Development and Characterization of an Interprocess Communications Interface and Controller for Bipedal Robots

James David Burton

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Tomonari Furukawa, Chair
Brian Y. Lattimer, Co-Chair
Alan Asbeck
Alexander Leonessa

November 30, 2015
Blacksburg, Virginia

Keywords: Humanoid Robot, ROS, Bifrost, Interprocess Communications, Robot Controller

Development and Characterization of an Interprocess Communications Interface and Controller for Bipedal Robots

James David Burton

(ABSTRACT)

As robotic systems grow in complexity, they inevitably undergo a process of specialization whereby they separate into an array of interconnected subsystems and individual processes. In order to function as a unified system, these processes rely heavily on interprocess communications (IPC) to transfer information between subsystems and various execution loops. This thesis presents the design, implementation, and validation of the Valor ROS Controller, a hybrid IPC interface layer and robot controller. The Valor ROS Controller connects the motion control system, implemented with the internally created Bifrost IPC, developed by Team VALOR for the DARPA Robotics Challenge (DRC) with the high level software developed by Team ViGIR that uses the Robot Operating System (ROS) IPC framework. The Valor ROS Controller also acts as a robot controller designed to run on THOR and ESCHER, and is configurable to use different control modes and controller implementations. By combining an IPC interface layer with controllers, the Valor ROS Controller enabled Team VALOR to use Team ViGIR's software capabilities at the DRC Finals.

In addition to the qualitative validation of Team VALOR competing at the DRC Finals, this thesis studies the efficiency of the Valor ROS Controller by quantifying its computational resourceful utilization, message pathway latency, and joint controller tracking. Another contribution of this thesis is the quantification of end-effector pose error incurred by whole-body motions. This phenomenon has been observed on both THOR and ESCHER as one of their arms moves through a trajectory, however, it has never been studied in depth on either robot. The results demonstrate that the Valor ROS Controller adequately uses computational resources and has message latencies in the order of 50 ms. The results also indicate several avenues to improve arm tracking in Team VALOR's system. Whole-body motions account for approximately 5 cm of the end-effector pose error observed on hardware when an arm is at near full extension.

This work was funded by ONR through grant N00014-15-1-2128, and by DARPA through grant N65236-12-1-1002. HDT Global also supported this work by lending of a pair of Adroit manipulator arms to Team VALOR for the DRC Finals.

Acknowledgments

My foremost thanks goes to God, who by his grace sustained through my degree.

I greatly appreciate the tremendous work done by the members of Team VALOR towards the DRC, without which, my thesis would not have been possible. In particular I would like to thank Jason Ziglar and John Peterson for making a better programmer.

I would also like to thank Team ViGIR for sharing their code base with Team VALOR.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Prospect of Bipedal Robots

Robots are becoming evermore present in modern life as they move out of the nearly sole purview of research labs and manufacturing plants into military and disaster situations, and onto local roads and backyards. However, the majority of robots, excluding the various types of unmanned air vehicles, reside in one of two locomotion categories, either they are fixed in place or move on a combination of wheels and treads. Both of these categories impose their own limits on the application of robotics. The first category is a staple of manufacturing robots that perform precise and repetitive tasks, but it is obviously incompatible with mobile robots. Wheeled and tracked robots have shown great success in a variety of mobile applications, however they often have trouble in both unstructured and human environments [1]. A tracked robot may have as much difficulty navigating a collapsed building as reaching the top of the same building without an elevator. Legged robots, particularly bipedal ones, provide an alternative mode of locomotion that, as shown by nature, is widely adaptable.

If robots are to help humans in the widest capacities possible, then they must be capable of interacting with humans in environments designed for human habitability [2]. Neglecting the fact that humanity has a fascination with itself [3], the bipedal robot with a humanoid torso has great synergistic potential to work alongside humans. Although difficult to control robustly, bipedal locomotion has the potential to take robots wherever humans could go but it otherwise would be too dangerous to risk a person. Torso mounted manipulators

with hands of sufficient dexterity also give humanoid robots the potential to interact with tools and environments as a person would. Thereby requiring little adaptation to accept the introduction of such robots. Bipedal humanoid robots are only just beginning to leave the controlled environments of the research lab, thus their potential is just beginning.

1.2 Interprocess Communication in Robotic Systems

As robotic systems scale up in size, the issue of transferring data throughout the system becomes increasingly important. While a simple robot might be able to run all of its controls and perception gathering in one program, good software architecture design dictates that different functions in complex system be broken up into separate subsystems. Communicating data between these subsystems, which are often entirely distinct programs, requires interprocess communications (IPC) methods and frameworks. While there are many IPC implementations, they all share the requirement that both sides of the communication utilize the same framework. This presents a challenge when a system calls for using two IPCs that must transfer data between each other. Adding the need to rapidly convert time-sensitive data between IPCs compounds this challenge. While at heart this is an engineering problem, collecting data to validate the interface provides insight into how the overall system behaves.

1.3 DRC and SAFFiR Projects

This work comes out of two projects that spurred the development of bipedal robots at Virginia Tech. Over the past few years, the U.S. Navy's Office of Naval Research (ONR) funded the Shipboard Autonomous Fire Fighting Robot (SAFFiR) project to develop a robot to combat shipboard fires and carryout other hazardous tasks. Out of this came a force controlled lower body proof of concept [4] and the full sized humanoid Tactical Hazardous Operations Robot (THOR). Concurrently, the Defense Advanced Research Projects Agency (DARPA) provided funding for Team VALOR, where VALOR stands for Virginia Advanced Legged Operations Robot, to compete in the DARPA Robotics Challenge (DRC). This Challenge, in line with previous DARPA Grand Challenges, pushed academic and corporate teams to apply robotics to difficult yet practical problems around disaster response scenarios.

Work towards this challenge resulted in the development of a follow up full sized force con-

trolled humanoid robot to THOR called the Electromechanical Series Compliant Humanoid for Emergency Response (ESCHER). THOR was originally intended for both the ONR SAFFiR project and the DRC, however, it became apparent that THOR could not operate untethered or complete some of the DRC tasks. ESCHER improves on THOR's design in that it is capable of running without a power tether, has force controlled arms, and stronger legs. The work in this thesis was primarily done using ESCHER, however, it is also designed to work on THOR also well.

The DRC provided an opportunity for Team VALOR to collaborate with two other DRC teams, Team ViGIR [5] based in Blacksburg, VA and Team Hector [6] from the Universität Darmstadt in Germany. Team Hector spawned out of Team ViGIR after the DRC Trails, in which Team ViGIR was a participant along side Team VALOR [7]. These three teams shared software, which provided Team VALOR many of the high level systems necessary to field ESCHER. However, integrating the code developed by Team ViGIR with that developed by Team VALOR was an arduous task. One particular challenge that the work in this thesis addressed is Team ViGIR used an open source IPC framework called Robot Operating System (ROS) [8] while Team VALOR used an internally developed IPC framework for their robot control system. This collaboration presented opportunities to work with both IPCs and Team ViGIR's robot controller.

1.4 Thesis Objectives

- Develop a ROS interface for the Motion System designed for THOR and ESCHER that is compatible with Team ViGIR's software for the DRC Finals
- Expand the applicability of Team ViGIR's Atlas Controller and characterize its performance
- Quantify the end-effector pose error induced by a whole-body control framework

1.5 Summary of Original Work

The contributions of this thesis are the preliminary steps towards expand the applicability of the control systems developed for THOR and ESCHER. However, these contributions

are also tightly coupled to the constraints associated with preparing for the DRC Finals. As such, there are three primary areas of novel work: interfacing the THOR and ESCHER Motion System with ROS, generalizing components of Team ViGIR's Atlas controller, and characterizing various aspects of the overall system. These contributions culminate in the development of the Valor ROS Controller.

The Valor ROS Controller developed in this thesis serves as the primary ROS interface to the Motion System used on THOR and ESCHER. This capacity required creating interfaces between the two primary IPC layers, ROS and Bifrost, used in Team VALOR's system. Two thrust were take to accomplish this, state data and control signals were routed directly into the data structures of the Valor ROS Controller while auxiliary interfaces of the Motion System were directly converted between ROS and Bifrost. This ensured that the internal controllers and command interfaces remained consistent between Team VALOR's and Team ViGIR's system while also providing access to the unique capabilities of the Motion System.

To maintain comparability with Team ViGIR's system, many of the components of the Valor ROS Controller are based on equivalent parts from Team ViGIR's Atlas controller. However, that controller was designed specifically for Boston Dynamics Inc's (BDI) Atlas robot. It is rife with references to Atlas specific values and functions that make the controller unusable for other robots. A major portion of the work of creating the Valor ROS Controller was removing these Atlas specific references and reworking internal data structures while maintaining the underlying functionality. In some cases it was more practical to reverse engineer the proprietary Atlas interfaces instead of refactoring certain portions of Team ViGIR's system. Coupled with this was the presumption of the existence of BDI behaviors. This led to the creation of a state machine mapping between control modes of the Valor ROS Controller, BDI behaviors, and Motion System states. An advantage to this re-engineering was the opportunity to implement an increase multithreaded design for the Valor ROS Controller.

The final contribution of this thesis focuses on quantifying the performance of the Valor ROS Controller and the ability of the system as a whole to reach goal end-effector poses. Process performance was determined to show that the Valor ROS Controller uses an acceptable level of computational resources. A lower bound on the Bifrost transmission latency is presented along with the experimentally estimated latencies for several important message pathways. Results from analyzing the control arm signals from the Valor ROS Controller and Motion System show areas of improvement regarding control of the arms. The results presented in

this thesis conclude by quantifying the end-effector error induced by a whole-body control framework. These results provide a first look at the overall performance of Team VALOR's system for the DRC Finals.

1.6 Thesis Outline

This thesis is separated into seven chapters that flow from discussing background material to the design and implementation of the Valor ROS Controller, and then to system testing results prior to concluding. Chapter two introduces a brief literature review that explores prior work related to custom IPCs for robotic applications. It will also take a look at work aimed at quantifying errors in manipulator end-effector poses, especially for humanoid and whole-body controlled robots. Chapter three introduces the fundamental knowledge required to understand the development of the Valor ROS Controller. It covers the ROS and Bifrost IPCs, the THOR and ESCHER Motion System, and the system developed by Team ViGIR for the DRC Finals. Chapter four discusses the primary original work of this thesis, namely the development and implementation of the Valor ROS Controller. Chapter five presents the results collected to characterize the Valor ROS Controller and the overall system. These results explore the performance of the Valor ROS Controller as a process, the message latencies between various points in the system, and the end-effector errors associated with the whole-body control algorithm. The sixth chapter concludes and presents potential areas for future work.

Chapter 2

Literature Review

2.1 Brief Review of Bipedal Robotics

2.1.1 The Beginnings of Bipedal Research and Modern Examples

Academic research and development in bipedal robotics has been ongoing since the 1970s [9]. Ground breaking work was done by Dr. I. Kato and his lab at Waseda University. After working on a number of bipedal robots capable of walking like motion in the late 1960s [10], they developed the first anthropomorphic robot, WABOT 1 [11]. However, this early robot according to Brooks was "essentially a playback mechanism" since it did not actively respond to its environment [12]. Researchers at Waseda University went on to develop a prolific number of bipedal and humanoid robots in the WL (Waseda Legged) and WABIAN (Waseda Biped humANoid) families [13] [14] [10] [3]. Japanese universities and companies continued to push bipedal robotics through the 1980s and 1990s [3] with robots such as the BLR-G2 [15], BIPER series from Tokyo University [16], and early Honda ASIMOs.

Work on bipedal humanoid robots has proliferated around the world with too many different designs to cover in a brief review. However, several notable robots competed at the DRC Finals. Prior to the competition, DARPA gave eight teams Atlas [17] built by Boston Dynamics. The Atlas is a force-controlled hydraulically powered humanoid and represents the most advanced humanoid developed by a U.S. company. Another commercial robot used by multiple teams at the DRC Finals was the position-controlled THORMANG [18], which

is produced by the Korean company ROBOTIS. Only two other robots were used by more the one team, the Hubo [19] [20] and HRP–2 [21] [22]. Both these robots are older position controller platforms, although the Hubos heavily modified for the Finals. In addition to ESCHER, two other state of the art force-controlled robots were developed for the DRC. These are the WALK-MAN [23] robot developed by the Italian Institute of Technology and Jaxon created by the JSK Lab at the University of Tokyo and based on the STARO [24] robot.

2.1.2 Bipedal Control Methods

Concurrently with Kato's work, Dr. M. Vukobratovic his team developed the concept of zero-moment point (ZMP) for locomotion stability [25] [9]. This represented the beginning of developing control paradigms for legged robots. ZMP has been one of the core underpinnings of walking algorithms used on bipedal robots since its inception [25]. Early humanoid robots developed at Virginia Tech, such as CHARLI, used this concept for their locomotion algorithms [26].

Recent work Virginia Tech in bipedal control has focused on improving stability robustness on unstructured terrains by using alternatives to ZMP. Pratt *et al.* introduce the Capture Point (CP) [27] and Linear Inverted Pendulum Plus Flywheel model, which builds off work done by Kajita *et al.* [28], to incorporate horizontal centroidal dynamics. An extension to the CP is the Divergent Component of Motion (DCM) introduced in [29] and extend to 3D by Englsberger, Ott, Albu-Schaffer [30]. At Virginia Tech, Hopkins *et al.* [31]extended the DCM to incorporate time-varying natural frequencies and applied it to a locomotion algorithm that takes in ZMP reference trajectories. This approach was demonstrated on hardware in [32].

2.2 IPC Development for Robotics

Interprocess communications (IPC) have been a necessary and important part of robotics for decades, although there has generally been little consensus on IPC methods or implementations. In 1987, Gauthier *et al.* [33] recognized the need for better IPC standardization in robotics. However, there was still a wide range of IPCs when Gowdy [34] reviewed the

available IPC frameworks in 2000. Of those reviewed by Gowdy, only one or two are still mentioned in literature today. In particular, it is interesting to note that IPC/IPT [35] developed Carnegie Mellon University became the basis for a number of project specific IPC implementations, including Tartan Racing for the DARPA Urban Grand Challenge [36]. A few years later in 2006, Broten *et al.* [37] reviewed IPCs in regards to the needs of Defense Research and Development Canada and found that the choice of IPC framework is still highly dependent on the likely needs of an institution or project.

Although there have been some recent trends towards consolidating IPC frameworks, no framework as of yet covers all of the needs of the robotics community. Thus, in addition to the Robot Operating System (ROS) [8], there are a number of widely used and niche IPCs. Metta, Fitzpatrick, and Natale introduced YARP (Yet Another Robot Platform) [38] in 2006 as a full IPC and robotics development platform. Although similar in scope as ROS, it did not experience the same level of adoption. In 2010, the Lightweight Communications and Marshalling (LCM) [39] [40] IPC was developed at MIT as a low-latency message passing system for real-time systems. Dantum *et al.* [41] also developed an IPC library for extremely low-latency real-time applications called Ach in 2014. In addition to these, there are likely countless custom IPC implementations used by university and corporate researchers.

Chapter 3

Fundamental Concepts

3.1 ESCHER and THOR Anatomy Overview

THOR and ESCHER, shown in Figure 3.1, are two advanced electromechanical humanoid research platforms built by TREC. They both run the Whole-Body Controller developed by Michael Hopkins [32] and are capable of a variety of locomotion and manipulation tasks. In addition, each robot has a host of sensors for perceiving the world. This section will briefly describe each robot, however, further information can be found in these papers (citations for each robot if they exist).

3.1.1 Legs and Lower Bodies

The two 6-DOF legs of THOR and ESCHER's lower bodies each are each driven by electric linear series elastic actuators (SEA). SEAs, as developed by Pratt and Williamson [43] introduce an elastic element between an actuators output and its load. This decreases the rigid coupling between the actuator and the environment, and improves the effectiveness of force control methods. Linear SEAs, as opposed to rotatory SEAs, act as prismatic joints by moving their drive-shafts along the axial axis of the actuator. The linear SEAs used on THOR and ESCHER are similar in principle to other SEAs [44] [45], however, as discussed in Knabe *et al.* [46], they use a cantilevered titanium beam as the elastic element. This reduces the overall length of the actuator which allows the actuator to be placed into smaller packages. Further information on the design of THOR's may be found in [47] [48], while

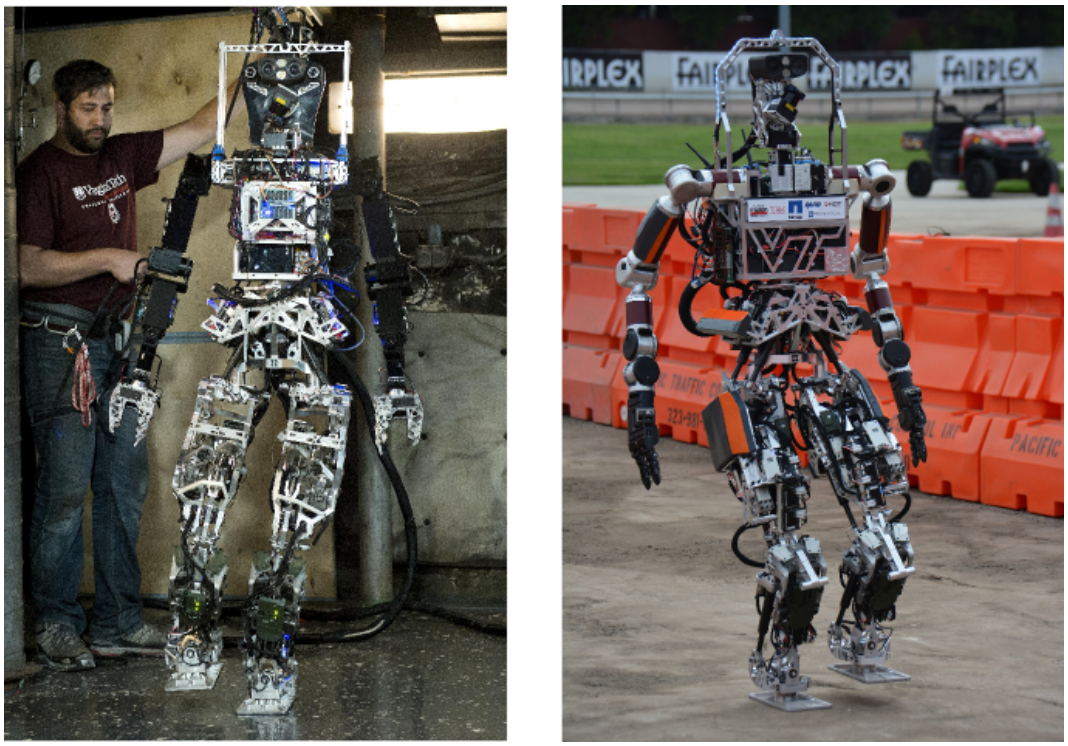

Figure 3.1: Left: THOR at the ONR SAFFiR demo on the U.S.S. Shadwell (photo courtesy of Virginia Tech /Logan Wallace). Right: ESCHER walking at the DRC [42].

Knabe *et al.* discussion the improvements made on ESCHER in [49].

3.1.2 Arms and Upper Bodies

As one would expect for humanoid robots, THOR and ESCHER each have an upper torso with two arms and a head. The chest of each robot is used to store computers and power electronics required to run the robots. ESCHER additionally has nearly half its chest cavity devoted to four lithium-polymer batteries that allow the robot to operate without a tether, a requirement for the DRC. The heads of each robot are identical pan-tilt mechanisms and carry the primary perception sensors of each robot.

THOR and ESCHER are each equipped with two seven degree of freedom arms for manipulation tasks. THOR uses ROBOTIS Dynamixel arms [50] with roll-yaw-roll wrists while ESCHER uses HDT Adroit arms [51] with the more standard roll-pitch-yaw wrist configuration, as shown in Figure 3.2. THOR's arms use custom made two degree of freedom grippers, while ESCHER uses four degree of hands from HDT. The Adroit arms have several advantages over the Dynamixal arms, foremost of which is the ability to do impedance control with them. In addition, the Adroit arms are stronger, more reliable, and have a greater range of motion than the Dynamixel arms.

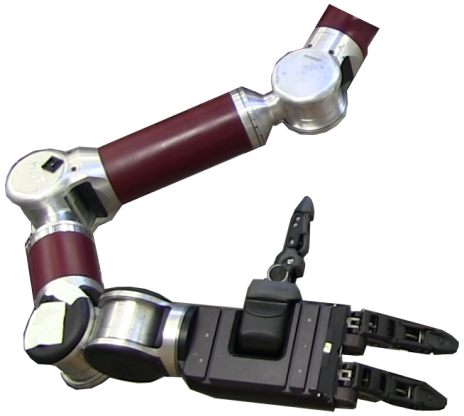

Figure 3.2: Picture of right HDT Adroit arm in Virginia Tech colors.

3.1.3 Perception Sensor Packages

The sensor packages of each robot are quite similar with most differences coming from which arms are used. Each robot has a pan-tilt head equipped with a Carnegie Robotics Multisense S7 [52], a rolling Hokuyo UTM–30LX-EW lidar [53], two mount points for two FLIR A35 thermal imaging cameras [54]. The Multisense S7 is a stereoscopic imager that provides RGBd images by running a Semi-global-matching algorithm on internal FPGA. This sensor is capable of producing 0.5–2 megapixel color images and colorized point clouds at 15Hz. The Hokuyo is mounted on an actuated carriage that rolls about the x-axis of each robot. This sensor provides a wide 3D point cloud of the environment around the robot, even including things over the shoulder. The FLIRs allow the robots to produce stereoscopic depth images using inferred. This lets the robots detect fires and *see* in smoke filled environments when the other sensors fail.

The multimodal sensor packages provide each robot, and any human operators, a variety of ways to view the world and the ability to adjust to changing environmental factors. For example, regular cameras and lidars provide rich data in nominal conditions, but quickly degrade in dusty or smoky environments. However, the FLIR cameras are able to see through smoke and dust.

Each robot has two inertial measurement sensors. An attitude, heading, and reference (AHRS) MicroStrain 3DM-GX4 (CHECK TYPE, cite page) that the Motion System uses for instantaneous accelerations and velocities, and a KVH 1750 Inertial Measurement Unit (IMU) [55] for odometry and mapping.

3.2 ROS and Bifrost

Anytime two or more concurrent computer programs interact by exchanging information, it is facilitated by an interprocess communication (IPC) method or framework. IPCs may range in complexity from sharing the same memory addresses to intricate message passing schemes. Silberschatz, Galvin, and Gagne present an overview of different IPC mechanisms in chapter 3 of [56] while Gray extensively discusses Linux based IPCs in [57]. Team VALOR's software stack utilizes two IPC frameworks, Bifrost and ROS [8]. These frameworks both abstract the the actual IPC implementations in order to simplify software development. Each framework

treats individual processes as nodes with connections to other nodes called topics. The follows sections will briefly discuss these IPCs.

3.2.1 Bifrost

Bifrost is a custom IPC protocol developed by Jason Ziglar for the SAFFiR and DRC projects. The initial intension was to use it as the primary IPC protocol for the software systems run on THOR and ESCHER, however, only the Motion System was mature enough to begin using Bifrost. As the scale of the projects became apparent, it was deemed too costly in time and resources to implement other subsystem using Bifrost. Instead of writing all of its own software, Team VALOR decided to seek out viable open source alternatives. Thus, the communication protocols used on THOR and ESCHER diverged as the Motion System continued development using Bifrost while the rest of the software transitioned to using ROS.

Bifrost is a Transmission Control Protocol (TCP) socket based distributed message passing protocol implemented in C++ based on the ZeroMQ library [58]. It allows processes, referred to as nodes, on one or more computers to communicate with each other using publisher-subscriber and request-reply interfaces. Each interface uses the same underlying idea of sending and receiving TCP messages, but they differ in how the nodes interact with each other. The publisher-subscriber interface allows nodes to publish data to any node that wishes to listen without regard to what those subscribing nodes do with the data. The request-reply interface, on the other hand, acts like a server-client interaction where a client node asks the server node to perform some action and then waits for a reply. Each node may have various interfaces and connections to other nodes. These connections between nodes are called topics, and are represented by strings called topic names. Once the Bifrost daemon is running, it opens a number of TCP sockets corresponding to the number of computers on the Bifrost network. Connections implicitly form if individual nodes have matching interface pairs with the same topic names. This allows for dynamic and anonymous networks of nodes. Since interfaces only rely on topic names, nodes do not have to know of the existence of other nodes.

Figure 3.3 shows an example of one Bifrost node communicating with two other nodes on one computer. Node A uses a publish-subscriber interface to publish a constant stream of data on a message topic. Nodes B and C subscribe to this same topic, thus data is transfered

unidirectionally from node A to nodes B and C. Node A also exposes a server using a request-reply interface. If node C has a client on the corresponding service topic, then it may make service calls to request information from node A.

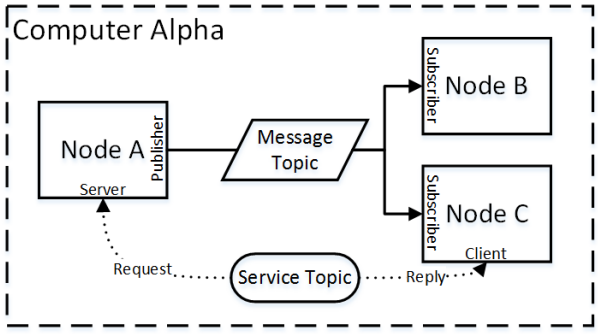

Figure 3.3: Bifrost Communication Diagram. Bifrost Nodes B and C are subscribed to a message topic from node A, while node C is also a client of node A.

3.2.2 ROS

ROS [8] is an open source communications framework for robotic applications. It currently boasts users from around the world in research, commercial, and industrial applications. In addition to handling communications between processes, ROS also integrates a number of useful diagnostic and visualization tools. There are hundreds of supported packages and countless user generated ones that provides features from sensor interfaces to mapping algorithms. This large base of support and available features makes ROS a valuable effort multiplier for small teams.

ROS initially started as an extension of work done at Stanford University and Willow Garage [8] [59]. Its first distribution was released in 2010 under the title Box Turtle, and since then it has had nine total major releases so far, from Box Turtle to Jade Turtle in 2015 [59]. During the DRC, Team VALOR used ROS Indigo Igloo, which was the latest distribution at the time.

Like Bifrost, ROS acts as an IPC communications layer between independent nodes. ROS is a data driven architecture that uses publisher-subscriber and server-client pairs in a similar way to Bifrost, However, ROS has number of important differences. ROS is natively implemented in number of languages, including C++ and Python, and has a number of wrappers for additional languages. ROS also requires the use of a name server, called ROS

Master, to maintain a list of nodes and available topics. Nodes communicate with the master server using the XML-RPC protocol. For peer-to-peer communications, nodes generally use TCP/IP sockets through the TCPROS protocol [60]. Figure 3.4 shows a simple diagram of two ROS nodes communicating with message and service topics. In addition to the concepts of messages and services, ROS also uses the idea of actions. These are similar to service calls, however, they are designed for long duration tasks that are capable of providing feedback. These communication interfaces provide ROS a great deal of flexibility for robotic applications.

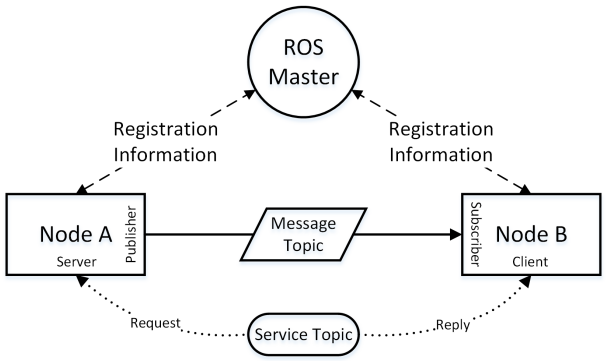

Figure 3.4: Simple ROS Node Diagram. This diagram shows two ROS nodes communicating with each other using message and service topics. Each node also communicates with the ROS Master server for up-to-date information about available topics.

3.3 ROS Control Overview

ROS Control [61] is a set of libraries and concepts in ROS for a robot agnostic control framework that is capable of supporting both real-time and non-real-time applications. It consists of several ROS packages that abstract robot hardware and controllers. These abstractions allow for controllers that are separate from specific hardware implementations, and thus reusable on multiple platforms or actuator configurations. ROS Control also provides methods for managing and switching between active controllers. This makes it a versatile and extensible control framework.

ROS Control in its basic form comprises of four components: controllers, a controller manager, hardware resource interfaces, and a hardware interface. Figure 3.5 [61] shows how data flows through these components to and from a robot. Controllers, which range

from simple PID position controllers to complex custom controllers, operate from inside a controller manager, which starts, stops, loads, and unloads controllers as needed. The controller manager also ensures that there are no resource conflicts between controllers, that is, only one controller is allowed to command an individual joint at a time. Each controller exposes its own command interfaces. While active, they use the given hardware resource interfaces to store commands and retrieve data from the hardware interface, which acts as a centralized storage repository. This process repeats each time an update is executed by the controller manager. The ROS Control end-user is then responsible for writing *write()* and *read()* functions that respectively send commands to the hardware and updates state data stored in the hardware interface.

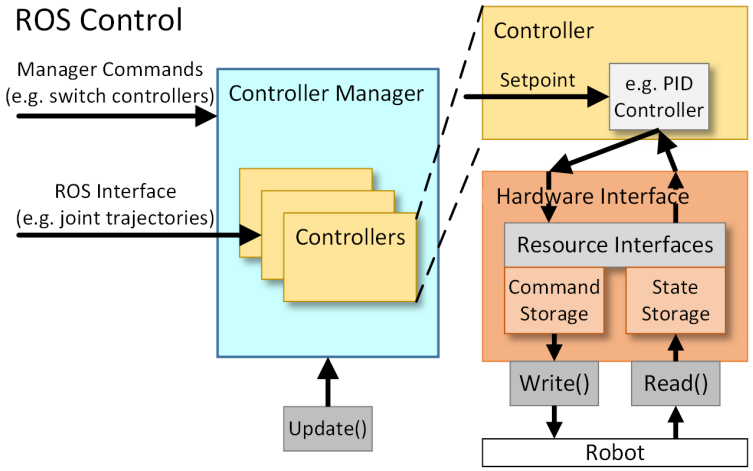

Figure 3.5: Diagram of basic ROS Control components and their interactions.

3.4 Review of the Motion System

The balance and locomotion software on THOR and ESCHER is the result of the work done by Michael Hopkins and Robert Griffin [62] [31]. This work culminate in the development of the Motion System, a collection of nodes and subsystems responsible for interfacing with hardware, balancing, and locomotion. The Motion System is the only part of Team VALOR's software stack that relies on Bifrost. This is because a preliminary version of it was developed before Team VALOR transitioned to using ROS. Because of this, the Motion System is entirely ROS free during operation and only depends on a couple of ROS robot description files for commonality with the rest of the software stack.

In the overall system that runs on THOR and ESCHER, the Motion System is often thought of as being at the bottom of the stack, in that it is the closet part of the system to the hardware that does not run on an embedded device. This means that all commands to the robot and proprioception from the robot go through the Motion System. As such, the Motion System acts as a gatekeeper to the robot. Only interfaces implemented by the Motion System may be used by higher level software, and any commands to the robot may be altered to maintain the current objectives of the Motion System.

The Motion System consists of three primary components: the Motion Manager, the Whole-Body Controller, and the Comms Manager. In addition there are various configuration and other supporting files that tune the system to work with different robot configurations. The Motion Manager is one of two nodes that start up when the Motions System starts. It acts as a wrapper for all of the joint level and robot level controllers, and provides exterior interfaces for other nodes to interact with the Motion System. The Motion Manager also handles all of the finite state machines (FSMs) used by the Motion System. Within the Motion Manager is the Whole-Body Controller, which is an implementation of a momentum-based approach to whole-body control. The Comms Manager is the second node of the Motion System, and manages the hardware interfaces of the robot. In addition to sending commands from the Motion Manager to each motor, it packages proprioception data and sends it further up the software stack. Each of theses three components work together to successfully run THOR and ESCHER.

The following diagram, Figure 3.6, shows each of the Motion System components in relation to each other. The Motion Manager on the left wraps and provides interfaces to the Whole-Body Controller, while the Comms Manager on the right transmits data to and from the robot's hardware. The following sections will cover each major component in more detail.

3.4.1 Motion Manager

The Motion Manager acts as the central hub of the Motion System. It is the primary interface to the robot and the Whole-Body Controller. The Motion Manager interacts with other nodes over a variety of Bifrost topics. Each topic has a message handler that routes the incoming command or data to the correct location. In addition, it publishes various state information about the robot and Motion System. It also sends joint setpoints from the Whole-Body Controller to the Comms Manager using a Motor message topic as shown in

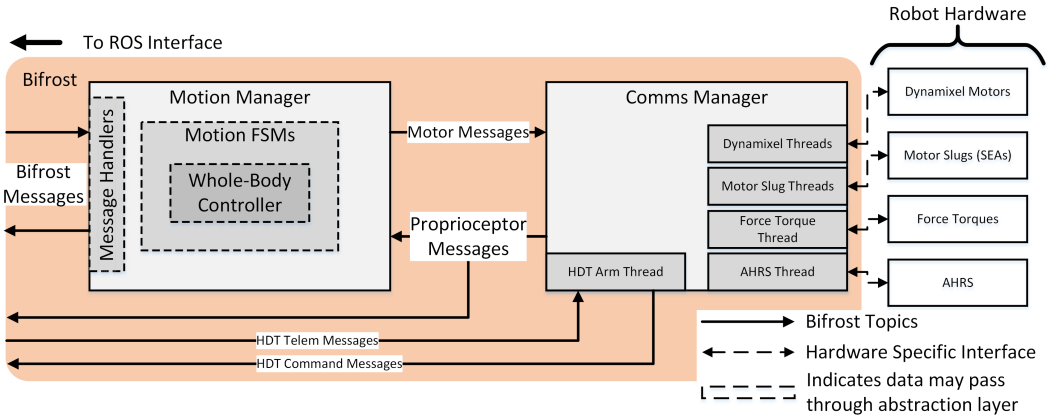

Figure 3.6: The relationships between each Motion System component. The Whole-Body Controller runs inside of the Motion Manager, which in turn sends commands to the Comms Manager to be sent to the hardware.

Figure 3.6. In this capacity, the Motion Manager has exclusive control over joint commands sent to the robot. By managing these interactions, the Motion Manager forms the core of the Motion System.

The FSMs in the Motion Manager play an important roll in how the Motion System behaves and reacts to different stimuli. Inside the Motion Manager, there are four separate FSMs. There is a Behavior FSM and three internal FSMs that manage specific parts of the robot. These three FSMs are the Locomotion, Manipulation, and Attention FSMs. Each of the internal FSMs have several states that either toggle different functions of the Motion System. The Locomotion FSM determines, as the name implies, the balancing and walking modes, the Manipulation FSM determines whether the arm joints are available for manipulation task, and the Attention FSM controls the head motors. Figure 3.7 shows the states for each internal FSM. The arrows indicate the allowed transitions from one state to the next. The Behavior FSM is a little different in that its states simply map to combinations of the other FSM states. Since it simplifies the possible combinations, it is the primary FSM other nodes interact with.

The Behavior FSM has a few primary states of operation, each with an associated manipulation state, and a few startup and error states. Figure 3.8 shows the Behavior FSM state transition map. Upon startup, the robot moves from the Idle state into the Stand Prep state. It then waits for a command to go into the Stand state, at which point the robot is fully balancing by itself and is accept additional commands. Footstep plans are executed in

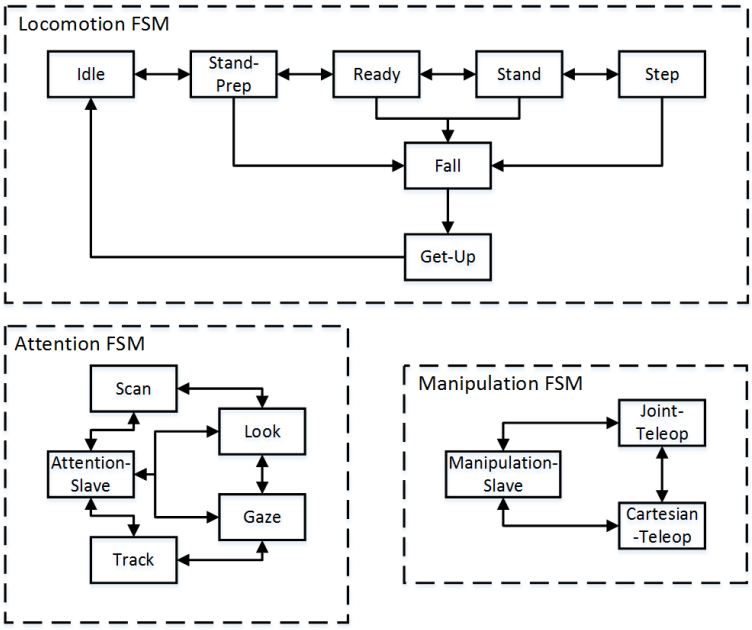

Figure 3.7: The three internal Motion System FSM states and their allowed transitions between states.

the Step state while linear and rotational velocity commands are executed in the Walk state. The three preceding states each have an associated manipulate mode that allows external commands to be sent to the arms. As a safety measure, the Behavior FSM can go into a Soft Stop mode that will maintain the robot's balance, but lock out command interfaces. From certain states, the Behavior FSM can also be put into the Freeze state that locks the joints, or a subset of joints, at their current positions. Two other states of to note are the Fall and Get Up states. At the moment, if the robot loses its balance, then any of the operational states enter the Fall state. However, once in the Fall state, the robot is considered dead since there is not yet a get up routine for either THOR or ESCHER.

Table 3.1 shows the corresponding states of the internal FSMs for each Behavior state. This table shows that, for all of the potential combinations of internal FSM states, the behaviors of the robot revolve primarily around how it moves about and whether it is using its arms at the same time. The Manipulation and Attention FSMs are basically relegated to binary switches determine when the arms and head joints may receive commands.

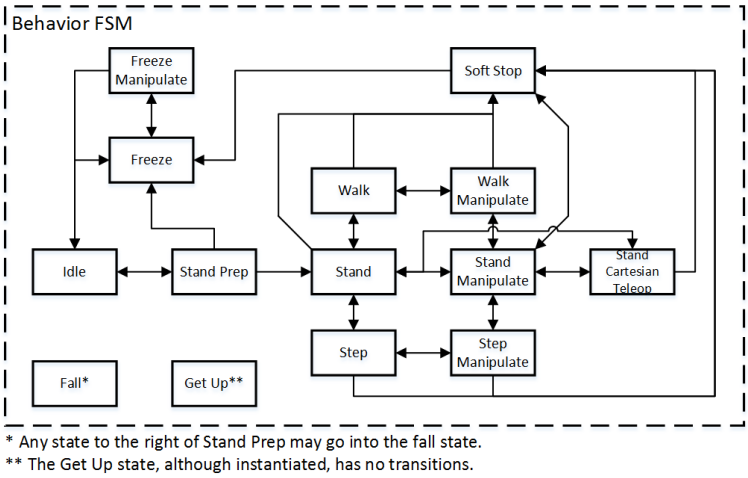

Figure 3.8: Behavior FSM state transition map.

Table 3.1: Motion System Behavior state to internal FSM state mapping.

	Motion System Internal FSM States		
Behavior States	Locomotion State	Manipulation State	Attention State
Fall	Fall	Slave	Slave
Freeze	Ready	Slave	Slave
Freeze Manipulate	Ready	Joint Teleop	Look
Get Up	Get Up	Slave	Slave
Idle	Idle	Slave	Slave
Soft Stop	Stand	Joint Teleop	Look
Stand	Stand	Slave	Look
Stand Cartesian Teleop	Stand	Cartesian Teleop	Look
Stand Manipulate	Stand	Joint Teleop	Look
Stand Prep	Prep	Slave	Slave
Step	Step	Slave	Look
Step Manipulate	Step	Joint Teleop	Look
Walk	Walk	Slave	Look
Walk Manipulate	Walk	Joint Teleop	Look

3.4.2 Whole-Body Controller

The Whole-Body Controller is the magic behind the robust balancing and locomotions behaviors displayed by THOR and ESCHER. It implements an optimization-base whole-body controller based on the Divergent Component of Motion as described in [32] [62]. The Whole-Body Controller computes optimal joint torques for all of the joints, except for the joints in the neck and grippers of each robot, given frictional contract constraints on the feet. It also integrates the joint torques to calculate position and velocity setpoints for non-force controlled joints. Although it is conceptually convenient to consider the Whole-Body Controller a separate system, it runs in the same overall update loop as the Motion Manager.

3.4.3 Comms Manager

In the Motion System, the Comms Manager acts as the monolithic interface to the robot hardware. Inside of the Comms Manager, each hardware component has its own loop that runs in an independent thread. While running, the Comms Manager's main loop periodically sends new commands to each of these other threads from Motor messages and aggregates new sensor data into Proprioceptor messages. Unlike the Motion Manager, Comms Manager is different between simulation and hardware and is specific to each robot since each robot has a slightly different hardware configuration. Referring to Figure 3.6 again, the ESCHER Comms Manager uses five threads to interact with the hardware. Four of the threads use hardware specific protocols to directly communicate while the HDT arm thread uses Bifrost topics, that later get converted into ROS topics, to send and receive data. This somewhat awkward configuration is because the HDT arms use ROS based drivers.

3.5 Review of the ViGIR Atlas Controller

The ViGIR Atlas Controller (VAC) is in many ways similar to Team VALOR's Motion System. It runs all of Team ViGIR's control algorithms and handles communications with the Atlas robot that ViGIR received from DARPA. All control commands and a number of sensor streams, including joint states, inertial readings, robot diagnostics, go through the VAC. The design of the VAC extends uses and extends ROS Control's concepts to create an abstracted finite state machine that can run multiple concurrent controllers for the same

hardware resources. Figure 3.9 shows the position of the VAC within Team ViGIR's software system, in between Team ViGIR's high level software capabilities and the Atlas robot.

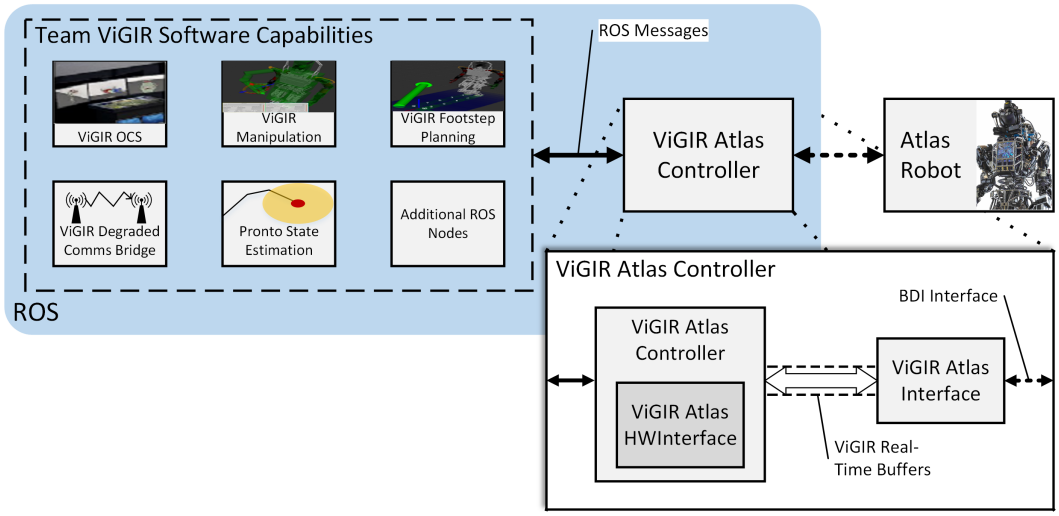

Figure 3.9: The ViGIR Atlas Controller in Team ViGIR's software system.

As shown in the breakout box in Figure 3.9, the VAC consists of three main components: the ViGIR Atlas Controller, the ViGIR Atlas HWInterface, and the ViGIR Atlas Interface. To be clear, VAC refers to the entire system, while the spelled out ViGIR Atlas Controller refers to an individual component.

The ViGIR Atlas Controller contains three ROS Control controller managers, called Control Mode Manager, Joint Manager, and Robot Manager, that handle controllers related to different aspects of the robot. The Joint Manager and Robot Manager run the active joint level and robot level controllers respectively while the Control Mode Manager determines which controllers are active. The ViGIR Atlas Controller also runs the main controller loop on the controller and handles command rectification to prevent controllers from sending conflicting signals.

Team ViGIR took the ROS Control concept of controller managers, which are capable of seamlessly handling transitions between controllers, a step further by creating a control mode ROS controller run by its own controller manager to act as a finite state machine. The Control Mode Manager, which runs this control mode controller, is able to command the other controller managers switch active controllers to correspond to the current control mode of the robot. This combination of control mode controller and controller manager allows for the specification of control modes that designate the active joint and robot controllers, and

thus different robot functions.

The ViGIR Atlas HWInterface implements the hardware interface class required by ROS controllers. It acts as a resource storage location for individual controllers and the ViGIR Atlas Controller.

The ViGIR Atlas Interface handles communicating with the robot. This interface runs in a separate thread to ensure rapid and continuous communication with the robot. In addition to sending commands and receiving data from the Atlas robot, the ViGIR Atlas Interface converts between Team ViGIR's control modes and BDI's Behavior states used by low level computers on the Atlas.

To ensure close to real-time communications within the VAC, Team ViGIR implemented a simple shared-memory based IPC using two element circular buffers. These ViGIR Real-Time Buffers (RTB) stored data in two memory locations and used C++ pointers to access and store data. Figure 3.10 shows the process of writing data into one of these RTBs. Once new data gets written to the memory location pointed to by the write pointer, the write and read pointers get switched so that the read pointer now points to the memory location containing the new data. Since this pointer manipulation is atomic (i.e., pointer assignments take one clock cycle), these buffers guarantee that data does not become corrupted by two threads trying to read and write the same memory location at the same time. These RTBs formed the backbone of the communication pathways between the ViGIR Atlas Controller and the ViGIR Atlas Interface.

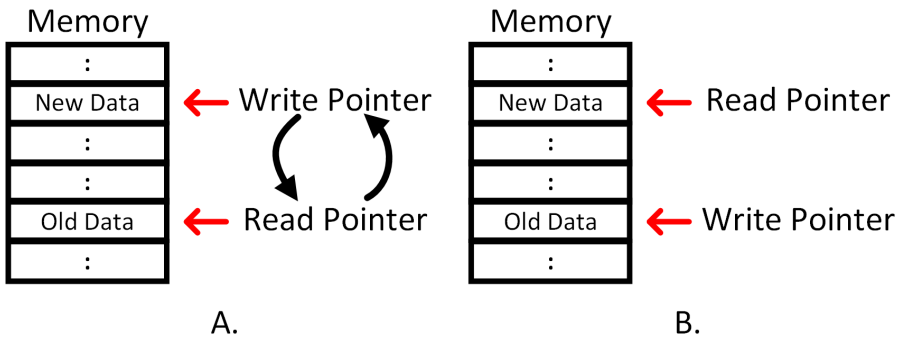

Figure 3.10: ViGIR Real-Time Buffer (RTB) process of writing new data. (A) New data is written to memory and write and read pointers are switched. (B) Read pointer now points to new data.

Chapter 4

The Valor ROS Controller

Competing in the DRC required interfacing with and integrating Team ViGIR's comprehensive software development with Team Valor's organic Motion System. To fail to do so would mean that Team Valor would a have walking humanoid robot without the ability to operate it in the DRC's degraded communications environment. In addition, all of Team ViGIR's manipulation and operator control station software stacks would be wasted. However, the success of this integration would come down to a few key areas, the primary of which was the gap between Team ViGIR's Vigir Atlas Controller (VAC) and Team Valor's Motion System. All of the ROS based commands sent to the VAC would have to be sent on to the Motion System as Bifrost commands. It is this seemingly simple action, converting between ROS to Bifrost, at the focal point Team ViGIR's and Team Valor's systems that forms the basis of this work.

The result of this work is called the Valor ROS Controller. The following sections of this chapter will go into the design and implementation of this system. First, the system goals and overall design will be discussed. This is followed by a look at the system implementation that includes the class hierarchy and treading structure. And finally, this section will conclude with a discussion of the data collected and collection methods used to validate and benchmark the Valor ROS Controller.

4.1 System Design

The design of the Valor ROS Controller follows the VAC and extends it for Team Valor's needs. In additions to interfacing between ROS and Bifrost topics with data common to both Team ViGIR's and Team Valor's systems, the Valor ROS Controller also needed to fulfill several other goals. The following sections will look all of the goals and requirements of the Valor ROS Controller, and steps necessary to get there.

4.1.1 Design Goals and Other Considerations

The overarching design goals of the Valor ROS Controller were relatively straight forward given the seemingly well defined problem at hand. From a highly view, the problem was to connect Team ViGIR's ROS based system to the Motion System by taking incoming data on ROS topics and republish that data on Bifrost topics, and vise versa. However, it is also important to look beyond the problems immediately at hand, and to consider the future of the systems built of the DRC. With this in mind, high level goals for the Valor ROS Controller were as follows:

- Expose the Motion System's capabilities and act as a bridge between ROS and Bifrost topics while maintaining Team ViGIR's interfaces as much as possible
- Implement as quickly as possible so that whole system testing can begin
- Stand as a possible path forward for moving the Motion System into ROS and C++
- Be as robot agnostic as possible

The first two goals extend directly from the necessities surrounding the DRC project and were the only hard requirements. If the Valor ROS Controller failed to interface between ROS and Bifrost using Team ViGIR's interfaces or took too long to implement, then it would be useless during the DRC. The last two goals were designed to look towards the future after the DRC was finished. Once the decision was made in TREC to move to ROS as the primary communications layer, it has been desired to move the Motion System into ROS as well. In addition, this would give an opportunity to port the parts of the Motion System that use Lua into C++. The last goal stems from a desire to see the Valor Escher Controller work on

THOR as well as ESCHER. While it is likely impossible to agnostic to the robot hardware involved, this goal pushed towards a more interchangeable design.

In addition to the general goals discussed above, their were also several other considerations that lead to the design of the Valor ROS Controller. During the SAFFiR project, I wrote a node to convert between equivalent ROS and Bifrost topics called the Ros-Bifrost Bridge. This experience showed that direct ROS-Bifrost conversion was possible for relatively simple and well defined pathways. However, it also showed that some messages, such as trajectory commands, require interpolation and spooling into messages streams for the Motion System to correctly handle them. Team ViGIR's large, seemingly ever changing, and poorly documented system would also make a direct one-to-one conversion scheme very difficult to implement. Moreover, some of the interfaces that Team ViGIR used had no direct equivalents in the Motion System and would require aggregate data from different Motion System topics or by filling through different means. Even though the approach used for the SAFFiR project appeared insufficient for the DRC, insights and techniques from the Ros-Bifrost Bridge were incorporated into the Valor ROS Controller.

Other consideration included minimizing computer resources used by the Valor ROS Controller and minimizing the latency introduced by converting between ROS and Bifrost. Since the Valor ROS Controller would be acting as the interface to the Motion System, and thus running on the same computer as the Motion System, it could not take so many computer resources that it slowed down the Motion System. Likewise, any significant additional latency induced by the Valor ROS Controller could not negatively affect the effectiveness of the Motion System. Latency would be particularly detrimental to the OCS and state estimation system since they rely on data from the robot to properly show the location of the robot and its joints. Although these considerations are strongly dependent on implementation details, they did play a significant roll in the final design of the Valor ROS Controller.

4.1.2 Overview of Overall Design

With the introduction of the Motion System, Team Valor's overall system architecture grew in complexity. The use of both ROS and Bifrost added to the complexity of Team Valor's software architecture for the DRC by requiring a layer nodes to interface between the ROS and Bifrost IPCs. Figure 4.1 shows the high level system architecture used by Team Valor. Like Team ViGIR's system (shown in Figure 3.9), ROS nodes for different high

level subsystems and planners communicate with the robot through a central node, which in Team Valor's case is the Valor ROS Controller. The one exception to this is ViGIR's Footstep Planner which passes footstep plans through the Footstep Converter. However, unlike Team ViGIR's design, the Valor ROS Controller and Footstep Converter form the ROS-Bifrost IPC interface layer. This means that these nodes straddle ROS and Bifrost in order convert messages pass messages between each IPC in addition to their other rolls. On the Bifrost side of the IPC interface layer is the Motion Manager, which, as discussed in Motion Manager (subsection 3.4.1), is the final arbiter of commands executed on the robot. The Valor ROS Controller is a mission critical node because it forms the bulk of the IPC interface layer.

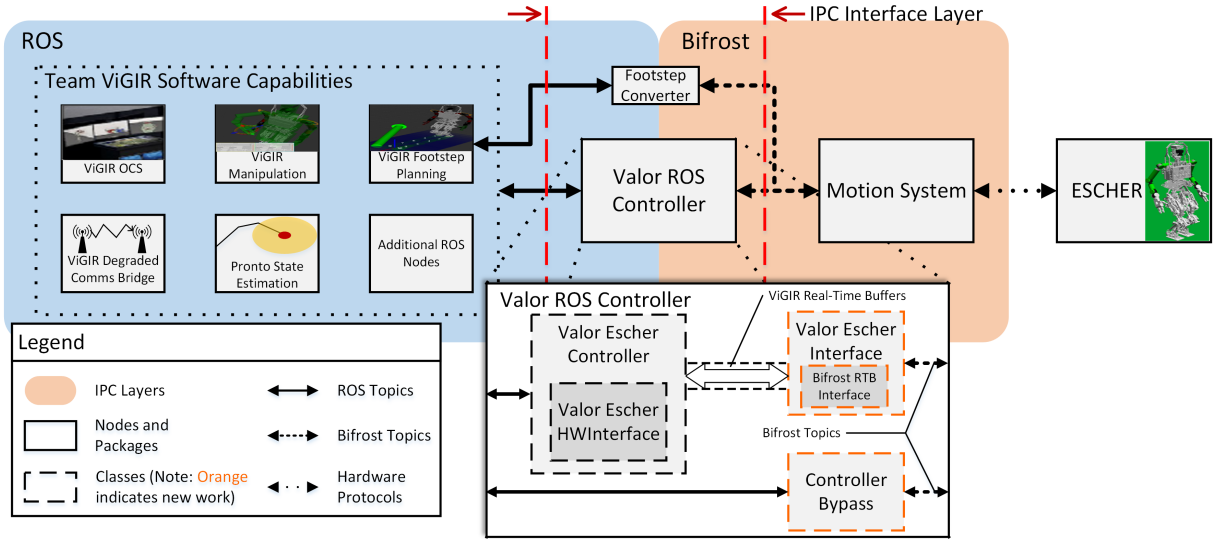

Figure 4.1: The Valor ROS Controller in the overall system.

The design of the Valor ROS Controller itself follows the design of Team ViGIR's VAC. Figure 4.1 shows the Valor ROS Controllers place in the overall system used for the DRC. The three primary components of the VAC have been renamed the Valor Escher Controller, Valor Escher HWInterface, and Valor Escher Interface to reflect their tailoring for Team Valor. The ViGIR RTBs are still used to convey data between the controller and interface. Along with various implementation modifications, there are a couple of major additions shown in the exploded view of the Valor ROS Controller in Figure 4.1. The new Bifrost RTB Interface is what makes the Valor ROS Controller capable of communicating over Bifrost channels. This then allows the controller to map the interfaces common to both Team ViGIR and the Motion System to each other. The other addition is the Controller Bypass, acts as

way of exposing interfaces unique to the Motion System. These additions will be further discussed in the following sections and in the System Implementation (section 4.2) section of this chapter.

4.1.3 Modifications to Team ViGIR's System

Even though Team ViGIR's VAC provided a good foundation for the Valor ROS Controller, it still required extensive modifications in some areas. To begin with, it was necessary to remove the computations and hooks associated with their partially implemented whole-body robot controllers since these functions were handled by the Motion System. This included control schemes for balancing, whole-body poses, and gravity compensation for the atlas arms. There were also numerous configuration files that needed reworking for ESCHER and THOR. These ranged from files defining each controlled joint to those that specified the allowed modes of operation. Finally, all of the modified files had there names changed to reflect their new purposes.

Another major area that need changing was the Atlas specific nature of many of the components in the VAC. All of the references to BDI's proprietary Atlas interface class and the numerous hard coded Atlas specific values used throughout the controller had to be removed. Not only did these references limit the controller to Atlas, they also made it impossible for Team Valor build compile the controller since Team Valor did not have access to BDI's software. The internal data types used by the controller to pass information between its various parts also required modification and simplification. For example, several data types had fields for hydraulic pressure and BDI error messages which are important information for hydraulically powered Atlases, but are utterly useless for electric robots. Further more, the robot interface, the Vigir Atlas Interface which became the Valor Escher Interface, had to be nearly entirely rewritten to work with ESCHER and THOR. In the end, any code from ViGIR's VAC had to be combed line by line before it could be used in the Valor ROS Controller.

4.1.4 Additions to Team ViGIR's System

Interfacing with Bifrost and the unique capabilities offered by the Motion System required adding several additions to Team ViGIR's controller. Figure 4.1 shows two of these additions

in the exploded view of the Valor ROS Controller. The first is the Bifrost RTB Interface that was incorporated into the Valor Escher Interface. The second, and most obvious, addition is the Controller Bypass that routes data around the main parts of the controller. There was also additional work do to map the various Control Modes, BDI FSM states, and Motion States to each other. Each of theses additions will be discussed in more detail below.

The primary addition to make the Valor ROS Controller work with Team Valor's system is a class to directly interface between Bifrost topics and the internal ViGIR RTBs. This class, called the Bifrost RTB Interface, associates Bifrost topics with RTBs related to their message contents. It allows Bifrost subscribers to directly place data into their associated RTB and Bifrost publishers to publish new commands straight from their associated RTB. Multiple Bifrost topics can populate or publish from the same RTB, thus allowing the controller to more efficiently move data around by using less RTBs. The rewritten Valor Escher Interface extensively uses this class to do the heavy lifting of communicating with the Motion System.

The second major addition focuses on exposing the Motion System's unique interfaces. The Controller Bypass does exactly what its name implies by routing data around the Valor Escher Controller and the Valor Escher Interface. It is based off of the experience gain from making the Ros-Bifrost Bridge for the SAFFiR project. Like the Ros-Bifrost Bridge, the Controller Bypass has ROS-Bifrost subscriber and publisher pairs for each topic going through the bypass. The Controller Bypass runs in its own thread and is kept separate from the reset controller in order to prevent unnecessary latency while converting topics unrelated to Team ViGIR's interfaces in the main part of the controller. By bypassing the main parts of the Valor ROS Controller, the Controller Bypass provides an efficient interface for the Motions System's unique capabilities.

A less distinctive, albeit still important, design addition to the Valor ROS Controller was mapping the states from the various FSMs used in the controller and Motion System to each other. This first required specifying the allowed Control Modes that the OCS may command the Valor ROS Controller to enter. Those modes were then mapped to the BDI Behavior states that were already widely referenced in the controller. The BDI Behaviors were then mapped to the Motion System Behavior states. Once the mapping were determined, the Valor Escher Interface was tasked with ensuring that only allowed transitions were executed to ensure that the current Valor ROS Controller state did not desynchronize from the Motion System state.

4.2 System Implementation

The Valor ROS Controller's implementation stems from its need to interface with Bifrost and match the ROS interfaces used by Team ViGIR. It also needed to abstract as much robot specific information as possible so that it would be robot agnostic. To accomplish this, the Valor ROS Controller uses the C++ application program interfaces (API) provided by ROS and Bifrost, and follows the initial framework provide by Team ViGIR. Its classes and and internal data types are also written in a fashion that avoids imposing any *a priori* knowledge about the robot platform. As part of its data transmission role, it was also important to well construct the data pathways and cycles within the controller. The Valor ROS Controller also needed to run quickly and efficiently which led to a multithreaded implementation. This section will cover these topics in greater detail.

4.2.1 Threading Architecture

The Valor ROS Controller leverages multithreading to increase its speed and efficiency. The Valor ROS Controller has six primary threads shown by Figure 4.2, as well as several background threads created by the ROS Multithreaded spinner to handle ROS' overhead. Each major loop of the system operates within its own thread. By splitting the work of the controller between multiple threads, each part of the system can run at higher frequencies than if a single thread had to sequentially go through each loop. This also helps the Valor ROS Controller to better utilize the computing resources at hand. Not only does it allow the controller to take advantage of the multiple cores on the robot computers, is also lets the operating system release processing resources for threads that are entering a timed wait. For example, certain loops may hit timed waits that temporarily suspends execution of the thread, thus releasing the core that the thread was running on for other threads to use. By using a multithreaded design, the individual parts of the Valor ROS Controller run faster and the latency of ROS-Bifrost message passing is reduced.

Figure 4.2 also shows the high level execution path that the Valor ROS Controller uses. Upon startup, the Main thread spawns a thread to handle signal interrupts from the operating system. When the Main Thread instantiates the Valor Escher Controller class, it creates the Valor Escher Controller Update thread. This then spawns the Valor Escher Interface Update thread and Controller Bypass Update threads during initialization, and the Valor Escher

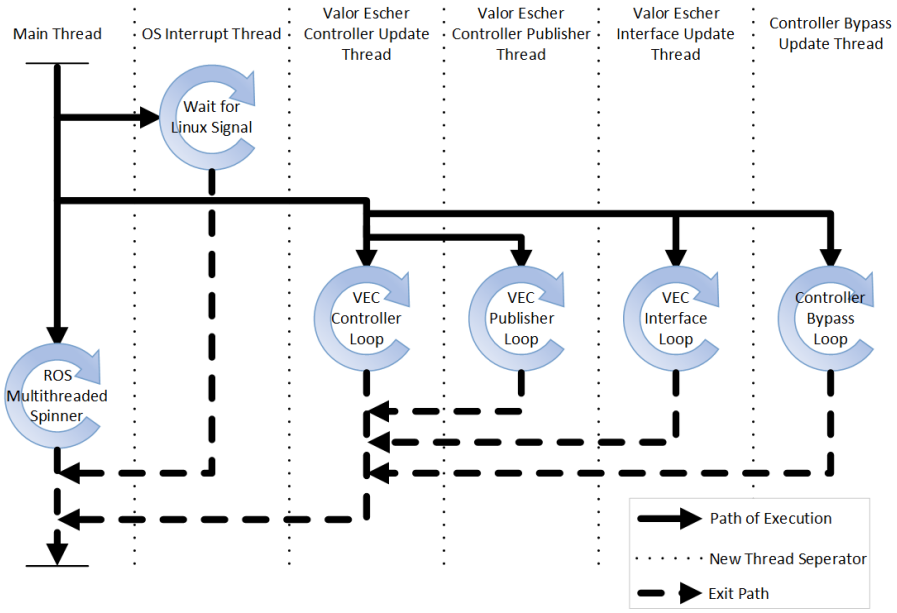

Figure 4.2: Threading architecture and process flow of the Valor ROS Controller.

Controller Publisher thread just before entering its own update loop. After initialization of the Valor Escher Controller, the Main thread then enters a ROS Multithreaded Spinner which creates several threads to handle ROS callbacks their associated overhead. Each of theses threads will contentedly run their own loops as long as no system interrupt commands are received. However, once an interrupt or terminate signal is caught by the OS Interrupt thread, then the following exit procedures occurs. The OS Interrupt thread calls the Valor Ecsher Controller's cleanup function, which changes run flags associated with each loop to false, thus prompting the other threads to exit. Further more, the OS Interrupt thread calls `ros::shutdown()`, which causes the ROS components of the controller to quit. As each thread ends, it joints back up with its spawning thread to ensure that there are no detached threads still running after the controller has shutdown.

The multithreaded nature of the Valor ROS Controller greatly increases its overall speed, however, it also introduces the problem of needing to pass data between each of its threads in manner that does not corrupt or invalidate that data. This is were Team ViGIR's Vigir RTBs show their strength. Since each buffer uses a combination of mutual exclusion locks and atomic pointer operations, there is no possibility of a thread reading data from one of the buffers while another thread is overwriting that data. This helps to prevent esoteric runtime errors caused by corrupted data from occurring.

4.2.2 Control Mode Mappings

The Valor ROS Controller operates in one of several states that determine the active joint level controllers and the states of the Motion System's finite state machines. In Team VALOR's system, as with Team ViGIR's, these states are called Control Modes and are part of a high level finite state machine overseen by a control mode ROS controller. In general, defining useful control modes depends on the robot's capabilities and the operational scenarios that it will encounter. However, since the Motion System was under development with the anticipation of integrating with Team ViGIR's software, the control modes chosen for the Valor ROS Controller resemble Team ViGIR's.

While defining control modes for the Valor ROS Controller is as simple as altering some configuration files, changing to another control mode is only effective if the Motion System changes to the corresponding internal states. Adding to the implementation difficulties was the need to maintain parts of ViGIR's system that referenced BDI's Atlas specific behaviors. This required creating a method of mapping the top level control modes of the Valor ROS Controller to BDI's behaviors, and then down to the Motion System's behaviors. Figure 4.3 shows the resulting mapping from control modes to the different sets of behavior states. Each black arrow indicates the equivalent mapping between states. Note that the `Upper Body Control` state in the BDI Behaviors column is a binary state that in BDI's system determines whether the arm joints may receive commands. It is used for a similar purpose in the Valor ROS Controller, however, state transitions require a two step process when it is changed. The Motion System's behavior interface only allows transitioning to a manipulation state from the corresponding non-manipulation state. When ever the control mode of the Valor ROS Controller changes, the Valor Escher Interface sends two state transition commands to the Motion System; the first to change the base state and the second the second to transition into either the manipulation or non-manipulation variants. The red lines in Figure 4.3 indicate these follow up commands.

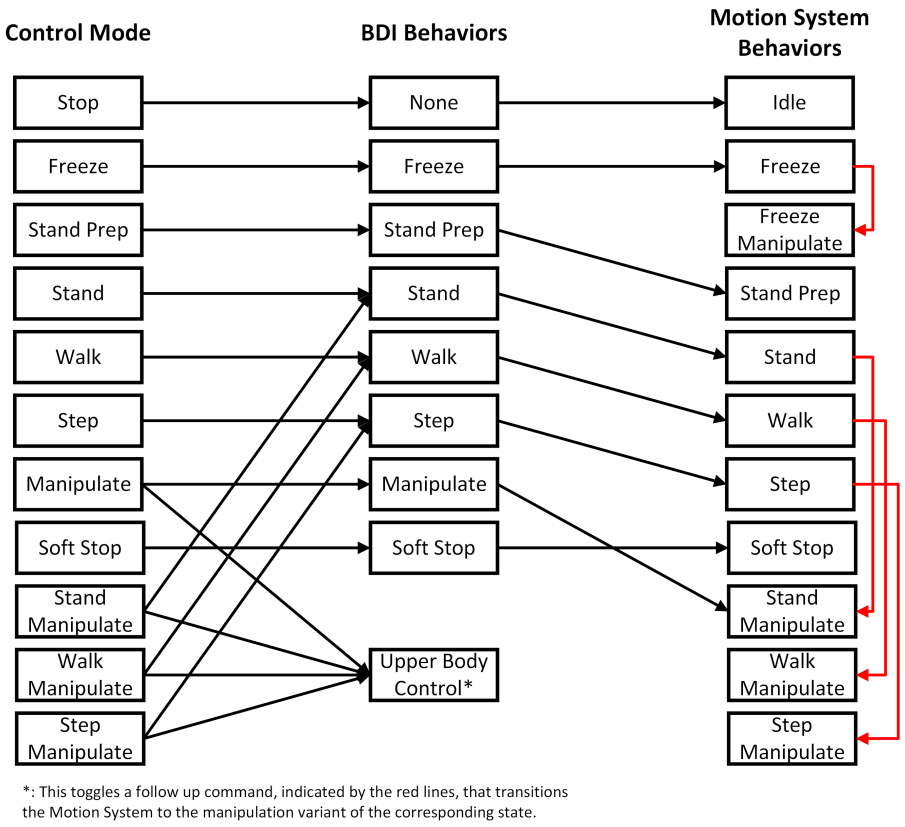

Figure 4.3: Mapping of Valor ROS Controller control modes to BDI Behavior States and Team VALOR's Motion System Behavior States.

4.3 Data Collection

4.3.1 Collection Methods

The data collection techniques in this thesis focus on recording various parts of the Valor ROS Controller and Motion System while the entire software stack was running. The primary areas of interest were: process statistics, Bifrost topics, and ROS topics. Each targeted area required different collection methods depending on its origin. Process data for the Valor ROS Controller was recorded by logging the output of a Linux tool for process monitor, while data about the controller and robot were recored by logging ROS and Bifrost messages. This section will discuss how the data from each area was recorded.

The process statistics for the Valor ROS Controller were measured using **top** [63], which is a standard Linux tool for displaying system performance and process monitoring. By default **top** operates by periodically updating a text display in a terminal window, however, this is not conducive to logging the results. As an alternative, **top** has a batch mode that can be used in a shell script. This mode simply returns a string containing the data that would have been displayed using the normal mode. To collect the process data over a sustained period of time, a shell script was written that would call the batch mode of **top** at a configurable frequency. The percent CPU and memory used by the Valor ROS Controller was then parsed out and written to a comma-separated values (CSV) file.

The ROS and Bifrost messages passed throughout the system contained made up the majority of the data collected. Theses messages shed the most light on the performance of the Valor ROS Controller and its interaction with the Motion Manager since they contain commands generated by each system and the response of the robot to those commands. The ROS messages were straight forward to log using the `rosbag` tool that is integrated into ROS. This tool records the messages of specified ROS topics and places them into log file that is called a bag file in ROS terminology. The Bifrost messages, however, required an additional step. Since there is no tool similar to `rosbag` in Bifrost, the Bifrost messages were converted into equivalent ROS messages using a separate node and stored in the same ROS bag file as the native ROS message. This also made it possible to convert the timestamps on the Bifrost messages from monotonic time to clock time, and thus ensuring that the ROS and Bifrost messages were time synchronized. The node accomplished this by calculating the difference between the OS wall clock and the monotonic clock upon stating the node, and adding that

to the Bifrost timestamps as an offset. The output ROS bag file from this process contained all of the relevant information pertaining to both the Valor ROS Controller and the Motion System.

Comparing the ROS and Bifrost messages from different points in the system also reveals how much latency there is in the system and between each component. For the purposes of measuring latency statistics, one can think of recording certain topics as *tapping* into the system. If two topics with equivalent data are chosen on either side of a component, then it is possible to estimate the latency added by that component. This is, in a sense, analogous to using a voltmeter or pressure taps to measure the difference in voltage or pressure caused by some object. For example, comparing messages containing joint angles from the Motion System with those that the Valor ROS Controller passes into ROS shows how much latency was add by the Valor ROS Controller while it was converting the data from Bifrost to ROS. As far as latency is concerned, the key points of interest are just before the Valor ROS Controller, between the Valor ROS Controller and the Motion System, between the Motion Manager and the Comms Manager, and just as commands are sent to hardware. Comparing messages at each of these locations gives the additional latency incurred by the Valor ROS Controller over simply communicating with the Motion System using Bifrost, as well as the total time it takes for a command to reach hardware.

The above discussion is summarized in Figure 4.4. This figure shows a high level view of what messages where collected and where they came from in the interaction between the Valor ROS Controller and the Motion System. Bifrost messages in and out of the Valor ROS Controller were direction recored into a ROS bag file using `rosbag`. While, Bifrost messages first went through a messages converted node before they were stored into the ROS bag. Each of the red lines shows where the latency between different parts of the system were measured from. The lines are mainly for illustrative purposes, since the actual beginning location for latency calculation depends on which direction the messages are flowing. The latencies calculated from the data collected will include some overhead from each of the IPC layers, which may vary with the path taken. However, this overhead should be small compared to the node latencies and are, in any case, a cost of doing business. The latency results will later be discussed in terms of latency delineations shown here.

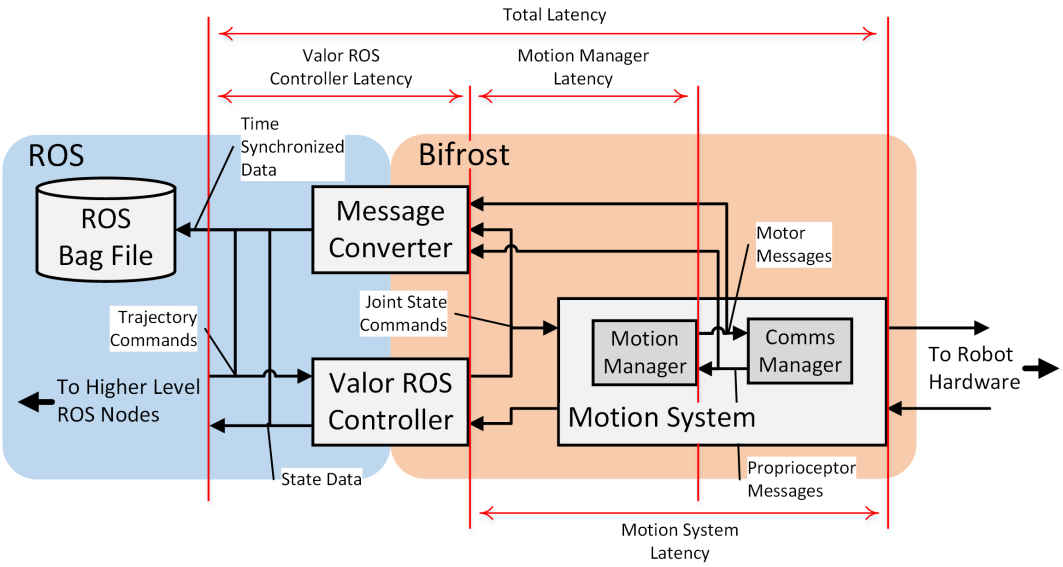

Figure 4.4: Data Collection Tap Points and Latency Intervals. This diagram shows the import Valor ROS Controller and the Motion System connections where ROS and Bifrost messages were recored from. It also shows the latency segments that will be used to describe the latency results.

4.3.2 Data Processing

In most cases, generating useful and informative results takes more than simply collecting raw data. This is especially true when the data collected covers so much of a system as represented by the messages passing through the Valor ROS Controller. A number of steps were necessary for processing and extracting the desired information from the data. These steps can be grouped into two categorizes: The steps it took to get the data into a workable format, and the steps necessary to analyze the data. The first category represents a processing pipeline while the second shows how to the desired information out of the collected data. This section will cover the processing pipeline for the ROS bag files collected, and the analysis steps necessary to calculate message latencies and end-effector trajectories.

Bag to CSV Processing Pipeline

The results shown in this thesis were done using MATLAB [64], however, it was first necessary to convert the recorded data into a format usable by the available version of MATLAB. The recorded ROS bag files contained messages with the relevant information about he Valor

ROS Controller and the Motion System. While ROS bags are useful for streaming raw data into ROS nodes, they are not very portable. There is a Robotics System Toolbox for MATLAB that can read bag files [65], but it is not available at Virginia Tech. The default version of MATLAB does however support CSV files. Even after this file conversion, the files would still need parsing to correct extract their data. Thus, it became necessary to create a pipeline to convert bag files in to CSV formatted files and read them into a MATLAB workspace.

This pipeline consisted of two parts, a ROS bag to CSV converter and a set of data extraction MATLAB functions.

Message Latency Calculations

There are two general ways of estimating the time delay, or latency, between to time-varying signals. The first, and most straight forward, is to plot both signals with respect to time and to measure the time difference between key points, such as peaks or zero-crossings. This works reasonably well if each signal is recorded with the same reference time, and the signals are well structured and without significant noise. The other method is to cross-correlate the signals to find the shift that causes the largest cross-correlation. This method only requires the signals themselves, and works well in general, even for noise and seemingly random signals. Both methods, however, require that the compared signals be reasonably similar to each other to have any meaning. What counts similar enough depends on the circumstances and is often up to interpretation. Although both methods would work for the majority of data recorded for this work, the cross-correlation method was used for this analysis because it is more general and can be applied on data sets with corrupted timestamps.

The cross-correlation method looks for the shift that causes the largest cross-correlation between two signals. For real valued signals, one way of defining cross-correlation begins with the Expectation Operator $E\{\}$. The expected value of a function $x(t)$ across all time (i.e., from $-\inf$ to inf) is defined as the mean of the function $\overline{x}$ as shown in Equation 4.1[1].

$$\overline{x} \equiv E\{x(t)\} = \lim_{T\to\infty}(\frac{1}{2T})\int_{-T}^{T} x(t)\,\mathrm{d}t \tag{4.1}$$

[1]The notation and derivation used for these equations follows the lecture notes of ME5564 [66] as taught by Dr. S. Southward.

In discrete time cases, the Expectation Operator becomes a summation instead of an integral as shown in Equation 4.2. Here, N is half the total number of samples while x_n represents and individual sample.

$$E\{x(t)\} = \lim_{N\to\infty}(\frac{1}{2N})\sum_{n=-N}^{N} x_n \tag{4.2}$$

We can now define the cross-correlation $R_{xy}(\tau)$ for two signals $x(t)$ and $y(t)$ as shown by equations Equation 4.3 and Equation 4.4 for continuous and discrete time respectively. τ and k represent how much the second signal is shifted by in time and number of samples respectively.

$$R_{xy}(\tau) \equiv E\{x(t) \star y(t+\tau)\} = \lim_{T\to\infty}(\frac{1}{2T})\int_{-T}^{T} x(t)y(t+\tau)\,\mathrm{d}t \tag{4.3}$$

$$R_{xy}(k) = \lim_{N\to\infty}(\frac{1}{2N})\sum_{n=-N}^{N} x_n y_{n-k} \tag{4.4}$$

The next step to estimating the time delay between $x(t)$ and $y(t)$ is to determine the shift that causes the maximum cross-correlation. This is represented by the arg max function. The delay in number of samples $Lag(k)$ is given by Equation 4.5. In practice, calculating the arg max invovles calculating $R_{xy}(k)$ for many values of k and returnging the value of k that corresponds to the largest value of $R_{xy}(k)$.

$$Lag(k) = \arg\max_{k} R_{xy}(k) \tag{4.5}$$

To get the time delay $Lag(t)$, simply divide by the mean sample rate $\overline{f_s}$ as shown in Equation 4.6.

$$Lag(t) = \frac{\arg\max_{k} R_{xy}(k)}{\overline{f_s}} \tag{4.6}$$

Where $\overline{f_s}$ is given by Equation 4.7. Here, Δt is the total duration of the signals and is given by $t_{last} - t_0$.

$$\overline{f_s} = \frac{N}{\Delta t} = \frac{N}{t_{last} - t_0} \tag{4.7}$$

In actuality, the calculations represtned above were handled by a few MATLAB functions and some scripting. MATLAB's `xcorr` function did the cross-correlation calculations while a combinations of the `max` function and vector index manipulation found the lags.

For the cross-correlations method to work properly, it is necessary for both signals to have the same sample rate, and, ideally, the same number of samples. However, this not always the case in a system like the Valor ROS Controller where different topics may be set to publish at different rates. This meant that when comparing two signals, the signal with the lower native sample rate had to be resampled using MATLAB's `resample` function. This ensured that the later calculations would come out properly.

4.3.3 Summary of Experiments

Process Characterization

The tests to characterize the Valor ROS Controller process statistics ran ESCHER both in simulation and on hardware in the full DRC Finals configuration. This included the OCS, and onboard planning and perception system. The robot was put into the Stand control mode and left standing for five minutes while the CPU and memory usages were logged. This setup provided data of the Valor ROS Controller operating under nominal conditions with its default controllers active and messages passing through all of its primary message pathways.

IPC Latency Tests

The only specific tests to determine latency in the system focused on measuring the transmission latencies of ROS and Bifrost. Determining the message pathway latencies required running the full system, thus data for this was recorded during the full system tests described below. Determining the minimum transmission latency for each IPC involved passing messages containing a signal double between threads. A node for each respective IPC would create a publisher thread and a subscriber thread. The publisher thread would

continuously publish messages with a single value that monotonically increased overtime, and record the time of publication. Each time the subscriber thread received a message, it would record the current time. Upon shutting down, the node would match the values of the sent and received message and subtract the corresponding sent times from the received times. Messages were sent at a frequency of ten hertz for at least two minutes.

End-Effector and Message Latency Tests

Generating data for end-effector trajectories and determining message pathway latencies required running the full DRC Finals configuration of the robot. To reconstruct the end-effector trajectories, joint angles from the right arm were recorded while it was executing a set of predefined configurations and arbitrarily planned trajectories. The predefined configurations came directly from the OCS and included arm positions that ranged from raising the arm straight out to the side to placing the end-effector in front of the face. For the arbitrary configurations, MoveIt! generated trajectories to place the right arm in arbitrary configurations, both in front of and behind the robot. In addition to these, several tests were run with the robot standing or moving its neck joints. During all of these tests, messages from each segment in the message pathways of the system were recorded for determining the message pathway latencies.

Chapter 5

Numerical and Experimental Results

This chapter is split into four sections that cover three areas directly related to the Valor ROS Controller and one area dependent on the entire system. The first three sections will look at the process statistics, message pathway latency, and setpoint tracking of the Valor ROS Controller. These sections will show that the Valor ROS Controller fulfills its role as an interface to the Motion System by promptly converting messages between ROS and Bifrost while utilizing an acceptable level of computational resources. The final section will then describe the systemic affects of whole-body motion on tracking end-effector poses. This section quantifies the phenomenon of the robot translating its torso pose as an arm moves through a trajectory, which has previously only been qualitatively observed.

5.1 Valor ROS Controller Process Statistics

The Valor ROS Controller is potentially a resource intensive process. In addition to running a complement of controllers, it must also constantly convert messages between the ROS and Bifrost IPCs. The central place in the overall software stack that the Valor ROS Controller occupies, gives it considerable preference for computational resources. However, it is still only one process in a menagerie of competing processes, many of which are also important to the core functionality of the overall system. Thus, it is important to ensure that Valor ROS Controller does not utilize an inordinate amount of computing resources.

As discussed in Collection Methods (subsection 4.3.1), two key process statistics, percent

CPU usage and percent memory usage, where collected using the Linux program **top**. Table 5.1 shows the results for the Valor ROS Controller while running in simulation and on hardware. For a multi-core processor, the **top** program reports the percent CPU usage as a percentage normalized by the number of cores present on the processor. It bases percent memory usage on the amount of memory utilized out of the total usable memory in the computer. The simulation results were collected on a computer with 12 cores and 32 Gigabytes of RAM while the computer on ESCHER running the Valor ROS Controller had eight cores and 16 Gigabytes of RAM.

Table 5.1: Basic process statistics for the Valor ROS Controller. Simulation and hardware percent CPU and memory usage for the Valor Escher Controller process.

Test Condition	**Simulation**	**Hardware**
%CPU		
Mean	36	53
STD	8	10
Max.	100	113
Min.	13	15
%Memory		
Avg.	0.3	0.6

A *prima facie* inspections of Table 5.1 suggests that the Valor ROS Controller does in fact consume an excessive portion of the processing resources. It utilizes over a third of a core in both simulation and half of a core on hardware. However, these average values give only a partial picture since they do not show how the Valor ROS Controller uses its processing resources or how it compares with other processes in the system. The multi-threaded nature of the Valor ROS Controller means that its load is spread across multiple cores on the processor. Even though it is cumulatively using half of one of the processor cores, each thread is using much smaller portions of individual cores. In comparison to other processes, the Valor ROS Controller does not use an unreasonable amount of processing resources. For example, the Motion System uses over one and a half cores by itself. In addition, on hardware, the Valor ROS Controller runs on the computer that is solely dedicated to the Motion System, which means that there is an abundance of processing overhead.

The standard deviation, maximum, and minimum values in Table 5.1 indicate the volatility of the Valor ROS Controller's CPU usage during operation. This stems from the multiple threads that are part of the Valor ROS Controller design. Not every thread is constantly active, thus the overall CPU usage varies as threads idle, such as when waiting for new

messages or while sleeping as part of a control loop, at different rates. This variability has the perhaps unintuitive benefit of letting the operating system better dynamically schedule other processes and shift process load as different programs require more CPU time [67]. Since a subset of threads within the Valor ROS Controller are at any particular moment entering an idle state, the operating system may place alternative process on the vacated cores. As the threads become active again, the operating system may assign them to less utilized cores in order to more evenly distribute the processing load between cores. The volatility in the CPU usage of a program with a wide variety of responsibilities such as the Valor ROS Controller also indicates that it is only using resources as it needs them. A constant CPU load suggest that a program is constantly looping through the same set of instructions, which may be desirable or an indication that the program is deadlocking.

Given its roll and criticality in the overall system, the Valor ROS Controller uses a reasonable and acceptable level of computational resources. It uses much less CPU power than the Motion System or the nodes associated with the OCS. The Motion System consumes an entire core to run the whole-body controller and additional half of a core to communicate with the hardware components on the robot, while the OCS nodes take up nearly six cores by themselves. In regards to memory usage, the Valor ROS Controller's impact is practically negligible. All of its variables and stored data structures consume less than 100 megabytes (using hardware values: $0.006 * 16GiB * 1024\frac{MiB}{GiB} = 98.3MiB$). These process specific results demonstrate that Valor ROS Controller satisfies the requirement that it not consume an inordinate amount of computational resources.

5.2 Messages Latencies

As the primary interface between the ROS and Bifrost IPCs, it is critically important that Valor ROS Controller passes messages with as little additional latency as possible. By comparing messages sent from sequential stages of the system, as discussed in Data Collection (section 4.3), it is possible to determine the accumulated latency incurred by the Valor ROS Controller. However, latency values for the Valor ROS Controller are uninformative without a baseline reference to judge them by and to put them into context. Thus, this section begins by reviewing ROS and Bifrost's message passing performance in simple scenarios. With these initial metrics in mind, the latency statistics for the Valor ROS Controller will then conclude this section.

5.2.1 Baseline Message Latencies for ROS and Bifrost

As IPC protocols, both ROS and Bifrost should promptly pass messages to their destinations. Since the time taken by a message in transport is lost time that could have been spent processing data, an ideal IPC would pass data instantaneously, however, real systems must currently settle for as fast as possible. Both ROS and Bifrost rely on TCP sockets as the basis for their message passing protocols. While not necessarily the fastest method for IPC (cite - If I can ever find it again), it is robust and yields respectable results. The following data represents baseline latencies for ROS and Bifrost.

One of the simplest IPC use cases is sending data from one thread to another. Figure 5.1 shows message latency over time for both ROS and Bifrost while sending an individual double-precision value from a publisher thread to a receiver thread. The black solid and dashed lines are trend lines associated with the ROS and Bifrost datasets respectively. The latencies of both IPCs overlap considerably, but ROS consistently produces slightly lower latencies than Bifrost. Bifrost's performance is quite good considering that one person developed it over the span of about a month.

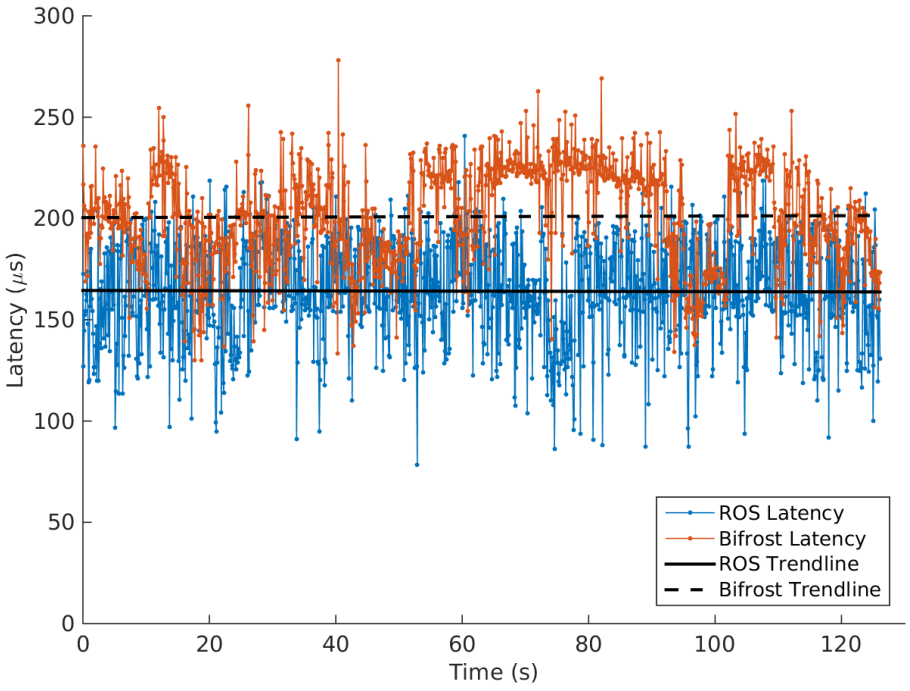

Figure 5.1: ROS and Bifrost messages latencies sampled over time.

Table 5.2 presents the statistical properties associated with the data in Figure 5.1. Although

there is significant overlapping in the figure above, the mean ROS latency is 36.8 μs, or about 18%, lower than the mean Bifrost latency. The standard deviations of both datasets are practically equivalent.

Table 5.2: Calculated mean and standard deviation of baseline message latency for ROS and Bifrost IPCs.

	IPC	
	ROS	**Bifrost**
Mean (μs)	165.5	202.3
STD (μs)	25.5	24.6

These results by themselves show the relative performance of ROS and Bifrost in a context relevant to Team VALOR. To verify results and to the better insight, it is important to compare the results with outside sources if possible. Dantam *et al.* compared ROS with several other IPC implementations while their own IPC [41]. The ROS results presented above match the latencies measured by Dantam *et al.* for ROS using TCP. This validates the measurement method used to determine the latency incurred by ROS message passing and puts Bifrost into the greater context of IPCs in general.

The above data represents lowest latency that can be expected from either ROS or Bifrost. Since general applications involve numerous topics and the preponderance of messages sent by either system are more complex than a single value, actual message passing latencies experienced by any deployed system will be higher. These results, however, present a baseline to help determine how much of the latency observed in a system is from message passing or computation.

(Possible paragraph about affect of message size.) Another aspect that affects TCP performance is message size (cite - I know I saw a reference somewhere). Each time a message is sent using a TCP based IPC protocol, the contents of the message are encoded into TCP packets. Larger message require more TCP packets to transmit, and thus it takes longer to process large messages.

5.2.2 Message Pathway Latencies of the Valor ROS Controller

As the primary interface to the Motion System, the Valor ROS Controller attempts to convert messages between ROS and Bifrost as rapidly as possible, however it inevitably adds

additional latency along each message pathway. Any additional latency in the data streaming to and from time sensitive systems, such as state estimators and high level controllers, has the potential to degrade the quality of those systems. Quantifying upper bounds on the latencies incurred by the Valor ROS Controller will help to expose one of the of introducing an IPC conversion layer. If these are modest in comparison to the overall data transmission times, then introducing the Valor ROS Controller has not substantially negatively impacted the system as a whole.

There are three primary message pathways of interest relate to how quickly the robot responds to joint level commands, how quickly proprioception data is relayed to higher level systems, and how quickly the robot changes control state. The first of these pathways for joint level commands takes reference trajectory commands from the higher level planners and interfaces, and passes various setpoints down through the Valor ROS Controller, Motion System, and on to the individual motor controllers. This requires the most message conversions and computational work of any of the message pathways. Disseminating proprioception data from the robot to the rest of the ROS system is the simplest pathway in that it only requires one message conversion. The state transition pathway comprises of two steps, transmitting a state transition command from a higher level system to the Motion System, and relaying the reported change in state back up to the higher level system. Each of these pathways heavily rely on both the Valor Escher Controller and the Motion System.

To analyze each message pathway, it is most desirable to split the path into a series of segments and nodes. Each segment would represent message transmission time through an IPC layer and each node would represent the computational time required to generate the next message in the pathway. However, the Valor ROS Controller and the systems it interacts with are intricate programs, thus it was not always possible to cleanly delineate the time taken to send a message through an IPC layer from the computational time associated with generating that message. Unlike the simple situations presented in Baseline Message Latencies for ROS and Bifrost (subsection 5.2.1), calculating the time it takes a message to reach its destination based on the time difference between when the message was sent and received would require drastically rewriting the underlying code of the system. One relatively simple way to overcome this limitation is to expand the notion of latency to include the time it takes to convert from one message type to another. This pathway latency is calculable by comparing message streams that contain equivalent data at different locations in a message pathway using the methods described in Message Latency Calculations (section 4.3.2). This

has the added benefit of providing a means of calculating the upper bounds on the message conversion times by subtracting the lower bound on the originating IPC transmission time from the pathway latency.

Figure 5.2 displays the average pathway latencies recorded for the message conversions related to the pathways of interest in the Team VALOR system. Each bar represents the average time delay from when a message was sent to when the following message in the pathway was sent. Thus, the latency times recorded include the transmission and message processing times. The labels below each bar indicate the messages involved and the direction in which messages were processed in the message pathway. The meaning of these labels will be discussed in greater detail later on. The red error bars mark one standard deviation of uncertainty in the recorded values. The primary system where each conversion takes place is designated by color.

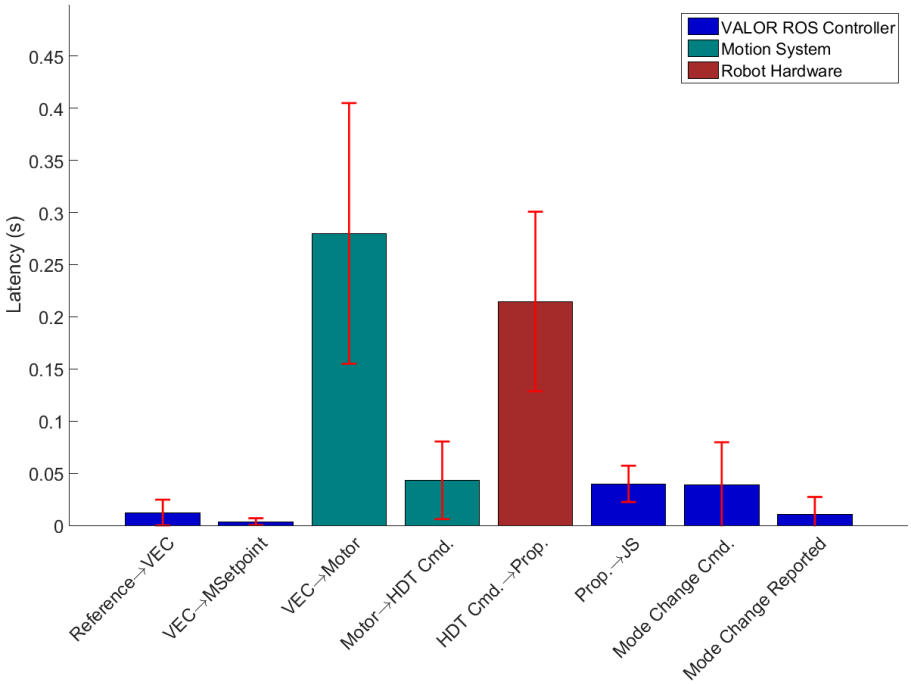

Figure 5.2: Average message latency between key points in the Valor ROS Controller and Motion System. The red error bars represent one standard deviation in the recorded data.

Each of the bar labels in Figure 5.2 represent steps along one of the message pathways of interest. Starting on the left, `Reference→VEC` is the first step in the joint command pathway. This bar represents the delay from when a reference command from a higher level system is sent to when the controllers within the Valor Escher Controller begin responding.

The `VEC→MSetpoint` bar indicates the time it takes for control values calculated by the controllers of the Valor Escher Controller to be sent out as setpoint commands to the Motion System. Next, the `VEC→Motor` bar is the time difference between the internal control values of the Valor Escher Controller and the motor commands computed by the Motion System. This is the step where commands go through the Whole-Body Controller, and thus is the most computationally intensive step. The final step in teh joint command pathway, represented by `Motor→HDT Cmd.`, is the time it takes motor from the Motion System commands to be sent out as setpoints to the low level motor controllers. In this case, the final destination was the HDT arm motor controllers. The next bar to the right, `HDT Cmd.→Prop.` represents the time delay between when commands were sent to the individual motors to when the Motion System received proprioception values corresponding to those commands. This bar most closes represents the general plant response of the robot. The `Prop.→JS` bar indicates the delay incurred by converting proprioception data from Bifrost to ROS joint state messages. The last two bars on the right, `Mode Change Cmd.` and `Mode Change Reported`, are part of the state transition pathway and represent the time it takes to transmit a mode change command to the Motion System and the time it takes to confirm the mode change respectively. A table describing each of the above abbreviations can be found in appendix [Message Abbreviations][].

The latency results presented show time associated with each individual conversion and processing segment of the message pathways. Summing these results for each respective pathway produces average estimates of the total time it takes for a particular message to reach its destination. Figure 5.3 displays these results for the three message pathways of interest. As before, the red lines indicate one standard deviation in the estimate. Each step in the pathways are separated into distinctive colors as shown by the legend.

5.3 Individual Joint Control Signals

As a controller, the Valor ROS Controller primarily focuses on producing persistent setpoints for individual joints. In this role, it converts very discrete piece-wise linear reference commands from higher level systems, such as MoveIt!, into smooth control signals for the Motion System. This ensures that there are no discrete jumps in the setpoints sent to the Motion System, which, as demonstrated by changing ESCHER's control mode to Stand while the arms are not in there nominal positions, cause potentially unsafe joint velocities.

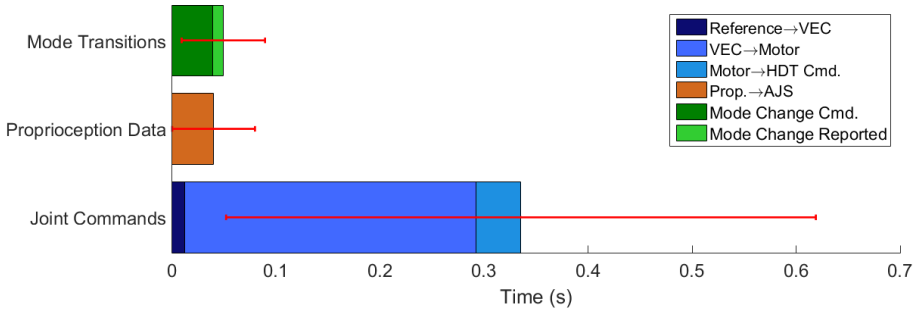

Figure 5.3: Overall transmission and computation time for the mode transitions, proprioception data, and joint command message pathways. Each color indicates a specific message conversion and processing segment in the pathway.

Guaranteeing reach their desired setpoints remains the responsibility of the Motion System since the actual admissible control values given the constrains on the Whole-Body Controller may preclude the setpoints specified by other systems.

5.3.1 Trajectory Controller Tracking

While it is not the Valor ROS Controller's responsibility to close the control loop in Team VALOR's DRC configuration, it is still informative to explore the control signals to individual joints in order to assess the tracking quality of each system downstream from the reference command. In the ideal scenario, the commands generated by the Motion System would closely track those sent by the Valor ROS Controller, and the proprioception data would show that each joint reaches the its desired setpoint. This section will look at the control signals associated with joints from the neck and right arm. These are representative of the control interactions between the Valor ROS Controller and the Motion System, and show that the Valor ROS Controller produces adequate control signals for the Motion System.

?? shows the position and velocity control signals and the proprioception data for the neck yaw joint on ESCHER. The left graph presents the position reference commands, the internal Valor Escher Controller control commands, the Motion System's motor commands, and the joint angles reported by the joint encoder. The right axis compares the velocity commands from the Valor Escher Controller and the Motion System with the joint velocities recorded by the proprioception sensors. Note that no reference values are given because the reference command in this case did not specify target velocities.

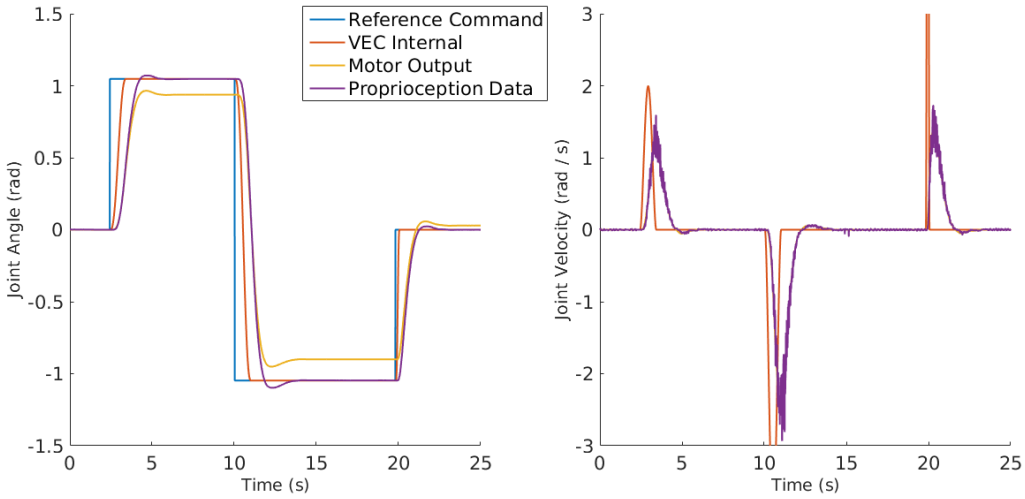

Figure 5.4: Comparison of control signals and proprioception data for the neck yaw motor. (Left) Position commands and reported position over time, (Right) velocity commands and reported velocity over time.

The arm joints receive the most direct commands from higher level systems, since the Motion System handles computing commands to the leg actuators while standing and walking. Thus it is important to evaluate the tracking performance of the various systems as they command the arm joints. In simulation, the arm joints achieve their reference angles and track the commanded positions and velocities from the Valor ROS Controller nearly perfectly. However, Figure 5.5 shows that on hardware not all of the arm joints reach their commanded positions. During the time period shown in the figure, the right arm moved through three trajectories that placed the arm up in front, the elbow bent 90 degrees, and offset to the right forward of the robot. Between each trajectory, the arm was reset to point nearly straight down the side of the robot. The blue lines in Figure 5.5 represent the final reference position for each joint while the purple lines indicate the actual positions recorded by the joint encoders. The red and yellow lines are the control signals from the Valor ROS Controller and Motion System respectively. For several of the joints, there are consistent errors between the commanded and reported positions. More over, the Motion System itself appears to produce position commands with a steady-state offset. Of particular note is the right wrist yaw, which gave Team VALOR difficulty during the Door task at the DRC Finals [42].

Figure 5.6 displays the corresponding arm joint angular velocity commands and sensor data to Figure 5.5. In this case, all of the joints except for the right wrist yaw follow the commands

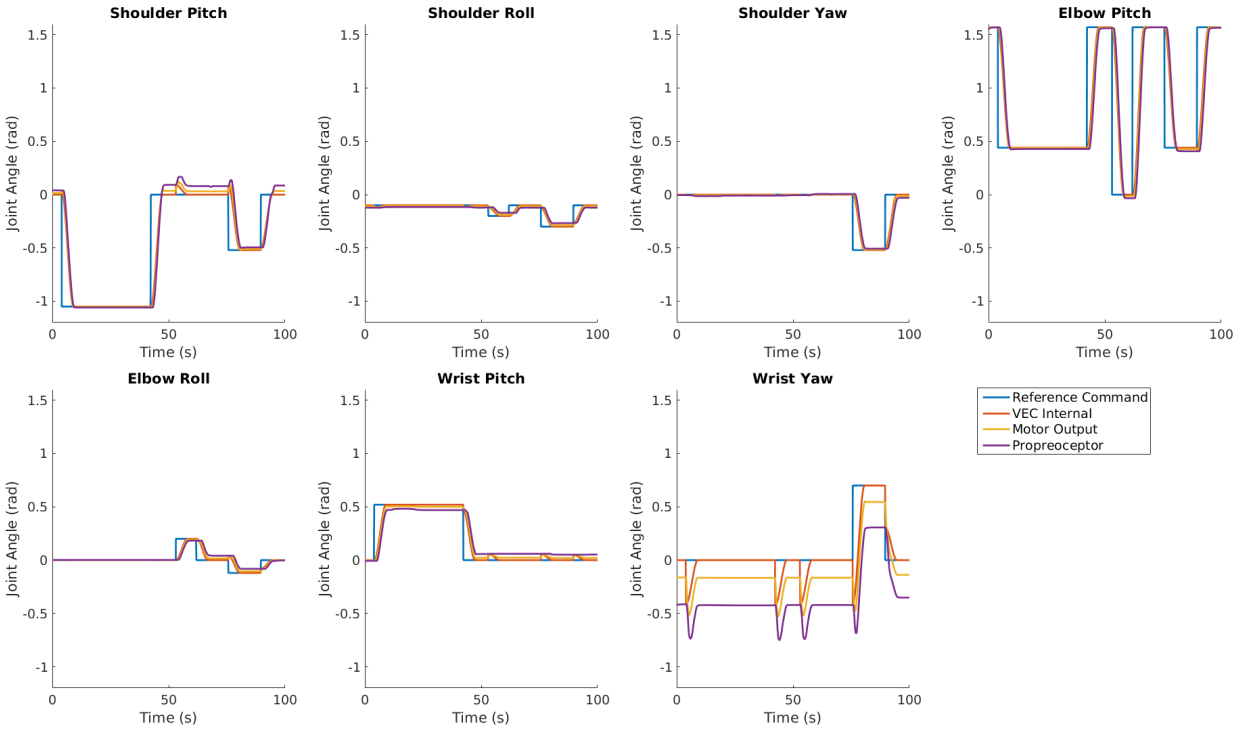

Figure 5.5: Individual angle verses time plots for joints in the right arm of ESCHER.

from the Valor ROS Controller, which the Motion System treats as reference values. The Motion System is sending remarkable high and constantly positive velocity commands to the wrist yaw motor without any apparent affect on the joint's position. This tends to suggest that some other factor in the wrist yaw motor is causing it to fail to follow the control signals it receives. Overall, however, the Motion System's joint controllers for the arms display good velocity tracking.

There are several factors that may be contributing to the errors observed on hardware, these include improperly tuned gains, backlash in the motors, and lack of gravity compensation. In manipulation modes, the Motion System uses PD controllers to determine position and velocity setpoints. As shown by Tomei [68], PD controllers are sufficient to control manipulators even in the presence of elastic elements such as the harmonic drives in the HDT arms. However, in the rush to get everything ready for the DRC Finals, Team VALOR neglected to rigorously tune the PD gains for the arms used by the Motion System. Another issue is the substantial amount of backlash inherent in the HDT arms used on ESCHER. With the arm at nearly full extension, Wittenstein [69] observed approximately 7 cm of play in the end-effector due to cumulative backlash in the motors. Lack of gravity compensation

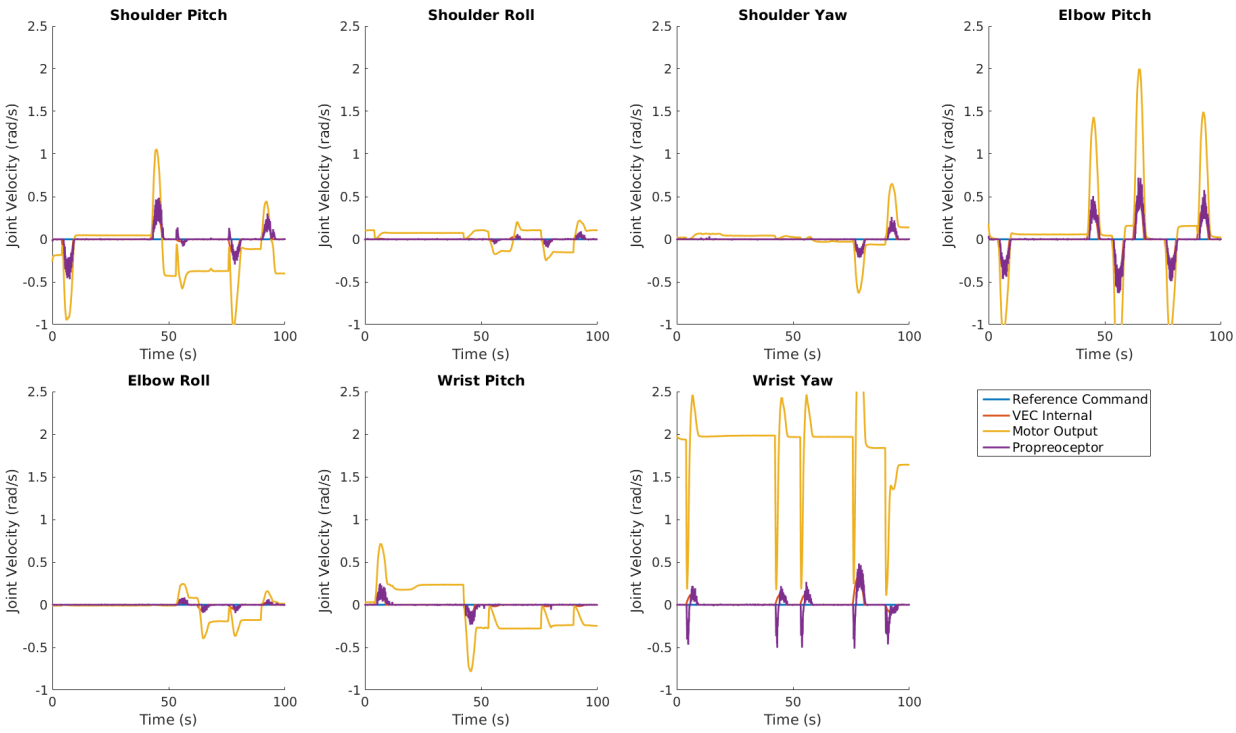

Figure 5.6: Individual angular velocity verses time plots for joints in the right arm of ESCHER.

in the arm control signals from the Motion System also contributed to the steady-state error observed in the pitch joints. Hsia [70] shows that gravity compensated PD controllers degrade under uncertainty in link mass values, thus it follows that completely uncompensated PD control signals will not drive gravity affected actuators to their intended positions.

5.4 End-Effector Trajectories

During manipulation tasks, precise and predictable positioning of the active end-effector is vitally important. The difference between a successful grasp and failure is often on the order of single centimeters. In order to simplify implementation, the manipulation planners and controllers implemented on ESCHER assume a fixed base frame for the manipulators. In reality, humanoid whole-body controlled robots such as ESCHER have a floating frame of reference. This disconnect between a planner that assumes a stationary frame of reference and the actual robot introduces stead-state errors between the desired and final end-effector poses. This section will look at these errors in Team VALOR's system by first verifying that the robot achieves the commanded end-effector pose under the supposition of a fixed base frame. It will then reevaluate these results after incorporating the changes in torso pose, which is effectively rigidly connected to the base link of each arm, as one of the arms moves through a trajectory.

5.4.1 Fixed Base

When discussing end-effector trajectories on a whole-body controlled humanoid robot, it is instructive to first assume that the base of the robot is fixed in position. That is, that the base link of the manipulator upon which the end-effector is mounted remains stationary in space. This was the underlying assumption of MoveIt! as utilized by Team VALOR and is commonly assumed when sending a robotic arm to a predefined configuration. This allows for verifying that manipulator trajectories are being executed correctly and for comparing the affects of whole-body control on resulting end-effector and joint poses. All of the data presented in this section was taken on ESCHER while the whole-body control algorithm was running, however, the forward kinematic solver used to reconstruct this data assumed a fixed base link for the robot.

Figure 5.7 shows reconstructed simulation data of the right arm of ESCHER moving through a series of command joint configurations. The blue line represents the desired joint configuration, with each dot corresponding to a joint and each line segment a link in the chain. The red line shows the current arm configuration while the small red, green, and blue axis represents current end-effector pose with the red, green, and blue lines corresponding to the x, y, and z axes respectively. The curved gray line indicates the path followed by the end-effector as the arm executes each trajectory. This figure confirms that under a fixed base assumption, the arm does indeed appear to reach the command joint configuration and end-effector pose.

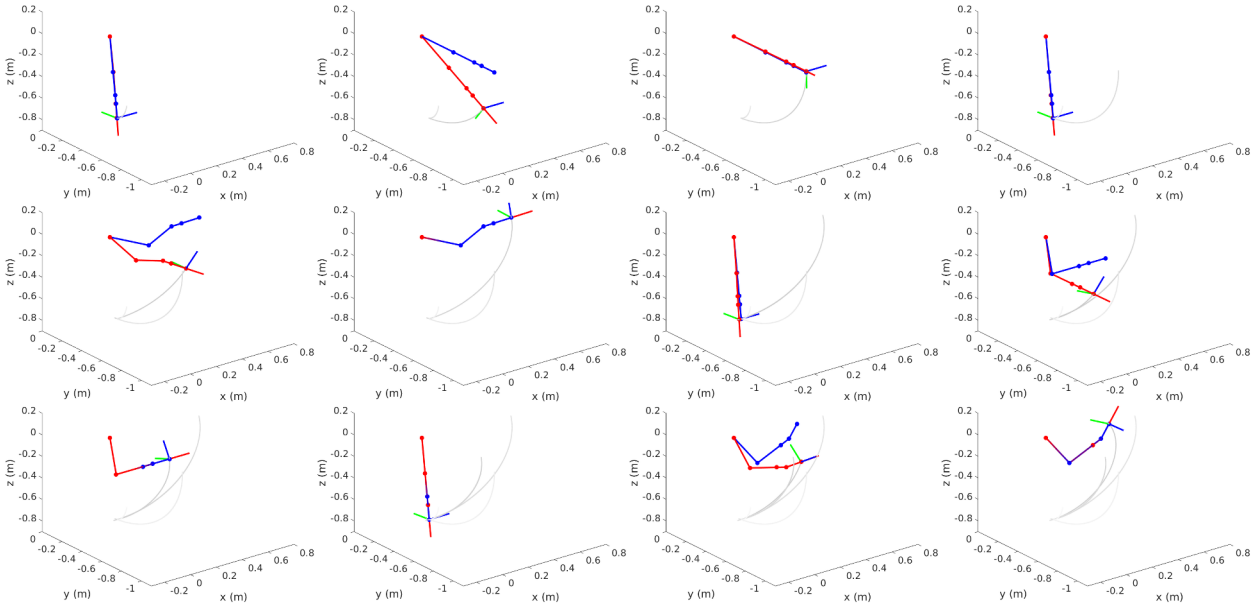

Figure 5.7: Simulation image series (from MATLAB) of right HDT arm moving through a trajectory. This shows the relative positions of each joint and end-effector pose assuming a fixed base.

Simulation End-Effector Errors

During manipulation tasks it is critically important that the end-effector achieves its command pose. In simulation, this should be trivial for a well functioning system. Figure 5.8 demonstrates that this is the case for the Valor ROS Controller. This figure shows the end-effector error, broken out into its position (top) and orientation components (bottom), between the reference pose and the commands from the Valor ROS Controller. The data in this figure corresponds to the images shown in Figure 5.7. Two things to note from Figure 5.8

are in many cases the pose error components converge at the same time and that there are hystereses in the errors. (EXPLENATION)

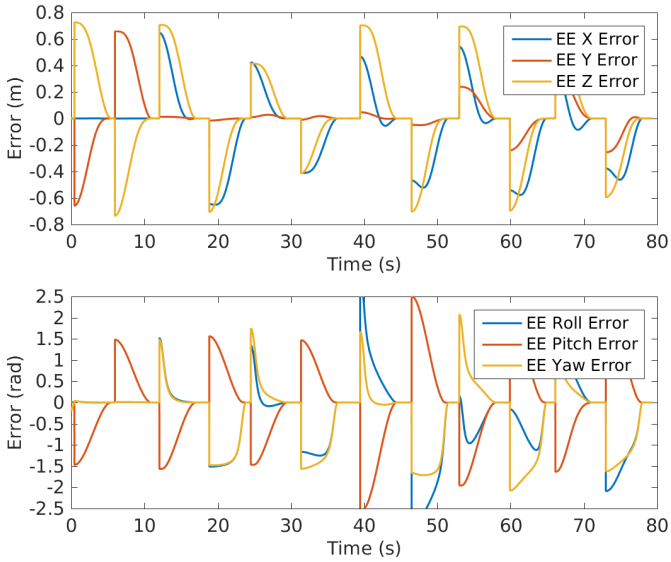

Figure 5.8: This figure shows the position errors (top) and orientation (bottom) errors of the commanded end-effector pose relative to the desired pose as the right arm moves through a series of predefined configurations.

Commands from the Valor ROS Controller do not fully portray how the entire system responds to arm trajectory commands. To get an idea of how the robot responds, it is necessary to look at the proprioception data from the robot. Figure 5.9 displays the end-effector component errors between the commands from the Valor ROS Controller and the current end-effector pose computed from joint angles. Again, the top graph shows the position errors and the bottom graph shows the orientation errors. As one might expect, since the plant of a robotic system separates sensor data from commands, the difference between the commanded and actual end-effector poses builds up for a period of time before it converges to zero again. However, the small maximum error between the commanded and sensed poses, on the order of 2 cm, relative to errors between the reference setpoints and commands, indicates that the time constant of the Motion System and robot is less than the Valor ROS Controller time constant.

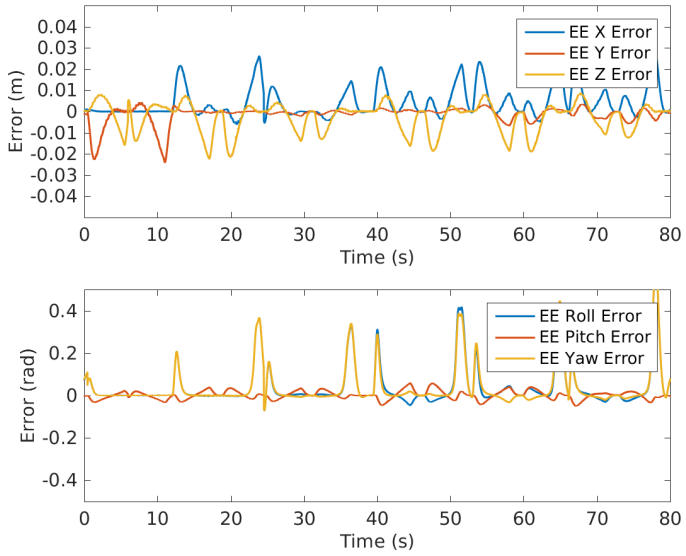

Figure 5.9: End-effector error between the pose commanded by the Valor ROS Controller and the posed measured by the simulated proprioception data.

5.4.2 Floating Base

Whole-body controlled humanoid robots such as ESCHER generally do not have a fixed reference frame from which their manipulators can base trajectories on. The compliance in ESCHER's legs, in conjunction with the whole-body control methodology, results in an under-constrained, or floating, base frame for the arms. In addition, the current implementation of the Whole-Body Controller and its Motion System interfaces does not prioritize maintaining a specific end-effector pose given a set of joint configurations. This means that, as an arm moves through a trajectory to a desired final configuration, the rest of the robot adjusts to maintain the current center of mass position. Thus, if one of the arms moves up and forwards by rotating the should pitch motor, the legs will shift the torso of the robot down and backwards in order to compensate change in the center of mass caused by moving the arm. Figure 5.10 demonstrates this phenomenon using simulation data. As before, the blue indicates the desired arm configuration in Cartesian space while the red line represents arm position computed from the sensor data. This figure shows the same data as Figure 5.7, however, it has been transformed to reflect the changes in torso pose over time.

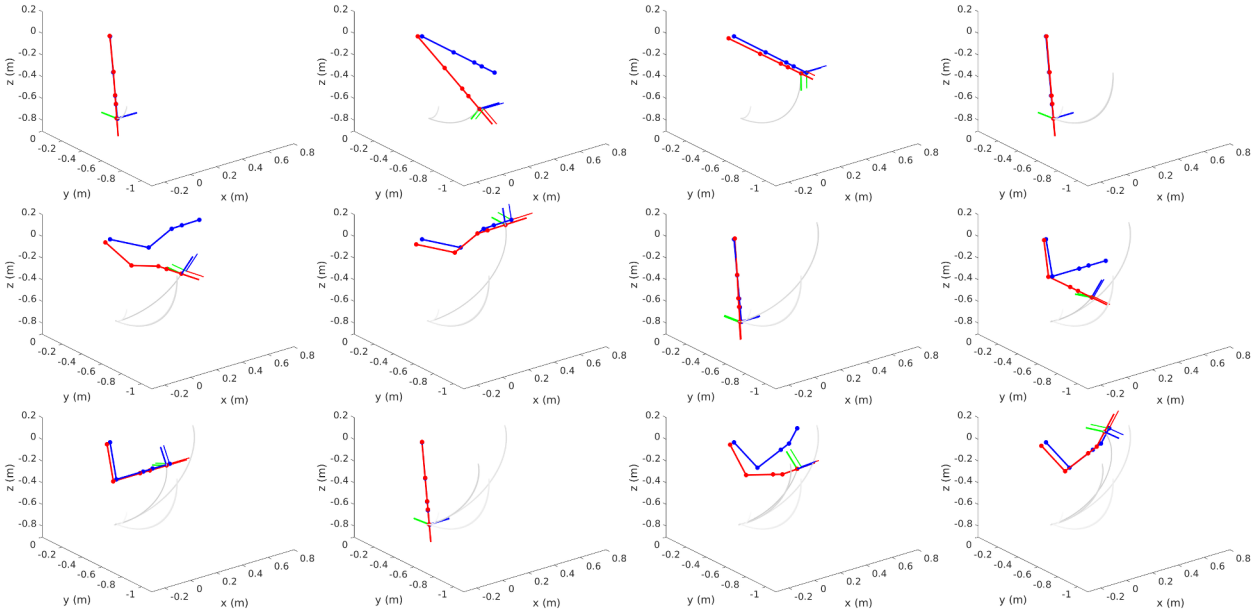

Figure 5.10: Simulation Image series (from MATLAB) of right HDT arm moving through a trajectory. This shows the relative positions of each joint and end-effector pose as the whole-body controller adjust the floating base of the robot to maintain its original CoM.

Simulation End-Effector Errors

Unlike in the fixed case, Figure 5.10 clearly shows a steady-state error between the end-effector pose corresponding to the commanded joint configuration and the actual end-effect pose achieved. However, as shown by Figure 5.11, the final end-effector position errors in simulation are small compared to the distance traveled by the end-effector to reach the goal pose. Even with changes in torso pose, the orientation errors still converge to zero. This is because the Whole-Body Controller puts high weights on maintaining the torso orientation. Thus, the torso only translates through space.

Although Figure 5.11 appears to belie the magnitude end-effector error caused by the whole-body control algorithm. Figure 5.12 shows that the final end-effector error is indeed significant. This figure compares the proprioceptive data under the fixed base assumption with the same data after taken into account the floating base. The top graph shows the floating base end-effector position components subtracted from the end-effector position calculated from the fixed base data. The bottom graph displays the absolute error magnitude of the component differences, that is, the magnitude of the distance between the end-effector positions according to fixed or floating base frame. This set of relatively standard arm

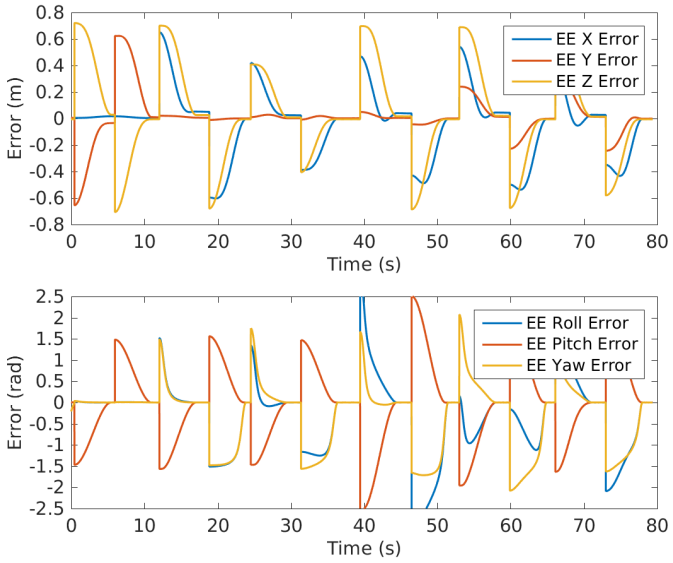

Figure 5.11: Graphs of the end-effector pose position errors (top) and orientation errors (bottom) as the right arm moves through a series of predefined trajectories. This is based on simulation data and reflects ESCHER's floating base frame.

configurations results in absolute differences between the desired end-effector poses and actual end-effector poses of approximately 3–6 cm. Differences in the x, y, and z components routinely run over 2 cm and as high as 5 cm.

The errors in end-effector position incurred by the Whole-Body Controller may seem modest, but they have the same order of magnitude as many grippers and objects for manipulation. For example, the HDT grippers used on ESCHER at the DRC Finals are 10 cm and have 7 cm long fingers while the door handle on the DRC door was 7 cm long and extended 3 cm from the door. More over, degenerative situations can occur since these error are not compensated for by any of the planners or controllers in the current system. Since the torso of the robot moves in the opposite direction of the end-effector, to maintain the position of the center of mass, extending the arm towards an object may cause the object to exit the available workspace of the robot. This happened several times during practice for the DRC Finals. Placing the robot so that the manipulation target was reachable by the OCS model, which assumed a fixed base, would result in having to manually adjust the torso pose to bring the target back into the workspace of the robot.

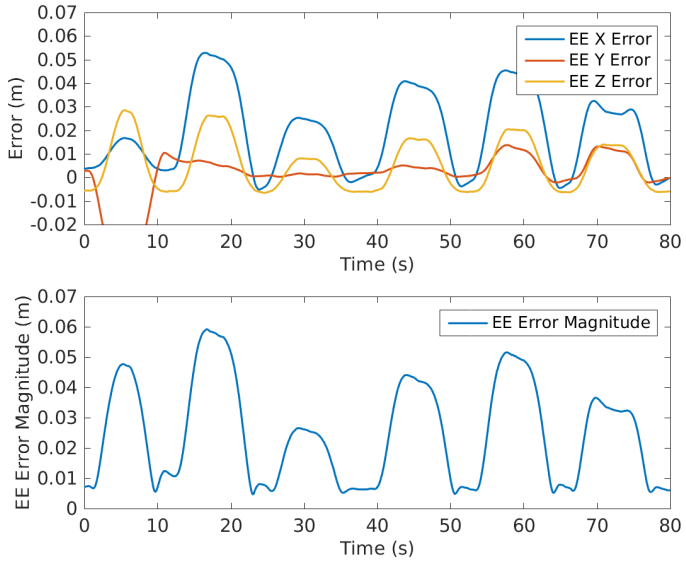

Figure 5.12: Differences between the end-effector pose calculated from proprioceptive data in simulation based on a fixed or floating base frame. (Top) Floating base frame end-effector positions subtracted from fixed base frame end-effector positions. (Bottom) Magnitude of component differences.

Hardware End-Effector Errors

The simulation results presented so far cleanly show the affects of the Whole-Body Controller shifting the torso pose of the robot as an arm moves to a new configuration. In reality, the unmodeled complex interactions between physical components tend to conflate various influences. However, Figure 5.13, demonstrates that the torso does indeed translate as the arm moves from its initial position, even as hardware peculiarities contribute additional errors to the final end-effector pose. As previously, the blue line indicates the desired arm configuration assuming a stationary base frame, while the red line describes the current arm configuration as reconstructed from proprioception data. This figure is based on data taken with ESCHER while configured for the DRC Finals.

The error in the end-effector pose between the reference joint configuration and the configuration commanded by the Valor ROS Controller follows the same pattern as seen in Figure 5.11. On hardware, the commands from the Valor ROS Controller converge towards the reference, but there remains a steady-state error from the unaccounted for shifts in torso pose. However, unlike the previously shown simulations results, there are large persistent errors between the commanded end-effector pose and the actual pose recorded by the joint

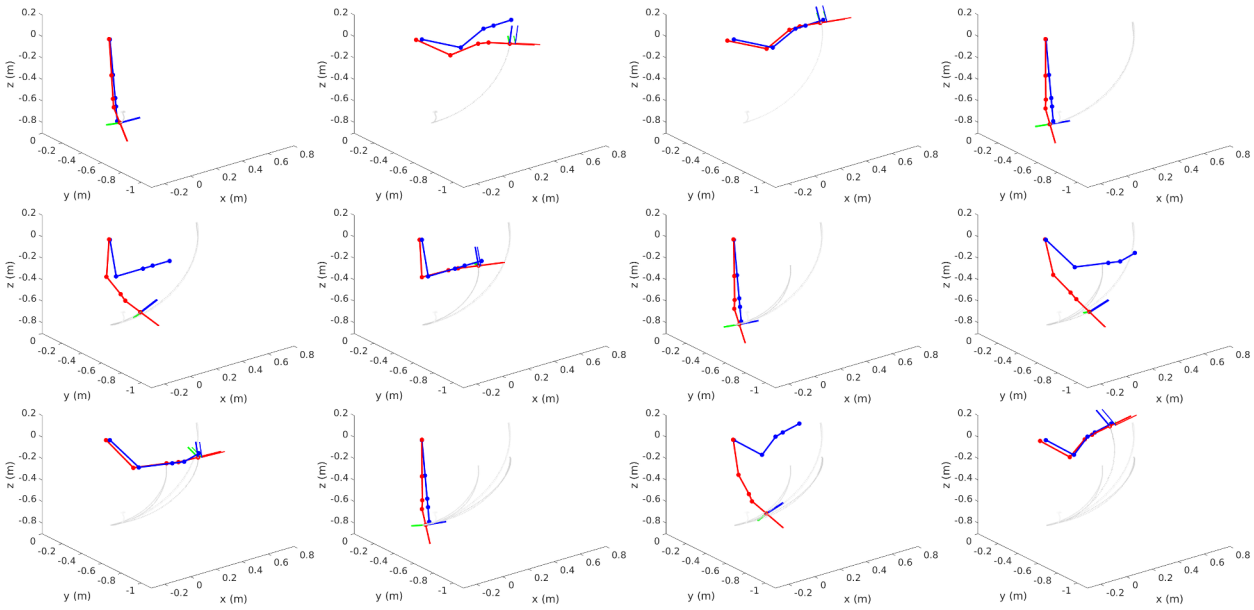

Figure 5.13: Image series of the right arm moving through a series of trajectories based on hardware data. This shows the relative positions of each joint and end-effector pose as the whole-body controller adjusts the floating base of ESCHER to maintain its original CoM.

encoders. Figure 5.14 presents the position and orientation component errors derived from the Valor ROS Controller commands and the robot proprioception data. This figure indicates that some number of joints in the right arm did not reach their reference positions as expected. This is directly related to the errors seen on hardware in Trajectory Controller Tracking (subsection 5.3.1). Since individual joints failed to track the setpoints sent by the Valor ROS Controller, it is unsurprising that the end-effector pose contains significant errors. The large yaw error seen the bottom plot of Figure 5.14 is almost exclusively the result of the failure of the wrist yaw joint to reach its command position.

Figure 5.15 compares the total end-effector error between the reference goals and proprioception data with the error due to translations in torso pose incurred by the whole-body control controller. This shows that slightly less than half of the error observed on hardware is attributable to the changes in torso pose. The Y-component error in the top plot in Figure 5.15 suggests that a major portion of the total error comes from the previously mentioned issues with the right wrist yaw motor. However, even if the right wrist yaw motor was operating correctly, there would still be errors unattributable to the whole-body controller.

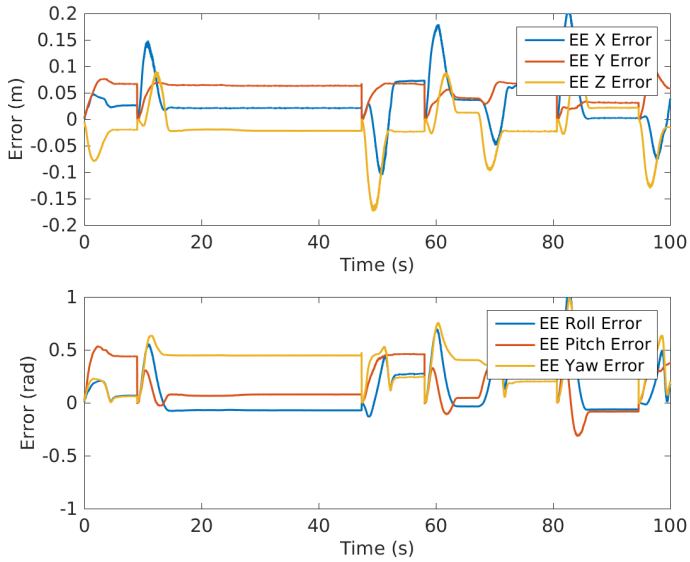

Figure 5.14: (Top) Position component errors observed on hardware between Valor ROS Controller end-effector pose and calculated pose from proprioception data. (Bottom) Orientation component errors.

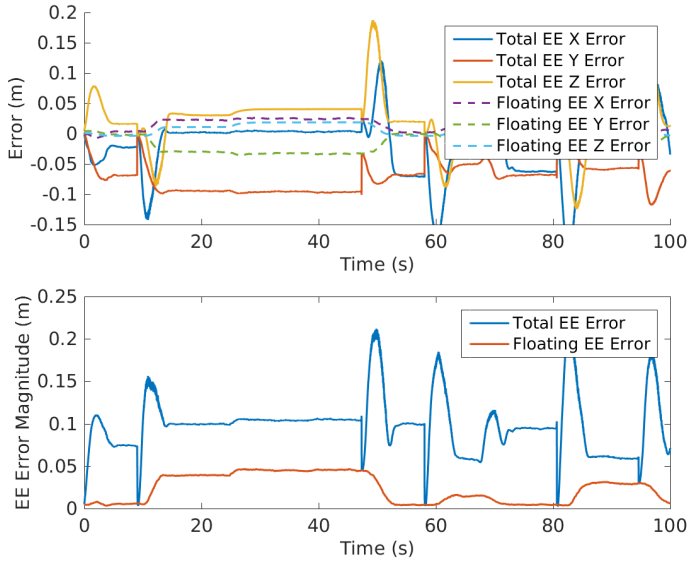

Figure 5.15: (Top) Position component errors of the total end-effector error (solid lines) and floating base error (dashed lines). (Bottom) Magnitudes of total end-effector error and floating base error.

5.5 Summary

The results presented in this chapter demonstrate that the Valor ROS Controller fulfills its role as an IPC interface and high level controller. They also quantify the end-effector error induced by the Whole-Body Controller maintaining a specific center of mass position. The first section quantified computational load of the Valor ROS Controller. Lower bounds on ROS and Bifrost message latencies were determined (THIS IS A POOR SENTENCE). The message pathways in the Valor ROS Controller quick pass data between ROS and Bifrost. The internal controllers in converge and adequately define setpoints for the Motion System, although they do not attempt to override the Motion System in eliminating steady-state error. The final section shows the affects of a whole-body control framework that prioritizes maintaining the center of mass on end-effector poses given a quasistatically defined trajectory in simulation and on hardware.

Chapter 6

Conclusion and Future Work

This thesis presents the development, implementation, and testing of the Valor ROS Controller which formed an integral part of Team VALOR's software system for the DRC Finals. The Valor ROS Controller provides command interfaces for operators and planners, interfaces with the Motion System running on THOR and ESCHER, and acts as an IPC interface layer between ROS and Bifrost. It thus enabled the integration of Team ViGIR's ROS based software with Team VALOR's whole-body control framework. To produce as much compatibility as possible with Team ViGIR's higher level systems, elements of Team ViGIR's Atlas Controller were generalized to form the basis of the Valor ROS Controller. The thesis culminates in verifying the efficient function of the Valor ROS Controller and quantifying end-effector error incurred by whole-body motions.

6.1 Conclusion

The Valor ROS Controller was a critical component of the system fielded by Team VALOR at the DRC Finals. As the primary control and IPC interface between ROS and the Bifrost based Motion System on ESCHER, the Valor ROS Controller ensured that proprioception data from the robot flowed up to the ROS network of nodes, and that commands reached the Motion System. The work presented in this thesis, along with the hard work of the members of Team VALOR, culminated in ESCHER participating in the DRC Finals, where it was the first of only two robots to walk across the 200 ft (61 m) dirt portion of the course. Figure 6.1 shows ESCHER stepping on to pavement as it finishes the dirt section of the

course. Detailed results from the DRC Finals can be found in [42].

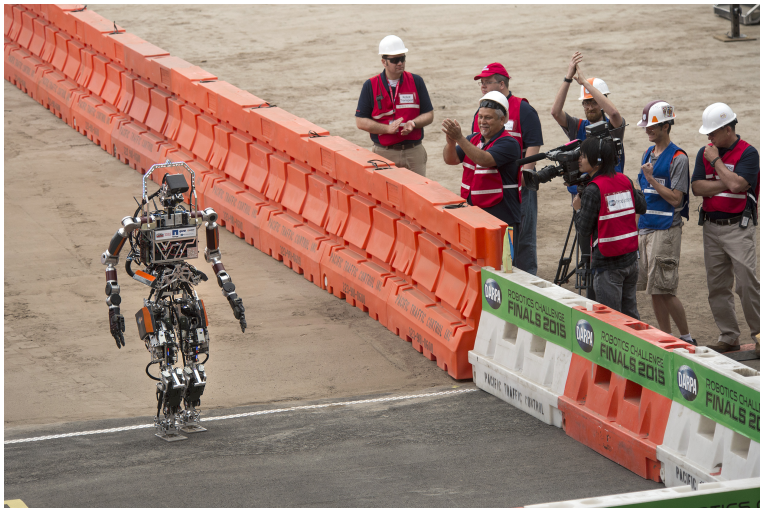

Figure 6.1: ESCHER completing the dirt section of the DRC Finals course. (photo courtesy of Virginia Tech /Logan Wallace)

The presented process statistics and message pathway latency results, along with the qualitative result of ESCHER competing in the DRC Finals, demonstrate that the Valor ROS Controller performs adequately as an individual process and as an IPC interface. On hardware, the Valor ROS Controller consumes approximately 53% of the processing time of a hypothetical single CPU core. In reality, the multithreaded nature of the Valor ROS Controller means that its load is spread over multiple CPU cores. The key message pathways through the Valor ROS Controller transfer proprioceptive data, mode change commands, and trajectory commands have latencies of 40 ms, 49 ms, and 335 ms respectively. Most of the last time is spent in the Motion System.

The Bifrost transmission latency lower-bound is 18% longer than ROS's. However, both IPCs produce transmission latencies that are two orders of magnitude lower than the shortest observed pathway latencies. Thus, for communications between processes on the same computer, the difference in transmission latency between ROS and Bifrost is negligible.

The recorded arm joint control signals from the Valor ROS Controller and Motion System, and proprioception data from the robot show that on hardware a number of the arm joints fail to reach their commanded positions. The substantial control signals sent from the Motion System to the right wrist without apparent effect indicate that that actuator is malfunctioning. This is the same joint that caused trouble for Team VALOR at the DRC

Finals. The steady-state errors displayed by the other joints can likely be reduced by tuning the PD gains in the Motion System. However, to truly eliminate the steady-state errors will likely require incorporating gravity compensation into the controllers.

Quasistatically planned arm trajectories alone are insufficient for precise placement of end-effectors on whole-body controlled robots. End-effector error caused by the whole-body control framework shifting ESCHER's torso to maintain its center of mass setpoint reaches over 5 cm in simulation. The noted joint position errors compound the end-effector error seen on hardware. However, nearly half of the 10 cm maximum observed steady-state end-effector error is attributable to the whole-body controller. Even with the near perfect actuation seen in simulation, the error due to whole-body motion during extended arm motions requires adjusting the end-effector pose before attempting to grasp an object.

6.2 Future Work

6.2.1 A Unified ROS Controller

In an ideal configuration, the Motion System and Whole-Body Controller would utilize the same communication framework and software paradigm as the overall system. From a research perspective, the Motion System, or at least its interface, should be highly extensible in order to facilitate various control schemes and new research directions. While the Valor ROS Controller was designed with adaptability with multiple robots in mind, its extensibility is fundamentally limited by the capabilities of the current Motion System.

The current system configuration has the Valor ROS Controller acting as a control and interface layer between higher level ROS nodes and the Motion System. This is not sustainable from either a controls or system design perspective. Stacking an additional layer of controllers on top of the Motion System controllers introduces significant system complexity and potential for anticipated interactions. The results presented in Messages Latencies (section 5.2) also suggest that there are substantial delays from passing messages through an interface layer which may adversely affect ROS based planners and high level controllers. In addition, the Motion System in its current form as based on Bifrost will be very difficult to maintain since the transition to ROS has deprecated Bifrost. This means that the Motion System will become a great hindrance to further research efforts

as intimate knowledge of its design disappears and the limitations that its implementation impose become even more rigid.

One potential solution is to move the core contributions of the Motion System into a unified ROS based controller. A couple of examples of ROS based whole-body controllers are Team IHMC's whole-body controller developed for the DRC Trials and Finals [71] [72] and ControlIt! from the he University of Texas at Austin [73] [74]. The Valor ROS Controller also has the potential to become the primary controller by incorporating the Motion Systems controllers as individual ROS controllers. In any event, this transition represents a significant amount of implementation work, however, if done correctly it will maintain current capabilities of the Motion System while also extending the longevity of the overall system and making it more flexible.

Bibliography

[1] Karthik Kappaganthu and C. Nataraj. "Optimal Biped Design Using a Moving Torso: Theory and Experiments". In: *Biped Robots*. Ed. by Armando Carlos De Pina Filho. InTech, Feb. 4, 2011. ISBN: 978-953-307-216-6.

[2] Shuuji Kajita et al. *Introduction to Humanoid Robotics*. Vol. 101. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. ISBN: 978-3-642-54535-1 978-3-642-54536-8.

[3] "Bipedal Robots and Walking". In: *Bipedal Robots: Modeling, Design and Walking Synthesis*. Ed. by Christine Chevallereau et al. ISTE, 2009, pp. 1–45. ISBN: 978-0-470-61162-3.

[4] Derek Lahr. "Design and Control of a Bipedal Robot". PhD thesis. Blacksburg, VA: Virginia Polytechnic Institute and State University, 2014.

[5] *Team ViGIR*. URL: `http://www.teamvigir.org/` (visited on 09/20/2015).

[6] *Team Hector — DRC Finals*. URL: `http://www.theroboticschallenge.org/finalist/team-hector` (visited on 09/21/2015).

[7] Stefan Kohlbrecher et al. "Human-robot Teaming for Rescue Missions: Team ViGIR's Approach to the 2013 DARPA Robotics Challenge Trials". In: *Journal of Field Robotics* 32.3 (May 1, 2015), pp. 352–377. ISSN: 1556-4967. DOI: `10.1002/rob.21558`.

[8] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 2009, p. 5.

[9] Shuuji Kajita and Bernard Espiau. "Legged Robots". In: *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 361–390. ISBN: 978-3-540-23957-4 978-3-540-30301-5.

[10] Hun-ok Lim and Atsuo Takanishi. "Biped walking robots created at Waseda University: WL and WABIAN family". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 365.1850 (Jan. 15, 2007), pp. 49–64. ISSN: 1364-503X, 1471-2962. DOI: 10.1098/rsta.2006.1920.

[11] Ichiro Kato. "Development of WABOT 1". In: *Biomechanism* 2 (1973), pp. 173–214.

[12] Rodney Brooks. "Humanoid robots". In: *Communications of the ACM* 45.3 (2002), pp. 33–38.

[13] S. Sugano and I. Kato. "WABOT-2: Autonomous robot with dexterous finger-arm–Finger-arm coordination control in keyboard performance". In: *1987 IEEE International Conference on Robotics and Automation. Proceedings*. 1987 IEEE International Conference on Robotics and Automation. Proceedings. Vol. 4. Mar. 1987, pp. 90–97. DOI: 10.1109/ROBOT.1987.1088025.

[14] S. Hashimoto et al. "Humanoid Robots in Waseda University—Hadaly-2 and WABIAN". In: *Autonomous Robots* 12.1 (Jan. 2002), pp. 25–38. ISSN: 0929-5593, 1573-7527. DOI: 10.1023/A:1013202723953.

[15] A. Sano and J. Furusho. "Control of torque distribution for the BLR-G2 biped robot". In: *Robots in Unstructured Environments*. 5th International Conference on Advanced Robotics. Vol. 1. 1991, pp. 729–734.

[16] Hirofumi Miura and Isao Shimoyama. "Dynamic Walk of a Biped". In: *The International Journal of Robotics Research* 3.2 (June 1, 1984), pp. 60–74. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836498400300206.

[17] Boston Dynamics. *Atlas: Anthropomorphic Robot*. URL: http://archive.darpa.mil/roboticschallengetrialsarchive/files/ATLAS-Datasheet_v15_DARPA.PDF (visited on 11/16/2015).

[18] Chi N. Thai. "ROBOTIS' Robot Systems". In: *Exploring Robotics with ROBOTIS Systems*. DOI: 10.1007/978-3-319-20418-5_2. Springer International Publishing, 2015, pp. 5–17. ISBN: 978-3-319-20417-8 978-3-319-20418-5.

[19] Ill-Woo Park et al. "Mechanical design of humanoid robot platform KHR-3 (KAIST Humanoid Robot 3: HUBO)". In: *2005 5th IEEE-RAS International Conference on Humanoid Robots*. 2005 5th IEEE-RAS International Conference on Humanoid Robots. Dec. 2005, pp. 321–326. DOI: 10.1109/ICHR.2005.1573587.

[20] Ill-Woo Park et al. "Mechanical design of the humanoid robot platform, HUBO". In: *Advanced Robotics* 21.11 (Nov. 1, 2007), pp. 1305–1322. DOI: 10.1163/156855307781503781.

[21] K. Kaneko et al. "Design of prototype humanoid robotics platform for HRP". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002. Vol. 3. 2002, 2431–2436 vol.3. DOI: 10.1109/IRDS.2002.1041632.

[22] K. Kaneko et al. "Humanoid robot HRP-3". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008. Sept. 2008, pp. 2471–2478. DOI: 10.1109/IROS.2008.4650604.

[23] F. Negrello et al. "A Modular Compliant Actuator for Emerging High Performance and Fall-Resilient Humanoids". In: (2015).

[24] Y. Ito et al. "Development and verification of life-size humanoid with high-output actuation system". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014 IEEE International Conference on Robotics and Automation (ICRA). May 2014, pp. 3433–3438. DOI: 10.1109/ICRA.2014.6907353.

[25] Miomir Vukobratovic and Branislav Borovac. "ZERO-MOMENT POINT — THIRTY FIVE YEARS OF ITS LIFE". In: *International Journal of Humanoid Robotics* 1.1 (2004), pp. 157–173.

[26] Jeakweon Han. "Bipedal walking for a full-sized humanoid robot utilizing sinusoidal feet trajectories and its energy consumption". PhD thesis. Virginia Polytechnic Institute and State University, 2012.

[27] J. Pratt et al. "Capture Point: A Step toward Humanoid Push Recovery". In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*. 2006 6th IEEE-RAS International Conference on Humanoid Robots. Dec. 2006, pp. 200–207. DOI: 10.1109/ICHR.2006.321385.

[28] S. Kajita et al. "The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation". In: *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001. Proceedings*. 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001. Proceedings. Vol. 1. 2001, 239–246 vol.1. DOI: 10.1109/IROS.2001.973365.

[29] T. Takenaka, T. Matsumoto, and T. Yoshiike. "Real time motion generation and control for biped robot -1st report: Walking gait pattern generation-". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009.* IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009. Oct. 2009, pp. 1084–1091. DOI: 10.1109/IROS.2009.5354662.

[30] J. Englsberger, C. Ott, and A. Albu-Schaffer. "Three-dimensional bipedal walking control using Divergent Component of Motion". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Nov. 2013, pp. 2600–2607. DOI: 10.1109/IROS.2013.6696723.

[31] Michael Hopkins, Dennis W. Hong, Alexander Leonessa, et al. "Humanoid locomotion on uneven terrain using the time-varying divergent component of motion". In: *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on.* IEEE, 2014, pp. 266–272.

[32] Michael A. Hopkins, Dennis W. Hong, and Alexander Leonessa. "Compliant Locomotion Using Whole-Body Control and Divergent Component of Motion Tracking". In: *IEEE International Conference on Robotics and Automation, 2015. Proceedings. ICRA '15.* Robotics and Automation (ICRA), 2015.

[33] D. Gauthier et al. "Interprocess communication for distributed robotics". In: *IEEE Journal of Robotics and Automation* 3.6 (Dec. 1987), pp. 493–504. ISSN: 0882-4967. DOI: 10.1109/JRA.1987.1087141.

[34] Jay Gowdy. *A qualitative comparison of interprocess communications toolkits for robotics.* Technical Report CMU-RI-TR-00-16. Pittsburgh, PA: Carnegie Mellon University, 2000, pp. 1–20.

[35] Jay Gowdy. *Ipt: An object oriented toolkit for interprocess communication.* Technical Report CMU-RI-TR-96-07. Pittsburgh, PA: Department of Computer Science, Carnegie-Mellon University, 1996.

[36] Matthew McNaughton et al. "Software Infrastructure for a Mobile Robot". In: *Experience from the DARPA Urban Challenge.* Springer Science & Business Media, Oct. 1, 2011, pp. 19–42. ISBN: 978-0-85729-772-3.

[37] Greg Broten et al. "Software systems for robotics an applied research perspective". In: *International Journal of Advanced Robotic Systems* 3.1 (2006), pp. 11–16.

[38] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. "YARP: yet another robot platform". In: *International Journal on Advanced Robotics Systems* 3.1 (2006), pp. 43–48.

[39] David Moore, Edwin Olson, and Albert Huang. *Lightweight communications and marshalling for low-latency interprocess communication*. Technical Report. Cambridge, MA: Massachusetts Institute of Technology, 2009.

[40] A.S. Huang, E. Olson, and D.C. Moore. "LCM: Lightweight Communications and Marshalling". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Oct. 2010, pp. 4057–4062. DOI: 10.1109/IROS.2010.5649358.

[41] Neil T. Dantam et al. "Multiprocess Communication and Control Software for Humanoid Robots". In: *IEEE Robotics and Automation Magazine* (2014). DOI: DOI: 10.1109/MRA.2014.2356937.

[42] Coleman Knabe et al. "Designing for Compliance: ESCHER, Team VALOR's Compliant Biped". In: *Journal of Field Robotics* ().

[43] G.A. Pratt and M.M. Williamson. "Series elastic actuators". In: *1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings*. 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. Vol. 1. Aug. 1995, 399–406 vol.1. DOI: 10.1109/IROS.1995.525827.

[44] Jerry Pratt, Ben Krupp, and Chris Morse. "Series elastic actuators for high fidelity force control". In: *Industrial Robot: An International Journal* 29.3 (June 1, 2002), pp. 234–241. ISSN: 0143-991X. DOI: 10.1108/01439910210425522.

[45] J.W. Sensinger and R.F. Weir. "Improvements to Series Elastic Actuators". In: *Proceedings of the 2nd IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*. Proceedings of the 2nd IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. Aug. 2006, pp. 1–7. DOI: 10.1109/MESA.2006.296927.

[46] Coleman Knabe et al. "Design of a Compact, Lightweight, Electromechanical Linear Series Elastic Actuator". In: (Aug. 17, 2014), V05BT08A014. DOI: 10.1115/DETC2014-35096.

[47] B. Lee. "Design of a humanoid robot for disaster response". Masters Thesis. Blacksburg, VA: Virginia Polytechnic Institute and State University, 2014.

[48] Coleman Knabe. "Design of Linear Series Elastic Actuators for a Humanoid Robot". Masters Thesis. Virginia Polytechnic Institute and State University, 2014.

[49] C. Knabe et al. "Design of a Series Elastic Humanoid for the DARPA Robotics Challenge". In: Humanoid Robots (Humanoids), IEEE/RAS International Conference on. Oct. 2015.

[50] *PLATFORM - ROBOTIS*. URL: http://www.robotis.us/platform/ (visited on 09/20/2015).

[51] *Adroit® Manipulator Arm — HDT Global*. URL: http://www.hdtglobal.com/series/adroit-manipulator-arm/ (visited on 09/20/2015).

[52] *MultiSense S7 — Carnegie Robotics*. URL: http://carnegierobotics.com/multisense-s7/ (visited on 08/28/2015).

[53] *Scanning range finder UTM-30LX-EW — Photo sensor — PRODUCTS — HOKUYO AUTOMATIC CO.,LTD.* URL: https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx_ew.html (visited on 08/28/2015).

[54] *FLIR A65 / A35 / A15 / A5 - Compact Thermal Imaging Cameras For Machine Vision*. URL: http://www.flir.com/automation/display/?id=56341#tab2 (visited on 08/28/2015).

[55] *1750 IMU Inertial Measurement Unit*. URL: http://www.kvh.com/Military-and-Government/Gyros-and-Inertial-Systems-and-Compasses/Gyros-and-IMUs-and-INS/IMUs/1750-IMU.aspx (visited on 09/07/2015).

[56] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. 7th ed. Hoboken, NJ: J. Wiley & Sons, 2005. 921 pp. ISBN: 978-0-471-69466-3.

[57] John Shapley Gray. *Interprocess Communications in Linux*. Prentice Hall Professional, 2003. 624 pp. ISBN: 978-0-13-046042-4.

[58] *Distributed Messaging - zeromq*. URL: http://zeromq.org/ (visited on 09/07/2015).

[59] *ROS.org — Powering the world's robots*. URL: http://www.ros.org/ (visited on 09/22/2015).

[60] *ROS/Concepts - ROS Wiki*. URL: http://wiki.ros.org/ROS/Concepts (visited on 09/22/2015).

[61] *ros_control - ROS Wiki*. URL: http://wiki.ros.org/ros_control (visited on 11/16/2015).

[62] Michael A. Hopkins. "Dynamic Locomotion and Whole-Body Control for Compliant Humanoids". PhD thesis. Blacksburg, VA: Virginia Polytechnic Institute and State University, 2014.

[63] *top(1): tasks - Linux man page*. URL: http://linux.die.net/man/1/top (visited on 09/18/2015).

[64] *MATLAB - The Language of Technical Computing*. URL: http://www.mathworks.com/products/matlab/?s_tid=hp_fp_ml&nocookie=true (visited on 09/19/2015).

[65] *Robotics System Toolbox Documentation*. URL: http://www.mathworks.com/help/robotics/index.html?nocookie=true (visited on 09/19/2015).

[66] S Southward. *L6 - Correlation, Convolution, Power Spectrum*. Class Notes, ME5564. Feb. 2014.

[67] Darryl Gove. *Multicore Application Programming: for Windows, Linux, and Oracle Solaris*. Addison-Wesley Professional, Nov. 9, 2010. 466 pp. ISBN: 978-0-321-71141-0.

[68] P. Tomei. "A simple PD controller for robots with elastic joints". In: *IEEE Transactions on Automatic Control* 36.10 (Oct. 1991), pp. 1208–1213. ISSN: 0018-9286. DOI: 10.1109/9.90238.

[69] Nikolaus Wittenstein. "Force Feedback For Reliable Robotic Door Opening". Masters Thesis. Blacksburg, VA: Virginia Tech, July 17, 2015. 71 pp.

[70] T. C. Hsia. "Robustness analysis of a pd controller with approximate gravity compensation for robot manipulator control". In: *Journal of Robotic Systems* 11.6 (Jan. 1, 1994), pp. 517–521. ISSN: 1097-4563. DOI: 10.1002/rob.4620110606.

[71] *ihmcrobotics / ihmc_ros / wiki / whole-body-controller — Bitbucket*. URL: https://bitbucket.org/ihmcrobotics/ihmc_ros/wiki/whole-body-controller (visited on 10/30/2015).

[72] Siyuan Feng et al. "Optimization-based Full Body Control for the DARPA Robotics Challenge". In: *Journal of Field Robotics* 32.2 (2015), pp. 293–312.

[73] Chien-Liang Fok et al. "ControlIt! - A Middleware for whole-Body Operational Space Control". In: *International Journal of Humanoid Robotics* (Oct. 2015). DOI: 10.1142/S0219843615500401.

[74] Chien-Liang Fok and Sentis Luis. "Integration and Usage of a ROS-based Whole Body Control Software Framework". In: *Springer Book on the Robot Operating System (ROS)*. June 2015.