Model-Based Robot Control in Human-in-the-Loop Cyber Physical Systems

A Dissertation Presented
by

Velin Dimitrov

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

May 2017
To my parents.
Contents

List of Figures vi
List of Tables xi
List of Acronyms xii
Acknowledgments xiii
Abstract of the Dissertation xv

1 Introduction 1
1.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Challenges in HiLCPS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Prior Work in Shared Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.4 Our Focus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
1.5 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2 Shared Control 12
2.1 Shared Control Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.2 Redundancy Resolution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.3 Shared Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.4 Experimental Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
2.5 Results and Findings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
2.6 Doorway Traversal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

3 Rovers 36
3.1 ORYX 2.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
3.2 Overview of ORYX 2.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.2.1 Passive Averaging Suspension . . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.3 Kinematic Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.4 Feedforward Velocity Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.4.1 Terrain Profile Estimate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.4.2 Compensating Wheel Velocities . . . . . . . . . . . . . . . . . . . . . . . . 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>Experimental Setup and Results</td>
<td>47</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusions</td>
<td>49</td>
</tr>
<tr>
<td>3.7</td>
<td>AERO 2013</td>
<td>50</td>
</tr>
<tr>
<td>3.8</td>
<td>Platform Design</td>
<td>53</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Navigation Sensors</td>
<td>54</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Sample Detection and Classification</td>
<td>54</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Manipulation</td>
<td>55</td>
</tr>
<tr>
<td>3.9</td>
<td>Previous Work</td>
<td>55</td>
</tr>
<tr>
<td>3.10</td>
<td>Navigation Overview</td>
<td>56</td>
</tr>
<tr>
<td>3.10.1</td>
<td>Hierarchical Control</td>
<td>56</td>
</tr>
<tr>
<td>3.10.2</td>
<td>Fault Tolerance</td>
<td>57</td>
</tr>
<tr>
<td>3.11</td>
<td>Local Planner</td>
<td>58</td>
</tr>
<tr>
<td>3.11.1</td>
<td>Driving with Tentacles</td>
<td>58</td>
</tr>
<tr>
<td>3.11.2</td>
<td>Extended Driving with Tentacles Algorithm</td>
<td>58</td>
</tr>
<tr>
<td>3.12</td>
<td>Global Planner</td>
<td>60</td>
</tr>
<tr>
<td>3.12.1</td>
<td>Carrot Path</td>
<td>61</td>
</tr>
<tr>
<td>3.12.2</td>
<td>Task-based Planning Strategies</td>
<td>61</td>
</tr>
<tr>
<td>3.13</td>
<td>Sample Detection and Retrieval</td>
<td>62</td>
</tr>
<tr>
<td>3.14</td>
<td>Arm Controller</td>
<td>63</td>
</tr>
<tr>
<td>3.15</td>
<td>Results</td>
<td>63</td>
</tr>
<tr>
<td>3.16</td>
<td>AERO 2014</td>
<td>64</td>
</tr>
<tr>
<td>3.17</td>
<td>Vision System</td>
<td>66</td>
</tr>
<tr>
<td>3.18</td>
<td>Navigation and Planning</td>
<td>70</td>
</tr>
<tr>
<td>3.18.1</td>
<td>Local Drive Planner</td>
<td>71</td>
</tr>
<tr>
<td>3.18.2</td>
<td>Global Drive Planner</td>
<td>72</td>
</tr>
<tr>
<td>3.18.3</td>
<td>Arm Planners</td>
<td>72</td>
</tr>
<tr>
<td>3.18.4</td>
<td>High Level Planner</td>
<td>73</td>
</tr>
<tr>
<td>3.19</td>
<td>Analysis of Teleoperation and Autonomous Tasks</td>
<td>77</td>
</tr>
<tr>
<td>3.20</td>
<td>Teleoperation with ORYX 2.0</td>
<td>78</td>
</tr>
<tr>
<td>3.21</td>
<td>Autonomous Operation with AERO</td>
<td>81</td>
</tr>
<tr>
<td>3.22</td>
<td>Shared Control and Behaviors</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>Humanoid Robots</td>
<td>85</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>85</td>
</tr>
<tr>
<td>4.2</td>
<td>Robot Hardware</td>
<td>87</td>
</tr>
<tr>
<td>4.2.1</td>
<td>ATLAS Robot</td>
<td>87</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Control Station</td>
<td>88</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Robot Hands</td>
<td>89</td>
</tr>
<tr>
<td>4.3</td>
<td>Software Architecture</td>
<td>91</td>
</tr>
<tr>
<td>4.4</td>
<td>DRC Driving a Utility Vehicle Task</td>
<td>92</td>
</tr>
<tr>
<td>4.5</td>
<td>Approach</td>
<td>94</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Steering</td>
<td>94</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Velocity Control</td>
<td>96</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Simulation</td>
<td>99</td>
</tr>
<tr>
<td>4.6</td>
<td>Results</td>
<td>99</td>
</tr>
</tbody>
</table>
4.7 Points ........................................................................................................ 102
4.8 Valkyrie ...................................................................................................... 103
4.9 Hardware .................................................................................................... 106
4.10 Software .................................................................................................... 108
4.11 Experimental Results and Capabilities ..................................................... 110
  4.11.1 Mobility ............................................................................................... 110
4.12 Best Practices ............................................................................................ 112
4.13 Full-size Humanoid Robot Dataset ............................................................ 115
4.14 Future Work and Conclusions .................................................................. 119

5 Conclusion .................................................................................................... 120

Bibliography .................................................................................................... 122
List of Figures

2.1 The shared control architecture consisting of the knowledge base, action engine, achievable action gate, perception engine, low-level robot control, human robot interface, context engine, cloud engine, and operator interface. Not all HiLCPS will necessarily implement all blocks. The architecture is split in both cyber and physical domains, and also human and robot domains depending on where elements are located and who contributes information to each element. ................................................................. 13

2.2 The action engine provides desired actions for the robots to complete. Various control schemes can be implemented here including teleoperated control, supervised or guarded autonomy, shared control, and autonomous control. ............................................................... 15

2.3 An architecture for HiLCPS robot controllers. Each robot may have different controllers providing different inputs at different time-scales. In addition, there may be coupling between the controllers as well. The architecture allows for both parallel and series combinations of controllers to be modeled. ................................................................. 16

2.4 The blocks highlighted in green are developed in the redundancy resolution-based formulation for integrating user input and autonomous behavior in the shared control system. The formulation includes splitting the task into primary and secondary tasks that can be simultaneously achieved leveraging the redundant DOF: ....................................................... 18

2.5 The environment where test subjects were asked to control a Turtlebot mobile robot. The users had to drive the robot from the starting position to a finish position marked on the ground in normal, time-delay, and drift conditions. ......................................................... 24

2.6 The view of the experiment from the robot perspective. The autonomous path computed by the navigation package is shown in green on the map and in the field of view of the robot camera. ................................................................. 26

2.7 A boxplot showing the time to complete the navigation task using the robot in all three cases: normal, time-delay, and utilizing drift. ......................................................... 27

2.8 A boxplot showing the distance to complete the navigation task using the robot in all three cases: normal, time-delay, and utilizing drift. ......................................................... 28

2.9 A boxplot showing the percentage degradation in time for each test subject compared to normal operation to complete the navigation task. ......................................................... 29

2.10 A boxplot showing the percentage degradation in distance for each test subject compared to normal operation to complete the navigation task with time delay and drift. ......................................................... 30
2.11 The $\alpha$ parameter controls the level of control authority between user and robot. This figure shows how the parameter changes throughout the course of an experiment for one user. Initially, the user is mainly in control of the robot with a little guidance provided by the autonomous navigation. As the goal is approached, the control authority shifts toward a more equal share, helping guide the robot to the target.

2.12 The setup for the blended shared control experiment for doorway traversal. Users are asked to navigate a Turtlebot through an open doorway while the blended shared controller is fusing their input with the autonomous navigation trajectory. The performance of the human–robot team is compared for drift and time-delay scenarios.

2.13 A boxplot showing the times in seconds to complete the doorway traversal task for the normal, time delay, and drift scenarios. The results are similar to the simpler navigation task.

2.14 A boxplot showing the times in seconds to complete the doorway traversal task for the normal, time delay, and drift scenarios. The results are similar to the simpler navigation task.

3.1 AERO (left) and ORYX 2.0 (right) at Bond Construction during field testing.

3.2 ORYX 2.0 collecting a sample rock from the moon-analog crater field at Johnson Space Center Analog Test Facility during the 2012 NASA RASC-AL Robo-Ops competition. ORYX 2.0 collected a total of 13 rocks and one "alien" life-form taking first place in the competition.

3.3 The development of the ORYX 2.0 rover marked the beginnings of our shared control implementation. The work resulting in a low-level controller that assisted the operator to drive in straighter trajectories over rough terrain. The implementation contributed mainly to the robot control block, but is relevant due to the foundation it laid for AERO and the later developments.

3.4 ORYX 2.0 shown driving over an obstacle. The passive averaging suspension serves two related purposes. It allows all four wheels to be on the ground while traversing obstacles for maximum torque and halves the roll angle experienced by the chassis of the robot.

3.5 A rendering of the robot showing the chassis and passive averaging suspension implemented through a 6-bar linkage composed of rocker linkages and a differencing arm. The passive averaging suspension keeps all four wheels on the ground irrespective of the terrain the robot is driving on and halves the roll the chassis experiences.

3.6 The kinematic model for the rover showing the frames used to track the chassis position, rocker angle, and wheel positions relative to the ideal ground plane.

3.7 A diagram showing the process for taking the proprioceptive sensor information, passing it through the kinematic model, and calculating the compensated wheel velocities.

3.8 A figure showing the compensation of the wheel velocities based on the terrain slope underneath the wheel. The controller aims to keep the desired heading velocity constant over the slope.
3.9 A picture showing the physical experiment setup with ORYX 2.0 and a trapezoidal obstacle. The rover is driven over the obstacle while the yaw deviations are measured using the onboard IMU.

3.10 A diagram showing the dimensions (in meters) of the trapezoidal obstacle used to conduct the experiment.

3.11 Yaw deviations from two separate trials where the controller is enabled and disabled. The controller provides an obvious improvement helping decrease the robot yaw deviations.

3.12 AERO, while originally designed to be fully autonomous, contributes to many of the blocks in the shared control framework enabling the implementation of robust autonomy specifically in the areas of perception, navigation, and manipulation. Limited contributions to the knowledge base are also present given the classifiers used to find unknown objects.

3.13 AERO along with its VR tag homing beacon.

3.14 The three levels of the hierarchical control. The supervisor coordinates the mission-level tasks. The global planner plans beyond the sensor horizon on the global map. The local planner plans on a local map representing a snapshot of the current environment around the robot.

3.15 A visualization of the tentacles in a speed set. As the robot’s velocity increases, the tentacles shift closer to the center line and become longer.

3.16 A flow diagram showing the components of the vision system for detecting samples. The raw image streams are inputs, and after the samples are identified in the frame, it is passed to the stages of the manipulator controller to be picked up.

3.17 A plot of the position of the robot in the global frame which includes a dynamic obstacle (a person moving in front of the robot) at location \( x \approx 0.0, y \approx 8.75 \). The blue line is the path the robot took. The green dashed line is the carrot path. And the purple stars are points where the carrot path was re-planned.

3.18 Visualization of the robot dodging an obstacle.

3.19 Benchtop test showing the disparity map generated by the stereo cameras, and the two samples detected by the classifier. The sample positions are passed to the arm controller which moves the manipulator to retrieve the samples.

3.20 AERO during field testing in a gravel mining pit.

3.21 A sample subset of the training images used for the precached hook sample of the NASA SRR Competition. Over 3000 such images were used to generate the classifier for the hook.

3.22 Four screen shots from the ROS visualization tool, rviz, showing the visual odometry package tracking the motion of the robot. The robot drives forward first, takes a slight right, and then a slight left. The red line traces out the path the visual odometry package calculated for the robot. The visual odometry was tested and proved useful in certain cases, but ultimately not used during competition.

3.23 An overview of the organization of the planners on the robot and their tasks.
3.24 The finite state machine used to complete level one that represents the current high
level task that the robot is trying to accomplish, such as driving off the platform,
picking up the sample and driving toward the platform. Each of the states represents
a different requested action that is dispatched to a lower level planner to execute.
Recovery behaviors for each action are also specified so that the robot can perform
a recovery behavior that makes sense in the current context. ................. 73
3.25 The beacon covered with AprilTags that was placed on the starting platform. .... 76
3.26 The ORYX 2.0 primary operator graphical user interface (GUI) is shown. Various
values such as internal temperatures of components and battery voltages are pro-
vided on the left. A large area in the center shows the views from the cameras.
The pose of the robot, angle of camera mast, network information, and a small sec-
ond camera view are all provided on the right. The secondary operator has a very
similar GUI, but the mast camera is generally displayed in the center panel. There
are few buttons on the GUI, since the robot is primarily controlled with through
commercially available gaming controllers with two sticks and multiple buttons]. .... 79
3.27 ORYX 2.0 picking up a purple sample rock in JSC sand pit. The clear polycarbonate
plate and small piece of tape in the corner proved very effective in lining up the
manipulator to conduct a pickup operation. ........................................ 80
4.1 The DARPA Robotics Challenge greatly propelled the development of humanoid
robots for disaster response within supervised autonomy frameworks. The contrib-
utions of our team included elements from both the human and robot halves of the
framework. The contributions mainly focused on the right side though, with limited
but important contributions to the cyber elements on the left. ................... 86
4.2 ATLAS robot and a stick figure showing the position and orientation of joints. .... 87
4.3 A visualization of the network layout between the robot sensors and the operator
control station. .............................................................. 88
4.4 The layout and functions of each display in the control station and which Operator
Control Unit (OCU) they connect to. ........................................... 89
4.5 Left to right: Robotiq, Sandia and iRobot hands. ................................ 90
4.6 A flow chart showing the software architecture and communication between differ-
ent subsystems. .............................................................. 92
4.7 The ATLAS robot at the start of the course at the DRC Trials. ................. 93
4.8 A rendering showing the slalom course the robot needs to navigate with the utility
vehicle as part of the DARPA Robotics Competition Trials. The robot and vehicle
start at the top of the course and must reach the light gray finish zone to receive a
point. ........................................................................ 93
4.9 The robot uses its stereo cameras and LIDAR to estimate the velocity of the vehi-
cle. Using this information and the desired velocity commanded by the user, the
robot actuates the throttle through a PI control loop. Images from the camera are
compressed and sent over the bandwidth limited and latency-induced link back to
the operator. The operator controls the steering throughout the driving task by com-
manding the robot to specific steering angles. ................................. 95
4.10 A simple scenario demonstrating the LIDAR-based velocity measurement showing
the vehicle and one LIDAR scan point through time as the vehicle moves. The
estimated velocity is calculated for every scan point, after which they are ranked by
a confidence metric calculated by the intensity of the return, and top results averaged
to estimate the vehicle velocity.

4.11 The view provided back to the operator during the 2013 DRC Trials. This image is
from the competition run on the actual course. Despite the high compression and
noticeable fuzziness of the image, the operator is able to navigate around the traffic
barriers.

4.12 The vision-based velocity estimate (red) and desired velocity (blue) during the 2013
DRC Trials. The data is from the competition run on the actual course. The drop in
velocity in the middle is attributed to the robot’s foot slipping off the throttle. The
operator was able to intervene and remedy the situation.

4.13 Development on the NASA Valkyrie humanoid robot contributes to two separate
areas of the shared control framework. Task validation for the SRC contributes to
robot control blocks of the framework demonstrating that a humanoid robot is capa-
bile of completing tasks in environments designed for astronauts. The generation of
the first dataset with a full-size humanoid robot will enable the research community
without access to a physical robot to contribute to the context and cloud engines.

4.14 A hardware diagram showing the location of all the sensors and joint actuators on
the Valkyrie robot. Valkyrie has a total of 32 degrees of freedom (DOF) with 6 DOF
legs, 7 DOF arms, 3 DOF back, and 3 DOF neck, not counting the DOF on the fingers.

4.15 A view from Rviz showing the TF frames of the joint locations of the 32 DOF on
the robot.

4.16 Left Hip Torques with a 1.0s transfer time and a 1.0s swing time. Failed Attempt.
First 12s show a side step with a 10s pause, followed by a failed first step.

4.17 Left Hip Torques with a 2.0s transfer time and 2.0s swing time. Successful Attempt.
First 15s show a side step with a 10s pause, followed by 7 successful 0.3 m steps.

4.18 The sequence of images shows Valkyrie picking up a box with a whole body control
algorithm.

4.19 Leg joint torques during the box pickup. The joint torques not shown do not change
significantly during the pickup operation. The data shown is from the left leg, but
the right leg data is very similar both quantitatively and qualitatively. Torque limits
for the hip and knee pitch joints are 350 Nm and 205 Nm for the ankle pitch.

4.20 Pelvis position of Valkyrie while walking forward as measured by the internal state
estimator on the robot and the ground truth data from the Qualisys motion tracking
system.

4.21 The distance error of the internal robot state estimator as a function of time with
respect to the ground truth.
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Blended Shared Control Experiment Setup</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Navigation System Failure Modes</td>
<td>57</td>
</tr>
<tr>
<td>4.1</td>
<td>A description of ATLAS arm joints</td>
<td>88</td>
</tr>
<tr>
<td>4.2</td>
<td>Robot hand specifications</td>
<td>90</td>
</tr>
<tr>
<td>4.3</td>
<td>WPI-CMU DRC Team Points Breakdown at the DRC Trials</td>
<td>103</td>
</tr>
</tbody>
</table>
List of Acronyms

**DARPA**  Defense Advanced Research Projects Agency Agency of the US Department of Defense responsible for emerging technologies for use by the military.

**DRC**  DARPA Robotics Challenge. A robotics competition sponsored by DARPA focusing on developing humanoid robotics that can address disaster response scenarios.

**HiLCPS**  Human-in-the-Loop Cyber Physical Systems A broad class of systems of systems that generally include a human input to the system or one of the subsystems integrated within the context of a traditional cyber physical system. The human-in-the-loop part defines the system as semiautonomous, adaptive, and dynamic due to the introduction of an impossible to deterministically model human input. The second part of the term refers to cyber physical systems that focuses on the seamless integration of computational algorithms and physical components.

**RIVeR**  Robotics and Intelligent Vehicles Research The Robotics and Intelligent Vehicles Research Laboratory focuses on ground, underwater, and aerial robotic systems within the Electrical and Computer Engineering Department at Northeastern University since 2015. The lab was originally founded at Worcester Polytechnic Institute as a part of the Robotics Engineering program in 2010.

**SRC**  Space Robotics Challenge The Space Robotics Challenge is a NASA Centennial Challenge following in the footstep of the DRC. It is intended to spur the development of methods and approaches to enable the use of humanoid robots in space exploration and development tasks. The Valkyrie humanoid robot is a NASA designed robot at the center of the challenge, intended to be a testbed for algorithms and software.

**SRR**  Sample Return Robot Challenge The Sample Return Robot Challenge is a NASA Centennial Challenge from 2012 to 2016 to develop autonomous navigation, perception, and mobility capabilities focused on space exploration. Robots had to locate and retrieve various geologic samples in an unknown, GPS-denied environment.
Acknowledgments

I wish to thank all the individuals who have supported me throughout the process of conducting my research and putting together my dissertation. Henry Ford once said "Coming together is a beginning; keeping together is progress; working together is success." I have had the opportunity to work with an outstanding group of individuals at both WPI and Northeastern over the last six years, and they have defined my successes.

As a first-year graduate student, I had the pleasure to work with four undergraduates who designed, built, and tested the ORYX 2.0 rover. I want to acknowledge Jon Anderson, Tom Carlone, Joseph Amato, and Michael Fagan for all their contributions resulting in ORYX 2.0. From the time spent in the machine shop to their efforts in modelling the robot in MATLAB, the team’s work seeded our efforts in shared control for the next five years. Their professionalism, dedication, and drive are unmatched.

After the ORYX 2.0 project, I was fortunate to lead the AERO team for the next two years. Mathew DeDonato, who I had met earlier at Teledyne Benthos, was finishing his Masters’ degree at WPI while helping out with AERO. His contributions to the leadership of the team, software development skills, and general engineering skills were a constant support in our efforts. Adam Panzica contributed heavily to the navigation and control of the robot, while long nights with Samir Zutshi resulted in classifiers that could find our samples. Supreeth Rao trained those classifiers on a painstakingly hand cropped dataset for the best performance. Additionally, Michael Fagan helped with the design and modification of our Husky robot to make it what it is today. Finally, I knew I could always count on Mitchell Wills to fix any software problem I could ask of him. The onboard robot intelligence would have never been possible without his contributions.

Mathew DeDonato moved on to lead the ATLAS project and our participation in the DARPA Robotics Challenge. I was lucky to work under his leadership with the rest of the DRC team. There are too many team members to acknowledge them all individually, but I would like to call out a few that I interacted with regularly. Kevin Knoedler took his years of software project management experience in software and applied it to help push the team through the VRC and on through the DRC Finals. Without his contributions, I would not have had the opportunity even work with ATLAS and likely Valkyrie as well. He and I worked on making the driving task a reality, and one of my proudest moments occurred when the robot and vehicle crossed the finish line at the DRC trials. Felipe Polido always had a friendly smile, and I counted on to help me with whatever I needed whether it required mental thought or physical strength. Siyuan Feng, Hirotaka Moriguchi, Eric Whitman, X. Xinjilefu, and Prof. Chris Atkeson taught us who had absolutely zero experience with humanoid robots up to that point how we should approach the tasks of the DRC, and without their support and guidance, we would not have enjoyed the success we did.

Thanks to the success of the DRC team, we received the opportunity to work with the
NASA Valkyrie humanoid robot. Without Murphy Wonsick’s co-leadership, the team would be farther behind in the task validation for the SRC, and the first humanoid robot data set would not exist. Even under the tightest deadlines, Xianchao Long has never refused to generate a trajectory to implement a task for me utilizing his constrained motion planning approach. Thanks to Pauline Maurice and Prof. Dagmar Sternad for helping us with the motion capture system for our humanoid robot dataset. James Tukpah helped crop and organize the data. I would like to also acknowledge Jordan Allspaw, Carlos Ibarra Lopez, and Adam Norton from UMass Lowell for their assistance in setting up and maintaining the robot, and Prof. Holly Yanco for providing a facility like NERVE Center for the robot to live in.

I would like to also thank a few people who were not part of specific projects with me, but without their assistance, my graduate experience would have been significantly different. Vinayak Jagtap helped me set up countless networks for our smart environments, and I knew I could always count on him to assist me with any system administration task that needed to be done. Jayam Patel and Nandan Banarjee kept the mood light when the stress would get too much. Anas Abou Allaban made significant contributions to the implementation and testing of our blended shared control implementation. Tarik Kelestemur has helped answer many of my basic vision and perception questions. Finally, Stephen Alt is like a little brother, and it has been a joy to mentor you.

All of these projects would not be possible without the constant backing of our support staff. Tracey Coetzee, Deb Baron, Andy Baron, and Marleny Ortiz from WPI were always there to support and cheer us on. Michael Jocelyn, Steve Keane, Fred Dellaporta, Eddie Tyrance, and Stephen Holland have helped support all the out of the box ideas we want to try at Northeastern. There are many more support staff at both institutions that have enabled us to do the work we do, and their efforts do not go unnoticed.

Finally, I would like to thank my committee members for the advice, discussions, and opinions that have shaped my years as a PhD student. I have known Prof. Sonia Chernova since I started at WPI, and I could always count on her to ask tough yet very thoughtful questions that have led me to a deeper understanding and appreciation for the design of experiments focusing on the interactions between humans and robots. I had met Prof. Deniz Erdoğmuş prior to moving to Northeastern, but he made us feel welcome as part of a larger research community at Northeastern when the lab first moved. His advice and support, especially with respect to the analysis of our blended shared control data, has been very beneficial.

Words cannot express the quantity and quality of guidance, support, and encouragement Prof. Taşkın Padır has provided me with over the last six years. They say that the most important aspect of a PhD is your advisor, and I could not have asked for a better mentor, advocate, and inspiration. Thank you for giving a chance to a student that everyone else passed on and providing me the environment to succeed.

Velin Dimitrov
April 24, 2017
Boston, MA
Abstract of the Dissertation

Model-Based Robot Control in Human-in-the-Loop Cyber Physical Systems

by

Velin Dimitrov

Doctor of Philosophy in Electrical and Computer Engineering

Northeastern University, May 2017

Dr. Taşkın Padır, Advisor

It is evident that over the last decade, complex problems and challenges in a diverse set of applications have led to the emergence of innovative approaches and research in human-in-the-loop cyber physical systems (HiLCPS). Robots play an integral role as physical agents within a HiLCPS and enable safer and more effective interactions within dirty, dull, and dangerous environments. Previous approaches and methods do not extend well this new class of systems due to an inability to incorporate human input and scale with the complexity and variety of components. As a result, there is a need to investigate, design, implement and validate novel shared control techniques that integrate humans and robots in a heterogeneous team to enable reliable, robust, and sufficiently agile interaction, communication, and operation. We posit that there are certain tasks humans are (and will be) superior to robots such as perception, intuitive control, and high-level decision-making, on the other hand, there are tasks robots can (or should) perform such as precise low-level motion planning, solving an optimization problem, and operating in hazardous situations. Therefore, the investigation of new control interfaces and shared control methods that can effectively delegate tasks and blend the control between the robot and human operator will enable us to field robot systems that act in direct support of humans.

It is possible to classify most robots that are currently deployed in applications in two categories: (i) fully autonomous robots performing specific tasks, and (ii) tele-operated robots solely dependent on operator input. We acknowledge that not all human-robot interaction fall into these two categories but they represent a wide majority of systems that are currently in use. As we attempt to close the gap in between these two classes, new control techniques are needed to dynamically shift the level of control between the human operator and the intelligent robot using system...
modeling, human-robot interaction, and system engineering tools within a holistic design framework. While completely autonomous exploration robots may someday be commonly utilized, we aim to show that selectively adding high-level shared control behaviors to execute real-world tasks can significantly improve time to completion with little addition of risk and system complexity.

We present a shared control architecture to enable the systematic design, modeling, and implementation of elements necessary for effective integration of robots in HiLCPS, with examples of systems in assistive, disaster, and space robotics. The implementation of shared control concepts in wheeled ground vehicles and bipedal humanoid robots are described in detail with emphasis on the challenges and problems encountered in their successful implementation. Additionally, contributions to the NASA RASC-AL Robo Ops Challenge, NASA Sample Return Robot Challenge, DARPA Robotics Challenge, and NASA Space Robotics Challenge are noted as examples of conducting research within the confines of competitive challenges. We identify common methods and concepts between the different applications of HiLCPS and show their progression and adaptation for different cross-domain implementations. The main contributions of the presented work are the common shared control framework, demonstrated in assistive, disaster, and space robotics, the addressing of challenges in its implementation under various constraints, and the best practices for integration on physical robot systems discovered and validated through testing in simulation and real-world scenarios.
Chapter 1

Introduction

1.1 Introduction

It is evident that over the last decade, complex problems and challenges in a diverse set of applications have led to the emergence of innovative approaches and research in human-in-the-loop cyber physical systems (HiLCPS) [83]. Robots play an integral role as physical agents within a HiLCPS and enable safer and more effective interactions within dirty, dull, and dangerous environments. Previous approaches and methods do not extend well this new class of systems due to an inability to incorporate human input and scale with the complexity and variety of components. As a result, there is a need to investigate, design, implement and validate novel shared control techniques that integrate humans and robots in a heterogeneous team to enable reliable, robust, and sufficiently agile interaction, communication, and operation. We posit that there are certain tasks humans are (and will be) superior to robots such as perception, intuitive control, and high-level decision-making, on the other hand, there are tasks robots can (or should) perform such as precise low-level motion planning, solving an optimization problem, and operating in hazardous situations. Therefore, the investigation of new control interfaces and shared control methods that can effectively delegate tasks and blend the control between the robot and human operator will enable us to field robot systems that act in direct support of humans.

HiLCPS are a broad class of systems of systems that generally include a human input to the system or one of the subsystems integrated within the context of a traditional cyber physical system. Breaking the HiLCPS term down, the human-in-the-loop part defines the system as semi-autonomous, adaptive, and dynamic due to the introduction of an impossible to deterministically model human input. The second part of the term refers to cyber physical systems, a rapidly growing
CHAPTER 1. INTRODUCTION

area of research over the last decade or so, that focuses on the seamless integration of computational
algorithms and physical components. This definition of HiLCPS covers any robotic system or set
of systems that interact with the environment and include a human in some way. The human may
be relegated to a trivial operator role or may be an equal partner in the system interacting closely
with the computational and physical elements of the robotic system. Recent HiLCPS application
areas cover a vast spectrum and have included wheelchairs for locked-in individuals [28, 38, 37],
behavior aware energy management systems [2], exoskeletons [51], and smart environments [23].

The process for modeling and control of HiLCPS is different from the approach traditionally taken in robotics. Elements of the system, especially those relating to the human input, are
difficult to model and do no lend themselves well to traditional control methods. Our terminology
in this thesis assumes a broad description of modeling and control. There are two types of models
both described in Chapter 2: the shared control framework as a model for the interactions between
different elements of the HiLCPS and a blended shared control model describing the blending of
human and autonomous input in a dynamic manner. Two types of control exist respective to the
type of model being considered. In the shared control framework, control refers to the information
flows through the framework and their integration into the individual blocks. In the blended shared
control implementation, control is referred to in the more traditional sense of how user input affects
the system state variables as the system progresses through its operation. This is a broader inter-
pretation than traditionally accepted for modeling and control, but necessary given the challenges
posed by HiLCPS.

It is possible to classify most robots that are currently deployed in applications in two
categories: (i) fully autonomous robots performing specific tasks [89, 56, 29], and (ii) tele-operated
robots solely dependent on operator input [7, 60, 14]. We acknowledge that not all human-robot
interaction fall into these two categories but they represent a wide majority of systems that are cur-
rently in use. As we attempt to close the gap in between these two classes, new control techniques
are needed to dynamically shift the level of control between the human operator and the intelli-
gent robot using system modeling, human-robot interaction, and system engineering tools within
a holistic design framework. While completely autonomous exploration robots may someday be
commonly utilized, we aim to show that selectively adding high-level shared control behaviors to
execute real-world tasks can significantly improve time to completion with little addition of risk and
system complexity.

The development of HiLCPS has also led to the emergence in new challenges when trying
to apply traditional tools to solve the associated problems. HiLCPS tend to involve collaborative
CHAPTER 1. INTRODUCTION

development efforts with contributions from a wide variety of disciplines. Each discipline brings it own associated set of tools, metrics, and language which make collaboration more difficult than within a discipline. Understanding a key concept in one domain will not be sufficient for researcher in HiLCPS. The tools to encourage cross-domain collaboration are either non-existent or insufficient today.

In addition to a variety of disciplines, HiLCPS commonly span several different scales, from fine granular details at the lowest level to abstract coupling and interactions at the highest levels, at the same time. The flow of information and interfaces between the levels need to be well understood to grasp the operation of the HiLCPS but without a formal methodology to analyze those interactions the complexities of the systems are lost or ignored as disturbances.

1.2 Challenges in HiLCPS

Because of their nature, with abstract elements in the cyber sphere and concrete objects in the physical environment, interactions within the human-in-the-loop CPS require design and analysis through different methods than traditional systems. An additional complexity arises from researchers using an entirely different language to describe the same core principle in a different domain. In [12], the lack of communication between different communities was directly tied to a lack of common language. Ultimately, one of the goals of research in HiLCPS should be to focus on an architecture and associated methodologies to enable and encourage the cross-pollination of ideas, approaches, and solutions between the domains. Many concepts in HiLCPS are shared across domains, but several challenges exist making the jump across domains more difficult and ultimately limiting the potential impact of novel concepts.

At the 2013 IEEE Systems, Man, and Cybernetics Conference held in Manchester, UK, a workshop was held on shared control where almost every system presented would be considered a HiLCPS. Application domains included haptic control for more efficient deep sea exploration, control of multiple rovers in a space environment, assistive curve negotiation in vehicles, and assistive robotics to improve quality of life. Modeling these HiLCPS within each domain is a challenge because the systems are complex, information travels through many pathways within the system, information interfaces are not traditional, and the systems tend to be hybrid in nature with very asynchronous communication. As highlighted in the research literature, comparison of these systems is possible within the same application domain, but currently very difficult or impossible in some cases across different domains.
CHAPTER 1. INTRODUCTION

The haptic feedback that works to enable more intuitive curve negotiation in a vehicle may also be beneficial to an operator controlling a space rover worlds away hampered by communication delays and bandwidth limitations. Many of these systems are employing concepts of model-based control and design, where operators are directly interacting with a virtual model of the system as opposed to directly with the system. This is a powerful concept, and each domain currently has its own approach to implementing such systems. The concepts similar between the domains, but divergent enough in implementation to limit their applicability.

In addition, many application domains have similar challenges with respect to performance metrics, optimization, and scalability that can be addressed in a common architecture since the goals are similar across the domains. Performance metrics for traditional systems, such as transient response, bandwidth, settling time, and disturbance rejection have been well studied, but are not necessarily as effective describing HiLCPS because of the challenges outlined above. A set of performance metrics that works well for describing HiLCPS, especially the coupling within the system, is extremely beneficial. The right metrics allow direct comparison of HiLCPS across application domains. Significant previous work exists in performance metrics for human-robot systems such as in [90, 108, 24, 18, 20, 62, 82] which can be extended and modified to work in HiLCPS applications. [97] provides a survey of performance metrics focusing on assistive robotics technologies.

The performance metrics would have an added benefit of enabling a better understanding of optimization in HiLCPS. Traditional optimization methods from control theory work well, but are sometimes difficult to implement in HiLCPS because most rely on good system models or good observability in the system. Many HiLCPS because of complex coupling, are difficult to model and the system may not be fully observable or deterministic, especially with respect to the human operators, and will likely be highly nonlinear. Good performance metrics of the system can encourage the development of new control and optimization methods for HiLCPS. For example in [26], the effects of changing reliability are studied and how that affects the implicit trust in the robotic system from the user’s perspective.

Finally, most of the HiLCPS systems need to integrate the concept of scalability: both in terms of the quantity of systems within the HiLCPS and the capabilities of those systems. Conditions in many HiLCPS change quickly and the systems need to dynamically adapt to continue operating in the desired mode. The modeling and control methods need to be able to dynamically scale with respect to the number of individual systems that compose the HiLCPS and their capabilities. Systems need to be able to come online and offline within minimal disruption to the HiLCPS
CHAPTER 1. INTRODUCTION

A common architecture can enable the comparison of scaling strategies in different HiLCPS.

1.3 Prior Work in Shared Control

Shared control within the context of robotics has been studied as early as the late 1980s [44, 5, 57, 22, 103, 88]. Early work was based on implementations where the robot control was transferred between teleoperation and autonomous modes with the goals of addressing transmission time-delays [44], developing context-aware navigation procedures, characterizing efficiency in human-robot interaction [22]. More recently, research results have been reported that implement shared control architectures based on speech recognition [81] and brain-computer interfaces [96]. Others such as in [17] have considered how the level of control can be dynamically modulated between human and autonomous input as the system completes different tasks. [28] presents the development of a modular system to enable quick retrofitting of existing wheelchairs enabling semi-autonomous operation. The presented work complements the previous results reported in literature by treating the human-robot team as a whole system on its own and develop models and control techniques within this architecture.

In [86], Sheridan introduces many of the concepts that would shape shared control in the coming years. He defines the concepts of traded and supervisory control, where control of the system in question is passed off back and forth between autonomous control and user control. In general the architecture that Sheridan proposes allows human input to the system to occur at a high, abstract level which is how shared control has been considered. In addition, tasks in the architecture are generally delegated to the human or robot system. The split between human and robot tasks is application dependent, and needs to be carefully considered depending on the specifics of the applications.

One of the biggest challenges that Sheridan highlights is how to enable the system to efficiently complete a task shared between human and robot, but at the same time allow the human to explore the operation of the system. The natural exploration process any human operator undergoes leads to errors in task completion, that the human operator monitors and learns from. Thus an interesting phenomena occurs where the human must be allowed to diverge from what may be optimal so that they can acquire an intuitive understanding of how the system works. This results in a short-term versus long-term trade-off in terms of how much error is acceptable and how quickly the exploration and learning process should ideally progress.
CHAPTER 1. INTRODUCTION

Later Sheridan updates the concepts and approaches to holistically design and analyze systems that rely on both humans and automation in [87]. After case studies of human interaction and automation in aircraft, vehicle, nuclear power, and healthcare systems, in addition to others, Sheridan describes the details of design for systems that blend humans and automation. Metrics for performance such as speed, accuracy, robustness, trust, error, and workload are considered, and how they may be leveraged to analyze the systems.

Finally, Sheridan highlights various interfaces for providing awareness back to the operators, and making the best use of inputs to relay decisions to the system. Model-based prediction for operator feedback is advocated as a flexible and adaptable approach to improving operator situation awareness. In more recent work, immersive virtual and augmented reality experiences with flexible and dynamic interfaces were proven to be highly effective in planning for space missions [67, 32]. Even looking at the DRC Finals, human-robot interaction was limited mainly to traditional keyboard and mouse interfaces [68], and would likely have benefited from more innovative approaches to interface design if the timeline allowed.

In [92], Tilbury formulates a model predictive controller to enable a user to navigate a high-speed mobile base through a simulated environment. The control law merges the human operator input and steers the robot away from the obstacles that have been placed in front. They present results showing that the shared control implementation gives the user a higher degree of control authority of the system while reducing the risk of a collision with an obstacle. The shared control implementation outperforms both the traded control, where user and robot switch control authority, and the pure teleoperated scenarios. In addition, it allows for the control authority to be changed to those extremes if needed.

Finally, in [27] describes in detail another blended shared control system implemented through the INEEL robot control architecture in [10]. The system transitions through four separate modes that control the level of autonomy: teleoperation, safe, shared, and autonomous. The first and last mode are evident, while the other two pose interesting questions related to shared control. In the safe mode, the user is in control of the robot but the autonomous functions prevent the user from going into unsafe situations. In the shared control mode, the robot velocity is blended by a parameter similar to Tilbury’s approach. Our approach differs in that we dynamically change the blending parameter as described by Enes in [36]. In addition, the presented architecture enables finer control of the subdivision of control between human and robot depending on the type of shared control modality that is set.

Goil et al. utilizes a blended shared control approach similar to the one presented in
CHAPTER 1. INTRODUCTION

our work. The work differs though in the implementation of the blending parameter which mixes the autonomous and human control inputs. In [41], they present a system focused on assisting a wheelchair bound individual to navigate through a doorway in a shared control manner. The blended equations are the same as in our work, but the blending parameter is selected as a function of the covariance of angular input of a learned user model and the difference between the user and planner desired angular velocity. This approach learns the covariance expected in the user input when navigating through the doorway, and uses the learned model to select the blending parameter. When little covariance is expected, the robot has more control authority over the system whereas when more covariance is expected, the user is afforded more control over the system. The disadvantage of this method though is it reliance on the learned model, which is task specific and not necessarily generic depending on the specific tasks that the human-robot team is expected to complete.

Gopinath et al. follow yet another approach to solve the problem of blending parameter selection while using the same linear blending model. In [42], they implement a framework that allows the user to request more or less autonomy while completing a manipulation task with a robotic arm. The framework solves an optimization problem for the configuration parameters for how the blended shared control parameter should change. A measure of the systems confidence in the user’s intent is calculated and used to gradually change the blended shared control parameter between the configuration parameters that the optimization problem has solved. The framework is tested a series of able-bodied individuals as well as individuals with spinal cord injuries, and analyzes their performance in three manipulation-focused tasks.

1.4 Our Focus

We explore various schemes to consider and treat the human-robot team as a whole. This remains an open and exciting research area due to the challenges in modeling the uncertainties in human actions and hybrid nature of the system. Previous work has focused on understanding and modeling user intent to improve operator control of robotic systems [94, 103, 73, 55]. In the following work, we consider different control methods that constitute shared control specific to robotics in disaster response, space exploration, and assistive robotics applications. Specifically, examples of traded control, indirect shared control, coordinated control, collaborative control, and blended shared control techniques are described and explained in detail. A similar approach has been presented in [36] in which Zermelo’s navigation problem is the basis of a shared control architecture.

Disaster response robotics has been an intense area of research focus for the last few
CHAPTER 1. INTRODUCTION

decades. Search and rescue robots were deployed to the site of the World Trade Center after the September 11, 2011 terrorist attacks [18]. Used to search in crevasses and air pockets for survivors, the robot allowed search and rescue teams to stand a safer distance away and reach pockets that were unreachable otherwise. In that instance though, the search and rescue teams stayed close to the robots and could intervene if they encountered problems such as their tethers jamming in the debris.

More recently, the disaster at the Fukushima Daiichi power plant brought the attention on disaster response robotics back into focus. Unfortunately, the explosions in the reactor left debris, radiation, and other hazards that severely limited the utility of the robots sent in the aftermath of the disaster. [65] details many of the challenges that were resolved before robots could be deployed in inspection tasks at the damaged reactors. Despite the promise of robotics to reduce risk and exposure to workers, they are very limited by the challenges of presented by real-world disasters. The DARPA Robotics Challenge [75] was organized following the Fukushima disaster, and greatly propelled disaster response robotics forward. In addition though, it highlighted the shortcomings of robotics, especially with respect to operator’s abilities to successfully control them in challenging time delay scenarios with degraded infrastructure impeding the robots [68].

In a different application domain, the same challenges of time delay and challenging terrain also present themselves limiting humanity’s ability to explore farther into space. Many scenarios in robotic planetary exploration require interaction of multiple agents, both human and robotic, but such interaction is impeded by significant time-delays and bandwidth restrictions of the communication channels. Our research focuses on the intersection between autonomous and teleoperated systems, and more specifically on how autonomous behaviors in a system can be combined with user input to enable shared control modalities previously not possible. Prior robotic missions to the Moon and Mars have followed mostly teleoperated mission scenarios, consisting mainly of highly scripted and preplanned task sets uploaded for semiautonomous execution.

As humankind’s exploration capabilities on the Moon and Mars continue to improve [43], and as missions evolve to include multiple robotic platforms (including rovers, micro-rovers, hoppers, and orbiters) with varying complexity and resources, a paradigm shift in mission design approach is required to rely more on the autonomy of these systems. While this has proven to be an effective approach especially for minimizing mission risks, it poses significant challenges when scaled to multiple systems. In order to effectively implement shared control operation though, more robust and reliable autonomous behaviors are needed. In addition, these behaviors must be validated in a variety of different environments so their actions are well understood.
CHAPTER 1. INTRODUCTION

We believe the limitations that these robots encounter in conducting disaster response or space exploration missions are not due to physical limits. There are opportunities given robust autonomous capabilities, intuitive user interfaces, and a shared control architecture that provides a structured method for humans and robots to work in a coherent heterogeneous team to enable future HiLCPS to alleviate these challenges. The principles to put these robots to work in the real-world exist, and the right way to leverage them needs to be experimentally derived, both in simulation and hardware. According to [83] the future of shared control HiLCPS leverages transparent input interfaces, good context-aware models to enable human intent inference, a well-executed shared governance of the system, modularity/re-configurability, and distributed architectures.

The lack of an architecture that can effectively model cross-domain HiLCPS implementations limits the ability of shared control HiLCPS to leverage the concepts outlined. Such an architecture needs to be developed and designed to be generic enough to be functional in different application domains, but at the same time have specificity allowing the intricacies of specific implementations to be captured. Our work present a similar shared control architecture to both Tilbury and Desai’s formulation in Chapter 2 where we leverage the kinematic model of the system. Instead of the formulating the user input in the control law for the system, we formulate the user input as a constraint on the kinematic model. In addition, we consider a system that is redundant with respect to the task space dimensions, so the redundancy can be taken advantage of to implement a primary and secondary task. We integrate the human input to implement the shared control implementation in one of these tasks depending on the specific application.

1.5 Contributions

The main contributions of the presented work are the common shared control framework, demonstrated in assistive, disaster, and space robotics, the addressing of challenges in its implementation under time delay, bandwidth limitations, and disturbances, and the best practices for integration on physical robot systems discovered through testing in simulation and real-world scenarios. The shared control architecture enables the systematic design, modeling, and implementation of elements necessary for effective integration of robots in HiLCPS with examples of systems in assistive, disaster, and space robotics [31]. The implementation of shared control concepts in wheeled ground vehicles and bipedal humanoid robots are described in detail with emphasis on the challenges and problems encountered in their successful implementation.
CHAPTER 1. INTRODUCTION

- A common shared control framework demonstrated in the SRR, DRC, and SRC
  - The SRR, while an autonomous challenge, contributed significantly to the development of robust autonomy focused on navigation in GPS-denied environments and manipulation of known and unknown objects.
  - The DRC contributed to the development of shared control methods to enable a humanoid robot to drive an unmodified vehicle with severe bandwidth and latency restrictions.
  - The SRC effort contributed to the generation of the first full-size humanoid data set focused on bridging the gap between simulation and real-world performance to enable higher fidelity cloud based simulation and context aware methods in the future.

- Addressing of time delay, bandwidth limitations, and unmodeled disturbances frequently encountered in HiLCPS shared control implementations
  - The effects of time delays on shared control, especially between user input and autonomous system, are explored through a user study with real robot hardware.
  - Bandwidth limitations are addressed through the use of aggressive compression and extensive testing on the sample operations with ORYX 2.0 and driving with the ATLAS robot.
  - The effects of unmodeled disturbances such as drift caused by shifting terrain or wind in outdoor environments are characterized through the user study as well as from direct, hands on experience with ORYX 2.0 and ATLAS.

- Best practices for testing and development of shared control in simulation and real-world
  - Experimentally derived and tested shared control approaches are intended to provide templates for future developments in similar HiLCPS

In addition, addressing the challenges of shared control implementation under time delay, bandwidth restrictions, and unmodeled disturbances is a significant contribution of the presented work. We identify common methods and concepts between the different applications of HiLCPS and show their progression and adaptation for different cross-domain implementations. The application areas outlined above all have constraints that can be broadly grouped in categories. For example, bandwidth and latency restrictions are common constraints that we encountered in all
CHAPTER 1. INTRODUCTION

three application areas. The distances required for interplanetary communication, the degraded infrastructure of a disaster site, and the physical limitations of individuals with disabilities all present challenges to effective implementations of shared control.

Finally, contributions to the NASA RASC-AL Robo Ops Challenge, NASA Sample Return Robot Challenge (SRR), DARPA Robotics Challenge (DRC), and NASA Space Robotics Challenge (SRC) are noted as examples of conducting research within the confines of competitive challenges. The rules and regulations of competitions reinforce the use of best practices for integration of shared control on physical robot systems in order to be successful. These best practices are discovered through extensive testing and validation in simulation and real-world scenarios. For example, the WPI-CMU DRC team was the only ATLAS team to complete the driving task at the DRC Trials, and the fastest team out of all the teams to complete the task. This success is attributed to a specific testing plan including full-scale testing when needed utilizing the best practices learned in the previous incremental testing. The process for developing, testing, and refining the shared control implementation resulted in a set of best practices that can be used as guidelines for development of future shared control implementations.

This dissertation is organized as follows: Chapter 2 introduces our common shared control framework in detail. Each block’s functionality and applicability is outlined along with descriptions of the interconnections between the individual blocks. A formulation for how human and autonomous input can be merged to implement a blended shared control algorithm on a redundant system is then presented. Finally, a series of experiments demonstrating the performance of the blended shared control approach on navigation task implemented on a small differential drive robot are presented. Chapter 3 details the development of the Oryx 2.0 and AERO rovers for the NASA RASC-AL Robo-Ops Challenge and NASA SRR Centennial Challenge, respectively. The contributions of each project to the further development of the shared control framework are highlighted. Chapter 4 similarly describes the work and contributions to shared control of the Atlas and Valkyrie humanoid robots for the DRC and NASA SRC respectively. Additionally, the first full-size humanoid dataset and its implications for the shared control framework are discussed. Finally, Chapter 5 offers a concise overview of the contributions and possible future directions enabled by the presented work.
Chapter 2

Shared Control

The shared control framework we present here is a product of the experiences, approaches, and methods that we have employed in space exploration, disaster response, and assistive robots. We describe the specific lessons of those experiences in the subsequent chapters, but first we introduce the framework that aims to integrate humans and robot into a heterogeneous, collaborative team to solve a given task not well suited to humans or robots by themselves.

2.1 Shared Control Architecture

Any shared control architecture needs to be easily adaptable and extensible to grow as research thrusts push in different directions. The architecture should be able to accommodate standard and non-traditional metrics that can be used to evaluate human-in-the-loop HiLCPS and are relevant in cross-domain applications. Concepts such as trust between agents in the system, the performance of the system, the control effort required within the system, and efficiency of control need to be easily quantified and described within the architecture. In addition to these interface-level metrics (metrics between subsystems within the HiLCPS), metrics internal to the subsystems should be available to encourage system-level optimization and model-based control of these HiLCPS. Complex intricacies in the systems can then be easily evident as opposed to being buried in the details and coupling of the individual subsystems.

The structure of the architecture is split into quadrants related to where (cyber and physical) each component resides and who (human and robot) contributes the information in each component. The components in the cyber realm operate in an abstract area that could be in the internet (on a cloud platform for example), could be on the robot, or could be distributed across several different
CHAPTER 2. SHARED CONTROL

areas. The cyber realm is not characterized by physical location but the availability of significant computation power and bandwidth within the realm, where heavy computing and intelligence can be easily implemented. The components in the physical realm are associated with the tangible objects of the system such as the on-board computing of the systems and interfaces the operators utilize.

A second categorization splits the architecture between components that directly utilize mostly human-centric information or robot-centric information. The components base their outputs on the information that is available, so this is a natural dichotomy based on how the control within the system is split between the autonomous agents and the human-driven ones. The human robot interface provides an abstraction for the information transfer between the two realms encompassing information generated by the operators and also automatic contextual information from the other elements. It is separate, but includes information from the operator interface. In reality, these boundaries are fuzzy and not clear-cut defined in a real system, but such a categorization helps with the comprehension of information sources and flow within the system.

![Diagram of shared control architecture](image)

Figure 2.1: The shared control architecture consisting of the knowledge base, action engine, achievable action gate, perception engine, low-level robot control, human robot interface, context engine, cloud engine, and operator interface. Not all HiLCPS will necessarily implement all blocks. The architecture is split in both cyber and physical domains, and also human and robot domains depending on where elements are located and who contributes information to each element.
CHAPTER 2. SHARED CONTROL

Figure 2.1 shows the proposed architecture that corresponds to the features outlined above. The architecture is composed of several elements that we have identified to be common to HiLCPS: a knowledge base, action engine, perception engine, achievable action gate, robot control, human-robot interface, context engine, cloud engine, and operator interface. The architecture is by no means complete, not all HiLCPS will necessarily include each element, and will expand, consolidate, and change as research development progresses. It should be noted that the robot does not need to be a robot in the traditional meaning of the word. Any autonomous agent would fit the architecture as well.

The Knowledge Base stores global strategies and approaches for the system, providing options to the action engine further down the line. The knowledge base is aware of the history of the action engine, and provides the high-level abstract goal that the system should try to achieve along with which control modality is most appropriate to accomplish the given task. For most systems, the knowledge base operates on an infrequent and asynchronous time frame, changing goals and approach on an event-based principle. Because of this, the knowledge base can store an immense amount of information and has the time to search through it to create a relevant plan for the action engine. In addition to the history of the action engine, the knowledge base receives information from the context engine to adjust global strategy based on the context of the environment, human operator, and system model. As the system operates and encounters various scenarios, the knowledge base stores information on which approach performed well based on the HiLCPS metrics so the system dynamically adapts to changing conditions.

The Action Engine takes the global strategy and plan given by the knowledge base and generates a set of potential actions that the robots can take to achieve the tasks. The action engine has access to the information from the perception engine, which provides limited information on the state of the robot, and passes its desired actions to the achievable action gate. The action engine, depending on the HiLCPS, may be completely in the cyber realm or may be infringing in the physical realm as well, especially if it is distributed in terms of computing. The action engine is where the different control modalities are implemented, as shown in Figure 2.2. The HiLCPS can be teleoperated, controlled through supervised or guarded autonomy, shared controlled, or fully autonomous. In many situations, the context engine (through the knowledge base) will affect which control modality is most appropriate in a given scenario.

The Perception Engine has a high-bandwidth, low-latency link with the robot sensors collecting and sorting the information from those sensors. The primary purpose of the perception en-
CHAPTER 2. SHARED CONTROL

Figure 2.2: The action engine provides desired actions for the robots to complete. Various control schemes can be implemented here including teleoperated control, supervised or guarded autonomy, shared control, and autonomous control.

gine is to interpret the sensor information and pass along only the information relevant to generating a set of desired actions to the action engine. Any system-wide state estimation (not robot-specific state estimation) should occur in the perception engine since it is the first place that information from all the robots is aggregated. In addition, the perception engine should be able to detect failures in the individual robot so the desired actions can be dynamically adjusted. Methods to detect soft failures, where sensor information is still available but is of questionable credibility, should also be present to arbitrate discrepancies between the individual robots.

The Achievable Action Gate takes the desired actions provided by the action gate and checks if they are actually achievable and relevant for the system. For example if the action engine requests a set of actions that are out-of-bounds for a given robot, the action gate will modify those actions so they are as close to the desired ones as possible but within the capabilities of the robot. The action gate is especially important for the teleoperation control modality since the action gate will limit the ability of the operator to accidentally command the robots into potentially dangerous situation either for the environment or the robots themselves. The operator can change the desired actions through the action gate depending on the control modality.

The Robot Control block accepts the actions from the action gate and translates them to low-level commands that the motion controllers on each robot can accept and accomplish. We have identified that many HiLCPS have parallel and series control loops, and this is one of the
CHAPTER 2. SHARED CONTROL

Figure 2.3: An architecture for HiLCPS robot controllers. Each robot may have different controllers providing different inputs at different time-scales. In addition, there may be coupling between the controllers as well. The architecture allows for both parallel and series combinations of controllers to be modeled.

intricacies that should be captured to enable comparison of cross domain HiLCPS. Figure 2.3 shows the proposed architecture for modeling robot controllers. Individual controllers can provide inputs directly to the system or go through another controller, enabling it model both parallel and series combinations of controllers. For simplicity, the diagram shows only two controllers, but it can be scaled depending on the complexity of the system. The switch, which form the parallel series structures, in many cases are set in a permanent state by the specifics of the system, but may be dynamic in certain cases if the system switches architecture or modes. In addition, the controllers do not need to be on the same timescales. $t_1$ and $t_2$ provide independent triggers that enable the controllers to run on different timescales.

The Cloud Engine receives limited state information about the robot system through the robotic system through the human robot interface and passes the information to the cloud, harnessing the combination of significant human and computational resources, to provide simulations and model-based awareness and control improvements. For example, a simplified model of the robot system can be run in the cloud engine, controlled in simulation through a crowd-sourcing platform, and those results can be passed to the context engine, providing a better sense of the environment, or to the operator interface. The cloud engine, like the action engine, can straddle the grey boundary between the cyber and physical realm with parts in an abstract computation environment and parts
CHAPTER 2. SHARED CONTROL

implemented on the operator side.

The Context Engine receives information from the human-robot interface similar to the cloud engine to understand the environment and situation around the robots. In addition, it can utilize the information from the cloud engine to help decipher the information and context in an accurate and efficient manner. The contextual information is sent to the knowledge base to be recorded and included in the plan later sent to the action engine. This step is critical to enable HiLCPS that are aware of their environment and can dynamically adapt to changing and dynamic conditions.

Finally, the Operator Interface serves as the carrier of information between the HiLCPS and the operator or potentially multiple operators. The interface can change based on the information provided by the cloud engine. Model-based awareness algorithms running in the cloud engine can change the perspectives of the information presented on the operator interface. The operator interface provides only a subset of the information flowing through the human robot interface, the context engine can send contextual information directly through as well. All information through the operator interface does not necessarily need to be validated by the operator always, but it is available for introspection if need.

Despite the simple nature of the diagram, we feel this is a preliminary implementation of a very powerful approach to model shared control of space exploration systems that has potential impacts across the whole range of application domains. It can enable the cross-pollination of ideas and approaches, the ability to compare and contrast ideas in common structure, and allow researchers to describe shared control systems in a common language. One of the central challenges of the framework is the integration of the human input with the autonomous behaviors across the human robot interface inside the action engine and achievable action gate.

In the next sections, we describe the theory behind one approach for enabling shared control by integrating the human inputs as constraints on the kinematics of the system. Figure 2.4 highlights the elements of the shared control framework that the kinematic redundancy resolution formulation addresses. User and autonomous input to the system is blended through a combination of primary and secondary tasks leveraging the redundancy in the the system. The top part of the shared control framework, focusing on the action engine, achievable action gate, and robot control blocks provides the autonomous input depending on the shared control modality to be implemented, and the operator interface provides the human input. The achievable action gate fuses the two inputs based on the formulation below.
CHAPTER 2. SHARED CONTROL

Figure 2.4: The blocks highlighted in green are developed in the redundancy resolution-based formulation for integrating user input and autonomous behavior in the shared control system. The formulation includes splitting the task into primary and secondary tasks that can be simultaneously achieved leveraging the redundant DOF.

2.2 Redundancy Resolution

For the sake of simplicity, we will demonstrate the concepts with a simple 5-DOF example, a robot whose base is constrained to the plane (3-DOF) with a 2-DOF arm consisting of a panning and tilting shoulder joint. The end-effector of the arm can be off the plane that constrains the robot base. The variable name conventions and general methodology follows the formulation in [71] and the key equations are reproduced here for easy reference. Because of the vector and matrix representations, the concept scales easily to higher DOF systems. The pose of the robot and the joint angles of the robot form a set of generalized coordinates in Eqn. 2.1, where the terms are lateral displacement, forward displacement, robot orientation, and the two joint angles of the arm respectively.

\[
\eta = \begin{bmatrix} x_v & y_v & \theta_v & \theta_1 & \theta_2 \end{bmatrix} \tag{2.1}
\]

The task space coordinates are accordingly in Eqn. 2.2. The lateral displacement, forward
displacement, and orientation are trivial given the robot is constrained to the plane. The $z_q$ term is only used to describe the height of the end-effector on the arm above the ground, since the arm is not constrained to the plane.

$$p_q = \left[ x_q \ y_q \ z_q \ \theta_q \right]$$ (2.2)

The rest of the section highlights the important results of the derivation from [71] that we utilize to implement the shared control through the redundancy inherent in the system. The kinematic constraints can be summarized in a very compact form reproduced in Eqn. 2.3.

$$A(\eta)\dot{\eta} = 0$$ (2.3)

Depending on the specific task we want to accomplish, the 5-DOF system may be redundant and we will demonstrate several examples of shared control implemented by introducing pseudovelocities that make use of the redundancy in the system. Pseudovelocities are formed by the transformation matrix $B(\eta)$ such that

$$\dot{\mu} = B(\eta)\dot{\eta}$$ (2.4)

where $\dot{\mu}$ is the pseudovelocity vector and the matrix $B$ is chosen such that $\begin{bmatrix} A^T & B^T \end{bmatrix}^T$ is invertible.

So, we can write,

$$\begin{bmatrix} A \\ B \end{bmatrix}^{-1} = \begin{bmatrix} \Sigma & \Gamma \end{bmatrix}$$ (2.5)

where $\Sigma$ and $\Gamma$ satisfy the following conditions: $A\Sigma = I_3$, $B\Sigma = 0$, $A\Gamma = 0$, and $B\Gamma = I_5$.

We resolve the redundancy in the system through the Moore-Penrose pseudoinverse where $\zeta$ is the null-space vector.

$$\dot{\eta} = \Gamma\dot{\mu} = \Gamma(J_q\Gamma)^\dagger\dot{p}_q + \Gamma\{I - (J_q\Gamma)^\dagger(J_q\Gamma)\}\dot{\zeta}$$ (2.6)

If we want, we can accomplish a primary task that is of lower dimensionality, and simultaneously specify a secondary task to implement simultaneously [66]. We will utilize this formulation to represent the human input to the shared control system as either part of the primary or secondary
task. Let’s assume that we can formulate the primary and secondary task by relating the pseudo-velocities to the task space points that correspond to the task through the appropriate Jacobian matrix.

\[
\dot{p}_p = J_p(\mu)\dot{\mu} \quad (2.7)
\]
\[
\dot{p}_s = J_s(\mu)\dot{\mu} \quad (2.8)
\]

Then the equivalent result to Eqn. 2.6 including both the primary and secondary tasks becomes,

\[
\dot{\eta} = \Gamma \{ J_p^\dagger \dot{p}_p + (I - J_p^\dagger J_p) \tilde{J}_p^\dagger (\dot{p}_s - J_s J_p^\dagger \dot{p}_p) \\
+ (I - J_p^\dagger J_p)(I - \tilde{J}_s^\dagger \tilde{J}_s)\dot{\zeta} \} \quad (2.9)
\]

Eqn. 2.9 represents the generalized velocities required to accomplish both the primary and secondary tasks through the redundancy resolution. By formulating the tasks in Eqns. 2.7 and 2.8, we can implement different shared control modalities and using Eqn. 2.9 we can tell the robot in joint space how to achieve the tasks.

2.3 Shared Control

In [36], Zermelo’s navigation problem is used as an example to consider a range of methods as to how shared control may be implemented including traded control, indirect shared control through cues, coordinated control, collaborative control, virtual constraint, and blended shared control. This categorization is very thorough and covers a wide range of possible shared control modalities. We use these categories as a template to show how each shared control modality, with the exception of indirect control through cues, can be implemented using the primary and secondary task allocation from the previous section. We leave out the cue-based control modality because it involves modeling the “internal” control loop of the human response, and that would be better accounted for in a dynamical model.

Traded control involves having the autonomous agent and human agent trading off control of the platform on a time-based or event-based trigger. The system can either be in full autonomous mode or under complete teleoperation, with no intermediate option. This can be simply expressed with a couple of cases,
CHAPTER 2. SHARED CONTROL

\[
\dot{p}_p = \begin{cases} 
\dot{p}_u & \text{when } t_1 = 1 \text{ and } t_2 = 0 \\
\dot{p}_a & \text{when } t_1 = 0 \text{ and } t_2 = 1 
\end{cases}
\]  

(2.10)

where \(t_1\) and \(t_2\) correspond to the time or event triggers on the parallel control loops shown in Fig. 2.3 allowing the robot control block to transfer between teleoperation and fully autonomous operation. \(\dot{p}_u\) and \(\dot{p}_a\) correspond to the task space velocity inputs corresponding to the user and autonomous system respectively. Since full control of the platform is exerted, both inputs are 5-DOF, essentially the derivative of \(p_q\) from Eqn. 2.2. There is no secondary task, since the primary task is a full 5-DOF input and there is no inherent redundancy.

An expected use case for this type of control for a rover would be when either a task is very delicate, difficult, and not very dynamic, so the user has full control of the platform or very dull, so full autonomous control can be given to the rover.

**Coordinated control** involves easing the human operators task, by allowing them to control the robot in a lower dimensionality. The robot takes care of mapping the lower dimension input to the full system. An example of this can be envisioned where the camera the operator is using for situational awareness and driving is mounted on the first link of the arm. The operator may want to keep the camera view steady as the robot is moving, then the operator is providing a 3-DOF input consisting of the desired translation in the plane and the desired camera orientation. The robot resolves the redundancy between the rotation of the arm and base to achieve the desired platform motion while holding the camera view. With this in mind, the primary and secondary tasks are described by \(\dot{p}_p\) and \(\dot{p}_s\) as follows,

\[
\dot{p}_p = \begin{bmatrix} x_v \\ y_v \\ \dot{\theta}_1 \end{bmatrix}
\]  

(2.11)

\[
\dot{p}_s = \begin{bmatrix} \dot{\theta}_v \\ \dot{\theta}_2 \end{bmatrix}
\]  

(2.12)

In Eqn. 2.11 the user input provides the primary task as the desired position of the robot and desired direction to point the camera mounted to the arm. The secondary task, Eqn. 2.12 takes care of minimizing the rotation in the base since it is easier to pan the arm rather than the whole base and also keeping the end-effector off the ground.

**Collaborative control** is very similar to the coordinated control above, but the user is given a lower dimensionality of inputs that is a direct subset of the joint space of the robot as opposed
CHAPTER 2. SHARED CONTROL

to an arbitrarily defined linear combination. In this case, the user is given full control of the robot base position, orientation, and arm orientation. The robot controls the arm elevation to keep the arm above the ground. This mode would be very useful for alignment of a sample before attempting to collect the sample with the end-effector, a task we identified as a good candidate for additional automation in [30]. The primary and secondary tasks are described by,

\[
\dot{p}_p = \begin{bmatrix} \dot{x}_v & \dot{y}_v & \dot{\theta}_v & \dot{\theta}_1 \end{bmatrix} \\
\dot{p}_s = \begin{bmatrix} \dot{\theta}_2 \end{bmatrix}
\] (2.13)

(2.14)

Eqn. 2.13 assigns the primary task for the robot to follow the commands of the human operator for positioning the robot base and orientation of the arm. Eqn. 2.14 assigns the secondary task to the robot to keep the end-effector above the ground so the user can slowly "bump" the platform or arm into a better position to pickup the sample.

Virtual constraint control involves defining a constraint, usually in task space, that limits the input the user can have on the system. The user has full control of the robot as long as the constraint is fulfilled. For example, assume that a ground-facing sensor is mounted on the arm and a survey needs to be conducted in front of the robot in a series of lines perpendicular to the direction of travel for the robot. The sensor height needs to be modulated above the ground to either follow terrain or conduct the survey at different heights. Because the survey lines need to be straight, but the arm is a fixed length, we can utilize the redundancy to follow the survey line with the end-effector while allowing the user to guide the location of the line and sampling point on that line without worrying about following the line itself. The primary and secondary tasks are described by,

\[
\dot{p}_p = \begin{bmatrix} \dot{x}_v & \dot{y}_v & \dot{\theta}_v \end{bmatrix} \\
\dot{p}_s = \begin{bmatrix} \dot{\theta}_2 \end{bmatrix}
\] (2.15)

(2.16)

Eqn. 2.15 defines the primary task of keeping the end-effector over the line by moving the base within the plane, and rotating the first link of the arm so the sensor is over the scan line. This primary task is driven by the user input which is a desired location of the line in front of the robot, and the desired location on the scan line that needs to be surveyed. Eqn. 2.16 sets the secondary task to minimize the rotation of the base (implemented by the controller minimizing vibrations so the sensor is relatively undisturbed during scanning) and also keep the sensor at the desired height.
A sensor that might require this type of operation would be a ground penetrating radar (GPR) used to create a 3-D model of the subsurface strata.

Blended shared control is the most collaborative modality where the human input and autonomous input are both controlling the aspects of the robot simultaneously. Implementation of blended shared control is difficult because the two inputs may be conflicting, and such cases must be easily detected. This type of control would be useful when approaching a region of interest that the rover has autonomously detected as something that needs to be further explored. The region may not be well defined, so a little assistance from the human operator would help guide the rover to the right place. In this case, both the human and robot drive towards the region of interest. If the two inputs are essentially going towards the same goal, the robot constantly receives positive reinforcement that it is on the correct path. On the other hand if the goal of the two inputs diverges too much, the robot can realize that there is a misunderstanding between where the human operator wants to explore and where the robot thinks it should be going. The primary task is a composition of both human and autonomous input described by,

\[
\dot{p}_p = \alpha \dot{p}_u + (1 - \alpha) \dot{p}_a = \begin{bmatrix}
\alpha \dot{x}_u + (1 - \alpha) \dot{x}_a \\
\alpha \dot{y}_u + (1 - \alpha) \dot{y}_a \\
\alpha \dot{\theta}_v,u + (1 - \alpha) \dot{\theta}_v,a
\end{bmatrix}
\]

\[
\dot{p}_s = \begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2
\end{bmatrix}
\]

In Eqn. 2.17 both the user input and robot behavior are providing the translation motion and heading the robot should be facing. This is similar to the shared control formulation in [27] and later further developed in a model predictive controller in [92]. The approach presented here is tested on a real robot in Section 2.4 under time delay and drift scenarios as opposed to the simulation based approach taken in [92]. The two inputs are averaged, and if the user and robot are heading in the same general region, the two vectors and their average should be very similar and thus the robot will head in that direction constantly reassured by the operator input that it is behaving correctly. Eqn. 2.18 takes care of keeping the arm out of the way while navigating.

An antagonism metric needs to be defined so that if two inputs differ too much, then the system can recognize that it needs to stop and re-evaluate what it thinks it should be doing. There are many possible ways to define such a metric, but one simple way is using the vector dot product to judge how far apart the vectors are in both heading and magnitude. Eqn. 2.19 calculates the dot product of the two input vectors and normalizes the results so if it approaches zero, below a
CHAPTER 2. SHARED CONTROL

certain threshold, the system decides the two inputs are diverging from the same goal. The min/max ensures that the metric approaches zero no matter which vector is larger.

\[
\beta = \frac{\dot{p}_u \cdot \dot{p}_a}{\max(\dot{p}_u \cdot \dot{p}_u, \dot{p}_a \cdot \dot{p}_a)} = \min \left( \frac{\dot{p}_u}{|\dot{p}_u|}, \frac{\dot{p}_a}{|\dot{p}_a|} \right) \cos(\gamma)
\]  

(2.19)

Of course, weightings or other more complicated metrics could be implemented depending on what characteristics are important, but these would be defined on a task-centric basis.

2.4 Experimental Setup

Figure 2.5: The environment where test subjects were asked to control a Turtlebot mobile robot. The users had to drive the robot from the starting position to a finish position marked on the ground in normal, time-delay, and drift conditions.

In order to validate the blended shared control implementation, we conducted a study with 9 college aged volunteers. Utilizing the Robot Operating System (ROS) and a Turtlebot, the necessary mapping, navigation, and control modules to test the algorithm were quickly prototyped. Figure 2.5 shows the laboratory environment where we conducted the experiments. The Turtlebot was used to map and localize in environment with the gmapping package, and the standard navigation package generates plans to goals within the map, allowing the Turtlebot to navigate in a fully autonomous manner. The Turtlebot does not have a manipulator like the formulation above, but it is not needed to demonstrate the blended shared control approach because both the robot and user control the robot motion at any given time.

The Turtlebot attempts to drive towards a goal location marked in the environment while the user provides input through the keyboard. The user input and Turtlebot control input are merged according to the equations shown above. Users are asked to drive the robot through the environment
CHAPTER 2. SHARED CONTROL

to the marked final goal position while looking at the physical robot. Three different scenarios are tested: a normal situation with no added latency or system disturbances, a time-delay scenario where the user input is buffered and delayed a set user amount, and finally a drift scenario where a constant drift in one direction is applied, similar to the current in a river. While the user is driving the robot around, we record the time and distance it takes the user to complete the task.

Drift and time delay scenarios were selected as the basis for this test for a few different reasons. The prior work in blended shared control for semiautonomous wheelchairs [41, 42] and mobile robots [27] do not consider the effects of external disturbance or time delay, common challenges we have encountered in the [HILCPS] we have implemented. The drift scenarios is applicable to rovers in space exploration because the robots regularly operate on shifting gravel or regolith that cause drift from the path that the robot attempts to follow. Sampling on steep inclines, described in the next chapter in greater detail, is difficult with pure teleoperation due to the shifting of the robot. In humanoid systems, unmodeled disturbances such as wind in outdoor environments have hindered the completion of tasks such as the door task in the [DRC]. The time delay challenges are always present, and have been proven to cause difficulty in controlling semiautonomous systems in literature from prior work. Characterizing the effects of time delay on blended shared control is an important contribution to enable better comparison of blended shared control approaches with respect to real-world performance.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Performance Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Delay: 1 sec</td>
<td>Robot Odometry (meters)</td>
</tr>
<tr>
<td>Drift: 0.2 rad/sec</td>
<td>Time to Completion (seconds)</td>
</tr>
</tbody>
</table>

Table 2.1 shows the parameters used for the blended shared control experiment in the time delay and drift scenarios. The performance metrics and their units are also called out. The robot odometry is utilized since it is readily available and an accurate measurement of the distance the robot traversed to complete the task. In addition, the time to complete the task is calculated from when the user first commands the robot to move. The navigation task is considered complete when the robot is within a small region around the goal point because aligning the robot perfectly to the goal pose is not possible with sensor and input noise.

Figure 2.6 shows the view from the Turtlebot Kinect sensor. The robot computes an autonomous path, shown in green, that it attempts to follow but allows the user to deviate from the path. As the robot gets closer to the goal location, the $\alpha$ parameter transfers more control to the
CHAPTER 2. SHARED CONTROL

Figure 2.6: The view of the experiment from the robot perspective. The autonomous path computed by the navigation package is shown in green on the map and in the field of view of the robot camera.

robot offloading the difficult task of precise alignment to the robot from the user.

2.5 Results and Findings

The time and distance to complete the navigation task was recorded for the 9 individuals in the three different scenarios. Each participant was asked to drive the same robot to the same goal location from a starting point next to them. The test participants could see the robot and goal location from the computer used to control the robot.

Figure 2.7 summarizes the times it took users to complete the navigation task. The box-plots show that the average time for task completion increased significantly when time delays are introduced. Intuitively, this behavior is expected, but it is interesting to note that the variance in the times also increases significantly, not just the average, indicating that the abilities to control a system with time delay have larger disparities than a regular teleoperation task, even with the blended shared control. The drift scenario shows surprising results though. The time to complete the task on average was faster than the normal scenario. We speculate that this is due to the users leveraging the drift to drive the robot towards the goal and trusting the blended shared control to bring the robot to the goal faster than in the normal scenario.

We run a Wilcoxon signed rank test to determine if the increase in time to complete
CHAPTER 2. SHARED CONTROL

Figure 2.7: A boxplot showing the time to complete the navigation task using the robot in all three cases: normal, time-delay, and utilizing drift.

the task is statistically significant. The test is similar to the paired Student’s t-test but makes no assumptions about the underlying distributions. The test was formulated as follows:

**Null Hypothesis 1.** *The times to complete the navigation task under the normal scenario and the time delay scenario come from the same distribution.*

Utilizing the `signrank()` command in MATLAB, the null hypothesis is rejected at the 5% significance level with a p-value of 0.0273. This result indicates that the time delay scenario data does not come from the same distribution as the normal scenario, meaning the result is in fact statistically significant. Users do in fact take longer to complete the task when the time delay is introduced.

Figure 2.8 shows the boxplots of the total distance the robot traveled to complete the task for all users and experiments. The results are remarkable because they show that the time delays did not result in significantly more distance covered than the normal scenario, but when drift is applied, the robot travels significantly farther with more variance. This is the inverse of the results
CHAPTER 2. SHARED CONTROL

Figure 2.8: A boxplot showing the distance to complete the navigation task using the robot in all three cases: normal, time-delay, and utilizing drift.

when tracking time. These results seem to indicate that users do not travel farther when time delays are introduced, but the drift scenario pushes them off-course and take a significantly longer route. Despite the longer distance covered, they do not take more time in the drift scenario, but they do take a longer time but about the same distance in the time delay scenario.

As before, we run a Wilcoxon signed rank test to determine if the increase in distance to complete the task is statistically significant. The test was formulated as follows:

**Null Hypothesis 2.** *The distance traveled to complete the navigation task under the normal scenario and the drift scenario come from the same distribution.*

The null hypothesis is rejected at the 5% significance level with a p-value of 0.0117. This result indicates that the drift scenario data does not come from the same distribution as the normal scenario, meaning the result is in fact statistically significant. Users do in fact drive the robot a farther distance when the drift is introduced.
CHAPTER 2. SHARED CONTROL

Figure 2.9: A boxplot showing the percentage degradation in time for each test subject compared to normal operation to complete the navigation task.

Figure 2.9 shows the degradation of each individual user’s time in completing the navigation task in both the time-delay and drift scenarios when normalized with respect to their unaffected task. The results are similar to the previous figure, but show that in almost every case, the individual performances degrade significantly with time-delay and stay about the same with drift on a per user basis.

Figure 2.10 is similar, showing the degradation of the task performance with respect to the distance traveled on a per user basis. We see similar results showing that users tended to travel significantly farther with drift, and only slightly farther with time delay.

Figure 2.11 shows how the control authority of the whole systems shifts dynamically between the user and robot from the beginning to the end of the navigation task for a single user. At the beginning, $\alpha$ is near 0.9 indicating the system is biased towards the user input with slight guiding from the robot. As the target location is approached, control authority starts shifting to the robot in order to help guide the user into the target location. The parameter asymptotically approaches 0.5 where the user and robot equally share control authority over the system.
CHAPTER 2. \textit{SHARED CONTROL}

Figure 2.10: A boxplot showing the percentage degradation in distance for each test subject compared to normal operation to complete the navigation task with time delay and drift.

Overall, the results are consistent with intuitive expectations. The main contribution of this work is quantifying the degradation of performance with time delays and drift disturbances in combination with a blended shared control approach used. The blended shared control results in performance where users may travel farther due to drift, but take about the same time to complete the task as an unaffected run or take longer to complete the task due to time delay, but travel about the same distance as an unaffected run. This correlation was not expected, and could be the basis for future developments based on the presented results.

2.6 \textbf{Doorway Traversal}

In [41], Goil et al. present a machine learning approach to implement navigation through a doorway utilizing blended shared control on a semi-autonomous wheelchair. They utilize the same linear blending functions as in our implementation, but the alpha parameter selection is carried out as a function of the covariance of the angular input of a learned user model and the autonomous
Figure 2.11: The $\alpha$ parameter controls the level of control authority between user and robot. This figure shows how the parameter changes throughout the course of an experiment for one user. Initially, the user is mainly in control of the robot with a little guidance provided by the autonomous navigation. As the goal is approached, the control authority shifts toward a more equal share, helping guide the robot to the target.

Angular velocity input. They implement the system in simulation and provide conclusions on its applicability and performance. While a direct comparison is difficult to complete due to the differences in the blending parameter implementation, we demonstrate our blended shared control on a similar doorway traversal problem.

Figure 2.12 shows the setup for the doorway traversal experiment with a Turtlebot. Users are asked to navigate the robot through an open doorway with the blended shared controller running. Disturbances consisting of time delay and drift are added and then compared to runs without any disturbances. The time delay and drift parameters are the same from Table 2.1. This experiment is more difficult for the operators than the previous one because they encounter more obstacles and tighter tolerances to complete the navigation task through the doorway. A total of 10 college aged volunteers provided the data for the doorway traversal task.
CHAPTER 2. SHARED CONTROL

Figure 2.12: The setup for the blended shared control experiment for doorway traversal. Users are asked to navigate a Turtlebot through an open doorway while the blended shared controller is fusing their input with the autonomous navigation trajectory. The performance of the human-robot team is compared for drift and time-delay scenarios to the normal scenario.

The robot starts away from the door and needs to travel a short distance before turning left. Once the turn is complete, the robot needs to travel around the blue bucket shown in Figure 2.12 and then through the door. The run ends when the robot reaches the location marked by a bin directly past the door. The time to complete the navigation task and distance traveled are recorded for each run, and used as metrics for comparing the performance of each run.

Figure 2.13 shows the time it took to complete the doorway traversal task for the 10 runs in all three scenarios as boxplots. The results are similar, as expected, to Figure 2.7 and indicate that users took longer to complete the task with time-delay and only marginally longer with drift. It is interesting to note that qualitative testing with the time-delay scenario and no blended shared controller led to a robot that was almost impossible to accurately control through teleoperation. The blended shared controller enables the users to traverse the doorway, albeit at a slower pace, but nevertheless successfully.

As before, we run a Wilcoxon signed rank test to determine if the increase in time to complete the task is statistically significant. The test was formulated as follows:

**Null Hypothesis 3.** The times to complete the doorway traversal task under the normal scenario and the time delay scenario come from the same distribution.
CHAPTER 2. SHARED CONTROL

Figure 2.13: A boxplot showing the times in seconds to complete the doorway traversal task for the normal, time delay, and drift scenarios. The results are similar to the simpler navigation task.

The null hypothesis is rejected at the 5% significance level with a p-value of 0.0137. This result indicates that the time delay scenario data does not come from the same distribution as the normal scenario, meaning the result is in fact statistically significant. Users again took longer to complete the doorway traversal task with time delay disturbances than in the normal scenario.

Figure 2.14 shows the distance users traversed to get past the doorway for the three scenarios as boxplots. Qualitatively the results follow the same pattern as before, but the scale of the differences is much smaller in this case. We speculate that the reason for this is two-fold. This experiment involved longer distance travel with more constraints to the possible paths. Both the bucket and doorway limited the paths the operators could take, and thus may have limited the variability in the distances. Despite this, the time-delay and drift scenarios do take slightly longer paths as would be expected when compared to the undisturbed case.

We run a final Wilcoxon signed rank test to determine if the increase in distance to complete the task is statistically significant. The test was formulated as follows:

Figure 2.13: A boxplot showing the times in seconds to complete the doorway traversal task for the normal, time delay, and drift scenarios. The results are similar to the simpler navigation task.
CHAPTER 2. SHARED CONTROL

Figure 2.14: A boxplot showing the times in seconds to complete the doorway traversal task for the normal, time delay, and drift scenarios. The results are similar to the simpler navigation task.

**Null Hypothesis 4.** *The distance traveled to complete the doorway traversal task under the normal scenario and the drift scenario come from the same distribution.*

The null hypothesis is rejected at the 5% significance level with a p-value of 0.0488. This result indicates that the drift scenario data does not come from the same distribution as the normal scenario. Users drive the robot a farther distance when the drift is introduced, but not by much in this specific experiment.

2.7 Conclusion

The presented blended shared control formulation and two experiments are intended to show, at a low implementation level, how human input can be merged with autonomous behaviors as part of the shared control framework. In the shared control framework, this implementation would be split between the action engine, generating the autonomous trajectory, the operator interface,
CHAPTER 2. SHARED CONTROL

taking the human input, and the achievable action gate performing the blended shared control fusion. The output of the gate would be checked for any robot constraints such as torque/velocity limits and holonomic constraints, and passed to the low-level controllers to implement the actions. The main contribution is the demonstration of the blended shared control formulation from beginning to end on a real robot, and the presentation of results characterizing the behavior of the approach.
Chapter 3

Rovers

Over the last 5 years, the RIVeR Lab has designed, built, and implemented several rover platforms within the frameworks of the NASA RASC-AL Robo Ops and Sample Return Challenges. There are several benefits to conducted research in parallel with competition development. The biggest benefit comes from the fact that the competitions provide set deadlines and benchmarks that need to be met. This helps bound the scope of the design, force steady progress, and significantly reduces risk at a systems level. Secondly, the competition structure helps keep research problems well-grounded with respect to real-world problems. Robotics in the future will be looked upon as a transformational technology, and the success will be grounded in real-world problems, not abstract formulations devised in meeting rooms. Competitions force the research team to solve real-world problems in a structured and systematic way. Finally, the last benefit is that many of these robotic competitions in the last few decades have formed strong research communities around specific problems. We have been lucky to be a part of a few of these communities, and that has helped push the research forward.

In this chapter, we describe three different rover systems that have been developed by RIVeR. Oryx 1.0 and 2.0 are mainly teleoperated platforms, with emphasis on speed and traversal respectively. AERO is a mainly autonomous platform with emphasis on sensing, computation, and manipulation capabilities. Each robot with its distinct advantages required different approaches to solve the task at hand. We describe the operation, challenges faced, and novelty that each of these systems has brought. The chapter will focus mainly on Oryx 2.0 and AERO since those two platforms have the most development and progress.

1. Oryx 1.0, is a research mobility platform originally designed for the inaugural NASA RASC-
CHAPTER 3. ROVERS

1. AERO Robo Ops competition. It consists of a rigid frame with four independent drive motors directly coupled to the chassis, kinematically similar to AERO. Due to the specific motor controller, motor, and gearbox combination, ORYX 1.0 is a very fast robot capable of covering significant area quickly. It carries a light payload of two cameras, wheel encoders, and a 4G/WiFi receivers simulating high gain/low gain antenna modules.

2. ORYX 2.0, shown in Figure [3.1] is a research mobility platform designed originally for the NASA RASC-AL Robo Ops competition [4]. Aimed at operating in analog Martian and lunar environments, this rover has passive kinematic suspension with skid steering and is designed to operate on uneven terrain. Even though the passive averaging rocker-bogie suspension with skid steering is not beneficial in all cases [95], it provides ORYX 2.0 with the flexibility to handle a variety of terrain while maintaining simplicity of the mechanical design. ORYX 2.0 has a footprint of about 1m by 1m with a weight of about 45kg, and with 31 cm diameter wheels is capable of traversing obstacles up to 20 cm high. ORYX 2.0 is also equipped with a number of sensors including a InterSense InertiaCube3 IMU, 12-bit absolute encoder measuring rocker angle, and drive motor controllers providing feedback on the wheels’ velocity, position, and current draw.

Figure 3.1: AERO (left) and ORYX 2.0 (right) at Bond Construction during field testing.

3. AERO, the Autonomous Exploration Rover, shown in Figure [3.1] is a research platform designed originally for the 2013 and 2014 NASA [SRR] Centennial Challenges. The rover is comprised of a differential-drive four-wheeled mobility platform and 6-DOF manipulator with a fixed suspension. The simple mechanical design increases reliability significantly at
the cost of ground compliance. AERO has a footprint of about 99cm by 67cm with a mast height of just under 1.5m and weighs about 80kg. While AERO is not as fast as ORYX 1.0 or able to travel over very rough terrain like ORYX 2.0, it carries more precise sensors and possesses significantly more computation power. AERO is instrumented with four Allied Vision Manta G-095C cameras in a two stereo pair configuration, a KVH 1750 fiber optic gyroscope based IMU, a 50m SICK LMS151 LIDAR, and wheel encoders.

Both ORYX 1.0 and ORYX 2.0 were designed with the NASA RASC-AL Robo-Ops competition in mind. The competition is an engineering challenge organized by the National Institute of Aerospace and funded by NASA designed to encourage teams to create teleoperated rover prototypes that can find and pick-up brightly colored rock samples. The challenge is held at the Johnson Space Center rock yard which has analog test areas simulating various terrains similar to ones on the Moon and Mars. The rovers need to navigate a hilly terrain, sand pit, crater field with loose gravel, and rock field with large obstacles collecting as many samples as possible within the 1 hour time limit. The competition rules allow teams to control their rovers from their host institution over a 4G cellular radio resulting in limited bandwidth and slight time delays. Time delays up to 1 second and bandwidths as low as 250 kilobits/s are possible, especially if the cellular radio fails-over to 3G, making teleoperation more difficult than in other applications.

### 3.1 ORYX 2.0

One of the tasks that operations with ORYX 1.0 showed to be difficult at the NASA Desert RATS field testing was to drive in straight trajectories over rough terrain. ORYX 2.0 was designed from the ground up to handle rough terrain with it’s passive averaging suspension. This provides an extra degree of freedom in order to have ground compliance, but makes driving in a straight line more difficult. In order to make teleoperation of the robot easier over rough terrain, we developed a feedforward velocity controller to enhance the precision of straight line trajectories for a rover with passive averaging suspension on rough terrain.

Figure 3.3 shows the work on ORYX 2.0 making contributions to the robot control blocks toward enabling shared control. At the time, ORYX 2.0 had difficulty driving in straight trajectories over rough terrain. Eventually, the solution involved blending the human teleoperation input and the robot’s proprioceptive sensors to compensate for the disturbances caused by the uneven ter-
CHAPTER 3. ROVERS

Figure 3.2: Oryx 2.0 collecting a sample rock from the moon-analog crater field at Johnson Space Center Analog Test Facility during the 2012 NASA RASC-AL Robo-Ops competition. Oryx 2.0 collected a total of 13 rocks and one "alien" life-form taking first place in the competition.

While the results are not monumental, they represent an important incremental step towards effective shared control on mobile robots.

Planetary rovers such as Carnegie Mellon University’s (CMU) Scarab [6, 105], NASA’s Jet Propulsion Laboratory (JPL) Sample Return Rover (SRR) [48], NASA’s Rocky 7 Rover [100], and CMU’s Nomad [104] have demonstrated the effective uses of this suspension technique in terms of mobility and endurance. While such passive suspensions improve the mobility and terrain traversability for rovers, they pose complexity in controls with the added degree of freedom in the suspension system. Modeling the 3D kinematics of the suspension has been used in prior work to achieve more precise trajectory control [49, 84].

Zoe, another rover designed by CMU, is an example of a rover with passive suspension and passive steering that employs a 3D kinematic model as part of a controller to improve trajectories over uneven terrain [84, 102]. The controller estimates terrain slopes at each wheel using the kinematic model assuming that only positive obstacles exist in the environment. Wheel velocities are then compensated to take into account the sloping terrain. The aim is to reduce wheel slip and increase accuracy of executed trajectories. Simulation and field testing results show improved performance in trajectory following, compared to controllers that did not compensate based on the kinematics of the robot.
CHAPTER 3. ROVERS

Figure 3.3: The development of the ORYX 2.0 rover marked the beginnings of our shared control implementation. The work resulting in a low-level controller that assisted the operator to drive in straighter trajectories over rough terrain. The implementation contributed mainly to the robot control block, but is relevant due to the foundation it laid for AERO and the later developments.

The controller developed for ORYX 2.0 is based on the motion controller for Zoe, specifically utilizing the feedforward velocity scaling and attempts to improve trajectories using a purely 3D kinematic approach based on proprioceptive sensors. One limitation of Zoe’s control approach and similar methods reported in literature is a reliance on global position feedback or assumptions on the type of obstacles encountered in the environment [84]. These methods are typically implemented through inertial measurement sensors or visual odometry. The issue is that these methods are hindered by drift and slow update rates [58]. Alternatively, approaches exist that rely on dynamic models but they are computationally intensive and involve modeling complex interactions between the ground and wheels [13]. To address these limitations the developed controller relies solely on proprioceptive feedback, the inertial orientation data and feedback on the position of the averaging suspension (rocker angle).

With sensor data indicating the orientation of the rover chassis and position of the averaging suspension, a kinematic model is utilized to find the wheel positions relative to the rover’s
Figure 3.4: ORYX 2.0 shown driving over an obstacle. The passive averaging suspension serves two related purposes. It allows all four wheels to be on the ground while traversing obstacles for maximum torque and halves the roll angle experienced by the chassis of the robot.

The sensor data is used to find pseudo-global wheel heights, providing an estimation of the terrain profile. The motivation for the controller is to minimize deviations in yaw and wheel slip, which result when using constant wheel speeds over uneven terrain. While this approach can effectively be used with similar results to more complex methods, it has several limitations. It assumes no wheel slip and cannot verify forward progress through feedback. In addition, the controller relies on the assumption that an ideal ground plane exists, and that at least one wheel is on this ground plane. The controller is more effective when more wheels are on the ground plane because the ground plane estimate is more accurate.

3.2 Overview of ORYX 2.0

ORYX 2.0 is a research mobility platform designed at Worcester Polytechnic Institute [4]. Aimed at operating in analog Martian and lunar environments, the rover has passive kinematic suspension with skid steering and is designed to operate on uneven terrain. Even though the passive averaging rocker-bogie suspension with skid steering is not beneficial in all cases [95], it provides ORYX 2.0 with the flexibility to handle a variety of terrain while maintaining simplicity of the mechanical design. ORYX 2.0 has a footprint of about 1 m by 1 m and with 31 cm diameter wheels is capable of traversing obstacles up to 20 cm high. Preliminary field testing with ORYX 2.0 showed large deviations in yaw and wheel slipping when traversing uneven terrain, which inspired the development of this controller.
CHAPTER 3. ROVERS

ORYX 2.0 is also equipped with a number of sensors. To measure the yaw, pitch, and roll of the chassis an InterSense InertiaCube3 is integrated under the deck of the robot. Feedback from the position of the suspension rocker is provided by a 12-bit absolute encoder. Finally, drive motors are controlled by Maxon’s EPOS2 controllers, which provide feedback on wheel velocity, position, and current draw.

Unlike many skid steering rovers, ORYX 2.0 has each wheel driven by a separate motor, allowing the velocity of all four wheels to be independently controlled. This feature enables the design of a velocity controller that can adjust wheel velocities according to an estimates of terrain the rover is driving over.

3.2.1 Passive Averaging Suspension

Passive averaging suspensions are commonly used to achieve ground compliance, ensuring that four wheels remain in contact with the ground, creating higher stability, and better weight distribution. The averaging effect of the suspension is an additional benefit as the roll of the chassis experiences is halved when compared to a non-compliant suspension.

Figure 3.5: A rendering of the robot showing the chassis and passive averaging suspension implemented through a 6-bar linkage composed of rocker linkages and a differencing arm. The passive averaging suspension keeps all four wheels on the ground irrespective of the terrain the robot is driving on and halves the roll the chassis experiences.

Multiple methods of implementing the differencing suspension have been demonstrated in prior work including using geared differentials and differential linkages. ORYX 2.0 has a differential linkage system, depicted in Figure 3.5. This approach places the suspension components outside the
CHAPTER 3. ROVERS

chassis and in a small form factor relative to the chassis size. Figure 3.5 shows the chassis design and the rocker differencing suspension.

To meet the mobility requirements for obstacle traversal, 35 cm diameter wheels which result in a ground clearance of 16 cm were chosen. Figure 3.4 depicts the rover design showing the low-profile chassis and relative wheel size. Since traversing obstacles half the diameter of the wheels is trivial, this configuration meets the design requirement of traversing 15 cm obstacles. However, given sufficient traction and the design features of the passive suspension, the rover is capable of easily exceeding this limit.

3.3 Kinematic Model

Rovers with passive suspension can be implemented in several ways; however, they all generally have similar kinematics where one degree of freedom defines the locations of the wheels relative to the chassis. In the case of ORYX 2.0, two rocker arms on either side of the rover passively rotate in opposite directions, constrained by a 6-bar mechanical linkage. Figure 3.6 shows the kinematic model of ORYX 2.0.

Three kinematic parameters are needed to fully characterize the rover. The first parameter is the wheel base ($\rho$), defined as the distance between the left and right sides of the rover, illustrated by the red line connecting one rocker assembly to the other in Figure 3.6. The second parameter is the length of the rocker arm ($\ell$), defined as the distance between the rocker pivot and center of the wheel; illustrated by the distance between coordinate frames one and two. The third parameter is the acute angle between the wheels on one rocker arm ($\phi$), defined as the angle between the rocker arm and the horizontal axis. For ORYX 2.0 these parameters are as follows:

$$\rho = 0.861 \text{ m}, \quad \ell = 0.330 \text{ m}, \quad \phi = 8.5^\circ$$

The kinematic model has four inputs: rocker arm angle ($\theta$) and the yaw ($\alpha$), pitch ($\beta$) and roll ($\gamma$) of the chassis. For the purposes of this implementation yaw is assumed to be zero, and thus ignored in the model. The rocker arm angle defines the position of the passive degree of freedom, defining the position of the two suspension members. The yaw, pitch, and roll, are rotations around the z-axis, y-axis, and x-axis respectively, of coordinate frame zero. Utilizing this information and applying trigonometric principles, it is possible to calculate each wheel’s pose relative to coordinate frame zero. This forms the basis of the kinematic model.
CHAPTER 3. ROVERS

The equations above summarize the kinematic model to determine the wheel heights. With the three model parameters and three inputs from the orientation sensor and suspension encoder, the kinematic model solves for the height of each wheel relative to coordinate frame zero in

\begin{align}
H_{FR} &= -\frac{\rho \sin \gamma}{2} + \ell \cos \gamma \sin(\theta - \beta - \phi) \\
H_{BR} &= -\frac{\rho \sin \gamma}{2} - \ell \cos \gamma \sin(\theta - \beta + \phi) \\
H_{FL} &= \frac{\rho \sin \gamma}{2} - \ell \cos \gamma \sin(\theta + \beta + \phi) \\
H_{BL} &= \frac{\rho \sin \gamma}{2} - \ell \cos \gamma \sin(\theta + \beta - \phi)
\end{align}

Figure 3.6: The kinematic model for the rover showing the frames used to track the chassis position, rocker angle, and wheel positions relative to the ideal ground plane.
CHAPTER 3. ROVERS

the center of the chassis.

3.4 Feedforward Velocity Control

The feedforward velocity controller is designed to estimate terrain profile under each wheel at discrete time steps, and adjust the velocities to attempt to maintain a constant heading. Terrain slopes are estimated based on the kinematic model and an ideal ground plane generation algorithm. The flow diagram in Figure 3.7 shows the process of the feedforward controller from the receiving of the sensor data, passing it through the kinematic model, and finally sending the compensated wheel velocities to the motor controllers. The proposed controller relies on the following core assumptions:

- An ideal ground plane exists
- At least one wheel is on or near the ideal ground plane
- The rover’s forward velocity is constant and equal to the desired forward velocity

3.4.1 Terrain Profile Estimate

The kinematic model only calculates the positions of each wheel relative to the rover, making it insufficient to provide sufficient information on the underlying terrain profile. In order to successfully adjust wheel velocities, an accurate estimate of terrain slopes is required. To determine an estimate of terrain slopes, pseudo-global wheel heights are calculated using a combination of the

![Figure 3.7: A diagram showing the process for taking the proprioceptive sensor information, passing it through the kinematic model, and calculating the compensated wheel velocities.](image)
CHAPTER 3. ROVERS

kinematic model and a ground plane estimator algorithm. These wheel heights are pseudo-global because they are with respect to the ideal ground plane, which inherently is relative to the rover and in most cases tracks the actual ground plane well enough to implement the compensated wheel velocities.

Algorithm 1: Estimating the ideal ground plane and adjusting wheel heights to form pseudo-global values

1. Determine largest magnitude wheel height based on kinematic model
2. If this is a positive value
   (a) Determine the smallest wheel height
   (b) Subtract this value from all wheel heights
   (c) Repeat at next time step
3. If this is a negative value
   (a) Determine the largest wheel height
   (b) Subtract this value from all wheel heights
   (c) Repeat at next time step

The algorithm takes in four wheel heights which are provided by the 3D kinematic model. It then converts these local values into pseudo-global wheel heights as described, and repeats at each time step when the kinematic model updates due to new sensor data. While simplistic in nature, this algorithm proves to be effective in a variety of simulated scenarios, is computationally efficient, and can handle both positive and negative obstacles.

3.4.2 Compensating Wheel Velocities

Pseudo-global terrain heights are calculated at each time step but an estimate of the forward progress of the robot is also needed to calculate terrain slopes. There are various ways to calculate this forward progress such as with odometry, visual odometry, or inertial measurement units. Despite the availability of odometry data, a simplifying assumption of a constant forward ve-
locity is implemented. This simplification does significantly reduce the performance of the system but avoids problems related to wheel slippage and a need for complex dynamic models.

Figure 3.8: A figure showing the compensation of the wheel velocities based on the terrain slope underneath the wheel. The controller aims to keep the desired heading velocity constant over the slope.

Using the assumption of constant forward velocity, the controller takes in the desired forward velocity and adjusts each wheel’s velocity based on the desired forward velocity and the estimate of the terrain slopes. Figure 3.8 demonstrates the calculation of the compensated velocity graphically. The change in height is the difference between two pseudo-global heights calculated across one time step. Knowing the time between the measurements, the difference is mapped to an estimate of wheel vertical velocity. The resulting compensated velocity \( v_{\text{comp}} \) is simply the magnitude of the estimated vertical velocity and assumed forward velocity, show in Equation 3.5.

\[
v_{\text{comp}} = \sqrt{\left(\frac{\Delta h}{t_1 - t_2}\right)^2 + v_d^2}
\]  

(3.5)

The process is repeated for each of the four wheels, and at each time step the controller calculates new velocities and sent to the motor controller for each wheel. An update frequency of 20Hz or time step of 50 milliseconds is used for the implementation of the controller.

3.5 Experimental Setup and Results

Figure 3.9 shows the experimental setup where ORYZ 2.0 is driven on flat ground over an obstacle while logging data with and without the controller enabled. The obstacle is a trapezoid
CHAPTER 3. ROVERS

with dimensions shown in Figure 3.10. While traversing the trapezoid the roll, pitch, and rocker angle data are logged so that they can be used as the inputs for the MATLAB simulation in [16] and the feedforward controller.

![Rover Image]

Figure 3.9: A picture showing the physical experiment setup with ORYX 2.0 and a trapezoidal obstacle. The rover is driven over the obstacle while the yaw deviations are measured using the onboard IMU.

The performance of the controller was simulated in MATLAB in [16]. In order to verify its operation, identical experiments were repeated with and without the controller enabled on the real robot traversing the obstacle. The controller was implemented on ORYX 2.0 utilizing ROS (Robot Operating System) and ran at 20 Hz. The success of the controller was analyzing using the changes in yaw as measured by the onboard IMU. These yaw changes correspond to the deviations from the straight line trajectory due to the obstacle. In an ideal scenario, the controller would keep the yaw at 0 degrees.

Figure 3.11 shows the yaw deviations from two trials, one with the controller enabled and one without the controller. The experiment was repeated multiple times and similar results were found in each case, with final yaw errors between 7 degrees and 9 degrees for no controller, and between 1 degree and 3 degrees when the controller is activated. Overall, this large reduction in yaw achieved shows that the implemented controller performs well. The controller fails to compensate fully for the slope in the cases where the front wheel and rear wheel are on the slope at separate times. Occurring at the beginning and end of the traversal, as a result the yaw is observed increasing.
Figure 3.10: A diagram showing the dimensions (in meters) of the trapezoidal obstacle used to conduct the experiment.

at these times. Similarly, the over compensation during the transition period results in a change in yaw in the negative direction. This error improves overall performance by balancing out the other two main sources of yaw error and proved repeatable over multiple trials.

3.6 Conclusions

The presented controller proved successful in improving accuracy of straight line trajectories over rough terrain, in a purely kinematic approach, and marked the first, albeit simplistic implementation of a shared control algorithm for our Earth analog space exploration rover. The controller is implemented on ORYX 2.0 and proved successful in experiments. While such experiments only investigated positive obstacles the kinematics and algorithm used behave identically for negative obstacles.

One limitation and source of error encountered during experiments with the rover was the apparent time delay between the location of the rover on the trapezoid and the compensated velocities. Another issue is that the velocity is calculated between two adjacent time steps. When new sensor data is updated, the desired wheel velocity can be calculated between that current time
CHAPTER 3. ROVERS

Figure 3.11: Yaw deviations from two separate trials where the controller is enabled and disabled. The controller provides an obvious improvement helping decrease the robot yaw deviations.

and the last time step. The limitation is that the desired velocities will always be one time step behind or require some means of estimating the next state of the rover to compensate for this effect. Despite these limitations, the yaw controller represents an important contribution that seeded and spurred the development of the shared control implementations to follow and the shared control framework as a whole.

3.7 AERO 2013

AERO, shown in Figure 3.13, was built after the success of using ORYX 2.0 in the Robo-Ops Challenge. The timing worked well because the NASA Sample Return Robot (SRR) Centennial Challenge was announced almost immediately after finishing development on ORYX 2.0. The logical next step after Oryx 2.0 was to design and implement a fully autonomous rover. AERO is comprised of a differential-drive four-wheeled mobility platform and 6-DOF manipulator designed to participate in the Sample Return Challenge. Fusing a combination of data from a fixed, forward-facing stereo vision system, LIDAR, and IMU, AERO implements a simultaneous localization and mapping (SLAM) algorithm to mark what areas are searched and return to the starting platform at the end of the competition. A second panning stereo vision system on a mast is used to locate and
identify samples using object classifier and texture-based algorithms.

Figure 3.12: AERO, while originally designed to be fully autonomous, contributes to many of the blocks in the shared control framework enabling the implementation of robust autonomy specifically in the areas of perception, navigation, and manipulation. Limited contributions to the knowledge base are also present given the classifiers used to find unknown objects.

Figure 3.12 shows AERO’s contributions to the framework’s knowledge base, action engine, perception engine, achievable action gate, and robot control blocks. The SRR Challenge is a fully autonomous robot challenge hence AERO does not make significant contributions to the human-centric bottom half of the diagram. Despite this, the contributions to the top half of the framework are relevant to the development of robust autonomy, many of which were also utilized in the DARPA Robotics Challenge with the ATLAS robot.

The knowledge base is highlighted in AERO’s implementation of the shared control framework due to our work on training the vision classifiers to detect the unknown metallic samples for the challenge. The classifiers do not represent the fullest extent of what is possible to be developed in the knowledge base. Instead, they are one example of the type of information that may be stored in the knowledge base and shared between robotic systems or called up when unknown objects need to be located and manipulated.
CHAPTER 3. ROVERS

The action engine, perception engine, achievable action gate, and robot control blocks were improved throughout the two year development of AERO. Significant development resulted in a highly customized version of a commercially available platform. Sensing and information extraction algorithms formed the basis of the perception engine, finding obstacles in the path of the robot and samples that needed to be picked up. An accurate state estimator based on an extended Kalman filter was also implemented as part of the perception engine. Cognition and path planning algorithms composed the action engine. The computation model for the cognitive and path planning functions of the robot was succinctly encapsulated in a finite state machine. Finally, path execution and acting models were included in the robot control blocks to implement the functions required by the action engine as arbitrated by the action gate.

The SRR Challenge is an annual NASA Centennial Challenge hosted by Worcester Polytechnic Institute for the first time in June 2012. The premise of the competition is to encourage teams composed of engineers, students, and tinkerers to build fully-autonomous robots that can navigate a large outdoor area, find and collect various geologic samples, and return them to the starting pad within the time limit. The caveat is that only space-compatible technologies are allowed, meaning that global positioning systems, sonar, compasses, magnetometers, and similar technologies are not allowed. These limitation create significant challenges to accurate localization and navigation, requiring solutions similar to what many military systems use in GPS-denied or GPS-corrupted areas. Samples for collection are split into three categories, easy, intermediate, and difficult, based on how much information and detail is provided on each sample apriori. For example, the easiest samples are fully described, and the difficult ones are just noted to be metallic objects of interest that visually do not appear to belong in the environment. For level one only a single easy sample is on the field, but for level two a combination of many easy, intermediate, and difficult samples are on the field.

This section presents a navigation system and vision system that allow the rover to avoid obstacles, navigate towards the samples, recognize the samples, and retrieve them. At a conceptual level, the system is designed with hierarchical control layers. The supervisor, global planner, and local planner layers handle the state of the robot, current path to the target, and avoiding obstacles respectively.

In order to locate and identify samples, AERO utilizes stereo vision object recognition, localization, and grasping algorithms to control the manipulator and retrieve the samples. Our algorithm first extracts the location information of the object of interest using disparity maps for location in 3D space. An object recognition algorithm determines the type of object and its orientation to plan a proper approach vector and grasping strategy.
CHAPTER 3. ROVERS

Figure 3.13: AERO along with its VR tag homing beacon.

AERO’s mission can be split into three main subtasks: navigating and localizing within the large outdoor area, identifying and classifying samples, and retrieving the samples with a manipulator.

3.8 Platform Design

The system architecture is designed with the main subtasks in mind. For example, AERO leaves the maximal amount of space on top of the robot for sample storage. The sensors are selected to comply with the competition rules, but also provide useful data to complete every subtask. A 6-DOF manipulator was selected to provide the most flexibility in sample handing, especially with the hard, undefined samples.

Inside AERO, a Roboteq MDC2250 dual output 60A motor controller implements closed loop velocity control on the primary drive motors. The control loop is closed using standard quadrature output optical encoders. The primary battery pack is also inside the robot towards the front consisting of sixteen 40Ah CALB LiFePO4 cells in an 8s2p configuration to provide 25.6V, 80Ah nominally. LiFePO4 cells were selected because of their good compromise between energy density,
CHAPTER 3. ROVERS

safety, and charge cycles. Two Manzanita Micro MK3x8 battery management systems ensure the safety of the lithium battery pack. Towards the front on a server motherboard, dual 8-core Intel Xeon processors provide the main computing hardware complemented by a NVIDIA Tesla K20 GPGPU co-processor. The Tesla GPGPU excels at image processing because of its highly parallel nature and significantly increases the vision processing capabilities of AERO.

3.8.1 Navigation Sensors

The main sensors AERO uses to navigate are the LIDAR, IMU, mast-mounted stereo vision system, and wheel encoders. GPS and other satellite based navigation aids are not used because they are not compatible with challenge rules. We selected a LMS151 LIDAR from SICK because of its 50 meter maximum range and excellent outdoor performance. The LIDAR directly feeds the SLAM algorithm by very accurately providing ranging data to trees and man-made features in the environment. A KVH 1750 fiber optic ring gyro IMU provides accelerations and angular velocities to enable AERO to dead-reckon when no good LIDAR features are available. A fiber optic ring gyro was selected because of its excellent stability and very low drift rates, providing accurate dead-reckoning for extended times without absolute positioning information from the LIDAR. The mast-mounted cameras periodically pan and extract trees from the scene to help localize the robot as well. Finally, wheel odometry from the motor encoders is fused with all the available data in an extended Kalman Filter (EKF) to localize the robot better than any one sensor can by itself.

3.8.2 Sample Detection and Classification

Sample detection and classification is entirely implemented by the computer vision system. The top mast cameras identify anomalies in the grass that could potentially be samples and mark them on a probabilistic map on the robot. The robot inspects each potential sample from a close distance using the fixed, front-mounted stereo vision system. The easy and medium samples are identified and classified using a Linear Binary Pattern (LBP) classifier. Because the features of the easy and medium samples are known ahead of time, the robot is preloaded with a training set of data helping it identify these samples. The hard samples are identified by their generally different appearance in the environment. The metallic hard samples are extracted from the grass background using simple normalized RGB color filtering. In addition, the fixed forward facing vision system extracts the location, major axis, and bounding box of each samples in order to assist in planning a suitable approach vector for the manipulator.
CHAPTER 3. ROVERS

3.8.3 Manipulation

A Kinova Jaco 6-DOF manipulator was selected to collect samples. It is a commercially available system that provides AERO with the needed flexibility to pick up samples of different sizes and place them on different locations on its top plate. The manipulator has a three-finger underactuated and compliant gripper with individual control over all fingers. The manipulator utilizes brushless DC motors with Harmonic drives resulting in low power consumption while still providing up to 1kg payload.

3.9 Previous Work

Our navigation algorithms are all based on the principle of driving with tentacles. Dillmann et. al.[61] present a method for extending the driving with tentacles algorithm[101]. They propose a system that allows for the control of an autonomous off-road vehicle that is more robust than the original work due to its ability to incorporate additional sensor data into the selection process. However, the algorithm only adds the ability to add additional obstacles and in a generally binary representation of existing. It did not adapt the underlying principle, and thus does not use the additional information to its fullest extent.

Quinlan et. al.[77] present a the concept of a real-time deformable global path in their work. They present a system where an initial global path is calculated, and as new sensor data is incorporated into the model the path is deformed locally to remain collision free. However, the computations are relatively expensive locally. In addition, the global path is never re-planned. If new sensor data has produced the possibility of a new global path that is more optimal, but beyond the reach of a local deformation, it will not be explored. Finally, collisions are only taken into consideration when deforming the path.

Mataric et. al.[63] present the concept of integrating global-tasks into behavior-based robots. In their method, a rough local path is planned, and behaviors are biased such that the robot will tend to follow the global path. In their work however, the behavior set is limited to a small series of hard-coded actions such as ’turn-left’, ’go backwards’, etc. The system also takes up all of the computational resources on the robot, but this work is fairly dated and may have been strongly affected by technology limitations. Finally, there is no concept of exploring, the focus is solely on following a collision-free path between global points making unsuitable for space exploration.

The Haar algorithm in [59] and histogram of gradients (HOG) are two of algorithms
already implemented in the OpenCV library for object detection. These are well known algorithms, and have been modified in various ways for many tasks. For the generation of disparity maps from stereo cameras, Doppelmann in [35] explains the important parameters such as SAD window size and demonstrates how they should be set. In our work, we use the standard OpenCV implementations of these algorithms leveraging the performance optimizations of OpenCV.

### 3.10 Navigation Overview

The implementation of driving with tentacles involves the generation of a set of arcs that emanate from the front of the robot. The navigation algorithm uses the following terminology.

**Tentacle:** A circular arc in space extending outwards from the front of the robot. Its properties are determined by the speed of the robot and the its position in a speed set.

**Speed Set:** A set of tentacles that correspond to a particular linear velocity.

**Pseudo-Subsumption:** A heuristic method to implement subsumption levels of in a behavior-based system, making them soft constraints instead of hard constraints.

**Object of Interest (OoI):** Objects on the field being explored which have been identified as samples.

#### 3.10.1 Hierarchical Control

The AERO navigation system at its core uses the concept of hierarchical control which can be divided into three levels (Fig. 3.14).

![Supervisor (High Level Tasks)](Supervisor.png)

**Figure 3.14:** The three levels of the hierarchical control. The supervisor coordinates the mission-level tasks. The global planner plans beyond the sensor horizon on the global map. The local planner plans on a local map representing a snapshot of the current environment around the robot.
CHAPTER 3. ROVERS

Table 3.1: Navigation System Failure Modes

<table>
<thead>
<tr>
<th>Failure</th>
<th>Effect Up Chain</th>
<th>Effect Down Chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>NA</td>
<td>Global planner receives no new mission-tasks</td>
</tr>
<tr>
<td>Global Planner</td>
<td>Robot does not complete mission-task</td>
<td>Local planner receives no new goal points</td>
</tr>
<tr>
<td>Local Planner</td>
<td>Robot stops moving</td>
<td>NA</td>
</tr>
</tbody>
</table>

The supervisor is the top layer of control implemented by a finite state machine and is responsible for mission level task planning and coordinating robot state. Examples of mission-level tasks include searching the field for objects of interest, navigating to a specific OoI, and collecting an OoI once it has been identified as a sample. It manages feedback from the OoI classification, sample collector control, as well as the navigation system. For the purposes of this work, we are only concerned with current mission task output of the supervisor.

The global planner is the layer of control below the supervisor (Section 3.12). It is responsible for planning in the global scale and implementing the mission-task specified by the supervisor. The lowest layer of control is the local planner (Section 3.11) which uses pseudo-subsumption. It is responsible for sending velocity commands to motor controllers and for dynamic obstacle avoidance.

3.10.2 Fault Tolerance

The hierarchical control allows the navigation system to have a high degree of fault tolerance. Systems further down in the chain can operate independently of systems higher up the chain. For example, if the supervisor fails, the global planner will simply continue attempting to complete its last mission-task. If the global planner fails, the local planner will attempt to reach the last goal it received following its standard behaviors, and then stop. Likewise, failures at the lower levels of the control hierarchy always leave the robot in a controlled, defined state. The failure modes are summarized in Table 3.1.
3.11 Local Planner

The local planner implements the low level platform drive control and dynamic obstacle avoidance. As an input, it receives a goal in the global frame which it transforms into the local frame, and the current sensor data. It produces as its output the set of control velocities, \([v \omega]\).

3.11.1 Driving with Tentacles

Driving with tentacles is an efficient path-planning method [101] which utilizes a set of arcs (tentacles) as potential paths to follow for the robot. Each tentacle is defined by its radius and is grouped into speed sets. The tentacles in faster speed sets are grouped more closely together with smaller radii but a longer arc-length, and those in slower speed sets further apart with larger radii but smaller arc-length (Fig. 3.15). In [101], tentacle selection is performed by determining which tentacle is the longest before running into an obstacle given a radius of safety for the robot.

Figure 3.15: A visualization of the tentacles in a speed set. As the robot’s velocity increases, the tentacles shift closer to the center line and become longer.

3.11.2 Extended Driving with Tentacles Algorithm

In order to improve the algorithm in [101], AERO implements a modified version which takes into consideration a number of behaviors suitable for a robot performing a search-and-return
mission. The behavior-based extension of the algorithm is inspired by [9]. The modification combines the advantages of the tentacles algorithm, computational speed and simplicity and easy deterministic mapping to control outputs, with added behaviors aligned with the rover’s mission. The behaviors include:

- Move towards some predetermined goal destination
- Move towards unexplored terrain
- Move away from previously explored terrain
- Move away from ‘difficult’ terrain

The desired end behavior will select the longest tentacle which moves the robot closest to the goal (if any) while weighing the passing through the least amount of previously explored terrain, the least amount of difficult terrain, and exploring the most new area. To accomplish this, the local planner is provided an occupancy grid generated by the global planner. For each new occupancy grid, the local planner iterates across each point on each tentacle in a speed set and examines the corresponding point in the occupancy grid’s point trait. There are two values that are updated at each point $p_n$ on tentacle $k$ based on the point trait in the occupancy grid: The distance along the tentacle that has been traversed as in equation (3.7), and the length modifier given by equation (3.8). In addition, if the goal point or an obstacle is intersected, the iteration across that tentacle is halted.

$$\Delta l[n] = \|p[n], p[n-1]\|_2$$  \hspace{1cm} (3.6)$$
$$ll[n] = \Delta l[n] + ll[n-1]$$  \hspace{1cm} (3.7)$$
$$lm[n] = lm[n-1] - \Delta l[n] \cdot \begin{cases} 
   DW & \text{if FREE_HIGH_COST} \\
   TW & \text{if TRAVERSED} \\
   -UW & \text{if UNKNOWN} \\
   0 & \text{if FREE_LOW_COST}
\end{cases}$$  \hspace{1cm} (3.8)$$

where $DW$, $TW$, and $UW$ are weighting parameters for biasing the behavior by their respective point trait.

$$lg = -\text{GoalWeight} \cdot \begin{cases} 
   0 & \text{If no goal} \\
   \|p_{[n_{end}]} , p_{goal}\|_2 & \text{If goal}
\end{cases}$$  \hspace{1cm} (3.9)$$
Equation (3.10) presents that the final effective length of the tentacle is the summation of its actual length before reaching any obstacles (if any) and modifiers based on the desired behaviors. Therefore the longest, and thus 'best' tentacle will naturally become the one which best satisfies the behavioral criteria. The priority of the behaviors can be adjusted by modifying the DiffWeight, TravWeight, UnknWeight, and GoalWeight parameters.

It is important to note that the local planner does not actually attempt to drive the entire length of the selected tentacle. It produces a single set of control inputs to the platform. With frequent updates and properly tuned weighting parameters, the robot is able to follow a path towards the goal while avoiding obstacles. No guarantees at all are made on the optimality of this path, or that it will not become stuck in local minima. When used in tandem with a global planner, however, the majority of these pitfalls can be avoided.

In addition, as this is mainly a heuristic approach, instability can result when two tentacles on opposite sides of the robot’s center line have similar fitness values that swap back and forth rapidly. The resulting effect on the robot will be oscillations or jitter. To help damp these oscillations, a tunable rate-change damping coefficient or low pass filter is applied on the resulting tentacle selection. For example, if the previously selected tentacle was at index 30 and the next tentacle is selected to be at index 70 (the equivalent tentacle on the opposite side of the centerline), with the rate limit is set to 5, the damper will instead select tentacle 35. If tentacle 70 is selected again, it will select 40, and so on.

### 3.12 Global Planner

The global planner is responsible for planning in the global world frame and completing the mission-level task specified by the supervisor. It takes as its input the current mission-task, and a locally centered global map generated by other means such as a SLAM implementation. It produces as its output a series of way-points in the global frame that it can provide to the local planner. Due to the fact that global planner is not responsible for actually generating control inputs to the platform and that the local planner can improve the paths created due to imperfect data, the global planner can run at a low update frequency, on the order of once every 30 seconds. This saves computational resources on the robot needed for mapping and vision algorithms.
3.12.1 Carrot Path

Due to the behavioral properties of the local planner, the global planner can leverage its way-point connecting abilities to both save computation resources and compensate for imperfect data on the global map. Instead of attempting to produce a continuous path, the global planner instead produces a series of loosely-connected way-points called a carrot path. The carrot path strategy allows the global planner to relax many constraints on the path planner algorithm it uses. Even if it produces paths that are invalid because they cross an obstacle boundary, the local planner will compensate. If the point turns out to be unreachable due to imperfect data when the path was created, it will be corrected at the next re-planning phase. This would be undesirable in a robot operating in an enclosed environment with very good mapping data, but is actually beneficial in an exploring robot in poorly mapped spaces as it will generally cause it to explore more of the map. It also again saves computational resources as the planning doesn’t have to happen on a very fine search space.

3.12.2 Task-based Planning Strategies

The global planner also varies its strategies based on the current mission-task provided by the supervisor. Currently, there are two planning strategies that it uses: AStarCarrot, which is a simple modification of the A* algorithm to make it to produce carrot paths, and RRTCarrot, which is a similarly simple extension of the Bi-Directional Rapidly-exploring Random Tree Connect algorithm described in [54] to produce carrot paths and allow it to gracefully time-out with a partial path. Both planners operate in a purely 2D space. Orientation is ignored due to the fact that the only use-case that orientation matters, collecting an OoI, is handled by a control system that supersedes the navigation system entirely.

AStarCarrot is used when the robot is attempting to navigate to a specific point, such as an OoI. Its path-costs are based on the same weighting values as the local planner, and its heuristic is the Euclidean distance to the goal point. Its search-space is discretized to approximately \( \frac{1}{100} \) the distance between points in the Carrot Path, to ensure that the output is at least reasonably sure to be reachable. The final path is then culled to only contain points spaced for the carrot path.

RRTCarrot is used when exploring the global map for OoI’s. Normally RRT algorithms are avoided in 2D space because they produce very sub-optimal and guaranteed inconsistent between planning runs paths. However, when searching for OoI’s this random behaviour is desirable because it encourages a higher rate of exploration. To enforce at least some level of consistency
and reduce the likelihood of the planner sending the robot to one end of the field in one plan and then the other end on the next plan due to random chance, a rough ‘search pattern’ of very distantly spaced way-points are connected in series using the RRTCarrot planner. Similarly to AStarCarrot, the ‘step-size’ when connecting nodes in the RRTCarrot is set to be approximately $\frac{1}{10}$th the distance between points in the carrot path, and the final output is culled.

### 3.13 Sample Detection and Retrieval

To detect the samples and classify them, we implement the standard OpenCV version of HOG algorithm. Fig. 3.16 details the flow of information in the vision system, and how the sample location is passed to the manipulator controllers. Once an object is detected, its 3D location information is extracted from the disparity maps generated by the stereo cameras. The manipulator controller then moves the arm to the desired position using a inverse Jacobian velocity controller that takes the current arm position and desired position as inputs.

![Figure 3.16: A flow diagram showing the components of the vision system for detecting samples. The raw image streams are inputs, and after the samples is identified in the frame, it is passed to the stages of the manipulator controller to be picked up.](image)

62
In order to train the HOG, training images of each sample need to be provided to train the classifier. Initially, a cloth-lined and lit table-top stage was used to generate the images. This allowed the background to be easily removed from the image, and leave just the sample in the image. After extensive testing and training sessions though, this method did not provide good detection rates.

In order to test the effect of the background on the training images, a new set of images was generated with a white noise background behind the sample. The detection rate using this set of images increased significantly. We presume the detection rate increases because the white noise background averages out and the classifier ignores it, unlike a plain background.

### 3.14 Arm Controller

The arm controller node determines the type of grip to use based off of the object classification, and its orientation. The arm controller calculates a rough path and approaches the object based on the best gripping strategy for the particular object. The velocity controller node receives a desired point from the arm controller, and converts it into a set of joint velocities required to get to the point. The first part of this node is the position control. When the desired position of the arm is received, the node first calculates the current position of the arm by feeding the joint angles, through the forward kinematics. Based off of the two points the linear and rotational errors are calculated. These errors are then each fed through separate PD controllers. The output of the controllers are then fed through a gain in order to convert the output into Cartesian velocity. The second part of this controller uses the arm Jacobian calculated using the arm kinematics.

### 3.15 Results

The navigation and vision algorithms described above were implemented with Robot Operating System (ROS) and OpenCV in Ubuntu Linux 12.04 on AERO. The software system was designed with flexibility and ease of implementation in mind. ROS provides a set of libraries and development tools on top of Linux tailored to robotics development that make it easy to develop software in multiple language across multiple systems. The ROS architecture involves creating a number of nodes that communicate using an inter-process message passing system. In addition, there are a large collection of software tools and libraries that implement a number of algorithms and common tasks in robotics. Data collection for testing and visualization of data was made especially easy with these tools.
CHAPTER 3. ROVERS

The test environment consisted of a series of impassible static obstacles, and one randomly moving dynamic obstacle near the center of the path from the start to the goal position. An example of the path taken by the robot can be seen in Fig. 3.17. It demonstrates the robot successfully dodging the dynamic obstacle while still moving towards the goal. A snapshot visualization of the data of this event can be seen in Fig. 3.18.

![Figure 3.17: A plot of the position of the robot in the global frame which includes a dynamic obstacle (a person moving in front of the robot) at location \( x \approx 0.0, y \approx 8.75 \). The blue line is the path the robot took. The green dashed line is the carrot path. And the purple stars are points where the carrot path was re-planned.](image)

Fig. 3.19 shows the output of the vision processing algorithms. The samples are detected, and their positions with respect to the arm are calculated. The arm controller then moves the manipulator into position to retrieve the samples. During benchtop testing, the sample was successfully retrieved multiple times.

3.16 AERO 2014

After developing many of the capabilities and necessary modules in 2013, we improved the robot for the 2014 competition. In this section we present our work towards more robust and reliable autonomous operation for sample return rovers. The work was guided by the 2014 Sample
Return Challenge, and focused on the integration of vision-based navigation and object recognition algorithms for AERO, shown again in Figure 3.20. The presented work builds on experiences from developing the rover for the 2013 SRR Challenge [29], and previous work with the ORYX 2.0 rover [16, 30, 4]. New training procedures for the cascaded classifiers used previously in [29] led to significantly improved detection and tracking performance. In addition, modifications were made to the previous software architecture to speed-up detection and therefore enable detection in situations where the rover is moving. A GPU based implementation was also developed with significant improvements in detection rates in specific situations. AERO also saw improvements in navigation performance, with a completely redesigned navigation architecture based on the best practices suggested in the new Robot Operating System (ROS) control framework\footnote{http://wiki.ros.org/ros\_control}. A new high level state machine significantly improved in handling conditions that previously caused navigation faults, and tighter integration between the subsystems in the navigation architecture led to better accuracy. A new method to integrate global information for virtual reality tags was developed and led to improvements in docking back at the starting location. We discuss all of these improvements in detail, and provide insight into our experiences with the algorithms and how they need to be
Our long-term motivation was to leverage these improvements in autonomous operations to design, develop, and validate a HiLCPs framework for enabling operators to control multiple semi-autonomous rovers to complete cooperative manipulation tasks. Control inputs to the system from operators are limited to high-level, abstract commands, and the framework is expected to autonomously handle the coordination between the robotic systems to comply with the intent of the operator commands. Cooperative manipulation will be required to explore areas where the abilities of one rover system are insufficient, because the terrain is too rough, steep, or loose, and specifically when a sample cache may need to be handed off between two systems. These actions will be safer with less risk to mission success and the health of the individual systems with improvements in the autonomous behaviors and actions of the system as a whole.

### 3.17 Vision System

To detect the samples, classify them, and retrieve using the vision system, we implement the standard Haar cascade classifiers \[99\] from the OpenCV libraries within the ROS system. Once
an object is detected, its 3D location information is extracted from the disparity maps generated by the stereo cameras. The details of the detection algorithms is not explained, because the approach has not significantly changed from the 2013 competition. Improvements in the structure and ROS implementation of the detection algorithms were made, but the core approach is the same as in \[29\] \[30\].

Significant improvements though were made in detection performance, by generating a better training data set, and iterative adjustment of the training parameters exposed by the OpenCV Haar cascade training utility. In order to train the classifier, training images of each sample need to be provided to classifier training utility. In the previous year, we utilized images of the sample with cloth-lined, lit table-top stages, and random noise backgrounds, resulting in detection performance that was usable but limited in cases of mixed-shade or unexpected orientations of the sample object. We determined that the practice of taking a few sample images and generating thousands of synthetic images using distortions may be sufficient in many cases, but does not provide the necessary robustness in the variable conditions expected during the competition.

We therefore changed the approach to utilize over 3000 hand-cropped images for the precached hook objects in various conditions to generate a very diverse and wide-ranging data set.
CHAPTER 3. ROVERS

Figure 3.21: A sample subset of the training images used for the precached hook sample of the NASA SRR Competition. Over 3000 such images were used to generate the classifier for the hook.

A small utility that varies the exposure was used during the collection of the data sets to simulate varying lighting conditions. In addition, data sets from various locations were included with various leaves, sticks, and natural debris in the negative sample set to help eliminate false positives from debris in the field. Training on the data-set with over 3000 positive images and 9000 negatives would take on the order of 2-3 days on a modern Core i7 computer. By running multiple simultaneous instances of the single-threaded training program, we could quickly generate multiple classifiers with different training parameters, leading us to a set of parameters for the training that yielded very good classification, even in darkly shaded areas, areas with leaves and sticks, and mixed shade.

In addition to using the CPU based Haar cascade classifier provided by OpenCV, the GPU based classifier, trained on using the same data-set, was also tested. When running on the NVIDIA Tesla K20 GPGPU inside AERO the classifier ran 7 times faster while utilizing only 20% of the GPU’s capacity and with minimal CPU usage. Although the GPU classifier significantly outperforms the CPU classifier, the number of detections were significantly fewer than the CPU. Using

2https://github.com/RIVeR-Lab/aero_srr_14/blob/hydro-devel/vision/src/cascade_classifier.cpp
the CPU based classifier introduced additional delay before the results were reported; however, in practice the delay was not a limiting factor for the speed that the robot could operate at. The effects of the delay that may have caused error in the detected position of the sample while the robot was moving were mitigated by ensuring that the robot had reached a full stop before attempting to accurately detect the position of a sample.

A feature we implemented on AERO, but did not end up using during competition because we only tested it in specific cases, was visual odometry through the ROS implementation of libviso2 \cite{libviso2}. Despite not using it during competition, the algorithm does work well and provides another source of information that can be provided to the Kalman filter resulting in a more accurate estimate of the pose of the robot throughout its operation. The algorithm works by tracking how objects within the frame of the stereo cameras move, and then correlates that motion to the motion of the

Figure 3.22: Four screen shots from the ROS visualization tool, rviz, showing the visual odometry package tracking the motion of the robot. The robot drives forward first, takes a slight right, and then a slight left. The red line traces out the path the visual odometry package calculated for the robot. The visual odometry was tested and proved useful in certain cases, but ultimately not used during competition.
stereo vision system. Since the camera sensors are over a megapixel each, the processing rate of the visual odometry is slow compared to the dynamics of the system. To overcome this issue, we tweaked several parameters of the default implementation to reduce the number of regions the algorithm is searching, tailor them to be more sensitive in the yaw of the platform by making the regions tall and skinny across the camera frame, reduce the number of points matched in each region, and decimating the input images to a quarter of the source resolution. These modification did not significantly affect the accuracy of the algorithm, but raised the frame rate of the processed pose estimates to about 12-15 frames per second. These simplification proved sufficient to provide usable visual odometry data from the stereo cameras.

3.18 Navigation and Planning

The implementation of the planning architecture for the robot is done by creating multiple levels of planners that ranges from controlling individual components of the robot to planning the high level tasks throughout the entire run. The planning process is divided into separate systems in order to make each system both easy to understand and to modify. The levels communicate by passing desired actions to a lower level controller that will execute the action and then communicate back the result (success or failure). This allows the higher level controller to try again or attempt to find a different way of completing the task. Figure 3.23 shows the organization of the different planning systems and the tasks they perform.

At the highest level there is a single finite state machine that represents the current high level task that the robot is trying to accomplish, such as driving off the platform, picking up the sample and driving toward the platform. Each of the states represents a different requested action that is dispatched to a lower level planner to execute. This high level view of the robot’s current state allows for easy debugging and reuse of similar actions.

While a finite state machine manages the high level operation of the robot, lower level planners manage individual components of the robot. Separate planners run for the arm and drive controllers that allow the high level planner to command actions such as move the arm end effector to a pose or drive the base to a location. The drive system has two levels of planning, a higher level planner that generates waypoints that the robot will use to get to a final goal and a lower level drive planner that generates a velocity for the robot to get to each waypoint.
3.18.1 Local Drive Planner

A local drive planner is implemented using the ROS base local planner\footnote{http://wiki.ros.org/base_local_planner}, which implements the Dynamic Window Approach for collision avoidance as described in \cite{39}. This approach works by sampling the robot’s control space and then projecting the result of those control inputs into the future. The resultant trajectories are then scored based on a number of factors including obstacle proximity and goal proximity, ignoring trajectories that would cause the robot to collide. The best control input is then chosen based on score and then executed. This process is executed continuously until the robot reaches the desired waypoint generated by the global planner. If none of the trajectories are valid, because the robot got too close to an object for example, the local planner supports a number of recovery behaviors including backing away from an obstacle in front of it.

As the local planner’s goal is in close proximity to the robot (within 10 meters), the planner only operates on a small costmap that is generated as a subset for each new laser scan. As
it does not operate on a global costmap that is generated from multiple laser scans, the planner is not affected by potential incorrect localization that causes errors in the global costmap, which could lead to the robot crashing into an obstacle. Additionally because it only has to search paths inside of this region, requiring much less computation than planning an entire route through the environment. This allows the planner to run at 20 Hz allowing the robot to react quickly to changes; while the environment of the competition is static, at times the robots 2D perception of the world through the laser scanner could change quickly. For example, as the robot approaches a hill it slowly inclines upward (if the hill is not too steep) causing it to see the hill further out as it moves upward. If the robot did not replan quickly enough it would see the hill as a wall and try to drive around it.

### 3.18.2 Global Drive Planner

In order to plan a path to locations that are not near by to the robot, a global drive planner is used to generate local waypoints to the goal that would be sent to the local planner as a goal. The ROS navfn package\footnote{http://wiki.ros.org/navfn} global planner implementation is used; it implements a global planner that computes a plan from a start position to end position using Dijkstra’s algorithm. Because this algorithm has to plan over the entire known environment it can not run nearly as frequently as the local planner. Instead the planner only runs when it receives a new goal or when the local drive planner is unable to find a path to the next waypoint in the current plan. When it runs, a new plan will be generated from the current position to the current goal. When it is detected that the robot has reached (or is very near) the most recent waypoint, a new waypoint will be sent to the local planner.

### 3.18.3 Arm Planners

Although our arm controller does support planning a path to a desired end effector pose in Cartesian space, it does not support specifying an approach vector or easily avoiding the robot itself. Instead of using a full motion planner to plan the desired trajectory, a number of finite state machines are used to implement planners to execute actions such as picking up a sample or stowing the arm using the controller’s built in planning. They are each represented as sequence of positions either in Cartesian space or state space of the arm that the arm would move to. This gives well defined trajectories for the different actions that the arm needs to perform. When picking up a sample, the location of the sample is known to be in a small bounded area on the ground in front of the robot.
This allows the intermediate poses that the arm takes to remain the same except for the poses where the fingers initially grasp the sample.

3.18.4 High Level Planner

The high level planner guides the robot through the operation of the competition. For level one of the competition, it can be divided into three sections: navigating to the approximate location of the sample, finding and collecting the sample, and returning to the start. These are implemented in a single finite state machine shown in Figure 3.24 and described below.
3.18.4.1 Navigating to the Sample

After the robot finishes booting, it begins executing the state machine. The first thing that is done is to wait for the hardware to finish initializing. This involves waiting for all of the drivers and software to start up and the IMU to finish calibrating. The arm is then placed in the stowed position so that it does not obscure the cameras. Once this is complete the robot prepares to leave the platform by software shuttering the laser scanner so that it does not place obstacles on the local or global costmap. This is done in a number of places throughout the state machine in situations that are known to be safe, but in some cases can lead to the laser scanner indicating that there is an obstacle in front of the robot when there is not. In this case the laser scanner is shuttered because when the robot leaves the platform it tilts down so that the laser scanner sees the ground. Since it is known that there are no obstacles directly in front of the platform, the laser data can be safely ignored as the robot drives off. The robot then drives straight off the platform by driving a specified distance away from the start and then unshutters the laser.

After successfully leaving the platform the robot then begins to navigate towards the known approximate location of the sample. As the level one sample was known to be behind the platform a path was chosen that would ensure that the robot traveled around the platform. To accomplish this the robot first drives to a position off to the side of the platform and then drives toward the approximate location of the sample.

3.18.4.2 Sample Location and Collection

Once the robot reaches the edge of the zone that the sample can be found in, it begins searching for the sample. It starts by driving to the center of the approximate location. If it sees the sample while it is driving it will stop and attempt to pick it up. Otherwise when it reaches the center it will begin a spiral search pattern centered there with each rotation separated by a meter ensuring that the entire area is thoroughly searched. If at any point during the spiral the sample is detected, then the robot will attempt to pick it up.

When the robot sees the sample, it immediately stops and waits two seconds so that it comes to a complete stop and any vibrations dampen. After this, the robot attempts to detect the sample again. This is done so that an accurate 3D position can be estimated for the sample using the disparity image from the stereo camera pair. Once the position is estimated, the robot then chooses what to do based on how far away the sample is. Because the position estimation is less accurate at farther distances, the robot does not immediately try to drive up to the sample and pick it up.
based on one detection, but instead slowly moves closer to it using multiple detections. If the robot is far away from the sample then it will drive to a position near the sample (about 1 meter away), detects the sample again, and repeats. If the robot is already at that distance then it drives up to the pickup position (so the sample is 10-20 cm away), detects the sample and repeats. If the robot is then detected to be in the pickup position then the arm is commanded to pick up the sample. If at any point in the process the sample is no longer detected the robot would abort trying to pick up the sample and restart the spiral search pattern.

After the robot attempts to pick up the sample, the robot would try to detect the sample again in order to ensure that it was actually picked up. If the sample was not detected then the robot would assume that it successfully picked up the sample and return to the starting platform. If the sample was detected again then the sample was not successfully picked up and another pickup would be attempted by repeating the detection, drive and pickup process. If the arm is interrupted or never completes the pickup trajectory after a timeout, such as if it ran into something or could not plan to a given configuration, then the robot will abort the pickup and stow the arm. After stowing the arm the robot will then drive backward a meter and attempt to pick up the sample again. By doing this the robot is able to reposition in order to potentially improve the position for the pickup.

After the sample is detected the laser scanner is shuttered for the entire process of navigating to the sample and picking it up. This is done so that the sample is not accidentally detected by the laser scanner and interpreted to be an obstacle making it impossible to drive to it. This is also known to be safe because the samples are known to not be located near any kind of obstacle. Once the sample pickup is completed or abandoned the laser scanner is unshuttered so that the robot does not run into anything as it continues driving.

3.18.4.3 Returning to the Start Platform

After the robot successfully picks up the sample it begins to return to the starting platform. To do this the robot drives towards the starting platform while searching for the fiducial markers that were placed on the platform. If the robot does not see the fiducials it begins a spiral search pattern until it sees them. Once the platform is identified the robot drives to a position in front of the platform. The laser scanner is shuttered again so that it does not accidentally see the platform as an obstacle and then drives onto the platform and stops, completing the challenge.

In order to accurately detect the location of the home platform the AprilTags\textsuperscript{70} fiducial system was used because they allow for the calculation of the relative position of the camera to
CHAPTER 3. ROVERS

the tag in addition to providing a collection of many unique tags that can be used. A simple C++ library\[\text{5}\] is available that allows for efficient extraction of the tag positions from an image. Many of these tags were placed in various orientations on the starting platform such that at least one could be seen from any angle as shown in Figure \[\text{3.25}\]. Because the position of each tag was well known relative to the platform the position of the platform could easily be computed. Each detection of the beacon was then used as an input to a Kalman filter that would estimate the position of the platform. This estimated position was used as a reference frame when generating goals for the robot to drive to when returning to the platform.

![Figure 3.25: The beacon covered with AprilTags that was placed on the starting platform.](image)

The work on the vision system in Section \[\text{3.17}\] is an implementation of the perception engine from the shared control framework where all the relevant sensor data is aggregated from the different platforms. It is the first place that has access to data from multiple sources, so sensor fusion algorithms to confirm localization of robots, samples of interest, or landmarks can be implemented. In addition, the work on the navigation and planning system in Section \[\text{3.18}\] is applicable to the action engine, achievable action gate, and low-level robot control. The more robust autonomy that can be pushed off to the individual robots, results in less operator load and commands that can be at higher more abstract levels.

\[\text{5}\]http://people.csail.mit.edu/kaess/apriltags/

76
CHAPTER 3. ROVERS

Efforts to combine observations from multiple systems are already providing unprecedented results, especially with the collaboration between ground based rovers, the Mars Reconnaissance Orbiter (MRO), and the human operators of each system. For example, work in [15] demonstrates collaboration between orbital maps and rovers for effective long range localization. As these collaborations begin to include more systems, the infrastructure of the shared control framework can provide a guide to enable structured development of collaborations between multiple robots.

3.19 Analysis of Teleoperation and Autonomous Tasks

Overall, we learned several valuable lessons through the implementation of ORYX 1.0, ORYX 2.0, and AERO. Working within the confines of the Robo-Ops and Sample Return Challenges provides unique problems that need to be addressed, and resulted in two systems that demonstrated sample acquisition tasks for space exploration. At the conclusion of the 2014 Sample Return Challenge, we took the opportunity to reflect on the lessons learned throughout both competitions, how certain tasks were difficult to do using teleoperation, and how others were difficult to fully automate. The next sections highlight these lessons learned, their contribution relevant to systems developed in the future, and how to best leverage heterogeneous human-robot teams capabilities from a holistic perspective.

We identified and compared a number of approaches that have been reported in literature and could be utilized in future missions such as teleoperation, teleoperation with latency, supervised autonomy, shared control, and full autonomy modes of control. We outline the control framework for the Oryx teleoperated robots, the procedures developed for the operators to efficiently, in terms of robot resources and operator input required, execute the mission tasks, and the operator control interface for displaying sensor data to the user. In addition, we describe the autonomous control framework for AERO, the algorithms for searching unknown areas, and the testing procedures to validate the developed autonomy algorithms. Based on the experiences of the Oryx operators and the performance of AERO, we devised semiautonomous shared control behaviors that tasks in a planetary exploration mission would benefit most from added autonomy. For example, behaviors that constrain and simplify multi-DOF manipulator motions significantly reduce operator load, and lead to more efficient sample pickup or coring operations. Ultimately, behaviors that enable swarm or formation exploring would enable a single operator to control heterogeneous teams of exploration robots.

An emphasis is placed on a minimal set of semiautonomous behaviors that would allow
flexible operation of the rover and ease of memorization for the operator. In addition, because the high-level shared control framework allows human-in-the-loop control at a task level, a small set of flexible behaviors can be combined by the operator to accomplish a variety of tasks. As exploration missions go farther and become more complex, high-level shared control with emphasis towards fully autonomous exploration will allow the rovers to explore more terrain than ever before in an efficient manner.

3.20 Teleoperation with Oryx 2.0

Significant amounts of research have been conducted in the previous couple of decades on effective teleoperation. In [19], the authors provide a thorough overview of many of the techniques and challenges in working with a teleoperated system. Many of the challenges, especially the ones associated with time-delay, were experienced during operations with Oryx 2.0. The general teleoperation architecture with Oryx 2.0 for the Robo-Ops competition requires two operators to drive the platform, control the manipulator, and assimilate the proprioceptive and visual data from the rover. All of the data required to maintain the safety of the rover and find samples is available to both operators and both can control the rover if needed. This is functionally similar to the operation of an airplane, where both the pilot and co-pilot have a clear distinction of responsibilities at one period of time, but both have access to the same information and can take complete control if needed. The rover is controlled with commercially available gaming controllers since they provide a very intuitive method to control a skid-steer platform and plenty of buttons for behaviors and action scripts.

The separation of responsibilities between operators usually means the primary operator is in control of the base platform and manipulator. The graphical user interface (GUI) displays either the front-facing drive camera, to drive to a requested location, or the arm-mounted camera, to line up a sample for pickup. The secondary operator is in control of the pan-tilt unit of the mast camera, constantly scanning for potential samples. In addition, the secondary operator helps the primary operator confirm a successful sample pickup and storage using the mast camera. If small corrections during pickup or sample storage are needed and the secondary operator has a better vantage point, they can take control of the manipulator or base and make the adjustment. This arrangement has been experimentally shown to work very well, and in fact during the 2012 competition, there was no period of time that a sample was not either in the process of pickup or in view of the mast camera and secondary operator. The limiting factor for the number of samples cached was the time it took
CHAPTER 3. ROVERS

Figure 3.26: The ORYX 2.0 primary operator graphical user interface (GUI) is shown. Various values such as internal temperatures of components and battery voltages are provided on the left. A large area in the center shows the views from the cameras. The pose of the robot, angle of camera mast, network information, and a small second camera view are all provided on the right. The secondary operator has a very similar GUI, but the mast camera is generally displayed in the center panel. There are few buttons on the GUI, since the robot is primarily controlled with through commercially available gaming controllers with two sticks and multiple buttons.

to line samples up and successfully manipulate them.

Several small but very relevant features were implemented on ORYX 2.0 to assist the teleoperation of the platform. By far the most important aspect is to be able to provide the operators of situational awareness during missions. In Figure 3.26, the camera mast visualization on the right panel of the GUI was implemented to significantly reduce the time to line up a potential sample and improve the communication between the operators. When a sample is found, the secondary operator can immediately tell the primary operator how many degrees they need to turn to line the sample up. The visualization helps the primary operator confirm the information and make a mental note of about how much the platform needs to turn. This process work well, so the sample does not need to be visible in the primary drive camera. If lined up properly, the primary operator can just drive straight forward until the sample enters the field of view and pickup can be initiated. Previous research focused on developing a controller that would keep the robot driving a straight line, even over rough terrain [16].
Sample pickup proved to be one aspect of operations that was especially tricky. Initial testing indicated that lining up the manipulator to the sample and conducting a pickup took excessively long and sometimes resulted in significant forces applied to the end effector due to operator misjudged distances. Figure 3.27 shows ORYX 2.0 picking up a purple sample along with the clear polycarbonate plate and small piece of tape in the corner. These additions may not seem critical, but the clear plate allows the operator to do two things: (i) align the sample to the scoop and (ii) because the optical properties of the plate change as it bends, the operator receives visual feedback if they are pinching the sample enough to pick it up or if the sample is in the bucket and the scoop is completely closed. In addition, this visual feedback allows the operator to open a small gap in the scoop to drain any extra sand in the scoop without dropping the sample. The small piece of tape in the corner indicates to the operator when they are just barely off the ground helping reduce unnecessary wear and tear on the scoop and reduce the amount of extra debris picked up along with the sample. Sample drop off is a completely automated sequence saving the operators time. Once the drop off sequence is initiated, the rover can immediately start driving to the next sample. In the mean time, the secondary operator can confirm a successful pickup with the mast camera.
Overall, the success of the teleoperation depended heavily on how experienced the operators were, and how much time they had to familiarize themselves with the platform. Many of improvements implemented to make operation easier proved critical in efficiently locating and picking up samples. The examples provided here are demonstrated in a relatively simple mission scenario, but the concepts and principles are relevant to teleoperation in significantly degraded conditions.

### 3.21 Autonomous Operation with AERO

Because of the requirements of the Sample Return Challenge, the algorithms developed for AERO had to function completely autonomously. This is an advantage in some aspects because the algorithms do not need to account for the actions of the operators, but makes the challenge difficult since the algorithms need to flexibly handle a variety of possible scenarios. Several approaches and algorithms we employed worked well, and many did not perform as expected in relatively open and sparse environments searching for samples. AERO’s mission can be split into three main sub-tasks: navigating and localizing within the large outdoor area, identifying and classifying samples, and retrieving the samples with a manipulator.

AERO’s navigation system at its core uses the concept of hierarchical control which can be divided into three levels shown previously in Figure 3.14 [29]. By splitting the navigation system into a hierarchical structure, the behaviors that require high loop rates and low latency such as obstacle avoidance can be done quickly, while the relatively slow planning phases can be implemented on a longer timescale. This allows the tentacles driving algorithm to operate faster than the dynamics of the robot preventing a crash into an obstacle, while allowing the high level planning to take place concurrently. This saves computational resources and provides an effective way to navigate and search for samples.

Sample detection and classification is entirely implemented by the computer vision system. The top mast cameras identify anomalies in the grass that could potentially be samples and mark them on a probabilistic map on the robot using very basic RGB threshold filters. The robot inspects each potential sample from a close distance using the fixed, front-mounted stereo vision system, and utilizes the classifiers previously described. Throughout the testing of AERO, sample detection proved to a consistent strength of the robotic system. The robot would detect samples at a similar distance as a human looking at the image stream, but the robot has a significant advantage that it knows the 3D location of the sample with respect to the robot. This proved to be the key
advantage that makes the robotic perception better for sample pickup than a system wholly reliant on operator perception.

The Jaco manipulator mounted on AERO has significant complexities with its inverse kinematics because of the specifics of the last three joints on the manipulator. They are designed at a 55 degree angle to reduce the number of potential pinch zones making the manipulator safer to use in unstructured environments. While the approach for picking up a sample, locating it in image-centric coordinates and controlling the arm in robot-centric coordinates to pick up the sample, is architected as a relatively simple pipeline, its operation proved to be faster, more efficient, and less prone to errors than the teleoperation of the ORYX 2.0 manipulator.

3.22 Shared Control and Behaviors

A spectrum of control autonomy exists for various robot systems ranging from teleoperation, teleoperation where latency hinders direct teleoperation, supervised autonomy, shared control, and full autonomy. Arguably, shared control, where the user input and various autonomous agents both have control of the system simultaneously is the least understood and studied. Most realizations of robotic systems in reality fall either into the teleoperated or autonomous categories. The rovers currently on Mars implement much supervised autonomy, executing their scripted behaviors autonomously with humans-in-the-loop ready to modify the scripts in case the rovers sense an unexpected issue. Of the few systems that do implement some sort of shared control, few are intuitive or work well. This is not due to deficiencies in the principle, but rather progressive attempts to solve a problem that is very difficult. The principle of shared control, if implemented in an intuitive and transparent manner can significantly improve the ability to search a large, unknown area and find samples of interest.

In [36], Zermelo’s navigation problem is used as an example to consider a range of methods as to how shared control may be implemented including traded control, indirect shared control through cues, coordinated control, collaborative control, virtual constraint, and blended shared control. We use these categories as a template to propose a set of shared control behaviors for rovers in sample return mission scenarios that would be useful considering the experiences and approaches that worked well with ORYX 2.0 and AERO.

In traded control, the user and rover can transfer full control of the platform between each other on either an event-driven or schedule-driven schedule. This modality of control would be especially useful while a rover is completing a tedious and monotonous task that happens to include
CHAPTER 3. ROVERS

a part that is intricate. The rover can complete the majority of the task by itself, but needs a little help to finish a critical part. A useful behavior during sample pickup might include allowing the rover to autonomously position its manipulator to pickup a sample in a gross manner, and then transfer control to a human operator to fine-tune the grasp and pickup strategy. Once secured by the manipulator, the rover can regain control and autonomously store the sample. The supervised autonomy of rovers as described above is essentially a form of traded control.

Indirect shared control through cues guides the inputs of the operator to ultimately change the behavior of the robot platform. Visual, auditory, tactile, and haptic feedback can all be utilized to implement indirect shared control through cues. In a sample return mission, haptic and tactile feedback could be very useful for the retrieval of samples. One challenge of retrieving samples with ORYX 2.0 is the inability to judge the forces on the end effector and sample well. Haptic feedback could be effective in confirming a good grasp. In addition, the cues can be used to help the operator align the rover base for optimal retrieval position where the operational space of the manipulator is optimal.

In coordinated control, the rover has full control but accepts operator inputs at a lower dimensionality than the task the robot is currently trying to complete. For example if the operator does not care about the orientation of the rover but would like to control the position and minimize the energy required to move the rover, a coordinated control algorithm can accept position inputs from the user and plan motions for the rover that minimize the energy used. This can be further extended by allowing tracking errors of the controllers to vary in different dimensions. For example if the lateral error of the rover is not critical, but the forward position must be carefully controlled, the coordinated controller can take that into account and allow the rover to stray laterally and conserve energy.

In collaborative control, the rover has control of some degrees of freedom while the operator has control of others. This is useful in situations where sample retrieval may be difficult due to poor stability of the platform. The collaborative controller can work to keep the platform stable on the difficult terrain, while the operator can focus on controlling the manipulator to retrieve the sample. Dangerous terrain and situations can quickly mentally overload even the best trained operators. By allowing the rover to autonomously handle a portion of its safety critical tasks, the operator can focus on completing the task quickly and efficiently reducing the risk to the rover.

Virtual constraints are already implemented on many robotic systems, not limited to space systems. In this control modality, certain inputs are not allowed based on the state of the system. In many cases, these inputs are not allowed for safety reasons. For example, a rover can sense when
its center of mass is leaving the polygon of stability and tipping is an imminent hazard. The virtual constraint can limit the inputs to allow the rover to travel in the safe direction away from the tipping hazard. Other applications include speed limits or speed minimums depending on the stability or structure underneath the rover. For example, to reduce the risk of sinking in very soft sand, it may be beneficial to drive over the sand relatively quickly until more stable terrain is found. This would have been useful for ORYX 2.0 where it got stuck several times in the gravel craters due to its inability to climb out slowly.

Finally, blended shared control offers a very broad range of possibilities where the operator input and autonomous controllers on board the rover combine their effects on the robot and control the rover simultaneously. The most exciting application for this control modality is allowing the rover to execute certain motion primitive autonomously, but guiding the overall search strategy through the human input. The motion primitives will have to be generated with respect to the high level search patterns or methods that the human commands, but these could be generated offline through massively parallel simulation. This will allow an operator to guide a rover with few inputs, allowing them to efficiently control more than one rover. Consider the possibility of a swarm of exploration rovers where the general behavior of the swarm is controlled by the operator, but each rover is driving itself within the swarm.

The proposed behaviors are by no means exhaustive, but these are a relatively minimal set that given the experiences with ORYX 2.0 and AERO, we feel significantly improve operations of the sample return rover in Earth analog scenarios.
Chapter 4

Humanoid Robots

4.1 Introduction

Disaster response has been an impactful research focus in advancing the capabilities of intelligent robots [11] [18] [69]. The Defense Advanced Research Projects Agency (DARPA) Robotics Challenge (DRC) [75], announced in 2012, is the new frontier in efforts to effectively deploy robot systems in natural and man-made disaster situations. The DRC is aimed at advancing robotics research and development in multiple directions including perception, manipulation, mobility, and supervised autonomy. The challenge tasks are motivated by real disaster sites such as the Great Eastern Japan Earthquake in 2011 [65] and Hurricane Katrina in 2005. The tasks include a variety of manipulation (turning valves, clearing debris, opening doors, attaching a firehose, and operating power tools) and mobility (driving a vehicle, climbing a ladder, and traversing rough terrain) tasks. The DRC Trials took place on December 20-21, 2013 and the event was viewed as a formative assessment of the teams participating in the DRC Finals [1].

Humanoid robots have advantages for completing a wide variety of tasks in human environments such as turning valves [3], traversing rough terrain [45] and driving a vehicle [80]. However, despite receiving great attention to date, humanoid robot motion planning and control remain challenging research topics. The development of computationally efficient algorithms to solve the inverse kinematics (IK) problem for dual arm manipulation tasks has been the focus of [98]. Probabilistic IK solvers that rely on rapidly-exploring random trees are shown to be effective in re-grasping tasks. A full body balance control architecture for humanoid robots with the input
CHAPTER 4. HUMANOID ROBOTS

being desired contact forces is presented in [91]. Torso posture control is presented as an example demonstrating the control approach. The high degrees-of-freedom that are inherently present in humanoid robots have been used to develop control methods such as whole-body torque control with multipoint contacts [85]. Their controller takes into account the relationship between the contacts and the desired center-of-mass maneuvers to achieve balanced motions that rely on stability polygons determined by the contact points.

Figure 4.1: The DARPA Robotics Challenge greatly propelled the development of humanoid robots for disaster response within supervised autonomy frameworks. The contributions of our team included elements from both the human and robot halves of the framework. The contributions mainly focused on the right side though, with limited but important contributions to the cyber elements on the left.

Figure 4.1 shows the components of the shared control framework, mainly concentrated on the right side, that the DRC contributions constituted. The DRC specifically required supervised autonomy, so it was the first application where the shared control framework had significant information exchange through the human-robot interface. Selectively added robust autonomy and efficient human operator interfaces were the key features needed for successful competitors. The cloud engine is highlighted in this specific application of our shared control framework because the DRC allowed the use of field computers to offload heavy computational from the robots, described
CHAPTER 4. HUMANOID ROBOTS

in more detail in Figure 4.3. We utilized this ability to generate synthetic camera views to increase operator awareness. In addition due to the need for robust autonomy to allow control of the robot at a high, abstract level, the top of the framework is active similar to the AERO implementation.

4.2 Robot Hardware

4.2.1 ATLAS Robot

The Worcester Polytechnic Institute (WPI)-Carnegie Mellon University (CMU) DRC team, originally known as WPI Robotics Engineering C Squad (WRECS) which took the 2nd place in the Virtual Robotics Challenge in June 2013, participated in the DRC Trials as the only Track C team. The WPI-CMU DRC team was provided with an ATLAS robot, designed and built by Boston Dynamics specifically for the DRC. ATLAS is a 150 kg humanoid robot with 28 hydraulically actuated degrees of freedom (DOF): 6 in each arm, 6 in each leg, 3 at the torso, and 1 in the neck (Figure 4.2). Figure 4.2: ATLAS robot and a stick figure showing the position and orientation of joints.

Table 4.1 concisely describes the arm joints. In addition to load cells for force sensing at hands and feet and a fiber-optic inertial measurement unit (IMU) at the pelvis for estimating robot pose, each actuator on the arms has a linear potentiometer for position measurement and two pressure sensors to determine the joint forces based on differential pressure measurements.
CHAPTER 4. HUMANOID ROBOTS

Table 4.1: A description of ATLAS arm joints

<table>
<thead>
<tr>
<th>DOF</th>
<th>Joint</th>
<th>Min</th>
<th>Max</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SHY</td>
<td>-90°</td>
<td>45°</td>
<td>135°</td>
<td>Shoulder axial rotation</td>
</tr>
<tr>
<td>2</td>
<td>SHX</td>
<td>-90°</td>
<td>90°</td>
<td>180°</td>
<td>Shoulder perpendicular rotation</td>
</tr>
<tr>
<td>3</td>
<td>ELY</td>
<td>0°</td>
<td>180°</td>
<td>180°</td>
<td>Elbow axial rotation</td>
</tr>
<tr>
<td>4</td>
<td>ELX</td>
<td>0°</td>
<td>135°</td>
<td>135°</td>
<td>Elbow perpendicular rotation</td>
</tr>
<tr>
<td>5</td>
<td>WRY</td>
<td>0°</td>
<td>180°</td>
<td>180°</td>
<td>Wrist axial rotation</td>
</tr>
<tr>
<td>6</td>
<td>WRX</td>
<td>-67.5°</td>
<td>67.5°</td>
<td>135°</td>
<td>Wrist perpendicular rotation</td>
</tr>
</tbody>
</table>

The robot’s sensor suite also includes three IP (Ethernet) cameras positioned around the robot to allow for a near 360° view of the surroundings and a Carnegie Robotics MultiSense SL sensor head which provides visual input to the operator. The MultiSense SL contains a set of stereo vision cameras and a rotating LIDAR and can be used to produce a point-cloud to represent the robot view. Because of the high power and data requirements of the system, ATLAS is tethered to a base station. This tether supplies the robot with 480V power, a fiber-optic connection of 10 Gbit/sec for network communication, and water cooling. Sensors communicate directly to the Control Station over the fiber-optic network (Figure 4.3).

![Network Layout](image)

Figure 4.3: A visualization of the network layout between the robot sensors and the operator control station.

4.2.2 Control Station

The control station enables the operators control the robot remotely without line of sight, and is comprised of 5 computers: the field computer, the primary Operator Control Unit (OCU)
CHAPTER 4. HUMANOID ROBOTS

and three auxiliary OCUs. The field computer manages all communications with the robot, limiting and compressing the high resolution data from the robot to be sent to the OCUs, and runs the autonomous robot software. The primary OCU (OCU 1 in Figure 4.4) is tasked with decompressing and relaying information sent over the communications pipeline. The auxiliary OCUs act as terminals to accommodate information to and from the users. The field computer is connected directly to the ATLAS network through the fiber-optic connection, it is also connected to OCU 1 through a limited bandwidth connection provided by DARPA. At the DRC trials, the connection bandwidth and latency alternated every minute between a bandwidth of 1 Mbit/second and latency of 0.05 second each way (0.1 second round-trip) and a bandwidth of 0.1 Mbit/second and latency of 0.5 second each way (1 second round-trip). The primary OCU and auxiliary OCUs are interconnected via a standard gigabit network (Figure 4.3). All OCUs are able to run the user interface. In order to provide the operator with an unrestricted view of the interface, each computer has dual monitors, with OCU 1 and OCU 2 containing a second row of displays above the first. These secondary displays mirror the screens of the opposite OCU, giving each user awareness of what their co-operator is doing (Figure 4.4).

Figure 4.4: The layout and functions of each display in the control station and which Operator Control Unit (OCU) they connect to.

4.2.3 Robot Hands

Since manipulation is an essential requirement in the DRC tasks, we performed an experimental evaluation of the iRobot, Sandia, and Robotiq hands shown in Figure 4.5 that could be
CHAPTER 4. HUMANOID ROBOTS

interfaced with ATLAS. Table 4.2 provides a summary of design specifications of each robotic hand available for use during the competition.

Figure 4.5: Left to right: Robotiq, Sandia and iRobot hands.

Table 4.2: Robot hand specifications

<table>
<thead>
<tr>
<th></th>
<th>iRobot</th>
<th>Sandia</th>
<th>Robotiq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fingers #</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>DoF</td>
<td>5</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>Weight(kg)</td>
<td>1.53</td>
<td>2.95</td>
<td>2.3</td>
</tr>
<tr>
<td>Base size LxWxH(cm)</td>
<td>13x11.5x8.25</td>
<td>11.5x11.5x18</td>
<td>12.75x12.75x10.25</td>
</tr>
<tr>
<td>Drive type</td>
<td>Worm gear</td>
<td>Gears</td>
<td>Worm gear</td>
</tr>
<tr>
<td>Trans. type</td>
<td>Spectra braid line</td>
<td>Steel cable</td>
<td>Mechanical linkage</td>
</tr>
<tr>
<td>Max Current 24V</td>
<td>5 A</td>
<td>2.5 A</td>
<td>1.5 A</td>
</tr>
<tr>
<td>Mechanical Safety</td>
<td>Magnetic finger coupling</td>
<td>Mechanical fuse</td>
<td>Mechanical fuse</td>
</tr>
</tbody>
</table>

For the DRC trials, the WPI-CMU team opted to use one Robotiq end-effector. The end-effector control was kept separate from most other systems for the competition. The Robotiq hand provides several modes of operation, such as basic mode, wide mode, pinch mode and scissor mode. We can also control each finger independently. At the same time, we can implement position, speed, or force control on the hand. We designed a ros-visualization (rviz) panel as a GUI to control the hand. The Robotiq hand allows for finger tip modifications, and distal and proximal phalange finger pad alterations. Based on this a few task specific finger modifications were designed to improve performance. Micro spikes were added for finer grip strength on the debris task. A proximal distal extension for drill operation, and a hose attachment finger tip. In addition, a number of interchangeable passive end-effectors were designed specifically for each task. For example, we used hook-pipe hands on the ladder task, 6 in (15.25cm) pipe hand in the vehicle task for steering and valve task, and 18 in (45.75cm) pipe hand in the door task to turn the door handle.
CHAPTER 4. HUMANOID ROBOTS

4.3 Software Architecture

The overall goal in designing the software architecture is to enable human-in-the-loop-control of a complex robotic system over a limited bandwidth link. In order to meet this goal, tasks that require high bandwidth and/or low latency communications are run on the field computer. The field computer is connected to the ATLAS robot via 10G Ethernet. User interfaces and tasks that work with low bandwidth and high latency are run on the Operator Control Units (OCUs).

The software on the field computer is divided into several subsystems (Figure 4.6). field_state is the primary interface to the robot hardware and where all of the critical control loops that run synchronously with the robot 3ms controller cycle time are loaded. These control loops include full body manipulation, stepping and velocity control needed to accomplish the individual DRC tasks. The field_command process is responsible for compressing and managing data sent to the operator control unit over the DARPA controlled network (0.1-1 Mbit/second, 0.1-0.5 second latency).

The OCUs provide interfaces to the human operators controlling the robot. These include our custom user interface (WGUI), and modified versions of MoveIt! and RVIZ. In addition, one process is responsible for managing the low bandwidth communication link with the field computer which decompresses incoming data and compresses outgoing data. A separate process (OCU to ROS) acts as a bridge between our custom protocol and standard ROS messages as used by MoveIt! and RVIZ.

One of the keys to our success was providing good situational awareness to the human operator and allowing the human operator to control the robot at several different levels of abstraction. Providing good situational awareness comes down to compressing data well and making effective use of the limited bandwidth pipe between the field computer and OCU. To make good use of the pipe we sent smaller and more compressed data during the low bandwidth (0.1Mbit/second) times of the run. Sending a large data set (1Mbyte) such as a high resolution image over the 0.1Mbit/second pipe would take over a minute wasting valuable time during the tasks.

Various levels of human operator control were implemented allowing the operator flexibility to solve problems, even ones that were not anticipated in the testing and practice prior to the challenge. The lowest level of control allowed the operator individually control joint positions and forces. The next level up allowed the operator to control the arms and legs in a Cartesian space through our inverse kinematics solver. Finally, a full-body manipulation controller enabling the most generic movements but required significant effort to ensure balance constraints were not broken resulting in a fall. Each control mode had its own associated levels of automation, GUIs, and
CHAPTER 4. HUMANOID ROBOTS

Figure 4.6: A flow chart showing the software architecture and communication between different subsystems.

scripting. These allowed the operator to choose an appropriate level of abstraction and automation when attempting to complete a task.

4.4 DRC Driving a Utility Vehicle Task

The vehicle tasks consist of a small utility vehicle the robot must drive through a slalom course of highway safety barriers. Our approach focused on allowing the robot to control its speed, while implementing operator-assisted steering. It is evident that the driving a utility vehicle task presents itself as a novel application for humanoid robots (Figure 4.7), and it is essential to implement reliable supervised autonomy modules on the robot for constrained motion planning and control as well as sufficiently fast perception algorithms. We present our initial solution to develop supervised control of humanoid robots in driving tasks. Our main contribution is the development of the human-in-the-loop software control architecture based on a whole-system design approach.

In the overall disaster response mission scenario, driving a utility vehicle task comes first. The robot will need to reach the disaster site using a vehicle designed for human-use. Figure 4.8 depicts the course layout for the vehicle task. The vehicle task has two subtasks: (1) the robot begins in the vehicle, drives through the course and crosses the finish line, (2) the robot egresses from the vehicle and travels dismounted out of the end zone. The course has dimensions of 75 m (length)×12 m (width) and includes six barriers placed at 45°.
CHAPTER 4. HUMANOID ROBOTS

Figure 4.7: The ATLAS robot at the start of the course at the DRC Trials.

Figure 4.8: A rendering showing the slalom course the robot needs to navigate with the utility vehicle as part of the DARPA Robotics Competition Trials. The robot and vehicle start at the top of the course and must reach the light gray finish zone to receive a point.
CHAPTER 4. HUMANOID ROBOTS

4.5  Approach

Our approach to the driving task is summarized in Figure 4.9. The stereo cameras and LIDAR information is used to generate two independent estimates of the velocity of the vehicle. The operator can choose between which velocity estimate, stereo-camera or LIDAR based, is used by the robot depending on the environment. In visually dense environments, the stereo cameras with a modified version of the libviso2 package described in Section 4.5.2 provide very good data. In relatively flat/poor lighting or environments with little visual diversity, the LIDAR-based estimate provides better data. The desired velocity is provided by the operator and is passed to a PI controller that actuates the throttle using the ankle joint of the robot. Once set, the robot drives the vehicle at a steady and slow pace autonomously.

Most of the complexity in navigating difficult terrain is associated with the steering which human operators are very good at handling, even in latency degraded conditions. Thus, the steering of the vehicle is handled by the operator in real-time allowing the robot to adapt to difficult or changing conditions easily. Visual feedback is provided to the operator from the same stereo cameras. The images are compressed to preserve bandwidth and displayed to the operator. The operator then commands a steering angle, and the robot generates the appropriate arm joint angles to execute the desired steering angle.

In [36], several architectures of user-robot interaction with respect to control are discussed: traded control, indirect control, coordinated control, collaborative control, blended control, and virtual constraint control. Our approach follows the coordinated control architecture since the robot autonomously handles the throttle while the operator handles the steering. The structure also incorporates principles of supervisory control since the operator can command the robot to slow down or speed up depending on the situation. In essence, this implementation is very similar to the function of cruise control in most modern automobiles, except it is implemented with a humanoid robot.

4.5.1  Steering

Our initial approach to tackle the steering challenge was to utilize the inverse kinematics of the robot. We would generate joint angles corresponding to the robot arm moving the steering wheel in a circle. While in principle this approach should have been sufficient, it was problematic since each point was generated without considering the points before and after it. Since ATLAS used a pipe-based manipulator inside the steering wheel, keeping the pipe close to perpendicular
CHAPTER 4. HUMANOID ROBOTS

Figure 4.9: The robot uses its stereo cameras and LIDAR to estimate the velocity of the vehicle. Using this information and the desired velocity commanded by the user, the robot actuates the throttle through a PI control loop. Images from the camera are compressed and sent over the bandwidth limited and latency-induced link back to the operator. The operator controls the steering throughout the driving task by commanding the robot to specific steering angles.

to the plane of the steering wheel was critical. This reduces the forces on the steering wheel, and allows the robot to utilize stiffer gains improving controller tracking without damaging the robot or vehicle. Due to the kinematic constraints of the robot and vehicle, the generated circle was not sufficiently smooth or accurate.

Several other more complicated options were considered including modeling the steering wheel and developing an optimization-based controller that would trade errors in tracking a circle with keeping the manipulator in the proper orientation in the steering wheel. Instead of moving to a more complicated controller, we decided to attempt to generate a circular motion by physically grabbing and moving the robot manipulator inside the steering wheel while the robot was securely strapped into the vehicle. This generated a set of points corresponding to various steering angles that moved more smoothly from point to point. Unfortunately there were a couple issues with this
second attempt. The set of joint values is only valid for a single robot position, and the elbow joints of the manipulator reached torque limits when the robot played back the trajectory.

The position of the robot varied somewhat each time it was loaded into the vehicle. To counter this, we utilized a manual operator-run procedure that adjusts the generated joint angles using inverse kinematics based on the relative position of the steering wheel with respect to the robot. The use of inverse kinematics in this case resulted in smooth trajectories since the transitions between the original points were smooth and the needed IK corrections were small. The issue with the weak elbow joints was resolved by manually adjusting the recorded circular path by forcing the robot to utilize its stronger shoulder joints to avoid the torque limits of the elbow joints.

Feedback is provided to the operator based on color images from the sensor head of the robot. Through some simple simulations in Gazebo, we verified that despite the latency restrictions it was possible to utilize operator controlled steering with delays up to 2 seconds if the robot could maintain a slow, steady speed. The operator provided inputs are translated to the correct joint angles using a lookup table and the IK corrections described above moved the vehicle steering wheel. The main difficulty was providing sufficient operator situational awareness. The narrow field of view provided by the camera made keeping track of objects to the side somewhat difficult. The latency changed every minute from a low of 100 milliseconds to a high of 1 second, increasing operator cognitive load since mitigating the latency effects requires operator compensation.

### 4.5.2 Velocity Control

One of the key challenges in controlling the vehicle in the task is effectively managing the velocity. In order to control velocity, a good measurement or estimate of the velocity of the vehicle through time is needed. Humans utilize many senses to accomplish this task when driving, not limited to looking at the speedometer on the vehicle or observing the obstacles in the environment passing them by. Through testing in simulation, we determined the slalom course is most easily driven at speeds less than 2 m/s. At these low speeds the speedometer is not useful as it typically reads zero or is very inaccurate. Using teleoperation, a human could directly control the throttle based on the images coming back from the robot. While possible, this turns out to be fairly challenging and requires significant mental effort for a human. Controlling velocity with latency makes the task even more difficult. It is also challenging to maintain a consistent speed. One of the overarching goals for the driving task was to enable a single human operator to control the robot through a supervised autonomy method. Through simulation, we determined that despite a human
operator’s ability to steer is not severely restricted by the latency, the ability to control speed is affected significantly. We decided to implement a control architecture where the robot autonomously senses and controls its speed, allowing the operator to focus solely on steering.

As mentioned above, the vehicle speedometer cannot reliably measure small velocities so we had to use the sensors integrated on the robot to estimate the speed of the vehicle. From the MultiSense SL head on ATLAS, both a LIDAR and a stereo camera pair are available to view the external environment. We created an algorithm to measure velocity using the LIDAR data from the Hokuyo UTM-30LX-EW LRF, and modified an existing algorithm to measure velocity separately using the stereo cameras.

![Diagram](image)

Figure 4.10: A simple scenario demonstrating the LIDAR-based velocity measurement showing the vehicle and one LIDAR scan point through time as the vehicle moves. The estimated velocity is calculated for every scan point, after which they are ranked by a confidence metric calculated by the intensity of the return, and top results averaged to estimate the vehicle velocity.
CHAPTER 4. HUMANOID ROBOTS

While more complicated algorithms for matching LIDAR features exist [47], we chose to implement a simpler algorithm. The Hokuyo measures the distance to each point in a 270° arc using 1080 samples. Each sample contains both a distance, accurate to ±50mm at 30m, and an intensity value. The intensity is an 18 bit value representing the strength of the reflected laser signal. The ROS driver presents these values to the robot in two arrays of double values: one array for the distance values and another for the intensities. Each array index denotes the angular scan position from which those values originated. Index zero is the far right scan point, index 540 is directly in front of the LIDAR unit, and index 1079 is to the far left.

Internally, the software maintains a FIFO queue of the last $m$ scans. Both the distance values and the intensities are retained. After each complete scan, the robot evaluates every point in its scan history by looking at the distance changes at a single angle over time. Points outside of a pre-defined forward arc are discarded, as those distances are beyond the accurate range of the Hokuyo device. Finally, if there are not enough reliable data points in the current angular scan line, the line is discarded.

If enough reliable time-variant distances are available along this angular scan line, the processing continues. First, each distance measurement at this angle is adjusted for the angle of the scan line, $\Phi$. We are only interested in the forward component of any velocity. The total distance change for this scan line is computed, and the velocity is calculated from this total. A mean intensity is calculated for the set of distances in the scan line. The mean intensity is used as a confidence indicator of this velocity measurement. Each angular scan line can produce a single velocity and confidence vote. The robot puts each vote into a sorted list by descending confidence. Finally, it averages the top $q$ votes to produce its final velocity evaluation (Figure 4.10).

A second measurement of velocity was done using the information from the stereo cameras. Measuring the velocity using the stereo cameras was done with a modified version of the libviso2 stereo odometry package[40]. The key modification was to enable filtering out of features that were part of the vehicle or robot. These features tend to have zero velocity with respect to the robot. If the external scene has a lot of features, the zero velocity features from the vehicle and robot do not affect the algorithm. If the scene does not have as many features as the vehicle/robot, then the RANSAC algorithm is dominated by the zero velocity features even if the vehicle is moving. This is not typically a problem for fixed cameras mounted to a vehicle as the cameras will be positioned so as to not see the vehicle. With the robot inside the vehicle, it is not always possible to orient the cameras so that none of the vehicle is in view.

With two independent methods to measure the speed, we developed a simple controller to
CHAPTER 4. HUMANOID ROBOTS

command the robot to actuate the throttle. This was implemented with a PI controller controlling the ankle joint of the robot. The integral gain was relatively higher than most situations demand to help minimize offset errors from the positioning of the robot in the vehicle. The input values (desired/actual velocities), the integral term, and the output were all clamped to experimentally determined values. In addition, a feedforward path immediately set the output to the minimum value when a desired velocity of zero was commanded. This was done so that the robot stopped the vehicle immediately rather than waiting for the response time of the PI loop.

4.5.3 Simulation

Simulation was key for success in driving. We used the Gazebo simulator to simulate the robot, a drivable vehicle, and the DRC Trials course. Gazebo is an open-source 3D robotics simulator that can simulate robots in an outdoor environment [52] and provides controllable models of robots, vehicles, and worlds. The interface to the robot in Gazebo was set up to mimic the real hardware allowing the same software that is run on the simulator to be run on the real hardware. Joint controls can be sent to the simulated robot and sensor data comes back from the simulator, including LIDAR, camera images, and force sensors. Rather than developing all of our code on our ATLAS robot we were able to develop on the Gazebo simulator in parallel. The real hardware has a limited number of hours that are available each day and can be broken if it is not used carefully. The testing in Gazebo offloaded the high risk development testing without placing unnecessary burdens on the real robot hardware.

The simulator provided significant benefits allowing testing and extended operator training to be conducted in a simulated environment. The logistics of preparing for driving tests are complex and time consuming. A test that requires one hour can turn into four hours when all the logistics are added on. Using the Gazebo simulator we were able to test the bandwidth and latency effects on operator control and cognitive load easily. Only two full tests with robot and vehicle hardware were conducted before competing at the DRC Trials.

4.6 Results

Our robot successfully drove the course at the 2013 DRC Trials reaching the finish zone in 6 minutes 21 seconds and earning 1 point. This result was the fastest posted by any team at the
competition and earned Team WRECS a "Best In Task" award for the driving task. In addition, it was the only ATLAS robot to successfully drive.

Figure 4.11 shows the visual feedback provided back to the operator to help set the desired steering angle at the competition. The image is fuzzy and degraded due to the significant compression applied to keep bandwidth use low, but is still very usable for avoiding obstacles. The large highway barriers are easily visible and some sense of distance can be extrapolated for the context in the image.

Figure 4.11: The view provided back to the operator during the 2013 DRC Trails. This image is from the competition run on the actual course. Despite the high compression and noticeable fuzziness of the image, the operator is able to navigate around the traffic barriers.

Figure 4.12 shows the estimate of the vehicle velocity from the vision-based system. Because the course at the competition was visually very rich, we relied solely on the vision-based system instead of the LIDAR system because of higher noise in the LIDAR-based estimate. The large drop in velocity is the point in the run where the robot’s foot slipped off the throttle. Due to the supervisory control architecture we implemented, the operator was able to intervene, pull the foot out from under the pedal, and raise it back into position above the throttle. It is interesting to note that the periodic undershoot and overshoot of the estimated velocity from the desired velocity is due to the initiation of each turn around the slalom course and then subsequent straightening out. We speculate that this is due to changes in the friction dynamics of the tires around the turns and the very slow velocity we were attempting to drive.

While the current implementation proved successful at the 2013 DRC Trials, there was
Figure 4.12: The vision-based velocity estimate (red) and desired velocity (blue) during the 2013 DRC Trials. The data is from the competition run on the actual course. The drop in velocity in the middle is attributed to the robot’s foot slipping off the throttle. The operator was able to intervene and remedy the situation.

Given the experiences at the DRC, three improvements would make the system more robust and effective to real-world scenarios: improving the velocity control system, giving the operator more feedback to improve the steering system, and generalizing the approach so it can support more difficult scenarios and different vehicles. The LIDAR-based speedometer works well, but additional work is needed to reduce the noise and eliminate the need for a calibration factor. This will involve more detailed analysis of the raw data to better understand the scan to scan differences that generate...
the velocity output. In addition, if the vehicle is on large, flat surface with few markings, neither the LIDAR speedometer or visual odometry works well. Humans drive with three other sensors that help them know the speed without looking at the speedometer: noise, foot angle, and motion. The most obvious one is noise - as the vehicle goes faster there is more noise. Another one is foot angle - humans have a good intuition for what angle and force combination generate a given speed. A final sense is motion - at higher speeds a given surface produces more bumps or vibration.

The velocity PI servo worked quite well for controlling the vehicle on the DRC Trials course, but there were two obvious errors that could be reduced. In particular, starting a turn reduced the vehicle speed due to the added friction from turning. Some time elapsed before the controller was able to recover to the desired speed. When the wheels were straightened out again the speed would overshoot for some period of time. A predictive throttle that makes adjustments based on steering angle (or future steering angle) could reduce these effects and further assist the operator. In addition, modeling, then showing the operator the arc that the vehicle will drive with a given steering angle could help provide better feedback as to the state of the vehicle. This is critical since simple mistakes can be costly due to the latency in the communications system.

Finally, our approach at the DRC Trials focused on driving an apriori known vehicle and course. This is not a reasonable assumption in a typical disaster response scenario. The approach needs to be generalized so that the robot can drive a wide variety of vehicles, and more difficult terrain. Our approach relied on being able to align the robot very precisely within the vehicle, but this would not be possible in a generalized scenario. A method to adaptively model the vehicle interfaces such as the steering wheel, throttle, ignition, and shifter is needed so the robot can utilize vehicles with no apriori information.

4.7 Points

Team WPI-CMU scored a total of 11 out of possible 32 points. We acquired all 4 points in the valve task, 2 points on the terrain, ladder and hose tasks, and one point in the vehicle task. These 11 points gave us a ranking of 7th out of 16 teams. As a result, Team WPI-CMU advanced to the DRC Finals that took place in June 2015. The breakdown of points by task is presented in Table 4.3.

We described the details of our system architecture for human-in-the-control of humanoid robots for disaster response tasks, utilized varying levels of teleoperation and autonomy to complete the tasks. Based on the strengths of, and despite the deficiencies in, our approach, the WPI-CMU
### Table 4.3: WPI-CMU DRC Team Points Breakdown at the DRC Trials

<table>
<thead>
<tr>
<th>Task</th>
<th>Points</th>
<th>Interventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Terrain</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Ladder</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Debris</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Door</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Wall</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Valve</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Hose</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>11</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>

The team became a DRC finalist by ranking 7th out of 16 teams. The team also received the Best-in-Task: Vehicle Award by completing the first subtask in 6 minutes. The WPI-CMU DRC Team continues to improve the performance of the framework by incorporating new supervised autonomy and user interface tools toward meeting the requirements set forth by the DRC Finals.

#### 4.8 Valkyrie

NASA’s Johnson Space Center (JSC) participated in the DRC Trials along with our team. As a Track A participant, JSC had to develop their own robot hardware and software resulting in a humanoid robot called Valkyrie, based on legacy Robonaut hardware [78]. After the DRC Finals, NASA sponsored a new Centennial Challenge called the Space Robotics Challenge (SRC) aiming to leverage the successes of the DRC developments on humanoid robots for space exploration. There are several challenges that humanoid robots can help address in future space exploration missions. Space exploration robotics as a field has been a vibrant and active research area ever since the beginning of the human spaceflight program. A variety of satellites, probes, landers, and rovers have explored various celestial bodies within our solar system [74]. Future robotic exploration missions though will require advances in specific capabilities to enable human exploration to reach farther [64].

It will be impossible to send the equipment and supplies required for long-term crewed exploration missions to Mars in just a few launch opportunities. In order to successfully support crewed exploration on Mars, multiple launches spread over a period of years will be required to predeploy the necessary equipment, habitats, and supplies. These predeployed assets will need to be maintained over the years to ensure their availability and functionality for the crew when they
arrive to complete their scientific objectives. Because the predeployed assets are designed to be used by astronauts, a robot in humanoid form-factor is most likely to be able to complete generic maintenance and support tasks that arise during the course of the mission.

In order to better understand the challenges that need to be solved to enable the next generation of robotic space missions using humanoid robots, NASA organized the SRC. It is a NASA Centennial Challenge in which teams competed in a virtual challenge first, demonstrating competency with mobility, manipulation, and perception tasks on a virtual Valkyrie robot. Example tasks that the SRC focuses on include alignment of a communications dish, repair of a solar array, and finding/repairing an air leak in a habitat. These are representative tasks that demonstrate integrated mobility, manipulation and perception frameworks needed to effectively support habitat building, maintenance, and science support operations in unknown and unexplored environments. Leveraging the Centennial Challenges program quickly exposes many different approaches that teams take to complete the tasks. Many of the approaches and algorithms need to be significantly modified for testing and operation on the real robot.

NASA selected our team as one of two in the United States to receive a Valkyrie robot for validation of the SRC tasks on real robot hardware, and to develop the necessary methods and algorithms to enable humanoid robots to be used in future exploration missions. We describe in detail the capabilities of the robot, lessons learned, and best practices for operating and testing with the robot in addition to highlighting important specifics of the robot hardware and software. The intent of providing this information is to help university research groups without access to a full-size humanoid robot, citizen scientists, and other potential entities interested in working with humanoid robots better understand the limitations and considerations needed to successfully transition from simulation to real hardware. This includes participants and future users of the Valkyrie robot or similar humanoid robots targeting space exploration. Considering the practical transfer to real hardware early in the development process helps keep future development for humanoid robots in space exploration environments stay grounded with respect to the realistic capabilities and challenges that will be present in completing relevant tasks in space.

Figure 4.13 shows the shared control framework blocks that are focused on with the Valkyrie robot. Contributions are split into two different areas: the robot control blocks and context/cloud engines. The robot control blocks are developed in this application due to the need for task validation of the SRC. NASA’s desire when awarding the Valkyrie robot to the host institutions is to ensure that the robot is physically capable of accomplishing the tasks that teams are completing in simulation. The task validation is a key part that ensures the competition and simulations are well
CHAPTER 4. HUMANOID ROBOTS

Figure 4.13: Development on the NASA Valkyrie humanoid robot contributes to two separate areas of the shared control framework. Task validation for the SRC contributes to robot control blocks of the framework demonstrating that a humanoid robot is capable of completing tasks in environments designed for astronauts. The generation of the first dataset with a full-size humanoid robot will enable the research community without access to a physical robot to contribute to the context and cloud engines.

grounded with respect to the capabilities of the real robot.

Contributions to the context and cloud engines will be a gradual process that will not show their full benefits for several years. One hindrance to the context and cloud engines is that simulations of humanoid robots including ATLAS and Valkyrie loosely match the performance of the robot in the best case. Over the last few years, simulators such as the open-source robot simulator Gazebo have significantly improved, but are still not to the level needed for good contextual awareness and cloud-based interaction with models to be successfully implemented. Towards enabling this goal, we developed the first dataset from a full-size humanoid robot. The intent of the dataset is to provide the opportunity for researchers without a full-size humanoid robot to develop robust capabilities in simulation and have access to ground truth data from a real robot. The experiments we conduct on the real robot are also conducted in simulation to provide a direct comparison of simulated and real-world performance with the hope that these results will lead to better simulator behavior in the
Figure 4.14: A hardware diagram showing the location of all the sensors and joint actuators on the Valkyrie robot. Valkyrie has a total of 32 degrees of freedom (DOF) with 6 DOF legs, 7 DOF arms, 3 DOF back, and 3 DOF neck, not counting the DOF on the fingers.

4.9 Hardware

Valkyrie, shown in Figure 4.14, is a 32 degree of freedom (DOF) humanoid robot originally designed to compete in the DRC Trials in December 2013 [79]. The robot consists of 5 major mechanical sub-assemblies: two arms, two legs, and a torso. The arms consist of a total of 7 DOF arranged in a 3 DOF shoulder, 1 DOF elbow, and 3 DOF wrist. The first 5 DOF of each arm are implemented through series-elastic actuators (SEA) [72] enabling the implementation of torque and
CHAPTER 4. HUMANOID ROBOTS

impedance control. The last 2 wrist DOF are implemented with a lead-screw on a linear guide assembly. The 7 DOF redundant arms allow for the optimization of trajectories and provides much needed flexibility in accomplishing tasks: for example, selecting between an elbow up and elbow down arm configuration.

Each arm is equipped with a 6 DOF hand consisting of three 1 DOF fingers and a 3 DOF thumb. The thumb has independently controlled distal and proximal joint control in addition to roll control of the entire thumb assembly, allowing Valkyrie to touch each one of her other 3 fingers to her thumb. An array of pressure sensitive sensors encased in a rubber compound [93] cover the fingers and palms of each hand giving the robot a tactile sense of objects that are grasped. The fingers are controlled by cables connected to Harmonic Drive gearboxes driven by small frameless motors embedded in the forearm assembly. There is no direct force sensing available on these actuators, so the low-level controller for the fingers is position based. Through a combination of using the barometers in the fingers and palm, force controlled grasps can be realized.

Each leg consists of a total of 6 DOF arranged in a 3 DOF hip, 1 DOF knee, and 2 DOF ankle. The first 4 DOF of each leg are also implemented with SEAs, while the last 2 DOF are implemented with lead-screw actuators similar to the wrist. Precise torque control is implemented in the SEAs by measuring spring deflection with pre- and post-spring encoders. Torque control of the ankle roll and pitch assembly is implemented with two load cells on the mechanical linkages of the last two joints. In addition to the joint level torque sensing, ATI Mini58 6-axis force/torque (F/T) sensors mounted in the ankles provide sensor data to calculate wrenches representing ground reaction forces and center of pressure (CoP). Finally, embedded in each foot is an array of Tekscan force sensors that after a firmware update can be used in future work for more accurate ground sensing, especially over uneven terrain.

Valkyrie’s arms and leg flanges are connected to flanges on the torso through a Marman band clamp. The clamps have a pair of flats cut into them ensuring repeatable alignment, and due to the wedging design of the clamp, a modest tension on the clamp locks the two flanges together. The repeatability and simplicity of the design allow for quick disassembly and assembly without requiring joint recalibration. A single cable carries communications and power to each appendage further simplifying disassembly and assembly. There are 6 DOF in the torso, split between a 3 DOF back that enables the torso to rotate and lean, and a 3 DOF neck. Two IMUs are located in the torso, one near the pelvis and other near the left shoulder. Both IMUs can be used independently or in coordination to estimate the pose of the torso. Finally, two cameras in a stereo pair arrangement are mounted in the stomach which can be used as hazard avoidance cameras or to provide a low-angle
CHAPTER 4. HUMANOID ROBOTS

perspective for manipulation tasks.

Valkyrie’s head consists of a Carnegie Robotics Multisense SL sensor unit combining a rotating Hokuyo LIDAR and a stereo camera pair. An integrated FGPA in the Multisense SL processes the stereo camera data and provides registered point clouds from the camera data and LIDAR data. Integrated fully-dimmable LED lights can be used to illuminate the scene. In addition, the LIDAR rotation speed can be varied, trading off denser point clouds for longer scan times.

4.10 Software

IHMC has integrated Valkyrie’s model into the Simulation Construction Set (SCS), the simulator built specifically for testing humanoid walking performance and used for the DRC by Team IHMC. In addition, IHMC integrated the walking controller used on the ATLAS robot during the DRC on the Valkyrie robot [50, 53]. The SCS simulator, along with the Valkyrie models, is open source. Development on Gazebo support is a work in progress, lagging behind the performance of SCS, but allows for manipulation-based research to be conducted in simulation. SCS is not structured to support simulation of dextrous manipulators, and therefore is limited to mobility simulation. In addition, Gazebo allows the leveraging of previously available code and approaches from the DRC. For those reasons, the SRC uses the Gazebo simulator.

Valkyrie has two on-board computers: Link and Zelda. They are single board computers (SBC) with Intel i7-3615QE at 2.3 Ghz paired with 16GB of DDR3 1600 and a 240GB SSD. All of that is on a Congatec BS77 Type2 COM Express module, and the EFK XV1 carrier board provides peripherals. Both computers run Ubuntu 14.04 Server edition, and Link, which runs the onboard low-level controllers, additionally has realtime kernel patches to ensure controller performance. In addition, it interfaces with the Synapese drivers implementing the custom LVDS Robonet protocol used to communicate with the motor drivers and sensor boards throughout the robot. In general, Link only runs the low-level controllers, and Zelda runs any ROS nodes that are needed on the robot, including handling the Multisense SL sensor data.

Valkyrie has a ROS master running on the Zelda computer which allows for simplified development on the robot utilizing the standard ROS tools, messages, and services. Figure 4.15 for example shows the TF frames of major joints on the robot. Controllers, such as the custom forearm controller, use the ROS control framework and can be loaded like any other controller in ROS. Coupled with the IHMC walking controller, users can manually or through other nodes send the robot footstep messages to make the robot walk, arm pose messages to control the arms either
Figure 4.15: A view from Rviz showing the TF frames of the joint locations of the 32 DOF on the robot.

in joint space or Cartesian space, and full body messages for control of all the joints. The IHMC walking controller provides relevant information for balancing such as the ground contact forces as wrenches in the foot frame. Sensor data is also provided through the standard ROS messages. Most recently, the barometric pressure sensors embedded in the palms and fingers has been exposed through the API and will enable the implementation of more robust and intelligent grasping algorithms. In addition, effort and position information is provided for each SEA joint allowing for very precise forces to be exerted for prehensile and non-prehensile manipulation. The Multisense SL data is passed through without modification from the embedded computers and is provided in appropriate ROS topics by the standard Multisense driver.
4.11 Experimental Results and Capabilities

4.11.1 Mobility

In IHMC’s walking controller, there are two main parameters exposed through the API that affect walking: transfer time, the amount of time spent in double support, and swing time, the amount of time spent in single support. However, several other parameters indirectly affect walking as well, including the height of the pelvis from the ground plane, the distance between the centers of each foot, and the placement of the footsteps.

In order to find reliable transfer times and swing times to achieve robust walking, the pelvis height, feet separation, and footstep placement are held constant. The transfer time and swing time are increased by increments of 0.5s starting at 1.0s. For pelvis height, a value of 1.05 m from the ground is used, which is calculated by using the center of pressure (CoP) frames on each foot. This height was determined based on hands-on experience with the hardware, and finding that
CHAPTER 4. HUMANOID ROBOTS

Figure 4.17: Left Hip Torques during Walking 2.0s Transfer and 2.0s Swing Time. Successful Attempt. First 15s show a side step with a 10s pause, followed by 7 successful 0.3 m steps. The higher pelvis height versus the default 0.974 m allows for smoother walking and less falls. The separation of the feet was kept at 0.277 m, which is the default value in the SCS simulator, and keeps the feet and hips inline with each other. Finally, footsteps were kept at 0.3 m long, allowing for a full heel-to-toe footsteps. Each iteration attempts to take seven full steps forward with an initial side step to bring the feet the desired distance apart along with a 10 second pause before walking and a final step to bring the feet back inline.

Figures 4.16 and 4.17 respectively show the left hip’s yaw, pitch, and roll torques of one failed attempt, where the robot falls, and one successful attempt, where the robot completes a series of steps. Figure 4.16 shows the initial attempt at walking with 1.0s transfer and swing time, however, the robot failed on the first step. Figure 4.17 shows one of the successful attempts with the lowest combined transfer and swing times, with each time being 2.0s. A transfer time of 1.5s and swing time of 2.5s was also successful and showed similar results. Between both experiments, it can be seen that hip’s yaw and roll have significant qualitative and quantitative differences. When the robot
CHAPTER 4. HUMANOID ROBOTS

fell, the difference between the minimum and maximum yaw is 121 Nm where in the successful run the difference is only 48 Nm. Similarly, for the hip’s roll, in the successful run it can be seen that the torques create parabolic curves during walking where in the failed attempt, the controller almost immediately becomes unstable and is unable to handle the forces at such a short time interval. This is the expected results given the controller’s inability to track the desired hip yaw and roll torques well.

Figure 4.18 shows the constrained motion planning approach from [107] to use both arms to pickup a box with a virtual reality tag. The robot executes an aggressive squat and Figure 4.19 shows the measured torques on the relevant joints that complete the squat. The hip and knee pitch joints have 350 Nm design torque limit, while the ankle pitch has a 205 Nm limit. Despite the aggressive squat in the figure above, the robot actuators still have headroom. The motion is robust, having been tested successfully a couple dozen times over several testing periods.

4.12 Best Practices

It is important for repeatability of test results to keep robot initialization consistent. One example is the main pelvis IMU, which gets initialized as soon as the robot is turned on. This makes it important to maintain the robot stationary when switching the power supply on, in order for the IMU to calculate its bias offsets accurately. Based on experiments, it is recommended that torque offsets are re-calibrated every time the robot motor power is switched on, this results in more stable walking, and less risk of triggering an emergency stop. The torque offsets are found by placing the robot in a calibration pose with known joint torques and calculating the offsets in the joint torque sensors.

It is well known that additional considerations must be made when running code on a physical robot as opposed to simulation. For Valkyrie and SCS, there are two key differences that must be considered during development.

First, there are torque limitations on the physical hardware that are not enforced in simulation. Each of Valkyrie’s joints has a torque limit and if at any point these limits are surpassed, the controller will engage the emergency stop to prevent any damage to the motors. However, in simulation there are no such limitations allowing the simulated version of Valkyrie to move faster than the physical version. This is particularly evident in walking. As discussed earlier, a transfer and swing time of 1.0 s with a step size of 0.3 m is too fast for the physical hardware to success-
fully accomplish. However, in simulation, Valkyrie is able to handle these parameters and can walk without failing.

Secondly, there is a difference in the understanding of the robot’s position relative to the world frame. In simulation, the world transform frame is located in a fixed location on the ground

Figure 4.18: The sequence of images shows Valkyrie picking up a box with a whole body control algorithm.
Figure 4.19: Leg joint torques during the box pickup. The joint torques not shown do not change significantly during the pickup operation. The data shown is from the left leg, but the right leg data is very similar both quantitatively and qualitatively. Torque limits for the hip and knee pitch joints are 350 Nm and 205 Nm for the ankle pitch.

and Valkyrie is initially located directly on top of the frame with the pelvis sharing the same z-axis and feet on the y-axis. However, on the real robot the world transform is created when the robot first powers on, and undergoes movement during the start up and stand process. This causes the world and robot relative positions change significantly. In addition, there is a drift between the world frame and the robot, meaning that over time the distance between the world and robot will change in a way that is not accounted for purely by the robot’s movement. To counter this effect, when sending any command from the world frame, the transform from world frame to the robot frame must be found and applied to any commanded position before being sent to the robot.

Through iterative sessions testing the different walking parameters on the robot, Valkyrie’s walking controller displays significantly different behavior depending on the type of surface she is walking on. Initially during testing, the robot walked on a polished concrete surface which would
result in slipping during toe-off when the walking parameters were pushed beyond conservative values. This behavior was not very surprising given the similar experiences with the ATLAS robot in the DRC. On the other end of the spectrum, the robot’s walking controller is unpredictable walking on soft or roughly textured surfaces, likely due to uncertainty in the robot ability to detect the ground reaction forces in the walking controller. The robot is unsure of where the ground contact is exactly and if that uncertainty is above a certain threshold, the controller will E-stop to prevent damage to the robot.

Commercial low-pile carpet on top of an 1/8” thick plywood results in the most consistent and predictable walking behavior for the robot and is recommended for any future studies that focus on altering and optimizing walking parameters. The commercial carpet provides enough stiffness for good controller certainty in footsteps, but provides the necessary friction to implement aggressive walking behaviors without the support foot slipping considerably. It is likely that softer, residential grade carpeting will not provide enough stiffness for the walking controller and will result in similar behavior to other soft surfaces tested such as gym mats and cushioned rubber flooring.

4.13 Full-size Humanoid Robot Dataset

In the future, we envision that humanoid robots will find numerous practical applications as personal assistants in the home, nursing assistants in hospitals, first-responders in disaster relief, and astronauts in future space missions. The DRC demonstrated that state-of-the-art humanoid robots are slower than humans by an order of magnitude in performing tasks such as turning valves, using hand drills and flipping electric switches. Furthermore, the DRC robots relied heavily on prescribed motions and hence lacked autonomy to carry out the simulated tasks relevant to disaster response. The reliability in completing tasks is prohibitively low to make robots practical even for an overly simplified set of tasks.

As a result, there is a need to develop new perception, planning, locomotion and manipulation methods to dramatically advance the capabilities of humanoid robots. However, accessibility to a full-size bipedal humanoid robot, due to size, complexity and costs associated with acquisition, infrastructure and maintenance, remains to be the limiting factor for the research community to systematically validate algorithms. Currently, no datasets exist for full-size humanoid robots, but the accessibility issue can be partially mitigated by a dataset from Valkyrie. We have completed a set
CHAPTER 4. HUMANOID ROBOTS

of experiments demonstrating the mobility and manipulation capabilities of the robot while logging relevant sensor data from the robot to create the first full-size humanoid robot dataset.

We include data from sets of experiments in the following categories focused on isolating the different functionalities and modalities available on the robot. The set is not meant to be exhaustive, but representative of the capabilities developed on the robot in the past year and a half. These experiments can currently run in simulation and the real robot, helping provide a basis for comparison of simulation and real-world results.

- **LOWER BODY:** Walk (Forward, Back, Left, Right, Rotate), Step Up, Step Down

- **UPPER BODY:** Arms - 9 Grid of target points, Figure 8 trajectory; Torso - Movement (Forward, Back, Left, Right, Rotate)

- **FULL BODY:** Box pick and place with dual arms.

Sensor data including proprioceptive data is logged internally on Valkyrie during every run. The same logs can be generated in simulation so the tools used for data analysis on the real robot are also applicable to the simulation data. From these logs, the following time-synchronized data streams are available for use by the wider research community seeking access to data from a humanoid robot.

- Joint displacements, velocities, torques

- End-effector pose

- Estimated center of pressure (CoP), center of mass (CoM), ground reaction wrenches

- State estimator pose estimate

- Ground truth end effector and pelvis pose

Position ground truth for both lower and upper body tasks is provided through a Qualisys external motion tracking and position sensing system. As the robot completes the tasks, the 32 DOF robot configuration is tracked and logged. Torque ground truth is not directly available, but before each experiment the robot is placed in a gravity-compensation mode and any biases present in the series elastic actuators are nullified, resulting in precise and repeatable torque data.

The dataset is beneficial to researchers who do not have access to a full-size humanoid robot to better understand the realistic dynamic response of the system performing various tasks, which is difficult to accurately simulate. In addition, the dataset provides a basis for comparing
CHAPTER 4. HUMANOID ROBOTS

future revisions of the Valkyrie robot and other humanoid robots. The set of experiments is reproducible to enable future direct comparison of the performance of such systems. Finally, we envision that this dataset will allow researchers to test new state estimation and humanoid localization without requiring physical access to a robot.

Figure 4.20: Pelvis position of Valkyrie while walking forward as measured by the internal state estimator on the robot and the ground truth data from the Qualisys motion tracking system.

Figure 4.20 shows a sample figure that can be generated using the dataset. The figure shows the state estimator pelvis pose and the ground truth from the motion capture system while the robot walks forward. The figure is generated by taking the data from the robot and motion capture system, decimating the robot state estimator data to match the rate of the motion capture system (100Hz), and then using the ABSOR tool\(^1\) which implements Horn’s method\(^2\) for aligning the global fixed frame of the motion capture system and robot’s world frame. Since the robot position is common to both sources of data, we can utilize this to our advantage to find the translation and offset of the two frames. Once properly aligned, we can plot the data and analyze the performance of the state estimator with respect to the ground truth.

Figure 4.21 shows the distance error of the robot state estimator as measured from the ground truth. The data shows that the robot state estimator accumulates approximately 4 cm of

\(^1\)https://www.mathworks.com/matlabcentral/fileexchange/26186-absolute-orientation-horn-s-method
CHAPTER 4. HUMANOID ROBOTS

Figure 4.21: The distance error of the internal robot state estimator as a function of time with respect to the ground truth.

offset over the course of eight steps traversing approximately 2 meters. Overall, the state estimation on Valkyrie is very good, especially considering no SLAM or vision based compensation is implemented in this data set. The robot in this case is dead-reckoning.

The dataset would be useful to researchers looking to implement better state estimation algorithms because clearly there are dynamics that are not modeled and understood well during the steps that lead to these error. In Figure 4.20, the yaw error that results after a step is clearly seen and likely a significant contributor to the final 4 cm error offset. The raw sensor data used by the internal state estimator is recorded for the run, so new state estimation algorithms can be quickly benchmarked against the existing implementation without requiring access to the physical robot.

The dataset contributes to the cloud and context engines of the shared control framework by providing researchers without access to the robot, the ability to contribute new approaches and algorithms to improve the performance of the simulator to more closely match reality, make improvements to the algorithms on the real robot such as the state estimator, and ultimately integrate these improvements into better cloud-based simulations and context-aware approaches. When the DRC first started, the Gazebo simulator couldn’t accurately model any of the tasks the ATLAS robot could complete. Over the last few years, the simulator has improved immensely and now Valkyrie

118
CHAPTER 4. HUMANOID ROBOTS

can be successfully simulated with reasonable overlap between reality and simulation. The intent is that this data will enable the next push to make the simulations more closely emulate reality.

4.14 Future Work and Conclusions

In order to successfully deploy humanoid robots in future missions, several more years of research are needed. Improving Valkyrie’s ability to grasp and manipulate objects, both with one and two hands, is a key capability that is still in its nascent stages. The SRC tasks involve significant manipulation capabilities that the robot currently cannot accomplish. Further work on the constrained motion planning and adding contact models will help generalize the grasping capabilities to more generic objects and tasks. Improvements in the operator interfaces and providing situational awareness will be key to providing the robustness and reliability needed for realistic mission scenarios.

In addition, the integration of Valkyrie into a large and scalable shared control framework, will be critical for successful heterogeneous human-robot teams. A successful Mars mission in the future will involve various robotic and human systems in tight collaboration to accomplish the tasks at hand. These types of missions and tasks are currently not well studied in humanoid robots. Work towards enabling humans and robot to work in close collaboration, specifically in space exploration scenarios, will need to be a primary focus in the upcoming decade before we see a robot on the surface of Mars, helping push humanity further into space.
Chapter 5

Conclusion

We have presented a shared control architecture to enable the integration of operator input and autonomous behaviors in HiLCPS for space exploration, disaster, and assistive robotics. Several examples demonstrating the development of different elements of the shared control framework were detailed. A method for integrating human input with the autonomous behavior of the system using redundancy resolution taking into account primary and secondary tasks was presented. In addition, a blended shared control implementation was demonstrated on a Turtlebot in two navigation tasks with time delay and drift disturbances. The results and implementation details show the benefits that blended shared control can provide and can serve as a basis or template for future research in this area.

Future work on the blended shared control formulation should focus on improving the algorithm through three key areas: the selection of $\alpha$, new time delay and drift models, and utilizing higher complexity systems. We have demonstrated that the blended shared control algorithm works on one system with a specific $\alpha$ selection. While the experiments show that the $\alpha$ parameter results in good performance, there may be better ways to select the parameter. Considerations should include how the user’s like the feel of the change in control authority over the system. Second, new time delay and drift models should be tested with the blended shared control.

In the presented example, constant time delay and drift models were utilized. A future experiment can recreate the presented setup, but use varying time delays and drifts to analyze how these transient disturbances affect the performance of the blended shared control. It may be that operators are better suited to compensate constant biases in time delay and drift, and the shared control architecture would be more useful in more degraded conditions.

Finally, the presented system is limited to a simple mobile robot. With the interest in
mobile manipulators and humanoid robots over the last few decades, future work could focus on exploring if these results scale with system complexity. The redundancy resolution shown above and utilized as the basis for the blended shared control in the presented experiment can handle primary and secondary tasks for high degree of freedom systems. How these systems can be controlled with a blended shared control algorithm is an open problem, and the handoff of control authority between human and robot is not trivial.

Future work on the shared control framework can focus in several different directions including proving the applicability of the architecture in new application domains, developing a more diverse set of metrics to measure performance, expanding the control modalities, and proving the architecture outside the research environment. We have demonstrated the architecture in the applications that inspired its development, space, disaster, and assistive robots, but span a vast variety of domains. In each domain will place emphasis on different elements in the architecture and utilize different methodologies which will show the ability to incorporate those differences. These new domains will inevitably lead to improvements and changes to the architecture.

Closely related to expanding to different domains, traditional performance metrics are not necessarily applicable in all, so new metrics that are specific to can be discussed and analyzed using the architecture. Integrating the work from into the architecture would be a possible path for improvements of metrics for. For example, traditional controllers are evaluated based on their tracking performance, but with human-in-the-loop systems, humans may be able to handle more tracking error subconsciously in exchange for better intuitive “feel” of a given controller. Some of these metrics will be application-specific, but the ultimate goal is to have a set of metrics that transcends applications so implementations can be easily compared quantitatively.

In Figure 2.2, we propose four different control modalities that could be part of the action engine. These are control modalities that make sense in our applications, but there are many other modalities that could be explored. In, shared control is broken down into six different modalities. It is conceivable that a system could implement all six modalities and dynamically switch between them depending on the situation.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


[40] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In Intelligent Vehicles Symposium (IV), 2011.


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


