



**Michigan
Technological
University**

**Michigan Technological University
Digital Commons @ Michigan Tech**

Dissertations, Master's Theses and Master's Reports

2016

Simulation of Scalability for Autonomous Mobile Microgrids

Nathan Beyers

Michigan Technological University, ntbeyers@mtu.edu

Copyright 2016 Nathan Beyers

Recommended Citation

Beyers, Nathan, "Simulation of Scalability for Autonomous Mobile Microgrids", Open Access Master's Thesis, Michigan Technological University, 2016.
<https://digitalcommons.mtu.edu/etdr/124>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etdr>



Part of the [Electro-Mechanical Systems Commons](#)

SIMULATION OF SCALABILITY FOR AUTONOMOUS MOBILE
MICROGRIDS

By

Nathan Beyers

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Mechanical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2016

© 2016 Nathan Beyers

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Mechanical Engineering.

Department of Mechanical Engineering - Engineering Mechanics

Thesis Advisor: *Nina Mahmoudian*

Committee Member: *Wayne W. Weaver*

Committee Member: *Gordon G. Parker*

Department Chair: *Dr. William Predebon*

Dedication

To my mother, father, brother, teachers, and friends

who have supported me throughout my journey in life and have helped me every step of the way. Without you, this work would not have been possible.

Contents

List of Figures	xi
List of Tables	xvii
List of Abbreviations	xix
Abstract	xxi
1 Introduction	1
1.1 Motivation	1
1.2 Past Work	7
1.3 Contribution and Outline	13
2 Test Setup and Design	15
2.1 Robotic Platform	17
2.2 Connector	23
2.2.1 Existing Solutions	26
2.2.2 Design Iterations	29
2.2.3 Current Design	32

2.3	Robotic Arm	34
2.3.1	Camera	36
2.3.2	Gripper	37
2.4	Wiring	39
2.5	Electrical Control	42
2.6	Sources and Loads	43
3	Computation Environment	45
3.1	Layers of Navigation	47
3.2	Linux Operating System	51
3.3	The Robot Operating System	52
3.4	Data Visualization	55
3.5	Gazebo	57
3.6	Worlds	58
3.7	Husky Simulator	60
3.7.1	Model Modifications	61
3.7.2	Sensor Simulation	62
3.7.3	Multiple Robots	65
4	Results and Conclusions	69
4.1	Scenarios	70
4.2	Results	72
4.2.1	Development	73

4.2.2	Replication of previous work	75
4.2.3	Uneven Terrain	76
4.2.4	Connection Scaling	79
4.2.5	Load Testing	80
4.2.6	Diverse Agents	83
4.3	Conclusions and Future Work	86
	References	89
	A Validation Code Samples	93
A.1	SerialSample.py	93
A.2	GPSLocalization.py	95
A.3	environment_scan.py	97
	B Simulation Code Samples	105
B.1	NASLab.world	105
B.2	husky_empty_world.launch (Modified)	107
	C Letters of Permission	111

List of Figures

1.1	An example of a microgrid with diverse power generation, storage, and load systems [1]	3
1.2	An example of a robotic microgrid which shows the different types of agents [2]	8
1.3	Electrical schematics showing the basic structure of different types of microgrid agents [3]	10
1.4	An overview of the architecture involved in a mobile microgrid platform	11
2.1	An overview of the necessary hardware components for an autonomous microgrid	16
2.2	A size comparison of the previously use DaNI robots to the current Husky platform	17
2.3	The new Husky platform demonstrating its ability to handle rugged terrain	18
2.4	Added components for Husky Vehicle	19
(a)	Lidar	19

(b)	GPS	19
(c)	Wireless Adapter	19
2.5	A Husky vehicle with the top removed to access the mini-ITX computer	21
2.6	The full set of Husky robots used for testing	22
2.7	A robotic arm holding a traditional plug to illustrate the tolerance and alignment challenges of a robotic connection	24
2.8	A test of the self-aligning surface on the 3D X-face connector [4] . .	27
2.9	The first generation robotic connector used for initial testing	29
2.10	Second-generation connector with cup-shaped receptacle	30
(a)	Male Connector	30
(b)	Female Connector	30
(c)	Completed Connection	30
2.11	Third-generation connector with a cone and funnel shaped design and rings as contacts	30
(a)	Male Connector	30
(b)	Female Connector	30
(c)	Completed Connection	30
2.12	Fourth-generation connector with smaller size and locking center pin	31
2.13	CAD model showing the connection process of the current connector design	32

(a) Male Connector	32
(b) Female Connector	32
(c) Completed Connection	32
2.14 Demonstration of how the connector locking pins retract when the end plunger is pulled	33
(a) Pins extended	33
(b) Pins retracted	33
2.15 3D printed prototype of the current connector design	34
(a) Separated	34
(b) Connected	34
2.16 The CAD model of the mount to secure the camera to the arm . . .	37
2.17 The CAD model of the gripper designed to hold robotic connectors	38
2.18 The completed arm assembly with gripper and camera mounted . .	39
2.19 The CAD model of the spool system which prevents twisting of the connector wire	40
2.20 The 3D printed spool and mount with the slip ring added	41
2.21 Location of RS-232 communications port on the computer	42
2.22 Completed source and load cabinets for microgrid demonstration . .	44
3.1 An example of the path planner selecting a subgoal	48
3.2 An overview of the software architecture involved in a mobile microgrid simulation	50

3.3	Sample graphical map of transformation frames in the ROS environment	53
3.4	Visualization of lidar point cloud with the corresponding simulation view	56
3.5	The GUI and graphical display for the Gazebo simulator	57
3.6	An office building model loaded from the default Gazebo library . .	59
3.7	The modified Husky model for a bus agent robot	62
3.8	Sample data from the EKF showing the filtered tracking of four robots with reference to the experimental setup	63
3.9	The structure and flow of the launch files used by the Husky simulator	65
3.10	The formation of multiple robots beginning a test in the modified simulation environment	68
4.1	Two robots avoiding obstacles in the Husky playpen world	73
4.2	Comparison of a real lab test to the simulated lab workspace	75
4.3	A robot navigating the uneven terrain of a world created from a heightmap	76
4.4	A scaled up version of in-lab testing using four robots	79
4.5	A test of obstacle avoidance using eight robots	81
4.6	Time-lapse images of three agent microgrid setup simulation	84
	(a) Deployment	84
	(b) Navigation starts	84

(c)	Source agent in position	84
(d)	Cabling to bus agent	84
(e)	Running cable	84
(f)	Completion	84
4.7	A large-scale demonstration which assigns robots unique roles . . .	85
C.1	Permission letter for figure 1.1	112
C.2	ASME permission letter for figure 1.2	113
C.3	IEEE permission letter for figure 1.3	114
C.4	IEEE permission letter for figure 2.8	115

List of Tables

4.1	A timing table for the simulation speed during load testing. The first column represents the number of robots simulated. The second column identifies whether the controller was running on the same computer as the simulator or outsourced to a different machine. The third column displays the average speed of the simulation as a percentage of real-time throughout the duration of the test.	82
-----	--	----

List of Abbreviations

ABS	Acrylonitrile Butadiene Styrene
AC	Alternating Current
CAD	Computer Aided Design
DoF	Degrees of Freedom
DTR	Data Terminal Ready
EKF	Extended Kalman Filter
GB	GigaByte
GPS	Global Positioning System
HIL	Hardware-In-Loop
IR	Infrared
I/O	Input/Output
IMU	Inertial Measurement Unit
NAS	Nonlinear and Autonomous Systems
PCB	Printed Circuit Board
PEBB	Power Electronics Building Block
RAM	Random Access Memory
SIL	Software-In-Loop
SLAM	Simultaneous Localization and Mapping

URDF	Unified Robot Description Format
2D	Two Dimensional
3D	Three Dimensional

Abstract

Microgrids are small-scale, decentralized systems for generation, storage, and management of electricity to provide power within a local area. By establishing these microgrids using autonomous robots, they can be made safer and more easily reconfigurable. This thesis presents a plan and outlines the infrastructure needed for scaling up previous experimental work on such systems. This work brings together a variety of tools in the form of both software and hardware to set the stage for how further development can be completed. An architecture for physical testing has been built to validate the full potential of navigation and making electrical connections in unknown environments. This is done by selecting a robotic platform and integrating it with a robotic arm, a specialized locking electrical connector, a spool for laying wire, and a camera for visual servoing. This hardware architecture has also been modeled in simulation for expanded development, testing, and analysis. The simulation environment has been constructed in ROS using the program Gazebo. This environment has been customized to allow for simulation of multiple robots which match the physical system. Scenarios have been tested in simulation to show that it compares to real-world demonstrations and to scale them up with more robots.

Chapter 1

Introduction

1.1 Motivation

Every day, we see more and more tasks traditionally done by humans being taken over by robots and computers. They have revolutionized nearly every industry from manufacturing to travel to housework. If you have a smart phone in your pocket, you are carrying a powerful computer with you nearly everywhere you go. It is capable of connecting to millions of other devices. You can use it to control a drone and view the camera output on-screen. Or you could use it to command an autonomous vacuum to start cleaning your carpet. But there are many other areas in which robotic systems play a role. They are crucial to much of the critical infrastructure that we rely on

day to day. That same phone in your pocket was likely assembled by a series of robots, each performing specialized tasks to optimize the cost, speed, and quality of production. It was then likely carried in a plane which uses an autopilot system to maintain course in the air or over land to shipping center. Perhaps it was then directed by a sophisticated distribution algorithm to be delivered to your door. All this before even making it to your hands.

There's a very good reason for the prevalence of robots in so many diverse areas. Robots don't get bored, tired or frustrated with their jobs. They can continue to do repetitive tasks hour after hour and day after day while maintaining the same quality results. Robots can also be customized to achieve things not humanly possible. They can achieve nanometer precision, they can lift thousands of pounds with ease, and they can have virtual eyes in the back of their head. In a world where safety is key, robots all but eliminate the potential for injury or loss of human life in many dangerous jobs. In the past, the practical uses of robotics may have been limited to relatively simple tasks. But in recent times, that is changing. For example, self-driving vehicles exist today which have shown to be safer than the average human driver [5].

The reason for this change is the ever-increasing technology in computers. The processor in a robot can be capable of thousands of calculations per second, giving robots much better reaction times than their human counterparts. This means decisions

which can be made by a computer are getting more and more complex. More data can be considered and more scenarios can be evaluated successfully. In short, robots are allowing us to make breakthroughs in nearly every field of science. Through creative application and synthesis with other groundbreaking technologies, robots can open up a world of new possibilities.

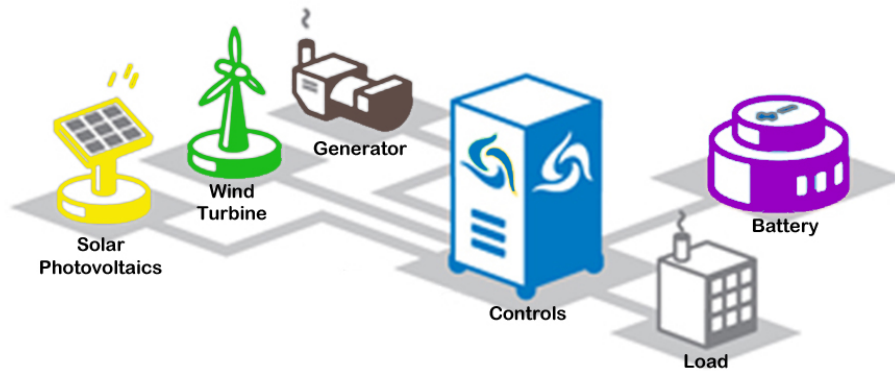


Figure 1.1: An example of a microgrid with diverse power generation, storage, and load systems [1]

One of these groundbreaking technologies to which robotics may be applied is a sub-field of electrical engineering and power systems: microgrids. Microgrids are small-scale distributed systems for electrical power distribution [6]. A microgrid is a small set of systems consisting of many nodes which manage the flow of energy from generation through transmission and storage to an eventual use in a particular load [7]. Figure 1.1 shows a microgrid unit formed from several nodes connected together. Each node has a role, whether it is generating power, controlling the flow of power, or using it. The system in the figure could exist in isolation or be connected to other

similar systems in a network. In these networks, connections must be monitored and managed to ensure proper delivery of electricity to its destination. The area where microgrids really thrive is in small, often isolated systems. The small size means that they can be easily set up and the interconnected nature allows for reconfigurability and tolerance to system failures. These features would be enhanced by mobile robotics to allow for unmanned deployment and dynamic connections between components.

With my work on mobile microgrid systems, I aim to provide tools that will revolutionize the way electrical systems are constructed and configured. When autonomous robotics and microgrid systems are brought together, each piece of technology helps to overcome challenges of the other.

For mobile robots, power is a major concern. Robots need a constant source of power to drive and to run on-board computers and sensors. This means that they generally must be taken out of operation for recharging or refueling. But if these robots could be tapped directly into a power source, as they would be if the mission was to configure a microgrid, that energy would be readily available to them. The connection to a power system would allow for continuous operation and optimization of resources.

For microgrids, one of the biggest obstacles is the infrastructure needed for creation. A lot of work goes into constructing a microgrid and that often incurs high cost and risk for those building it. But if a microgrid could be set up using autonomous mobile robots, the cost and risk could be reduced. This would allow for microgrids to be

deployed in areas where they have not been possible or cost effective in the past. The applications of microgrids will be greatly expanded.

One such application would be in forward operating bases for military groups. These groups require power on their bases for everything from mission operations to defenses to lighting. If these bases are established in a dangerous area, it can be very risky for troops to set up the electrical infrastructure. If this step could be done in advance through the use of autonomous robotics, it would limit the time workers are put at risk. The electrical systems used for defense could already be active by the time humans get there, so the threat would be much lower.

Another area where a mobile microgrid would lend itself quite well is in disaster areas. During events such as widespread disease outbreaks like the Ebola emergency [8], hospitals and quarantine areas need power more than ever. Rapid-response Microgrids could be used to manage power at existing locations or to quickly establish new sites for combating the contagion. Again, this would allow the people who would otherwise need to enter the area and create the power grid to stay out of harm's way. The robots could also work to re-establish power when an existing grid has been damaged. In the Hurricane Katrina disaster, power was lost in critical buildings like hospitals. Even with generators, power couldn't last long. People were unable to call for help because cellular towers were down [9]. So workers had to go in, entering into dangerous areas, and restore the damaged system.

Microgrids set up by robots would also be incredibly helpful in space missions. It is a common topic of discussion in current news that humanity may visit Mars within this century [10]. And when they get there, it would be very helpful if a microgrid for distributing power were already in place. Robots would be ideal for this task. They could be sent out to the Red Planet to set up this network on their own. And if these robots are autonomous, they wouldn't be limited by the painstaking communications latency needed to receive commands from earth. We could see a sustainable robotic system which operates for an extended period of time without any human intervention, getting the planet ready for future missions.

Since autonomous mobile microgrids could be so useful in such a wide range of situations, from military bases, to disasters, to exploration, it makes a lot of sense to start developing the technology that would be up to such a task. This is the reason myself and my fellow researchers are working to lay the foundation for these types of systems, which could have a huge impact on the future of power distribution. To accomplish this mission requires providing an architecture for establishing a microgrid using mobile robotics and then the validation of this architecture to show that it can be feasible, reliable, and scalable to operate in such harsh environments.

1.2 Past Work

Microgrids are a relatively new, but rapidly growing field. Much work has been done on the advantages and considerations of such systems. They fill a useful role in-between more traditional power systems and have been successfully applied in specialized areas. They are smaller and more manageable than the massive, centralized power systems such as the nuclear, hydroelectric, and coal power plants which provide electricity to hundreds of homes. Yet microgrids are more robust and versatile than most distributed systems like a backup generator providing power to a single home. The way a microgrid operates is by linking small, modular units together [11]. These units can then be configured, added, or removed as needed.

The concept of using autonomous mobile robots to establish microgrids however, is a much more limited field. Most of the foundation for this work has been done by the Nonlinear and Autonomous Systems lab at Michigan Technological University. In previous work, my colleagues demonstrated the viability of creating a robotic microgrid with the ability to autonomously reconfigure. This strategy allows for optimization of resources without the need for human interaction. [12]

A small-scale proof of concept was implemented using National Instruments DaNI robots to run cable and connect power from a battery (source) to a light (load)

[13]. These small robots operated using a rotating IR scanner mounted on a servo to scan the environment and detect obstacles. An initial demonstration included a single robot running the wire to provide power from the battery to the light. The robot was deployed into an environment with both obstacles and targets. Using data from an overhead motion-capture camera system for localization of the DaNI and the targets, the robot plans a route. It then uses feedback from the IR scanner to gather data about its surroundings. This data is then interpreted as either a gap or an obstacle. This demonstration laid the groundwork for identifying the needs of a mobile microgrid architecture.

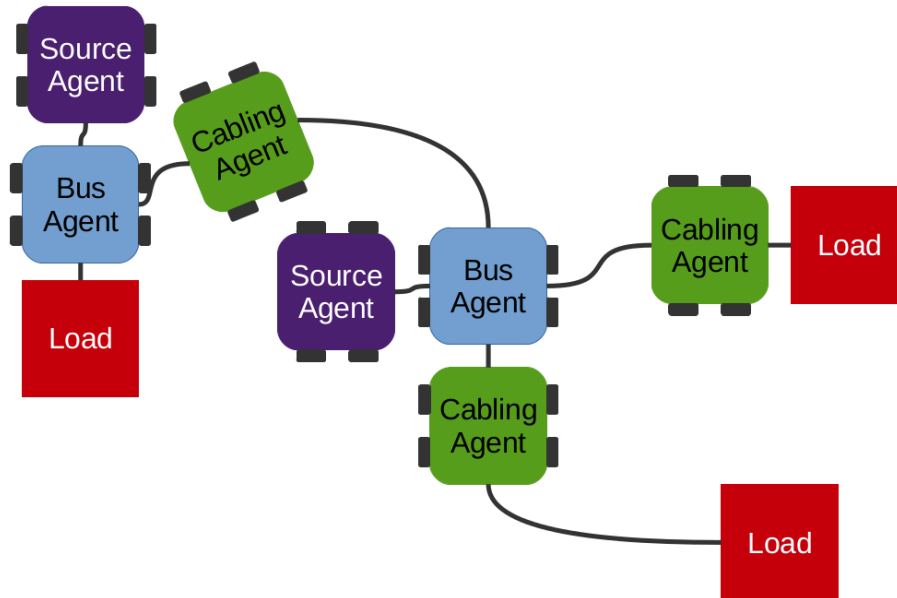


Figure 1.2: An example of a robotic microgrid which shows the different types of agents [2]

A second demonstration later expanded upon this work by showing two robots being

deployed to the field. These robots simultaneously navigated to separate nodes and then met up in a central location to establish a connection [2]. In this work, the different components of an autonomous mobile microgrid architecture were identified. In addition, the work discusses the use of at least three types of specialized “agents” needed for a microgrid. These agents, defined below, are based on a breakdown of the tasks for microgrid setup and operation.

† **Source Agent:** This agent provides or stores power for the system. It will be equipped with mobile power generation technologies such as solar panels or diesel-electric generators. The source agent should also have a battery bank for the storage of unused energy. Source agents do not have to move around much during their operation, so they may be less mobile than other agents.

† **Cabling Agent:** The most mobile agent in the system is the cabling agent. This robot runs the wire to make connections between nodes of the microgrid. As a result, this vehicle may need specialized perception equipment in addition to wire and connection gear. Once the grid is set up, cabling agents will be able to reconfigure the grid as needed.

† **Bus Agent:** This agent acts as the control and power management system for the microgrid. It allows for many connections made to it and has methods of transforming voltages based on the needs of the system. Bus agents will serve as the links from sources to the loads they power.

Figure 1.2 shows how each of these types of agents would link together to form a microgrid. This setup includes connections to fixed loads that are in need of power. Seven robots are shown to illustrate the multiplicity of each style of agents. The same framework could be scaled up with more agents to provide power to an arbitrary number of loads.

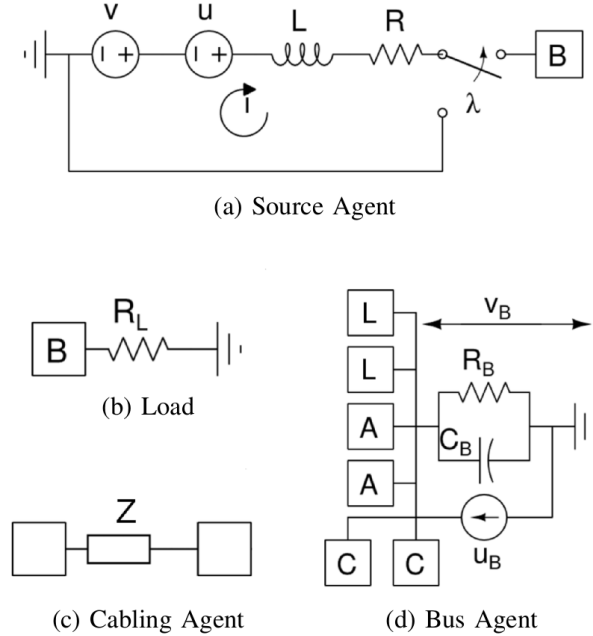


Figure 1.3: Electrical schematics showing the basic structure of different types of microgrid agents [3]

Following the definition of different types of agents, some of the electrical structures for a mobile microgrid were identified [3]. Figure 1.3 shows the basic electrical representation of each of the agents. The source agent consists of a power supply and a battery for storage. A boost converter is used to step up voltage during transmission. A load can be modeled as a resistor which dissipates power. The cabling

agent provides a direct line between two points and can be described with only the impedance of the wire used. Finally, the bus agent acts as a control circuit while accepting multiple connections. Using these electrical building blocks, it is possible to characterize the electrical properties and needs of the completed microgrid.

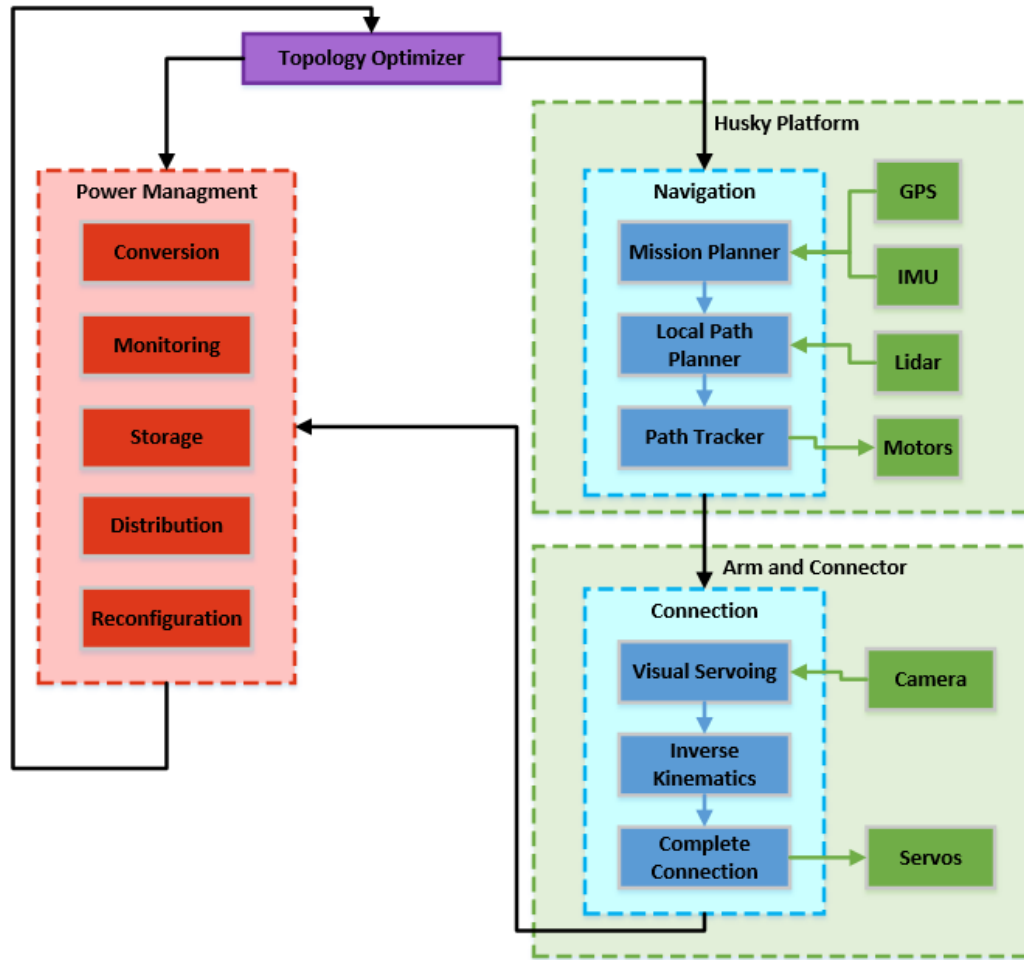


Figure 1.4: An overview of the architecture involved in a mobile microgrid platform

In addition to the electrical components, a software architecture for such a system

was outlined. The proposed architecture is shown in figure 1.4. A topology optimizer handles the planning for how the robots should be arranged and configured to maximize efficiency, minimize time, and plan for available resources. Below the topology level, two systems are identified: the the electronics system and the navigation. The electronics consists of the power management components which regulate power and control the microgrid. The navigation system is made up of the mobile robot platform and components used to maneuver around the environment and establish connections.

The proposed path planning algorithm for an autonomous mobile robot consists of three parts:

1. A high level mission planner generates a route from the current position of the robot to its desired destination.
2. This path is then augmented by a lower-level local path planner which uses robot sensors to detect any obstacles and reacts by determining a way around them.
3. Finally, the proposed route is followed using a path tracker which directly controls the motors of the robot.

A unique part of the work being done on this project that differs from more common autonomous vehicle control algorithms is that the robots know very little about their

environment prior to starting the mission. Many popular systems, such as most self-driving cars, have detailed map information for localization. Other robots learn about an environment and plan a route at the same time using a method known as simultaneous localization and mapping [14]. However, if vehicles are required to navigate in unknown or rapidly changing environments, then they must be capable of path planning without a known map. This type of map-less navigation is well suited to the microgrid architecture.

1.3 Contribution and Outline

The work until now on autonomous power distribution systems has been largely conceptual. It has focused on identifying the need for such a system, outlining how it could be done, and proving the capabilities on a small scale, in a very controlled environment. But real world applications have yet to be demonstrated. To further develop the technology, an effort is needed to scale up the demonstration. This is being done simultaneously in three ways:

1. The physical scaling up of the robots to include larger vehicles.
2. Through simulation of robots in a microgrid configuration to test the robustness of the system.

3. By moving the tests from the laboratory environment into outdoor, real-world terrain.

The work presented in this thesis encompasses all these efforts in scaling the system. My contribution involves the design and integration of both hardware and software components to bring autonomous mobile microgrid systems to the next level.

The content of this thesis is outlined as follows. In the next chapter, I discuss the design work and infrastructure needed to begin using larger robots with more sophisticated sensors. This chapter identifies the hardware components and explains how they fit together for a realistic demonstration. The third chapter provides an overview of the simulation tools being used as well as the modifications which allow for the microgrid vehicles to be simulated. The fourth chapter discusses the results of simulations and validation. This chapter gives an evaluation of the performance for the system under more extreme conditions such as more obstacle dense environments and scenarios with larger numbers of robots. These simulated results are then compared with observations from physical testing to draw conclusions about the future of autonomous mobile microgrids.

Chapter 2

Test Setup and Design

Based on the previous work with mobile microgrids described in Chapter 1, several necessary components were identified for the scaling up of the demonstration into more realistic scenarios. In this chapter, the hardware components needed for a real-world demonstration are defined and integrated. The goal for this setup was the design and experimental validation of a scaled up microgrid architecture which could serve as a base for expansion and future development.

Figure 2.1 shows a basic overview of the various components which build up to a complete autonomous microgrid. As a starting point, a robotic platform for all three types of agents (source, cabling, and bus) is needed. For the agents to link together,

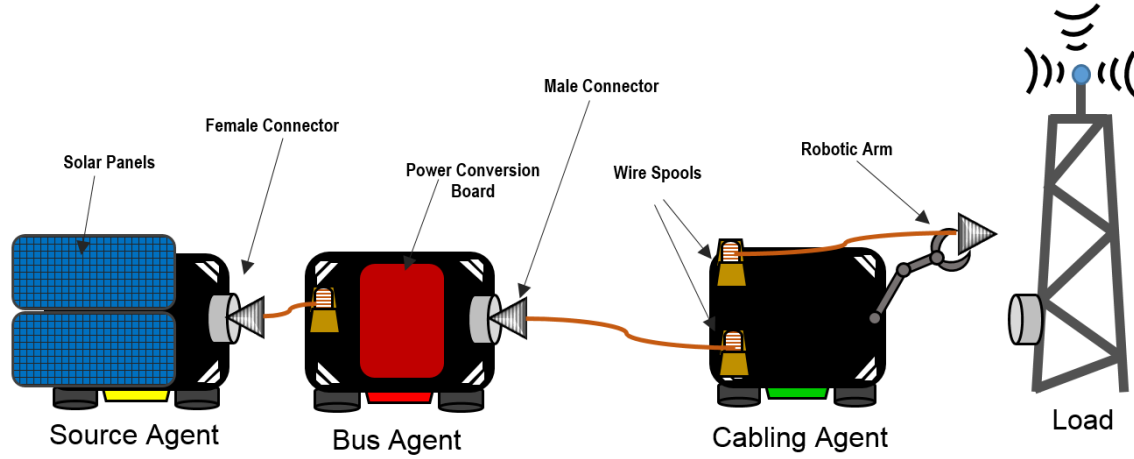


Figure 2.1: An overview of the necessary hardware components for an autonomous microgrid

a method for making connections and running wire must be developed. These connections must be completed by an actuator such as a robotic arm. The bus agent will be equipped with a Power Electronics Building Block (PEBB) board which is being developed by a collaborator for the project. This board must be capable of interfacing with the computer on the robotic platform. Additionally, electrical components are needed to represent the voltage sources and the resistive loads in the microgrid system.

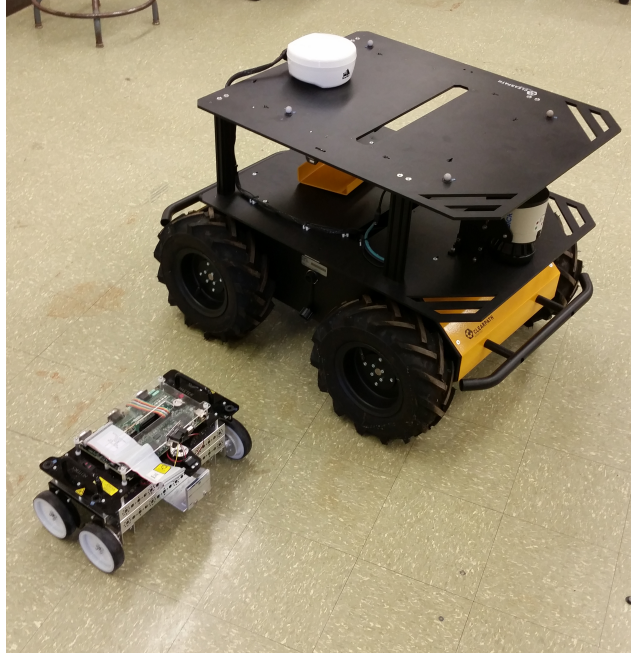


Figure 2.2: A size comparison of the previously use DaNI robots to the current Husky platform

2.1 Robotic Platform

After a successful proof of concept, the next step is to demonstrate the capabilities of an Autonomous microgrid system in the real world. For this, a larger robotic platform is needed. The previous platform, the DaNI robot from National Instruments stands about 10 inches tall and has a wheel diameter of 4 inches. It works well for the lab setting, but lacks the capability and sensors needed for operation in the real world. If the robotic vehicles are required to operate in locations such as a disaster area or rough Martian soil, they must be able to traverse small obstacles or uneven

terrain. Additionally, these robots must have a visual sensor with the speed, resolution, and sight distance to be useful in outdoor environments. Since the new platform is no longer operating solely in a lab equipped with an overhead camera system for millimeter accurate precision, an alternative method of localization is needed. For the new platform, a combination of GPS and other sensors is used. Appendix A.2 provides code which demonstrates an example of this localization method. Figure 2.2 shows the size comparison between the previously used DaNI platform and the new vehicle.



Figure 2.3: The new Husky platform demonstrating its ability to handle rugged terrain

The scaled-up version of the demonstration uses Husky robots from Clearpath. This robot was designed with off-road environments in mind, as demonstrated in figure 2.3, so it is well suited for the scaling effort. The robot weighs about 120 lbs fully equipped and measures roughly 2 feet tall with 13 inch wheels. This gives it 5 inches

of clearance to operate in snow, grass, sand, or mud as well as climb over small rocks and branches. The vehicle also operates with a differential drive controller similar to the one used by the DaNI robots. This means that each half of the robot is controlled independently by a separate electric motor. The differential drive allows the robot to turn in place as opposed to traditional automobile steering. With a max payload of 165 lbs, there is plenty of capacity for the tools, sensors, and accessories needed for the equipment used on the different types of agents in the microgrid demonstration [15].

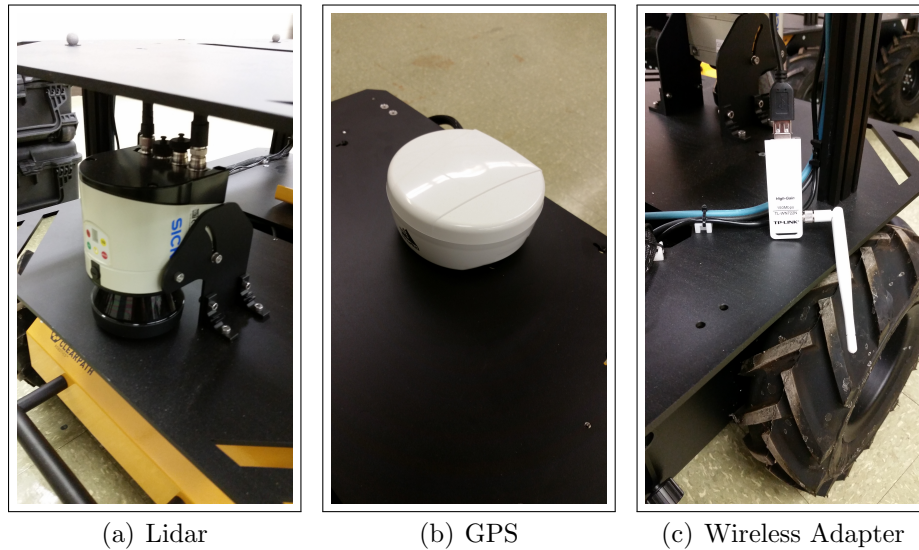


Figure 2.4: Added components for Husky Vehicle

The Husky platforms used for our work are customized with a suite of sensors to meet the needs of outdoor navigation. To maintain a more uniform architecture, all agent types utilize the same sensors for this stage of development. Some of these are displayed in figure 2.4. The sensors include:

† **A Sick LMS151 lidar unit:** This system, shown in figure 2.4(a), provides the vision for the robot. It scans on a single plane using a reflected laser signal to determine a set of measurements with distance and angle coordinates. Each sweep provides 1080 datapoints in a 270 degree aperture angle at the front of the robot. The sensor is much faster and with a wider field of view than the previously used IR sensor, allowing for faster navigation with better awareness of obstacles. The sight distances is also much improved. The maximum range of the sensor is 50 meters, allowing the robot to see well in front of it to recognize obstacles and plan around them in advance. The Sick LMS 151 is designed for outdoor use, so it is resistant to the elements and corrects for fog and reflective surfaces [16].

† **NovAtel SMART6-L:** This GPS sensor, seen in figure 2.4(b), will take the place of the overhead motion capture camera system used in the lab. It is resistant to weather and movement, making it useful for on outdoor vehicles. Using multiple channels and constellation tracking, the unit is capable of accuracy within 1.2 meters. It also provides localized data for recording movement as if it were on a 2-dimensional plane which is useful for robot tracking within a small area [17].

† **MicroStrain 3DM GX3-25:** This is an IMU, which provides information on the heading or attitude of the vehicle as well as position, velocity, and acceleration data. This is done by using a 3-axis accelerometer and gyroscopes to

measure linear and rotational changes in velocity [18]. It is also equipped with a calibrated magnetometer which can determine cardinal directions based on measurements of Earth's magnetic field. This sensor is able to provide a heading for the robot as the overhead camera system did previously. This is needed because the GPS alone can not provide a reliable direction when stationary.

† **Wheel Encoders:** Encoders are mounted on each wheel to measure rotational information. This is useful in the odometry of the robot because it can be provide an estimate for number of turns and distance traveled. However, when a loss of traction occurs the vehicle will slip and so encoder data will drift unless corrected by another source.



Figure 2.5: A Husky vehicle with the top removed to access the mini-ITX computer

These sensors are linked to a mini-ITX computer, which is located in the internal bay of the vehicle, behind the battery. This computer can be accessed by removing the

top of the Husky, as shown in figure 2.5. The mini-ITX is equipped with a quad-core processor, 8 Gigabytes of RAM, and a 120 Gigabyte solid state hard drive to provide on-board computing for each robot. A 24 Volt, 200 amp-hour lead acid battery provides power for roughly 3 hours of runtime. Additionally, the computer can be connected to a wireless network via the TP-Link adapter shown in figure 2.4(c).



Figure 2.6: The full set of Husky robots used for testing

Our lab has also scaled up the number of robots being tested. Previous tests were done with only two vehicles to show the mechanics of making a connection. But a usable microgrid scenario would require many more robots, each with specialized jobs. In order to demonstrate a more complex scenario which captures this, our resources include a fleet of four robots. This allows for a source agent, a bus agent, and two cabling agents in real world testing. The fleet of robots we have to work with are shown in Figure 2.6.

It is worth noting that although the Husky platform is capable of running outdoors

on reasonably rough terrain, it is still not expected to be the final platform for a microgrid. A full microgrid would require heavier components such as large solar panels or generators and heavy cabling equipment. Vehicles might also need to move over larger obstacles and hazardous terrain. It is likely that the vehicle used in the field for a full scale microgrid would be larger to handle a bigger payload and would be customized for its desired environment of operation. The reason for using the current platform is the need to continue development and shift that development to a more sophisticated platform which is capable of operating outside.

2.2 Connector

In order to demonstrate that the robotic microgrid system is capable of transmitting electricity, it is necessary to autonomously make connections between the robots and sources or loads within the network. This is the primary job of the cabling agent. For this agent, an electrical connector capable of transmitting power must be added. However, robots encounter unique challenges that we humans do not often think about when plugging in an appliance. Picture the robotic arm in figure 2.7 attempting to connect a standard plug into an outlet. The main difficulties are:

- † **Alignment:** For a human connecting an electrical outlet, it is easy to see both ends of the connector in 3-dimensions, correct any error with a slight movement,



Figure 2.7: A robotic arm holding a traditional plug to illustrate the tolerance and alignment challenges of a robotic connection

and receive both a tactile and visual response when the connection is successful. For a robot, this task is not so trivial. A plug requires a high degree of precision in order to be connected. If a robot attempting such a connection is misaligned by just a few millimeters, the attempt will fail. To achieve the required precision requires very precise sensors and actuators as well as more sophisticated control algorithms. All of these factors contribute to the overall cost of the connection mechanism.

† **Rotation:** Most connectors can only be coupled when they meet at a particular

rotation about the axis of connection. Because of this, even if the alignment is perfect to make the connection, the connection may fail due to incorrect rotational orientation. Correcting of this orientation requires a degree of freedom which might otherwise be unnecessary in such a system.

† **Connection Force:** A standard electrical outlet requires significant force to make a connection. When using small-scale mobile robots, the actuator making the connection is often not capable of producing such forces. For this reason, a larger and generally more costly actuator must be used or a different connector must be substituted.

In addition to the mechanical requirements of a connection, there are certain electrical considerations which must also be taken into account. Firstly, since the connector will be operating as part of a microgrid and transmitting electricity, it must be capable of handling large current loads. Secondly, power will likely be transmitted in the form of three-phase alternating current. To accomplish this, a connector will need three live pins, a ground, and likely another pin for confirming connection success. This means that a connector will likely need at least five pins to ensure full functionality.

For these reasons, many standard off-the-shelf connectors are not suitable for a robotic power-grid. An alternative design is needed to better suit the unique set of challenges and advantages when working with robots. Next, I describe the set of the existing connector designs which were explored.

2.2.1 Existing Solutions

Since providing power to a mobile robot is not a unique challenge to microgrids, many other robotic systems have managed to overcome some of the difficulties in making a connection autonomously. One example is the Roomba, which is a robotic vacuum cleaner that maneuvers around the floor until returning to a base station to recharge. This is done through a combination of infrared and radio signals which allow the robot to determine its location and proximity to the base station. With this position knowledge, the robot approaches the base station until the wheels sit in a specially designed groove. Once in position, spring-loaded pins contact metal surfaces on the underside of the robot for charging [19].

The contacts on the Roomba work similarly to the design tested by Kartoun et. al. [20]. This design uses a set of powered rails at set heights. The robot drives toward the rails and flexible prongs positioned at matching heights contact the rails for a connection. Another solution in academia comes from researchers at the University of Southern California [21]. This solution uses a ball-and-cup shaped connection. A ball shaped probe on the vehicle is guided into a cup-like mating component on the base station. The cup acts as a funnel to overcome positioning errors and the flexibility in the mechanism reduces the problems caused by poor alignment.

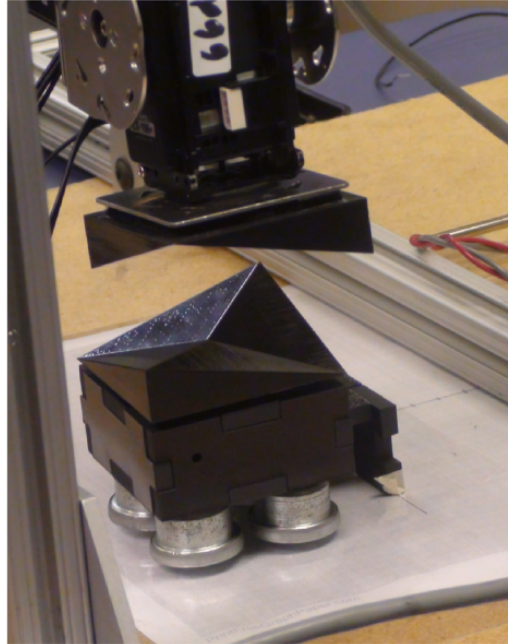


Figure 2.8: A test of the self-aligning surface on the 3D X-face connector [4]

Researchers from the Mod Lab out of University of Pennsylvania have also done research on self-aligning connectors which would be useful for autonomous robotics [4]. The design they have come up with uses 3-Dimensional X-shaped grooves which allow the connector faces to rotate and line up with one another when pressed together. They have compared their own design to several other mainstream designs discovered this style to have a superior tolerance to offset.

However, none of these connectors exactly fit what we were looking for. The primary reason for this is the need for a locking mechanism to keep the connector in place as the robot moves around. Many of the other designs are intended for robots that return to a docking station for charging and during this time they are inactive. However, for the

continuous operation that is required of a microgrid, our robots need to be constantly moving. They can't be attached to a large base station and the connection must be resistant to tension in the cable. A locking mechanism allows a connection to be made and fixed in place as the robot continues to move about its environment. Other work has considered this as well. The same researchers at Mod Lab have developed a locking mechanism for their 3D X-Face connector [22] for self-configurable robots to link with one another. The DRAGON connector [23] also features a locking mechanism which uses an electrically actuated memory alloy to secure and release the connection.

But after considering these designs, we decided to develop a new one of our own. The reasoning was that making our own design would give us complete freedom in the functionality. This meant that we could be free to customize our design to scale with the robots. As the microgrid gets larger and more complex, the connector must as well. It must have the possibility to add more contacts, larger wires and contacts capable of handling high current, and an adjustable size which will work for both the current Husky robots and later generations of the project. Ultimately, we can make a connector which works with our robots rather than making our robots work with a connector.



Figure 2.9: The first generation robotic connector used for initial testing

2.2.2 Design Iterations

The design process went through several stages, each one helping to teach lessons and refine the design. The first was a prototype which used magnets for alignment. As the two halves of the connector were brought near to each other, two strong, polar magnets brought them together and caused them to align. This connector used a right angle design, shown in figure 2.9, to pull both sets of contacts onto their respective mates. However, the magnets needed to bring the contacts together in just the right way and often didn't align perfectly, causing failed connections.

The second design used a cup-shape to help with alignment and kept with the idea of magnets for a locking mechanism. A single magnet on a rotationally symmetric connector would attract to another at the center of the cup. This design is shown

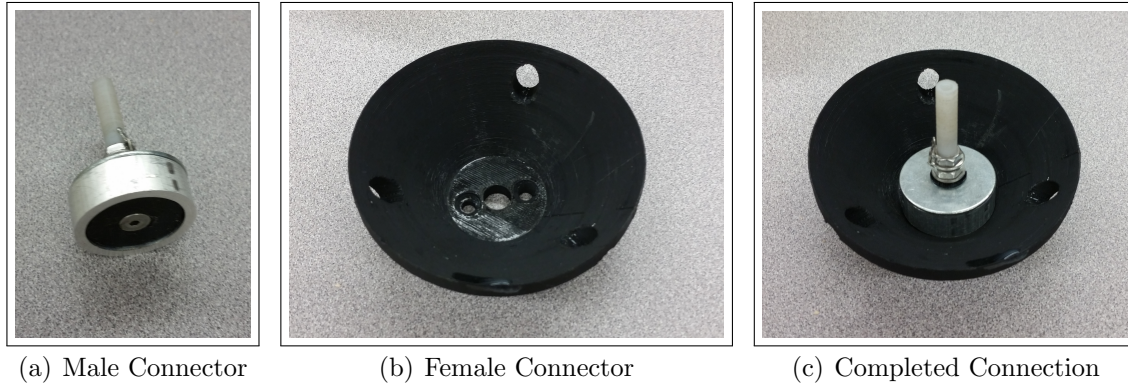


Figure 2.10: Second-generation connector with cup-shaped receptacle

in figure 2.10. Magnets are used in many modern day connectors [24], but they had some disadvantages in our tests. Because they were fixed pole magnets, the forces were largely uncontrollable. Connectors were difficult to disconnect and stuck to other magnetic materials when brought within close proximity. Additionally, these magnetic connector designs did not have the full five contacts needed for three-phase AC power.

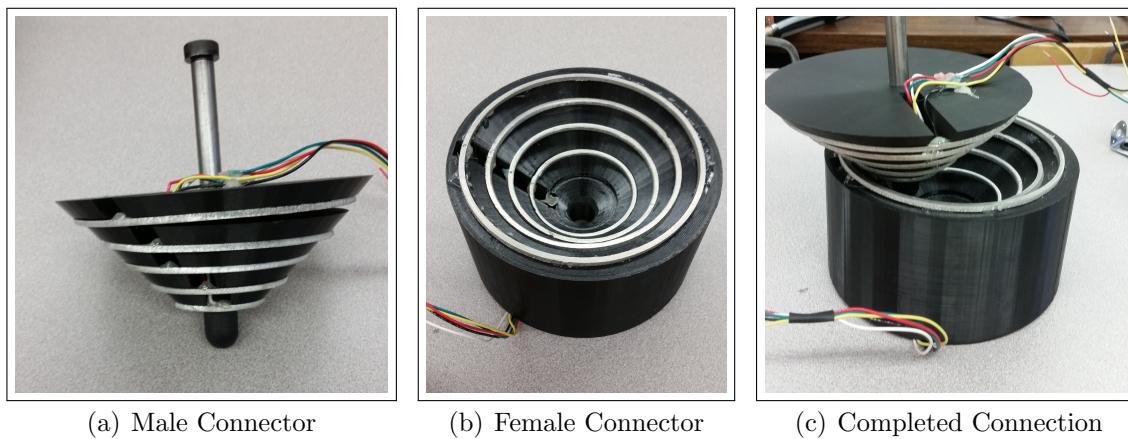


Figure 2.11: Third-generation connector with a cone and funnel shaped design and rings as contacts

The next design was a shift to a cone-shaped male connector and a funnel-shaped mate. Both pieces have a set of conductive rings on these surfaces which press together when in contact. The cone shape is sectioned and separated by springs which compress to ensure that all rings can remain in contact simultaneously. The idea is that the funnel and cone will correct offset and alignment and the cylindrical symmetry means rotation about the axis of connection will not have any effect. This design was never implemented due to its large size and difficulty to machine the rings. It also had no locking mechanism to hold the connection together. The third-generation design can be seen in figure 2.11.

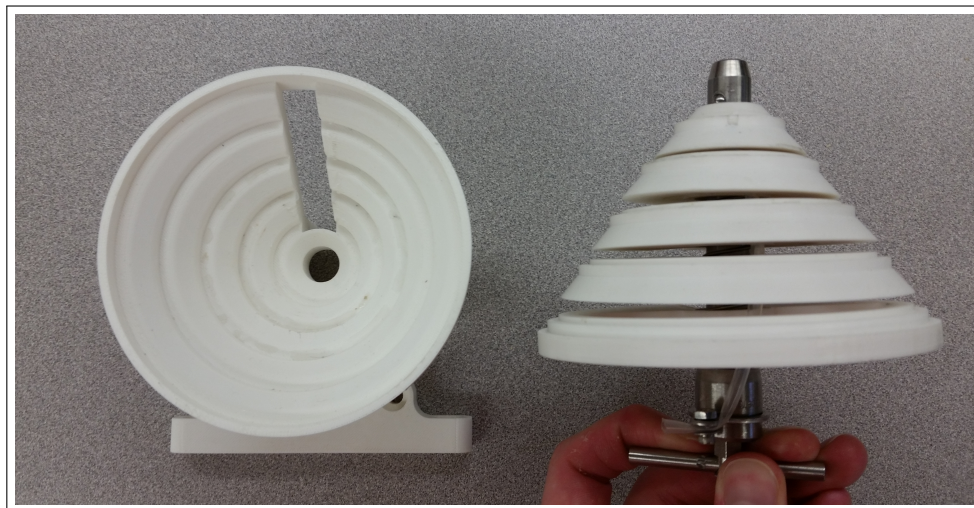


Figure 2.12: Fourth-generation connector with smaller size and locking center pin

The following connector prototype was smaller, used easy-to-make rings from PCB, and featured a special locking bolt at the center. These features are shown in figure

2.12. However, the locking mechanism required a pin at the end to be pulled separately and then released once the connector is in position. Furthermore, the male part was still too heavy for a robot to easily handle. A still lighter version was needed.

2.2.3 Current Design

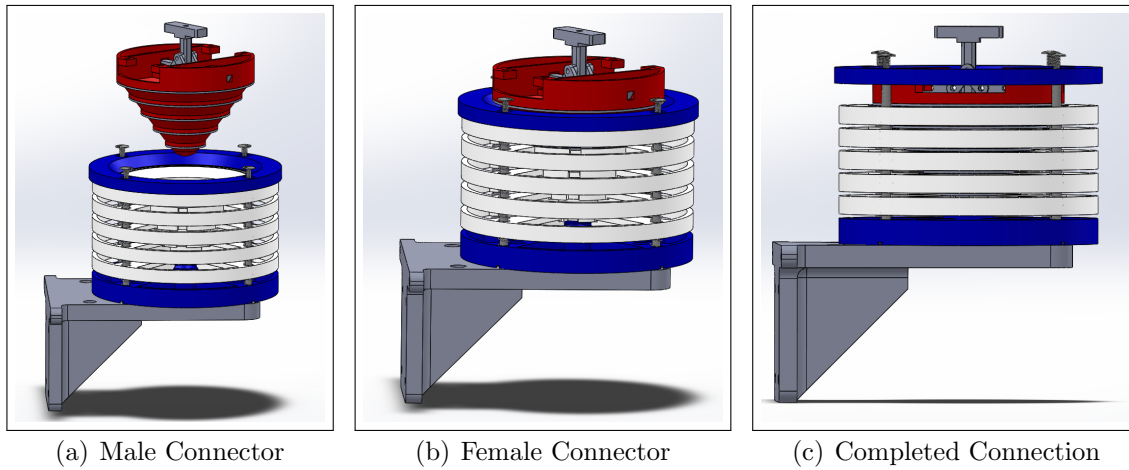


Figure 2.13: CAD model showing the connection process of the current connector design

The current design for the connector sticks to the cone and funnel shape, but moves the springs, sections, and slides to the fixed female part. This allows the male connector to be hollow, making it lighter and leaving space for a better locking mechanism. The 3D model for this design is shown in Figure 2.13. This figure goes through the steps of a successful connection. The connector starts at a distance with imperfect alignment in figure 2.13(a). Then, in figure 2.13(b), the conical shape brings the two halves of the connector into alignment. Finally, in figure 2.13(c), the connector is

pressed into place, compressing the springs to ensure mating of all contacts and to allow the locking mechanism to hold it in place.

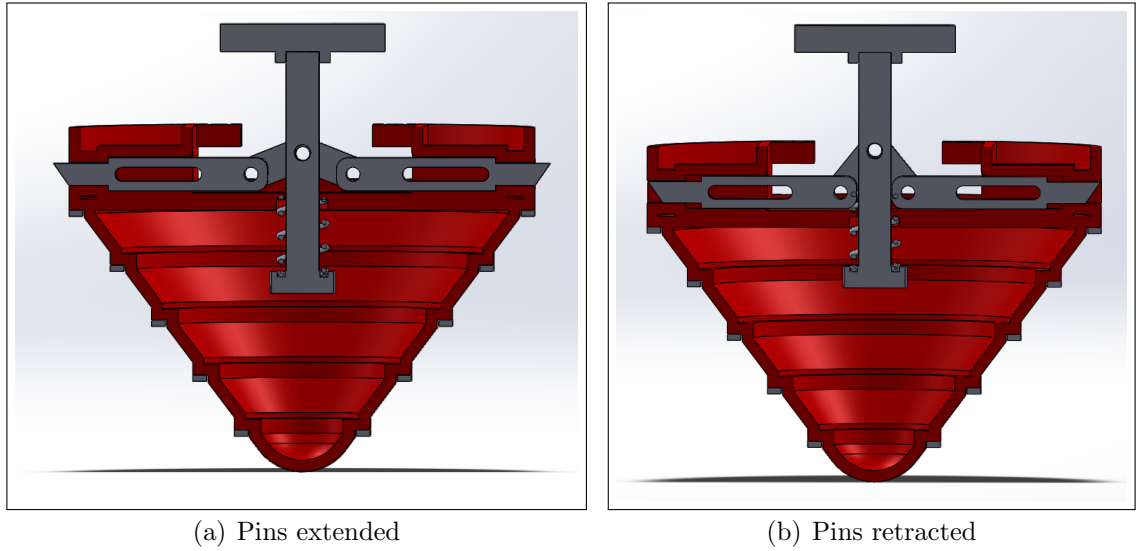


Figure 2.14: Demonstration of how the connector locking pins retract when the end plunger is pulled

The new locking mechanism uses a central, spring-loaded plunger which retracts two pins that extend from the body of the connector. Then the connector is pressed into position, the female part compresses, all contacts are made, and the pins extend under a lip in the female connector to lock the connection in place. To release, the plunger can be pulled and the connection is free to separate. The locking mechanism is shown in the sectional view in figure 2.14 where the linkage can be seen to translate the vertical pulling of the plunger to the horizontal motion of the pins.

Figure 2.15 shows the completed prototype of the current connector design. This design was 3D printed from plastic and assembled for testing. Initial results show

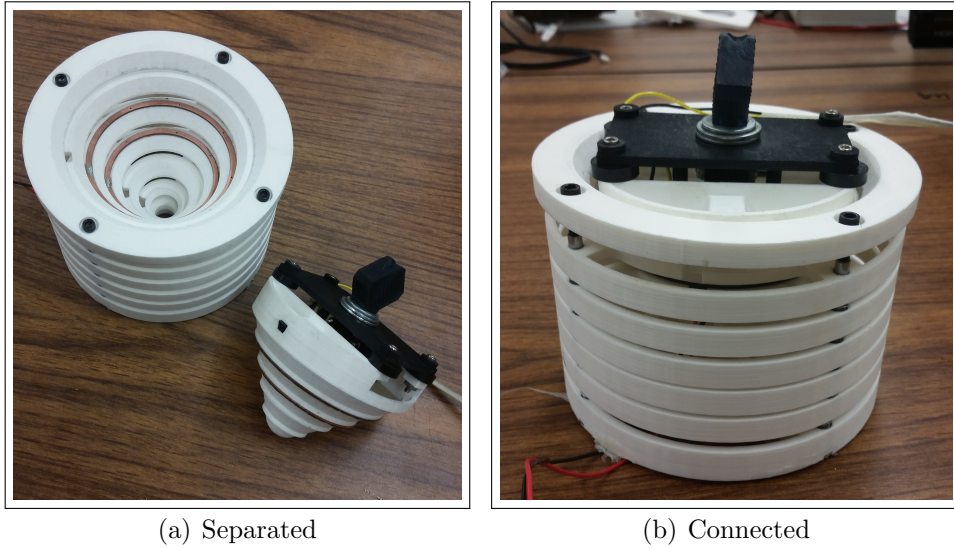


Figure 2.15: 3D printed prototype of the current connector design

that it can be used with the robotic arm and that the electrical connections are successful when mated. However, friction from the rough edges of the rings can prevent a successful alignment from occurring. Future revisions to the design will likely be needed to improve the connection success rate of the design.

2.3 Robotic Arm

When moving from lab testing to outdoor environments, the problem of making a connection has some additional challenges. Since cabling agents will be operating in potentially rough terrain, a connector cannot be assumed to remain at a fixed height as the robot moves. A system must be added which can compensate for height changes

and offsets due to uneven terrain. In these three-dimensional, outdoor environments, positioning the connector is not a trivial task. One reason for this is that the GPS has a lower accuracy than the overhead camera system, so the positioning is not reliable enough to make a connection on its own. In addition, uneven ground found in the real world can cause a misalignment between the two mating sides of the connector that the robot cannot overcome no matter how well it is positioned.

To correct for the accuracy and alignment issues when scaling up, a new system is needed to provide additional positioning control on the connector in 3D space. To do this, we have selected the Widow-X robotic arm from Trossen Robotics [25]. This arm has six servos with high resolution rotational control, which allow it to move in space relative to the robot. This arm can fine-tune positioning of the connector, as long as the robot can get close enough to the connection point. The robotic arm is assembled from laser-cut ABS plastic and aluminum parts to maintain a strong and lightweight frame capable of holding onto an electrical connector and pressing it into place. The gripping mechanism allows for up one-half kilogram of weight to be carried. It is sized to provide a 41 centimeter dome of reachable area within which a connection can be made.

The arm will be secured on the top of the Husky robot to act as a manipulator for making connection between nodes of the microgrid. In the future, it may be possible to use the manipulator to perform additional tasks such as clearing small obstacles

from the path of the robot.

In order to use the connector with the robotic arm, the arm had to be outfitted with custom equipment for visualization and gripping. This was accomplished by installing a camera and a gripper for the connector.

2.3.1 Camera

The arm must be able to position the connector to make a connection. This means an additional sensor is needed to locate the mating connector once the robot has navigated to the general area of the target. We do this through the use of a camera and image tracking. When the robot approaches its target, the camera will activate and search for a set of markers. Once found, the location of these markers gives a very accurate knowledge of where the connector must be placed. The controller can use this to move the arm and make the connection.

To attach the camera to the arm, a special mount was created. The 3D model is shown in figure 2.16. This mount attaches to the end effector of the arm and allows for an adjustable angle for the camera. By adjusting this angle, an optimal approach can be found. Currently, a 45 degree angle is used to provide a balance between arm reach and force to make the connection. With the information from the camera view and the servo angles, the robot can compute the position of the connector relative

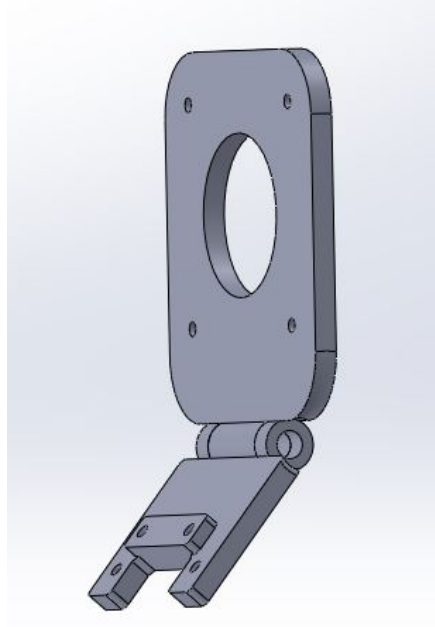


Figure 2.16: The CAD model of the mount to secure the camera to the arm

to the rest of the arm. using this knowledge, the inverse kinematics for making a connection can be developed.

2.3.2 Gripper

A new part is also needed to allow the arm to hold the connector while making a connection. This gripper must perform two functions simultaneously: it must be able to hold and release the connector while at the same time operating the locking mechanism. This is done by kinematic coupling of the two motions. The gripper is designed with smooth, rounded jaws which close together on the plunger end of

the connector. Due to the angle on the connector, the jaws slide underneath and pull the plunger outward as they close. This motion retracts the locking mechanism and allows the connector to be removed when gripped. This coupling simplifies the process of making a connection. While other connectors have a two-stage connection which involves first positioning and then locking, this design allows both to be done in one single motion of the gripper.

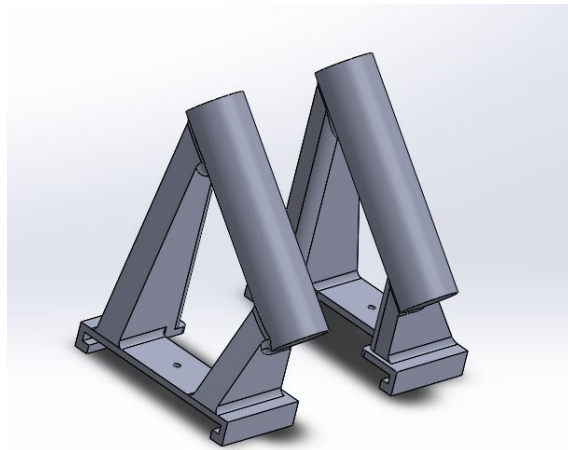


Figure 2.17: The CAD model of the gripper designed to hold robotic connectors

The design for the gripper is shown in Figure 2.17. The jaws have a 45 degree slope to match the camera angle on the arm. At the base of each is a track which is identical to the slides found on the standard end-effector for the arm. The servo on the Widow-X attaches via a linkage to the hole at the center of each half of the gripper, so its rotation is translated to a linear opening and closing motion, similarly to the standard tool. This means the end effector can be directly swapped out for

the custom design.

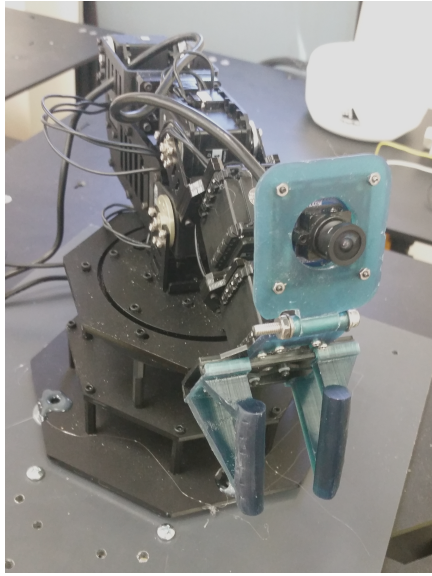


Figure 2.18: The completed arm assembly with gripper and camera mounted

The customized robotic arm is shown in figure 2.18. In this image, the camera and gripper are attached and the arm has been mounted to the top plate of one of the Husky robots, where it will be able to pick up and place connectors.

2.4 Wiring

As cabling agents travel and make connections, they must continuously lay wire as they move. During early tests, wiring was run off a simple spool of wire with one end connected to the robot and the other attached to the connector. After the connection was made, the robot would drive to the next objective and the wire would unwind

from the spool. However, this led to a problem: the fixed end caused the wire to twist as the spool rotated. This created a failure point in the wire when the twisting forces exceeded the material strength. In order to mitigate this type of failure, a new component was designed for holding and laying cable.

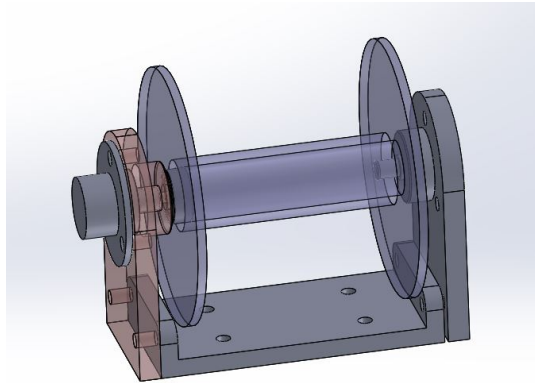


Figure 2.19: The CAD model of the spool system which prevents twisting of the connector wire

A custom spool system was designed to allow for rotation without a twisting of the wire. This was done using a slip-ring electrical connector at the base of the spool. This connector is a wire with a rotational point at the center which allows free rotation. Starting with this slip-ring in mind, a spool system was designed around it. The design of the fixture consists of two sides and a base. Each side has a boss where a bearing can be press-fit in place as shown in the semi-transparent view in figure 2.19. These bearings allow for the free rotation of the spool on the fixture. Both sides then bolt to a base, mounted to the robot, to be held in place.

The fixture can hold a manufacturer spool to avoid having to wind the wire around

and the additional cost of a new part. However, in future work, it may be beneficial to have a method for easily retracting wire that has been uncoiled. For this, a new spool with a gear or pulley may be used which can be rotated via an electric motor. In the interest of lower cost and increased simplicity, this has not yet been implemented.

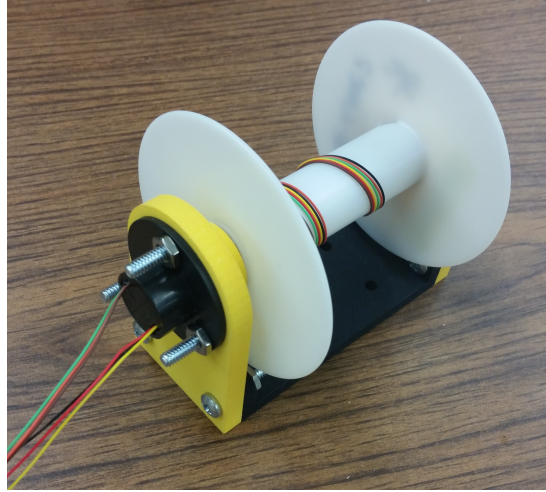


Figure 2.20: The 3D printed spool and mount with the slip ring added

The design was manufactured using 3D printing and is shown in figure 2.20. Once assembled, the design was tested and observed to resolve the issue of twisted wire. Based on the volume of wire, the current size will be adequate for at least 100 feet of wire, more than enough for either indoor or outdoor testing. The design is also readily scalable for use on larger microgrid systems which have more cable with added weight and larger diameter.

2.5 Electrical Control

Another part of a successful microgrid demonstration involves the control and management of power. Robots cannot be navigating in an unknown environment with live wires as this would be dangerous. Instead, power must be switched on after a successful connection. This is a key requirement for the bus agent, which regulates the flow of electricity with the PEBB board. This board must be electrically linked to the computer of the robot so that the electronics and navigation systems work together. To model this additional complexity, a method was needed to control power from the robot computer.



Figure 2.21: Location of RS-232 communications port on the computer

For a controllable electrical output, one of the RS-232 connections on the mini-ITX board was used. The port can be accessed on the Husky vehicle by removing a the top panel from the vehicle and connecting directly to the I/O panel of the computer, as seen in figure 2.21. Within this port, the DTR pin was used as a binary controllable

voltage. This is done using python code and the “serial” module. Given a COM port number and a baud rate, the pin can be set to values of ± 25 volts. This voltage may be used to power up wires once a connection has been made or to shut them down when not in use. Because this is done in python, the same language that most of our vehicle control algorithms is written in, it can be directly integrated with the existing code. In addition, other pins and multiple RS-232 ports could be used to control multiple components simultaneously.

The current version of the code can be found in Appendix A.1. This code was run to demonstrate a timed switching of the signal, which was tested using a multimeter. These tests verified the changing voltages and showed how the signal can be controlled. For the next stage of demonstration, this signal will be used to activate power for the electronics control board.

2.6 Sources and Loads

To complete the demonstration of a microgrid on a larger scale, the vehicles, sensors, connections, wiring, and communications have all been described, so the remaining pieces are the sources and loads. The loads have been modeled as cabinets which open to allow electrical equipment to be placed inside. For demonstration purposes, these cabinets can house a 12 volt lead-acid battery to provide power as a source. For

the load, a 12 volt DC motor is attached to a spinning pole. When this pole receives power, it spins to show that a successful connection has been made.

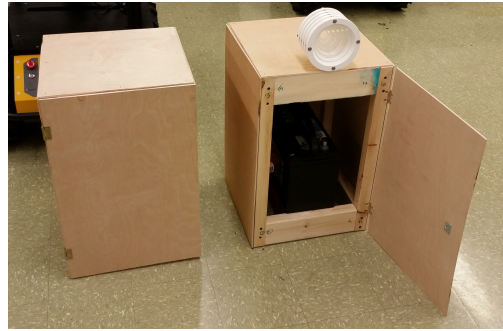


Figure 2.22: Completed source and load cabinets for microgrid demonstration

As a simple solution, the cabinets were constructed from wood and fasteners. This allowed them to be cheap, reasonably lightweight, and easily machinable. The finished design was created in 3D modeling software and constructed. Figure 2.22 shows the completed design with a mounded connector and battery.

Chapter 3

Computation Environment

The previous chapter outlined the hardware components needed for scaling up of previous development for an autonomous mobile microgrid project. This chapter will focus on the complementary software involved in the process.

The computational setup is a very critical part of the design and scaling of an autonomous system. This setup includes everything from the type of computers and controllers running the vehicles to the operating systems and specific programming languages that are used. Depending on the selection of these components, different tools become available to help meet the goals of the project.

In addition to controlling the vehicles and facilitating the communication between

sensors and actuators, computers can assist in the development of autonomous systems in another way: simulation. Simulation allows either hardware-in-loop (HIL) or software-in-loop (SIL) development to be done in a virtual environment. This is a new tool for the microgrid project that gives us significant advantages when used alongside real-world testing.

One of the main advantages of simulation is cost. By doing development in a virtual environment, many of the hardware components can be left out of the test. For example, a sensor could be easily added to a simulation model to test whether it works for a particular task. This is a low cost way to determine whether the sensor is compatible before purchasing it. The simulated system can also be tested for scalability. It can help to determine how many robots can operate on a network without having to purchase an arbitrarily large number of robots. If a problem with the code causes a vehicle to collide with an obstacle, a simulation can just be modified and reset whereas the same problem in a field test might result in costly damage to equipment.

Another advantage is time. Setting up either an indoor or outdoor test with real robots is a very time consuming process. It often requires swapping components, moving and positioning vehicles, setting up the environment with both targets and obstacles, and configuring each robot for the particular test. Simulation skips much of this. Tests can be configured and run with a simple script and possibly even sped

up to simulate faster than real-time. Testing code in simulation allows code to be changed and tested on the spot, rather than waiting until after the conclusion of a test. By shortening the development cycle, many bugs can be caught earlier on, before they cause further problems.

In addition, testing code in a virtual environment allows for repeatability. Simulations can be configured the exact same way each time they are run, a benefit which would not be impossible in a real-world scenario. This means that problems can be easily repeated for better debugging and new versions of code can be directly compared under the same conditions.

As we move to more powerful computers and adopt simulation into our development cycle, it is useful to outline the tools used and the methods by which that development occurs.

3.1 Layers of Navigation

This section describes the general layout for the autonomy code to be used in the autonomous microgrid setup. This is an important consideration because the layout of the code impacts the selection of development environments for future work. The autonomy system consists of several layers.

At the highest level is the mission planner. This manages the overall destination that the robot will attempt to move to, based on its current position. This layer is also in charge of determining when the vehicle has reached a target and when to continue to the next one. The global path planner operates with position data from both the robot and the target to approach. Since it does not have any prior information about the map nor any potential obstacles along the way, it will attempt to travel in a straight line to the target.

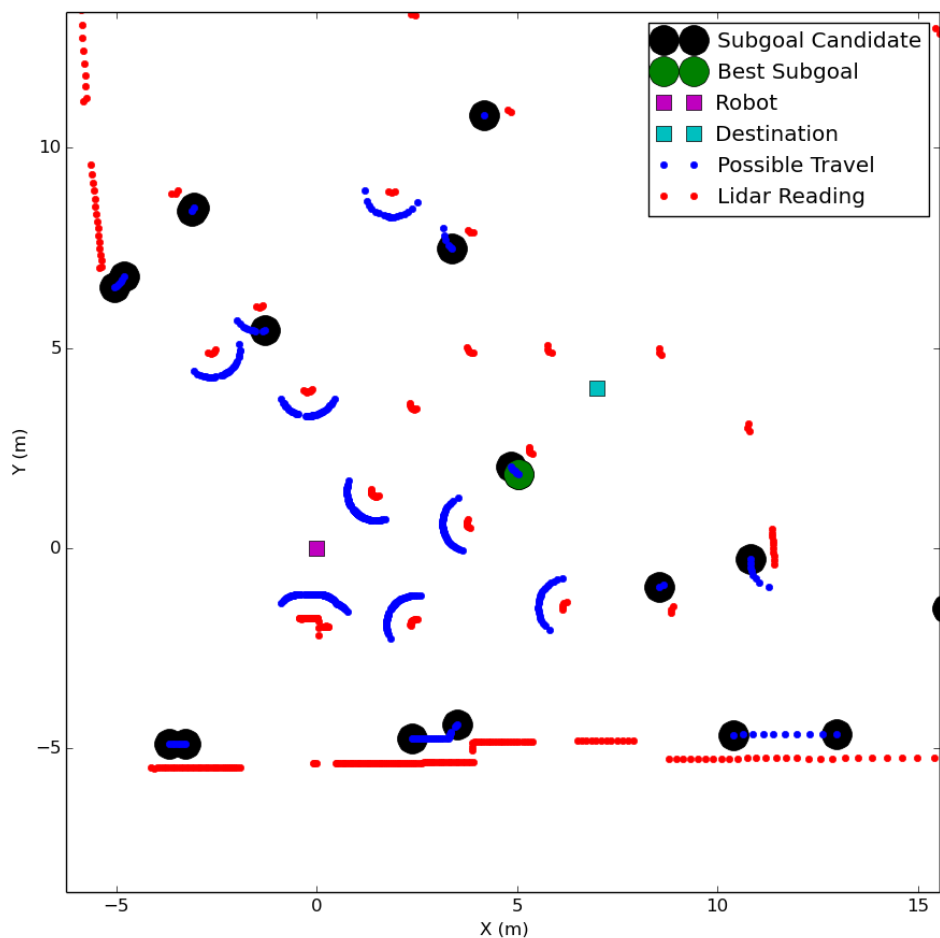


Figure 3.1: An example of the path planner selecting a subgoal

Operating below the global planner is a local path planner which takes over when an obstacle is encountered. This is essentially the obstacle avoidance algorithm. It uses primarily lidar information to determine where obstacles are located, find viable gaps to get around it, and then determine the direction of travel which best matches the global planner. In figure 3.1, the planner has scanned the environment and determined how far the robot can travel in any direction. Using this information, several “subgoals” are determined to get around obstacles. Finally, the best subgoal is chosen based on the location of the desired destination.

The actual movement commands are sent by the path tracker algorithm. This takes the path determined by the local and global planners and projects it into a set of points for the vehicle to follow. Essentially, this breaks the current path up into sub-destinations and tells the robot how to follow them. This tracker will also throttle the speed for the vehicle based on its proximity to obstacles.

Data from the sensors feeds into different layers of the software to provide information used to make decisions. To manage all these layers, it is helpful to design with an object oriented approach. A sample of the mission planner code is given in appendix A.3 showing the use of objects to represent levels of the navigation system. With this approach, each piece of the algorithm can exist as a separate object which stores relevant data and interacts with other objects. Multiple instances can be created for many robots operating in the same network. To fit this multi-level, object oriented

design approach we have selected Linux systems running ROS as the development environment. These systems are described in more detail in the following sections.

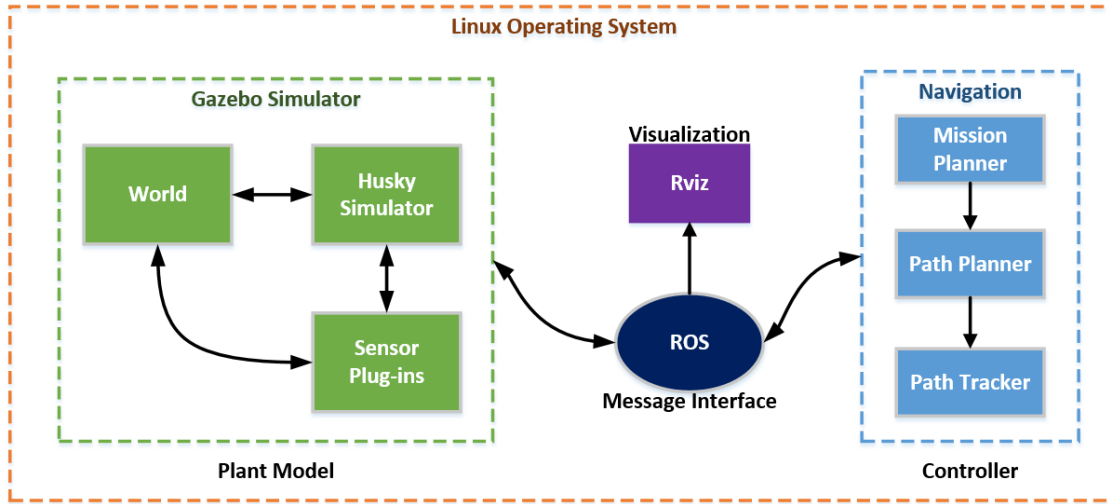


Figure 3.2: An overview of the software architecture involved in a mobile microgrid simulation

Figure 3.2 shows how all these various software components fit together. The navigation system described above makes up the controller for the simulation. This software runs identically on either the physical Husky robot or the simulated version known as the plant model. These components work together, assisted by the ROS interface, all within a Linux system.

3.2 Linux Operating System

For working with the robots, our operating system of choice is Linux. The mini-ITX computers on the Huskies each runs Ubuntu 12.04 while the two workstations for development use Ubuntu 14.04. This is very versatile platform since it does not rely only on embedded code compiled directly to the robot. Because each Husky computer has Linux, a keyboard and monitor can be connected and used to install software or write code directly. The robots can also be connected to a wireless network and accessed remotely in order to make changes, start tests, or transfer software.

For the workstations, Linux allows for a wide variety of programs which include browsers, video editing tools, word processors, and integrated development environments for code. This give workstations full functionality for development of software and for configuring and running simulations. On the vehicles, a much more sparse version is used to keep processing power focused only on the path planning and autonomy algorithms. However, additional software can easily be added as needed.

These systems are all linked together via a local wifi network. This allows all robots to be able to send and receive data and communicate with one another as they travel around the lab. Also included on this network are virtual machines. These instances of Linux which can be run from a host operating system such as Windows to facilitate

the use of tools from both systems simultaneously.

The Linux environment is well suited for larger-scale, collaborative software development which is necessary for scaling up the project. It allows for source code control using tools such as git to share code and track changes through the development process. It also provides more freedom for different languages and tools over the previous embedded environment.

3.3 The Robot Operating System

To handle messages and low-level communication between components of the robot, the Robot Operating System (ROS) is used. ROS is a set of standards, packages, and messaging components which allow the sensors and actuators to work together.

What this looks like on the robots is the ROS core system running at the center of several “nodes.” Nodes are additional pieces of hardware or software which can be either imported or designed by the user. Each node can send and receive information in the form of messages called “topics,” which utilize a subscriber/broadcaster relationship. For example, each of the sensors exists on the system as a node, broadcasting the information that it outputs as a topic. These topics are formed with a template and contain information such as the message type and the time at which it was sent.

The motor controller also subscribes to a topic which is used to tell the vehicle how to move. Between these is the autonomy controller which publishes commands to the motor controller based on readings from the sensors.

Internal tools allow the user to view topics, information about them, and the types of data being published across them. ROS also provides an interface for publishing these topics through the command window. Using this interface, virtual or real robots can be driven manually and faked sensor information can be generated. New topics can be created as well, such as the positioning information broadcast from the overhead camera system.

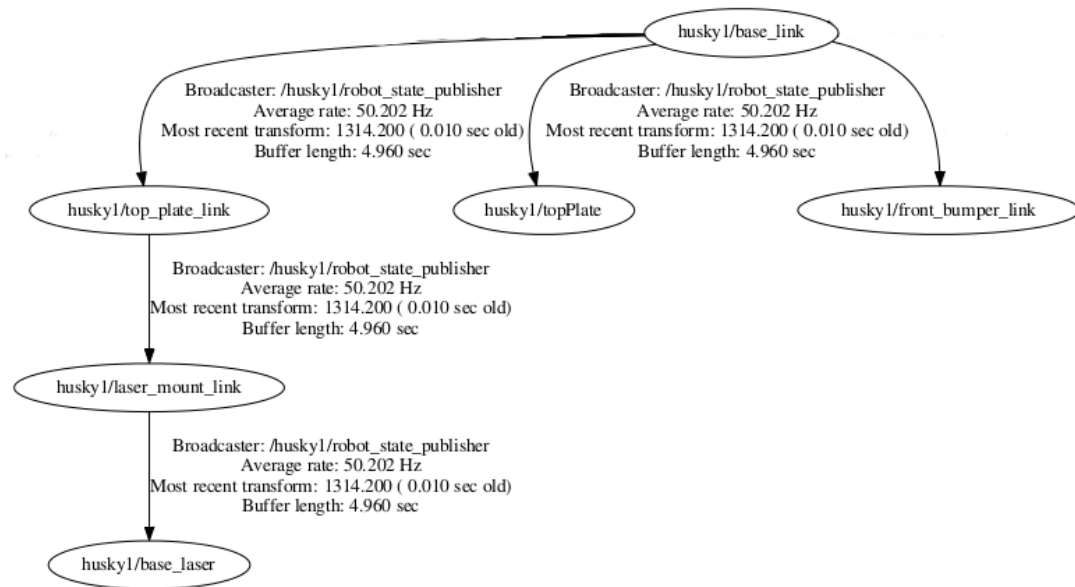


Figure 3.3: Sample graphical map of transformation frames in the ROS environment

In addition, ROS keeps track of positioning information in the “tf” library. This library stores either static or dynamic links between components and broadcasts how they change over time. The coordinates of these components, known as frames, can be mapped out to show how each interacts with one another, as seen in figure 3.3. Through this positional awareness information that can easily be implemented with ROS, multiple robots with different configurations and multiple joints can all be handled simultaneously. This is a feature which is needed for a diverse robotic system such as a microgrid.

The suite of packages on our installation of ROS further help development in a variety of ways. One particularly helpful tool is the use of bag files. These are files which save all information published by a robot over the ROS network. This allows for data to be stored during a test and later analyzed for results. For example, the Lidar data may be examined to identify what may have caused the robot to stop unexpectedly in the field. This data can also be played back as if it were being broadcast in real-time, for use in replicating events that occurred in testing.

As any software, ROS is constantly changing and evolving. There are many different distributions which can be used. For the robots we use in lab, the legacy ROS hydro is installed since this is what was shipped on the robots and is proven to work. For simulation, the newer ROS Indigo distribution was chosen for some of its more advanced simulation tools.

3.4 Data Visualization

One of the most important parts of autonomous systems is data from the sensors; both the real-world and the simulated ones. So when designing such a system, it is very helpful for debugging and understanding if the data coming from the sensors can be interpreted. This process is known as data visualization. Data can be visualized in many different ways, such as time series, scatter plots, and 3D renderings. For example, with the GPS data that our robots produce, it is useful to display position information as path: a timestamped map of where the robot has traveled.

One very helpful tool for visualizing many different types of ROS data is the program “rviz.” This software contains methods for displaying the model of a robot and tracking frames of reference. It also allows data from cameras, lasers, IMUs, and other devices to be shown relative to this model. One of the main sensors on the robot is the lidar, which scans in a plane and outputs a list of angles and distances from each scan. Using rviz, this information can be converted into a point cloud and displayed on a monitor as shown in Figure 3.4.

The image at the top of the figure shows the robot in a simulated world, positioned before a set of traffic barrels, barriers, and other common objects. The simulated lidar performs a scan over 270 degrees in front of the robot and rviz then generates

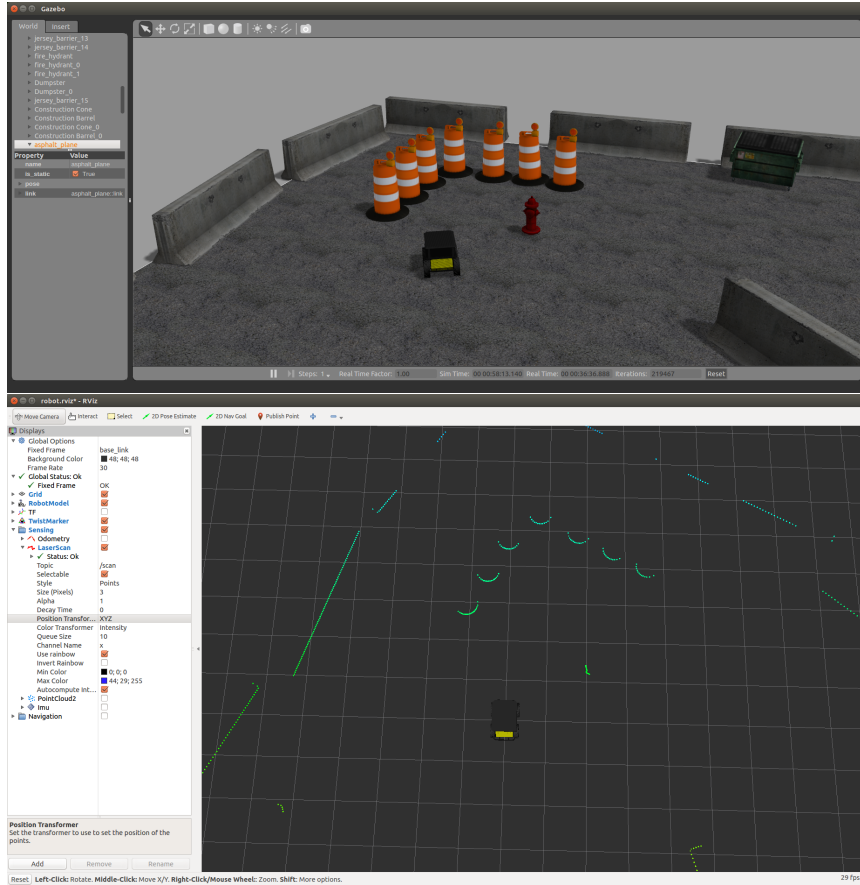


Figure 3.4: Visualization of lidar point cloud with the corresponding simulation view

the point cloud in the lower image. Using this method of visualization, we can see what data the control algorithm is working with. It is often easy to recognize why the robot is behaving in a way that might otherwise appear erratic.

3.5 Gazebo

The tool used to simulate the vehicles, movement, and sensors is a program called Gazebo. Gazebo interfaces directly with ROS and acts as a node which broadcasts topics concerned with the position and joint states of robots. The program actually consists of two different parts: the physics engine and the graphical display.

The physics engine portion computes interactions between the robot and the various elements in the environment. It uses several different solvers to simulate motion dynamics and track the results as vehicles move. This includes complex interactions such as slip that occurs at the robots wheels on a smooth surface or the resultant momentum of objects after a collision. The environment can be fully customized to change solvers or to edit the values for gravity and simulation speed.

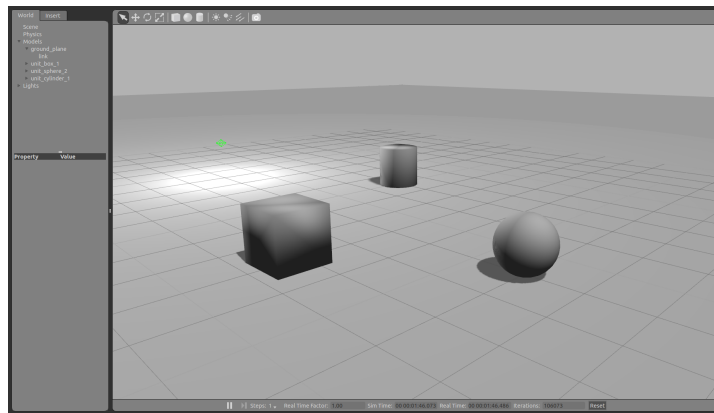


Figure 3.5: The GUI and graphical display for the Gazebo simulator

The graphical display shows the user what is happening in the simulation. It does not need to be run, but it is a good method for observing the results of a particular test. Using this tool, lighting and texture effects can be added to enhance the visual qualities of the demonstration. The display updates as the simulation runs to show the user the states of the robots and how the models are moving. The graphical display is shown in figure 3.5 along with the GUI for managing changes to the simulation.

Gazebo allows models to be either built or imported from 3D modeling files. This means accurate representations of robots and obstacles can be loaded and viewed in the simulation. These models are in a Unified Robot Description Format (URDF) file which contains XML code to configure properties such as mass, moment of inertia, and coefficient of restitution. It can also be used to configure the visual display, initial position, and the collision surface of a part.

3.6 Worlds

The graphical user interface (GUI) for Gazebo also allows for the creation of worlds. These worlds are the environment in which a simulation will be run. Worlds can be pre-built and loaded into the environment on startup. Pre-built worlds are included with the installation or found online and include anything from small workspaces to floorplans of entire buildings, such as the offices shown in figure 3.6.

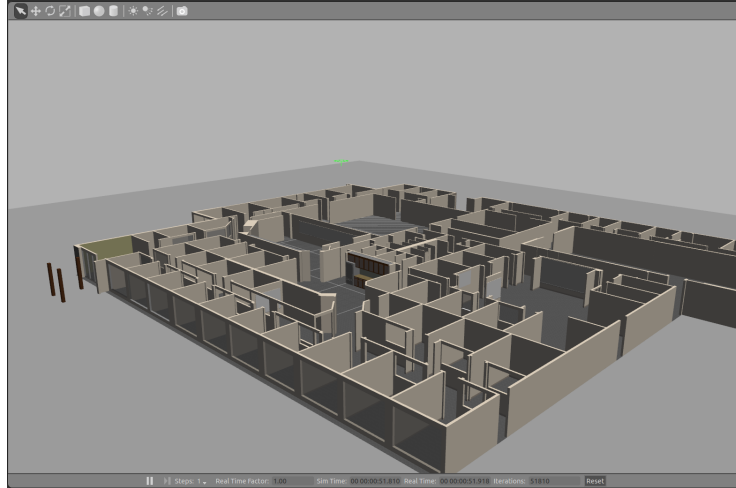


Figure 3.6: An office building model loaded from the default Gazebo library

Alternatively, a graphical user interface allows models to be built on the fly by the user. A library of models exists which can be dragged and dropped directly into the simulation. These models include objects such as traffic cones, concrete barriers, stationary robots and user-created 3D models which have been added to the directory. In addition, the GUI allows for lighting effects, positioning, creation of simple shapes, and viewpoint control. Once all these settings have been modified, the world can be saved and used in future simulations. Saving will automatically generate an XML file containing all the properties of the world. An example of how the properties are stored in the XML file is shown in appendix B.1. By manually editing this code, more advanced changes can be made to the world. These advanced changes include fine-tuning positioning and color settings which cannot be set in the properties window of the GUI.

Another advanced tool in creating worlds is the ability to generate terrain. For this

task, I used a heightmap. This is an image file in which the pixel colors represent a height value for the world at that point. Gazebo can use this file to generate a filtered ground surface with uneven elevation. This type of world is very useful for testing the 3D movement that a robot might undergo when testing outdoors.

Worlds are a crucial part of the simulation. They allow for modeling a wide variety of scenarios and verifying that the robots can operate successfully in them. Worlds can readily be changed to add new obstacles, scaled to give a larger working area, and modified to test out different robot configurations.

3.7 Husky Simulator

To simplify the process of simulation, a basic Husky robot simulator is provided by Clearpath. This simulator works by mimicking many of the ROS topics of the actual vehicle. It is configured to simulate Lidar, IMU, and GPS sensors. The Husky simulator includes a model for the base-level model of the Husky and can be controlled with commands to the motor controller much like the physical robot. Early work from this project tested the Husky simulator performing a basic dead-reckoning square in comparison to the actual vehicle running the same code [26].

This Husky simulator served as the base for the simulation work. It was cloned

and customized to better represent the vehicles we are working with and has been modified to allow for the simulation of multiple robots. These changes occur in the launch files for the simulator. The launch files are where all the information about the simulation is configured. The main areas for changes were to the robot model, the sensor plug-ins, and the robot namespace.

3.7.1 Model Modifications

Since the Husky simulator only provides the model for the base model Husky robot, some changes were made to reflect the actual look of the robot. This required a change to the Husky description URDF file, which identifies all the CAD models for the robot and links them together. The additional parts for our customized Husky were obtained from Clearpath and added to the model. The colors and properties were set to match our vehicles. Later, in order to differentiate between the different types of vehicles in the microgrid setup, the colors were changed for the robots. Yellow for source agents, red for bus agents, and green for cabling.

In the future, it will be possible to further modify these models to include various additional components such as the robotic arm, spool, and connectors for cabling robots, the electronics control board for bus robots, and a generator or solar panels for source robots. However, the basic goal of the demonstration, to show how the

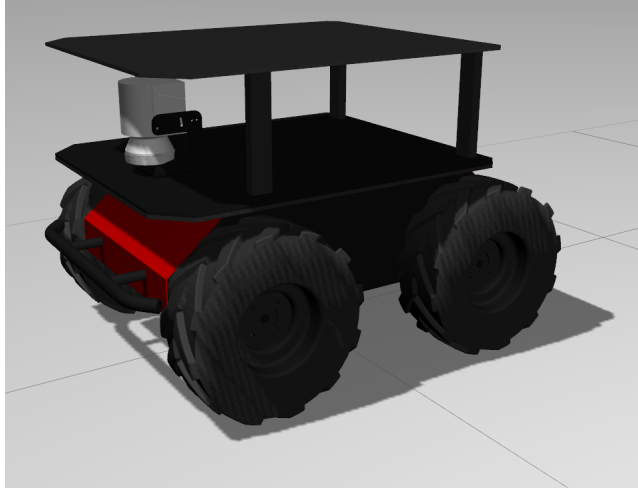


Figure 3.7: The modified Husky model for a bus agent robot

vehicles can navigate in the environment and collaboratively establish a microgrid, can be done without these additional components, so they have not been included. An example of a modified bus agent model is displayed in figure 3.7.

3.7.2 Sensor Simulation

Another difference between the default Husky simulation and our robots is with the sensors. Our vehicles use more advanced sensors with higher accuracy and different characteristics than the base model. This meant the sensors in the model had to be adjusted. Sensors are simulated through the use of plug-ins in the launch file. A particular sensor is imported into the robot description and then configured by a set of properties such as noise levels and reading frequency. I looked up these properties for the sensors described in chapter 2 and edited the plug ins to accurately reflect

them. To validate the results, sample data was recorded from both real and simulated sensors and the results were compared visually. Models for the GPS, lidar, and IMU have all been modified.

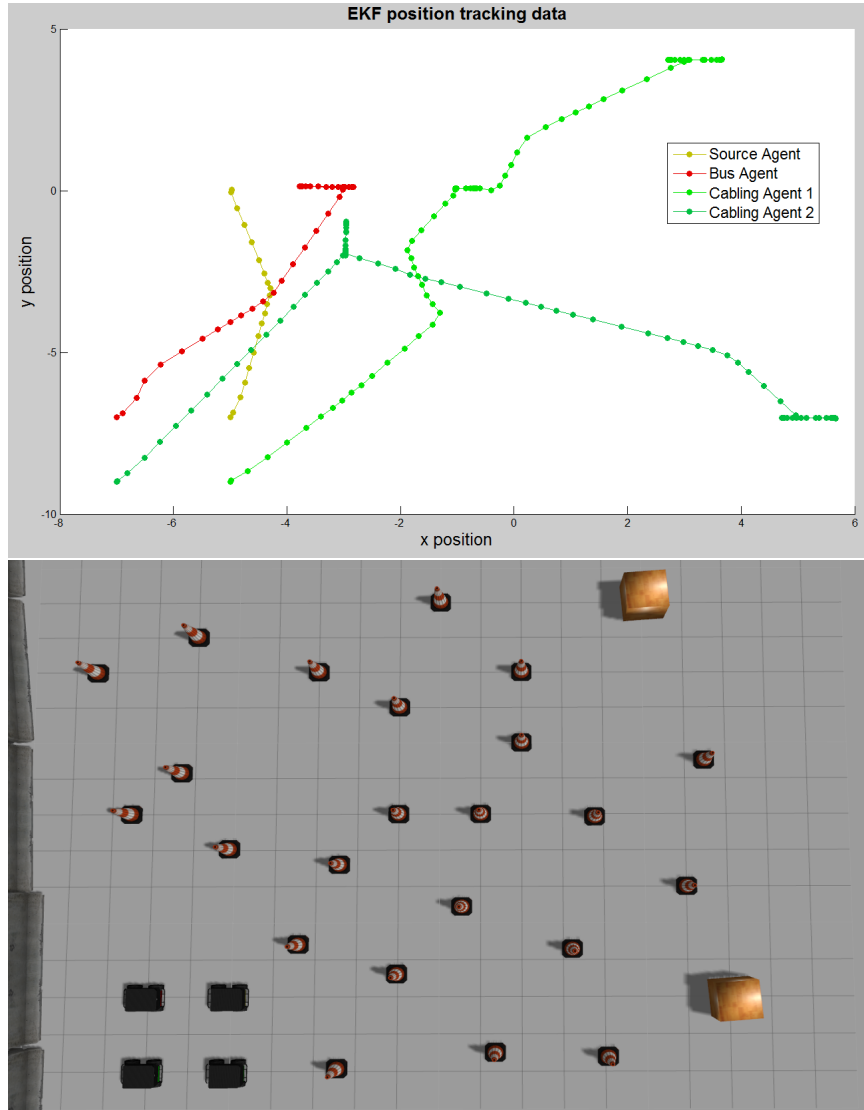


Figure 3.8: Sample data from the EKF showing the filtered tracking of four robots with reference to the experimental setup

Since the simulated GPS data alone has a wide range of variation to match the data

received from the actual GPS sensor, it cannot be relied on exclusively for determining the location of the robot. Early tests showed serious error in robot positioning when approaching a target. To counter these errors, a method of sensor fusion was used to combine data from multiple sensors for a more accurate estimation. This was done using an extended Kalman filter from the “robot_localization” package. With this package, data from multiple sources are combined and assigned a weight based on their respective covariance values. Over time, the Kalman filter gains change and the measurements supplement each other to combat sensor drift. For the simulation, combining data from the GPS, IMU, and odometry from the wheel encoders led to a much more accurate vehicle localization value. The tracking data from the GPS is shown in figure 3.8. In this plot, the paths of four robots moving in the workspace are shown with the world setup alongside for reference. The filtering smooths out the paths, compensating for noise in the GPS unit. This figure also illustrates how the robots behave in the simulation, with close groupings of points representing a robot stopping to make a connection. A similar EKF will likely need to be implemented on each actual robot for better positioning upon moving out of the lab setting and into real-world testing.

3.7.3 Multiple Robots

The largest task in modifying the Husky simulator to suit our needs was allowing it to simulate multiple robots simultaneously. The default simulator only creates one robot, located at the origin of the world. This virtual robot publishes and subscribes to a standard list of topics and tracks a standard set of transformations among the various components.

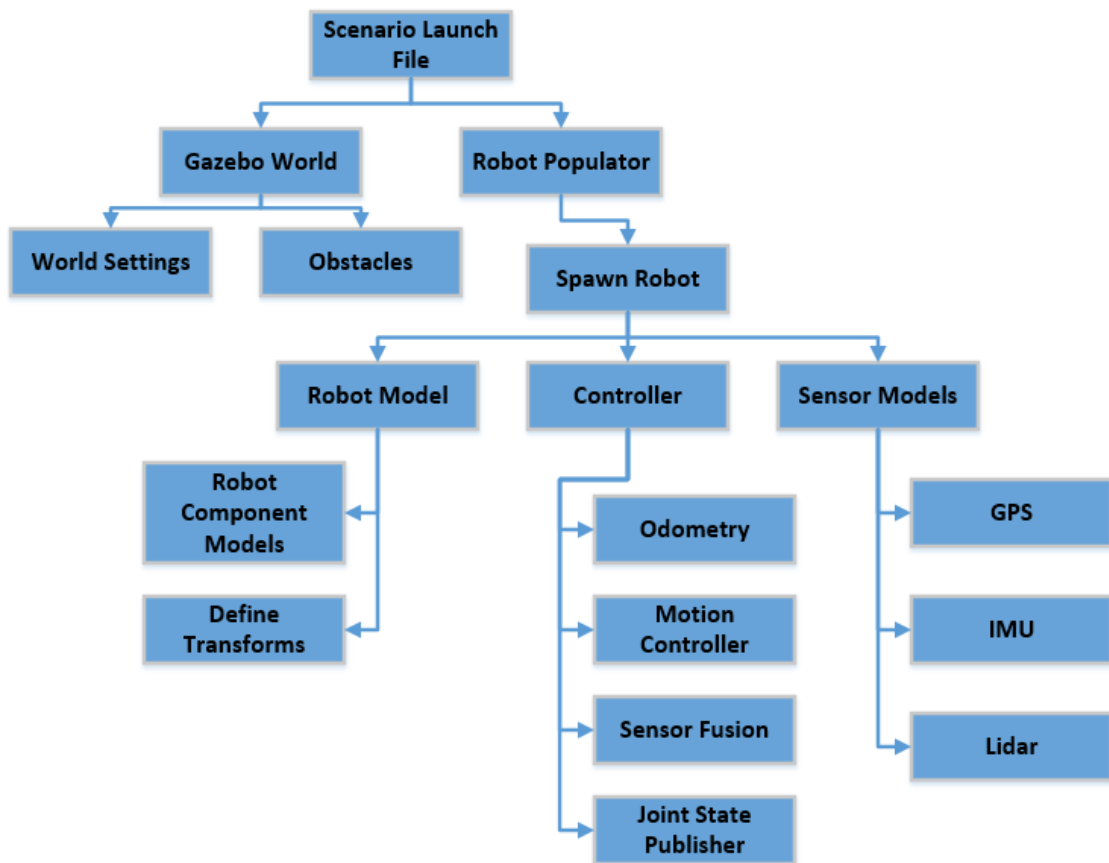


Figure 3.9: The structure and flow of the launch files used by the Husky simulator

To scale up the simulation to include multiple robots, several arguments had to be added to the launch file. Figure 3.9 shows the different levels of the launch structure. A scenario file starts both the world and the robot models, which consists of components for control, sensors, and the visual for the platform. When a launch file is called, it has a specific list of arguments which can be set by the calling file. These arguments can be passed down to subsequent levels and used to configure parameters for the robot. Within this structure, the changes start with the “robot populator.” This file was edited to call multiple instances of the “spawn robot” file, each with a unique set of parameters. The modified file can be found in appendix B.2.

The first addition was position arguments for each robot. These exist in the form of six degrees of freedom which have been added to the spawn file: translation in the x, y, and z directions as well as roll, pitch, and yaw, angles. The spawn file is what calls the robot description, plug-ins, and controller nodes and places them in the Gazebo environment. With the addition of position arguments, the robot can be placed at any specified starting location within the world. This means robots no longer start on top of one another, they can be set to spawn in a particular formation as if they were deployed to the field.

Next was the addition of a namespace. A namespace is essentially a wrapper for many ROS topics to keep them independent. Without this, the lidar on all robots would publish to the same “scan” topic, creating an interfering and unusable data

stream. The subscribed topics have similar problems. The “cmd_vel” which controls the the robots would be shared among all of them, so the robots would all move in the same way. The solution to this is an extension for the topics. For example, the lidar data topic becomes “huskyN/scan” and the controller is “huskyN/scan” where N is a unique number assigned to each robot. In this case huskyN is the namespace under which all topics for that particular robot are located. This argument is passed down to all launch files associated with creating topics and keeps all information separate. One additional difficulty of this method, however, is that the .xacro files which configure the sensor plug-ins cannot accept arguments. This means that each robot needs a unique description file to publish sensor data under it’s own namespace.

With robots being assigned starting positions and all the information being sent and received on unique topics, the infrastructure for multiple robots is almost complete, however, one last piece is needed so that robots know where they are located. Because all robots operate using the tf library, each piece of the model has identical frames, such as “odometry,” which describes how far the robot has traveled. Similar to the namespace problem, robots publish interfering transforms because they all use the same frames. Again, the solution is a wrapper on each of these frames called a Frame ID. Once this is added, the tf structure goes from a single tree with all robots writing to the same place to N identical trees which are kept separate from each other.

After changes to the model, sensors, and launch files, the simulation environment



Figure 3.10: The formation of multiple robots beginning a test in the modified simulation environment

is complete. With this in place, many realistic Husky robots can be simulated and testing can begin. An initial formation of many robots in this customized simulation environment is shown in figure 3.10

Chapter 4

Results and Conclusions

With all this underlying work completed in the software, we have accurate models, using accurate sensors, which can be created and run simultaneously in the same world. This means that useful, scaled-up scenarios can be tested in simulation. In addition, the infrastructure is in place to do physical testing as well. This chapter will focus on the parallel testing done both in simulation and the real world, a discussion of the results of that testing, and what this means for the future of an autonomous microgrid project.

4.1 Scenarios

This section summarizes the different worlds used for testing the simulation environment. These worlds were carefully selected for missions to validate the environment and to test the limits of what could be done in simulation. The worlds are described below.

† **The Husky playpen.** This is a simple world with a variety of different obstacles that allows basic development and testing to be done. It is a pre-built map that allows testing of obstacle avoidance techniques.

† **A simulated version of NAS Lab.** This is a model of our workspace at Michigan Tech, where early testing of the Husky vehicles was done. With both the simulation and real demo working in a similar environment, I was able to ensure that the simulation was behaving properly. This was very valuable in fine-tuning simulation parameters like sensor noise and vehicle speed.

† **Uneven terrain.** Since most of the previous development has been done on very flat surfaces, it is important to test in an environment where this was not the case. Hills and elevation changes introduce a number of new challenges and the goal of this world is to help us to understand and deal without the need for as much outdoor testing.

† **Empty microgrid configuration.** This world serves as a larger-scale parallel to a video demonstration of connecting a microgrid in our lab. In this video, two robots connect to blocks and then to each other. This represents cabling robots making a connection between a source and a load to connect power. The empty microgrid world serves the same purpose, but it is larger than our lab-space, so it allows four robots to operate simultaneously.

† **Obstacle avoidance load testing.** This world consists of 8 different robots and a field of obstacles. There are no simulated sources or loads, so the goal of the robots is to navigate to the opposite side of the field while avoiding obstacles and each other. The purpose of this scenario is to test the abilities of the robots when navigating in large numbers. This world is also used to test how scaling in the number of robots affects simulation performance. An added benefit is that the robots can be tested in an obstacle-dense environment with other robots acting as moving obstacles.

† **Diverse agents test.** This is a set of simulations to test the robot performance for each type of robot (cabling, bus, or source) operating in its own role. On the simplest level, one robot of each type is deployed. The source robot moves to a suitable location to provide power while the bus agent and cabling agent link it to a load. This represents a single unit of the microgrid. For a mid-level demonstration, a second load and cabling agent is shown. This scenario would represent a single source with the bus agent distributing power among

multiple loads in a unit. Finally, a larger scenario shows an interconnection between multiple units. A second source and bus agent are added as well as a third cabling agent for a total of seven robots. The additional agents establish another source to load unit and then the two units are combined through a connection between the buses. This interconnection between multiple sources and loads shows the true value of an autonomous, reconfigurable microgrid. With the proper control, power could be directed, distributed, and stored in a robust way to ensure reliable connections.

4.2 Results

Each of the scenarios described above was set up in the simulation environment to be tested and the results and observations of each test are outlined in this section. Where applicable, these simulated results are compared to physical validation using the architecture developed in chapter 2 to verify the accuracy of the simulated environment. In addition, each section will include the lessons learned and any changes which had to be made to the simulation or navigation algorithms as a result.

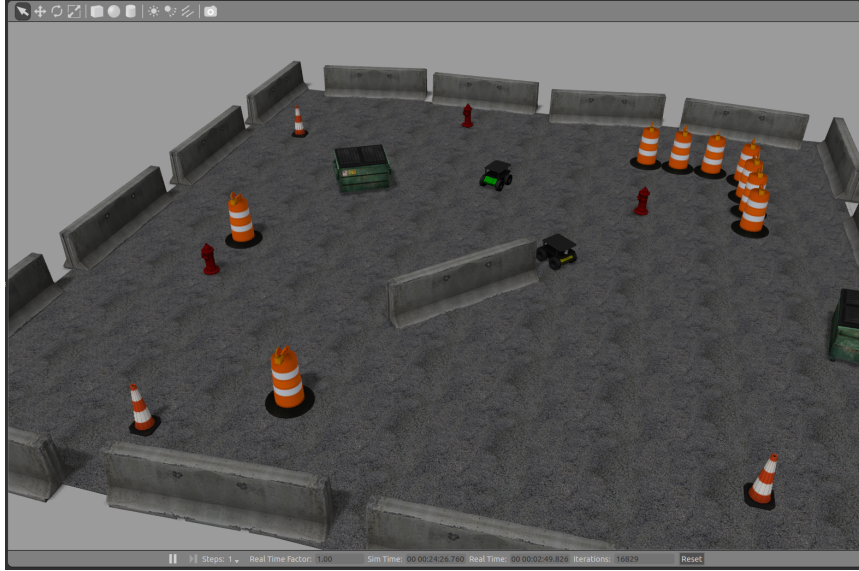


Figure 4.1: Two robots avoiding obstacles in the Husky playpen world

4.2.1 Development

The Husky playpen scenario was used to modify existing obstacle avoidance code to fit both the real and virtual Husky robots. The world contains a variety of different obstacles and scenery which are useful for observing how the software responds. This world was also used as the primary example-case when changing the simulation files to allow for multiple robots. As a result, much of the debugging work was completed in this scenario. One early observation that resulted from this testing was that that robots needed an initialization point at the mission start or the localization could not be completed. Previously, all robots assumed a start position and angle of 0. A set of parameters needed to be added to the launch file to describe the location of each

robot relative to the others.

By the end of the simulation, two robots were able to run simultaneously, avoiding obstacles and each other while traveling between a set list of points. Figure 4.1 shows this world populated by two robots during a test for the local path planner.

During tests in this environment, there were occasions when the vehicles would briefly bump one another while passing. This is a result of the robots executing a turn command when an obstacle enters the space beside the rear wheel. This location is a blind spot for the lidar sensor which controls avoidance. This suggests that for future, larger scale vehicles, a more complete suite of sensors will be needed to close such gaps in the perception system. Even with these measures in place, it will always be necessary to exercise caution when approaching an autonomously operating vehicle.

Another problem which occasionally arose during testing was the tendency for the robot to swivel back and fourth between headings while nearing the corner of an obstacle. An analysis of the Lidar data and the local path planner in these cases showed that the desired heading rapidly switches between points at the corner of the obstacle and the point in the horizon beyond the obstacle. This occurs until the robot eventually moves far enough around the corner to pass the barrier. The cornering performance is made slower, but this does not cause the mission to fail.

4.2.2 Replication of previous work

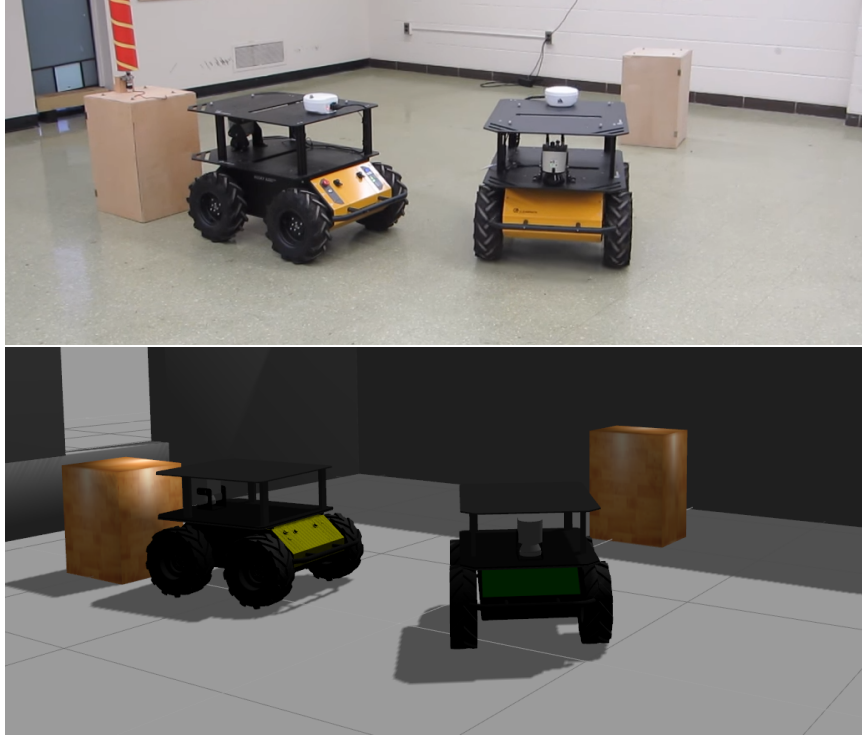


Figure 4.2: Comparison of a real lab test to the simulated lab workspace

The simulated model of NAS lab was used to demonstrate that the new simulation environment could replicate the results of the in-lab testing that was also being done. An example of this testing is shown in a video released by NAS Lab [27]. Figure 4.2 shows a comparison of the same connection software running in both the lab and virtual environments.

This development environment allowed for a direct comparison between the behavior of the real robots to those used in simulation. Testing went quite smoothly and

showed the simulated robots could complete the same path as the real ones on the first try. This means that before clearing space in the lab, setting up the network, and pushing code to the robots, the capability now exists to conduct lab tests in simulation first and perfect them, saving time during real-world demonstrations.

The scale model of our workspace also helps in another aspect of scaling. It allows for tests to be performed with realistic GPS data rather than the extremely accurate camera system positioning. Tests have shown that with the addition of sensor fusion through the use of an extended Kalman filter, the positioning will still be accurate enough to achieve similar results when moving tests outdoors.

4.2.3 Uneven Terrain

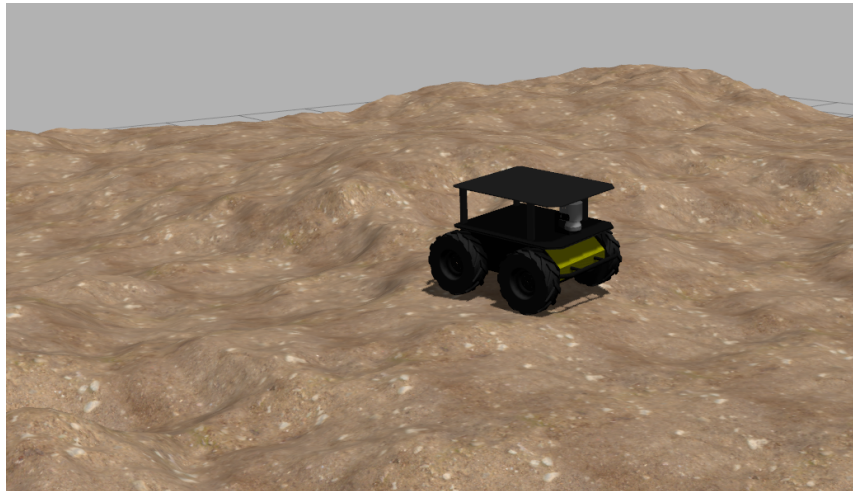


Figure 4.3: A robot navigating the uneven terrain of a world created from a heightmap

Once the GPS localization code was complete, the Husky robots could be taken outside for the first time. However, this test uncovered some problems that were not seen in the lab or previous simulations. The robots were observed to move erratically and often get stuck before reaching their destinations. An analysis of the recorded data during these outdoor tests showed that the problem was due to the terrain. Since the lab and simulation environments both used perfectly flat areas of operation, the lidar sweep was always parallel to the floor. But in the real world, the robots would often tilt forward and interpret the ground as a large obstacle with no path around it. To assist with tackling this problem, a world was created using randomly generated uneven terrain. Figure 4.3 shows a simulated robot placed on this setting.

However, testing showed that simulation for these cases was not practical. Even in a small world with a single robot and no obstacles, the simulation was too slow to offer any help. Simulations for these cases ran at about 3 percent of real time. For testing uneven ground, a real world trial in off-road terrain was usually preferred to simulation. This result highlights the need for simultaneous development of both simulation and real-world tests. Each has advantages in certain areas and disadvantages in others. By using them in combination, we can make sure to capture the best of both worlds.

An analysis of computational resources during the rough terrain simulation showed the limiting factor to be processing speed. This is because the complex geometry of

the ground means a lot of calculations for the physics engine even to perform simple tasks such as movement or calculating lidar readings. In the future, it may be possible to continue work in this area using one of several possible solutions.

† One method for analyzing rough terrain is to simulate it with simpler models.

Very short, wide cylinders could be added to the world file which would be tall enough to tilt the robot when driving over them. These simple shapes would simplify calculations while still replicating the problem of lidar reflections off the ground.

† Another possible method for improving speed would be to run the simulation on a more powerful computer. The simulation could possibly be adapted for use on a supercomputer where processing power is vastly improved. Simulations could be completed much faster and with much more complexity.

† A third way that simulation of a mobile microgrid on rough terrain could potentially be more successful is by using different simulation settings. It is possible that through software updates or changing solver settings to a lower resolution, Gazebo may be able to handle the physics within a more realistic amount of time.

Simulation of rough terrain is an area which will surely need more development in the future. But for this stage in development, lessons about operating in this type of

environment are left to the results of testing on the actual robots. Based on the results from early outdoor testing, a solution to this problem is currently being developed for more smooth operation. This solution will likely incorporate data from the IMU to correct for tilt and limit the range of the lidar readings accordingly.

4.2.4 Connection Scaling

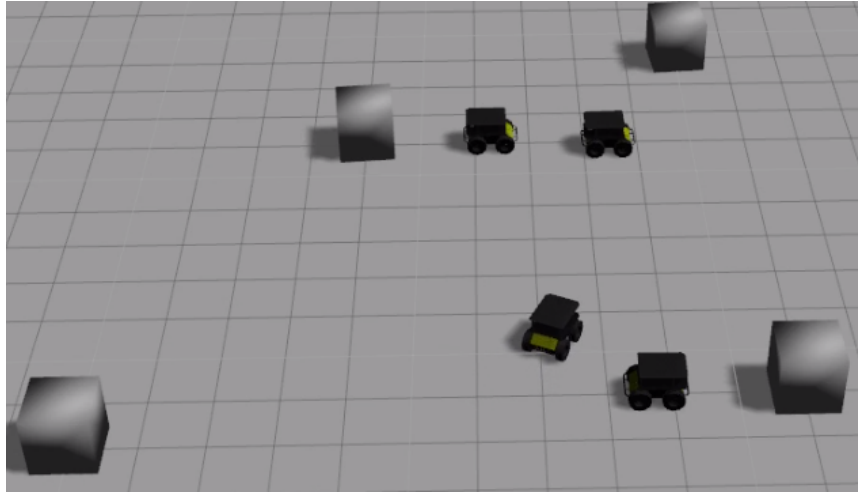


Figure 4.4: A scaled up version of in-lab testing using four robots

Scaling in number of robots has been one of the most successful applications so far for the simulation environment. The first of these tests was a simple, open-world environment with only sources and loads, as seen in figure 4.4. This scenario specifically tests the global path planning level of the autonomy algorithms. Testing is done in an open world with no obstacles except for other robots and source or load blocks. Vehicles are assigned the task of connecting to each of the blocks and

then to each other to simulate the connection-making process of a microgrid setup. This simulation is the next step up from the simulation of the virtual NAS lab which demonstrated the same result with only two robots.

The results from this scenario are used to show that robots can locate each other, as well as targets when making connections in a virtual setting. It shows how many different nodes in the microgrid can be connected together. This scenario was useful in evaluating the impact of initial positioning of the robots. It can be used to test what deployment patterns for the robots will allow all connections to be completed in the shortest amount of time.

4.2.5 Load Testing

In the next scenario, shown in figure 4.5, the limits of the computer simulation were examined. For this test, eight robots were deployed surrounding a relatively obstacle-dense world. Each robot was tasked with navigating to the other side of the world while running the full navigation software. Changing the number of robots in the simulation allows an analysis of how the number of robots affects simulation speed.

For this test, it is important to note a difference between the simulation environment and a real world test. For simulation, both the plant and the controller must be modeled. The plant is the Husky robot virtualized using Gazebo and the controller

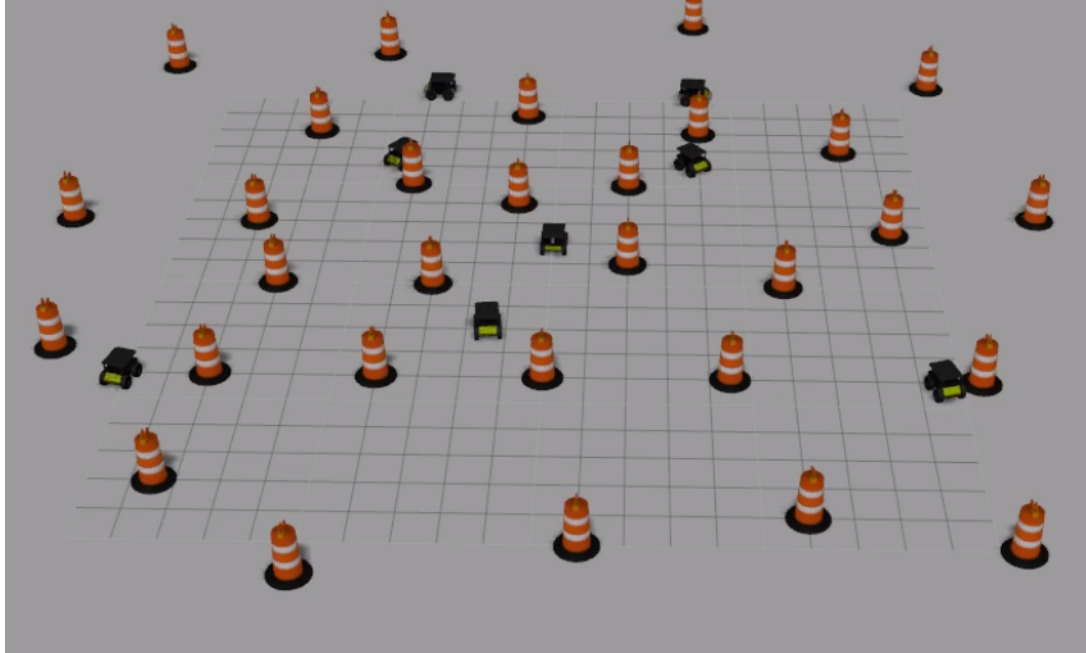


Figure 4.5: A test of obstacle avoidance using eight robots

is the actual python code running the path planner. In a real world test, each robot contains its own computer which is used to run the controller. However, in simulation, a single machine must run the controller for all robots as well as the simulation environment. This means that for an eight robot simulation, there are eight full path planners, and eight virtual robots sending and receiving data. Having all of these components running on a single machine is very computationally intensive. This becomes evident when looking at the data in table 4.1.

The table shows the simulation speed as compared to real-time based on the number of robots in the simulation. For a two robot case, the simulation runs essentially in real time. However, the speed decreases considerably as more robots are added. With eight robots, a five minute demonstration would take nearly 18 minutes to run in

Table 4.1

A timing table for the simulation speed during load testing. The first column represents the number of robots simulated. The second column identifies whether the controller was running on the same computer as the simulator or outsourced to a different machine. The third column displays the average speed of the simulation as a percentage of real-time throughout the duration of the test.

Number of Robots	Path Planning	Simulation speed (as a percent of real-time)
2	On Machine	98
4	On Machine	67
8	On Machine	28
8	Outsourced	71

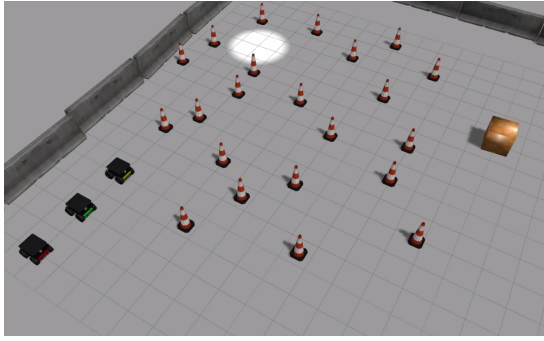
simulation. To alleviate some of the stress on the computer’s processor, I leveraged the intranet network used in the lab. I ran the simulation on one computer and broadcast the ROS information to another computer which runs the controller for all robots. The last line of the table shows that with the path planning outsourced to a second machine, the simulation speed increased by a factor of more than two.

This demonstration of the ability to combine computers for better simulation is an important result. It means that if additional computers are added to simulate control algorithms, the scalability of simulation for mobile microgrids can be further improved. So far, the scaling of our microgrid system simulation is only limited by available processing power.

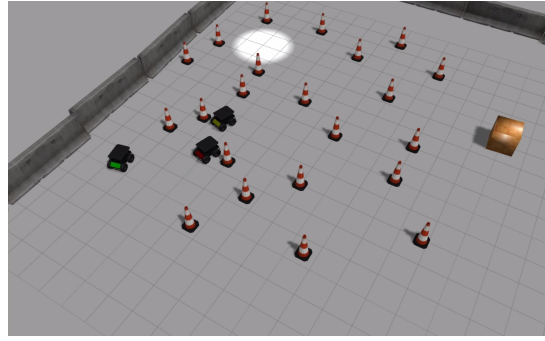
4.2.6 Diverse Agents

All the previous pieces have come together for a full demonstration of the scenario discussed in the first chapter of this thesis. For this simulation, each of the robots is defined as a different type of agent. Source agents are modeled as yellow robots, which navigate to a fixed location and stay there for power generation as a robot with a solar panel array might do. Red bus agents follow closely behind and connect to these sources, then act as management systems to regulate power. Finally, green cabling agents act to connect these buses to loads and to one another.

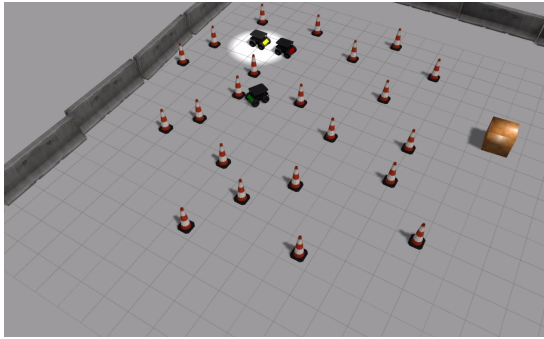
Figure 4.6 shows the steps of the microgrid construction process using one of each of the three types of agents. In 4.6(a), the robots have been deployed to the field of operation and are waiting in formation for navigation to start. Cones represent obstacles in the path of the robots, the block to the right of the field is a load in need of power, and the white spot in the upper left indicates a suitable location for the source agent. The robots then begin moving in 4.6(b). They navigate together toward the desired source location while avoiding obstacles. In 4.6(c) the source agent has arrived and the bus agent is connecting with it. Next, the cabling agent connects to the bus agent in 4.6(d) and begins toward the load in 4.6(e). Finally, in 4.6(f), the cabling agent connects the load to the rest of the microgrid, completing the setup. Now the power can be activated and managed by the bus agent. This has been the



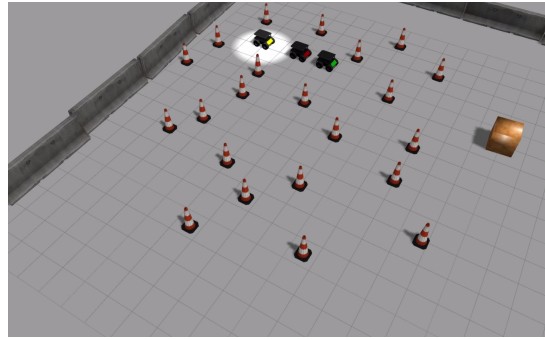
(a) Deployment



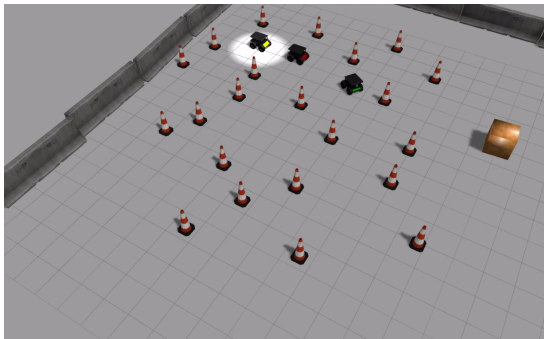
(b) Navigation starts



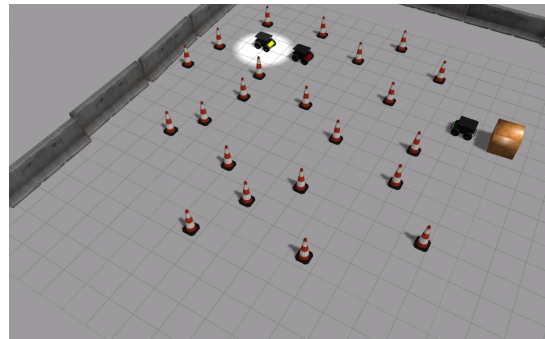
(c) Source agent in position



(d) Cabling to bus agent



(e) Running cable



(f) Completion

Figure 4.6: Time-lapse images of three agent microgrid setup simulation

first example of diversification in robot types for an autonomous mobile microgrid where the source, bus, and cabling agents are considered to function differently.

The time-lapse shows the simplest configuration of a microgrid with only one of each



Figure 4.7: A large-scale demonstration which assigns robots unique roles

type of agent. However, the grid size can be expanded to power more loads through the addition of more agents. The scenario pictured in 4.7 shows the start of a seven robot demonstration. This demonstration displays another key characteristic of a scaled microgrid setup: it represents an interconnection of units in the microgrid. A unit is considered to be a source-bus-load connection. For this demo, multiple sources are combined with multiple loads, all on the same electrical network. This means that if one unit runs low on power, it can be supplemented by the rest of the grid so that all critical loads can stay active.

4.3 Conclusions and Future Work

This work has shown that a microgrid with multiple units and specialized agents is possible in simulation, the tools for further development are in place. The next step of the process is to bring all aspects that have been completed so far into a single demonstration. This demonstration should include a real-world portion where physical connections are completed using a robotic arm and managed using an electrical control board. All the hardware components will come together to show a successful electrical grid being set up autonomously both in a lab setting and in the real world. These same demonstrations can be simulated in the software along with expanded simulations which show what could be accomplished with a larger number of vehicles and more diverse environments.

The work outlined in this thesis has put the infrastructure in place for advancing to such a demonstration. These tools will provide a lasting way to continue development on autonomous mobile microgrids as the technology improves and will eventually be useful for transitioning these systems into real-world use in the field.

While simulations are currently capable of handling eight robots, larger numbers of robots result in increasingly slow simulation speeds. Further investigation into different solvers in the simulator, computational resources, and compatibility with

supercomputers may result in the ability to simulate far larger microgrid scenarios as well improve simulation of off-road environments.

Further development can also be done on simulators for different types of vehicles. Microgrid systems will not be limited to just ground-based platforms. A similar approach could be used with underwater vehicles, aerial vehicles, or any combination. For these possibilities, simulation tools must also be created to account for robots operating in a fluid.

Another potential area for expanding the capabilities of the simulator would be to allow for the testing of connections. A more complex simulation environment could feature an electrical connector and robotic arm to extend the scenario beyond navigation to include the task of making electrical connections as well.

References

- [1] Microgrid regulatory policy in the us. November **2014**.
- [2] Moridian, B.; Bennett, D.; Mahmoudian, N.; Weaver, W.; Robinett, R. In *Dynamic Systems and Control Conferences*, 2015.
- [3] Moridian, B.; Bennett, D.; Mahmoudian, N.; Weaver, W.; Robinett, R. In *Safety, Security, and Rescue Robotics, 2015 IEEE International Symposium on*, 2015.
- [4] Eckenstein, N.; Yim, M. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1227–1233, 2014.
- [5] Automated vehicle crash rate comparison using naturalistic data. January **2016**.
- [6] Weaver, W.; Mahmoudian, N.; Parker, G. In *TARDEC Ground Vehicle Systems Engineering and Technology Symposium, Dearborn, MI*, 2012.
- [7] Lasseter, R. In *Power Engineering Society Winter Meeting, 2002. IEEE*, Vol. 1, pages 305–308 vol.1, 2002.

- [8] Gostin, L. O.; Friedman, E. *Lancet* **2014**, *384*, 1323–1325.
- [9] Kwasinski, A.; Weaver, W. W.; Chapman, P. L.; Krein, P. T. Sept **2009**, *3*(3), 277–287.
- [10] Kanas, N. In *Humans in Space*; Springer, 2015; pages 109–117.
- [11] Dimeas, A. L.; Hatziaegyriou, N. D. Aug **2005**, *20*(3), 1447–1455.
- [12] Moridian, B. **2014**.
- [13] Moridian, B.; Bennett, D.; Mahmoudian, N.; Weaver, W.; Robinett, R. In *The International Federation of Automatic Control, The 19th World Congress*, 2014.
- [14] Montemerlo, M.; Thrun, S. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, Vol. 2, pages 1985–1991 vol.2, 2003.
- [15] Husky unmanned ground vehicle. Clearpath. **2016**.
- [16] 2d laser scanners: Lms1xx / lms15x / outdoor. Sick. **2016**.
- [17] Smart6-l. Novatel. **2016**.
- [18] 3dm-gx3 -25. Sensing, L. **2016**.
- [19] Automatic charging station for autonomous mobile machine. September 30 **2008**.
- [20] Kartoun, U.; Stern, H.; Edan, Y.; Feied, C.; Handler, J.; Smith, M.; Gillam, M. In *Automation Congress, 2006. WAC '06. World*, pages 1–8, 2006.

- [21] Silverman, M. C.; Nies, D.; Jung, B.; Sukhatme, G. S. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, Vol. 1, pages 1050–1055 vol.1, 2002.
- [22] Eckenstein, N.; Yim, M. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2846–2851, 2014.
- [23] Nilsson, M. Dec **2002**, 7(4), 473–474.
- [24] Nagy, Z.; Abbott, J. J. In *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*, pages 1–6, 2007.
- [25] Widowx robot arm kit mark ii. Robotics, T. **2016**.
- [26] Scaled up microgrid by naslab. Beyers, N. **2016**.
- [27] Navigation system for autonomous mobile microgrid. Moridian, B. **2016**.

Appendix A

Validation Code Samples

This section provides samples of code developed for both the path planning and validation of the autonomous microgrid system.

A.1 SerialSample.py

This code provides a simple example for testing a programmatically controllable output pin. It uses the python “serial” library to power a pin at alternating high and low voltage for five seconds. This was used in testing along with a multimeter to show that the vehicles could activate an external power board.

```

#Copyright      2016 Nonlinear and Autonomous Systems ↵
    Laboratory
#Michigan Technological University

#Author(s): Nathan Beyers
#Created: 2016-09-06


import sys, serial
from time import sleep

#configure port and baud rate
COM_PORT = 1
BAUD = 115200

#connect to rs232 port
ser = serial.Serial(COM_PORT-1, BAUD, timeout=0.5, ↵
    rtscts=0)

#Toggle the DTR pin on for 5 seconds, then off for 5 ↵
    seconds.
time = 0
while (while time < 5) :
    print "Voltage: High"
    ser.setDTR(1)
    sleep(5)
    print "Voltage: Low"
    ser.setDTR(0)
    sleep(5)
    time = time + 1

```

A.2 GPSLocalization.py

This code was created to replace the localization information from the overhead camera system with GPS and IMU data. Based on the initial location of the robot, the GPS data is mapped to an odometry measurement relative to the initial position. This data is then fed into the program and combined with compass data from the IMU to determine a position in 2D space. This position, which include x and y coordinates as well as yaw angle, is used by the mission planner to direct the robot to its target.

```
#!/usr/bin/env python

#Copyright: Nonlinear and Autonomous Systems Laboratory
#Michigan Technological University

#Author(s): Nathan Beyers
#Created: 2015-9-30

import rospy
from sensor_msgs.msg import NavSatFix, Imu
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Pose2D
import tf
from math import pi

class gpsLocalization(object):
```

```

def __init__(self, namespace, odomTopic, imuTopic, ←
    xOffset, yOffset):
    rospy.init_node(namespace + 'gpsLocalization') #start←
        the control node

    rospy.Subscriber(odomTopic, Odometry, self.←
        updatePosition)
    rospy.Subscriber(imuTopic, Imu, self.updateRotation)

    self.rate = rospy.Rate(10) #rate at 1 Hz

    self.pose_msg = Pose2D() #create a 2D pose message ←
        and initialize values
    self.pose_msg.x = float('nan')
    self.pose_msg.y = float('nan')
    self.pose_msg.theta = float('nan')

    self.ang_msg = Pose2D() #create a 2D pose message ←
        and initialize values
    self.ang_msg.x = float('nan')
    self.ang_msg.y = float('nan')
    self.ang_msg.theta = float('nan')

    self.roll = 0
    self.pitch = 0
    self.yaw = 0

    self.xOffset = xOffset
    self.yOffset = yOffset

def getStates(self, num):
    # get position
    agent_id = 0 #Change for multiple agents

```

```

        x = self.pose_msg.x + self.xOffset
        y = self.pose_msg.y + self.yOffset
        z = 0
        yaw = self.pose_msg.theta
        pitch = 0
        roll = 0
        return agent_id, x, y, z, yaw, pitch, roll

def updatePosition(self, data):
    self.pose_msg.x = data.pose.pose.position.x
    self.pose_msg.y = data.pose.pose.position.y

def updateRotation(self, data):
    quaternion = [data.orientation.x, data.orientation.y,
                  ,data.orientation.z, data.orientation.w]
    euler = tf.transformations.euler_from_quaternion(←
        quaternion, axes = 'rzxy')
    self.pose_msg.theta = euler[0] #+3*pi/2

if __name__ == "__main__":
    loc = gpsLocalization()

```

A.3 environment_scan.py

This code shows the operation of the basic mission planner for the system. A goal is defined and objects for the path planner and path tracker are created. Additionally, sensor inputs are configured and subscribed to. The mission planner guides the robot

toward this path and calls the path planner upon each new lidar scan. The path planner determines the best course, given nearby obstacles, and calls the path tracker to throttle speed and drive the robot in this direction.

```
#!/usr/bin/env python

#Copyright: Nonlinear and Autonomous Systems Laboratory
#Michigan Technological University

#Author(s): Nathan Beyers
#Created: 2015-9-30

import rospy
from numpy import arctan2, sqrt, arange, pi, sin, cos, ←
    save
from sys import path, exit, argv
path.append('/modules/')
from gpsLocalization import gpsLocalization
from sensor_msgs.msg import LaserScan
from actuate import ROS2DimActuate
from tracker import PlanarTracker
from time import sleep, time
from path_planning import GapFinder

every_other = 3
increment = pi * .5 / 180
angles = arange(-3*pi / 4, 3*pi / 4 + increment, ←
    increment)[0::every_other] #Simulated husky
kp = .4 / 1
kd = .3
```



```

log_length = 4096
log = [[]] * log_length
i = 0
stage = 0
finished_logging = False
temp_var = 1
temp_var_2 = temp_var * log_length

class Navigation(object):

    def __init__(self):

        #set parameters
        self.gap = .7 #space needed to pass through
        self.agent_id = 0
        self.topicConfig()

        self.connection = gpsLocalization(argv[3],self.↵
            gpsTopic,self.imuTopic)    #Connection which ↵
            will give current position
        self.path_planner = GapFinder(self.gap)    #Finds ↵
            gaps that the robot can enter
        self.actuation = ROS2DimActuate(self.↵
            controlTopic)    #Controls the motion of the ↵
            robot
        self.actuation.setAngularVelocityLimit(.5)    #↵
            Sets the maximum velocity
        #Create a tracker which knows how to move the ↵
            robot and get it's position
        self.tracker = PlanarTracker(self.actuation.↵
            actuate, self.connection.getStates)
        #Tell the tracker which robot to command

```

```

self.tracker.setID(self.agent_id)

self.distance = []
self.prev_closest_reading = 0.0
self.prev_time = time()
self.crash_avert_velocity = 0.0

print 'Starting the Navigation'
sleep(2)
self.subscriber = rospy.Subscriber(self.↵
    lidarTopic, LaserScan, self.move, queue_size↵
    =1) #Call move for each laser scan

rospy.spin()

def topicConfig(self):
    if len(argv)>1:
        self.lidarTopic = '/' + argv[3] + '/scan'
        self.gpsTopic = '/' + argv[3] + '/odometry/↵
            filtered'
        self.imuTopic = '/' + argv[3] + '/imu/data'
        self.controlTopic = '/' + argv[3] + '/↵
            cmd_vel'
        self.target_x = float(argv[1])
        self.target_y = float(argv[2])

    else:
        self.lidarTopic = '/scan'
        self.gpsTopic = '/navsat/enu'
        self.imuTopic = '/imu/data'
        self.controlTopic = '/cmd_vel'
        self.target_x = 5 #destination coordinates
        self.target_y = 4

```

```

def move(self, data):
    agent_id, x, y, z, yaw, pitch, roll = self.↵
        connection.getStates(self.agent_id) #Get ↵
        localization info
    #print 'x: ', x,'y: ', y,'theta: ', yaw

    print yaw
    print '-----'
    global i
    global stage
    global finished_logging
    distances = list(data.ranges)[0::every_other] #↵
        store the range readings from the lidar
    self.path_planner.filterReadings(distances, ↵
        angles) #filter the results

    closest_reading, closest_reading_angle = self.↵
        path_planner.getMinimumReading()
    closest_reading = min(closest_reading, 2 * self.↵
        gap)
    time_now = time()
    self.crash_avert_velocity = (self.↵
        crash_avert_velocity + (closest_reading - self.↵
        .prev_closest_reading) * kd / (time() - self.↵
        prev_time)) / 2
    self.crash_avert_velocity = min(0.0, self.↵
        crash_avert_velocity)

    controlled_velocity = (closest_reading) * kp + ↵
        self.crash_avert_velocity
    controlled_velocity = max(0.0, min(↵
        controlled_velocity, 1.0))

```

```

self.actuation.setTangentialVelocityLimit(min←
    (.2, controlled_velocity))

i += 1
if i % temp_var is 0 and i < temp_var_2:
    log[i / temp_var] = [x, y, yaw, self.←
        path_planner.readings_polar]
print self.target_x, x
print self.target_y, y
diff_x = self.target_x - x
diff_y = self.target_y - y
self.distance = sqrt(diff_x**2 + diff_y**2)

if self.distance < .1:
    stage += 1
    print 'ARRIVED!!!!!!!!!!!!'
    if finished_logging is False and i >= ←
        temp_var_2:
            self.tracker.saveLog()
            save('loginfo', log)
            finished_logging = True
    self.target_y = self.target_y * -1
    self.target_x = self.target_x * -1
    exit()

angle = arctan2(diff_y, diff_x) - yaw
subgoal_distance, subgoal_angle = self.←
    path_planner.planPath(self.distance, -angle)
subgoal_angle2 = -subgoal_angle

self.tracker.moveToDynamicPoint(←
    subgoal_distance, subgoal_angle2)

self.prev_closest_reading = closest_reading

```


Appendix B

Simulation Code Samples

This section provides samples of code developed for the simulation of the autonomous microgrid system.

B.1 NASLab.world

This code demonstrates a small portion of the world file generated for the virtual representation of the Nonlinear and Autonomous Systems laboratory. The full file is thousands of lines, so only a small section is displayed here. This code provides the first steps to creating a bookshelf in the simulation environment. First, the shelf is defined as a link with given transformation data so that Gazebo can keep track of its

position. Next, the collision surfaces are defined for the physics engine and properties such as mass are assigned. Finally, a visual is created which will be displayed for the user. This visual defines a set of simple shapes which make up the shelf and gives them an imported pattern to look like wood.

```
<model name='bookshelf'>
  <pose>-5.47897 -4.4727 0 0 -0 3.12502</pose>
  <link name='link'>
    <pose>-5.47897 -4.4727 0 0 -0 3.12502</pose>
    <velocity>0 0 0 0 -0 0</velocity>
    <acceleration>0 0 0 0 -0 0</acceleration>
    <wrench>0 0 0 0 -0 0</wrench>
  </link>
</model>
<model name='bookshelf'>
  <static>1</static>
  <link name='link'>
    <inertial>
      <mass>1</mass>
    </inertial>
    <collision name='back'>
      <pose>0 0.005 0.6 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.9 0.01 1.2</size>
        </box>
      </geometry>
      <max_contacts>10</max_contacts>
      <surface>
        <bounce/>
        <friction>
          <ode/>
```



```

        </friction>
        <contact>
            <ode/>
        </contact>
    </surface>
</collision>
<visual name='visual1'>
    <pose>0 0.005 0.6 0 -0 0</pose>
    <geometry>
        <box>
            <size>0.9 0.01 1.2</size>
        </box>
    </geometry>
    <material>
        <script>
            <uri>file://media/materials/scripts/gazebo.↵
                material</uri>
            <name>Gazebo/Wood</name>
        </script>
    </material>
</visual>
...

```

B.2 husky_empty_world.launch (Modified)

This XML code shows the modified launch file which populates the Gazebo world with multiple Husky robots. Initially, several parameters are set such as configuration options for the robot sensors. Next, properties of the simulator are configured and

the location of the robot model file is defined. After this, three Husky robots are spawned. Each of these robots is given a custom namespace, transform prefix, and initial position. These values are passed down to all subsequent launch files.

```
<?xml version="1.0"?>
<!--
```

Software License Agreement (BSD)

```
\file      husky_empty_world.launch
\authors   Paul Bovbel <pbovbel@clearpathrobotics.com, ↵
          Devon Ash <dash@clearpathrobotics.com>
\copyright Copyright (c) 2015, Clearpath Robotics, Inc., ↵
          All rights reserved.

-->
<launch>
```

```
<arg name="world_name" default="worlds/empty.world"/>

<arg name="laser_enabled" default="true"/>
<arg name="ur5_enabled" default="false"/>
<arg name="kinect_enabled" default="false"/>

<include file="$(find mtu_gazebo_ros)/launch/↵
  empty_world.launch">
  <arg name="world_name" value="$(arg world_name)"/> <↵
    !-- world_name is wrt GAZEBO_RESOURCE_PATH ↵
    environment variable -->
  <arg name="paused" value="false"/>
  <arg name="use_sim_time" value="true"/>
  <arg name="gui" value="true"/>
```

```

    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
</include>

<param name="robot_description" command="$(find xacro)↵
    /xacro.py '$(find mtu_husky_gazebo)/urdf/↵
    mtu_description.gazebo.xacro '
    laser_enabled:=$(arg laser_enabled)
    ur5_enabled:=$(arg ur5_enabled)
    kinect_enabled:=$(arg kinect_enabled)
    " />
<!--Spawn Huskies in their respective Namespaces-->

<group ns="husky1">
    <param name="tf_prefix" value="husky1" />
    <include file="$(find mtu_husky_gazebo)/launch/↵
        mtu_multi_spawn_husky.launch">
        <arg name="laser_enabled" value="$(arg ↵
            laser_enabled)"/>
        <arg name="ur5_enabled" value="$(arg ur5_enabled)"↵
            />
        <arg name="kinect_enabled" value="$(arg ↵
            kinect_enabled)"/>
        <arg name="x" value="-5.0"/>
        <arg name="y" value="-7.0"/>
        <arg name="namespace" value="husky1"/>
    </include>
</group>

<group ns="husky2">
    <param name="tf_prefix" value="husky2" />
    <include file="$(find mtu_husky_gazebo)/launch/↵
        mtu_multi_spawn_husky.launch">

```

```

    <arg name="laser_enabled" value="$(arg ←
        laser_enabled)"/>
    <arg name="ur5_enabled" value="$(arg ur5_enabled)"←
        />
    <arg name="kinect_enabled" value="$(arg ←
        kinect_enabled)"/>
    <arg name="x" value="-5.0"/>
    <arg name="y" value="-9.0"/>
    <arg name="namespace" value="husky2"/>
</include>
</group>

<group ns="husky3">
    <param name="tf_prefix" value="husky3" />
    <include file="$(find mtu_husky_gazebo)/launch/←
        mtu_multi_spawn_husky.launch">
        <arg name="laser_enabled" value="$(arg ←
            laser_enabled)"/>
        <arg name="ur5_enabled" value="$(arg ur5_enabled)"←
            />
        <arg name="kinect_enabled" value="$(arg ←
            kinect_enabled)"/>
        <arg name="x" value="-7.0"/>
        <arg name="y" value="-7.0"/>
        <arg name="namespace" value="husky3"/>
    </include>
</group>

</launch>

```

Appendix C

Letters of Permission



Conor Walsh

Apr 7 (4 days ago) ☆



to me ▾

Hi Nathan,

Go for it! Send us a copy when you finish.

Good luck!

On Wed, Apr 6, 2016 at 4:48 PM, Nathan Beyers <ntbeyers@mtu.edu> wrote:

Hello,

My name is Nathan and I am currently working on a Master's thesis at Michigan tech. My work is involving microgrids and I really like the image you use to explain them in the article: "Microgrid Regulatory Policy in the US" on your website

I am hoping to include this image in my thesis to help explain the components of a microgrid. Would this be possible?

Thank you,

--

Nathan Beyers

Michigan Technological University
B.S. Mechanical Engineering

--

Conor Walsh

Figure C.1: Permission letter for figure 1.1

4/6/2016

Michigan Technological University Mail - RE: ASME Digital Collection Feedback -- permission for DSCC2015 paper



Nathan Beyers <ntbeyers@mtu.edu>

RE: ASME Digital Collection Feedback -- permission for DSCC2015 paper

Beth Darchi <DarchiB@asme.org>
To: Nathan Beyers <ntbeyers@mtu.edu>

Wed, Apr 6, 2016 at 11:54 AM

Dear Mr. Beyers,

It is our pleasure to grant you permission to publish the ASME **Figure 3 only** from "Design of Mobile Microgrid's Hierarchy for Power Distribution," by Barzin Moridian, Daryl Bennett, Nina Mahmoudian, Rush Robinett and Wayne W. Weaver, Paper No. DSCC2015-9866, cited in your letter for inclusion in a thesis entitled Simulation of Scalability for Autonomous Mobile Microgrids to be published by Michigan Technological University (ProQuest).

Permission is granted for the specific use as stated herein and does not permit further use of the materials without proper authorization. Proper attribution must be made to the author(s) of the materials. As is customary, we request that you ensure proper acknowledgment of the exact sources of this material, the authors, and ASME as original publisher. Acknowledgment must be retained on all pages printed and distributed.

Many thanks for your interest in ASME publications.

Sincerely,



Beth Darchi

Publishing Administrator

ASME

2 Park Avenue, 6th Floor

New York, NY 10016-5990

Tel: 1.212.591.7700

darchib@asme.org

Figure C.2: ASME permission letter for figure 1.2



Title: Robotic power distribution system for post-disaster operations

Conference Proceedings: 2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)

Author: B. Moridian; N. Mahmoudian; W. W. Weaver; R. D. Robinett

Publisher: IEEE

Date: 18-20 Oct. 2015

Copyright © 2015, IEEE

LOGIN

If you're a [copyright.com](#) user, you can login to RightsLink using your [copyright.com](#) credentials. Already a [RightsLink](#) user or want to [learn more?](#)

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

Copyright © 2016 Copyright Clearance Center, Inc. All Rights Reserved. [Privacy statement](#). [Terms and Conditions](#). Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Figure C.3: IEEE permission letter for figure 1.3



Title: Area of acceptance for 3D self-aligning robotic connectors: Concepts, metrics, and designs
Conference Proceedings: Robotics and Automation (ICRA), 2014
Author: N. Eckenstein; M. Yim
Publisher: IEEE
Date: May 31 2014-June 7 2014
Copyright © 2014, IEEE

LOGIN
If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials. Already a RightsLink user or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

Copyright © 2016 Copyright Clearance Center, Inc. All Rights Reserved. [Privacy statement](#) [Terms and Conditions](#).
Comments? We would like to hear from you. E-mail us at customer@copyright.com

Figure C.4: IEEE permission letter for figure 2.8