ROBOT-OBJECT INTERACTION LANGUAGE

by

Christin Danelle Shelton

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 04, 2012

Keywords: robot, object, interaction

Abstract

As the world anticipates receiving helper robots into their homes there are still a few more hurdles that the robotics field must leap over in order for us to reach such a goal. The main issues that the discipline faces are the snowflake robot designs (no two are alike) and the large learning curve for robot programming. These issues create a difficulty and an inconsistency in design and usability. To resolve these issues, a language structure has been designed to communicate with the programming language of any robot enabling the robot to manipulate any object. Thus, the hypothesis submitted in this proposal states that any robot is capable of interacting with any real world object to the robots optimality and within its limitations.

Acknowledgments

I must first thank the Lord God Almighty for trusting me to take on such a challenge. He has given me strength to accomplish great things in Jesus' name. He is my ultimate and eternal partner, friend and Savior. I would also like to thank my committee and other faculty members for their advice, flexibility and encouragment. I would never be able to list all of my friends here, but ones who stand out are Drs. Wanda Eugene and Shanee Dawkins who refused to leave me behind. Thank you for you prayers and all the work you've done to get me to the finishline. All of my other friends know who they are and you know what you've done. Thank you. As for my family, there is no way that I could ever begin to thank you. You have been a light in a dark dark world. To my Uncle Victor: thank you for being my inspiration. My sister who is still praying for me this very moment and is on call and ready to encourage at a drop of a hat. Her children, who are like my own and love me just for being their "TiTi". My mother who words could never combine together to begin to be enough to mean or give a true thanks. Thank you for telling me that I didn't have to do this but that I could. Thank you for teaching me to identify myself in Christ and not in academia. And last but never least (in fact he's second to God) is my loving husband. Thank you for your support and your patience especially since I finished when we should have been in our honeymoon stage and just relaxing together. Thank you for loving me through this. God sent you to me at the perfect time.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The world has been anticipating the day that robots will become a major part of our everyday lives. The service and entertainment industries are two fields on which robots will have a significant impact (Makatchev & Tso, 2000). Researchers are eagerly designing robots that will be able to provide effective in-home care to the disabled and the elderly (Edsinger & Kemp, 2006, 2007, 2008; C. C. Kemp, Anderson, Nguyen, Trevor, & Xu, 2008; Nguyen et al., 2008). The movie *I, Robot*, featured an elderly woman who had a live-in robot assistant (Figure 1.1). Her robot was able to recognize and interact with all of the worlds objects both inside and outside of her home. Now, imagine a paraplegic who drops her remote control or needs a book on an unreachable shelf. Given a service robot, it should be able to locate and identify the object and know how to appropriately handle it. As technology advances, we are sure to discover an optimal robot design that will allow them to attend to us all.

Regarding entertainment, RoboCup (shown in Figure 1.2) is a 16-year-old initiative that joins Artificial Intelligence and Robotics in an effort to promote research and education. RoboCup is an international conference and competition in which the participants take robots and program them to operate as soccer players. The robots are programmed to compete by searching for, acquiring and passing a small colored ball back and forth, and hopefully into a goal. RoboCup has included Search and Rescue events since 2000, and it aspires to play the ultimate robotics soccer game featuring life-size humanoid robots. Although its underlying motivation is not entertainment, RoboCup events and tournaments provide entertainment for both its participants and spectators (A Brief, n.d.).

Figure 1.1: *Granny and her helper robot in I, Robot.*

Figure 1.2: *RoboCup soccer tournament.*

Whether the focus of development is service or entertainment, robotics technology is widely accessible for relatively low-budget research, particularly for primary to graduate education (Stone, 2007). There are many manufacturers that build robots for development (Lapham, 1999; Wörn, Wurll, & Henrich, 1998), but research labs are building more and more robots in-house to perform and test for their particular needs. Because of this vast accessibility, the growth of the robotics industry has been exponential. The designs of robots have infinite possibilities. No two robot designs will be exactly alike, even more different than the humans who design them.

Before robots can become a part of our everyday lives, consistency is needed among the various robot technologies. Research has been conducted that enables robots to interact with particular real world objects (e.g., grasping, fetching, finding, etc.). Research also indicates that robots are able to safely exchange and pass objects back and forth with humans (Edsinger & Kemp, 2007). Domo, an assistive humanoid robot created for passing objects, particularly tools, is shown in Figure 1.3. Kemp (2008) designed and built El-E, a robot for object manipulation research that uses a green laser to select a number of objects on a level surface (e.g., floor, table, etc.). The robot detects the laser and is directed to the objects location, and calculates how to grasp the object and carry it to another place or a person (C. C. Kemp et al., 2008). These robots are examples of different designs and capabilities. Despite these differences, there should be one method enabling each for interaction via programming, also allowing each robot to have the same manipulation potential. Based on this prior research, one can state that robots are able to appropriately interact with various objects (i.e., pick up a book, set a cup down) within its own limitations. However, this has been proven difficult due to the lack of consistency in robot programming technology for object manipulation.

In addition to the barriers of design and development in assistive robot technology, the learning curve for robot programming is an issue, as well (Stone, 2007). Manufacturers all use different programming languages, syntaxes, and structures, requiring programmers to be

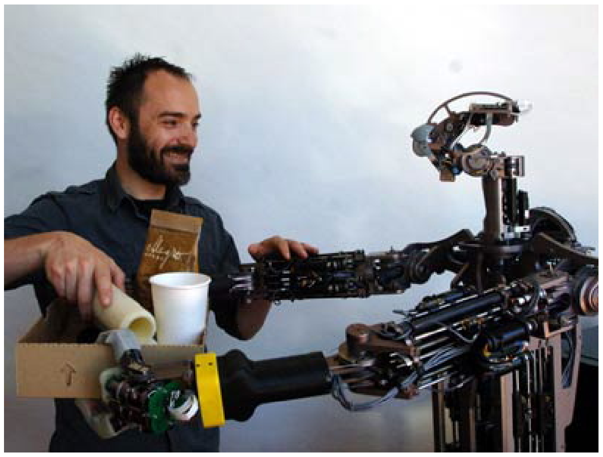


Figure 1.3: *Domo, a robot that passes objects with humans.*

proficient in many, or specific, languages. Prior research has determined that using computer programming languages that have sustained over time, such as C and Java, has decreased the learning curve and made robot programming easy to learn due to more computing power and cheap memory. Learning the commands and their use makes any type of programming difficult to learn, and mastering any programming language is a considerable time investment (Lapham, 1999). Students of robotics can find a large learning curve frustrating, which may cause them to avoid robotics in the future (Anderson, Thaete, & Wiegand, 2007). Essentially, there is a need to diminish this learning curve and allow any robotics novice to program any robot to interact with any object without having to learn a new robot programming language for each robot. The following section will affirm the goals, designed approach, and planned contributions of the language grammar and system presented in this dissertation research.

1.2 Overview of Research Goals and Contributions

The chief goal of this research is to establish a standard system that will automate code that will create the opportunity for any robot to appropriately interact with any real-world object. The submitted hypothesis states that any robot is capable of interacting with any real world object that is within its limitations. It is anticipated that the findings of this research will support and evolve the design and development of robot manipulation for any environment.

The results presented here are expected to contribute to the fields of Human Robot Interaction, Robot Programming Languages, and Robot Manipulation.

Overall, this research will achieve the following:

- Establish a standard automated robot language production system
- Diminish the learning curve for robot programming for objects in any environment
- Improve robot programming capabilities

1.3 Research Problem

Robots today come in many shapes, sizes, and forms (Wörn et al., 1998; Lapham, 1999). However, the world is still waiting for a mainstream robot to penetrate the home as the modern day PC that will assist in various roles. Robot manipulation has been proven to be one of the more difficult areas of the robotics industry as research has yet to deliver a robot that may serve as a helper for the elderly and disabled or for the purposes of entertainment. Due to the exponential and perpetual growth of the industry, new technological advances not only produce aesthetically new designs, but also new functions, capabilities, and platform designs and language (Stone, 2007). This has created the problem of each robot design having a unique platform and language. Much of the research in the Human-Robot Interaction (HRI) field is dedicated to robot autonomy in a human's everyday environment; however, because of the lack of design standards, each robot must have a unique method of interaction. The learning curve has become great and time-consuming for both robotics novices and experts (Wörn et al., 1998).

Researchers recognize a need for a common or standard language to make robot programming easier (Lapham, 1999). Some focus their robotics research on manipulation or designing and building a robot intended for precise set of tasks. However, there is a lack of research conducted to design and develop a method that will assist robot programming in relating to any real-world object.

Most robots that are capable of manipulation are restricted by the tasks for which they were developed. This means that if a robot needs to interact with new objects, the programmer must go in and manually program or script each new object each time a new object is presented. Even if an algorithm is used to detect and recognize unknown objects there is still a limited set of shapes that are detectable and manipulation will not be optimized for each object. Furthermore, should a new programmer join the research team, she must know and perhaps learn the particular programming language that is being used on each individual robot.

The problem being addressed is the lack of a standard method that discovers whether or not a robot is appropriate for any real-world object and produce code that will make the objects accessible. This study will support the hypothesis that a standard architecture that can discover a robot and its capabilities will be the basis of a system for interaction with objects. For the purposes of this study, a small, diverse subset of simulated objects will be held by a database and exported into the grammar structure. The website will output a block of code that corresponds to the language on which the robot operates.

1.4 Organization of the Research

The organization of this dissertation will proceed following a research agenda. Chapter 2 reviews the background information that relates and analogizes the system of this dissertation to the areas of robot programming, robot manipulation, plug and play networks. Chapter 3 will provide details of the system design and implementation and includes a scenario to give further understanding of how to use the system. Chapter 4 follows with the disclosure of experimentation of the system mentioned in Chapter 3. The results of the experiments performed are discussed in Chapter 5. The summary and conclusions of Chapter 5 are in Chapter 6 along with the contributions made by this work and issues for future work.

Chapter 2

Literature Review

This chapter identifies two areas of robotics technology that explore the aspects of getting a robot to do what you want through design and development. Presented here are various robots and languages that best demonstrate the technologies of robot programming and robot manipulation.

2.1 Programming Languages

This section of the review intends to introduce the understanding the difficulty of learning a new high-level programming language to be succeeded with a discussion of programming languages that specifically support robotics technology. With any language, and particularly mainstream programming languages (e.g. C++ and Java), there is said to usually be quite a steep learning curve (Katz & McCormick, 2000). David Brooks (1999) offers advice for students who want to succeed in a programming class. With many books that are written to teach a programming language, no matter the level programmer, there is usually some prerequisite of knowledge that the write expects has already been attained by the reader and determining the most useful background involves understanding the history and ancestry of each language and comparing the syntax and concepts of each. As a result, Brooks (1999) advises that the student find out what their textbook assumed about the reader. For example, it is suggested that knowledge of Windows programming will help a programmer pick up Visual C++ more quickly (DelRossi, 1993) and C/C++ programmers will more easily acquire C# and Java (Mueller, 2009; Grimes, 2001). On the contrary, Visual Basic programmers will find it difficult to grasp C# due to the varying syntactical styles and fundamental concepts (Mueller, 2009; Simon et al., n.d.). Likewise, programmers of Java, C++ or C#

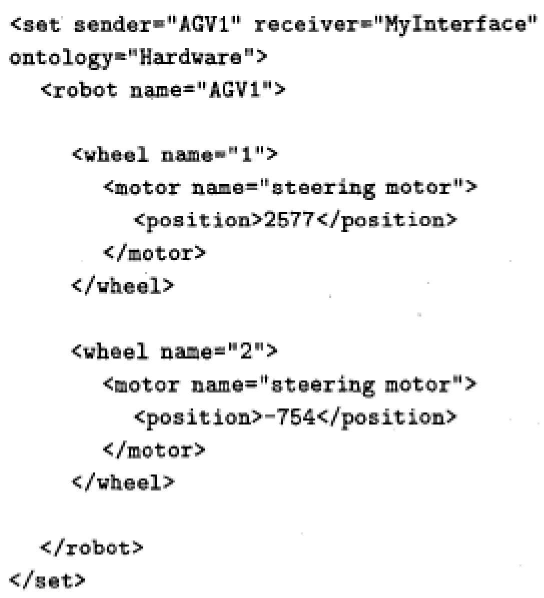
will conceptually relate to Python, but syntactically will struggle with the new language (Smith, 2011). Therefore, experience helps to pick up other language. While those with experience may struggle with a new language, true novices will have the most difficulties.

Other factors involved in the learning curve include time. Time involves the learning of the syntax and nuances of the new language (Pappas & Murray, 1995). Additionally, programmers must learn to use a number of tools and techniques to implement the new language (Warth, 2011), as well as, understand the environments help, warning, and error messages (Pappas & Murray, 1995). The following section will cover various programming languages that have been written to support robots and robotics technology. It will further establish these issues in programming languages specifically for robotics.

2.2 Robot Programming Languages

2.2.1 RoboML

RoboML (or Robo Markup Language) is a human-robot interface software prototype that utilizes Extensible Markup Language (XML) (see Figure 2.1) as communication language for robot agents. This agent-based interface is meant to act as a common language for "robot programming, agent communication, and knowledge representation (Makatchev & Tso, 2000). RoboML recognizes the need for open standards with regards to robot hardware and software. It sought to answer the need by facilitating communications between real-time agents (user clients) and embedded software and user interface agents via a proxy agent. Its focal points are issues of human-robot interfacing using the Internet: 1) an agent-based architecture and 2) a common markup language (Makatchev & Tso, 2000). Makatchev (2000) chose XML to develop RoboML due to its aptness for demonstrating what can be conveyed by the more common languages for programming robots, its usefulness for human users to manipulate using simple software and its availability for cross-platform applications. It is well believed that XML is a convenient language for describing various data structures



```
<set sender="AGV1" receiver="MyInterface"
ontology="Hardware">
  <robot name="AGV1">

    <wheel name="1">
      <motor name="steering motor">
        <position>2577</position>
      </motor>
    </wheel>

    <wheel name="2">
      <motor name="steering motor">
        <position>-754</position>
      </motor>
    </wheel>

  </robot>
</set>
```

Figure 2.1: *Snippet of RoboML code; XML structured*

(Makatchev & Tso, 2000). The work on the formation of a tentative language is decreased with XML (Makatchev & Tso, 2000).

2.2.2 RobotScript

The Universal Robot Controller (URC) is an open-architecture controller that uses a Windows platform. The URC works with most robots and increases flexibility and capabilities over other typical controllers. The operating system for the URC is Windows NT, which is an enterprise-wide network connected to the robot controller by the user for logging data, backing up programs, and handling other tasks for communication. The creators of the URC believe it is the idyllic stage for a common programming language for robots and that RobotScript is that language. RobotScript is meant to allow a single language to command all of the robots in a factory (Lapham, 1999).

RobotScript was developed over the course of a number of steps. The first relevant step was to decide the purpose of the programming language, which was originally to command a robot and enable the user to design user interfaces or correspond with other software

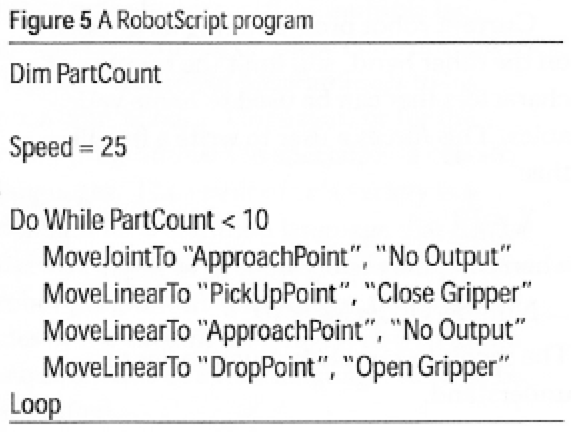


Figure 5 A RobotScript program

```
Dim PartCount

Speed = 25

Do While PartCount < 10
    MoveJointTo "ApproachPoint", "No Output"
    MoveLinearTo "PickUpPoint", "Close Gripper"
    MoveLinearTo "ApproachPoint", "No Output"
    MoveLinearTo "DropPoint", "Open Gripper"
Loop
```

Figure 2.2: *Snippet of Robotscript code; VBScript-like*

applications. RobotScript presents a solution that requires programming proficiency in a new language. Its creator, Lapham, desired that the language be effortless and yet function as a typical programming language for robots that only determined the motion, which is essential to the robots path (Lapham, 1999). Next, it was decided that RobotScript would be a robot library for a pre-existing computer language. Microsofts Visual Basic Scripting Edition, or VBScript (see Figure 2.2), was selected for its easy-to-learn syntax and interpreted compiler. The library must match the syntax in VBScript (Lapham, 1999).

2.2.3 Urbi

Urbi is a universal platform created to provide all robots with software compatibility, establishing a standard way for the reuse of components from one robot to another. Gostai, the company that created Urbi, created this platform to tackle the demands of Artificial Intelligence and robot programming for autonomous robots. Two major focal points of the Urbi platform concentrate on flexibility and simplicity. According to Gostai, flexibility is being universal, working with any robot on any OS with any programming language. Simplicity requires that the platform be easy to acquire for all experts, kids and hobbyists as well as have as little documentation as possible (Gostai, n.d.).

“Urbi is a middleware which includes component architecture” called UObject. Urbi presents a new scripting language, urbiScript, to handle parallelism and events in robot programming. It is interfaced through liburbi with languages like Java, C++ (see Code samples in Listings 2.1 & 2.2), Ruby, Matlab, Python, and others. This is accomplished in urbiStudio, which holds a number of graphical programming tools. Urbi is used by various research labs and companies; however, there is the learning curve to consider. One still needs to learn one or more specific programming languages in addition to the new language that Urbi introduces.

Listing 2.1: Urbi Basic Function Definition; C++/C-like

```
// Define myFunction
function myFunction()
{
  echo("Hello world");
  echo("from my function!");
};
[00000000] function () {
  echo("Hello world");
  echo("from my function!");
}

// Invoke it
myFunction();
[00000000] *** Hello world
[00000000] *** from my function!
```

Listing 2.2: Urbi Return Function Definition; C++/C-like

```
function sum(a, b, c)
```

```
{
  return a + b + c;
};
[00003553] function (var a, var b, var c) { return a. + (b). + (c) }
function sum2(a, b, c)
{
  return a + b + c;
}|;
sum(20, 2, 20);
[00003556] 42
```

2.2.4 Robot Manipulation

Whether at home or at work, we desire that robots be capable of "physically altering the world through contact" (C. Kemp, Edsinger, & Torres-Jara, 2007). Most research in robot manipulation has occurred in various controlled environments, e.g., industrial plants and uncluttered research areas. Within those environments, successful demonstrations of research robots autonomously performing complicated manipulation tasks have relied on some combination of known or simplified objects, fixed or organized environments, or "narrowly defined, task-specific controllers" (C. Kemp et al., 2007).

Merriam-Websters (Merriam-Webster.com, n.d.) online dictionary defines 'manipulate' as "to treat or operate with or as if with the hands or by mechanical means especially in a skillful manner". In keeping with the spirit of this definition, for the purpose of this hypothesis, manipulation, or appropriate interaction, is interpreted as gripping and carrying objects according to the robots affordances. The limitations of robots today enable a robot to only grip and pick up a cell phone but not to accurately press the buttons to make an effective call. Any mention of a robot manipulation is restricted to its end-effectors or grippers (Figure 2.3).

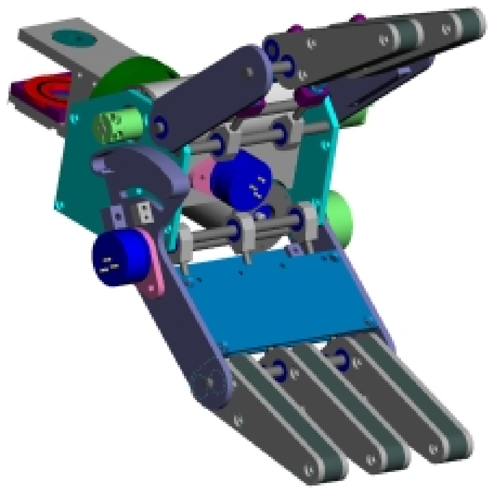


Figure 2.3: *An example of a gripper.*

Robot designs for manipulation often carry several restrictions. Robots are often designed to carry out fixed tasks, e.g., search and rescue and object passing. Therefore, their end-effectors are rarely at their optimum usage and restricted to the precise objects they were designed to interact with. The issue here is if a programmer or developer chooses for her robot to interact with any additional objects she will have to make changes to her program manually.

El-E, a robot shown in Figure 2.4, makes use of a green laser pointer that highlights the object the robot is to detect, locate, grasp, and hand-over to the user. El-E acquires an object and places it upon a flat surface, e.g., tables, floors, and bookshelves. Flat surfaces are found throughout human environments due to the need for people to randomly place objects on top of their things (C. C. Kemp et al., 2008). After acquiring the object, the El-E can set the object on a surface appointed by the laser above the floor, then trail the laser along the floor, or carry it to a designated, seated person. Should the user misplace or, perhaps in the case of a disabled user, drop the laser pointer, the robot is rendered useless. The laser interface hinders the addition of modalities, such as speech and touch recognition because the robot is restricted to finding the green light. Subsequently, due to limitations of the laser interface and the eliminated possibility of any other modality, if the desired object

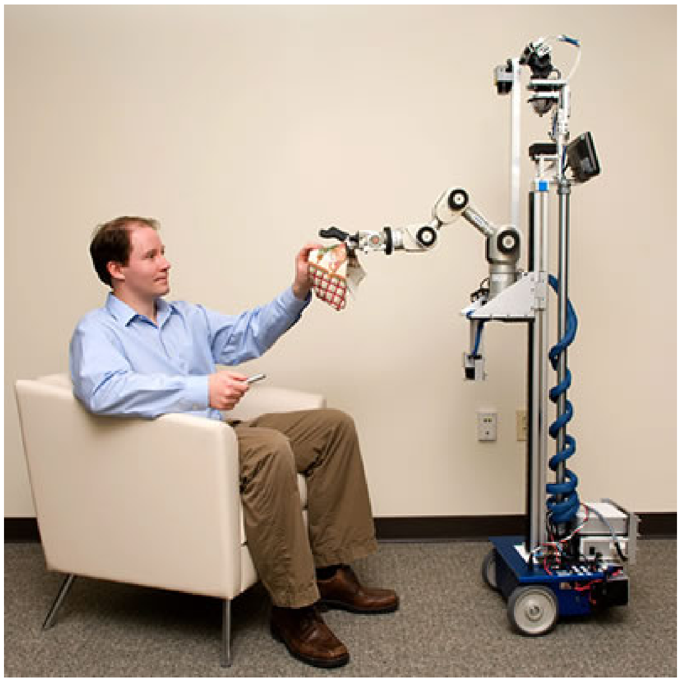


Figure 2.4: *El-E, a robot that delivers objects with a laser interface.*

is in another room, the user must go to the room with the robot in tow and sit down, if not already seated.

There are a couple of questions concerning object manipulation that need to be addressed. What set of objects will be used for research purposes or what objects does the robot need to interact with? How are known objects to be recognized? If the task requires interaction with additional objects, how does the robot recognize and know how to interact with the unknown? In the book The Design of Everyday Things, Norman (1990) states that objects in human environments have similar design features that make using them easier. Figure 2.5 is a well-known image of an example of an unusable or at least difficult-to-use design. Edsinger and Kemp (2006) found that manipulation is simplified by developing behaviors that suit those features. They show that the handling of a substantial set of tools can be determined by the tools tip (Edsinger & Kemp, 2006). However, exact autonomous grasping of what once was a foreign object still remains a challenge (Saxena, Driemeyer, Kearns, Osondu, & Ng, 2006). In order for a robot to acknowledge and manipulate any new

Figure 2.5: *An unusable everyday object (teakettle).*

object each object must be scripted by hand (Saxena et al., 2006). Often, object recognition requires the incorporation of an algorithm. Saxena (2006) uses a supervised learning algorithm to predict how to grasp new objects through vision. When deciding which objects to use, typically, the robots purpose or its current task will definitively make that determination. However, when the research calls specifically for a robot to learn many various types and shapes of objects, a training set may be required (Saxena et al., 2006).

Currently, this research does not support physically plugging in a robot machine so that the hosting computer is able to discover information about it and transfer and receive data to and from it.

2.3 Universal Console Overview

This section serves strictly as an analogy to the approach of the proposed study in this paper. The science of Plug and Play and the applications mentioned, the Universal Remote Console and the Personal Universal Controller, represent research and devices of comparative architectures. In addition to similar formats, the following consoles also employ XML to structure their languages.

2.3.1 Plug and Play

In the history of data processing technology hardware modules were assembled and linked by wires for various calculating operations. This process was complex often requiring soldering and wire cutting for configuration changes and it was most frequently used by companies that processed large amounts of data. However, due to the permeation of the personal computers throughout the general public there was pressure to automate the configuration of these devices for use by those who are less skilled with the wire connection techniques. After several attempts at self-configuration among various companies, Microsoft released Windows 95, which dbuted an attempt at fully automated device detection and configuration called Plug and Play (PnP). Briefly, the implementation of plug and play has three basic requirements. The operating system being used must have handlers in support of plug and play media that finish the process of configuration through the BIOS started for each device. The BIOS, the core utility of the plug and play process, reads a file that has the information about the installed PnP devices called the Extended System Configuration Data (ESCD). The BIOS enables the plug and play and detects the devices (Grabianowski & Tyson, 2001).

”Some bus types such as Peripheral Component Interconnect (PCI) and Universal Serial Bus (USB) take full advantage of Plug and Play” (Microsoft, 2003). Today, interfaces like USB are now thought of as a common method of connection for peripheral devices (Computer Desktop Encyclopedia, 2008). As it is quite similar to URC (expounded upon in the following section) Universal Serial Bus is the realized dream of any person who has ever used and personal computer and desired a quick and easy way to link all of their accessories (e.g. scanner, printer, digital camera) to it (Universal Serial Bus, n.d.). Connecting an antiquated component can be rather daunting task that would require some “computer savvy” or, perhaps, a bit of luck. However, today, for PCs and peripheral devices that are USB-compliant, one can simply plug them in and power them on. It is an automatic process that takes no skill whatsoever. Anyone can take advantage of this technology (Universal Serial Bus, n.d.).

USB is reviewed here for analogous purposes. USB is a direct connection method for hosts and peripherals. The system developed for and presented in this dissertation currently does not require or use the connection of a robot to any platform host. It acts as middleware to "discover" the robots information. This is all done with minimal effort from the user. As previously mentioned the only task required for use of any peripheral that connects via USB is to plug it in. Likewise, the user only need possess knowledge of robot name and what objects the user would like the robot to make use of.

2.3.2 Universal Remote Console

The idea of a remote console for universal usability aims to transform the way that people will one day exploit technical devices. The Universal Remote Console (URC) is an instrument intended to operate compatible target devices. The problem is a lack of a standard for managing target devices using a single device. This standard would include various manufactured devices. A single user interface is rendered to accommodate the users requirements and preferences. Home security systems, thermostats, and public kiosks are examples of networked-based applications and tools, which operate using a built-in user interface. URC researchers are looking to take advantage of electronic devices, such as cell phones, wrist watches, PDAs and other wearable and handheld computing devices to create remote consoles for all of the technologies found on a users network and operating them from anywhere. The electronic consoles would have to accommodate the diverse interfaces of each target technology (Zimmermann, Vanderheiden, & Gilman, 2003).

Waloszek (2005) believes a URC is anticipated by people who struggle with setting up video recorders and operate "smart" microwaves. With a URC, people will be able to choose their own applications as remote controls for their other devices (Waloszek, 2005).

2.3.3 Personal Universal Controllers

Appliances for the home and office are steadily becoming more elaborate as software enables new capabilities. More functionality typically means a more difficult user interface. Many researchers, including Nichols, propose a separation of the interface from the appliance. A user would carry a single device that would operate all the appliances and applications in her domain. The device, the Personal Universal Controller (PUC), downloads a specification from the target device and automatically generates an interface for the remote control (Nichols et al., 2002).

2.4 Robotics Middleware

This subsection was written to extend the introduction. Similarly mentioned in Chapter 1, there is a prolonged absence of credited, commonly adopted software architecture, termed middleware, in the robotics field (Smart, 2007).

Bakken (2003) describes middleware as ...

> "a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system. In doing so, it provides a higher-level building block for programmers than Application Programming Interfaces (APIs) that are provided by the operating system. This significantly reduces the burden on application programmers by relieving them of this kind of tedious and error-prone programming. Middleware is also informally called plumbing because it connects parts of a distributed system with data pipes and then passes data between them".

Smart (2007) asserts that nearly all researchers and institutions continue to formulate their own systems, except that the software engineering community set the standards and that, naturally, these institutions develop software that are specifically designed for their own systems, machines, and middleware. If we have a tried and true set of middleware, it can be reused and optimized with confidence over many applications (Gill & Smart, 2002). Therefore, Smart (2007) believes sharing is difficult and more than often impossible because re-use of architecture-specific technology by another institution often involves reimplementation for their different system, machines, and middleware. Costly time is taken away from studies and experimentation due to the needless, extraneous programming and debugging that takes place. If various institutions were able to implement the same algorithms, the varying systems would be capable of comparisons amongst themselves, more bugs discovered and removed, and time put to better use. A necessary prerequisite for such implementation re-use is a common communications middleware (Smart, 2007).

Middleware, now crucial to innovative robotics, is situated "between hardware and software", making it easier to develop programs (Ahn et al., 2006). Speaking in terms of software, middleware joins individual, unique, "interoperable platforms" that they might share information and system modules (Jagiello, Tay, Eronen, Fernhill, & Park, 2006). Essentially, robotics technology involves "significant interaction and coordination of diverse hardware and software elements" (Gill & Smart, 2002). Middleware can offer a common programming model across language and/or platform boundaries, as well as across distributed end systems (Gill & Smart, 2002).

CORBA is the most commonly used robotics middleware to date. Common Object Request Broker Architecture, or CORBA, is Object Management Groups "open, vendor-independent architecture and infrastructure" that computer programs use to interoperate over networks (*CORBA FAQ*, n.d.). However, it is not capable of accommodating the dynamics of the computing world. Therefore, other middleware have been developed that conform. Among are .Net, Jini and UPnP (Universal Plug and Play) (Ahn et al., 2006). Microsoft developed .NET to afford an association among the many programs operating on Windows (Ahn et al., 2006). Jini, which is not an acronym or moniker for anything, is a distributed computing environment written entirely in Java that supplies plug and play for networks. Clients are discovered and are available for devices to late and use for various operations (Newmarch, 2010). Of the few mentioned, UPnP has most recently been investigated as a middleware technology for use in robotics technology. UPnP is another technology developed by Microsoft, but, unlike .Net, is a "pervasive peer-to-peer network" architecture connecting smart appliances, computers, and devices that are wireless. It is distributed, open networking architecture taking advantage of TCP/IP and the Web to enable seamless proximity networking in addition to control and data transfer among networked devices in the home, office, and everywhere in between (Universal Serial Bus, n.d.).

Networks are capable of being joined and left dynamically by various devices, thus, more middleware is being developed to accommodate. Robots, like networks, must be "dynamic distributed computing environments" because soon each module will be dynamically installed and uninstalled.as well (Ahn et al., 2006).

The notion of robotics middleware lends itself to this dissertation as the system presented is an architecture built with components (robots and objects) that must work well together, yet, have need for an agent to join them, so to speak, for interaction.

The middleware system of this dissertation is inherently different from typical robotics systems in that the set of components that are desired to interact with are non-networkable items. Everyday real-world objects such as balls, tables, or flat screen televisions are unable to be dynamically discovered on a network. The most common way for objects to be found is through visual interfaces using cameras. This method of discovery is out of the scope of this research dissertation acts as a broker between the simulated robot and the simulated platform.

Chapter 3

System Design

3.1 Robot-Object Interaction Language

3.1.1 Overview

The purpose of the overall design is to feed any given robot details about an object that it is to interact with. Each robot could be in a different environment and have various manipulators and different methods of kinematics. The Plug and Play aspect is taken into account by the simulator's capability to virtually and asserted seamlessly introduce any generic or specific robot into any environment.

Robot-Object Interaction Language (ROIL), the system presented in this dissertation, is an attempt to answer the need to create a provisional method for robots to access various objects both in the real world and simulation.

With the basic programming skills of reading a file, knowing the object is simplified by ROILs ability to acknowledge the appropriate methods to include in a program. The ROIL system, as seen in Figure 3.1, was designed so that a user could use an internet browser to input robot information, select an object, and find out if that object is appropriate for interaction with the robot in question. With this information, the user can include the objects information in the robot's program and, as a result, properly interact with it.

At present, Plug and Play is easily relatable to inserting or "plugging in" a flash drive or some other USB-enabled device into a system. The device is then automatically discovered by the system with no need for configuration or user intervention for use, or "play". ROIL aims to mimic this concept by allowing the user to "plug", or input, little information into the system: the robots name and the robots programming language, the language to "discover"

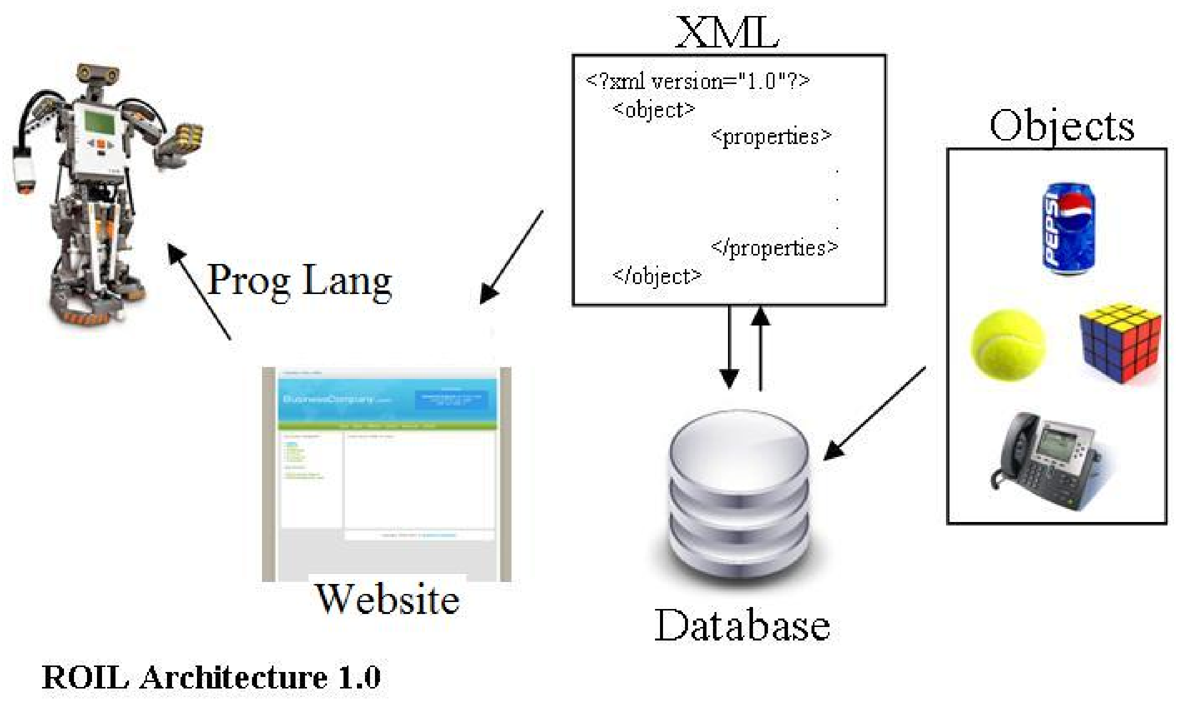


Figure 3.1: *ROIL Architecture*

the code needed for interaction, if capable. Based on this model, little coding regarding inserting objects into the environment or giving the robots code the information to access the objects is required on the part of the user.

The system, Robot-Object Interaction Language, consists of four major components. The first component is the database which stores all robot and object data. The second component consists of the the XML structures, or the "languages", that link all four components. The third component is web-based and essentially acts as an USB-enabled plugged-in robot. The fourth component enables the user to make use of or manipulate the provided robot data to her will and benefit.

3.1.2 Component I MySQL Database

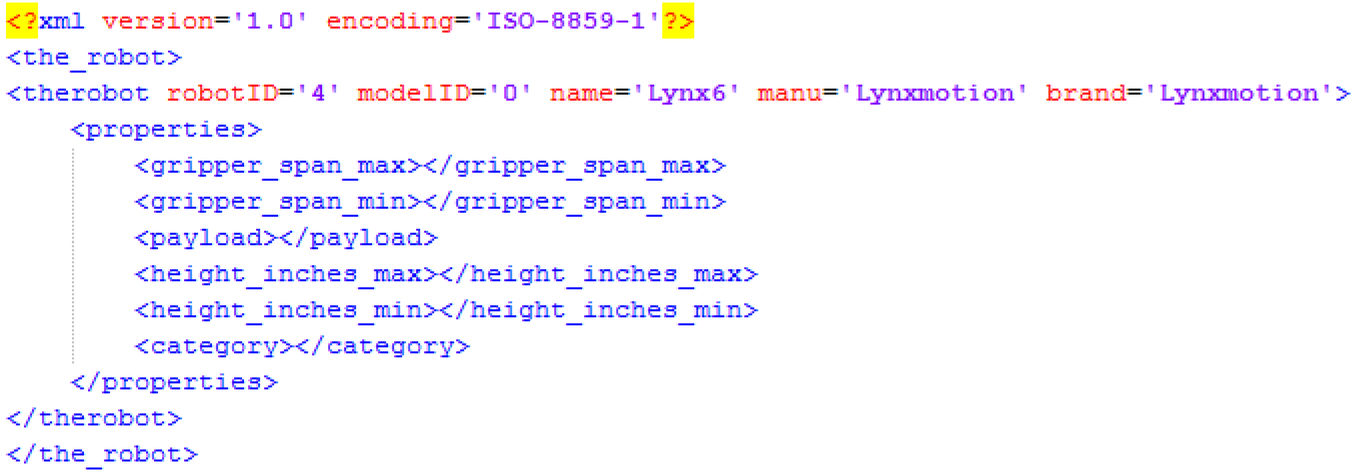
The MySQL database (see Appendix A) consists of four tables and are accessed solely by the website (Component III). The first table accessed *robotinfo* stores specifications about robot systems and includes the robots name, span of gripper, payload, gripper category, as well as the manufacturing information. The next table accessed by the website *sim_objects* contains the objects and their descriptive attributes. Attributes include the generic name of the object, the general geometric shape, dimensions, and weight. The final table of note *code* contains the actual generic code that must be appended by the user and called by her program, the instructions on how and where to append, and an additional file (*ROILfile.someextension*, discussed in Component II) to download. The particular instructions and code to append depend on the language the user writes her program in. The programming language attribute is, in fact, a foreign key linking to the auxiliary fourth table that only contains unique programming languages and a corresponding automatically incremented identification number.

3.1.3 Component II XML Structures

Concerning the two XML structures to be discussed in this section, all data is populated from the *robotinfo* table in the database.

Robot XML Structure

The XML structure (example in Figure 3.2 and Appendix B) designed to represent the robots begins with a node called *therobot* which has five attributes robotID, modelID, name, manu (or manufacturer), and brand. The field *robotID* is generated by the database from the automatic incrementation function applied to the field in the database table. The only direct child of *therobot* node is *properties* which has seven child nodes of its own. The seven children are all the rest of the fields of the *robotinfo* table being used in experimentation and are as follows: the span of the gripper (minimum and maximum), payload, height in inches (minimum and maximum), and the category. The category refers to the four basic prehension types of robot grippers found in Monkmans *Robot Grippers* (2007). The first category is called astrictive which is described as “a binding force produced by a field”



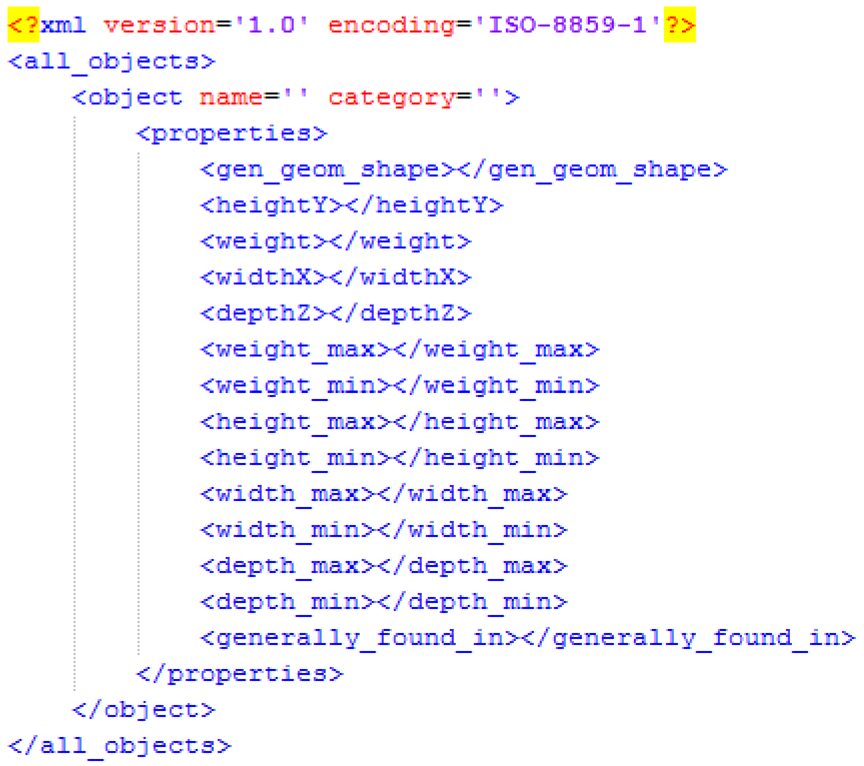
```
<?xml version='1.0' encoding='ISO-8859-1'?>
<the_robot>
<therobot robotID='4' modelID='0' name='Lynx6' manu='Lynxmotion' brand='Lynxmotion'>
    <properties>
        <gripper_span_max></gripper_span_max>
        <gripper_span_min></gripper_span_min>
        <payload></payload>
        <height_inches_max></height_inches_max>
        <height_inches_min></height_inches_min>
        <category></category>
    </properties>
</therobot>
</the_robot>
```

Figure 3.2: *Object XML Structure without data*

(Monkman, 2007). Field types include "vacuum suction, magnetism, or electrostatic charge" (Monkman, 2007). A contigutive gripper involves direct contact with the surface of the object as "contigutive mean touching" (Monkman, 2007). Examples include chemical and thermal adhesion. Ingressive grippers permeate the surface of the objects and can either be intrusive (e.g. using pins) or non-intrusive (e.g. using hooks or loops). Finally, impactive describes "a mechanical gripper" whereby apprehension is achieved by, for example, forces that impact against the surface of the object (Monkman, 2007). *Category* is the last child node of *properties* and *properties* is the only child of *therobot*, thus completing the structure.

Object XML Structure

The XML data for the objects (Figure 3.3 and see Appendix C) is populated from the *sim_object* table in the database. The root node called *all_objects* is capable of having an infinite number of child nodes called *object*. The *object* node has two attributes (name and category) and a single child node. Name is the generic name for the object. Category refers to the four robot end-effector prehension categories mentioned in the previous subsection. The only child of *object* is *properties* which encapsulates its thirteen child nodes that describe the objects. The node *gen_geom_shape* is short for the general geometric shape of the object



```xml
<?xml version='1.0' encoding='ISO-8859-1'?>
<all_objects>
    <object name='' category=''>
        <properties>
            <gen_geom_shape></gen_geom_shape>
            <heightY></heightY>
            <weight></weight>
            <widthX></widthX>
            <depthZ></depthZ>
            <weight_max></weight_max>
            <weight_min></weight_min>
            <height_max></height_max>
            <height_min></height_min>
            <width_max></width_max>
            <width_min></width_min>
            <depth_max></depth_max>
            <depth_min></depth_min>
            <generally_found_in></generally_found_in>
        </properties>
    </object>
</all_objects>
```

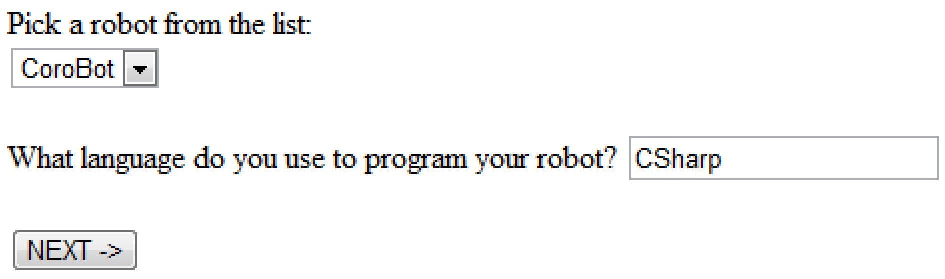
Figure 3.3: *Object XML Structure without data*

(e.g., sphere, box, capsule). See Figure 3.4 as an example of various geometric shapes. In terms of weight and dimension (i.e. height, width, depth), depending on the object, there is either a specific weight and size or a general range used to describe the object. Therefore, either the explicit set of dimensions and weight will be populated in the XML structure or the set with bounds. The last child of *properties* is *generally_found_in* which denotes the environment in which the object is most commonly found. The structure closes *properties*, *object*, and *all_objects* concluding the XML design for objects.

3.1.4 Component III - Website

Components III and IV of this system together essentially frame the plug and play model. The third component determines of the manipulation relationship between the robot and the object. It involves a website that, upon receiving a small set of information, generates a page containing a generic code and a canned file, and provides instruction on how to

Figure 3.4: *Various geometric shapes/objects.*



Pick a robot from the list:

CoroBot

What language do you use to program your robot? CSharp

Figure 3.5: *The index page of the website.*

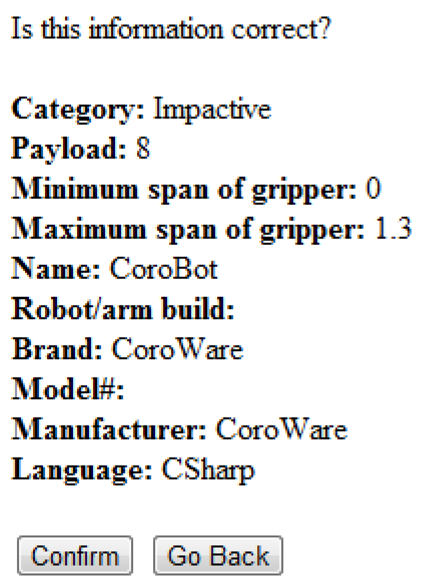
use them and where to place them (i.e. the main source code and in the same directory, respectfully). The first page of the website (Figure 3.5) is a simple form that asks the user to choose which of the available or canned robots she is using and to input the programming language her code is written in. The robot names populate a dropdown list and are retrieved from the *robotinfo* table in the database. Upon submission, the two fields are passed on to the next page.

The second page (Figure 3.6) only retrieves and displays the robots specifications from the *robotinfo* table in the database so that the user may verify that the robot chosen is, in fact, what she specifically wanted. If not, she has the option to go back and change the values on the previous page. If the robot specifications are correct, then the robots information is hidden in html input tags and submitted to the next page *objectindex.php* (Figure 3.7). Upon loading, the robot XML file (Figure 3.2; also see Appendix B) is created by being fed the selected robot's data. Displayed on the page is a multiple select list occupied with each object stored in the database table *sim_objects*. When all of the desired objects are selected the objects are sent to the final page of the website *createXMLfile.php* (Figure 3.8). On this page each object populates the child node *object* discussed in Component II and are written to a newly created XML file called *selectedobject.xml* (Figure 3.3 and see Appendix C)that uses the object XML structure. Using the PHP function *SimpleXML* the XML structures are loaded into two variables and using if statements the child node of *properties* in both XML files are logically compared. For example, the weight of the object is compared to the payload of the robots gripper. The results of the comparison are displayed on the page. Next, the instructions, ROILfile, and the paste-in code are retrieved from the table *code* depending on the programming language inputted on the first page. They are all also displayed on the page; only *ROILfile.someextension*, the canned file, is downloaded using a displayed hyperlink.

The page will produce results such as "CoroBots gripper span is too small to fit around the book" and "The TriBots payload is able to carry the ping pong ball." Below, on this same page, are the code in a scrollable window, a link to the canned file, and the instructions produced directing the user to take and incorporate the provided code in their existing program. This page is the result of the compilation and comparison of the two XML files. This comprises the third component of the system design.

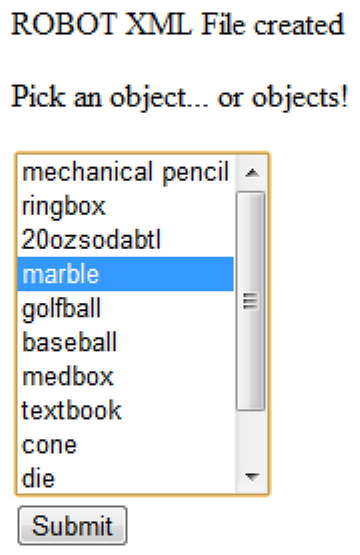
The Canned File - ROIL.someextension

This file is generally a class that reads the XML object file uploaded on the server and is read from the main source code. The code in *ROIL.someextension* begins with the creation


Is this information correct?

Category: Impactive
Payload: 8
Minimum span of gripper: 0
Maximum span of gripper: 1.3
Name: CoroBot
Robot/arm build:
Brand: CoroWare
Model#:
Manufacturer: CoroWare
Language: CSharp

Figure 3.6: *The second page of the website.*


ROBOT XML File created

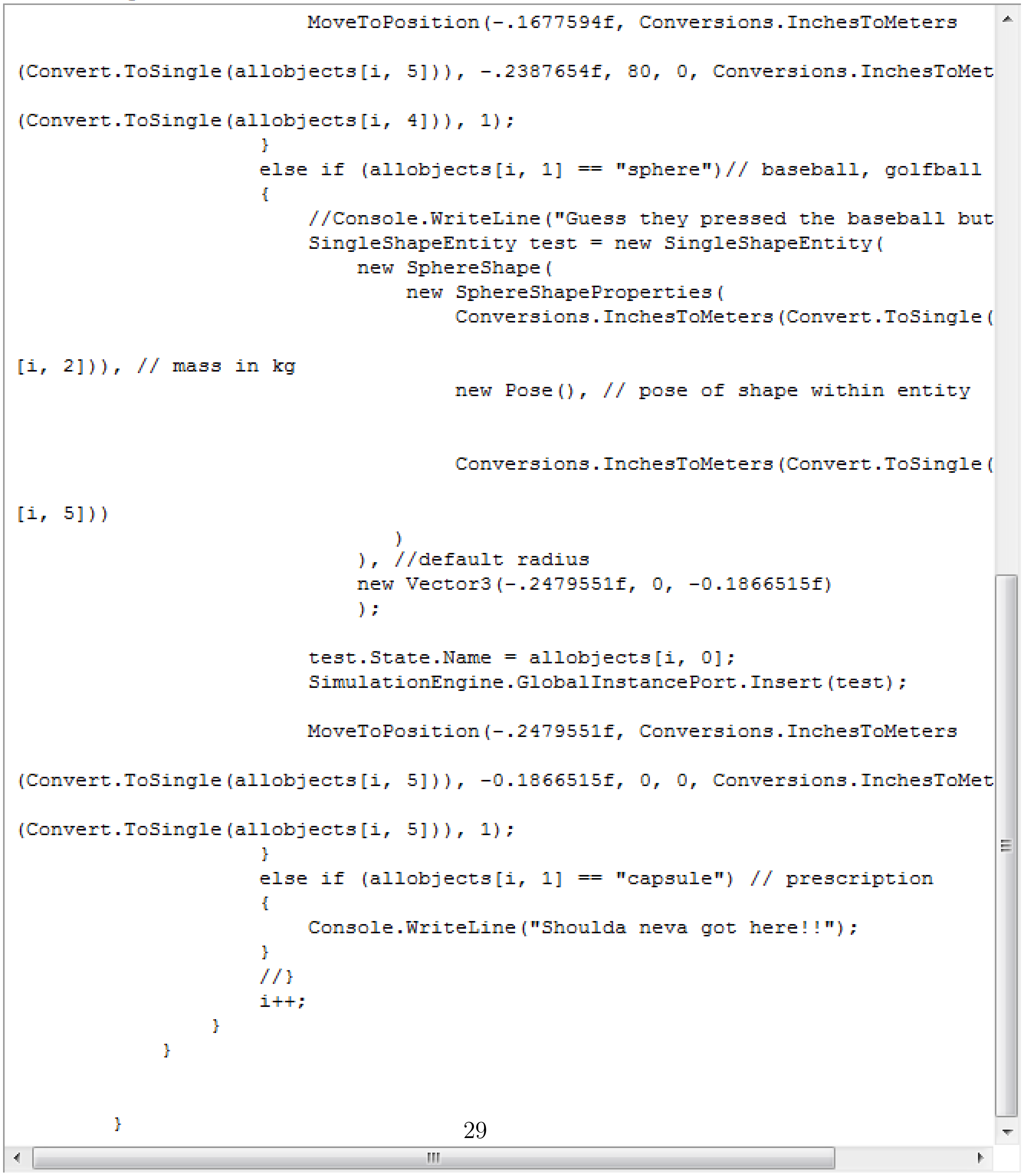
Pick an object... or objects!

Figure 3.7: *The third page of the website.*

The gripper will not handle the marble as is too big or too small to grip the object.
The robot may be able to pick up the marble but it may be fairly difficult to manuever.
This robot's end-effector is in the proper category to appropriately handle the marble.
Language: CSHARP

Instructions: Paste the following code in your main class and call it from the Start() method.

The code to paste in:

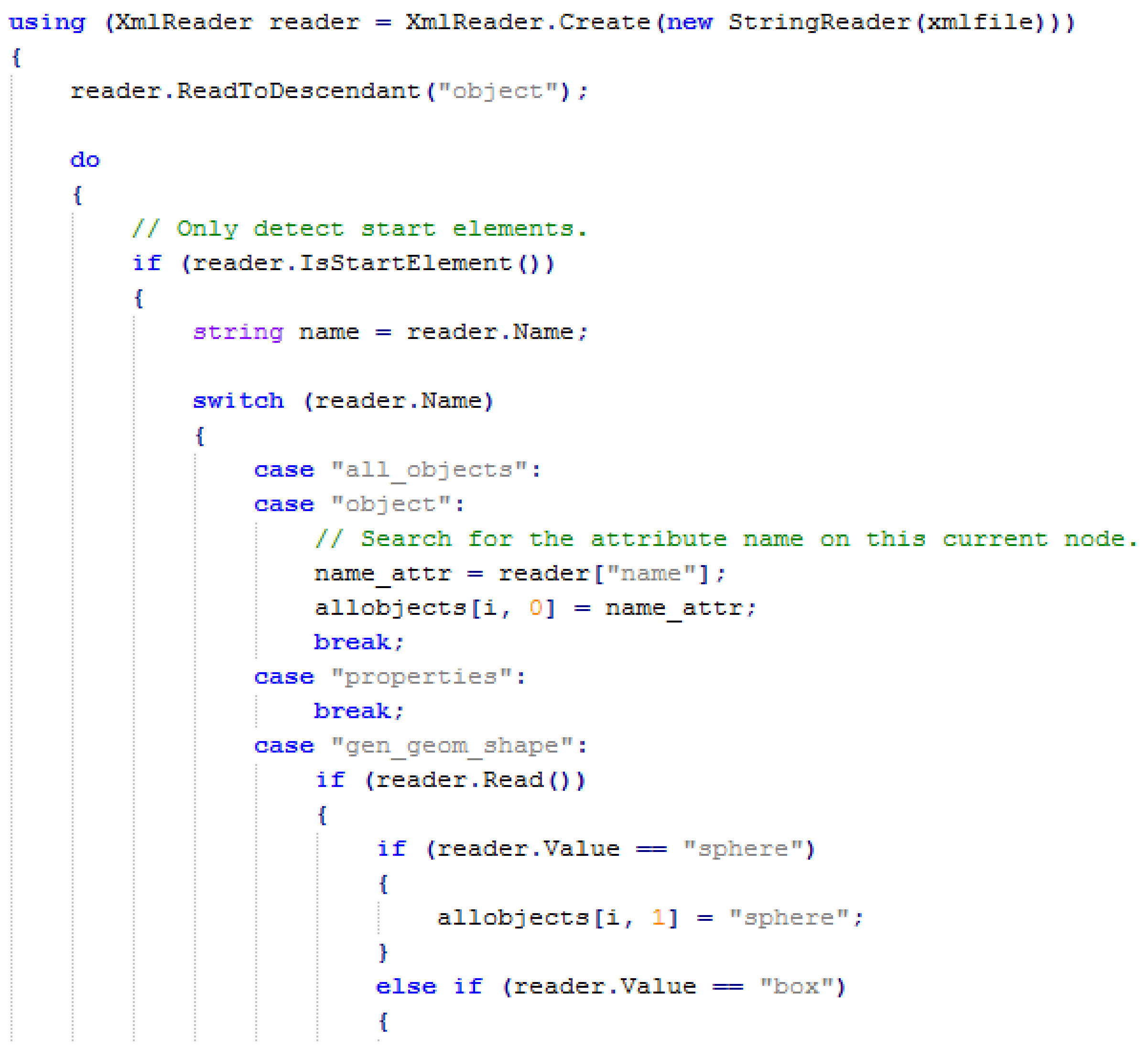


```
                        MoveToPosition(-.1677594f, Conversions.InchesToMeters

(Convert.ToSingle(allobjects[i, 5])), -.2387654f, 80, 0, Conversions.InchesToMet

(Convert.ToSingle(allobjects[i, 4])), 1);
                    }
                    else if (allobjects[i, 1] == "sphere")// baseball, golfball
                    {
                        //Console.WriteLine("Guess they pressed the baseball but
                        SingleShapeEntity test = new SingleShapeEntity(
                            new SphereShape(
                                new SphereShapeProperties(
                                    Conversions.InchesToMeters(Convert.ToSingle(

[i, 2])), // mass in kg
                                    new Pose(), // pose of shape within entity


                                    Conversions.InchesToMeters(Convert.ToSingle(

[i, 5]))
                               )
                            ), //default radius
                            new Vector3(-.2479551f, 0, -0.1866515f)
                            );

                        test.State.Name = allobjects[i, 0];
                        SimulationEngine.GlobalInstancePort.Insert(test);

                        MoveToPosition(-.2479551f, Conversions.InchesToMeters

(Convert.ToSingle(allobjects[i, 5])), -0.1866515f, 0, 0, Conversions.InchesToMet

(Convert.ToSingle(allobjects[i, 5])), 1);
                    }
                    else if (allobjects[i, 1] == "capsule") // prescription
                    {
                        Console.WriteLine("Shoulda neva got here!!");
                    }
                    //}
                    i++;
                }
            }


        }
```

of a multidimensional array that is set as the return type of a function *Readfile()*. Next, the object XML file is loaded into a variable named *xmlfile* and a builtin function that reads XML files is utilized to traverse the nodes and access each element. Because the programming for experimentation of this dissertation was specifically written in C# XMLReader is the function seen in Figure 3.9. Here, XMLReader is used twice. The first use determines how many objects are in the XML file, which is the first dimension in the array. The second dimension, which is the number of attributes and properties in the XML file, is determined using a switch statement that accesses each element to determine what is in the file and its value.

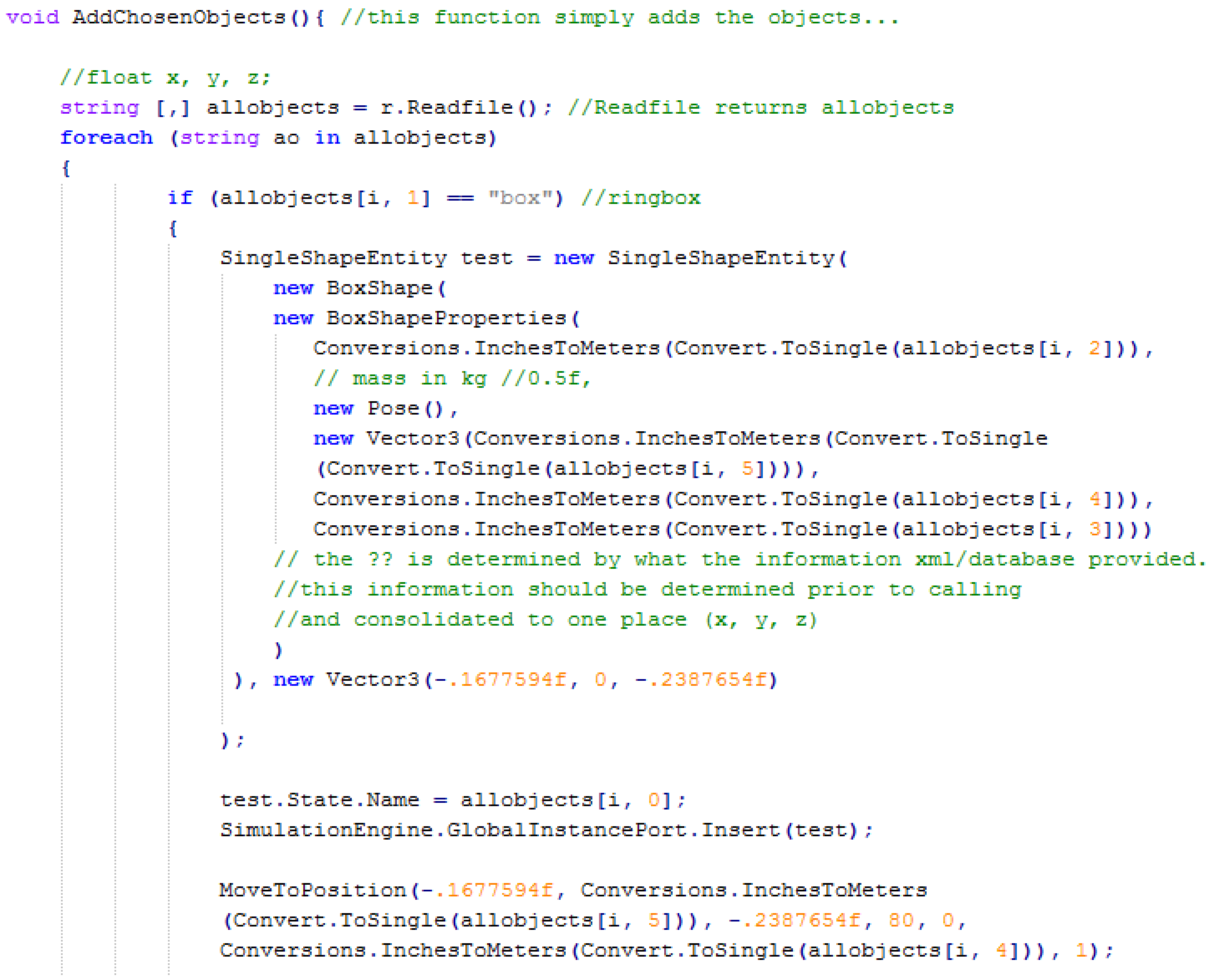
The Appended Code

The snippet of code (Figure 3.10) the user is asked to append to her code is produced in the same language as the program she is writing her program in. This pasted code snippet is a function that calls the function *Readfile()* (Appendix D) in *ROILfile.someextension*. The array of objects that is returned is traversed using a foreach statement to create and insert each object in, in this case, the simulated environment. Microsoft Robotics Developer Studio is programmed in C#, for example, and the code to create and insert an object involves identifying the basic or complex geometric shape of the object. If the object is a cube or rectangular shape, as exhibited in Figure 3.4, then the block of code requires a specific set of dimensions or volume, depending on the shape (e.g. cube requires length, width, and height). Also required are the mass and position of the object. The object must be uniquely named and then inserted. Besides changing the position, dimensions, etc., the user also has the option to add information, such as a mesh file, to give the shape the appearance of the object as seen in the real world.



```
using (XmlReader reader = XmlReader.Create(new StringReader(xmlfile)))
{
    reader.ReadToDescendant("object");

    do
    {
        // Only detect start elements.
        if (reader.IsStartElement())
        {
            string name = reader.Name;

            switch (reader.Name)
            {
                case "all_objects":
                case "object":
                    // Search for the attribute name on this current node.
                    name_attr = reader["name"];
                    allobjects[i, 0] = name_attr;
                    break;
                case "properties":
                    break;
                case "gen_geom_shape":
                    if (reader.Read())
                    {
                        if (reader.Value == "sphere")
                        {
                            allobjects[i, 1] = "sphere";
                        }
                        else if (reader.Value == "box")
                        {
```

Figure 3.9: *Object XML Structure without data*



```
void AddChosenObjects(){ //this function simply adds the objects...

    //float x, y, z;
    string [,] allobjects = r.Readfile(); //Readfile returns allobjects
    foreach (string ao in allobjects)
    {
            if (allobjects[i, 1] == "box") //ringbox
            {
                SingleShapeEntity test = new SingleShapeEntity(
                    new BoxShape(
                    new BoxShapeProperties(
                       Conversions.InchesToMeters(Convert.ToSingle(allobjects[i, 2])),
                       // mass in kg //0.5f,
                       new Pose(),
                       new Vector3(Conversions.InchesToMeters(Convert.ToSingle
                       (Convert.ToSingle(allobjects[i, 5]))),
                       Conversions.InchesToMeters(Convert.ToSingle(allobjects[i, 4])),
                       Conversions.InchesToMeters(Convert.ToSingle(allobjects[i, 3])))
                    // the ?? is determined by what the information xml/database provided.
                    //this information should be determined prior to calling
                    //and consolidated to one place (x, y, z)
                    )
                 ), new Vector3(-.1677594f, 0, -.2387654f)

                );

                test.State.Name = allobjects[i, 0];
                SimulationEngine.GlobalInstancePort.Insert(test);

                MoveToPosition(-.1677594f, Conversions.InchesToMeters
                (Convert.ToSingle(allobjects[i, 5])), -.2387654f, 80, 0,
                Conversions.InchesToMeters(Convert.ToSingle(allobjects[i, 4])), 1);
```

Figure 3.10: *Object XML Structure without data*

3.1.5 Component IV Simulated Robot Environment

The fourth component requires the produced generic code from Component III, which is placed in the users main source code as noted in the instructions. The generic code simply places the selected object in the robots simulated environment with real world dimensions and in a random position. The user is able to modify the code, however, to change the objects dimensions in order for it to be scaled to fit their particular environment. The position, mesh and every other part of the code are editable, as well.

Together, Components I, II, III, and IV comprise the Robot-Object Interaction Language scheme. With these items established the next step in this dissertation research is to examine the efficiency of the design and architecture of the system.

Chapter 4

Experiment

Upon completing the system implementation of ROIL based on the approaches outlined in the previous chapter, a formal experiment was conducted to validate this system. The objective of this evaluation focuses on the system performance with respect lines of code, and redundancy, which supports the hypotheses. This chapter reports on the experiment with the simulated CoroBot, simulated Lynx 6 Arm, and the simulated TriBot.

4.1 Experiment Design

4.1.1 Overview

The goal of this experiment is to evaluate ROIL with respect to its ability to connect to various robots, create and insert appropriate code, and work effectively via simulation. A positive outcome would involve the addition of fewer lines of code and faster processing time of task completion. This outcome should also include manipulation with less redundancy and as effective as a simulation without ROIL.

Both the control and variable setups predominately consist of the coding in the Microsoft Robotics Developer Studio. However, the control setup, ROIL, is essentially two parts itself, where the plug and play feel is given by the website, but the platform language takes advantage of the generated code to do the actual task at hand.

4.2 Materials

Platform

The software used to implement the simulation was Microsoft Robotics Developer Studios 2008 R3 (MRDS) (Microsoft, n.d.), a platform for developing robotics applications. This platform was chosen for its powerful graphics engine and its portability, flexibility, affordability, and set of templates. The Visual Simulation Environment allows for testing robotic applications using a 3D physics-based simulation engine. MRDS is portable due to the fact that one can use the simulation code and deliver it to robot hardware with few changes. Its flexibility is attributed to the fact that it supports a wide variety of robots. Programming in MRDS is typically done using Visual Studio and the .NET framework, and is generally executed in C#.

Specifically, Visual studio 2008 and the .NET Framework 4.0 were used to code the simulated robots, environments and objects for MS Robotics Studio. The experimentation was performed on a single Dell Optiplex 755 running Windows Vista Business SP1. A local server was needed to upload and access the XML files. The server was downloaded via XAMPP (*Apache Friends - XAMPP*, n.d.), a cross-platform (X) distribution package which includes Apache, MySQL, PHP and Perl (AMPP) for Windows platforms XP and above. The website was created on the local server using PHP. MySQL houses the data describing the object set and robots and code.

One other technology used to support this research includes LineTally. LineTally version 1.7 is simple freeware that counts the number of lines in the source code of 61 different program languages. It determines how many lines are code, comments, mixed (code and comments), blank, the sum, and the respective percentages of each (*LineTally*, 2008). LineTally is the software that was used to count the lines of code for that would be analyzed for experimentation.

4.2.1 Participants

There were no human participants in this research study. To use human participants would have required a severely limited group or community, as well as, a very specific and small segment of the worlds population. Human participation would also be time consuming (e.g. learning a new language, applying said language) proceeding weeks and months per robot programmer and program. A certain level of education would be preferred in computer science or engineering, as well as, efficient programming experience. Therefore, the researcher of this study was the sole participant to program the interaction using one language with three robots on one platform.

Robot Descriptions

Three robots were used in this study. Although each robot was written on the same platform and in the same language, each has different features, functions and capabilities due to their varying designs and construction. In addition, a different programmer originally coded each.

TriBot: This 3-wheeled vehicle has multiple sensors. Lego Mindstorms NXT (1999) follows the typical building block scheme that its brand is known for and, therefore, despite some instructions, users are not restricted to any building specifications or guidelines. However, the TriBot used in this experimentation was built specifically according the instructions as the only required pieces were the grippers, which are included in the manufacturers instructions. Broadly, the TriBot (Figure 4.1) has 1 degree of freedom (DOF) and its grip opens to approximately 7 inches and closes at approximately 2 inches. It is powered by the Lego Mindstorms NXT brick. The simulated TriBot used in this research is a software package originally programmed by the development company, SimplySim (n.d.) and was altered for use in this research.

CoroBot: The CoroBot, in Figure 4.2, with an arm has the following dimensions: 12 x 13 x 10 (x 16 inches with arm). It has a 4-wheel drive, a 14 inch long arm, 4 DOFs in the

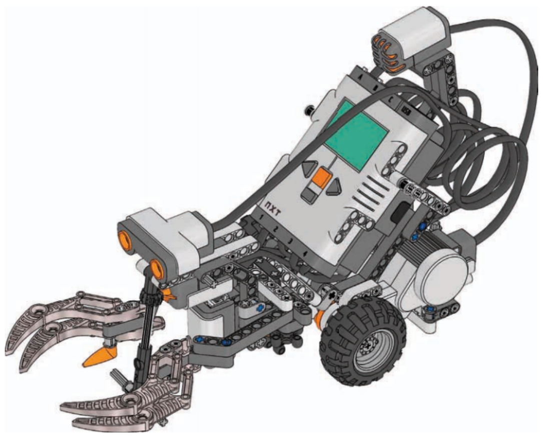


Figure 4.1: *Lego Mindstorms TriBot*

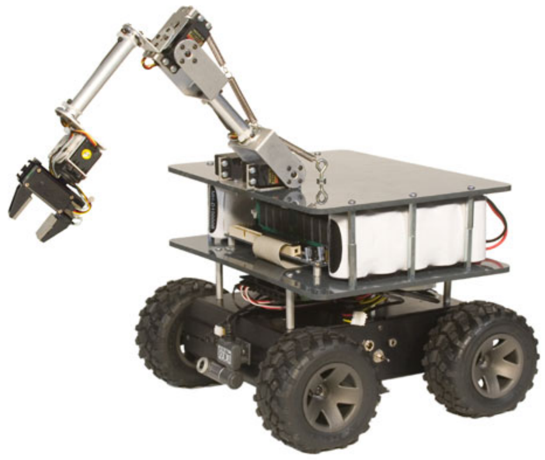


Figure 4.2: *CoroWares CoroBot*

arm, a gripper span of 1.3 inches. CoroBot has a gripper sensor, an arm payload capacity of 8 ounces and is supported on Windows XP in C, Linux Ubuntu, and Player. It is powered by a standard CPU/motherboard and is designed, built, and sold by CoroWare (CoroWare, n.d.). The simulated version of the CoroBot was written by CoroWare developers and was released to the robotics community and was altered for the cause of this dissertation study.

Lynx L6 Robotic Arm: The Lynx 6, in Figure 4.3, is itself an articulated arm. Although its production has been discontinued there are similar robots being sold by its creator company Lynxmotion (e.g. AL5A Robot Arm). This arm, built for hobbyist, boasts 6 degrees

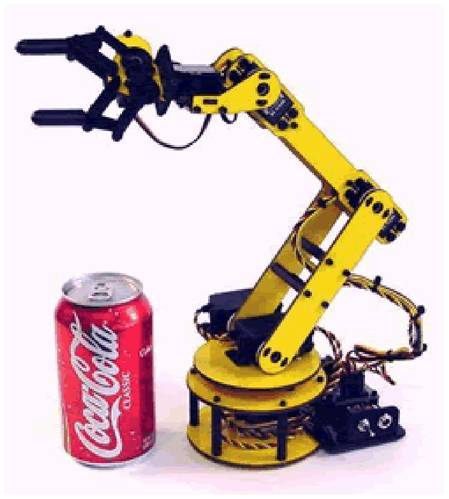


Figure 4.3: *Lynxmotions Lynx6 Arm*

of freedom (DOF) with a base height of 3 inches and a 5.083 inch median reach and a 4.5 ounce lift capacity. Its maximum grip span is 2.25 inches and is powered by an SSC-32 controller. In MRDS, the Lynx 6 is immobile. It is not an extension of another driving robot (Lynxmotion, n.d.). The Lynx L6 Arm simulation was written by Microsoft Robotics Developers and was altered to suit the purpose of this dissertation research.

4.2.2 Simulation Environment

Microsoft Robotics Developer Studio 2008 R3 (MRDS) is a simulation engine that was developed to be both an area for those who do not have physical robot hardware to program and for others to use as a testing site before applying their code to their robot machine. MRDS has a great number of varied simulated robots and environments to take advantage of and the software makes it easy to add or create many more (Johns & Taylor, 2008). It is reasonable to suggest that student-led research that requires a number of robots of various forms and capabilities is reasonably better suited in a simulation environment. This is such research, in that three different robots are used but are not readily available in their hardware form.

Each robot is packaged with its own individual default simulation MRDS environment. The Simulated Lynx 6 is inserted into an uncluttered environment. It sits on a wooden floor that is surrounded by an infinite amount of open space. The CoroBot is situated in a two room house. It begins in an office that has a desk, chair, and bookshelf. The Tri Bot is defaulted in a room that has a poster on the floor that has colors for its color sensor and other default objects like plants in the background. The first environment is more likened to a research lab/area where the last two environments are the most similar to real world human environments.

4.2.3 Setup

The variable group of the study focuses on the concept of handwritten programming without any automation. In the experimentation, two factors are represented: ROIL and non-automated, handwritten code. The latter factor has four sets of object groupings. There are three robots each with a default environment and twelve objects to be created, inserted, and accessed by each robot. The four groupings of the variable setup are described as follows:

- A set of nine random objects was inserted in each robots program.
- A set of six random objects was inserted in each robots program.
- A set of three random objects was inserted in each robots program.
- Each object was inserted individually in each robot's environment

The code produced by ROIL replaces each object grouping, thus, only four programs (one per object grouping) utilizing ROIL per robot are necessary for experimentation.

4.2.4 Units of Measurement

Note that in Chapter 2 none of the languages or software discussed made mention a unit of measurement to gauge its performance. For each software project, there was no accessible or publicized study found to establish a unit of measurement for this dissertation research. After further consideration, it was determined that lines of code and redundancy would be the most appropriate measurements of efficiency.

When researching the lines of code metric, what was found was that the metric was typically utilized for measuring the growth of a project (Cunningham & Cunningham, Inc., n.d.), the progress or effort a programmer is making (Cunningham & Cunningham, Inc., n.d.), and cost estimates (Tuxtips.org, 2011). In these cases, for those who accept LOC as a valid form of measurement, the greater the number of lines of code is considered beneficial; however, Andy Hertzfeld (2011), an early Apple Macintosh developer, recounts an anecdote about the author of QuickDraw, describing how his goal of programming "was to write as small and fast a program as possible". Although LOC as a metric is widely argued to be a useless metric for measuring productivity or progress in software development (Marx, 2008) it may also be argued that fewer LOC produce less complex yet more precise (Charlton, 2008) software, fewer defects (Cunningham & Cunningham, Inc., n.d.). Counting lines of code in this experimentation is not relatable to designing "clever code (James, 2007) or code tuning, where one take five lines of code and shrink them down to one. This smarter code often reduces readability (Lucas, 2008). The advantage of reducing lines of code, which involves getting rid of unnecessary code, especially if they are redundant, includes increasing maintainability or reducing the complexity of bugs (Lucas, 2008). Other advantages decrease in compilation time and often readability. Steve McConnell (1993), the author of *Code Complete: A Practical Handbook of Software Construction*, believes that using lines of code to measure software estimation is an acceptable place to start as long as its limitations are kept in mind. While speed is not a factor explored within the scope of this dissertation research, size is. The smallness in program size is determined advantageous.

Redundancy was also resolved to be a source of performance measuring for the ROIL system. Again, this metric is employed adversely to its typical appliance. Software redundancy is more often used to prevent failure and as a form of backup but usually with more critical software (e.g. medical and space) (Teach-ICT.com, n.d.). Here, it is promoted as a source of fault due to the facts that the software platform that the programs are written in is heavy in video graphics and can be greatly delayed with redundancy, as well as the fact that purposed redundancy opposes the desire to decrease the number of lines of code.

A decreased count of lines of code and redundancy in the programs do not directly correlate to robot programming standardization. Determining standardization would require the inclusion of human participants and qualitative experimentation, which was not feasible for or within the scope of this research. Official standardization will require ROIL to be widely used amongst the robotics industry. The deduction of the two measuring units recommends themselves to optimal and efficient usage of the software. ROIL, being one place for users of various robots to gain code for object interaction, is in itself a commonality.

4.2.5 Tasks

The immediate task at hand is to get the robot and object relating and producing some results and generate a generic code waiting for specifications (e.g. position) from the user. Subsequently, the task of the generic code should insert the chosen object(s) into the environment.

Briefly, the task of each robot is to have access to information about the object with which it is to interact. The user must use a form of modality to make that determination, (e.g. she speaks to the robot, presses a button or makes a gesture). The use of the information is beneficial both before and during the simulation. The user is able to use the information when informed of the compatibility of the object with her robot pre-simulation. Moreover, the robot is capable of using the information by adjusting its grip span, arm reach (depending on the inverse kinematics model used). The researchers task included learning a new language.

Again, briefly, the tasks are:

- Establish the relationship between the robot and the objects
- Generate a bare, generic code in the language of the robot

  - Code should create and insert the various objects

- End user provides specifications, modality for interaction

4.2.6 Variable Environment

The variable experiment demonstrates what a robot programmer would ordinarily do without any assistance. The program is a fully developed environment and is completely coded by hand. The testing of this environment demonstrates the increased number of lines and redundancy that may result from a lack of a standardized or common support. The following sections explain the reasoning behind the methodology of inserting both a single object and multiple objects in the simulation environments for experimentation.

Single Object

In the variable experimentation the decision to insert a single object in each robot environment was based on the assumed simplicity and ease of doing so when compared to inserting multiple objects at a time. When inserting a single object it would be sensible to code as if there were only one object as opposed to coding for the likelihood of multiple objects. This is not especially difficult except in the case where the user decides to change the number of objects she would like to include. Furthermore, if the lone object itself must change whether this is known at the beginning of the project or is decided later on then the programmer must decide whether or not to initially include each object in the code and continually go in and change which one is used every time that change is needed. Otherwise, she will have to go in and rewrite the code to replace the old object with the new one. Either decision is a nuisance without some type of array, struct, class or database that stores the information of each object.

Multiple Objects

During experimentation for the variable group when coding to insert more than one object complexity is further increased when considering any changes that may be required concerning which objects to use. Coding the programs with multiple objects was an absolute necessity in order to show ever growing amount of code as the number of objects coded increases.

4.2.7 Control Experiment

The control experiment is the system developed during this research that is purposed to decrease the amount of work and code on behalf of the end user. That system is the Robot-Object Interaction Language (ROIL). ROILs main goal is to simplify the programming of code for an object or objects that will interact with various robots.

4.2.8 Scenario

A graduate student pursuing a Masters in computer engineering with a focus on robotics, Zahara, is working a manipulation project with her advisor and two others. Zahara and her group are using the Lynx 6 and are simulating it in Microsoft Robotics Developer Studio. Their code is written in C#. Zahara is responsible for gathering objects on which they can test the arm and determine if they are compatible for interaction with the Lynx 6 Arm. She goes to the ROIL website and checks to see if her robot is listed. She sees that it is and chooses her robot from the dropdown list on the page and types in the language in which they will code their robot. She presses the “Continue” button, and on the next page, she is asked to verify that the information displayed is the specifications for the Lynx 6. If the robot is not correct she may press the “Go Back” button and choose another robot or she may press the “Continue” button to proceed. Zahara presses “Continue” and now has a list of random objects that she can find in the real world. She wants the Lynx 6 to pick up and move around the following objects: an empty ring box, a single die, a baseball, a marble, a large prescription bottle, and a pencil.

After pressing the "Choose" button, the next page informs her that the robot is able to handle the widths of all the objects except the baseball. It states that all of the payloads are acceptable, as well. If she scrolls down a bit she will see a set of instructions that tell her to copy the generic block of code presented further down on the same page and to place it in the main class of her source code. It also tells her where to place a few other lines of code in the same class. Finally, Zahara is directed to a link of a file that she should simply include it in the same directory of her main file.

Now Zahara makes sure her windows form code is able to call and access each object. When she runs her simulation she uses the two windows forms she created to drive the Lynx 6s arm and to set the grip span of the gripper.

Chapter 5
Results

This section discusses the results of the comparisons of the quantitative data from the programs that were coded. Each section will give the initial look at the resulting numbers and the following section will analyze the significance of the collected data. The data presented was collected from the programs written for the three robots previously mentioned and the ROIL system programs (control group) to determine if there were any improvements in the numbers of lines of code and the amount of redundancy.

5.1 Data Analysis

5.1.1 Lines of Code

The main purpose of this research is to provide a service that eases a programmer into developing code by providing a standard in coding that would permit their robots to interact with objects. To validate this research, results were explored from the comparison of the number of lines of code and redundancy of three robots programmed to interact with one, three, six, or nine object(s) to the ROIL system. This section reviews the data collected concerning the number of lines of code from programs with four sets of objects for the robots to interact with.

Where eight programs per robot were written to create a sample population (each to code a different combination of objects), only one program was necessary for experimentation per robot using ROIL. The following table, 5.1, displays the number of lines of code (LOC) for the control group. These three quantities, one per robot, were used and referred to repeatedly for comparison to the variable group.

Robot simulation	**Lines of Code**
CoroBot	190
Lynx6	1070
TriBot	137

Table 5.1: Lines of code count per simulated robot for ROIL (control group)

Figures 5.1 , 5.2, and 5.3 show each robots LOCs by number of objects and also where ROIL fares in comparison at first glance. As seen in Figure 5.1, ROILs LOC are less than all sets of the TriBots simulated code with more than one object. However, comparing ROIL with the CoroBot programs (Figure 5.2), ROIL fared differently as the variable groups with one and three objects both have fewer lines of code than the control group. Therefore, the control groups lines of code only fare better compared to the CoroBots variable programs with six and nine objects.

In Figure 5.3, it is seen that all four sets of objects in Lynx6 programs have an overall greater number of LOC than that of the control group.

Figures 5.4, 5.5, 5.6, and 5.7, below, describe the difference in percentage of lines of code for each set of objects within the variable group from the control group, ROIL. The average of each set of objects was found as the starting point and the percentage of difference was established for each robot of the control group. Hence, a positive percentage denotes a decrease in the LOC, the desired result. Figures 5.4 and 5.5 both show that programs in the variable group with nine and six objects both average to have a decrease in LOC of 40.82%, 26.99% and 24.85% and of 26.29%, 13.09%, and 22.18%, respectively.

In Figure 5.6, CoroBots three-object programs average to have 5.63% less LOC than the control. However, the TriBot and Lynx6 both have a decrease in lines of code using ROIL with the same number of objects.

Displayed in Figure 5.7, TriBots and CoroBots lines of code with a single object average to have 15.21% and 30.36%, respectively, less LOC than ROIL. Lynx6 LOC all has a greater

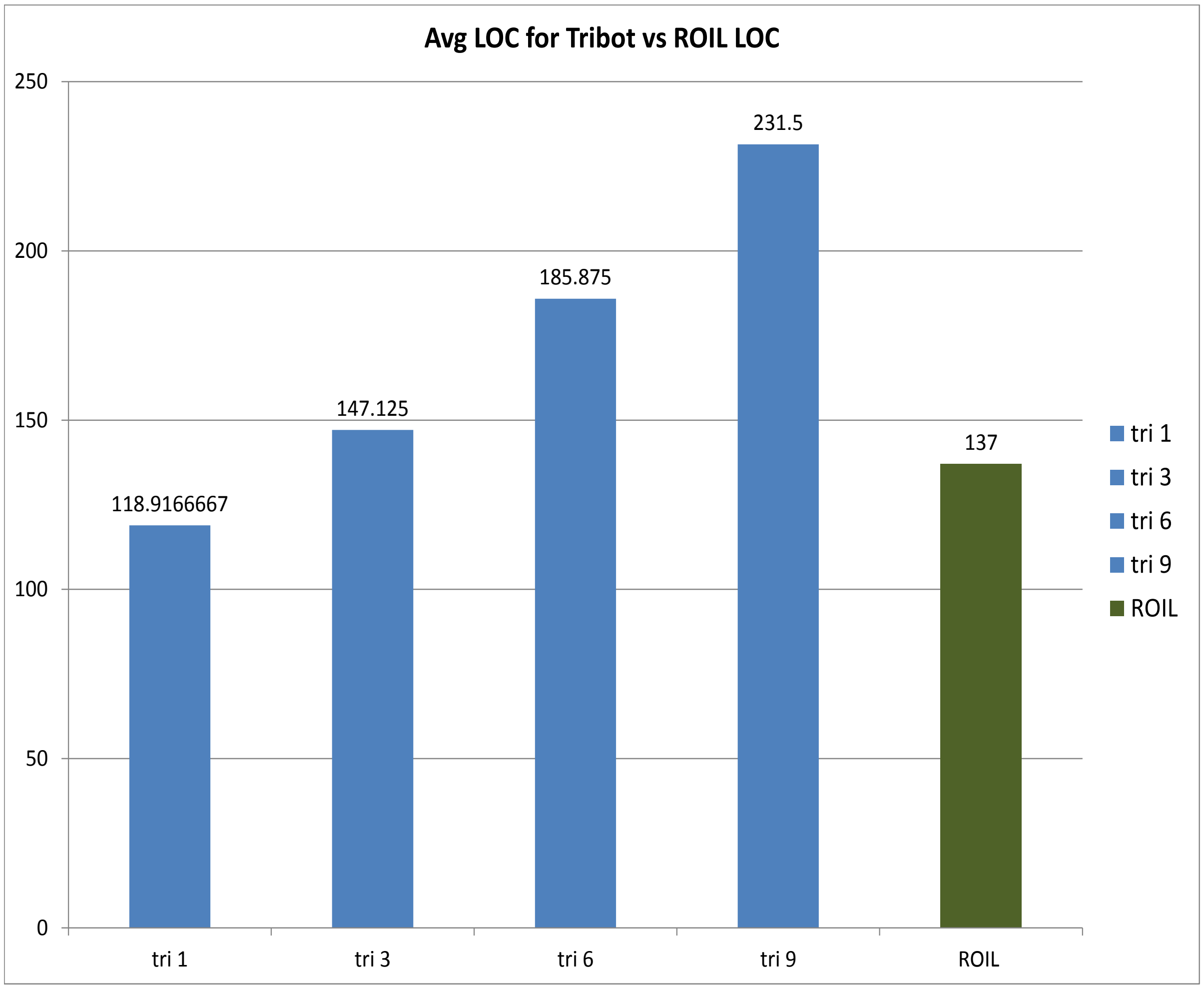


Figure 5.1: *The lines of code count in the Tribot program vs the count in ROIL*

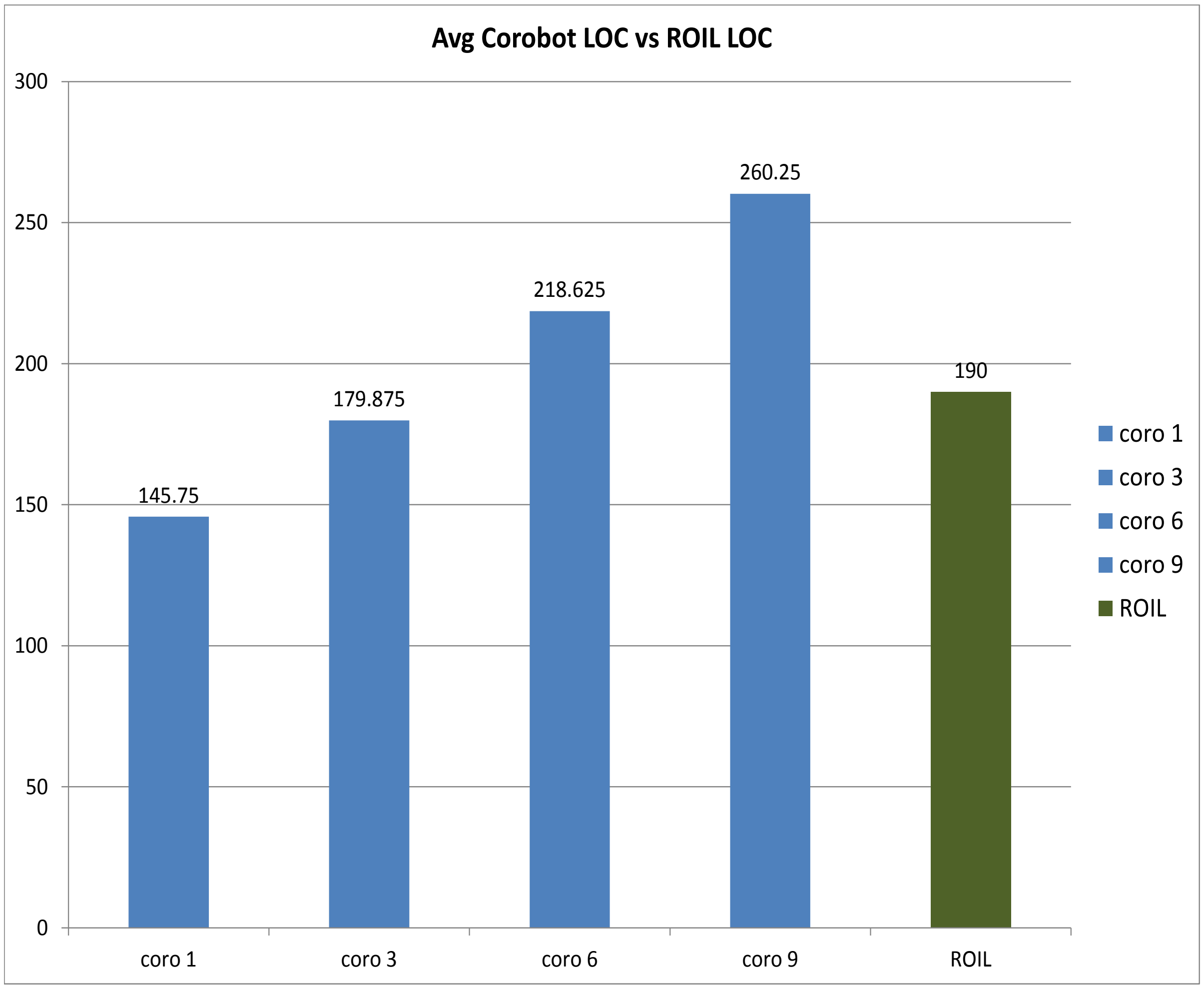


Figure 5.2: *The lines of code count in the CoroBot program vs the count in ROIL*

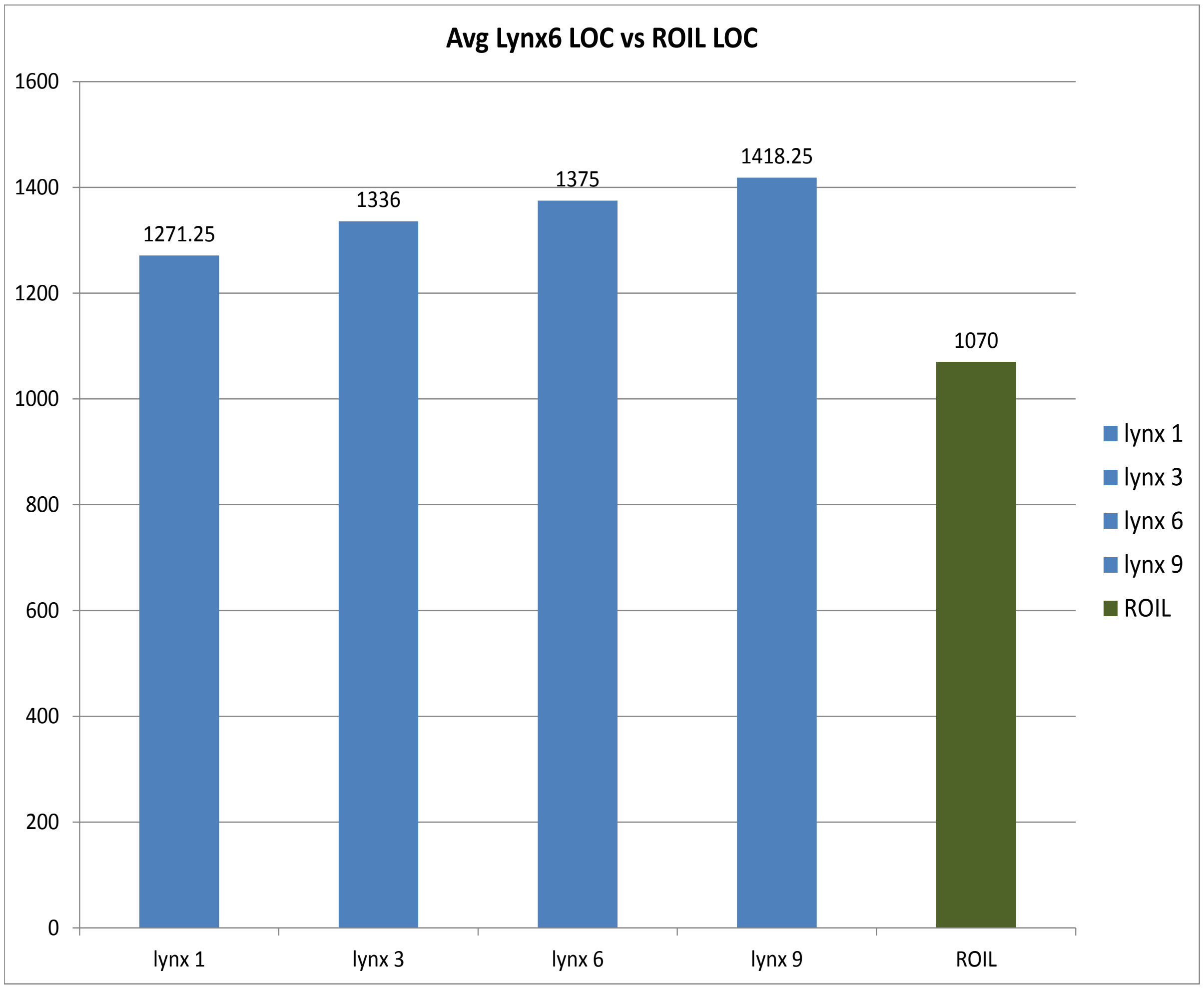


Figure 5.3: *The lines of code count in the Lynx6 program vs the count in ROIL*

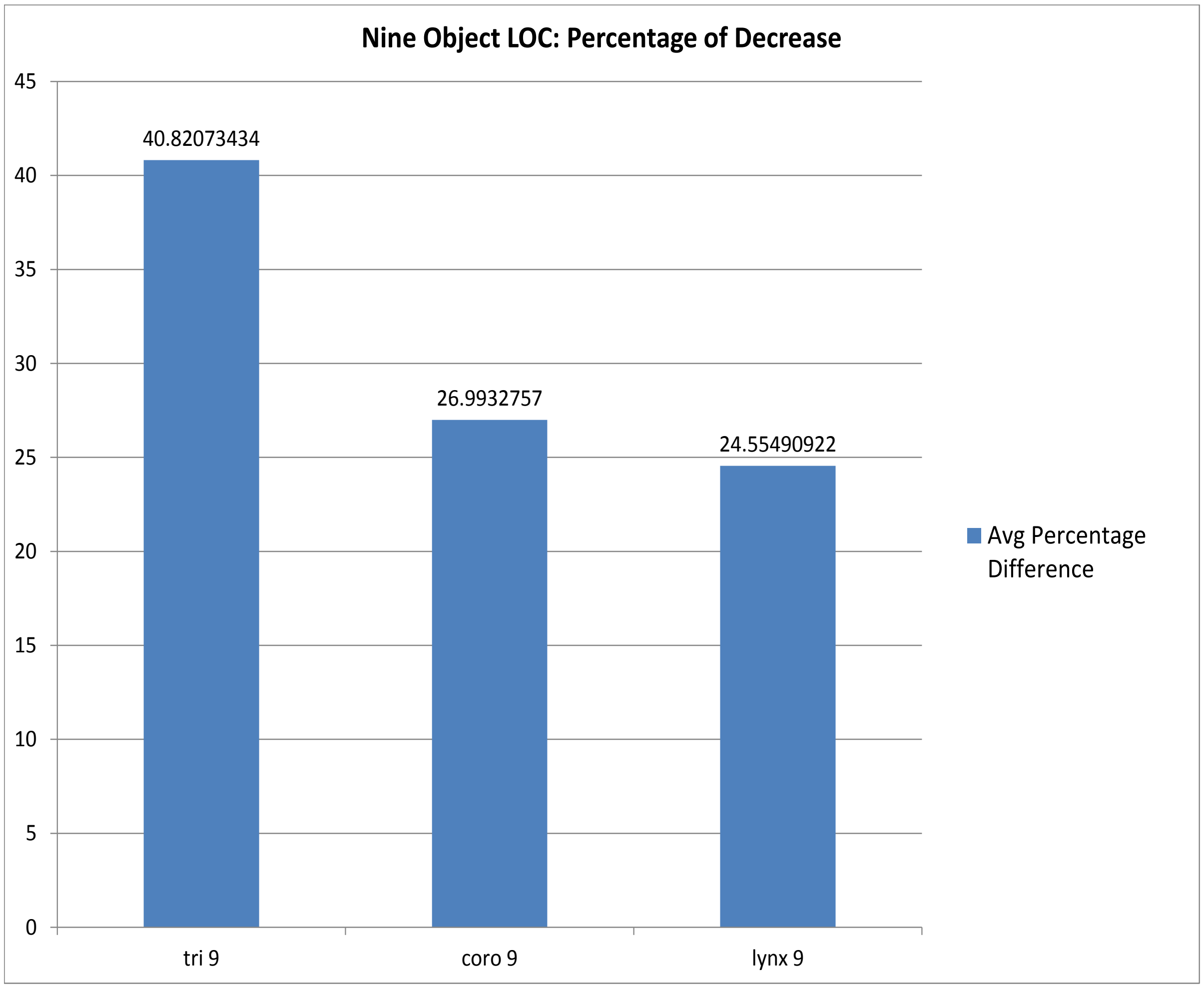


Figure 5.4: *The percentage of decrease in the LOC count of each robot's program that inserted nine objects.*

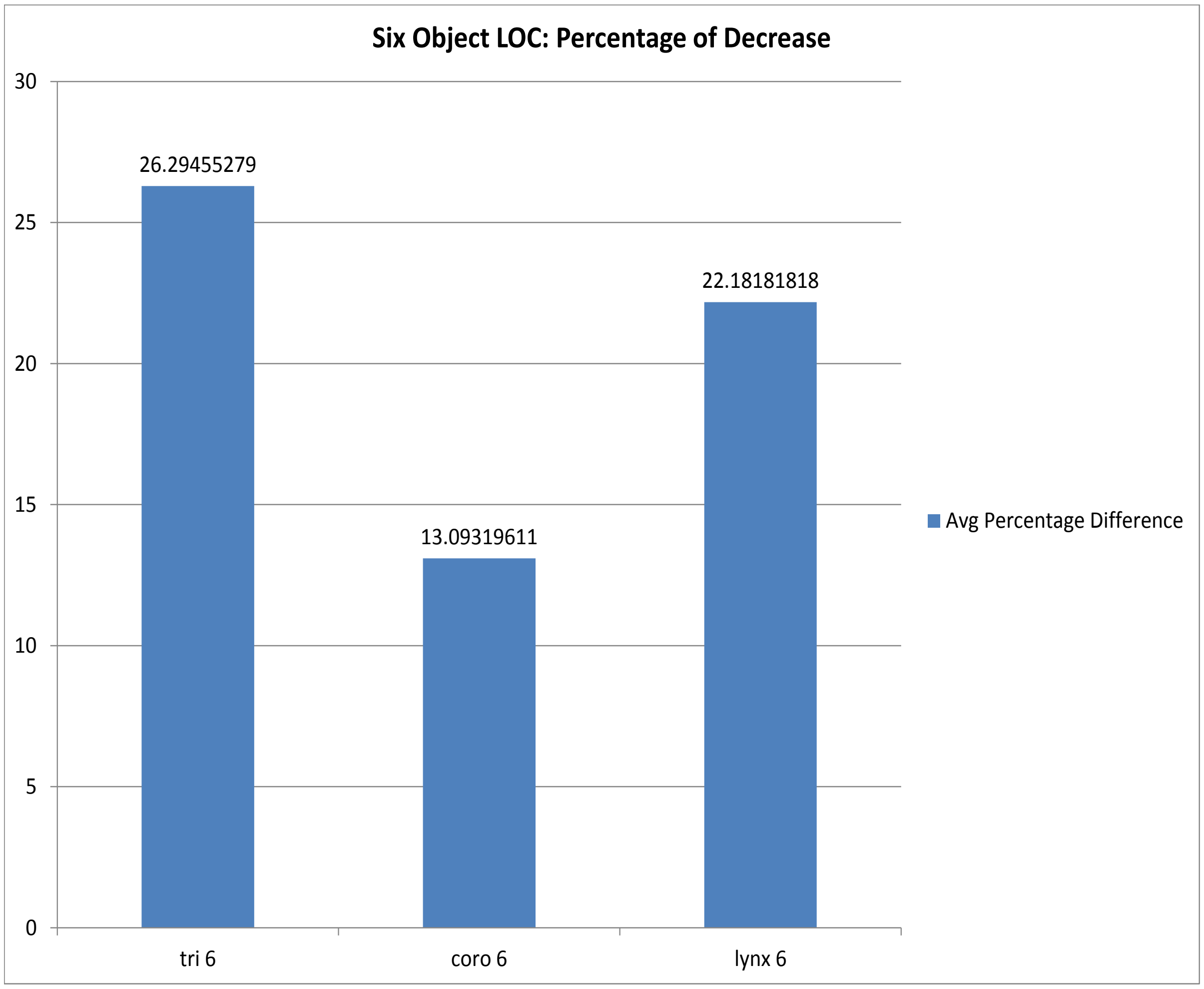


Figure 5.5: *The percentage of decrease in the LOC count of each robot's program that inserted six objects.*

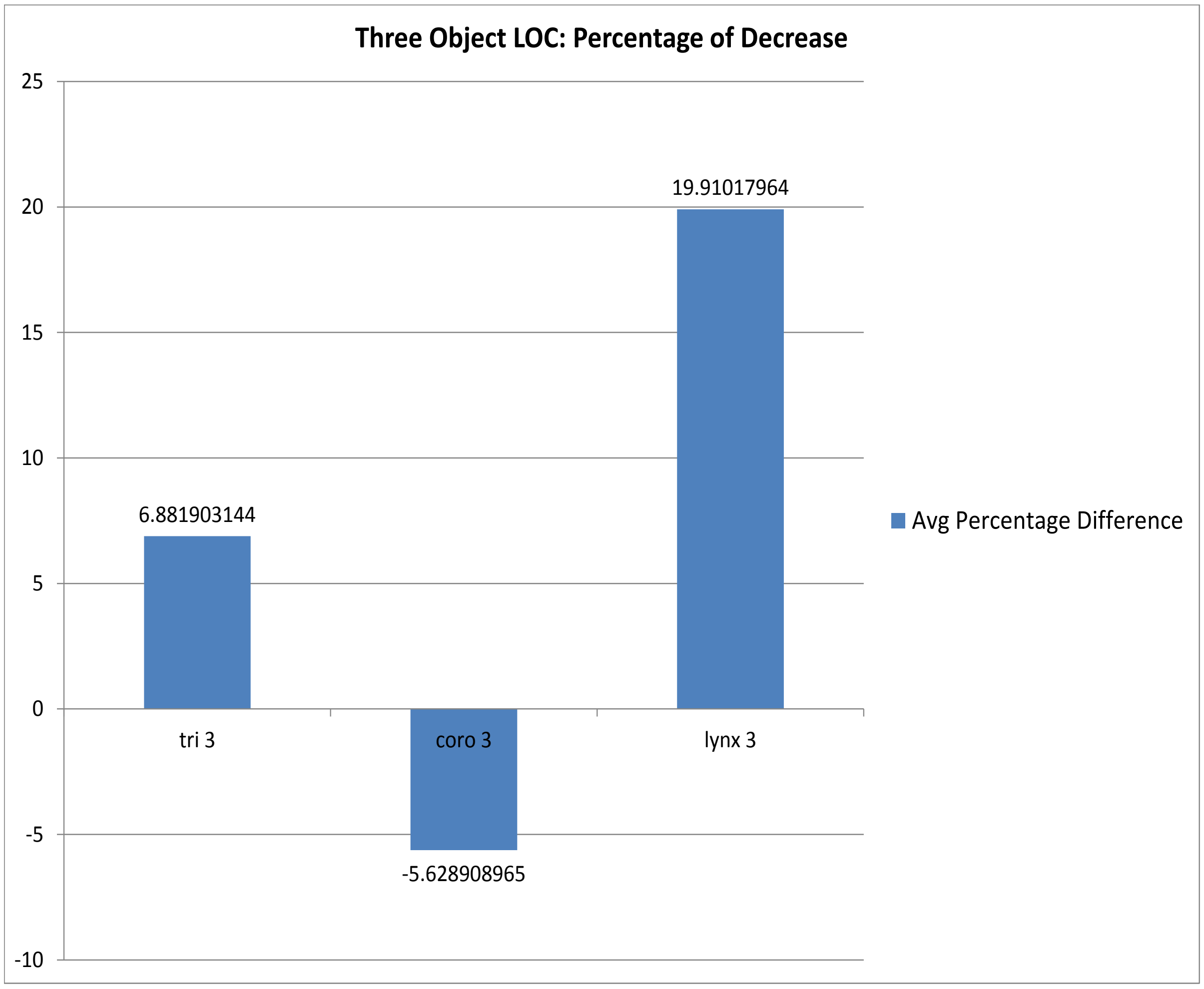


Figure 5.6: *The percentage of decrease in the LOC count of each robot's program that inserted three objects.*

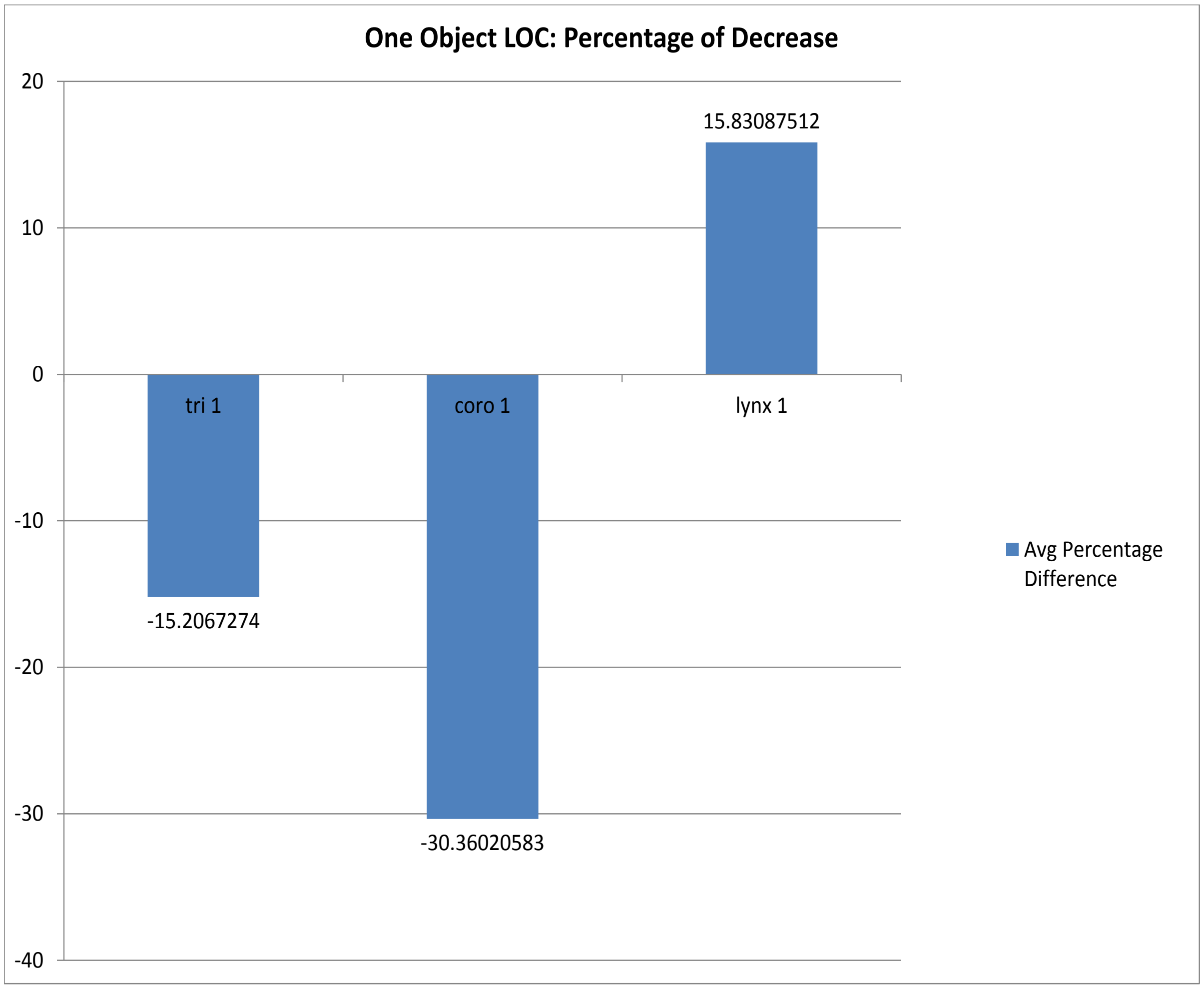


Figure 5.7: *The percentage of decrease in the LOC count of each robot's program that inserted one object.*

percentage difference at an average of 15.83%, also seen in Figure 5.7. What will be explored next is whether or not these differences in program line tallies are significant.

Significant Difference of Lines of Code

As previously discussed one of the determinations to be made from this research is whether or not using ROIL, the control group, decreases the number of lines in each robots code creating a more efficient method of coding, independent of the number of objects being inserted in the environment. In the previous section, the control groups LOC are generally shown to be fewer than the variable groups LOC across all programs, but are the differences between the line of code counts of any significance? Also, are the LOC of the variable group fewer than those of the control group *significantly* fewer? To determine if the numbers are significantly different a One Sample T-test was employed to make that determination. A One Sample T-test is utilized for statistics that involve a known or specified value that is compared to a sample mean to determine if that mean is significantly different. The One Sample T-test is a comparison of the average sample and the population with an adjustment for the number of cases in the sample and the standard deviation. The hypothesis to be examined states that the difference between the lines of code of ROIL and the variable programs is not zero. The null hypothesis states that the difference is zero. We wish to reject the null hypothesis to show a significant difference between the control and variable programs. Here, control groups LOC values from Table 5.1 are the specified values and the variable group mean values are analyzed. Tables 5.2–5.25 display the results of the comparisons of the two groups produced by R, a statistical software (Gentleman & Ihaka, 2011). "R is a language and environment for statistical computing and graphics." John Chambers and his colleagues at Bell Laboratories instigated this GNU project, which is a free software that compiles various operating systems while the user has full control over all functionality (including data analysis and graphics) (Gentleman & Ihaka, 2011).

Set of Nines The following tables (i.e. Tables 5.2, 5.3, 5.4, 5.5, 5.6, & 5.7) display the statistics and test results for the simulated robot programs that contain nine objects. In the tables called *stats* the abbreviation in the third row of the first column is short for the robot and the number of objects in the program. For example, if the Tribot has nine objects then it is abbreviated as tri9. The N is the number of experimental results (number of LOC counts) and mean is the average number of lines among the N programs. The tables below stats are test. The test value (also called mu) in this table is ROILs LOC count that is the sample of the robots programs are being compared to. The notation t is the standard deviation, and the result of the t-test. *Df* is the degree of freedom which is N-1 and lower and upper represent the confidence interval bounds. Recall from Figure 5.6 that for all three robot programs containing nine objects the LOC were greater than the control groups LOC. To determine significance the produced p-value was analyzed given a 95% confidence level. As seen in Tables 5.2 & 5.3, with a known test value (from Table 5.1) of 137 and 7 degrees of freedom, the LOC in the TriBot (tri9) programs was significantly reduced as the p-value, 7.50e-10, is less than 0.05.

Stats		
	N	Mean
tri9	8	231.5

Table 5.2: The input to retrieve results of TriBot with 9 objects

Test				
	Test value= 137			
		95% conf interval of the difference		
t	df	lower	upper	p-value
44.5477	7	226.4839	236.5161	7.50E-10

Table 5.3: The statistical results of TriBot with 9 objects

Displayed in Tables 5.4 & 5.5, with a known test value (from Table 5.1) of 190 and 7 degrees of freedom, the LOC count for the CoroBot (coro9) programs was significantly reduced as the p-value, 6.05e-09, is less than 0.05.

Stats		
	N	Mean
coro9	8	260.25

Table 5.4: The input to retrieve results of CoroBot with 9 objects

Displayed in Tables 5.6 & 5.7, with a known test value (from Table 5.1) of 1070 and 7 degrees of freedom, the LOC count for the Lynx6 (lynx9) programs was significantly reduced as the p-value, 8.38e-14, is less than 0.05.

Test				
	Test value= 190			
		95% conf interval of the difference		
t	df	lower	upper	p-value
33.018	7	255.219	265.281	7.50E-10

Table 5.5: The statistical results of CoroBot with 9 objects

Stats		
	N	Mean
lynx9	8	1418.25

Table 5.6: The input to retrieve results of Lynx with 9 objects

Test				
	Test value= 1070			
		95% conf interval of the difference		
t	df	lower	upper	p-value
163.6802	7	1412.219	1423.281	8.3E-14

Table 5.7: The statistical results of Lynx6 with 9 objects

Set of Sixes The following tables, 5.8, 5.9, 5.10, 5.11, 5.12 and 5.13, display the statistics and test results for the simulated robot programs that contain six objects. Recall from Figure 5.5 that for all three robot programs containing six objects the LOC were greater than the control groups LOC. To determine significance the produced p-value was analyzed given a 95% confidence level. As seen in Tables 5.8 & 5.9, with a known test value (from Table 5.1) of 137 and 7 degrees of freedom, the LOC in the TriBot (tri6) programs was significantly reduced as the p-value, 1.36e-07, is less than 0.05.

Stats		
	N	Mean
tri6	8	185.875

Table 5.8: The input to retrieve results of TriBot with 6 objects

Test				
	Test value= 137			
		95% conf interval of the difference		
t	df	lower	upper	p-value
21.0857	7	180.394	191.356	1.36E-07

Table 5.9: The statistical results of TriBot with 6 objects

Displayed in Tables 5.10 & 5.11, with a known test value (from Table 5.1) of 190 and 7 degrees of freedom, the LOC count for the CoroBot (coro9) programs was significantly reduced as the p-value, 6.86e-06, is less than 0.05.

Stats		
	N	Mean
coro6	8	185.875

Table 5.10: The input to retrieve results of CoroBot with 6 objects

Displayed in Tables 5.12 & 5.13, with a known test value (from Table 5.1) of 1070 and 7 degrees of freedom, the LOC count for the Lynx6 (lynx9) programs was significantly reduced as the p-value, 3.23e-13, is less than 0.05.

Test					
	Test value= 190				
		95% conf interval of the difference			
t	df	lower	upper	p-value
11.8663	7	212.9208	224.3292	6.86E-06

Table 5.11: The statistical results of CoroBot with 6 objects

Stats		
	N	Mean
lynx6	8	1375

Table 5.12: The input to retrieve results of Lynx6 with 6 objects

Test				
	Test value= 1070			
		95% conf interval of the difference		
t	df	lower	upper	p-value
134.9618	7	1369.656	1380.344	3.23E-13

Table 5.13: The statistical results of Lynx6 with 6 objects

Set of Threes Tables 5.14, 5.15, 5.16, 5.17, 5.18, and 5.19 display the statistics and test results for the simulated robot programs that contain three objects. Recall from Figure 5.6 that two of the three robot programs containing three objects had a positive percentage difference meaning there was a decrease in LOC after using ROIL. To determine significance the produced p-value was analyzed given a 95% confidence level. As seen in Tables 5.14 & 5.15, with a known test value (from Table 5.1) of 137 and 7 degrees of freedom, the LOC in the TriBot (tri3) programs was significantly reduced as the p-value, 0.002424, is less than 0.05.

Stats		
	N	Mean
tri3	8	147.125

Table 5.14: The input to retrieve results of TriBot with 3 objects

Test				
	Test value= 137			
		95% conf interval of the difference		
t	df	lower	upper	p-value
4.608	7	141.9436	152.3064	0.002424

Table 5.15: The statistical results of TriBot with 3 objects

CoroBot was the singular robot program in this group containing three objects that had a negative percentage difference. Displayed in Tables 5.16 & 5.17, with a known test value (from Table 5.1) of 190 and 7 degrees of freedom, the LOC count for the CoroBot (coro3) programs was significantly increased as the p-value, 2.19e-03, is less than 0.05.

Stats		
	N	Mean
coro3	8	179.875

Table 5.16: The input to retrieve results of CoroBot with 3 objects

Test				
	Test value= 190			
		95% conf interval of the difference		
t	df	lower	upper	p-value
-4.7092	7	174.7909	184.9591	2.19E-03

Table 5.17: The statistical results of CoroBot with 3 objects

Displayed in Tables 5.18 & 5.19, with a known test value (from Table 5.1) of 1070 and 7 degrees of freedom, the LOC count for the Lynx6 (lynx3) programs was significantly reduced as the p-value, 6.04e-13, is less than 0.05.

Stats		
	N	Mean
lynx3	8	1336

Table 5.18: The input to retrieve results of Lynx6 with 3 objects

Test				
	Test value= 1070			
		95% conf interval of the difference		
t	df	lower	upper	p-value
123.4494	7	1330.905	1341.095	6.04E-13

Table 5.19: The statistical results of Lynx6 with 3 objects

Set of Singles Tables 5.20, 5.21, 5.22, 5.23, 5.24, and 5.25 display the statistics and test results for the simulated robot programs that contain one object. Recall from Figure 5.7 that only one of the three robot programs containing one object had a positive percentage difference meaning there was a decrease in LOC after using ROIL. To determine significance the produced p-value was analyzed given a 95% confidence level. TriBot was the one of the two robot programs in this group containing a single object that had a negative percentage difference. As seen in Tables 5.20 & 5.21, with a known test value (from Table 5.1) of 137 and 11 degrees of freedom, the LOC in the TriBot (tri1) programs was significantly reduced as the p-value, 6.82e-10, is less than 0.05.

Stats		
	N	Mean
tri1	12	118.9167

Table 5.20: The input to retrieve results of TriBot with 1 object

Test				
	Test value= 137			
		95% conf interval of the difference		
t	df	lower	upper	p-value
-19.5518	11	116.881	120.9523	6.82E-10

Table 5.21: The statistical results of TriBot with 1 object

CoroBot was the other robot program in this group containing three objects that had a negative percentage difference. Displayed in Tables 5.22 & 5.23, with a known test value (from Table 5.1) of 190 and 11 degrees of freedom, the LOC count for the CoroBot (coro3) programs was significantly increased as the p-value, 1.91e-12, is less than 0.05.

Stats		
	N	Mean
coro1	12	145.75

Table 5.22: The input to retrieve results of CoroBot with 1 object

Displayed in Tables 5.24 & 5.25, with a known test value (from Table 5.1) of 1070 and 11 degrees of freedom, the LOC count for the Lynx6 (lynx3) programs was significantly reduced as the p-value, 9.60e-05, is less than 0.05.

Test				
	Test value= 190			
		95% conf interval of the difference		
t	df	lower	upper	p-value
-33.6508	11	142.8558	1148.6442	1.91E-12

Table 5.23: The statistical results of CoroBot with 1 object

Stats		
	N	Mean
lynx1	12	1271.25

Table 5.24: The input to retrieve results of Lynx6 with 1 object

Test				
	Test value= 1070			
		95% conf interval of the difference		
t	df	lower	upper	p-value
5.9494	7	1196.798	1345.702	9.60E-05

Table 5.25: The statistical results of Lynx6 with 1 object

5.1.2 Redundancy

Programmers, at times, create the same code again and again, which causes unnecessary sum of code lines and increases the programs intricacy since a change might be made in one of the routines. One could be left behind and create problems (Teach me how to, 2011). Debugging becomes hard as the lines look so similar that finding the problem is next to impossible. Also redundancy in code increases the time it takes for code to run and process (Php Web Scripting, n.d.). Thus, to measure optimization using the ROIL system, redundant code within each program was discovered and tallied. Where lines of code were measured per robot and set of objects, contrarily, redundancy was only measured per set of objects. This is because the only part of each program that was counted was the lines that exploited the differences between the redundancy of the variable and control groups, specifically the code that creates and inserts the objects into each environment. This was permissible due to the fact that all objects were coded in the same language, causing each robot to have no differentiation. Therefore, redundancy in the variable group is partially based on the number of objects. Only one program was necessary for experimentation using ROIL, thus, the control group has a single value which totals to 14.

The following table, Table 5.26, displays the redundant code counts of each program by number of objects - nine, six and three.

8 programs	**Number of objects**		
	nine	six	three
a	42	21	14
b	42	21	14
c	42	21	14
d	42	21	14
e	42	21	10
f	42	28	10
g	42	21	10
h	42	35	10

Table 5.26: The counts of redundancy of each program by number of objects

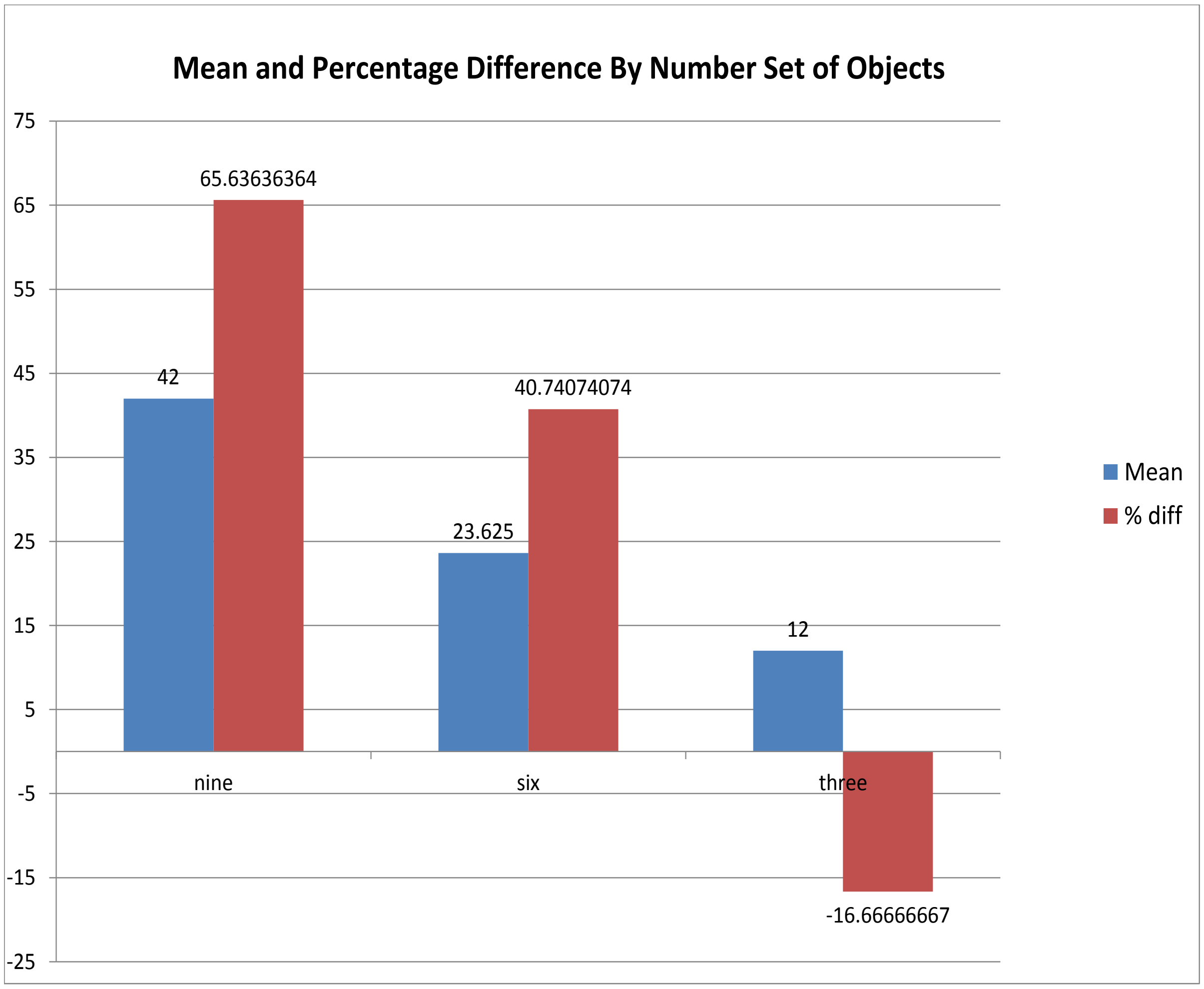


Figure 5.8: *The mean and percentage of difference by number set of objects*

For the eight programs written in the control group that contain nine objects, the amount of redundancy is consistent with forty-two lines of redundant code per program. It is seen that for the nine-object programs the number of redundancies is reduced using ROILs code. The mean, 42, is higher than 14 and there is a 65.64% decrease in redundancy, in Figure 5.8.

For the eight programs written in the control group that contain six objects, the amount of redundant code is reduced using ROILs code. The mean, 23.63, is higher than 14 and there is a 40.74% decrease in redundancy in Figure 5.8.

Table 5.26 displays the results of the redundancy counts within the eight written programs of the variable group of three objects. The mean, 12, is lower than 14 and there is a 16.67% increase in redundancy using ROIL, which is seen in Figure 5.8.

Table 5.27 displays the results of the redundancy counts within the twelve written programs of the variable group where each contains a single object. As seen in the table, the number of redundancies within each program is zero as one object is coded. Zero is, of course, less than the fourteen redundancies of the control group. As previously mentioned, the redundancies are related to the number of objects written in the code, therefore, no redundancies here is expected.

Programs	**Redundancies**
marble	0
baseball	0
book	0
box	0
cone	0
die	0
domino	0
golfball	0
mechanical pencil	0
prescription bottle	0
ringbox	0
soda bottle	0

Table 5.27: The number of redundancies among the programs with one object.

The next section looks to discover whether or not any of these differences in redundancy are significant.

Significant Different of Redundancies

As previously discussed, the research presented here seeks to determine if using the ROIL system, the control group, decreases redundancy in each robots code creating a more optimal method of coding, independent of the number of object being inserted in the environment. In the previous section, the redundancy of the ROIL system has been shown to be generally lower than the variable groups. Are the differences between the redundancies of each group of any significance? Also, are the redundancies of the variable group that are fewer than those of the control group *significantly* fewer? To determine if the numbers are significantly different a One Sample T-test was employed to make that determination. Here, control groups redundancy value 14 is the specified value and the variable group mean values are analyzed. Tables 5.28, 5.29, 5.30, 5.31, 5.32, 5.33, 5.34 and 5.35 reveal the results of those comparisons produced by R.

Tables 5.28 & 5.29 display the statistics and test results for redundancy among the coding that contain nine objects. Recall from Figure 5.8 codes containing nine objects had a positive percentage difference meaning there were decreases in LOC after using ROIL. To determine significance the produced p-value was analyzed given a 95% confidence level. Unfortunately, as seen in Tables 5.28 & 5.29, with a known test value of 14 and 7 degrees of freedom, the software returned an error quoting data are essentially constant due to the amount of redundancy being the exact same amongst all code containing nine objects. Therefore, no p-value was returned and significance could not be determined by the software.

Stats		
	N	Mean
redof9	8	42

Table 5.28: The input to retrieve results of redundancy among programs with 9 objects

Test				
	Test value= 14			
		95% conf interval of the difference		
t	df	lower	upper	p-value
N/A	7	N/A	N/A	N/A

Table 5.29: The statistical results of redundancy among programs with 9 objects

Recall from Figure 5.8 codes containing six objects had a positive percentage difference meaning there were decreases in redundancy after using ROIL. To determine significance the produced p-value was analyzed given a 95% confidence level. After employing the one sample t-test, the degree of freedom produced 7, the p-value, 1.22e-03, is less than 0.05, seen in Tables 5.30 & 5.31, therefore, the redundancy in ROIL code is significantly lower than that of the variable group containing six objects.

Stats		
	N	Mean
redof6	8	23.625

Table 5.30: The input to retrieve results of redundancy among programs with 6 objects

Test				
	Test value= 14			
		95% conf interval of the difference		
t	df	lower	upper	p-value
5.2271	7	19.27086	27.97914	1.22E-03

Table 5.31: The statistical results of redundancy among programs with 6 objects

The result of this t-test leads to the understanding that objects containing nine objects also have a significantly decreased amount of redundancy upon using the ROIL system. Recall from Figure 5.8 that codes containing three objects had a negative percentage difference meaning there was an increase in redundancy after using ROIL. To determine significance the produced p-value was analyzed given a 95% confidence level. Seen in Tables 5.32 & 5.33, after employing the One Sample T-test, the degree of freedom produced 7, the p-value, 3.32e-02, is less than 0.05, therefore, the redundancy in ROIL code is significantly greater than that of the variable group containing three objects.

Stats		
	N	Mean
redof3	8	12

Table 5.32: The input to retrieve results of redundancy among programs with 3 objects

Test				
	Test value= 14			
		95% conf interval of the difference		
t	df	lower	upper	p-value
-2.6458	7	10.21251	13.78749	3.32E-02

Table 5.33: The statistical results of redundancy among programs with 3 objects

There is a presumed increase in redundancy due the fact that there is an entire lack of redundancy amongst the coding of a single object. Unfortunately, due to the consistency of zero redundancy, the t-test essentially failed, as seen in Tables 5.34 & 5.35. However, despite the failure, a p-value was returned by the R application with a degree of freedom of 11 and a p-value that is less than 2.2e-16. Therefore, there is a significant increase in redundancy using ROIL compared to the variable group of a single objects.

Stats		
	N	Mean
redof1	12	0

Table 5.34: The input to retrieve results of redundancy among programs with 1 object

Test				
	Test value = 14			
		95% conf interval of the difference		
t	df	lower	upper	p-value
-Inf	11	NaN	NaN	<2.2E-16

Table 5.35: The statistical results of redundancy among programs with 1 object

Chapter 6

Summary and Conclusion

6.1 Summary

The ROIL system was created to facilitate robot programming, specifically code that creates objects that the robots may interact with. ROILs main purpose is to provide code that will give more efficient, optimally functioning code chunks as well as reduce the learning curve of the programmer. Through experimentation the ROIL system has demonstrated that it is a viable option for programming object code into simulated environments for robots. The ROIL system reduced the lines of code for all robot programs that contained nine and six objects. Tests showed that there was a significant reduction is lines of code in these categories. There was also a significant decrease in lines of code of the Lynx6 robot arm with three and one object(s) and the TriBot with three objects , as well. However, it must be noted that there was a significant increase in lines of code for the CoroBot programs that contained three objects and one object as well as the TriBot program written with a single object.

When comparing redundancy of the ROIL system to that of the variable group the results were evened out. The redundancy of objects with nine and six objects was significantly reduced using the ROIL system. Recall that the statistical significance for programs with nine objects was determined by the significance of six-object codes due to the fact that the software was unable to return a p-value because the amount of redundancy didnt change amongst all programs with nine objects. Also note, that the redundancy within code with three or less objects was significantly less than the ROIL code.

6.1.1 Conclusion

This research has shown that the ROIL system is a viable option to programming objects into simulated robot programs. The ROIL system significantly reduced the lines of code for programs that use six or more objects among all three robots simulated. In addition, the ROIL system reduced the lines of code count within the Lynx6 and TriBot programs that use three objects and the Lynx6 programs that only had one object. Also, redundancy was reduced when ROIL was used in programs that had six or more objects.

Research on writing more efficient, optimal code is minimal and usually debated via programming bloggers using small chunks of code with quick measurement of completion times, number of runs or loops, or readability. Furthermore, programmers as a result of these unreliable determinations often shun using lines of code as a unit of measurement. This shunning is also backed by the fact that more often than not researchers are comparing lines of code amongst various languages (Tuxtips.org, 2011). However, it is found to be quite useful and a reasonable measure taken when used to compare programs that are written in the same language to determine progress or complexity (Tuxtips.org, 2011). Lines of code are a starting point to determine if there may be any difference at all amongst the programs. It is also understood that less code correlates to fewer bugs and errors (Cunningham & Cunningham, Inc., n.d.). It is determined by this research that there is an overall significant difference when using the ROIL system when comparing lines of code. Like lines of code, redundancy is also a controversial measurement, but again, similarly, reduces the likelihood of bugs and also makes the code easier to read and debug (Teach me how to, 2011). However, further research is needed in order to determine other qualities of ROIL such as readability and process time completion to further provide a basis for efficiency and optimality.

6.1.2 Contributions

This dissertation research has made the following contributions to the field of Human Robot Interaction:

- A grammar written in XML was established to create and show a commonality in robot machines and robot programming .
- The learning curve for robot programming for object manipulation was decreased due to the systems capability of generalizing object manipulation per language and delivering the code to the user.
- This dissertation demonstrates the ability for a robot to approach and optimally grip any object within its capabilities and limitations.

6.1.3 Directions for Future Research

There are a number of questions and concerns that remain concerning this dissertation and the direction that it should take:

1. The research experimentation was done in simulated in environments. Many institutions use simulated environments to perform studies. Therefore, other simulated environments must be tested. Additionally, despite their frequent use in robotics research, simulated environments are not adequate enough to resemble a real environment which is the greater target for the end purpose of this study. Therefore, real robots and objects must be acquired and used in the future to determine that the results are same.
2. In this study, only one programming language was used to code the robots for experimentation. The number and variations of languages, in both simulated and real environments, must be increased in order for ROIL to be determined useful in the community.
3. RFID devices and tags will be used to gain a more accurate description and position of the worlds infinite, unique objects. In addition, cameras and color tags will be used to broaden the studys capabilities and for use of the various studies at various institutions.

4. Due to the fact that measuring efficiency using the counting lines of code is often considered debatable, research will also continue exploring other performance measurements including time of completion, manipulation accuracy, identification accuracy, manipulation optimality, and code error frequency.

References

Ahn, S. C., Lee, J. W., Lim, K. W., Ko, H., Kwon, Y. M., & Kim, H. G. (2006). Upnp sdk for robot development. In *Siceicase international joint conference* (pp. 363–368).

Anderson, M., Thaete, L., & Wiegand, N. (2007). Player/stage: A unifying paradign to improve robotics education delivery. In *Workshop on research in robots for educatoin at robotics: science and systems conference.*

Apache Friends - XAMPP. (n.d.). `http://www.apachefriends.org/en/xampp.html`. ([Online; accessed 5-March-2012])

Bakken, D. E. (2003). Middleware. In *Encyclopedia of distributed computing.* Kluwer Academic Press.

Brooks, D. R. (1999). *C programming: the essentials for engineers and scientists.* Springer–New York.

Charlton, J. (2008). *Lines of Code as a Measure of Progress.* `http://devlicio.us/blogs/casey/archive/2008/05/16/lines-of-code-as-a-measure-of-progress.aspx`. (Web log comment)

Computer Desktop Encyclopedia. (2008). *USB.* `http://www.pcmag.com/encyclopedia_term/0,2542,t%3DUSB&i%3D53531,00.asp`. The Computer Language Company Inc.

CORBA FAQ. (n.d.). `http://www.omg.org/gettingstarted/corbafaq.htm`. ([Online; accessed 2-March-2012])

CoroWare. (n.d.). *CoroBot Classic programmable mobile robot platform.* `http://robotics.coroware.com/corobot`. ([Online; accessed 2-March-2012])

Cunningham & Cunningham, Inc. (n.d.). *Lines of code.* `http://www.c2.com/cgi/wiki?LinesOfCode`. (Web log comment)

DelRossi, R. A. (1993). Visual c++ is a strong development tool. *InfoWorld*, *25*, 101.

Edsinger, A., & Kemp, C. C. (2006). *Manipulation in human environments* (Nos. 102–109). (Humanoid Robots)

Edsinger, A., & Kemp, C. C. (2007). *Human-robot interact for cooperative manipulation: Handing objects to one another.*

Edsinger, A., & Kemp, C. C. (2008). *Two arms are better than one: A behavior based control system for assistive bimanual manipulation* (Vol. 370). Springer.

Gentleman, R., & Ihaka, R. (2011). *R (2.14.1).* `http://cran.cnr.berkeley.edu/bin/windows/`. (Software)

Gill, C. D., & Smart, W. D. (2002). Middleware for robots. In *Intelligent distributed and embedded systems: Papers from the 2002 aaai spring symposium* (pp. 1–5).

Gostai. (n.d.). *Urbi.* `http://www.gostai.com/index.php`.

Grabianowski, E., & Tyson, J. (2001). *How PCI Works: Plug and Play.* `http://computer.howstuffworks.com/pci4.htm`.

Grimes, R. A. (2001). *Malicious mobile code: virus protection for windows.* O'Reilly & Associates–Sebastopol, CA.

Hertzfeld, A. (2011). Revolution in the valley: The insanely great story of how the mac was made. In (chap. -2000 Lines of Code). O'Reilly Media.

Jagiello, J., Tay, N., Eronen, M., Fernhill, D., & Park, F. (2006). A robotic middleware. *Components*, 1–6.

James, J. (2007). *Is SLOC a valid measure of quality or efficiency?* `http://www.techrepublic.com/blog/programming-and-development/is-sloc-a-valid-measure-of-quality-or-efficiency/499`. (Web log comment)

Johns, K., & Taylor, T. (2008). *Professional microsoft robotics developer studio.* Wiley Publishing, Inc.–Indianapolis.

Katz, J. K., & McCormick, D. L. (2000). *The encyclopedia of trading strategies.* McGraw-Hill–New York.

Kemp, C., Edsinger, A., & Torres-Jara, E. (2007). Challenges for robot manipulation in human environments. *Perception*, *1*(14), 20–29.

Kemp, C. C., Anderson, C. D., Nguyen, H. T., Trevor, A. J., & Xu, Z. (2008). A point-and-click interface for the real world: laser designation of objects for mobile manipulation. *Proceedings of the 3rd ACMIEEE international conference on Human robot interaction*, 241–248. (ACM)

Lapham, J. (1999). Robotscript: The introduction of a universal robot programming language. *The Industrial Robot*, *1*, 17.

Lego Mindstorms NXT. (1999). `http://mindstorms.lego.com/en-us/Default.aspx`.

LineTally. (2008). `http://download.cnet.com/LineTally/3000-2229_4-10785145.html#ixzz1nn5orMF1`. (Software)

Lucas, A. (2008). *Lines of Code - Dispelling The Myths.* `http://www.callingshotgun.net/geekery/lines-of-code-dispelling-the-myths/`. (Web log comment)

Lynxmotion. (n.d.). *AL5A Robotics Arm CoroBot Kit.* `http://www.lynxmotion.com/c-27-robotic-arms.aspx`.

Makatchev, M., & Tso, S. K. (2000). Human-robot interface using agents communicating in an xml-based markup language. *Proceedings 9th IEEE International Workshop on Robot and Human Interactive Communication IEEE ROMAN 2000*, 270–275. (IEEE)

Marx, D. (2008). *Lines of Code and Unintended Consequences.* `http://marxsoftware.blogspot.com/2008/09/lines-of-code-and-unintended.html`. (Web log comment)

McConnell, S. (1993). *Code Complete: A Practical Handbook of Software Construction.* Microsoft Press – Redmond, WA.

Merriam-Webster.com. (n.d.). *Manipulate.*

Microsoft. (n.d.). *Microsoft Robotics Developer Studios 2008 R3 (2.2.76.0).* `http://www.microsoft.com/download/en/details.aspx?id=17386`. (Software)

Microsoft. (2003). *How Plug and Play Works.* `http://technet.microsoft.com/en-us/library/cc781092(v=ws.10).aspx`.

Monkman, G. J. (2007). *Robot Grippers* (1st ed.). Wiley-VCH.

Mueller, J. (2009). *C design and development.* Wrox.

Newmarch, J. (2010). *Jan Newmarchs Guide to Jini Technologies (1st ed.* `http://jan.newmarch.name/java/jini/tutorial/Jini.html`. APress.

Nguyen, H., Anderson, C., Trevor, A., Jain, A., Xu, Z., & Kemp, C. C. (2008). El-e : An assistive robot that fetches objects from flat surfaces. *Robotics*.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., & Pignol, M. (2002). Generating remote control interfaces for complex appliances. *Proceedings of the 15th annual ACM symposium on User interface software and technology UIST 02*, *2*, 161–170.

Norman, D. (1990). *The design of everyday things.* Doubleday Business–New York.

Pappas, C. H., & Murray, W. H. (1995). *The visual c++ handbook.* Osborne McGraw-Hill–Berkeley, Calif.

Php Web Scripting, L. (n.d.). *Elimination of code redundancy.* `http://phpwebscripting.com/phpwebsite/index.php?module=article&view=4&page_num=2`.

Saxena, A., Driemeyer, J., Kearns, J., Osondu, C., & Ng, A. Y. (2006). Learning to grasp novel objects using vision. *Learning*, *12*(39), 1–10.

Simon, R., Nagel, C., Watson, K., Glynn, J., Skinner, M., & Evjen, B. (n.d.). *Professional c#.*

SimplySim. (n.d.). `http://www.simplysim.net/DL/NXT-MSRDS-R3.msi`. (Software package)

Smart, W. D. (2007). Is a common middleware for robotics possible? *Development*.

Smith, P. (2011). *Software build systems: principles and experience.* Addison Wesley–Upper Saddle River, NJ.

Stone, P. (2007). Intelligent autonomous robotics: A robot soccer case study. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *1*.

Teach-ICT.com. (n.d.). *Software Redundancy.* `http://www.teach-ict.com/gcse_computing/ocr/211_hardware_software/reliability/miniweb/pg6.htm`.

Teach me how to. (2011). *Redundant code.* `http://www.teach-ict.com/gcse_computing/ocr/211_hardware_software/reliability/miniweb/pg6.htm`. (Web log comment)

Tuxtips.org. (2011). *Lines Of Code The Most Meaningless Metric.* `http://www.tuxtips.org/?p=4`. (Web log comment)

Universal Serial Bus. (n.d.). *What is UPnP? UPnP Forum.* `http://http://www.usb.org/about/features/`.

Waloszek, G. (2005). *Universal Remote Console.* `http://www.sapdesignguild.org/editions/edition9/urc.asp`. (SAP User Experience, SAP AG)

Warth, A. (2011). *Experimenting with programming languages.* ProQuest, UMI Dissertation Publishing.

Wörn, H., Wurll, C., & Henrich, D. (1998). Automatic off-line programming and motion planning for industrial robots. In *Isr98, 29th international symposium on robotics.*

Zimmermann, G., Vanderheiden, G., & Gilman, A. (2003). Universal remote console - prototyping for the alternate interface access standard. *Universal Access Theoretical Perspectives Practice and Experience*, 524–531.

Appendices

Appendix A
MySQL Database

```
-- Database: `roildb`
-- Table structure for table `code`
--

CREATE TABLE IF NOT EXISTS `code` (
  `codeID` int(11) NOT NULL AUTO_INCREMENT,
  `programminglanguage` int(3) DEFAULT NULL,
  `pasteInFile` longtext,
  `ROILfile` mediumblob,
  `instructions` text,
  PRIMARY KEY (`codeID`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

--
-- Table structure for table `programminglanguages`
--

CREATE TABLE IF NOT EXISTS `programminglanguages` (
  `plID` int(3) DEFAULT NULL,
  `plname` varchar(35) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- --------------------------------------------------------

--
-- Table structure for table `robotinfo`
--

CREATE TABLE IF NOT EXISTS `robotinfo` (
  `robotID` int(11) NOT NULL AUTO_INCREMENT,
  `modelID` int(10) NOT NULL DEFAULT '0',
  `robot_manufacturer` varchar(100) DEFAULT NULL,
  `robot_brand` varchar(100) DEFAULT NULL,
  `modelNumber` varchar(100) DEFAULT NULL,
  `robotname` varchar(150) DEFAULT NULL,
  `essential_parts` varchar(250) DEFAULT NULL,
  `additional_parts` varchar(250) DEFAULT NULL,
  `built_how` varchar(25) DEFAULT NULL,
  `gripper_span_max` float DEFAULT NULL,
  `gripper_span_min` float DEFAULT NULL,
  `payload` float DEFAULT NULL,
  `height_inches_max` float DEFAULT NULL,
  `height_inches_min` float DEFAULT NULL,
  `end_effector_exists` char(3) DEFAULT NULL,
  `baseheight` float DEFAULT NULL,
  `category` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`robotID`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=7 ;
```

```
-- --------------------------------------------------------

--
-- Table structure for table `sim_objects`
--

CREATE TABLE IF NOT EXISTS `sim_objects` (
  `objectID` int(11) NOT NULL AUTO_INCREMENT,
  `objectName` varchar(50) DEFAULT NULL,
  `category` varchar(50) DEFAULT NULL,
  `o_weight_max` float(20,5) DEFAULT NULL,
  `o_weight_min` float(20,5) DEFAULT NULL,
  `o_height_inches_max` float(20,5) DEFAULT NULL,
  `o_height_inches_min` float(20,5) DEFAULT NULL,
  `fidcolor1` char(11) DEFAULT NULL,
  `fidcolor2` char(11) DEFAULT NULL,
  `fidcolor3` char(11) DEFAULT NULL,
  `o_weight` float(20,5) DEFAULT NULL,
  `o_heightY` float(20,5) DEFAULT NULL,
  `o_widthX` float(20,5) DEFAULT NULL,
  `o_depthZ` float(20,5) DEFAULT NULL,
  `orientation` varchar(100) DEFAULT NULL,
  `generally_found_in` varchar(50) DEFAULT NULL,
  `o_width_inches_max` float(20,5) DEFAULT NULL,
  `o_width_inches_min` float(20,5) DEFAULT NULL,
  `gen_geometric_shape` varchar(30) DEFAULT NULL,
  `o_depth_inches_max` float(20,5) DEFAULT NULL,
  `o_depth_inches_min` float(20,5) DEFAULT NULL,
  `1 or 2 hands?` tinyint(1) DEFAULT NULL,
  PRIMARY KEY (`objectID`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=14 ;
```

Appendix B

XML Structure - Robot

Listing B.1: Empty object XML structure

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<all_objects>
        <object name='' category=''>
                <properties>
                        <gen_geom_shape></gen_geom_shape>
                        <heightY></heightY>
                        <weight></weight>
                        <widthX></widthX>
                        <depthZ></depthZ>
                        <weight_max></weight_max>
                        <weight_min></weight_min>
                        <height_max></height_max>
                        <height_min></height_min>
                        <width_max></width_max>
                        <width_min></width_min>
                        <depth_max></depth_max>
                        <depth_min></depth_min>
                        <generally_found_in></generally_found_in>
                </properties>
        </object>
</all_objects>
```

Listing B.2: Example of a street cone data

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<all_objects>
        <object name='cone' category=''>
                <properties>
                        <gen_geom_shape></gen_geom_shape>
                        <heightY></heightY>
                        <weight></weight>
                        <widthX></widthX>
                        <depthZ></depthZ>
                        <weight_max>10.00000</weight_max>
                        <weight_min>1.50000</weight_min>
                        <height_max>36.00000</height_max>
                        <height_min>12.00000</height_min>
                        <width_max></width_max>
                        <width_min></width_min>
                        <depth_max></depth_max>
                        <depth_min></depth_min>
                        <generally_found_in></generally_found_in>
                </properties>
        </object>
</all_objects>
```

Listing B.3: Example of an official golfball data

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<all_objects>
        <object name='golfball' category=''>
        <properties>
                <gen_geom_shape></gen_geom_shape>
                <heightY></heightY>
                <weight></weight>
                <widthX></widthX>
                <depthZ></depthZ>
                <weight_max>1.62000</weight_max>
                <weight_min>0.00000</weight_min>
                <height_max></height_max>
                <height_min>1.68000</height_min>
                <width_max></width_max>
                <width_min></width_min>
                <depth_max></depth_max>
                <depth_min></depth_min>
                <generally_found_in></generally_found_in>
        </properties>
    </object>
</all_objects>
```

Appendix C

XML Structure - Object

Listing C.1: Empty robot XML structure

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<the_robot>
<therobot robotID='' modelID='' name='Lynx6' manu='' brand=''>
        <properties>
                <gripper_span_max></gripper_span_max>
                <gripper_span_min></gripper_span_min>
                <payload></payload>
                <height_inches_max></height_inches_max>
                <height_inches_min></height_inches_min>
                <category></category>
        </properties>
</therobot>
</the_robot>
```

Listing C.2: Example of an the Lynx6 Arm data

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<the_robot>
<therobot robotID='4' modelID='0' name='Lynx6' manu='Lynxmotion'
brand='Lynxmotion'>
        <properties>
                <gripper_span_max>1.25</gripper_span_max>
                <gripper_span_min>0</gripper_span_min>
                <payload>4</payload>
                <height_inches_max>14</height_inches_max>
                <height_inches_min>0</height_inches_min>
                <category>Impactive</category>
        </properties>
</therobot>
</the_robot>
```

Appendix D

ROILfile - C#

```
  public class ROIL
{
        public ROIL()
        {
    }

    int objcount = 0;
    static string[,] allobjects; // = new string[i, 7];
    string height;
    string width;
    float widthf;
    string depth;
    float depthf;
    float height_max = 0;
    float height_min;
    float width_max = 0;
    float width_min;
    float depth_max = 0;
    float depth_min;
    float heightf;

    string fiducial_rgb1 = "";
```

```
#region
public string[,] Readfile()
{
    string phpfile = "http://localhost/diss/selectedobject.xml";
    Uri httpuri = new Uri(phpfile);

    string xmlfile = Convert.ToString(DisplayFileFromServer(httpuri));
    //string xmlfile = Convert.ToString(ftpuri);

    #region XMLReader

    int i = 0;

    using (XmlReader reader = XmlReader.Create(new StringReader(xmlfile))
    {
        while (reader.Read())
        {
            // Only detect start elements.
            if (reader.IsStartElement())
            {
                // Get element name and switch on it.

                if (reader.Name == "object")
                {
                    ++objcount;
                }
                }


            }
        }
       allobjects = new string[objcount, 7];


        string ggs = "";
        string name_attr;
        string weight = "";
        float weightf;
        float weight_max = 0;
        float weight_min;



        using (XmlReader reader = XmlReader.Create(new StringReader(xmlfile))
        {
            reader.ReadToDescendant("object");

            do
            {
                if (reader.IsStartElement())
                {
                    string name = reader.Name;

                    switch (reader.Name)
                    {
                        case "all_objects":
```

```
case "object":
    name_attr = reader["name"];
    allobjects[i, 0] = name_attr;
    break;
case "properties":
    break;
case "gen_geom_shape":
    if (reader.Read())
    {
        if (reader.Value == "sphere")
        {
            allobjects[i, 1] = "sphere";
        }
        else if (reader.Value == "box")
        {
            ggs = "box";
            allobjects[i, 1] = ggs;
        }
        else if (reader.Value == "capsule")
        {

            ggs = "capsule";
            allobjects[i, 1] = ggs;
        }
        else if (reader.Value == "rectangle")
        {
            ggs = "rectangle";
```

```
                                        allobjects[i, 1] = ggs;
                                    }
                                    else Console.WriteLine("no shape");
                                    //break;
                                }
                                break;
                            case "heightY":
                                if (reader.Read())
                                {
                                    string holdvalue = reader.Value.Trim();
                                    if ((!string.IsNullOrEmpty(holdvalue)) ||
(holdvalue != "0"))
                                    {
                                        height = reader.Value;
                                        allobjects[i, 5] = height;
                                    }
                                    else { height = ""; }
                                }
                                break;
                            case "weight":
                                if (reader.Read())
                                {
                                    string holdvalue = reader.Value.Trim();
                                    if ((!string.IsNullOrEmpty(holdvalue)) ||
(holdvalue != "0"))
                                                                {
                                        weight = reader.Value;
```

```
                    allobjects[i, 2] = weight;
                    //goto case "depthZ";
                }
                else
                {
                    weight = "";
                }
            }
            break;
        case "widthX":
            if (reader.Read())
            {
                string holdvalue = reader.Value.Trim();
                if ((!string.IsNullOrEmpty(holdvalue)) ||
(holdvalue != "0"))
                {
                    width = reader.Value;
                    allobjects[i, 4] = width;
                }
                else { width = ""; }
            }
            break;

        case "depthZ":
            if (reader.Read())
            {
                string holdvalue = reader.Value.Trim();
```

```
                                        if ((!string.IsNullOrEmpty(holdvalue)) ||
(holdvalue != "0"))
                                        {
                                            depth = reader.Value;
                                            allobjects[i, 3] = depth;
                                        }
                                        else { depth = ""; }
                                    }
                                    break;

                                case "weight_max":
                                    if (reader.Read())
                                    {
                                        string holdvalue = reader.Value.Trim();
                                        string compare = "";
                                        if (holdvalue != compare)
                                        {
                                            weight_max = Convert.ToSingle
(reader.Value);
                                            //goto case "weight_min";
                                        }
                                        else { weight_max = 0; }
                                    }
                                    break;
                                case "weight_min":
                                    if (reader.Read())
                                    {
                                        string holdvalue = reader.Value.Trim();
                                        string compare = "";
                                        if (holdvalue != compare)
                                        {
                                            //weight_max += 0;
                                            weight_min = Convert.ToSingle
(reader.Value);
                                            weightf = (weight_max + weight_min)/2f;
                                            weight = Convert.ToString(weightf);
                                            allobjects[i, 2] = weight;
                                            //goto case "depthZ";
                                            //continue;
                                        }
                                        else { weight_min = 0; }
                                    }
                                    break;
                                case "height_max":
                                    if (reader.Read())
                                    {
                                        string holdvalue = reader.Value.Trim();
                                        string compare = "";
                                        if (holdvalue != compare)
                                        {
                                            height_max = Convert.ToSingle
(reader.Value);
                                            //goto case "height_min";
                                        }
```

```
                    else
                    {
                        //goto case "heightY";
                        height_max = 0;
                    }
                }
                break;
            case "height_min":
                if (reader.Read())
                {
                    string holdvalue = reader.Value.Trim();
                    string compare = "";
                    if (holdvalue != compare)
                    {
                        height_min = Convert.ToSingle
(reader.Value);
                        //take avg diameter
                        heightf = (height_max + height_min)/2f;
                        height = Convert.ToString(heightf);
                        allobjects[i, 5] = height;
                        //goto case "fiducial_rgb1";
                    }
                }
                break;
            case "width_max":
                if (reader.Read())
                {
```

```
                                        string holdvalue = reader.Value.Trim();
                                        string compare = "";
                                        if (holdvalue != compare)
                                        {
                                            width_max = Convert.ToSingle
(reader.Value);
                                        }
                                        else
                                        { //goto case "widthX";
                                            width_max = 0;
                                        }
                                    }
                                    break;
                                case "width_min":
                                    if (reader.Read())
                                    {
                                        string holdvalue = reader.Value.Trim();
                                        string compare = "";
                                        if (holdvalue != compare)
                                        {
                                            width_min = Convert.ToSingle
(reader.Value);
                                            widthf = (width_max + width_min)/2f;
                                            width = Convert.ToString(widthf);
                                            allobjects[i, 4] = width;
                                        }
                                        else { width_min = 0; }
```

```
                    }
                    break;
                case "depth_max":
                    if (reader.Read())
                    {
                        string holdvalue = reader.Value.Trim();
                        string compare = "";
                        if (holdvalue != compare)
                        {
                            depth_max = Convert.ToSingle
(reader.Value);
                        }
                        else { depth_max = 0; }
                    }
                    break;
                case "depth_min":
                    if (reader.Read())
                    {
                        string holdvalue = reader.Value.Trim();
                        string compare = "";
                        if (holdvalue != compare)
                        {
                            depth_min = Convert.ToSingle
(reader.Value);
                            depthf = (depth_max + depth_min)/2f;
                            depth = Convert.ToString(depthf);
                            allobjects[i, 3] = depth;
```

```
                                        }
                                        else { depth_min = 0; }
                                    }
                                    break;
                                case "fiducial_rgb1":
                                    if (reader.Read())
                                    {
                                        string holdvalue = reader.Value.Trim();
                                        string compare = "";
                                        if (holdvalue != compare)
                                        {
                                            //do something
                                            allobjects[i, 6] = fiducial_rgb1;
                                        }
                                    }
                                    break;
                            }
                        }

                    } while (reader.Read()); //i++;
                    if (reader.Name == "object") { i++; }
                }
                #endregion

                return allobjects;
            }
        #endregion
```

```
    public static string DisplayFileFromServer(Uri serverUri)
    {
        string fileString;

        // Get the object used to communicate with the server.
        WebClient myrequest = new WebClient();

        // This example assumes the FTP site uses anonymous logon.

        try
        {
            byte[] Data = myrequest.DownloadData(serverUri);
            fileString = System.Text.Encoding.UTF8.GetString(Data);
            return fileString;
        }
        catch (WebException e)
        {
            Console.WriteLine("The file could not be read: ");
            Console.WriteLine(e.ToString());
            return e.Message;

        }
    }
}
```

Appendix E

ROILfile - Code Append

```
        int i = 0;
        ROIL r = new ROIL();



void AddChosenObjects(string obje)
{
        string[,] allobjects = r.Readfile();
        foreach (string ao in allobjects)
        {
            if(obje == ao){
                i = Array.IndexOf(allobjects, obje);
                MoveToPosition(Convert.ToSingle(allobjects[i, 6]),
                                Conversions.InchesToMeters(
                                Convert.ToSingle(allobjects[i, 5])),
                                Convert.ToSingle(allobjects[i, 8]), 80, 0,
                                Conversions.InchesToMeters(
                                Convert.ToSingle(allobjects[i, 4])), 1);
            }
        }
}

void AddChosenObjects(){ //this function simply adds the objects...
```

```
    //float x, y, z;
    string [,] allobjects = r.Readfile(); //Readfile returns allobjects
        foreach (string ao in allobjects)
    {
            if (allobjects[i, 1] == "box")
            {
                SingleShapeEntity test = new SingleShapeEntity(
                    new BoxShape(
                    new BoxShapeProperties(
                       Conversions.InchesToMeters(
                                                Convert.ToSingle(
                                                allobjects[i, 2])),
                       new Pose(), new Vector3(
Conversions.InchesToMeters(
Convert.ToSingle(Convert.ToSingle(
allobjects[i, 5]))),
Conversions.InchesToMeters(Convert.ToSingle(
allobjects[i, 4])),
Conversions.InchesToMeters(
Convert.ToSingle(
allobjects[i, 3])))
                 )
                 ), new Vector3(-.1677594f, 0, -.2387654f)

                );
```

```
                    test.State.Name = allobjects[i, 0];
                    SimulationEngine.GlobalInstancePort.Insert(test);

                    MoveToPosition(-.1677594f, Conversions.InchesToMeters
                                    (Convert.ToSingle(
                                    allobjects[i, 5])), -.2387654f, 80, 0,
                                    Conversions.InchesToMeters(
                                    Convert.ToSingle(allobjects[i, 4])), 1);

                }
                else if (allobjects[i, 1] == "sphere")
                {
                    SingleShapeEntity test = new SingleShapeEntity(
                        new SphereShape(
                            new SphereShapeProperties(
Conversions.InchesToMeters(
Convert.ToSingle(allobjects[i, 2])),
new Pose(),
Conversions.InchesToMeters(
Convert.ToSingle(
allobjects[i, 5]))
)
), //default radius
new Vector3(-.2479551f, 0, -0.1866515f)
);

                    test.State.Name = allobjects[i, 0];
                    SimulationEngine.GlobalInstancePort.Insert(test);

                    MoveToPosition(-.2479551f, Conversions.InchesToMeters(
                            Convert.ToSingle(allobjects[i, 5])),
                            -0.1866515f, 0, 0, Conversions.InchesToMeters(
                            Convert.ToSingle(allobjects[i, 5])), 1);

                }
                else if (allobjects[i, 1] == "capsule")
                {
                    Console.WriteLine("");
                }
            //}
    i++;
        }


}
```

Appendix F

Lines of Code - Results

Table F.1: TriBot with 1 Object

TriBot_single	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
baseball	115	64	4	51	234
book	121	61	1	49	232
box	121	62	1	49	233
cone	116	60	4	48	228
die	121	61	1	48	231
domino	121	65	1	50	237
Golfball	114	63	4	50	231
Marble	114	62	4	48	228
Mechanical Pencil	120	61	4	53	238
Prescriptionbottle	121	60	3	50	234
ringbox	123	61	1	51	236
sodabtl	120	61	3	49	233

Table F.2: TriBot with 3 Objects

TriBot_geom	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
capsules.cs	253	74	8	86	421
spheres.cs	236	83	10	86	415
3aboxes.cs	257	77	1	88	423
3bboxes.cs	252	80	4	88	424
31Ea.cs	249	79	6	91	425
31Eb.cs	248	83	7	91	429
31Ec.cs	249	77	6	89	421
31Ed.cs	248	79	7	89	423

Table F.3: TriBot with 6 Objects

Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
6a.cs	179	72	15	77	343
6b.cs	189	66	10	76	341
6boxes.cs	198	70	4	76	348
6c.cs	186	71	12	77	346
6d.cs	181	68	15	76	340
6e.cs	187	73	11	82	353
6f.cs	178	71	17	80	346
6g.cs	189	76	18	85	368

Table F.4: TriBot with 9 Objects

Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
9a.cs	222	74	18	88	402
9b.cs	236	72	12	89	409
9c.cs	236	72	12	89	409
9d.cs	229	75	15	92	411
9e.cs	228	74	17	93	412
9f.cs	226	78	17	94	415
9g.cs	237	83	18	96	434
9h.cs	238	75	9	89	411

Table F.5: CoroBot with 1 Object

CoroBot_single	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
baseball	156	69	12	47	284
book	140	70	13	41	264
box	147	62	10	39	258
cone	148	62	10	39	259
die	143	61	13	39	256
domino	147	62	10	40	259
Golfball	147	66	10	41	264
Marble	140	63	13	41	257
Mechanical Pencil	140	70	13	42	265
Prescriptionbottle	146	61	13	39	259
ringbox	147	61	12	40	260
sodabtl	148	61	10	40	259

Table F.6: CoroBot with 3 Objects

CoroBot_geom	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
capsules.cs	176	61	17	45	299
spheres.cs	168	72	19	53	312
3aboxes.cs	184	64	13	53	314
3bboxes.cs	189	69	10	56	324
31Ea.cs	181	67	15	56	319
31Eb.cs	180	71	16	54	321
31Ec.cs	181	65	15	53	314
31Ed.cs	180	67	16	54	317

Table F.7: CoroBot with 6 Objects

Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
6a.cs	213	73	24	65	375
6b.cs	223	67	19	63	372
6boxes.cs	232	71	13	65	381
6c.cs	220	72	21	65	378
6d.cs	215	69	24	63	371
6e.cs	221	74	20	69	384
6f.cs	212	72	26	67	377
6g.cs	213	77	24	70	384

Table F.8: CoroBot with 9 Objects

Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
9a.cs	253	75	27	76	431
9b.cs	267	73	21	76	437
9c.cs	264	78	23	76	441
9d.cs	260	76	24	77	437
9e.cs	259	75	26	79	439
9f.cs	257	79	26	80	442
9g.cs	253	79	27	78	437
9h.cs	269	76	18	77	440

Table F.9: Lynx6 with 1 Object

Lynx6_single	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
baseball	1372	986	366	66	282
book	1312	986	349	56	264
box	1312	986	349	56	264
cone	1307	986	348	59	262
die	1312	986	349	56	263
domino	1312	986	353	56	264
Golfball	1305	986	352	59	264
Marble	1305	986	350	59	264
Mechanical Pencil	1310	986	348	59	264
Prescriptionbottle	930	986	190	46	175
ringbox	1166	986	0	0	175
sodabtl	1312	986	348	56	263

Table F.10: Lynx6 with 3 Objects

Lynx6_geom	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
capsules.cs	1340	334	63	269	2006
spheres.cs	1323	344	65	270	2002
3aboxes.cs	1344	337	56	268	2005
3bboxes.cs	1339	339	59	271	2008
31Ea.cs	1336	339	61	273	2009
31Eb.cs	1335	343	62	271	2011
31Ec.cs	1336	337	61	269	2003
31Ed.cs	1335	339	62	270	2006

Table F.11: Lynx6 with 6 Objects

Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
6a.cs	1368	345	70	282	2065
6b.cs	1378	339	65	281	2063
6boxes.cs	1387	342	59	280	2068
6c.cs	1375	344	67	282	2068
6d.cs	1370	341	70	280	2061
6e.cs	1376	346	66	286	2074
6f.cs	1368	349	70	286	2073
6g.cs	1378	349	73	291	2091

Table F.12: Lynx6 with 9 Objects

Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
9a.cs	1411	347	73	292	2123
9b.cs	1425	345	67	292	2129
9c.cs	1422	350	69	290	2131
9d.cs	1418	348	70	292	2128
9e.cs	1417	347	72	295	2131
9f.cs	1415	351	72	294	2132
9g.cs	1411	351	73	293	2128
9h.cs	1427	348	64	291	2130

Table F.13: ROIL Lines of Code

ROIL					
Source File Path	Code Lines	Comment Lines	Mixed Lines	Blank Lines	Total Lines
CoroBot_ROIL.cs	190	68	23	46	327
LynxL6Arm_ROIL.cs	1070	218	54	202	1544
TriBot_ROIL.cs	137	64	13	45	259

Table F.14: Average and Percentages of Decrease for Each Robot

LOC	avg	**LOC**	%diff
tri 1	118.9167	tri 1	-15.2067
tri 3	147.125	tri 3	6.881903
tri 6	185.875	tri 6	26.29455
tri 9	231.5	tri 9	40.82073
ROIL	137	ROIL	
LOC		**LOC**	
coro 1	145.75	coro 1	-30.3602
coro 3	179.875	coro 3	-5.62891
coro 6	218.625	coro 6	13.0932
coro 9	260.25	coro 9	26.99328
ROIL	190	ROIL	
LOC		**LOC**	
lynx 1	1271.25	lynx 1	15.83088
lynx 3	1336	lynx 3	19.91018
lynx 6	1375	lynx 6	22.18182
lynx 9	1418.25	lynx 9	24.55491
ROIL	1070	ROIL	

Appendix G

Redundancy - Results

Table G.1: Amount of Redundancy for Programs with One Object

File/object	**same object type**	**redundant objects**			**results**
		s	**c**	**b**	
marble		0	0	0	0
baseball		0	0	0	0
book		0	0	0	0
box		0	0	0	0
cone		0	0	0	0
die		0	0	0	0
domino		0	0	0	0
golfball		0	0	0	0
mechanical pencil		0	0	0	0
prescription bottle		0	0	0	0
ringbox		0	0	0	0
sodabtl		0	0	0	0

Table G.2: Amount of Redundancy for Programs with Three Objects

File/object	**same type**	**red objs**			multi NO red obj	**results**
		s	**c**	**b**		
capsules			2			14
spheres		2				14
3aboxes				2		14
3bboxes				2		14
31Ea		0	0	0	2	10
31Eb		0	0	0	2	10
31Ec		0	0	0	2	10
31Ed		0	0	0	2	10

Table G.3: Amount of Redundancy for Programs with Six Objects

File/object	**same object type**	**redundant objects**			**results**
		s	**c**	**b**	
6a		2	1	0	21
6b		0	0	3	21
6c		1	1	1	21
6d		1	1	1	21
6e		1	1	1	21
6f		2	2	0	28
6g		2	1	0	21
boxes (6)	5				35

Table G.4: Amount of Redundancy for Programs with Nine Objects

File/object	**same object type**	**redundant objects**			**results**
		s	**c**	**b**	
9a		2	1	3	42
9b		0	1	5	42
9c		1	2	3	42
9d		1	1	4	42
9e		1	2	3	42
9f		2	2	2	42
9g		2	0	4	42
9h		0	0	6	42

Table G.5: Average and Percentages of Decrease for Each Set of Objects

	Redundancy	
Number of objects	**Mean**	**% diff**
nine	42	65.63636
six	23.625	40.74074
three	12	-16.6667

Table G.6: Redundancy in ROIL

	Amount of Redundancy
ROIL	14