CS 430: Senior Project
Robotics

Students
Nathan Woods, Jennings Anderson
Shae Tiegen, Forrest Laskowski

Professor
Steve Harper

Term
Spring 2012

Abstract

"Robo"

Carroll College 2012
Mathematics, Computer Science, & Engineering

Contents

1 Introduction **1**
1.1 Overview 1

2 Development Environment **2**
2.1 Requirements 2
2.2 RSLoad Script 2
2.3 Individual Load Scripts 3
2.4 Integrating with Eclipse 4
2.5 Source Version Control 5

3 Robot **7**

4 Robot **7**

5 Logic **7**

6 Conclusion **7**

1 Introduction

1.1 Overview

After procuring robots from MSU's robotic department, we familiarized ourselves with the IntelliBrain2 Micro Controller from Ridgesoft mounted atop the Ridgesoft Deluxe Educational Robot:

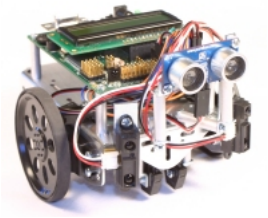

Figure 1: IntelliBrain2 Micro Controller on the standard Ridgesoft base

This small robot allowed us to learn the basics before moving onto a larger frame. This robot ran off two independent continuous rotation servos (ran each wheel independently), a range finding sensor, and two light sensors pointed downwards to follow a line. Next, we moved onto a larger, faster robot mounted on an Axial Remote Control Rock Crawling car base:

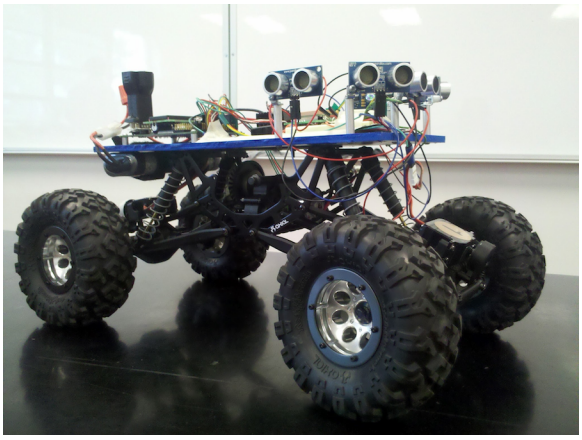

Figure 2: Axial RC car mounted with IntelliBrain2 Controller

We modified the base to include 4 wheel steering. We mounted 5

2 Development Environment

The IntelliBrain2 can be programmed in Java. A specific RoboJDE library is required at compile to interface with the board. There are numerous options to turn Java code into running code on the robot that we will explain here.

2.1 Requirements

First, download the development software from the Ridgesoft Website:

http://ridgesoft.com/robojde/robojde.htm

The software, API, required libraries, and physical wiring diagrams can be found on this website.

There is an included development application, *RoboJDE*, that uses RoboJDE project files $(.rjp)$ to program the robot. This editor is great for getting started, but is quickly outgrown. The software comes with many example files and as long as the host computer has a Serial Port (We used an OptiPlex GX620), then there is no additional software required to get the first programs loaded to the robot.

This environment, however, is all plain text with no syntax coloring or error detection. Moreover, We found Eclipse to be the best editor because it can be configured to automatically compile to the robot using the rsload script.

2.2 RSLoad Script

Included in downloads for all platforms is a script called rsload. rsload is a command line utility for loading code to the robot. The best documentation on it can be found here:

http://ridgesoft.com/robojde/2.0/docs/RoboJDEGuide.pdf

Some important arguments to know for rsload are:

-**port** Specifies the port the robot is connected to, typically COM1 on Windows or /dev/ttys0 on Linux, but if using a USB adapter, this could be /dev/tty.usbserial or COM5. Understanding particular drivers and the OS is important here.

-**run** If included, the robot will automatically run when load is complete.

-**bank** Specify to load to either Flash or Ram; we always loaded to Flash and in some cases, it was required to use this argument.

-**verbose** Good for debugging purposes.

The ultimate required argument is the name of the class that includes main. An example call:

```
rsload -bank Flash -port /dev/tty.usbserial -run Controller
```

This command will load the Controller class to the robot over the USB to Serial adapter. When load is complete, it will run the program. It should be noted that in this example, Controller.class is located in the same directory as rsload.

2.3 Individual Load Scripts

For greater compatibility, we wrote a customizable load script that would compile and then load a specific file, given the file path and the name of the file containing the function, $main$.

```
#Instructions:
#Update both filePath and fileName variables to represent where the files are.
#These should all be relative to Dropbox.  (Note in eclipse cases, compiled files
#may be found in the /bin/ directory of the project.  However, these should be
#pointed to the java file for compiling.
#Don't put a '/' on the end of the filePath:
#The fileName is the main controlling class.
#Warning! Every .java file in the path BETTER compile.
#This is an example:
#filePath=/home/robotics/Dropbox/Robo/Development/Forrest
#fileName=MusicTest
################################################################
####################### WHAT YOU SHOULD EDIT ###################
filePath=/home/robotics/Dropbox/Robo/Development/Master/Robot/src
#Note, don't end with /
fileName=Controller #Note, no .java or .class! This has 'main function'
######################## END WHAT YOU SHOULD EDIT ###############
################################################################
clear
echo "Welcome to the CS 430 Load Script"
echo "This will fail if RoboJDE is running"
echo ""
echo "Copying to Temp Build Path to not upset Eclipse..."
rm -f /home/robotics/Dropbox/Robo/Development/TempLoadNoTouchie/*
cp $filePath/* /home/robotics/Dropbox/Robo/Development/TempLoadNoTouchie/
```

```
echo "Files Copied To Temp Build Path:"
cd /home/robotics/Dropbox/Robo/Development/TempLoadNoTouchie
ls -al
echo "Press Enter to Compile"
read nothing
#Let's compile it first:
echo "Compiling..."
javac -classpath "/home/robotics/RoboJDE/RoboJDE.jar" *.java
echo "Compilation Complete"
echo "Make sure Robot is in Bootstrap Mode, Turn ON then press STOP,
                                                    Then Press Enter"
read nothing
echo "Initiating Robot Loader:"
sh /home/robotics/RoboJDE/rsload -port /dev/ttyS0 -bank flash $fileName
echo ""
echo "Press Enter to Quit"
read nothing
```

This script was written for the robotics user account on an OptiPlex GX620 running Ubuntu 11.10 with the robot connected to the serial port. It will copy the required files to a temporary directory, compile them and then load them to the robot.

The main purpose of a script like this is to abstract the robot development to simple java files in a single directory. The script takes care of including the necessary files for interfacing with the board, so the user can write simple Java code to feed this script.

Next, we take this concept one step further by integrating with Eclipse.

2.4 Integrating with Eclipse

Eclipse is a robust open source development environment that allows for much customization. The two requirements for this project are:

1. Include the RoboJDE.jar archive in the compile path
2. Write an external build tool to automatically call the rsload script.

Once a project is created in Eclipse, preferences can be set by right-clicking on the project root. Ensure here that the Java Build Path includes the RoboJDE.jar file. It is convenient to put a copy of this file in the workspace directory so that it can be referenced locally.

There are explicit instructions for setting up an eclipse external tool in the read me, but I have included a screen shot here for further description:

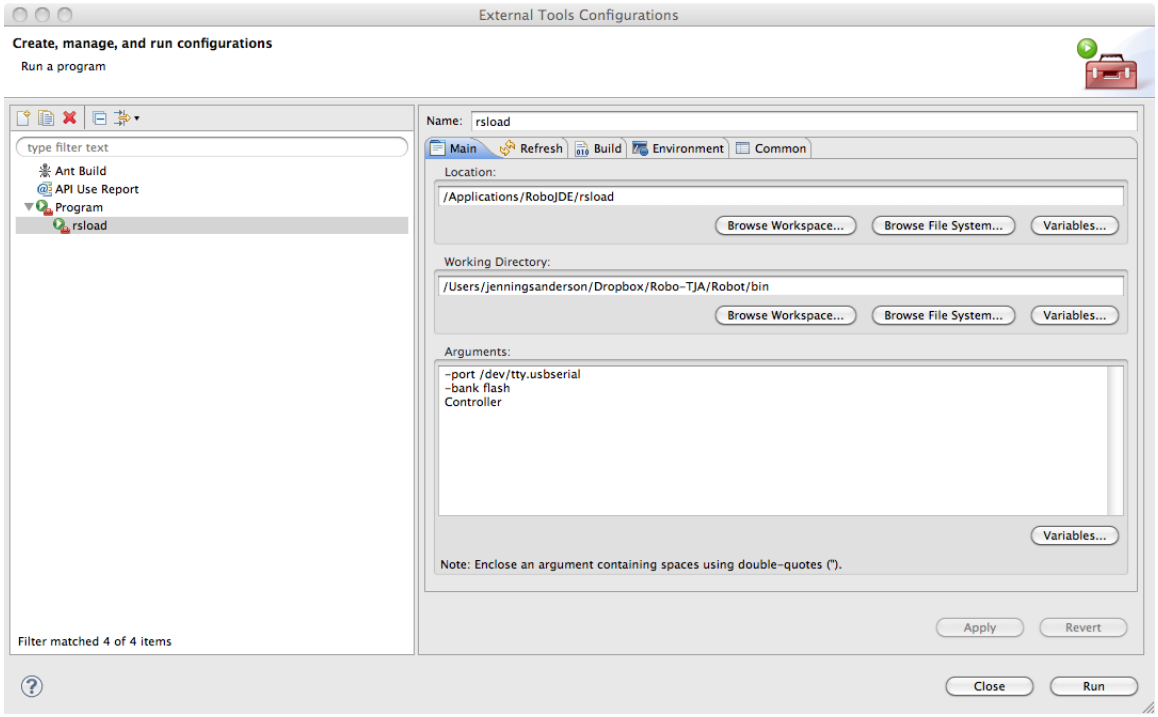

Figure 3: Eclipse External Tool Configuration

This tool configuration differs from the suggested configuration with static references for convenience. Note that the Working Directory references the bin directory, where Eclipse puts the compiled class files. The name of the class with the function, $main$ in it is $Controller$.

2.5 Source Version Control

The importance of a steadfast source version control method should not be over looked; unfortunately, we did so for months until turning to git and github.com. We recommend to spend the time in the beginning familiarizing yourself with the processes of branching, forking, and merging with the github interface.

We found an open source utility, EGit installed through the Eclipse market place, that allowed us to integrate the Eclipse environment with github. It should be known and user be warned that github integration with Eclipse is very powerful and will overwrite the local workspace, so one must be comfortable with the github commands and Egit interface before fetching or pushing from the remote repository.

Originally, we used Dropbox, which worked as a great backup tool at the end of the day, but hard for real-time development. This is because Dropbox is not true version control software.

SVC Anecdote on Dropbox

While Dropbox is a simple, convenient, and magical application in today's computing, it is not a substitute for source version control. We initially used Dropbox to sync files across all of our computers for team development. The problem here is that changes propagate immediately. Backups are kept in the depths of Dropbox's servers, but are tedious to scan through and recover. However, the real kicker occurs when Dropbox meets DeepFreeze, the Enterprise software that Carroll uses to fight malicious intentions. If Dropbox is installed on a user's profile, it will by default sync all files to a folder on the local hard drive. When the computer restarts, changes to the C drive are erased and Dropbox is fooled into believing the user deleted all files in their Dropbox directory. This change is then propagated to the rest of the team, so come Monday morning, all code is erased across every team member's computer. *Thanks, Forrest*.

3 Robot

4 Robot

5 Logic

6 Conclusion