**IntelliBrain2 Graphical User Control**

**Dmitry Korobov**

**CS 470 Project Write-up**

**April 29, 2009**

Table of Contents

[Abstract 1](#_Toc228959464)

[1. Introduction 1](#_Toc228959465)

[2. Project Overview 2](#_Toc228959466)

[2.1 Data Files 3](#_Toc228959467)

[2.2 Prior Robot Control 4](#_Toc228959468)

[3. Project Requirements 4](#_Toc228959469)

[3.1 GUI Specifications 4](#_Toc228959470)

[3.2 System Specifications 5](#_Toc228959471)

[4. System Design 6](#_Toc228959472)

[4.1 Robot’s Module Design 6](#_Toc228959473)

[4.2 GUI Module Design 7](#_Toc228959474)

[4.3 Data Structures 9](#_Toc228959475)

[4.3 System Architecture 10](#_Toc228959476)

[4.4 Algorithms 10](#_Toc228959477)

[5. Software Development Process 10](#_Toc228959478)

[5.1 Testing and Debugging 11](#_Toc228959479)

[5.2 Development Challenges 11](#_Toc228959480)

[5.3 Work Breakdown 11](#_Toc228959481)

[6. Results 12](#_Toc228959482)

[7. Summary and Conclusions 12](#_Toc228959483)

IntelliBrain2 Graphical User Control

# Abstract

[The "IntelliBrain2" robotics controller is RidgeSoft's second generation robotics controller which incorporates the most popular features of the original "IntelliBrain2" controller and the "IntelliBrain2" expansion board into a single board.](http://www.ridgesoft.com/intellibrain2/intellibrain2.htm) The "IntelliBrain2" robotics controller was designed specifically for educational robotics applications can interface with sensors and effectors including hobby servos, DC motors, infrared sensors, sonar sensors, wheel encoders, vision sensors, compasses, GPS devices, speech synthesizers, etc. For example, robot can navigate on its own using servos and range sensor to determine if the path ahead is blocked. The result of this project is a high-level user interface to control robot’s actions.

# 1. Introduction

This project is a senior computer science project developed for Dr. Mock as part of the CS470 class. This project created a graphical user interface for the IntelliBrain2 controller made by RidgeSoft. Robot is designed for educational purposes of learning Java programming language. Robot is controlled by user programs written in Java and loaded into robot’s cache or RAM memory. Robot is capable of interfacing with various sensors and adapters. This particular project involves Bluetooth serial adapter, infrared range sensor, infrared line sensor, sonar motion sensor, servos, LCD display, and IntelliBrain2 controller. Figure 0 shows the robot used for this project.

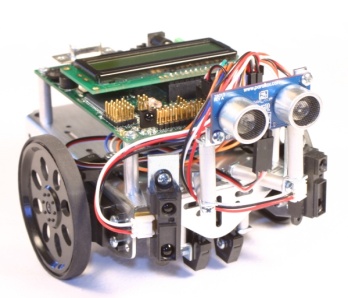


Figure 0, “IntelliBrain2” robot.

# 2. Project Overview

The client for this project is Dr. Mock and I. “IntelliBrain2” robots are capable of performing basic tasks such as moving in various directions, hearing and seeing objects ahead of them, making sounds, providing sensors’ feedback and so on. However there is no easy way of communicating human intentions to the robot.

The scope of this project is to visualize robotic controls and let humans use “IntelliBrain2” robots for various tasks on a higher level. The initial goals of this project were to develop a graphical user interface and a search algorithm that will let the robot recognize shapes and colors. The project was underestimated and later the scope was cut down to the GUI due to time constrains. To achieve the goal of the project two modules were written. The first module is the Robot’s state machine. The second module is the Java desktop application.

This project will deal with various types of data. Most of the data is the signals communicated between the laptop and the robot through the serial port either wired or Bluetooth. Serial link will deliver commands from the laptop to the robot and feedback from the robot to the laptop. Figure 1 is the initiation process diagram.

Figure 1. Initiation Process Diagram

Communication between the laptop and the robot is done through the Bluetooth serial adapter. Since we’re using a one-on-one connection, there’s no need for addressing or any kind of communication protocol. All the data can be directly written and read from the open connection. The laptop will write commands only and read feedback only, while the robot will always read commands only and write feedback only. Commands will be uniquely identified by a series of ASCII characters and will represent both commands and data necessary to perform any particular command.

## 2.1 Data Files

Data files are minimal and are not crucial for the main functionality. The two data files used are: 1) script file and 2) song file.

The script file is used to store more complex procedures that are made of smaller atomic actions. For example, running in a square pattern is accomplished by first running straight then turning right. These two actions repeated 3 more times will make the robot drive in a square pattern. All the procedures are written in the text area the way would write regular java program. In order to preserve the procedure we store it as a plain text file that can be loaded later.

The second data file is an xml file that stores a song that can be sent over to the robot to play. Figure 2 shows part of the xml file that stores the “Twinkle Twinkle” song.

  <?xml version="1.0" encoding="UTF-8" ?>

[**-**](file:///E:\NetBeansProjects\Hive\src\hive\HappyBirthday.xml) <root>

[**-**](file:///E:\NetBeansProjects\Hive\src\hive\HappyBirthday.xml) <note>

  <tone>C</tone>

  <dura>3</dura>

  </note>

[**-**](file:///E:\NetBeansProjects\Hive\src\hive\HappyBirthday.xml) <note>

Figure 2. “Twinkle Twinkle” xml file.

As you can see from the Figure 2 the song consists from note that in turn consist of a tone and duration. Notes can alternatively be represented by integers to ease the process of looking them up in the robot’s module.

## 2.2 Prior Robot Control

Prior to the development of the graphical user interface to control the robot a student would need to write a module for the robot and load it using RoboJDE. Pushing the start button on the robot would launch the module. Once the module is executing it is impossible to switch behavior or pause execution unless you program the start button to act as a logical switch between methods or commands to execute. This adds some flexibility but lacks convenience. It is hard to follow the robot around to push the button assuming you know exactly when to push it. Once the robot is turned off you have to reload the module.

# 3. Project Requirements

The requirements are fairly general and can be broken into two sections: 1) GUI interface, and 2) robot’s module.

## 3.1 GUI Specifications

1. The GUI program should be a standalone desktop application, since serial communication is not platform independent. Even though java.comm package contains several common implementations it relies on a platform dependent native communication library which may not be available on a given system.
2. A graphical window should be created to include various sections of the robot’s control interface. It should include the settings menu to enter setup information such as what communication port is used: com1, com2, etc.
3. A direct control section should be created to host every possible robot’s atomic action. Each action should have a button to initiate the action and terminate it respectively. To list some of the actions: move straight, turn left, stop, sample distance ahead, read camera statistics, and so on. Once started, action should continue until explicitly terminate with a stop command.
4. A script section should be created to allow sequences of atomic actions. Atomic actions from the direct control section should be droppable to the script section for simplicity of use. This would provide the correct wording that is necessary for the script. This section is important in a sense that you won’t need to load your modules to the robot using the original loading method. With the scripter it is possible to load any number of modules without power cycling the robot itself. This section should contain 4 buttons: save, load, clear and run. Save button will allow saving the script as a text file. Load button will allow for loading robot command from robot to the script panel. Clear button will clear the script panel. Run button will parse the script into serial commands to be sent over to the robot for execution.
5. A sensor section should provided 7 boxes to display sensor readings. The boxes are: 1) display, 2) led, 3) camera color tracking, 4) camera histogram, 5) sonar range readings, 6) infrared range readings, 7) line sensor readings.

## 3.2 System Specifications

The GUI control should be implemented using Java language and rxtxSerial library. Both JDK and rxtxSerial library need to be present on the computer to run the application. The resulting application should be executable on most desktop systems since the serial library includes most popular serial port drivers for Windows, UNIX, and Mac computers. At a minimum, the system should have at least 1 logical serial port. If the serial port is physically absent you can use USB adapter to emulate a serial connection either wired or wireless. This application was written and tested on the Windows XP machine using JDK 1.6.

# 4. System Design

System design is split into two separate pieces: 1) robot’s module, and 2) GUI module.

## 4.1 Robot’s Module Design

A majority of the work centers on a state machine implementation. This module is essentially an object oriented model of the hardware components. Each hardware component is implemented as a class that acts as a state machine for that particular hardware piece. A typical state machine for most classes is shown below in figure 2.

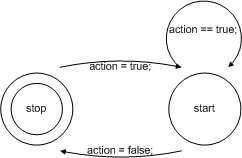


Figure 3. Typical State Machine

All the classes run as separate threads. There is a total of 15 classes. The main runnable class contains pointers to all the other objects and is able to modify their memory space to change their states based on the command that it receives from the laptop. All the threads run using the state machine shown in figure 3. The only exceptions are: 1) the serial class, 2) the display class, 3) the main class, and 4) the led class. Those classes have only one active state. Each thread runs explicitly with normal priority and yields to other normal threads after each complete state cycle. Overall it is a round robin design.

Modeling classes to represent hardware pieces allows for the flexibility of higher level usability. Once the all the state machines are in place you no longer need to modify the robot’s module. It is now capable of performing any procedure written in a higher level language.

## 4.2 GUI Module Design

Figure 4 shows the design of the graphical user interface for the "IntelliBrain2" robot.

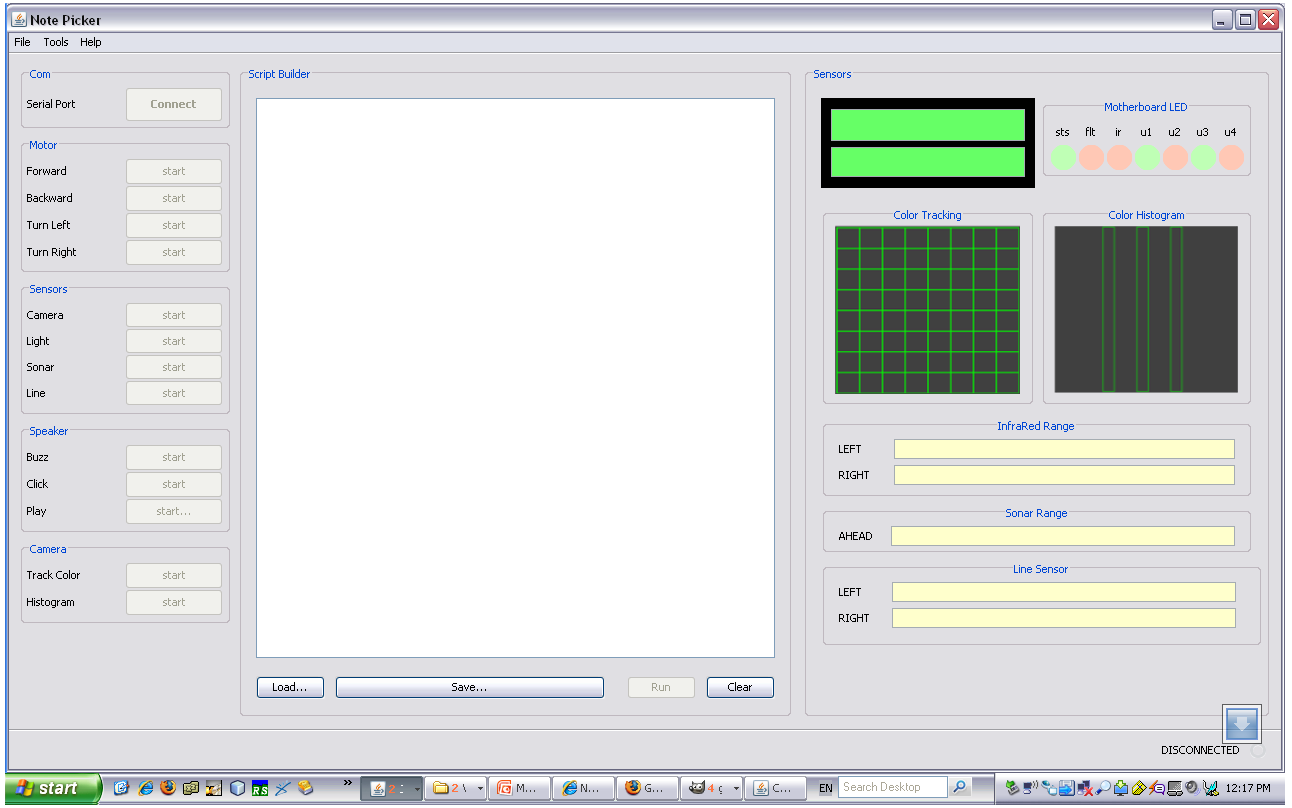


Figure 4. GUI screenshot

The interface contains three main functionality panels, tool bar, and the status bar. The three panels are: 1) atomic actions panel, 2) script panel, and 3) sensors’ panel.

The atomic action panel contains buttons to perform actions that cannot be split into several simpler actions. Atomic actions will be used to perform more complex actions. For example, running in a square pattern will require repeating straight move and turn 4 times. Each atomic action includes a short description label and a button that toggles between start and stop every time you press it. Buttons are activated once the serial communication channel is established. Once the button is pressed a relative command is generated and sent over to the robot via the serial connection. Once the action is initiated it continues indefinitely until that action is explicitly stopped.

The script panel contains the text panel for writing non-atomic procedures in visual basic style language. A sample script is show below in Figure 5.

if (camera center color == green)

move forward

end if

if (light 2 ≤ 10)

move stop

end if

Figure 5. Sample script.

The script panel contains 4 buttons: 1) save, 2) load, 3) clear, and 4) run. Save button saves the script to plain text file. Load button loads scripts from files to the script panel. Clear button clears the text area of the script panel. The run button launches script execution. Once the run button is pressed the script is converted into a single string. The string is parsed based on defined tokens into meaningful syntactical units. Each unit is converted into a command and is sent over to the robot. In essence the script is converted into machine states to be applied to the robot’s module. All the evaluations happen on the laptop. For example, “camera center color == green” will generate command to read the color of the center pixel on the camera. The color value will then be compared on the laptop to determine if the color is indeed green. Robot’s module may be extended to include evaluation operations, but this was not done to avoid extra load on the robot’ CPU. Script panel was not fully implemented due to time constrains.

The sensors’ panel includes all the information provided by the active sensors. The sensors are: 1) display, 2) led, 3) camera’s color tracking, 4) camera’s histogram, 5) infrared range sensor, 6) sonar range sensor, and 7) infrared line sensor. The display and LEDs are passive. They can only be invoked from within the robot’s module for debugging and information retrieval. If their memory state changes it will be reflected in the corresponding hardware and will be sent over to the laptop for graphical representation.

Tool bar contains the settings menu that lets you pick the correct serial port to communicate with the robot. The status bar contains the progress bar and the connection status label.

## 4.3 Data Structures

The data structures used in this project are relatively simple. Most of the data is stored in arrays of bytes or strings. The most complex data structure used in this project is FIFO queue.

Serial communication is implemented as runnable classes in both the robot’s module and the laptop GUI. In order to avoid lost commands due to thread yielding to each other a serial command buffer was implemented using a FIFO queue. For example, in a scenario where the serial thread is yielding and the line sensor writes a command to send over to the laptop and the display writes in a line to be updated, the earlier command will be overwritten with the later one before the serial thread wakes up. To avoid lost commands, commands are instead written to the buffer and are later sent out by the serial thread. The buffer is initialized at size 5. If the buffer is full it will auto-double itself to accommodate all the commands. The buffer is clear logically by reassigning head and tail.

Commands are store in byte arrays. Each command is 25 bytes long with 7 bytes being the command the rest to include the data necessary for command execution. A sample command would be “r i i 0 t 0 0 d 0 0 0 0 0 0 0 0 … ‘\r’”. First two bytes identify the state machine; in this case it is infrared range sensor. Bytes 2 and 3 identify whether it is a right or left sensor; in this case it is the left sensor. “t 0 0” identifies the duration of the execution; in this case it is indefinite. The rest of the string is data preceded by the ‘d’ character. Commands are terminated with carriage return character ‘\r’.

Based on the overall functionality I recommend trying to let objects writing directly to the output stream of the serial connection class to shorten the wait time, since the output is not part of the running serial thread.

## 4.3 System Architecture

Since the robot needs to be loaded with all the basic movement patterns and the serial port listener should be polling for commands before you launch the desktop application we will use RoboJDE to upload all the robot’s modules. Once the modules are uploaded, RoboJDE should be terminated to free the serial port for the new communication link between the application and the robot.

Upon initialization of the main window the user must first go to the settings menu and setup the serial communication mode. Once all the settings are applied, application will attempt to establish communication with the robot. You will receive a feedback in the status bar at the bottom of the window. If the link was established successfully the red “no connection” line of text will change to green “connected” line. Once the connection is established all the controls should be functional. Sensors will only display data while the sensors are in use.

Check Figure 1 for the sequence diagram.

## 4.4 Algorithms

Most of the algorithms are very straight forward. Serial communication is accomplished based on putting together commands that correspond to user actions. Threads act based on the flags telling them whether they need to perform any specific actions or not. Threads run as infinite loops that never stop.

# 5. Software Development Process

Due to the somewhat vague understanding of robot mechanics and overall project goals the development process was a lengthy road of discovery. For the most part I used the extreme programming technique. I would implement a single action to act as test, see how it performs if it does at all, and integrate it with overall interface. If I liked the way if functioned in the system I kept the prototype and later built on top of it.

## 5.1 Testing and Debugging

I spent a majority of my time in testing and debugging. I spent a lot of hours trying to debug serial communication and get the laptop and the robot talk to each other. Another major area of debugging was multi threading. Once the thread were added it was hard to determine why some threads will not perform.

Serial communication was very new to me and while implementing it I ran into a lot problems. The major problem was the overall understanding of the functionality. Once I figure out what it is I ran into minor problems such as using bytes for data and various intermittent bugs due to differences in baud speed. For example I didn’t realize that while converting byte array into strings you lose ‘\r’ characters or that the baud rate needs to be set in both hardware and software for the communication to happen. The major bug that stayed unresolved is the situation when initializing Com2 port corrupts data received on the Com1 port. The bug is consistent and manifests itself in flipped bits and dropped bytes in the Com1 stream. Due to this bug the camera is not functional.

Multiple thread scheduling was tried to improve the responsiveness of the threads.

As it stands now sensors are not responsive enough, while movement and sounds are extremely responsive. The nature of such behavior is not clear.

## 5.2 Development Challenges

The major problem with the development process was time constraints and the learning curve. During the development I realized that I highly underestimated the scope of the project and overestimated my ability to meet scheduled deadlines. A lot of features listed in the initial proposal were not implemented.

## 5.3 Work Breakdown

The Gantt chart in figure 6 shows the anticipated work schedule. This timeline was originally proposed but was extremely hard to follow mainly because of my lack of understanding the requirement and their implementations.



Figure 6. Projected timeline

# 6. Results

The GUI control for the “IntelliBrain2” robot was implemented in its basic form. A lot of features proposed originally were not implemented due to time constrains and a large learning curve. More work is required for the final product to be functional as described in the original proposal.

# 7. Summary and Conclusions

This project resulted in a GUI control that allows a higher level of interaction with the “IntelliBrain2” robot. The interface allows users to use the robot without thinking of how to trigger actions on the robot end of the interaction.

I learned a lot about serial communication and multi-threading. Not all parts of the original proposal were implemented and in order to have a fully functional product more work is required. For example the script panel functionality is incomplete. Some major bugs remain unresolved. Camera is not functional due to mysterious interference between Com2 and Com1 on the robot’s motherboard.

Appendix A: User Manual

**Minimum System Requirements**

Windows XP or higher, mainly due to USB 2.0 drivers

At least one USB port or Serial port

JDK 1.5 or higher

RoboJDE

NetBeans

rxtxSerial library

**Installation/Starting the Program**

Launch RoboJDE and open the Robo project. Press Load button to load the project’s source code into the robot’s memory. If the load button is not active, go to the Tools menu, pick Setting option and select the correct Com port. If the button is still disabled use HyperTerminal to clear the connection buffers. And repeat the procedure.

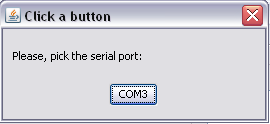
Download rxtxSerial library from <http://www.rxtx.org/> under the download link and follow the setup instructions under the install link.

Launch NetBeans. Open the project Hive and run it.

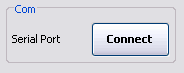
**Using the Program**

Once the hive project is launched you will see the GUI part of the application.

Navigate to the Tools menu and select the Settings option. You will be prompted for the serial port to communicate with the robot.



Once the port is selected press “Connect” button. Functional buttons will unlock. Be careful. It is possible to initialize a connection without being physically connected to anything. Make sure you picked the right port.



Press Start Button on the robot.

Use buttons in the GUI to start and stop robot’s action. The robot’s module is threaded so it is possible to launch multiple actions as long as they belong to different hardware.

**Exiting**

When you would like to exit the application, choose F)ile, E)xit or click the X in the upper right corner. Press stop button on the robot.

Appendix B: Code Listing

See attached zip files.