CSM #5
Scribbler Robot

Field Session 2009

CLIENT:

John Jackson

WRITTEN BY:

Rachel Benefiel
Brian Findley
Eric Young

Abstract

The goal of this project was to make software to be used in a summer program designed to promote excitement and interest in computer science and programming among middle school and high school students. The project called for the design of a fun and interesting lab that focuses on computer science for the Colorado School of Mines led summer technology camp for middle school students and the after school technology club for high school students. The lab uses a programmable robot called a Scribbler and a collection of sensors called SunSPOTs to be both interesting and educational. In order for the students to be able to acclimate themselves to the aforementioned tools and instruments as quickly as possible within a limited time frame a new and more intuitive graphical user interface needed to be constructed. This project called for the redesign of the preexisting visual programming interface of the Scribbler robot to one that places more emphasis on user interface design. Some key user interface concepts and ideas covered by the redesign include drag and drop functionality, intuitive image designs, non intrusive hints and properties, and adding graphical representations of the Scribbler robot program via a sandbox environment. The program also places heavy emphasis on automation by converting the list of images that the user programs with, to pbasic, compiling the pbasic code via an external compiler, and sending the code to the Scribbler robot all with a single button click.

Introduction to Project

The purpose of the CSM 5: Scribbler™ Robot Project is to design software and labs around the Scribbler™ Robot (Figure 1) and SunSPOT sensors (Figure 2). The lab and software will be used in both a free technology summer camp (Tech Camp 101) and an after school technology club held by the Colorado School of Mines (CSM) every year. These opportunities are meant to generate interest in the computer science fields for both High School (tech club) and Middle School (tech camp) students with the intent of potentially recruiting them into the department.

Figure 1: Scribbler

The software created for this project (Figure 3) was made to provide a better alternative to the visual programming software (Figure 4) that came with the Scribbler robot. By utilizing different user interface design concepts, we created a "drag and drop" programming environment that we hope is received as a more intuitive environment. The software was designed for students with little or no programming experience to receive a hands-on introduction to basic programming concepts. As per the client's specifications, the graphical user interface (GUI) must be both user friendly and easy to use, but also age appropriate for high school and middle school students. The software has been developed to allow its user to write programs for the Scribbler robot, show its translated code, and show an approximation of the robots actions on the computer screen.

Figure 2: SunSPOT

The lab that was created to work in conjunction with our software will consist of multiple parts. The first part of the lab will allow the students to gain some practice and familiarity using the GUI by calibrating the robot and by programming simple movements and shapes. Once the students understand how to use the GUI, they will practice using loops and conditional statements to take more complex actions. After this, the students will be able to program the robot to run an obstacle course or maze. Finally, the SunSPOT sensors will be added to the robot, and the students will be able to map the intensity of the light around a room. The GUI was programmed in Java using Eclipse IDE, the SunSPOT sensors were programmed in Java using the NetBeans IDE, and the commands that the robot takes in pbasic.

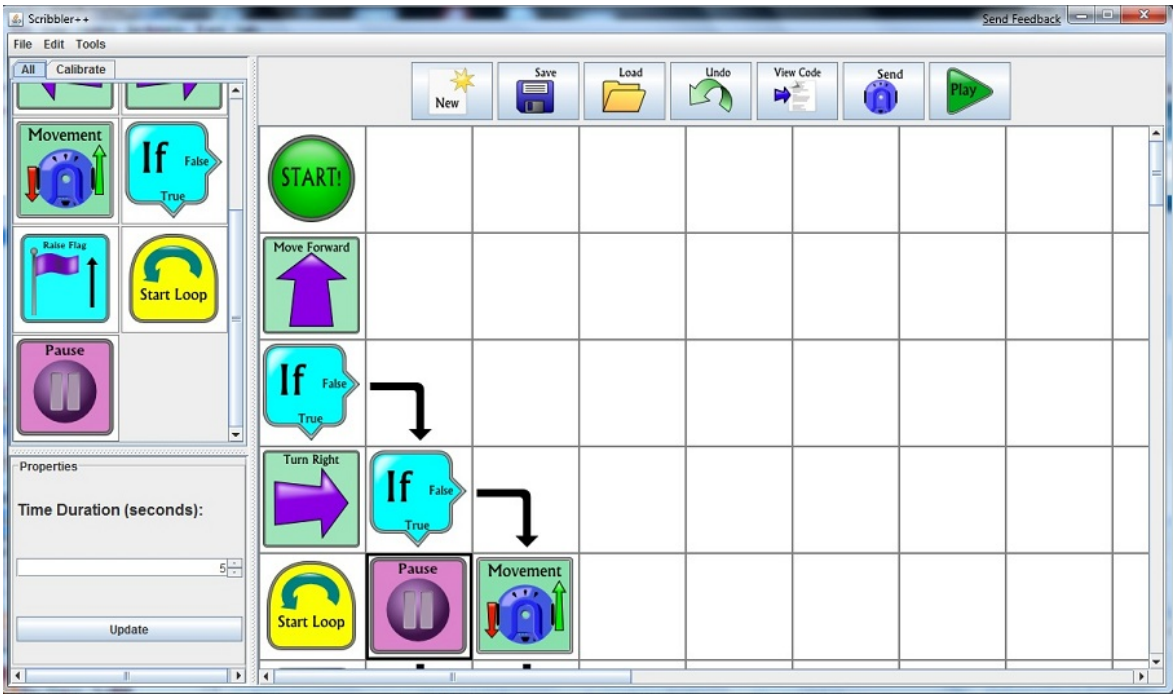


Figure 3: New Scribbler Programming Software

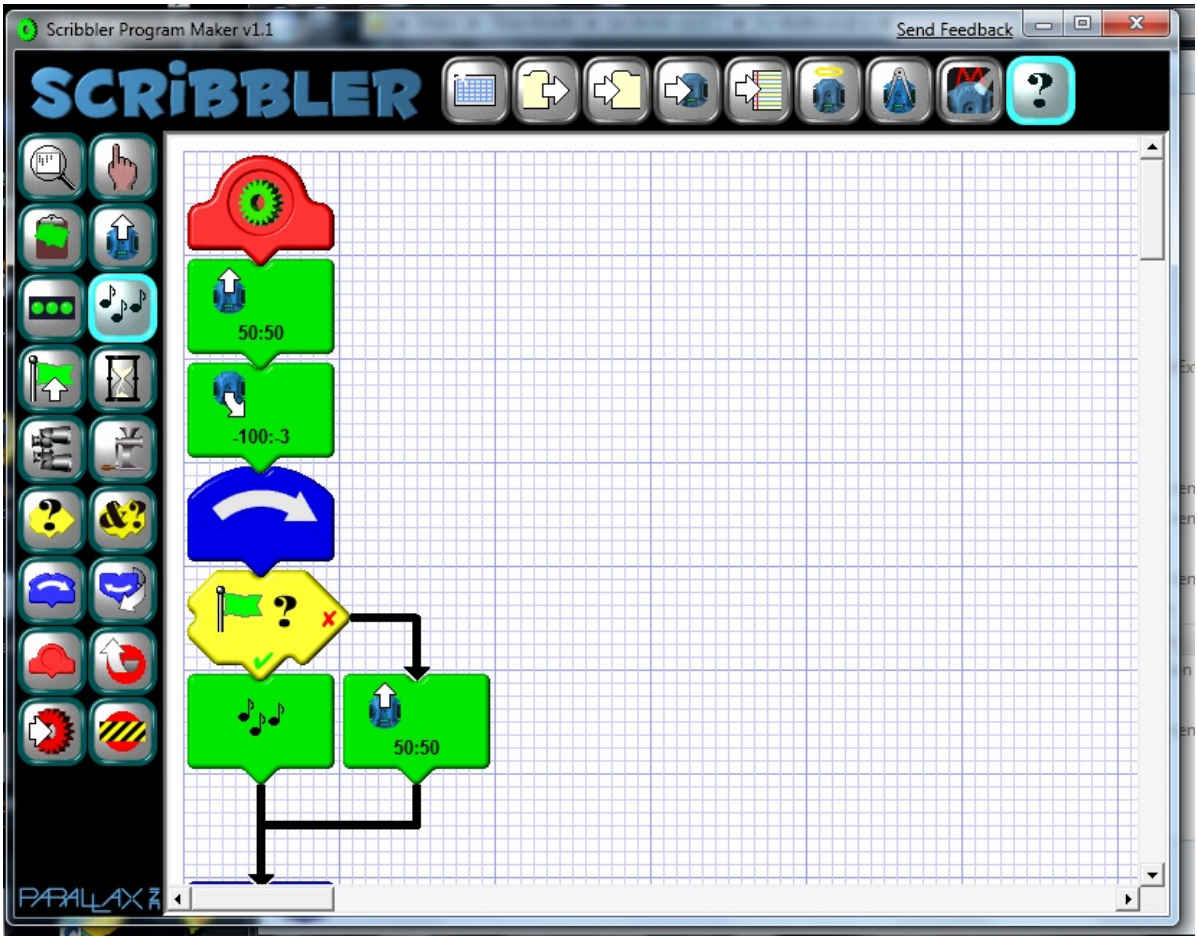


Figure 4: Original Scribbler Programming Software

Requirements

Functional Requirements

1. GUI functionality should include:
   -Coding a program that can send commands to the Scribble Robot.
   -Providing an execution mode that approximates the location of the robot without needing to send the robot commands.
   -The ability for the user to save/load their current worksheet for use later
   -The ability to translate/output pbasic code.

2. Have the software be automated so that the students do not need to open other programs to send the commands to the robot, everything will be included in the GUI.

Non-Functional Requirements

1. Use the SunSPOT's light sensors and Scribbler robot
2. Create a mock write up of the possible lab the students will complete
   -The lab should consist of multiple parts
   -The difficulty of the parts should get progressively more difficult
3. When programming SunSPOTs Java and NetBeans must be used
4. Be very user friendly
   -Include images and the command's properties to help the user understand what each command will do.
5. Six week time frame to finish project

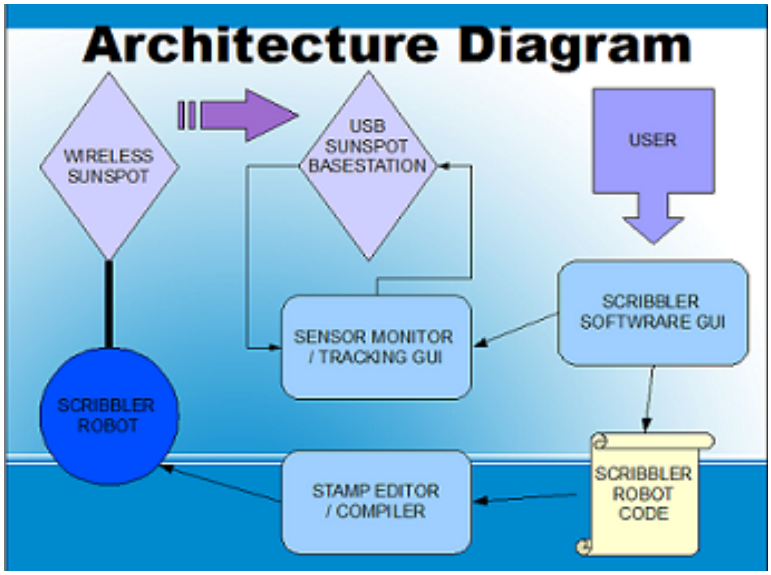


Figure 5: Architecture Diagram

As depicted in Figure 5, the students will program the robot visually in the software designed by our group then upload the program/instructions to the robot. The robot's code is then compiled and sent to the robot by calling the Stamp Editor software when our program uses the command line, so that the user does not need to open any other program. After the robot is programmed, the user has the option of letting the Scribbler™ software open a new window for the map which approximates the movements and position of the robot and map light readings onto a color contour map.

Design

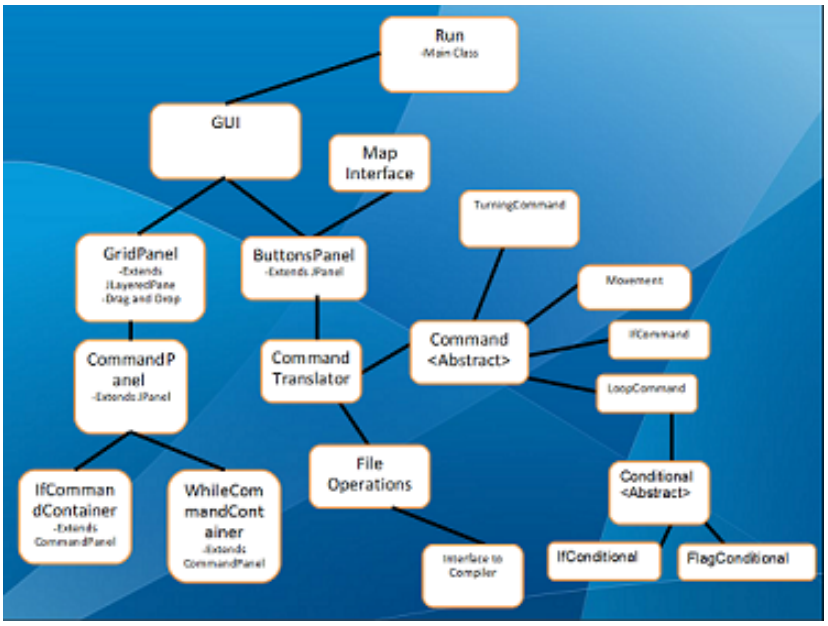


Figure 6: UML

As the purpose of this software is to design a programming environment for the Scribbler robot that is more intuitive and user friendly than the original, heavy emphasis was placed on the user interface from the beginning of the project. As such, the program was divided into two main components: the front end and the back end (Figure 6). The front end included tasks such as designing the GUI, creating drag and drop functionality, and creating images for the software to use. The back end of the software dealt with tasks such as translation of the visual commands of the user to pbasic (the language required to program the Scribbler), compiling the code, and sending the code to the Scribbler robot. In order to accomplish as much as possible during the given time constraints two of the team members worked on the front end and the last member on the back end. As the software progressed, one of the two members from the front end started development on the map, which consisted of utilizing the SunSPOTs and approximating the location of Scribbler to form a contour map.

The functionality of the front end relied heavily on the creation of a data type that could both store data and be represented visually. This data type is called the CommandPanel, an extension of Java's JPanel. CommandPanel's contain the name of the command, a JLabel with its corresponding image, and data prevalent to its command. Because of the embedded image in the CommandPanel each command is seen by the user as the image, however when clicked or interacted with, the data within is either shown or altered. The drag and drop functionality and insertion of commands relied heavily on the mouse listeners of each CommandPanel. There exists two more classes that contain commands and they are IfCommandContainer and WhileCommandContainer, both of which are extensions of CommandPanel. These classes were created to cope with the special properties of if and loop commands. As both of these commands can store other commands within themselves, they were given array lists to store all the embedded commands, a function the original CommandPanel did not have.

The overall design of the back end's translation protocol was to make an interface that can take commands and data from the visual interface and turn them into pbasic code that the robot can understand. The starting layer is the interface where the user can have a visual representation of the commands and options. Next the visual components were placed into classes derived from a base command class. The command classes hold a function to take the options and print the proper pbasic code to a text file. The compiler is then called and the executable is sent to the robot over serial. This method was chosen because commands could be easily added by extending the base command class for future commands and the only code editing would have to be with the interface between the GUI and the commands. Also the recursive nature of this structure allows embedded commands to properly be tracked so the pbasic comes out in the right order.

The map portion of the software utilizes the wireless SunSPOT that is attached to the top of the Scribbler via Velcro (Figure 7) and the base station SunSPOT that is attached to the computer for data gathering. By translating the commands from the visual code in a similar manner to the translation portion of the back end, an approximate path of the robot is made (if conditions have to be ignored in this simulation). The wireless SunSPOT is programmed to send out a light reading wirelessly to the base station every half second. This data is then compared to a gray scale of the range of possible light values and given a color value by the map. The map then draws a colored circle of the light reading on the map. By utilizing multi-threading the robot appears to travel in a fluid path to its destination. As the robot moves throughout the room the colored circles should begin to fill the map leaving a rough contour map of light in the room (Figure 8).

Figure 7: Mounted

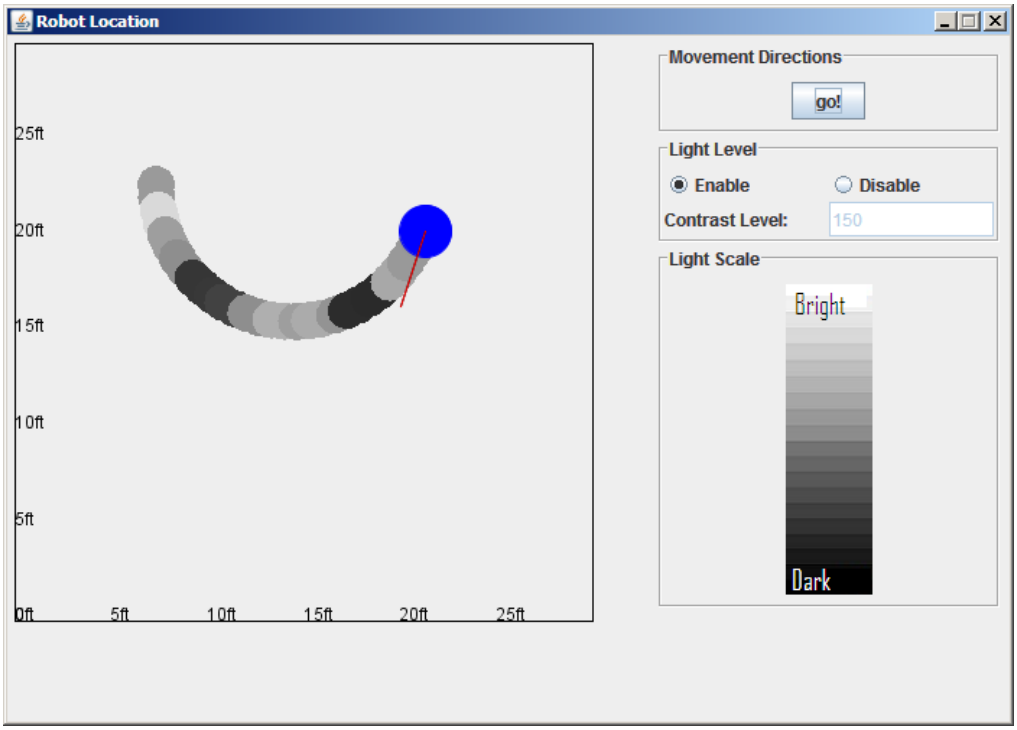


Figure 8: Map

Use Cases:

A) Setting up Robot/Lab (Instructor)

1. The instructor removes existing batteries and replaces with new ones.
2. Instructor connects robot to computer via serial-to-USB cable.
3. Instructor removes robot and plugs in two SunSPOTs using USB cables
4. Opens NetBeans, opens the light sensing project, and uploads it to the sensors

B) Students

1. User plugs in Scribbler robot via serial-to-USB cable
2. User opens the programming GUI
3. User drags commands onto the workspace to construct program
    1. Repeat until code is finished being designed
4. User clicks
    1. Send Button
        -User unplugs Scribbler robot
        -User places Scribbler into starting position
        -User clicks start on Scribbler (and or Play Button)
        -Scribbler moves according to the program
    2. Play Button
        -Map window pops up
        -Simulated robot moves around the map
        -Toggles light readings (paints data or not)

Implementation

Because the project was open ended, all software tools and programming languages that is used in the program was left up to us. As the GUI and visual aspects of the software had to be very intuitive and user friendly we chose Java as our programming language, as it has a robust library of graphical tools and is relatively easy to program user interfaces. Other factors that weighed into this decision is that the team as a whole has more experience in GUI programming using Java than any other language, as well as two of the three members are more familiar with Java programming in general. As Java was the decided on as the programming language of choice Eclipse was chosen as the programming environment because of the team's familiarity in it's use. When programming the SunSPOTs, NetBeans had to be used because it is currently the only supported development environment for SunSPOTs. All elements of the GUI were coded by hand instead of with a GUI builder, because certain functionality like drag and drop required more overall control of the architecture of the GUI than is offered by using a GUI builder.

The Java standard class libraries and no other external code was used in the software except for two exceptions. The Java standard libraries are robust enough that they were sufficient for the majority of tasks and functionality that was needed. The two exceptions to this being the use of a preexisting pbasic compiler that is necessary to convert our code into something manageable by the Scribbler robot and using the Sun libraries made for use with the SunSPOTs. To program and use the SunSPOTs the SunSPOT SDK was essential.

Results

After the software was mostly complete and testing of how well the software was at programming the Scribbler robot, certain qualities of the Scribbler robot were revealed. One of the more evident issues with the scribbler are the inconsistencies of moving. Factors such as being turned on after a period rest, warming up, battery life, wobbly wheels, and just time affect the path of the Scribbler. In order to compensate for this a calibration tab in the software was added and it allows for the reduction in power of the faster wheel. This calibration is useful and necessary, but it does not fix the problem entirely, every once in a while it must be done. After calibrating the left wheel by reducing power by 100 we were able to get a graph of the average velocity of the Scribbler with its corresponding power values as shown in figure 9.

Other details of note when running our software is that in order to use the SunSPOTs you must download the newest software on line instead of the version available on the CD. Another thing to note about the SunSPOT is that it claims that it works using both Windows XP and Vista, but after attempting to get it to work on three different Vista machines it never installed correctly.

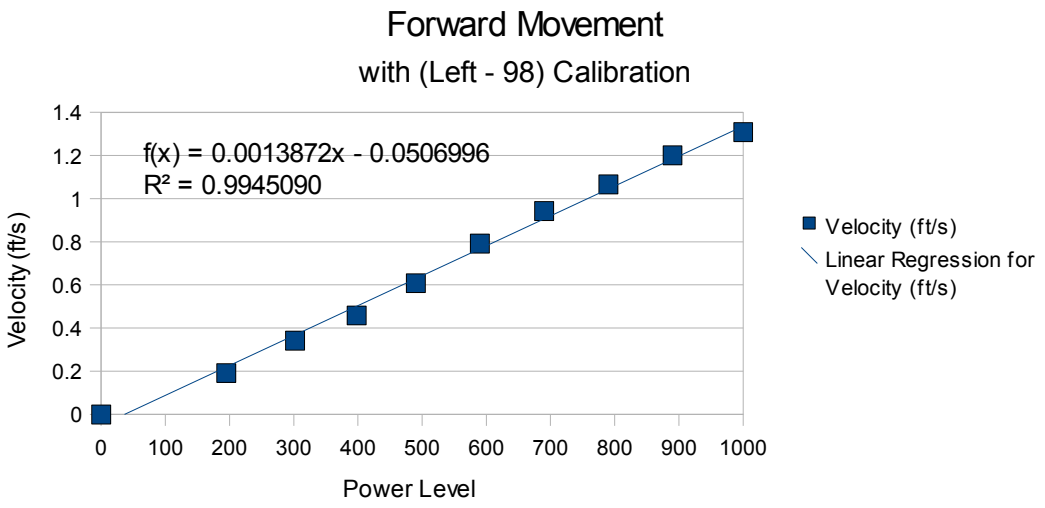


Figure 9: Velocity Graph after Calibration

Project Progression

In order for this project to be considered a success the following had to be accomplished: create a user friendly programming GUI, use SunSPOTs to detect/measure light, write a lab, map the robot's approximate location, and have as many visuals as possible to simplify understanding.

The current build of the software meets all of the requirements dictated by the client to be considered a success. The main GUI has its drag and drop interface as well as its non-obstructive properties and help management is more user friendly and quick to use than the original Scribbler software. The map GUI can wirelessly access light sensor data from the

Scribbler mounted SunSPOT and use that data to create contour maps. A four part lab was written with the functionality available in our software. The first part of which will allow the students to gain some practice and familiarity using the GUI by calibrating the robot and by programming simple movements and having the robot run in specific shapes. The second part will have the students practice using loops and conditional statements to learn computer logic fundamentals. The third part will involve programming the Scribbler robot around a known obstacle course. Finally, the SunSPOT sensors will be added to the robot, and the students will be able to map the intensity of the light around a room.

The functionality that has been programmed into the software include movement, if conditionals, loops, light sensors, and pauses. Movement has been divided up into five categories unlike the original software that just had a generic move. The five parts of movement include forward, backwards, turn left, turn right, and an advanced movement. The if statements allows for checking on the three on board light sensors to decide which actions to take. Loops allow for actions to be repeated a set number of times before breaking out of them. Pauses allow for periods of non movement for the Scribbler to take.

Future Directions

The project as it was detailed above is not actually the original project as dictated by the client during our first meeting. The original project placed heavy emphasis on the creation of a system as well as software to create an accurate map with a robot and its current location as it roams around the room. From the map and data gathered from a mounted SunSPOT a contour map of a room's temperature was to be created.

The original project had two major faults in the way that it was proposed. The first of which is that the SunSPOT which was supposed to record temperature using its on board temperature sensor was inaccurate because of the sensor's placement on top of its power supply. The second problem laid with the original intention of using a single SunSPOT's accelerometer to track motion and in turn the location of the Scribbler, a necessity for an accurate contour map. The accelerometer, according to our tests, was just not sensitive enough for certain motions and could not be relied on. To backup our findings a group from the 2008 field session that also used the SunSPOTS claimed them to be “unreliable” in tracking. For approximately a week and a half we worked to resolve these issues. The issue of unresponsive temperature sensors was fixed by using light sensors instead. The notion of realistic tracking of the SunSPOTs was also fixed by using two of the three SunSPOTs given to use and fixing them onto both of the Scribbler's wheels. The SunSPOTs are extremely sufficient in calculating turns, meaning we could calculate the speeds of each wheel and thus the location. However one of the SunSPOTs given to us was non functional and all three needed to be used to work (2 for the wheels and one connected to the computer) (Figure 10). This problem proved to be unresolvable within the time constraints of the field session so the current project was decided on, using an unrealistic approximation system for tracking motions (due to the Scribbler's inconsistencies). So for a future direction, if CSM buys another SunSPOT to replace the non functional one a more realistic tracking system and map can be made.

Future directions that this project should take are extending the currently used commands and sensor data. The scribbler has line sensors and IR sensors that we didn't have time to add but would be good to have added on. Also other functionality such as the sound device, collision detection, and LED control would be useful in future labs. Any other future work should be

done with the calibration utility and trying to make it more precise.

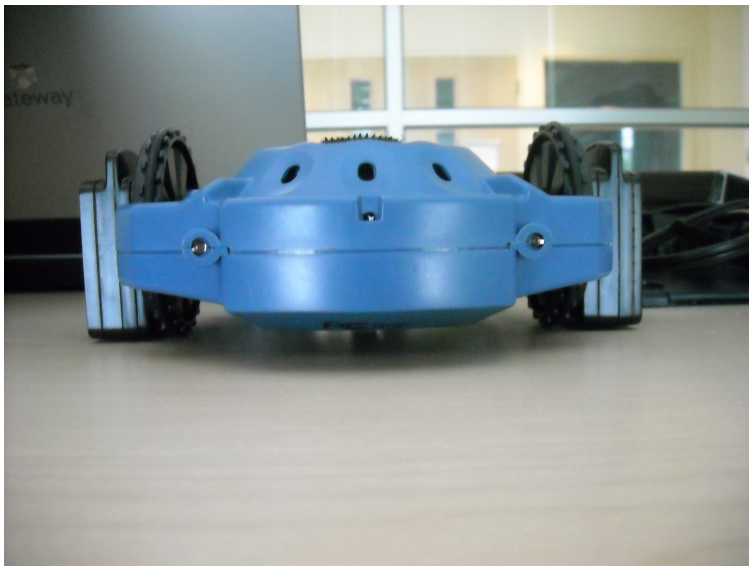


Figure 10: 3 SunSPOT Tracking

Conclusion

Overall the project was a success. The GUI successfully translates commands to pbasic code and the robot follows the commands in the proper order. We designed a lab that will be both challenging as well as fun and should teach the students basic programming logic and thought processes. A lot was learned from this project. To get the drag and drop functionality in the GUI, extensive research in the Java libraries was required. Also, the commands and syntax of pbasic was required to properly make the scribbler move. Other things learned over theses past six weeks include the significance of proper planning and designing, more communication between split programming parts (front end and back end), and scheduling to cope with longer projects.

Appendix:

1. Installation Guide
2. User Guide
3. Programmer's Guide
4. Lab

Installation Guide

2. Insert Scribbler robot programming software CD into computer disk drive
3. Open “Field Session 2009” folder
4. Open “CSM 5 Scribbler Robot” folder
5. Open “Program” folder
6. Click on the “Scribbler_v1.1.1_setup.exe” and install Scribbler software
7. Copy the “Scribbler2” folder onto the desktop
8. Open the “Scribbler2” folder on the desktop and open the “Scribbler++.jar” file to start program. The program should now work except for gather light data
9. Download and install the SunSPOT software from (preferably on Windows XP) http://sunspotworld.com/SPOTManager/SPOTManager.jnlp
10. Click yes to all the installation pop ups.
11. Open NetBeans (either already installed or installed with above link)
12. Click File → Open Project and go to Scribbler2 folder on the desktop and open “SendDataDemo-onDesktop” and “SendDataDemo-onSPOT” projects
13. Plug in one of the non base station SunSPOTs and right click on the “SendDataDemo-onSPOT” project and select “build and deploy”. Some text should stream in the console. Be careful to watch for a comment that says to reset the SunSPOT if it does tap the button on the side of the SunSPOT
14. After the program says it is complete unplug the SunSPOT and plug in the base station SunSPOT.
15. Right click the “SendDataDemo-onDesktop” project and click run. After a little while it should start receiving data from the wireless SunSPOT. If no data is being shown then tap the reset button on the wireless SunSPOT. If every second a white LED on the front of the SunSPOT goes off then it is transmitting data. This last step should be done before starting a session with our program if the user wants to collect light data.

User Guide

1. Open the “Scribbler2” folder on the desktop and open the “Scribbler++.jar” file
2. To start programming drag the desired command from the left panel of the program into the grid and in between the the Start and Stop commands.
    1. Move Forward – Movement that sets both wheels to go forward at the same speed
    2. Move Backward – Movement that sets both wheels to go backward at the same speed
    3. Turn Left – Moves wheels at same speed but in opposite directions (to turn left)
    4. Turn Right – Move wheels at same speed but in opposite directions (to turn right)
    5. Movement – Movement option that gives user direct control of each wheel independently
    6. If – Creates an if statement that splits the code into two paths. The one directly below the if is a true path and the one to the right is the false path. Right now ifs only work while using light sensors.
    7. Loop – Repeats any command with the loop a certain number of times as dictated by the properties. 0 is set to repeat the task indefinitely.
    8. Pause – Tells the robot to not do anything for a set amount of time
3. When the command is dropped a panel on the lower left of the screen should display the properties of the selected command. Change the settings there and click the update button to make changes permanent.
4. If you hover a command in the grid a toolTip of the properties is displayed.
5. At any time you can save, load, create a new file, send, play, undo, and view pbasic code in the button panel at the top of the program. The help file can be read if you go to tools → help. Right now there is no way to delete, just undo if you need to redo something. Undo may appear to not do anything but properties are being reverted so keep clicking if you need to.
6. When the program is made you can do three things. To view the equivalent pbasic code click the View Code button. This should open a pbasic editor called stamp editor. To see what the program should do click the Play button then the go button. This creates and runs a sandbox environment that displays what should happen. The send button takes the visual code that is made, translates it to pbasic, compiles the pbasic, and sends it to the Scribbler robot. To send the code to the robot plug the Scribbler to the computer using a USB to Serial cable then turn on the robot.
7. Before any serious coding is done calibration has to be done to compensate for the tendencies of the Scribbler to curve. Send the forward command to the robot and let the robot run. If the robot drifts one way or the other got to the calibration tab and adjust the forward calibration. Resend the program and test the Scribbler until the path of the Scribbler travels in a fairly straight line. After this is accomplished create a new file and place a backwards command into the grid. Check if the robot swerves and then change the backwards calibration. (For us for Forward and Backward we set the calibration to a reduction of 100 on the left wheel). If you choose to calibrate the Scribbler it is advised that you never go below a power level of 200 for any type of movement except for when you want a wheel to have a power of 0. This is because the calibration is subtracted from power and it is possible to get a negative value for power, which is handled by our program to mean it is not moving.
8. If you want to measure light data from around a room attach the wireless SunSPOT to the Scribbler by Velcro and attach the base station to the computer. Open NetBeans and right click the “SendDataDemo-onDesktop” project and click run. After a little while it should start receiving data from the wireless SunSPOT. Then program the robot and send it to the Scribbler as well as hit the play button. Hit the go on the map and the power on the Scribbler at the same time for accuracy and then hit the enable radio button to see the contour map.

Programmer's Guide

1. **Main GUI Tips/Comments**
    1. The reason there is currently no drag and drop functionality to reorganize the grid and delete functions is because of the special properties of the if and loop commands. To move or delete an if statement or while statement that contains commands within itself is tedious and prone to breaking. We just did not have the time to deal with that.
    2. If a new generic command is created you can just make it a regular CommandPanel and update the image list in the CommandPanel class. If special properties are needed you can extend the CommandPanel class to make proper adjustments(see the IfCommandContainer and the WhileCommandContainer). Also to add the command to the list of commands on the left of the program add it to the commandList in the EverythingTabPanel class. A lot of the conditionals needed for inserting a new command should be done in the GridPanel class. Also you must add the new command's corresponding image to the Command Images folder
    3. There may be some menu buttons that have no purpose. These were just place holders just in case we needed them.
    4. The size of the grid is currently set to be 40X20 commands. If you want to change these values change the sizeX and sizeY values in the GridPanel class.
    5. There is a JsplitPane between the tab of all commands and the Grid that does not appear to function. This was a deliberate decision to help the GUI look better.
    6. If you want to swap out the images that we use for other ones all you need to do is create 100X100 pixel images and name them exactly the same as we did and place them in the Command Images folder. No functionality should be lost if this is done.
    7. If you want to expand upon the already existing drag and drop functionality the mouse listeners in the CommandPanel and the GridPanel should be changed
    8. Ifs and Loops both have arrayLists to store other commands but these aren't filled until the Send button is clicked. The arrayLists are only being used right now to help with the translation process.They can be expanded upon to help with deletion and reorganization drag and drop.
2. **Translation Tips/Comments**
    1. When the Send button is clicked the commands in the grid are placed within their respective ifs and while loops, if applicable, and then within an arrayList in the GridPanel class called allActions. allActions is traversed and fed into the preTranslate function.
    2. The preTranslate function takes everything from the allActions and converts them to a child of the Command class and returned to the translator.
    3. Each command send to the translator will be stored in an ArrayList until translation time.
    4. When the translate function is called the abstract function appendToFile is called for each child command and they will each return an ArrayList containing the pbasic output to be stored to a file.
    5. To ensure the proper order of embedded commands such as commands contained in if statements or loops both the preTranslate function and the translate function has been set up to work recursively so any inner commands may also call preTranslate or translate.
    6. Any commands added on to the program should extend the Command class and follow a similar structure to the current child classes of Command.
    7. Any new conditionals added should extend the Conditional class and follow a similar structure to current children of Conditional.
    8. Going through two layers of translation is not an efficient or clean way of running preforming this task and future work should be done to consolidate this process into into one layer of translation with easier transition from the GUI to the translator.

3. **Map Tips/Comments**
    1. Some variable/function names might appear to be strange because this code was borrowed from one of the team member's earlier projects.
    2. Some more testing of the MoveRobotThread is necessary to match up the speed of the actual Scribbler robot with the simulated one.
    3. The map is supposed to only approximate the location of the robot.
    4. The user of the map can set the cap of the light values from the light sensor. This is because within a normal room's lighting there is slight variance and if you have a scale from the two extremes of light the SunSPOT can detect then there will be very little change in the map. However if you ever exceed the value that you input then the contour functionality breaks.
    5. The map uses a similar translation to the translation to pbasic part of this software to decide what to simulate on the map.
    6. Right now the loops in the simulation are artificially capped at 15 times.
    7. Ifs are ignored in the simulation because it is not possible to pull the Scribbler's on board light sensor to make decisions in the simulator.
    8. SunSPOTs only seem to work with XP and earlier operating systems. Even if you use the right operating system the SunSPOTs are still finicky.
    9. The SunSPOT light reading program is done outside of our software. in NetBeans, because NetBeans is required when programming with SunSPOTs and we weren't able to extract a jar file.
    10. To get the two software to talk to each other the SunSPOT program writes to a file and our software reads from that file and compares to a gray scale.
4. **Scribbler/SunSPOT Tips/Comments**
    1. The code that Scribbler robot uses is pbasic
    2. The Scribbler robot's speed and direction can change from battery life, warming up, and just time.
    3. We do not estimate the angle in properties because it is really sensitive and hard to find a good estimate.
    4. SunSPOTs must be programmed in Netbeans
    5. SunSPOTs need an windows XP or prior operating system to work.
       SunSPOTs are sensitive to crashes and might require the computer to reset to work

Scribbler Lab 1

Introduction:

Lab 1 will help you familiarize yourself with the Scribbler++ program, calibrate the Scribbler robot and learn the different commands that the Scribbler can follow.

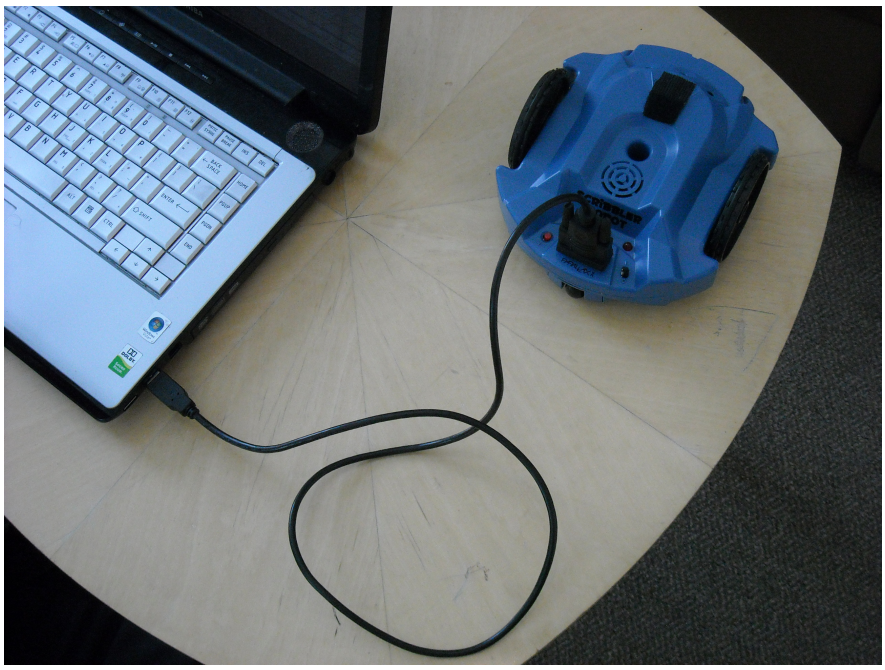
What you need:

- The Scribbler Robot.
- A USB to Serial Cable or Serial Cable
- The Scribbler++ program

Setup:

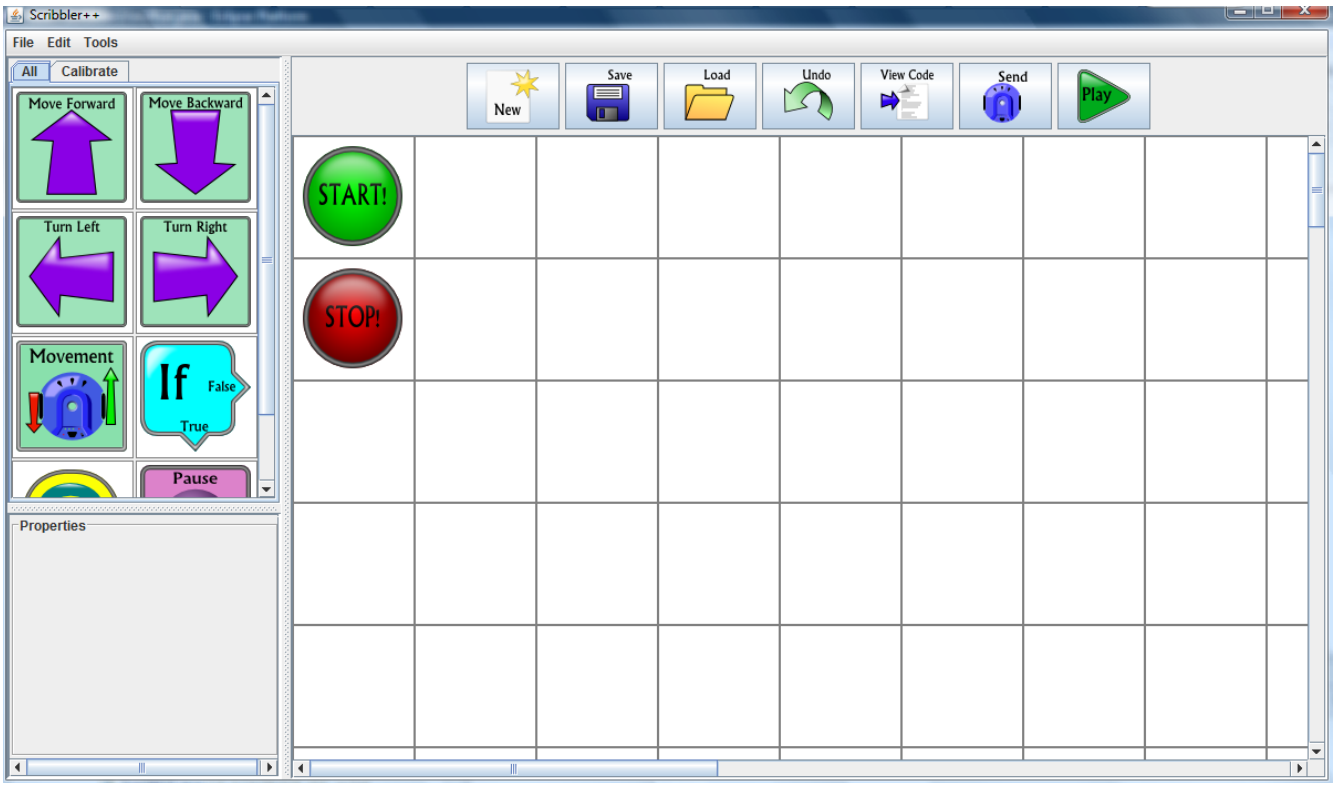
Start by obtaining a Scribbler Robot and connecting cable from your teacher.

Plug the connecting cable into the computer and into the robot as shown. Make sure the Scribbler is on.

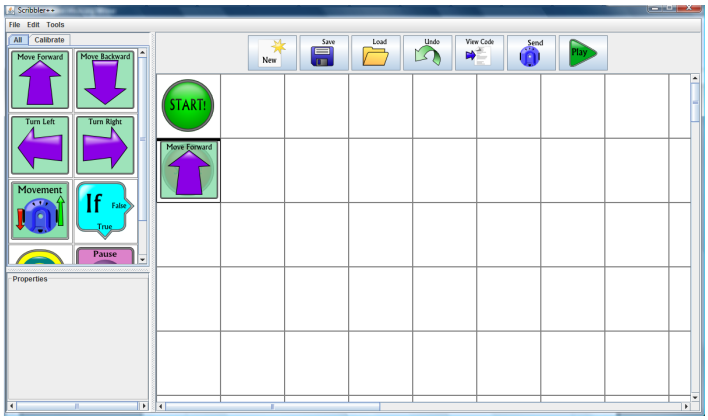


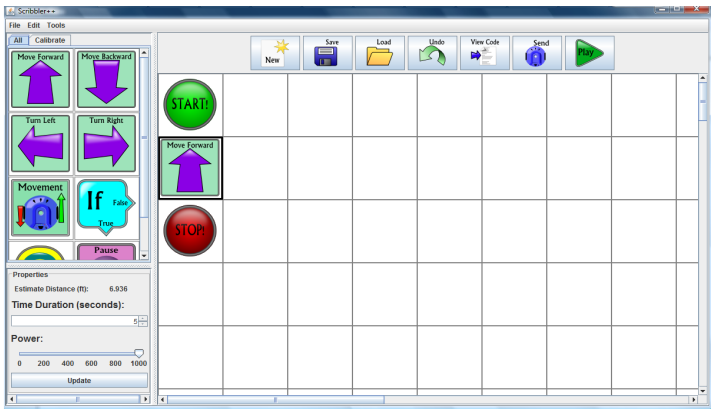
Next open the Scribbler++ program.

The window should look like the figure below. On the left there are two commands. "All" contains all the commands that can be used on the robot. "Calibrate" holds calibration settings.



Let's start with a simple move forward command. Click on the Move Forward button under the "All" tab and drag it in between the Start and Stop icon. You will see a black line when you are able to place it.

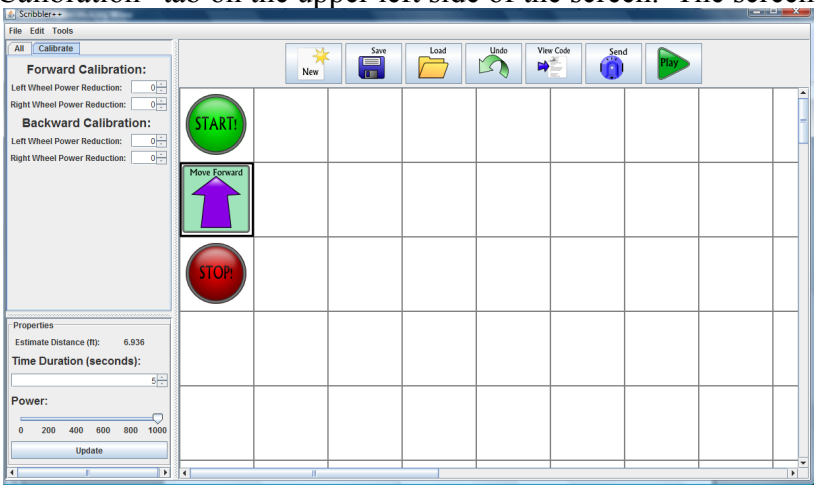




Now hit the send button at the top. The Scribbler robot will run so turn it off and take it to a clear area and let it run. Watch the scribbler and try to figure if it is veering right, left or going straight. If the Scribbler goes straight you can skip part 1.

Part 1, Calibration:

If your Scribbler didn't go forward in a straight line it is in need of calibrating. Click on the "Calibration" tab on the upper left side of the screen. The screen should now look like:



If your robot was turning to the left you want to reduce the power on the right wheel. If your robot was turning right you want to lower the power on the left wheel. Think about why it is the opposite wheel that has to be lowered. After keep altering the calibration until your robot is approximately going in a straight line.

Click the "New" button to start a new sheet and repeat the same process with the "Move Backward" command.

Part 2, Shapes:

Now that the Scribbler is calibrated, the next part of the lab will be to learn a little about the movement commands by making some basic shapes.

Start with trying to make a square, using the Move Forward and Turn Commands. After you have completed this try to make a circle with a radius of five or six inches. Once you have mastered circles try making the Scribbler spin in place.

Remember the different ways to use commands and be creative with them in the rest of this lab.

Part 3, Maze:

Now that you can send many different commands to the Scribbler your task will be to navigate a maze designed by your instructor. Take a careful look at the maze and try to come up with a plan to navigate it without hitting the walls. Estimate how the Scribbler should move and for how long to accomplish this task. When you believe your program is ready upload it to the Scribbler and try it out.

hint: You may want to try out the "Play Button" for a prediction on how the Scribbler will behave before taking it to the maze.

Part 4, Sensors:

The next part of this lab will be working with the on-board light sensors of the Scribbler. Look at the "Loop Command" and "If Command." Loop will repeat any code inside the command the number of time you tell it to. If you select "0" for number of loops it will repeat until the Scribbler is turned off. The "If Command" will let you have the robot execute a command if a certain condition is met. You will use this command to have the Scribbler react to light conditions within the maze.

Your job is now to program the Scribbler to follow light through the maze. Once you think your program is ready take the scribbler to the maze. Try to get your Scribbler to follow the light through the maze the fastest! Scribbler with the best time wins!

Part 5, Introduction to SunSPOTS:

Now for the final part of the lab get a configured SunSPOT set from your instructor and set it up as shown. The SunSPOT taking the light data should be mounted on top of the Scribbler and the base station receiving the data should be attached to the computer like shown.

Program the Scribbler to make a loop around the room once and pull up the map window with the "Play Button." Press the go button on the map and turn on the Scribbler at the same time and watch the SunSPOT record data along the Scribbler's path.