ISAT 252 Final Project - Raspberry Pi and GoPiGo Robots

For Dr. Teate’s ISAT 252 class final project, we were asked to use a Raspberry-Pi powered GoPiGo robot to issue basic commands and manipulate the robots’ movements and LED lights. This involved using pre-written Python libraries (classes and functions) to issue commands to the robot using SSH through either PuTTY or a Linux terminal. We encountered a number of problems with the initial setup, wireless connectivity, and SD card corruption, but overcame most of those issues to be able to at least manipulate the robot on a basic level. This report serves to outline the steps we took to get the robot to work, some of the issues and challenges we faced, as well as solutions to these issues and what we took away from this assignment.

Our robot consists of a GoPiGo control board and a Raspbery Pi B+ microcomputer. The Raspberry Pi runs a Linux operating system and runs the Python code to send commands to the GoPiGo control board, which in turn controls the motors and electronics of the robot itself. To connect to the robot, we utilized the PuTTY program on Windows and the built-in ssh daemon on Linux to use SSH and connect to the robot. SSH stands for ‘secure shell’ and is a method of securely and remotely connecting to a Linux machine to issue command line commands. For this project, Kadar utilized his Linux laptop for connectivity while Anna used a Windows laptop and PuTTY to SSH into the Pi. We did not face any issues with SSH related to SSH itself, and had to learn that the Pi is not as fast as consumer computers, we had to wait about 20-30 seconds for the connection to initiate and another 30-45 seconds to safely shut down the Pi once we were done using it.

One of the skills we developed in using SSH and PuTTY was a basic understanding of the Linux command line, including commands to navigate between folders (cd), create & open python files (sudo nano abcd.py), editing python code (nano abcd.py), executing python code (sudo python abcd.py) ,and safely shutting down from the command line (sudo shutdown -h now). We also learned that the ‘sudo’ command must be used each time a command needs to be used as an administrator, otherwise we’d get errors stating ‘Permission denied”.

Once connected to the Pi, we created a folder for our project using the ‘mkdir’ command and created our python file to further edit. We used the nano command to open the provided code (basic\_robot\_control.py) and adapted that code to control our robot. We utilized a menu consisting of numbers relating to direction and LED blinking and prompted the user what each command was and when to enter it. Using this code, we controlled the robot remotely for filming our videos.

In the course of completing this project, we faced a number of different issues and challenges and had to consult outside resources to overcome them. Some of the issues faced included issues with the motors, wireless connectivity, SD card corruption, batteries dying too fast, and firmware issues.

One of the first challenges we faced was the Robot turning in a circle, no matter what changes we made to our code. We discovered that this was a multiple-cause issue; the first cause was improper wiring on the motors and the second cause was an invalid trim setting on the motors, causing one motor to turn faster than the other. To fix the motor issue, we wired the motor according to the following wire structure: black, white, white, black. One thing that is tricky about this wiring is that the wires for the left motor (when looking at the bottom of the robot with the battery pack facing away from you) are wired farther to the right, and the wires for the right motor are wired to the left. Appendix Figure 1 shows an illustration of this. Once we did this, we verified that the motors operated properly by running left() and right() code, ensuring the correct wheel turned when the corresponding code executed. To address the issue of the motor trim, we updated the firmware to v1.3 using a guide found on Dexter Industries’ website [1]. We first verified the current trim level by issuing the ‘tr’ command. We then tested various trim levels using ‘tt’ and wrote the one that worked best for our robot using ‘tw’. In this case, we used a trim of 0 for the robot running on kitchen tiles, but a trim of 5 when running on a linoleum rubber surface. The wheels are slightly wobbly and don’t spin in a perfect circle, so the trim has to be adjusted based on the surface.

Another issue we faced was with wireless connectivity between the Pi and our laptops. Initially, we had the Pi set up on Dr. Teate’s netgear router using IKM\_THESIS as the SSID and were able to connect to it within the lab room, but we had concerns that we would not be able to connect the router on campus dorms and we also faced an issue of the Pi assigning a different IP address each time it was powered on and off. The solution to this was multi-tiered; first, we met off-campus at Kadar’s apartment so as not to have issues with JMU’s wireless security (which, as it turns out, isn’t an issue after all). Next, we were able to configure the Pi to have a static IP address by following the solution on the ElectroSchematics’ website [2]. Finally, we issued a command line command, ‘sudo shutdown -h now’, instead of toggling the power switch on the GoPiGo to safely shutdown the robot each time we were finished using it. Eventually, Kadar was able to configure a spare router to match the SSID of Dr. Teate’s router, which eliminated the need to transport Dr. Teate’s router. Additionally, Kadar was able to verify that the static IP fix worked by reformatting the SD card and testing the solution each time. It was discovered that the static IP fix worked by following that guide.

Another networking issue faced was the issue of not being able to SSH into the Pi if the Pi was not properly shut down. This was caused by the file system becoming corrupt by using a hard power on/off. We fixed this problem by connecting the Pi to an external monitor with a mouse and keyboard and running the ‘fsck’ command, following the prompts that showed up. This had to be done only if we didn’t properly shut down the Pi. For all other shutdowns, we issued the ‘sudo shutdown -h now’ command to safely shut down the Pi and didn’t have any issues as a result.

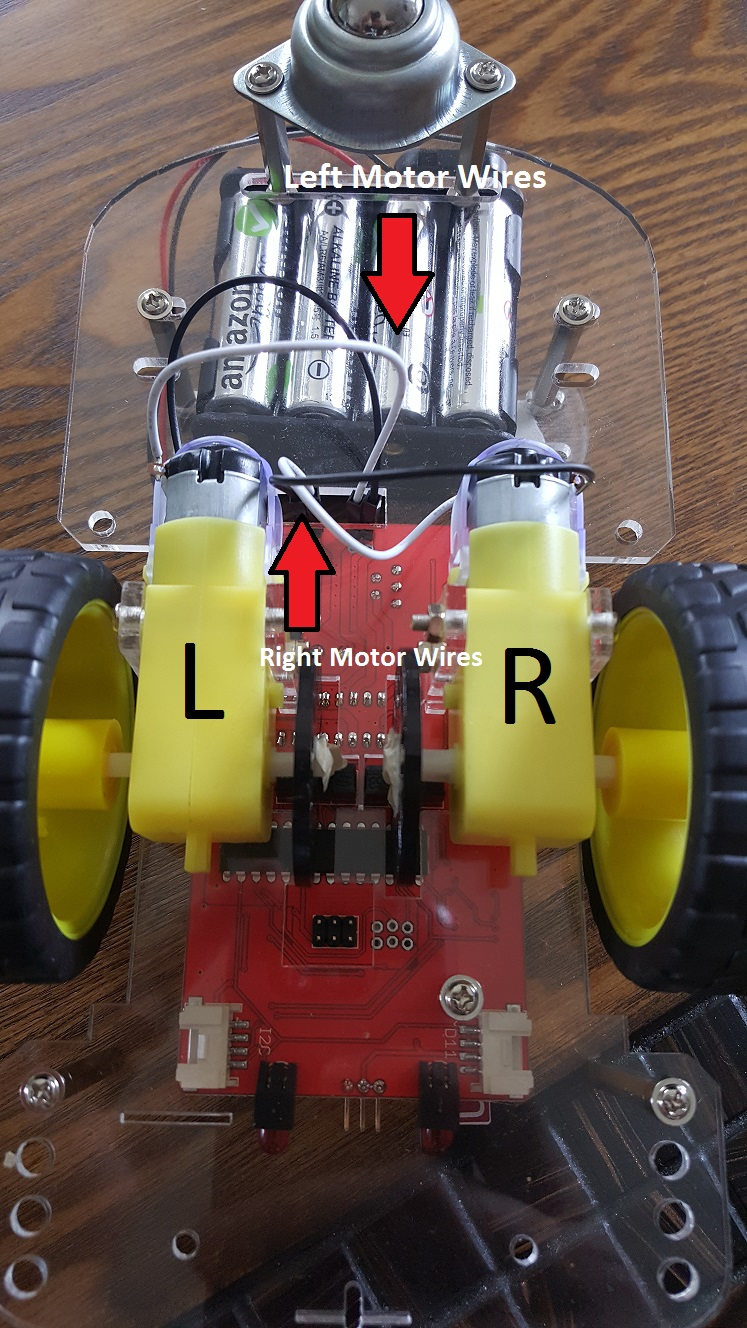
Yet another issue we faced was the motors of the GoPiGo turning at different speeds and causing the robot to drift to the left and right. We again consulted Dexter Industries’ website [3] and performed a firmware upgrade. With this firmware upgrade, we were able to adjust the ‘trim’ of the robots by issuing the tr, tt and tw commands described previously. Once done, this helped to keep the robot from drifting but, as previously mentioned, the robot has some inherent flaws in its’ construction that prevent it from navigating in a completely straight line. The accuracy of this navigation also depends on the material the Pi is moving across.

A final issue we faced, which prevented us from using the robot autonomously, was an IOError that popped up after performing a firmware update to v1.3. When using the robots on firmware 1.2, we never received an IOError in issuing commands to the Pi, but received these errors after the 1.3 upgrade. Appendix Figure 2 better illustrates the severity of these errors. In order to complete this project, we consulted with Dr. Teate and discussed the issues at hand and came to the agreement that we would film each of the robot’s movement as a validation that we understand how to interact with the robot and use the provided python packages to control and manipulate the robot. The IOErrors prevent us from moving the robot autonomously, but we have requested help on the Dexter Industries’ forum and hope to have a response, and resolution, in the near future.

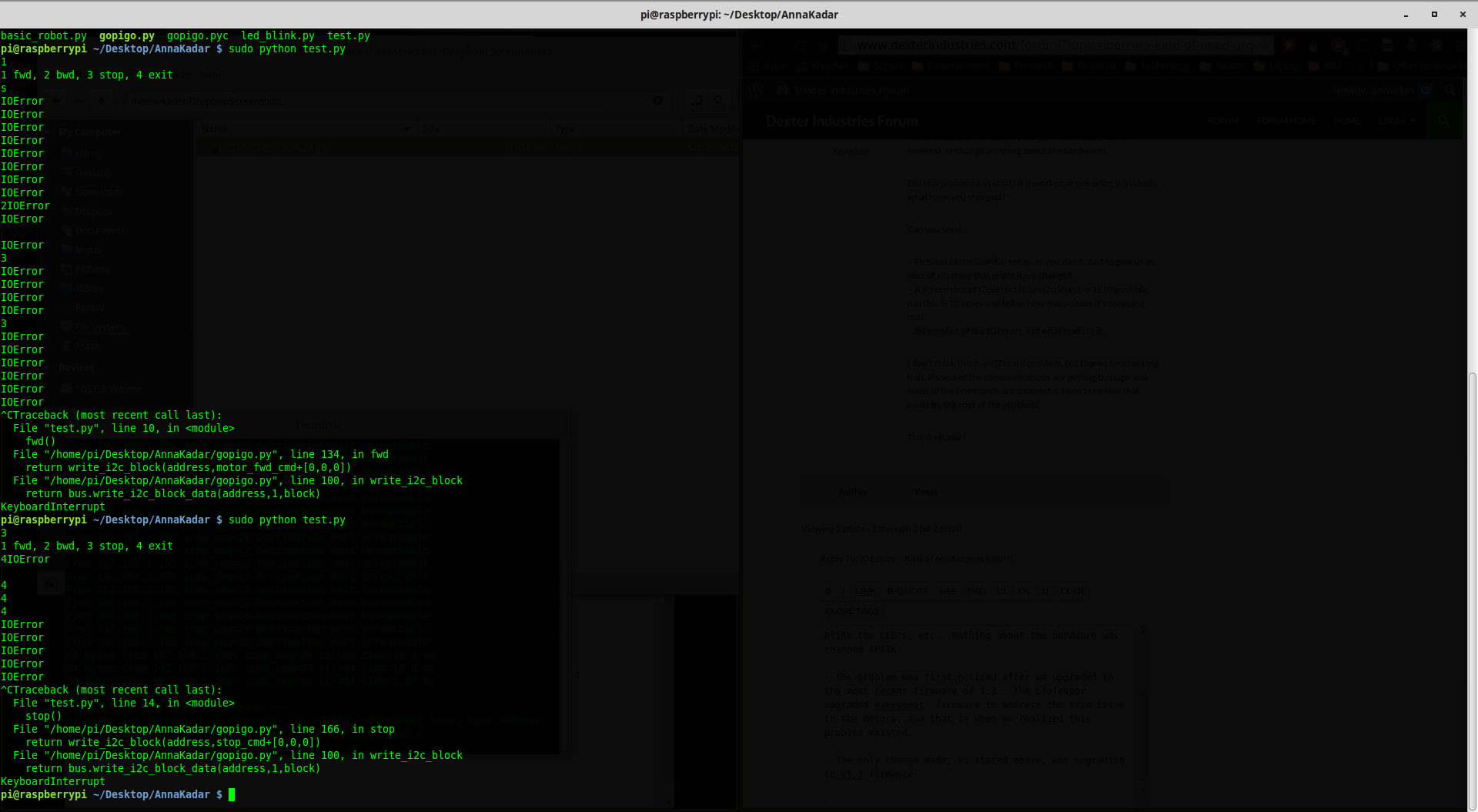
A challenge we faced during this project was making the best use of a limited resource provided to us, specifically the batteries that power the robot. We noticed that even after a few minutes of sustained usage, the batteries would almost completely drain and prevent us from at least connecting to the robot enough to modify our code. A solution to this problem was to use an external battery pack with USB cables to power our Raspberry Pi while we created and modified our code, then to use the toggle switch on the GoPiGo board and actually power the motors only when we wanted to test the code. This helped us to spend time troubleshooting our code without draining through a limited amount of batteries.

Overall, this project was incredibly fun, enriching, and it was exciting to use the newly-developed GoPiGo robot for our project. We were able to learn about using Linux and basic commands, became familiar with reading and adapting other peoples’ code and enjoyed using Python to control the robot. This project helped us gain an understanding for some basic networking fundamentals and also to further appreciate a real-world use of Python in robotics. We faced a number of issues and challenges but those issues only helped to further our understanding of the project as a whole.

Appendix - Diagrams & Figures of the GoPiGo Robot Project



*Figure 1 - Photo of the GoPiGo motor wiring.*



*Figure 2: Screenshot of the multitude of IOErrors we received when trying to run some basic python Code on the GoPiGo robot. We had to use ctrl+C to stop the process in order to manually re-run the code and stop the robot.*

References

1. <http://www.dexterindustries.com/GoPiGo/learning/python-programming-for-the-raspberry-pi-gopigo/add-trim-to-the-motors/>
2. <http://www.electroschematics.com/9496/static-manual-ip-wireless-raspberry-pi/>
3. <http://www.dexterindustries.com/GoPiGo/getting-started-with-your-gopigo-raspberry-pi-robot-kit/5-using-your-gopigo-for-the-first-time/>