Developing a Modular Robotics Platform for Teaching Programming

Samantha Kacir
Advisor: Douglas Blank

Submitted in partial fulfillment of the requirements for a B.A. in Computer Science at Bryn Mawr College

May 2017

Abstract

The purpose of this project is to develop an inexpensive modular Raspberry Pi based robotic platform to teach programming. It has been established through a number of studies that learning to program using robots has quite a few advantages over traditional programming courses; however, there are few kits that are both easy to get started with and that enable the experienced programmer to develop more complicated projects. Of these, I was unable to find any that were comparable in price to college textbooks. I experimented with different hardware options in order to find an inexpensive and easy to use hardware solution and then wrote code to control it. The code was designed to be simple to understand and easy to expand. I also designed a 3D-printable housing for the control unit and a 3D-printable robot as proof of concept. I believe that with a good web application and an expanded library of robots, this project could easily serve as a framework for an introduction to programming.

Table of Contents

1. Introduction

As early as the 1960s people have been exploring using computers to learn traditionally difficult subjects such as mathematics or programming (Harel n.d.). Seymour Papert was a pioneer in this field, and described a "computer-controlled cybernetic animal" (essentially a physical or virtual robot) that could be used to help children connect with what they are attempting to learn (Papert 1980). Learning to program using robots has several advantages over traditional programming courses, which are "taught by introducing constructs such as variables, expressions, conditionals, iteration, functions, object-oriented concepts, etc." (Markham and King 2010, 204). This leads to examples being designed to academically illustrate or practice those concepts, which has led to criticisms of traditional programming courses as being "non-engaging, irrelevant, and boring" (Markham and King 2010, 204). Robots, on the other hand, are "cool and fun," making them generally more engaging for students (McGill 2012). They "are tools that support, complement and enhance learning environments" (Scott, et al. 2015, 1). Along with that, robots enable almost instantaneous, tangible feedback, making debugging easier and more enjoyable. "Personal robots in particular give each student unlimited, hands-on access as well as freedom to choose where and when they use them" thereby enabling students to learn on their own terms and turf rather than necessitating a trip to the lab (Markham and King 2010, 204). Finally, they help students to construct models of how programming works in their minds, which Papert postulates is essential to understanding something. As he says, "anything is easy if you can assimilate it to your collection of models. If you can't, anything can be painfully difficult." This is especially true in mathematics-based fields, such as programming, where it is rare for people to happen upon experiences leading to the creation of these models (Papert 1980).

Since robots are useful, fun, and educational, this project seeks to develop an inexpensive modular Raspberry Pi based robotic platform to teach programming. I use the term "modular" throughout this paper to refer to robots designed to be able to be taken mostly or completely apart and put back together with the same or similar parts for a completely different function. By inexpensive, I mean that the hardware should be comparable in price to college textbooks, i.e. under $200 and preferably closer to $100. While the platform's stated purpose is education,

in theory it is modular enough that robots made using this platform could be used for anything from a simple plant caretaker to a quadcopter, thereby opening it up to everyone to use. In other words, as has been an accepted standard of good software tool design, the platform "should make it easy for novices to get started (low threshold) but also possible for experts to work on increasingly sophisticated projects (high ceiling)" with the addition of "wide-walls" by giving anyone the ability to design their own robot if they have some way of creating STL documents for 3D-printing (Resnick, et al. 2005).

2. Background

Teaching through robotics has become more common since the turn of the century. Not only are they used in classrooms, but also outside of classrooms, as kits or instruction guides for self-teaching. There are even robotics kits that allow the user to choose the function(s) of the robot they build. However, there do not seem to be many inexpensive modular kits that the learner can build/design the parts themself without much prior robotics experience.

2.1 Robots

There are many classroom robots, by which I mean robots designed to teach programming in a classroom setting. The guiding principle of these robots is to have a premade robot with minimal abilities that can be programmed to do things out of the box. This is due to the fact that they are designed to be an introduction to programming rather than robotics, so the hardware interaction is minimized. One example of such is the Parallax Scribbler, which comes prebuilt with sensors and actuators for basic movement and environment interaction (Kumar 2008). It has been used at Bryn Mawr College and other institutions of higher learning to teach programming (Eilbert 2007).

There are kits and instructions to build robots that operate on much the same principles as the classroom robots. An example of this sort is the Qwerkbot (Figure 1), which is a small wheeled robot that can be inexpensively built using materials from a hardware store. For this sort of robot, while the learning to program aspect might still be touted, the focus is generally more on learning about the electronics or robotics aspect.

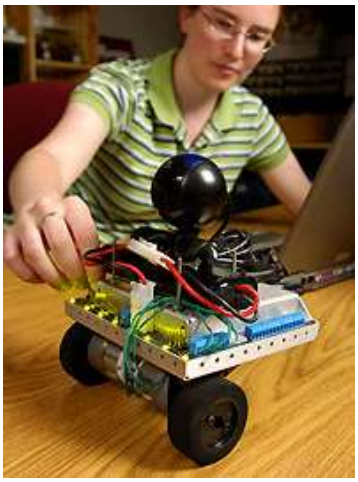

Figure 1 Qwerkbot
http://www.popcitymedia.com/innovationnews/5907qwerkbots.aspx

Modular robots, on the other hand, are almost universally designed as teaching tools. One doesn't get quite as much functionality out of a robot that is designed to be taken apart and put back together again in different ways as one that has a dedicated single use. However, it does mean that if one only needs a robot temporarily, or decides they want some additional functionality, they don't have to buy an entirely new robot or make potentially damaging

modification hacks. Modular robots are much less common than dedicated-use robots or semi-modular robots, though seem to be relatively popular where they are found. One of the most popular modular robot platforms for teaching is Lego MindStorms, described in section 3.2 (McGill 2012, 1).

2.3 Arduino

Arduino is an open-source electronics platform designed to have a simple and accessible user experience such that it is easy for beginners to get started but flexible enough for experts to use (Arduino AG 2017). Arduino components are generally one of four things: a controller, a shield, a sensor, or an actuator. An Arduino controller is a board that can have a program uploaded to it that it will run when it has power. A shield is a board that can be plugged into the controller or another shield in order to extend the functionality of the project (Arduino AG 2017).

Arduinos are generally seen as the standard in DIY robotics projects – they are relatively inexpensive, well-documented, and have good support. Along with that, with a control board or a shield such as the Arduberry or GrovePi, which are shields designed to connect the Raspberry Pi (see section 2.3) with Arduino components, they are generally plug-and-play with no soldering required. This makes them an excellent choice for beginners with or without an interest in robotics. Those with an interest can get an introduction to hardware, even at institutions that don't generally have robotics courses, and decide if they want to invest the time and effort into either learning more on their own or finding a mentor. Those without can merely plug the parts into the ports and get to the coding portion.

2.3 Raspberry Pi

A Raspberry Pi (Figure 2) is an embeddable microcomputer. In fact, the original model in 2012 was "the first mass-market 'single board' computer" (Geerling 2016). It has now been around long enough, and become popular enough, to find documentation and troubleshooting for many projects and problems. This is its main advantage against other

Figure 2 Raspberry Pi 3 Model B
https://www.adafruit.com/categories/105

microcomputers. The version I am using is the Raspberry Pi 3 Model B, which is one of the most recent ones.

The advantage of a microcomputer over other options, such as Arduino control boards, is versatility. The control boards are merely mirocontrollers so they cannot run an operating system and generally can only store one program at a time, which runs when the board has power. At a comparable cost to Arduino control boards such as the Uno and Leonardo, the Raspberry Pi has greater processing power, built-in wireless connectivity, and the ability to multitask. By adding in an Arduberry or GrovePi, that computing power is combined with the actuator-driving and sensor-reading ease of an Arduino (Orsini 2014). While this combination does require purchasing two boards rather than just one, that cost is offset by the ease of use. In addition, if one wants to take a break from robotics, one can take off the Arduino board to which the actuators and sensors are connected, pop out the microSD card containing the programming environment and put in one with just an OS, making the Pi simply a micro-computer again.

2.4 GrovePi

The GrovePi is a shield that plugs into the Raspberry Pi which makes it possible to "plug-and-play" Grove components (Seeed Studio 2016). I eventually chose the GrovePi (Figure 3) over the Arduberry for several reasons, namely that the GrovePi has better documentation, is simpler to use, and is cheaper.

Figure 3 GrovePi on Raspberry Pi
https://www.seeedstudio.com/GrovePi%2B-p-2241.html

Not only is the GrovePi well-documented, but it also has code in multiple programming languages for all of its standard Arduino components. Along with this, interacting with it is the same as interacting with any other Grove shield, so Grove's substantial userbase can be taken advantage of for ideas and debugging. In contrast, very few people seem to be using the Arduberry with Python so it is much harder to figure out what's going wrong with the interaction of the hardware and software. In addition, the Arduberry is designed to interface with Arduino shields rather than directly with components, meaning that putting them together tends to be a lot more hacky.

Along with that, Grove components tend to run a little bit cheaper than standard Arduino parts, ranging from one to twenty dollars with the majority in the four to eight dollar range. Standard Arduino components, on the other hand, tend to run between five and twenty-five dollars, with most seeming to be in the ten to fifteen dollar range. This difference, though seemingly small, begins to add up as multiple components are bought, especially for cash-strapped college students and professors.

Finally, while the code for controlling the components is included, it can be easily edited enabling those with experience to push the boundaries while still allowing beginners to get started quickly. In addition, Grove components are designed to interface easily not only with the board itself, but also with other hardware thanks to being situated on control boards reminiscent of puzzle pieces with pre-made holes. As can be seen in Figure 4, this makes them easy to secure on flat surfaces.

Figure 4 Project Made Using Grove Indoor Environment Kit for Intel

http://www.nkcelectronics.com/assets/images/grove_indoor_edison_02.jpg

2.5 3D-Printing

3D-printing allows users to quickly and cheaply, at least after the initial cost of the printer, create and replace parts. All that one needs is the printer itself, filament with which to print, and a file from which to print. This file can be created by the user or downloaded from a library and is generally of type .STL (from STereoLithography). .STL files are representations of 3D objects using triangles. 3D-printers can use this triangulated polyhedron in order to compose the object (Ennex Corporation 1999). One can even download a file from a library and modify it

to fit their project. Thus, beginners can quickly get started while those with the time and inclination have the freedom to customize their parts howsoever they choose. This combination of low cost and ease of access and modification makes it ideal for creating the structure and bodies of robots.

2.6 Project Jupyter

Project Jupyter is an open-source project with a goal of supporting "interactive data science and scientific computing across all programming languages" (Project Jupyter 2017). I am specifically using Project Jupyter's notebook, known as the Jupyter Notebook. The Jupyter Notebook is an open source web application that allows the sharing of code and documents (Project Jupyter 2017). In particular, I am using it to allow users to run code on the Raspberry Pi by starting an externally accessible Jupyter Notebook server on the Raspberry Pi. Once connected, they can run commands by starting up or opening a notebook and entering the commands there. They can also start up a terminal to safely shut down the Raspberry Pi.

3. Related Work

There are a few other robotics platforms with goals or implementations similar to mine. Of these, there are three that are most pertinent to my project with some key differences. The first of these is the Telepresence Robot Kit, or TeRK, developed by Carnegie Mellon University. This project consisted of instructions for building robots at home using easily accessible parts but is not particularly modular. The second is Lego MindStorms, which I mentioned above. This is a Lego based modular robotics kit, and is as simple to put together as most Lego products. However, it is difficult to use one's own parts with it, meaning that one is at the mercy of Lego in terms of both price and function. The final is the Poppy Project, which is a small collection of open-source 3D-printable robots.

3.1 TeRK

TeRK was a project of the Mobile Robot Programming Lab at Carnegie Mellon's Robotics Institute (Telepresence Robot Kit 2006). They designed the base hardware for it, including the Qwerk controller (Nourbakhsh, et al. 2007). TeRK was designed to be a library of designs for robots, referred to as 'recipes,' that anyone could build at home using parts from the hardware store; however, the project is no longer active (Robotics Institute 2010). The QwerkBot above is from a TeRK recipe. While not particularly modular, the design to be low cost (for 2007) and accessible are features particularly important to my project.

3.2 Lego MindStorms

Lego MindStorms are modular robots that come in kits. They are named after a book by Seymour Papert in which he explored the idea of using computers (and robots) to help children learn and get enjoyment from learning (Papert 1980). He helped Lego to create an actualization of these ideas with Lego MindStorms. Similar to the more common Lego products, these kits are a mix of specialized pieces, such as the actuators, and pieces that can be found in a variety of kits. While they are designed to be used in a classroom or other learning environment, people have used them to design more complicated robots, including a Lego duck maker and a Rubik's Cube solver. However, at $349.99 for the base kit as of February 2017, they are a bit pricy, especially considering that they are only designed to be extended with other Lego products

(LEGO 2016). Apart from the cost and hard-to-hack nature, the nature of my project should end up being similar to the modular nature of Lego MindStorms.

3.3 Poppy Project

A similar project that originated as a PhD thesis has become an open-source robotics platform known as the Poppy Project (Inria 2016). The Poppy Humanoid robot (Figure 5) was developed in order to study the impact of the body on sensorimotor development and cognition (Inria 2013). It was designed to easily and quickly conduct scientific experiments on sensorimotor learning, exploring morphology properties, and human-robot interaction.

> As an experimentation robotic platform, Poppy was designed to be affordable, lightweight, robust and safe, easy to use, versatile and fast and easy to duplicate or modify with the mid-term goal to make it easily reproducible by other lab through an Open Source distribution (hardware and software) (Lapeyre, Rouanet and Oudeyer 2013, 1).

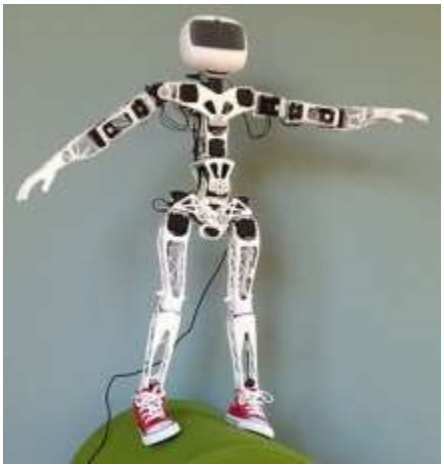

Figure 5 Poppy Humanoid
https://flowers.inria.fr/wp/wp-content/uploads/2013/05/poppyInria4.jpg

Matthieu Lapeyre and his advisor Pierre-Yves Oudeyer say that they were able to successfully accomplish this using 3D-printing, affordable off-the-shelf components and optimized mounting design (Lapeyre, Rouanet and Oudeyer 2013, 1). While more affordable than other humanoid robots of its size, the Poppy Humanoid runs a bit under $7000, which is quite expensive, especially for college students.

Since Poppy has moved to the public realm, users have gained the choice of three basic 3D-printable robots – a robotic arm (Figure 7), a robotic torso (Figure 6), or a robotic humanoid that looks about the same as and is similar to the original. The robotic arm, which runs about $285, can be capped with a lamp, a gripper, a pen holder, and more, which is where the modular part comes in. They are moved by DynaMixels, which are stackable actuators. Poppy creations are controlled by a Raspberry Pi or the similar Odroid board, both of which are small embeddable computers that run their operating systems off of memory cards.

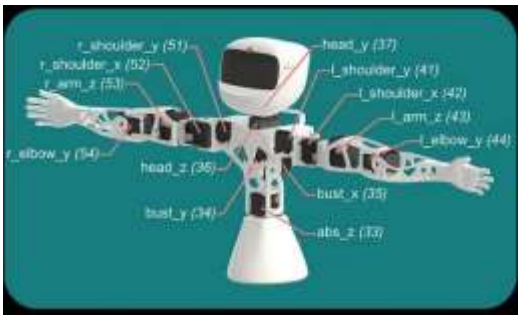

Figure 6 Poppy Torso
https://www.flickr.com/photos/poppy-project/26799515610/in/album-72157650211602329/

Figure 7 Ergo Jr. with Gripper and Ergo Jr. with Lamp
https://www.flickr.com/photos/poppy-project/26628822091/in/album-72157661379363331/

My project would ideally end up being a cheaper, more accessible version of Poppy. While DynaMixels are a viable option, they are expensive enough that for the basic design of my project will focus on standard servos. I believe that this will make my project more useful for hobbyists as well as those studying robotics

4. Project Design

My goal was to develop a Raspberry Pi based modular robotics system. In order to accomplish this, I had to do a lot of experimentation with different hardware. It was essential to find Raspberry Pi compatible hardware that would be intuitive to put together and develop a software base to control it. I needed to design a housing for the Raspberry Pi and peripherals such that it provided protection and could interface with robots without losing modularity. I also wanted to design a robot in order to ensure that the process would not be too arduous. It was necessary to develop a configuration file format that could be utilized by the software to control the robot. In short, my goals were to find the best and most economical hardware options, ensure that a library of robot designs could be easily created, and develop a way for the user to communicate with their robot. One thing to note is that I will be quite brand specific in this paper. While it is feasible, and in some cases relatively simple, to use or hack together alternatives, the goal is to create an introductory tool, so I will focus on the components that most easily enable that.

To go into more detail, each robot should consist of 3D-printed modular pieces with Arduino actuators that interface with a central housing containing the Raspberry Pi and a shield with which to connect the Arduinos. It should be comparable in price to college textbooks depending on the actuators and sensors chosen. The Adafruit Servo HAT is $17.50 as of the time of this writing. Since the Arduberry and GrovePi cost the same ($29.99 as of February 2017) which is just slightly under the cost of the Raspberry Pi, the base control boards should come out to just under $85. Arduino-compatible actuators and sensors generally run between $1 and $25, with most Grove (a specific brand of Arduino) compatible ones running between $4 and $8, so the entire cost of most robots should run well under $150.

One of my goals was to design the central housing. It needed to give protection without limiting access to the ports on the Raspberry Pi, the GrovePi, or the Servo HAT. I believe that the best design is a "bumper" case, which is essentially a frame that gives some protection against bumps. An example of this is shown in Figure 8. It also needs to be able to interface with the modular portions. While there are several ways to accomplish this, the most feasible would probably be to add rails on which it can fit. That way, apart from needing to take the size of the central housing into consideration, it would only be necessary to incorporate two rails into each robot design.

Figure 8 A Bumper Case Example
http://www.thingiverse.com/thing:629886

In order to allow users to easily get started with pre-made designs, it should have a web app where one can see different projects, choose the one they want to make, order the electronic components, and print the parts. There should also be a configuration file that includes the necessary information for the hardware-specific code in the project. This configuration file should have lines containing the name of the sensor or actuator, what it is (e.g. digital input or analog output), and to which port it is connected.

In order to create the projects, anyone should be able to choose their favorite CAD software and begin designing it based on what they want the robot to do. As mentioned above, the only limitation in the design is needing somewhere to interface the central housing. The designer should also print it out and ensure that the design is good. Along with that, they should create a configuration file; however, since that is relatively simple to do, a later user can add that as well.

In order to use the project base, learners will first need to set up the Raspberry Pi. This is done by downloading the operating system to a microSD card and plugging it into the Raspberry Pi. After the Raspberry Pi is working, the user will download the necessary packages and the software result of this project in order to be able to easily program their robot. More details on setting up the Raspberry Pi are located in Appendix A.

Once the Raspberry Pi has been set up, the user can code in one of three ways. The first of these methods is to connect a monitor, keyboard, mouse, and Ethernet cord to the Raspberry Pi and code on it like the computer it is. The second is similar, though uses a locally hosted Jupyter notebook rather than the terminal to code through (see section 2.6). If they follow that method, they can also disconnect the peripherals (except for the internet connection and power) after getting the IP address and connect to the Pi's Jupyter notebook from a different computer. This third method will allow portable robots, since all of the hardware can be run off of batteries. To be able to make it truly untethered, the Pi can be connected to WiFi and there can be a start-up script that starts the notebook server and displays the IP address, negating the need for a keyboard, monitor, or mouse.

Because of how Arduinos are coded, in order to actually influence the actuators and sensors, the program will need to know which components are connected to which ports. This information can be conveyed at the time of the robot's initialization by a configuration file or by typing out which part is connected to which port. At this point, the user can begin programming the robot.

For basic output such as LEDs, it is a simple matter of turning the control pin to HIGH (on) or LOW (off). For basic sensors, it is a matter of reading what the sensor senses. The trickier portion is for more complicated components, such as Arduino shields and LCD screens due to the fact that they require more specific commands. Since the Arduino IDE uses a derivative of *C* as a coding language, these commands need to be translated into Python and tailored to the Raspberry Pi's hardware. With the Arduberry, I experimented with RPi.GPIO and PyWiring and looked into Rhyduino to see which seems to be the most intuitive to use to control the Arduino input and output. The GrovePi, however, has an excellent library in many languages including Python, which is another reason I chose it.

For my robot class, the user should be able to use a single function with minimal information to write output and another to read input. It should also pull in some of the simpler built in functions either as they are or make them even more intuitive and allow the user to add in their own.

5. Project Implementation

I was able to find hardware options such that one or more robots can be built at a price comparable to college textbooks. I created a robot class in Python that allows simple and intuitive controls of a robot. I designed a central housing that should protect the control units of the robots without being overly bulky or interfering with the modularness of the platform. Finally, I designed one example robot and configuration file.

I started out trying the Arduberry with the Raspberry Pi but eventually decided to use the GrovePi. The GrovePi costs about the same as the Arduberry, but is a lot simpler to use with sensors and actuators. It also has both a larger code base and a larger user base, so it is easier to get started coding and get help when things go wrong.

Figure 9 Initial Setup with Arduberry

In order to develop the code for the users to use, I first had to learn how to use Grove actuators myself. I did this mostly by trial and error with some help from searching the internet. This was made easier by GrovePi's Python library. Once I was able to read input, I worked on generating output. I ran into a couple of bugs doing that, the most serious of which involved a never-ending squeal from a high-pitched buzzer. It was that which led me to hack together a Grove-compatible LED so if it got stuck on I could test it with visual aids rather than auditory. It turned out that the problem was that if output is set in an infinite loop and is stopped at the wrong time, it would get stuck on and would not respond to new code that should have turned it off. The best way of fixing this that I found was to switch ports or reset the pin mode (each must be set to "OUTPUT" or "INPUT"). Restarting the Raspberry Pi would also fix it. After I stopped using the demo code with the infinite loops, I no longer ran into the problem.

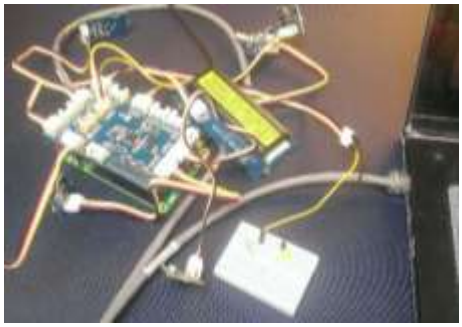

Figure 10 Experimentation with GrovePi

I did run into another issue; while looking into making servos work with the GrovePi, I discovered that they are not natively compatible. The reason being, as said by one of the lead developers for the GrovePi, is that even just one generates enough "noise" on the power supply to make the Raspberry Pi unstable or even restart it (Karan 2016). I looked into two possible

solutions. The first of these is using an Adafruit 16-Channel PWM / Servo HAT for Raspberry Pi which can control up to sixteen servos and costs under $20[1] but requires soldering the pins on. The second is using a Pixl board to control Robotis Dynamixel servos which comes ready to run but is about $30 along with the Dynamixel servos being a bit more expensive than standard ones. I ended up going with the servo HAT since it was easy to use and interface with the existing software.

I wrote a script that runs when the Raspberry Pi boots. This script runs commands that start the Jupyter notebook and display the IP address on a connected LCD screen as soon as the Pi is plugged in and connected to the internet. This makes it so that the user can begin coding from another computer immediately, without needing to connect the Raspberry Pi to a monitor.

```
In [8]: demoR.writeOutput('buzzer2', 1)
Out[8]: 1

In [9]: demoR.writeOutput('buzzer2', 0)
Out[9]: 1

In [10]: for x in range(0,10):
             print(demoR.readInput('lightSensor0'))
             time.sleep(.5)
689
672
755
768
566
341
204
242
241
678

In [11]: for x in range(0,10):
             print(demoR.readInput('pushButton6'))
             time.sleep(1)
0
0
1
1
0
0
0
0
1
0

In [13]: demoR.moveServo('servo0', 150)
```

Figure 11 Example Code

I have developed a base python class with which the user can communicate with the robot. It keeps track of which ports are in use through a dictionary and has functions that combine a lot of the necessary calls to set and use the sensors and actuators. It can be initialized with or without a configuration file. If no configuration file is given, it will prompt the user to input each component for a given type by first asking how many there are and then asking for their names and ports. There is also an addPort function for adding any components that might have been forgotten in initialization.

The main functions of the class are: readInput, which takes in the name of a sensor and returns the value the sensor outputs; writeOutput, which takes in a value along with the name of the actuator and turns the actuator on or off; and moveServo, which takes in a value along with the name of a servo and

[1] Prices as of February 2017

moves it by the specified value. Some example code can be found in Figure 11 and a list of the methods can be found in Appendix C. The full code can be found on GitHub.[2]

I have designed one robot and the central housing. The robot I decided to design is a plant sitter (see Appendix D). It consists of a pot and a stand, with spaces for a Grove Moisture Sensor and two Grove UV Sensors. A plant will go in the pot. The robot will be coded to move (or notify the user that it should be moved) towards the optimal amount of sunlight. Along with that, it will use the moisture sensor to determine if the plant needs to be watered. I experimented with adding a water pump so that the robot can automatically water the plant, but the current design flashes an LED instead. The central housing (Figure 12) is a bumper case as described above with the addition of supports for each board. These supports are designed to keep the GrovePi and Servo HAT horizontal rather than slanted. To incorporate it into a robot requires incorporating just two square rails. These rails make the central housing part of the robot by running in parallel either vertically or horizontally through the square holes in the central housing.

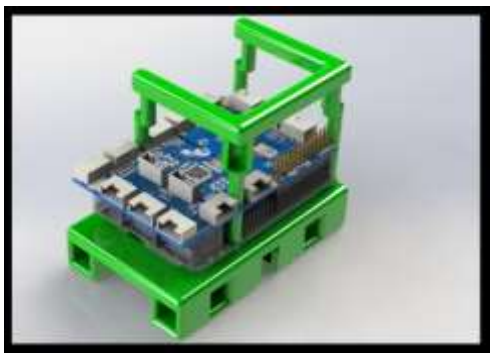

Figure 12 3D Model of Central Housing
Note that the model does not include the Servo HAT

Overall, I believe that this can be further developed as described in section 6 to create a viable teaching platform.

[2] https://github.com/samphibian/patzah-project.git

6. Future Work

What is left to be done is to expand the library of robots, implement an online database for easy access, and have a robot help forum. Expanding the library of robots is rather straightforward as it can be done by coming up with an idea for a robot, designing a body for it, and tweaking the design until it works. The online database should have several key components. The first of these is the display. I think there should be three browse options: by name, by picture, and by function. Along with this, when a robot design is selected, it should have a link to download the configuration file and links to the components. There should also be a base kit consisting of the Raspberry Pi, GrovePi, and central housing that is a separate listing linked to in the parts list. The final thing to make this a viable teaching platform is to make it popular. This would give it a good userbase that could then not only continue expanding the database, but also answer questions in some sort of forum. One additional thing that could be done is port the functionality to additional languages so that the robots are not restricted to Python users.

7. Conclusions and Summary

After much experimentation, I found hardware that works well together, is modular, and easy to use. I created an easily expandable Robot class in Python that allows easy reading and manipulation of sensors and actuators. I also designed a central housing for the control unit of the robot that gives protection without sacrificing access. Finally, I tested the design process and found that it was relatively straightforward to design a robot. Overall, I believe that this can be further developed to create a viable teaching platform with all of the benefits of personal robotics, the fun of being able to choose a robot from many options or design one's own, and the ease of coding in a popular language.

Acknowledgements

This project would not have turned out as it did without the support of many people. Thanks to Professor Blank without whom this project would not exist. Thanks to Calla, Jordan, and Sophie for proofreading and suggestions. Special thanks to Rob Cunningham and Rich Willard in the BMC Machine Shop for printing all of my prototypes and caring about how they came out. Thanks to Professor Kumar for his comments and suggestions. Thanks to SEEED for having such lovely documentation for the GrovePi. As always, thanks to my family for their continued support.

References

Arduino AG. 2017. *Arduino - ArduinoShields.* Accessed March 30, 2017. https://www.arduino.cc/en/Main/arduinoShields.

—. 2017. *Arduino - Introduction.* Accessed March 6, 2017. https://www.arduino.cc/en/Guide/Introduction.

Eilbert, Natasha. 2007. *Student Reaction to a Computer Science Course Using Robots | Institute for Personal Robots in Education Blog* . Accessed February 9, 2017. http://blog.roboteducation.org/node/22.

Ennex Corporation. 1999. *The StL Format.* Accessed April 6, 2017. http://www.fabbers.com/tech/STL_Format.

Geerling, Jeff. 2016. *Review: ODROID-C2, compared to Raspberry Pi 3 and Orange Pi Plus.* March 24. Accessed March 5, 2017. https://www.jeffgeerling.com/blog/2016/review-odroid-c2-compared-raspberry-pi-3-and-orange-pi-plus.

Harel, Idit. n.d. *Professor Seymour Papert.* Accessed April 7, 2017. http://papert.org/.

Inria. 2013. *Flowers Laboratory.* Accessed February 9, 2017. https://flowers.inria.fr/.

—. 2016. *Poppy Project - Open Source Robotic Platform.* Accessed January 17, 2017. https://www.poppy-project.org/en/.

Karan. 2016. *Raspberry pi, Python, and servo - GrovePi - Dexter Industries Forum.* June. Accessed February 22, 2017. http://forum.dexterindustries.com/t/raspberry-pi-python-and-servo/981.

Kumar, Deepak. 2008. *Learning Computing With Robots.* Institute for Personal Robots in Education.

Lapeyre, Matthieu, Pierre Rouanet, and Pierre-Yves Oudeyer. 2013. "Poppy Humanoid Platform: Experimental Evaluation of the Role of a Bio-inspired Thigh Shape." Atlanta, September 11.

LEGO. 2016. *LEGO® MINDSTORMS® EV3 | LEGO Shop.* Accessed February 9, 2017. https://shop.lego.com/en-US/LEGO-MINDSTORMS-EV3-31313 .

Markham, Stefanie A., and K. N. King. 2010. *Using Personal Robots in CS1: Experiences, Outcomes, and Attitudinal Influences* . ACM.

McGill, M. M. 2012. "Learning to Program with Personal Robots: Influences on Student Motivation." *ACM Transactions on Computing Education* 4:1-4:32.

Nourbakhsh, Illah, Emily Hamner, Tom Lauwers, Carl DiSalvo, and Debra Bernstein. 2007. *TeRK: A Flexible Tool for Science and Technology Education* . Carnegie Melon University.

Orsini, Lauren. 2014. *Arduino Vs. Raspberry Pi: Which Is The Right DIY Platform For You?* May 7. Accessed February 9, 2017. http://readwrite.com/2014/05/07/arduino-vs-raspberry-pi-projects-diy-platform/.

Papert, Seymour. 1980. *Mindstorms: Children, Computers, and Powerful Ideas.* New York: Basic Books, Inc. .

Project Jupyter. 2017. *Project Jupyter.* Accessed April 7, 2017. https://jupyter.org/.

Resnick, Mitchel, Brad Myers, Kumiyo Nakakoji, Ben Schneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. *Design Principles for Tools to Support Creative Thinking.* 30 October. Accessed February 9, 2017. http://www.cs.umd.edu/hcil/CST/Papers/designprinciples.htm.

Robotics Institute. 2010. *Robotics Institute - Telepresence Robot Kit.* Accessed March 30, 2017. http://www.ri.cmu.edu/research_project_detail.html?project_id=565&menu_id=261.

Scott, Michael James, Steve Counsell, Stanislao Lauria, Stephen Swift, Allan Tucker, Martin Shepperd, and Gheorghita Ghinea. 2015. "Enhancing Practice and Achievement in Introductory Programming with a Robot Olympics." *IEEE TRANSACTIONS ON EDUCATION* 249 - 254.

Seeed Studio. 2016. *GrovePi+.* Accessed March 6, 2017. https://www.seeedstudio.com/GrovePi%2B-p-2241.html.

Telepresence Robot Kit. 2006. *TeRK - Project Description.* April 25. Accessed February 28, 2017. https://www.cs.cmu.edu/~terk/project.html.

Appendix A – Setting up the Raspberry Pi:

- Install the Raspbian operating system on a microSD card (https://www.raspberrypi.org/downloads/)
- Download the folder I have created (most likely through a GitHub repository)
- Run the following commands:
    - sudo pip3.4 install pip –u
    - pip –version
        - should be pip 9.0.1 from /usr/... (python 3.4)
    - sudo pip install jupyter notebook -u
        - Note that running `jupyter notebook --config='/home/pi/.jupyter/jupyter_notebook_config.py' &` will start the Jupyter server configured as in the file in the specified path. This path might need to be edited to match where the configuration file is located.
    - Sudo crontab –e
        - Add the following commands to the file to launch the Jupyter notebook server on startup and display the IP address (note that the paths might need to be edited)
            - @reboot sleep 15 && sh /home/pi/displayAddress.sh >/home/pi/logs/cronlogIP 2>&1
            - @reboot sh /home/pi/launchJupyterServer.sh >/home/pi/logs/cronlogShell 2>&1
- Once this has been done, the Jupyter Notebook myPy.ipynb can be accessed by connecting to localhost:8888 from the Raspberry Pi or by connecting to [its IP address]:8888 from another computer. Code should either be in this notebook or have this notebook included.

Appendix B – Software Resources:

http://www.robotshop.com/blog/en/arduino-5-minute-tutorials-lesson-5-servo-motors-3636

http://www.instructables.com/id/Raspberry-Pi-Launch-Python-script-on-startup/?ALLSTEPS

http://jupyter-notebook.readthedocs.io/en/latest/public_server.html

http://blog.roboteducation.org/node/47

https://docs.python.org/2/tutorial/inputoutput.html

https://www.dexterindustries.com/GrovePi/get-started-with-the-grovepi/setting-software/

Fixing Raspberry Pi keyboard: https://www.raspberrypi.org/forums/viewtopic.php?f=6&t=5278

Get IP address at boot:

http://unix.stackexchange.com/questions/57852/crontab-job-start-1-min-after-reboot

http://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib/1267524#1267524

GPIO: http://maxembedded.com/2014/07/using-raspberry-pi-gpio-using-python/
http://www.thirdeyevis.com/pi-page-2.php

Appendix C – Main Class Methods

Function	Input	Output	Use
readInput	Sensor name	Sensor Value	Get the value of a sensor
writeOutput	Component name, Value	Passes the specified value to the specified component	Make something do something, e.g. turn an led or buzzer on/off
moveServo	Servo name, Value	Moves the specified servo by the specified value	Move a servo
readEncoder		Encoder Value	Get the value of an attached encoder (a device that measures changes in angles)
readTemp		Temperature	Get the current temperature from a Grove Temperature&Humidity Sensor
readHumidity		Humidity	Get the current humidity from a Grove Temperature&Humidity Sensor
display	Color, text	Displays text with given background color on LCD screen	Display text to a Grove LCD RGB Backlight

Appendix D – the PlantBot:

Example Code:

```
# plant robot

usens1 = 'uvSensor1'
usens2 = 'uvSensor2'
led2 = 'led2'
led3 = 'led3'
led4 = 'led4'
mSens0 = "moistureSensor0"

plantRobot = Robot('configFile')

oldu1 = -1
oldu2 = -1
oldm = -1
moistureMin = 300

while(True):
    u1 = plantRobot.readInput(usens1)
    u2 = plantRobot.readInput(usens2)
    m = plantRobot.readInput(mSens0)

    if(u1 != oldu1 or u2 != oldu2):
        oldu1 = u1
        oldu2 = u2
        if(u1 > u2):
            plantRobot.writeOutput(led2, 1)
            plantRobot.writeOutput(led4, 0)
            print("move plant to the right: ", u2)
        elif(u1 < u2):
            plantRobot.writeOutput(led2, 0)
            plantRobot.writeOutput(led4, 1)
            print("move plant to the left: ", u1)
        else:
            plantRobot.writeOutput(led2, 0)
            plantRobot.writeOutput(led4, 0)

    if(m != oldm):
        oldm = m
        if(m < moistureMin):
            plantRobot.writeOutput(led3, 1)
            print("water plant: ", m)
        else:
            plantRobot.writeOutput(led3, 0)

    time.sleep(1)
```

Figure 13 3D-Printed Stand

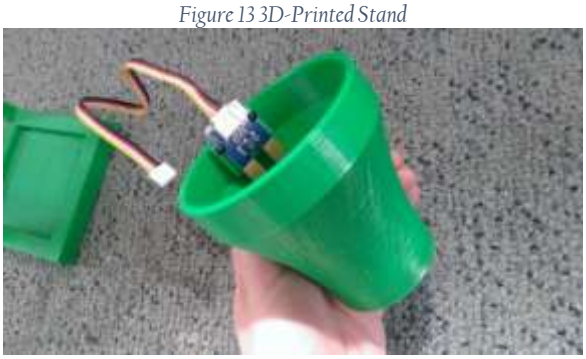

Figure 14 3D-Printed Pot

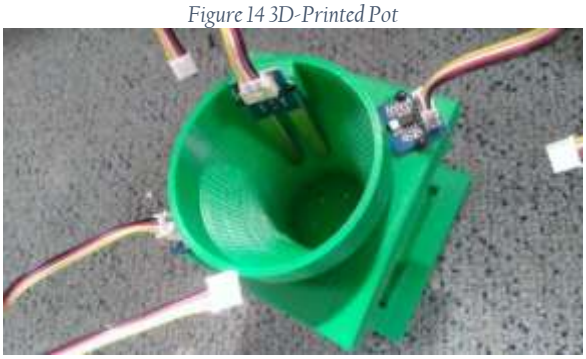

Figure 15 Pot in Stand

Figure 16 PlantBot with Dirt