May 17, 2014

Using the BeagleBone Black on a robot bicycle

Biorobotics and Locomotion Lab

Mukund Sudarshan

Freshman, MAE 1900 – 3 Credits

ms2666@cornell.edu, 607-379-2405

CUID: 3383050

Team members:

1. Ariam Espinal
    a. Junior
    b. aee49@cornell.edu
2. Arundathi Sharma
    a. Freshman
    b. ams692@cornell.edu
3. Diego Olvera
    a. M.Eng
    b. do98@cornell.edu
4. Frederick Koennecke
    a. M.Eng
    b. fmk27@cornell.edu
5. Kate Zhou
    a. Sophomore
    b. ykz2@cornell.edu
6. Mert Ürkmez (Non-credit)
    a. Freshman
    b. do98@cornell.edu

Table of Contents

Abstract

Title: Robot Bicycle — controlling an electric bike with a BeagleBone black

Abstract:

I was part of a team whose goal was to construct a self-stabilizing, self-propelling bicycle. My assignment was to research and implement the BeagleBone Black, an open-source mini-computer that runs Linux, in the specific context of this project. With the assistance of other members in Professor Andy Ruina's lab, the team understood that there were a few main tasks with the BeagleBone, such as **motor control for steering and propulsion, feedback control loops for balance and a failsafe mechanism**. I set about figuring out how to facilitate all of these using this portable computer. There are three main parts to this paper. First, the software setup of the BeagleBone Black will be covered. The software includes both the operating system running on the BeagleBone and the programming languages used. Then, each required task will be detailed, explaining how the setup process was useful in doing so. Finally, the hardware setup of the BeagleBone will also be explained.

Introduction

The motivation behind the robot bicycle is to investigate how humans maintain balance a bicycle while riding it and how the steering of the bike can help with this. A bicycle when pushed forwards is seen to correct its roll on its own.[1] While it is moving forwards, lateral oscillations about the horizontal plane of the bike are dampened as the steering adjusts itself. In theory, this self-stabilizing behavior can be predicted by certain equations of motion[2]. To understand if these equations are actually accurate in real life, we decided to build a bicycle with motors controlling propulsion and steering to see if balance could be maintained using just steering and propulsion.

In order to implement this idea, a computer was needed to take inputs from various sensors, regarding the bike's roll and steer angles, process these using an equation of motion and send outputs to the motors on the bike. For our purposes, we chose the BeagleBone Black to do all of this. To understand how the BeagleBone Black is ideal for us, it can help to think of it as a cross between an Arduino and a Raspberry Pi. While the simple infrastructure of the Arduino is mainly intended for hobbyists, the Raspberry Pi is targeted towards developers. This is chiefly due to the fact that the Raspberry Pi isn't as easy to setup and use, but is capable of performing tasks much faster. An example of this would include the speed of floating point calculations. While the Arduino seriously lacks in this aspect, the Raspberry Pi is notably faster. The BeagleBone tries to combine the best of both worlds. This makes the BeagleBone Black an ideal candidate for the computer to be used in the robot bicycle. Below is a comparison between the three. Only categories relevant to this project are mentioned.

Name	Arduino Uno R3	Raspberry Pi Model B	BeagleBone Black Revision A
Processor	ATMega 328	ARM 11	AM3358
Clock Speed	16 MHz	700 MHz	1 GHz
RAM	2 KB	256 MB	512 MB
GPIO Pins	14	8	65
Analog Input Pins	6	0	7
UART	1	1	4
Ethernet	N/A	Yes	Yes
Video Output	N/A	HDMI	HDMI
Floating Point Calculation	N/A	Yes	Yes

Table 1: Computer Comparison

[1] http://ruina.tam.cornell.edu/research/topics/bicycle_mechanics/overview.php
[2] Refer to Diego Olvera's 2014 paper on dynamics for the robot bicycle

Setup of BeagleBone Black

1. Installed latest version of Angstrom Linux v2013-09-04
2. Installed Python v2.7.6
3. Installed Adafruit BeagleBone Input Output (BBIO)
4. Use BBIO library to access[3]
    a. GPIO
    b. PWM
    c. ADC
    d. UART

Angstrom

For our purposes, **Angstrom Linux** was the distribution of choice. It is a non-processor intensive version of Linux that is designed to run on embedded devices[4]. Some reasons for us choosing Angstrom included it being the official language of the BeagleBone Black, meaning it comes with a custom version of Angstrom Linux when you buy it, but more importantly, it is the only operating system BeagleBone's manufacturers will release updates to and add libraries[5] to. Another important feature of Angstrom is that it allows the users to install Python, a general-purpose, high-level programming language[6]. The following is the process by which we installed Angstrom, using a 2012 MacBook Pro[7]. There are two options while installing an OS: running it off a microSD card, or running it off the internal memory. We will be exercising the latter option, as the former would limit the speed of programs to the read-write speed of the microSD card reader.

Download

We must first download the version of Angstrom Linux we want to install on the BeagleBone Black. It is recommended to use the latest one. Since Angstrom is the official language of BeagleBone for this device, they have a source conveniently located on their main website: http://beagleboard.org/latest-images/. The download is an image (.img) file, which contains data the BeagleBone Black requires to install Angstrom.

Copying image to microSD card

This is our way of transferring the image file from the computer to the BeagleBone Black. Using a 16GB microSD card and a microSD to SD card adapter for the MacBook Pro, we transfer this image using a program called ApplePi Baker. Intended for use with the Raspberry Pi, the assumption that it would also work with the BeagleBone Black was proved correct. This is a simple graphical user interface for a process that can also be done through a command line.

[3] These are explained in Appendix B
[4] https://learn.adafruit.com/beaglebone-black-installing-operating-systems/angstrom
[5] A collection of standard programs and subroutines for immediate use, usually stored on disk or some other storage device (http://dictionary.reference.com/browse/library)
[6] http://www.tiobe.com/index.php/paperinfo/tpci/Python.html
[7] http://beagleboard.org/Getting+Started

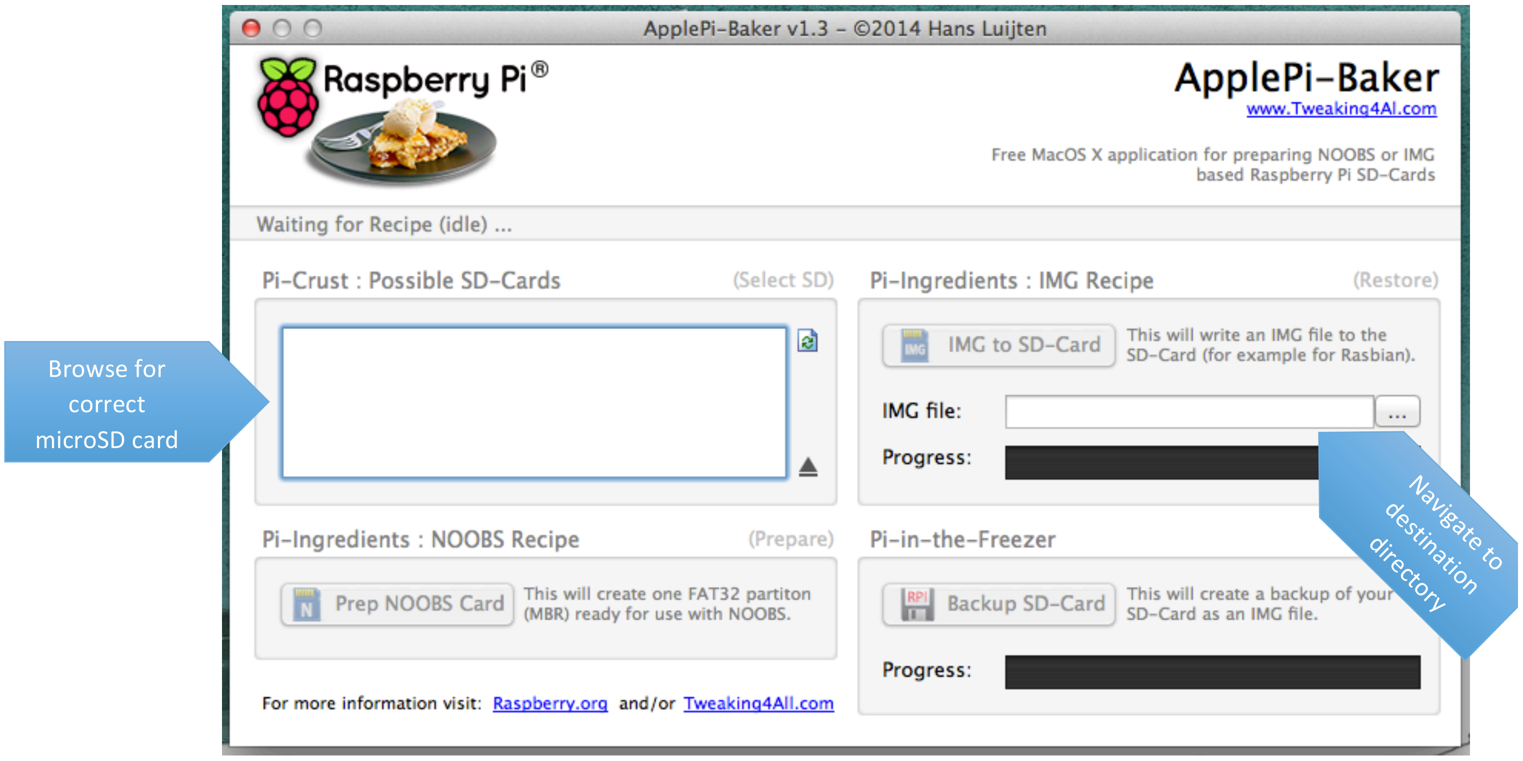

ApplePi-Baker v1.3 – ©2014 Hans Luijten
Raspberry Pi®
ApplePi-Baker
www.Tweaking4Al.com
Free MacOS X application for preparing NOOBS or IMG based Raspberry Pi SD-Cards
Waiting for Recipe (idle) ...
Pi-Crust : Possible SD-Cards
(Select SD)
Pi-Ingredients : IMG Recipe
(Restore)
IMG to SD-Card
This will write an IMG file to the SD-Card (for example for Rasbian).
IMG file:
Progress:
Pi-Ingredients : NOOBS Recipe
(Prepare)
Prep NOOBS Card
This will create one FAT32 partiton (MBR) ready for use with NOOBS.
Pi-in-the-Freezer
Backup SD-Card
This will create a backup of your SD-Card as an IMG file.
Progress:
For more information visit: Raspberry.org and/or Tweaking4All.com
Browse for correct microSD card
Navigate to destination directory

Figure 1: ApplePi Baker GUI

Flashing the BeagleBone Black

To make sure that the BeagleBone's internal storage uses the latest version of Angstrom, we must copy, or flash, the image file from the microSD card to the BeagleBone. Making sure the power source for the BeagleBone isn't connected, insert the microSD card. Holding down the "User Boot" button, power the BeagleBone until all four onboard LEDs are solid on, then let go of the button. The LEDs will now start to flash, indicating the transfer process has started. Wait for around 45 minutes and the 4 LEDs will be solid again. Disconnect the power and eject the microSD card to finish installing Angstrom[8].

[8] http://makezineblog.files.wordpress.com/2013/07/beaglebone-black-front.jpg?w=620&h=396

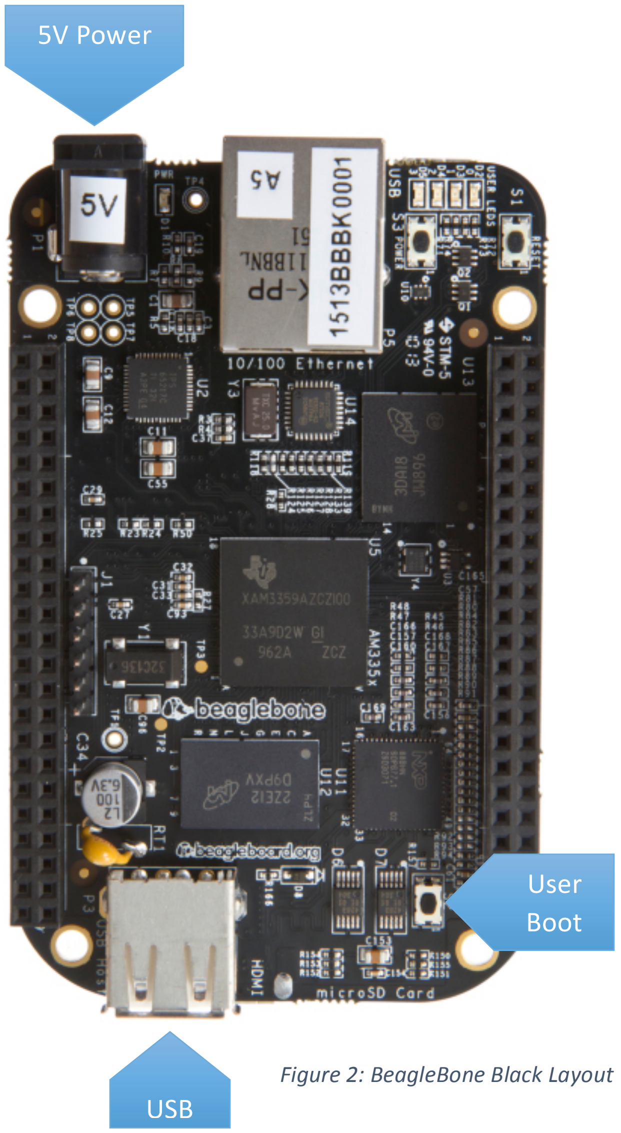


Figure 2: BeagleBone Black Layout

Python

Even though the BeagleBone Black comes with a version of JavaScript pre-installed, we are going to use Python in order to use serial communication. As of now, there are serial communications libraries written only in Python. To install Python on the BeagleBone, we need to first access its command line. The easiest way to do this is by using the **S**ecure **Sh**ell (SSH) protocol. SSH allows users to perform a variety of tasks but for our purposes, we'll be using SSH to remotely access the BeagleBone's command line via a 2012 MacBook Pro.

Driver installation

Obtaining the network driver for a Mac is simple; BeagleBone has conveniently placed it on the main website: http://beagleboard.org/static/Drivers/MacOSX/RNDIS/HoRNDIS.pkg. This driver ensures that applications that work over real Ethernet interfaces will work over a USB interface without modification, because they can't tell that they aren't using real Ethernet hardware[9]. We'll also be requiring a serial driver for the Mac as the latest models no longer come with serial ports: http://beagleboard.org/static/Drivers/MacOSX/FTDI/FTDI_Ser.dmg. This allows serial communication via USB.

SSH to BeagleBone Black

To SSH on a Mac is quite simple, open the "Terminal" program that is preinstalled, then type in the following command. Note that this is a static IP address and will not change.

```
ssh root@192.168.7.2
```

Upon being prompted for a password, we can simply hit "enter" as there is no pre-set password.

Installing Python[10]

Python is downloaded from an online repository so we must first connect to the Internet using an Ethernet cable. Simply entering the following lines one by one will install Python successfully.

```
/usr/bin/ntpdate -b -s -u pool.ntp.org

opkg update && opkg install python-pip python-setuptools python-smbus
```

The first line updates the time on the BeagleBone Black as there is no internal battery to maintain time. The next line updates opkg, a lightweight packet manager used in downloading and installing software. It then also installs python and the relevant packages needed for this project.

Adafruit BBIO

This is the simplest step of the install process. Once Python is successfully installed, simply type the following command in the terminal window. The command used here, "pip," is a package manager specifically designed for Python libraries. The Adafruit BBIO library is written in Python.

```
pip install Adafruit_BBIO
```

We are now able to access the various features of the BeagleBone Black via Python.

Programming Infrastructure

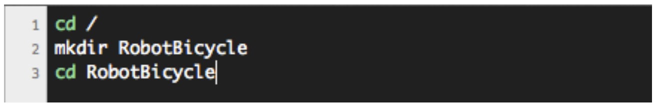
To do anything with our prepared BeagleBone Black, we need to write Python code, and then upload it to the BeagleBone to run it. There are three things we need to do:

[9] http://www.embedded.com/design/connectivity/4024491/Linux-based-USB-Devices
[10] https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/installation

1. Create a Python directory for our files
2. Create a method to automatically run certain files upon starting the BeagleBone Black
3. Create Python files

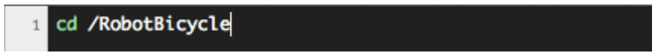
Creating a directory

Once we've SSHed into the BeagleBone, we enter the following commands:



```
cd /
mkdir RobotBicycle
cd RobotBicycle
```

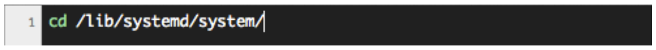
Here's what happens: we navigate to the root (top-most) directory of the BeagleBone, and then create a folder called "RobotBicycle." Finally we navigate to this directory where we can place files. Anytime we want to navigate to this folder, simply type in:



```
cd /RobotBicycle
```

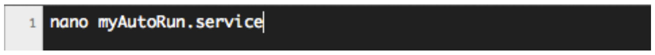
Autorun files

For this, we use a feature of Angstrom called "service files." Service files allow an action to be run upon startup[11]. Here, we create a custom service file to run a specific program in our directory. First, browse to the directory where service files must be placed:



```
cd /lib/systemd/system/
```

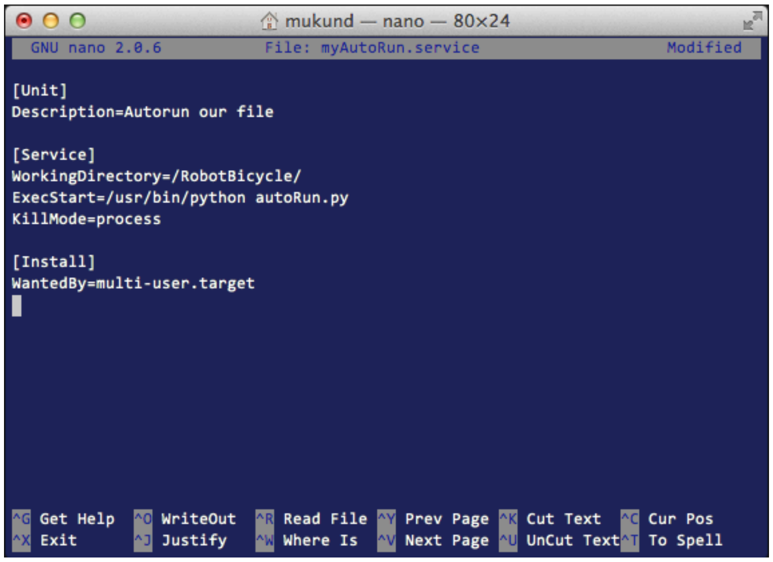
We then use the command below to open an edit window:



```
nano myAutoRun.service
```

Entering this code below, then exiting will prompt a message asking to save. Upon confirming, we will have created a service file.

[11] http://pcsupport.about.com/od/fileextensions/f/servicefile.htm



```
mukund — nano — 80×24
  GNU nano 2.0.6              File: myAutoRun.service                  Modified

[Unit]
Description=Autorun our file

[Service]
WorkingDirectory=/RobotBicycle/
ExecStart=/usr/bin/python autoRun.py
KillMode=process

[Install]
WantedBy=multi-user.target


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

This service file essentially looks for a file in the directory we created earlier. It then runs a Python program called "autoRun.py." It stops running after one successful execution. Be sure to create a file called "autoRun.py" in the correct directory even if it doesn't have any code in it.

All that remains is the activation of this file:

```
systemctl enable rfidreader.service
```

Creating Python Files

After browsing to the correct directory, use the following command to open up a Python edit window. It will look identical to the one above. Replace the square brackets and its contents with any name desired, and don't forget to include the Python extension.

```
nano [insert name here].py
```

To run the program, simply enter:

```
python [insert name here].py
```

Software Applications

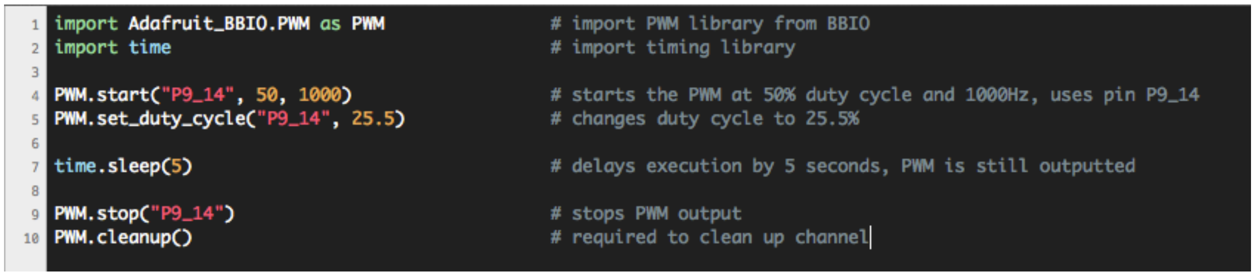
Motor control – Propulsion

For propulsion, we chose Golden Motor's Magic Pie 2X[12], a brushless DC electric hub motor. The Magic Pie 2X is a well-known motor used to convert regular bicycles into electric ones. It is sold as a kit including a motor controller and a throttle. In order to control the speed of the motor, we looked at how the throttle communicates with the motor controller. The throttle was simply a variable resistor, which took in 5V from the motor controller and fed back an analog signal of 0 to 5V.

Figure 3: Magic Pie 2X

In order to replicate this process using the BeagleBone Black, we decided that a **P**ulse **W**idth **M**odulation (PWM) signal, via a low-pass filter and an opto-isolator[13] would be useful in creating an analog signal. This particular setup is required, as the BeagleBone does not have an analog output on its own. The preloaded `analogWrite()` function simply outputs a PWM signal.

This required a program that simply involves outputting PWM at various duty cycles.


```
import Adafruit_BBIO.PWM as PWM              # import PWM library from BBIO
import time                                  # import timing library

PWM.start("P9_14", 50, 1000)                 # starts the PWM at 50% duty cycle and 1000Hz, uses pin P9_14
PWM.set_duty_cycle("P9_14", 25.5)            # changes duty cycle to 25.5%

time.sleep(5)                                # delays execution by 5 seconds, PWM is still outputted

PWM.stop("P9_14")                            # stops PWM output
PWM.cleanup()                                # required to clean up channel
```

The pin seen here, "P9_14," is a PWM pin on the BeagleBone Black. This program will change only slightly in the future, as it will take in commands from an RC receiver.

Feedback control loops for balance

In terms of balance, we needed the BeagleBone Black to use a dynamics equation[14] to take in various inputs like the roll angle, the steer angle, current, etc.[15] The BeagleBone then multiplies these states by a feedback matrix. The output of the equation is a torque value that goes to the steering. As this equation

[12] http://www.goldenmotor.com/goldshop/product/229.html
[13] Refer to Diego Olvera's 2014 paper on dynamics for the robot bicycle
[14] Refer to Diego Olvera's 2014 paper on dynamics for the robot bicycle
[15] Refer to Diego Olvera's 2014 paper on dynamics for the robot bicycle

is still a work in progress, we haven't yet been able to write it in Python. However, we do know that this system of actuation will be in the form of a feedback loop as shown below.

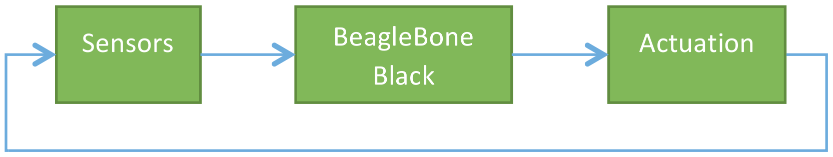


Figure 4: Motor Control Feedback Loop

The sensors will relay information such as roll angle, steer angle, etc. to the BeagleBone for processing which will then actuate the motors on the bike. This actuation will send new information via the sensors back to the BeagleBone, allowing for further changes to the actuation.

Safety

In the event of a malfunction with our robot, such as the computer crashing or not responding properly, we need a system to cut all power from the battery to the motors. Both an automated safety check and a manual override are required. For this mechanism, there are two main parts as shown in the diagram below:

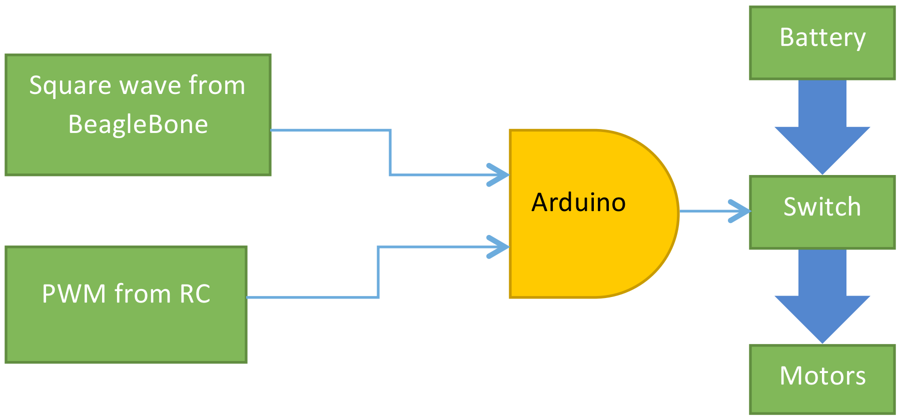
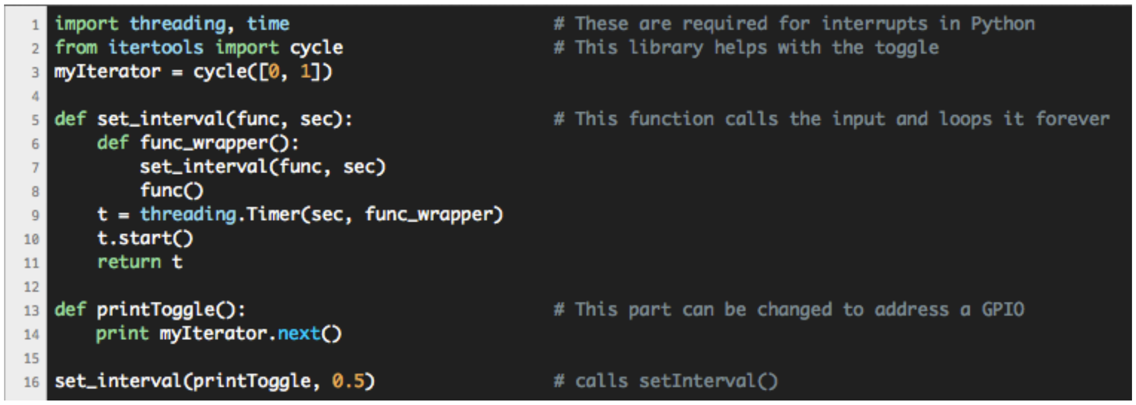


Figure 5: Safety Shutoff Mechanism

In the first part, there is a square wave coming out of one of the BeagleBone's GPIOs. In the second part, there is a constant duty cycle PWM from the RC receiver being read by an Arduino. Using the `pulseIn()` function on the Arduino, we are able to measure the duty cycle of both the square wave and the PWM. If these values are within their respective tolerances, a GPIO on the Arduino is set high and this turns the switch on, allowing the battery to power the motors on the bike. Essentially, the Arduino functions as an AND gate. The user can stop the PWM from the RC manually or the BeagleBone could crash. In either scenario, the output of the AND gate would be low, cutting power to the motors.

Timing function

Since at this point in time the BeagleBone is quite new, there aren't many libraries written for it. This is why we wrote our own functions for timing. For tasks like outputting a square wave, we needed a function that would toggle a GPIO at a set interval. This function exists in JavaScript and is called `setInterval()`. It takes in 2 inputs: a function and a time interval. The code below is the same function but in Python.



```python
import threading, time                                       # These are required for interrupts in Python
from itertools import cycle                                  # This library helps with the toggle
myIterator = cycle([0, 1])

def set_interval(func, sec):                                 # This function calls the input and loops it forever
    def func_wrapper():
        set_interval(func, sec)
        func()
    t = threading.Timer(sec, func_wrapper)
    t.start()
    return t

def printToggle():                                           # This part can be changed to address a GPIO
    print myIterator.next()

set_interval(printToggle, 0.5)                               # calls setInterval()
```

Steering

To steer the bicycle, we decided to use a DC motor found in the lab, a Yaskawa Minertia J02L. This motor takes in power via a powered PWM signal. In order to create a PWM of this magnitude, we make use of another opto-isolator; the low power PWM is amplified without the BeagleBone being affected by the higher current. Steering happens in another feedback loop as illustrated below.

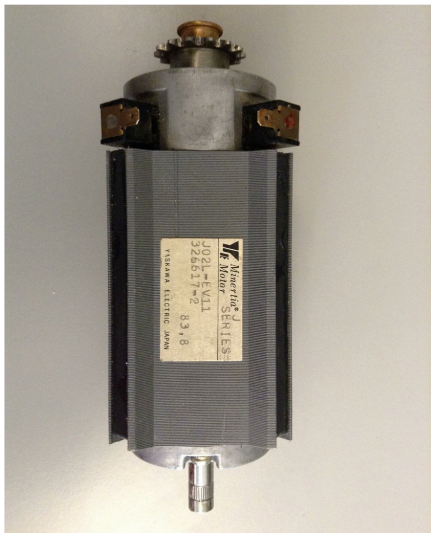


Figure 6: Yaskawa Minertia J02L

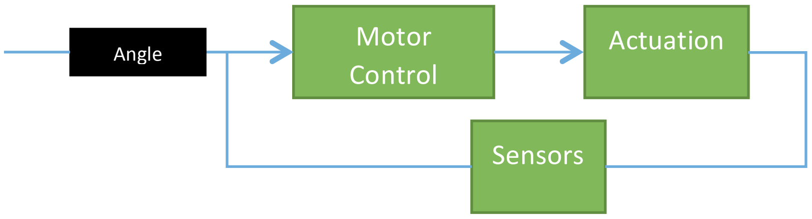


Figure 7: Steering Feedback Loop

The steering program running on the BeagleBone will send the motor control program the required angle. Then, the motor control program applies a torque to the motor. The new position is relayed back to the motor control program using a rotary encoder so that the program can decide whether or not to correct the position to match the angle.

Arduino

As mentioned earlier, there are very few libraries written for the BeagleBone Black, restricting our access to its features. As a result, there wasn't any simple way of reading PWM signals. This requires access to the onboard of the BeagleBone. To simplify the process, we used the Arduino, which has the `pulseIn()` function mentioned earlier. When the rotary encoder on the motor outputs a PWM signal to indicate the position of the rotor, the Arduino measures the duty cycle and outputs this value to the BeagleBone Black via its analog output pin.

Discussion

This was my first time working in a research lab and was quite confusing at first. A lot of my time during the first few weeks was spent on finding something for me to do that was useful to the larger goal of the project, but at the same time possible with my current skill level. For this project, I was tasked with two things: finding a suitable propulsion motor and interfacing the BeagleBone Black with our hardware. The first one alone took me at least two weeks to do as there weren't too many motors out there that would serve our purpose well. Not too many manufacturers design electric bike kits that are controllable by a computer. After this however, the pace picked up.

In order to start using the BeagleBone with Bonescript, a JavaScript library written to control the BeagleBone's various features, I had to first learn JavaScript. Then when we found out that there are no serial communication libraries in Bonescript, we had to switch to Python, the only language – supported on the BeagleBone – with which we could read data from our inertial measurement unit. For this, I had to learn Python and its programming style in order to make reasonable progress. After finalizing on Python, it was just a matter of writing the programs.

Even though UART libraries exist in Python for the BeagleBone, there are plenty still missing. This forced another team member and I to create our own programs emulating even simples functions like `setInterval()`. It was surprising to see none had been written already.

Goals for next semester

For next semester, there are two main goals with respect to using the BeagleBone Black. The first is getting the hardware up and running so that the software can be tested. The next one is to learn more about how to access the features on the BeagleBone in order to decrease dependence on external hardware.

Over the summer, our team will have hopefully completed all the hardware mounting and installation. Once we accomplish this difficult task, we will be able to test our programs together. As of now, each segment has only been tested separately. For example, the motor control program for the propulsion works, but we don't know if it will work with everything else running at the same time.

We were forced to use an Arduino this semester, as we were unable to find the correct libraries to help read PWM. During the next semester, we should focus more on learning how to create our own libraries. For example, we are aware that there are timers on the BeagleBone but are not sure how to access them. Once we do this, we no longer need an Arduino to read our PWM.

Appendix A

Please refer to PowerPoint file included in zip file.

Appendix B

Additional notes:

- Before the team decided on BeagleBone as the computer to use, I was tasked with finding a propulsion motor. The Magic Pie 3 was initially chosen but due to the embedded motor controller, the older Magic Pie 2X was more ideal for our purposes. The Magic Pie 3 also had more power than was necessary (1000W).
- Before deciding on Angstrom, we experimented with Ubuntu, as Ubuntu Linux would be used in other projects in the lab. However, there were no device tree overlays (DTO) that work well with

Ubuntu. This is a serious problem as high-level programs written in Python would not be able to access the pins on the BeagleBone. We also tried running both operating systems off a microSD card, but found this to be much slower in terms of startup.

- Key terms used in this paper:
    - General Purpose Input Output (GPIO) – generic pin on an integrated circuit whose behavior, including whether it is an input or output pin, can be controlled by the user at run time.[16]
    - Pulse Width Modulation (PWM) – modulation technique that conforms the width of the pulse, formally the pulse duration, based on modulator signal information.[17]
    - Analog to Digital Converter (ADC) – electronic circuit that converts continuous signals to digital numbers
    - Universal Asynchronous Transmitter Receiver (UART) – method of serial communication between devices
    - Device Tree Overlays (DTO) – way to describe hardware in a system. An example of this would be to describe how the UART interfaces with the system, which pins, how they should be multiplexed, the device to enable, and which driver to use.[18]
- Main works cited
    - Diego Olvera's 2014 paper on dynamics for robot bicycle

[16] http://en.wikipedia.org/wiki/General-purpose_input/output
[17] http://www.homepower.com/articles/solar-electricity/design-installation/sizing-grid-tied-pv-system-battery-backup
[18] https://learn.adafruit.com/introduction-to-the-beaglebone-black-device-tree/overview