Intelligent Power-Assist Bicycles

Design modification for improved portability

Ogni Goswami

American High School

Fremont, California

Internship Report

Summer 2015

Advisors: Professor M. Tomizuka

Dr. W. Zhang

University of California Berkeley

Department Mechanical Engineering

Mechanical Systems Control Laboratory

Abstract

The majority of commercially available power-assist bicycles utilize proportional motor controllers where the human's effort into the bicycle and the output of the bicycle's assist motor are set at a constant 1:1 ratio. While this is very useful for tasks such as riding up hills, the constant assist ratio is not suitable for adapting to variations in the terrain or to the level of fatigue of the user. To address this, the concept of *intelligent* power-assisted bicycle was envisioned. [1] In this bicycle, the user's heart rate is monitored as an indicator of the input effort and the amount of assist torque generated by the motor varies with this.

While this setup works as expected, there is a practical problem. The user is burdened with the inconvenience of carrying a bulky, expensive laptop in order to run the software that controls the level of assistance. In fact, this is not only just an inconvenience, but is a serious impediment against commercialization attempts. In order to solve this issue, we developed a highly portable system consisting of a compact myRIO platform from National Instruments and a Raspberry Pi mobile computation platform to run the controller instead. This report describes this new system of improved portability.

1. Introduction

The concept of electric bicycles has existed for over a century, [2] yet it has only seen wide-scale adoption during the past two decades, especially in Asia. [3] The electric bicycle possesses several key advantages over conventional bicycles and other modes of transportation. For instance, electric bicycles have zero emissions, which can make them a sustainable "green" solution against climate change. However, many other vehicles such as electric cars and trains also have zero emissions. What sets apart electric bicycles is that they are also naturally hybrid, and they alleviate so called "range anxiety": the fear that an electric transportation will simply run out of battery and have the user stranded. Also, the US is having an obesity "epidemic" [11], and electric bicycles can encourage more physical activity, possibly offering a solution. Various cities such as Los Angeles and New York suffer from extreme traffic congestion, and some, such as San Francisco and Portland, have taken steps to solve this by designating bicycle friendly areas. However a major issue for bicycles is their limited range of reach as they are strongly dependant on the physical abilities of the user. Electric bicycles can significantly increase this range, so they can solve urban congestion as fewer people will need to drive. Other advantages include their relative affordability and portability. In many cases electric bicycles are allowed to be transported inside public transports like trains or buses. Therefore, it is no surprise that the sales of electric bicycles have doubled throughout the past decade. [3]

The feature which sets electric bicycles apart from electric motorcycles is that they are meant to "assist" the user instead of the motors being the sole source of propulsion during use. However, the problem with most commercially available electric bicycles is that they utilize a proportional motor controller. The controller collects data through a torque sensor in order to determine the user's pedaling effort and provides a constant 1:1 assistance, which means that the assisted torque from the motor is equal to the magnitude of input torque from the user. Unfortunately this arrangement causes issues with user comfort, safety, and power efficiency as the human-machine-environment interaction is too "rigid". For instance, an input to output ratio above 1:1 is necessary for comfortable uphill ride or if the user is fatigued. On the other hand, a ratio below 1:1 is necessary for downhill rides or if lower speeds are preferred (such as in crowded areas).

Also, the user's riding habits may change depending on the weather and terrain. In order to satisfy this variety of conditions, the need for intelligent, or adaptive, electric bicycles arise.

To make the system adaptive and responsive to the user's needs, a setup was created using a Giant Suede E electric bicycle [4], Zephyr heart rate monitor [5] (Figure 1), and Sigma speed sensor [6] (Figure 2).

Figure 1: Zephyr Heart Rate Sensor [5]

Figure 2: Sigma Speed Sensor [6]

Figure 3: This is the setup of the intelligent power-assist bicycle. This is a stationary setup as the bicycle is mounted on a dynamometer

The original controller of the bicycle was replaced with a more easily modifiable version and the bicycle was placed on a dynamometer-like device so the conditions (such as terrain and weather) could be constant during the experiments. This entire setup was assembled as in Figure 3 above. The focus of this project was to replace the laptop computer running the assist algorithm with a compact and more portable solution.

Lack of convenience was identified as one of the major factors which prevented this system from being considered for commercialization. For instance, the lack of portability of the system was identified as a problem: To operate, the user must carry a bulky, expensive laptop to run the algorithm, as seen in Figure 4. That not only meant the consumer had to continuously run an entire, unnecessary OS and manually launch the program, but also that they had to track the laptop's battery life since the bicycle would not function without it.

Figure 4: Older system where user must carry an expensive, bulky, power consuming laptop

Figure 5: Proposed system where laptop is replaced with the efficient and compact myRIO platform

In order to solve this issue, a myRIO mobile computation platform from National Instruments, [7] which is designed specifically to run LabVIEW algorithms, was utilized. This would draw power directly from the bicycle's central battery pack, be of small mass, and only need to be switched on or off, as seen in Figure 5 above. The power demand of this system under full load would be 18 watts [7] while the power demand from the average laptop under full load is 91.2 watts. [12]

The myRIO would receive the user's heart rate data from the heart rate sensor through Raspberry Pi mobile computation platform, which streams the information from the Bluetooth module, as the myRIO cannot support Bluetooth drivers. This streaming configuration is illustrated in the schematic in Figure 6. This project focused only on interpreting the heart rate of the user, and the actual execution of motor commands is beyond the scope of this report.

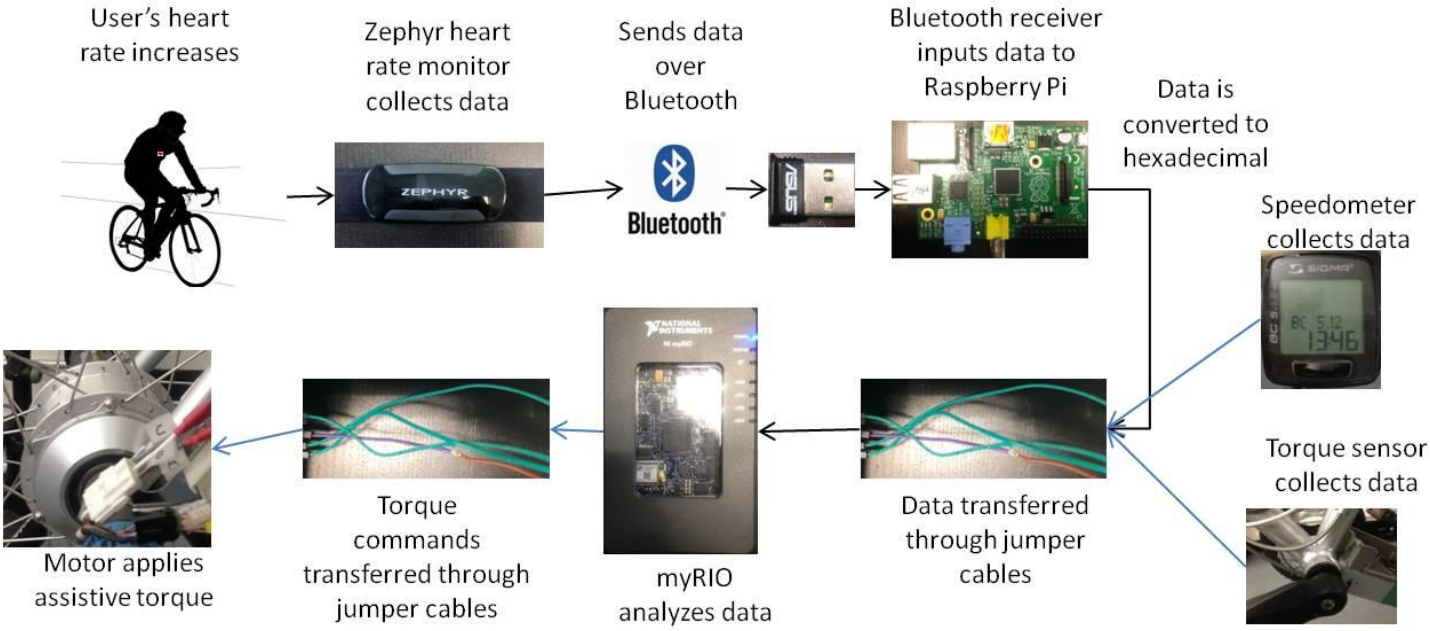


Figure 6: This flowchart represents the concept proposed before. The blue lines are proposed links which are to be made in the future to complete the system.

2. Method

The main component of the solution is to run the LabVIEW algorithm on the myRIO platform instead of on a laptop. The myRIO platform (Figure 7) is developed by National Instruments specifically to run LabVIEW portably. [7] It has various special features such a wide array of GPIO (General Purpose Input/Output) pins, Wi-Fi capabilities, and an FPGA (Field-Programmable Gate Array) board. [7]

Figure 7: The myRIO platform developed by National Instruments

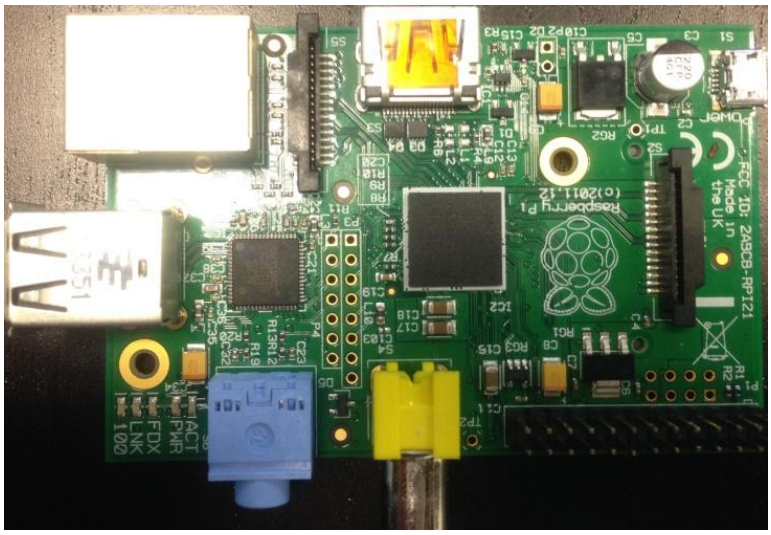


Figure 8: The Raspberry Pi mobile computation platform chosen for streaming to the myRIO

The original plan was to simply plug in the ASUS Bluetooth dongle [14] to the myRIO and upload the LabVIEW algorithm. However, we soon discovered that myRIO could not support Bluetooth drivers. Therefore, a "middle man" which could handle Bluetooth drivers and stream its data was necessary. A Raspberry Pi was chosen for this task as it had the portability of a credit card with

the power of a small computer. The Raspberry Pi is approximately a 9.0 cm x 5.5 cm Linux-based mobile computation platform with a 1 GHz processor and 512 Mb RAM [15] (Figure 8).

The Raspberry Pi's built-in Python shell was utilized to write the program shown in Figure 9 to stream the heart rate data from Bluetooth dongle to the myRIO. As detailed in the comments of the program below, the code reads incoming nix format [13] data from the heart rate sensor, converts it to a hexadecimal string, and then writes the string to the serial port to which the myRIO is connected. A legitimate real-time program would have two simultaneously running loops of reading and writing. However, as the Raspberry Pi was capable of running at the relatively high speed of 1 GHz, the loop in the program above ran quickly enough to *appear* as a real-time stream.

```
#!/usr/bin/env python
import time
import serial
i=0
#setting up infinite loop
while i>-1:
        ser1 = serial.Serial(
        #HR monitor serial port referred
                        port='/dev/rfcomm0',
                        baudrate = 9600,
                        parity=serial.PARITY_NONE,
                        stopbits=serial.STOPBITS_ONE,
                        bytesize=serial.EIGHTBITS,
                        timeout=1
                    )
        counter=0
        x = []
        data = []
        x=ser1.read(60),
        #reads first 120 bytes
        data.append(x),
        #adds data to the list
        ser = serial.Serial(
                #refers to USB serial output to myRIO
                port='/dev/ttyAMA0',
                baudrate = 9600,
                parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE,
                bytesize=serial.EIGHTBITS,
                timeout=1
                )
        counter=0
        for f in data:
                data1 = str(data)
        #converts data to string
        data2=data1.encode("hex"),
        #converts data to hexadecimal tuple
        for f in data2:
                data3= str(data2)
        #converts tuple to string again
        ser.write (data3),
        #writes data to serial port
        print data3
        #empties list for next round
        data = [];
        i += 1
```

Figure 9:This python script was developed to stream the heart rate data to the myRIO for computation

It was necessary to modify the LabVIEW algorithm as well. The original algorithm, which was developed by Xuan Fan, [1] was created for the laptop to run the motors using data from the torque and speed sensors. However, as the focus of this project was only streaming the heart rate through the myRIO, several modifications had to be made to the algorithm. In order to modify it for the myRIO, the VISA Serial data input in the program had to be changed. The original data input was the COM 3 port on the PC, which the algorithm was intended to run on. However, COM ports are limited to Windows and MS-DOS environments. [16] Since the myRIO is Linux based, [17] the data input in the program had to be changed to the UART (Universal Asynchronous Receiver/Transmitter) pins to which the Raspberry Pi was connected. Also, at the very end of the algorithm, a "Reset myRIO" block was necessary to clear the RAM of the myRIO for the next experiment. The circled portions in Figure 10 were modified from the original algorithm.

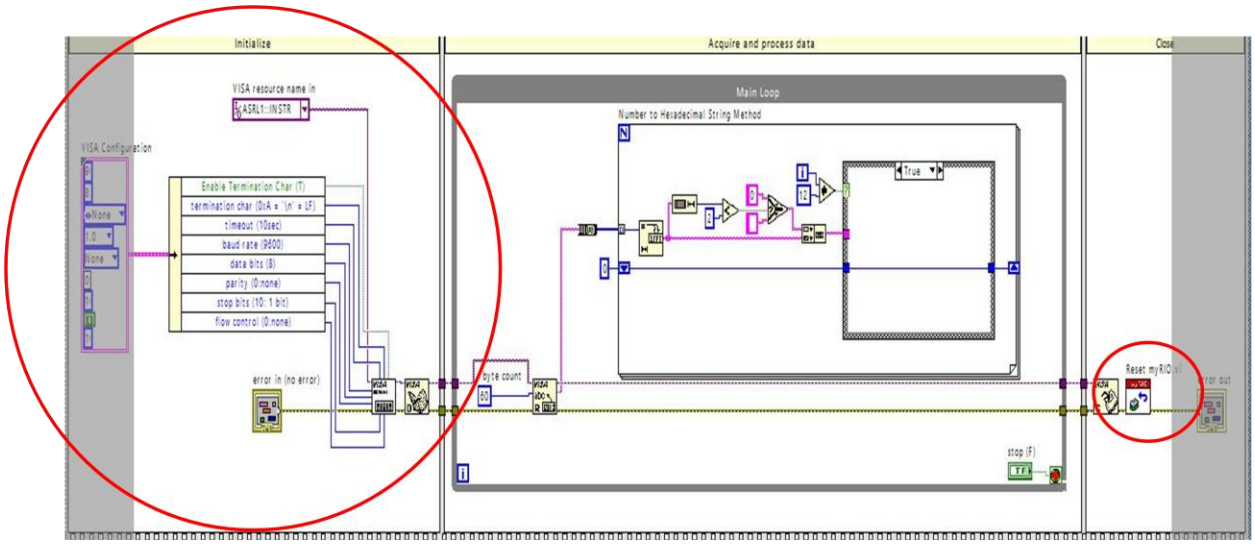
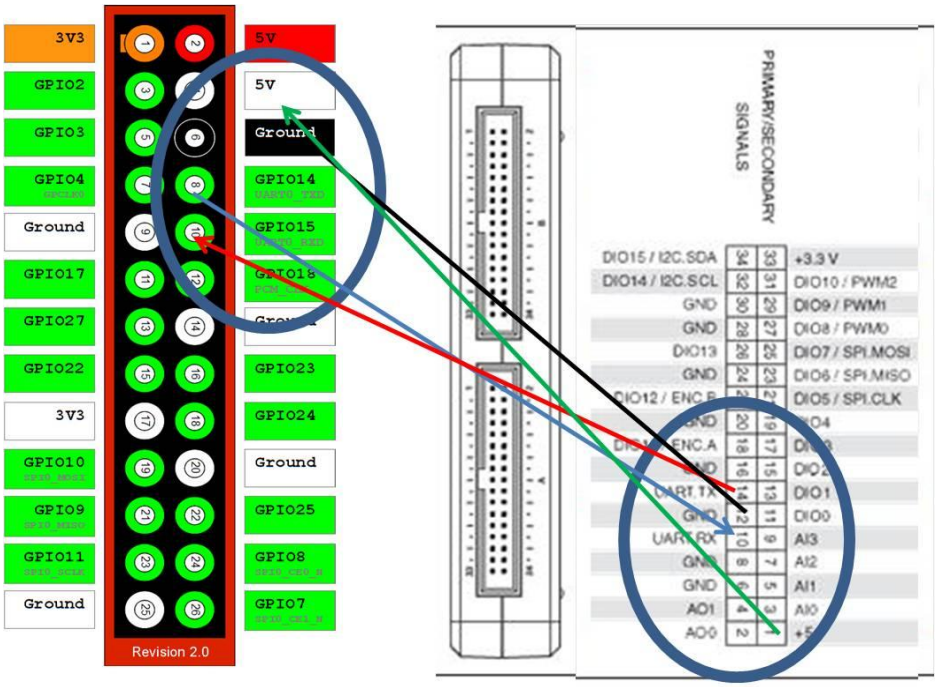


Figure 10: The original LabVIEW algorithm was modified so it could be run on the myRIO, independently of a computer.

From a hardware perspective, serial ports were assigned on the Raspberry Pi to the myRIO and the Bluetooth dongle, /dev/rfcomm0 and /dev/ttyAMA0 respectively. The myRIO was connected to the Raspberry Pi jumper cables between the GPIO pins. The UART RX (receiver) Pin of the myRIO, which is its Pin 10, was connected to the UART TX (transmitter) Pin of the Raspberry Pi, which is Pin 8. The UART TX Pin of the myRIO (Pin 14) was connected to the UART RX Pin of the Raspberry Pi (Pin 10). Also, the two grounding pins of the myRIO and the Raspberry

Pi (Pins 6 for both) were connected. In order to supply power to the Raspberry Pi, a 5 volt line was connected between Pin 1 on the myRIO and Pin 4 on the Raspberry Pi. This setup is shown in the schematic in Figure 11.



Raspberry Pi myRIO

Figure 11 [7] [8]: The pins connected with jumper cables are marked above

Finally, although it is not in the block diagram, a cable between the USB B port of the myRIO and a laptop was used to view the live heart rate of the user, for demonstration purposes. A photo of the entire configuration is shown in Figure 12.

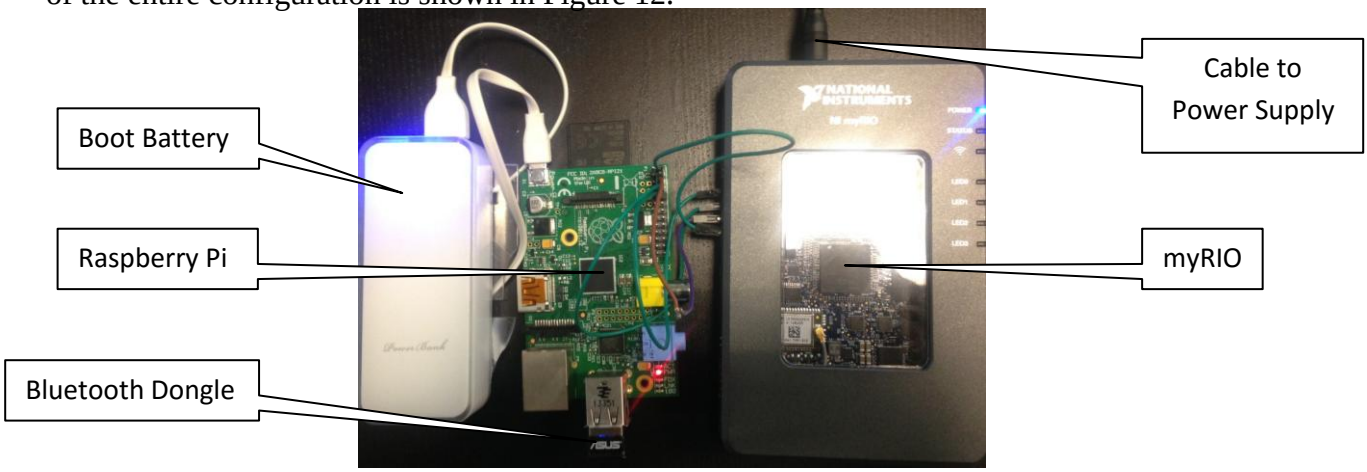


Figure 12: This is a photo of the configuration in Figure 11. The battery is connected to the Raspberry Pi since it is booting.

3. Results and Evaluation

In order to successfully execute the LabVIEW algorithm on the myRIO, the heart rate monitor was first assigned a serial port on the Raspberry Pi using the built-in Bluetooth manager. After the heart rate monitor was assigned the serial port number /dev/rfcomm0, the python script in the Raspberry Pi was executed using the built-in Python Shell. After hexadecimal data from the heart rate sensor started printing on the screen, the LabVIEW program was initiated. In order for the myRIO to function without a computer, the LabVIEW program needs to, at first, be run from a laptop while the myRIO is plugged in. After initiating the program on the myRIO, it began feeding the heart rate sensor into a gauge on the control panel of the LabVIEW algorithm, as seen in Figure 13.

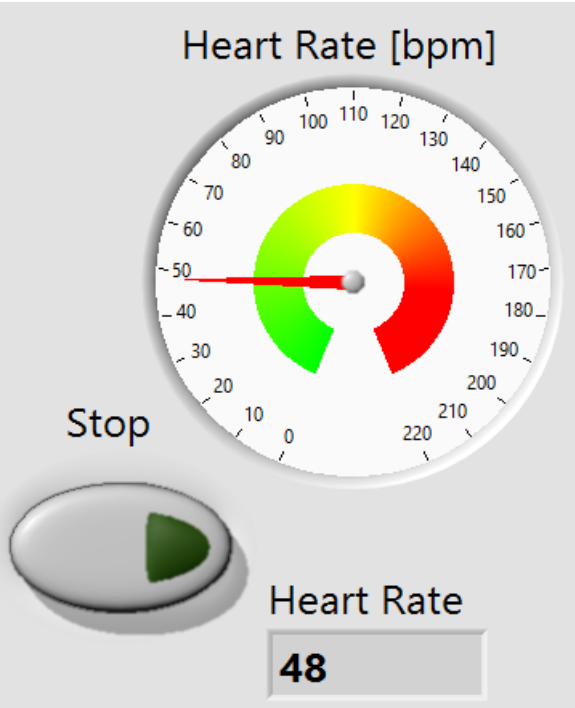


Figure 13: A gauge on the control panel of the LabVIEW algorithm displays the heart rate of the user as computed by the myRIO.

However, soon after the heart rate started showing up on the screen, we realized that the readings were inaccurate. While wearing the heart rate sensor, my displayed heart rate erratically fluctuated between 48 bpm (Beats per Minute) and 99 bpm, while the average heart rate of a

human at rest is approximately 72 bpm. [18] This may be a combination of a faulty heart rate sensor and bugs in the LabVIEW algorithm.

A possible theory is that these incorrect readings are caused by a “displacement” in the data stream. When the heart rate monitor streams data, it puts many “false” characters around the real heart rate value, possibly for verification purposes for the target system. The heart rate is the 60^{th} character of each transmitted data packet and all the values before and after it are for identification purposes, so the LabVIEW algorithm is set to ignore all the irrelevant values and only read that. However, when the Raspberry Pi streams data, it puts parentheses, quotation marks, and brackets around the sets of data because of the way it formats strings and arrays. Since the streamed data is displaced by the three characters (parentheses, bracket, and quotation mark), the algorithm may only be reading the 57^{th} character instead. Therefore, the value being displayed on the computer may not be a heart rate reading at all, but rather one of the verification characters. Either the LabVIEW algorithm or the python script may be modified in the future to resolve this issue.

4. Future Development

1. The ultimate goal of this project was to run LabVIEW on the myRIO platform and enable it to control the level of assistance supplied by the hub motor. For this to be successful, future work may include linking the torque sensor and speed monitor to the Digital I/O pins of the myRIO as inputs, and connecting the motor to another pin for output commands.
2. In an actual, outdoor implementation, the myRIO must use a battery instead of being plugged into a wall. Some research was done on this topic and it was concluded that the myRIO's maximum power intake was 12 volts at 1.5 amperes, or 18 watts, so eight AA batteries in parallel would satisfy its energy requirements (Figure 14).

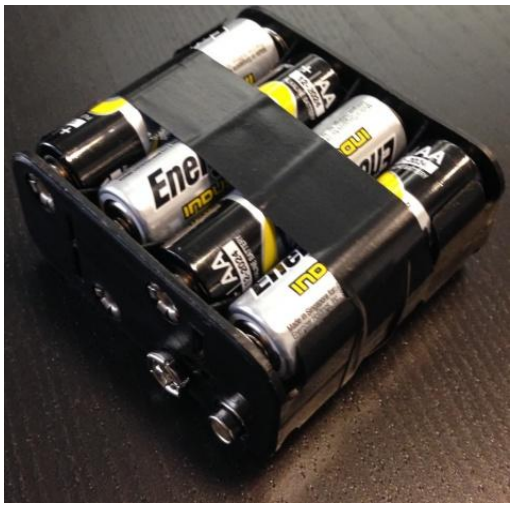


Figure 14: 8 AA Batteries in

parallel are required for the myRIO

3. The Raspberry Pi was able to function independently of a battery as it drew power from the 5 volt jumper cable from the myRIO. However, it required an external power source, such as a battery or wall outlet to boot. This is necessary to satisfy the surge of power drawn during booting. If the Raspberry Pi was booted without an external power source, it would crash as there would not be enough available current to boot. A solution to this which may be explored in the future is the undervoltage setting on the Raspberry Pi. If this is implemented, the user will only have to power on the myRIO for the Raspberry Pi to also automatically boot.

4. A major convenience would be to program the Raspberry Pi to automatically assign a serial port to the heart rate sensor and begin executing the python code. This is called a “headless bootup” as it is done without a keyboard or monitor. If this is implemented, the user will never even have to interact with the Raspberry Pi as they would only have to switch on the myRIO to allow current to flow to the Raspberry Pi to start executing the script.
5. So far, each time the LabVIEW algorithm needs to be run on the myRIO, the program needs to be initiated on a laptop with that algorithm before the myRIO can be unplugged. A modification to the myRIO to allow it to store the LabVIEW program and run it automatically when it boots would be very convenient.
6. Also, for future commercialization a more reliable heart rate monitor should be considered as the Bluetooth connection in the current one often erratically disconnects. A possibility is to attach a conductive heart rate monitor, similar to the type found in many treadmills (Figure 15), to the handlebars of the bicycle. This would eliminate the need for the user to have to strap the heart rate monitor onto their chest and can even eliminate the need for unreliable Bluetooth connections. If this is implemented, the Raspberry Pi would likely no longer be necessary as the myRIO could directly read the data from the heart rate sensor without a middleman. As seen in the modified block diagram in Figure 16, this would be a significant simplification over the current system.

Note: I have several solutions to the propositions above. If necessary, please contact me at ogni@ogniweb.com

Figure 15: [18] An example of the conductive heart rate monitors found in the handlebars of many treadmills. Installing such a monitor inside the bicycle handlebars will allow removal of the more cumbersome chest strap-on monitor

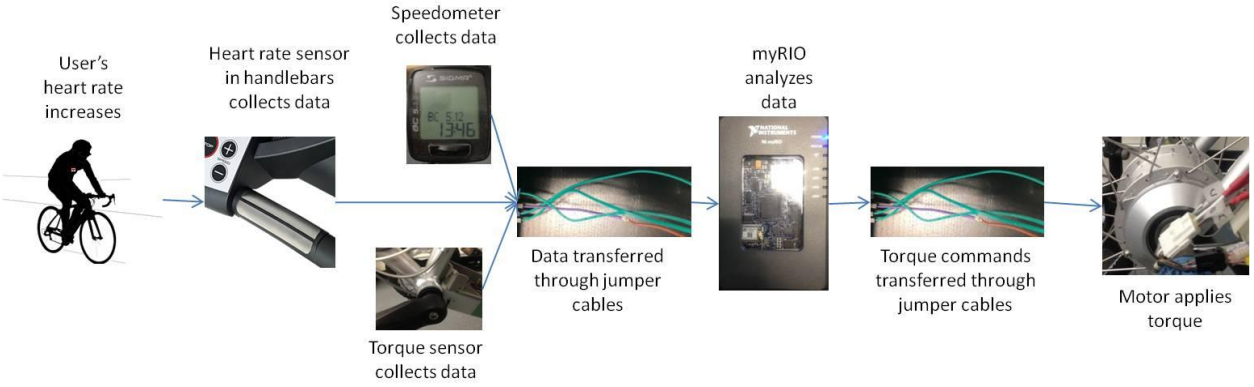


Figure 16: If the current Heart Rate sensor was replaced by one mounted in the handlebars, it would pose significant simplification of the system compared to the one proposed in Figure 6.

Bibliography

1. Fan, X. (2010). Intelligent Power Assist Systems Auto-Adaptive to Varying Human Characteristics and Environmental Conditions -First Year Progress Report. University of California, Berkeley. Retrieved from http://escholarship.org/uc/item/4m56v3j5
2. Bolton, O. Electrical Bicycle. Patent US552271 A. Dec.-Jan. 1895. Print.
3. Fairly, P. (June 1, 2005). China's Cyclists Take Charge. IEEE Spectrum. Retrieved from http://spectrum.ieee.org/transportation/advanced-cars/chinas-cyclists-take-charge
4. Giant Suede E Electric bicycle. http://www.giant-bicycles.com/en-us/bicycles/lifestyle/1272/29755/
5. Zephyr. Zephyr HxM BT HR Monitor. Retrieved from http://zephyranywhere.com/products/hxm-bluetooth-heart-rate-monitor
6. Sigma. Sigma Speed Sensor. Retrieved from http://www.sigmasport.com/us/service_center/zubehoer/bicyclecomputer
7. National Instruments. (2006). User Guide and Specifications: NI myRIO-1900. Retrieved from http://www.ni.com/cms/images/devzone/tut/image6042522872247360281.jpg
8. Wicker, N. (2014). How to Build a Handheld, Raspberry Pi-Powered Game Console. Lifehacker. Retrieved from http://lifehacker.com/how-to-build-a-handheld-raspberry-pi-powered-game-cons-1663675758
9. Maassen, M. (2008). Intelligent Power Assisted Bicycles: design of a motor driver for the experimental setup-Traineeship Report. University of California, Berkeley. Retrieved from http://www.mate.tue.nl/mate/pdfs/9635.pdf
10. Centers for Disease Control and Prevention. (2011). The Obesity Epidemic. Retrieved from http://www.cdc.gov/cdctv/diseaseandconditions/lifestyle/obesity-epidemic.html
11. Glaser, F. (April 10, 2011). Review Dell XPS 15 Notebook. Notebook Check. Retrieved from http://www.notebookcheck.net/Review-Dell-XPS-15-Notebook-i5-2410M-GT-540M.51186.0.html
12. Niemietz, R. C. (October 21, 2003). A proposal for addition of the six Hexadecimal digits (A-F) to Unicode. Retrieved from http://std.dkuug.dk/jtc1/sc2/wg2/docs/n2677
13. ASUS. ASUS Bluetooth USB Adapter. https://www.asus.com/us/Networking/USBBT400/
14. Raspberry Pi. Retrieved from https://www.raspberrypi.org/
15. HOWTO: Specify Serial Ports Larger than COM9. (October 26, 2013). Microsoft Support. Retrieved from https://support.microsoft.com/en-gb/kb/115831#appliesto
16. How Do I Access the Shell on a NI Linux Real-Time OS Device. National Instruments Support. Retrieved from http://digital.ni.com/public.nsf/allkb/9822A3A39B1D0CBB86257C55006B962A
17. Laskowski, E. R. What's a normal resting heart rate. Mayo Clinic. Retrieved from http://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979

18. NordicTrack C900 Pro Treadmill Review. Treadmill Adviser. Retrieved from http://www.treadmilladviser.com/nordictrack-c900-pro.html

Copyright Information

Acknowledgments

I would like to thank Professor M. Tomizuka for allowing me the exceptional opportunity to work at his research lab, Dr. C. Wang for his help with LabVIEW and this report, and Dr. W. Zhang for inspiration of this project. I also gratefully thank the members of the Mechanical Systems Control Laboratory at the University of California Berkeley for creating a friendly atmosphere to work at.