Closed-Loop Feedback System for Controlling Renal Perfusion Pressure via LabVIEW

A Technical Report for BME 4064

Presented to the Faculty of the School of Engineering and Applied Sciences
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Biomedical Engineering

Author

Pai Li
May 01, 2020

Technical Project Team Members
Pai Li

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature _____ Date _____

Approved _____ Date _____
Pin-Lan Li, M.D., Ph.D., Virginia Commonwealth University School of Medicine, Department of Pharmacology and Toxicology

Closed-Loop Feedback System for Controlling Renal Perfusion Pressure via LabVIEW

Pai Li[a,1], Pin-Lan Li, M.D., Ph.D.[b]

[a] Fourth Year Biomedical Engineering Undergraduate at the University of Virginia
[b] Professor of Pharmacology and Toxicology at Virginia Commonwealth University School of Medicine
[1] Correspondence: pl9ed@virginia.edu

Abstract

Renal perfusion pressure (RPP) plays a key role in pressure natriuresis, which in turn plays a key role in the long-term regulation of blood pressure. However, despite its importance in renal research, accessible methods of controlling RPP have been lacking. This project furthers the development of a LabVIEW program for controlling RPP from a proof of concept prototype into a working product. Telemetric RPP readings are sent from the transmitter via Excel to the LabVIEW program, which then infuses or withdraws a syringe attached to an aortic occluder directly upstream of the renal arteries. High RPP readings cause the program to infuse the syringe, inflating the aortic occluder and pinching the aorta, leading to lower downstream blood pressure and ultimately lower RPP. For low RPP readings, the reverse process occurs. While the original prototype was functional, it suffered from a number of design flaws that made troubleshooting and development difficult. The most notable consequence of these design flaws was the fact that the prototype's functionality could not be validated without the use of a live rat for RPP data. The current project improves upon the prototype design by removing bloat and modularizing subcomponents, allowing for hardware-independent unit testing of each individual component. The current iteration's design makes for a much more robust piece of software that is not only more accessible, but also easier to work with, develop, and troubleshoot.

Keywords: renal perfusion pressure, LabVIEW, feedback system

Introduction

Cardiovascular disease (CVD) and cerebrovascular disease (stroke) are two of the top five causes of death in the United States, accounting for roughly one in three and one in twenty deaths in the U.S., respectively. These numbers translate to someone dying from CVD once every 40 seconds and from stroke once every 4 minutes.[1,2] Monetary costs associated with CVD and stroke totaled $329.7 billion in 2018 and are projected to reach upwards of $1.1 trillion by 2035.[3] Hypertension (HBP), typically defined as systolic blood pressures greater than 140 mmHg or diastolic blood pressures greater than 90 mmHg, is a leading risk factor for both CVD and stroke.[4,5] Furthermore, the prevalence of HBP in adults in the U.S. was 34% in 2018.[3] Of these cases, 95-98% were of essential HBP with no clear attributable cause.[6] The current understanding of the pathophysiology of HBP is relatively weak and uncertain, especially considering its colossal presence in American public health.

Seeing as the vast majority of HBP cases are those of essential HBP with no known underlying cause, it is clear that the etiology of HBP is poorly understood.[6] Although HBP is ultimately the result of a number of complex underlying factors, HBP is generally regarded as a secondary product of impaired pressure natriuresis.[7,8] Indeed, pressure natriuresis is abnormal in almost all animal models of HBP.[9] Renal perfusion pressure (RPP) plays a crucial role in the body's regulation of blood pressure via pressure natriuresis and diuresis by way of controlling renal sodium and water output. An increase in arterial blood pressure leads to an increase in renal sodium and water excretion and a decrease renal sodium reabsorption. Since mean arterial pressure is the product of total peripheral resistance and cardiac output, cardiac output is the product of heart rate and stroke volume, and stroke volume is proportional to extracellular fluid volume (ECFV), it follows that blood pressure is proportional to ECFV. Therefore, decreases in ECFV via renal excretion lead to a subsequent decrease in blood pressure, and vice versa. This feedback mechanism for blood pressure control is extremely potent and results in a long-term feedback gain approaching infinity.[10]

In order to properly identify and understand the mechanism by which the vast majority of HBP cases arise, this complicated network of interacting underlying causes needs to be untangled, isolated, and studied. Thus, RPP, which plays an active role in pressure natriuresis via blood flow autoregulation, renin release, and sodium excretion, is a key component in the pathophysiology of HBP.

As a key physiological marker for pressure natriuresis, pathological values of RPP can cause renal dysfunction. However, pathogenesis of chronic hypertensive kidney damage is complex. Damage to renal function can occur either by direct damage to the kidneys (RPP-independent) or by changing RPP such that end-stage renal disease occurs (RPP-dependent). Investigating RPP-dependent factors requires controlling RPP as the experimental variable, whereas investigating RPP-independent factors may require maintaining RPP as a controlled variable. In either case, a reliable method for controlling RPP is necessary in HBP research. In fact, early renal research implementing mechanical servo-control of RPP were what revealed the importance of pressure natriuresis in the first place.[7,11] Furthermore, since the exact role of RPP is not fully understood, the ability to control RPP is a required component to eventually understanding both renal dysfunction and essential HBP.

Programmatic RPP Control

As previously mentioned, systems for programmatically controlling RPP have been developed in the past.[12–15] However, prior systems suffer from a number of issues surrounding implementation. Systems such as those developed by Hester et al. or Nafz et al. typically involved hardware implementations via mechanical servo-control.[12,15] Setting up a closed-loop feedback system on the hardware level requires a feedback circuit that works directly with analog signals from the RPP recorder (a Grass polygraph in the case of Hester et al.) and syringe pump, not to mention any other necessary components such as amplifiers and filters. It requires

specialized knowledge of relatively low-level concepts such as signal and data representation. A software-based feedback system, on the other hand, is much simpler to understand and implement. Most programming languages operate on a higher conceptual level that more closely mirrors human language, thus requiring less specialized knowledge to develop and operate.

To this end, Xia et al. developed a software-based approach to RPP control using LabVIEW.[16] A telemetric transmitter was transplanted into the infrarenal aorta via the femoral artery. This transmitter then fed the RPP data to DataQuest ART Gold Acquisition V3.10 (Data Sciences International), which could dynamically update the RPP readings to an Excel file. The LabVIEW virtual instrument (VI) would then read the input RPP and communicate with a syringe pump (NE-1000, New-Era Pump Systems) attached to an inflatable Silastic vascular occluder (DocXS Biomedical Products) implanted around the aorta above the infrarenal arteries. Thus, when RPP was too high, the VI would infuse the syringe, inflating the occluder, leading to lower downstream blood pressure, therefore lowering RPP. Conversely, when RPP was too low, the VI would withdraw the syringe, deflating the occluder, leading to the reverse effect. This setup produces a typical closed-loop feedback system for controlling RPP (Fig. 1).

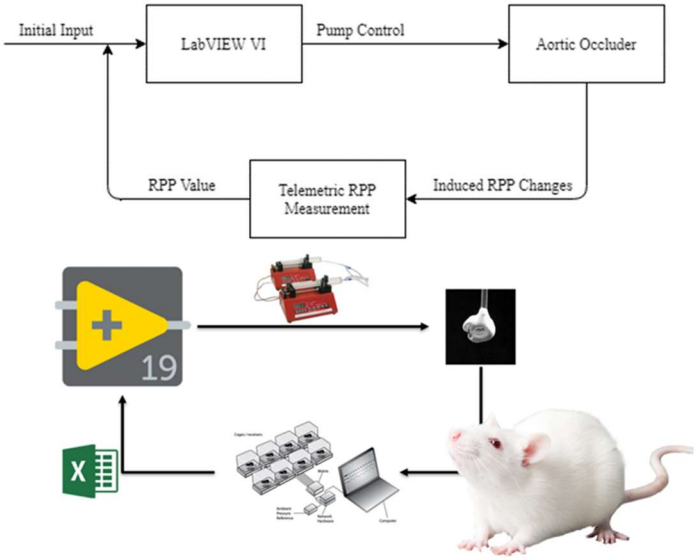

Fig. 1. System Block Diagram. Block diagram along with physical representation

However, their 2008 VI was an unstable proof of concept that suffered from critical crashes and malfunctions. The code contained a number of unnecessary or redundant sections, leading to a lack of readability which made further troubleshooting and development difficult. Their VI also suffered from a number of poor design choices likely stemming from the fact that it was developed as a proof of concept with the goal of demonstrating feasibility. For example, the entire program code was duplicated in order to accommodate a second pump (Fig. 2). This design choice meant any changes need to be made twice, and significantly increases the chances failure due to the interconnected nature of the program. Furthermore, although there were some attempts to organize the VI into components, by and large the VI was one giant block of code. Some of the few sub-VI's that did end up being developed performed multiple functions. Thus, unit testing was impossible, and troubleshooting errors was extremely difficult. A further impediment was the fact that no proper testing environment existed for the VI due to the way in which hardware constraints were integrated into the program. The only way to verify the VI's functionality was to monitor a live rat's RPP and ensure that its RPP stayed within the predetermined threshold values. There was no way to independently validate system decision making behavior or pump behavior. Thus, validation of any changes to the VI required full operational testing. Due to the highly iterative nature of software development, neither of these options are feasible in the long run.

Here, we take Xia et al.'s LabVIEW VI and developed the program from a proof of concept prototype to a working product. In addition, we developed a testing environment in order to verify changes and additions, as well as formal documentation and instructions on GitHub (see End Matter).

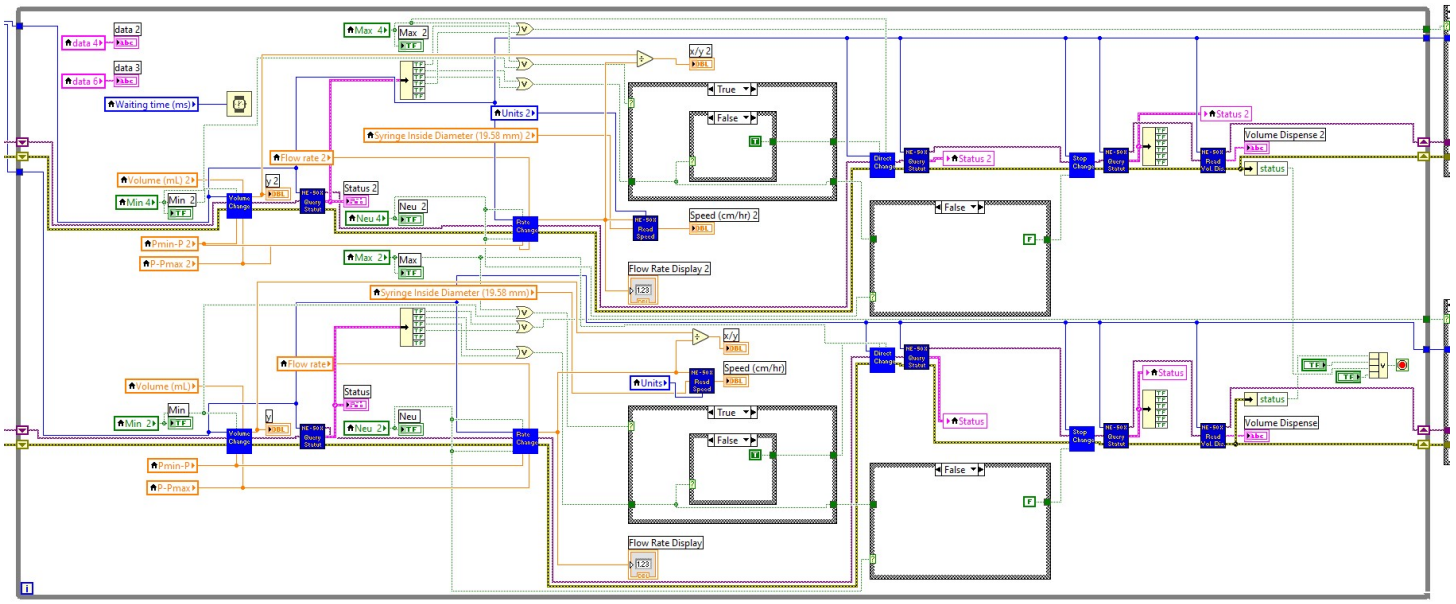

Fig. 2. Sample Main Program Block Diagram from Original VI. The entire block diagram was duplicated in order to accommodate a second pump. Many case structures in this section are unnecessary. Lack of documentation further hampers readability.

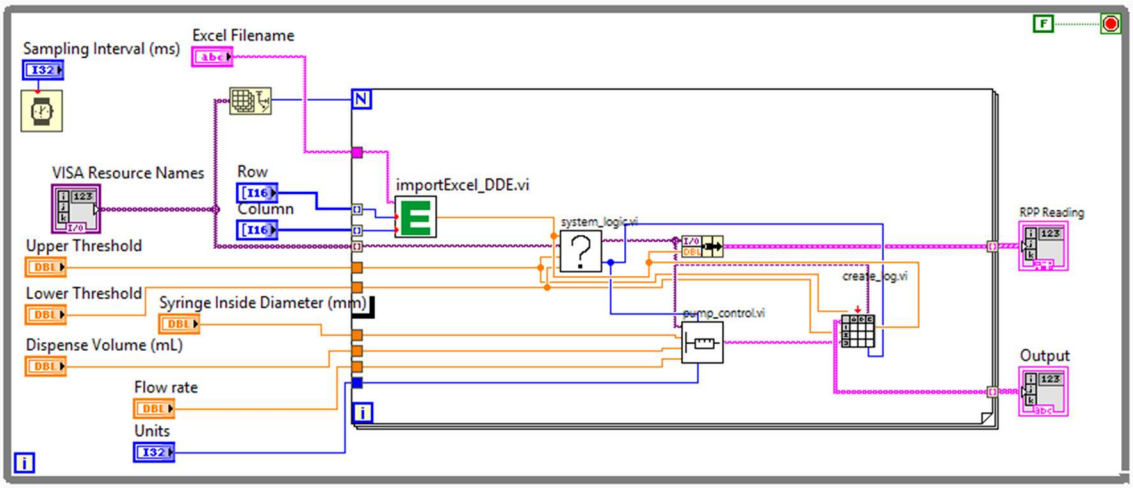

Fig. 3. Updated Main Program Block Diagram. Components of the program are separated out into discrete sub-VI's. An array is used to accommodate an arbitrary number of samples, identified by the VISA Serial Port of the pump associated with the sample.

Program Redesign

The primary goal was to fix the design flaws inherent in the original prototype. Most importantly, the components of the program needed to be broken down into modular components to allow for unit testing. Thus, the project had 3 main goals: redesigning the program framework, designing and developing a testing environment, and creating documentation.

Redesigning Framework

As mentioned previously, the main goal of the redesign was to separate and modularize components, specifically into discrete sub-VI's. In doing so, each component could be independently tested and verified, making troubleshooting and further development much simpler. Due to the lack of readability of the prototype, the program was redesigned with a bottom-up approach, where the basic components of the program were built anew.

The program could be broken down conceptually into 3 main functions: reading in RPP data, making a decision, and controlling the syringe pumps. Thus, sub-VI's were created for each component. Furthermore, additional pumps are added via an array and identified through their VISA Serial Port. As a result, it is necessary to correctly identify which pump corresponds to which sample when initializing the program. The new VI can theoretically accommodate any number of samples with the addition of a pump network address, although the fact that arrays iterate one at a time may cause issues with timing when it comes to extremely large numbers of samples. However, for a typical lab, and for Dr. Li's lab for whom this VI was designed for specifically, an array-based implementation for multiple pumps should not cause issues.

The input sub-VI (importExcel_DDE.vi) uses an OpenDDE implementation to read values from specific cells in an Excel spreadsheet. The choice to use OpenDDE to communicate with Excel rather than more traditional methods such as reading in a delimited spreadsheet was due to the nature of the transmitter software used in Dr. Li's lab. The software could only dynamically export live data to an Excel spreadsheet, necessitating communication with Excel. Protocols involving ActiveX and LabVIEW's Report Generation Toolkit were also explored. However, ActiveX methods depend on the version of Excel in question, meaning the sub-VI would need to be updated alongside each Office or Excel version upgrade. LabVIEW's Report Generation Toolkit, although more robust than OpenDDE, was not available on the computers in the lab. Thus, the lab would need to upgrade LabVIEW should they need to make any changes to the sub-VI. As a result, an OpenDDE-based approach proved best despite it being an older, clunkier implementation that requires the Excel file in question to be open.

The input RPP values were then fed into a logic sub-VI (system_logic.vi) which determines the appropriate course of action based on a straightforward series of conditional cases comparing the input RPP to the predetermined thresholds. The output is an integer corresponding to the cases used by New Era's drivers (Infuse: 0, Withdraw: 1, No Action: 2, Error: 3). This integer is fed into a pump control module (pump_control.vi), which writes the appropriate commands to a NE-1000 syringe pump.

The pump control sub-VI uses a number of drivers and sub-VI's from New Era's LabVIEW library. The general process of pump control starts with initialization steps: initialize pump, read firmware, turn off power failure mode, turn off safe mode, and clear memory. This is followed by setting specific parameters: syringe diameter, flow rate, target volume, and direction. The direction parameter is what is fed from system_logic.vi. The program then infuses or withdraws the syringe until the dispensed volume reaches the target volume. The volume dispensed is read from the syringe pump itself, hence the clear memory step during initialization. Once this step is complete, the pump is reset.

Values from every stage of the process is fed into a logging sub-VI (create_log.vi) which keeps track of the parameters involved and writes the data into separate csv files for each sample identified by the VISA Resource Name, along with the date and time. If a log file already exists, it appends the information to the end. This log file can then be used for both validation and for troubleshooting down the line.

Creating a Testing Environment

Validation of each component was done via unit testing during development. Each sub-VI can be run as an independent program to check if the actual outputs match the expected outputs for any given input. However, in order to test the functionality of the program as a whole, a method of generating spoofed data was necessary.

To this end, an independent Excel spoofing program (spoofExcel.vi) was developed that writes values to a specified cell(s) in an Excel spreadsheet. This program mimics the behavior of DataQuest's transmitter software, which updates live RPP readings to a specified cell in Excel. It can either generate random RPP values, or values based on an input csv file. This program can then be run in conjunction with the main VI to test holistic program behavior.

Creating Documentation

One unusual design constraint is the fact that the program will primarily be worked on and used by VCU School of Medicine students and faculty with potentially no background in software development or computer science. As a result, the program needs to be readable enough and accessible enough such that any needed changes or adjustments to the program itself can be done by the layperson. To help combat this issue, documentation for the program detailing was created on GitHub. The behavior, inputs, outputs, and block diagrams of each VI and sub-VI were listed in simple language along with a glossary with general computer science terms. In addition, comments were provided throughout the block diagram itself detailing the function and behavior of more complicated components.

Testing and Validation

In order to validate the program's functionality, it was run alongside the Excel spoofing program for about an hour with a sampling interval of 5 seconds, once with random RPP values and once with RPP values from an input sine wave. The random value testing resulted in 595 decisions, and the sine wave RPP values resulted in 568 decisions. The decisions made by the program were validated against the logged RPP and threshold values (Fig. 4a,b). Pass/Fail criteria was based on whether or not the program ever made an incorrect decision based on the input RPP value and thresholds. This process was done programmatically using an R script, which itself was validated by intentionally using an older, buggy iteration of the RPP control program (Fig. 4c). The pre-validation data was also then confirmed manually to ensure the R script could properly identify incorrect decisions made by the program. The current iteration of the program passed both test cases.

Discussion

The current iteration of the program represents a massive improvement over the original proof of design prototype. Modularization of the subcomponents of the program allows for much easier troubleshooting and better readability. Furthermore, by developing with a bottom-up approach, all of the added bloat from the original program has been removed. Most importantly, the program has been developed to a point where testing can be done without the need for live rats for data.

However, despite these improvements, there are a number of lingering concerns regarding the current state of the program. Due to the COVID-19 outbreak, operational testing in the lab was not possible. The first priority moving forward is to test the program in its operational environment. The computers used in Dr. Li's lab run on much older hardware and software, and while the current iteration was developed with backwards compatibility in mind, validation is still needed. Furthermore, while data flow is easier to follow in a visual language, logical structures often are not. Case structures, especially, tend to be messier when compared to traditional high-level programming languages that loosely follow the English language.

As a piece of licensed proprietary software, LabVIEW also has issues of access due to the way it is managed by National Instruments. Moving forward, it may be better to further refactor the program for development in an open source language such as Python. The benefits of open source software are pretty well known at this point, and switching the program over to a common open source language will allow for much greater access. It would also make it easier for others to improve upon the program and make adjustments to suit their individual needs. Not every lab is going to be using NE-1000 syringe pumps, after all.

Nonetheless, the current iteration of the program offers a simple, accessible way to control for RPP for use in renal research. Due to its modular nature, it is easy not only to adapt for other uses, but also to test

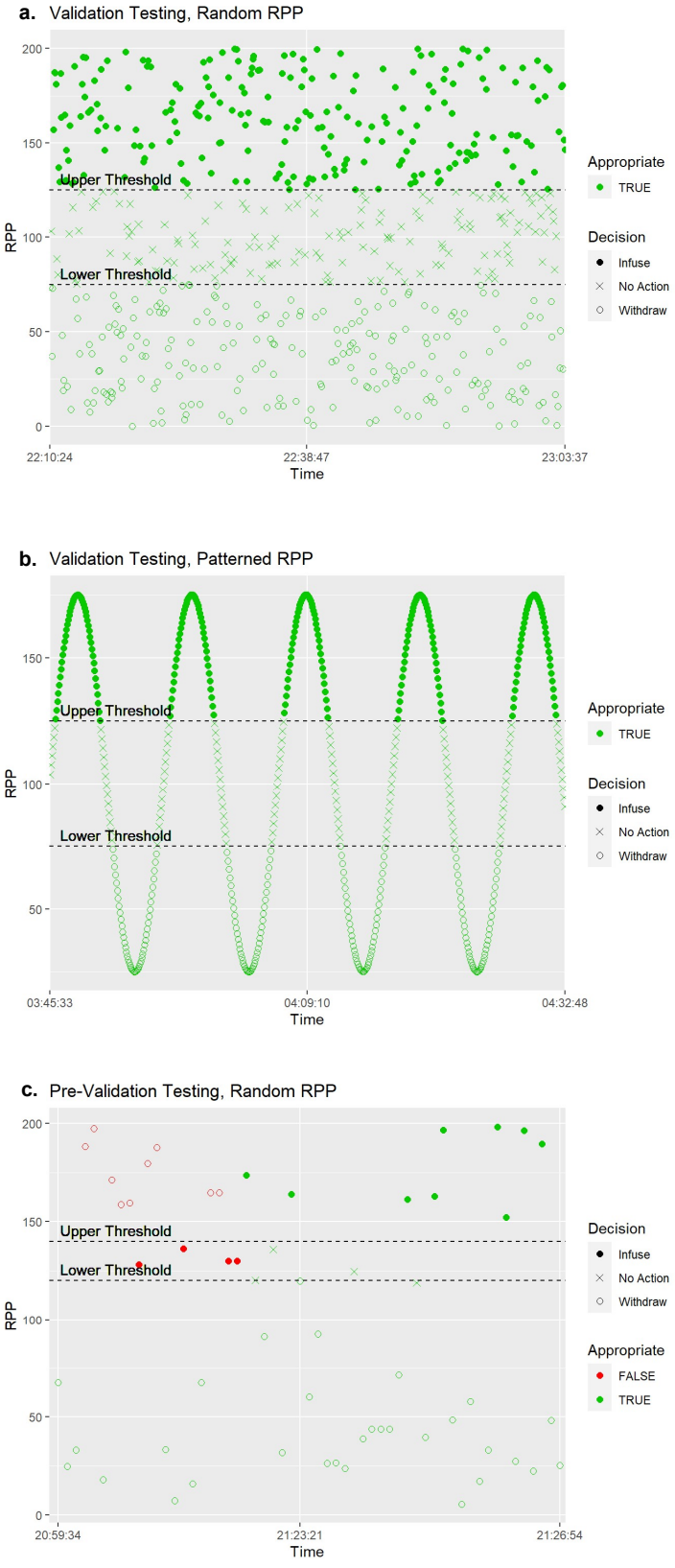

Fig. 4. Validation Testing Data. (a) validation data for randomly generated RPP values (b) validation data for sine wave RPP values (c) pre-validation using an intentionally buggy program

and validate changes made. Because each individual component can operate as a standalone VI, troubleshooting when things do go wrong is also much simpler. The current iteration retains prior functionality while also being a more robust piece of software.

End Matter

Author Contributions and Notes

P.L. designed and performed research, wrote software, analyzed data, and wrote the paper.
The author declares no conflict of interest.
This article contains supporting information online at:
https://github.com/pl9ed/RPP-Control

Acknowledgments

I would like to show my appreciation to Dr. Pin-Lan Li for giving me the chance to work on the program, as well as Min Xia for the program's original original development.

References

1. Johnson, N. B. et al. CDC National Health Report: leading causes of morbidity and mortality and associated behavioral risk and protective factors--United States, 2005-2013. MMWR Suppl. 63, 3–27 (2014).
2. Mozaffarian Dariush et al. Heart Disease and Stroke Statistics—2015 Update. Circulation 131, e29–e322 (2015).
3. Benjamin, E. J. et al. Heart Disease and Stroke Statistics-2018 Update: A Report From the American Heart Association. Circulation 137, e67–e492 (2018).
4. Fryar, C. D., Ostchega, Y., Hales, C. M., Zhang, G. & Kruszon-Moran, D. Hypertension Prevalence and Control Among Adults: United States, 2015-2016. NCHS Data Brief 1–8 (2017).
5. Merai, R. CDC Grand Rounds: A Public Health Approach to Detect and Control Hypertension. MMWR Morb. Mortal. Wkly. Rep. 65, (2016).
6. Beevers, G., Lip, G. Y. H. & O'Brien, E. The pathophysiology of hypertension. BMJ 322, 912–916 (2001).
7. Hall John E. The Kidney, Hypertension, and Obesity. Hypertension 41, 625–633 (2003).
8. Guyton, A. C. & Coleman, T. G. Quantitative analysis of the pathophysiology of hypertension. Circ. Res. 24, 1–19 (1969).
9. Ivy, J. R. & Bailey, M. A. Pressure natriuresis and the renal control of arterial blood pressure. J. Physiol. 592, 3955–3967 (2014).
10. Hall, J. E. & Guyton, A. C. Textbook of Medical Physiology. (Saunders, 2006).
11. Hall, J. E. et al. Mechanisms of escape from sodium retention during angiotensin II hypertension. Am. J. Physiol. 246, F627-634 (1984).
12. Hester, R. L., Granger, J. P., Williams, J. & Hall, J. E. Acute and chronic servo-control of renal perfusion pressure. Am. J. Physiol. 244, F455-460 (1983).
13. Mori, T. & Cowley, A. W. Role of pressure in angiotensin II-induced renal injury: chronic servo-control of renal perfusion pressure in rats. Hypertens. Dallas Tex 1979 43, 752–759 (2004).
14. Woods, L. L., Mizelle, H. L. & Hall, J. E. Autoregulation of renal blood flow and glomerular filtration rate in the pregnant rabbit. Am. J. Physiol. 252, R69-72 (1987).
15. Nafz, B., Persson, P. B., Ehmke, H. & Kirchheim, H. R. A servo-control system for open- and closed-loop blood pressure regulation. Am. J. Physiol. 262, F320-325 (1992).
16. Xia, M., Li, P.-L. & Li, N. Telemetric signal-driven servocontrol of renal perfusion pressure in acute and chronic rat experiments. Am. J. Physiol. - Regul. Integr. Comp. Physiol. 295, R1494–R1501 (2008).