OPEN-SOURCE HARDWARE IN CONTROLS EDUCATION

M. A. Hopkins and A. M. Kibbe
Electrical and Microelectronic Engineering Department
Rochester Institute of Technology

Abstract

In teaching undergraduate automatic controls, the laboratory experience is an important and irreplaceable component. Historically, good platforms for a controls laboratory have been expensive, because the equipment has typically been very specialized for educational purposes. Moreover, the equipment often is not physically robust in the face of student manhandling, creating major difficulties and costs in maintaining such a lab. The advent of inexpensive open-source controller hardware is revolutionizing this situation because it is now possible to have good controls-hardware capability at relatively low cost. The Arduino Mega 2560, in particular, is supported by Matlab, Simulink and LabView, and thus provides a great deal of flexibility in developing laboratory procedures for students to study controls.

The Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. The "motor shield" is an add-on (daughterboard) to the Arduino that further enables control of dc motors. This paper explains a method and hardware to connect an Arduino to a separately-powered dc-motor unit. Matlab and Simulink provide full support of the Arduino board for feedback-controller design. The Arduino board is powered by, and communicates with Simulink, through a standard USB connection.

Keywords — computers in education, controls education, controls laboratory, laboratory hardware, Arduino board, dc motor, motor control, motor control hardware.

Introduction

Educational hardware for electronics laboratories is generally very expensive, in part because the customer base, mostly schools and universities, is not large. Due to the relatively low number of total units produced, such hardware is almost never as refined or bug-free as a full-up commercial product, and in our experience independent third-party support is often not available. Using the Arduino board as a laboratory platform avoids all these problems.

The Arduino is a widely used, open-source platform for hobbyists and tinkerers, available at low-cost from multiple suppliers. In fact, the availability and low cost are such that many students could acquire their own boards, if necessary. The Arduino is designed to be user-friendly, and requires only basic electronics and programming knowledge to use. Engineering students in their second or third year should be capable of quickly learning how to use the Arduino, and how to interface it with other hardware components with relative ease. This is particularly true when the Arduino is programmed with the third-party support now available from The Mathworks (Matlab & Simulink) and National Instruments (LabView). For example, with Simulink the low-level Arduino programming details are "under the hood," transparent to the user, who does all the design work using Simulink's graphical programming.

The Arduino already has been used successfully in classroom and laboratory settings to teach various topics [1, 3], including photovoltaic cells [2] and C-programming [8], demonstrating the versatility and robustness of this platform. This article describes how the Arduino board is being used successfully at Rochester Institute of Technology, in the

Electrical Engineering program (RIT-EE) to implement dc-motor control in laboratories related to control signals, feedback loops, and transfer functions.

Hardware

The *Arduino Mega 2560* was selected as the Arduino board of choice, as it is among the more powerful of the Arduino models yet still maintains a reasonable $50 price tag, and is one of the boards that have been fully supported in Matlab, Simulink, and LabView for about two years. This board has 54 Digital I/O pins, 15 of which can double as pulsewidth modulation (PWM) pins, 16 analog inputs, and serial-communication capabilities via USB cable.

Arduino board input/output ports are capable of operating only in the *positive* voltage range. All signals produced by the board or any of its standard shields (daughterboards) are positive and input voltages below 0V are clipped and interpreted as 0V. Additionally, the analog inputs are not capable of reading voltages above a reference voltage (nominally +5V, but this can be set lower). Voltages above the reference level are similarly clipped. These limitations are addressed by the hardware-interface circuit discussed below.

The *Arduino Motor-Shield* is an inexpensive $25 add-on that plugs in to the Arduino. It contains various outputs for servos as well as two PWM driven analog outputs. This shield is capable of deriving its power solely from the Arduino motherboard, and thus does not require a separate supply (except for high-power applications).

The Arduino Mega/Motor-Shield combination can be powered entirely through the USB cable. In that case, the voltage output of the PWM signals have a maximum of 4 V. When powered by a separate, independent power supply, the Arduino voltage-range increases to approximately 0.5 volts below the supply voltage. This paper assumes Arduino is powered by USB only.

A wide variety of controls experiments can be designed around an independently-powered dc-motor unit. There are many such units available commercially, with the common feature of built-in electronics to power the dc-motor. Thus, very little power is required from the external control-hardware, in this case the Arduino/Motor Shield.

In discussing the dc-motor module, we assume it is a platform that includes a power amplifier for the control input, and an output voltage that is proportional to the motor velocity (*i.e.*, a tachometer). In this paper, both of these signals are assumed to be in the range of -5V to +5V. These two signals are sufficient to design velocity-control experiments. If a dc-motor unit also has an output signal that is proportional to motor-shaft position, then position-control experiments can be designed for the unit, too.

The motor module that is used at RIT-EE is the *MS15 DC Motor Control Module*, marketed by LJ Create. In this paper, we refer to this module as the *LJE motor board*.

Due to input/output mismatch between the Arduino and a typical motor module, a custom signal-conditioner shown in Figure 1 was designed and implemented (the full schematic is given in Appendix A, Figure A.1). The Arduino is indeed capable of driving an attached motor in either direction, but it can only accomplish this by switching a pulsewidth modulated (PWM) control signal between two different output pins when the control signal changes sign. The custom signal-conditioner described here uses just one of these PWM signals (the positive one) to output a voltage between -5V and +5V.

The signal-conditioner converts the PWM signal produced by the motor shield from square wave to a -5V to +5V DC signal. The conditioner uses a lowpass filter and a tunable voltage divider as inputs to a differential amplifier. Thus, a PWM signal at 50% duty cycle is the 'zero point,' and the conditioner outputs a voltage of zero. Duty cycles above 50% produce positive voltages, and those below 50% produce negative voltages.

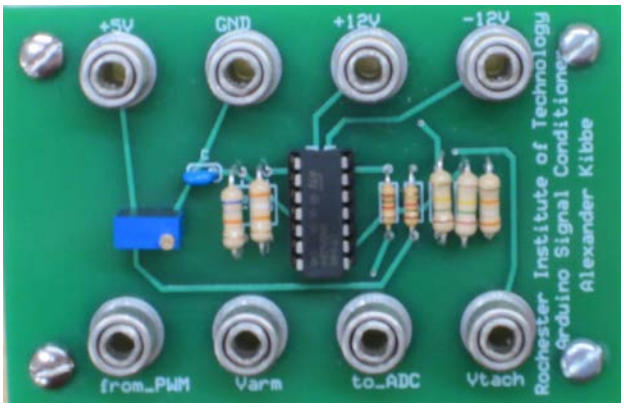

Figure 1. Low-cost custom signal-conditioner to interface Arduino and motor unit.

The signal-conditioner also contains a voltage-averaging circuit. This averaging circuit is needed because the tachometer (motor velocity) output voltage is between -5V and +5V, but the Arduino analog-to-digital converter (ADC) is capable of converting only voltages between 0V and +5V. The signal conditioner circuit shifts and scales the tachometer signal so it is between 0V and +5V. In effect, it averages the tachometer signal with a constant +5V, resulting in a voltage that is in the required range for the Arduino's ADC.

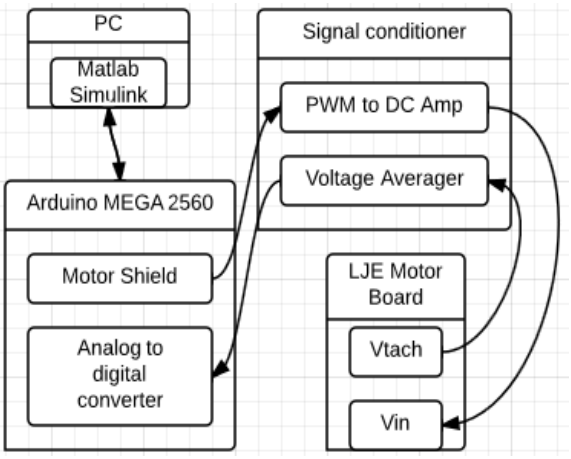

Figure 2. Block diagram of system components and interconnections.

The PWM drive signal produced by the Arduino is filtered and applied to a differential amplifier. The filter, shown in Figure 3, is first-order lowpass with a time constant of 22.5 ms, tailored to the fundamental frequency of the Arduino PWM signal. The PWM signal produced by the Arduino has a frequency of 490 Hz, and thus has a period of about 2.0 ms. Because the time constant of the lowpass filter is much longer than the period of the PWM signal, it effectively smooths the square-wave PWM signal. The filtered signal is buffered to reduce effects due to loading by the rest of the circuit.

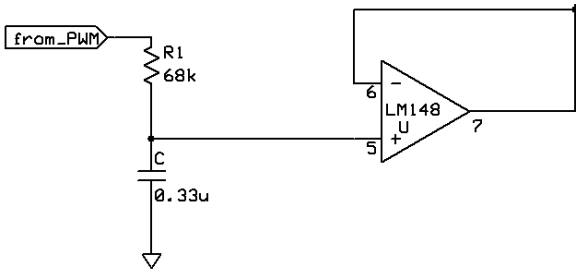

Figure 3. Lowpass filter with buffer.

The difference amplifier in Figure 4 subtracts the voltage applied to R2 from the filtered PWM signal applied to R3, and amplifies their difference by its gain (for the Arduino, the gain is around 3.8). The result is that the Arduino PWM signal, which typically has a range of about +1V to +4V, is smoothed and translated into the range -5V to +5V, so it is ready to be applied to the motor module input (*i.e.*, the armature voltage).

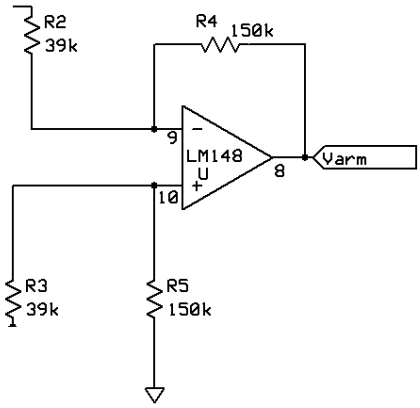

Figure 4. Difference amp / voltage subtractor.

The other input (R2) to the difference amplifier is fed by the buffered voltage-divider of Figure 5. The fact that this divider has a tunable

potentiometer allows the signal-conditioner to service any PWM signal whose voltage is between 0V and 5V. As such, the conditioner should also be compatible with non-Arduino PWM signal sources. It has served well with Arduino boards in actual laboratory use, even though there is some inherent variation in PWM signals from board to board. Figure 6 shows the input/output signals for three different duty cycles.

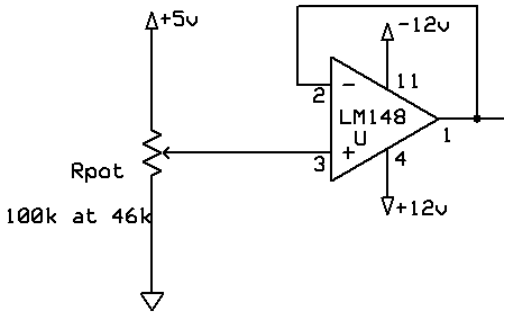

Figure 5. Voltage divider with buffer.

In the three plots of Figure 6, the pulsewidth modulation (PWM) input is lowpass-filtered and level-shifted to produce the output. These plots demonstrate the dependence of signal-conditioner output on the PWM duty cycle, and the fact that the output range includes negative voltages. As PWM duty cycle increases, so does the output voltage, and when PWM duty cycle is below 50%, the output is negative.

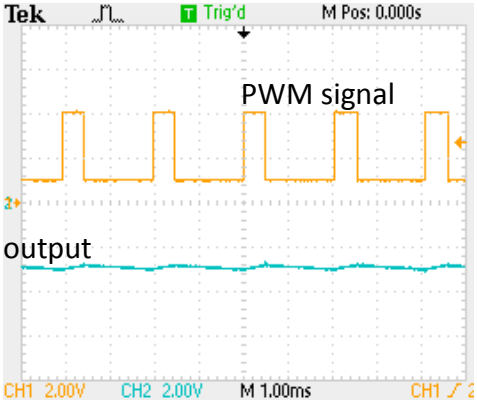

Figure 6a. 25% duty cycle**:** output is approximately -2.5 volts.

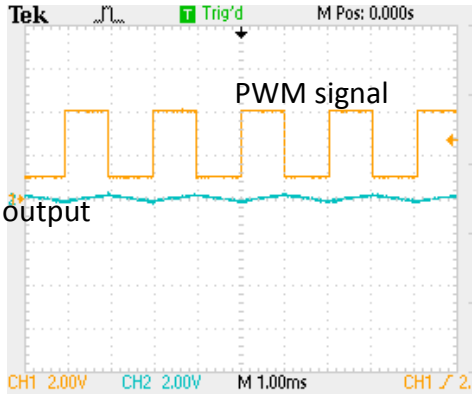

Fig. 6b. 50% duty cycle**:** output is approximately zero volts.

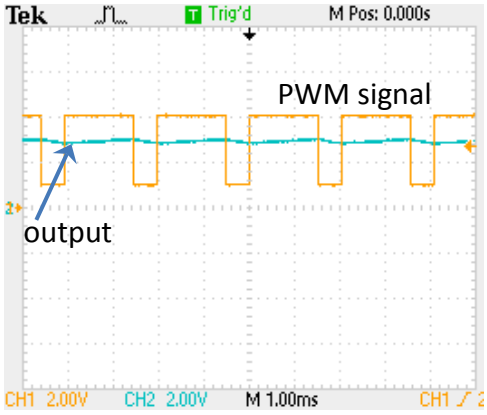

Fig. 6c. 75% duty cycle**:** output is approximately +2.5 volts.

Figure 6. PWM of various duty cycles, and resulting signal-conditioner output.

Simulink Control

To achieve closed-loop control, a controller design is created in Simulink and downloaded to the Arduino, to be executed. The example controller, shown in Appendix A Figure A.2, uses a feedback loop and a second-order PID transfer-function to generate the PWM output of the Arduino. Input and output signals require conversion between floating-point and integer values. The ADC data are unsigned integers from 0 to 1023 (10-bit ADC), which must be converted to floating-point values between -5V and +5V. Similarly, the controller output must be converted from a floating-point value,

between -5V and +5V, to an unsigned integer between 0 and 255.

The controller design in Simulink can be done either in the continuous-time domain, or in the discrete-time domain. If the design is in continuous-time, as it is in Figure A.2, the required conversion to discrete-time (for implementation in the Arduino) is not obvious to students during the download process. That makes continuous-to-discrete conversion an ***optional*** lesson in the curriculum of the controls course for which the laboratory is being done.

The hardware closed-loop test results of Figure 7 show that steady-state tachometer output voltage (labeled "output") almost exactly matches the square-wave reference input (labeled "command"), as it should under PID control.

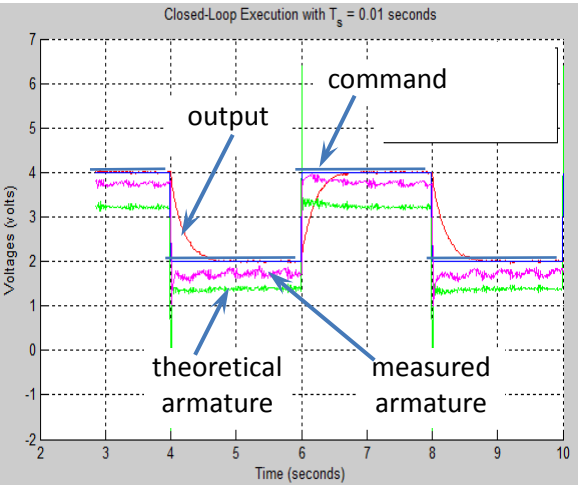

Figure 7. Hardware closed-loop test results.

The difference between the theoretical and the measured armature voltages, is due to a slight mismodeling of the dc motor, as well as significant nonlinearities in the motor unit drive mechanisms. Despite that, the integrating nature of the controller is able to compensate so that the measured tachometer output closely matches the command input, in steady-state. That is, the closed-loop system has dc gain that is very close to unity.

Related Products

Several other products exist that provide similar functions, however none of them completely suited our needs, due either to functional limitations, or to high price. Here are some examples of other commercial products that can be used in controls education.

The *Sparkfun Inventors Kit* [6] ($100) is a very good product that uses an Arduino board to interface a computer to various electronic components. The Arduino UNO can be controlled via Matlab/Simulink in the same way that the Arduino Mega is supported. However, the kit does not include a tachometer, so closed-loop velocity-control is not easily implemented. Additionally, the included motor is driven simply by the on-board PWM without lowpass filtering. In effect, the motor itself serves as the low pass filter for the PWM signal.

The *ScienceWiz Inventions Kit* [7] ($20) is a fairly simple set of do-it-yourself experiments, on a range of topics from motor control to radios. Like the previous one, this kit lacks tachometer feedback. While it does allow the dc-motor control to be configured as a generator, there is only one motor included, so tachometer feedback for velocity control is not possible. Connectivity to a computer for data gathering purposes is not supported.

The *QNET DC Motor Control Trainer* [5] from Quanser is a full-featured set of software, lab procedures, and equipment, including a dc-motor, that enables students to study controls through the NI-Elvis platform and LabView software. Due to the inclusion of written lab procedures, prices are difficult to compare but it is quite expensive.

Feedback, Inc., markets the *Servo Fundamentals Trainer* [4] which is capable of, among other things, acting as a motor speed controller. This is a full-featured product, in terms of controls and physics experimentation. However, the size and high cost are probably

only justifiable if this system is used to support a much wider range of courses than controls alone.

The Feedback, Inc., and Quanser products highlight the main appeal of the Arduino solution: its low cost. Another, less obvious but very appealing, aspect is that many students are eager to learn more about the Arduino board, and their familiarity with it can serve them well in later courses and projects.

Conclusion

The Arduino hardware, interface board, and Simulink design support described in this paper have been used successfully at RIT-EE for the past three academic terms, and again in the current term. Student evaluations of their experiences with the new hardware have generally been much better than those with the previous generation of more expensive, specialty hardware. The old hardware was less reliable. It was also much less accessible, being hidden inside a desktop computer (plugged into a PCI bus), and with two 50-pin ribbon cables for the rather balky interconnections to the dc motor unit.

Serious efforts have been made to "robustify" the use of this hardware in the lab, so that students are much less likely to accidentally burn out circuitry. For example, the only power connection for the Arduino board is the USB connection, itself. These efforts have paid off in terms of a better experience for the students, as well as significant reductions in the work and expense required to maintain the lab equipment.

Problems can arise in the lab when students make a wrong connection from the power supply to the interface board. That can damage the interface board. When that has happened, the interface boards have been easily repaired by a chip replacement (cost $0.45). No Arduino boards have failed, and none have been damaged. We believe this is because it is powered from the USB connection, so students don't have much opportunity to connect improper voltages to it. The Simulink interface to Arduino has worked flawlessly, so far.

Over the past 18 months, around 200 students, usually working in teams of two, have used this new hardware in lab. Students have had, in general, fewer technical problems with the Arduino hardware than they had with the previously-used specialty hardware. We attribute this largely to the robustness of the Arduino platform, as well as to the fact that this hardware implementation is much more obvious to students than the one previously used, which had a PCI plug-in board buried inside a desktop computer, and a very awkward electrical interface (two 50-pin ribbon cables). Another factor may be their ability to run the lab experiments from their personal laptop computers, if they have installed the student version of Matlab. Many of them have done that.

Feedback about the lab from students, including official course evaluations, has been almost uniformly positive, which we attribute largely to the fact that many students consider the Arduino to be something like state-of-the-art, want to know more about it, and feel that this lab gives them good exposure to it. Another major factor in student satisfaction is that on occasions when technical problems do arise, they are fairly easy to understand and to fix. Several students have gone on to use the Arduino in other projects, such as Senior Capstone projects.

References

1. D. Wilcher, "Physical Computing and DC Motor Control" in *Learn Electronics with Arduino* Apress, 2012, Ch 4. pp. 69-87

2. K Zachariadou *et.al.*, "A low-cost computer-controlled Arduino-based educational laboratory system for teaching the fundamentals of photovoltaic cells", *Eur. J. Phys.* 2012 **33** 1599

3. B. M. Hoffer, "Satisfying STEM education using the Arduino microporcessor in C programming", MS Thesis, East Tennessee State University, Publication Number: AAT 1520533, 2012.

4. Feedback Inc., Servo Fundamentals Trainer [online]: http://www.feedback-instruments.com/products/education/terms_and_conditions/servo_fundamentals_trainer

5. *Quanser QNET DC Motor Control Trainer* [online]: http://www.quanser.com/products/qnet_dcmct

6. *Sparkfun Inventor's Kit* [online]: https://www.sparkfun.com/products/12001

7. *ScienceWiz Inventions Kit* [online]: http://sciencewiz.com/products/science_Books_Kits_Inventions.php

8. J. Sarik and I. Kymissis, "Lab Kits Using the Arduino Prototyping Platform", *40th ASEE/IEEE Frontiers in Education Conference*, October 2010, Washington DC, pp.T3C1-T3C5.

Biographical Information

Mark Hopkins has been teaching electrical engineering at RIT in the controls area for over 25 years and has extensive related experience in the aerospace industry.

Alexander Michael Kibbe is an Electrical Engineering graduate of the Rochester Institute of Technology, Bachelors degree.

Appendix A. The Arduino-to-Motor Interface Board

Figure A.1 shows the circuit diagram of the interface board that goes between the Arduino board and the dc motor. The interface board is pictured in Figure 1 of the main text.

Figure A.2 shows a PID controller that was designed for the particular LJE motor board used in the lab, which has a dominant mechanical time-constant of about 0.3 seconds. The PID controller transfer function is

$$H(s) = \frac{2.5(s+5)(s+60)}{s\,(s+150)}$$

The controller is automatically converted by Simulink to discrete-time at a sample rate of 100 Hz when it is downloaded to the Arduino board. This controller was used in the hardware closed-loop testing reported in Figure 7 of the main text.

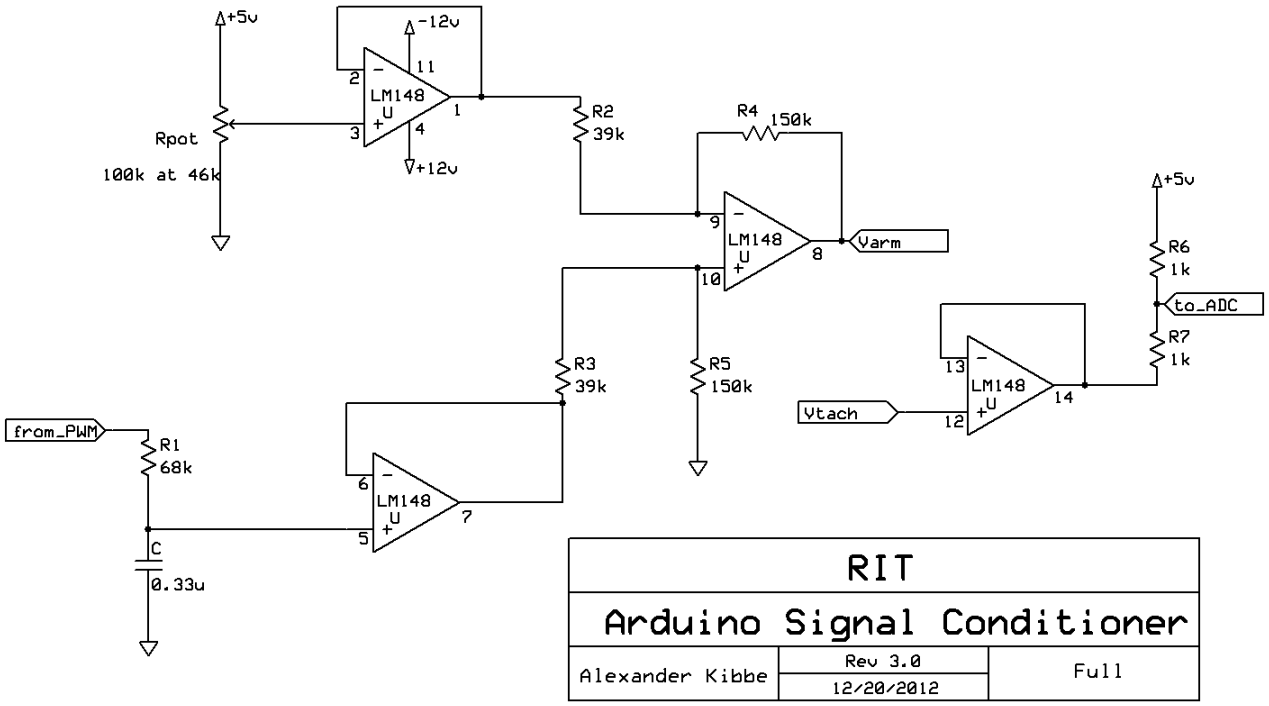

Figure A.1. Schematic of the Arduino-to-dc motor interface board (signal-conditioner).

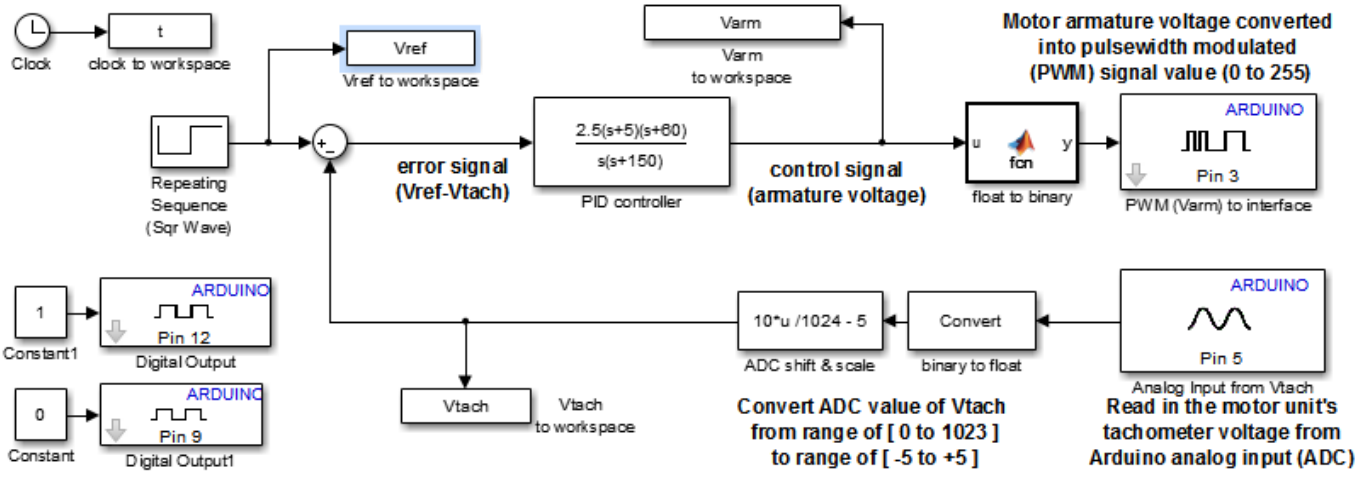

Figure A.2. Example of a PID dc-motor controller for Arduino, as a Simulink model.

Appendix B. A Pure-Integral Controller to Implement in Arduino

This appendix is based on one of the labs performed by students in the controls class for which this lab platform was developed. This lab procedure allows students to investigate pure integral control.

First, students implement Figure B.1 using a transfer-function model of the dc motor (simulation model) that they have determined in an earlier lab. This Simulink diagram enables students to simulate, *i.e.*, to predict, the behavior of the full closed-loop system, which is a dc motor together with the pure integral controller. The 0.8-volt offset at the input is intended to get the motor out of its deadzone before the onset of the step-change, which is programmed to occur about three seconds into the experiment, well beyond the settling time of the motor.

After the simulation is run, the Simulink diagram of Figure B.2 is downloaded to the Arduino target hardware, which runs the same closed-loop, except now with the actual dc-motor unit in the loop, instead of the transfer-function model.

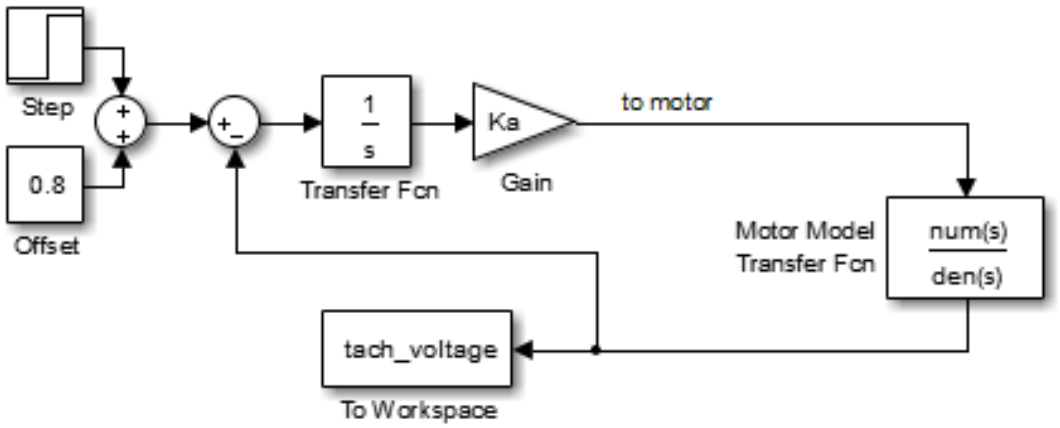

Figure B.1. Simulink diagram for *simulating* closed-loop behavior with pure integral control.

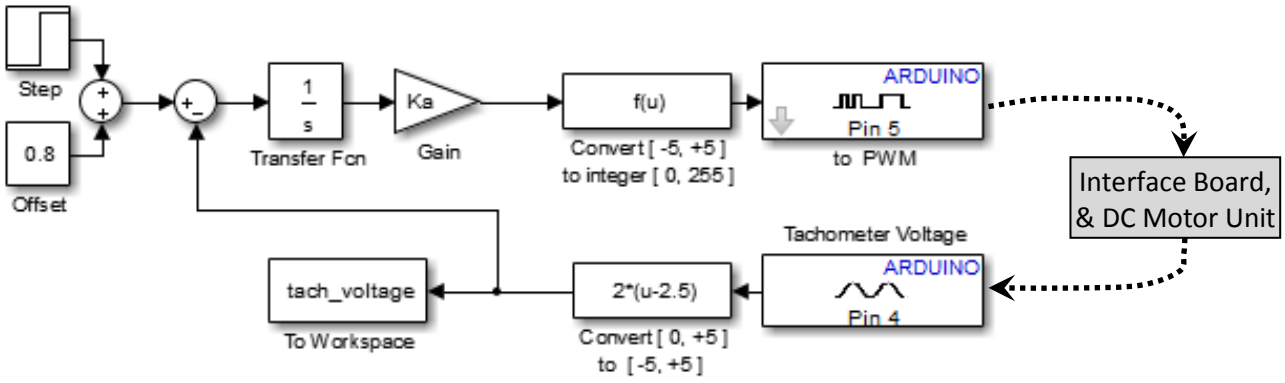

Figure B.2. Simulink diagram for putting hardware-in-the-loop, with pure integral control running on the Arduino.