



## **Introducing Mechanical Engineers to Microprocessors With Arduino Tank Robots**

### **Dr. David R Mikesell, Ohio Northern University**

David R. Mikesell is an associate professor of mechanical engineering at Ohio Northern University. His research interests are in land vehicle dynamics, autonomous vehicles, and robotics. He joined the faculty in 2007 after work in automotive engineering at Ohio State (PhD 2008), six years designing automated assembly machines and metal-cutting tools for Grob Systems, and four years' service as an officer in the U.S. Navy. He holds bachelor degrees in German (Duke 1990) and Mechanical Engineering (ONU 1997).

### **Dr. John-David S Yoder, Ohio Northern University**

John-David Yoder received his degrees (B.S., M.S., and Ph.D.) in mechanical engineering from the University of Notre Dame. He is Professor and Chair of the mechanical engineering at Ohio Northern University, Ada, OH. He has previously served as Proposal Engineer and Proposal Engineering Supervisor at Grob System, Inc. and Software Engineer at Shaum Manufacturing, Inc. He has held a number of leadership and advisory positions in various entrepreneurial ventures. He is currently a KEEN (Kern Entrepreneurial Education Network) Fellow, and has served as a Faculty Fellow at the Jet Propulsion Laboratory, Pasadena, CA and an Invited Professor at INRIA Rhone-Alpes, Monbonnot, France. Research interests include computer vision, mobile robotics, intelligent vehicles, entrepreneurship, and education.

# Introducing Mechanical Engineers to Microcontrollers With Arduino Tank Robots

## Abstract

Though microcontroller programming has traditionally been the dominion of electrical and computer engineers, other engineers must be familiar with the capability and integration of microcontrollers for interdisciplinary tasks. Ohio Northern University has started a two-week module on microcontroller programming in the Computer Applications course required for all mechanical engineers. The course begins with programming instruction in Matlab®, which is then applied in the microcontroller module. Each student had the in-class use of his or her own Arduino® Uno-based track-driven robot.

A number of lessons were learned that would help others to successfully implement this type of module. For instance, hobby-grade robots did not prove sufficiently robust for classroom use, and recently-standard MATLAB/Simulink® native support for Arduino processors is not yet a smooth integration.

Despite these and other difficulties, many students were engaged by the activity. Though less than 8% of the ME students had programmed Arduino or other microcontrollers before this class, 61% indicated that they “would like to play more with microcontrollers in the future.” Other survey results support the success of the trial and guided its revision for the next course offering.

## Introduction

Microcontrollers have dramatically increased in popularity and versatility over the past several years, and are now accessible to the hobbyist and general public. These inexpensive yet powerful devices can now be found in many household appliances and common industrial equipment. Even the most basic modern automobile contains at least 30 microprocessor-controlled devices, and some luxury cars have as many as 100.<sup>1</sup> Such a powerful tool, primarily the domain of electrical and computer engineers, should be introduced to mechanical engineers as well. Mechanical engineers are frequently called upon not only to design a system’s physical mechanism and actuation, but also to design, program, and package the control architecture. But even if they aren’t the primary programmers, they should understand how the interface has to work, as well as how the controller may be limited in memory, speed, and input/output.

Many schools employ microcontroller projects in first year coursework for electrical and computer engineers (e.g., Villanova<sup>2</sup>, Western Kentucky University<sup>3</sup>), but a growing list have incorporated microcontrollers into their first year coursework for all engineers. Louisiana Tech<sup>4</sup> and Western New England University<sup>5</sup> both seek to engage and motivate engineering undergrads in their very first term with build and program challenges for Arduino-powered robots. At Ohio State University, all ~1700 first-year engineers exercise their Matlab skills by programming an Arduino microcontroller to control model railroad crossing gates and regulate the speed of a train as it passes through different environments.<sup>6,7</sup> Giurgiutiu et al. cite over twenty US universities, as of 2005, which included microcontroller and mechatronics education in non-EE curriculum.<sup>8</sup>

Two studies<sup>9,10</sup> linked one or more projects in microcontroller-based robotics to increased retention among engineering students.

This type of engineering-wide integration demonstrates the power and appeal of the microcontroller, but it is not always possible to implement such exercises in courses common for all students. Curricular requirements and topical interest varies widely among engineering disciplines, and consensus for first-year topics is sometimes hard to obtain.

As part of the compromises made when Ohio Northern University transitioned from quarters to semesters, full courses in (C++) computer programming and 3D modeling were removed from the required list for mechanical engineers. A hybrid Computer Applications course at the sophomore level was then developed to teach introductory topics in both of these areas, with emphasis given to Matlab programming. The course outcomes are as follows. Upon completion of the course, students will be able to:

1. Complete a flowchart of how to solve a problem;
2. Use a computer program to solve an engineering problem;
3. Correctly and clearly plot the results of calculations;
4. Program a microprocessor; and
5. Use software to accurately represent a 3-dimensional object.

Prior to this curriculum change, mechanical engineers were not all exposed to microprocessor programming. A number of students employed them in club, competition, or capstone projects, but this was generally a minority. Department faculty decided to seize the opportunity in this new course to introduce microcontrollers to all mechanical engineering students. Not only is it an engaging way of exercising and reinforcing recently learned programming skills, but it opens the students' eyes to the flexibility, utility, and simplicity of these inexpensive devices.

The course begins with programming instruction in Matlab, which is then applied in the two-week microcontroller module. Each student had the in-class use of an Arduino Uno-based track-driven robot, as shown in Figure 1. The two-section class is held in a computer lab with sufficient hardware for each student to have his or her own desk and computer. Students performed tasks with the robot in three different ways: first through the Arduino Integrated Development Environment (IDE), then using Matlab, and finally with Simulink.

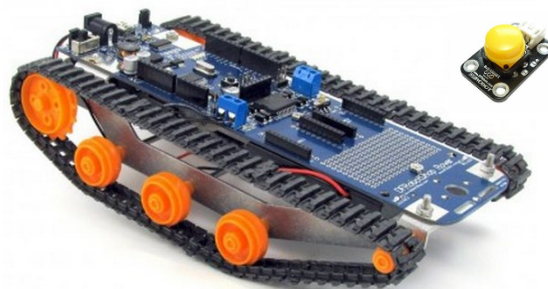


Figure 1: Arduino Uno-based mobile platform and pushbutton<sup>11</sup>

## Instructional Goals

Forty-nine (49) students, split into two sections, took this course in the spring of 2014. Earlier portions of this course were devoted to learning the Matlab user interface, programming syntax, and introductory programming skills such as loops and data structures. Students were also given instruction and assignments on Simulink and its interaction with the Matlab workspace.

The goal of this module was that students would be able to use both Matlab and Simulink to program an Arduino robot to perform basic tasks. In the software version employed in this study (R2013a), Mathworks® has enabled an Arduino to execute Matlab code while tethered via USB, whereas code built in Simulink may be compiled, uploaded and run on the Arduino without the tether. The assignments used may be found in the appendix.

A separate pushbutton was used to provide a manual user interface to the robot. This button was wired to analog pin 1 of the Uno board.

## Challenges

A number of lessons were learned that would help others to successfully implement this type of module. An inexpensive hobby-grade robot was purchased in order to have one robot per student. Unfortunately these robots did not prove sufficiently robust for the lab environment; about 30% of them failed during the two week module. The most frequent causes of failure were separation of the surface-mount USB socket and internal failure of the gearbox. Those considering this type of exercise might select a more durable device or purchase spares and have a technician handy for repairs.

Matlab<sup>12</sup> and Simulink<sup>13</sup> have added native Arduino support to the base package, no longer requiring the purchase of special toolboxes for a select list of microcontrollers. But the implementation is not yet flawless; for R2013a, at least, this required several lab-wide iterations on the installation process and one awkward workaround.<sup>14</sup> Also, in the authors' experience, Simulink blocks in the "Arduino IO Library" did not function, but those in the essentially duplicate "Simulink Support Package for Arduino Hardware" library did work (Figure 2).

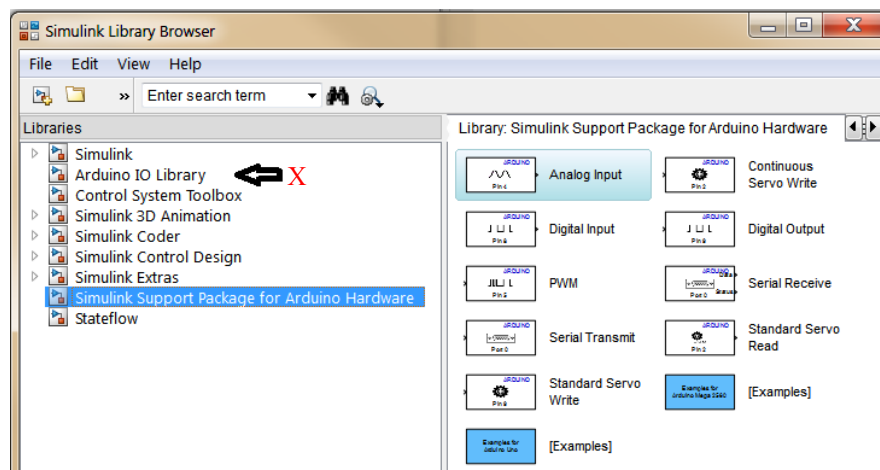


Figure 2: Simulink Block Library with Arduino Support

The programming concept of interrupts was new to virtually all students and proved challenging to communicate. Students were asked to change the robot's direction, using Matlab code, each time a separate pushbutton was depressed. The direction of movement should sequence through forward, left, right, reverse, and stop upon each button click. Very few students figured out on their own that their Matlab program continued to quickly loop while the button was pressed, thus the robot's next direction was assigned according to the (random) value of the button click counter when the button was released. A focused exercise on interrupts would have made the larger programming tasks more accessible.

It was also difficult for students to complete all assignments in class, even though substantial time was allotted for them to do so. Issuing robots to students for the duration of the project would enable them to work out of class, as well as encouraging some level of accountability in the robots' care.

Anyone teaching such a module on a Windows® platform should be prepared for other issues related to the operating system. Windows tends to assign the same COM port to a given Arduino board each time it is plugged in, but will change the port number for new boards of the same type. Students need to become familiar with the Windows Device Manager in order to determine the port assigned to their robot. And as with any USB device, Arduino boards are not always recognized when first plugged in; this, however, is typically remedied by repeating the action or trying a different USB port.

## Results and Assessment

Of the 49 students who took this class in spring semester of 2014, 39 completed an end-of-course survey. Of these 39 students, three said that the Arduino robot programming was the "best thing about the course," and three listed it as the worst. Though less than 8% of the ME students had programmed Arduino or other microcontrollers before this class, 56% enjoyed learning how to do so using Matlab/Simulink. Over half felt confident afterward that they could program the tank robot to do more interesting things than they were required to do in this class. 61% indicated that they "would like to play more with microcontrollers in the future," and 36% strongly agreed with that statement. Over half of the students found programming a microcontroller more enjoyable than their other programming assignments; only 13% found it less enjoyable.

This instruction module and survey will be repeated at the end of spring 2015. With the same number of robots available this year, the instructors plan to modify the exercises such that a robot will be issued to a student group (rather than one per student) to enable them to work on the assignment outside of class.

The questions shown in Table 1 were answered on a Likert scale where 1 = strongly disagree, 2 = disagree, 3 = neutral, 4 = agree, and 5 = strongly agree.

Table 1: Partial Survey Results (n = 39)

Survey Question	Avg. Response
I have programmed Arduino (or other) microcontrollers before this class.	1.36
I enjoyed learning how to program a microprocessor using Matlab/Simulink	3.56
I liked having my own robot to work with in class.	4.41
I would have preferred to work in groups of two.	3.69
After these exercises I think I could program the robot to do more interesting tasks.	3.46
I would like to play more with microcontrollers in the future.	3.82
I like Simulink more than Matlab.	3.31

Anecdotal answers to the survey question, “What was the hardest part of successfully programming the tank robot?”

- Getting the robot and computer connected. (4 related comments)
- Lack of access to the robot outside of class. (7)
- “Making sure the program doesn’t run through millions of times while you push the button.”
- “Debugging code. It was much harder to determine what was going wrong because there were so many reasons why something could be malfunctioning.” (4)
- “Making it move in the directions I wanted it to.”
- Getting the LED to blink.
- “Learning how to operate Arduino, Matlab, and Simulink together.”

The first two comments relate to some implementation challenges faced by the students and instructor. The next two comments point to key lessons that the students learned about the nature of embedded system programming. Regarding debugging, students can be taught to implement and test code in small sections, as well as using the LEDs available on the board to indicate which part of the code is active.

## Conclusions

The power, popularity, and ubiquity of microprocessors and microcontrollers make a compelling argument for engineers of all types to be instructed in their use. Ohio Northern University faculty created a module in a required course for mechanical engineers where students must apply and extend their recently-gained Matlab programming skills to accomplish several tasks with an Arduino-powered tank robot. Though the two-week module was limited in scope, survey results suggest that it was generally successful in increasing student interest in microcontrollers and giving them the confidence to tackle other microcontroller tasks.

Faculty interested in implementing this type of project should carefully evaluate their hardware options. Hobby-grade kits are more affordable, but spare robots and parts should be available to replace those which break. Microcontroller enthusiasts will employ a variety of programming

languages, but native Arduino support in Matlab and Simulink makes this a natural engineering software platform for introductory microcontroller instruction. It improves Matlab programming skills that many students will need for their control systems or experimental methods courses, or builds on the familiarity they already have.

## Bibliography

- 
- <sup>1</sup> Motavalli, J. "The Dozens of Computers That Make Modern Cars Go (and Stop)." *The New York Times* 4 February 2010: online. [www.nytimes.com/2010/02/05/technology/05electronics.html](http://www.nytimes.com/2010/02/05/technology/05electronics.html)
- <sup>2</sup> Mercede, F.J., "Hands-on projects to introduce Electrical and Computer Engineering," Frontiers in Education Conference, 2008.
- <sup>3</sup> Cambron, M. "Using the Arduino in Freshman Design," 6th First Year Engineering Experience Conference, 2014.
- <sup>4</sup> Living with the Lab, <http://www2.latech.edu/~dehall/LWTL/home/courses.html>, Louisiana Tech University.
- <sup>5</sup> First Year Program in Engineering, <http://www1.wne.edu/engineering/index.cfm?selection=doc.421>, Western New England University.
- <sup>6</sup> Adding Fun to First Year Computer Programming, <http://www.mathworks.com/company/newsletters/articles/adding-fun-to-first-year-computer-programming-classes-with-matlab-arduino-microcontrollers-and-model-trains.html>, Mathworks.
- <sup>7</sup> Fundamentals of Engineering I, <https://eeicourses.engineering.osu.edu/1181/content>, Ohio State University.
- <sup>8</sup> Giurgiutiu, V., Lyons, J., Rocheleau, D., and Liu, W., *Mechatronics/microcontroller Education for Mechanical Engineering Students at the University of South Carolina*, Mechatronics, 2005, **15**(9): pp. 1025-1036.
- <sup>9</sup> Saad, A., "A Project-based Approach to Learning Digital Logic Design Using Simple Mobile Robots," in Proc. Int'l. Conf. on Eng. Educ., Valencia, Spain, 2003.
- <sup>10</sup> Freuler, R., Fentiman, A., Demel, J., Gustafson, R., and Merrill, J., "Developing and Implementing Hands-on Laboratory Exercises and Design Projects for First Year Engineering Students," in Proc. ASEE Annual Conf. and Expo., 2001.
- <sup>11</sup> DFRobotShop Rover V2, <http://www.robotshop.com/en/dfrobotshop-rover-tracked-robot-basic-kit.html>, 28 Jan 2015.
- <sup>12</sup> Arduino Support from MATLAB, <http://www.mathworks.com/hardware-support/arduino-matlab.html>, 28 Jan 2015.
- <sup>13</sup> Arduino Support from Simulink, <http://www.mathworks.com/hardware-support/arduino-simulink.html>, 28 Jan 2015.
- <sup>14</sup> Arduino for Simulink Support Package, <http://www.mathworks.com/matlabcentral/answers/89753-arduino-for-simulink-support-package-installed-and-works-fine-for-me-but-not-for-other-users>, 10 Oct 2013.

## Appendix

### HW1: Arduino Robot Control using Matlab

Modify the *motor\_control\_start.m* program to accomplish the following:

- When the program starts, the rover should not move until the push button is pressed.
- Each time the button is pressed (“clicked”), the rover changes behavior:
  - Click 1: Rover moves forward at full speed
  - Click 2: Rover rotates left at half speed
  - Click 3: Rover rotates right at half speed
  - Click 4: Rover moves in reverse at half speed
  - Click 5: Rover stops.
  - Click 6: Go to “Click 1” behavior, continue to cycle through these modes.
- Each time the button is pressed, the LED attached to pin 13 will change state (on→off, or off→on).

#### *motor\_control\_start.m*

```
%=====
%*** STOP!! BEFORE RUNNING THIS CODE, CHANGE THE COM PORT ***
%*** in line 11 to whatever port number you see the Arduino ***
%*** attached to in the Arduino IDE program or Device Manager. ***
%=====
%*** If Matlab cannot connect to the COM port:
%*** (1) Close Matlab
%*** (2) Unplug the Arduino, wait a few sec, plug it back in
%*** (3) Open Arduino IDE and find proper COM port
%*** (4) Reopen Matlab and go. If necessary, change COM port.
%=====

clc;
if exist('a','var') && isa(a,'arduino') && isvalid(a),
    % nothing to do
else
    a = arduino('COMX'); % CHANGE THIS! e.g., a = arduino('COM4')
end

ML = 8; % Left motor direction control is pin 8
MR = 7; % Right motor direction control is pin 7
SL = 6; % Left motor speed control is pin 6
SR = 5; % Right motor speed control is pin 5

spd_L = 0; dir_L = 0; % Max speed = 255; Min speed = 0;
spd_R = 0; dir_R = 0;

a.pinMode(ML,'output'); % Before writing to any digital pin, you must
a.pinMode(MR,'output'); % first put it in 'output' mode.
a.digitalWrite(ML,dir_L);
a.digitalWrite(MR,dir_R); % a.digitalWrite(pin_number,output_value)
a.analogWrite(SL,spd_L); % a.analogWrite(pin_number,output_value)
a.analogWrite(SR,spd_R);

t = 0; % t = program time in seconds
```



```

tic
while t < 20
    x = a.analogRead(1);    % Read the button attached to analog pin 1
    if x < 500             % If button is depressed, turn motors on
        spd_L = 255; spd_R = 255;
    else
        spd_L = 0; spd_R = 0;
    end
    a.analogWrite(SL, spd_L);
    a.analogWrite(SR, spd_R);
    t = toc;
end

a.analogWrite(SL,0);      % Turn both motors off when program is done
a.analogWrite(SR,0);

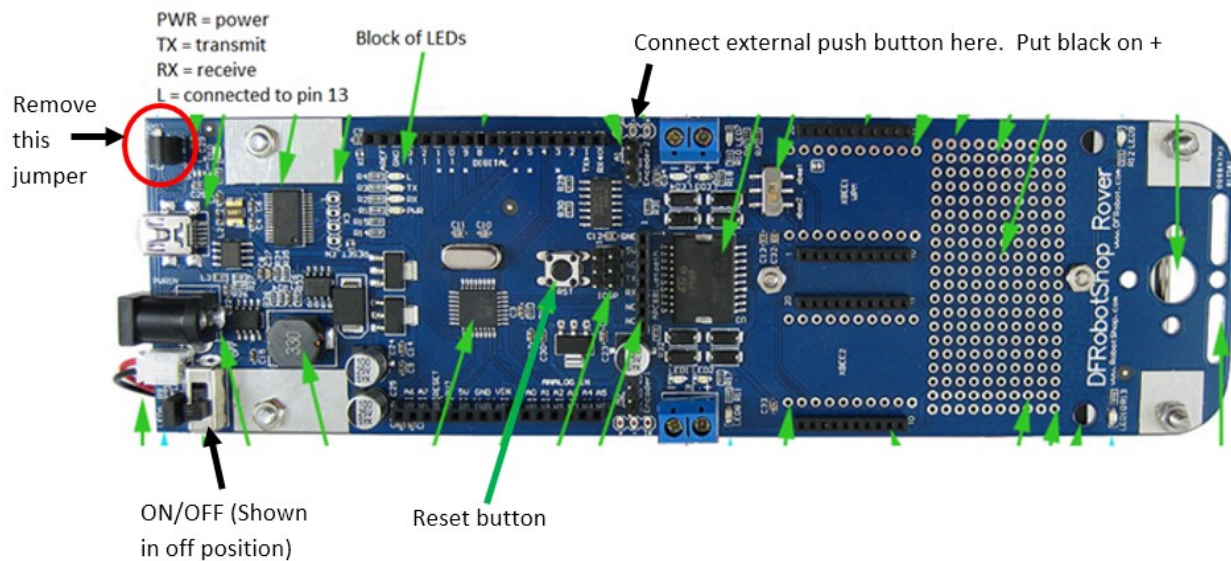
```

## HW2: Arduino Robot Control using Simulink

FIRST! Turn sheet over and go through “Simulink and the DF Robotshop Rover”

Write a Simulink program to accomplish the following with the untethered Rover:

- At program start, the Rover moves forward at full speed.
- When the button is pressed, the Rover rotates right

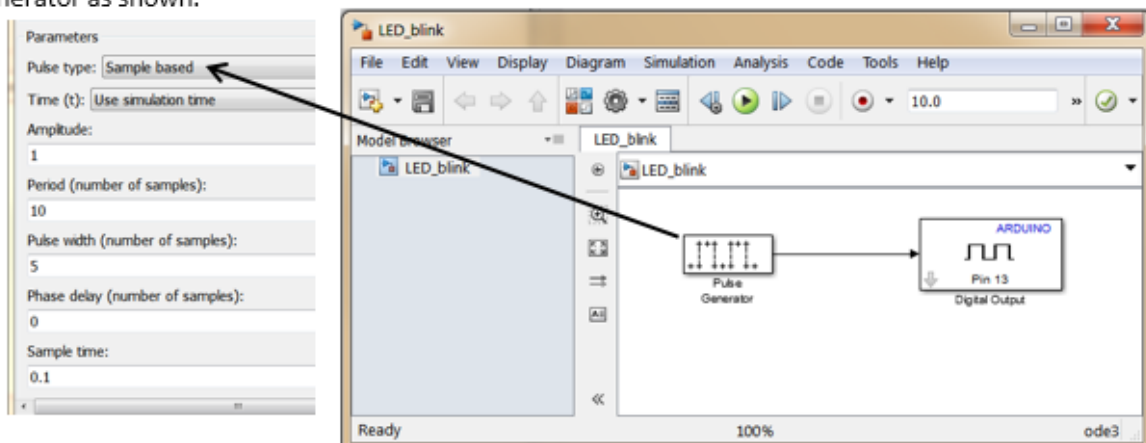



<HW2 REVERSE>

## Simulink and the DF Robotshop Rover

Open Simulink, write and deploy a sample program.

1. \*If you have already been communicating with the Arduino using Matlab, and an "Arduino" object exists in the workspace, you must type `clear` to remove this object before Simulink will be able to give the board new instructions.\*
2. Open the Simulink Library Browser and start a new model.
3. If you want to make a deployable (non-tethered) model, DO NOT USE THE "Arduino IO Library." YOU MUST USE THE BLOCKS IN "Target for use with Arduino hardware" (R2012b) or "Simulink Support Package for Arduino Hardware." (R2013a)
4. Form the model pictured below. Change the digital output to pin 13 and configure the pulse generator as shown.



5. Prepare the model to run using the following steps.
  - a. In the menu bar of the block diagram, Tools > Run on Target Hardware > Prepare to Run.
  - b. Select "Arduino Uno" in the Target Hardware drop down menu.
  - c. On the next screen, Host-board connection: Set host COM port: *manually*. Type in the COM port that you know the board is currently using. (Use the Arduino IDE or Device Manager to find out if you don't know.) Click OK.
  - d. Incidentally: The previous configuration step also automatically changes Simulink to an ode3 fixed step solver. The solver time step selection does not seem to matter.
6. Download and run the program. Tools > Run on Target Hardware > Run.
  - a. If all works as planned, the LED will start blinking and the message "Model successfully downloaded to 'Arduino Uno'" will be displayed in the lower right-hand corner of the model.
  - b. Unplug the Rover and you will see that, unlike programming with Matlab, the LED will continue to blink (the program will not stop).
    - i. If you turn the Rover off, the program will stop. But the program will restart when turned back on.
    - ii. The program will start from the beginning if you press the reset button 
7. Two ways to get the existing deployed Simulink program off the Rover:
  - a. Overwrite it with a new program.
  - b. Open the IDE and rewrite `adivoes.pde` to the board.

HW2 Sample solution.

