Unit 0 - Assignment 0

September 25, 2024 6:28 PM

Comp 444 Embedded / Robotic Programming

Introduction to robotic programming using the Arduino. The course progresses from first principles to advanced topics in robotic control.

Objectives: (https://comp.athabascau.ca/444/r1/unit00.html>)

- discuss robots in general, including the history and features of robots.
- describe robotic features including affectors, actuators, and control processes.
- discuss robotic control mechanisms including feedback, architectures, deliberative, reactive, hybrid, behaviour-based, and coordination.
- discuss emergent behaviour and distinguish this from normal robotic behaviour.
- discuss robot learning in the context of current robots.
- design and create robots to perform tasks from simple movement to complex interactions with the world.
- explore robotic concepts with hands-on experiments using the Arduino and the SparkFun Inventor's kit.
- articulate design decisions and create a diary describing learning experiences that form a portfolio of competence.

Expectations

- Senior level course, senior level expectations
- Students, tutors and professors must show genuine interest to learn and research
- Student must maintain diary on the Landing and participate in fellow student discourse.
- Students, tutors and professors must follow all standard AU policies and standards

Course Information / Reference Materials:

- Textbook: Matarić, M.J. (2007). The Robotics Primer. MIT Press.
- Workbook: https://sourceforge.net/projects/roboticsprimer/
- COMP 444 on Landing
- Landing Groups: https://landing.athabascau.ca/groups/member/timothybl
  - COMP 444
  - Programming and Problem Solving
  - Robotics and Embedded Controllers
- Arduino Wiki
- SparkFun Inventors Kit
  - All-in-one kit with board, wires, power, servos, actuators, sensors and more
- (Instructors links)
  - Sparkfun SIK Guide
  - Sparkfun Inventors Kit for Arduino
  - Arduino Homepage
  - Lady Ada Arduino Tutorial
  - SteamPunk R&D Intro

Portfolio / Weblog

- A log book must be maintained and will contain:
  - Answers to study guide and textbook questions
  - Log experiments (including outcome)
  - Record of readings, questions, forums postings, Arduino exercises, etc
- Each entry in the log should be linked back to Objectives
- Weblog will be maintained on the Landing
- Encouraged to try video recordings
- Will be required to submit weblog periodically

Assignments:

- 0: Setup Profile
  - Review study guide
  - Create landing profile and blog
  - Review course material
  - Setup and test development environment
- 1: Unit 1 and 2
  - Submit blog and diary
  - Designs, discussions, diary entries describing examples from unit 1 and 2
  - Working examples of LEDS, actuators, motors and sensing light levels
  - Answers from Unit 1 and 2 from textbook
- 2: Unit 3 and 4
  - Designs, discussions and entries from units 1 to 4
  - Examples of actuators and sensors
  - Answers from Unit 3 and 4 from textbook
- 3: Unit 5
  - Designs, discussions and entries from units 1 to 5
  - Working examples of multiple feedback control mechanisms
  - Answers from Unit 5 of the textbox
- Project
  - Combined knowledge from all units into a significant working project designed between student and tutor
  - Employ significant control elements from Units 5 to 11
  - Save all discussions into a PDF and submit as part of collaborative component

Unit 0 - Blog Entry

September 30, 2024 7:12 PM

I've read and summarized the course intro, now it's time to install Arduino IDE and setup the Spark Inventors Kit (SIK) configured. I'm pretty comfortable with the Arduino IDE for a variety of boards, but something I haven't tried is integrating Git. Git is a free (and optionally paid) software source control service that supposed to integrate nicely into the IDE. I wish to use this because I want the option to develop on both my portable Microsoft Surface, as well as my desktop PC, plus the added convenience and protection of source control is a big bonus. No better time to challenge myself!

ArduinoIDE was already installed, a simple update was all that was needed. Installing the SIK guide code was a breeze with the provdied Sparkfun Kit instructions. Git on the other hand… There is not native integration with Git and Arduino IDE. I was able to accomplish my goal by using Git folder implementation, and tested with a hello world project - Success! This will allow me to manage code better and work from multiple PCs.

After all the software installed, I connected followed the instructions in the SIK manual for Project 1A. It was a very simple setup, I sure enjoyed the breadboard - redboard place holder, nice touch by sparkfun Electronics. Setting up the board, LED, resistors, jumpers, USB and IDE was pretty straightforward, something I've done a dozen times before. To my surprise though, the program did NOT run the first time. I immediately knew the LED was in backwards, unplugged the board, switched the LED, and boom, problem solved - a nice blinking light.

Now, I wonder what else I can do with this? How about making the light blink S.O.S.? In order to do that, I'll need the code to make a light blink, an LED, and the timings of when the light should blink on and off. I've already got 2 of 3 requirements, a quick google and I've found that: A dash should be 3 units of length, a dot is 1 unit of length, the time between letters is 1 unit of length, and time between words is 3 units of length. Easy enough to encode those variables and make some functions to represent a dot or dash. It's up to me what a unit of measure is, and my first guess was clearly wrong:

(Seizure warning - Flashing light!)
P1A- Blink SOS - too fast (seizure warning!)

But after a quick tweak of one variables, we've got a great looking S.O.S. flashing light (I'll post the code at the bottom of this posting)
https://youtu.be/VuwL4KKGUFk

That about wraps up the first entry, and project. Was a lot of fun and looking forward to doing more. As for that S.O.S. program, it sure would be neat to do the opposite project; switch the LED for a button, and get the Arduino to read button presses into Morse code - maybe another time.

p.s. Is the Landing is in need of some more horsepower, plenty of timeouts while uploading.

Here is the code for the S.O.S.:

```
/*
  Sp4rkFUn Inv3nt0r’s K1t  ---- HACKED BY TIM BLAKE TO Samuel Morse!
  Turns your SIK into a life saving device, for anyone doing CMP 444 on the
open ocean.
*/
void setup() {
  pinMode(13, OUTPUT);      // Set pin 13 to output
}
int unitOfTime = 150; // Units of time, in milliseconds
int dotLength = 1;    // How long to show a dot for? 1 unit of time
int dashLength = 3;   // How long to show a dash for? 3 units...
int pauseLength = 1;  // How long do we wait better letters?
int wordLength = 3;   // How long do we wait between words?
void dash (){
  digitalWrite(13, HIGH);            // Turn on the LED
  delay(dashLength*unitOfTime);      // Wait for DASH units of time X UNITS of
time (ms)
  digitalWrite(13, LOW);             // Turn off the LED
  delay(pauseLength*unitOfTime);     // Wait for pause units of time X UNITS
of time (ms)
}
void dot (){
  digitalWrite(13, HIGH);             // Turn on the LED
  delay(dotLength*unitOfTime);        // Wait for two seconds
  digitalWrite(13, LOW);              // Turn on the LED
  delay(pauseLength*unitOfTime);      // Wait for two seconds
}
void nextLetter()
{
  digitalWrite(13, LOW);   // Turn on the LED
  delay(wordLength*unitOfTime);
}
void loop() {
// S is 3 dashes
  dash();
  dash();
  dash();
  nextLetter();
// O is 3 dots
  dot();
  dot();
  dot();
  nextLetter();
// S is 3 dashes again
  dash();
  dash();
  dash();
// take a quick nap
  nextLetter();
  nextLetter();
}
```

I then attempted the constant LED challenge and noted that approximately 12-13 millisecond delay was the limit for when the LED appeared to be constantly lit.
Clock cycles: Just out of curiousity I googled the datasheet for the Redboard to see if its clock cycle had any impact on my test. According to the datasheet the clock cycle is anywhere from 10 - 40 MHZ. According to UnitJuggler the upper limit, 40 MHZ, converts to 25 nanoseconds, and given 1 nanosecond is equal to 1 million milliseconds, there would be no impact.

Here is the code that I used:

```
/*
  SparkFun Inventor’s Kit
  Circuit 1A-Blink
  Turns an LED connected to pin 13 on and off. Repeats forever.
  This sketch was written by SparkFun Electronics, with lots of help from the
Arduino community.
  This code is completely free for any use.
  View circuit diagram and instructions at:
https://learn.sparkfun.com/tutorials/sparkfun-inventors-kit-experiment-
guide---v41
  Download code at: https://github.com/sparkfun/SIK-Guide-Code
*/
void setup() {
  pinMode(13, OUTPUT);      // Set pin 13 to output
}
int loopD = 13;  // loop delay, tweak this variable until light appears
constant
void loop() {
  digitalWrite(13, HIGH);   // Turn on the LED
  delay(loopD);              // Wait for two seconds
  digitalWrite(13, LOW);    // Turn off the LED
  delay(loopD);              // Wait for two seconds
}
```

Unit 1 - Introduction

September 25, 2024 8:34 PM

Note from Tim: Based on the instructions I understand we have a certain freedom to our blog. I've decided to structure based off of OneNote notebooks - I absolutely love OneNote and live by it professionally. Each unit will be a "Section", then sections will be broken into pages for introductions/descriptions of the unit, as well as a personal summary of readings, and SIK exercises. For each unit submission, I will do an extract of that unit to HTML or PDF. Final project submission will be a complete export of the notebook. I will include images/videos examples of exercises within the pages when possible, however they are also available via my git repo: https://github.com/malcolmsdad/COMP444/tree/main (FYI Malcolm is the name of one of my adorable dumb cats)

Learning Objectives

After completing this unit, you should be able to

- define the term *robot* and provide a brief history of robots and robotics.
- list the components of a typical robot.
- continue on with more advanced robotic material.

Readings

Please read the following chapters in the textbook:

- What Is A Robot? (Chapter 1)
- Where Do Robots Come From? (Chapter 2)
- What's in a Robot? (Chapter 3)

Questions to Ponder

At the end of each chapter in the assigned readings there are questions labelled "Food for Thought." Please answer these questions as best you can in your weblog, which will become part of your portfolio of competence submitted for marking during this course.

Exercises

Exercises for this unit can be found in the *Instructor's Weblog* on the Landing. Please follow along with the exercises and programs using your own Arduino and the SparkFun Inventor's Kit, and keep a record of your explorations in your own weblog.

Further Readings

At the end of each chapter in the assigned readings you will find a section titled "Looking for More." While the links and readings mentioned in this section are not assigned, please feel free to examine them if you are interested, or ask questions about them on the Landing.
Bookmark the brief history of the Arduino found on *Wikipedia* http://en.wikipedia.org/wiki/Arduino, and check out the official Arduino home page at http://www.arduino.cc/.

Unit 1 - Instructor Blog Entry

October 9, 2024 11:21 PM

As mentioned in the Study guide and this notebook, the companion text workbook is somewhat incomplete, and does not completely apply to our chosen hardware platform. The workbook also does not correspond well to the SIK Guide, in that the guide explores some hardware elements in different order than the text's workbook. However, this is not really a significant difficulty, as we want to first explore the various hardware elements in our Inventor's kit, after which we can tackle some of the text's robotic questions.

Unit 1 is a general discussion of locomotion. Unit 2 will further explore robotic locomotion, but for now we shall begin by working through the SIK tutorials.

Bearing that in mind, you should begin to work through the SIK Guide tutorials in Section 2, making notes of your progress, questions, challenges and solutions as you go. Due to the structure of the SIK Guide, I recommend you tackle the circuits and programs in the order they are presented, starting with Circuit #2 and continuing through Circuit #7. These circuits comprise the tutorial's coverage of sensor inputs, which will be put to further use in Unit 4, and pave the way for the next unit's circuits.

Activities using the text Companion Workbook: Read through the text companion workbook section titled Robotic Components. Answer the questions posed in this section of the workbook in your own weblog.

From <https://landing.athabascau.ca/pages/view/242993/unit-1>

Chapter 1 - Robots

September 25, 2024 8:36 PM

Well, here we go. The beginning of another course. I'm very much looking forward to this one though, this is much more aligned with my personal interests and hobbies.

I've been programming nearly my entire life, professionally for the last 20 years. I've also dabbled with embedded and micro-computing quite a bit as well. I'm an avid Raspberry PI hacker, have done a lot of projects with Arduinos, ESP32s and even Intel Galileo. Robotics is totally new to me though, so I'm really excited to see dive deeper into that.

Chapter 1: Robotics

This chapter provided a very brief history of robots, the original inventor of the term (Josef Capek, or his brother). It also described the evolution of the term robot, as well as defining the modern meaning of the word. (See Definitions section)
Primary concepts of robots are introduced such as the difference between teleoperation and autonomy. A great deal of emphasis was put into defining sensors and robot must sense its environment to be considered robotic.

Definitions

October 7, 2024 10:21 PM

Robot: an autonomous systems that exists in the real world. It can sense its environment and interact with it. It must be an autonomous system, that exists in a physical world, can sense its environment and act upon it in order to achieve some goals.

Autonomous: an actor capable of acting on the basis of its own decisions and is not controlled by a human.

Teleoperate: to operate a system from afar.

Sensors: Sensors are physical devices that enable a robot to perceive its physical environment in order to gain information

Embodiment: having a real body

Niche: environment or position in ecosystem

Food for Thought

October 9, 2024 5:24 PM

What else can you do from afar, by means of teleoperation? You can talk, walk and see, as in telephone, telegraph and television. There is more. Can you think of it?

The general theme for all of these question is: are they robots. In order to properly answer these questions, I believe it'd be best to review the definition and requirements to be declared a robot.

The definition of a robot is an automated machine that can detect is environment, and interact with it. It also most meet the following requirements:

- Be autonomous (not controller by human)
- Exist in physical world
- Sense its environment
- Can act upon its environment
- Achieve some goals

Is a thermostat a robot?
Yes and No. Like robots, thermostats can vary wildly.
Modern thermostats are intelligent, loaded with sensors, connected to the internet, paired with mobile apps and easy to program - a far cry from the days before IoT. The latest in thermostat technology sees the devices optimizing home energy use and comfort by through data collection and analysis such as downloading weather patterns to predict heating or cooling requirements, monitoring room or house occupancy to reduce energy use when not needed. Learning 'time to heat' statistics to predict when heating or cooling cycles should begin. In these situations, I would consider thermostats robots because:

- Some newer thermostats have the ability to learn users patterns and temperatures and adapt heating/cooling cycles based on those patters - this demonstrates autonomy
- Thermostats absolutely exist in the real world
- Thermostats physical capability is the manipulation of electronical currents to one or many devices such as HVAC appliances, such as fans, humidifiers, furnaces, boilers, air conditions, heat pumps etc. The goal of this action is to control the devices environment.
- The goal of these modern thermostats is to increase the comfort (temperature, humidity) and reduce energy usage.

Meanwhile, older thermostats, which consist of a simple dial, or basic display and temperature control button, are not robots. These thermostats rely on solid state or mechanical process to trigger heating/cooling cycles. These thermostats rely more on physics than robots. Comparing against the list earlier, we can see older thermostats:

- Exist in the physical world
- Can act up on its environment

But that's about the limit. Older thermostats:

- Do not sense their environment, although it may seem that they do, since they need to know the temperature of the environment, there is no perception in the thermostat when the furnace turns on due to cold air triggers - it's simply a thermodynamic reaction
- They have no goals to achieve, they simply operate

Is a toaster a robot?
A toaster is definitely not a robot. Toaster do not sense their environment and act upon it. They simply active heater coils for a user input amount of time.

Some intelligent programs, also called software agents, such as web crawlers, are called "soft bots" Are they robots?

No, these are not robots, because they do not satisfy the condition of "existing within the real physical world"

Is HAL, from the movie 2001, the Space Odyssey a robot?

Similarly, HAL is not a robot, because it does not satisfy the condition of "existing within the real physical world"

Chapter 2

October 9, 2024 10:05 PM

I appreciate the short and to the point chapter 1. Chapter 2 discusses more history of robotics.
The first topics is "Control Theory" which is defined as the mathematical analysis of automated control systems. I wasn't satisfied with this definition, so I did some secondary research. My initial thought was it was a specific theory describing control system, but in fact it’s much more broad; its an entire branch of applied mathematics that performs objective analysis of feedbacks to evaluate and optimize system responses to goals and inputs.

The chapter then discusses cybernetics which describes the study of biological organism and comparison against artificial systems, a perfect prelude into the next section about a series of biomimetic tortoises by Will Grey Walter.
These robots, named Elmer and Elsie, were early tricycle based robots, that were able to sense and navigate through their environment, and even follow basic instructions based on light or sound inputs.

These tortoises were used to introduce concepts including: analog, analog electronic circuit, reactive control, emergent behaviour and artificial life. The analog and analog circuitry are now much outdated, due to the invention of the transistor. Reactive control is a new concept I learned, and something I'm quite concerned with, because it does focus heavily in a robot I'm pondering for the final project. I'm looking forward to Chapter 14 to go in more depth. Reactive control, led to an interested observed phenomenon called emergent behaviour which meant how robots exemplified animal like, realistic behaviour.

The chapter continues with another evolution of the tortoises; Valentino Braitenberg, wrote of a series of thought experiments (gedanken experiments) who described a series of minimalist robots demonstrative an inhibitory connection between sensor inputs and the robots reaction could mimic social behaviour, and even love or aggression. He did this by introducing concepts of his robots (Braitenberg vehicles) light sensors (photophilic and photophobic) reaction to light in positive (excitatory) or negative (inhibitory) connections.

The study to these emergent behaviours and input/output connections led to Artificial Intelligence, which is the next section. AI was officially born in 1956, Dartmouth University in Hannover, NH, USA. Defined AI as requiring:

- Internal models of world
- Search through possible solutions
- Planning and reasoning to solve problems
- Symbolic representation of information
- Hierarchical system organization
- Sequential program execution

AI is also something I'm very much interested in and look forward to further reading.
The chapter then described some early AI implementations in robots including Shakey, Flakey and Hillaire, CART and Rover.
Finally, the concept "types of robotic control" is introduced, which describe the different directions of robotic study including reactive control, behaviour based control and now purely deliberative.

Definitions

October 14, 2024 9:45 AM

Control Theory: A mathematical analysis to automation control systems

Cybernetics: a field of study that analysis biological systems at the nerve cell (neuron) level. It studies and compares biological and artificial systems with the goal of finding common properties or principals between the two.

Biomimetic: Machines with properties similar to biological systems or imitate biological systems in some way

Braitenberg Vehicle: a simple robot initially equipped with a light sensor and servo and programmed to react in a variety of ways based on light inputs. These robots were supplemented with additional sensors to study their reactive behaviours.

Photophilic: Act of being attracted to light

Photophobic: Act of being afraid of light

Excitatory Connection: The stronger the input, the strong the motor output.

Inhibitory Connection: The stronger the input, the weaker the output.

Artificial Intelligence: field of study that defines intelligence in machines and what is required for a machine to be intelligent. Understanding of world, ability to research, plan and reason solution. Ability to represent information symbolically, have hierarchical system organization and execute programs sequentially.

Shakey: An early intelligence robot that lived in a carefully controlled environment, with limited sensory inputs, and given a task. The robot would then attempt to solve the task iteratively through trial and error.

Vision-Based Navigation: Navigation of a robot through it's environment using vision sensors. Initially demonstrated in the CART robot, it was very slow to react due to time to process visual information.

Types of Robot Control: Different approaches to robot control including reactive, hybrid and behaviour based controls.

Food for Thought

October 9, 2024 7:37 PM

How important is it for robots to be inspired by biological systems? Some people argue that biology is our best and only model for robotics. Others say that biology teaches us valuable lessons, but that engineering can come up with its own, different solutions. The airplanes is a popular example of non-biomimetic engineering; people's first attempts to build flying machines were inspired by birds, but today's planes and helicopters share little in common with birds. Does it matter what kind of robot you are building (biomimetic or not)? Does it matter if it is going to interact with people? For more on this last topic, wait for Chapter 22.

I disagree with robots being inspired by biological systems, both for defining or modelling artificial intelligence and form, or physical attributes. We hardly understand biology to any point that we are able to reproduce to its most basic element. Only last week did we (humans) finally map the brain of a fruit fly, if the penultimate robot is a bipedal humanoid (like demonstrated in some many movies and electronics shows), then that'd require understanding the biology of much more complex biology like humans.
Even on our tiny little planet, biology of creatures and regions ranges wildly. Some creatures rely upon hands and fingers, others use claws, or wings, some even use suction cups and beaks to interact with things - who is to say that some new form of mechanical interaction may occur. Perhaps humans could create a method of manipulating an object with gravitational waves, or ionic thrust. Those methods of interaction could be far superior to a couple fingers.
Robotics should not limited to what biology has evolved to offer, it should be design ed and implemented to best suits its goals or needs.

Chapter 3

October 9, 2024 10:05 PM

The chapter starts with a refresh on the definition of a robot. It then describes the main components; a physical body, sensors, efforts and actuators, and a controller. The chapter then goes into further detail of physical bodies, called embodiment and their limitations; including comparing human limitations to robot limitations.

Sensing is then examined, including describing sensor perception, storage of sensor information (called state), how that information is perceived (observably, partially, or hidden) as well as how often that information is categorized (discrete or continuous). Storage techniques and requirements are then described - in quite detail.

The chapter then moves onto the more physical properties such as Actions, Brains and Brawn. Here we learn how robots can act upon their environment using effectors and actuators. The types of activities are broken into two categories: Locomotion or manipulation. The concept of degrees of freedom is introduced as the limitation in which a manipulator can move.
The chapter used limitation as a prelude into the next section which described power limitations imposed on robots, and how power is a major problem in practice robots, including storage, isolation, performance and replenishing.

Finally the chapter describes the controllers importance to a robot and how they are the 'brains of the robot'. Controllers use hardware and software programming to use sensor input to decide on how what to do.
Pseudocode is introduced as the method the chapter will describe programming of a robot, and autonomy is introduced as the ability to make ones own decision and act upon them.

Definitions

October 14, 2024 10:28 AM

Embodiment: the act of having a physical body

Sensors: physical devices that enable a robot to perceive its physical environment

Sensing Perception: The process of receiving information about the world through sensors

Niche: Environments that animals or robots live or operate within.

State: The robots description of itself at any point in time.

Observable/Partially/Hidden State: a state may be observable, partially observable or hidden to a robot.

Discrete / continuous state: Discrete information is up or down, blue or red, whereas continuous is information such as miles per hour.

State space All possible states a system can be in; ie: a light switch is on or off

Space: Refers to all possible values or variations of something

External State: The state of the world as the robot can perceive is the external state.

Internal State: The state of the robot as the robot can perceive it.

Representation/Internal Model: all information relating to the world collected by sensors

Sensor/Perceptual Space: The space required for all a robots sensors, in every possible sensory reading or permutation.

Effectors: Devices that enable a robot to take action, do physical things. Ie: Flippers, wings

Actuators: Underlying mechanisms that do actual work for a robot such as motors or muscles.

Degrees of freedom: The amount of movement a manipulator can move.

Locomotion: The act of moving around, going places

Manipulation: The act of handling of objects

Pseudocode is an intuitive way to describe the controller without using any particular programming language.

Autonomy is the ability to make ones own decision and act upon them.

Food for Thought

October 9, 2024 7:52 PM

What do you think is more difficult, manipulation or mobility?

Think about how those abilities develop in babies, children, and then adults.

- How large do you think your sensor space is?
- Can you think of things or information in your life that is observable, partially observable, or hidden?

I believe that manipulation is a more difficult task to achieve for a robot. If we consider what the most common form of mobility for machine is based on wheels and circular motion. Circular motion is easily accomplished by the most basic motor. A set of wheels, a simple motor and a bump sensor and you can easily achieve mobility.
Manipulation however is much more difficult. In order to manipulate an object, the robot must first sense or perceive the object, its location and orientation, as well as how to manipulate it. The robot also required effector and actuators to manipulate with the object. The number of required components in order to manipulate in object is much more

SIK Exercises

October 12, 2024 10:52 PM

Recommended and individual SIK exercises will be listed within this section. (The first two excercises are part of Unit 0, however their code can also be linked from my git repo)

Experiences during the projects, and appropriate visual aids will be used.

Any illustrative material that cannot be pasted within my blog will be available at (typically GIFs):
https://github.com/malcolmsdad/COMP444/tree/318fbd46aea4ad804a20a212c5b05cf5c43c1bab/Excercises

NOTE: I have over 20 years professional programming experience, and been dabbling my entire life. I will not remark on theories, practices, syntax or methods that I have already used in my past - I will however, be completely open and transparent with my exercises including mentioning silly or careless mistakes. Silly 'noob mistakes' like missing semi-colon, or an incorrect casing can happen to any developer and actually cause a lot of headaches. I feel dealing with even this most basic mistakes important practice to catch and fix.

Circuit 1A: Blinking LED

October 14, 2024 10:58 AM

(Copied from Unit 0)

I connected followed the instructions in the SIK manual for Project 1A. It was a very simple setup, I sure enjoyed the breadboard - redboard place holder, nice touch by sparkfun Electronics. Setting up the board, LED, resistors, jumpers, USB and IDE was pretty straightforward, something I've done a dozen times before. To my surprise though, the program did NOT run the first time. I immediately knew the LED was in backwards, unplugged the board, switched the LED, and boom, problem solved - a nice blinking light.

Now, I wonder what else I can do with this? How about making the light blink S.O.S.? In order to do that, I'll need the code to make a light blink, an LED, and the timings of when the light should blink on and off. I've already got 2 of 3 requirements, a quick google and I've found that: A dash should be 3 units of length, a dot is 1 unit of length, the time between letters is 1 unit of length, and time between words is 3 units of length. Easy enough to encode those variables and make some functions to represent a dot or dash. It's up to me what a unit of measure is, and my first guess was clearly wrong:

(Seizure warning - Flashing light!)
P1A- Blink SOS - too fast (seizure warning!)

But after a quick tweak of one variables, we've got a great looking S.O.S. flashing light (I'll post the code at the bottom of this posting)
https://youtu.be/VuwL4KKGUFk

That about wraps up the first entry, and project. Was a lot of fun and looking forward to doing more. As for that S.O.S. program, it sure would be neat to do the opposite project; switch the LED for a button, and get the Arduino to read button presses into Morse code - maybe another time.

Challenge 1 Git source code:
https://github.com/malcolmsdad/COMP444/tree/ec3001ec11c7f952c27cc67c8e259b25fa814379/Unit%200/C1A%20Blinked%20LED%20-%20SOS%20Blink

Challenge 2: Constant LED:
https://github.com/malcolmsdad/COMP444/tree/ec3001ec11c7f952c27cc67c8e259b25fa814379/Unit%200/C1A%20Blinked%20LED%20-%20Constant%20Blink/SIK_Circuit_1A_BlinkingLED_Constant

I then attempted the constant LED challenge and noted that approximately 12-13 millisecond delay was the limit for when the LED appeared to be constantly lit.
Clock cycles: Just out of curiousity I googled the datasheet for the Redboard to see if its clock cycle had any impact on my test. According to the datasheet the clock cycle is anywhere from 10 - 40 MHZ. According to UnitJuggler the upper limit, 40 MHZ, converts to 25 nanoseconds, and given 1 nanosecond is equal to 1 million milliseconds, there would be no impact.

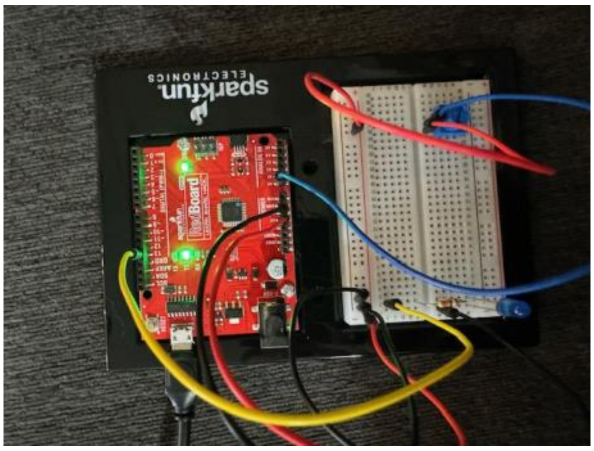
Circuit 1B: Potentiometer

October 13, 2024 11:07 AM

Circuit 1bB: Potentiometer
This task was not listed as part of the exercise list, but I wanted to give it a shot as extra practice.
I have had experience using potentiometers before. During covid, I found myself diving into several projects that I'd always wanted to try, including building a custom full size arcade emulator cabinet. This included Raspberry PI, dozens of arcade switches, USB encoder and an old heavyduty steering wheel. That steering wheel had no native drivers for linux, so I had to read the output of the potentiometer of the steeringwheel, and pedals, then translated them into respective degrees of turning, or throttle positions. The arcade was a total success, and currently I'm testing various infrared, camera based 'light guns' - lots of fun! Onto the actual exercise:

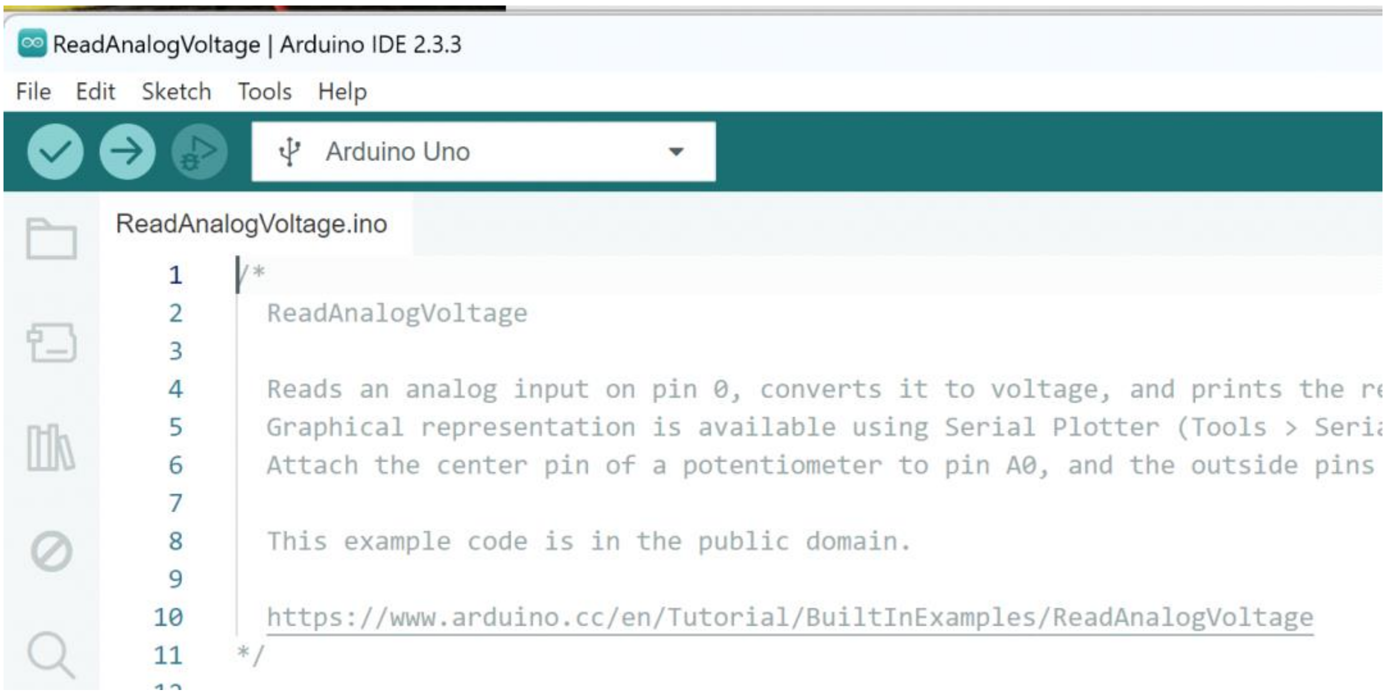
Attempt #1: Fail
Good thing too, the first attempt at running the application failed. I set up the wires as described in the instruction book, then uploaded "ReadAnalogVoltage" program, but nothing happened!

No LED:



Attempt #2 / Big Whoops
I double checked the wiring, and I must remark, my eyes are not what they used to be! I have to use a lighted magnifying to help me read the labels on the board and textbook. Pulled and rewired the potentiometer and LED, then re-uploaded. Still no flashing light! This time, I opened the serial plotter and confirmed that the values are being red from the potentiometer sensor. At this point I'm really confused, I can't find anything wrong with the wiring at all, lets check the code. Aha. Wow did I goof. The code I'm running doesn't interact with an LED at all, it simply reads the potentiometer and console outputs the values - I'm running the wrong code!

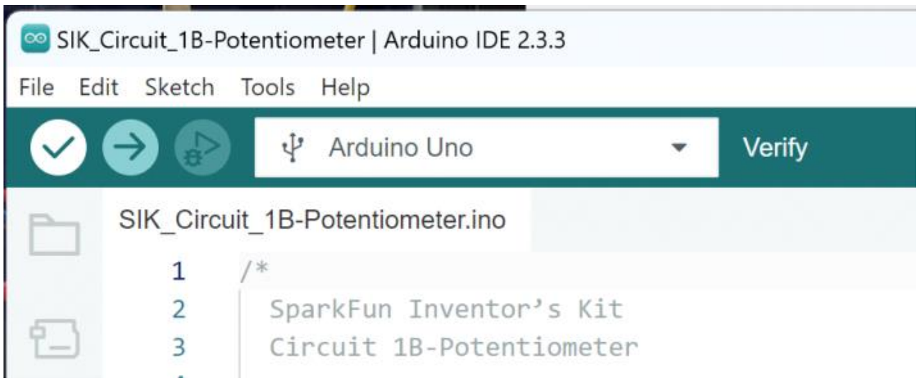
Wrong code:


ReadAnalogVoltage | Arduino IDE 2.3.3
File Edit Sketch Tools Help
Arduino Uno
ReadAnalogVoltage.ino

```
/*
  ReadAnalogVoltage

  Reads an analog input on pin 0, converts it to voltage, and prints the re
  Graphical representation is available using Serial Plotter (Tools > Seria
  Attach the center pin of a potentiometer to pin A0, and the outside pins

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/ReadAnalogVoltage
*/
```

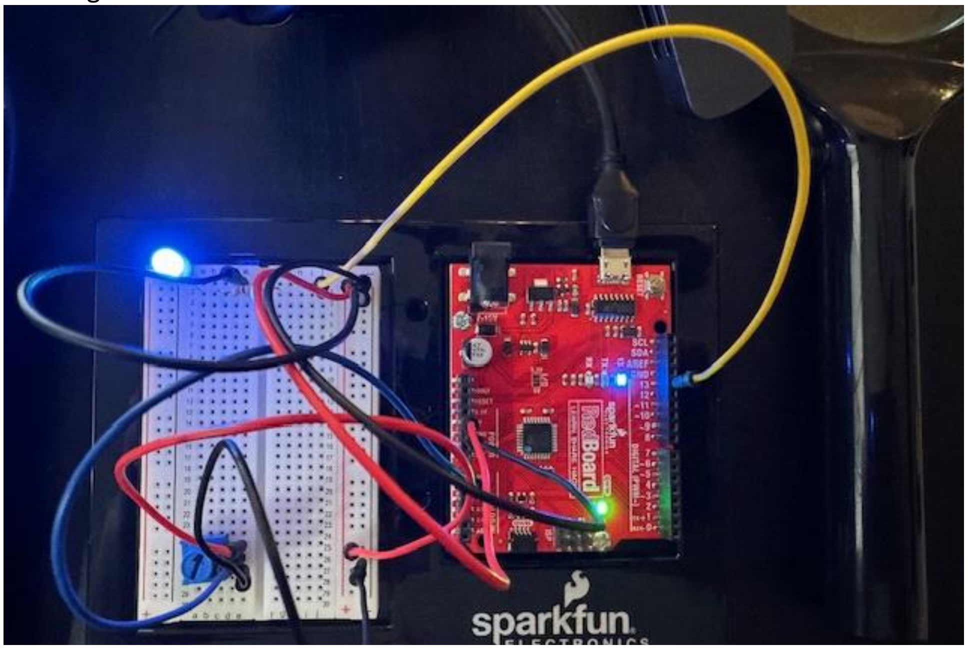
Attempt #3: Success
Uploading and running the correct code resolved the problem perfectly - the program ran perfect; LED responded to potentiometer as expected.

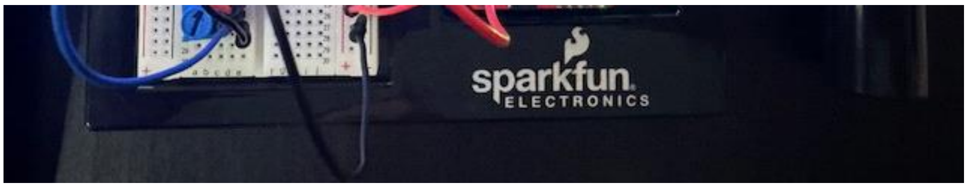
Correct code:


SIK_Circuit_1B-Potentiometer | Arduino IDE 2.3.3
File Edit Sketch Tools Help
Arduino Uno
Verify
SIK_Circuit_1B-Potentiometer.ino

```
/*
  SparkFun Inventor’s Kit
  Circuit 1B-Potentiometer
```

Flashing blue LED:




sparkfun
ELECTRONICS

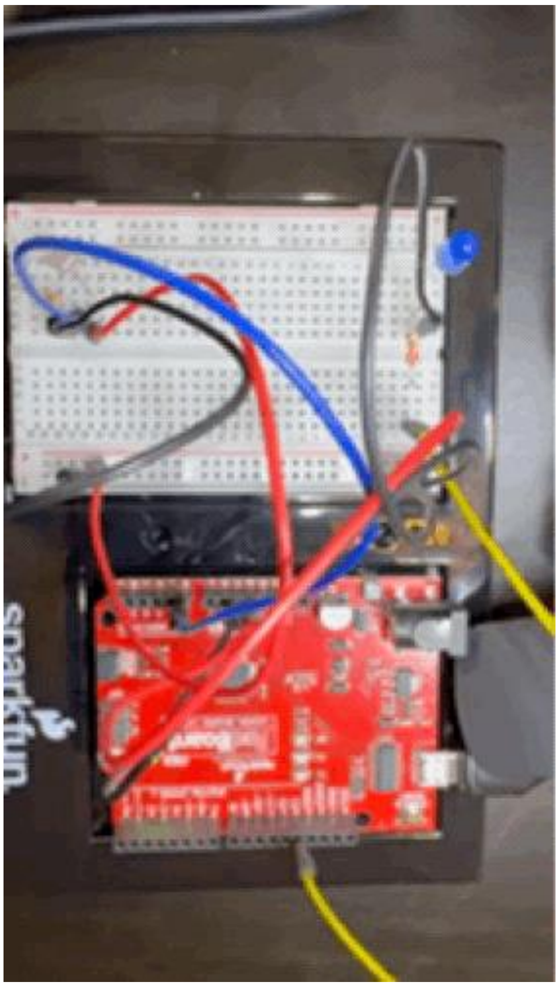
Circuit 1C: Photoresistor

October 13, 2024 11:19 AM

Time to learn how a photoresistor works.
A photoresistor changes the resistances based on how much light it receives.
Similar to potentiometer, photoresistors work by dividing voltages.
This practice introduced the concept of analog to digital conversion, as well as voltage dividing using resistance / voltage calculation.

Setup:
This was an easy one, it was very similar to the previous exercise 1b, with a swap of photoresistor for the potentiometer, and add a resistor.

Testing:
Initial test appeared to be a failure. The LED would not change when I moved my finger over the sensor. Upon reviewing the program instruction sheet, I noted that the some dim rooms may be too low for the threshold. I turned on another light, and immediately the resistor worked as expected - Success!



Project 1C
Photoresi...
(Animated GIF of LED reacting to finger)

Circuit 1D: Nightlight

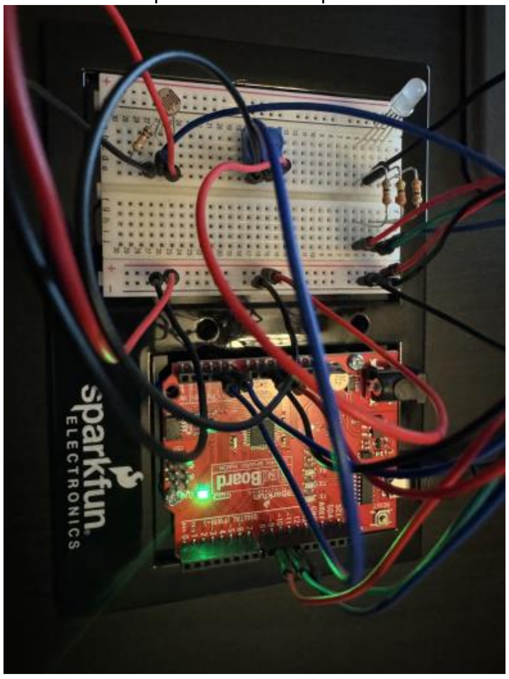
October 13, 2024 2:43 PM

A mulit-colour night light! This project introduces multi-colour LEDs (RGB LED) and pulse-width modulation (PWM).

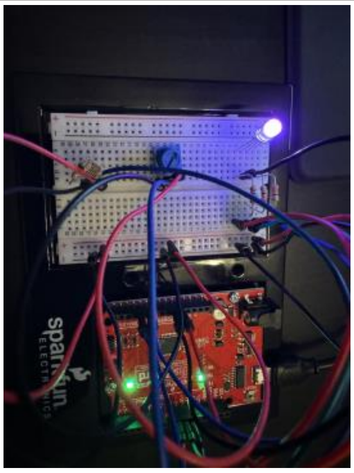
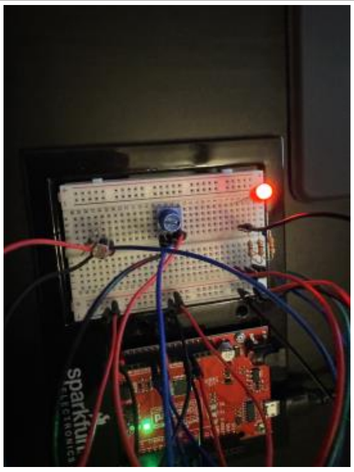
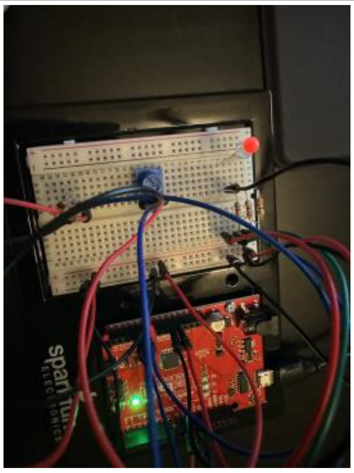
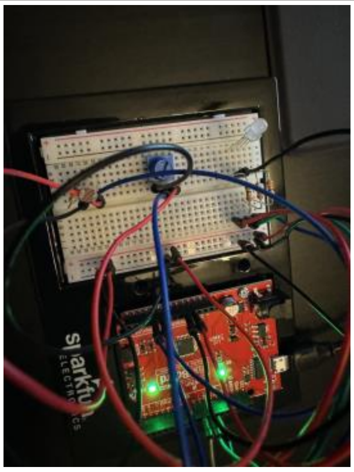
***Definition**: <u>Pulse width modulation</u> is a very powerful technique which allows analog variables to be controlled from a purely digital output, and with only a single data connection.*
Source: https://www.sciencedirect.com/topics/engineering/pulse-width-modulation

Note: Only a few pins on the RedBoard are capable (fast enough) for PWM, these are: 3, 5, 6, 9, 10, 11.

Attempt #1:
The setup was pretty straightforward, although we haven't had these many component on the breadboard at a time.
Here is the completed first attempt:



Unfortunately, after uploading, I found that the LED didn't operate exactly as suspected. The LED would light up, mostly. The potentiometer was able to adjust the colours of the LED, sort of. Here are the strange results of the LED:

			
Blue appears to work	Red appears to work	Partial red LED?	No LED at all!

Red and Blue appeared to light up, but the colours and LED performance didn't seem right. I disconnected from the USB and inspected the board. I found no issue with the jumpers, potentiometer or photosensor. I knew the orientation of the potentiometer and photosensor didn't matter, but the RGB LED orientation does matter. Upon closer inspection, I had installed the RBB backwards. That's twice I've done that, I must be more careful or get a better magnifying glass!

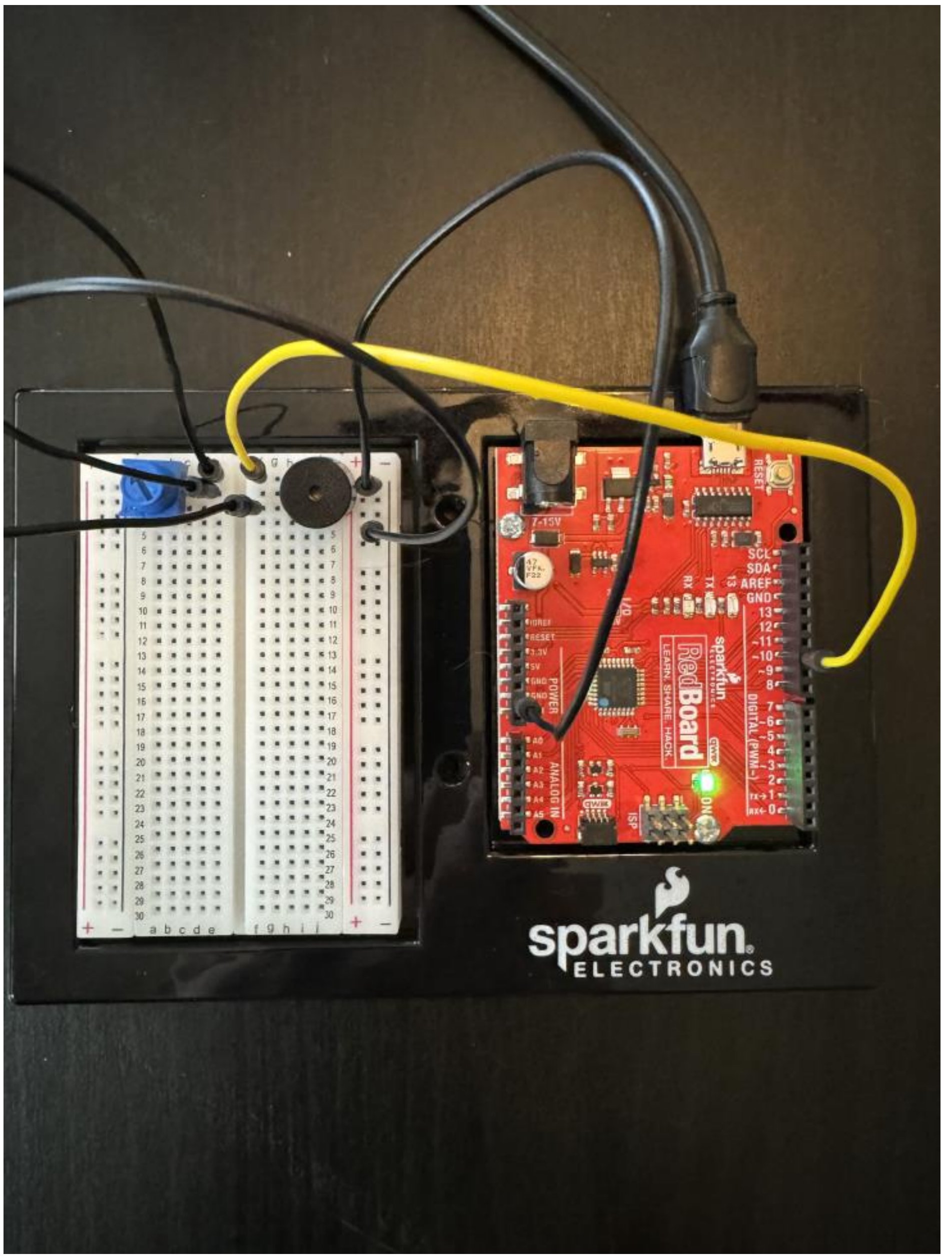
Here is the final result dialing through the colours:

(OneNote does not support GIF, click here)

Circuit 2A: Buzzer

October 13, 2024 1:47 PM

This exercise introduces the buzzer and the array coding technique. I had no problem setting up this board, and uploading the code, everything went perfect.



Coding Challenges:

Beat Length Multiplier: I found a very easy way to change the length that a beat is played. By adding a simple new variable:

```
double beatLengthMultiplier = 0.5; // multiple the lenght a tone is player for
by this..
```

Then modifying the line of code in the play() function that determines the beat length:

```
  int beatLength = 150 * beatLengthMultiplier;   //the length of one beat
(changing this will speed up or slow down the tempo of the song) ** HACK:
Alter beat length
```

And then, the song can play a very FAST version:

```
double beatLengthMultiplier = 0.1;
```

Or a very SLOW version:

```
double beatLengthMultiplier = 5;
```

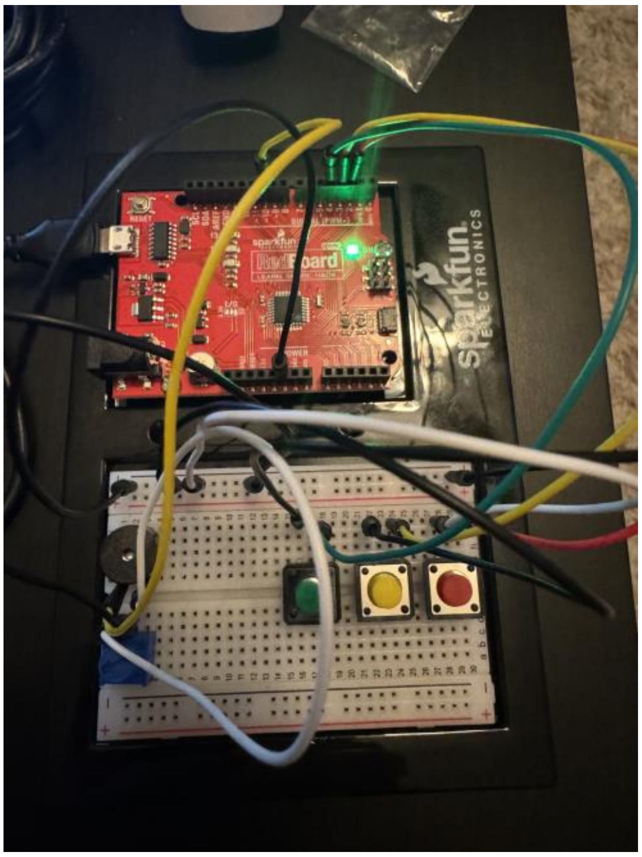
Code was tested and verified. Source code is available on git here:
https://github.com/malcolmsdad/COMP444/tree/1026481d2dadeb1114ccfb9bd9f6ef0295fe2d52/Unit%201/C2A%20Buzzer%20-%20BeatLength/SIK_Circuit_2A_Buzzer_BeatLength

Circuit 2B: Digital Trumpet

October 13, 2024 9:40 PM

The initial setup of this project was pretty straightforward, simply connected all wires as expected. This project introduces a few new concepts, including the use of buttons, pull up resisters, and binary number system.

Attempt #1: Success
The wiring and programming went smoothly, and the pietzo buzzer responded to button presses.



Coding Challenge #1: Modifying the tones. I defined a series of constant integers with the associated buzzer tone (in Hz). After uploading to the board, the program now produces different sounds for the buttons pressed.

The code is available here:
https://github.com/malcolmsdad/COMP444/tree/8f7106433b06aeed3080e606971437a1e18fdf83/Unit%201/C2B%20Trumpet%20-%20ChangeKey/SIK_Circuit_2B_DigitalTrumpet_ChangeKey

Coding Challenge #2: (Cont'd from #1):Multiple, different tones
By using several if statements, and an array I was able to analyze the buttons being collected and select a tone. The result is different tones based on the combination of buttons pressed.

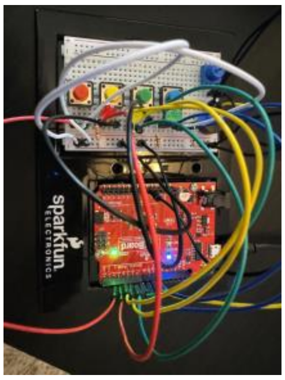
The code is available here:
https://github.com/malcolmsdad/COMP444/tree/8f7106433b06aeed3080e606971437a1e18fdf83/Unit%201/C2B%20Trumpet%20-%20MultipleTones

Circuit 2C: Simon Says

October 14, 2024 10:58 AM

A magnifying problem:
This was a very tricky one to set up. It contained 14 wires, 4 buttons and many other components. I've been finding working with the breadboard increasingly difficult for my aging eyes and oaf-like fingers, a magnifying glass helps, but often I require two hands. I found a lighted, magnifying glass visor on Amazon.ca (link here) that fit the job perfectly. I still issues navigating board when its filled with wires, but at least I can see what I'm doing!

Here is the initial board before programming:



A successful compile was a good start, the tones played perfectly, but again - for seeming the third time in a row - the LEDs didn't show. I must've reversed their polarity. That was it! Switched the LEDs polarity, and problem solved. The game works successfully; if I match 10 correct processes a winning chime is played, otherwise a losing one. I also confirmed the other behaviours such as timing out before inputs, incorrect button press (see attached GIF), the progression of correct inputs until the ultimate 10 are correct, and finally the increase in tones between sucessful rounds.

Here is an animated (no audio) GIF of a failure

I'm very familiar with much of the code, although I did learn the millis() function. The helper functions are very useful as well. I suspect that creating a library of them for later use would be handy!

Coding Challenges: 1 and 2: I was easily able to increase the difficulty of the program by increasing the array length to 15. Modifying the tones was also quiet easy, however the tones I came up with don't sound particularly nice. The code is here:
https://github.com/malcolmsdad/COMP444/tree/6251fbbb95543c5cce5f61ea51cfb016e7f6a616/Unit%201/C2C%20Simon%20Says%20-%20Difficult%20Tones/SIK_Circuit_2C_SimonSays_Difficult_Tones

I'd like to highlight the following changes:

I decided to remove the int that stored the number of rounds, instead I decided to use the length of the array to determine that. So to increase difficulty of the game, simply increase the size of the array. I've increased to 20 rounds here:

```
// This variable isn't needed! We could analyze the length to track number of required wins!
//int roundsToWin = 10;         //number of rounds the player has to play before they win the game (the array can only hold up to 16 rounds)
// the length of the button sequence will now determined the number of rounds to play
int buttonSequence[20];       //make an array of numbers that will be the sequence that the player needs to remember
```

I also created a helper function to easily determine the array length (instead of including STD):

```
// calculate the length of the array based on the size of the array allocated divided by the size of an element.
int gameSize(){
  return sizeof(buttonSequence)/sizeof(buttonSequence[0]);
}
```

And finally, I made an absolutely horrible random winning music function, it basically plays a random number of tones, a random number of times:

```
// winMusicRandom: This function will play a random tone a random number of times. Will sound very strange.
void winMusicRandom()
{
  for (int i=0; i<random(0, 8); i++){           // play a random number of tones
    tone(buzzerPin, random(1000, 3100), 150);   //play a random tone
    delay(175);
  }
}
```

Unit 2 - Introduction

October 14, 2024 3:02 PM

Unit 2: Robotic Movement 1 – Locomotion

What makes a robot move? What types of robotic movement are possible? In this unit you will explore locomotion in robots.

Learning Objectives

After completing this unit, you should be able to

- list the various mechanisms of robotic movement.
- discuss active versus passive actuation.
- describe different types of actuators.
- discuss the similarities and differences between DC motors, geared motors, and servo motors.
- describe and calculate the degrees of freedom in movement.
- discuss stability in robotic movement.
- discuss the creation of robots that move using various affectors or combination of affectors.
- employ the Arduino and the SparkFun Inventor's Kit to create circuits which control various affectors.

Readings

Please read the following chapters in the textbook:

- Arms, Legs, Wheels, Tracks, and What Really Drives Them (Chapter 4)
- Move It! (Chapter 5)

Questions to Ponder

At the end of each chapter in the assigned readings there are questions labelled "Food for Thought." Please answer these questions as best you can in your weblog, which will become part of your portfolio of competence submitted for marking during this course.

Exercises

Exercises for this unit can be found in the *Instructor's Weblog* on the Landing. Please follow along with the exercises and programs using your own Arduino kit, and keep a record of your explorations in your own weblog.

Further Readings

At the end of each chapter in the assigned readings you will find a section titled "Looking for More." While the links and readings mentioned in this section are not assigned, please feel free to examine them if you are interested, or ask questions on the Landing.

Assignment 1

It is time to submit your weblog for Assignment 1.

From <https://comp.athabascau.ca/444/r1>

Unit 2 - Instructor Blog Entry

October 14, 2024 3:04 PM

This unit begins the discussion on locomotion and robotic movement. It corresponds nicely to the SIK circuits involving motors. Work through the SIK Guide tutorials for Circuits #8 through #12, inclusive. Keep notes of your progress as before on the Landing.

Once you have completed the Circuits #1 through #12, you may wish to create a video of one or more of your favorite circuits in action if you have video recording capability. If you do, be sure to upload your video to your weblog on the Landing to share with everyone in the class.

Activities using the text Companion Workbook: Read through the text companion workbook section titled Locomotion. Answer the questions posed in this section of the workbook in your own weblog. You will not be able to do the exercises in the workbook, but compare what they are doing with the iRoomba to robot kits like Lego Mindstorms. How do they compare? How do the motors used in the iRoomba compare to the various motors in your Inventor's kit? Discuss how you might build a robot like the iRoomba using your Inventor's kit, including and additional parts you might have to acquire.

TME 1 is due at the end of this unit, so your diary on the Landing should encompass everything you have done to this point, including the tutorials through Circuit 7, inclusive.

Chapter 4

October 14, 2024 3:04 PM

This chapter discusses the various components of robotic actuators, or characteristics of movement. It first introduces effectors (legs) and actuators (muscles) and parallels them biological equivalents. IT then discusses that although the actions are similar, the way robots physically performs movements is quite different than biologics.

The chapter then discusses active vs passive actuation. Remember that actuators are the mechanisms that move the effectors. Passive actuation is then described as action of actuators using the robots potential energy, such as a soaring glider, or a walker going down a hill.
Active actuators however require some form of energy to act upon the effectors. These can include electric motors (the most common), hydraulic - relying on hydraulic fluid pressure; pneumatics - actuators relying on air pressure; finally photo-reactive materials which are mechanisms that perform physical work in response to amount of light around them. It is usually a small amount of movement - **if they do not perform movement, they are not actuators.** Also described were chemical, thermally reactive materials or piezoelectric materials, which reactive to push/pull like crystals.

Motors are then described in fair amount of detail. Discussion are the fact that motor are limited rotational movement, meaning they're very efficient for wheels. This includes description of the function and makeup of DC motors as well as their low cost, ease of availability, use or operation. Properties of DC motors are examined including ideal operations, properties of torque and rotation.

Continuing the discussion of torque leads to gearing, which can alter the torque of the motor by adjusting the output gear speed. Gearing, similar to cars is affected by the ratio of gears from the input gear to the output gear. That means that the intermeshing of the gear is important and can lead to 'backlash' or sloppy play back in forth of a motor. The multiplying effect of gears is then described by series of gears or ganged gears.

Servo motors are the next subject, as they are similar to DC motors, in that they exert rotational effort, however they servo to a specific position. Servos are often used for remote control cards and planes. Servos are typically DC motors with gearing for reduction (increased torque), position sensor on the motor shaft and electric circuit to control the motor.

The chapter then describes how the electric interaction with the servo creates a waveforms of energy. The properties of the waveform determine the duration of a pulse, or movement of the servo. This is called pulse-width modulation.

Next was degrees of freedom, controlled degrees of freedom, total degrees of freedom and how the number of controlled degrees of freedom determine if the robot is holonomic, nonholonomic or redundant. I found the example of a car being nonholonomic because it can't parallel park particularly entertaining.

Definitions

October 14, 2024 6:30 PM

Effector: any device on a robot that has an effect (impact or influence) on the environment. Legs or fingers are effectors.

Actuator: A mechanism that enables the effector to execute an action or movement. Muscles and tendons for animals.

Passive Actuation: Is the act of a mechanism using potential energy in the mechanism of the effector and its interaction with the environment instead of active power mechanisms

Electric motor: Most common robot actuator, relies on electric current

Hydraulic: actuators based on fluid pressure. Very powerful and precise but large and dangerous

Pneumatics: Actuators based on air pressure, typically very large and dangerous.

Photo-reactive Materials: Materials that perform physical work in response to light around them. NOTE: not actuators if there is no movement

Chemically reactive materials: Materials that react to chemicals, ie: fibers in acid bath causing linear motion

Thermally reactive materials: materials that react to changes in temperature

Pizoelectric materials: Materials that create electric charges when pushed/pressed

DC Motor: direct current motors convert electric energy into mechanical energy using magnets, loops of wires, and electric current.

Gears: Change the force and torque output of motors

Backlash: The amount of looseness between meshing gears is called backlash and makes the mechanism move sloppily back and forth, without turning the gear.

Gears in series; Ganged gears: gears can be organized in a series of ganged. This has the effect of multiplying their effect. Two gears of 3:1 in series results in 9:1.

Pulse-width modulation: The duration of the pulse (wave form shapes) that modulations the rotation of a servo motor.

Position control: The motor is driven so as to track the desired position at all times, resulting in a very accurate motor-driven actuator.

Torque control: The motor is driven to track the torque at all times, regardless of its specific shaft position.

Degrees of freedom: minimum number of coordinates required to completely specify the motion of mechanical system.

Translational DOF: Translational degrees of freedom are three degrees of freedom a body has in 3d space, x, y and z conventions.

Rotational DOF: Describes the other three degrees of freedom also known as rotation, including: roll, pitch and yaw.

Roll: is forward rotation

Pitch: lateral rotation when facing forward

Yaw: left / right rotation about center point when facing forward.

Controllable DOF: Each degree of freedom which has an actuator is a controllable DOF.

Uncontrollable DOF: Each degree of freedom that does not have an actuator

Trajectory: A complicated path that an object takes to get to a desired destinations

Holonomic: When controlled degrees of freedom is equal to the total degrees of freedom

Nonholonomic: When total number of controllable degrees of freedom is less than the total degrees of freedom.

Redundant: when the total controlled degrees of freedom is greater than the total degrees of freedom then the robot is redundant. A redundant robot has more ways of control than the DOT has to control.

Food for Thought

October 14, 2024 8:26 PM

How would you measure a motor's torque? What about its measurements are important for controlling robot movement. You'll know how in Chapter 7.

Torque is a measurement used to describe capabilities of rotational motors, a type of active actuator. Torque is measured in foot pounds, which means the number of pounds of force acting at a perpendicular distance of one foot from a pivot point. Basically, how hard a motor can push against something around its shaft axis. Torque can be affected by a number of things, including voltage, amperage, rotational speed (RPM), gearing and even resistance. "When a motor is spinning freely, with nothing attached to its shaft, then its rotational velocity is the highest but torque is zero." (Materic, *The Robotic Primer*, 2007, Page 34, Paragraph 2).
A faster spinning motor has much lower torque, which is sufficient for moving small items, however robots are typically larger, and require higher torque to achieve its goals, or even locomotion. Gearing can assist with that problem, by altering RPM of the motor, thusly increasing torque.

How many DOF are there in the human hand?
If you consider the wrist part of the human hand, then there are all 6 degrees of freedom within the human hand. The wrist is extremely complex and capable of performing rotational yaw, roll and pitch. The fingers make up the rest with the ability to move the finger tip translationally, forward, back, up, down and left.
At least, that's what I thought after reading the textbook. "In general, an object in 3d space has a total of six degrees of freedom" - which is the reason I said six DOF, however a quick google says I'm WAY off. According to dozens of posts online, there are 27 degrees of freedom! Those include counting many tiny movements of fingers like degrees of flexion or extension. Perhaps I need to do some more research in the COMP 444 blog group!

Chapter 5

October 14, 2024 3:04 PM

Chapter 5 discusses the various concepts of locomotion. The chapter initially defines locomotion as the act of moving a robot around its environment and then gives an example of several types of actuators / effectors, including: legs, wheels, arms, wings, flippers. Wheels are significantly easier to implement because the less DOF the less complex, also balancing is difficult to achieve within robots. Robots should be stable enough to stay up right. Stable being defined as not wobbling, leaning or following over to get the job done.
There are two types of stability, static stability (enough legs) or dynamic stability (effort expended to balance and maintain upright).
We then learn about the center of gravity with respects to stability, and if a robots COG is below its area of ground points than it'll fall over. The area covered by ground points is called the polygon of support.
Any robot that can walk while staying balance at all time is statically stable walking. The opposite is dynamic stability, where the body must actively balance in order to move. Dynamic stability is difficult to achieve and often results in a robotic terms as reverse pendulum, where the robot cannot stop and stay upright, and continually attempts to balance.
We then learn about the different properties of walking, including gait (how a robot moves, including order of lifts and lowering of legs and feet) Gait can vary hugely by robot, including robots gait abilities. Some types of gaits are examined including tripod gait where six legs are used to move a robot, and three maintain contact with ground at all times, or alternating gait.
Many other six legged robots are introduced including Gengis, a MIT created robot.
Then we move into discussing wheels, and surprisingly how they can be found in nature.
Ideal selection for robotics due to simplicity and inherently statically stable. They are not holonomic, but very close, and addition of wheels can assist with that limitation. Individual driving of wheels is called differential drive, and the ability to steer different wheels is differential steering.
Trajectory and paths are then examined. Robots getting there using 'any path' is more complex than robots following a path. Following a trajectory requires trajectory planning, aka motion planning. An involves the process of searching through possible trajectories and analyzing.
The last two sentences really stuck with me, as I'd like robot to be able to navigate: "Robots may not be so concerned with following specific trajectories as with just getting to the goal location. The ability to get to the goal is called *navigation*"

Definitions

October 14, 2024 8:49 PM

Locomotion: the way a robot moves from place to place. Derived from Latin *locus* (place) and English *motion*

Stability: A robot that does not wobble, lean or fall over easily is stable.

Static Stability: Able to stay stable without any effort

Center of Gravity: Average location of the weight of an object

Polygon of Support: The horizontal region that the COG must lie to achieve static stability

Statically Stable Walking: A robot that is able to maintain complete static stability while walking.

Dynamic Stability: a robot that is able to remain stable through effort and balancing

Invers Pendulum: A robot that continually balances to stay stable may appear like a reverse pendulum

Tripod Gait: A tripod gait is when a six legged robot leaves three legs on the ground at a time, like a tripod

Alternating Tripod Gait: A more efficient tripod gait where the legs that form the tripod alternate each step

Ripple Gait: When more than 6 legs are involved, an alternating gait is used, which creates a ripple effect, thusly called the ripple gait

Differential Drive: When wheels may be driven at driven rates or directions, the robot is said to have differential drive

Differential Steering: When wheels can be steered independently then it has differential steering.

Trajectory: The desired path the robot should follow to the goal

Trajectory planning: the act of researching all possible trajectories and identifying which is the best to achieve a certain goal.

Optimal Trajectory: The optimal trajectory is the best trajectory to follow to satisfy a certain set of conditions such as shortest, safest and most efficient.

Food for Thought

October 14, 2024 8:49 PM

How does an Web path planner, such as Mapquest or Google Maps, which finds a route for, say, a trip from your house to the airport, find the optimal path?

I can think of many possible ways that a web path planner could identify the trip from my house to the airport. The planner could do like so many children do with activity book mazes and just following random paths until we fall upon a correct one, then take note of it. Continue until all possible paths (within reason) have been identified.

It's at that point that more complicated analysis would have to be taken. Aspects such as the geometry of the route (shape, turning radius) as well as steering mechanism properties. Perhaps the user had input preference such as avoid toll highways, or gravel roads. If that's the case then those any route with those options must be removed.

Eventually a smaller subset of viable options would be reviewed and a default selected based on general user preferences or system behaviour. Perhaps those options would be presented to the user to select.

Another possible option, is if types of roads were predefined trajectories, such as arterial roads or highways, and determining optimal trajectory involved including those roads.

Perhaps the web path planner has been told to optimize fuel consumption, and, like UPS did with their ORION system, and never take left turns. (https://www.cnn.com/2017/02/16/world/ups-trucks-no-left-turns/index.html , CNN, *Why UPS trucks (almost) never turn left* February 23, 2017)

As we can see there are many possible ways that web path planners can determine the route, or suggest best route. I must say thought, ultimately it's the drivers responsibility actually follow the route:

https://github.com/malcolmsdad/COMP444/blob/dev/Excercises/Unit%202%20-%20Locomotion%20-%20Web%20Path%20Knows.gif?raw=true

SIK Exercises

October 14, 2024 3:04 PM

Circuits start at 3A: Servo Motors. Previous circuits are available in Unit 1, through 2

Circuit 3A: Servo Motors

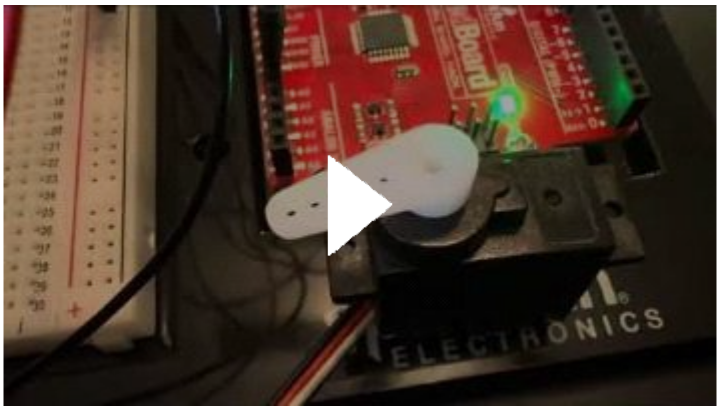
October 14, 2024 9:49 PM

Last time I used a servo motor was when I was a teenager a built a RC model airplane. I always thought they were fun little motors and could do some real interesting things with them. Who would've known 30 years later I'd be following through with that curiosity!

This project we see the PWM signal in action, using a third wire (along with power and ground) to the servo motor.
We also learn about PWM affecting duty cycle, where the length of time a PWM is referred two defines the duty cycle. The variation in the duty cycle is what tells the servo which position to go in its rotation.

This project requires we import a library, called servo.h. It provides a lot of helper functions to control servo motors, such as .write(90); - rotate 90 degrees.

Initial setup was a breeze! The addition of the lighted magnifying visor and all this practices from previous circuits, I'm getting pretty proficient at this. Unfortunately, then I ran into some serious problems. The servo that came shipped with my package is clearly defective. It worked for a second, followed my direction using the potentiometer, but then died.
The servo will respond to potentiometer inputs, however I have to give it a slight nudge in order to get it on its way. This still occasionally happens. Here is a video of that behaviour:
Faulty Servo SIK



Also worth mentioning is that the motor has fine film on the side of the servo, which appears to indicate a lubricant leak, also the motor can get quite hot to the touch when it seizes. Finally, it makes a slight buzzing sound when seized.

I will share the same YouTube link with the SparkFun team and see if they can provide any advice - maybe send a replacement? I'll post updates here.

As for the challenges.. I'm afraid that I'll have to skip two of them this time. I don't like the idea, but I feel that this seizing behaviour creates an already confusing development environment, and the first and third (Reverse Servo and Swap Sensor) may be too frustrating to test with an unreliable servo.

Adjusting the range should be something I can reliably test with the servo as it is.

Challenge #2: Change the range
I modified the code such that we can easily change the range of the servo. These two variables set the lower range and upper range of the sweep mapping:

```
int minServoPosition = 100; // the minimum degree position the servo can move
```

```
to. Default was 20.  The higher the number, the smaller the range
int maxServoPosition = 110; // the maximuim degree position the servo can move
to. Default was 160.  The lower the number, the smaller the range.
                            // The program will santize these values and no
permit under rotation, over rotation, over invalid input, ie: min > max.
```

You can try and enter invalid ranges, perhaps you want to break my servo even further? Well, I entered validations in the code to ensure to prevent this:

```
  // clean inputs variables
  int minP = min(minServoPosition, maxServoPosition); // ensure we get the
lower value of the two.. dont want no tricksters here
  int maxP = max(minServoPosition, maxServoPosition); // ensure we get the
higher value...
  // protect the device
  minP = max(20, minP);  // do not let values below 20
  maxP = min(160, maxP);  // do not let values above 160
  servoPosition = map(potPosition, 0, 1023, minP, maxP); //convert the
potentiometer number to a servo position from 20-160
```

I compiled and uploaded the program. I **believe** it worked, because the motion of the arm was relative to the motion of potentiometer, but by a much smaller ratio. The arm would only sweep a few degrees (10 by design).

The code is available here:
https://github.com/malcolmsdad/COMP444/tree/3e6842b353ed9b073e9c23bfb0ccb5186ed0712a/Unit%202/C3A%20Servo%20-%20Adjust%20Range/SIK_Circuit_3A_Servo_AdjustRange

Circuit 3B: Distance Sensor

October 14, 2024 11:14 PM

Okay! This is one I have been looking forward too! I've never used one of these but have always wanted to.

This kit uses an ultrasonic distance sensor, that uses high-pitched sounds, so high that humans can't hear them. I wonder if cats can…?
The other two concepts are datasheets and else if statements, things I'm extremely comfortable with, and use frequently.

Setup went well, this was an easy one:
https://github.com/malcolmsdad/COMP444/blob/dev/Excercises/Project%203A%20-%20Sound%20location.jpg?raw=true

Unfortunately.. even with my super vision - the echo sensor was off by a row and the program didn't work first try. I corrected this, and the application worked just great!

Corrected: https://github.com/malcolmsdad/COMP444/blob/main/Excercises/C3C%20-%201%20-%20Complete.png?raw=true

For my final project I want to combine input from two other echo devices, possibly even three. I found this cool listing on amazon and bought it: Dkardu 5PC kit When it arrives, I'll be sure to include any findings in my blog.

Challenge #1: Change unit

It was extremely easy to find a datasheet for the sensor, which provided the calculation to convert to CM. The datasheet had the same calculation for inches, so I'm confident the datasheet was accurate. I created a string unit variable and store either "cm" or "in". If the value is CM then outputs centimeters:

Serial Monitor Output:

```
const String unit = "cm";          // "cm" for centimeters, "in" for inches
```

```
21:10:30.746 -> 10.19 cm
21:10:30.777 -> 10.19 cm
Etc…
```

Or if the value is "in" it outputs inches:
Serial Monitor Output:

```
const String unit = "in";          // "cm" for centimeters, "in" for inches
```

```
21:11:27.628 -> 4.03 in
21:11:27.666 -> 4.03 in
Etc…
```

The code is available here:
https://github.com/malcolmsdad/COMP444/tree/092d7e56f3724189c8a8199937ba4a3b6786baaf/Unit%202/C3B%20Echo%20-%20Different%20Unit/SIK_Circuit_3B_DistanceSensor_DifferentUnit

Challenge #2: More Colours
Initially I was intending on doing four LED colours, but decided that was too easy, and the colours were meaningless and should be organized better. I then thought, if it's a distance sensor for warning for collisions, we should use something humans are more familiar with, like the good old traffic lights: Red, Yellow and Green. I decided to create variables to allow for a dynamic range. Ie: A safe (green) distance is 100 cm, but a dangerously close distance is 10cm. Yellow is caution, and exists directly in the middle of the range (ie: 100 - 10, or 45cm is yellow).

```
// create a distance range.. and assign colors based on that range.
float maxDistance = 12;  // show a green colour for this value
                         // show a yellow value for something in between
float minDistance = 1;   // show a red colour for this value
```

In order to do this I used my good old Paint.NET freeware image authoring software to create me a gradient with 10 points, that provided me with RGB values. I knew I was going to use a switch statement to equate a position within the range to a coordinating gradient colour, so I wanted to be able to draw RGB in a single statement.
That led me to creating a handy little analogWriteRGB(r,g,b) function.

```
// easy little helper to write to the led
void analogWriteRGB(int red, int green, int blue) {
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}
```

It all went well and here is a demo of the program altering the colour of the LED depending on distance:
https://github.com/malcolmsdad/COMP444/blob/main/Excercises/Project%203B%20-%20Distance%20Gradient.GIF?raw=true

The source code is available on my git hub:
https://github.com/malcolmsdad/COMP444/tree/092d7e56f3724189c8a8199937ba4a3b6786baaf/Unit%202/C3B%20Echo%20-%204%20Colours/SIK_Circuit_3B_DistanceSensor_FourColours

Circuit 3C: Motion alarm

October 15, 2024 10:10 PM

Setup was extremely easy - that's because it's practically identical to the previous board. Here is completed still running the last program!

Here is the assembled board:
https://github.com/malcolmsdad/COMP444/blob/main/Excercises/C3C%20-%201%20-%20Complete.png?raw=true

And the board in action:
https://github.com/malcolmsdad/COMP444/blob/main/Excercises/Project%203C%20-%20Motion%20alarm%20moving.GIF?raw=true

Unfortunately, it's at this point that my old server died. That was partially expected, as we saw that a few exercises ago. I'll be honest, I never contact sparkfun, instead I just Amazon a 2 pack for 12 bucks! It'll arrive in a few days.

For the challenge I decided to opt out of adding the mechanism, as I didn't want to risk the motor any more. I did modify the behaviour of the alarm by decreasing the pause between alarms proportionate to the proximity to the sensor. Ie: The closer you are the faster the alarm goes!

Unfortunately, the servo died before I started the challenge, so I couldn't confirm the servos behaviour, but the sound works great!

The code is available here:
https://github.com/malcolmsdad/COMP444/tree/7a60a783e5313f24e28a028ad48ff8c468fd8dca/Unit%202/C3C%20Motion%20Alarm%20-%20Urgent%20Alarm/SIK_Circuit_3C_MotionAlarm_UrgentAlarm

Circuit 4A: LCD "Hello, World"

October 15, 2024 10:14 PM

It has been a LONG time since I've done a Hello, World that I'm really interested in doing! I've never worked with a liquid crystal display. I'm not even sure how they convert data into display, so this should be good.
Liquid Crystal Displays use a special kind of crystal that turns opaque when a current is applied.

Well, that was anti-climatic. Wired it up, and it worked right away. What I learned is: thank goodness for libraries! The LiquidCrystal.h really does all the hard work for us. We just need to tell it which pins to connect to.

Coding Challenge:

For the coding challenge, I decided to combine all three tasks into one:

- Change the message
- Show hours, minutes, and seconds
- Count button presses

So in order to do that, I added a new button to the board and made it 'smart'. The program will check for both a button press and release, before it increases button count. Otherwise, the user could just hold down the button indefinitely. When the button is pressed (and released) the button count, which is displayed on screen, is increased, the language is switched to French (or back to English) AND a nice message is displayed, with a delay timer to hide the message.

Here is a gif of the demonstration:
https://github.com/malcolmsdad/COMP444/blob/main/Excercises/Project%204A%20-%20LCD%20Switch%20Language.GIF?raw=true

Circuit 4B: Temperature Sensor

October 15, 2024 10:14 PM

Introducing temperature sensors and algorithms! Important to note the 0.5 variance, mentioned in the textbox (and the datasheet, linked below). My board layout will vary slightly from that of the textbook, because I want to leave my button where it is, and temperature sensors are much easier to move around than buttons, note: the pin outs on my board will match the textbook exactly.

Yup, vey simple, and previous program is working still.
https://github.com/malcolmsdad/COMP444/blob/main/Excercises/C4B%20-%20Temperature%20Complete.png?raw=true

Challenge:

Again, I combined all three challenges into 1! This circuit monitors the potentiometer (controlling contrast) and a new photoresistor I installed. If either detect change or motion, then the display will switch to a 'bar graph' representation of the current temperature. I programmed it for 15 as the lower limit, and 40 degrees Celius as the higher. An RGB LED also display a colour coded representation of the temperature. If the temperature is too far off from room temperature, then it'll change from orange to red.

And here it is in action (sorry for the rotated GIF, I couldn't rotate it back!!)
https://github.com/malcolmsdad/COMP444/blob/main/Excercises/Project%204B%20-%20Temperature%20Challenge.gif?raw=true

And of course the source code:
https://github.com/malcolmsdad/COMP444/tree/d07bf1d8ea9a2b39cc44c2636921397d03fc9f99/Unit%202/C4B%20Temperature%20-%20Challenge/SIK_Circuit_4B_TemperatureSensor_Challenge

Notes

- Datasheet: https://sfe.io/TMP36