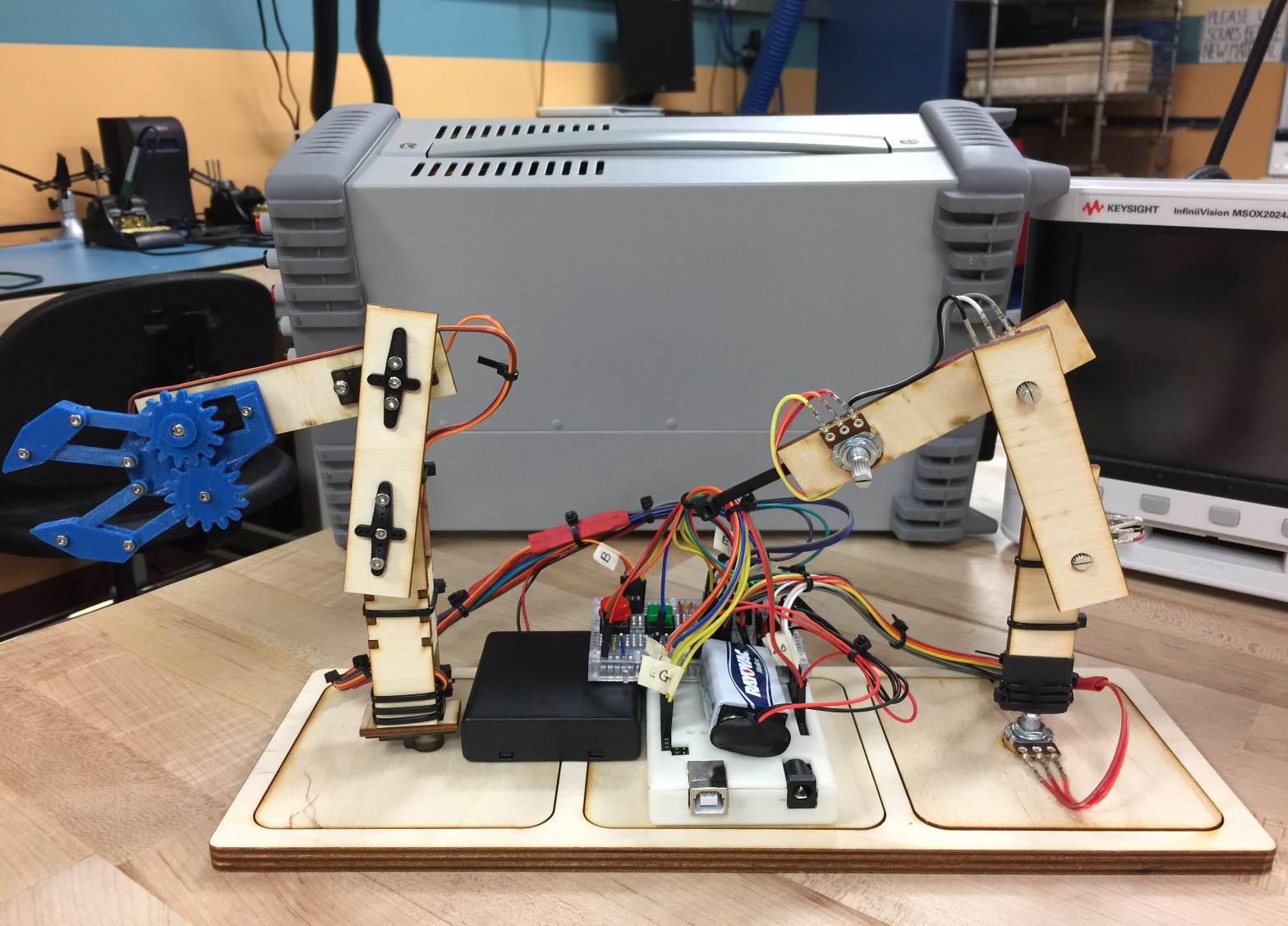
****

**Department of Electrical and Computer Engineering | Professor Truong Nguyen**



**Mini Robotic Arm – Automation**

Created by Project-in-a Box developers:

by Buu Truong, Jordi Medellin, and Auston Wang

Last Updated: 05/01/2017

Description

This project uses a robotic arm with a small gripper from your previous project, where a user motions the controlling arm, and the mechanical arm mimics the movement of the user controlled arm. Therefore, the user can perform precision tasks with the robotic arm. Most importantly, the arm can learn a set of motion by imitating the controlling arm and repeat such motion on its own.

Objective

The main goal of this project is to develop an algorithm that gives the robotic arm the ability to autonomously perform a set of motions, which is recorded during the time following the movement of the controlling arm. The algorithm requires us to write the motion of the controlling arm to the servo arm, and, at the same time, record such motion onto the Arduino memory for later used. We can achieve this by simulating a multithreading technique (proto-threading) on the Arduino.

Overview

**Challenge #1:** Setup the circuit for automation mode

**Challenge #2:** Program the automation function using protothread

**Challenge #3:** Memory Management

You Will Need

1. 1 - Fully assembled robotic arm
2. 1 - Arduino Uno
3. 1 - 40 pcs 1 pin male to male jumper cable wires
4. 1 - 4x1.5v AA battery case with on/off switch
5. 4 - AA batteries
6. 1 - 9v battery
7. 1 - 9v battery clip connector with lead wires
8. 1 - 5mm red LED
9. 1 - 5mm green LED
10. 2 - 10K ohm resistor
11. 2 - 330-ohm resistor
12. 2 - momentary button
13. 1 - set of 170 tie-points mini breadboard
14. 1 - Arduino ProtoShield prototype shield (replacement of breadboard)

Optional Supplies:

1. 1 - Arduino Uno case (recommend 3D printed case)

Challenge #1: Setup the circuit for automation mode

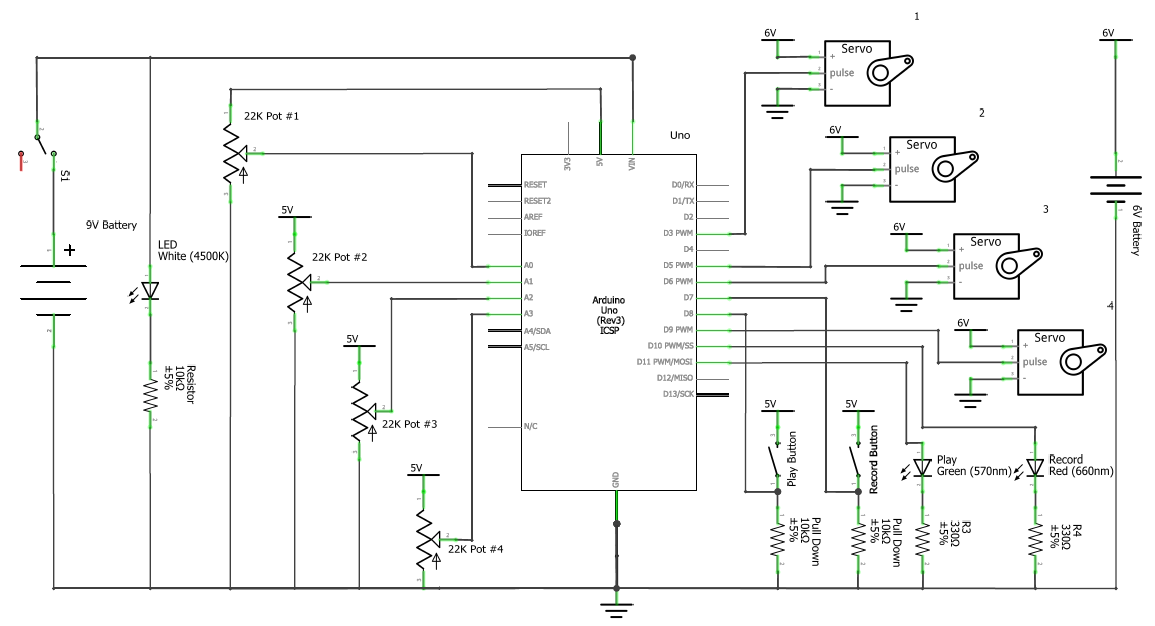
From the previous robotic arm circuit, you are going to add two extra buttons, two LEDs, and 4 resistors. One button is going to be the record button, where pressing it will put the robotic arm in recording mode. The other is the play button, where pressing it will initiate the arm to replay the set of motions that it has previously recorded in recording mode. Moreover, the red LED is a visual indicator of the recording state, where it will be on when the arm is recording, and the green LED is a visual indicator of the playing state. The two resistors of 10K ohm will be used as pull-down resistor for the buttons, and the other two 330 ohm resistors are part of the LED setup.

Question (NEED CHECK OFF FROM TA FOR POINT):

What is pull-down resistor? and why do you need it?

**Objective:** assemble the circuit board following the schematic diagram onto a breadboard, then solder all the connection onto the provided Protoshield

General Step:

1. Connect all components following the schematic below **\*\*NOTE: you are free to use different pin on the Arduino board for all connections.**
2. Skip to Challenge #2 to write and ensure the automation program is running smoothly
3. Solder all components that are on the breadboard onto the Protoshield

Challenge #2: Program the automation function using protothread

Protothread is a programing technique that allows a processor with a single core, like what you would find on Arduino board, to simulate multithreading algorithm. Before we continue our discussion on protothread, we should understand how multithreading functions. Multithreading is a programing technique that allows a processor with multi-core to process multiple programs at the same time, where each core in the processor is responsible for running one program. For example, a quad-core processor can run four different programs at the same time, parallel processing, because there are physically four different processor units on the chip. This is how we can listen to music on the computer while typing into the Microsoft word at the same time. Special exception, a single core processor with built-in program to divide up the physically single core into two or more virtual cores that can perform multithreading. This is called hyperthreading.

On the other hand, protothread do not divide the single core processor; rather, it utilizes a timer that allows it to switch (in microsecond interval) between multiple tasks and run them in a timely manner as though they are happening at the same time. For example, a protothread program that must read the temperature value from a sensor, store it on a SD card, and display the value on an LCD screen at the same time would read the temperature value at every 10 microseconds (ms), then the algorithm would switch to storing data at every 12 ms and, after that, then displaying the temperature value at every 14 ms. Most importantly, each task (thread) within the protothread program MUST run at constant time, **O(1)**, since any thread with longer time complexity would disrupt the timer. In Arduino, the millis function is used as a timer to enable protothreading. A library, TimedAction, is provided to help with implement of the millis function more cleanly and easily. \*\***NOTE: you do not need to use this library to implement the millis function as a timer, but it is highly recommended.**

**Objective:** write an algorithm that implement protothread technique that enable your robotic arm to record position values and replay those values at a press of a button. Specifically, when record button is pressed, your algorithm will write positions obtained from potentiometers to servos and store them into arrays at the same time, and when play button is pressed, your algorithm will write those stored values to the servos continuously.

General Steps:

1. Import the TimedAction library into your local Arduino library, and the file can be found in the class Google Drive folder. **\*\*NOTE: do not download the TimedAction directly off the documentation website, since it will not compile with current Arduino complier**
2. Access the documentation website to look at the example code to learn how to use the TimedAction library: <http://playground.arduino.cc/Code/TimedAction#Description>
3. Follow the flowchart to program your automation code. **\*\*NOTE: please look at the “definition of keywords in flowchart” to help with understanding of the flowchart**

**Hint:**

* What is millis() function with respect to Arduino?
* What is the different between delay() and millis()?
* How many arrays are needed to store position values for four servos?
* How do you implement multiple millis() functions?

Follow flowchart to program the automation function: play and record state initialized as false.

no

Play Mode:

false

playState on? true or false

false

Play button press?

Manual Thread():

Wait for 50ms

Read Potentiometers && Map to degree

Move Servos

false

recordState on? && usedArraySize < arrayCapacity?

Record button press?

no

yes

Set recordState == true

Turn recordLED on

Record Thread():

Store servo position into array

usedArraySize ++

true

Wait for 60ms

Turn playLED on && recordLED off

Terminate Manual Thread()

Terminate Record Thread()

Set playState == true

yes

indexCounter ++

Read recorded position from array

Move Servos

Wait for 60ms

true

true

arrayPointer <= usedArraySize?

Start here

**Definition of keywords in flowchart:**

* playState is the state of the program, where the program is currently writing the stored position values in arrays to servos.
* record button is a button that is used to initiate the process of storing position values obtained from potentiometers into arrays.
* play button is a button that is used to initiate the process of writing the stored position values in arrays to the servos.
* recordState is the state of the program, where the program is currently storing the position values obtained from potentiometers and writing those values to the servos at the same time.
* usedArraySize is a counter that keeps track of the already allocated index in an array.
* arrayCapacity is a value that indicates the maximum of elements an array can hold.
* recordLED a LED that give visual indication of the recordState.
* playLED is a LED that give visual indication of the playState.
* indexCounter is a counter that indicates the current index of an array. This is used exclusively in play mode.

**\*\*DISCLAIMER: multicore processor does implement an advanced version of protothread to execute multiple programs in the background**.

**\*\* Return to challenge #1 to finish the last step \*\***

Challenge #3: Memory Management

This automation algorithm will utilize all the Arduino Uno dynamic memory (RAM) because the position values are stored in RAM space. Therefore, the more RAM space you have, the more position values you can store, leading to a longer recording time. An easy way to expand RAM memory is efficient allocation of memory.

**Objective:** look at your algorithm, and change the **type** (unit of information) that is used to define your variables (i.e. int, short, byte)

General Steps:

1. find out the biggest whole number that position value (variable) can be
2. determine the best unit of information (type) to represent that value. For example, if the biggest whole number of your position value is 100, then you should not use an **int** type to define your position variable because an **int** is 32 bits, which can represent value up to 2,147,483,647. So, you will waste a lot of memory when using an int type to represent 100.
3. change the unit of information to the optimal type.
4. Repeat step 1 to 3 for other variables in your algorithm.

**Please email me at** [**buutruong1@gmail.com**](mailto:buutruong1@gmail.com) **for any question or concern regarding this project**