Arduino Nanocourse Day 3 Project

Behavior Box Software

Precise timing and logging of experimental events (stimuli, responses, rewards, etc.) are essential requirements when building a scientific device. Using what you've just learned about the Arduino software, evaluate and improve the performance of your behavior box.

Timing: Verify (and improve) behavior box timing

Precision

Measure the millisecond timing precision of your behavioral box including:

- Duration of the trial
- Duration of the trial start trigger (beep)
- Duration of period where a response results in a reward

Use the timing tips from today's lecture to improve the temporal performance of your box.

Reaction time

Add code to measure the reaction time in milliseconds for each trial by recording the time of the behavioral response relative to the start of the trial.

Millis rollover

How would your box handle rollover of the millis timer? Use the following code to test your box:

```
#include <util/atomic.h>

void set_millis(unsigned long ms) {
  extern unsigned long timer0_millis;
  ATOMIC_BLOCK (ATOMIC_RESTORESTATE) {
        timer0_millis = ms;
        //timer0_micros
  }
}

void setup() {
  set_millis(4294960000UL);  // set millis to ~7 seconds before rollover
  //...
}
```

Serial control: adjusting behavior box training parameters

Update your behavior box control code to allow configuration of trial parameters over the serial port.

Some parameters that you might want to configure are:

- Tone duration
- Trial time (30 seconds default)
- Response delay (3 seconds default)
- Reward duration/amount

To accomplish this you will need to produce the following:

- A serial protocol (or message format) used to specify what variable to change and what value to change it to
- Serial parsing code on the Arduino to decode or interpret messages sent from the computer
- Modified behavior box control code to use the newly configurable parameters

Serial protocol

Serial communication between two devices involves sequential transmission of bytes of information by sending a particular temporal pattern of highs and lows on a single wire. In addition to a defined hardware protocol (e.g. what voltage is ‘high’ vs what voltage is ‘low’), both devices must know the order and meaning of the bytes sent over the serial connection. Most devices you encounter will define their own serial protocol that you must follow to control and communicate with the device. We will define a protocol below that has some elements that you might encounter and is easy to use through the serial monitor provided with the Arduino IDE.

Message format

For our protocol, every message contains the following elements in order:

1. Identifier: a single letter specifying what variable we want to modify
2. Value: an integer or floating point number (as text) that we want to assign to the variable
3. Line ending: two characters that signify the end of a line (in this case the carriage return and line feed characters: \r \n)

Example message: i10\r\n
With: Identifier = i, Value = 10

Parsing serial messages

To parse messages of this format on the Arduino it is helpful to use the parseInt and parseFloat functions provided for you. Below is an example function that parses incoming Serial bytes and unpacks messages of the above format for two variables ‘my_int’ and ‘my_float’ with ids ‘a’ and ‘b’:

```
void check_serial_port() {
  if (Serial.available()) {
    char id = Serial.read();              // read the next serial byte as a char
    if (id == '\r') {                     // if id is a carriage return, do nothing
    } else if (id == '\n') {              // if id is a newline, do nothing
    } else if (id == 'a') {               // if id is 'a', assign value to my_int
      my_int = Serial.parseInt();         // read in value as an int
    } else if (id == 'b') {               // if id is 'b', assign value to my_float
      my_float = Serial.parseFloat();     // read in value as a float
    } else {                              // if any other id, report invalid identifier
      Serial.print("Invalid identifier ");
      Serial.println(id);
    }
  }
}
```

As this serial protocol expects line-ending characters, be sure the serial monitor in the Arduino IDE is setup to send both newline and carriage returns for each line. You can configure this using the drop-down menu near the bottom right of the serial monitor window next to the Baud Rate drop down. In this case, select “**Both NL & CR**”.

Modifying the behavior box code

Use and modify the above code to update your behavior box code to allow setting trial parameters over the serial connection. As a reminder, try to start ‘small’. It is a good idea to first write a test sketch that only parses serial messages, changes some variables and prints their values so that you know your serial communication is working.

Saving serial output: logging behavioral performance

Record behavioral performance measured with your new instrument.

Some metrics you might want to save are:

- Number of trials
- Outcome of each trial (was reward delivered?)

Formatting output

A common data format is ‘comma separated values’ or csv. These files can be read into most programs (Excel, Matlab, R, python, etc.). For each trial, output a single serial line that includes:

Trial number, Reward delivery (1 for reward, 0 for no reward)

For example if trial 3 resulted in a reward you should see an output that looks like this:

3,1

Saving serial output to a file

When directly connected to a computer, the Arduino appears as a serial connection. The serial monitor built into the Arduino IDE is useful for monitoring this connection but provides no way to save the output.

However, there are many ways to log serial input but one of the simplest is using a program called CoolTerm. It is available here:

http://freeware.the-meiers.org/

Once CoolTerm is installed the serial connection to the Arduino can be established by going to the Connection menu within CoolTerm and choosing Options, or instead using the Cmd+I keyboard shortcut.

Within the Options specify the settings for the serial port connected to the Arduino. Other than selecting the Port and confirming that the baud rate is set to 9600, all other settings can be left as defaults.

The connection then needs to be initiated by selecting the Connect icon from the menu bar, or using the Cmd+K shortcut.

To send a message to the Arduino, select the Send String option under the Connection menu or use the Cmd+T shortcut.

Lastly, to log the messages being sent from the Arduino choose the Capture Textfile option under the Connection menu or use the Cmd+R and specify the name and location of the text file to write to.

Please see example code shared on the class website for how to log to a file in Matlab, Python, and R.