Ecole Polytechnique de Bruxelles

University of Leeds

Johnston Lab Internship

User Guide: Devices Synchronization using Arduino MEGA controller as I2C master for data integration in tiff files

Author:
Maxime Verstraeten

Supervisor:
Jamie Johnston

September 13, 2019

Contents

List of Figures

1 Introduction

This user guide intends to provide the user with the information required to use and understand the synchronization device developed at JohnstonLab as well as giving some clues regarding troubleshooting.

The last Version of the software is available on : https://github.com/maverstr/i2cAurora

2 Objective

This device gathers from different experimental devices, either digital or analog signals, through different BNC connectors and some PS/2 connectors. It then is able to plot them in a user-friendly interface using Arduino Serial plotter to easily keep an eye on all the signals.

It also acts as an I2C master device to transmit data through ScanImage (http://scanimage.vidriotechnologies.com/display/SI2015/I2C+data+recording) which emulates an I2C slave on the computer. ScanImage can then interpret this I2C Data and put it in the description of tiff files for each frame.

This allows to have every signal values inside each frame metadata.

The μC also gets data from two optical mice that act as a motion sensor and does the computation for the path taken by the mouse.

3 Hardware

3.1 Arduino MEGA

The μC is the Arduino MEGA, chosen for the high number of analog and digital inputs as well as easy configuration and user-friendly plotting.

3.2 Enclosure

The enclosure contains the μC and hold it in place for simple manipulations. The enclosure also comes with 16 BNC holes and 2 PS/2 holes. The enclosure is chosen from RS: https://uk.rs-online.com/web/p/general-purpose-enclosures/3815164/ and CAD designed for documentation purpose and easy hole drilling.

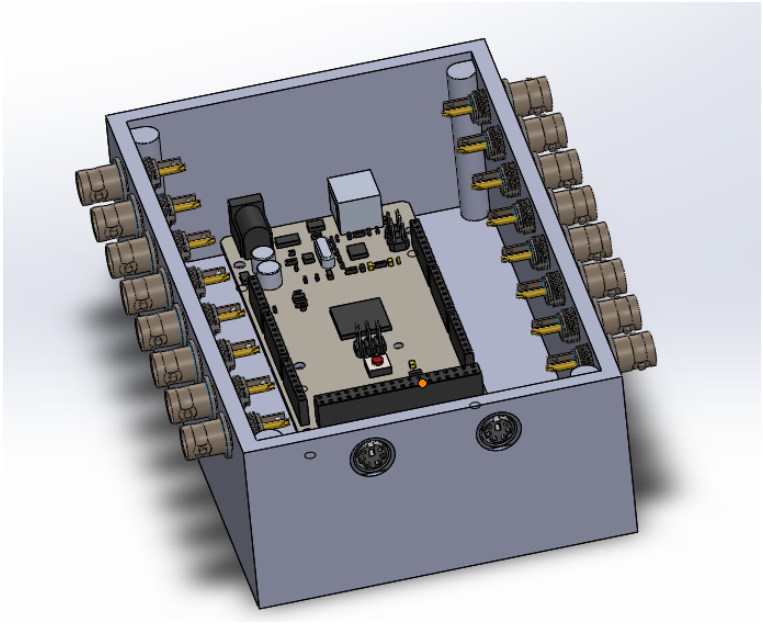


Figure 1: CAD View of the enclosure

The Plans for drilling holes are presented in appendix B and are available on scale 1:1 on demand.

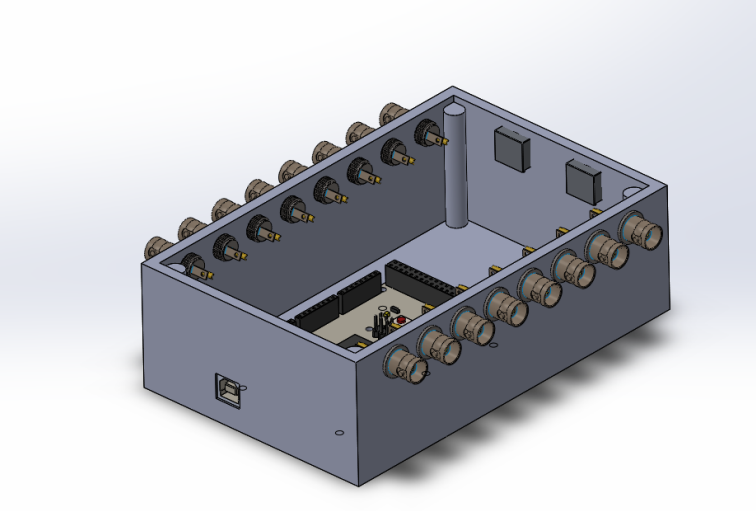


Figure 2: CAD View of the enclosure

3.3 Connectors

The connectors used to transmit data are BNC connectors. This is actually not the best option but the more practical to use as everything in the lab presents BNC connectors and BNC cables.

There are also 2 PS/2 connectors used for the motion sensor using optical mice which communicates using PS/2.

4 Software

The μC is programmed using the Arduino IDE. The code is quite simple and comes in 2 main parts: (full main code available in appendix A)

4.1 Motion sensor acquisition and computation

Using the developed libraries, the μC receives data from 2 optical mice and compute the path taken by a running mouse on a ball. This is all done in the additional .h and .cpp libraries.

```
#include "MotionSensorAcq.h"
```

The mice are setup on setup part using "motionSensorSetup()" with defined clock and data pins in variable definition. Then the data is acquired using

```
motionSensorAcq(&motionSensorValuesArray[0]);
```

which basically inserts the data into the pointed array.

4.2 Other data acquisition

The rest of the incoming data from other devices are simply processed using Arduino digital or analog read functions. Some data like the strain gauge are just re-scaled to proper values.

4.3 Plotting

The plotting is easily done by following the guidelines for Arduino Serial plotter.

4.4 I2C Communication

The integers value are first casted into String to be able to use the .toCharArray on them to form char arrays which can then be transmitted over I2C. The transmission itself is done using the Wire library.

4.5 I2C Slave

ScanImage acts as an I2C slave and is handled by MATLAB. This part will not be discussed here.

5 How to use?

Simply plug the connectors to the corresponding pins (or change them in the declaration part of the code). Use USB to power the Arduino and allow Serial communication to go through. Open the Serial Plotter using tools in Arduino IDE. You should see the relevant signal on the window. Note: the Serial rate is set to 500 000 bauds to allow fast communication of multiple signals at high sampling rate. You have to set the Serial Monitor and Plotter to this value.

The values are automatically sent over I2C when triggered externally by the Labjack. If you want constant I2C communication without requiring a trigger, overwrite cameraTriggerValue to 1. Note that this may have unexpected results.

6 Expected Results

When using the Serial plotter, the window should show the relevant signals in different colors. When opening the description of each frame of a tiff files (can easily be done using Python scripts (https://vidriotech.gitlab.io/scanimagetiffreader-python/), you should see a line of String data separated by spaces and representing the signals values in the order they were transmitted

7 Troubleshooting

7.1 Nothing appears in the Serial plotter and Serial Monitor

Check that the baud rate is set to 500 000.

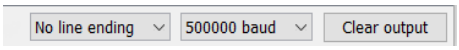


Figure 3: Set Baud rate to 500 000

7.2 Nothing plots on the Serial Plotter and the Serial monitor shows a growing single line of values

You must end your last Serial printing by a "LF" character (line feed), being "\n" or 0x0A in hex. Alternatively, you can also change your last print by:

```
Serial.print("yourData: "); Serial.println(yourData);
```

7.3 I can't send I2C data

Make sure that ScanImage is using the adress 0 as the I2C slave adress. If not, you can change the slave adress in the code (either in integer or hex):

```
Wire.beginTransmission(yourI2CSlaveAdressHere);
```

7.4 My new data doesn't appear on the plot

Check that you plugged your signal in the correct pin. Also check that this pin is selected in the code as either digitalRead() or analogRead(). Just follow the guidelines in the code for each section (pin definition, data acquisition, plotting, I2C communication). The guidelines are written again here-below:

```
/*TO ADD A NEW DATA:
 * add the corresponding pin in the code:
 *    int myDataPin = "my pin number"
 *
 * add an acquisition value variable
 *    int myDataValue;
 *
 * add an I2C array to transmit this value.
 * This array should have the size of the number of characters you want
    ↪ to
 * send +1.
 * i.e. to send 327, the array size is 4, because each char is one byte
    ↪ and
 * we want to transmit bytes.
 *    int myDataI2cArray["my required size"];
 *
 * add the pinMode in Setup()
 *    pinMode(myDataPin, INPUT);
 *
 * in the acquisition part (dataAcquisition() function)
 * add your data reading
 *  if it is a digital value:
 *    myDataValue = digitalRead(myDataPin)
 *
 *  if it is an analog value:
 *    myDataValue = analogRead(myDataPin);
 *
 *
 * add the value in plot:
 *     Serial.print("yourData: "); Serial.print(yourData); Serial.print
    ↪ (" ");
 *   Make sure that the last line of ArduinoPlotting is Serial.print(\n
    ↪ ");
 *
 * add the data in the bytes conversion function i2cDataTransform
 *    ((String)myDataValue).toCharArray(myDataI2cArray, my array size);
 *
 * add the i2C communication line inbetween Wire.beginTransmission(0x00
    ↪ );
 * and Wire.endTransmission();
 *    Wire.write(myDataI2cArray); Wire.write(" ");
  */
```

7.5 The plotting shows only a few signals and the rest are straight lines or very small

The plotter does not automatically sets a scaling. Each value share the same axis. Therefore, a very low analog value or a digital value (highest value being 1) will appear extremely small or as a straight line in comparison.

Simply multiply the value (in the Serial.print() only !) by a relevant gain.

By default, the gain is set so that max values is 5000 (legacy from strain gauge 5V max value).

A C++ Main Code

```
// /!\ OVERWRITE cameraTriggerValue for constant i2c communication without trigger
//SCB-19: red: SDA    black: SCL
//arduino: red: A4    black: A5
//mega : data: 20, clock : 21

#include "Arduino.h"
#include <Wire.h>

//#define _DEBUG_ // debug conditional compiling


/*TO ADD A NEW DATA:
 * add the corresponding pin in the code:
 *     int myDataPin = "my pin number"
 *
 * add an acquisition value variable
 *     int myDataValue;
 *
 * add an I2C array to transmit this value.
 * This array should have the size of the number of characters you want to send +1.
 * i.e. to send 327, the array size is 4, because each char is one byte and we want to transmit bytes.
 *     int myDataI2cArray["my required size"];
 *
 * add the pinMode in Setup()
 *     pinMode(myDataPin, INPUT);
 *
 * in the acquisition part (dataAcquisition() function)
 * add your data reading
 *  if it is a digital value:
 *     myDataValue = digitalRead(myDataPin)
 *
 *  if it is an analog value:
 *     myDataValue = analogRead(myDataPin);
 *
 *
 * add the value in plot:
 *      Serial.print("yourData: "); Serial.print(yourData); Serial.print(" ");
 *   Make sure that the last line of ArduinoPlotting is Serial.print(\n");
 *
 * add the data in the bytes conversion function i2cDataTransform
 *     ((String)myDataValue).toCharArray(myDataI2cArray, my array size);
 *
 * add the i2C communication line inbetween Wire.beginTransmission(0x00); and Wire.endTransmission();
 *     Wire.write(myDataI2cArray); Wire.write(" ");
  */

// ================================
// =====          GPIO          ====
// ================================
```

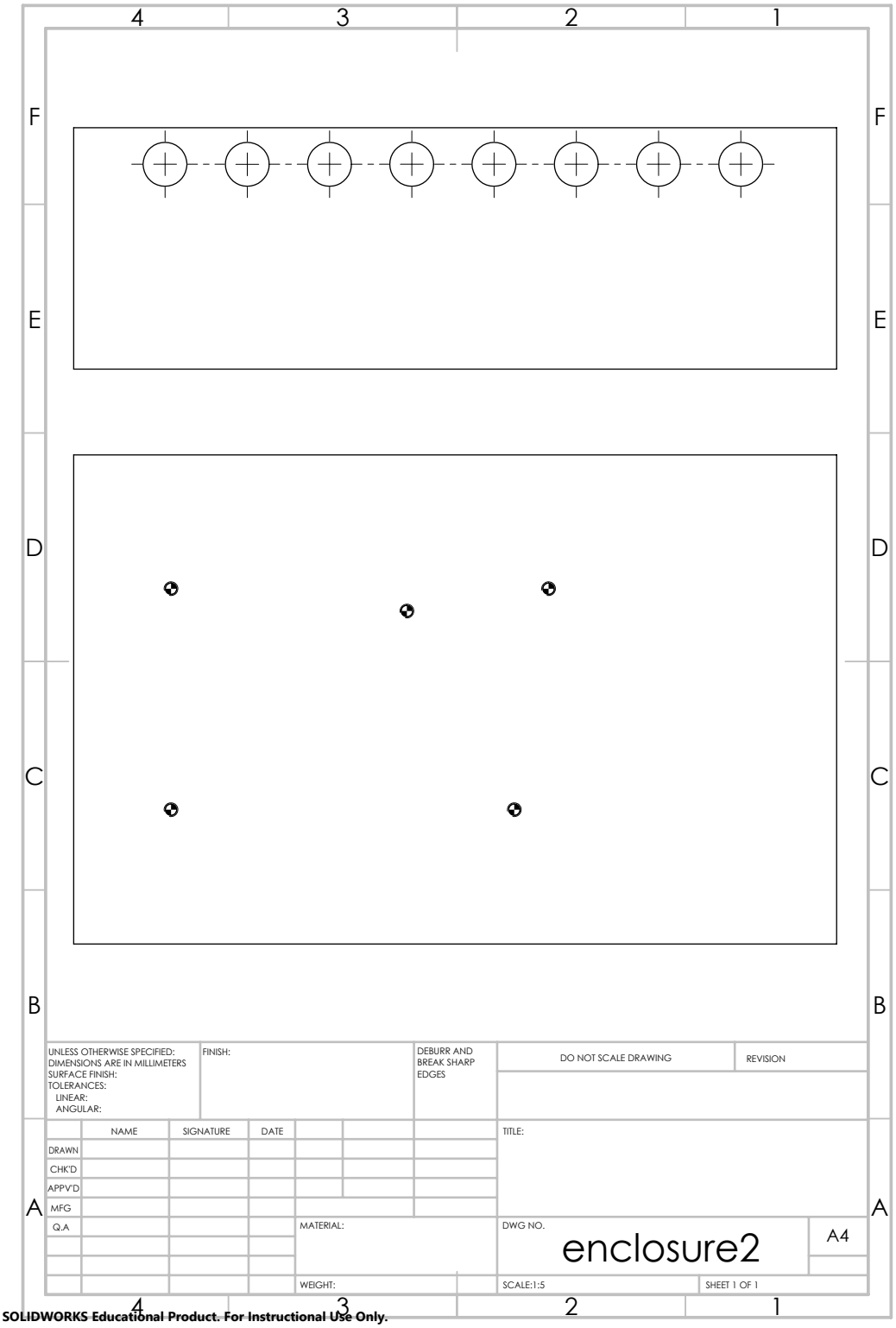
```
//motionSensor
int clockMouse1Pin = 6;
int dataMouse1Pin = 5;
int clockMouse2Pin = 3;
int dataMouse2Pin = 2;
#include "MotionSensorAcq.h"

//thermocamera
int thermRespirationPin = A5;

//Lick sensor
int lickPin = 8;
int lickValveActivationPin = 9;
int lickAirValveActivationPin = 10;

//strain gauge
int strainGaugePin = A0;

//camera trigger and stimulus
int cameraTriggerPin = 15; //
int stimulusPin = 16; //aurora 206 final valve

// ================================
// =====  Acquistion Values   ====
// ================================
int motionSensorValuesArray[3];
int thermRespValue;
int lickValue;
int lickValveActivationValue;
int lickAirValveActivationValue;
int strainGaugeValue;
int cameraTriggerValue; // == 1 when triggered
int stimulusValue;

// ================================
// =====          I2C          ====
// ================================
int DELAY_I2C = 1;
int i2cError = 0;

//I2C message
char strainGaugeI2cArray[5];
char cameraTriggerI2cArray[2];
char stimulusI2cArray[2];
//Motion sensor data
char motionXI2cArray[8];
char motionYI2cArray[8];
char motionZI2cArray[8];
//therm resp
char thermRespI2cArray[4];
//lick sensor
char lickValueI2cArray[2];
char lickValveActivationI2cArray[2];
char lickAirValveActivationI2cArray[2];

// the setup routine runs once when you press reset:
void setup() {
  pinMode(cameraTriggerPin, INPUT);
```

```
  pinMode(stimulusPin, INPUT);
  pinMode(thermRespirationPin, INPUT);
  pinMode(lickPin, INPUT);
  pinMode(lickValveActivationPin, INPUT);
  pinMode(lickAirValveActivationPin, INPUT);
  pinMode(strainGaugePin, INPUT);
  Wire.begin();
  Serial.begin(500000);
  motionSensorSetup();
}

void loop() {
  dataAcquisition();
  ArduinoPlot();
  i2cDataTransform();

  ////////////////////////////////////////////////////////////////////////
  /*OVERWRITE cameraTriggerValue so constant i2c communication without
    ↪ trigger*/
  ////////////////////////////////////////////////////////////////////////
  //cameraTriggerValue=1;

  if (cameraTriggerValue == 1) {
#ifdef _DEBUG_
    Serial.println("I2C ON, cameraTriggerValue == 1");
#endif
    i2cCommunication();

#ifdef _DEBUG_
    Serial.print("i2C error: "); Serial.println(i2cError);
#endif
    delay(DELAY_I2C);
  }
#ifdef _DEBUG_
  else {
    Serial.println("I2C OFF, cameraTriggerValue != 1");
  }
#endif
}

//
  ↪ ------------------------------------------------------------------------
  ↪

// ================================
// =====      Transmission      ====
// ================================
void i2cDataTransform() {
  ((String)strainGaugeValue).toCharArray(strainGaugeI2cArray, 5);
  ((String)cameraTriggerValue).toCharArray(strainGaugeI2cArray, 2);
  ((String)stimulusValue).toCharArray(strainGaugeI2cArray, 2);
  ((String)motionSensorValuesArray[0]).toCharArray(motionXI2cArray, 8);
  ((String)motionSensorValuesArray[1]).toCharArray(motionYI2cArray, 8);
  ((String)motionSensorValuesArray[2]).toCharArray(motionZI2cArray, 8);
  ((String)thermRespValue).toCharArray(thermRespI2cArray, 4);
  ((String)lickValue).toCharArray(lickValueI2cArray, 2);
  ((String)lickValveActivationValue).toCharArray(
    ↪ lickValveActivationI2cArray, 2);
```

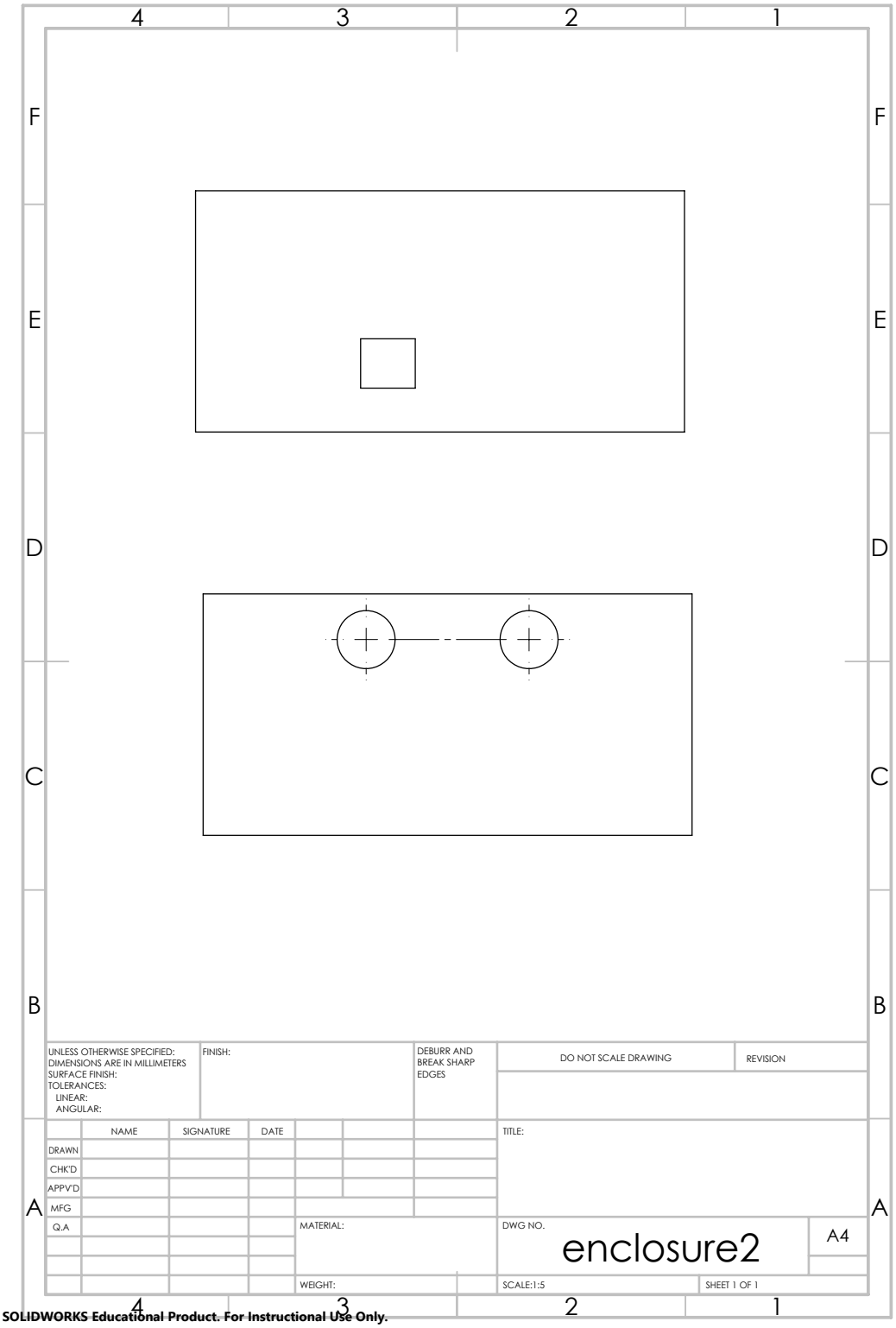
```
  ((String)lickAirValveActivationValue).toCharArray(
      ↪ lickAirValveActivationI2cArray, 2);
}

void i2cCommunication() {
  Wire.beginTransmission(0x00); // transmits to device with address 0
  Wire.write(strainGaugeI2cArray); Wire.write(" "); // sends strain
      ↪ gauge value
  Wire.write(cameraTriggerI2cArray); Wire.write(" ");// sends trigger,
      ↪ should be always 1
  Wire.write(strainGaugeI2cArray); Wire.write(" "); // sends final
      ↪ valve
  Wire.write(motionXI2cArray); Wire.write(" ");
  Wire.write(motionYI2cArray); Wire.write(" ");
  Wire.write(motionZI2cArray); Wire.write(" ");
  Wire.write(thermRespI2cArray); Wire.write(" ");
  Wire.write(lickValueI2cArray); Wire.write(" ");
  Wire.write(lickValveActivationI2cArray); Wire.write(" ");
  i2cError = Wire.endTransmission();
}

// ================================
// =====   Arduino plotting   ====
// ================================
void ArduinoPlot() {
  Serial.print("motionX: "); Serial.print(motionSensorValuesArray[0]);
      ↪ Serial.print(" ");
  Serial.print("motionY: "); Serial.print(-motionSensorValuesArray[1]);
      ↪  Serial.print(" "); //note the negative
  Serial.print("motionZ: "); Serial.print(motionSensorValuesArray[2]);
      ↪ Serial.print(" ");
  Serial.print("termResp: "); Serial.print(thermRespValue); Serial.
      ↪ print(" ");
  Serial.print("lickValue: "); Serial.print(lickValue *5000); Serial.
      ↪ print(" ");
  Serial.print("lickValveActivationValue: "); Serial.print(
      ↪ lickValveActivationValue *5000); Serial.print(" ");
  Serial.print("lickAirValveActivationValue: "); Serial.print(
      ↪ lickAirValveActivationValue *5000); Serial.print(" ");
  Serial.print("cameraTriggerValue: "); Serial.print(cameraTriggerValue
      ↪  *5000); Serial.print(" ");
  Serial.print("stimulusValue: "); Serial.print(stimulusValue *5000);
      ↪ Serial.print(" ");
  Serial.print("strainGaugeValue: "); Serial.print(strainGaugeValue);
      ↪ Serial.print(" ");

  Serial.print("\n"); //required to plot each time
}

// ================================
// =====    Acquisition     ====
// ================================
void dataAcquisition() {
  motionSensorAcq(&motionSensorValuesArray[0]);
  thermRespValue = analogRead(thermRespirationPin);
  lickValue = digitalRead(lickPin);
  lickValveActivationValue = digitalRead(lickValveActivationPin);
  lickAirValveActivationValue = digitalRead(lickAirValveActivationPin);
```

```
  cameraTriggerValue = digitalRead(cameraTriggerPin);
  stimulusValue = digitalRead(stimulusPin);
  strainGaugeValue = analogRead(strainGaugePin);
  strainGaugeValue *= (5000 / 1023.0); //in mV
}
```

B Enclosure Plan 1



C Enclosure Plan 2



References