CG1112 Engineering Principles and Practices for CEG

Week 8 Studio 1 – Jump Starting Alex

PARTIALLY GRADED STUDIO

1. INTRODUCTION

In this studio we start to put together Alex's firmware. You are provided with the main Alex firmware in Alex.ino, and some additional files for today's activities.

Alex's firmware is provided in the Arduino Wiring Language. You will earn points by converting all the Wiring Language to bare-metal. However for the final project you will not fail if you leave Alex's code alone and not convert anything to bare-metal. This is to allow you to focus on implementing cool features on Alex.

NOTE however that some activities today still require you to do bare-metal coding. Please read the studio and the source codes carefully.

For today's Studios you will be doing:

Activity 1: Setting Up Arduino Libraries on the Pi.
Activity 2: Exploring the Issues of Cross-Platform Communications (Graded).
Activity 3: Setting up and testing the wheel encoders.
Activity 4: Setting up Alex's motor control routines.

Only Activity 2 is graded. However you need to complete all activities and answer all questions so that you can finish your Alex project. In the next studio you will be controlling Alex remotely.

2. TEAM FORMATION

You will work within your **full project team** of 4 or 5 persons, not within the small sub-groups. This is a long studio so divide out the work. You may need to use both VNC and ssh at the same time to do this.

3. PREPARATION

a. Please ensure that Alex is fully assembled and functional. Also ensure that the wheel encoders are mounted. The left wheel encoder should be connected to Arduino UNO pin 2, and the right wheel encoder should be connected to pin 3.

b. The DRV8833 motor driver should be connected to the Arduino as specified in the Alex Assembly Guide:
   i) A1IN and A2IN to pins 5 and 6
   ii) B1IN and B2IN to pins 10 and 11.

   Note that these connections are DIFFERENT from what you did in the PWM studio in Week 4 Studio 1.

c. Ensure that the Remote Desktop on the Raspberry Pi is working and that you have installed Arduino on the Pi.

d. Connect to the Pi using an Ethernet cable or a hotspot.

e. Use VNC Viewer to connect to the Pi's remote desktop.

f. Connect the Arduino to the Pi if you haven't already done so.

4. SUBMISSION INSTRUCTIONS

This is a graded studio. Please submit your lab reports (AxxxxxY.docx renamed to the student number of the student submitting) to the respective folder within 15 minutes of the official end time of your respective.

This studio is worth 12 marks.

5. ACTIVITY ONE – SETTING UP ARDUINO LIBRARIES ON THE RASPBERRY PI

Step 1.

Transfer over the w8s2pi.zip file to the Pi and unzip it there. Use VNC to connect to the Pi.

You will see the following files:

i. serialize.zip (containing the serializing library for the Arduino)
ii. packet.h (We need this for Week 8 Studio 1)
iii. buffer.zip (containing the buffers library for bare-metal serial port programming)
iv. Alex.zip (containing Alex's core firmware routines)
v. serial.zip (containing source code for using the Pi's serial ports)

(Note: If unzip is not installed, install by typing "sudo apt-get install zip")

Step 2.

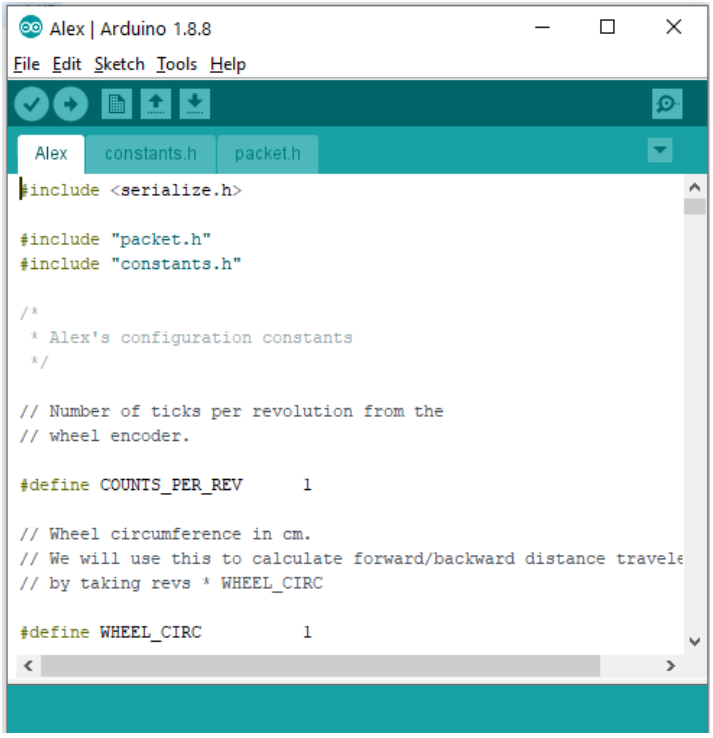
Go to the new directory, and unzip Alex's core code:

unzip Alex.zip

Unzip serial.zip as well. Do not unzip buffer.zip or serialize.zip.

Step 3.

Launch Arduino on the Pi, and open Alex.ino in the Alex directory. Your sketch should have 3 tabs:

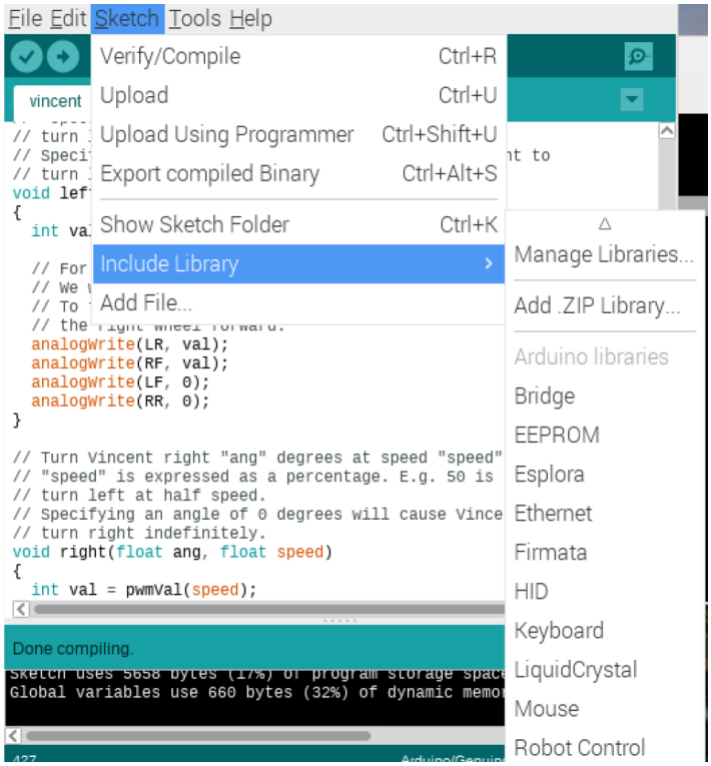


The Alex tab holds the main code, the constants.h tab holds commands you will use in Week 8 Studio 2, while the packet.h tab holds the definition for the command packets in Week 8 Studio 2. We will ignore these for now.

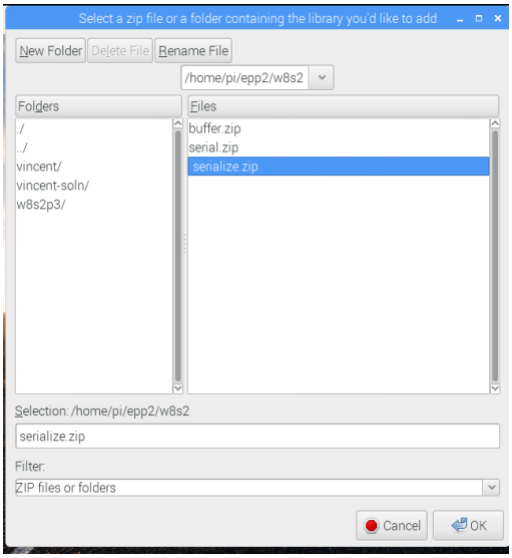
Step 4.

We will now add in the serialize library so that Alex.ino can compile properly:

a. Choose Sketch->Include Library->Add .ZIP Library…



b. Now navigate to the directory you just created, and click on “serialize.zip”:

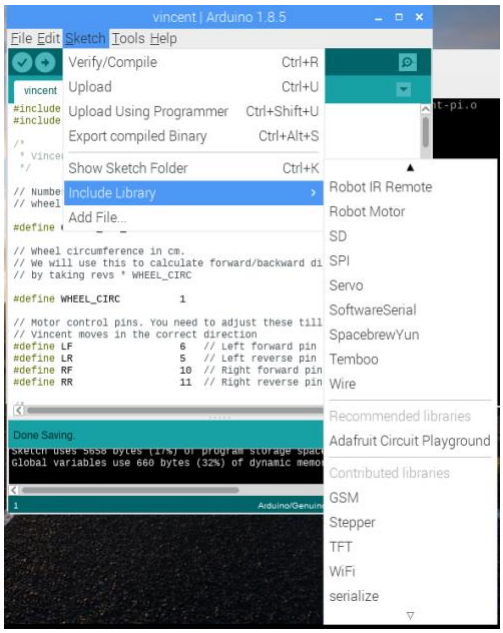


Click OK.

You will see the line “Library added to your libraries. Check “Include library” menu just under your source code window.

To include the serialize library in Alex.ino:

a. Click Sketch->Include Library->serialize (you need to scroll all the way down the list of libraries:



When you have done this you will see “#include <serialize.h>” appear at the top of the sketch.

We will not add the buffers library today.

6. ACTIVITY 2 – EXPLORING ISSUES IN INTER-PLATFORM COMMUNICATIONS (GRADED)

In this studio we will look at the issues of communicating between the Pi running on a 32 –bit ARM MCU, and the Arduino running on an 8-bit MCU.

Step 1.

Copy over the w8s2p3-pi.cpp and w8s2p3.zip files to the Pi. Unzip the w8s2p3.zip file in the new directory you made earlier, and ensure that w8s2p3-pi.cpp, serial.cpp and serial.h (unzipped from serial.zip earlier on) are in the SAME directory. This is important.

Step 2.

Open a terminal in the Pi, and use vim to open w8s2p3-pi.cpp. Set the PORT_NAME macro to the port where the Arduino is connected (For most of you this should be “/dev/ttyACM0”. For some of you it might be “/dev/ttyACM1”. Use Arduino IDE to verify.)

Now compile w8s2p3-pi using:

gcc w8s2p3-pi.cpp serial.cpp –pthread –o w8s2p3-pi

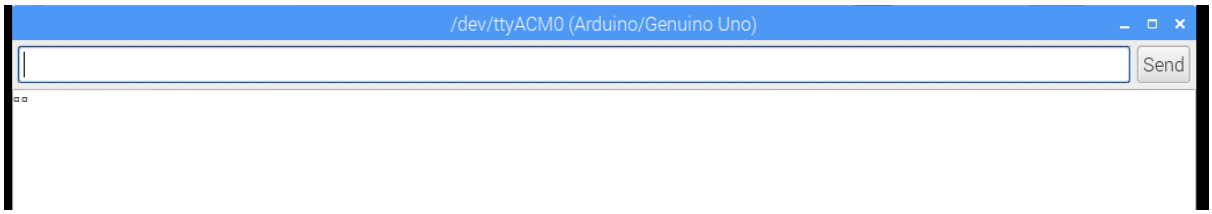
Note that the –pthread option is very important as we are using POSIX threads in our Pi side program.

(Note: If gcc is not found, install by typing “sudo apt-get install build-essential” to install gcc)

Step 3.

Open the w8s2p3.ino file in Arduino IDE (unzipped from w8s2p3.zip in Step 1) on the Pi. Upload it to the Arduino.

Now open Serial Monitor at 9600 bps. In the text box on top press “s” and the return/enter key. You will see an output similar to this:



Question 1. (2 marks)

Examine the code in w8s2p3.ino in the Arduino IDE. What do you think you are seeing in the output?

Step 4.

Close Serial Monitor. In the Pi's terminal window, type:

"./w8s2p3-pi"

Question 2. (1 mark)

Examine the values of x and y in w8s2p3.ino. Are the values the Pi is receiving correct?

Hit CTRL-C to exit w8s2p3-pi.

Step 5.

Now edit w8s2p3-pi.cpp using vim, and change the data types for x and y in the TData data structure from int to int32_t:

```
typedef struct
{
    // Uncomment the line below for step 7 of activity 4.
    //char c;
    int32_t x;
    int32_t y;
} TData;
```

Recompile w8s2p3-pi.

Step 6.
In Arduino IDE running on the Pi, do the same thing. Locate the definition for the TData data structure and change the types for x and y from int to int32_t:

```
typedef struct
{
  // Uncomment the line below for Step ??
  //char c;

  // Uncomment the line below for step ??
  //char padding[3];
  int32_t x;
  int32_t y;

} TData;
```

Now compile and upload to the Arduino, and run w8s2p3-pi in the terminal.

Question 3. (3 marks)

Is x and y printing correctly now? What do you think caused w8s2p3-pi to print the incorrect answers in Question 7 above?

Step 7.

Hit CTRL-C to exit w8s2p3-pi. Now use vim and open w8s2p3-pi.cpp, locate the line //char c; in the definition of TData, and uncomment it by removing the //:

```
typedef struct
{
    // Uncomment the line below for step 7 of activity 4.
    char c;
    int32_t x;
    int32_t y;
} TData;
```

Towards the end of w8s2p3-pi.cpp, you will see a line that says //printf("c received is %c\n", test.c);

Uncomment this line as well:

```
    // Uncomment the line below for Step 7 of Activity 4
    // in Week 8 Studio 2.
    printf("c received is %c\n", test.c);
```

In w8s2p3.ino on the Arduino IDE: Uncomment the // char c; statement by deleting the "//".

Further down in w8s2p3.ino, search for the line that shows // test.c='a'; Uncomment this line as well:

```
// Ucomment following line for Step 7
test.c = 'a';
```

Finally in loop, locate the line that says //test.c++; and uncomment this:

```
// Uncomment following line in Step 7
test.c++;
```

Recompile w8s2p3-pi in the Pi terminal, and w8s2p3.ino in Arduino IDE and upload to the Arduino. Run w8s2p3-pi.

Question 4. (3 marks)

Take note of the size of TData on the Pi and on the Arduino. Is there a difference in size? Is this difference affecting the answers for x and y?

Step 8.

Break w8s2p3-pi by pressing CTRL-C.

In Arduino IDE, edit the definition of TData by typing in "char dummy[3];" between "char c;" and "int32_t x;":

```
typedef struct
{
  // Uncomment the line below for Step 7
  char c;
  char dummy[3];
  int32_t x;
  int32_t y;

} TData;
```

Recompile w8s2p3.ino on the Arduino IDE and upload to the Arduino. In the Pi terminal window recompile w8s2p3-pi.

Run w8s2p3-pi.

Question 5. (3 marks)

Are the x, y and c fields being printed correctly now? What does this tell you about how the gcc compiler on the Pi compile TData versus the compiler on the Arduino?

7. ACTIVITY THREE – THE WHEEL ENCODERS

Step 1.

Open the Alex.ino sketch using Arduino, either on your laptop or on the Pi via VNC Viewer. We will be using Alex.ino some subsequent studios. You can use Alex.ino as the starting point for your final project.

Question 6. (Ungraded)

Implement the setupEINT function in BARE-METAL to set both INT0 and INT1 trigger on the falling edgeof pulse on pins 2 and 3 respectively. Remember to enable the interrupts in EIMSK.

Your ISR for INT0 should increment the leftTicks variable, while the ISR for INT1 should update the rightTicks variable.

Cut and paste your code into your report and explain them.

Question 7. (Ungraded)

Compile and run your code on the Arduino, and start the Serial Monitor, setting it to 9600 bps. Based on what you see in Serial Monitor, are the wheel encoders giving you false readings?

Why do you think this is happening? (Hint: As per the documentation for the wheel encoders, the Hall Effect Sensors have open collector outputs. Google for your answer and explain) (2 marks)

As mentioned above the Hall Effect Sensors are open collector devices, and to use them properly you will need to put pull up resistors on the output of the sensors.

Happily it turns out that the Atmega has built in pull-up resistors on all its GPIO pin. To activate the pull-up resistor:

- Set the pin to INPUT mode.
- Drive the pin HIGH by writing a 1 to it.

Question 7a. (UNGRADED)

Implement the "enablePullups" function in BARE-METAL by setting pins 2 and 3 to be INPUT, then driving them HIGH. Do not use Arduino functions like pinMode and digitalWrite. Cut, paste and explain your code in your report.

Question 7b. (UNGRADED)

Has the false outputs stopped? Explain why.

Now that we have our wheel encoders playing nice with us, we are going to calibrate them.

<u>Step 2</u>.

Use a marker and make a small reference mark on the rim of one of Alex's wheels, as shown below by the black dot below (circled). This reference mark is to help you know when you've turn the wheel a full circle.

Step 3.

Reset the Arduino by pressing on the RESET button. Using the mark you made as a reference, turn the wheel one full circle, taking note of the count on the screen.

Repeat this step 2 or 3 more times, resetting the Arduino each time and being as precise as possible about how you turn the wheel. Take note of the value you get.

Question 8. (UNGRADED)

Alex's gearbox has a 48:1 ratio and there are 8 magnetic poles on the disk magnets that come with the wheel encoder. What is relationship between this information and the values you got from turning the wheel one revolution? (Note: Depending on how you mounted your wheel encoders, the 48:1 ratio may not have any impact on you. If this is the case, explain why).

Step 4.

Now change the COUNTS_PER_REV constant from 1 to the value you have just found. Recompile and upload to the Arduino, start Serial Monitor, and turn the wheel exactly one revolution. Verify that the revolution count is (roughly) correct. You may still find some residual false-triggering.

Question 9. (UNGRADED)

Using a ruler or anything handy like your team-mate's finger, estimate the circumference of Alex's wheels. Write down this circumference in cm, and update the WHEEL_CIRC constant with this circumference.

Congratulations! Now you have a crude, not that reliable but still workable way of estimating how far Alex has travelled just based on the readings from the encoders! Set this aside for now, and we will be using it again in Week 9 Studio 2.

8. ACTIVITY FOUR – SETTING UP ALEX'S MOTOR ROUTINES

In this rather short activity, we will be setting up the constants for Alex's motor routines are properly set up. If you had followed the Alex Assembly Guide, this activity shouldn't take you any more than 5 minutes. Tops. Promised. Confirm + chop.

From here on, we will be programming exclusively on the Pi. So we will use Arduino IDE running on the Pi and NOT ON YOUR LAPTOP.

Step 1.

If you haven't already done so, transfer Alex.ino to the Pi and connect the Arduino to the Pi's USB port.

Connect to the Pi using VNC Viewer and start up Arduino.

Step 2.

In loop locate these lines:

```
void loop() {
// Uncommment the code below for Step 2 of Activity 3 in Week 8 Studio
// forward(0, 100);
// Uncomment the code below for Week 9 Studio 2
```

Uncomment the "forward(0,100);" instruction by removing the "//".

Compile and upload to the Arduino. Connect up the battery to the motors and switch it on. Alter the LF, LR, RF and RR constants until Alex runs in the correct direction.

Question 10. (UNGRADED)

Complete the following table in the report:

	Arduino Pin	GPIO Pin (e.g. PD5, PB3, etc)	OCxA/OCxB pin (e.g. OC1A, OC0B etc)
LF			
LR			
RF			
RR			

Step 3.

Change the leftISR and rightISR to compute the distance travelled forward. For now we will assume that Alex can only move forward. We will address this issue again in Week 9.

Question 11. (UNGRADED)

Copy your code for leftISR and rightISR to your report.