Aerospace Electronic Engineering and Computing - EEEE1006

Academic year 2023-2024

COMPUTING COURSEWORK 2 – SOLVING ENGINEERING PROBLEMS WITH A PROGRAMMING LANGUAGE

INTRODUCTION

This coursework will help you revise the Matlab knowledge gained in lectures 6-9, through an Arduino used as an I/O interface. This coursework is worth 30% of the module credits and will cover the following topics:

- 2D data: matrices, plotting and extracting data
- Functions
- Input/Output to/from files and Arduino interface
- Algorithms
- Version control

Please read the Assessment Guidelines below before attempting this coursework, as well as the mark sheet given on Moodle, so you know how the marks will be awarded.

The **submission deadline** for this coursework is: **3pm Thursday 9th May 2024.**

ASSESSMENT GUIDELINES

- All code sections are to be uploaded on your git repository as Matlab script file(s) (i.e., '.m'). There will be also .m functions and .txt files to be submitted. The submission template for this file is available on the course Moodle page, under the heading: **Matlab - Coursework 2.**
- Zip your git repository folder, containing all the requested files, and submit it to Moodle.
- **Put your name and e-mail address at the start of the script** preceded by the '%' character – the markers will use this e-mail address to give you feedback.
- For questions which require you to write text (i.e., descriptive sentences), include these as comments using the '%' character at the beginning of the line. Include comments in your script file describing what part(s) of the question you are answering (if appropriate), what the program is doing and any features of note – you will get marks for this.
- You will need or be asked to comment part of the codes while answering the questions. Don't worry, these will be uncommented and checked upon assessment.
- Save your script with a name in the following format: Firstname_Surname_studentid.m.
  For example: **Julia_Smith_13344423.m**

- Ensure that you upload before the submission deadline, as late submissions without extenuating circumstances will be penalised (minus 5 marks and additional minus 5 marks every 24 hours).
- See “Coursework 2 Markscheme.xls” on Moodle to know how marks are awarded.

A WORD OF CAUTION – Remember this is individual work, not group work. Work handed in must be entirely your own and not copied from anyone else. Discuss the coursework with your friends if needed, but answer the questions and write the code yourself. You may be awarded 0 marks if the work is copied.

ENGINEERING PROBLEM SCENARIO

You are working at an aerospace company and are responsible for ensuring the comfort of passengers on board. In particular, you are tasked with developing a system that monitors the cabin temperature to make sure it is within the range 18-24 °C, which is a typical temperature range of comfort for aircraft cabins. When in-range, a constant green light will indicate the temperature of comfort is reached, however when the temperature is below or above the range, there will be intermittent yellow or red lights flashing, respectively. It is also important to log historical temperature data during take-off, which takes about 10 minutes, to ensure the cabin temperature is not affected by the change of outside temperature.

MATERIALS

- 1 PC/laptop with MATLAB installed
- 1 Arduino UNO
- 1 USB A to USB B data cable
- 1 red LED
- 1 yellow LED
- 1 green LED
- 1 breadboard
- 1 thermistor (MCP 9700A)
- 3 220 Ω resistors
- 8 jumper wires

PRELIMINARY TASK – ARDUINO AND GIT INSTALLATION [10 MARKS]

Information and support on the following tasks is provided in project introductory sessions.

a) GitHub

- Create a GitHub account.
- Ensure git is installed on the computer being used (it is already installed on those in the computer lab)
- Create a repository where you will work and include the files for this project.
- Include the coursework template to the repository.
- Link your current MATLAB local working folder to your repository (Right click on “current folder” area - Source Control\Manage files…).
- Include all the files in this project in your repository and/or your working folder. You can work locally or in cloud, as long as your submitted file is your actual repository zipped folder, containing all your files and the “.git” folder.
- As you create your .m files, implement version control for them (Right click on the file - Source Control\Add to Git).
- As you proceed with programming, for example after writing some code or after a session of programming, commit the changes you make to your repository (Right click - Source Control\View and commit changes…).

- Optional: to synchronise your local working folder with your git repository, or, “Push”, so that your files are in cloud, (Right click - Source Control\Push).
- Keep this working practice as you proceed through your coursework.

b) MATLAB/Arduino configuration

- From the menu tab “Home”, “Add-Ons”, install the “MATLAB Support Package for Arduino Hardware” add-on.
- Configure the Arduino Hardware Support Package according to the prompts on-screen.
- Keep in mind the Arduino Add-on documentation contains much valuable information for your coursework tasks!

c) Arduino communication

- Open your submission template and start programming in the “PRELIMINARY TASK” section.
- Establish communication between MATLAB and Arduino. To do this, assign a variable *a* to the “Arduino()” function. In parentheses, specify the port the Arduino is connected to, and the type of Arduino used. You will be able to use this variable *a* to address the Arduino in any future command.
- Connect the GND and 5V pins of the Arduino to the ground bus “–“ and to the power bus “+”.
- Connect one of the LEDs to the breadboard, with the two legs on two different lines.
- Using one jumper wire, connect the line where the long LED leg sits to a digital channel of the Arduino.
- Using one 220 Ω resistor, connect the line where the short leg of the LED sits to the ground bus “-“.
- In the MATLAB command line, write “*writeDigitalPin(a,'DX',1)*”, where “*a*” is your initialised variable, X should be substituted with the number of the chosen digital bus, and 1 means “high” level. This will apply 5V tension to the LED, lightening it.
- Switch off the LED by using the “*writeDigitalPin(a,'DX',0)*” command, this will bring the applied voltage to the “low” level, which is 0 V.
- Now on your submission template create a loop that makes the LED blink. Use the two commands above in a *for* loop, interval with a *pause(1)* command, so that your program makes the LED blink at 0.5 s intervals.
- This point in time would be an appropriate one to commit your changes to your git repository – keep doing this as you move on to the next tasks!
- You have successfully made MATLAB and Arduino communicate, power an LED and write digital signals! Now you are ready to pass on to the next tasks.

TASK 1 – READ TEMPERATURE DATA, PLOT, AND WRITE TO A LOG FILE [20 MARKS]

This question lets you practice the *sprintf* and *fprintf* commands, which are used in Matlab and other programming languages. It is important to be able to format computer text output, especially when dealing with limited size displays.

Remember you can find instructions for the *sprintf* and fprintf command by typing '*doc sprintf*' and '*doc fprintf* 'in Matlab.

a) Connect the temperature sensor to the breadboard, and link its power terminals to a 5 V voltage supply and to ground, and its output terminal to one Arduino analogue channel of choice. Note that the microcontroller power supply can generate noise which disturbs the temperature reading – you may want to use a resistor to stabilise this. Include a photograph of the implementation in your report.

b) Create a variable "*duration*" with value 600 – this indicates the acquisition time in seconds. Create the arrays that will contain the acquired data. Read the voltage values from the temperature sensor approximately every 1 second for 10 minutes (the time indicated in "*duration*"). Then convert the voltage values into temperature values by using the temperature coefficient T_C and the zero-degree voltage $V_{0°C}$ which you can find in the sensor documentation. Calculate three statistical quantities over the full dataset acquired: minimum, maximum and average temperature.

c) Create a temperature/time plot with appropriate axes' titles indicating the quantity (e.g. time) and units, using the functions "*plot*", "*xlabel*" and "*ylabel*".

d) This part of the program prints the recorded cabin data to a screen. Using the *sprintf* command, make Matlab print the information in the format shown in the box below. Use the date when the data was recorded, the actual location, and substitute the 'Xs' with the values you recorded/calculated. Minute 0, Minute 1 etc mean the instant at 0 seconds, at 60 seconds etc. You should be able to format the information exactly as shown in Table 1, including:

- Separate lines for data entries.
- Spaces between lines.
- Correct alignment of the data entries using tabs.
- Any special characters (look at the documentation!).
- Two decimal digits for temperature values (does not need to work for negative numbers).

e) Using the *fopen* command, open a file called 'cabin_temperature.txt' with writing permission. Write the same data formatted as above into the cabin_temperature.txt log file. Make sure to close the file at the end of your script. Open the generated file in Matlab using *fopen* to check that the data has been written correctly.

Table 1 – Output to screen formatting example

Data logging initiated - 5/3/2024
Location - Nottingham

Minute	0
Temperature	XX.XX C
Minute	1
Temperature	XX.XX C
Minute	2
Temperature	XX.XX C
Minute	3
Temperature	XX.XX C
Minute	4
Temperature	XX.XX C
Minute	5
Temperature	XX.XX C
Minute	6
Temperature	XX.XX C
Minute	7
Temperature	XX.XX C
Minute	8
Temperature	XX.XX C
Minute	9
Temperature	XX.XX C
Minute	10
Temperature	XX.XX C
Max temp	XX.XX C
Min temp	XX.XX C
Average temp	XX.XX C

Data logging terminated

TASK 2 – LED TEMPERATURE MONITORING DEVICE IMPLEMENTATION [25 MARKS]

This section will allow you to implement an LED temperature monitoring device, where a set of LEDs will blink in real-time according to the recorded temperature. You will need to build on the breadboard configuration you have developed in Task 1.

a) Connect the three LEDs sensor to the breadboard, link their input terminals to three Arduino digital channels of choice, and their output terminal to ground. Include a photograph of the implementation in your report.

b) Now you are tasked with developing a function for displaying the temperature values and controlling the LEDs according to the measured temperature. For this, you will need to continuously monitor the temperature and to power the various LEDs according to the following directives:

- When the temperature is in the range 18-24 °C, the green LED should show a constant light.
- When the temperature is below the range, the yellow LED should blink intermittently at 0.5 s intervals.
- When the temperature is above the range, the red LED should blink intermittently at 0.25 s intervals.

Continuously means, the task should progress indefinitely rather than for a specific time. Make sure the data acquisition still follows the correct timing!

Before writing any code: Draw a flowchart to outline your implementation plan for this function and upload it to your git repository, indicating the decision points and the loops used.

c) Now, write the actual function "temp_monitor.m" (create a separate file and commit to your repository) which implements the logic you have outlined in your flowchart. Call the function from your coursework template file to use it.
NOTE: Remember that a function has a different workspace to the one where from where it is called from – you will need to pass the Arduino object you have created at the beginning of the submission template (the *a* variable) to the function!

d) The first part of the function is about monitoring and plotting the temperature. Create a single live graph that shows the recorded values of the temperature as time progresses, at intervals of approximately 1 s. For this, the usual *plot* command will create the graph, and the additional commanda *xlabel* and *ylabel* will generate proper axes titles. However, note that the dataset changes with time – and the graph needs to update and show accordingly! You may find useful other functions like *xlim* and *ylim*, which will ensure the graph shows a span which is appropriate for the dataset. As the loop progresses, the command *drawnow* will make your graph updated.

e) Now you can add some commands that will switch your LEDs on or off depending on the temperature values. To check the function is working correctly, you can hold the temperature sensor in your hand, move in a cold area e.g. close to a window, and/or change the temperature range temporarily to check it is operating correctly.

f) Note that LEDs need to blink according to a certain period, as well as the live graph has to update according to a certain period – make sure you take both of these into account for the overall temporisation of the function.

g) Write some short documentation (max 100 words) to be included in your function, which can be retrieved using the command *'doc temp_monitor'* and briefly explains the function use and purpose.

h) **After writing the code:** Draw a second flowchart in a similar fashion to the first one, which reflects the actual implementation that you have done. Do a commit to your git repository of this second, updated version.

TASK 3 – ALGORITHMS – TEMPERATURE PREDICTION [25 MARKS]

In case the cabin temperature is varying too quickly or approaching the (upper or lower) threshold of the comfort range, it would be convenient to detect this so that adjustments in the cabin temperature control system can be actioned in advance. In this task, you are asked to devise a way to alert if the temperature is varying too quickly (more than 4°C/min), and to also predict the temperature value expected in 5 minutes' time.

a) **Before writing any code:** Read through Task 3. Draw a flowchart to outline your implementation plan for this function and upload it to your git repository, indicating the decision points and the loops used.

b) Create a function temp_prediction.m that you will call from the coursework template file. Devise a way to continuously calculate how fast, in °C/s, the temperature is changing, and print the value to screen. Note that the temperature change rate over time is the derivative of the data collected. Also keep in mind that in the short-term there may be noise spikes, which are smoothened out over the medium term.

c) Now that you know how fast the temperature is changing, let the function print out to screen at every cycle the current temperature, and the temperature expected in 5 minutes (assuming the rate of change remains constant over these 5 minutes).

d) Your function should also constantly monitor the temperature and generate a constant green light if the temperature is kept stable within the comfort range. If the rate of change in temperature is greater than +4°C/min (increase), a constant red light should show instead. If the rate of change in temperature is greater than -4°C/min (decrease), a constant yellow light should show. You can hold the thermistor with your fingers to test these!

e) Write some short documentation (100 words) to be included in your function, which can be retrieved using the command *'doc temp_prediction'*.

TASK 4 – REFLECTIVE STATEMENT [5 MARKS]

Please write a reflective statement (400 words max) describing challenges, strengths, limitations, and suggested future improvements of your project. Write this as a comment within your submission template file in the appropriate section at the end.

TASK 5 – COMMENTING, VERSION CONTROL AND PROFESSIONAL PRACTICE [15 MARKS]

a) Comment the code throughout in a way that would allow another programmer to be able to understand and contribute to the code you wrote.

b) Commit the changes to your git repository as you progress in your programming tasks. There should be at least one commit of your code files for each of the four programming tasks (preferably more to ensure good backup of your code).

c) Hand the Arduino project kit back to the lecturers with all parts and in working order.

END