CTEC3426 – Telematics Project

Michael Bull, P12190492

April 2015

Project Supervisor: Alexis Le Compte, alexis.lecompte@dmu.ac.uk

Contents

List of Figures

Chapter 1

Introduction

For this project there has been software requirement of a graphical user interface (GUI) that can provide a real time display of, and provide monitoring tools for, a control area network (CAN) in the Orange Mobile Communications laboratory, located in room 1.01 of the Queens Building.

The software must provide tools to display and monitor the messages being transferred on the CAN, as well as offer the ability to issue messages to the CAN.

The software must also provide various information regarding the CAN development kit, including data for:

- The motor
- The heater
- The four switches
- The four light-emitting diodes (LEDs)
- The current temperature
- The keypad

As well as being able to manually control the development kit, the system must provide a level of automatic control. The automatic control should regulate the temperature of the development kit by following the set of rules listed below:

- A target temperature must be set, with the ability to start/stop the system from attempting to reach this target.
- The temperature must increase when the development kit is below the target temperature
- The temperature must decrease when the development kit is above the target temperature

With these automated controls built into the system, it is also important that a level of manual control is provided for the changing of the system temperature. This will be done by providing manual control for the remote motor and the remote heater.

For the data in the CAN to be monitorable and analysed by others it must be extracted into a readable format. The extracted data may then be sent over the popular text message service short message service (SMS).

Chapter 2

Specifications

1. **View messages on CAN**
   CAN unit will display incoming messages on the CAN
2. **Send messages to CAN**
   CAN unit will send outgoing messages to the CAN
3. **Display motor status**
   GUI will display status of the remote motor, either Off, Forward or Reverse
4. **Display heater status**
   GUI will display the status of the remote heater, either Off or On
5. **Display switch statuses**
   GUI will display the status of all four switches, either Off or On
6. **Display LED statuses**
   GUI will display status of all four LEDs, either Off or On
7. **Display current temperature**
   GUI will display the current temperature of the CAN unit in degrees celsius
8. **Display keypad status**
   GUI will display the status of the keypad; the currently pressed button
9. **Set automatic temperature**
   CAN unit will automatically turn on the remote heater to increase its temperature when below the target temperature, or turn off the remote heater to decrease its temperature when above the target temperature, in order to move to a temperature goal set by the user. The user may start and stop the automated temperature control at any point.
10. **Manual control motor**
    User may manually enable/disable the remote motor
11. **Manual control heater**
    User may manually enable/disable the remote heater
12. **Send SMS data**
    CAN unit will extract and format the data from the CAN and send it via SMS

This page is intentionally left blank.

Chapter 3

Message Formats

3.1 Outgoing CAN Message

For an outgoing CAN message, the identifier is 29 bits long and formatted using hexadecimal. This means that the outgoing identifier must always start with either a 0 or a 1, and followed by 7 hexadecimal digits. An example of this is the output identifier being set to 0123ABCD. After an output identifier is chosen, the board will broadcast messages starting with the chosen identifier. After the identifier follows 8 bytes of data, the first 5 of which contain the status of each component on the board. These 8 bytes of data are shown in Figure 3.1.

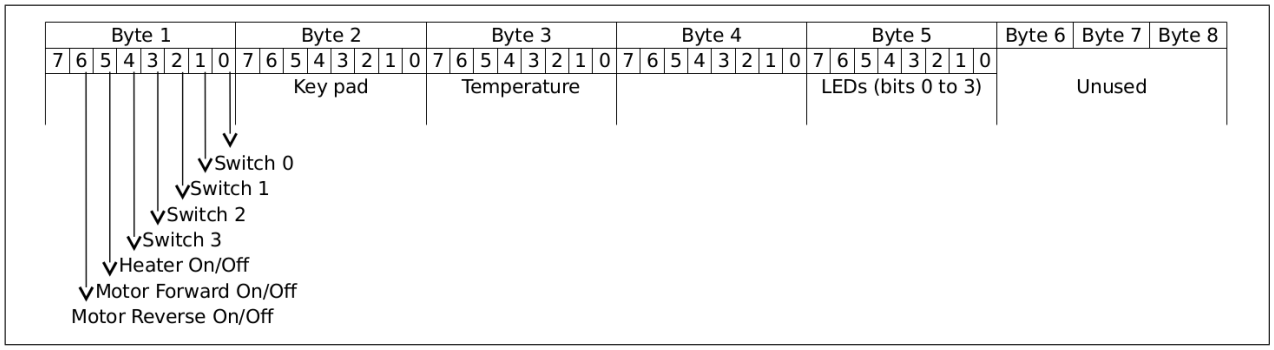

Figure 3.1: The outgoing CAN message format

3.2 Incoming CAN Message

Similar to an outgoing message, the incoming message starts with a 29 bit long identifier in hexadecimal format. Again this will be a number starting with either 0 or 1, followed by 7 hexadecimal digits. After this 8 bytes of data are sent, however only the first byte contains meaningful data regarding the status of the board. This data is shown in Figure 3.2.

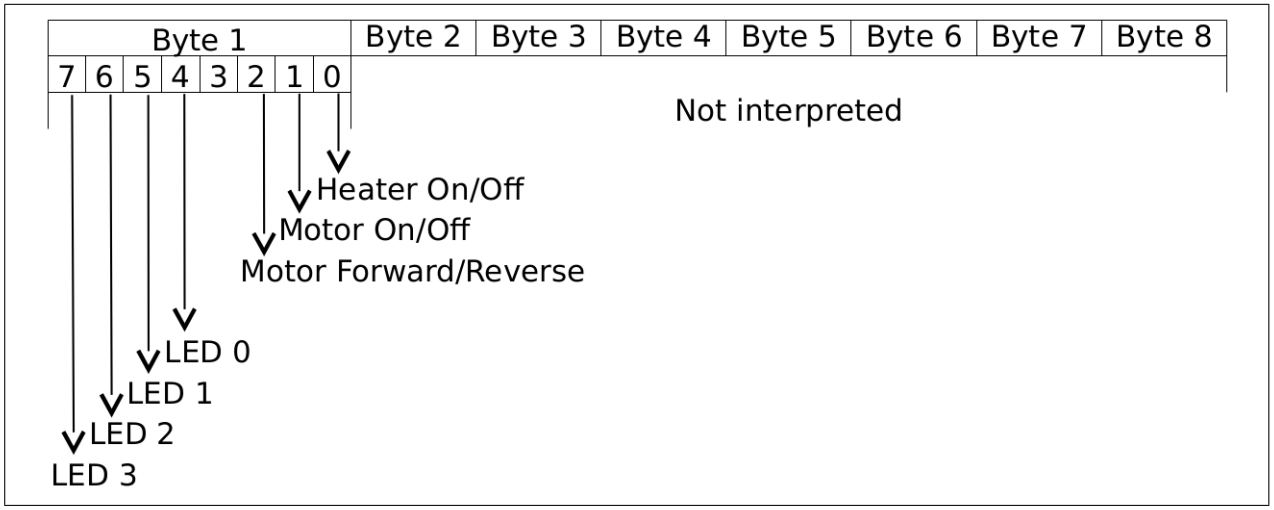

Figure 3.2: The incoming CAN message format

3.3 Outgoing SMS Message

In order to achieve the 160 character limit of SMS, the characters must be constructed from 7 bits, compared to using the 140 character limit and having 8 characters constructed from 8 bits. If we look at the ASCII table for the characters provided when using 8 bits (referred to as the 'extended ASCII table'), we can quickly identify that the extra characters will not be required for the chosen SMS format, and instead an extra 20 character limit would be preferred.

The only symbol that may be useful in the extended ASCII table is the degrees ° symbol for representing the current temperature in degrees celsius, however this is not worth sacrificing an extra 20 characters.

Figure 3.3 shows an example SMS message with the data from a CAN formatted to a 160 character limit.

Temperature: 41, Motor=On,
Heater=Off, Keypad=5,
Switches=<On,On,Off,Off>,
LEDs=<On,Off,On,Off>

Figure 3.3: The outgoing SMS message format

Chapter 4

Test Report

Test	**Outcome**
Does the unit display the motor status when it is Off?	PASS
Does the unit display the motor status when it is going Forwards?	PASS
Does the unit display the motor status when it is Reversing?	PASS
Does the unit display the heater status when it is Off?	PASS
Does the unit display the heater status when it is On?	PASS
Does the unit display the status of switch 1 when it is Off?	PASS
Does the unit display the status of switch 1 when it is On?	PASS
Does the unit display the status of switch 2 when it is Off?	PASS
Does the unit display the status of switch 2 when it is On?	PASS
Does the unit display the status of switch 3 when it is Off?	PASS
Does the unit display the status of switch 3 when it is On?	PASS
Does the unit display the status of switch 4 when it is Off?	PASS
Does the unit display the status of switch 4 when it is On?	PASS
Does the unit display the status of LED 1 when it is Off?	PASS
Does the unit display the status of LED 1 when it is On?	PASS
Does the unit display the status of LED 2 when it is Off?	PASS
Does the unit display the status of LED 2 when it is On?	PASS
Does the unit display the status of LED 3 when it is Off?	PASS
Does the unit display the status of LED 3 when it is On?	PASS
Does the unit display the status of LED 4 when it is Off?	PASS
Does the unit display the status of LED 4 when it is On?	PASS
Does the unit display the current temperature in degrees celsius?	PASS
Does the unit display the status of the keypad?	PASS
Can the user start the automated temperature control?	PASS
Can the user stop the automated temperature control?	PASS
Can a target temperature be set?	PASS
Is the heater turned off when below the target temperature?	PASS
Is the heater turned on when below the target temperature?	PASS
Can the user manually start the remote motor?	PASS
Can the user manually stop the remote motor?	PASS
Can the user manually start the remote heater?	PASS
Can the user manually stop the remote heater?	PASS
Can the unit send an SMS message of the data in the CAN?	PASS

This page is intentionally left blank.

Chapter 5

Analysis

For the product, C# was utilised as the chosen programming language to write the software in. This provided the project with an object-oriented programming language and the required tools for GUI creation. The adoption of C# resulted in the potential use of Visual Studio as an integrated development environment (IDE), the use of which provided the development phase with a visual GUI constructor tool, allowing for design of the product's front end with simple functionality such as dragging buttons and labels onto the GUI. The only downside with the adoption of this language is the lack of cross-platform compatibility with the product's deployment, as the product is compiled to an executable file that is designed to be ran on a Windows environment. Alternative languages such as Java could be used to solve the issue of cross-platform compatibility, however C# was still deemed to be most appropriate with its GUI creation tools and inclusion of classes such as the `SerialPort` class which was utilised heavily in the product's implementation. The deployment process for the application simply involves building the project's solution to an executable file, resulting in one single `.EXE` file that a user may open to be presented with the functionality they require.

Further improvements to the GUI created with the use of Visual Studio may include accessibility options, catering to users with visual impairments. A high visibility option that changes the colour scheme of the product and increases the size of the text/GUI elements on screen may assist those with visual impairments, and would be a natural extension to the product given further development time.

5.1 Remote Control

Remote controlling of a board was implemented by sending a CAN message constructed to the specification outlined in Figure 3.2 to the board. The board would then read this incoming message and act upon the data read within the constructed message. This was written with a helper method that created an array of 8 integers which would be set to either 0 or 1 based on the command and their position within the array, e.g. a command to turn the motor on would involve constructing an array in which the 7^{th} element is set to '1'. The helper method would then convert this array of 8 integers into a hexadecimal number and pad it with the expect remaining empty bytes. As the message is expected to be 8 bytes long, with only the first byte containing command information, 7 bytes were remaining. In order to write the remaining 7 bytes, they must be appended to the message in hexadecimal form,

thus 7 bytes of '00' (0 in hexadecimal) must be appended to the message to ensure a full message is being sent to the CAN.

An alternative to the use of the integer array constructed when sending commands to the CAN would be the incorporation of bitwise operations. Operations such as NOT, AND, OR, and XOR, could be applied to the data sent to the CAN to avoid manual construction of an array of integers, thus resulting in less code, however this was not implemented due to the complexity of bitwise operations. Further research and understanding of bitwise operations could prove useful if development were to continue on the project.

5.1.1 Manual Control

As the specification requires, a level of manual control is provided to the user, allowing them to enable the motor, disable the motor, reverse the motor, enable the heater, and disable the heater. They can also remotely control which LEDs are enabled. This is achieved by pressing the button associated within each control group within the GUI (Figure 5.1), e.g. pressing the 'Reverse' button under the 'Motor' controls.

Further functionality could be provided to the user, allowing them to set the speed of the motor and the power of the heater. Additional visual representations, such as a lightbulb icon illimunating, could be added to the LED controls; providing a further visual depiction of the currently enabled LEDs.

5.1.2 Automatic Control

With the user being able to manually control certain elements of the board, it was also important that automation of such control was also implemented. Once the user has set a target temperature for the board, the system will continually analyse the current temperature of the board and act upon it with regards to the target temperature. This is achieved by enabling the motor and disabling the heater when the current temperature is above the target temperature (thus reducing the board's temperature over time), and disabling the motor and enabling the heater when the current temperature is below the target temperature (thus increasing the board's temperature over time). The user may also interject with this automation at any time, as the manual control of the board is still supported whilst the board continues to aim for a target temperature.

This level of automation provides simple functionality, however alternatives were considered in the form of fuzzy logic and hysteresis. As the system does not read the decimal point of the temperature in celsius, the temperature may often fluctuate around the target temperature; sporadically enabling and disabling the motor and heater as the temperature continually moves above and below the target temperature. This fluctuation will result in the motor and heater constantly changing their states, thus requiring further power and energy from the board.

This could be fixed with two approaches; the first of which is hysteresis. Providing the system with a lower and upper bound for which the temperature should fall into allows the system to enable/disable the motor and heater based on the temperature either falling above or below the defined range of accepted temperatures. This results in the motor and heater

states depending upon the history of the temperature; thus providing the system with a level of hysteresis. The result of this prevents the heater and motor constantly switching on and off as the board's current temperature drifts around a specific target temperature, as such changes to the motor and heater will only occur once the board's current temperature either enters or leaves the lower or upper bound of a range of defined temperatures.

The second approach considered was the utilisation of fuzzy logic. A calculation may be made within the system that compares how close the current temperature is to the specified target temperature, the result of which will then be fuzzified into a value that represents how much the board falls into the 'cold' and 'hot' fuzzy sets. These fuzzy sets represent the states in which the board is too cold (and should therefore turn the heater on and motor off), and is too hot (and should therefore turn the heater off and motor on). The fuzzified values can then be placed into a defuzzification method which converts each value into an output that can be sent to the board, e.g. converting a value calculated as belonging to the 'cold' set by 50% into turning the heater on by 100% and disabling the fan. The incorporation of this fuzzy inference system would provide the system with smooths transitions around the target temperature, but would also require the utilisation of the power settings for the motor/heater motor to result in smooth transitions that do not instantly change it from on to off, but instead gradually decrease in power.

Both approaches would also mean the system could incorporate the decimal point of the temperature in celsius, without the issue of the current temperature fluctuating around the target temperature. With added precision, such drifting around the set point would increase the issue of the motor/heater constantly changing on/off, however, the utilisation of hysteresis or fuzzy logic would avoid this problem and provide the system with further level of precision with regards to automatic temperature control.

Providing the issues detailed above were addressed, further work could then be pursued to provide a greater level of functionality with regards to monitoring the automatic temperature control. Saving a limited amount of temperature data to the program's memory would allow it to construct a graph representing the change in temperature of the board over a certain period of time. This would provide an improved visual depiction of how the automated control is affecting the board over a certain amount of time, allowing the user to monitor and manage the board more effectively.

5.2 Effectiveness

Overall the implementation fulfils every requirement outlined in the specification, however also leaves plenty of oppurtunity for further development to enhance future revisions of the product. Further research into subjects such as hysteresis and fuzzy logic could greatly improve the efficiency of the product, ensuring that power is not wasted as the automatic temperature control experiences fluctuation around the set point, and further development time may be used to provide the user with extended functionality such as the ability to monitor the board's temperature over a period of time. With none of this functionality outlined in the specification it was also important to ensure that development time was focused on the core functionality that was required by the user, and as such the specification has been effectively fulfiled.

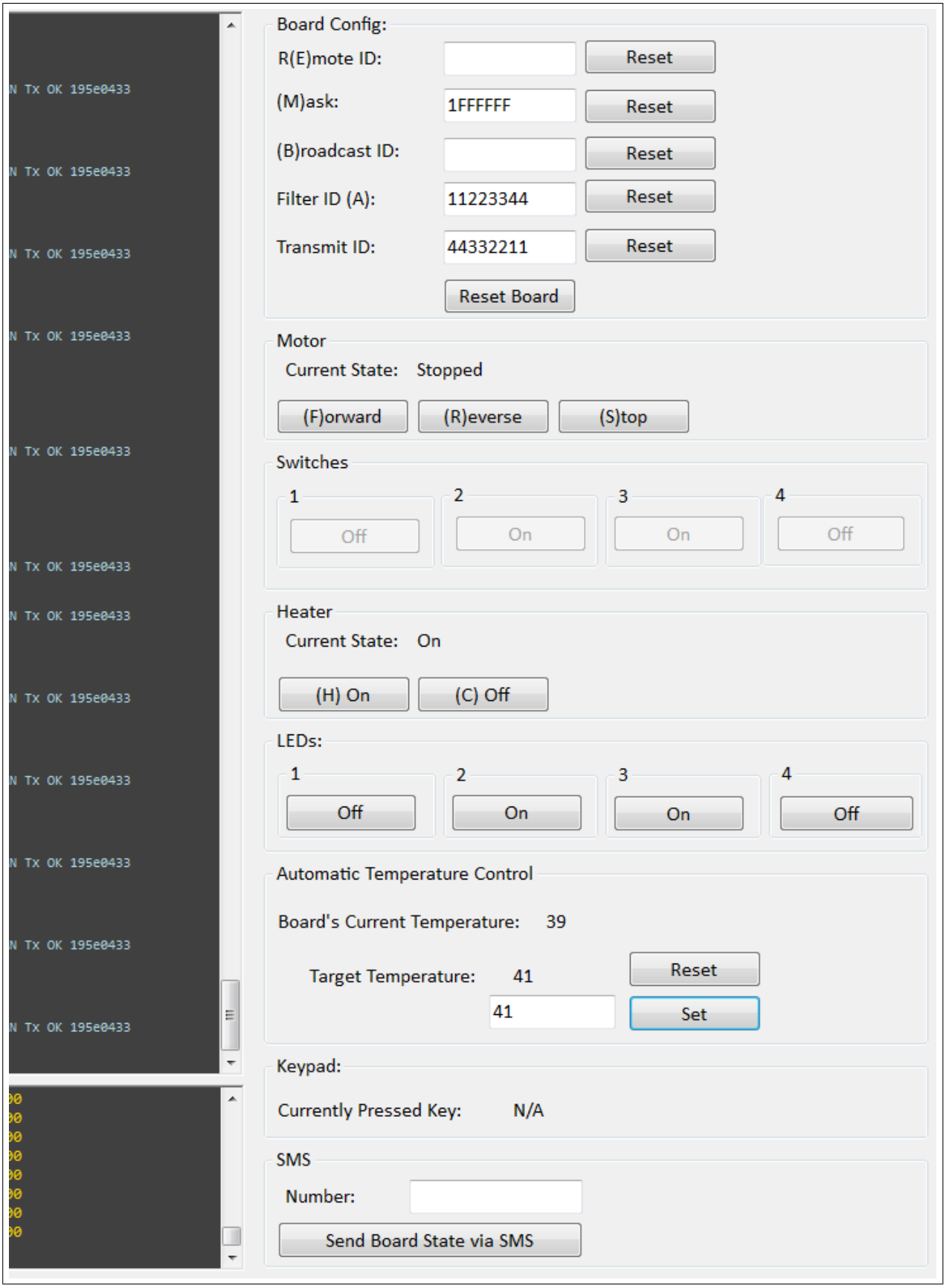

Figure 5.1: Screenshot of the state of a board displayed in the GUI

Chapter 6

User Guide

To enable control of a remote board, the program must be ran on two systems, each with a board connected to it. Once the programs are up and running, a configuration process must take place for both boards.

6.1 Configuration

Step 1. Identify which board you wish to control and which you wish to send commands from. For this example we will have **Board B** controlled by **Board A**.

Step 2. Set up **Board B** to broadcast its current state on the CAN by entering an 8 digit hexadecimal number into the 'Broadcast ID' text field.

As you type it will update the board with this broadcast ID, there is no need to click any GUI elements for it to start using the broadcast ID you typed.

For this example, we will set **Board B** to broadcast with the ID of **11223344**.

Step 3. Now we will allow **Board B** to be controlled by commands issued with a custom remote ID, ensuring that we are not being controlled by any board on the CAN.

This is achieved by entering an 8 digit hexadecimal number into the 'Remote ID' text field.

For this example, we will allow **Board B** to be controlled with commands issued to the remote ID of **44332211**.

Step 4. Now we must allow **Board A** to read the state of **board B**.

This is achieved by listening to the messages CAN and filtering them to only read messages from **Board A**. We can execute this with the combination of a mask and filter, both of which must be entered as 8 digit hexadecimal numbers.

For this example, we will set the mask to be **1FFFFFFF**. We must also set the 'Filter ID' text field with the ID that **Board B** is broadcasting with; **11223344**.

Step 5. Finally we must allow **Board A** to issue commands to **Board B**, achieved by sending commands with the remote ID decided upon two steps ago.

For this example, we set **Board B** to only accept commands with the remote ID of **44332211**, therefore we should transmit our messages with the ID of **44332211**. This is achieved by entering the remote ID of **Board B** into the 'Transmit ID' text field.

6.2 Usage

Now that the boards have been configured to allow remote control, the GUI provides controls that will issue commands to the board. It also displays the current state of the board as it is being read from the broadcasted CAN messages.

The motor, heater, and LEDs can be controlled simply by pressing the buttons within the GUI, at which point the GUI will reflect this change and the board will be sent a message with this command.

The temperature of the board is also displayed, with the ability to set a target temperature. By default, the target temperature is displayed as 'N/A', indicating that the board currently has no available temperature to aim for; therefore it will not attempt to regulate the temperature of the board. A target temperature can be set by typing into the numerical text field, then pressing the 'Set' button, at which point the target temperature will now display the user's input, and the system will respond to it.

If at any time the user wishes to change the target temperature, they may simply type again into the text field and press the set button, at which point system will respond to this and the GUI will reflect the change. If the user wishes to stop the automatic temperature control, the 'Reset' button will change the target temperature back to 'N/A', and thus the system will stop regulating the units temperature. This will also disable the board's heater and motor, bensuring that the board does not continue to overheat or cool down after the automatic temperature control has been disabled.

The ability to send a textual representation of the board's state via SMS is also provided within the GUI. After entering a valid SMS number for the data to be sent to, the user may press the 'Send Board State via SMS' button, at which point the system will extract the relevant data and construct a message (outlined in Figure 3.3) before sending it via the SMS protocol. An example of a mobile device receiving the state of a board is shown in Figure 6.1.

Figure 6.1: Screenshot of the SMS message sent to a mobile device

This page is intentionally left blank.

Acronyms

CAN control area network. i, 1–3, 5–7, 9, 10, 13–15

GUI graphical user interface. ii, 1, 3, 9, 10, 12, 13, 15

IDE integrated development environment. 9

LED light-emitting diode. 1, 3, 7, 10, 15

SMS short message service. ii, 2, 3, 6, 7, 15