Report: Room Detector

Jeremiah Griffin

Spring 2017

Contents

List of Figures

List of Tables

1 Introduction

The room detector uses inputs from several sensors to determine what room or environment the device is currently in. It uses a statistical model to associate ambient factors with a user-defined room number. The network must be trained by the user before it may be used. This is done online by placing the device in each room to be detected, entering training mode, associating either a new or existing room number with the training data, and then moving the device through the room. The longer the device is trained, the more accurate its classifications become. Once training is complete, the device may enter detection mode and be moved between any of the trained rooms. If brought to an untrained room, it will make its best guess at classifying the room as one that was trained.

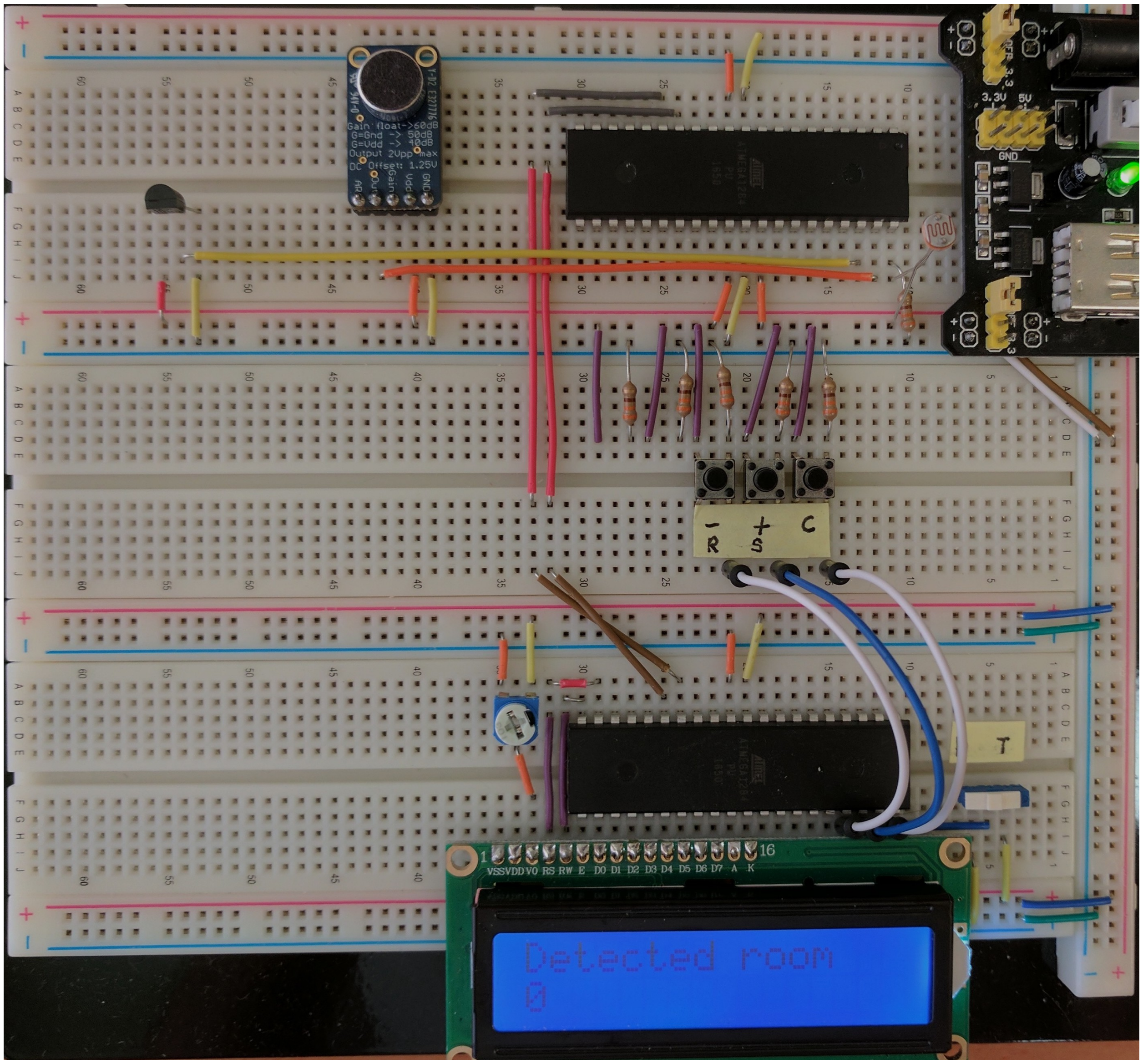

Figure 1: Overview photo of project hardware

2 Hardware

2.1 Parts

The hardware components that were used in this project are listed below. The equipment that was not taught as part of the course material is listed in bold.

Part Number	Component	Quantity
ATmega1284p	Microcontroller	2
	SPDT toggle switch	1
	SPST push-down button	3
	10KΩ trimmer potentiometer	1
	330Ω 10% resistor	4
	CdS photoresistor	1
LCM-S01602DTR/M	Liquid crystal display	1
TMP36	Temperature sensor	1
CMA-4544PF-W	Electret microphone	1
MAX9814	Microphone amplifier	1

Table 1: Listing of parts used in design

2.2 Pinout

3 Software

The software designed for this project was implemented using the PES standard. The overall design as a task diagram is included below.

In the following sections, the individual tasks from the above diagram are briefly described in addition to their inputs and outputs. The state machines for these tasks are presented in appendix A.

3.1 Tasks

4 Complexities

4.1 Complete

- Using EEPROM to persist the classification model state
- Using USART to offload computation to a dedicated microcontroller
- Using ADC multiplexing to collect analog readings from multiple sensors
- Using a temperature sensor and microphone obtained outside of the lab kit
- Using a statistical model to perform input classification using fixed-point arithmetic

4.2 Incomplete

- Using a neural network to perform input classification using fixed-point arithmetic
  This complexity could not be completed due to hardware limitations. Its role in the project was instead filled by the statistical model listed above. The intended design for the neural network was a multi-layer feed-forward network, with one input layer of size 4, one hidden layer of size 10 with a linear activation function, one hidden layer of size 10 with a hyperbolic tangent activation function, one hidden layer of size 16 with a linear activation function, and one output layer of size 16 with a softmax activation function. This topology was chosen as the best for fulfilling the goal of classifying

three inputs, with one bias input, into one of 16 outputs. However, this network proved too large for the ATmega to process. When including the amount of memory necessary for storing training state, the network became too large for storage in the ATmega's 16 kilobyte SRAM. Additionally, even using fixed-point arithmetic in place of traditional floating point, the times for classification became slower than was considered acceptable and the times for training made it obvious that training would only be possible offline using a faster device, which was deemed to be beyond the scope of this project. As a result of these limitations, the neural network was replaced with a simpler statistical model at the cost of accuracy and robustness when faced with input noise.

5 Links

GitHub Repository https://github.com/nokurn/roomdetect

6 Known Issues

- The classification algorithm does not cope well with noise and may confuse environments with a large degree of noise as being equivalent regardless of the specifics of that noise.
- The temperature sensor is not measured with sufficient precision for its readings to adequately influence the results of the classification.
- The microphone primarily distinguishes between noisy and quiet environments with no consideration of the qualities of the sound such as timbre and pitch.
- The classifier most heavily relies on the measurements from the photo-resistor, despite unbiased treatment of all readings in the input vector. This is likely due to the aforementioned reasons: low analog precision and the need for pre-processing of some readings.

7 Future Work

- The largest improvement might come from replacing the compute microcontroller with a higher-power compute unit, possibly integrating floating-point arithmetic, and using a more robust classification model such as a neural network.
- By using an external analog-digital converter with higher precision, or simply replacing the ATmega with a microcontroller with a higher-precision integrated ADC, sensors with small output ranges would contribute more to the accuracy and confidence of the device's classifications. This would reflect immediately in the ability of the device to distinguish between environments with slightly different temperatures.
- Applying a Fourier transform to the microphone signal and using the individual components as input to the classifier would improve the quality of classifications with regard to environmental sounds. For example, if one room contains several computers with spinning fans and another room has a single ceiling fan, yet the volume level in both rooms is similar, a Fourier analysis of the sound waves would enable the classifier to distinguish between the two by the timbre and pitch of the sound in addition to its loudness.
- Adding additional sensors would increase the size of the classifier's feature vector and its accuracy. Barometric pressure, humidity, altitude, or even GPS would be likely candidates for inclusion in the system.

A Appendix: State Machines

A.1 Master Microcontroller

Figure 2: State machine for master mode task

Input `PA0`

Output `device_mode: enum`

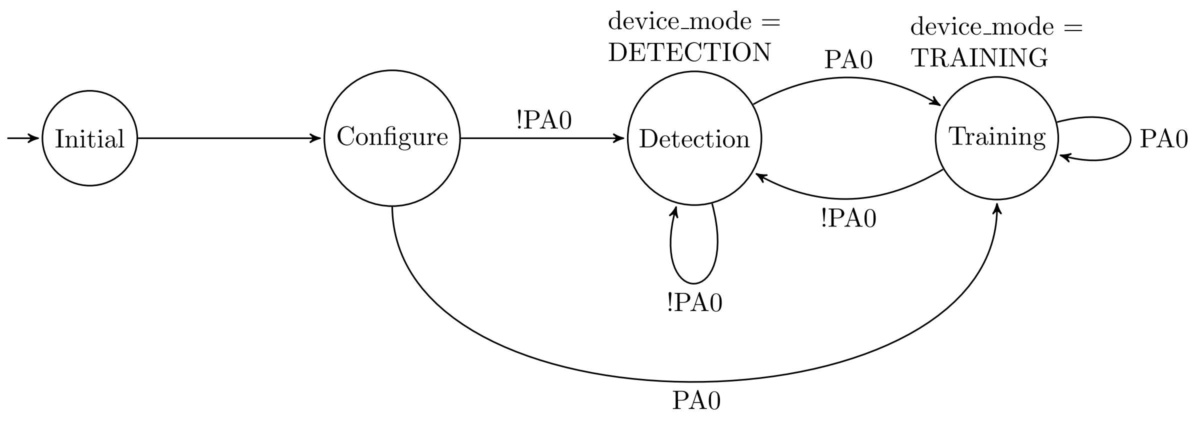

Figure 3: State machine for master detection task

Input `device_mode`; `PA2`; `PA3`

Output `detected_room: uint8`; `usart_tx_queue`

Local `tick_count: uint16`

Figure 4: State machine for master training task

Input `device_mode`; `PA1`; `PA2`; `PA3`

Output `training_room: uint8`; `is_training: bool`; `usart_tx_queue`

Local `tick_count: uint16`

Figure 5: State machine for master display task

Input `detected_room`; `training_room`; `is_training`; `should_display_save`; `should_display_reset`; `should_display_erase`; `is_lcd_ready`

Output `should_lcd_clear`; `should_lcd_write`; `lcd_buffer`; `lcd_position`

Local `last_room: uint8`

A.2 Compute Microcontroller

Figure 6: State machine for compute mode task

Input `usart_packet_rx_queue`

Output `device_mode: enum`; `training_room: uint8`

Figure 7: State machine for compute reading task

Input `should_poll_readings: bool`; `adc`

Output `reading_vector: accum[]`; `are_readings_ready: bool`

Local `readings: accum[]`; `index: size`; `samples: size`

Figure 8: State machine for compute detection task

Input `device_mode`; `reading_vector`; `are_readings_ready`

Output `detected_room: uint8`; `should_poll_readings`

Figure 9: State machine for compute training task

Input `device_mode`; `reading_vector`; `are_readings_ready`; `training_room: uint8`

Output `should_poll_readings`

A.3 Common Infrastructure

Figure 10: State machine for common LCD task

Input `should_lcd_clear: bool`; `should_lcd_write: bool`; `lcd_buffer: char[]`; `lcd_position: uint8`

Output `is_lcd_ready: bool`

Local `index: size`

Figure 11: State machine for common USART TX task

Input `usart_tx_queue: uint8[]`; `usart_tx_queue_size: size`; `usart_tx_queue_head: size`; `usart_tx_queue_tail: size`

Output `usart_tx_queue`

Figure 12: State machine for common USART RX task

Input `usart_rx_queue`

Output `usart_rx_queue: uint8[]`; `usart_rx_queue_size: size`; `usart_rx_queue_head: size`; `usart_rx_queue_tail: size`

Figure 13: State machine for common USART packet RX task

Input `usart_rx_queue`

Output `usart_packet_rx_queue: usart_packet[]`; `usart_packet_rx_queue_size: size`; `usart_packet_rx_queue_head: size`; `usart_packet_rx_queue_tail: size`