Robotic Petting Zoo.

The robotic petting zoo project was produced at the SoBA Lab, within the Psychology Department, at Bangor University. The technical requirement for the project was to control a robotic vacuum cleaner to display a certain personality and play a characteristics sound when it bumped into any objects as it went about doing the cleaning task. This document details how to go about doing this for anyone who wants to try to give their Roombas a personality.

Requirements:

The robotic vacuum cleaners selected for the project was Roomba 605. The reason for selecting a Roomba was the readily available control specification for the Roomba Open Interface and the external Mini- DIN connector. The connector allows for easy two-way serial interface for reading the Roomba sensors as well as controlling it. A relatively older version of Roomba was used, as newer versions are better at obstacle avoidance. We also avoided the iCreate version, as we needed the cleaning attachments to be present and working.
We used an Arduino to control the Roombas but you can also use a RaspberryPi instead. Instructions here is for using an Arduino only.

So, things required

- Roomba 605
- Arduino Uno
- Breadboard
- Breadboard Jumper Wires
- Two 1N4001 Diodes
- A micro SDHC card (we used a 8GB card)
- Micro SD SDHC Shield TF card Adapter, 6pin SPI for Arduino
- A re-chargeable, self-powered, mini speaker with 3.5mm jack

For diagnostic purpose:

- One Blue LED
- One Yellow LED
- Two 100 Ohm resistors

Software requirements:

- Arduino IDE
- TMRpcm library (https://github.com/TMRh20/TMRpcm)

Circuit Diagram

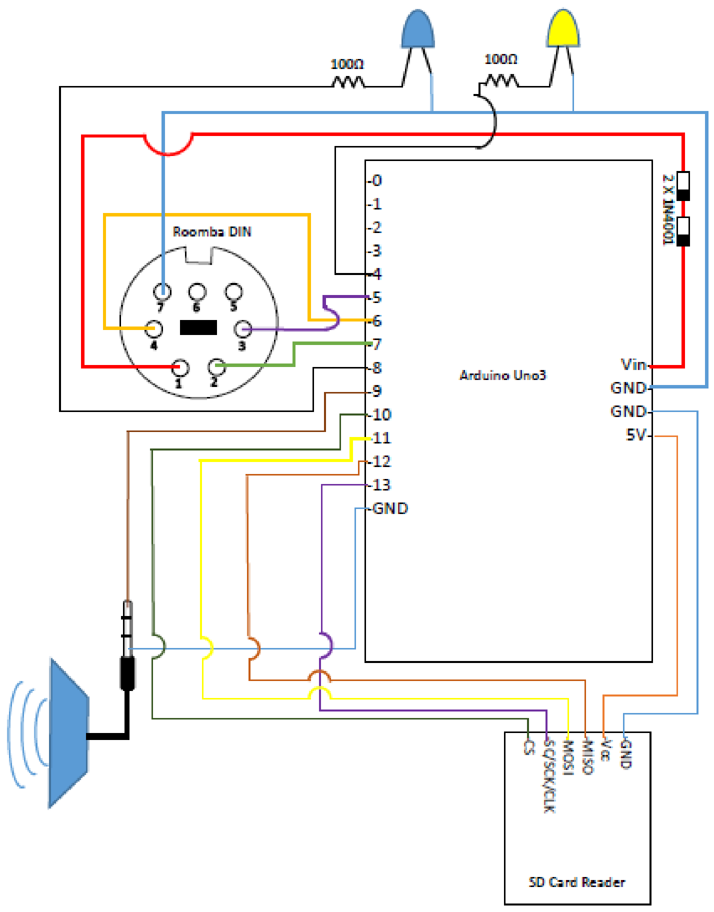
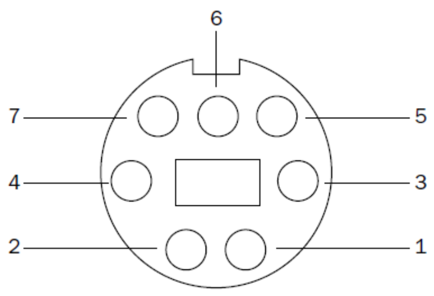


Figure 1: Arduino connection diagram



Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 – 5V Serial input to Roomba
4	TXD	0 – 5V Serial output from Roomba
5	DD	Device Detect input (active low) – used to wake up Roomba from sleep
6	GND	Roomba battery ground
7	GND	Roomba battery ground

Figure 2 Roomba DIN pin description

Design Outline:

The Roomba powered the Arduino rather than a separate battery pack. We used this approach so that we only had to worry about charging the Roomba between petting zoo sessions. The speakers were charged separately. We found that the speaker coped well throughout the day with only one overnight charging. In any case, we did not want too much power to be drawn from the Roomba.

The character selection was through a configuration file in the SD card. The SD card also had the sound files for the character in .wav format. The Roomba personalities created were:

- Excited
- Surprised
- Happy(giggly)
- Scared
- Angry
- Apologetic

The sound corresponding to each personality was pre-recorded. The sound files had to be of unsigned 8-bit PCM format, with mono channel and 16000Hz/sec sample rate. There could be multiple sounds per personality. All sounds for a personality had to be of the same length so that they could correctly synchronize with the movement. The sound file name had a fixed prefix for each personality followed by an index. The index was selected at random on bump to decide which sound file would play.

The Arduino switched the Roomba between clean mode and character display mode on bump. The Arduino constantly commanded the Roomba to return its bumper sensor values during the clean mode. There were a few options for the bump sensors and we used the two boolean values for left bump and right bump returned by packet ID 7. We did not use the IR signal based Light Bump sensor because they were too sensitive and did not give the Roomba a chance to come out of the bump or sometime triggered on the slightest touch. The Light Bumps gave an integer value for the strength of bump. We would have had to process that to detect bumps strong enough to act on. So, for the project we reverted to using the Boolean that was already giving this.

On detecting the left or the right bump the Arduino would run the commands for the Roomba switched off cleaning to do its characteristic moves as well as play the sound. Part of the function of the character movement was to take the Roomba away from the obstacle so that the Roomba is not stuck. So depending on whether the right or the left side was bumped the Arduino changes the turn away direction of the character move.

The code can run in **debug mode** in which the Arduino can remain attached to the IDE serial monitor through the USB and the Roomba. In this case, the Arduino is powered through the USB so the Vin must not be connected to the Roomba Vpwr. All other connections must be setup. In debug mode, all serial prints will show on the IDE serial monitor, the Roomba bump sensors will be read and the characteristic sound will play. However, in debug move all move and cleaning commands from the Arduino to Roomba are deactivated. Because we wanted to be able to do this, we did not use the hardware Tx/Rx pins 0 and 1 to communicate with the Roomba configured other pins to do so instead.

To prevent Roomba from making its own system sounds we opened it up and disconnected its speakers. This gave us trouble when the Roomba ran into internal error conditions, as we could not hear the diagnostic messges. So we followed had our own LEDs that gave indicator of when things were going wrong and had an initial start movement sequence to check that the Roomba was correctly initialised.

Things that caused problems:

Memory was the biggest trouble we faced. Using SD card with Roomba takes up a lot of the RAM. Our problems were compounded by the fact that we had to increase the buffer size in pcmConfig.h to 100bytes from 64bytes to get a reasonably good quality for the character sounds. So we had to optimize memory usage by decreasing the size of global variables, removing nested function calls where possible, localising variables, putting string in the PROGMEM.

The second trouble we ran into was not putting in enough delay between commands causing conflicts. This was especially true with reading the sound files from the SD card before trying to play them and not giving the Roomba enough time to finish a move.

We sorted out most of these issues, but we had kept a soft restart in code as a backup in case problems were detected.

References:

We built on and from the information that was already available on the internet. Here are the references to the sources this project drew from. You will have to refer back to them if you need background details.

- Specification for iRobot Roomba 600: http://www.irobotweb.com/~/media/MainSite/PDFs/About/STEM/Create/iRobot_Roomba_600_Open_Interface_Spec.pdf
- iRobot Arduino Tutorial : https://www.irobotweb.com/-/media/MainSite/PDFs/About/STEM/Create/Arduino_Tutorial.pdf?la=en
- Playing .wav file on Arduino:
  - https://www.youtube.com/watch?v=gi9mqIha8n0
  - https://diyhacking.com/arduino-audio-player/
- Disconnecting Roomba speakers: http://www.schneordesign.com/Avi/irobot/roomba_spk1.htm
- Optimizing Arduino memory: https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram