Raspberry Pi robot

Intelligent Embedded Systems - Final report

Niemistö, Jesse Muona, Leo Hilden, Matias

February 20, 2014

Contents

1 Introduction

This is a final report for the course Intelligent Embedded System (id 582711) at the CS Department of the University of Helsinki. This course was held during Spring term 2014, third period. The course is a self-study course that consists of an embedded systems project. For our project we chose to build a Raspberry Pi robot, which utilizes Pi's camera module, audio output and electric DC motors.

Raspberry Pi is a cheap arm-computer that was originally created for the purpose to help more people to get into programming. We chose to use this one-chip-computer for our project because it runs on Linux (among other operating systems), it has a camera module available, it has GPIO pins for controlling a motor control chip, and all of our team members already had one lying around.

This document includes our project idea, requirements and initial design for the robot, implementation of the robot, evaluation of our project, and a summary of lessons learned. This is the tale of our awesome little raspi robot.

2 The idea

We got our initial idea from the video game Portal by Valve Corp. In the game, there were these "cute" robot turrets that recognized movement, talked to the source of the movement and shot them. So our initial idea was to create a robot similar to these turrets, so that we will use Raspberry Pi's camera module to recognize movement, target the movement source, and play a random audio clip to greet the source of the movement.

Our initial idea had two DC motors to control the turret's movement and a laser- or LED-pointer to point the target. However this idea evolved during the project just to use one motor and only turn the camera towards the movement source.

3 Requirements and initial design

At the start of our project we wrote our requirements and then initial design. The design consists more on actual planning of hardware part than the code to be created. This section can be divided into five parts: required features, audio system design, camera design, motor control system design, and software design.

3.1 Required features

These were the required features at the start of our project when designing the robot:

- Horizontal and vertical rotational movement.
- WAV (or ogg) audio playback.
- Motion detection based on frame difference algorithm.
- Aiming at detected movement.
- Remote control.
- Works on top of Linux.

We used these features a guideline for our project design. At this stage it is good to mention that some of these features were modified during the project.

3.2 Audio design

For audio we used the 3.5mm jack audio output connector on the Raspberry Pi with our self-built audio amplifier, which is connected to a single speaker. The single speaker can only output mono audio, but this won't matter since audio contains only speaking. The used speaker was small and low powered. We also made a simple audio jack connector. This was made with a bought 3.5mm mono jack connector, which was soldered to ground and audio.

In the heart of the audio amplifier stands the LM386 circuit, which is a "Low Voltage Audio Power Amplifier" and was more than enough for our use case. The figure in 3.2 shows the breadboard view that we used used for the audio amplifier. The build is same to that what is shown on the LM386 datasheet, "amplifier with Gain = 200".

- Between PINs 1 and 8 is a 10uF capacitor, which causes a gain on 200. In desibels this is around 20.
- PIN 2 is places to ground.
- PIN 3 takes the audio input, and is connected to ground through 10k ohm resistor.
- PIN 4 is placed to ground.

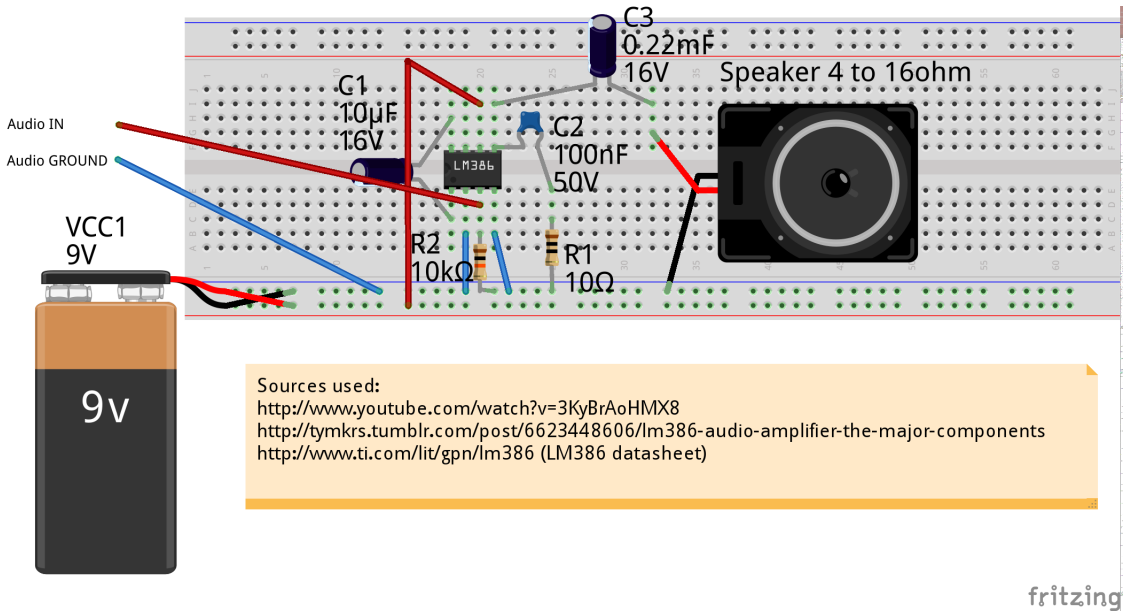

Figure 1: Breadboard view for the audio amplifier.

- PIN 5 is the amplified audio output.
- PIN 6 takes power
- PIN 7 is bybassed, doesn't take connection.

3.3 Camera design

We decided to use a simple plug-in Raspberry Pi camera module, which we will use to take pictures that we compare to previous ones to detect movement. Since Raspberry Pi has a ready-to-use socket for camera cable, no extra cables or power supplies are needed. To use the camera we will use Pi's Userland code, that comes with raspistill-program which we can use to take still pictures.

3.4 Motor control design

To control the motors, we planned our design around L293D motor control chip, which is cheap and easily available. Our design consisted of two motors, one to turn the camera horizontally and one to turn the camera vertically. This design used two L293D chips, one for each motor.

Figure 3.4 shows how the chips would be connected into the Raspberry Pi. Design uses GPIO pins 17 and 18 to control the first motor and pins 22 and 23 to control the second motor. The two AA batteries will supply power for both of the motors, and Raspberry Pi will supply power for the motor control chips.

3.5 Software design

Our project requires software for three main things: camera control, audio control and motor control. Even though we use a ready-to-use software to take pictures with the camera, we need to write our own software for image analysis, that is used to detect movement in the camera. We will write our own audio playback code against pulseaudio sound server. As for sound files, we are planning to use .ogg files. For the motor controlling we plan to use either direct /dev/mem code, or an additional WiringPi gpio library to control the GPIO pins.

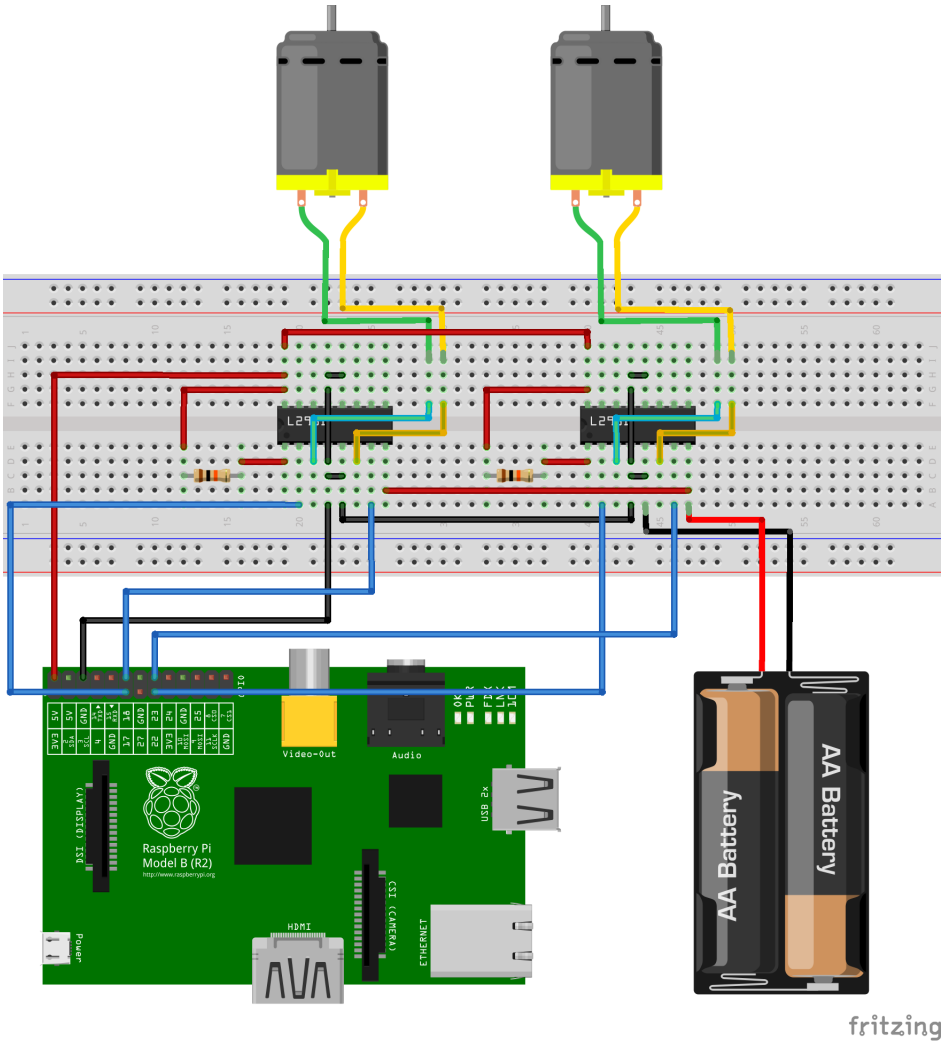

Figure 2: Motor controllers design in breadboard view.

We will also write other code, like the logical code for the whole software, threading for different parts, and a remote control code for testing purposes. And hopefully we don't need to write multiple hacks.

4 Implementation

For the implementation, we ran into few problems along the way. Implementation was divided into three separate parts: motor related stuff, audio related stuff and camera related stuff. These parts were implemented as paraller. During the implementation, few features needed some modification and later of the project timetable few "nasty" code and hacks were implemented to get the wanted results.

4.1 Audio

The implementation of the audio amplifier required bying the parts and assembling them. There weren't really any problems with this, and everything worked as was planned. Refer to chapter 3.2 for what was the plan.

The speaker didn't require any assembling but was simply connected to the audio amplifier. For the case of our speaker we used a beer can. Twas used for flavor.

The software part wasn't that easy. We used Pulseaudio audio system for interacting with the underlying audio hardware. We decided to use the Pulseaudio provided Threaded API, as we wanted to play sounds in a non-blocking way. The Initial implementation worked well on test cases, and on development computers. However, when moved to the actual enviroment, which was Raspberry Pi under load, the problems started. Firstly, Pulseaudio allows under no conditions connections from a root user. This caused a few problems as our software at the time required root user (wiring Pi actually). After working around this (dropping root), we noticed that audio was crackling as Raspberry Pi couldn't handle all the load. This required making some things sleep during audio playback.

This wasn't all thought. It seems that at least on the current Raspberry Pi board something with the audio driver or Pulseaudio was bugging, and the whole audio playback would jam for some unknown reason. Working around this actually required a erally evil workaround. Luckily all our samples were really short (under three seconds), so if we notice that three seconds has passed by and our audio subsystem says it is still playing, we send SIGINT to pulseaudio to kill the connection. This made it possible that our robot is able to continue its smooth operation.

4.2 Motor control system

Largest modifications to the design had to be done to the motor controlling system. After implementing the planned two-motor-system, we noticed that it would be hard to create correct platform that would be aimed in two dimensions. In addition to this, we noticed that we couldn't implement the correct calculations for vertical movement during project's timetable. Thus we changed the motor system to use only horizontal motor.

The hardware implementation of our motor system is shown in figure 4.2. The AA batteries now have to supply only one motor, and we had to implement code for only two GPIO pins. The L293D motor control chip we used gets it's VCC1 power from Raspberry Pi's 5V pin, and uses that same line trought a 10k ohm resistor to give the enable signal for our motor chip.

Motor control's software had some changes as well. After we started to use WiringPi library the software had to be run as root, as it was WiringPi's requirement. This caused us some problems with pulseaudio, which has a lot of problems when run as root. A fix to this was to remove WiringPi linking from our software and add WiringPi library only as a system requirement. That is, it should be installed on the Raspberry Pi normally. Then we used system calls to use gpio-program, that was bundled with WiringPi and could be run in non-root environments. This is one of the hacks we had to do to get things working.

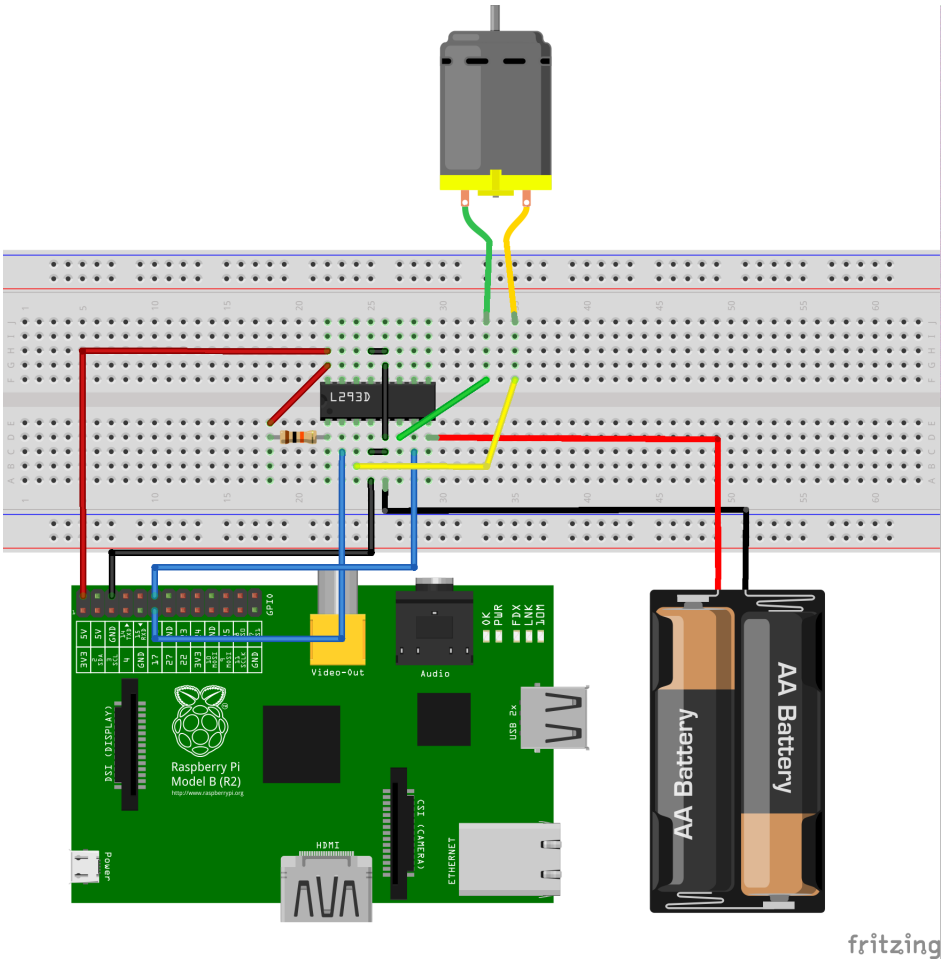

Figure 3: Motor controller implementation in breadboard view.

4.3 Camera

Implementing the taking of pictures was not a problem due to the usage of raspistill-program. The program turned to not be optimal for continual usage but served our purposes well enough. Image difference calculation and rotation angle calculation were purely mathematical algorithms and thus did not need to interact with Raspberry Pi hardware. This made their implementation quite straightforward.

5 Evaluation

Motor controlling system worked well. Mainly due to the hack we made and the fact that the L293D chip is a pretty simple control chip. Using the GPIO pins is pretty straight forward with the WiringPi library. Only bad thing in the motor control system implementation is that we had to downgrade our project to use only one motor instead of two.

Audio amplifier, when connected to the speaker had suprisingly good sound and volume. Actually we had to drop the volume to one fourth of what was the full volume not to break our ears. The speaker in a can outputted a good and clear sound, better than any of us could have hoped for.

However, when audio jack is not connected, and even sometimes when connected, the audio amplifier and speaker takes audible interference. A high-piched noise that eats into your mind. There were a few workarounds, that mostly was duct taping the wires together which acted as a shield for the wires.

The software side of the Audio, when coupled with the hacks it required, works quite well. At times one can notice that the hardware can't really handle all the load, and audio makes a few cracks. We really shouldn't also have used Pulseaudio Threaded API. This mostly adds overhead, especially since Raspberry Pi is a single processor system.

6 Summary

The project started with a funny idea. And the idea continued to live on through the project. Thus all team members agree that this project a great success. There were few hacks along the way and few features had to be modified or dropped, but the robot still was quite similar as the initial idea. Our internal project timetable actually went pretty badly, because implementation took way more time than we originally though. However we were able to get the robot in a complete state.

When analyzing the keys to success, this that comes to mind is the fact that all team members knew each others beforehand, and the project was really fun to work with. Also the amount of documentation for the Raspberry Pi helped alot.

7 Links

- Demonstration video: `http://www.youtube.com/watch?v=bQQ_EZM07OI`
- Project github repository: `https://github.com/domewolf/raspimobot`
- Raspberry Pi userland repository (contains raspistill): `https://github.com/raspberrypi/userland`
- WiringPi homepage: `https://projects.drogon.net/raspberry-pi/wiringpi/`
- Raspberry Pi documentations: `http://elinux.org/RPi_Hub`