OSP Project Specification
Team ⟨sql injection⟩

Neil Ang	"Alfred" Yang Yuan	Val Lyashov
s3251533	s3363619	s3366222

Semester 2, 2013

Contents

1 Introduction 2
2 System Overview 2
3 Design Considerations 5
3.1 Goals and Objectives 5
3.2 Assumptions and Dependencies 6
3.3 General Constraints 6
3.4 Development Methodology 7
4 Architecture 8
4.1 System Design 8
4.2 Data Design 10
4.3 Program Design 10
5 Testing Issues 12
5.1 Types of Testing to be Conducted 13
6 Roles and Responsibilities 13
6.1 Val Lyashov 13
6.2 "Alfred" Yang Yuan 13
6.3 Neil Ang 14

A Reflections of the Raspberry Pi Cross-Compiler Build 15

Bibliography 17

1 Introduction

Our project will attempt to modernise the phonograph by using the *Raspberry Pi*® for subject tracking, digital recording and audio processing. A directional microphone will be installed on a custom mount that can be programmatically moved in an X and Y direction. A connected camera will then be used to detect the subject (e.g. through face detection) and adjust the position of the microphone accordingly. A secondary goal of the project is to then process the audio through an Internet based voice-to-text service, and produce saved audio recordings with completed transcripts.

The final solution will be a relatively cheap, portable and smart recording device suitable for use in a lecture theatre or classroom. It would be ideal for assisting students in note taking, or for lecturers teaching in learning spaces that haven't been outfitted with full featured recording systems (such as the *Lectopia*[1] or *Echo360*[2] systems used at *RMIT*).

2 System Overview

This is an ambitious project that involves complex processing and interaction with hardware. To distribute the CPU workload and modularise the solution, the functionality will be spread over two *Raspberry Pi*®*s*. This will have the added benefit of making it easier to work on as a group, as the components can be worked on independently of each other.

The first *Raspberry Pi*® will be responsible for subject tracking using computer vision. It will detect the subject via an attached camera and communicate directly with a servo controller to move the microphone mount. In an effort to conserve power, this *Raspberry Pi*® will also have a motion sensor connected to one of its General Purpose Input/Output (GPIO) ports. When no motion is detected, it will power down the connected peripherals. Figure 1 illustrates the connected components for this device.

The second *Raspberry Pi*® will be responsible for the audio recording and processing. It will have a shotgun (or directional) microphone connected to it via a USB sound card. As audio is recorded it performs voice-to-text processing through a web service. A USB wireless card will be attached to support connection to the Internet. Figure 2 illustrates the connected components for this device.

Both *Raspberry Pi*®*s* will also be networked together via their ethernet ports for inter-device communication. For a detailed summary of all involved hardware and its purpose in the project, see Table 1.

[1] http://www.rmit.edu.au/teaching/technology/lecturecapture
[2] http://www.rmit.edu.au/teaching/technology/echo360

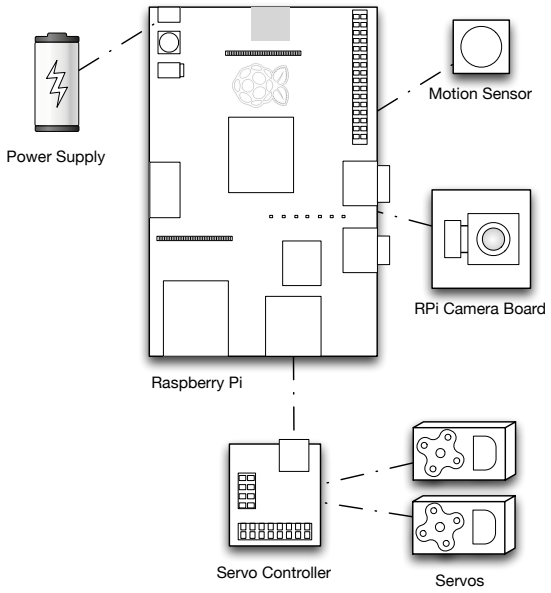

Figure 1: Connected hardware for the computer vision component of the project. It includes a power supply, motion sensor, RPi camera board, servo controller and two servos.

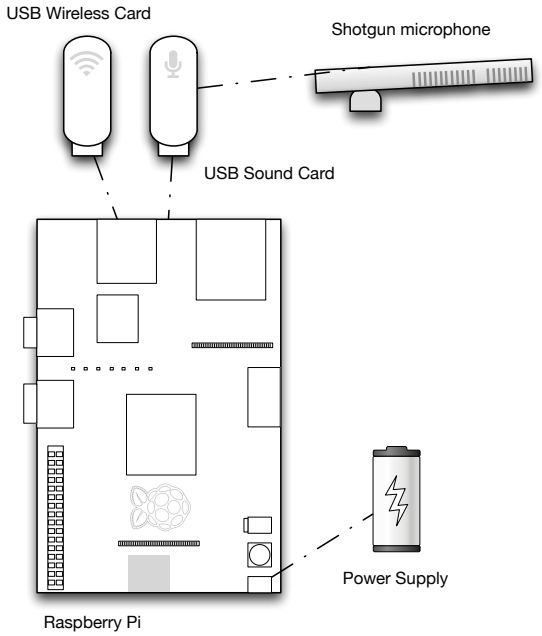

Figure 2: Connected hardware for audio recording component of the project. It includes a power supply, sound card, shotgun microphone and wireless card.

Hardware	**Qty**	**Purpose**
Raspberry Pi®	2	Due to the inherent complexity of audio and video processing, the solution workload will be split across two *Raspberry Pi®s*. The first will handle the computer vision and controlling the servos. The second is for processing audio and data storage.
Camera	1	Either a RPi Camera board or USB web cam will be used for subject tracking through computer vision.
Servos	2	Two servos will be used to control the microphone mount. One servo will manoeuvre the mount on an X axis, while the other is used for the Y axis.
USB servo controller	1	This will be used to improve the movement accuracy of the servos.
USB sound card	1	The *Raspberry Pi®* has no in-built audio input jack, so an external sound card will be used to facilitate connecting the microphone.
Shotgun microphone	1	This microphone will provide targeted audio for recording voice and minimising background noise.
Motion sensor	1	To conserve power before and after a recording session, a motion sensor will be used to detect the presence of a subject in the room. The system will enter a "standby mode" if no movement is detected.
Power supply	2	To take advantage of the lightweight nature of the *Raspberry Pi®* and components, an external battery pack would be an ideal supply to keep the system portable.
Wireless card	1	The voice-to-text processing will be performed remotely, hence an unobtrusive Internet connection will be required.

Table 1: Summary of hardware

3 Design Considerations

Early experiments have indicated that face detection and audio processing will be very CPU intensive. To ensure the system runs smoothly, we will need to optimise the system as much as possible. For example, although using Python would be acceptable for prototyping scripts, the final implementation of the software will be re-written in C/C++ for optimal performance.

As this is a fairly complex project, we will also need to find ways to modularise the hardware and software components. This is so they can be developed in parallel by different group members, as well as making it easier to redesign components of the system if any problems arise.

Finally, a physical design consideration will be to build a sturdy, easy to assemble and portable system. Ideally we will try to construct a battery powered, low consuming system. Powering the solution by battery will not be difficult, however, we will need to invent intelligent ways to reduce resource usage, therefore increasing battery performance. This will include powering hardware on and off when not in use and keeping CPU usage to a minimum on both devices. Although we will endeavour to design the project to be power conservative, we will only explore battery power if we have time remaining at the end of the project.

3.1 Goals and Objectives

This project aims to provide an easy way to audio record and annotate lectures and other presentations. As students, it is a solution we would use ourselves, however, we are also attempting to meet all learning objectives of the assignment.

Personal goals

Design a solution that is to be relatively portable, requires minimal setup and start-up time.

Learning objective 1

The *Raspberry Pi*® is a resource constrained device, so for this objective we will ensure that our system runs efficiently with the hardware available. We will achieve this by running performance tests for all software components and introducing clever ways to reduce CPU usage and conserve power (e.g. using the motion sensor).

All source code and project documentation for this project will be version controlled through *git*, and hosted on a private[3] *GitHub* repository along with our project wiki and development logs.

[3] Access to this repository is available upon request.

Learning objective 2

The system will be built on a bare-bones linux distribution and designed so that each sub-component of the main software application will be responsible for interfacing with only one hardware feature (e.g. the *comms* code will be the only process/thread that can access the ethernet port, and the *face-detect* process/thread will be the only code to access the camera). Building the solution in this manner, with no other programs competing for resources is how we will effectively manage the hardware.

Learning objective 3

We will cover this in detail in the final portfolio through diagrams and descriptions of the system design.

Learning objective 4

As a team we have assigned primary roles which make best use of our individual skills. However, to increase each group members personal learning in areas they are not already accomplished in, we have allowed for some overlap in the roles. For example, Val who has expertise in Unix and hardware, will also attempt to prototype some of the hardware interfaces in Python. Neil, who does not have much hardware or C/C++ experience, will then port the Python scripts to C/C++. The primary responsibilities for the roles will still lie with designated group members, but everyone will have the opportunity to try something they have not worked on before.

3.2 Assumptions and Dependencies

Due to time constraints, we will be dependent on a third party framework for subject detection. To date we have completed some preliminary research and experimented with OpenCV[4] for facial recognition.

When recording an audio session through our product, we have also made the assumption that the system will be placed front-on to the subject, and the recording environment will be reasonably lit.

3.3 General Constraints

A major constraint of this project is the processing power of the *Raspberry Pi*®. Preliminary investigations suggested that depth sensing would be a more accurate subject tracking technique, however the *Raspberry Pi*® would not have the processing power required for skeletal mapping. We have opted to use simpler face detection instead. However its effectiveness will be limited by the light in the room, the direction the subject is facing and any obstruction of facial features.

[4] http://opencv.org/

As we are building moving parts into the project, we will also be constrained by the total weight of the hardware. There is a physical limit to the servos torque and we have to ensure we do not exceed the weight limit in our mount design.

Finally, the biggest constraint of the project is time. The project is very ambitious, with many individual items that could not turn out as planned. Adapting to these issues will become increasingly limiting as we approach the delivery date.

3.4 Development Methodology

The project will be constructed over three main stages, in isolated parts by all group members.

Stage 1

The first stage will attempt to solve the hardest issues of face recognition and physical movement. We will also need to demo our cross-compiler and develop the project specification.

Objective	**Members**	**Week**
Lab demo	All	5
OpenCV tracking	Alfred, Neil	4, 5, 6
Servo movement	Val	4, 5, 6
Project specification	Neil	6

Stage 2

Stage two will also encapsulate the mid-semester break. We will combine what was built in the first stage and start work on implementing audio features of the second *Raspberry Pi*®.

Objective	**Members**	**Week**
Servo mount	Val	7
Audio recording	Val	8
Pi device comms	Alfred	7
Server complete	Alfred	8
PIR monitor	Neil	7
Integrated face tracking /w servo	Neil	8

Stage 3

The final stage will involve putting all the finished pieces together. We will also need to deliver our presentation and portfolio in this stage.

Objective	Members	Week
Data storage	Neil	9
Voice-to-text	Val	9
Client complete	Alfred	9
Demo	All	10
Portfolio	All	12

Stage 4

In the event that the group finishes the project early we can optionally explore power improvements to the project. This would include benchmarking and tracking power usage with the hardware, re-designing all the hardware to be powered by battery, and integrating an in-progress battery indicator into the project.

4 Architecture

The construction of the system will be complex due to the number of items to be achieved. These include a technical design with compatible hardware, a stripped back operating system, an efficient software layer, and a physical design that is portable.

4.1 System Design

The project hardware will consist of two *Raspberry Pi®s* working in parallel as a complete system. Communication between the two *Raspberry Pi®* devices will be undertaken through a central messaging system, which will enhance modularity of the project components.

The pan and tilt motion of the project will be provided by two servos connected to a USB *Pololu Maestro* servo controller. The benefit of an external hardware controller is much greater accuracy and reduced latency of servo responses. Communication is achieved through the virtual serial port over the USB, utilising *Pololu's* proprietary communications protocol[5]. The external controller requires a 5V rail to power servo movements. Early

[5]http://www.pololu.com/docs/0J40/5.c

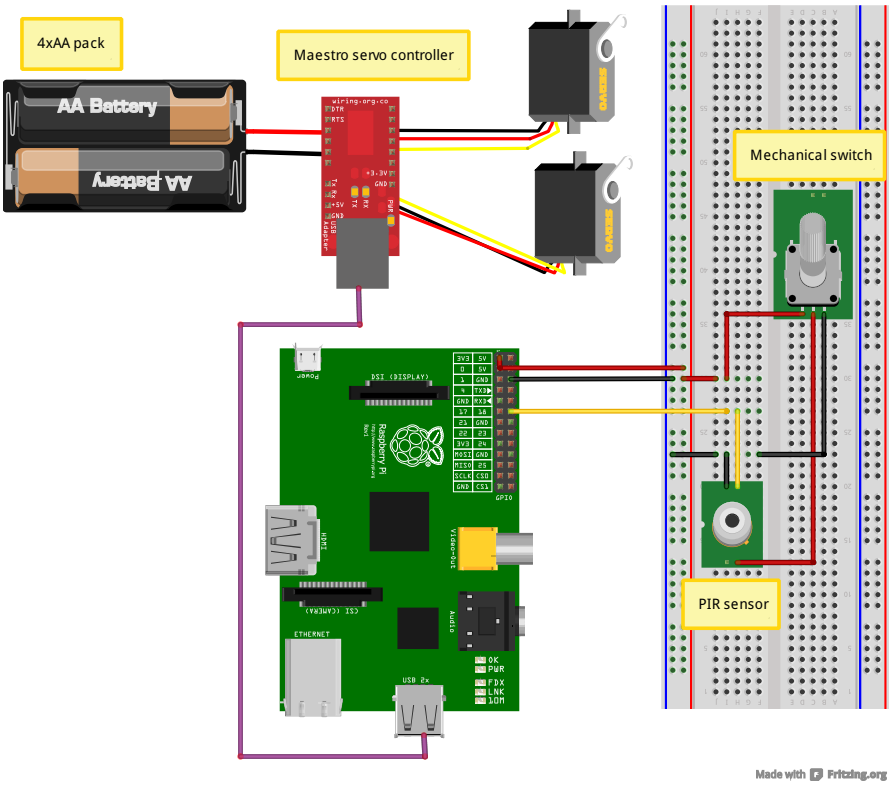

Figure 3: Wiring design for servo controller and PIR-sensor.

experimentation showed the *Raspberry Pi®s* on-board 5V is unreliable for burst usage, causing the device to restart. Figure 3 provides a high level illustration of how the servo hardware is wired to the *Raspberry Pi®*.

As the *Raspberry Pi®* devices do not have built-in microphone input ports, sound recording will be enabled through a USB sound card. Initial tests show a requirement for this sound card to be physically connected away from the *Raspberry Pi®*, as the circuit design generates notable line noise.

Power will be delivered by a single powered USB hub that is also connecting the first *Raspberry Pi®* to the USB camera and external servo controller. An additional 5V of power will be delivered by an external 4xAA battery pack for movement of the servos.

In order to reduce overall power consumption of the system, a passive infrared (PIR) proximity sensor will be utilised to detect motion in line-of-sight of the RPi camera board.

4.2 Data Design

It is undecided whether the program will be implemented through threads or sub-processes. Regardless of the method used, we will need to devise a solution for inter-process communication. For inter-device communication we will implement a message queue protocol over TCP, utilising the *Raspberry Pi®s* native RJ45 connector.

One notable issue that is still to be addressed is the efficient storage of audio data from recording sessions. The group has already started researching fast encoding algorithms and formats to reduce data offloading requirements and increase the operational time of the end-device. Possible solutions explored to date include external hardware mp3 encoders as well as fast FLAC encoding software solutions.

4.3 Program Design

There will be a custom main program running on each *Raspberry Pi®*. The program on the computer vision device will act like a *server*, and the audio recording device will behave like a *client*. These programs will spawn and manage sub-processes or threads to be responsible for individual tasks. For simplicity they will just be referred to as "sub-processes" in the below descriptions. Figure 4 is a high level summary of the components that make up each main program.

server and client

The *server* will have three state based modes, which will inform how its sub-processes behave. It's states will be "standby", "seeking" and "active". The "standby" mode will halt any unnecessary power usage, the "seeking"

Figure 4: A modularised representation of each programs sub-processes.

mode will inform the processes to seek out a target, and the "active" mode will be used for normal operation.

The *client* will also be state based, but only use "standby" and "active" modes. It's state will be set and updated by the *server* program.

pir-monitor

The *pir-monitor* will be responsible for recording motion detection in the room. It will listen to a passive infrared sensor (or PIR-sensor) connected to the GPIO pins on the device. Whenever it detects movement in the room, the sub-process will pipe out the current timestamp to a shared memory location. The *server* will periodically check the timestamp for recency, and use this information to assist in choosing the next system state.

face-detect

The *face-detect* is responsible for controlling the connected camera hardware and looking for faces in captured images. It will continually request an image from the camera and process the image to determine where a face is located. The coordinates of a face will be communicated with the *servo-ctrl* for updating its position.

servo-ctrl

The *servo-ctrl* is responsible for orienting the microphone and camera in the right direction. It will receive instructions from the *face-detect* sub-process and adjust its positioning to try and centre the camera. The *servo-ctrl* will be able to adjust the servos 180° on an X and Y axis.

voice-ctrl

The *voice-ctrl* is responsible for handling the microphone hardware. It will start and stop the voice recording and handle saving the sound recordings to hard disk. Ideally it will recognise gaps in audio patterns and save the data in "chunks" of speech for easier processing.

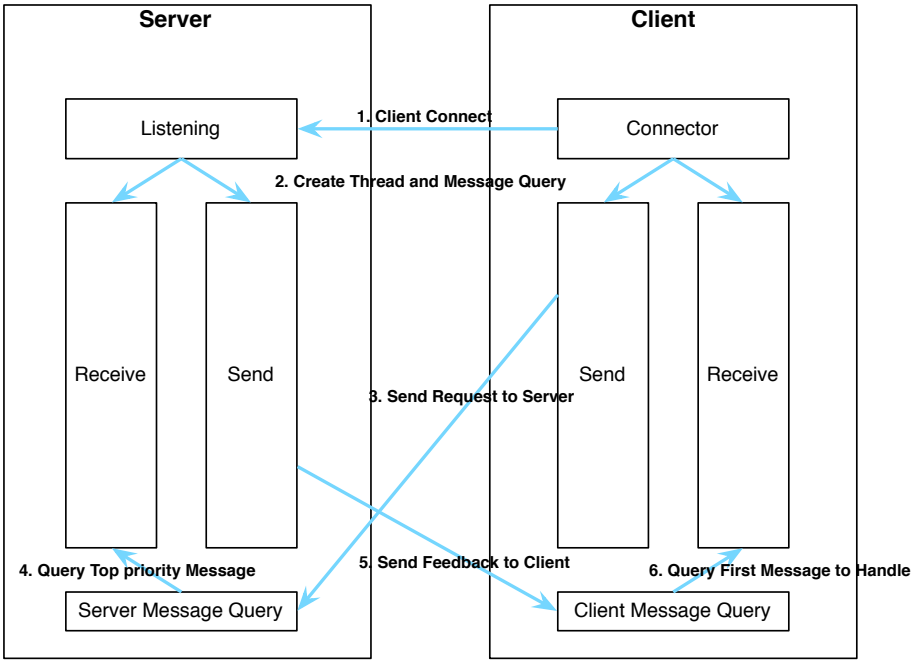

Figure 5: High level illustration of inter-device communication.

text-to-speech

The *text-to-speech* sub-process will monitor for new recording files and transmit them to the Internet for conversion. Once the audio recordings have been translated, it will concatenate the audio files together and add the running transcript.

comm

Both the *client* and *server* will communicate with each other via a message queue implemented over TCP or UDP. The communication will be managed by a *comm* sub-process on both devices. Figure 5 is a high level diagram of how the communication will occur.

5 Testing Issues

There will be multiple hardware dependent components to be built for the project and limited hardware ownership within the group. The project can be built in parts, however they will need to be brought together to test everything works in unison.

5.1 Types of Testing to be Conducted

We will need to regularly run hardware tests to ensure that we are using all the system resources efficiently.

We also need to run tests on the audio quality being captured. We have already found that even with the direction microphone, interference can be caused from the hardware circuitry.

Additional tests yet to be explored from our initial brainstorming session include:

1. *Power consumption:* a multimeter has been acquired to begin benchmarking power consumption of the *Raspberry Pi*® at various stages of operation.

2. *Computer vision target tracking:* as a way to simulate field testing, the team is exploring using lecture theatre video captures (available through channels such as OpenCourseWare, YouTube, etc.) to identify any issues that may arise from environmental factors. For example, poor lighting conditions.

6 Roles and Responsibilities

Due to an unfortunate circumstance, our final group did not form until week four. As a result, there are cross-overs in our teams individual strengths. The roles have been divided with some shared responsibilities for delivering the final solution. Although at different levels, all three group members will be writing software, documenting and experimenting with hardware.

6.1 Val Lyashov

Val has been working with computer components for many years. He will take on the role of designing the hardware solution and providing the operating environment. He will also be responsible for drafting some software interfaces to the hardware in Python, which will later be ported to C/C++ by the other members.

6.2 "Alfred" Yang Yuan

Alfred is an experienced programmer who has professionally worked with C++ for over 3 years. He will be responsible for the final software architecture and optimisation of the software layer. His role will also involve improving the code written by the other team members to be multi-threaded and run efficiently with the hardware.

6.3 Neil Ang

Neil has a strong programming background and a particular interest in intelligent system design. He will be responsible for implementing the subject tracking and writing some C/C++ interfaces to the hardware. He will also take on responsibility for the technical writing and leadership of the group.

Appendix A

Reflections of the Raspberry Pi Cross-Compiler Build

Did only one person build the image or did each member of the team do it?

Each member individually completed the cross-compiler task, but only one member was chosen to demonstrate it in the laboratory.

If more than one did it, did team members do it independently or did they do it collaboratively?

Every member completed it individually. The provided instructions were fairly straightforward so there was no need to work on it collaboratively.

Did they have any problems? If so how did they overcome them? What were the fixes?

Since we followed the instructions precisely there were no issues with the cross-compile. However, the instructions for setting up the .inputrc resulted in two members unable to press the 'c' key on their keyboard.

An improved version of the .inputrc script for backwards searching was later found on Stack Overflow[1]:

```
"\e[A":history-search-backward
"\e[B":history-search-forward
```

[1] http://stackoverflow.com/questions/1030182/how-do-i-change-bash-history-completion-to-complete-whats-already-on-the-line

Do they understand what the recipe scripts did? Why was a cross compiler necessary?

Val had prior knowledge of cross-compiling and understood the scripts the best. The cross-compiling was necessary to build an image that would be runnable on the *Raspberry Pi*® hardware.

Why did the script use uClib and not glibc? What are the trade-offs?

No sure. We didn't spend much time investigating this.

Did using this recipe help you gain confidence in using the UNIX command line? What man page entries did you use?

Each member already had a basic understanding of unix, so we were already confident with using the command line. We didn't find the need to look up any man pages to complete the task.

Bibliography

[1] Alon. *Control of Pololu Maestro Micro 6 channel servo control board*. 2010. URL: https://github.com/alon/pololu_maestro.

[2] Romain Beauxis et al. *Shine*. 2013. URL: https://github.com/savonet/shine.

[3] Chet Corcos. *Learning OpenCV with Xcode*. 2012. URL: https://sites.google.com/site/learningopencv1/installing-opencv.

[4] Pololu Corporation. *Pololu Maestro Servo Controller User's Guide*. URL: http://www.pololu.com/docs/0J40/all#5.h.1.

[5] Evangelos Georgiou. *Tutorial: OpenCV 2.4.3 Face Tracking — Detection using VS 2010 C++*. 2013. URL: http://mymobilerobots.com/myblog/academic/tutorial-opencv-2-4-3-face-tracking-detection-using-vs-2010-c/.

[6] UMass Graphics and Vision Research. *Realtime Tracking With a Pan-Tilt Camera*. 2010. URL: http://umassgv.blogspot.com.au/2010/07/realtime-tracking-with-pan-tilt-camera.html.

[7] Robin Hewitt. *How OpenCV's Face Tracker Works*. 2007. URL: http://www.cognotics.com/opencv/servo_2007_series/part_3/sidebar.html.

[8] Richard Hirst. *ServoBlaster*. 2013. URL: https://github.com/richardghirst/PiBits/tree/master/ServoBlaster.

[9] Skand Hurkat. *CarDetect*. 2011. URL: https://github.com/skandhurkat/CarDetect.

[10] Liz. *Facial recognition: OpenCV on the camera board*. 2013. URL: Facialrecognition:OpenCVonthecameraboard.

[11] Malakai. *Raspberry Pi Camera Extended documentation*. 2013. URL: http://www.raspians.com/raspberry-pi-camera-extended-documentation/.

[12] mojocorp. *Control of Pololu Maestro servo control board*. 2010. URL: https://github.com/mojocorp/Pololu_Maestro.

[13] Pierre. *OpenCV and Pi Camera Board !* 2013. URL: http://thinkrpi.wordpress.com/2013/05/22/opencv-and-camera-board-csi/.

[14] Mike Pultz. *Accessing Google Speech API / Chrome 11*. 2011. URL: http://mikepultz.com/2011/03/accessing-google-speech-api-chrome-11/.

[15] *raspberry-gpio-python*. URL: https://code.google.com/p/raspberry-gpio-python/.

[16] OpenCV dev team. *Cascade Classifier*. 2013. URL: http://docs.opencv.org/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html.

[17] OpenCV dev team. *GPU Module Introduction*. 2013. URL: http://docs.opencv.org/modules/gpu/doc/introduction.html.

[18] x-io Technologies. *Camera control and stabilisation via PC*. 2011. URL: http://www.x-io.co.uk/camera-control-and-stabilisation-via-pc/.

[19] Philipp Wagner. *OpenCV Face Recognition Lag*. 2012. URL: http://answers.opencv.org/question/3334/opencv-face-recognition-lag/.

[20] Wikipedia. *OpenMAX*. URL: http://en.wikipedia.org/wiki/OpenMAX.