University at Buffalo
Commencement System

Design Document

May, 2016

Revision History

Release Number	Date	Revision Description
1.0	5/11/2016	Initial Release

Table of Contents

1.0 - General Information

1.1 - Origins of the System

- The system was developed for a computer engineering senior capstone class for the University at Buffalo. Nine students worked on the system in two separate groups. Specifically a team which developed the web application and a team to develop the hardware and software. The system went through two major iterations before the final system was delivered.
- The system was built to solve the following problems:
    - The pronunciation of students' names is frequently done poorly by the readers during commencement. This problem is solved by pre-recording the names with the students' pronunciation preference in mind
    - It can be difficult to keep track of the order of the graduating students for reference for the photographer. It is currently done by taking a student's name card and putting it on a fishline as the students come up. The integration with the camera in the 'Eventual Goals' section of this document shows how this could be further improved

1.2 - Points of Contact

1.1.a - Professor Kris Schindler

- kds@buffalo.edu
- Oversaw the project during it first year of implementation
- Knowledgeable with technical system design

1.1.b - Christine Human

- chuman@buffalo.edu
- Coordinated logistics requirements of the system.
- Initial client and recipient of System Release 1.0

1.1.c - Katie Menke

- kemenke2@buffalo.edu
- Production manager for the commencement ceremony

1.1.d - Dave Jordan

- dsjordan@buffalo.edu
- Technical director at Alumni Arena

1.1.e - Eric Burlingame

- ebp3@buffalo.edu
- Audio technician at Alumni Arena

2.0 - System Summary

2.1 - Web Application Summary

2.1.1 - Overview

Re-recording web application for student names

- Utilizes Google App Engine host.
- Stores students' database in Google Cloud Datastore.
- WebAudioRecorder.js it used to record audio input from users.
- Generated unique links for each student through java hashcode.

2.2 - Hardware Summary

2.2.1 - Overview

- The scanning station accepts QR codes as inputs and plays audio files in response to the scanning event. The station continuously takes pictures, (via the camera) and attempts to decode them with a QR code decoding algorithm. If it successfully decodes the image, it passes the decoded information, in this case it is the scanners UBIT, to a lookup function. The lookup funciton ensures that there is a recording linked to that UBIT. If there is not a recording, negative feedback is provided. If there is a recording, the wav file is played out of the audio output device that is selected.

2.2.2 - Parts

Part	Detail
Raspberry Pi II	Model B
PiCamera	Compatible w/ Model B
White LEDs	5mm - 1.7V Super White
Piezo Buzzer	
RGB LED	5mm - 2.7V

3.0 - Getting Started

3.1 - Scanner Station

3.1.1 - Power Setup

- **Requirement** - The scanning station requires a 5V and 2.5 Amp (minimum) DC Power Supply via the micro USB port.
    - *NOTE*: A higher quality power supply is recommended as lower quality power supplies can generate noise on the audio output due to poor grounding or rippled input voltages.
- Procedure:
    - Connect micro USB cable to the Raspberry Pi on the scanner station
    - Plug power adapter into wall outlet

3.1.2 - Audio Setup

- *NOTE*: Audio output can be connected with two different connections, 3.5mm audio jack, or HDMI. The system can only output out of one device at a time.
- Procedure:
    - Power down the device
    - Connect the audio output cable to its port
    - Power on the device
    - Setup the audio output by entering the following command into a terminal
        - If using HDMI audio
            - “Amixer cset numid=3 2”
        - If using 3.5mm Jack audio
            - “Amixer cset numid=3 1”

3.1.3 - Physical Connections

3.1.3.a - Camera

- The camera has a ribbon cable connecting it to the Raspberry Pi on the scanner station.
- Procedure:
    - Power down the device
    - Open the ribbon cable port door on the scanner station.
    - Gently insert the ribbon cable.
    - Close the ribbon cable port door on the scanner station.

3.1.3.b - RGB LED

- The RGB LED has three connections to the scanner station Raspberry Pi
  - GND - (pin 34)
  - 3.3V - GPIO 19 (pin 35)
  - 3.3V - GPIO 26 (pin 37)
  - NOTE: Only connect and disconnect the connections when the device is powered down

3.1.3.c - White Illumination LEDs

- The White Illumination LEDs have two connections to the scanner station Raspberry Pi
  - GND - (pin 30)
  - 3.3V - GPIO 5 (pin 29)
  - NOTE: Only connect and disconnect the connections when the device is powered down

3.1.3.d - Piezo Buzzer

- The Piezo Buzzer has two connections to the scanner station Raspberry Pi
  - GND - (pin 14)
  - 3.3V - GPIO 27 (pin 13)
  - NOTE: Only connect and disconnect the connections when the device is powered down

3.1.3 - User Access Levels

- In order to access the GPIO pins which control the piezo buzzer, white LED's and RGB LED, the system must be run as *'sudo'*
- To run the scanner software in `*sudo*` mode, start the program with the command `*sudo python __init__.py*`

3.1.4 - Running Software

- Starting scanner software
  - Procedure
    - Navigate to folder `Desktop/scanner/`
    - Open a terminal window
    - Enter the command:
      - `sudo python __init__.py`
      - *NOTE*: If prompted for a password, enter the sudo password

4.0 - Name Recordings

4.0 - How to Obtain Student Names

- Potential graduates participating in graduation can come not only from the current semester, but possibly previous semesters, and even previous years. To generate a list of names of potential student participants, contact the school's office whose degrees are offered. They will be able to generate a list of students for current and previous semesters. If possible, obtain the each student's UBIT from the respective offices.

  These list will most likely just consist of full names. The system operates primarily on student UBIT's. You will need to obtain these unique identifiers. To do this, we have written a piece of software which when given list of student names, can generate the associated names. It does this by scraping the University at Buffalo's website and writing the data to text file.

4.1 - How to Record

- To record the student names, the use of a recording room in CFA was utilized. The use of this recording room ensures each sound file is free of unwanted noise. It also ensures all the recordings have the same volume and audio settings. The recording booth offers very specific sound setting adjustments. The settings that provide the best sound file are as follows:

Setting	Type	Value
Highs:	Knob	-2
Mids:	Knob	-1
Lows:	Knob	+1
Frequency:	Knob	+2
Microphone Gain:	Knob	~50dB
Channel Gain:	Slider	0dB
Master Gain:	Slider	0dB

4.2 - Recording Software

- SoundTrack Pro was the recording software used to record the names. The program offers many different file types for saving the audio files. A decision to use .wav file was made due its lossless compression. The settings used for the bitrate, and bit depth are as follows:

Setting	Value
Bitrate	192000 Hz
Bit Depth	32 bits
Endianness	Little Endian

5.0 - Device Software

5.1 - The Repo

5.1.1 - Program Source Files

- `__init__.py` is the entry point for the software, as well as the bulk of the runtime
- The only local import `__init__.py` requires is the adjacent file `ScannerIO.py`, which serves as an API to interact with the device status hardware

5.1.2 - QR Source Files

- `qr.py` defines a pair of convenience functions for decoding and encoding QR codes, `qrencode` and `qrdecode`
- `encoder.py` utitlizes the functions of `qr.py` to generate QR codes from a directory full of `.wav.` files, encoded by filename (omitting `.wav`)
- `qrs/` is a directory of QR coded `.png`s, representing the names of the recordings in the `namewavs` directory

5.1.3 - Log File

- The terminal output from the software running live at UB's 2016 SEAS Commencement Ceremony for the graduation Computer Engineers

5.1.3 - Archive

- A arbitrarily-organized directory containing previous versions source and media files

5.2 - The Scanner Software

5.2.1 - Software Summary

- The scanner software runs on the UB student developed Python source files `__init__.py` and `ScannerIO.py`
- Standard library imports
    - `subprocess` for issuing Linux commands in the background
    - `multiprocessing` symbols:
        - `Manager` for creating shared data objects

        - `Lock` for ensuring mutual exclusion of access to a shared data object at any given time
        - `Process` for spawning threads
    - `os` for `listdir`
    - `io` for create in-memory file object through `BytesIO`
    - `time` for `sleep`
    - `RPi.GPIO`
- 3rd party library imports:
    - `zbar` for QR code scanning
    - `PIL` symbol `Image` for image processing
    - `picamera` for accessing the device camera
- Flags, which can be set in the `__main__` block at the end of the file:
    - SCAN_THREAD_COUNT: Used to control the number of scan threads to spawn. Production version of the project ran 3 scan threads
    - LOOP_SLEEP: There is a sleep command set up in each thread loop, but currently this sleep time is set to zero. This runs just fine, but including some sleep times in the loops may reduce overheating of the device
    - SUPRESS_APLAY: Boolean that chooses whether the attempt audio output should be printed to `stdio`
    - LIGHTS_ON: Boolean that chooses whether the white LEDs above the scanning platform should be on or off
    - ENABLE_CAMERA_PREVIEW: Boolean that chooses whether a camera stream should be shown on the display

5.2.2 - Software Walkthrough

5.2.2.a - Initialization of `__init__.py`

- The `__main__` block at the end of `__init__.py` initializes the software as follows
    - Flags are set
    - The audio hardware is initialized
        - `amixer` currently sets the output to HDMI (last arg is 2)
        - `audio_primer` is a piece of custom software which reduces the delay of audio played through HDMI
    - Shared data structures are created
    - Parallel processes are started. The only process worth parallelizing multiple times is the scan thread, since the decode steps are the bottleneck of the software

        - One capture thread is created, as only one camera object can be instantiated at a time
        - The number of scan threads created depends on the corresponding flag
        - One say thread is created, as to avoid multiple announcements made in parallel
    - The `raw_input()` call is made to wait for a user to kill the program by hitting `Enter`
    - The processes are killed safely

5.2.2.b - The Capture Process

- Creates a camera object
- Loops the following
    - Saves an image to a stream object
    - Puts this stream onto the shared stream stack
    - Trims the stack if necessary to save memory
- The status light shows when camera capture is occurring. It's easy to see that most of this loop spends time in the camera capture

5.2.2.c - The Scan Process

- This function is executed across multiple processes
- A `zbar` scanner object is created
- Loops the following
    - A stream object is popped from the stream stack
    - The stream is analyzed using zbar
    - If symbols are detected, they are decoded and added to the data queue

5.2.2.d - The Say Process

- A hash data structure is created to detect if a wave file exists for a given user
- A set data structure is created to guard against repeated name announcements
- Loops the following
    - Data is pulled of the data queue
    - The data is compared to the two data structures mentioned above
    - If the data is a ubit name with a file that hasn't been announced, it is announce with `aplay`
    - The user gets a confirmation beep in this case, a denial beep otherwise

6.0 - Future Plans

6.1 - Needs To Be Done

- Confront heat problems: if the device is going to be run for an entire ceremony, it will be running for over an hour, which means it's likely to get very hot. During the 2016 commencement ceremony the device only had to run for about 2 minutes
- Software cleanup: The software is currently very stable, but could use some touching up. There are definitely some `continue` cases which don't handle everything that would normally happen at the end of a loop
- Scale: The system currently handles about 30 student audio files. If the project is to be scaled to the rest of SEAS or larger, it will have to handle thousands. Likely a database and an external storage source is needed
- User adjustability: Possible provide a GUI which offers a user controls for announcement delay, announcement volume, beep volume, any of the flags currently defined in the program
- Automatic logging: Both the output of the terminal during a run as well as the order of the students should be automatically logged to a file. The files should be updated in real time so that the records are persistent in case of a crash
- Appearance: Some paint would be nice

6.2 - Could Be Done

- Boot to program: make the system act like an embedded system. There is a configuration file at `/etc/supervisor/` which gives a user the option to issue some commands on startup
- Automatic deployment: This is a part of the project that would make this system a miracle for the different departments to use, as it would streamline the whole process from start to finish. Ideally it would work as follows:
    - Students would have a preferred pronunciation recording on file at all times. Perhaps they would be asked to update their recording every year during some sort of other UB student info update
    - During the weeks before a graduation ceremony, a list of eligible seniors for a given department would be generated. It would either be through scraping the UB databases, or from submission of a file from a department head if that wasn't possible, or any other way that makes sense
    - Unless this list already contains ubit names or person numbers, the scraper would run on the UB directory to acquire this information. These

are important because they are unique identifiers which can be used for each student
    - After the unique identifiers are acquired, an email is sent to remind department heads that it's time to have the names recorded. They would obviously be responsible for organizing the who, where, and when this occurs
    - The QR codes would be generated and ready to be printed. This year they were printed on 2x2 cardstock at FedEx Kinko's. It might be better to print them on rectangular cardstock next time to allow room for thumbs to hold them
    - A thumb drive would be populated with the wave files to be plugged into the device for that ceremony. The software on the device would have to be modified to locate the files at this location
    - The device runs at the corresponding ceremony!

6.3 - Eventual Goals

- Camera integration: We had the idea to integrate the system with a camera facing the students after their scan. The camera would automatically take a photo of the student he/she passes it, and their name would be matched up with their scan
- RFID: RFID could be used to quickly and more accurately check scan student tokens, but we were under the impression that QR codes would be a cheaper alternative, and easier to prototype