Software Overview

Year: 2019 Semester: Fall Team: 1 Project: IntelliFace

Creation Date: September 5, 2019 Last Modified: September 6, 2019

Author: Rtvik Sriram Bharadwaj Email: [bharadwr@purdue.edu](mailto:bharadwr@purdue.edu)

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Software Overview** | 5 | x2 | 10 |  |
| **Description of Algorithms** | 5 | x2 | 10 |  |
| **Description of Data Structures** | 5 | x2 | 10 |  |
| **Program Flowcharts** | 5 | x3 | 15 |  |
| **State Machine Diagrams** | 5 | x3 | 15 |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** | 5 | x2 | 10 |  |
| **Formatting and Citations** | 5 | x1 | 5 |  |
| **Figures and Graphs** | 5 | x2 | 10 |  |
| **Technical Writing Style** | 5 | x3 | 15 |  |
| **Total Score** | 100 | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

1.0 Software Overview

The IntelliFace module at a high-level consists of two firmware components, the STM32F0 MCU that handles asynchronous tasks and sensor readings, and the Nvidia Jetson which runs when invoked to perform high-computational tasks which include facial recognition, running the dashboard, and interlinking with the STM32F0 through USART to display sensor data.

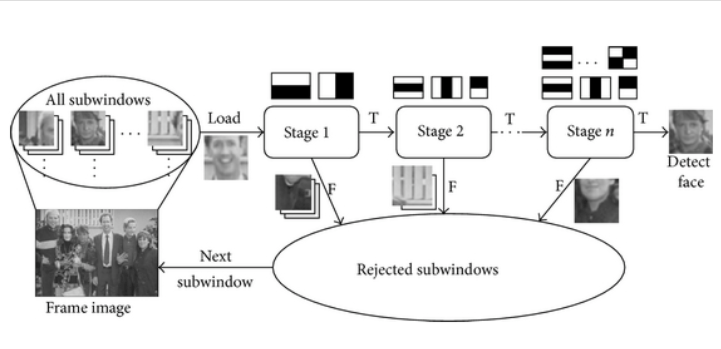
The STM32F0 based software is currently being written as a state machine, performing the asynchronous task of passively recording IR values to sense the user’s presence as well as recording temperature to monitor for an anomaly, and invokes the Jetson to run a command based on these inputs. The other task it performs are started once the user wakes the Jetson (in low power mode, described below) through the asynchronous tasks and is successfully authenticated, which is light detection to adjust screen brightness. At this stage, the IR sensor waits for the users command, and if the shutdown gesture if sensed, will send the Jetson into rest mode, and return to the asynchronous process state [1].

The Jetson in its initial state is on low-power mode monitoring the USART relay from the STM32F0 MCU, waiting for a temperature anomaly alert or a user presence detected alert. In the former case, the Jetson immediately moves into wake mode and utilizes a Slack or PagerDuty API to notify the user of the alarm. In the latter case, the Jetson boots up a passive webcam driver which records video and waits for a recognized user to be detected within a minute. In the successful recognition case, it will open the Jetson dashboard and proceed to read the temperature, light, and infrared values. It passively controls screen brightness using the ambient light values, displays the temperature values at a regular interval onto the dashboard, and asynchronously waits for user gesture detection using the values read from the IR sensor, all sensor values being passed through the USART channel. Upon successful exit gesture detection, it returns to a low-power mode. Additionally, In the case of a user recognition failure, or a system error during runtime (if any), it returns to a low-power mode.

During the UI operation, the Jetson performs background API requests, most notably stock from Yahoo, weather from AccuWeather, and news from CNN RSS feeds. It will also perform regular in-room temperature readings as read from the USART relay, and display any temperature anomaly alarms to the screen if required.

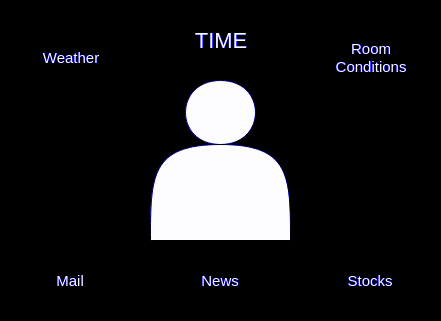
2.0 Description of Algorithms

The Jetson’s initial camera operation will rely on the Haar Cascade Classifier [2], which uses the Python OpenCV library [2]. This is a lightweight facial recognition image classifier which simply detects the presence of a person in the frame, the success upon which the Jetson captures a still frame and sends to the appropriate cloud-based facial recognition API after upload to an online Oracle Cloud database [5], which was picked due to the team’s familiarity with it and the availability of a large amount of free cloud credits due to a team member’s former work period at their cloud division. The facial recognition API returns a match percentage [4]. If the match level is sufficient for authentication, the UI is unlocked and displayed, as described below.



*Figure 1: Haar Cascade Algorithm Mechanism*

The UI is created and displayed using the Python Tkinter library (currently) for creating reactive dashboards [3]. It relies on multiple API requests sent to various servers contingent upon successful responses, upon which the dashboard is continuously updated. Tkinter was selected due to its familiarity from previous courses and work terms, namely from ECE 36400.



*Figure 2: Basic Dashboard Layout Plan*

The online APIs which the dashboard relies upon will be performed using carefully threaded operations to maximize the performance of the board, along with the sensor data read from the USART relay read from a different thread [6]. Threading is important for performance management when handling multiple operations simultaneously and synchronously, and our decision to proceed with Python as opposed to a more popular language for threading support like Golang and Erland was due to lack of exposure to these in previous coursework, as well as incompatibilities with the rest of the operations.

The APIs are all expected to be returned in a JSON format, which was chosen as it was the most widely available API format and allows the easy re-purposing of existing Python supported parsing functions.

The gesture detection will be performed through an asynchronous threaded reader on the Jetson side as sent from the STM32F0, and when the sensor is completely covered during wake state will trigger a shutdown. Similarly, when the sensor is completely covered during the sleep state, it triggers the Jetson wake mode. This is planned to be done using passive falling edge trigger readers on the Jetson side.

The STM32F0 component, primarily handling the passive sensor management, will be performed asynchronously using timers and interrupt handlers [1].

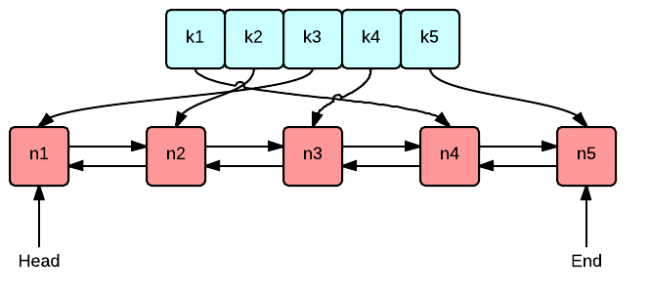
3.0 Description of Data Structures

The Jetson, performing the Haar Cascade Algorithm for initial user detection and image capture uses graph traversal techniques such as Depth First Search and bilinear interpolation in its image analysis to find a face, along with minor linear algebra matrix operations to handle the pixel data.

The APIs all rely upon JSON parsing, which is an inexpensive operation compared to the legacy systems primary method of field-bound data transmission, which is XML [8]. In consideration for the API method were also gRPC and protocol buffers which are benchmarked as being lighter and faster, which were however dropped due to the majority of the group’s unfamiliarity with this concept.

The captured images, if stored upon the Jetson would cause a storage issue long-term, and hence the storage of the data will involve uploads to an online database, currently set to Oracle Cloud Bare-Metal DBs due to their low-cost and high-speed. This will return a unique URL which contains the raw captured image, thereby negating any storage adversities. The image is scrapped immediately after upload, with handlers to ensure that the clean-up takes place as scheduled.

To map the captured image with the unique URL, we will be using a Least Recently Used (LRU) Cache mechanism of storage [7]. This involves a doubly linked list tied to a Hash Map, whereby older data is automatically ejected in a queue format, while preserving the latest image IDs and their unique URLs. This mechanism offers an O(1) time complexity for lookups and inserts, and a O(n) space complexity where n refers to the maximum number of images that are needed for storage.



*Figure 3: LRU Cache Architecture*

The facial recognition API used by all cloud-providers in consideration for our project require a sample training set of recognized users to successfully match the newly provided image with. While this is not implemented locally on our side, their models all utilize re-configurable neural networks to perform algorithms encompassing but not limited to point anchoring, linear regression, and classification.

As there is no data retention required on the STM32F0 MCU, and all sensors send data through the USART, there are no complex data structures required in this area. The data is to be streamed real-time as a comma separated line of data which is appropriately read, split, and analyzed by the Jetson.

4.0 Sources Cited:

[1] ARM STM32 (2019). *General-purpose timer cookbook for STM32 microcontrollers*. [online] Available at: https://www.st.com/resource/en/application\_note/dm00236305.pdf [Accessed 6 Sep. 2019].

[2] Docs.opencv.org. (2019). *OpenCV: Face Detection using Haar Cascades*. [online] Available at: https://docs.opencv.org/3.4.1/d7/d8b/tutorial\_py\_face\_detection.html [Accessed 6 Sep. 2019].

[3] Docs.python.org. (2019). *tkinter — Python interface to Tcl/Tk — Python 3.7.4 documentation*. [online] Available at: https://docs.python.org/3/library/tkinter.html [Accessed 6 Sep. 2019].

[4] Amazon Web Services, Inc. (2019). *Amazon Rekognition – Video and Image - AWS*. [online] Available at: https://aws.amazon.com/rekognition/ [Accessed 6 Sep. 2019].

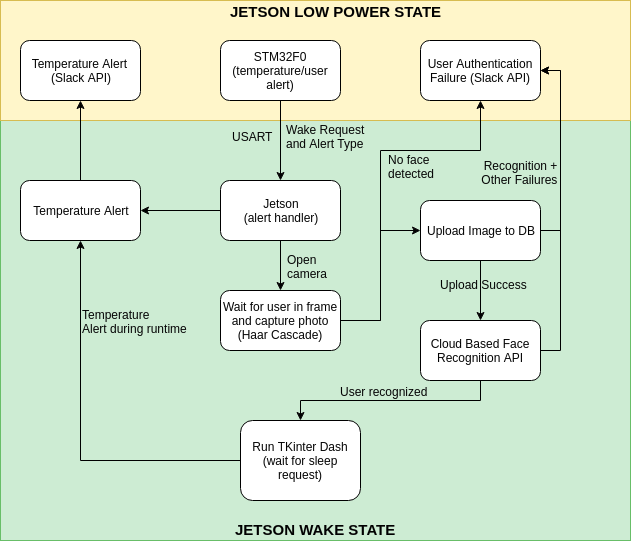
[5] Cloud.oracle.com. (2019). *Cloud Database on Bare Metal - Oracle Cloud Infrastructure*. [online] Available at: https://cloud.oracle.com/database [Accessed 6 Sep. 2019].

[6] Docs.python.org. (2019). *threading — Thread-based parallelism — Python 3.7.4 documentation*. [online] Available at: https://docs.python.org/3/library/threading.html [Accessed 6 Sep. 2019].

[7] Singhal, K. (2019). *How to implement LRU cache using HashMap and Doubly Linked List*. [online] Medium. Available at: https://medium.com/@krishankantsinghal/my-first-blog-on-medium-583159139237 [Accessed 6 Sep. 2019].

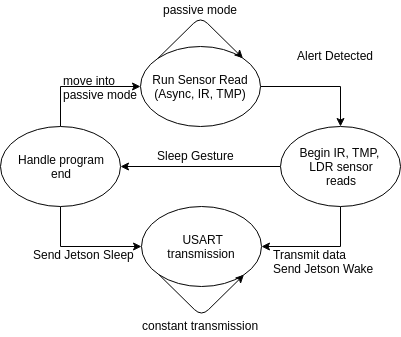
[8] Wyse, J. (2019). *Why JSON Is Better Than XML*. [online] Blog.cloud-elements.com. Available at: https://blog.cloud-elements.com/json-better-xml [Accessed 6 Sep. 2019].

Appendix 1: Program Flowcharts

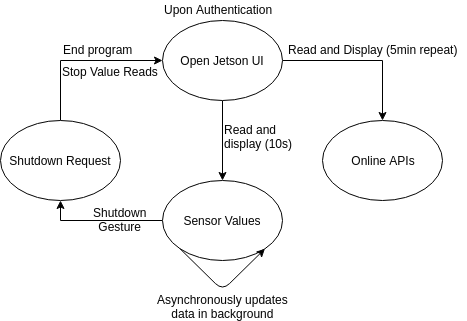
**

*Figure 1.1: Overall machine flowchart*

Appendix 2: State Machine Diagrams



*Figure 2.1: STM32F0 side State Machine*



*Figure 2.2: Jetson Dashboard side State Machine*