Project Final Report

EECS 488 - Embedded Systems Design
Case Western Reserve University

Steven Chen Hao Nyeo (cxn152)

May 9, 2019

Contents

1 Abstract

The purpose of this final document is to elaborate on the surveillance system developed during the Embedded Systems Design (EECS 488) class project: *A Low Cost System for Video Surveillance*. The document will include the approaches of tackling the implementation of surveillance systems with comparatively low installing and engineering cost. The sections will include the motivation and significance of the project, available products on the current market, personal approach of implementation and its methods, and also the outcomes of the project.

2 State of the Art

2.1 Closed-circuit Television (CCTV)

The Closed-circuit Television (CCTV), shown in Figure 1, is a product on the market that satisfy the video surveillance functionalities and is widely implemented on both public infrastructure and private properties for security and safety purposes. Depending on the quality, features, and the number of cameras during the installationhe cost of a CCTV system typically varies around 600 USD to 2,000 USD, and averages at around 1,339 USD [1].

Similar products on the market that would utilize surveillance cameras, shown in Figure 2, include the following:

2.2 Car Dash Camera

The car dash camera is a surveillance product on the market that is used to monitor and record driving conditions and possible incidents on the road. The system typically consists of a sole camera that focuses on the front end of the car to monitor upcoming traffic. The cost of a typical car dash cameras are around 70 USD to 300 USD [2].

2.3 Monitoring Cameras for Pets and Infants

The pet and infant cameras are built specifically for monitoring the actions pets and infants in households when direct interaction with pets and/or infants is

Figure 1: Closed-circuit Television (CCTV) [4]

not accessible. Typical costs for the pet and infant cameras are approximately 30 USD to 300 USD [3].

3 Motivation

The Raspberry Pi circuit board is a multi-purpose single-board computer that is often used in hardware and software projects requiring signal controlling and processing. Because of its low cost at around 35 USD and its ease of programmability [7], the Raspberry Pi is widely acknowledged as an inexpensive alternative for chips built specifically for certain signal processing tasks. In the case of this surveillance system project, the Raspberry Pi acts as the main embedded device that executes the surveillance functionalities, including movement and human detection.

After the evaluation of the installation cost of a readily available surveillance system on the market, it is seen that the cost of surveillance system could possibly be lowered by using the Raspberry Pi as the customized medium of communication between devices of a surveillance system.

The goal of the project is to lower the cost of implementing a stable, reliable, and resource-efficient surveillance camera system. The Raspberry Pi boasts its advantage in a low amount of power usage and its customizability, yet is still equipped with adequate features and performance to conduct image processing. In this project, the majority of the design will be focused on programming the functionalities of the surveillance system on the Raspberry Pi.

Figure 2: State of the Art

(a) Car Dash Camera [5]

(b) Pet Monitor [6]

4 Program Design and Features

The main parts of the surveillance system include video capturing, a finite-state machine, frame differencing, and human detection. The system is written in C++, heavily utilizes the OpenCV2 library, and could be compiled and executed on any machine with C++ and OpenCV (2 and above) installed. As for the alarm system, curl should also be installed on the computer for remotely sending a Gmail notification through the Simple Mail Transfer Protocol (SMTP) services provided by Gmail. Hardware requirements include surveillance cameras that could be accessed by the OpenCV library, the most common being USB cameras or Webcams. The computing unit, preferably a Raspberry Pi circuit board, should be able to support image processing and plug-and-play of the cameras.

4.1 The Mat Class

The *cv::Mat* class from the OpenCV library can be used to store an array of values [8] and is commonly used to instantiate Mat objects for storing image frame pixels. An instantiated Mat object contains a header that stores the information regarding the starting address of the matrix, and a pointer for reading and writing values to the matrix. [9] Therefore, each pixel could be accessed by assigning a pointer to a certain address within the Mat object, which benefits the counting of pixels during the implementation of frame differencing. In the project, a Mat object instance is created to store frames captured by the camera in the form of a matrix.

4.2 OpenCV VideoCapture

The program should be launched with at least one camera connected to the host device. The *VideoCapture* object will be instantiated to store the camera pipe. The *imShow* function will display the frame in a window. An infinite while loop will continuously update the frame object and display the frame image, generating a stream of images that mimicks video recording. Since each frame is separated individually from each other between updates, the frame object could be used to process detection of movement with the frame differencing algorithm [10], which will be explained in section 4.4.

The capturing and storing of frames is demonstrated with the code below:

```
VideoCapture cap(video0);
if (!cap.isOpened())
    cout << "Unable to open camera" << endl;

while (1) {
    Mat frame;
    cap >> frame;

    /* Code for state machine */

   imshow("Video", frame);
}
```

4.3 Finite-state Machine

The basic structure of the program will be modeled after a finite-state machine, in which each of the event listed in the Basic Operation section of the project prompt will represent a state. The state transition diagram is shown in Figure 3.

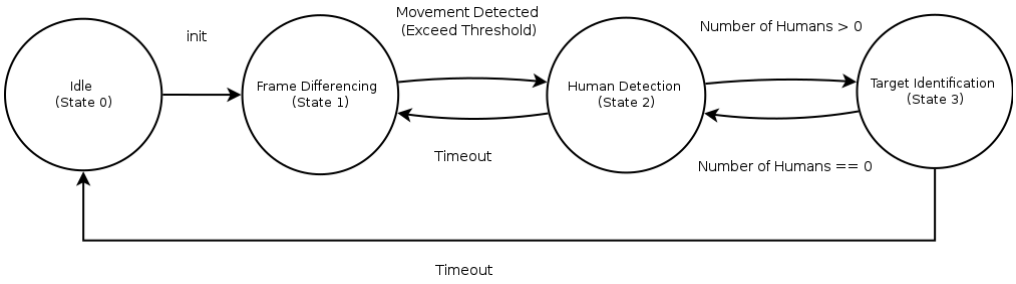


Figure 3: State Machine Design for the Surveillance System

Each state will have its own modules, within contains the functions necessary to fully execute each event:

- **State 0:** Idle

  In state 0, no action is taken and the embedded device goes into power saving mode.
  The state machine could transition to State 1 when movement detection is turned on.

- **State 1:** Movement

  In state 1, the state machine will keep scanning for movement using the implemented frame differencing algorithm. The threshold of movement could be altered and defines the trigger of state transition. The state machine will transition to State 2 when movement larger than the defined threshold is detected. Otherwise, the state machine will stay in State 1. The frame differencing algorithm and the threshold definition will be explained in section 4.4.

- **State 2:** Approaching Target

  In state 2, the state machine will trigger the functionalities for human detection for a defined limited amount of time before automatically transitioning back to State 1 to save computing power. The state machine will transition to State 3 when a human target is identified. Otherwise, the state machine will stay in State 2 and attempt to identify human figures in the picture. Human detection will be explained in section 4.5.

- **State 3:** Suspicious Target

  In state 3, the state machine will draw out the target on the frame as long as the number of humans identified on the screen is greater than zero. If the number of targets of frame is equal to zero, then the state machine would transition back to state 2. Otherwise, the state machine will stay in state 3. The state machine will trigger the alarm if the state machine maintains in state 3 for more than the defined limited amount of time. Once the alarm is triggered, the state machine will send an email to the designated email address of the user.

- **Manual Mode:**

  The remote owner takes control and the state machine halts. This state cannot be achieved through state transition and is not integrated into the state machine.

The code below depicts the structural overview of the state machine:

```
switch (state) {
   case 0:
      /** Power saving mode */
      break;

   case 1:
      /** Frame differencing algorithm */

        // state transition from 1 to 2 and start timer for movement
            timeout
        if (moved_frame_count > MVNT_THRESHOLD) {
         state = 2;
        } else {
            state = 1;
        }
        break;

    case 2:
      /** Human detection algorithm */

        state = 2;
        // If human is detected then start timer for human detection
            timeout
        if (human.size() > 0) {
         start = time(0);
            state = 3;
        }
        // If current time elapse times out then transition to state 1
        else if (time_elapse(start, time(0)) > DIFF_TIMEOUT)
         state = 1;
      break;


    case 3:
      /** Human detection algorithm */

        if (human.size() > 0) {
         /** Drawing human detection */
         state = 3;

         if (time_elapse(start, time(0)) > HUMAN_TIMEOUT) {
            /** Send alert */
         }
        }
        else
         state = 2;
      break;
}
```

4.4 Frame Differencing

The frame differencing algorithm is implemented with the comparison between two frame objects. Before entering the while loop, a Mat object is generated to store the previous frame captured by the camera. Inside the while loop, another Mat object object is created to store the current frame captured after the previous frame. Using the operator-overloaded subtraction of the Mat object, the difference between each individual pixel from the current frame and the previous frame could be calculated using the minus sign. The pixels from the returned Mat object will indicate the differences at each pixel position. e.g. if the frame differencing image is black, then it shows that all the pixels in the current and previous frame have no difference, and therefore shows 000 as the color.

If the colors between a certain pixel of the current frame and the previous frame are the same, the difference of the pixels would be zero. Otherwise, the color difference will be quantified in floating point numbers. The differences of the pixels wil be stored in another Mat object. After the difference between the pixels is stored, the previous frame is updated by the current frame, and the while loop continues to capture the next frame [11].

The code below demonstrates the calculation of frame difference:

```
Mat prev_frame;    // Capture and store previous frame
cap >> prev_frame;

while (1) {
    Mat frame;
    cap >> frame;

    /** Calculate frame difference */
    Mat frame_diff = frame - prev_frame;

    /** Update previous frame with current frame */
    prev_frame = frame.clone();

    imshow("Video", frame);
}
```

Using the matrix provided by the Mat object storing the pixel differences, a threshold (*MVNT_THRESHOLD*) could be set to indicate the sentivity of movement detection.

With a for loop and a pointer at the header of the Mat object, the frame differencing algorithm could traverse through the whole matrix with a pointer and identify the magnitude of the pixel differences between the frames in the form of floating numbers. In this case, the floating point numbers for each pixel will be typecasted to integers so that the integers could be used for movement identification. In other words, since typcasting a double will truncate the floating points, the pixels with typecasted values greater than 1 would be marked

as true and those with typecasted values less than 1 would be marked as false.

If the typecasted value of a certain pixel exceeds 1, the algorithm will increment *moved_frame_count*, a variable that keeps track of the total number of pixels with values greater than 1 in the differenced frame.

After the whole matrix of data from the Mat object is traversed, the *moved_frame_count* variable will be compared to *MVNT_THRESHOLD* for movement detection. If the number of marked pixels is greater than the threshold, movement in the frame is considered detected and the state machine will transition its state.

The code below demonstrates the calculation of the difference between pixels from the current and previous frame:

```
#define MVNT\_THRESHOLD 300
```

```
float* pixels = &frame_diff.at<float>(0);
int moved_frame_count = 0;

switch (state) {
case 1:
    /* Count moved pixels */
    for (unsigned int i = 0; i < 307200; i++)
      if ((unsigned int) pixels[i])
         moved_frame_count++;

    /* State assignment and transition */
    state = 1;
    if (moved_frame_count > MVNT_THRESHOLD) {
      start = time(0);
      state = 2;
    }
    break;
```

To save computing power and prevent sampling errors caused by camera glitches that will be mentioned in section 7.1, the number of pixels examined could be reduced. Instead of traversing through the whole matrix, the for loop could skip a certain number of pixels just enough to determine movement. For example, for a camera with a resolution of 640 x 480 resolution, there are 307200 pixels. Instead of processing all 307200 pixels, the for loop could process just 3072 pixels, with each chosen pixel having a distance of 100 pixels from the next chosen pixel. Note that this approach should not be overdone to retain accuracy, and that the threshold should also be adjusted to match the original sensitivity of movement detection.

The code below is an approach that attempts to speed up the pixel-counting process:

```
#define MVNT_THRESHOLD 30
```

```
/* Count moved pixels */
for (unsigned int i = 0; i < 3072; i += 100)
    if ((unsigned int) pixels[i])
        moved_frame_count++;
```

4.5 Human Detection

The human detection functionality implemented in the surveillance system uses the cascading classifiers available from the OpenCV GitHub repository as XML files. Initially, the XML file defining the numeric values of a human face or body is loaded into a CascasdeClassifier object.

Depending on the loaded XML file, this technique could be applied to both facial recognition and human detection [12]. In the case of this project, since the cameras are expected to be installed on the roof of the factory, the program will load the full body XML and function as if the camera is attempting to identify pedestrians.

Using the frame grabbed and retrieved from the camera, the detector compares the templates loaded from the XML file to the frame and identifies pixel groups in the frame that closely resembles the numeric values from XML file. The algorithm draws a rectangle around the identified object. Note that the frame should be translated to black and white before feeding the frame to the cascade classifier in order for the cascade classifier to generalize colors into gray scales and avoid having to recognize various colors.

```
/* Initial setup for cascade classifier */
CascadeClassifier detectorBody;
detectorBody.load("haarcascade_fullbody.xml");

/* Convert the picture to black and white */
cvtColor(frame, gray_frame, CV_BGR2GRAY);
detectorBody.detectMultiScale(frame, human, 1.1, 2, 0 | 1, Size(40, 70),
    Size(80, 300));
```

Another candidate for implementating human detection is using the OpenCV Histogram of Oriented Gradients (HOG). However, due to accuracy and performance disadvantages in the default detectors, the cascading classifier technique is chosen over HOG [13].

4.6 Timing

Timing plays a significant role in the project since timing conditions are used in the system to trigger state transitions. Assume that a human target is identified for a certain period amount of time that, instead of being recognized as

a bypassing pedestrian, would in fact be considered suspicious. This time limit could then be defined as the timeout for state transition.

Similarly, we can apply the same assumption on voids of human detection. Whenever the human detection algorithm fails to identify humans in the frame for longer than a certain amount of time, it can be assumed as if the movement that resulted in the enabling of human detection is in fact not caused by humans. Therefore, after this amount of time is achieved, it is convincing enough for the algorithm to give up on human detection and fall back to frame differencing.

Below is a helper function used throughout the program for calculating time elapses.

```
/* function for time elapse */
double time_elapse(time_t start, time_t end) {
   return (double) end - (double) start;
}
```

The code below shows the basic logic of how timing triggers a certain action, and in the case of this project, timing triggers state transition.

```
time_t start;
// if the current time and the start time differs more than TIMEOUT then
    action
if (time_elapse(start, time(0)) > TIMEOUT) {
   /** Action */
}
```

Throughout the project, we can see that timing is used in the cases mentioned above.

```
time_t start;
while (1) {
    switch (state) {
      case 1:
        /** Frame Differencing */

          /* state transition from 1 to 2 and start timer for movement
              timeout */
          if (moved_frame_count > MVNT_THRESHOLD) {
           start = time(0);
           state = 2;
          } else {
              state = 1;
          }
          break;

       case 2:
         /** Human Detection */

          state = 2;
```

```
            // If human is detected then start timer for human detection
                timeout
            if (human.size() > 0) {
             start = time(0);
                state = 3;
            }
            // If current time elapse times out then transition back to
                state 1
            else if (time_elapse(start, time(0)) > DIFF_TIMEOUT)
             state = 1;

          break;

        /** Drawing human detection */
        case 3:
            if (human.size() > 0) {
             /** Draw human */
                state = 3;

                // If elapsed time since start is greater than the defined
                    timeout, send alert message
                if (time_elapse(start, time(0)) > HUMAN_TIMEOUT) {
                  /** Send alert message! */
                }
             }
             else
                state = 2;
          break;
    }
}
```

4.7 Sending Email Notification

In the project, the alarm system does not contain a physical alarm bell. The *alarm* function will trigger curl on the computer to send an email message via the Gmail SMTP service.

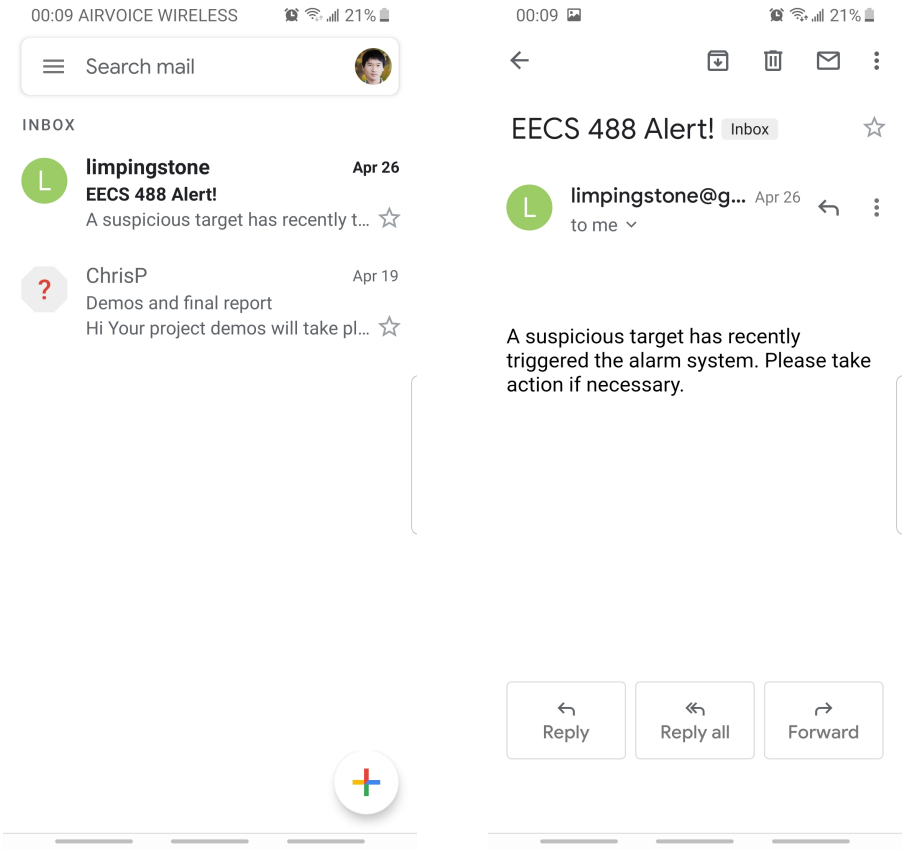
The SMTP service with curl is transplanted from Daniel Stenberg's libcurl example [14] and integrated into the *alert* function as the default alarm notification mechanism for the surveillance system. Figure 4 shows the preview of the email notifications received when the alarm is triggered.

5 Current Product

5.1 Installation

The program is distributed with the main.cpp file, the cascade classifier XML file, and a makefile. To build the system from the source code, the operating

Figure 4: Email Notification Preview



(a) Alert Email Sent to Inbox

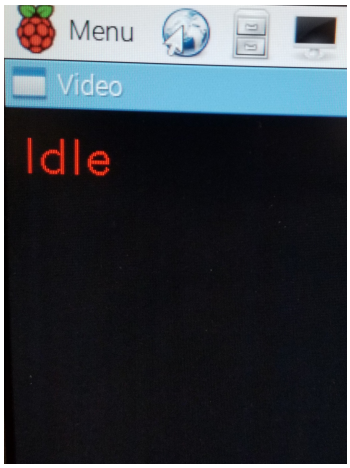
(b) Alert Message

system will require OpenCV2, the g++ compiler, and curl installed. By typing in the *make* command into the UNIX terminal, the compiler will generate a binary file.

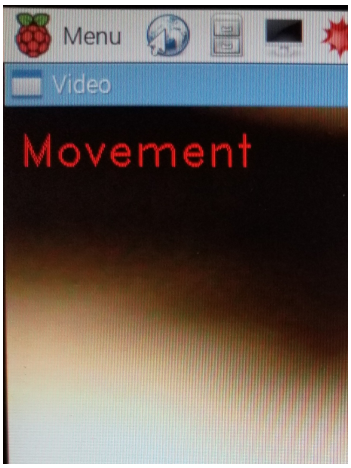
5.2 Display Interface

Once the program is built, the user will be able to launch the surveillance program by executing the compiled binary file. The program will begin displaying the frames captured from the camera. The program will display the current state of the surveillance system on the upper left corner of the frame. Figure 5 shows the program interface during frame differencing.

Figure 5: User Interface of the Surveillance System during Frame Differencing



(a) Idle State



(b) Movement Detection State

Figure 6 shows the program interface during human detection. The elapsed time and current status of the program is also elaborated on the console in the background.

6 Results and Analysis

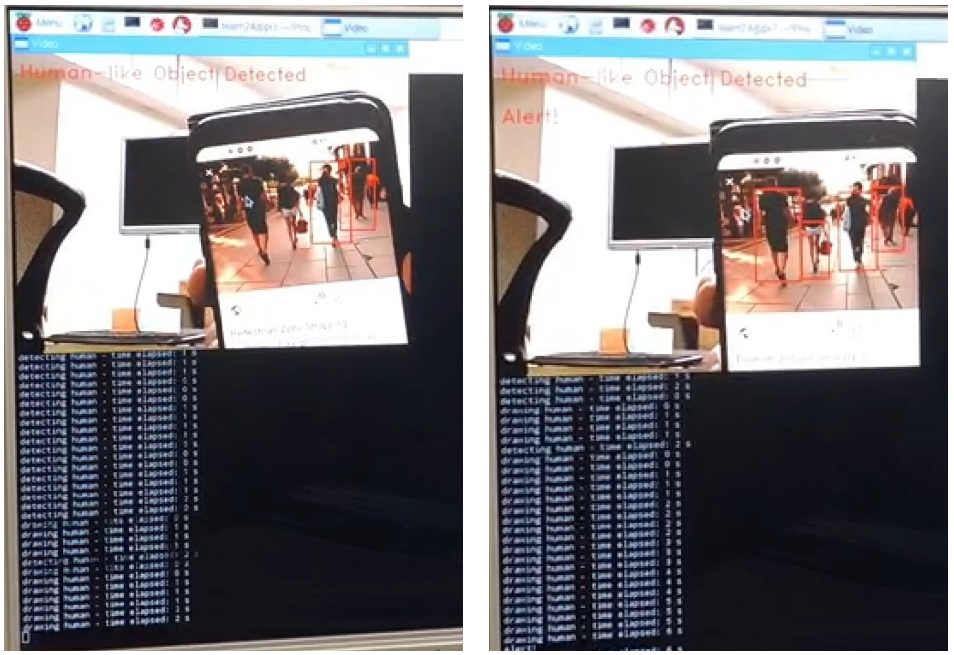
6.1 Performance

The Raspberry Pi is able to handle frame updating at around 10 frames per second. The program sees no significant decrease in performance when the frame differencing algorithm is implemented. Therefore, the performance of the frame differencing algorithm does not impact the overall performance on updating the frames. However, when the human detection mechanism is launched, the frame refreshing frequency decreases to around 3 frames per second, indicating that with the use of the cascade classifiers, the human detection algorithm requires extensive CPU computing power.

Another candidate for human detection is HOG, mentioned in section 4.5. HOG runs at around 0.5 frames per second on the Raspberry Pi, which is even slower than the cascade classifier technique. In addition, human detection using HOG lags much more than the cascased classifier technique.

Attempts on multithreading to increase performance of the cascade classifier technique for human detection have been tested with both OpenMP and p-threads. Yet, the effect of implementation did not see a noticeable increase in the frame rate since the semaphores in the multithreading process prevent the frames from simultaneously showing up using the *imshow* function.

Figure 6: User Interface during the Human Detection State



(a) Human Detected

(b) Timeout and Sending Alert

Another attempt for performance improvement was experimenting on skipping every other frames in order to prevent the frames from lagging behind as the process load builds up with time. The attempt did not improve performance since the capturing of the next new frame is achieved only when the human detector is done processing the current frame.

6.2 Cost Analysis

The table below depicts the cost of the physical components of the project. The cost of each individual cameras, the micro SD card, and the alarm system is based on an estimated current retail price on Amazon.com. For future development of the system, a 128 GB SanDisk is chosen as the sample to incorporate recording and frame storing functionalities. The price of each individual camera is obtained by searching for surveillance cameras, and the price for the alarm is estimated using the manufacturing costs of fire alarms. The total estimate of budget cost to fully implement a reliable surveillance system would be around 286.90 USD, which is significantly less than the average of 1,339 USD for a CCTV system from the market.

Cost Entry	Units	Total Cost (USD)
Raspberry Pi	1	$ 35.00
Cameras	8	$ 207.92
Micro SD Card	1	$ 19.99
Alarm System	1	$ 23.99
Total		$ 286.90

Table 1: Items Cost

6.3 Engineering Cost

The design and coding of the current program are not included as the accounting cost of the surveillance system. Installing the complete surveillance system will require time to install 8 cameras, install the program on a Raspberry Pi, and secure the communication between the 8 cameras and the Raspberry Pi with either wired or wireless options. With one technician at the scene, this should all take around 4 to 5 hours depending on the magnitude of the factory.

6.4 Varying Cost

Other varying costs include monthly fees for internet connection, and the total cost varies depending on the time duration of the surveillance system on duty. Maintenance costs of the program will differ according to how critical the software vulnerability or hardware damage is.

7 Future Prospects

7.1 Improvements

Below is a list of improvements identified that could enhance user satisfiability of the surveillance system:

- **Performance of Human Detection**

  As mentioned in the performance section 6.1, it is seen that when human detection is launched, the frame rate significantly decreases to 3 frames per second. The surveillance system will require a more efficient approach of human detection to be able to support higher frame resolutions, meanwhile able to decrease the time delay caused by comparison with the cascading classifier XMLs.

  Hypothetical approaches of solving performance issues is to conduct parallel programming with multiple Raspberry Pi's, yet as mentioned in section 6.1, having multiple threads did not increase the frame rate, and therefore the hypothesis is less probable. Another approach is to outsource the computing portion to personal computers or cloud computing servers,

yet this approach requires stable and reliable internet connection to avoid being bottlenecked by slow internet speed or not being able to retrieve computing results from the cloud server and cause the program to crash.

- **Camera Glitches**

  Camera glitches occassionally occur and could cause the frame differencing algorithm to accidentally trigger human detection.

  An approach to prevent the frame corruption problem from influencing the frame differencing algorith is to simply remove and skip the corrupted frame altogether. This could be done by detecting the integrity of the frame before feeding the frame to the frame differencing algorithm.

  However if removing corrupted frames is not available for implementation, another way to alleviate the errors of processing corrupted images is by only testing sample pixels instead of testing every single pixel in the Mat object, as mentioned in section 4.4. Although this solution cannot completely solve the frame corruption problem, the solution increases the possibility to neglect a certain group of pixels in the frame to influence the number of pixels marked as moved.

- **Multiple Cameras**

  Originally, the prompt stated that there should be 8 cameras around the factory. This project, however, has not shifted its focus to having multiple instances of camera active simultaneously. There were versions where multi-threading and process forking came in handy as solutions for this problem. Yet for some vendors, there exists some unresolved conflicts between two and more cameras. This project focuses to have one Raspberry Pi solely deal with one camera, as handling the load for image processing with 8 cameras would be far beyond the demand of one Raspberry Pi.

7.2 Viable Features

- **Container for the Frame Window**

  The *imshow* function is provided by the OpenCV library. The function will launch a window on the graphics user interface with the frame captured from the camera. Since the window is frequently opened and closed with each iteration of the while loop, the program will seem as if the frames are updated.

  However, since the windows are continuously updated, the window cannot be closed by clicking on the close button like other programs. Whenever the window receives the kill signal, a new window with a new frame is generated by the while loop. So far, the only way provided by many OpenCV programs is detecting whether the escape key (Esc) is pressed. The surveillance system anticipates a more user-friendly interface that is responsive to mouse clicks.

- **Recording Functionality**

  As mentioned in section 6.2, users can utilize the additional storage space on the Raspberry Pi to store recordings of the frames for future reference and evidence tracking. This could be a promising feature to be incorporated in the future.

8 Summary

This project regards the implementation of a surveillance system on a Raspberry Pi. The major focus of the project has been on the smoothness of transition between the states for the state machine. Currently, the program user interface could clearly display the status of the camera by printing the state on the screen without any state transition failures. This is a significant step in the development of the program in that the foundation of the project - the state machine - does not crash the whole system.

After the state machine was developed, the focus shifts to efficient ways of communicating the user with alerts when the alarm system has been triggered. With the implementation of curl into the system, the program is able to send an email whenever the trigger has been set off. The user would be notified nevertheless that there has been an incident at the factory and should visit the scene for details.

This project has its flaws and points of improvements in the future. However, the program has so far demonstrated that it is possible to combine frame differencing, human detection, timing, and alert sending into one state machine effectively and reliably.

9 References

References

[1] HomeAdvisor, "2019 Security Camera Installation Costs — CCTV Surveillance System Prices," *HomeAdvisor*, 2019. [Online]. Available: `https://www.homeadvisor.com/cost/safety-and-security/install-a-surveillance-camera/`. [Accessed: May 9, 2019].

[2] Anthony Alaniz, "The Best Dash Cams for Your Next Road Trip," *Popular Mechanics*, 2019. [Online]. Available: `https://www.popularmechanics.com/cars/how-to/g9/5-dash-cams-tested/`. [Accessed: May 9, 2019].

[3] Safety.com, "The Best Pet Cameras," *Safety.com*, 2019. [Online]. Available: `https://www.safety.com/pet-monitors/`. [Accessed: May 9, 2019].

[4] OSRAM (2019). `https://media.osram.info/im/img/osram-dam-1457779/c,x,0,y,764,w,4800,h,2664/s,x,1260,y,0/727326_OS_awareness_safety_picture_camera.jpg`. [image] Available at: `https:`

//www.osram.com/os/applications/surveillance-cctv/index.jsp. [Accessed: May 11, 2019].

[5] https://images-na.ssl-images-amazon.com/images/I/61BpozoUvSL._AC_UL200_SR200,200_.jpg. (2019). [image]. [Accessed: May 11, 2019]

[6] https://m.media-amazon.com/images/I/61hTOr-fwnL._AC_UL436_.jpg. (2019). [image]. [Accessed: May 11, 2019]

[7] Raspberry Pi Foundation, "Featured Products," *Raspberry Pi Foundation*, 2019. [Online]. Available: https://www.raspberrypi.org/products/. [Accessed: May 9, 2019].

[8] OpenCV, "cv::Mat Class Reference," *OpenCV*, 2015. [Online]. Available: https://docs.opencv.org/3.0.0/d3/d63/classcv_1_1Mat.html. [Accessed: May 9, 2019].

[9] OpenCV, "Mat - The Basic Image Container," *OpenCV*, 2019. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html. [Accessed: May 9, 2019].

[10] OpenCV, "How to open usb cam instead of webcam in VideoCapture?," *OpenCV*, 2018. [Online]. Available: http://answers.opencv.org/question/188821/how-to-open-usb-cam-instead-of-webcam-in-videocapture/. [Accessed: May 9, 2019].

[11] Stack Overflow, "How to find the differences between frames using OpenCV?," *Stack Overflow*, 2014. [Online]. Available: https://stackoverflow.com/questions/9998195/how-to-find-the-differences-between-frames-using-opencv. [Accessed: May 10, 2019].

[12] Funvision, "Fast Opencv people pedestrian detection Tutorial," *funvision.blogspot.com*, Mar. 11, 2016. [Online]. Available: https://funvision.blogspot.com/2016/03/opencv-31-people-detection-at-13-fps-by.html. [Accessed: May 10, 2019].

[13] Madhawa Vidanapathirana. "Real-time Human Detection in Computer Vision - Part 1," *Medium*, 2018. [Online]. Available: https://medium.com/@madhawavidanapathirana/https-medium-com-madhawavidanapathirana-real-time-human-detection-in-computer-vision-part-1-2acb851f4e55. [Accessed: May 10, 2019].

[14] Daniel Stenberg. "libcurl example - smtp-mail.c," *curl*, 2017. [Online]. Available: https://curl.haxx.se/libcurl/c/smtp-mail.html. [Accessed: May 10, 2019].