FIT3161 Project Design & Prototype Specification for Group 5: Remote Video Surveillance System Project

TABLE OF CONTENTS

1. Synopsis

Our project, proposed by Clayton Campus tutor Daniel Jitnah, involves the act of constructing a scalable multi-client video monitoring surveillance system. The project involves employing multiple computers as nodes in the system, with a central computer acting as the main surveillance network, and utilizing USB webcams hooked to computers, to perform motion-detection of its area. The aim of this proposed system is to address resource concerns commonly seen in other public surveillance models whereby now, video streams will be sent in short clips to the server only when valid motion has been detected by the camera's surroundings. A motion detection technique known as background subtraction will be the chosen methodology by which we will perform motion detection, with the ViBe method proposed by Marc Van Droogenbroeck being the main backbone by which we will create our motion detection algorithm. A more extended, and higher-level functional requirement sheet is delivered by our tutor and guide Daniel, and will be documented below. This document aims to specify and design a high-level architecture for the system, detailing important software modules, components, and data flow which will occur across the system. A detailed plan for software quality control will be developed via architectural drivers, which will allow us to test the functionality, accuracy, and reliability of our system at any point during development. Architectural drivers drive & guide the design of the software architecture and system.

2. Architectural Design

2.1 Project Decomposition & Data Flow

We firstly decompose our system architecture by illustrating the physical data flow of the system via a systems design flowchart, identifying inputs, outputs, and the flow of data. We also define several additional flowcharts that helps illustrate the main components of our system.

Data Flow Diagram

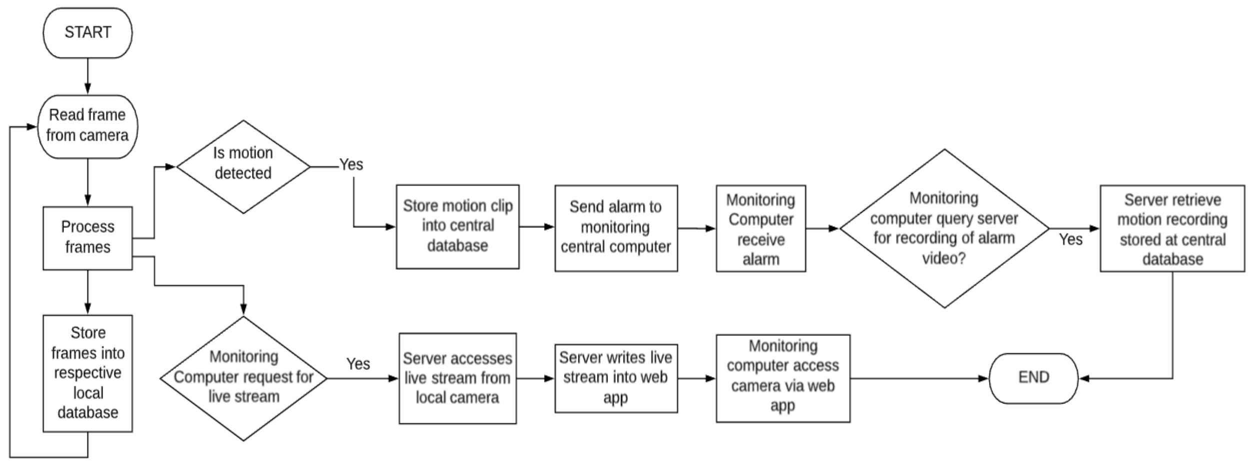

START
Read frame from camera
Process frames
Store frames into respective local database
Is motion detected
Yes
Store motion clip into central database
Send alarm to monitoring central computer
Monitoring Computer receive alarm
Monitoring computer query server for recording of alarm video?
Yes
Server retrieve motion recording stored at central database
Monitoring Computer request for live stream
Yes
Server accesses live stream from local camera
Server writes live stream into web app
Monitoring computer access camera via web app
END

Figure 1: Data Flow Diagram for Remote Video Surveillance System

Level 0 System Context Diagram

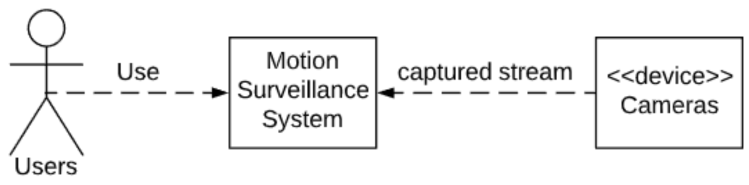


Figure 2: Context Diagram for Remote Video Surveillance System

Level 1 System Container Diagram

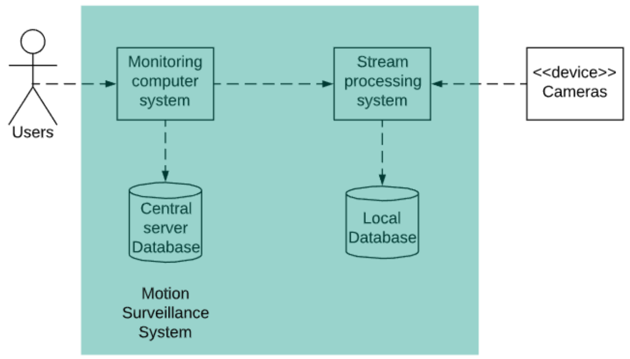
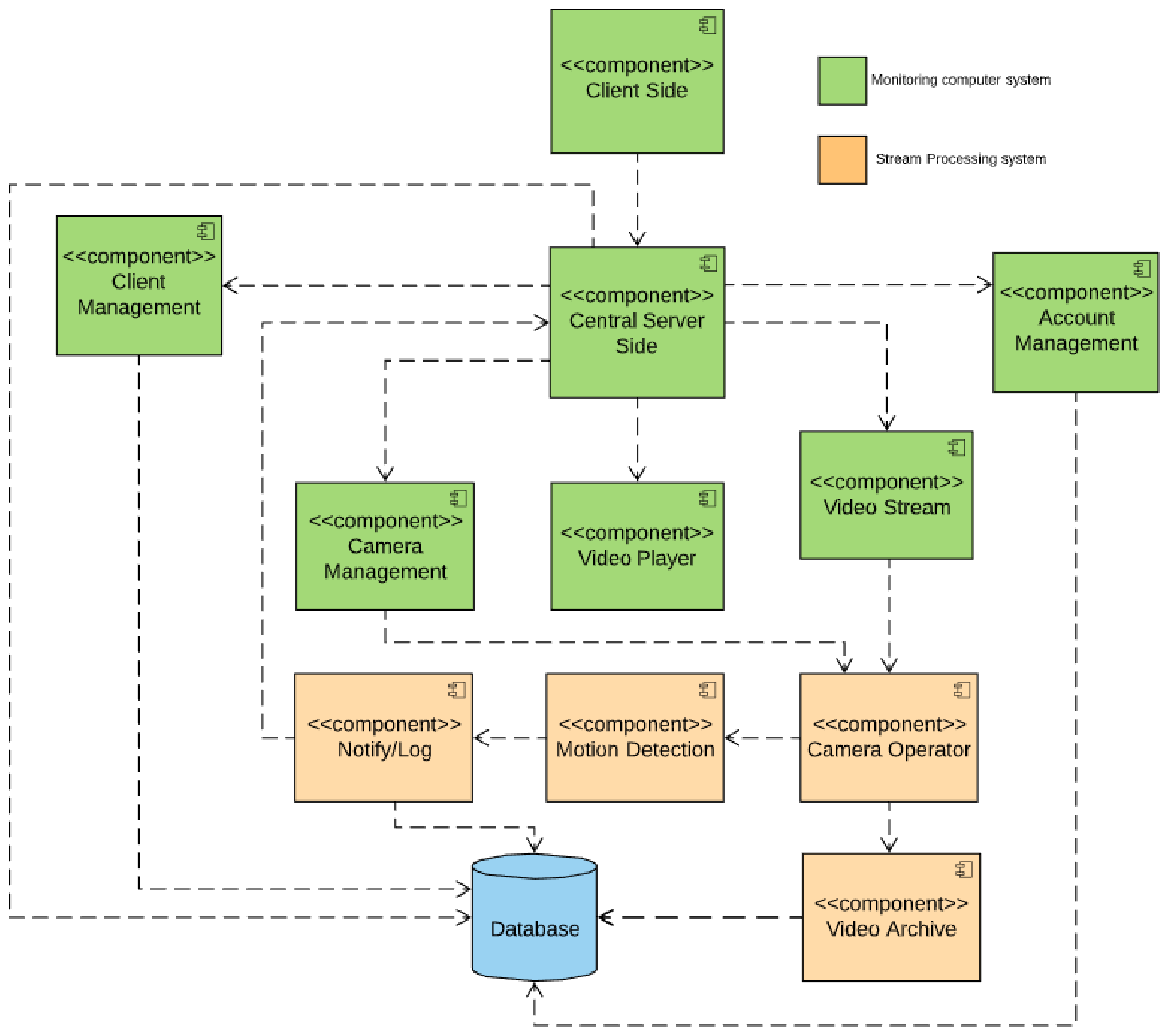


Figure 3: Container Diagram for Remote Video Surveillance System

Level 2 System Components Diagram



<<component>> Client Side
Monitoring computer system
Stream Processing system
<<component>> Client Management
<<component>> Central Server Side
<<component>> Account Management
<<component>> Camera Management
<<component>> Video Player
<<component>> Video Stream
<<component>> Notify/Log
<<component>> Motion Detection
<<component>> Camera Operator
Database
<<component>> Video Archive

Figure 4: Components Diagram for Remote Video Surveillance System

Background Subtraction Flowchart

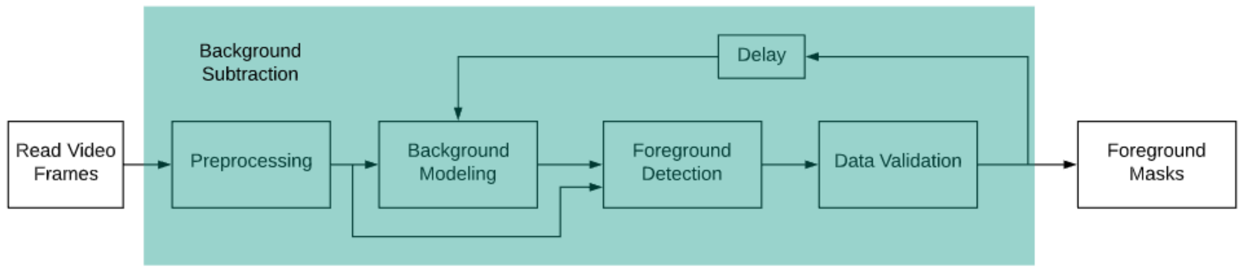


Figure 5: The Flow Diagram of a generic background subtraction algorithm (Shekhawat, 2011)

2.2 Software Design Modules

Several important software modules are required for the success of our system. These software components have been highlighted in Figure 4 of our components diagram. A deeper, more extensive list of the specific functionalities required is detailed below.

- Process of each local camera performing live video streaming → storing video data in a specific format → writing to local hard drive
- Process of sending video snapshot during motion capture to central server database
- Motion Detection algorithm
- Local computer database Schema (has many event logs due to constantly streaming)
- Central server database Schema (has many clients → has many webcams)
- Web Application Interface & Design Plan (Javascript using MEAN STACK)
- Live Streaming performed on web application server using Node.js technologies
- FTP server for handling transmissions between local computers and the central sever

2.3 Architectural Drivers

2.3.1 High-Level Functional Requirements

A set of High-Level functional requirements are derived based on the project specifications delivered by Daniel in order to facilitate and encapsulate project scope, direction, and progress.

1. Ability to record video frames from a camera and store it into a local database
2. Ability to process video frames into a valid video format for viewing
3. Ability to detect motion from a camera and issue an alert message to the central monitoring computer
4. Ability by a local computer to store snapshot of its video upon motion detection to the central database, hence the snapshot can be played anytime.
5. Ability for the central monitoring computer to view any camera stream connected to it on demand
6. Ability for the central monitoring computer to store all motion detection clips alongside a valid schema for storing event logs, client information etc.
7. Ability for the central monitoring computer to view past streams from any local camera via accessing the local computer’s video database.

8. Ability from the central monitoring computer to associate different cameras to different clients and manage this association (via database management)
9. Ability to carry out management tasks on the monitoring computer via interaction with a web application
10. Ability to perform all desired activities from the central monitoring computer onto each local computer via a robust web application.
11. Ability to design a valid and responsive motion detection algorithm embedded into all local IP cameras using the background subtraction methodology via ViBe.

2.3.2 Quality Attribute Requirements

Quality attributes represent measurable properties of a software system. They allow us to assess the maintainability, usability, testability of our system. We propose several quality attribute scenarios of several qualities which will help measure the success of our software system.

Performance Qualities

1. Video frames processed and stored in the database must be of the same quality as the camera's live stream.
2. Processing time for writing a specific timestamp of the live stream into the central server upon motion detection must be fast (5-10 seconds).
3. Web application developed to perform live streaming and management operations must be fast when performing web and database operations

Reliability & Availability Qualities

1. Web application developed must be robust and free from fatal crashes.
2. If a local camera happens to be unavailable, the central monitoring computer is able to immediately notice.
3. Every local camera must be able to consistently write video frame data towards its respective local data store with accurate timestamps without delays.
4. Central monitoring computer is able to tap into any local camera at any time without fail.

Usability Qualities

1. The supervisor operating the central monitoring computer should be notified of motion detection of an unspecified site in a way that is clear.
2. Usability of the web application developed to inspect live streaming and perform management operations should be easy to use and intuitive.

2.3.3 Architectural Constraints & Concerns

1. Security of local camera's being accessed by unauthorized users
2. Performance of viewing multiple live streams at once from the central computer
3. Scalability of handling larger bandwidth transmissions between central server and X local cameras
4. Scalability of central storage database, particularly if we outsource to a cloud service such as MongoDB

3. Software & Hardware

3.1 Software Specifications

3.1.1 C++ with OpenCV

C++ programming environment will be used solely for the purpose of handling the functional requirements for video stream processing. During any video camera stream, the video will consist of multiple frames per second which make up the stream. To process this, we utilize a crucial library called OpenCV for read/write operations. We plan to utilize the C++ compiler for performing video read/write operations which offers fast computation. The IDE used for development will potentially be Visual Studio due to its support for C++. C++ will also be used to develop our motion detection algorithm, which will be embedded into all the IP cameras of the system.

<u>Reading Video Procedure</u>
We create a **VideoCapture** object via OpenCV to store the video stream into a specified file. Use functions **imshow()** to read the video and display the video frame by frame in a window, and **waitKey()** to pause each video frame in the video ; by passing the millisecond as parameter into it. (E.g. If we read a 40-FPS video, the number of milliseconds we pass into **waitKey()** is 25, if we read and display each frame in 25ms).

<u>Writing Video Procedure</u>
To write the video stream as a video file, we create a **VideoWriter** object via OpenCV. Parameters we supply to the object are the output file's name, the file format, FourCC, which is a 4-byte code specifying the video codec, the number of FPS, and the frame size. Hence a sample declaration of a **VideoWriter** will be of the form:
cv2.VideoWriter('output.avi',cv2.VedioWriter_fourcc('M','J','P','G'),10,(frame_width,frame_height))

3.1.2 Javascript with MEAN Stack

We will use Javascript with the MEAN stack to build a web application which can service all the central monitoring functionality alongside perform management operations on our system.

Web Application UI and Design Functionality

The web application's User Interface will be built using Angular.JS, a client-side framework offered from the MEAN Stack. It allows us to create a dynamic and responsive web interface. The application will offer the live streaming of all local cameras in the system, and the ability to perform management operations such as altering daytime/nighttime monitoring specs, performing CRUD operations on the database etc.

Web Application Database

The central database which will store the client's information, alongside motion capture recordings of all local cameras connected to the central computer, is established via MongoDB, another MEAN stack tool. The user will be able to query this database via the web application in order to play a certain motion-detected video.

Web Application Server-Side Routing

The Web Application will be connected to a server using Node.js & Express.js server-side framework offered by the MEAN Stack. This allows the user to perform routing operations such as tapping into the IP camera of any local camera.

3.1.3 FTP Server

Our central server will need to implement a file transfer protocol (FTP) in order to satisfy the functionality of sending past video clips stored in local computers, into the central computer's database. The FTP server network will be established in the system by ensuring all hardware is connected over the same Local Area Network (LAN). After establishing an FTP, the monitoring computer will be able to query central server to access any IP camera from any local computer, and is able to tap into the live stream for any computer. The central server may potentially use Real Time Streaming Protocol (RTSP) to view the IP camera's live stream.

3.1.4 Google Virtual Machine

As a reliable prototype server, we will first utilize Google's Virtual Machine to act as our server for our system.

3.2 Hardware Specifications

As a system which should be able to demonstrate scalability, There will always:

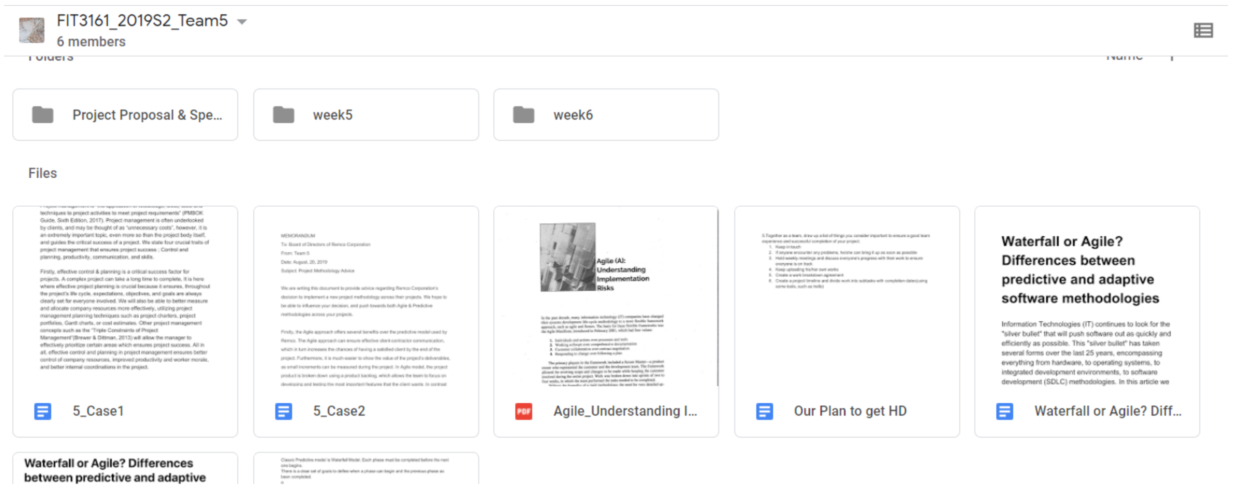
1. One Central Monitoring Computer – Running the web application, performing live streaming, management operations, view past clips etc.
2. One Central Server – Handles transactions between Central computer and Central Database, and also handles the live feed between Central computer and any local cameras
3. One Central Database – Stores all video clips where motion detection has been found across local cameras. Also stores all relevant information about the camera's information, and client information.
4. Wireless network system – Ensuring all nodes are connected in the same system to facilitate data transfer across devices
5. X number of Local Cameras – Will be local computers (laptops) hooked with a USB webcam. Performs live streaming of their current location and continuously writes image to its hard disk. Implements a motion detection algorithm which will notify the central computer.

3.3 Collaborative Tools

Being a group project, we plan to utilize several collaboration tools to facilitate project delivery.

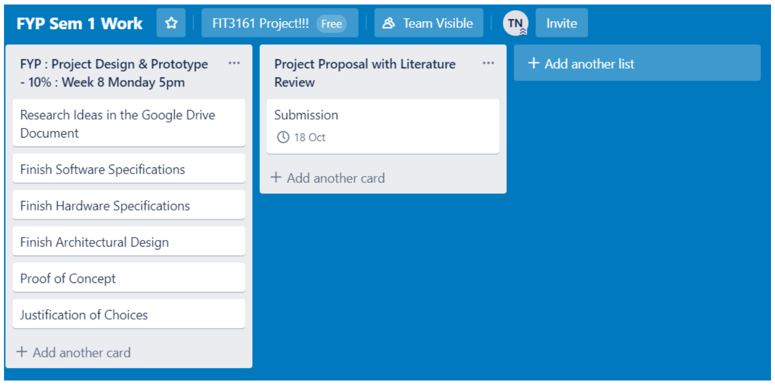
3.3.1 Google Drive

Reliable file storage & synchronization service which allows us to collaborate on the same document which creating deliverables. A small snapshot of our work with Google Drive is below.


FIT3161_2019S2_Team5
6 members
Project Proposal & Spe...
week5
week6
Files
5_Case1
5_Case2
Agile_Understanding I...
Our Plan to get HD
Waterfall or Agile? Diff...
Waterfall or Agile? Differences between predictive and adaptive software methodologies
Waterfall or Agile? Differences between predictive and adaptive

3.3.2 Trello

We will use Trello, which is a list-making application, allowing us to collaboratively manage and track the team's progress towards completing certain aspects of the project. Each team member can assign new tasks to perform.



FYP Sem 1 Work | FIT3161 Project!!! Free | Team Visible | TN | Invite

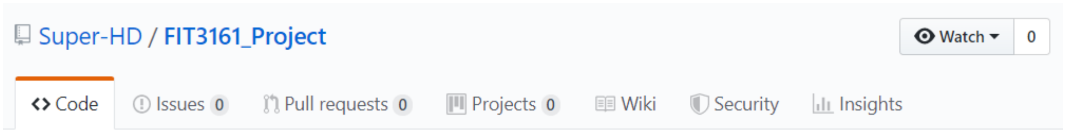
FYP : Project Design & Prototype - 10% : Week 8 Monday 5pm
- Research Ideas in the Google Drive Document
- Finish Software Specifications
- Finish Hardware Specifications
- Finish Architectural Design
- Proof of Concept
- Justification of Choices
- + Add another card

Project Proposal with Literature Review
- Submission 18 Oct
- + Add another card

+ Add another list

3.3.3 GitHub

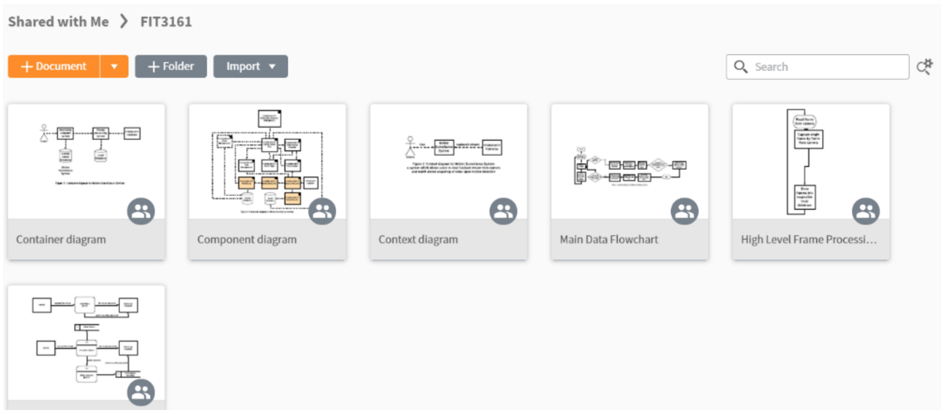
GitHub will be our main platform for establishing a collaborative software development environment. We are all educated on the topic of version control, and will make sure to develop code progressively in a push/pull manner. A private GitHub repository has been established within our group, which will be the repository where our entire system development software with reside in. Version control will allow us to manage our code progress, and revert back changes if anything happens to go wrong throughout our development timeline.



Super-HD / FIT3161_Project

3.3.4 LucidChart

We have and will keep using LucidChart to collaboratively create important flowcharts and diagrams that are required for project delivery.



Shared with Me > FIT3161

4. Justification of Choices

4.1 Development Language

Choosing C++ over Python for video stream processing

When using OpenCV (powerful library for image processing) for Python, we realize that it is just a wrapper around the original C/C++ code, meaning, when we call a function in OpenCV in Python, the code is actually running the source at C/C++. Hence, the native python OpenCV implementation runs much slower than C++ native implementation, which is the main reason that we chose to develop our video stream processing environment and motion detection algorithm in C++.

4.2 Motion Detection Algorithm

We are going to use the ViBe (Visual Background Extractor) algorithm, a background subtraction method to implement the motion tracking function. Background subtraction is one of the most widely used methods in motion detection and is performed by separating out foreground objects from the background in a sequence of video frames. This is done by embedding a "background model" as the normal state of a scene. We then determine if motion has occurred by moving objects by determining the difference between the current frame, and the background model's frame.

Most techniques rely on a probability density function (PDF) or statistical parameters of the underlying background generation process. Afterwards, the question arises as to how the which background sample should be updated. Parks et al. distinguishes between recursive techniques, which maintain a single background model that is updated with each new video frame, and non-recursive techniques which maintain a buffer of *n* previous video frames and estimate a background model based solely on the statistical properties of these frames (Droogenbroeck & Barnich, 2014). We have researched several motion detection methodologies before selecting ViBe, such as Optical Flow & Frame Subtraction. Optical flow is the distribution of apparent velocities of movement of brightness patterns in an image (Shafie & Ali & Hafiz, 2009). This can arise from relative motion of objects and the viewer, giving important information about the spatial arrange of the objects viewed and the rate of change of this arrangement (Shafie & Ali & Hafiz, 2009). The frame subtraction method compares the current image frame with the previous image to find the change in the value of pixels above threshold to detect motion. We believe that background subtraction is the preferred methodology as frame subtraction is not an accurate method, and optical flow is a more complicated, and mathematically intensive algorithm which is not preferred due to system performance constraints.

The background updating strategy for ViBe will incorporate three mechanisms. (1) a memoryless update policy, ensuring a smooth decaying lifespan for the samples stored in the background pixel models. (2) a random time subsampling to extend the time windows covered by the background model models. (3) a mechanism that propagates background pixel samples spatially to ensure spatial consistency, and to allow the adaptation of the background pixel models that are masked by the foreground. (Droogenbroeck & Barnich, 2014).

A step-by-step approach to the ViBe algorithm, demonstrated by Droogenbroeck & Barnich, will be the method by which we implement our background subtraction methodology on all local cameras within our system.

1. We start by defining the pixel model to estimate the background.
2. Supply a number of sample pixels for each pixel in the model.
3. Supply a matching threshold between a pixel value and a sample.
4. Supply the number of matches needed to incorporate a pixel into the background of the scene.
5. This approach will compare the classifying value of a point to its closest values among the set of samples by defining a sphere around this point.
6. A pixel value is then classified as background if the cardinality of the set intersection of this sphere and the set of samples is above a given threshold.
7. To handle new motions that appear in the screen, the model has to update regularly.
8. Furthermore, to ensure spatial consistency of the whole image model, and to handle some likely practical situations such as small camera movements or slowly evolving irrelevant background objects, a spatial information propagation method should be implemented.
9. With the above implementation, the ViBe algorithm could afford the use of a strictly conservative update scheme and deal with several concomitant events evolving at various speeds.

4.3 Central Computer Management Capabilities

The scope of capabilities that the central computer can perform upon its local computers are yet to be concluded. We aim to develop a system which will allow ease of extension if additional features are desired. Some management tasks that we have thought upon are of the following:

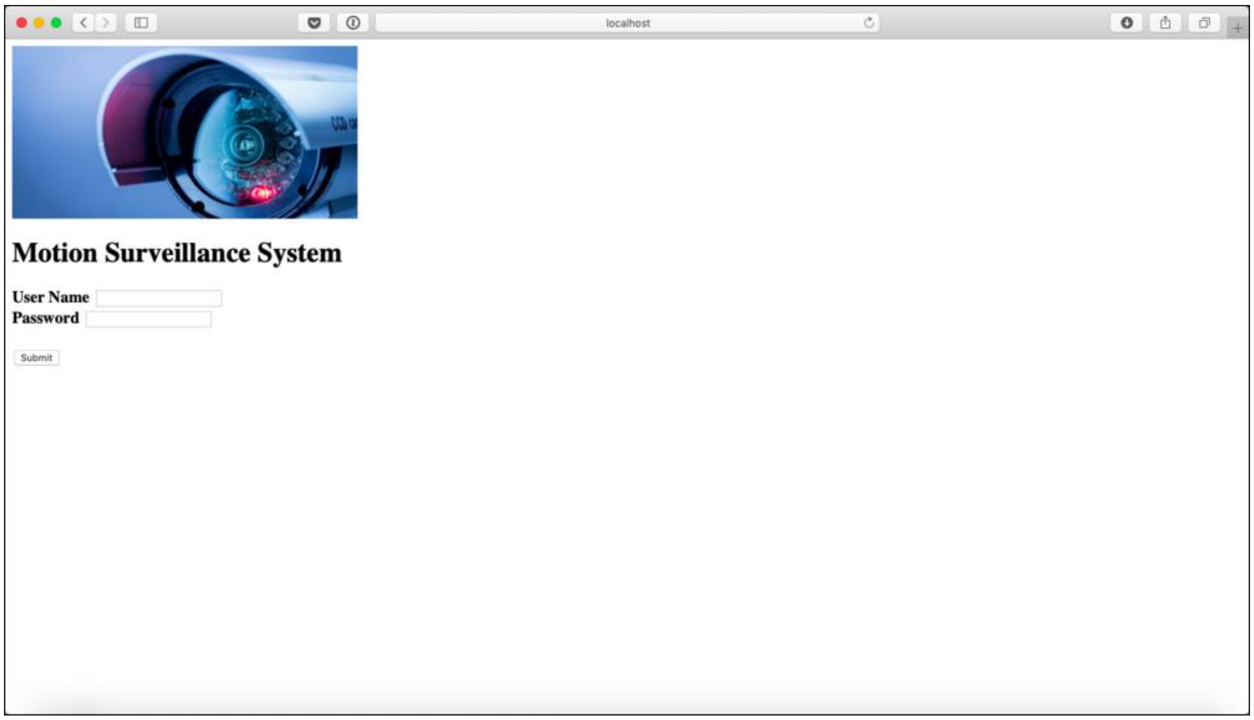
1. Able to perform CRUD (Create, Read, Update, Delete) on the local video storage of each camera
2. Able to alter the functionality of the motion detection algorithm being run on each local camera such as:
   - Disable/Enable motion detection at any time
   - Alter the criteria of detection depending on environment variables (how fast a motion should occur to constitute as detection, daytime/night-time differences)

5. Proof of Concept

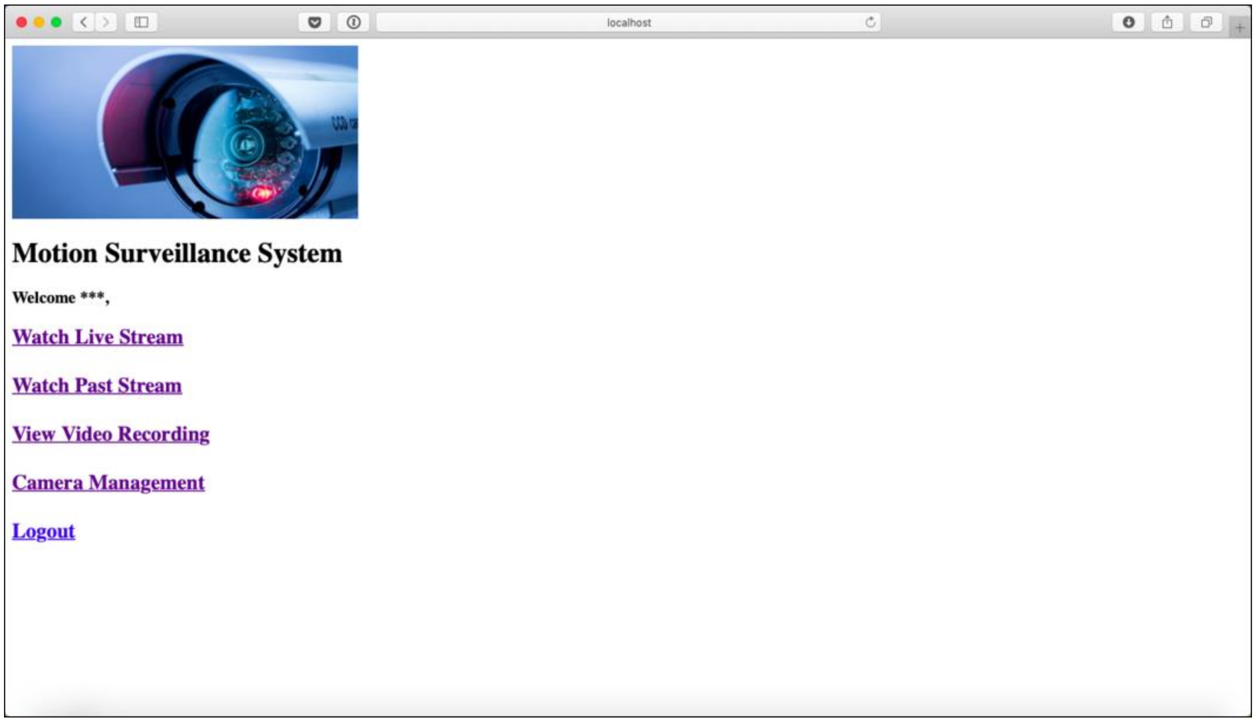
5.1 Web Application Prototype Concept

The web application will be the main module that the supervisor interacts with. It consists of three components: browser component, central server, and the database server. The server side will be developed using Javascript, MEAN Stack, and an FTP implementation. The supervisor will handle the presentation logic of the application. A prototype web application has been developed with six main interfaces: Login page, Home page, Live Stream Page, Past Stream Page, Motion-Detection Recordings Page, and Camera Management Page. So far, the web application only runs on the computer’s localhost, future implementations will extend it towards an online server, accessible by all nodes in the system.

Login Page – Prompts the supervisor to input a username and password to start using the web application.



Home Page – Displays all functionalities of the system.


localhost

Motion Surveillance System

Welcome ***,

Watch Live Stream

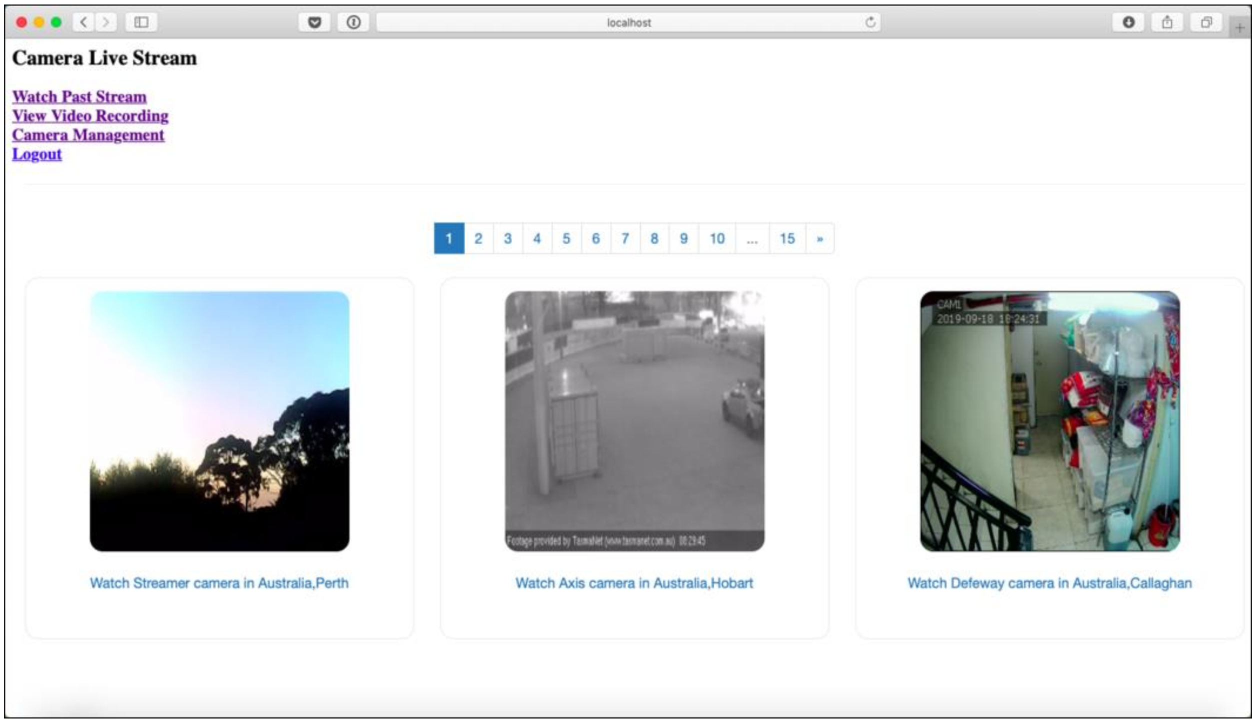
Watch Past Stream

View Video Recording

Camera Management

Logout

Live Stream Page – Shows all available camera's live stream. The supervisor can click on any specific camera to direct to that cameras page, displaying a better video resolution.


localhost

Camera Live Stream

Watch Past Stream
View Video Recording
Camera Management
Logout

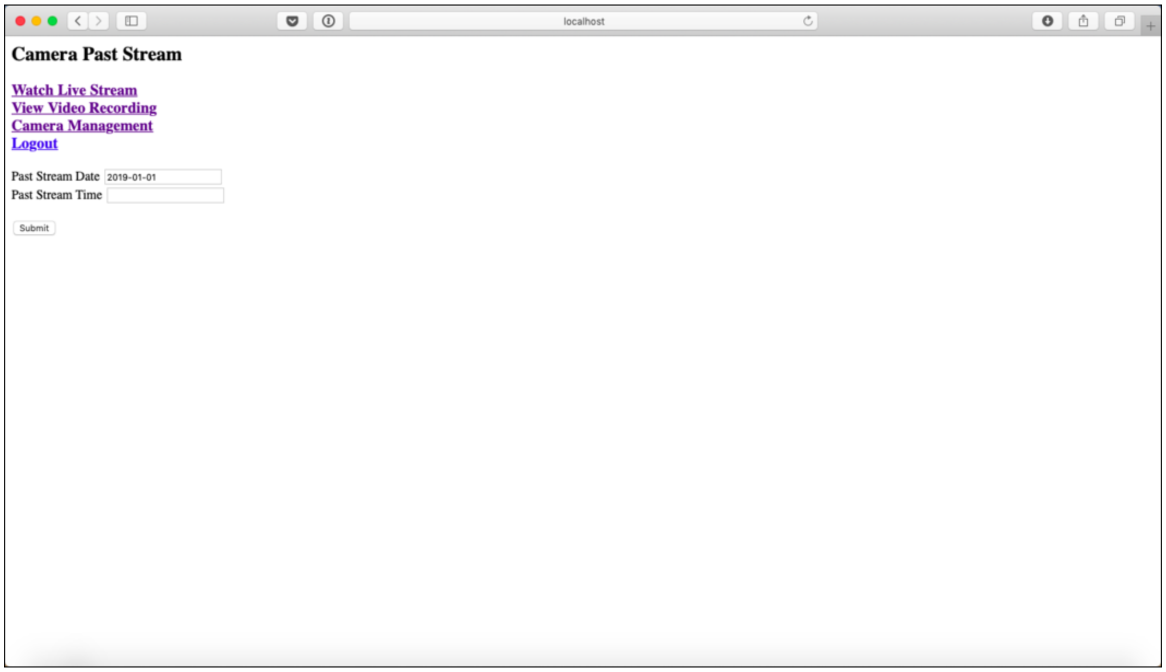
1 2 3 4 5 6 7 8 9 10 ... 15 »

Watch Streamer camera in Australia,Perth

Watch Axis camera in Australia,Hobart

Watch Defeway camera in Australia,Callaghan

Past Stream Page – Prompts supervisor to input the date and time of the past stream they intend to watch. After submitting, the web application plays a past stream.

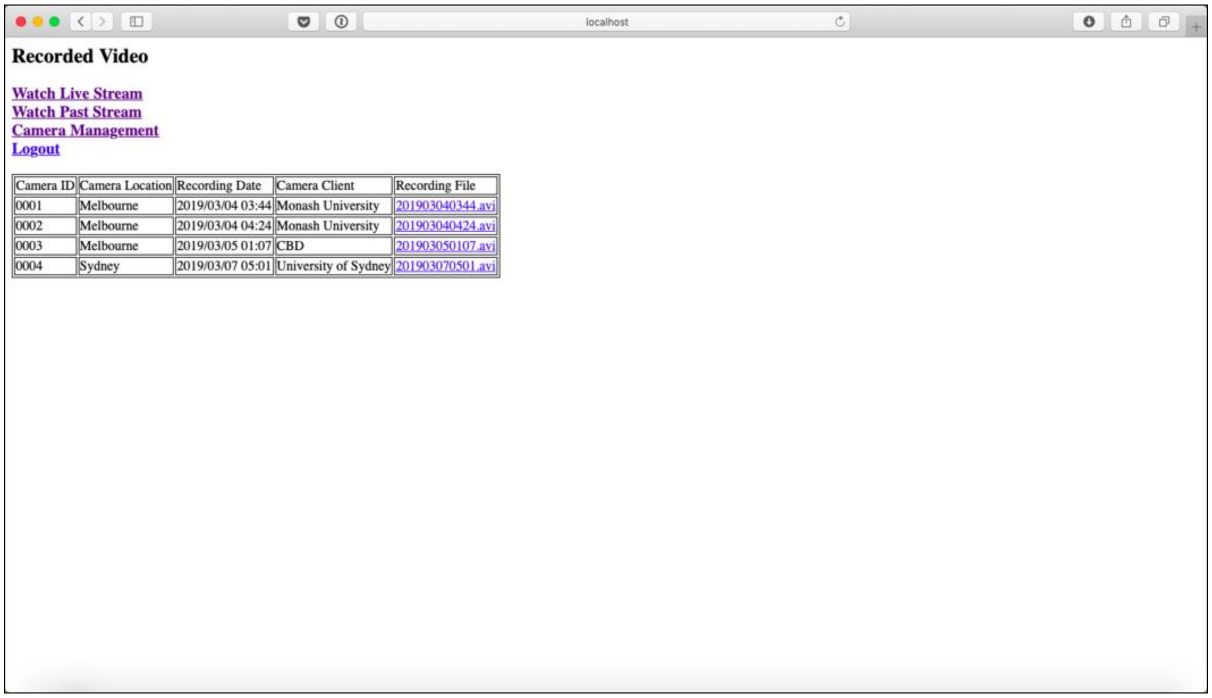


Camera Past Stream

Watch Live Stream
View Video Recording
Camera Management
Logout

Past Stream Date 2019-01-01
Past Stream Time

Motion-Detection Recordings Page – Displays a table of video snapshots of camera streams that have experienced motion detection. The supervisor will be able to filter the table based in the camera ID, location, client, or the time of recording. A recording file is stored which by default, will be a 20 second clip of the motion detected video.

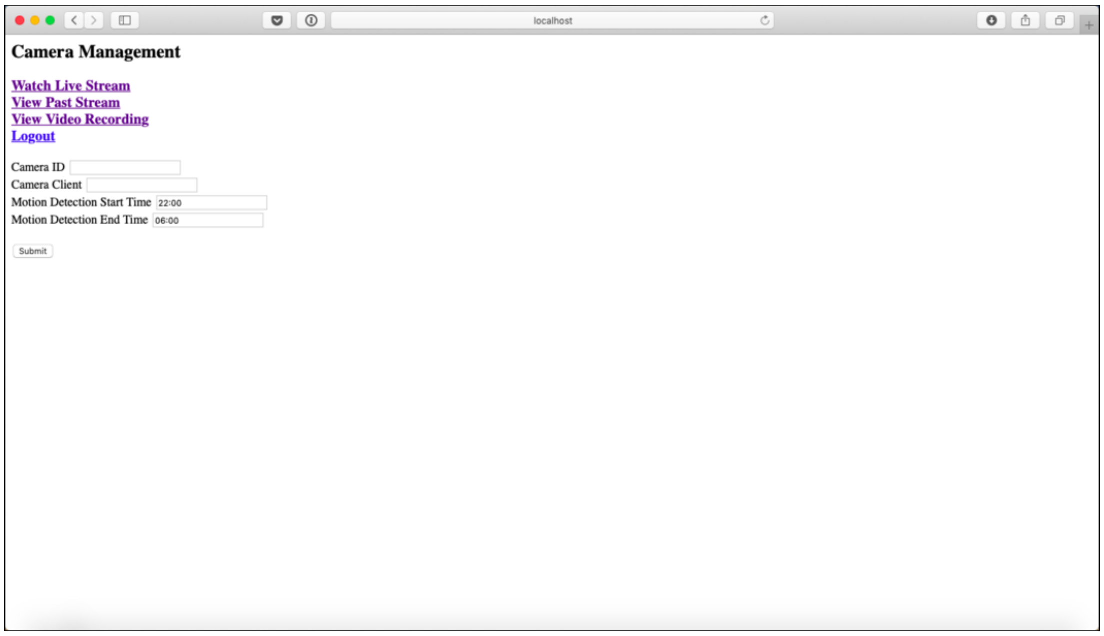


Recorded Video

Watch Live Stream
Watch Past Stream
Camera Management
Logout

Camera ID	Camera Location	Recording Date	Camera Client	Recording File
0001	Melbourne	2019/03/04 03:44	Monash University	201903040344.avi
0002	Melbourne	2019/03/04 04:24	Monash University	201903040424.avi
0003	Melbourne	2019/03/05 01:07	CBD	201903050107.avi
0004	Sydney	2019/03/07 05:01	University of Sydney	201903070501.avi

Camera Management Page – Enables the supervisor to specify and alter behaviors of its local cameras. Functionalities are subject to change and get extended.



Camera Management

[Watch Live Stream]
[View Past Stream]
[View Video Recording]
[Logout]

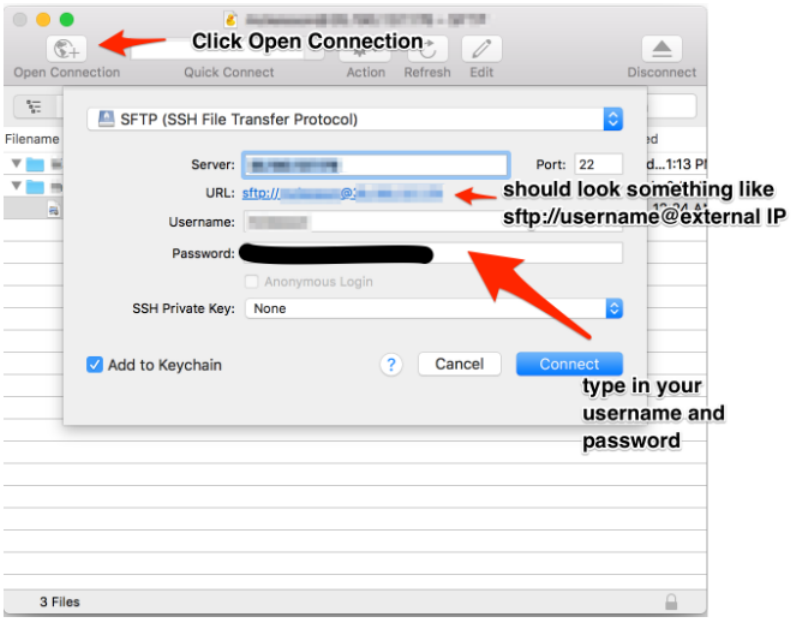
Camera ID
Camera Client
Motion Detection Start Time 22:00
Motion Detection End Time 06:00

5.2 Server Prototype Concept

Our server which will power the web application of our system, will be established in our Google Virtual Machine. All local computers will be able to transmit local video data by connecting with the server. The FTP Server will also be built inside the Google Virtual Machine using Google's Compute Engine services.

<u>Connecting a local computer (Mac Computer in this case) to the FTP server</u>

The local computer will need to connect to the external IP of the central server, and be authenticated by the server through account management.



SFTP (SSH File Transfer Protocol)

Server: [redacted] Port: 22

URL: sftp://[redacted]

Username: [redacted]

Password: [redacted]

SSH Private Key: None

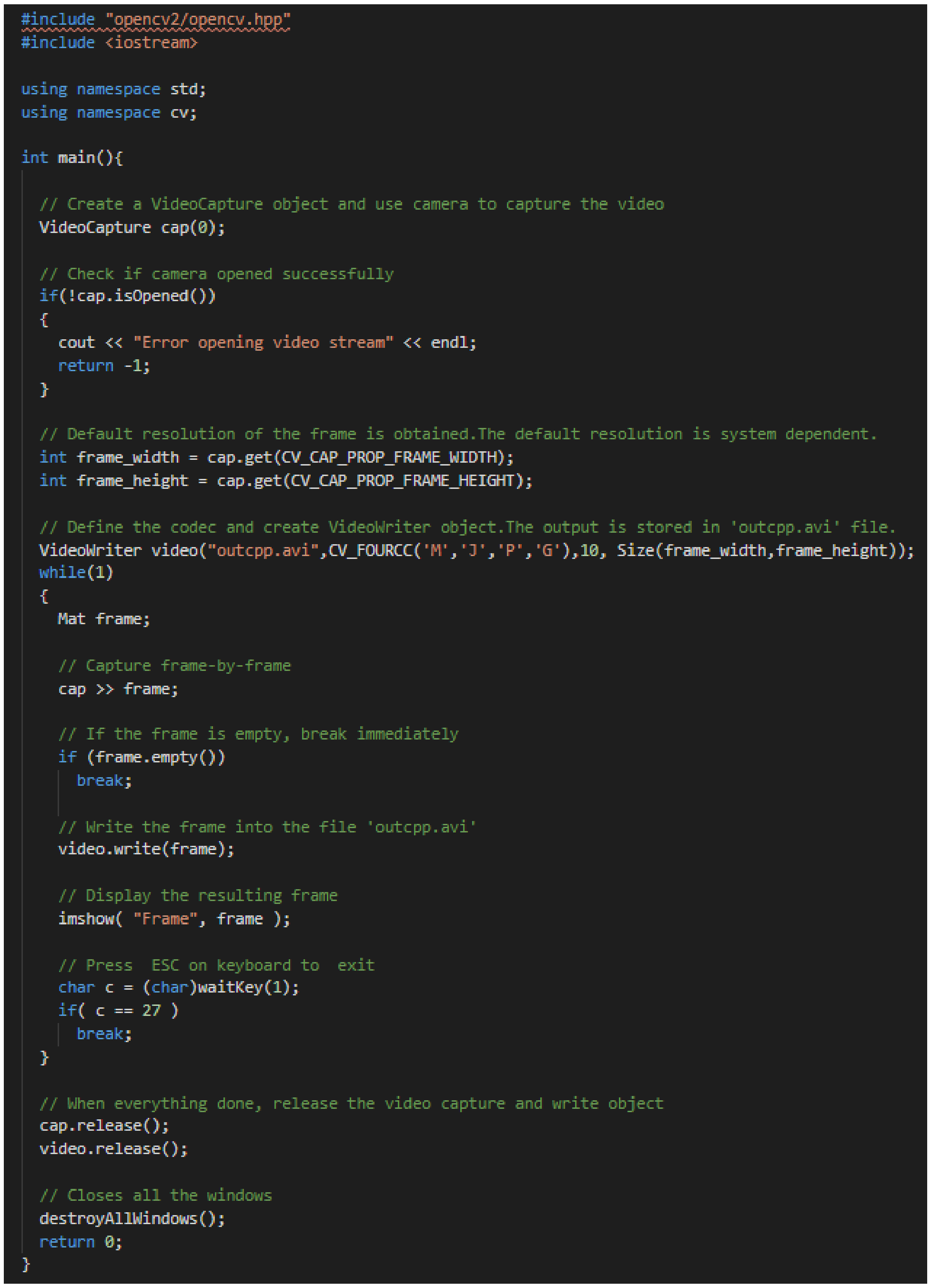
[✓] Add to Keychain

Image taken from Bianca Padilla's Medium Article "Set up your SFTP server" (Padilla, 2018).

5.3 Video Processing & Motion Detection Concept

We plan to utilize OpenCV's libraries in our C++ development environment in order to perform video processing. A code block taken from Saha at the "learnopencv.com" domain shows this operation.

Using OpenCV to capture live stream & write to a file (Code taken from Saha, 2017)



```cpp
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace std;
using namespace cv;

int main(){

  // Create a VideoCapture object and use camera to capture the video
  VideoCapture cap(0);

  // Check if camera opened successfully
  if(!cap.isOpened())
  {
    cout << "Error opening video stream" << endl;
    return -1;
  }

  // Default resolution of the frame is obtained.The default resolution is system dependent.
  int frame_width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
  int frame_height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);

  // Define the codec and create VideoWriter object.The output is stored in 'outcpp.avi' file.
  VideoWriter video("outcpp.avi",CV_FOURCC('M','J','P','G'),10, Size(frame_width,frame_height));
  while(1)
  {
    Mat frame;

    // Capture frame-by-frame
    cap >> frame;

    // If the frame is empty, break immediately
    if (frame.empty())
      break;

    // Write the frame into the file 'outcpp.avi'
    video.write(frame);

    // Display the resulting frame
    imshow( "Frame", frame );

    // Press  ESC on keyboard to  exit
    char c = (char)waitKey(1);
    if( c == 27 )
      break;
  }

  // When everything done, release the video capture and write object
  cap.release();
  video.release();

  // Closes all the windows
  destroyAllWindows();
  return 0;
}
```

<u>Background Subtraction Algorithm</u>

We plan to develop our own motion detection algorithm via the ViBe methodology of background subtraction. A visualization of how our algorithm will work can be imagined via the following images and code blocks.

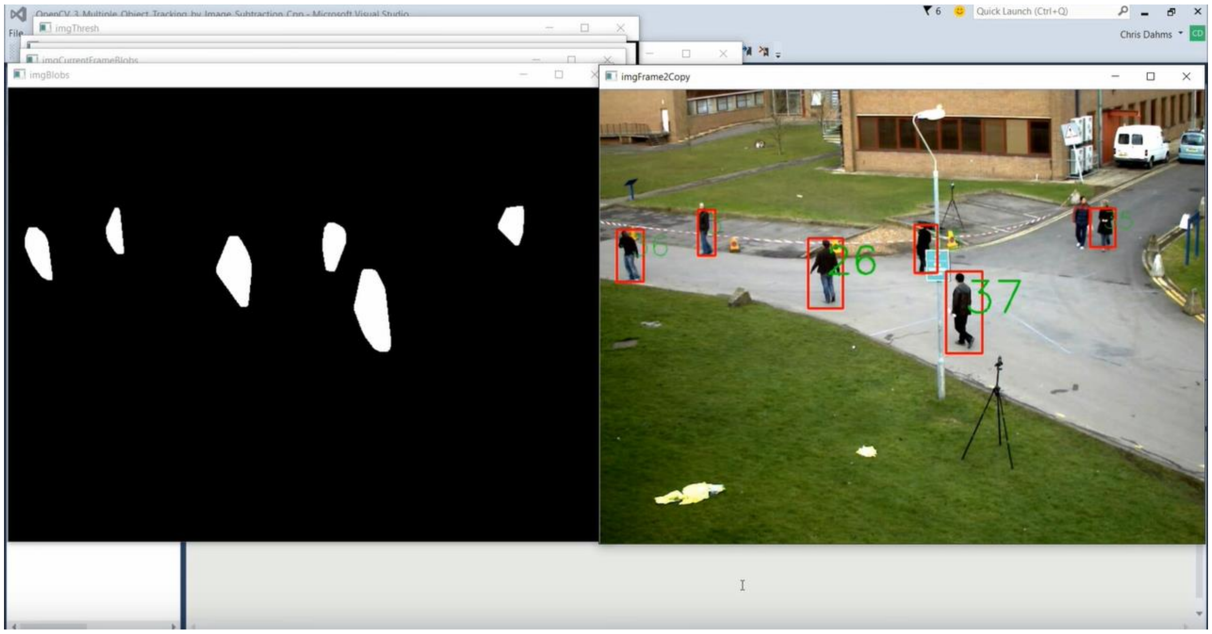


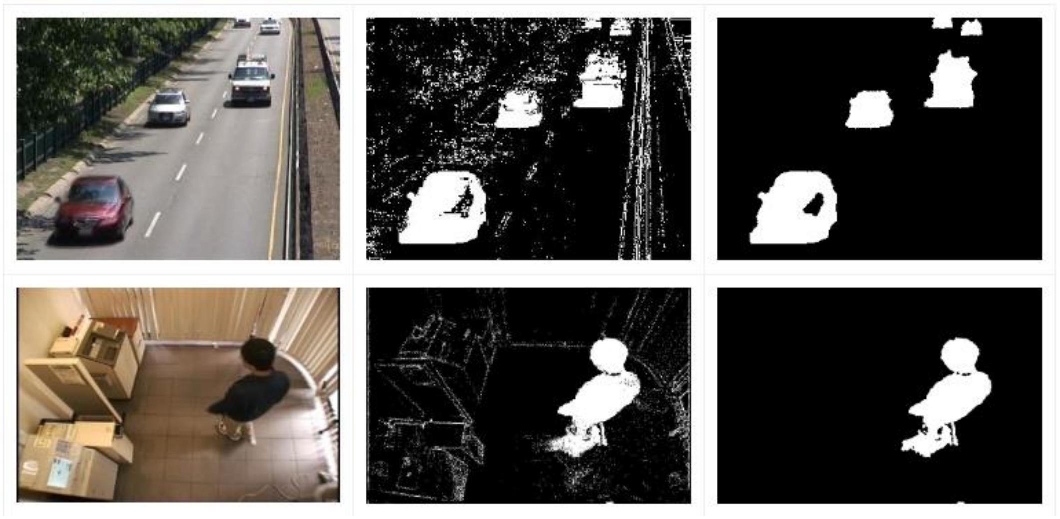
Image taken from Chris Dahm's OpenCV Object Tracking Tutorial (Dahms, 2016).

```
int width;                              // width of the image
int height;                             // height of the image
byte image[width*height];               // current image
byte segmentationMap[width*height];     // classification result

int numberOfSamples = 20;       // number of samples per pixel
int requiredMatches = 2;        // #_min
int distanceThreshold = 20;

byte samples[width*height][numberOfSamples]; // background model

for (int p = 0; p < width*height; p++) {
  int count=0, index=0, distance=0;

  // counts the matches and stops when requiredMatches are found
  while ( (count < requiredMatches) && (index < numberOfSamples) ) {
    distance = getDistanceBetween(image[p], samples[p][index]);
    if (distance < distanceThreshold) {
      count++; }
    index++;
  }

  // pixel classification
  if (count < requiredMatches)
    segmentationMap[p] = FOREGROUND_LABEL;
  else
    segmentationMap[p] = BACKGROUND_LABEL;
}
```

Pixel classification algorithm of ViBe (Droogenbroeck & Barnich, 2014).

```
int width;                    // width of the image
int height;                   // height of the image
byte image[width*height];     // current image
byte updatingMask[*height];  // updating mask (1==updating allowed)

int subsamplingFactor = 16;  // amount of random subsampling

int numberOfSamples = 20;    // number of samples per pixel
byte samples[width*height][numberOfSamples];  // background model

for (int p = 0; p < width*height; p++)
  if (updatingMask[p] == 1) { // updating is allowed

    // eventually updates the model of p (in-place updating)
    int randomNumber = getRandomIntegerIn(1, subsamplingFactor);
    if (randomNumber == 1) { // random subsampling
      // randomly selects a sample in the model to be replaced
      int randomNumber = getRandomIntegerIn(0, numberOfSamples - 1);
      samples[p][randomNumber] = image[p];
    }

    // eventually diffuses in a neighboring model (spatial diffusion)
    int randomNumber = getRandomIntegerIn(1, subsamplingFactor);
    if (randomNumber == 1) { // random subsampling
      // chooses a neighboring pixel randomly
      q = getPixelLocationFromTheNeighborhoodOf(p);
      // diffuses the current value in the model of q
      // (uncomment the following check to inhibit diffusion across the
          border of the updating mask)
      // if (updatingMask[q] == 1) {
        int randomNumber = getRandomIntegerIn(0, numberOfSamples - 1);
        samples[q][randomNumber] = image[p];
      // }
    }

  } // end of "if"
} // end of "for"
```

Background model updating algorithm of ViBe (Droogenbroeck & Barnich, 2014).



Segmentation maps generated via ViBe with left – original image, middle – unfiltered segmentation map, and right – same map but filtered with area (Droogenbroeck & Barnich, 2014).

6. References

1. Droogenbroeck, M., & Barnich, O. (2014). Retrieved from http://www.telecom.ulg.ac.be/publi/publications/mvd/VanDroogenbroeck2014ViBe/

2. IBM Knowledge Center. (2019). Retrieved 19 September 2019, from https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/intro/src/tpc/db2z_componentsofwebapplications.html

3. Padilla, B. (2018). Set up your own (free!) SFTP server using Google Cloud Platform. Retrieved 19 September 2019, from https://medium.com/@biancalorenpadilla/sftp-google-cloud-storage-d559fd16e074

4. Saha, A. (2017). Read, Write and Display a video using OpenCV (C++/ Python) | Learn OpenCV. Retrieved 19 September 2019, from https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/?fbclid=IwAR0Q9MREM3c6fEmsw5SGEsKHl_k33KJntXm-4dKIMmLTOTG9PYNiKCP7Hdo

5. Shafie, A., Ali, M., & Hafiz, F. (2009). Motion Detection Techniques Using Optical Flow. Retrieved from https://publications.waset.org/8745/pdf

6. Shekhawat, R. (2011). Background subtraction. Retrieved 19 September 2019, from https://www.slideshare.net/ravi5raj_88/background-subtraction