Software Design Document of Vigil

Hussein ElSherif, Khaled Salem, Mahmoud ElShennawy

June 15, 2017

1 Introduction

1.1 Purpose

This software design document describes the architecture and system design of our graduation project VIGIL, which is an abnormal behavior detection system aimed at the detection of abnormal behaviors related to car theft.
The intended target audience of this document is the committee members for our graduation project.

1.2 Scope

In this document, we define the architecture and design of Vigil.

1.2.1 Goal

Our goal is to detect a set of predefined abnormal behaviors in real time by processing over GPU.

1.2.2 Objectives

1. To accurately detect the predefined abnormal behaviors of Vigil that were stated in the SRS document.
2. To speed up our processing of abnormal behaviors using CUDA cores.

1.3 Overview

This is our SDD for our graduation project, and it's composed as follows:

1. Reference Material.
2. Definitions and Acronyms.
3. System Overview.
4. System Architecture.
    - Architectural Design.
    - Decomposition Description.
    - Design Rationale.
5. Data Design.
    - Data Description.
    - Data Dictionary.
    - Component Design.
6. Human Interface Design
    - Screen Images.
    - Screen Objects and Actions.
7. Requirements Matrix.
8. Appendices.
9. References.

1.4 Definitions and Acronyms

Term	Definition
HOG	Histogram of Oriented Gradients
ROI	Region Of Interest
SVM	Support Vector Machines

2 System Overview

Vigil is one of the very first surveillance systems that offer the concept of automated surveillance. Vigil should be able to detect a trained data set of abnormal behaviors. Vigil is targeted at the detection of abnormal behaviors related to car theft. These actions consist of the following:
1- Hand swings to break glass.
2- Jumping into cars.
3- Loitering.
4- Object swinging to break glass.

Vigil should then proceed to save videos of such behaviors if they were detected in the region of a car, and deploy an alarm along with a visual cue to the security guard observing the cameras to report the incident.

3 System Architecture

3.1 Architecture Design

3.1.1 Architectural Diagram

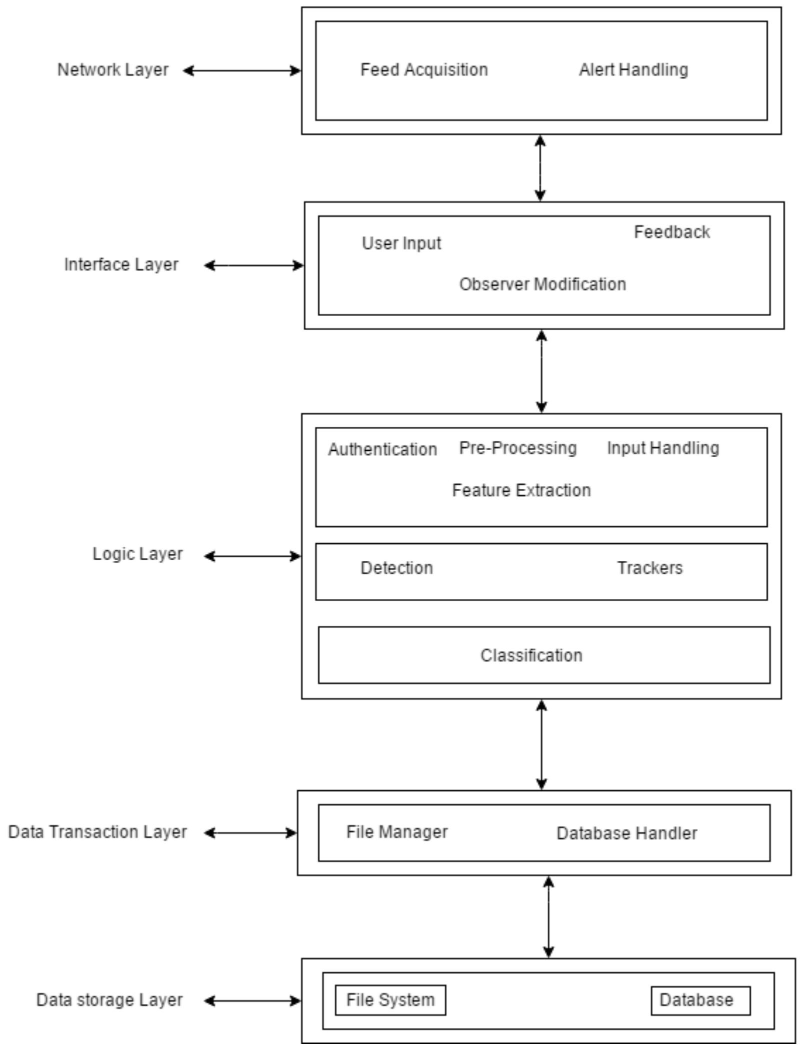
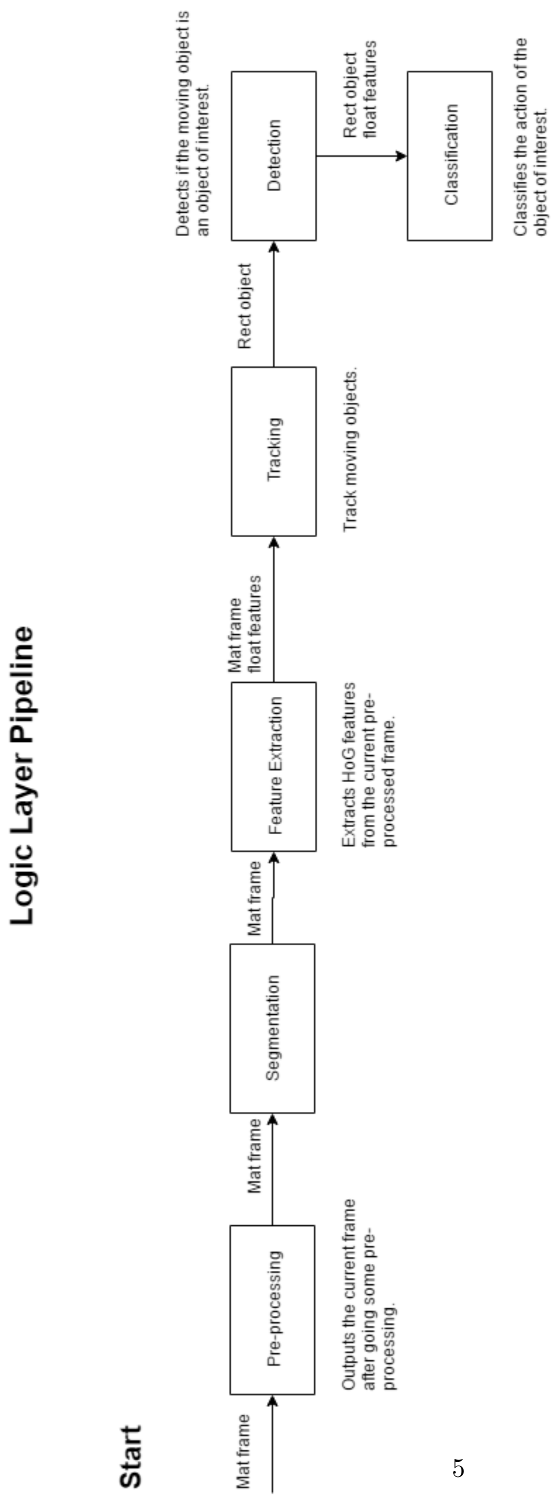


Figure 1: Architectural Design

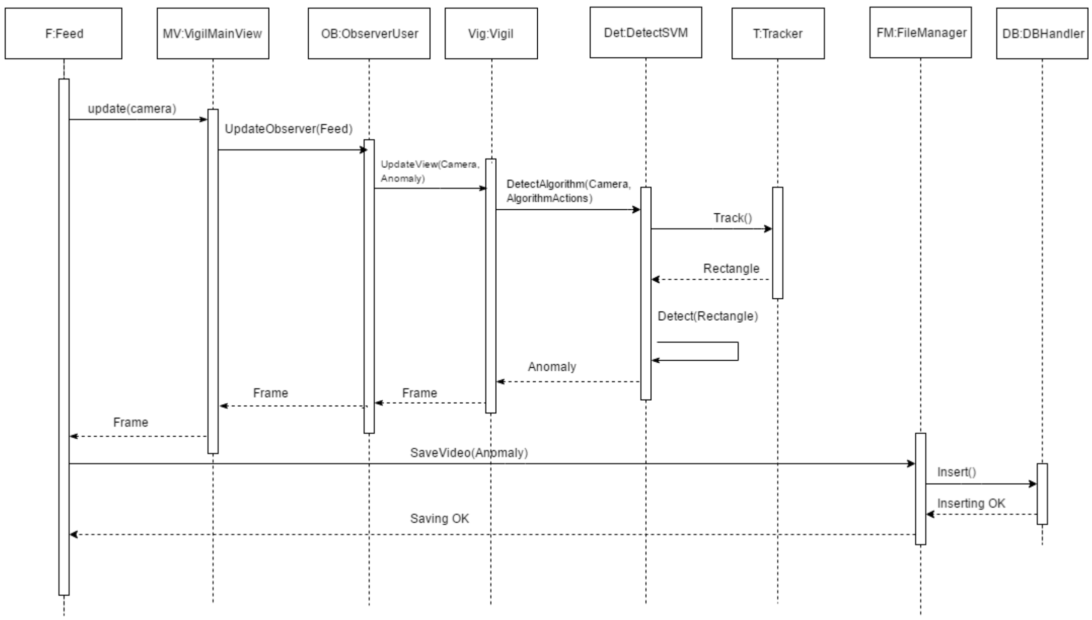
3.1.2 Logic Layer Pipeline



3.2 Sequence Diagrams

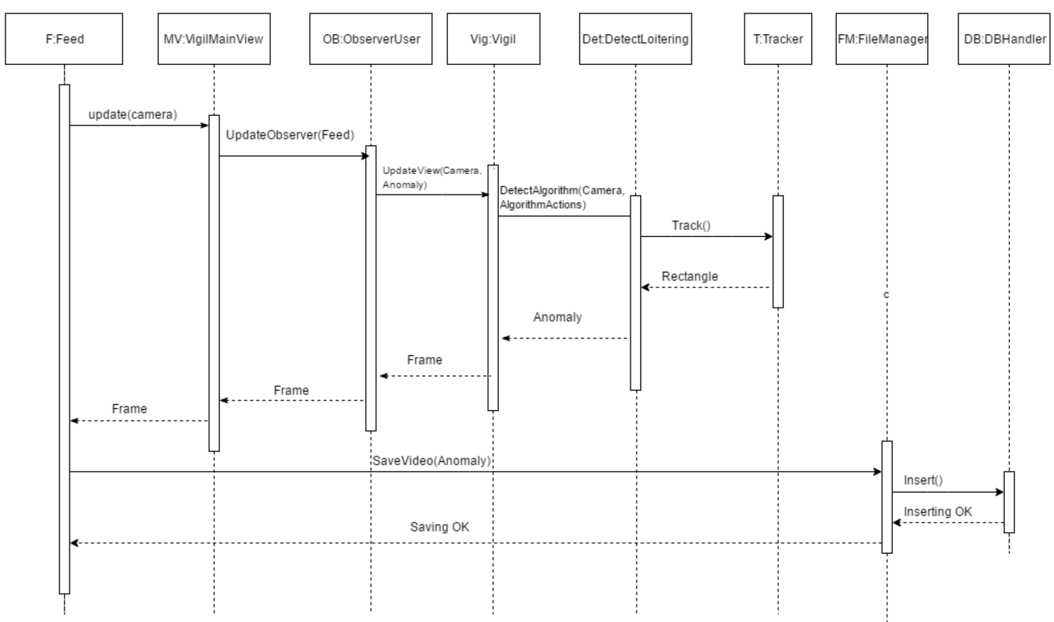
1- Action Detection.

1)Detect Actions



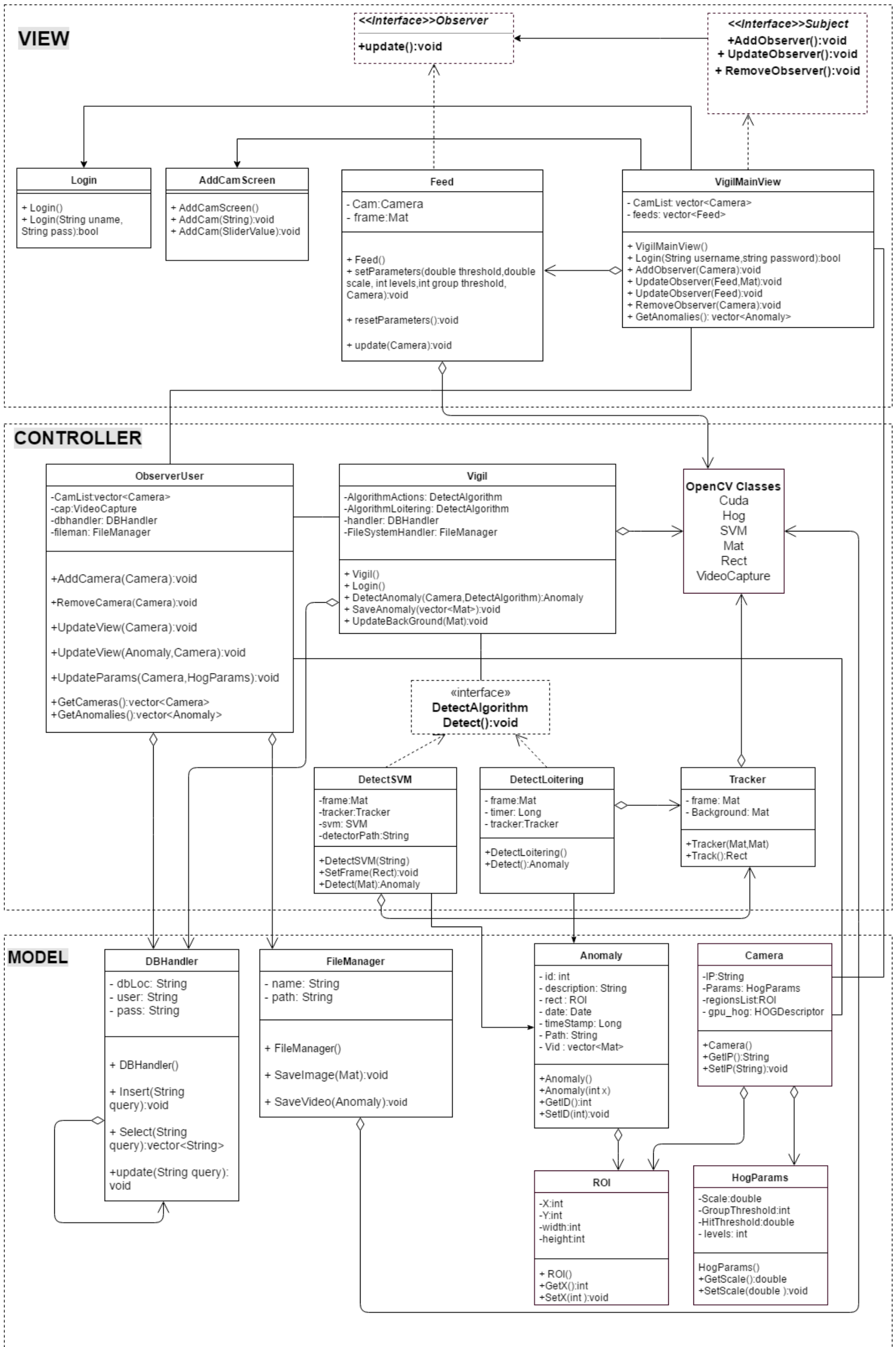
2- Loitering Detection

2)Detect Loitering



3.3 Decomposition Description

3.3.1 Class Diagram



3.3.2 Decomposition

Table 1: Interface layer description

Component	**Description**
User Input	This handles all user input for changing parameters.
Observer Modification	This handles addition of new cameras.
Output	This is the output of VIGIL including feedback

Table 2: Logic Layer - Top

Component	**Description**
Authentication	Authenticates user's credentials/Input.
input Handling	Applies the changes done by the user to detection parameters.
Pre-Processing	Handles resizing, background subtraction, etc on source frames.

Table 3: Logic Layer - Middle

Component	**Description**
Trackers	Handles tracking humans for detection application.
Detection	Handles detection of abnormal behaviors.

Table 4: Logic Layer - Bottom

Component	**Description**
Classification	Handles classifying of abnormalities detected in the previous layer.

Table 5: Data Transaction Layer

Component	Description
FileManager	Handles saving to file system.
Database handler	Handles insertion into database.

Table 6: Data storage layer

Component	Description
File System	The file system of Vigil.
Database	The system's actual database.

3.4 Design Rationale

Layered architecture provides the system with flexibility, maintainability, and scalability by separating the user interface, from program logic, from data transactions so that we can remove any possible clashes between components.
It also further enables flexibility to the system by allowing different parts to be developed independently of the others, which boosts flexibility and development speed.

Advantages:
1- Flexibility.
2- Maintainability.
3- Scalability.
4- Component reuse.

However, it also imposes the following Disadvantages:
1- Slight negative impact on performance.
2- Adds complexity to simple applications.

4 Data Design

4.1 Data Description

Video stream: Stored in OpenCV Mat structures.
Features: Stored into a float vector and processed as a 1D array alongside its label.
Descriptor: Stored into a .yml format file to be used by OpenCV's SVM descriptors.

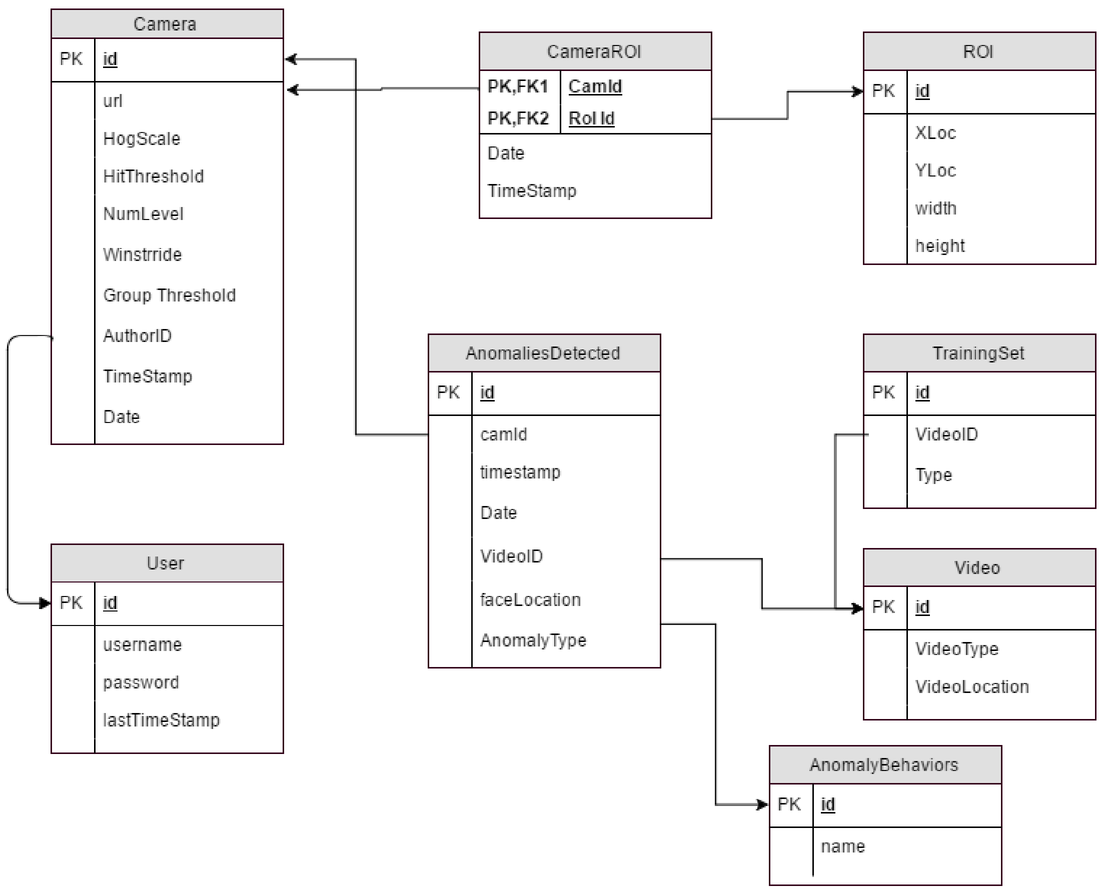


Figure 2: Database Design

4.2 Database description

4.2.1 AnomalyBehavior

A table containing a list of all abnormal behaviors the system can detect.

Table 7: AnomalyBehavior

Attribute	**Description**
ID	ID of the behavior.
Name	The name of behavior.

4.2.2 ROI

A table which contains the ROI of the whole system and their parameters.

Table 8: ROI

Attribute	**Description**
ID	ROI id.
Xloc	ROI's x location.
Yloc	ROI's y location.
Width	Width of ROI.
Height	Height of ROI.

4.2.3 AnomaliesDetected

A table which contains a list of the detected anomalies on the system along with some of its various properties.

Table 9: AnomaliesDetected Description

Attribute	**Description**
ID	Entry ID.
CamID	ID of camera that detected the anomaly.
Timestamp	Time of anomaly detection.
Date	Date of anomaly detection.
VideoID	ID of the saved video.
FaceLocation	Location of the saved Face of suspect.
AnomalyType	Type of detected anomaly.

4.2.4 Camera

A table which contains a list of all cameras registered on the system along with their various properties.

Table 10: Camera Description

Attribute	**Description**
ID	Cam ID.
URL	IP Address of the Cam.
HOGScale	Scale multiplier for the trained HOG images.
HitThreshold	Hit threshold for HOG detection
numLevels	Number of levels in image for HOG detection.
Winstride	HOG Parameter
GroupThreshold	HoG parameter for grouping detection rectangles.
AuthorID	ID of last user who changed CAM settings.
Date	Date of last settings update.

4.3 Data Dictionary

Table 11: VigilMainView

Method/Object	**Method/Object Parameters**
VigilMainView()	Void
Login() Camera	String, String AddObserver
UpdateObserver	Camera, Anomaly.
UpdateObserver	Camera
RemoveObserver	Camera
GetAnomalies()	Void

Table 12: Feed

Method/Object	**Method/Object Parameters**
Feed()	Void
SetParameters	double Threshold, double scale, int levels, int groupThreshold
resetParameters	Void
Update	Camera

Table 13: AddCamScreen

Method/Object	**Method/Object Parameters**
AddCamScreen()	Void
AddCam	String ip
AddCam()	SliderValue

Table 14: Login

Method/Object	**Method/Object Parameters**
Login()	String, String
Login()	Void

Table 15: ObserverUSer

Method/Object	**Method/Object Parameters**
Camlist	Vector Camera
Cap	Video Capture
dbHandler	DBHandler
FileMan	FileManager
AddCamera()	Camera
RemoveCamera()	Camera
UpdateView()	Camera
UpdateView()	Anomaly, Camera
UpdateParams()	Camera, HOGParams
GetCameras()	Vector Camera
GetAnomalies()	Vector Anomaly

Table 16: VIGIL

Method/Object	**Method/Object Parameters**
AlgorithmActions	DetectAlgorithm
AlgorithmLoitering	DetectAlgorithm
handler	DBHandler
FileSystemHandler	FileManager
VIGIL()	Void
Login()	Void
DetectAnomaly()	Camera, DetectAlgorithm
SaveAnomaly()	Vector
UpdateBackground()	Mat

Table 17: Tracker

Method/Object	**Method/Object Parameters**
frame	Mat
Background	Mat
Tracker()	Mat, Mat
Track()	Void

Table 18: DetectLoitering

Method/Object	**Method/Object Parameters**
frame	Mat
timer	Long
tracker	Tracker
DetectLoitering()	Void
Detect()	Void

Table 19: DetectSVM

Method/Object	**Method/Object Parameters**
frame	Mat
tracker	Tracker
svm	SVM
DetectorPath	String
DetectSVM()	String
setFrame()	Rect
Detect()	Mat

Table 20: Login

Method/Object	**Method/Object Parameters**
Login	String uname, String pass

Table 21: Camera

Method/Object	**Method/Object Parameters**
Camera()	Void
GetCamera()	Void
SetCamera()	Void
IP	String
Params	HOGParams
RegionList	ROI
GPU$_HOG$	HOGDescriptor

Table 22: DBHandler

Method/Object	**Method/Object Parameters**
DBLoc	String
User	String
Pass	String
DBHandler()	Void
Insert()	String
Select()	String
Update()	String

Table 23: FileManager

Method/Object	**Method/Object Parameters**
Name	String
Path	String
FileManager	Void
SaveImage()	Mat
SaveVideo()	Vector

Table 24: HOGParams

Method/Object	**Method/Object Parameters**
Scale	Double
GroupThreshold	Double
HitThreshold	Double
Levels	int
GetScale()	Void
GetHitThreshold()	Void
GetGroupThreshold()	Void
Set()	Void
Params	HOGParams

Table 25: ROI

Method/Object	**Method/Object Parameters**
X	Int
Y	Int
Width	Int
Height	Int
GetX()	Void
GetY()	Void
SetWidth()	Int Width
SetHeight()	Int Height
SetXY()	Int X, IntY

Table 26: Anomaly

Method/Object	Method/Object Parameters
ID	Int
Description	String
rect	ROI
date	Date
path	String
TimeStamp	Long
Anomaly()	Void
Anomaly	Int X
GetID()	Void
SetID ()	Int
GetRect()	Void
SetRect()	ROI
GetDate()	Void
SetDate()	Date

5 Component Design

5.1 Segmentation[1,3]

Background subtraction algorithm is consisting of 4 major steps[1]

1. Pre processing
2. background modeling
3. Foreground Detection
4. Data validation

Pre-processing : the process of changing the raw data which is the input video sequences into a format that can be read for the next phase[1].

Background Modeling : Background subtraction is the method used in computational to separate foreground objects from the background in the sequence of video frames[1].

Foreground Detection : moving object that separated from the background model or scene after the step of background separation. The method will detect moving object and classify the process of pixels as foreground and background[1].

Data Validation : Data validation stages function as examiner and eliminator where it examines candidate mask and eliminates pixels that are not related with target moving objects and only provide the foreground masks output[1].

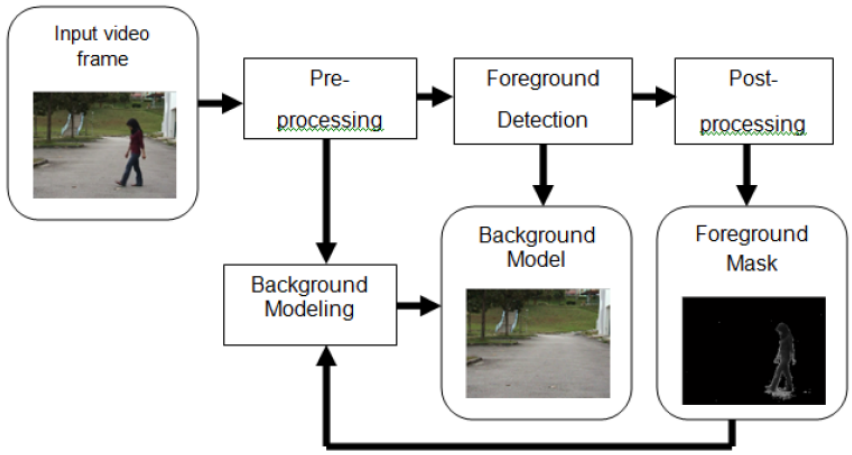


Figure 3: Background subtraction steps illustration

5.1.1 Mixture of Gaussian(MoG)

[1,3] Why MoG ? MoG method is chosen due to its low rate of complexity, memory consumption and suitability for outdoor environment along with its robustness and also it can handle multi-modal distributions

In MoG, the background is known as parametric frame of values where each pixel location is represented with number of Gaussian functions as probability distribution function as given[1].

$$F\,(i_t = \mu) = \sum_{i=1}^{k} \omega_{i,t} \, . \, \eta\,(\mu, \sigma)$$

Furthermore, the advantage of MoG is that it can extend to colour video sequences that can solve the shadows effect

5.2 Tracking[4,5,6,7]

Mean shift, which was proposed in 1975 by Fukunaga and Hostetler, is a nonparametric, iterative procedure that shifts each data to local maximum of density function[5].

Consider having the histogram of a set of pixels, we want to draw a rectangle over the area of maximum density of this set. The centroid of pixels inside the rectangle won't match the centroid of the pixel density so, we continuously move our rectangle such that the centroid of the rectangle and the centroid of the pixel density match. Figure 4 is a representation on what happens with the meanshift algorithm

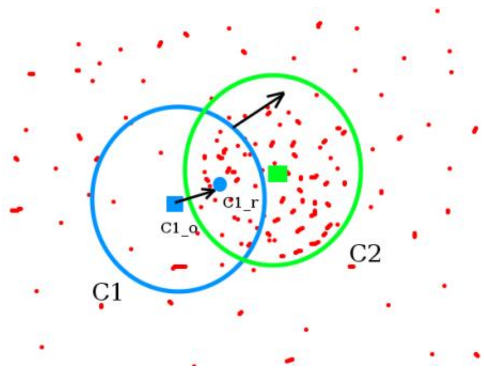


Figure 4: Meanshift

How to Calculate the Mean Shift Algorithm[4]

1. Choose a search window size.
2. Choose the initial location of the search window.
3. Compute the mean location in the search window.
4. Center the search window at the mean location computed in Step 3.
5. Repeat Steps 3 and 4 until convergence.

The method has been applied to the task of tracking a football player marked by a hand-drawn ellipsoidal region[5].

5.2.1 Camshift[4]

Since the probability distribution of the object can change and move dynamically in time, the mean shift algorithm is modified to deal with dynamically changing probability distributions. The modified algorithm is called the Continuously Adaptive Mean Shift (CAMSHIFT) algorithm.

CAMSHIFT tracks dynamically changing probability distributions. Many approaches for using histograms to identify visual objects have been suggested[4].

How to Calculate the Continuously Adaptive Mean Shift Algorithm[4]

1. Choose the initial location of the search window.
2. Mean Shift as above (one or many iterations); store the zeroth moment.
3. Set the search window size equal to a function of the zeroth moment found in Step 2.

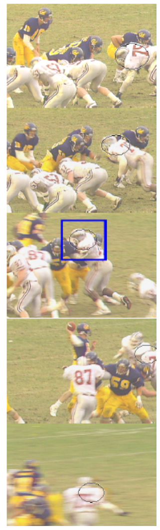


Figure 5: Meanshift example

4. Repeat Steps 2 and 3 until convergence (mean location moves less than a preset threshold).

The method has been applied to a traffic car comparing it with the meanshift algorithm the marker is applied statically[7]

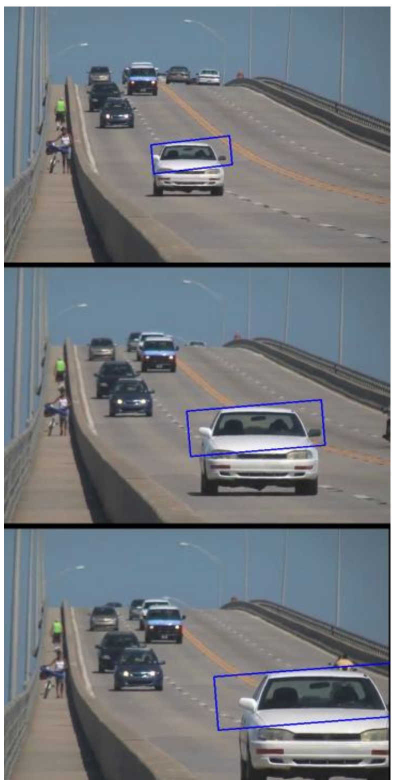


Figure 6: Camshift results

Figure 7: Meanshift results

5.2.2 Feature Extraction[2]

What the HOG does in summary is extracting features based on the histogram of gradients.

1. first the horizontal and vertical gradients are calculated using the following kernels:

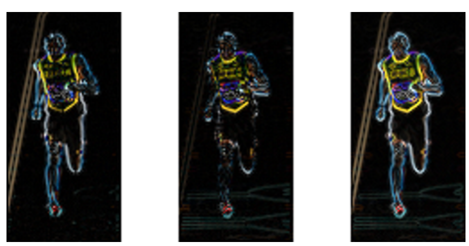
-1	0	1

-1
0
1

2. then we find the magnitude and direction of the gradients by converting from Cartesian coordinates to polar coordinates to obtain a magnitude and a direction at every pixel.

$$g = \sqrt{g_x^2 + g_y^2}$$
$$\theta = \arctan \frac{g_y}{g_x}$$

3. Then these are the resultant gradients: Left : Absolute value of x-gradient. Center : Absolute value of y-gradient. Right : Magnitude of gradient.



4. the magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels(RGB), and the angle is the angle corresponding to the maximum gradient. Empirically having the angle between 0 to 180 degrees gives much better results.

5. Then we calculate the histogram of the gradients by first dividing the image in 8*8 cells to make the representation less noisy.

6. The histograms must be normalized to be independent of the lighting conditions, to obtain better results we must work on a 16*16 block.

7. And in the end normalized histogram features are used for detection.

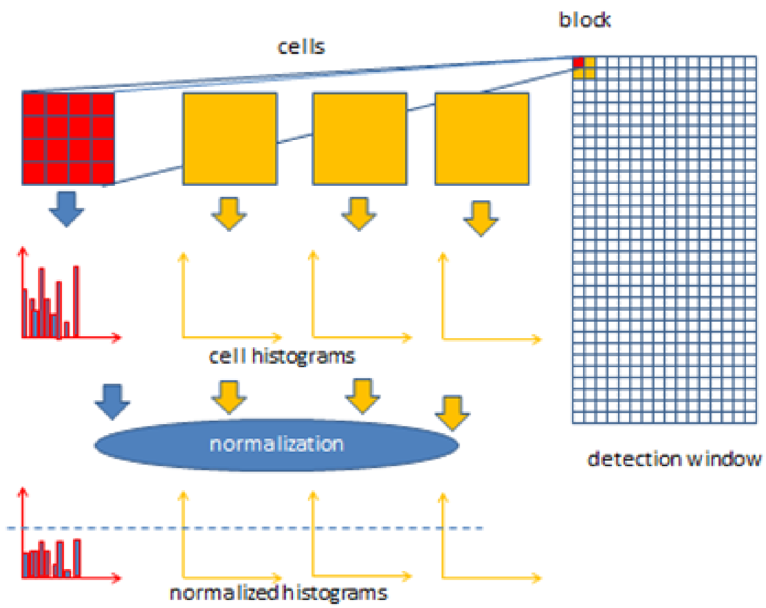


Figure 8: Hyper planes and vectors

5.2.3 Detection and Classification[9]

The classification and detection were done using SVM (Support vector Machines). The Support vector machines is a machine learning algorithm that is mainly used for classification. Our approach in using SVM is One Against All (OAA)[9], basically we classify each action independently; then we run our classification on all actions afterwards. If we're trying The support vector machine is responsible for taking a set of data and creating hyperplanes between these data vectors that become the margin between the classification of this data and the others, as shown in the figures below.

1. First we have to identify the best hyperplane *separator* between two classes. Ideally the best hyperplane is the one that segregates the two classes better and maximizes the distance between nearest data points.

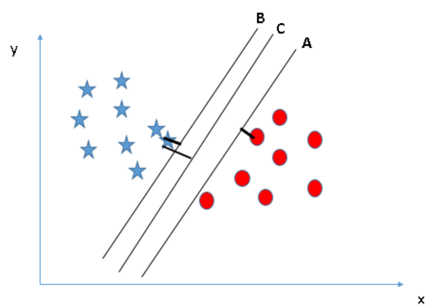


Figure 9: SVM Margin

2. If there exists outliers, SVM has a feature to ignore them.

Figure 10: SVM outlier

6 Human Interface Design

6.1 Screen Images

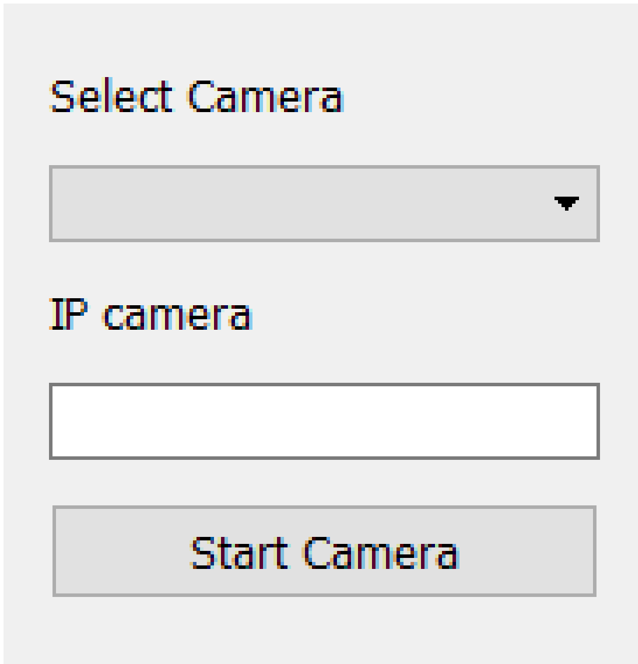


Figure 11: Select Camera screen

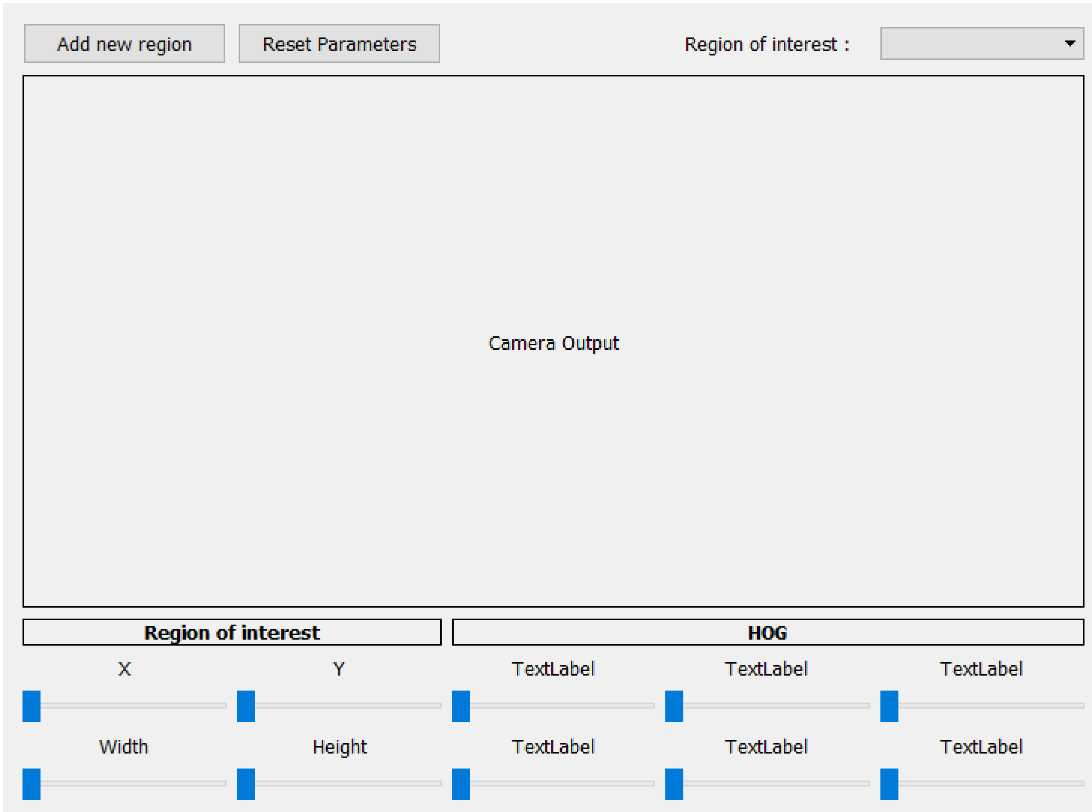


Figure 12: Output Screen

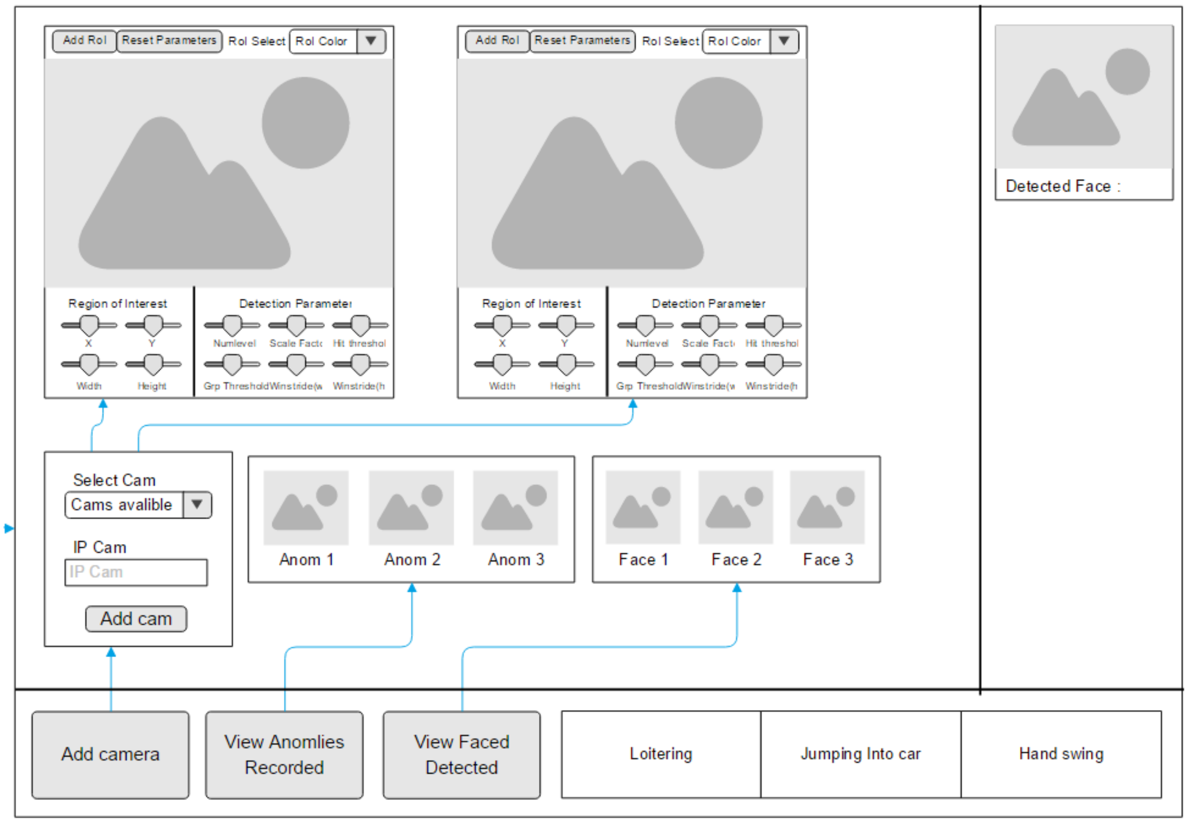


Figure 13: Main Screen

6.2 Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

6.2.1 Select Camera

The user will be able to select which camera to view on VIGIL, whether the camera is connected to the computer or is an IP camera.

6.2.2 Output screen

In this screen, the feed of the camera is shown, also the user can add a new ROI, or reset detection parameters.

6.2.3 Main Screen

In this screen, the user monitors cameras and views recorded anomalies.

7 Test results

The primitive data-set we created consists of 6710 frames extracted from 2 recording sequences of 6 people doing the same actions twice on 2 different cars.

Below is our Primitive testing results:

Table 27: My caption

Test	Accuracy	Number of frame samples
Jumping into Car	95.2%	2093
Object Swinging	51.3%	1127
Punching	65.4%	772
Pushing	100%	1004

8 Requirements Matrix

8.1 SRS Requirements

Req1: Add Camera
Req2: Set ROI
Req3: Detect Loitering
Req4: Detect Hand Swing
Req5: Detect Jumps
Req6: Save Videos of Anomalies
Req7: Feedback Alert
Req8: Remove/Modify ROI
Req9: Alter Parameters
Req10: Reset Parameters
Req11: View Detected Anomalies

8.2 Test Cases

TC1: Adding a camera to the main feed.
TC2: Running Detection and Classification on all Cameras/Videos.
TC3: Changing HoG Parameters on one of the cameras.
TC4: Adding/Changing ROI on one of the cameras.
TC5: Viewing saved anomalies on system.

Table 28: Requirements Traceability Matrix

	Total	Req1	Req2	Req3	Req4	Req5	Req6	Req7	Req8	Req9	Req10	Req11
TC1	1	X										
TC2	7	X	X	X	X	X	X	X				
TC3	3	X								X	X	
TC4	3	X	X						X			
TC5	1											X

9 References

1. S. S. Mohamed, N. M. Tahir, and R. Adnan, Background modelling and background subtraction performance for object detection, 2010 6th International Colloquium on Signal Processing its Applications, 2010.

2. N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)

3. T. Bouwmans, F. E. Baf, and B. Vachon, Background Modeling using Mixture of Gaussians for Foreground Detection - A Survey, Recent Patents on Computer Science, vol. 1, no. 3, pp. 219237, Sep. 2010.

4. G. R. Bradski, Computer Vision Face Tracking For Use in a Perceptual User Interface, 1998.

5. D. Comaniciu, V. Ramesh, and P. Meer, Real-time tracking of non-rigid objects using mean shift, Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662).

6. Z.-Q. Wen and Z.-X. Cai, Mean Shift Algorithm and its Application in Tracking of Objects, 2006 International Conference on Machine Learning and Cybernetics, 2006.

7. Meanshift and Camshift, OpenCV: Meanshift and Camshift. [Online]. Available: http://docs.opencv.org/trunk/db/df8/tutorial$_{py_m}eanshift.html.[Accessed: 25 - Apr - 2017]$.

8. M. Komorkiewicz, M. Kluczewski, and M. Gorgon, Floating point HOG implementation for real-time multiple object detection, 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012.

9. J. Manikandan and B. Venkataramani, Design of a modified one-against-all SVM classifier, 2009 IEEE International Conference on Systems, Man and Cybernetics, 2009.