Design and Development of a Detection and Tracking System for Moving Objects

VERIFEYE
Modernize Security

Sunny Patel  Samantha Husack  Ethan Wallace  Alexander Hurst
Research Area

Design a System that Detects and Tracks Objects, specifically pedestrians, in Various Contexts.
The Problem

Traditional Security Systems are Not Ideal for All Users
The Problem(s) with Traditional Systems

- **Very long term storage unfeasible**
- Single Point of Access
- Not extensible without $$$ (hardware or software)
- Often not accessible off-premises
The Problem(s) with Traditional Systems

Very long term storage unfeasible

Single Point of Access

Not extensible without $$$ (hardware or software)

Often not accessible off-premises
The Problem(s) with Traditional Systems

- Very long term storage unfeasible
- Single Point of Access
- Not extensible without $$$ (hardware or software)
- Often not accessible off-premises
The Problem(s) with Traditional Systems

Very long term storage unfeasible

Single Point of Access

Not extensible without $$$ (hardware or software)

Often not accessible off-premises
The Solution
Objectives

- Easily Scalable
- Multiple Access Points
- Secure
- Available Off-Premises
- Little Latency
Objectives

- Observe Paths taken by Objects
- Live Video Streams with Identified and Tracked Objects
- Secure Access
  - Secure Login
- Administrator Ability to Add and Remove Cameras
The Solution
A security system that is:

- Distributed tracking-based security system
- Multi-platform
- Portable
- Extensible in hardware
  - Supports the addition of any number of cameras
- Extensible in software
How VERIFEYE Works

Client Devices
Display Footage

Server and DB
Heavy Lifting

Cameras
Put the Eye in VerifEye

Available on IOS, Web and Android

With non-proprietary, modifiable protocols

Of any shape, size and number
How VERIFEYE Compression Works
Modes of Tracking - Past Footage

VerifEye can handle security in the following configurable ways

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Method Used</th>
<th>Storage Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-traffic, low area-of-coverage</td>
<td>Traditional - Record the entire scene, overlays bounding boxes</td>
<td>Matches today’s storage rates</td>
</tr>
<tr>
<td>Low traffic, high area-of-coverage. Non-safety critical</td>
<td>Minimized - Record only the actors and their movements. Not whole video.</td>
<td>Vastly improves storage rate.</td>
</tr>
</tbody>
</table>
The Design
A thorough overview
Requirements

- Mobile Application + Web Application
- Security
- Multiple Cameras
- View footage from server
  - Live
  - Past
- Track Objects in Footage
Design: The System

- **Camera**: Records Video
- **Tracking Server**: Detects and Tracks Objects
- **Central Database**: Save User and Video Data
- **API**: Link Between Access Point and Database
- **System Access Points**: User Interacts with System
Design: System Architecture
Design: Server

Goal of server: Track objects of interest

How it works:

- Custom trained Neural Network for tracking
- SORT - a Kalman Filter based Multiple Object Tracking model
- Additional logic for occlusion and re-entry
- FFmpeg and OpenCV for decoding, processing and encoding
Design: Database & NFS

- **User Information**
  - Login credentials
  - User Preferences
  - User Information

- **Camera Information by organization**

- **Video Data**

- **Tracked Objects as video streams**
Design: API Endpoints

- Asynchronous, distributed event bus
- Scalable architecture
- Video data is chunked and sent asynchronously as a byte stream
  - Vert.x is able to stream video very efficiently by bypassing userland
Design: UI Endpoints

● Mobile application built with Flutter
  ○ View Cameras, add Favorites, Profile Settings

● Web application built with Angular
  ○ Feature parity with mobile application + admin portal
The Implementation
Behind the Scenes
Cameras

Any camera that uses a non-proprietary interface

- USB
- MIPI
- SDI

Also supporting IP Cameras for low-cost, low-energy solutions
Server: Python

- Runs on windows, linux or EC2 instance
- An Dell Precision 3520 can handle 2 720p streams at 30fps
- Deep neural network + Multiple object tracker
- Inference -> Track
Testing the Server

- Checked the mAP of the detections
  - 78-96%
- Visually checked the following*:
  - False positive crossovers
    - 5 (on exit & re-entry)
  - Lost actors
    - 1
  - False positive switches
    - 41

*over 20 minutes of concatenated video
Engineering Tradeoffs

- Latency vs Extensibility
  - Is low latency really important?
  - Solution: Enable both (HLS + option for UDP)

- Tracking accuracy vs privacy
  - Reduce number of false positive switches by re-identifying actors?
  - Not worth it
API and API Testing:

- REST
- Java-based (vert.x)
- “Glue” for other services making up the application
- Postman API design suite
  - Scriptable interface
System Access Points

- Mobile Application
- Web Application
- Interact with REST API for data, video and security
Using the BLoC architecture

- Streams of Events -> Streams of States
- Allows better separation
- Powerful state management
- Rich Debugging
- Hierarchy of BLoCs
Design: Web Application

● Angular
  ○ Extensible, Secure
  ○ Open Source, built on TypeScript

● Material Design
Existing Solutions
Image Recognition

- Microsoft Computer Vision API
- Amazon Rekognition
- Google Cloud Vision API
- Open CV
- Clarifi
- Torch

We use YOLO

Deep Neural Convolutional Net
Object Tracking

- Tracker by Vicon
- Qualisys
- Kinovea
- Noraxon

We use

SORT

Kalman Filter Based Tracking
Implications of Development

VerifEye started off as an application. It is now a powerful Open Source Platform.

- Each component of VerifEye is valuable in its own right
  - Multiple Object Tracking with occlusion considerations is a currently open problem
    - Applicable to self driving cars
  - Open source distributed code for serving video
    - Applicable to many other use cases
    - Internal, secure video streaming for large organizations
  - Extensible, flexible dual-platform code that handles live and stored video
    - Useful for content-sharing
Demo and Testing Setup

Testing Setup:

4 Computers (Server, API, Observer, User)
Demonstration