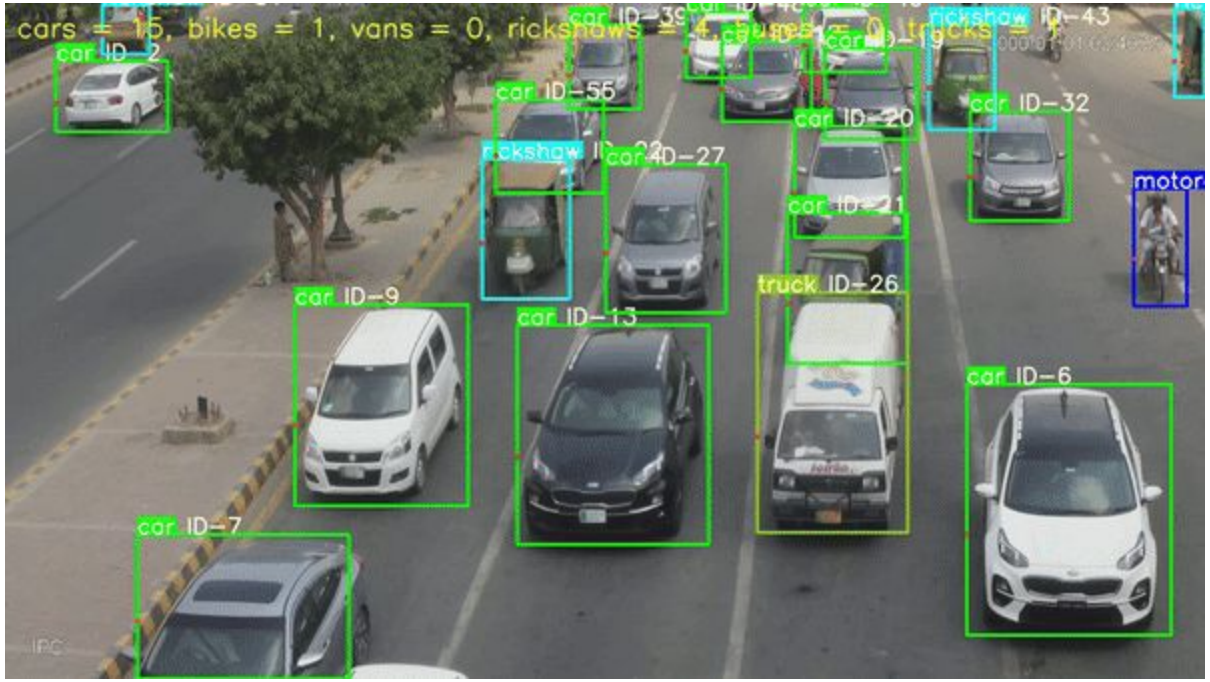
Motion Analysis

Last updated: 19-11-24

Solem, J. E., 2012. Programming Computer Vision with Python: Tools and algorithms for analyzing images. "O'Reilly Media, Inc." (Ch. 10).

Szeliski R., Computer Vision - Algorithms and Applications, Springer, 2011 (Ch. 7).

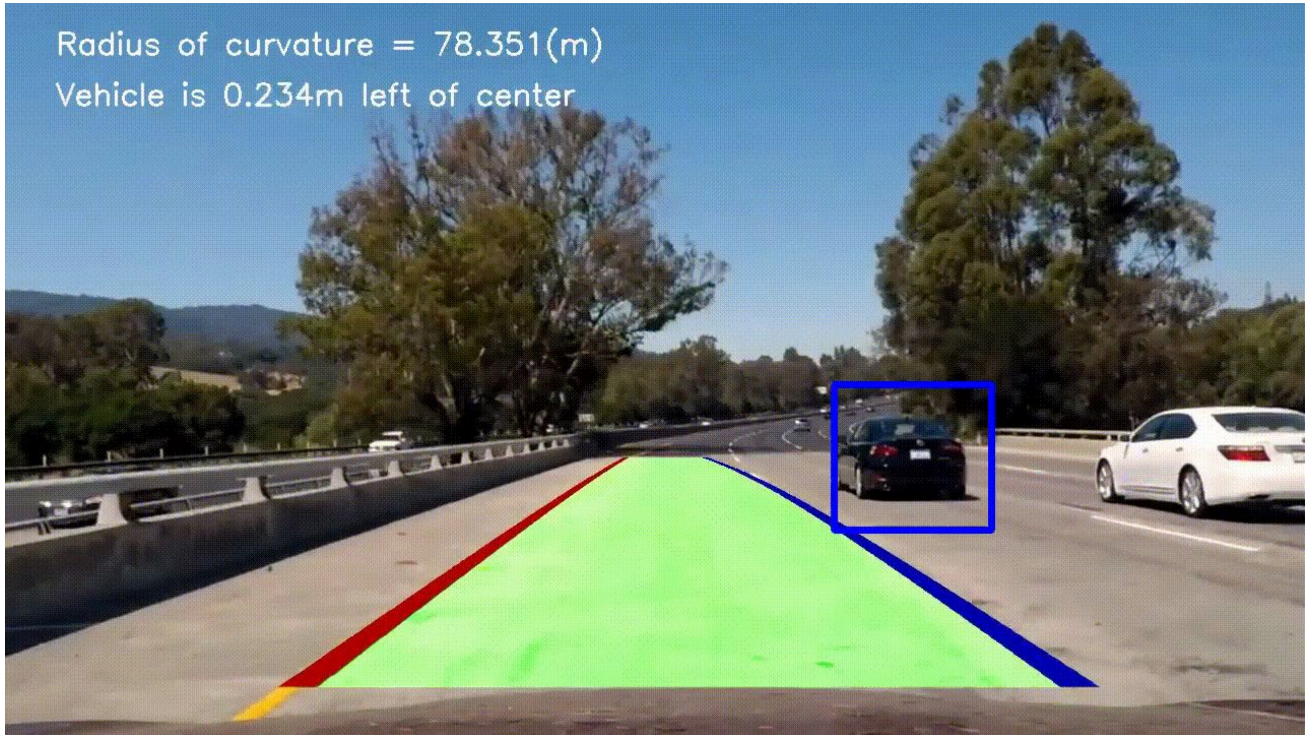


Dr. Zulfiqar Habib, Professor
http://vig.cuilahore.edu.pk

Motion Analysis

is a critical area in computer vision that deals with extracting information about moving objects from image sequences.

By analyzing how pixels change over time, we can gain valuable insights into the movement of objects in a scene. This information has numerous applications in various fields, including robotics, surveillance, and human-computer interaction.



Motion Analysis – Key Tasks

Motion detection: Identifying the presence of motion in a scene. This is often the first step in motion analysis and can be achieved through various techniques like background subtraction or frame differencing.

Object tracking: Following the movement of an object over time. Tracking algorithms use motion information to estimate the object's trajectory and predict its future location.

Trajectory analysis: Understanding the path an object takes in a scene. Trajectory analysis can reveal information about the object's motion patterns and behavior.

Video analytics: Classifying the actions performed by objects in a scene. This involves recognizing complex motions and interpreting their meaning in the context of the scene.

Object Tracking

Frame # 73

Gray

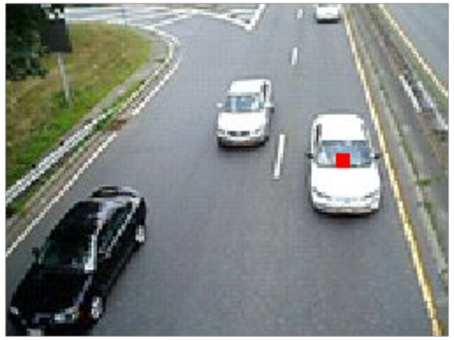
Binary

Car tracking

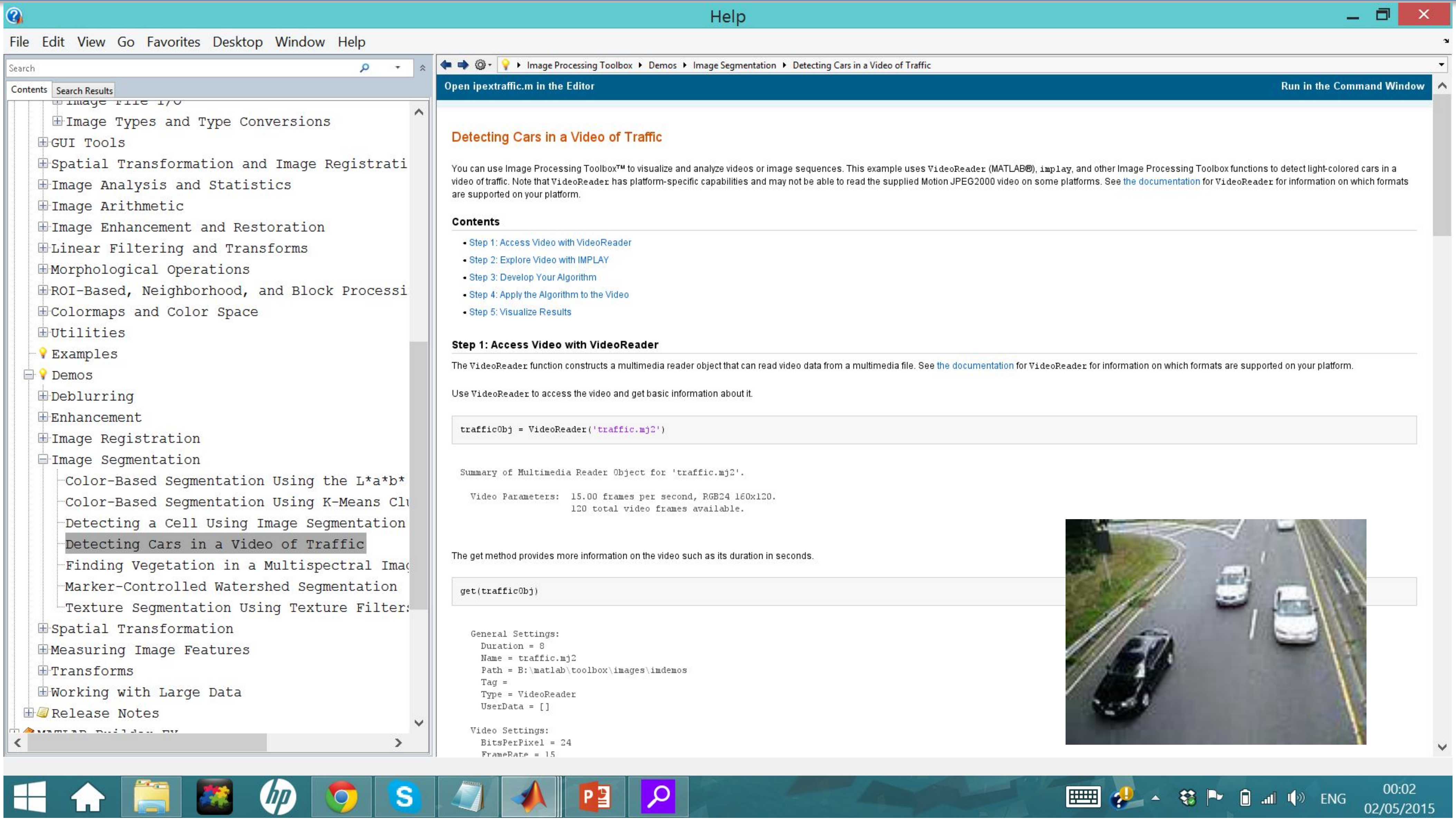
No Noise

One Object

Tagged Car



Tracking White Cars in a Video of Traffic



Tracking White Cars in a Video of Traffic

Input: traffic.mj2, a video file

Output: Video with identification and tracking of white cars

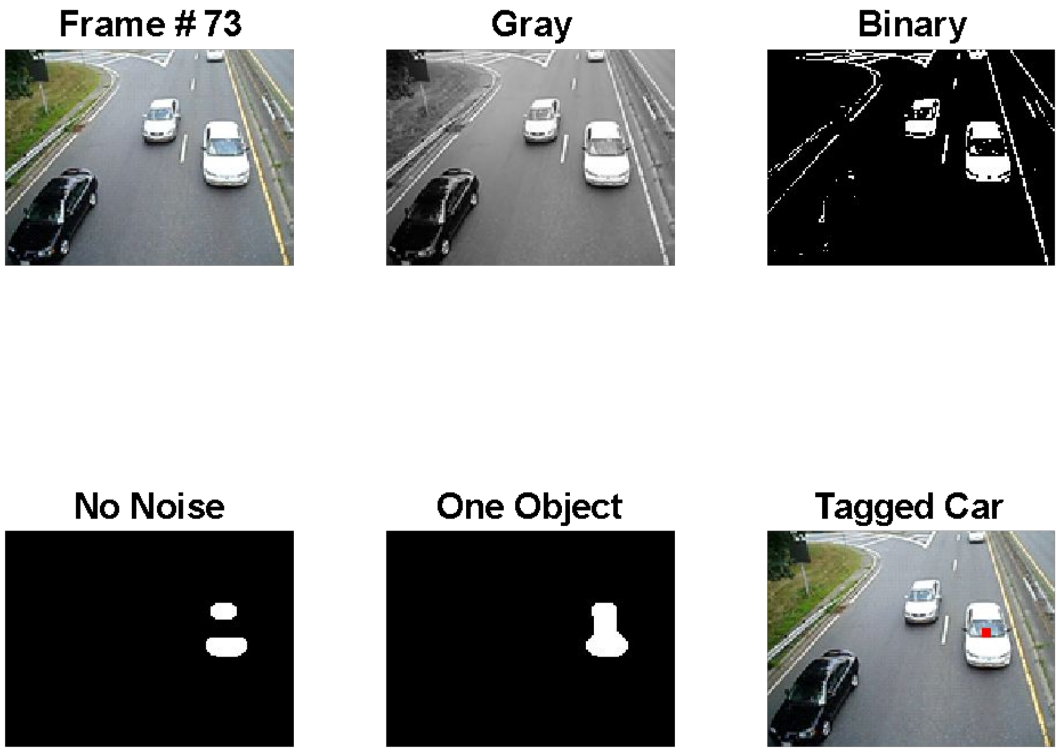
We can use Image Processing Toolbox to visualize and analyze videos or image sequences.

This project uses VideoPlayer, mmreader, implay, and other Image Processing Toolbox functions to detect light-colored cars in an MJ2 or AVI video of traffic.

Tracking White Cars in a Video of Traffic

Top-Down Approach

1. Read video
2. Play video and find representative frame(s)
3. Apply image processing and labelling on representative frame(s) to find most suitable parameters through experiments
4. Develop your algorithm
5. Apply and test the algorithm on video



Tracking White Cars in a Video of Traffic

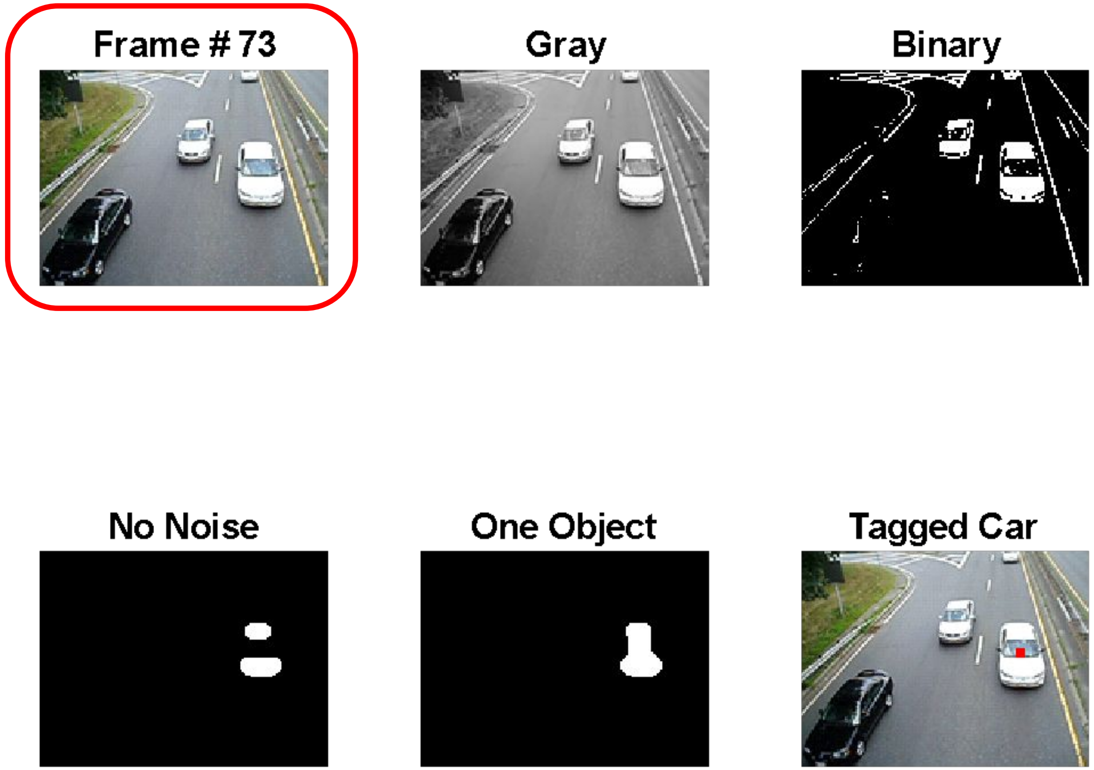
Step 1: Read video

```
trafficObj = VideoReader('traffic.mj2')

trafficObj =

  VideoReader with properties:

   General Properties:
            Name: 'traffic.mj2'
            Path: 'E:\Dropbox\Dropbox-Habib-Active\Teaching-Current\Computer Imaging\…'
        Duration: 8
     CurrentTime: 0
             Tag: ''
        UserData: []

   Video Properties:
           Width: 160
          Height: 120
       FrameRate: 15
    BitsPerPixel: 24
     VideoFormat: 'RGB24'
```

Tracking White Cars in a Video of Traffic

Step 2: Play video and find representative frame(s)

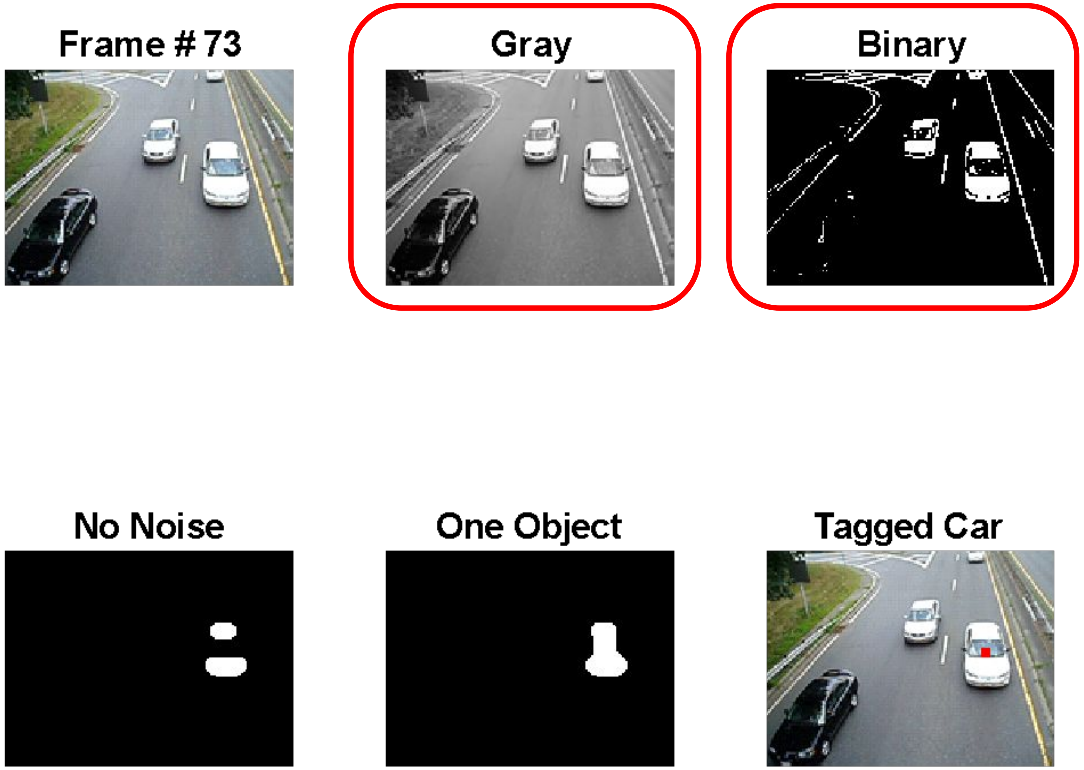
```
implay(trafficObj);
fn = 73;
f = read(trafficObj, fn);
```



Tracking White Cars in a Video of Traffic

Step 3: Apply image processing and labelling on representative frame(s)

```
fGray = rgb2gray(f);
t = double(multithresh(fGray,3))/256
t =
    0.2656    0.5273    0.7578
fBW = im2bw(fGray, t(3));
```



Tracking White Cars in a Video of Traffic

Step 3: Apply image processing and labelling on representative frame(s)

```
se5 = strel('disk', 5);
fNoNoise = imopen(fBW, se5);
se10 = strel('disk', 10);
f1Obj = imclose(fNoNoise, se10);
```

```
0 0 0 0 0 1 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 1 0 0 0 0 0
```

Frame # 73

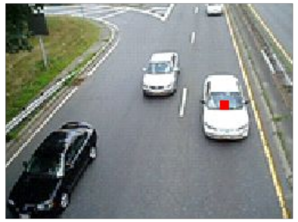
Gray

Binary

No Noise

One Object

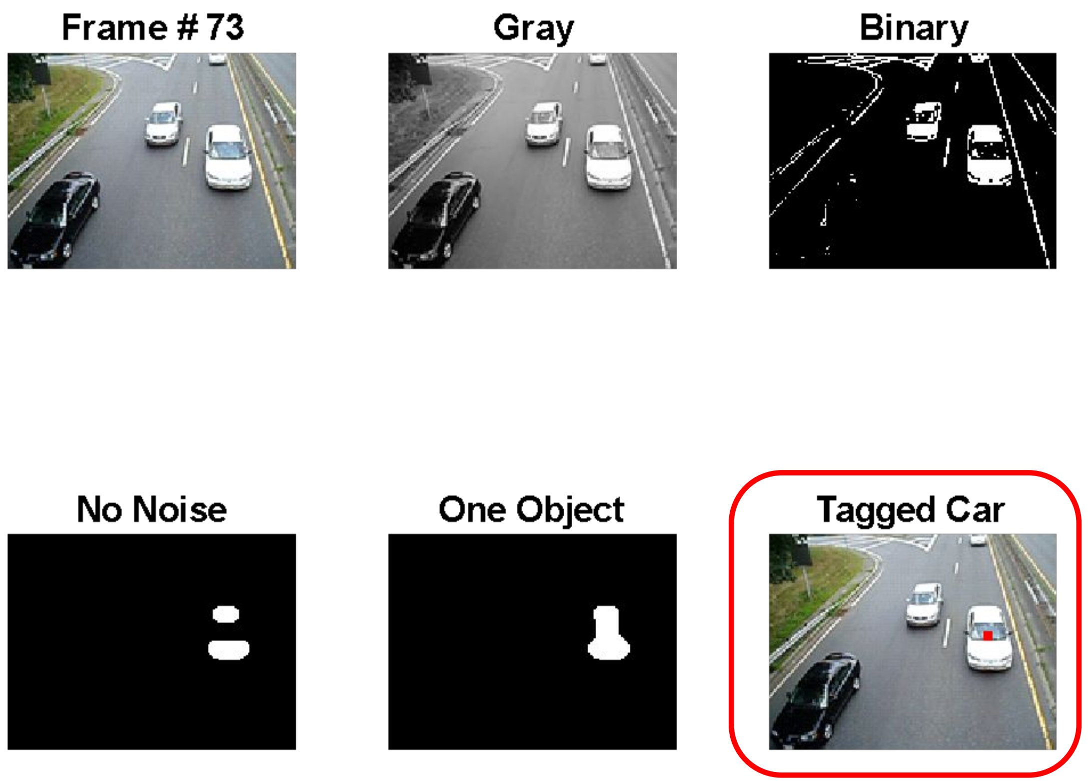
Tagged Car



Tracking White Cars in a Video of Traffic

Step 3: Apply image processing and labelling on representative frame(s)

```
width = 2; % size of label
taggedCars = f;
[r, c] = find(f1Obj==1);
rbar = round(mean(r));
cbar = round(mean(c));
row = rbar-width:rbar+width;
col = cbar-width:cbar+width;
taggedCars(row,col,1) = 255;
taggedCars(row,col,2) = 0;
taggedCars(row,col,3) = 0;
```

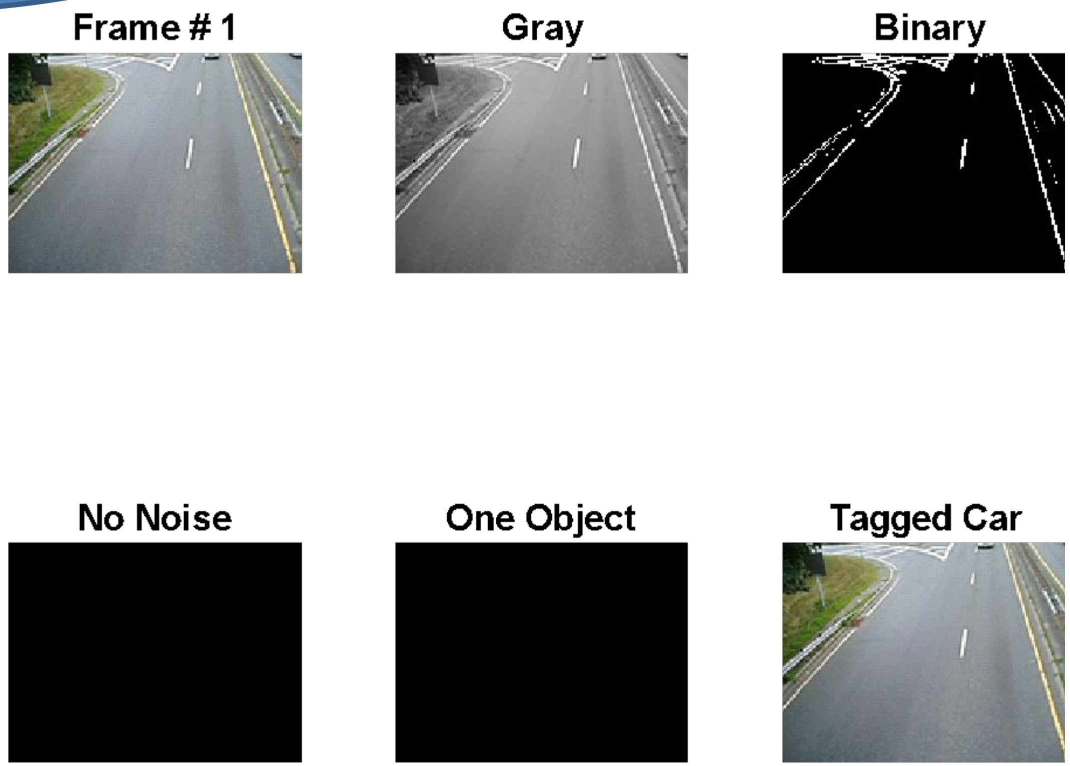


Tracking White Cars in a Video of Traffic

Step 3: Apply image processing and labelling on representative frame(s)

Does not work if no car in frame

```
width = 2; % size of label
taggedCars = f;
[r, c] = find(f1Obj==1);
rbar = round(mean(r));
cbar = round(mean(c));
row = rbar-width:rbar+width;
col = cbar-width:cbar+width;
taggedCars(row,col,1) = 255;
taggedCars(row,col,2) = 0;
taggedCars(row,col,3) = 0;
```



Tracking White Cars in a Video of Traffic

Step 3: Apply image processing and labelling on representative frame(s)

```
width = 2; % size of label
taggedCars = f;
if length(find(f1Obj==1)) > 0
   [r, c] = find(f1Obj==1);
   rbar = round(mean(r));
   cbar = round(mean(c));
   row = rbar-width:rbar+width;
   col = cbar-width:cbar+width;
   taggedCars(row,col,1) = 255;
   taggedCars(row,col,2) = 0;
   taggedCars(row,col,3) = 0;
end
```

```
subplot(2,3,1),imshow(f),title(['Frame # ', num2str(fn)]),
subplot(2,3,2),imshow(fGray),title('Gray'),
subplot(2,3,3),imshow(fBW),title('Binary'),
subplot(2,3,4),imshow(fNoNoise),title('No Noise'),
subplot(2,3,5),imshow(f1Obj),title('One Object'),
subplot(2,3,6),imshow(taggedCars),title('Tagged Car');
```

Tracking White Cars in a Video of Traffic

Step 4: Develop your algorithm

Use finalized values of parameters in previous steps for the development of algorithm.

Tracking White Cars in a Video of Traffic

Step 5: Apply and test the algorithm on video

The car-tagging application processes the video one frame at a time in a loop. Need to modify algorithm if does not work on all frames.

Because a typical video contains a large number of frames, it would take a lot of memory to read and process all the frames at once.

A small video (like the one in this example) could be processed at once, and there are many functions that provide this capability.

For faster processing, pre-allocate the memory used to store the processed video.

Tracking White Cars…

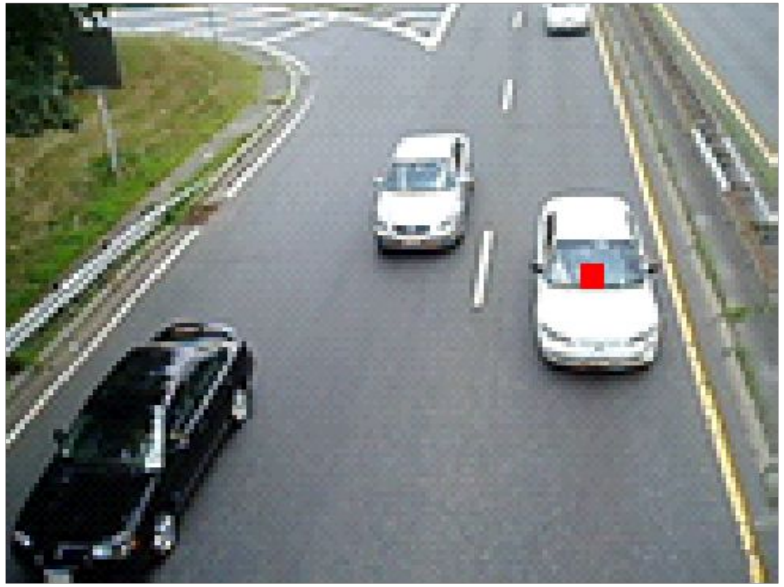

```
clc; clear all; close all hidden
spath = cd; cd('..'); addpath(genpath(pwd)); cd(spath);
trafficObj = VideoReader('traffic.mj2');
nframes = get(trafficObj, 'NumberOfFrames');
width = 2; % size of label
se5 = strel('disk', 5);
se10 = strel('disk', 10);
f = read(trafficObj, 1);
taggedCars = zeros([size(f,1) size(f,2) 3 nframes], class(f));
```

Tracking White Cars…


```
for k = 1 : nframes
    f = read(trafficObj, k);
    fGray = rgb2gray(f);
    t = double(multithresh(fGray,3))/256;
    fBW = im2bw(fGray, t(3));
    fNoNoise = imopen(fBW, se5);
    f1Obj = imclose(fNoNoise, se10);
    taggedCars(:,:,:,k) = f;
```

Tracking White Cars


```
if length(find(f1Obj==1)) > 0 % labelling if frame has a car
        [r, c] = find(f1Obj==1);
        rbar = round(mean(r)); cbar = round(mean(c));
        row = rbar-width:rbar+width; col = cbar-width:cbar+width;
        taggedCars(row,col,1,k) = 255;
      taggedCars(row,col,2,k) = 0;
      taggedCars(row,col,3,k) = 0;
    end % if
end % for

frameRate = get(trafficObj,'FrameRate');
implay(taggedCars,frameRate);
```



Tracking White Cars: Python…

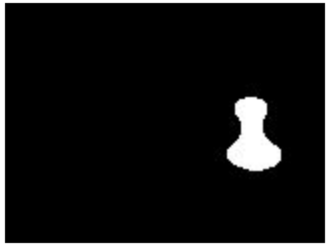
#Parameters Setting

```
from google.colab import drive
drive.mount('/content/drive')
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from skimage.filters import threshold_multiotsu

trafficObj = cv2.VideoCapture('/content/drive/MyDrive/Images/traffic.mj2’) # Read the video file
fn = 73 # Set the frame position
trafficObj.set(cv2.CAP_PROP_POS_FRAMES, fn) # Read the frame
ret, frame = trafficObj.read()
cv2_imshow(frame) # Display the frame
fGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Convert to grayscale
cv2_imshow(fGray)
```

Tracking White Cars: Python…


```
# Generate multi-threshold values
t = threshold_multiotsu(fGray, classes=4)  # Divide into 3 classes
print("Threshold values:", t)
print(t[2])
# Binary image
fBW = np.where(fGray > t[2], 255, 0).astype(np.uint8)
cv2_imshow(fBW)
# Create structuring elements for morphological operations
se5 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11)) # disk of radius 5
se10 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (21, 21)) # disk of radius 10
# Morphological operations
fNoNoise = cv2.morphologyEx(fBW, cv2.MORPH_OPEN, se5)
cv2_imshow(fNoNoise)
f1Obj = cv2.morphologyEx(fNoNoise, cv2.MORPH_CLOSE, se10)
cv2_imshow(f1Obj)
```

```
Threshold values: [ 69 132 193]
193
```



Tracking White Cars: Python


```
# Store the original frame in taggedCars
taggedCars = frame

# Label cars if present
width = 2 # size of label
if np.any(f1Obj == 255):
    r, c = np.where(f1Obj == 255)
    rbar = int(np.mean(r))
    cbar = int(np.mean(c))
    row = slice(rbar - width, rbar + width + 1)
    col = slice(cbar - width, cbar + width + 1)
    taggedCars[row, col, 0] = 0   # Blue channel
    taggedCars[row, col, 1] = 0   # Green channel
    taggedCars[row, col, 2] = 255 # Red channel

cv2_imshow(taggedCars)
```

Tracking White Cars: Python…

#Car Tracking Video

```
from google.colab import drive
drive.mount('/content/drive')
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from skimage.filters import threshold_multiotsu

# Read the video file
trafficObj = cv2.VideoCapture('/content/drive/MyDrive/Images/traffic.mj2’)

nframes = int(trafficObj.get(cv2.CAP_PROP_FRAME_COUNT)) # Get the total number of frames
width = 2 # Size of the label
# Create structuring elements for morphological operations
se5 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11)) # disk of radius 5
se10 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (21, 21)) # disk of radius 10
```

Tracking White Cars: Python…


```
# Initialize an array to store tagged frames
frameWidth = int(trafficObj.get(cv2.CAP_PROP_FRAME_WIDTH))
frameHeight = int(trafficObj.get(cv2.CAP_PROP_FRAME_HEIGHT))
frameRate = trafficObj.get(cv2.CAP_PROP_FPS)
taggedCars = np.zeros((frameHeight, frameWidth, 3, nframes), dtype=np.uint8)

# Prepare video writer to save the output
fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for MP4
outputPath = '/content/drive/MyDrive/Temp/tagged_traffic.mp4'
videoWriter = cv2.VideoWriter(outputPath, fourcc, frameRate, (frameWidth, frameHeight))
```

Tracking White Cars: Python…


```
# Loop through all frames
for k in range(nframes):
    # Read the current frame
    ret, frame = trafficObj.read()
    if not ret:
        break
    # Convert to grayscale
    fGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Generate multi-threshold values
    t = threshold_multiotsu(fGray, classes=4)  # Divide into 3 classes
    # Binary image
    fBW = np.where(fGray > t[2], 255, 0).astype(np.uint8)
    # Morphological operations
    fNoNoise = cv2.morphologyEx(fBW, cv2.MORPH_OPEN, se5)
    f1Obj = cv2.morphologyEx(fNoNoise, cv2.MORPH_CLOSE, se10)
    # Store the original frame in taggedCars
    taggedCars[:, :, :, k] = frame
```

Tracking White Cars: Python

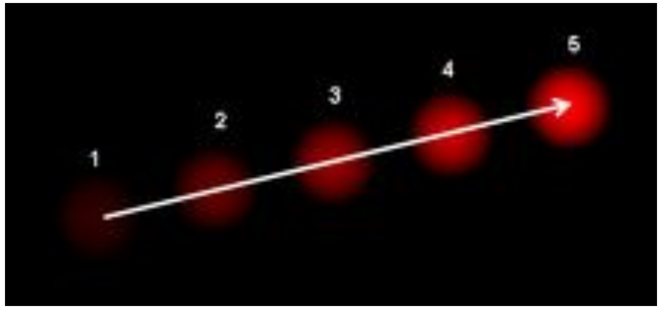

```
    # Label cars if present
    if np.any(f1Obj == 255):
        r, c = np.where(f1Obj == 255)
        rbar = int(np.mean(r))
        cbar = int(np.mean(c))
        row = slice(rbar - width, rbar + width + 1)
        col = slice(cbar - width, cbar + width + 1)
        taggedCars[row, col, 0, k] = 0     # Blue channel
        taggedCars[row, col, 1, k] = 0     # Green channel
        taggedCars[row, col, 2, k] = 255  # Red channel
    # Write the tagged frames to the output video
    videoWriter.write(taggedCars[:, :, :, k])

# Release the video capture and writer objects
trafficObj.release()
videoWriter.release()
if len(taggedCars) > 0:
    print(f"Tagged video saved to {outputPath}")
else:
    print("No tagged frames found.")
```

Task

1. Apply your algorithm on traffic.avi and resolve issues if any.
2. Try the strategy of consecutive frame difference for noise removal and detection of moving objects.
3. Track dark cars in a given video clip.
4. Count white cars in a given video clip.

Tracking Objects Using Optical Flow

Optical flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene.



In computer vision, the **Lucas–Kanade method** is a widely used differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade. It assumes that the flow is essentially constant in a local neighbourhood of the pixel under consideration, and solves the basic optical flow equations for all the pixels in that neighbourhood, by the least squares criterion. (Wikipedia)

Lucas–Kanade (LK) - Algorithm

Optical flow vector between two images is calculated from the solution of the following optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

I_x, I_y, and I_t are the spatiotemporal image brightness derivatives, where

u is the horizontal optical flow.

v is the vertical optical flow.

Lucas–Kanade (LK) - Algorithm

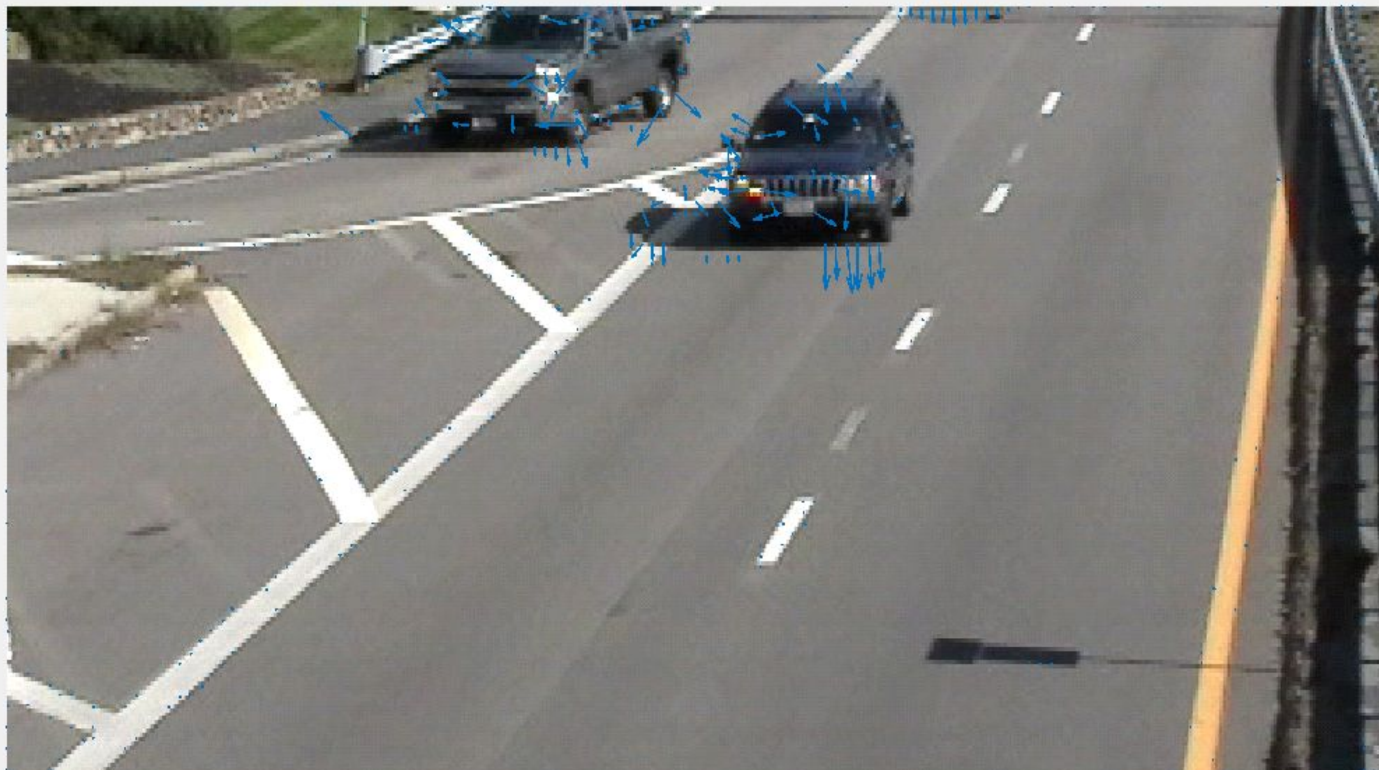
The LK optical flow algorithm [1] is a simple and efficient technique which can provide an estimate of the movement of interesting features in successive images of a scene. Movement vector (u, v) is associated to every such "interesting" pixel in the scene, obtained by comparing the two consecutive frames. It is also less sensitive to image noise.

Disadvantage - errors on boundaries of moving object. The optical flow equation can be assumed to hold for all pixels within a window centered at p, i.e., the velocity vector for local image must satisfy. It is a purely local method and cannot provide flow information in the interior of uniform regions of the image.

[1] Barron, J. L., D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. "Performance of optical flow techniques." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),*236-242. Champaign, IL: CVPR, 1992

Tracking Cars Using LK Optical Flow


```
vidReader = VideoReader('visiontraffic.avi','CurrentTime',11);
opticFlow = opticalFlowLK('NoiseThreshold',0.009); % to estimate the direction and speed of the moving objects
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of Optical Flow Vectors');
hPlot = axes(hViewPanel);
while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = rgb2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
    hold on
    plot(flow,'DecimationFactor',[5 5],
              'ScaleFactor',10,'Parent',hPlot);
    hold off
    pause(10^-3)
end
```



Exercise

Record your own video of traffic of around one minute and count light and heavy traffic separately, with accuracy in %age.

Face Detection and Tracking

Object detection and tracking are important in many computer vision applications including activity recognition, automotive safety, and surveillance.

To track the face over time, Kanade-Lucas-Tomasi (KLT) algorithm can be used [1]. While it is possible to use an cascade (waterfall like) object detector on every frame, it is computationally expensive. It may also fail to detect the face, when the subject turns or tilts his head. This limitation comes from the type of trained classification model used for detection.

[1] Suhr, J.K., 2009. Kanade-lucas-tomasi (klt) feature tracker. Computer Vision (EEE6503), pp.9-18.

Face Detection and Tracking

Therefore, the face can be detected only once, and then the KLT algorithm tracks the face across the video frames.

A simple system can be developed for tracking a single human face in a live video stream captured by a webcam by dividing the tracking problem into the following three parts.

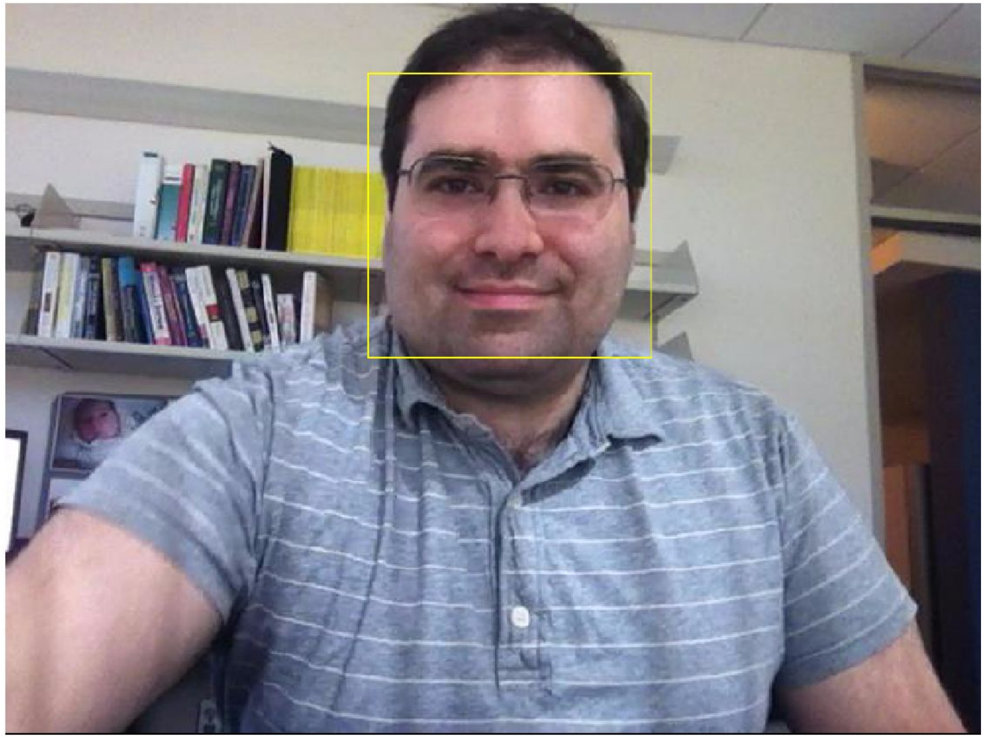
Face Detection and Tracking

1. Face Detection

First, we detect the location of a face in a video frame by using Viola-Jones detection algorithm (cascade object detector in Matlab) and a trained classification model for detection.

By default, the detector is configured to detect faces, but it can be used to detect other types of objects.

Detected face

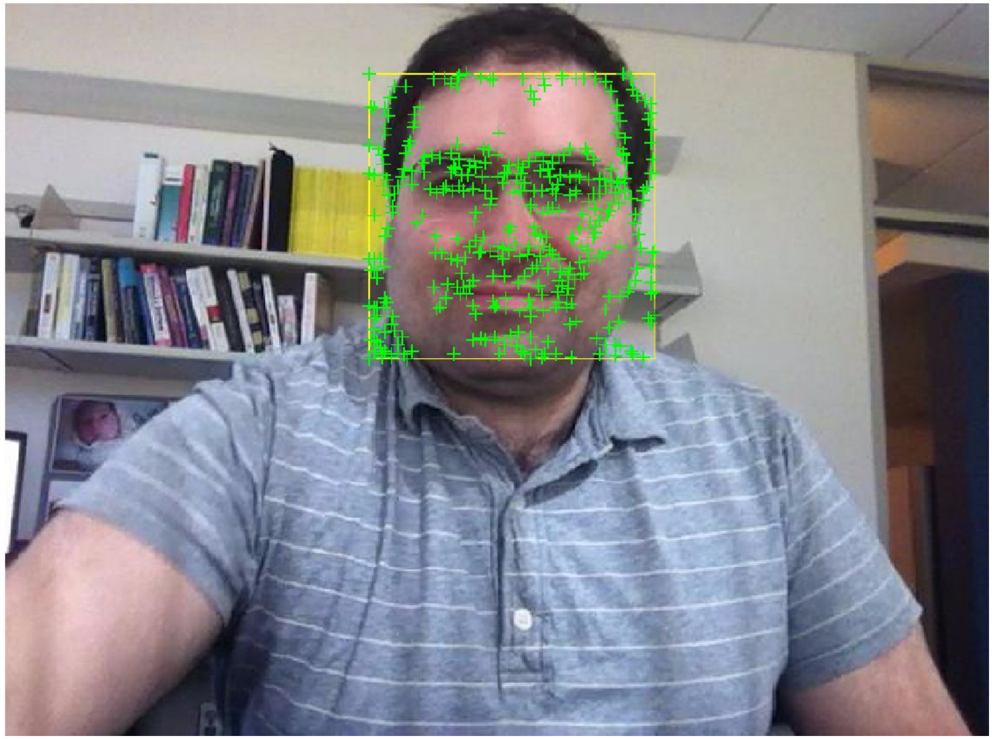


Face Detection and Tracking

2. Identification of Facial Features to Track

The KLT algorithm tracks a set of feature points across the video frames. Once the detection locates the face, the next step is to identifies feature points that can be reliably tracked [1].

Detected features

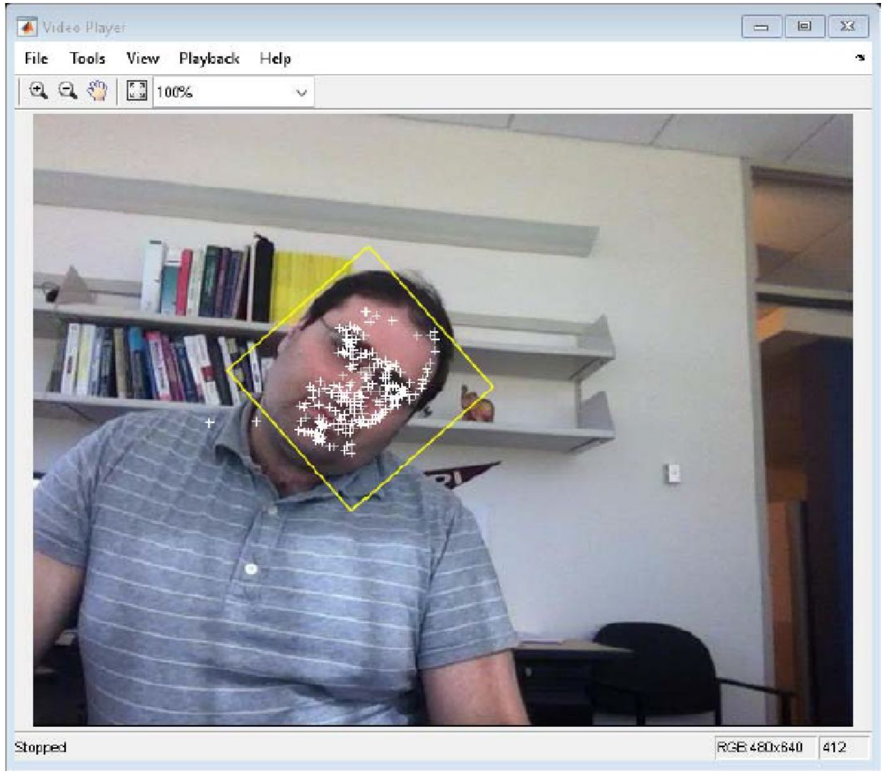


[1] Shi, J., 1994, June. Good features to track. In 1994 Proceedings of IEEE conference on computer vision and pattern recognition (pp. 593-600). IEEE.

Face Detection and Tracking

3. Initialization of a Tracker to Track the Points

Once the feature points identified, object can be tracked. For each point in the previous frame, the object tracker attempts to find the corresponding point in the current frame. Then translation, rotation, and scaling are estimated between the old points and the new points. This transformation is applied to the bounding box around the face.



Face Detection and Tracking - Code

Help

sound | Face Detection and Tracking Using the KLT Algorithm

Documentation | All | Examples | Functions | Blocks | Apps | Find in page

CONTENTS Close

Face Detection and Tracking Using the KLT Algorithm

This example shows how to automatically detect and track a face using feature points. The approach in this example keeps track of the face even when the person tilts his or her head, or moves toward or away from the camera.

Open Live Script

Introduction

Object detection and tracking are important in many computer vision applications including activity recognition, automotive safety, and surveillance. In this example, you will develop a simple face tracking system by dividing the tracking problem into three parts:

1. Detect a face
2. Identify facial features to track
3. Track the face

Detect a Face

First, you must detect the face. Use the `vision.CascadeObjectDetector` object to detect the location of a face in a video frame. The cascade object detector uses the Viola-Jones detection algorithm and a trained classification model for detection. By default, the detector is configured to detect faces, but it can be used to detect other types of objects.

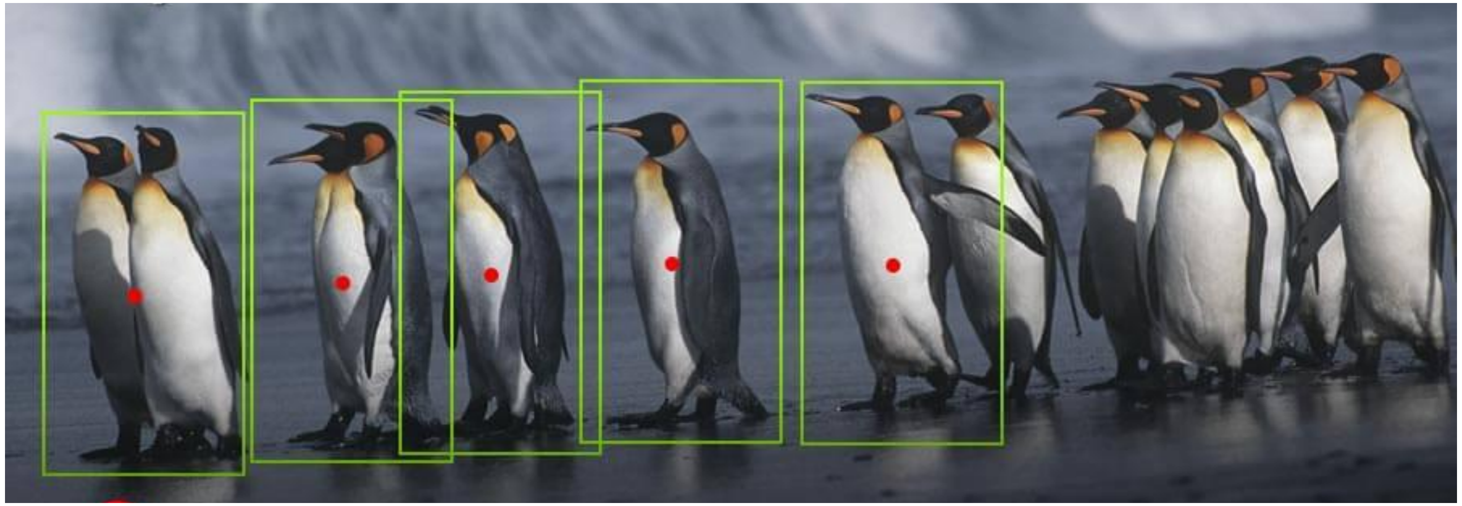
```
% Create a cascade detector object.
faceDetector = vision.CascadeObjectDetector();

% Read a video frame and run the face detector.
videoFileReader = vision.VideoFileReader('tilted_face.avi');
videoFrame      = step(videoFileReader);
bbox            = step(faceDetector, videoFrame);

% Draw the returned bounding box around the detected face.
videoFrame = insertShape(videoFrame, 'Rectangle', bbox);
figure; imshow(videoFrame); title('Detected face');
```

Kalman Filter

- The Kalman Filter is an optimal recursive data processing algorithm used for estimating the state of a dynamic system from a series of noisy measurements.
- It was developed by Rudolf E. Kálmán in the late 1950s.
- In computer vision, the Kalman Filter is widely used for tracking and estimation tasks.
- It helps in dealing with noisy measurements and predicting the state of objects in a dynamic environment.

Kalman Filter - Why

It can address the following challenges:



- Sensor noise: Cameras and other sensors introduce noise into measurements
- Occlusions: Objects can be partially or fully hidden from view
- Background clutter: Background noise can make it difficult to track objects

Kalman Filter Cycle

Prediction: Estimates the state at the next time step based on the previous state and a system model

Measurement update: Incorporates a new measurement to refine the predicted state

Kalman gain calculation: Determines how much weight to give to the prediction and the measurement

Estimate update: Combines the predicted state with the measurement using the Kalman gain

Kalman Filter - Advantages

- Progressive method
  - No large matrices has to be inverted
- Proper dealing with system noise
- Track finding and track fitting
- Detection of outliers
- Merging track from different segments

Kalman Filter - Applications

- Object tracking: Tracks the movement of objects in a video sequence (e.g., self-driving cars)
- Image stabilization: Reduces camera shake in videos
- Feature tracking: Tracks key points or features in an image sequence (e.g., facial feature tracking)
- Sensor fusion: Combines data from multiple sensors, e.g., camera and LiDAR (Light Detection and Ranging)

Kalman Filter - Code

Use Kalman Filter for Object Tracking

R2023a

Open Script

This example shows how to use the `vision.KalmanFilter` object and `configureKalmanFilter` function to track objects.

This example is a function with its main body at the top and helper routines in the form of nested functions.

```
function kalmanFilterForTracking
```

Introduction

The Kalman filter has many uses, including applications in control, navigation, computer vision, and time series econometrics. This example illustrates how to use the Kalman filter for tracking objects and focuses on three important features:

- Prediction of object's future location
- Reduction of noise introduced by inaccurate detections
- Facilitating the process of association of multiple objects to their tracks

Challenges of Object Tracking

Before showing the use of Kalman filter, let us first examine the challenges of tracking an object in a video. The following video shows a green ball moving from left to right on the floor.

```
showDetections();
```

Video

Video data is a series of images over time. Video in binary or intensity format is a series of single images. Video in RGB format is a series of matrices grouped into sets of three, where each matrix represents an R, G, or B plane.

Video file format often refers to either the *container format* or the *codec*.

Container format is like a box that holds different elements of a video such as the actual video, audio, subtitles, etc. It's the file that wraps everything together, such as MPEG (Moving Picture Experts Group), or AVI (Audio Video Interleave) or JPEG (Joint Photographic Experts Group).

Codec is a method or recipe used to compress and decompress the video itself.

So, the container is the box, and the codec is the way the contents are packed inside.

Video Reading

To read a video file, any application must:

- Recognize the container format (such as AVI).
- Have access to the codec that can decode the video data stored in the file. Some codecs are part of standard Windows® and Macintosh system installations, and allow you to play video in Windows Media® Player or QuickTime.
- Properly use the codec to decode the video data in the file.

```
>> info = mmfileinfo('shuttle.avi');
>> info.Video
ans =
  struct with fields:
    Format: 'MJPG'
    Height: 288
     Width: 512
```

The file, shuttle.avi, uses the MPEG codec

Common Video File Formats

- AVI (Audio Video Interleave) (.avi)
- Motion JPEG 2000 (.mj2)
- Moving Pictures Expert Group 4 (MPEG-4) (.mp4)

.AVI file is one of the oldest and most compatible video file formats. Many different codecs can be used with an .AVI file, which means that this format has more flexibility in choosing a balance between quality and size. However, these files tend to be larger than many other formats, which makes it less ideal for the web and more ideal for storing movies on a computer. AVI files can be both lossy or losslessly compressed, depending on the codecs used to encode the audio and video streams within the container.

Common Video File Formats

.MPG, .MP2, .MPEG, .MPE, .MPV files can play audio/video media, or simply audio. They are low in file size but also relatively low in quality. They also have lossy compression, meaning their quality will degrade after being edited numerous times. These files are best used when video will be recorded once and never edited.

.MP4, .M4P, .M4V are similar to .MPG files in that they can contain audio *and* video, or can simply be solely audio files. Their file formats are lossless, which makes them ideal for editing as they won't lose quality through subsequent edits and file saves. These are used for streaming video via the internet. They are generally higher in quality than .WEBM files, but tend to be larger in file size. .M4V files are proprietary iTunes files that share the same qualities of .MP4 and .M4P files. M4V files are DRM copy-protected.

Best Video Format?

Unfortunately, there is no single “best” video format. The best video format for you depends on how you would like to balance the quality and size of the video file. Some formats are extremely small and are great for web streaming, but are low quality. Other formats are high quality and the right choice for commercial videography but are very large in size.

Video Analytics

Video Analytics (VA) is the capability of automatically analyzing video to detect and determine temporal and spatial events.

Temporal events in a video are about things happening over time, like motion or changes from one frame to the next. Spatial events are about what's happening in one frame at a particular moment, like the position of objects in a single shot. Temporal is time-related, spatial is space-related.

VA is used in a wide range of domains including entertainment, health-care, retail, automotive, transport, home automation, flame and smoke detection, safety and security.

Video Analytics

Many different functionalities can be implemented in VA. Video Motion Detection is one of the simpler forms where motion is detected with regard to a fixed background scene. More advanced functionalities include identification, behavior analysis, video tracking and anomaly detection.

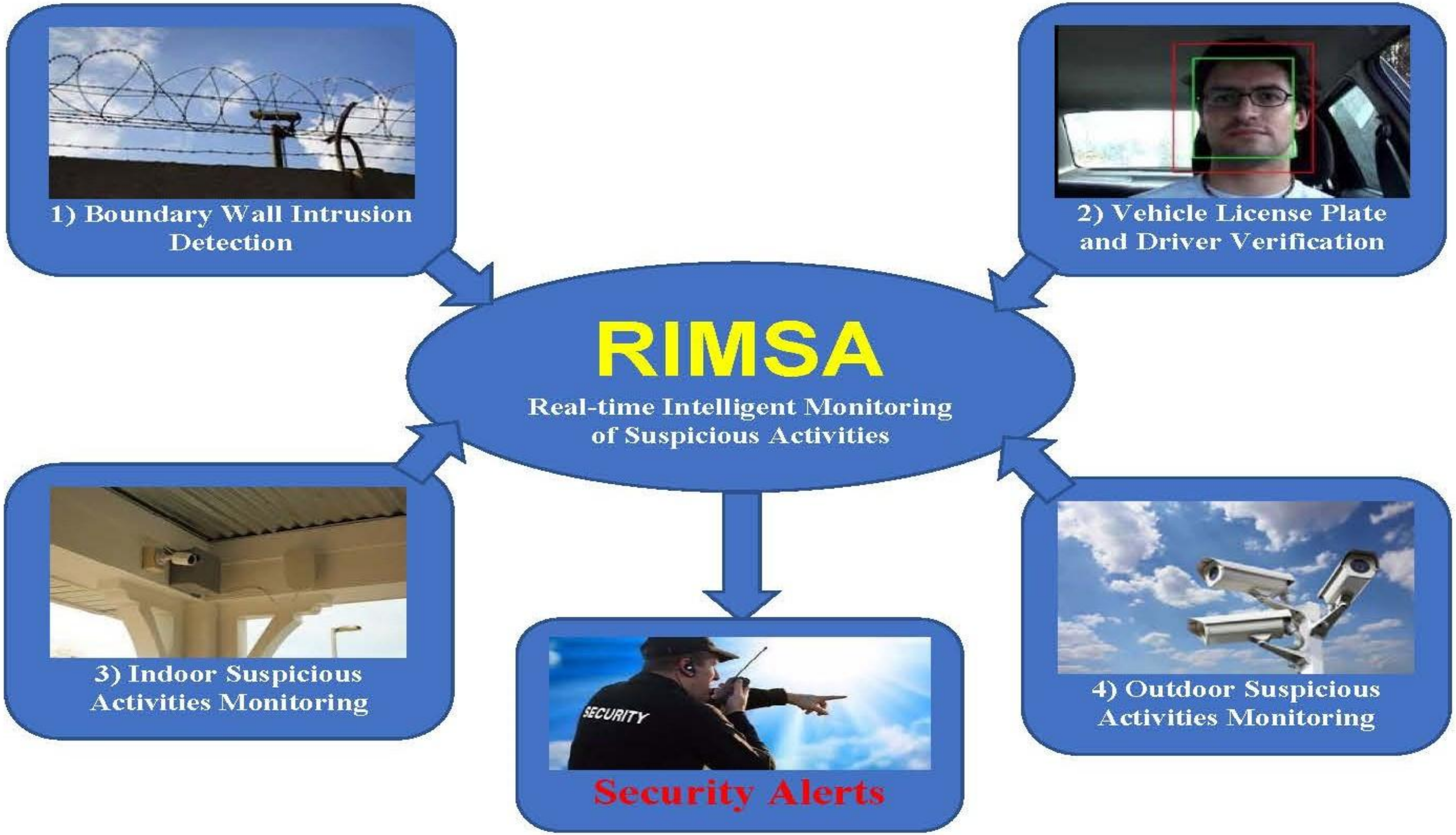
VA relies on good input video, so it is often combined with video enhancement technologies such as video denoising, image stabilization, and super-resolution.

Video Analytics

Smart cars

Mobileye Vision systems currently in high-end BMW, GM, Volvo models

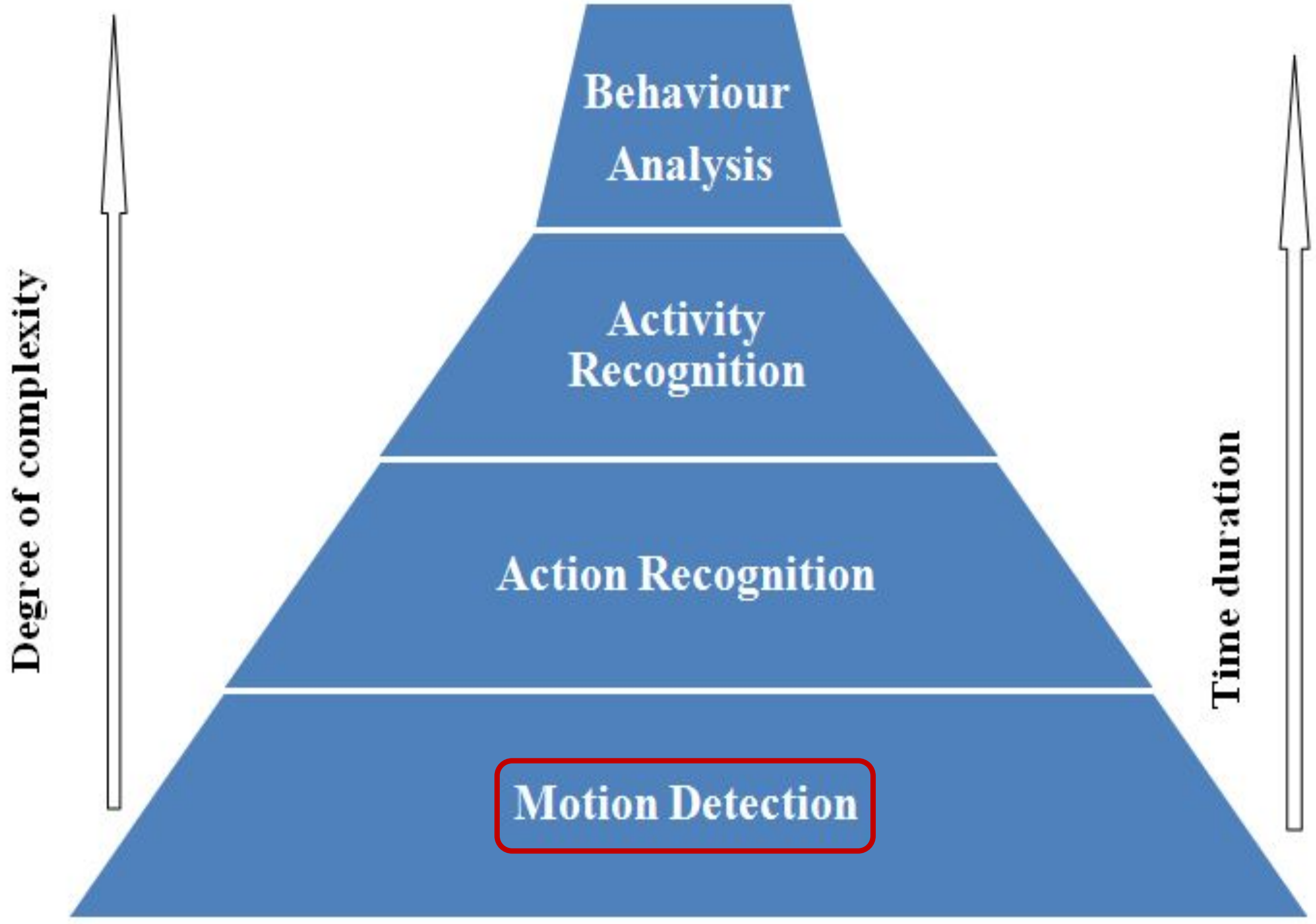
Video Analytics



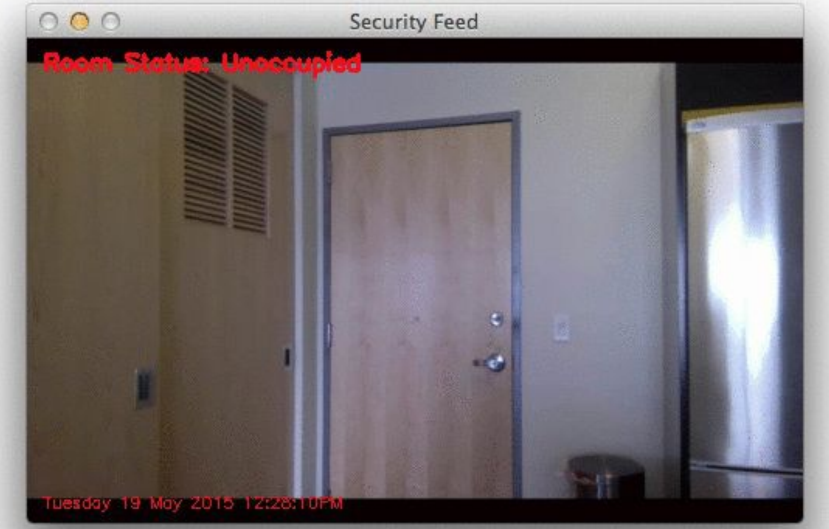
Applications

Video Analytics

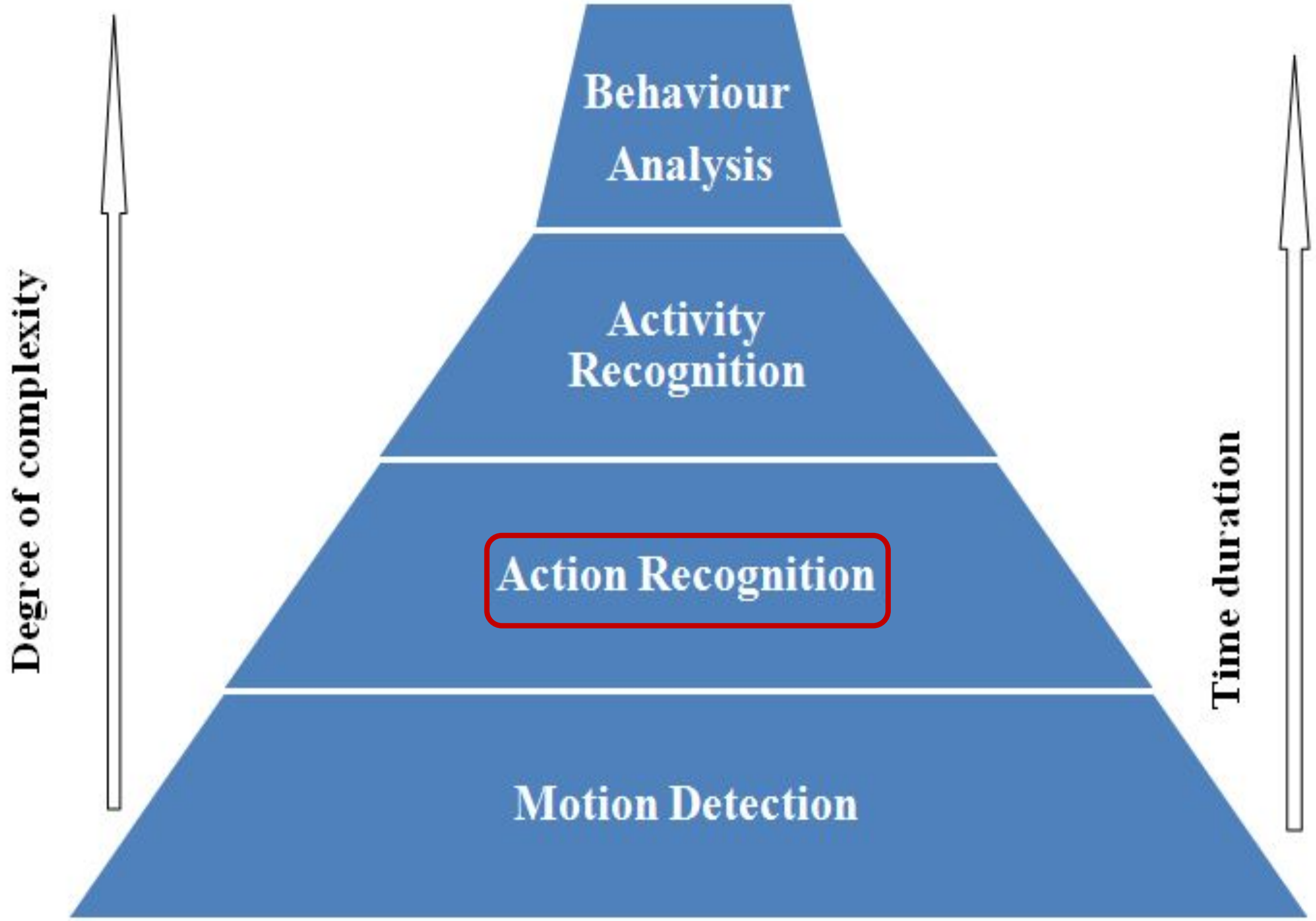
Video Analytics



Categories of anomalies detection

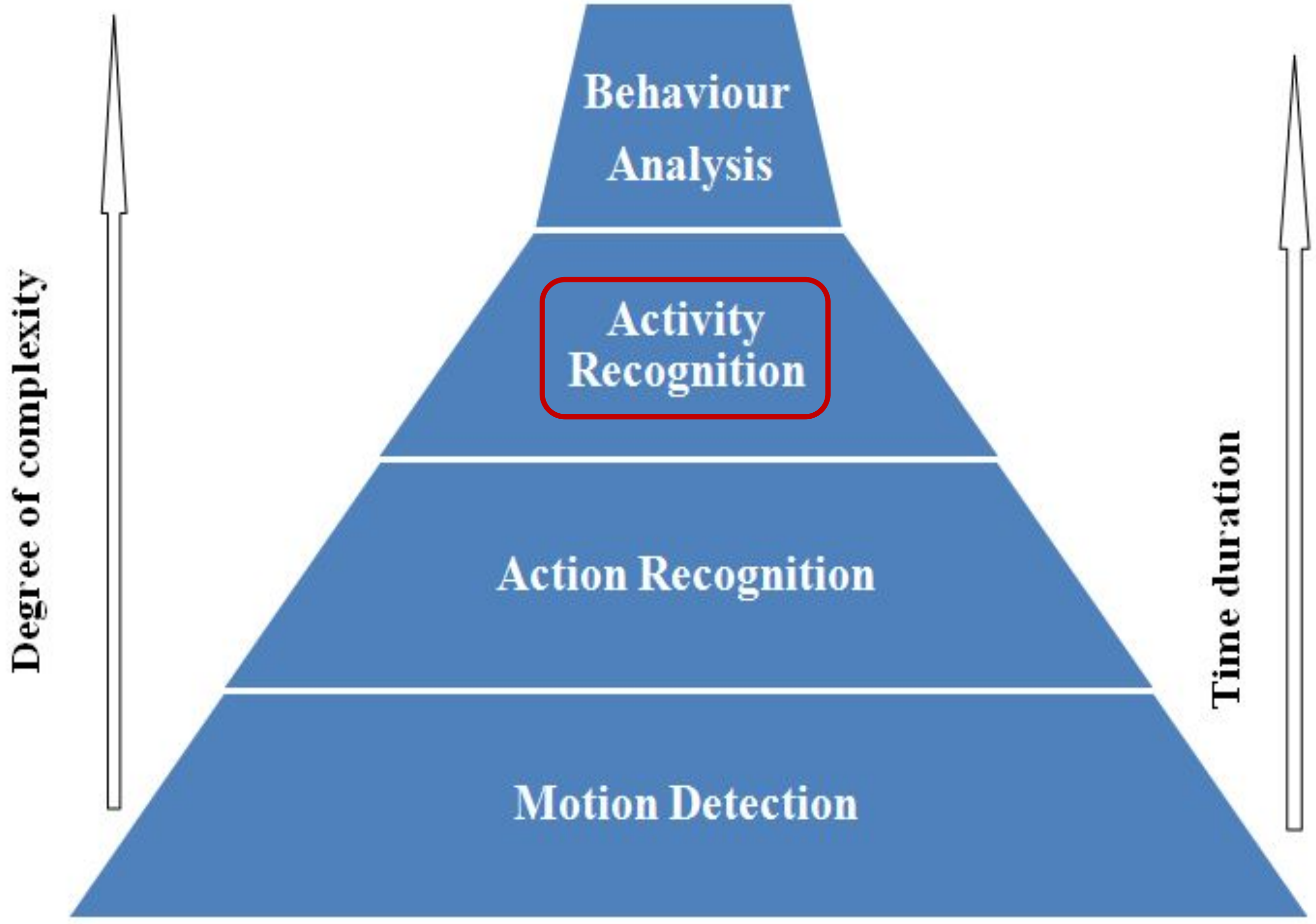


Video Analytics



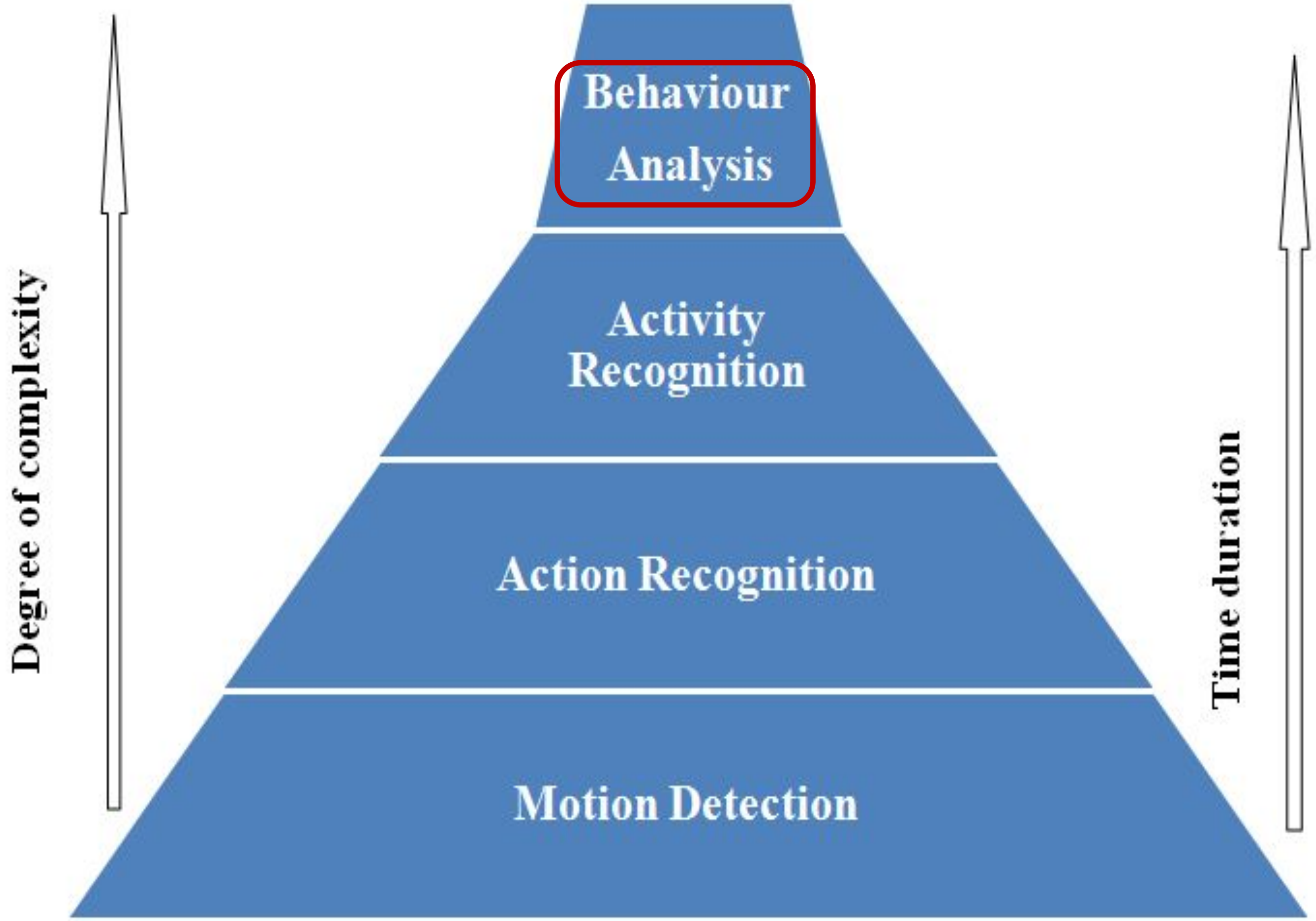
Categories of anomalies detection

Video Analytics



Categories of anomalies detection

Anomaly Detection



Categories of anomalies detection

Webcam Image Acquisition

With previous versions of the Image Acquisition Toolbox™, the files for all of the adaptors were included in your installation. Starting with version R2014a, each adaptor is available separately through support packages. In order to use the Image Acquisition Toolbox, you must install the adaptor that your camera uses.

Webcam support packages: Home => Add-Ons => Get Add-Ons

1. **Image Acquisition Toolbox Support Package for OS Generic Video Interface**
2. **MATLAB Support Package for USB Webcams**

Webcam Image Acquisition

% List of available webcams

```
>> webcamlist
```

```
ans =
  1×1 cell array
    {'HP Truevision HD'}
```

```
>> imaqhwinfo
```

```
ans =
  struct with fields:
    InstalledAdaptors: {'winvideo'}
        MATLABVersion: '9.6 (R2019a)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '6.0 (R2019a)'
```

```
>> info = imaqhwinfo('winvideo')
>> info = imaqhwinfo('winvideo', 1)
```

```
info =
  struct with fields:
      AdaptorDllName: 'C:\ProgramData\MATLAB\....\mwwinvideoimaq.dll'
   AdaptorDllVersion: '6.0 (R2019a)'
         AdaptorName: 'winvideo'
           DeviceIDs: {[1]}
          DeviceInfo: [1×1 struct]
```

```
info =
  struct with fields:
          DefaultFormat: 'MJPG_1280x720'
    DeviceFileSupported: 0
             DeviceName: 'HP Truevision HD'
               DeviceID: 1
  VideoInputConstructor: 'videoinput('winvideo', 1)'
 VideoDeviceConstructor: 'imaq.VideoDevice('winvideo', 1)'
       SupportedFormats: {1×14 cell}
```

Webcam Image Acquisition


```
>> cam = webcam % Create any available webcam object and connect with camera with light on but no preview window

>> preview(cam)  % Show preview window

>> img = snapshot(cam);  % Capture frame

>> imshow(img); % Show frame as an image

>> closePreview(cam) % Close preview window but camera light still on

>> clear('cam');  % Switch off camera light and also close preview if open
```

Live Motion Detection


```
clc; clear;
cam = webcam; % Create any available webcam object and connect with camera with light on but no preview window
%preview(cam);  % Show preview window
PrevFrame = rgb2gray(snapshot(cam));
load gong.mat; % Sound file
for idx = 1:100
  rgbFrame = snapshot(cam);
  NextFrame = rgb2gray(rgbFrame);
  Dist = sqrt(sum((PrevFrame(:) - NextFrame(:)).^2)); % Euclidean distance between frames
  if Dist > 2000 % Less value more sensitive
     msg = 'Motion start';
     sound(y); % Sound from gong.mat
  else
     msg = 'No motion';
  end
  imshow(rgbFrame), title(msg);
  PrevFrame = NextFrame;
  hold on;
end
%closePreview(cam); % Close preview window but camera light still on
clear('cam'); % Switch off camera light and also close preview if open
```

Webcam Image Acquisition: Python…


```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();
```

Webcam Image Acquisition: Python…


```
    // Resize the output to fit the video element.
    google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

    // Wait for Capture to be clicked.
    await new Promise((resolve) => capture.onclick = resolve);

    const canvas = document.createElement('canvas');
    canvas.width = video.videoWidth;
    canvas.height = video.videoHeight;
    canvas.getContext('2d').drawImage(video, 0, 0);
    stream.getVideoTracks()[0].stop();
    div.remove();
    return canvas.toDataURL('image/jpeg', quality);
  }
  ''')
display(js)
data = eval_js('takePhoto({})'.format(quality))
binary = b64decode(data.split(',')[1])
with open(filename, 'wb') as f:
  f.write(binary)
return filename
```

Webcam Image Acquisition: Python


```
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))
```

Exercise

Write an algorithm and code for live motion detection and tracking of human only. Hint: may use skin detection.

Case Study

You have recently joined the traffic control department of Lahore as a programmer.

Your 1st task is to make a flow chart and write a pseudo code of your program which can track heavy vehicles (truck etc.) from the video of traffic. Implement your program and do a test run on a video of traffic.

Now you are promoted as a system administrator. Your next task is to design a system which can track & count the number of vehicles at any crossing with the help of a surveillance camera. Make a flow chart which can guide your programmer to implement the system. Data should be accessible for any slot of times.