Efficient Video Coding in ADAS

Presented by,

Nithil Anantheshwar Rao

Imma Nummer: 93851

Subject: Efficient Video Coding

Submitted to,

Prof. Dr.-Ing. Christian Langen

Hochschule Karlsruhe
University of
Applied Sciences

HKA

Where Efficient Video Coding Is Used In ADAS

Automotive Applications

- Lane Keeping Assist
- Adaptive Cruise Control
- Pedestrian Detection

Project Objective

Preprocessing

Initial cleanup and noise reduction

Compression

Data size reduction with quality trade-offs

Lane Detection

Extract road markers for driving assistance

Object Detection & Tracking

Identify and follow moving objects

Raw Video Input

Video Source

RGB video stream

Challenges

High redundancy, irrelevant background details

Next Step

Preprocessing to prepare data

Preprocessing Techniques

Grayscale Conversion

Eliminates color channels to reduce computation

Gaussian Blur

Removes high-frequency noise, aiding edge detection

MATLAB Code (GrayGauss)

```matlab
function graygauss(inputVideo)
    % GRAYGAUSS Shows all video processing stages in one window
    % Displays original, grayscale, and blurred video in tiled layout
    % Usage: graygauss('video_path.mp4')

    %% 1. Initialize Video Reader
    if ~exist(inputVideo, 'file')
        error('Video file not found: %s', inputVideo);
    end
    vr = VideoReader(inputVideo);

    %% 2. Create Single Figure with Tiled Layout
    fig = figure('Name', 'Video Processing Pipeline', ...
                'Position', [100 100 1200 800], ...
                'NumberTitle', 'off');

    % Create tiled layout (1 row, 3 columns)
    t = tiledlayout(fig, 1, 3, 'Padding', 'none', 'TileSpacing', 'compact');

    % Create axes for each video stream
    ax1 = nexttile(t); h1 = imshow(zeros(vr.Height, vr.Width, 3, 'uint8'));
    title(ax1, 'Original Video');

    ax2 = nexttile(t); h2 = imshow(zeros(vr.Height, vr.Width, 'uint8'));
    title(ax2, 'Grayscale Conversion');

    ax3 = nexttile(t); h3 = imshow(zeros(vr.Height, vr.Width, 'uint8'));
    title(ax3, 'Gaussian Blur (σ=2)');

    %% 3. Real-Time Processing Loop
    try
        while hasFrame(vr) && isvalid(fig)
            % Read current frame
            originalFrame = readFrame(vr);

            % Processing pipeline
            grayFrame = rgb2gray(originalFrame);
            blurredFrame = imgaussfilt(grayFrame, 2);

            % Update displays
            set(h1, 'CData', originalFrame);
            set(h2, 'CData', grayFrame);
            set(h3, 'CData', blurredFrame);

            % Control playback speed and update display
            pause(1/vr.FrameRate);
            drawnow;
        end
    catch ME
        disp(['Processing stopped: ' ME.message]);
    end

    %% 4. Cleanup
    if isvalid(fig), close(fig); end
    close(vr);
end
```

DCT-Based Compression

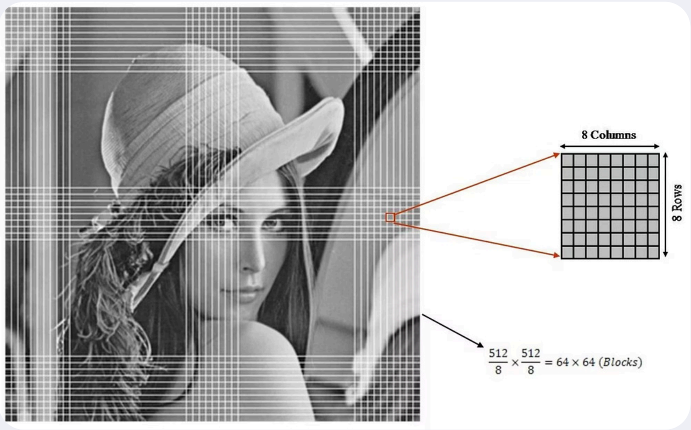
Purpose

Reduce data size, keep key visual features

Method

Block-wise DCT plus quantization

(Discrete Cosine Transform)

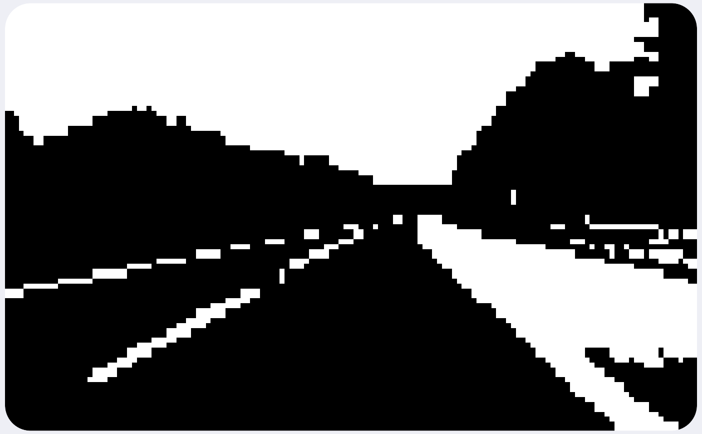


8×8 pixels

Reduce Spatial Redundancy

Lower Q

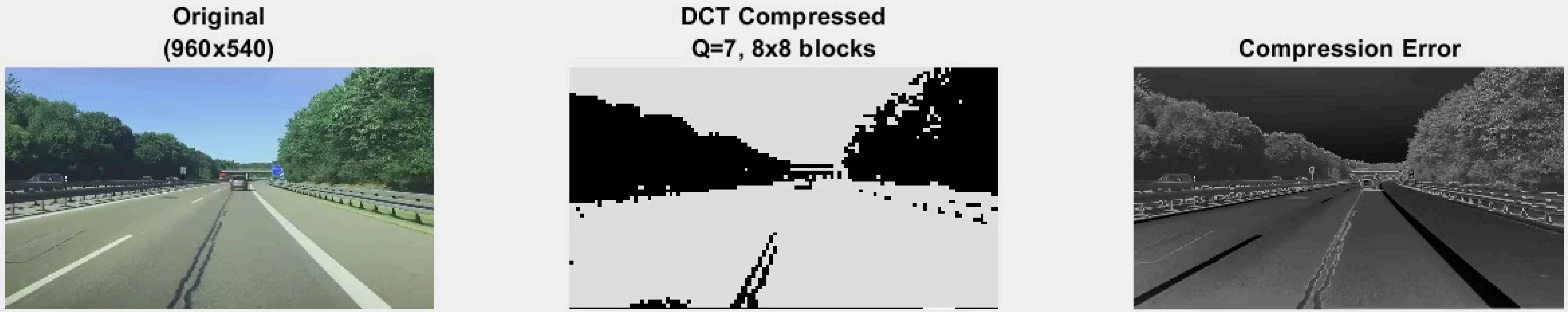
Low Q: High quality, low compression



Higher Q

High Q: Low quality, high compression

Discrete Cosine Transform



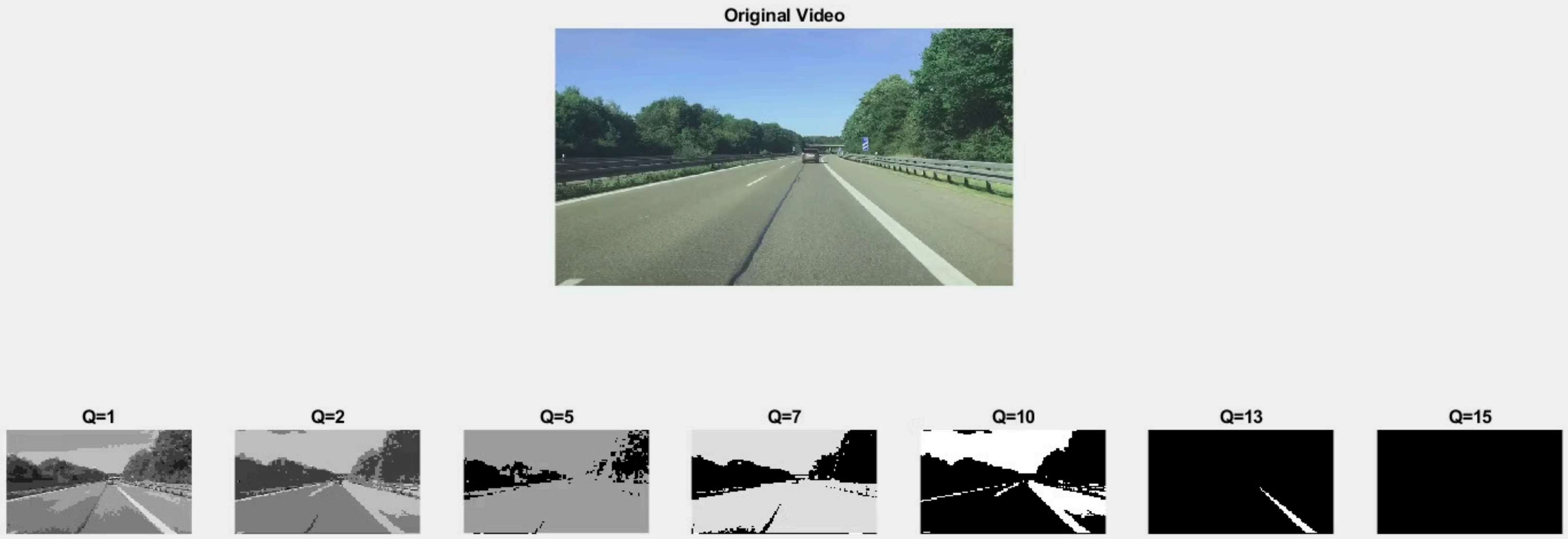
Video is divided into 8×8 blocks where DCT reduces spatial redundancy. The brighter areas in the error image show where more data was lost during compression.

MATLAB Code (DCT)

```matlab
function DCT(inputVideo, Q)
    % DCTVIDEOCOMPRESSION Demonstrates DCT-based video compression
    % Shows original, compressed, and error frames side-by-side
    % Usage: dctVideoCompression('video.mp4', Q)
    %   Q: Quantization factor (higher = more compression)

    %% 1. Initialize Video
    if ~exist(inputVideo, 'file')
        error('Video file not found: %s', inputVideo);
    end
    vr = VideoReader(inputVideo);

    %% 2. Create Display Window
    fig = figure('Name', 'DCT Video Compression', ...
                 'Position', [100 100 1200 400]);

    % Original video
    ax1 = subplot(1,3,1);
    h1 = imshow(zeros(vr.Height, vr.Width, 3, 'uint8'));
    title(sprintf('Original\n(%dx%d)', vr.Width, vr.Height));

    % Compressed video
    ax2 = subplot(1,3,2);
    h2 = imshow(zeros(vr.Height, vr.Width, 'uint8'));
    title(sprintf('DCT Compressed\nQ=%d, 8x8 blocks', Q));

    % Error visualization
    ax3 = subplot(1,3,3);
    h3 = imshow(zeros(vr.Height, vr.Width, 'uint8'));
    title('Compression Error');
```

```matlab
    %% 3. DCT Processing Pipeline
    while hasFrame(vr) && isvalid(fig)
        % Read and convert frame
        original = readFrame(vr);
        gray = im2double(rgb2gray(original));

        % DCT Compression
        dctFun = @(block) round(dct2(block.data) ./ Q);
        dctBlocks = blockproc(gray, [8 8], dctFun);

        % Reconstruction (inverse DCT)
        idctFun = @(block) idct2(block.data * Q);
        compressed = blockproc(dctBlocks, [8 8], idctFun);

        % Convert back to display format
        compressed8 = im2uint8(compressed);
        errorImg = im2uint8(abs(gray - compressed));

        % Update displays
        set(h1, 'CData', original);
        set(h2, 'CData', compressed8);
        set(h3, 'CData', errorImg);

        % Control playback speed
        pause(1/vr.FrameRate);
        drawnow;
    end

    %% 4. Cleanup
    if isvalid(fig), close(fig); end
    close(vr);
end
```

Role of Quantization Factor



Q Value	What It Means	Visual Quality	Compression
Low Q (e.g., 1–3)	Less quantization	High quality	Low compression
Medium Q (e.g., 5–10)	Balanced quantization	Good quality	Moderate compression
High Q (e.g., 13–15)	Aggressive quantization	Low quality	High compression

MATLAB Code (DCT Comparison)

```matlab
function dctComparison(inputVideo, Q_values)
    % DCTVIDEOCOMPARISON Shows original vs multiple DCT-compressed versions
    % Usage: dctVideoComparison('video.mp4', [1,2,5,7,10,13,15])

    %% 1. Initialize Video
    if ~exist(inputVideo, 'file')
        error('Video file not found: %s', inputVideo);
    end
    vr = VideoReader(inputVideo);

    %% 2. Create Figure with Tiled Layout
    fig = figure('Name', 'DCT Compression Comparison', ...
                'Position', [100 100 150*length(Q_values)+300 500]);

    % Create tiled layout (1 row for original + N rows for Q values)
    t = tiledlayout(fig, 2, length(Q_values), 'TileSpacing', 'compact');

    %% 3. Initialize Displays
    % Original video
    ax0 = nexttile(t, [1 length(Q_values)]);
    h0 = imshow(zeros(vr.Height, vr.Width, 3, 'uint8'));
    title(ax0, 'Original Video');

    % Create axes for each Q value
    h = gobjects(1, length(Q_values));
    for i = 1:length(Q_values)
        ax = nexttile(t);
        h(i) = imshow(zeros(vr.Height, vr.Width, 'uint8'));
        title(ax, sprintf('Q=%d', Q_values(i)));
    end

    %% 4. Real-Time Processing
    while hasFrame(vr) && isvalid(fig)
```

```matlab
    %% 4. Real-Time Processing
    while hasFrame(vr) && isvalid(fig)
        % Read frame
        original = readFrame(vr);
        gray = im2double(rgb2gray(original));

        % Process for each Q value
        compressed_frames = cell(1, length(Q_values));
        for i = 1:length(Q_values)
            % DCT Compression Pipeline
            dctFun = @(block) round(dct2(block.data)/Q_values(i));
            quantized = blockproc(gray, [8 8], dctFun);

            % Reconstruction
            idctFun = @(block) idct2(block.data*Q_values(i));
            reconstructed = blockproc(quantized, [8 8], idctFun);

            compressed_frames{i} = im2uint8(reconstructed);
        end

        % Update displays
        set(h0, 'CData', original);
        for i = 1:length(Q_values)
            set(h(i), 'CData', compressed_frames{i});
        end

        pause(1/vr.FrameRate);
        drawnow;
    end

    close(vr);
```

Lane Detection Pipeline

1 **Canny Edge Detection**

Detects strong and weak edges in the image by computing image gradients. Essential for identifying lane boundaries in high-contrast areas.

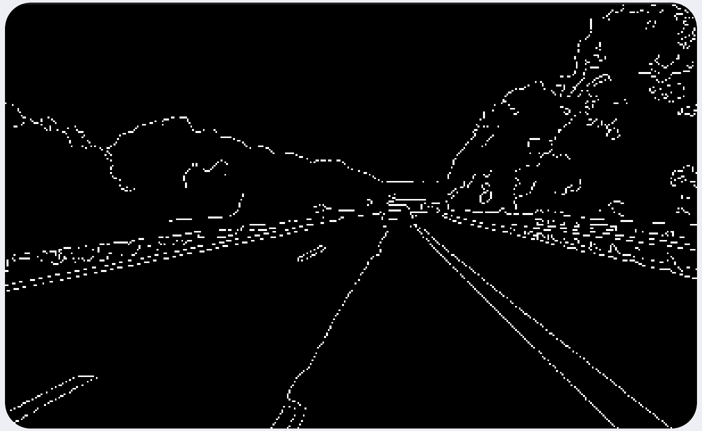
2 **Region of Interest**

Applies a mask to focus only on the relevant part of the image (e.g., road area), reducing noise and false detections outside the drivable space.

3 **Hough Transform**

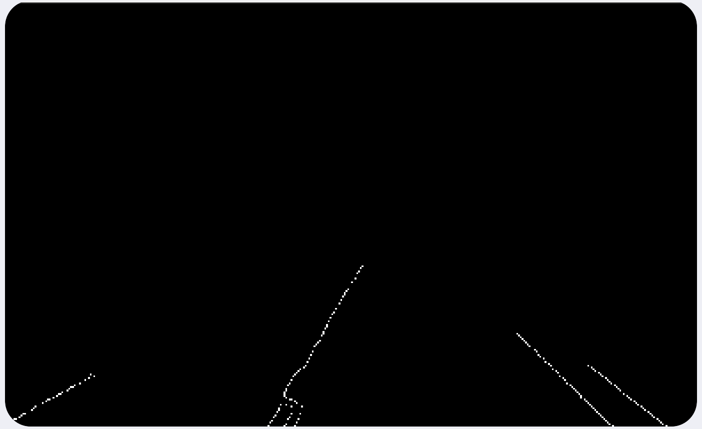
Detects lines by converting edge points into a parameter space and finding co-linear arrangements. Ideal for detecting lane markings from edges.

Lane Detection Pipeline



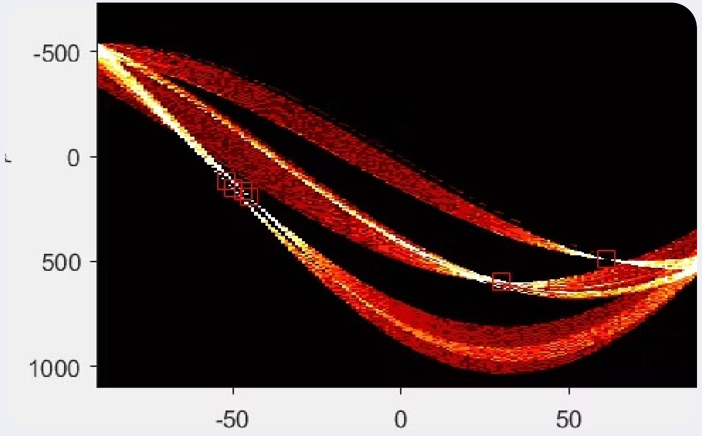
Canny Edge Detection

Detects Edges



Region of Interest

Removes the irrelevant parts



Hough Transform

Detects straight lines from edges.

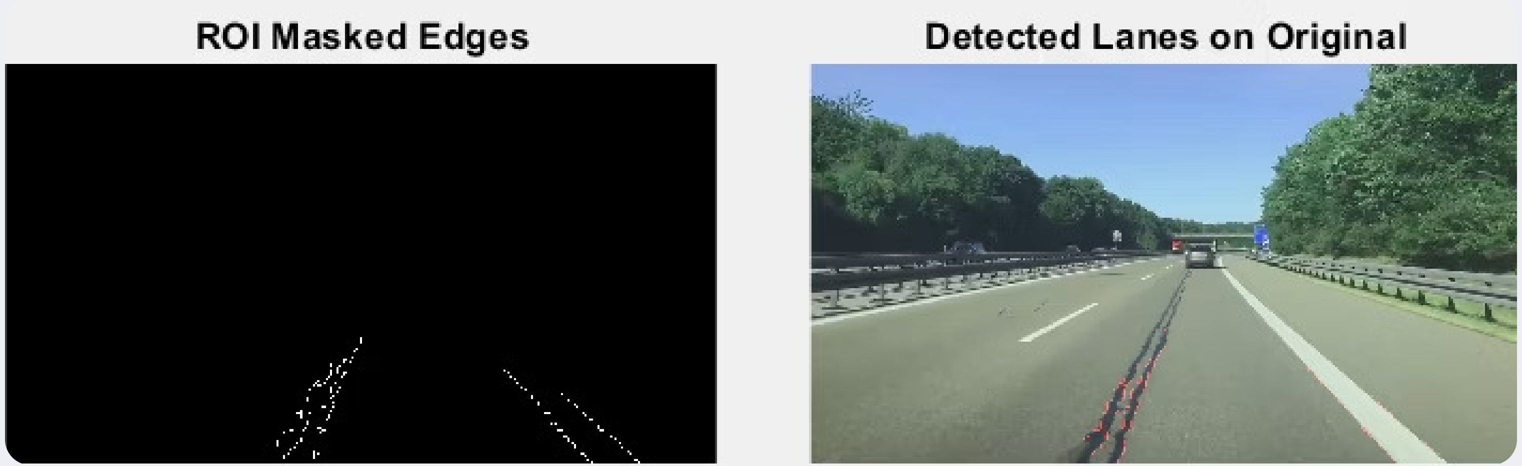
MATLAB Code (CannyROI)

```matlab
function cannyroi(inputVideo)
    % LANEDETECTIONWITHDCT Shows pipeline from DCT compression to lane detection
    % Compares DCT output with Canny edges and ROI masking
    % Usage: laneDetectionWithDCT('your_video.mp4')

    %% 1. Initialize Video
    if ~exist(inputVideo, 'file')
        error('Video file not found: %s', inputVideo);
    end
    vr = VideoReader(inputVideo);

    %% 2. Create Figure with Tiled Layout
    fig = figure('Name', 'Lane Detection Pipeline with DCT', ...
                'Position', [100 100 1200 400]);

    % Create 1x4 tile layout (DCT compressed | Canny edges | ROI edges | Combined)
    t = tiledlayout(1, 4, 'Padding', 'none', 'TileSpacing', 'compact');

    %% 3. Initialize Processing Parameters
    params.cannyThresh = [0.1 0.3];   % Canny edge thresholds
    params.roiHeight = 0.6;           % ROI covers lower 60% of image
    params.gaussianSigma = 2;         % Blurring strength
    params.dctThreshold = 0.1;        % DCT compression threshold (keep 10% of coeffs)

    %% 4. Processing Loop
    while hasFrame(vr) && isvalid(fig)
        % Read and preprocess frame
        frame = readFrame(vr);
        gray = rgb2gray(frame);

        %% Stage 0: DCT Compression (Your Existing Pipeline)
        dctFrame = performDCTCompression(gray, params.dctThreshold);
```


```matlab
        %% Stage 0: DCT Compression (Your Existing Pipeline)
        dctFrame = performDCTCompression(gray, params.dctThreshold);
```

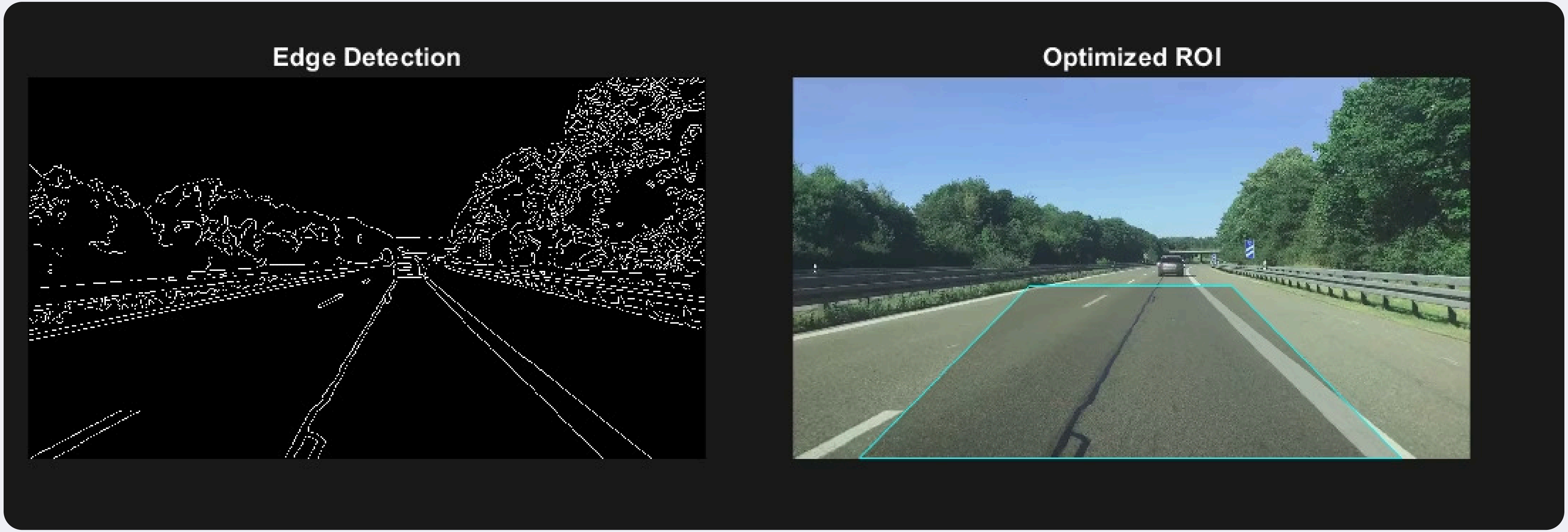

```matlab
        %% Stage 1: Edge Detection on DCT Output
        blurred = imgaussfilt(dctFrame, params.gaussianSigma);
        edges = edge(blurred, 'Canny', params.cannyThresh);

        %% Stage 2: ROI Masking
        [rows, cols] = size(edges);
        roiY = round(params.roiHeight * rows);
        roiPoints = [1, rows; cols/2, roiY; cols, rows];
        roiMask = poly2mask(roiPoints(:,1), roiPoints(:,2), rows, cols);
        maskedEdges = edges & roiMask;

        %% Stage 3: Combined Visualization
        combinedVis = frame;
        [y, x] = find(maskedEdges);
        combinedVis(sub2ind(size(combinedVis), y, x)) = 255; % Mark edges in red

        %% Display Results
        % DCT compressed frame
        nexttile(1);
        imshow(dctFrame, 'Border', 'tight');
        title(sprintf('DCT Compressed (%.0f%% coeffs)', params.dctThreshold*100));

        % Canny edges
        nexttile(2);
        imshow(edges, 'Border', 'tight');
        title('Canny Edge Detection');

        % ROI masked edges
        nexttile(3);
```

Focused Lane Detection



- Only the lower road portion is processed for edge detection and line extraction, improving both speed and accuracy.
- Detected lane lines are highlighted using the masked region.

Optimized ROI



Detecting Moving Objects

Frame Differencing

Concept:

- Subtract previous frame from current
- Highlight pixels that have changed
- Motion = pixel intensity difference

Blob Analysis

Concept:

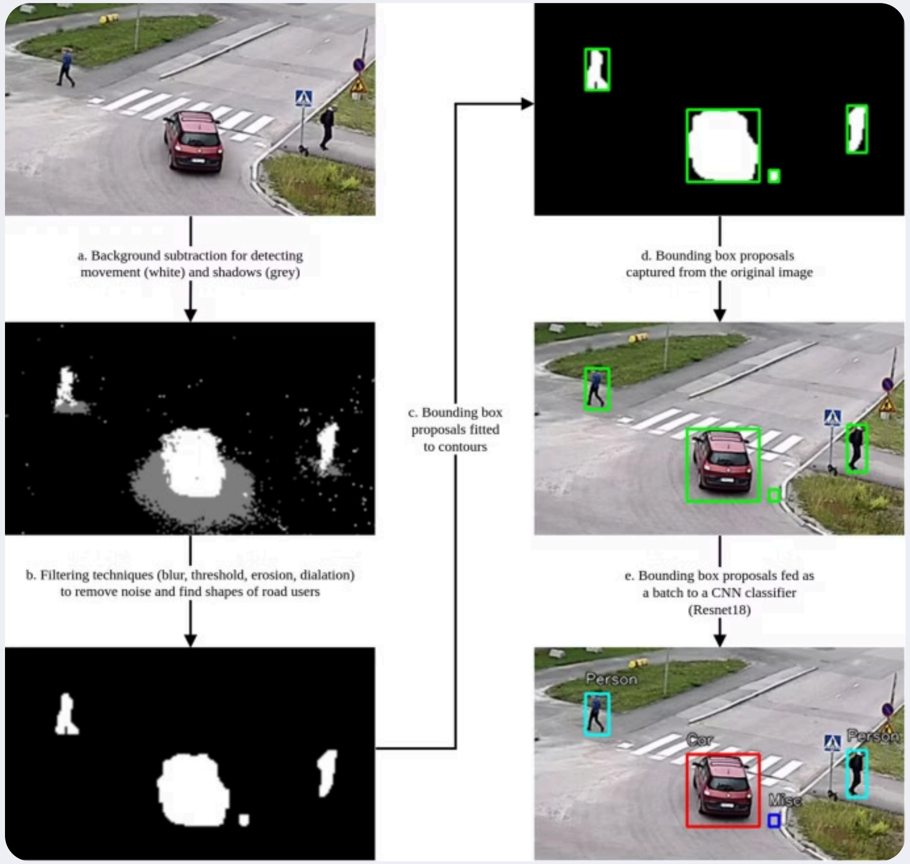
- Detect connected regions in the binary mask
- Group them as "blobs" representing objects
- Draw bounding boxes to track

Kalman Filter

Concept:

- Takes current position + velocity
- Predicts next position
- Corrects prediction if the actual observation is available

Frame Differencing and Blob Analysis



Frame differencing detects movement, and blob analysis turns it into meaningful tracked objects.

Image Source: **Journal of Big Data**

Kalman Filter-Based Tracking

Predictive Tracking

Estimates object location despite occlusion

Unique IDs

Assigns consistent identifiers to tracked objects

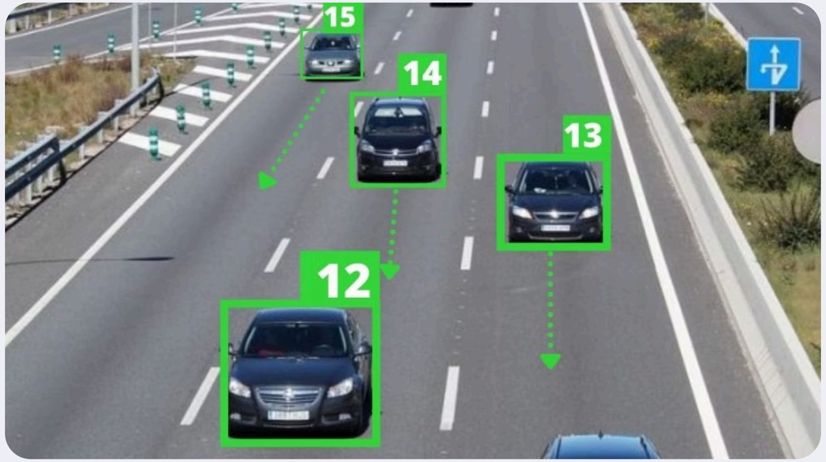


Image Source: [Pysource]

Gaps Compared to Industry Use

- **Real-time Embedded Systems:** Integration with hardware like Jetson or STM32 missing
- **Deep Learning Models:** YOLOv12 could improve detection over frame differencing
- **Semantic Segmentation:** Needed for enhanced lane marking accuracy
- **Sensor Fusion:** Combining LiDAR and video for robust environment perception

Summary & Takeaways

Essential Role of Video Processing

Efficient video coding is key for reliable ADAS performance.

Educational Pipeline

This project simulates core industry workflows simply and clearly.

Future-ready Foundation

Provides groundwork for real-world ADAS application expansion.

Thank You!

Please feel free to ask any questions or share your thoughts.