

Machine Vision Algorithm Development and Simulation with NXP Vision Toolbox for MATLAB[®] on S32V Processors

Mike Doidge

NXP Automotive Microcontroller & Processors

June 2019 | Session #AMF-AUT-T3653



SECURE CONNECTIONS
FOR A SMARTER WORLD

Agenda

- NXP Vision Toolbox – Introduction
- Embedded System Development Process with MATLAB for S32V
- Object & Feature Detection Demo
- CNN SqueezeNet Demo

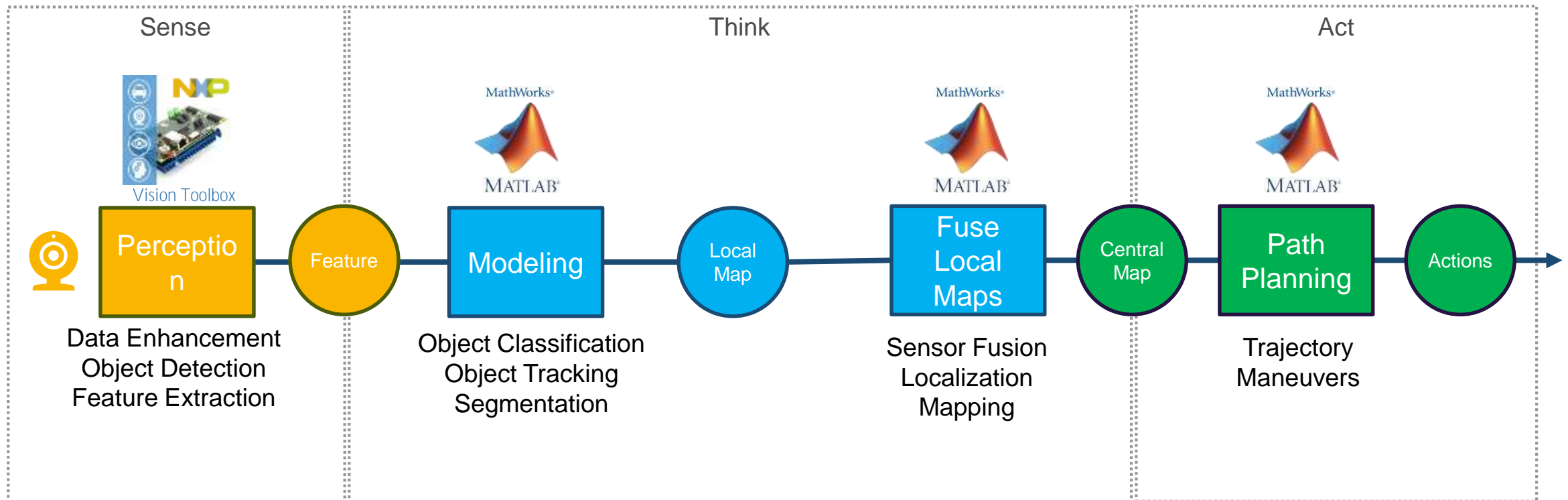
NXP Vision Toolbox – Introduction

Faster Time to Market



What is Vision Toolbox and How Can It Help?

“Typical” ADAS computing domain partitioning: Sense – Think – Act



Tools Ecosystem

MathWorks Tools for Fast Prototyping & Validation

Perception

Modeling

Fusion

Acting

- [Image Processing](#)

Processing, 3D workflow, Analysis



- [Computer Vision](#)

Object detection, Recognition, Calibration, Stereo Vision, Tracking, Extraction



- [Deep Learning](#)

Design, Train, Transfer, Pretrained, Framework interop



- [Advanced Driving System](#)

Labeling, Sensor Fusion & Tracking, Classification, Scenario Generation

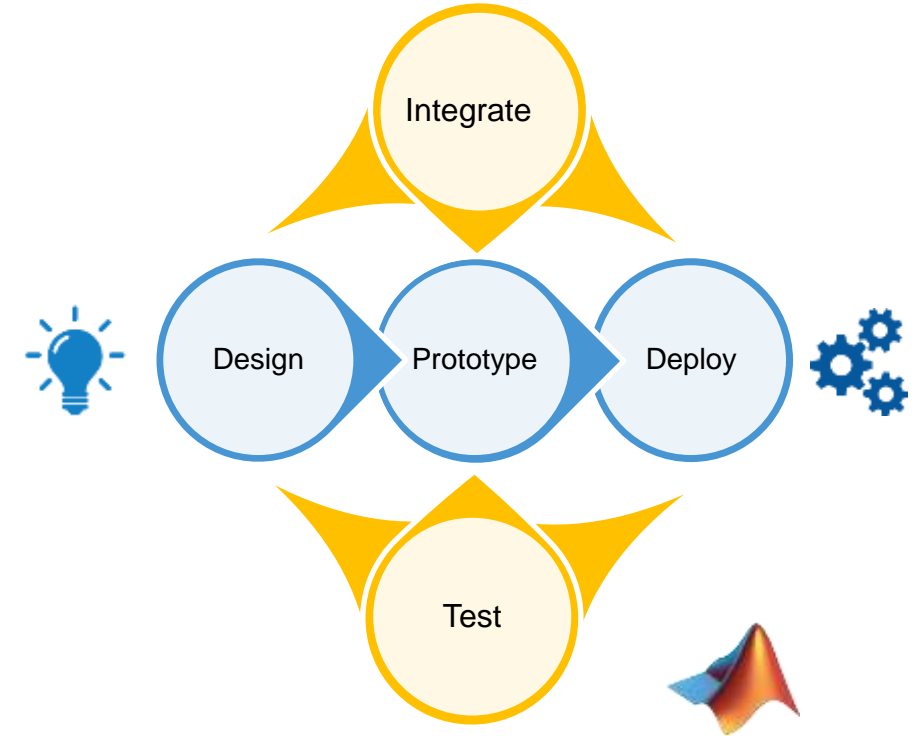
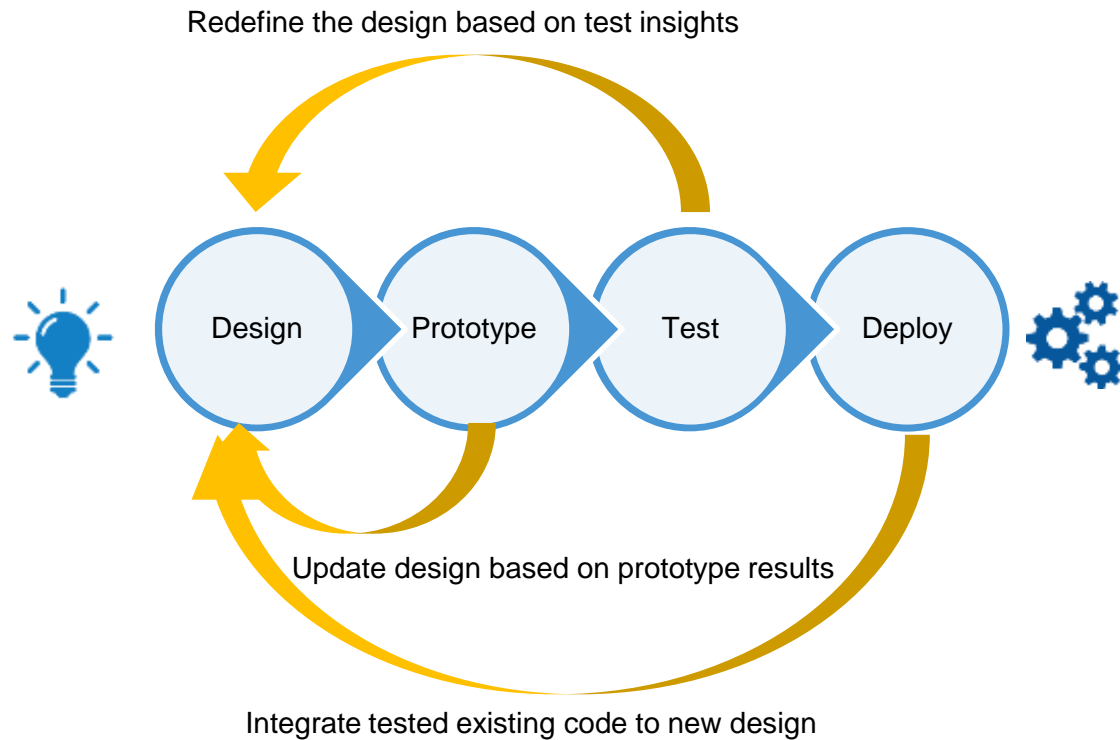


- [Robotic System](#)

Coordinate transformation, ROS, Ground Vehicle Algorithms, Code Generation, Log File and Analysis



Traditional vs. Model-Based Design Dev. Process

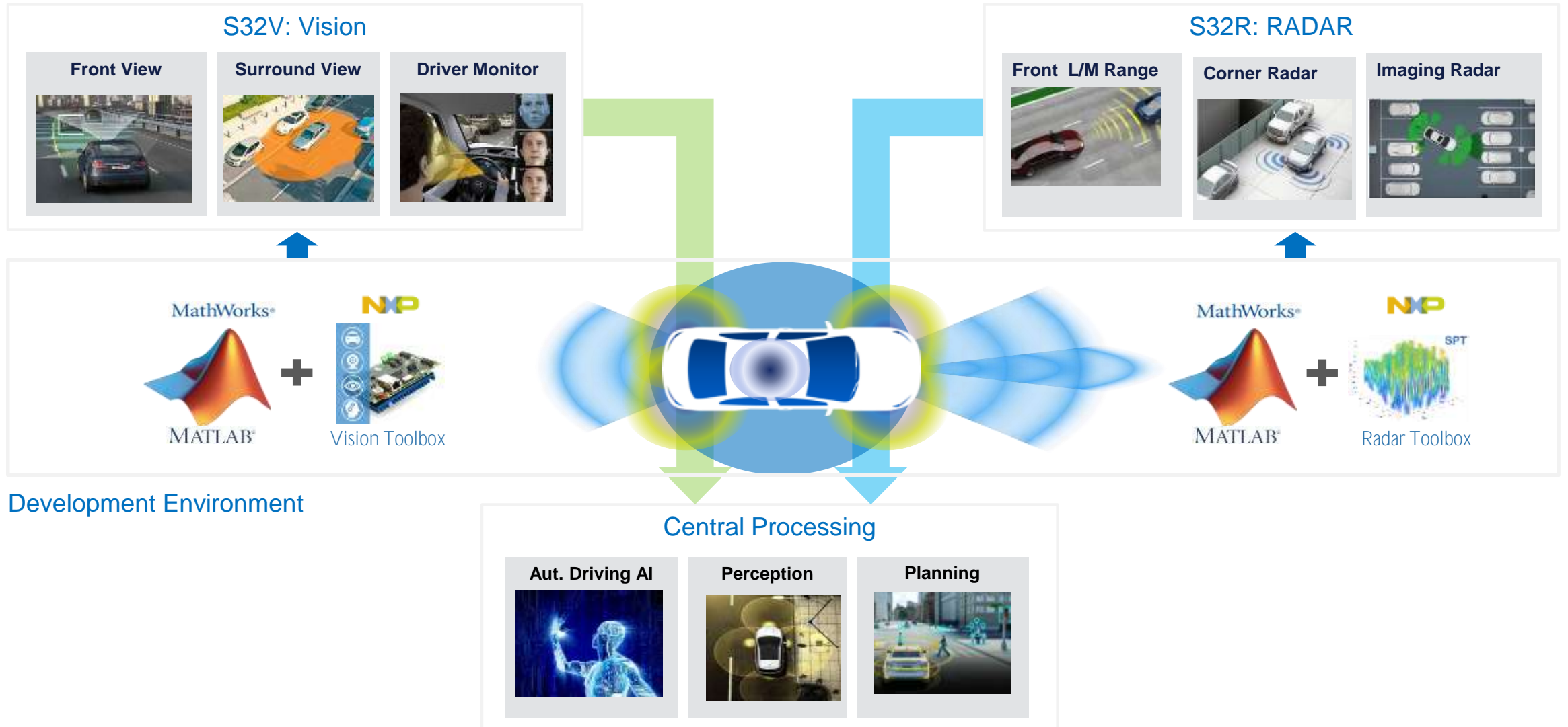


Embedded System Development in MATLAB Environment for S32V

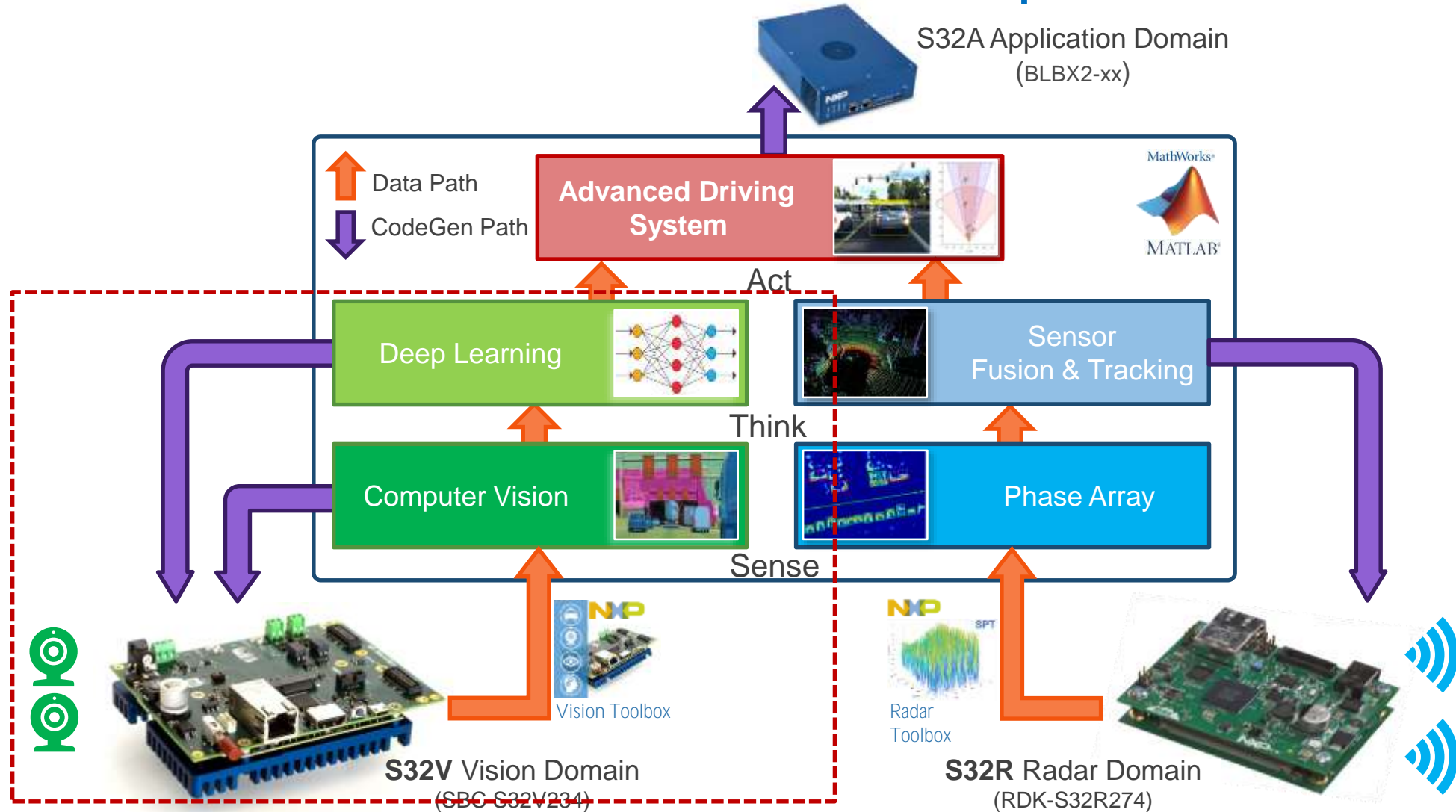
Simulate, Test, Build and Deploy



NXP HW & SW Solutions for ADAS



ADAS Solution with MATLAB – Concept



NXP Vision Toolbox – Preview

The screenshot displays the MATLAB R2018a environment with the NXP Vision Toolbox lane detection example open. The interface is divided into several panes:

- Current Folder:** Shows the file structure of the toolbox, including folders like 'examples', 'apps', and 'face_detection', and files like 'laneDetection_camera.m' and 'laneDetection_image.m'.
- Editor:** Contains two files:
 - laneDetection_image_ismix.m:** A script titled "Lane Detection" that reads an input image and displays it. The code includes:

```
inImagePath = 'data/lane_detection.jpg';
inImgMat = nxpvt.imread(inImagePath);
if isempty(inImgMat)
    fprintf('Failed to open input image: %s.', inImagePath);
    return;
end
nxpvt.imshow(inImgMat);
```
 - laneDetection_camera.m:** A function titled "Lane Detection camera" that initializes objects and creates a shape inserter. The code includes:

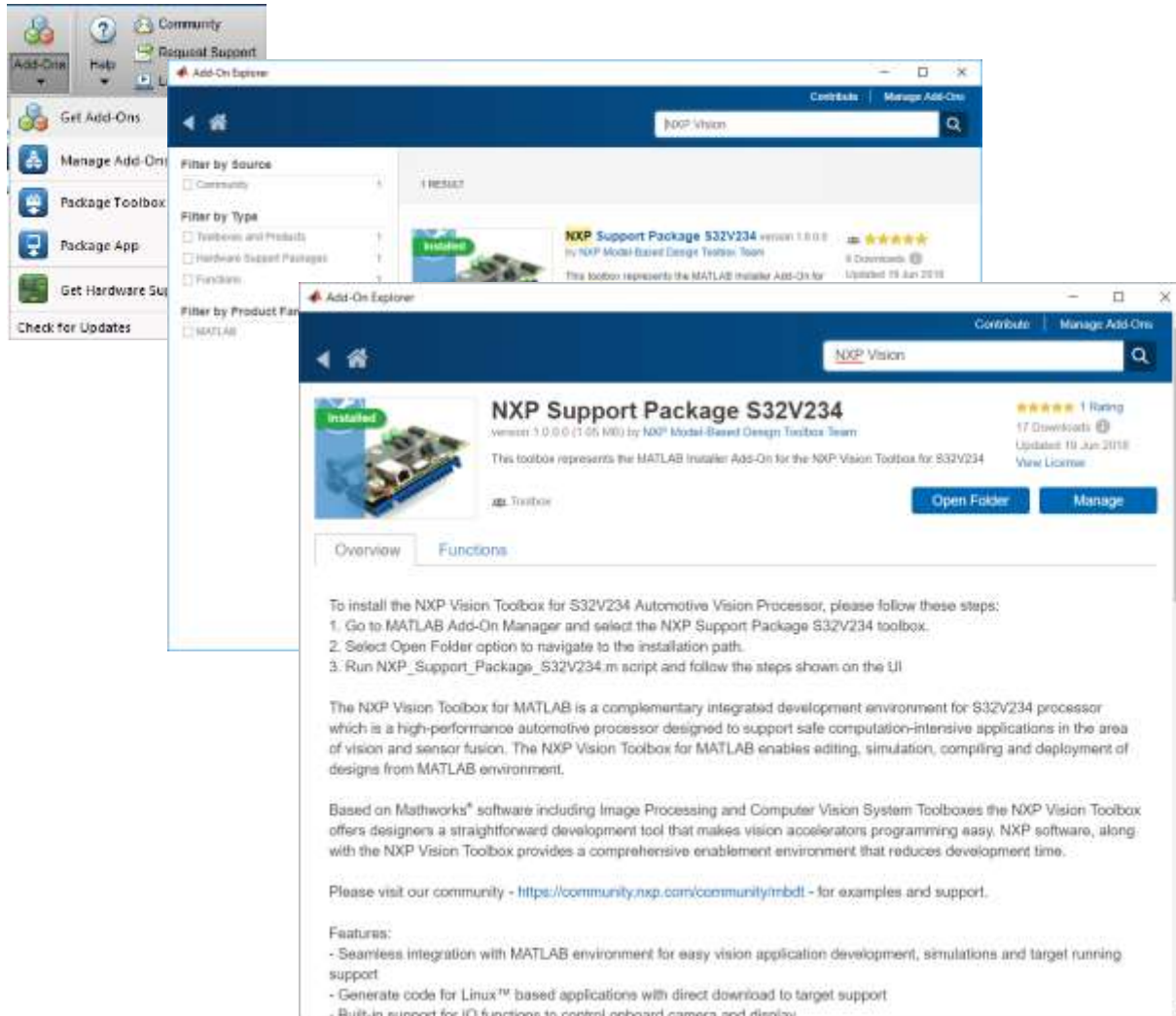
```
function laneDetection_camera() %#codegen
%% Initialize Objects
% Video Reader
width = uint32(1280);
height = uint32(720);
ratio = 1; % Resize ratio

if coder.target('MATLAB')
    input = nxpvt.videoinput('winvideo', 1, ...
    else
    input = nxpvt.videoinput('sony', 1, width, height, ratio);
end

%% Create Shape Inserter
hShapes = vision.ShapeInserter('Shape', 'Lines', ...
    'BorderColor', 'Custom', ...
    'LineWidth', 4, ...
```
- Workspace:** Lists variables in the workspace, including 'inImagePath', 'inImgMat', 'laneImageAlg', 'laneVerticesA', 'ratio', and 'resizeObj'.
- Command Window:** Displays instructions for using the toolbox, including a usage example:

```
config.MakeJobs = 0;
config.Optimize = true;
config.Deploy = true;
config.TargetIpAddress = '134.27.169.142';
config.DeployPath = '/examples/main_video';
config.RemoteFilename = 'main_videoreader.el';
config.ExtraFiles = {'./data/Megamind.avi', 'nxpvt_codegen/main_videoreader.m', config};
nxpvt_codegen('main_videoreader.m', config);
```
- Figures - Figure 2:** Shows a video frame with green lines overlaid on the road, representing the lane detection results.
- Documentation:** Provides a table of contents and a description of the 'nxpvt.apu.harris' function, which finds corners in the input data.

NXP Vision Toolbox – Installation



The screenshot shows the MATLAB Add-On Explorer interface. The search results for "NXP Vision" are displayed, with the "NXP Support Package S32V234" selected. The package details include the version (1.0.0.0), the developer (NXP Model-Based Design Toolbox Team), and the update date (19 Jun 2018). The package is marked as "Installed". Below the package details, there are instructions for installation and a list of features.

To install the NXP Vision Toolbox for S32V234 Automotive Vision Processor, please follow these steps:

1. Go to MATLAB Add-On Manager and select the NXP Support Package S32V234 toolbox.
2. Select Open Folder option to navigate to the installation path.
3. Run NXP_Support_Package_S32V234.m script and follow the steps shown on the UI

The NXP Vision Toolbox for MATLAB is a complementary integrated development environment for S32V234 processor which is a high-performance automotive processor designed to support safe computation-intensive applications in the area of vision and sensor fusion. The NXP Vision Toolbox for MATLAB enables editing, simulation, compiling and deployment of designs from MATLAB environment.

Based on Mathworks® software including Image Processing and Computer Vision System Toolboxes the NXP Vision Toolbox offers designers a straightforward development tool that makes vision accelerators programming easy. NXP software, along with the NXP Vision Toolbox provides a comprehensive enablement environment that reduces development time.

Please visit our community - <https://community.nxp.com/community/mbdt> - for examples and support.

Features:

- Seamless integration with MATLAB environment for easy vision application development, simulations and target running support
- Generate code for Linux™ based applications with direct download to target support
- Built-in support for IQ functions to control onboard camera and display



The screenshot shows the "NXP Vision Toolbox for S32V234: Installation Guide" window. It provides a step-by-step installation guide for the NXP Vision Toolbox for S32V234 Automotive Vision Processors. The guide includes a welcome message, a description of the toolbox, and a list of steps for installation.

Welcome to step-by-step installation guide for NXP Vision Toolbox for S32V234 Automotive Vision Processors

The NXP Vision Toolbox enables editing, simulation, compiling and deployment of designs from MATLAB environment in conjunction with NXP Vision SDK and NXP ARM/APEX Build Tools that needs to be installed separately.

Based on Mathworks® software including Image Processing® and Computer Vision System Toolboxes® the NXP Vision Toolbox offers designers a straightforward development tool that makes vision accelerators programming easy. NXP software, along with the NXP Vision Toolbox provides a comprehensive enablement environment that reduces development time.

Note: A valid NXP account is needed to access to IPUs Download and License generation pages. Use <https://www.nxp.com/webapp/signup/register> to create a new account. The NXP account gives access to the content on the NXP's Model-Based Design Community <https://community.nxp.com/community/mbdt>.

Sign up now!

Download, Install and Activate NXP Vision Toolbox

Step 1: Download the NXP Vision Toolbox from NXP website and install it as Add-On using MATLAB installer

Go To NXP Download Site

Install MLTBX File as Add-On

Verify Vision Toolbox Installation

Step 2: Generate a free-of-cost license from NXP website, download and install into toolbox License folder

Generate License File

Activate NXP Vision Toolbox

Verify Vision Toolbox License

Download & Install NXP Vision SDK and Build Tools for A53/APU

Step 3: Download and install NXP Vision SDK software package

Go To VSDK Download Site

Install VSDK and A53/APU Compilers

Step 4: Wait until the SDK installation is completed and then create S32V234_SDK_ROOT and APU_TOOLS system variables.

```
ie: S32V234_SDK_ROOT = [VSDK_local_dir]\S32V234_sdk
ie: APU_TOOLS = [Compiler_local_dir]\APU_compiler_v1.0
```

(Optional) Download SD Card Pre-built Image

Step 5: Download a pre-built SD Card Image for S32V Evaluation Board. Check NXP Vision Toolbox Quick Start Manual instructions to learn how to deploy the pre-built image on your own SD Card and boot up the S32V platform directly from MATLAB

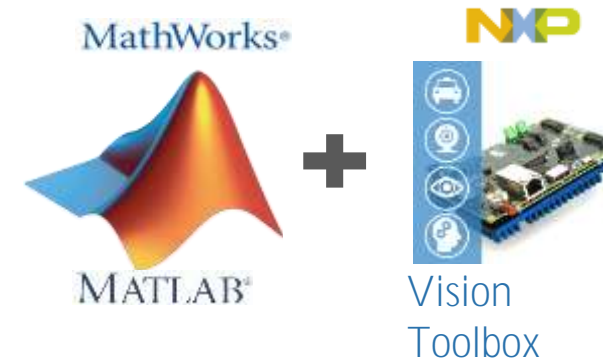
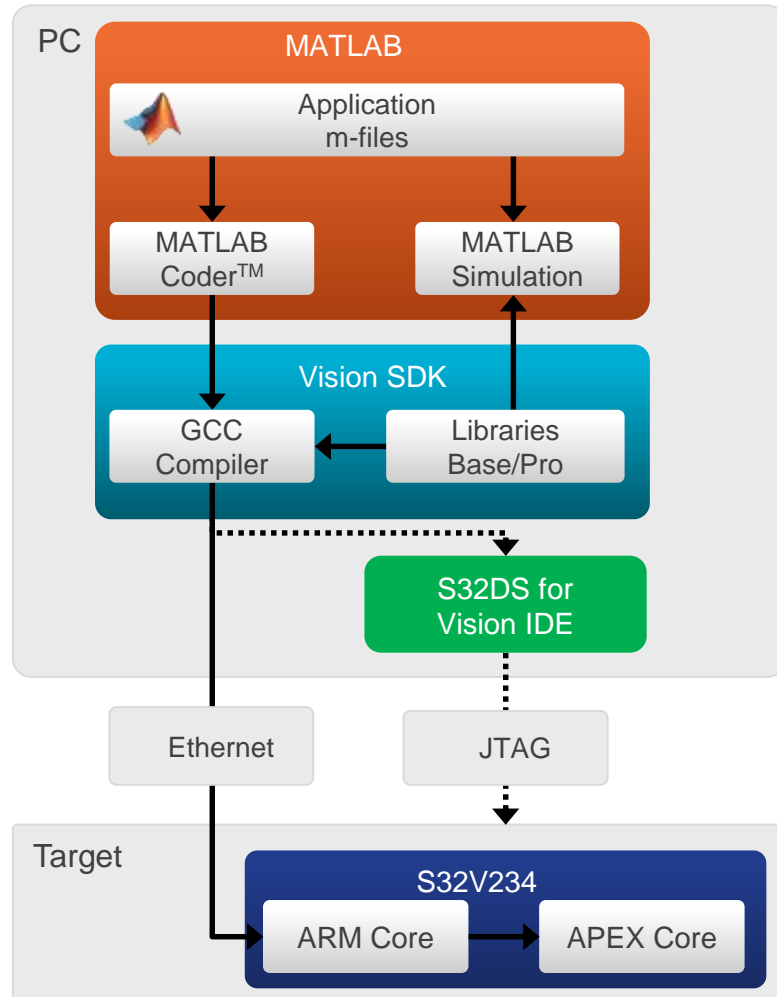
Download SD Card Image

Go To Support Site

Go To NXP Vision Toolbox Site

Close

NXP Vision Toolbox – Software Development Flow



- Single MATLAB Environment for complete development, simulation, build & running on the NXP targets
- Support for I/O functions (cameras & displays)
- Automatic deploying & running on S32V234EVB and S32V234SBC
- Ready to use examples for [Computer Vision Toolbox applications](#)

Objects & Features Detection Demo on S32V



Object Detection – How To

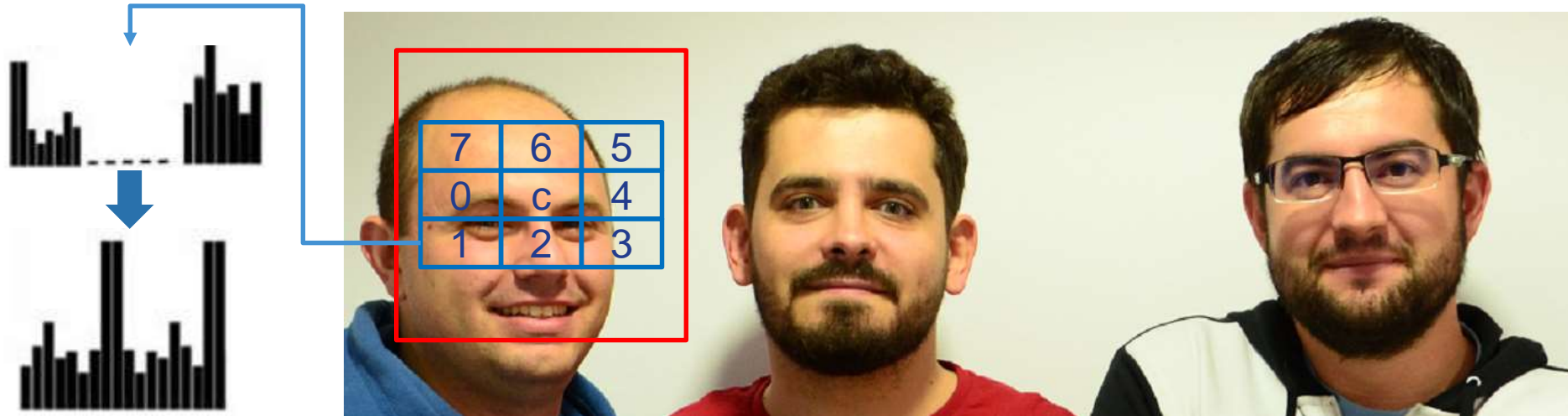
- Cascade Detectors: Local Binary Pattern (LBP) or Haar



- The object detector has an associated **sliding window** that is sliding from left to right and top to bottom to all possible positions
- The sliding window is also scaled between minimum object to maximum object, forming the **window pyramid**
- All possible detections in a specific range are merged into an object and if their number is larger than the **merge threshold** then we have a detection

Local Binary Pattern – Object Detector

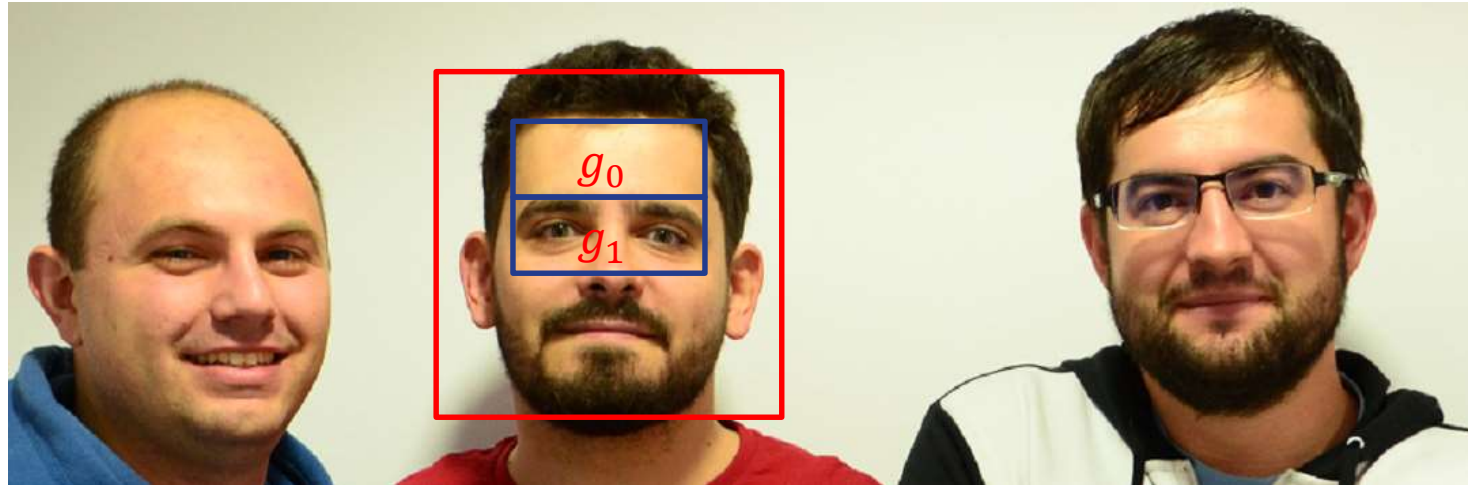
- LBP feature = rectangle divided in 3 columns and 3 rows resulting 9 smaller rectangles of equal size



- The values of each smaller rectangle are added resulting a 3 by 3 matrix M. The centre value, $M_{2,2}$, is compared with the other values, $M_{i,j}$ ($i,j \neq (2,2)$)
- If the centre value is greater than the neighbour's value then 0 is written, otherwise a 1 is written, resulting a 8-bit number
- The 8-bit number is mapped to value which is the feature value of the input image

Haar – Feature Detector

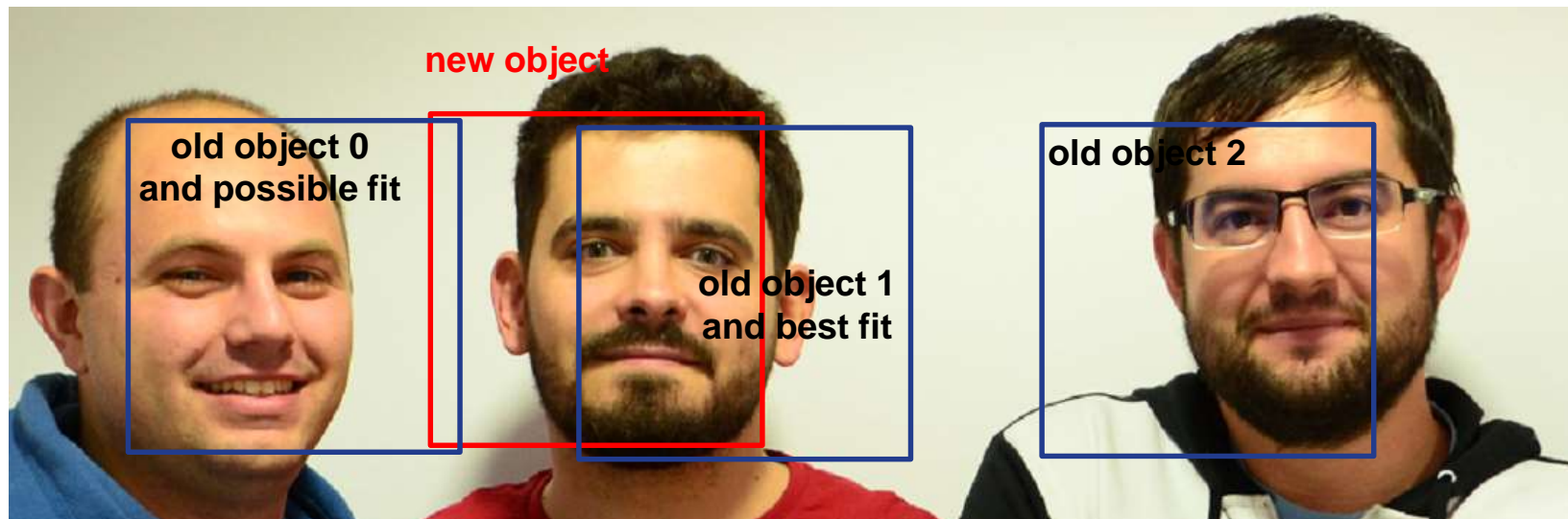
- Haar-like feature is made of 2 or 3 rectangles each one with an associated weight, that satisfies $\sum_{i=0}^{k-1} g_i w_i h_i = 0$, where k is the number of rectangles, and g_i is the weight, w_i is the width, h_i is the height of the rectangle i



- The feature sum is $\sum_{i=0}^{k-1} g_i \sum_{r=y_i}^{y_i+w_i-1} \sum_{c=x_i}^{x_i+h_i-1} I(r,c) / w \cdot h$, where I is the input image, w is the input image width, h is the input image height
- By comparing the feature sum with a threshold we select one of the two values which is the feature value of the input image

Kalman Filter

- The filter has as input the list with the new detections, the old list of filtered objects, and the time elapsed from the last call, and as output the new list of filtered objects
- The best fit is the old object with the closes center from the new object center
- If there are not any old objects that overlap then an object with no history is created
- In the image below the blue rectangles are the old object and the red rectangle is the new one



Kalman filter (cont'd)

- Uses a model and observations to make more precise estimations and it is supposed that both the model and observations are affected by noise
- The noise information is passed to Kalman filter through two matrices

$$Q_k = E[W_k W_k^T] = \begin{pmatrix} \frac{1}{4}\Delta t^4 \sigma_{qx}^2 & \frac{1}{2}\Delta t^3 \sigma_{qx}^2 & 0 & 0 & 0 & 0 \\ \frac{1}{2}\Delta t^3 \sigma_{qx}^2 & \Delta t^2 \sigma_{qx}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4}\Delta t^4 \sigma_{qy}^2 & \frac{1}{2}\Delta t^3 \sigma_{qy}^2 & 0 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^3 \sigma_{qy}^2 & \Delta t^2 \sigma_{qy}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{qw}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{qh}^2 \end{pmatrix}$$

$$R_k = E[V_k V_k^T] = \begin{pmatrix} \sigma_{rx}^2 & 0 & 0 & 0 \\ 0 & \sigma_{ry}^2 & 0 & 0 \\ 0 & 0 & \sigma_{rw}^2 & 0 \\ 0 & 0 & 0 & \sigma_{rh}^2 \end{pmatrix}$$

- The variances were found using empirical research

Object & Feature Detection with S32V

```
features_detection_s32v234_camera_roi_kf_main.m x +
1 % Connect with S32V board and Create a camera object
2 s32Obj = nxpvt.s32v234('134.27.168.249');
3 input = nxpvt.cameraboard(s32Obj, 1, 'Resolution', '720x1280');
4
5 face_detector = nxpvt.CascadeObjectDetector('data/lbpcascade_frontalface.xml', ...
6     'ScaleFactor',1.1, 'MinSize',[150 -1], 'MaxSize',[400 -1], 'SkipOdd',1, 'MergeThreshold',4);
7 eyepairb_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_eyepair_big.xml',...
8     'ScaleFactor',1.1, 'MinSize',[150 -1], 'MaxSize',[-1 -1], 'SkipOdd',1, 'MergeThreshold',1);
9
10 coder.varsize('filteredObjects');
11 filteredObjects = {FilteredObject([0 0 0 0])};
12
13 while true
14     frame = input.getsnapshot();
15
16     % Get faces
17     [face_bbox, ~] = step(face_detector, frame);
18     [face_bbox_kf, filteredObjects] = filter_objects(face_bbox, filteredObjects, T);
19
20     % Get features
21     eyepairb_bbox = loop1_eyes(face_bbox_kf, frame, eyepairb_detector);
22
23     % Display results
24     nxpvt.cv.rectangle(frame, face_bbox, [255, 0, 0], 2);
25     nxpvt.cv.rectangle(frame, eyepairb_bbox, [0, 0, 255], 2);
26     nxpvt.imshow(frame);
27 end
```

STEP 1: Create an object to connect with the NXP S32V from MATLAB.

STEP 2: Initialize the LPB & Haar Detectors

STEP 3: Initialize the Kalman Filter

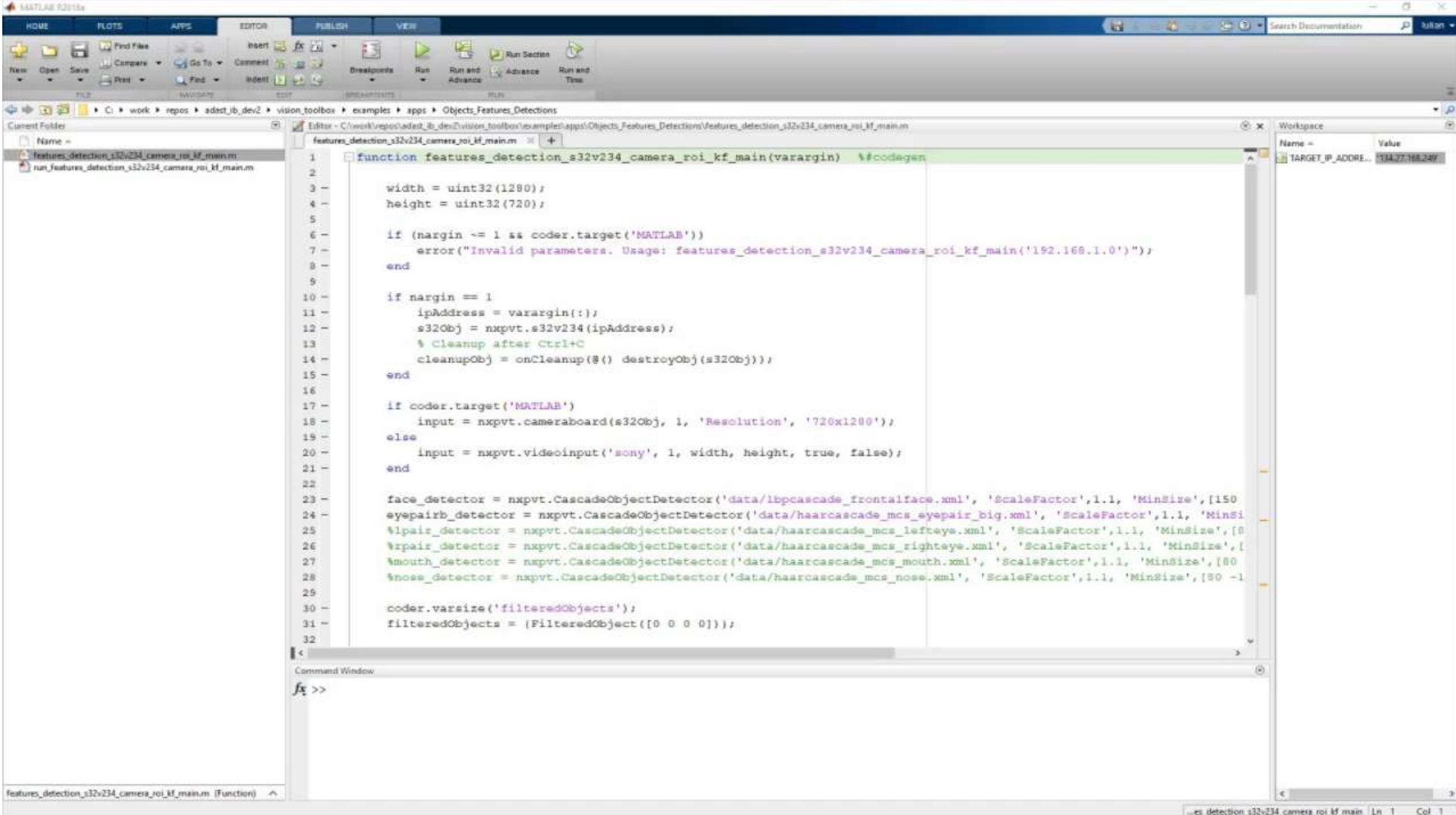
STEP 4: Get video frames from the S32V SonyCamera

STEP 5: Run Object Detector and filter out the detections using Kalman filter

STEP 6: Run Feature Detector only within the filtered ROI

STEP 7: Display prediction results and capture frames on the screen for validation

Object & Feature Detection – Demo on S32V HW



The image shows the MATLAB R2016a IDE interface. The main editor window displays a function named `features_detection_s32v234_camera_roi_kf_main`. The function code is as follows:

```
1 function features_detection_s32v234_camera_roi_kf_main(varargin) %codegen
2
3     width = uint32(1280);
4     height = uint32(720);
5
6     if (nargin <= 1 && coder.target('MATLAB'))
7         error("Invalid parameters. Usage: features_detection_s32v234_camera_roi_kf_main('192.168.1.0')");
8     end
9
10    if nargin == 1
11        ipAddress = varargin{:};
12        s32Obj = nxpvt.s32v234(ipAddress);
13        % Cleanup after Ctrl+C
14        cleanupObj = onCleanup(@() destroyObj(s32Obj));
15    end
16
17    if coder.target('MATLAB')
18        input = nxpvt.cameraboard(s32Obj, 1, 'Resolution', '720x1280');
19    else
20        input = nxpvt.videoinput('sony', 1, width, height, true, false);
21    end
22
23    face_detector = nxpvt.CascadeObjectDetector('data/lbpcascade_frontalface.xml', 'ScaleFactor',1.1, 'MinSize',[150
24    eyepairb_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_eyepair_big.xml', 'ScaleFactor',1.1, 'MinSi
25    %lpair_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_lefteye.xml', 'ScaleFactor',1.1, 'MinSize',[0
26    %rpair_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_righteye.xml', 'ScaleFactor',1.1, 'MinSize',[
27    %mouth_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_mouth.xml', 'ScaleFactor',1.1, 'MinSize',[80
28    %nose_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_nose.xml', 'ScaleFactor',1.1, 'MinSize',[80 -1
29
30    coder.varsize('filteredObjects');
31    filteredObjects = (FilteredObject([0 0 0]));
32
```

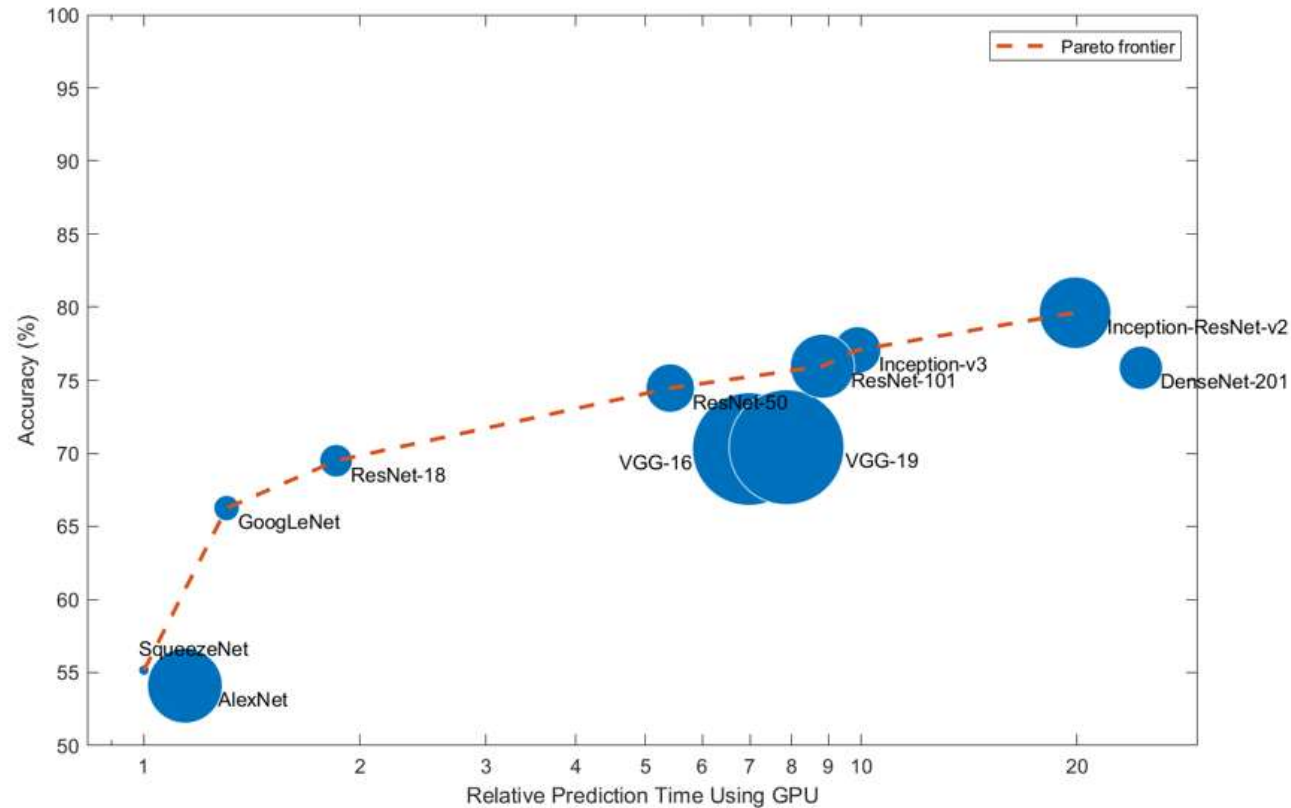
The Command Window at the bottom shows the prompt `>>`. The Workspace window on the right shows a variable `TARGET_IP_ADDR..` with the value `'192.168.1.0'`.

CNN Demo on S32V



CNN Selection

[Deep Learning Toolbox](#) in MATLAB provides a set a Pretrained networks to speedup the SW development

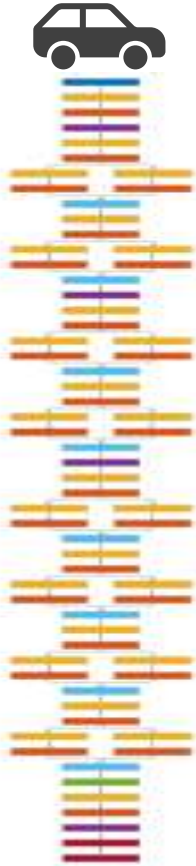


Network	Depth	Size MB	Params. (Millions)	Img. Size
alexnet	8	227	61.0	227x227
vgg16	16	515	138	224x224
vgg19	19	535	144	224x224
squeezenet	18	4.6	1.24	227x227
googlenet	22	27	7.0	224x224
inceptionv3	48	89	23.9	299x299
densenet201	201	77	20.0	224x224
resnet18	18	44	11.7	224x224
resnet50	50	96	25.6	224x224
resnet101	101	167	44.6	224x224
inceptionresnetv2	164	209	55.9	299x299

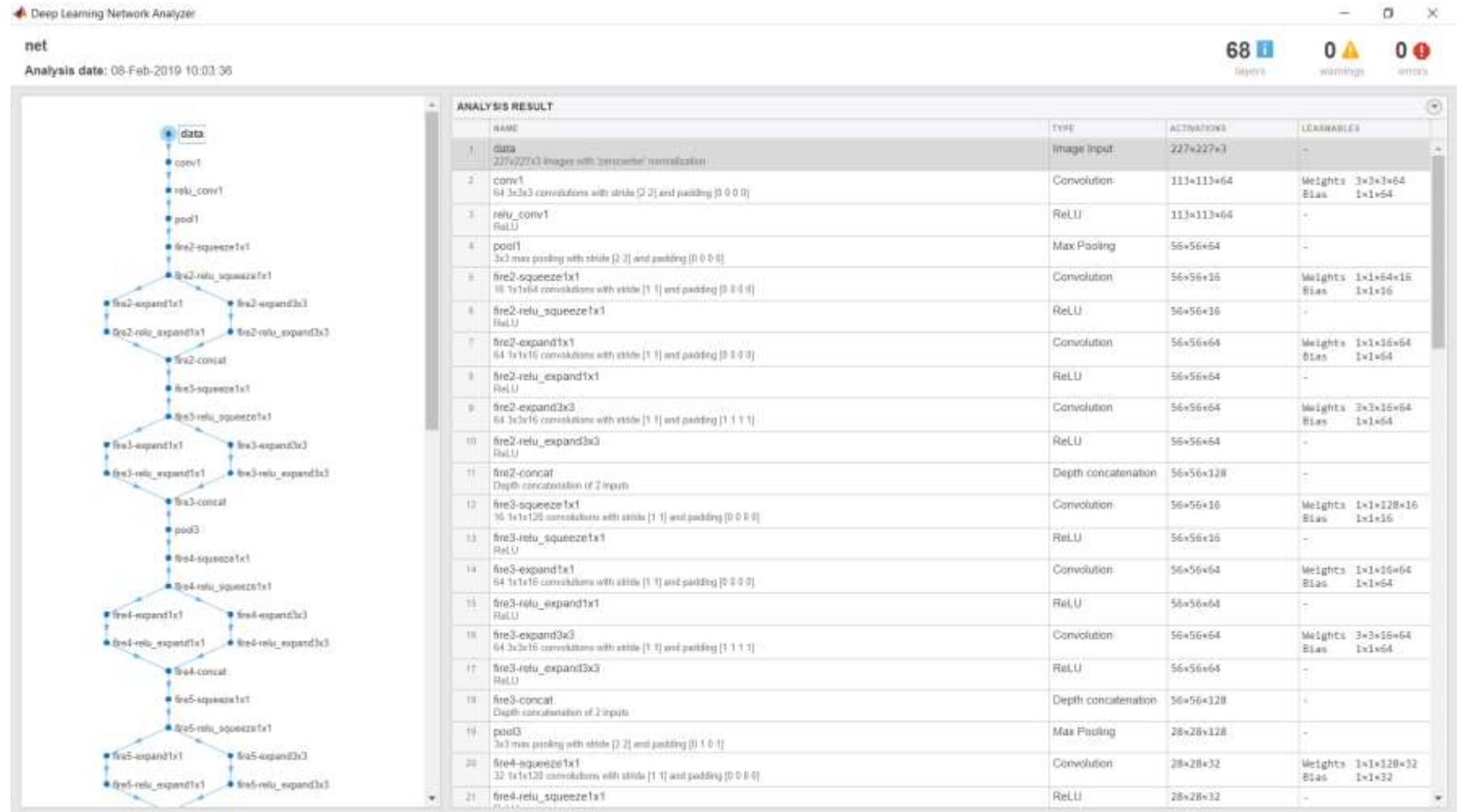
Source: [MathWorks Website](#)

SqueezeNet Model Used for Demo

Input image



Output prediction: CAR



CNN Object Prediction with S32V

```
TestMySqueezeNet.m x +
1 % Connect with S32V board and Create a camera object
2 s32 = nxpvt.s32v234('134.27.168.249', 123);
3 cam = nxpvt.cameraboard(s32, 1, 'Resolution', '720x1280');
4
5 % Process frames from S32V SonyCam and Predict
6 while (1)
7
8 % Get video frames from S32V SonyCam
9 img = cam.snapshot().data;
10
11 % Resize image to fit SqueezeNet network
12 [origwidth, origheight, ~] = size(img);
13 img = imresize(img, [227,227]);
14
15 % Predict with the SqueezeNet network
16 [proc, classIdx] = mySqueezeNet(img);
17
18 % Display results
19 load classNames
20 img = imresize(img, [origwidth, origheight]);
21 img = insertText(img, [10, 10], [classNames{classIdx(1)} ' ' ...
22 num2str(proc(1) * 100) '%'], ...
23 'BoxOpacity', 0, 'TextColor', 'green');
24 nxpvt.imshow(img);
25 end
```

STEP 1: Create an object to connect with the NXP S32V from MATLAB.

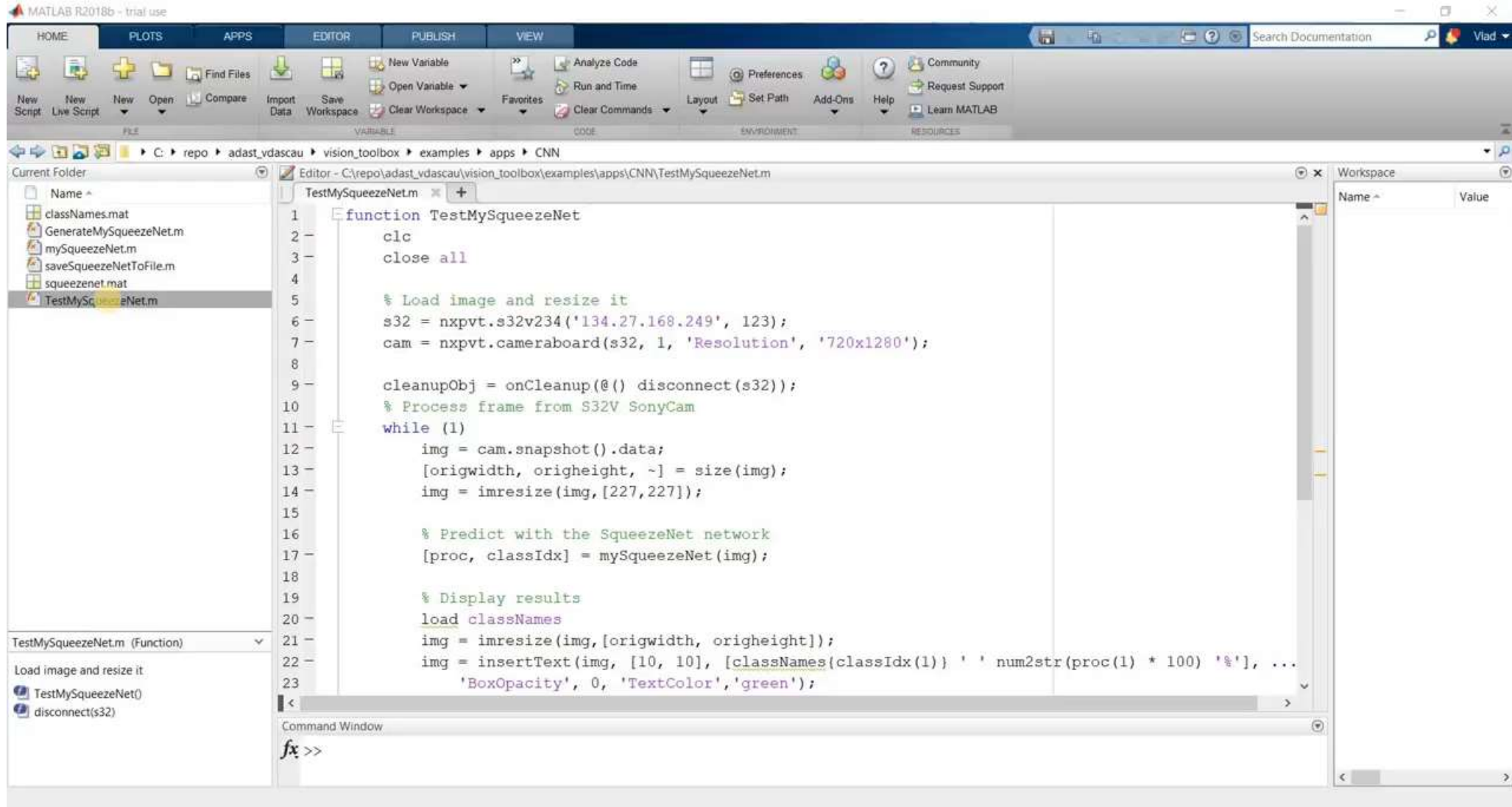
STEP 2: Read video frames in real-time from S32V onboard ISP SonyCam

STEP 3: Run the SqueezeNet CNN to detect the objects within the video frames captured by S32V ISP

STEP 4: Display prediction results and capture frames on the screen for validation

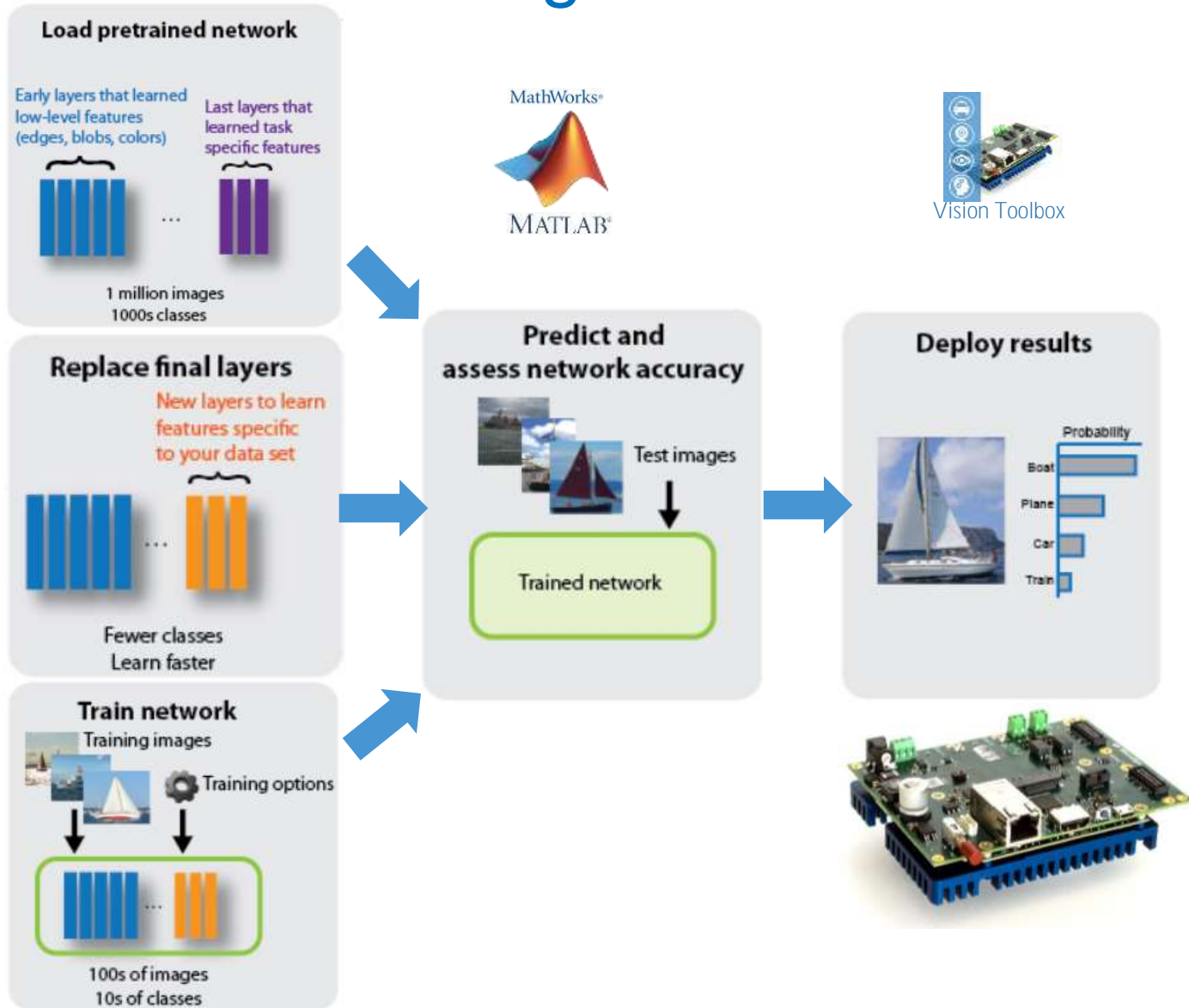


SqueezeNet CNN – Demo (Simulation & S32V HW)

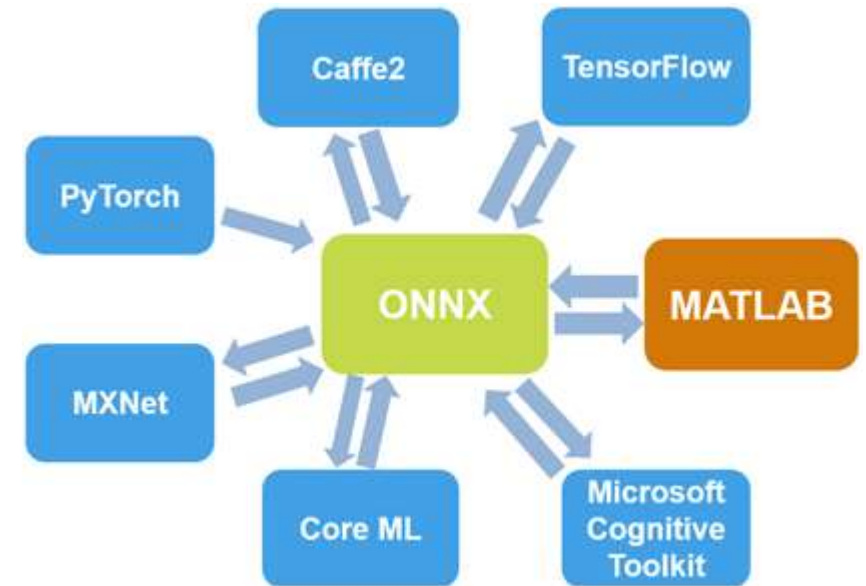


```
function TestMySqueezeNet
1
2     clc
3     close all
4
5     % Load image and resize it
6     s32 = nxpvt.s32v234('134.27.168.249', 123);
7     cam = nxpvt.cameraboard(s32, 1, 'Resolution', '720x1280');
8
9     cleanupObj = onCleanup(@() disconnect(s32));
10    % Process frame from S32V SonyCam
11    while (1)
12        img = cam.snapshot().data;
13        [origwidth, origheight, ~] = size(img);
14        img = imresize(img, [227, 227]);
15
16        % Predict with the SqueezeNet network
17        [proc, classIdx] = mySqueezeNet(img);
18
19        % Display results
20        load classNames
21        img = imresize(img, [origwidth, origheight]);
22        img = insertText(img, [10, 10], [classNames(classIdx(1)) ' ' num2str(proc(1) * 100) '%'], ...
23            'BoxOpacity', 0, 'TextColor', 'green');
```

CNN Fine Tuning & Reuse



Deep Learning Framework Interoperability



Source of Images: [MathWorks Website](https://www.mathworks.com)

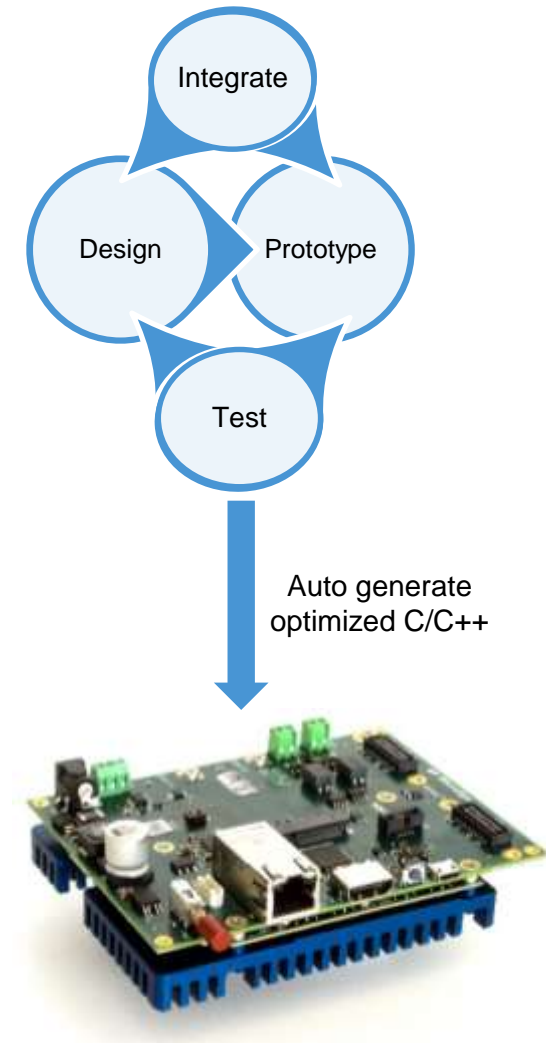
Conclusions



Achieve Efficient Development Workflow

Using MATLAB and various toolboxes, design engineers can:

- Maintained one design in MATLAB
- Design faster and get to C quickly
- Test more systematic and frequently
- Focus on algorithm improvements



Getting Help

[Vision Toolbox Online Community](#)
[Examples & Help](#)

[Vision Toolbox home page](#)
www.nxp.com/visiontoolbox

The screenshot shows the NXP Vision Toolbox Online Community website. At the top, there is a navigation bar with 'Home', 'Content', 'People', 'Subspaces and Projects', and 'Calendar'. Below this is a search bar with the text 'Search or Ask a Question in VISION'. The main content area features a 'Vision Toolbox for MATLAB News' section with several news items, including 'NXP Vision Toolbox for S32V234 2018.R1.EM1 - Product Release Announcement'. To the right, there is a 'Top Participants' list with names like 'dimitri.daniel.popa', 'sara woo', and 'Paul Vlase'.

The screenshot shows the Vision Toolbox for MATLAB home page. At the top, there is a navigation bar with 'OVERVIEW', 'DOCUMENTATION', 'DOWNLOADS', and 'DEVELOPMENT TOOLS'. Below this is a 'Jump To' section with links for 'Overview & Features', 'Supported Devices', 'Target Applications', and 'System Requirements'. The main content area features an 'Overview' section with a description of the NXP Vision Toolbox for MATLAB, followed by a 'Features' section with a list of features. Below the features, there are 'User Guide' and 'Download' buttons. At the bottom, there is a diagram showing the NXP Vision Toolbox for MATLAB architecture, which includes 'PC/MATLAB', 'Application m-files', 'MATLAB Coder', and 'MATLAB Simulation'.



**SECURE CONNECTIONS
FOR A SMARTER WORLD**