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

Patrick Menter

NXP Automotive Microcontroller & Processors

June 2019 | Session #AMF-AUT-T3653

SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

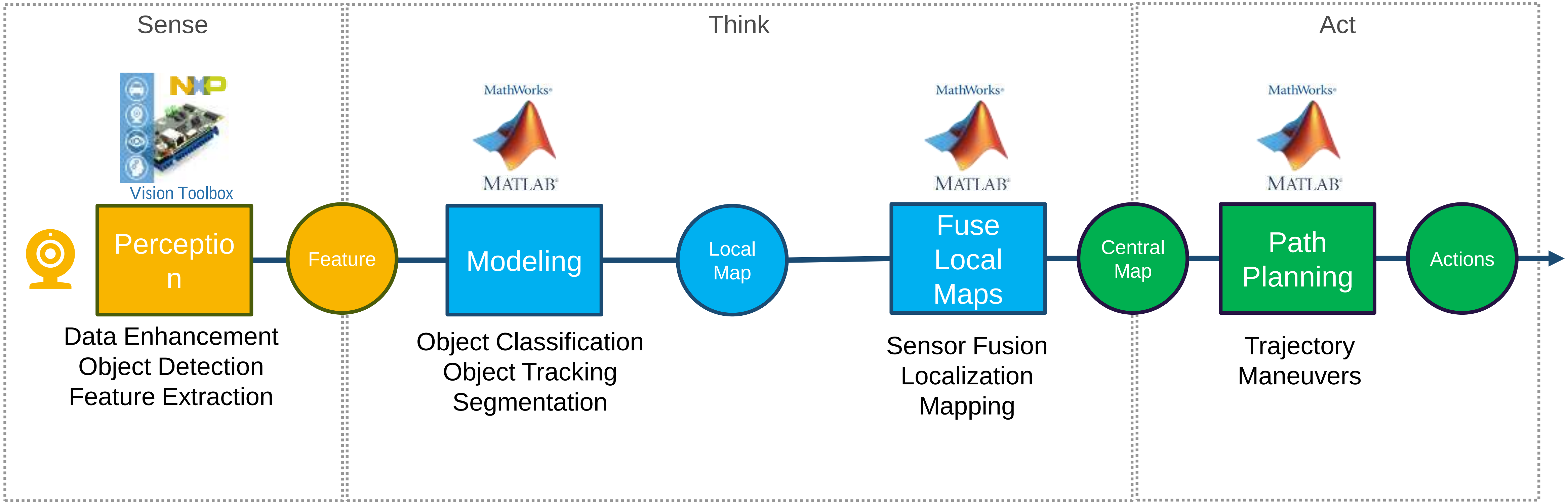
- NXP Vision Toolbox – Introduction
- Embedded System Development Process with MATLAB for S32V
- Object & Feature Detection Demo
- CNN SqueezeNet Demo

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

Perception

- Image Processing

  Processing, 3D workflow, Analysis

Modeling

- Computer Vision

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

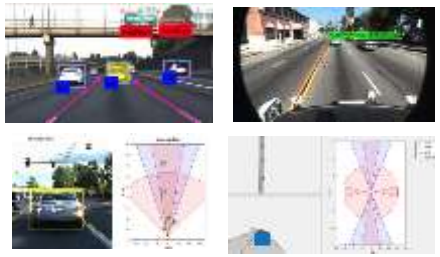
- Deep Learning

  Design, Train, Transfer, Pretrained, Framework interop

Fusion

- Advanced Driving System

  Labeling, Sensor Fusion & Tracking, Classification, Scenario Generation

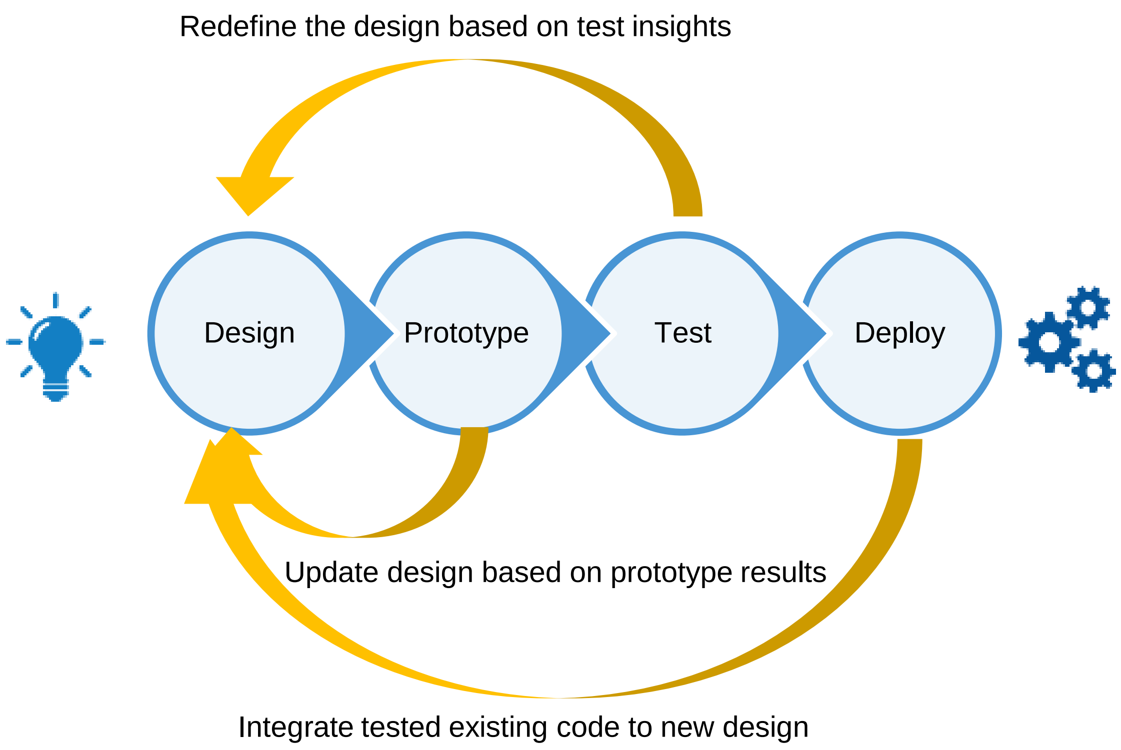


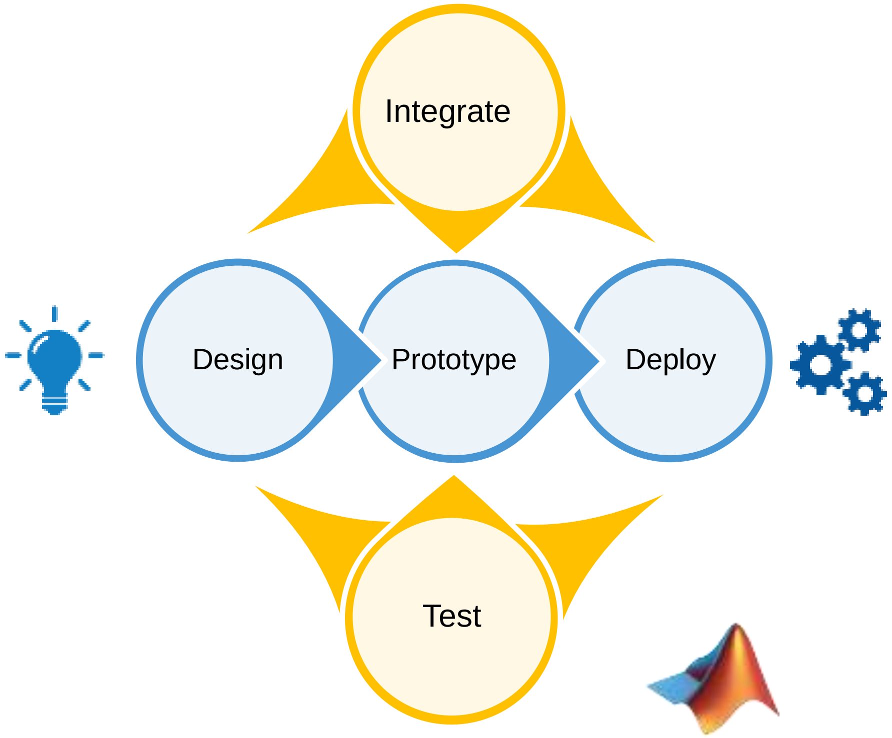
Acting

- Robotic System

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

Traditional vs. Model-Based Design Dev. Process

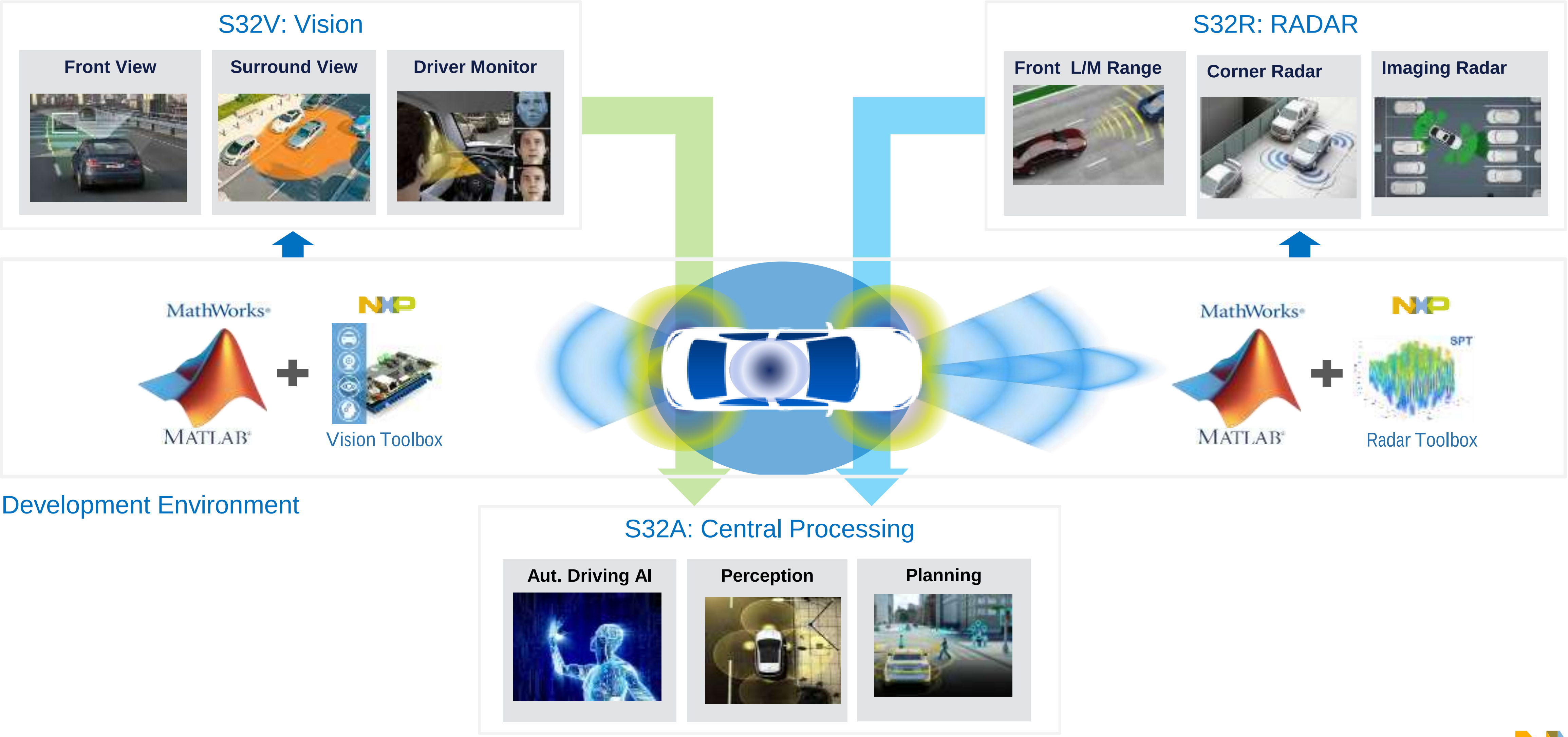




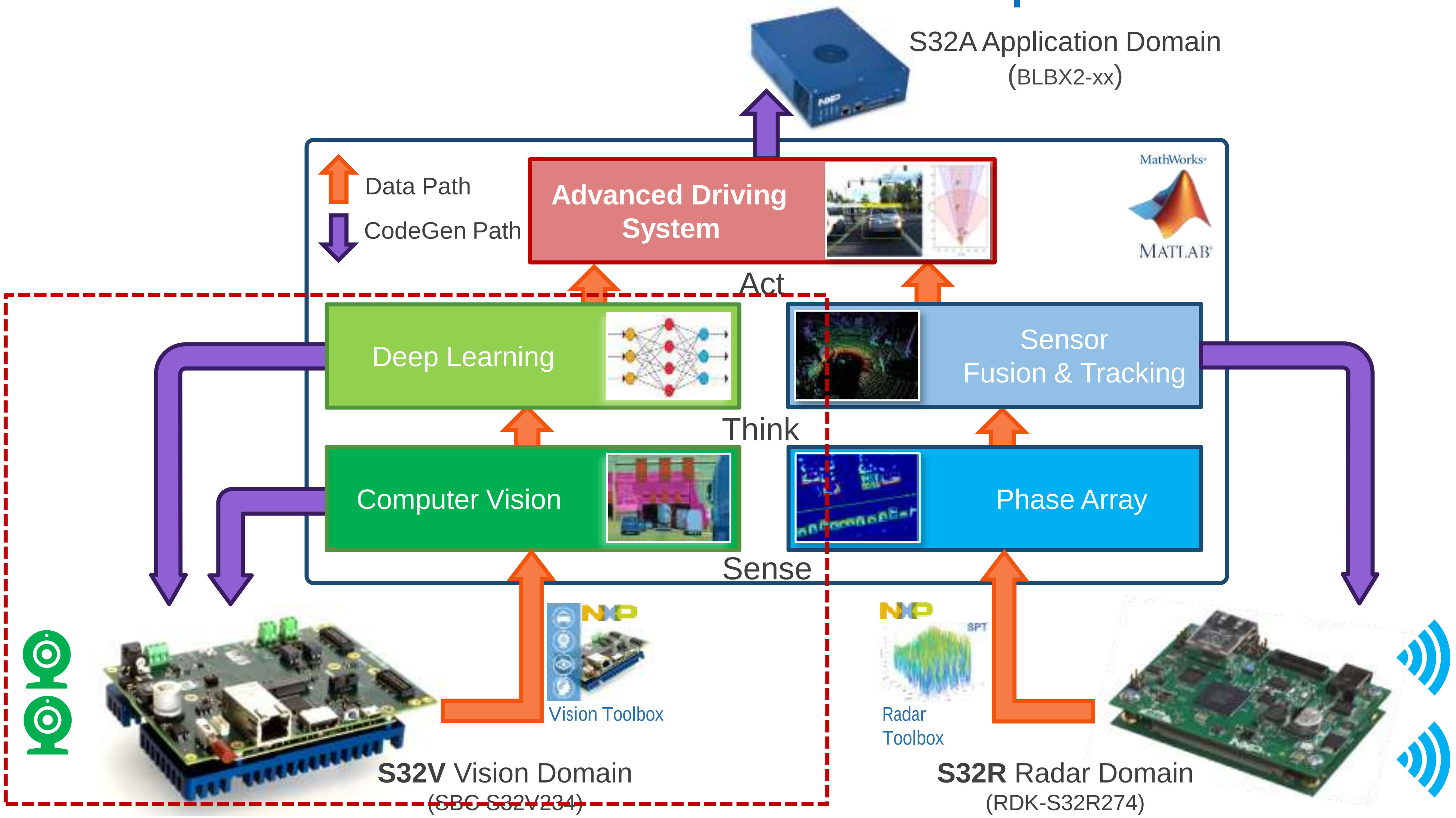
Embedded System Development in MATLAB Environment for S32V

Simulate, Test, Build and Deploy

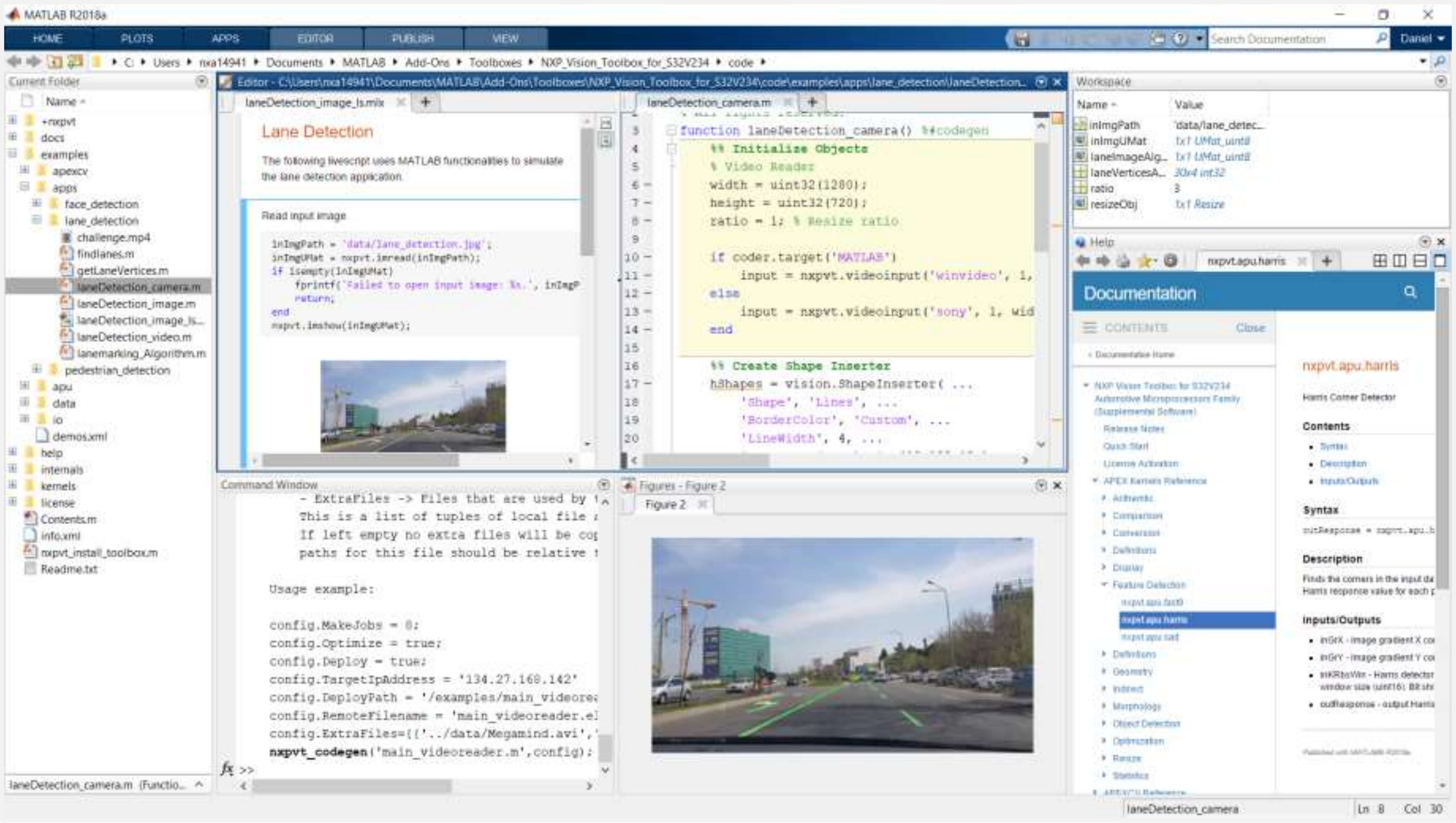
NXP HW & SW Solutions for ADAS



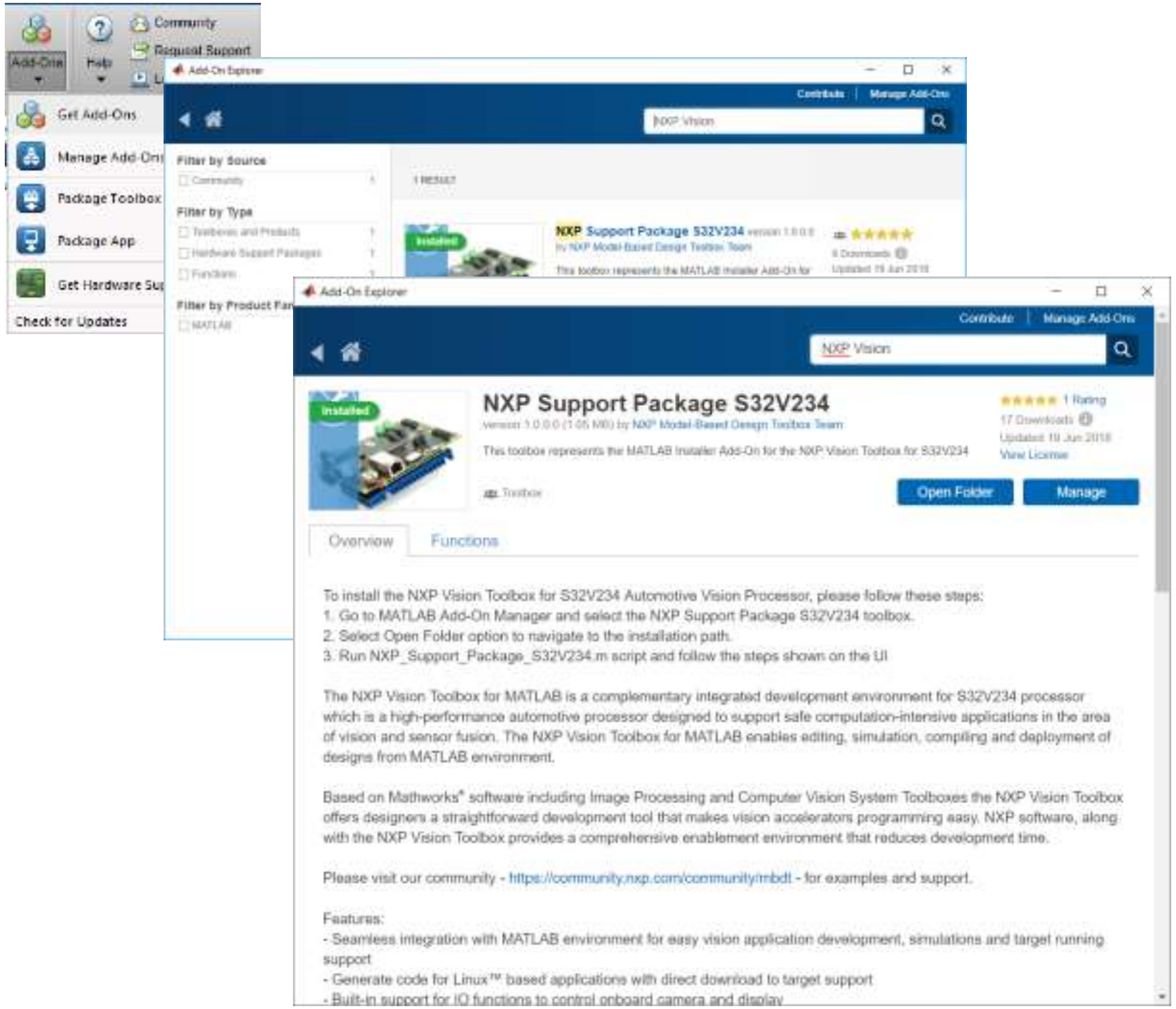
ADAS Solution with MATLAB – Concept

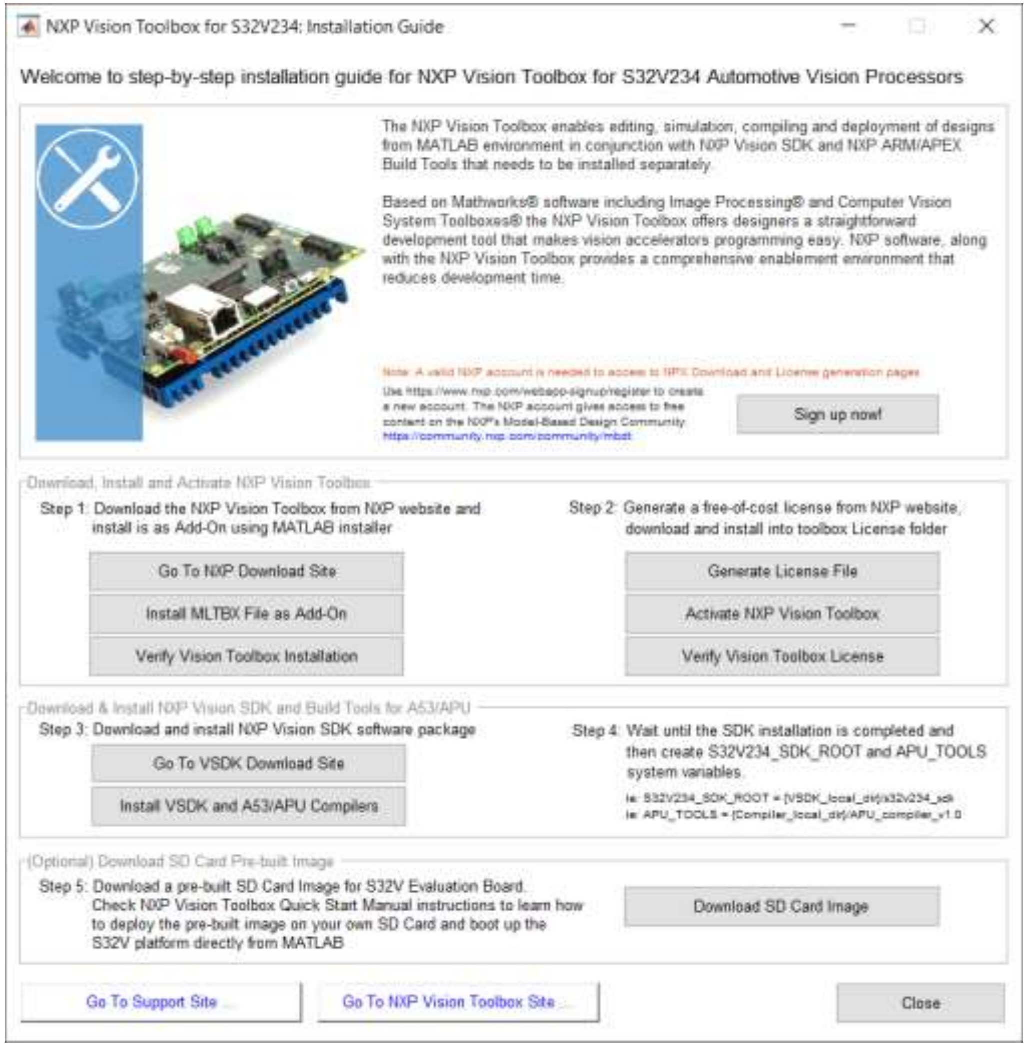


NXP Vision Toolbox – Preview

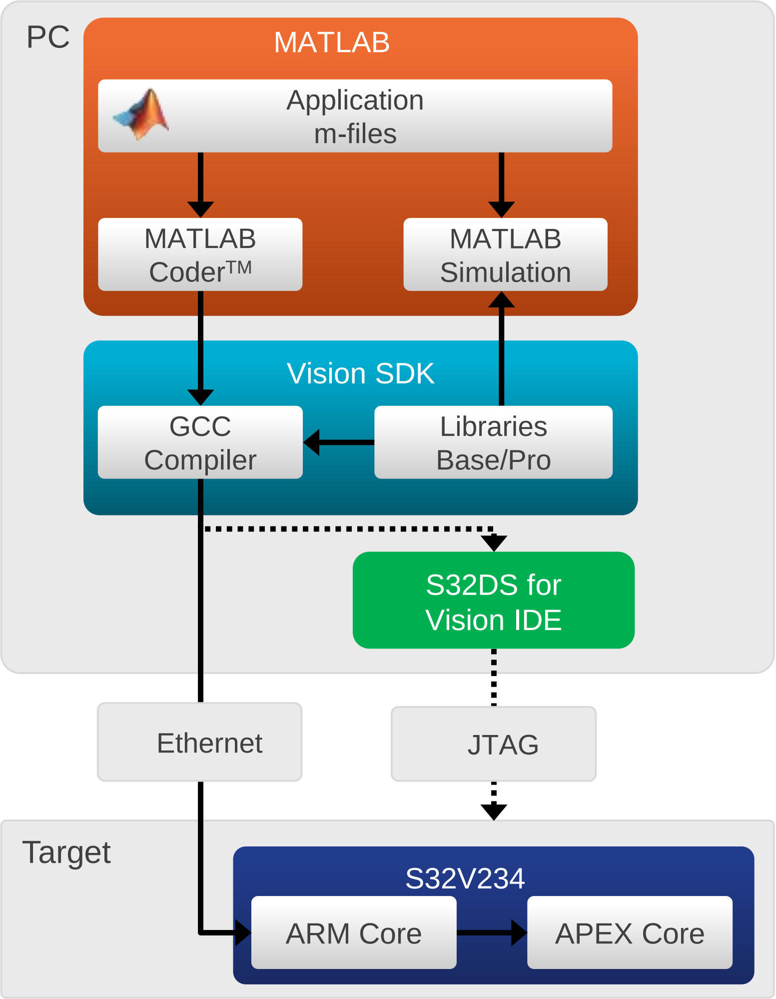


NXP Vision Toolbox – Installation





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

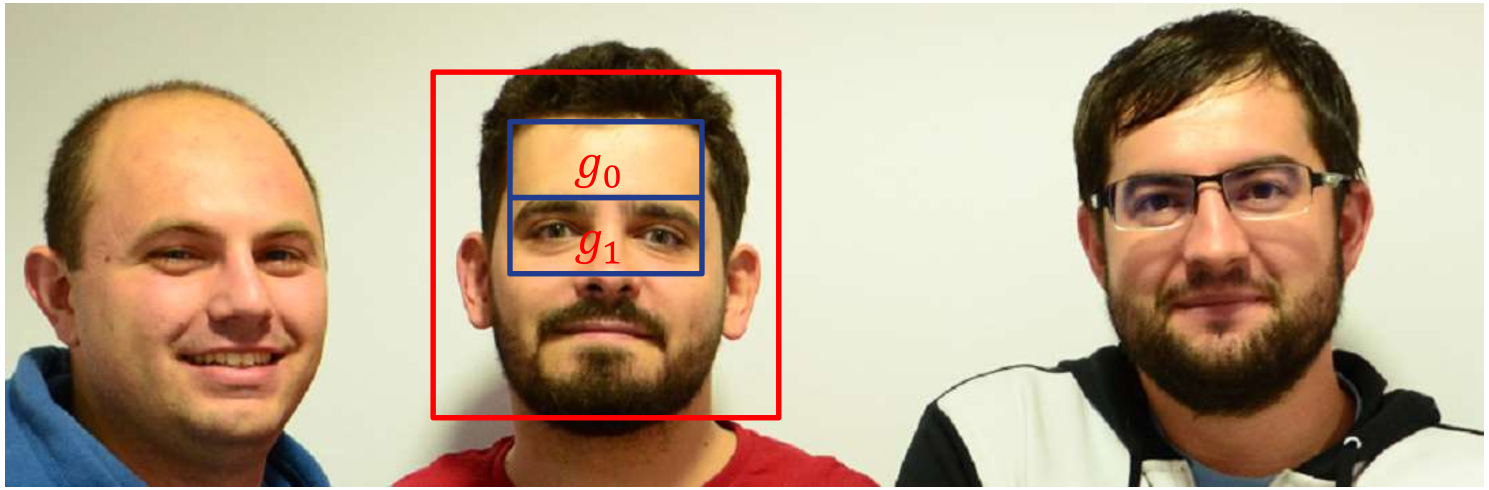
Local Binary Pattern – Object Detector

- 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, $M_{i,j} \quad (i,j) \neq (2,2)$
- If the centre value is grater 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

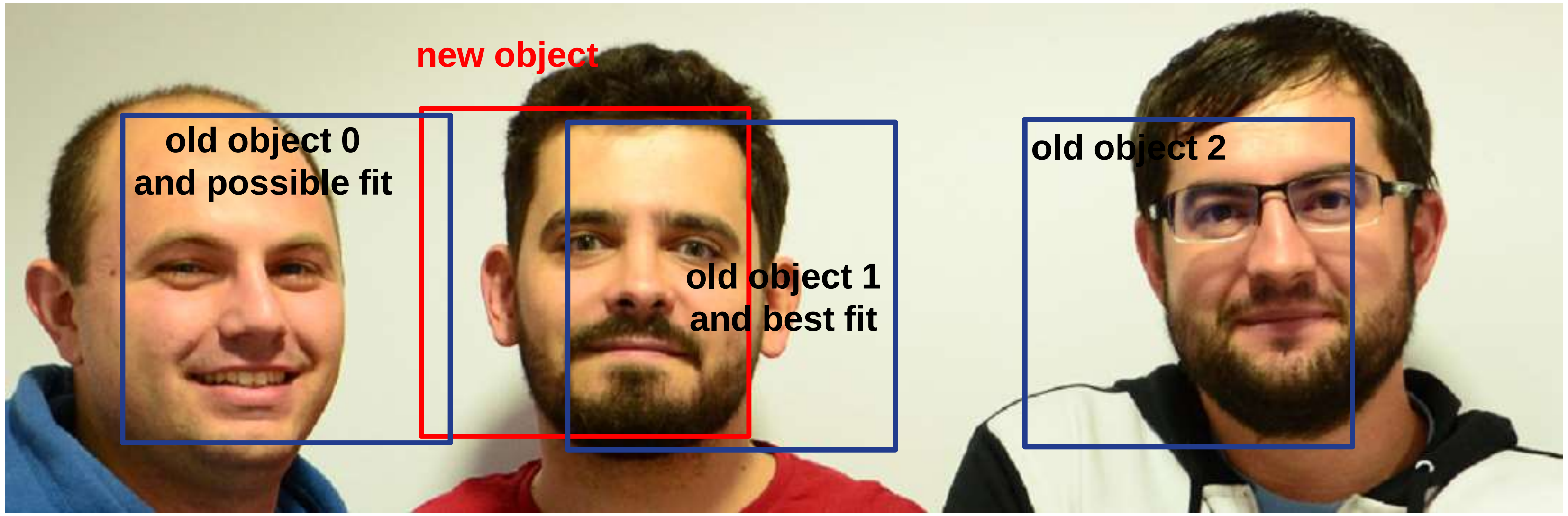
- Haar-like feature is made of 2 or 3 rectangles each one with an associated weight, that satisfies $\sum_{i=0}^{k-1} g_i w_i h_i = 0$, where k is the number of rectangles, and g_i is the weight, w_i is the width, h_i is the height of the rectangle i



- The future 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 compering 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

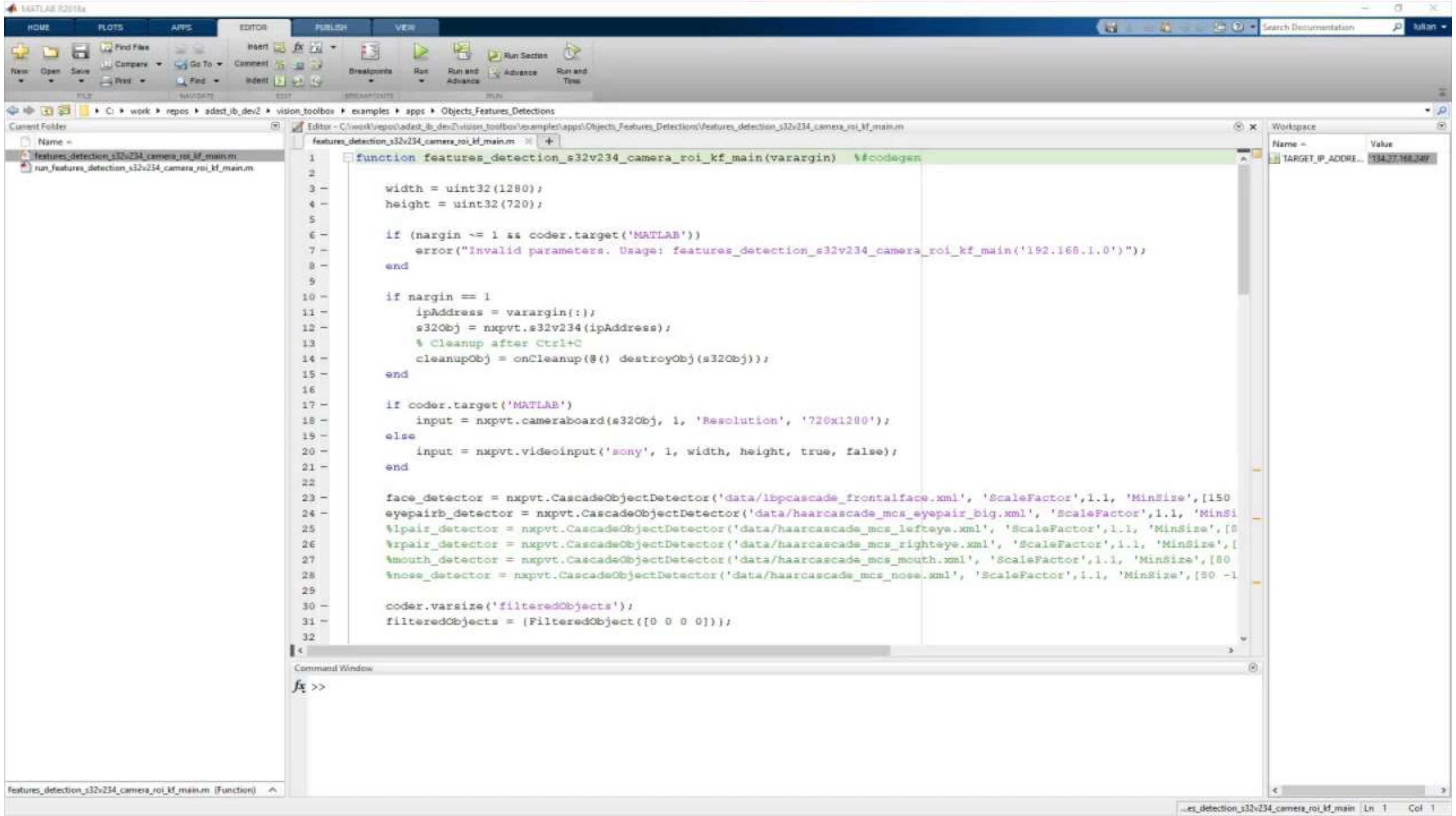
Object & Feature Detection with S32V

features_detection_s32v234_camera_roi_kf_main.m

```matlab
% Connect with S32V board and Create a camera object
s32Obj = nxpvt.s32v234('134.27.168.249');
input = nxpvt.cameraboard(s32Obj, 1, 'Resolution', '720x1280');

face_detector = nxpvt.CascadeObjectDetector('data/lbpcascade_frontalface.xml', ...
    'ScaleFactor',1.1, 'MinSize',[150 -1], 'MaxSize',[400 -1], 'SkipOdd',1, 'MergeThreshold',4);
eyepairb_detector = nxpvt.CascadeObjectDetector('data/haarcascade_mcs_eyepair_big.xml',...
    'ScaleFactor',1.1, 'MinSize',[150 -1], 'MaxSize',[-1 -1], 'SkipOdd',1, 'MergeThreshold',1);

coder.varsize('filteredObjects');
filteredObjects = {FilteredObject([0 0 0 0])};

while true
    frame = input.getsnapshot();

    % Get faces
    [face_bbox, ~] = step(face_detector, frame);
    [face_bbox_kf, filteredObjects] = filter_objects(face_bbox, filteredObjects, T);

    % Get features
    eyepairb_bbox = loop1_eyes(face_bbox_kf, frame, eyepairb_detector);

    % Display results
    nxpvt.cv.rectangle(frame, face_bbox, [255, 0, 0], 2);
    nxpvt.cv.rectangle(frame, eyepairb_bbox, [0, 0, 255], 2);
    nxpvt.imshow(frame);
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

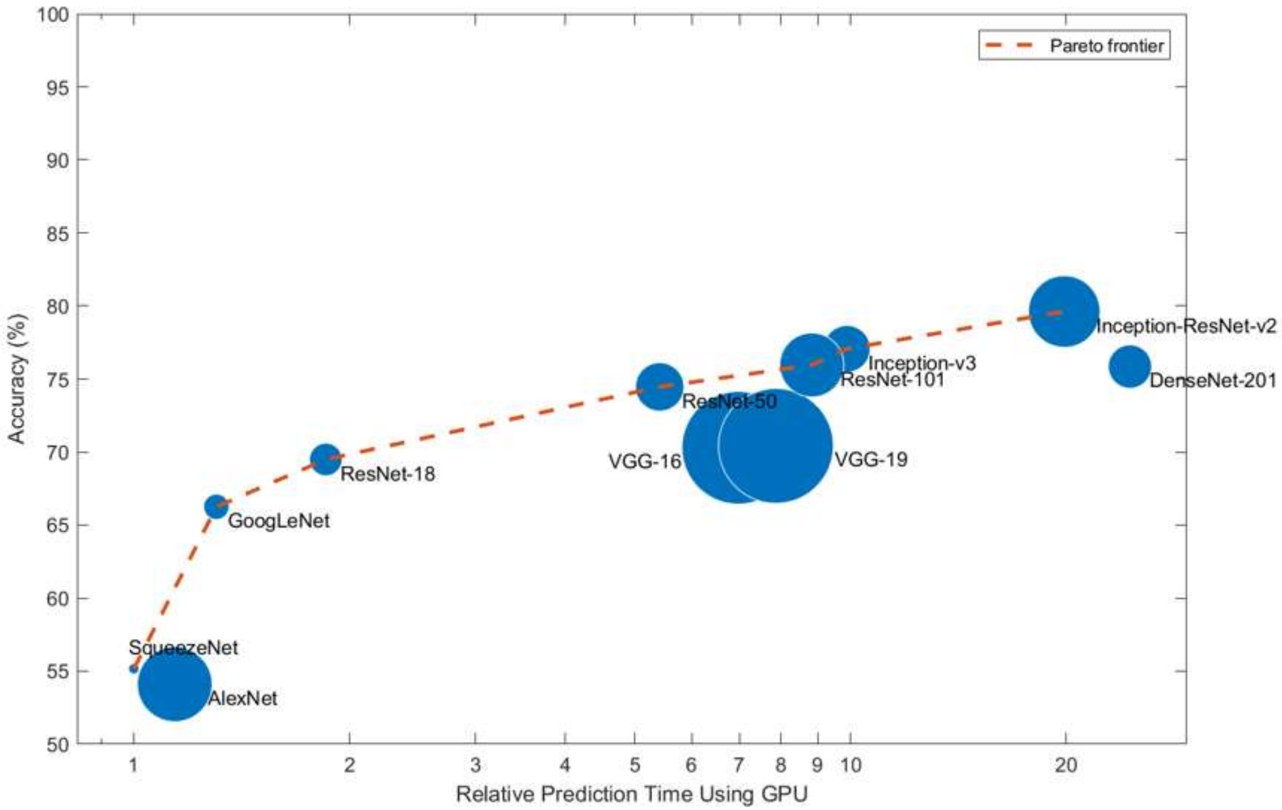
Object & Feature Detection – Demo on S32V HW



CNN Demo on S32V

CNN Selection

Deep Learning Toolbox in MATLAB provides a set a Pretrained networks to speedup the SW development

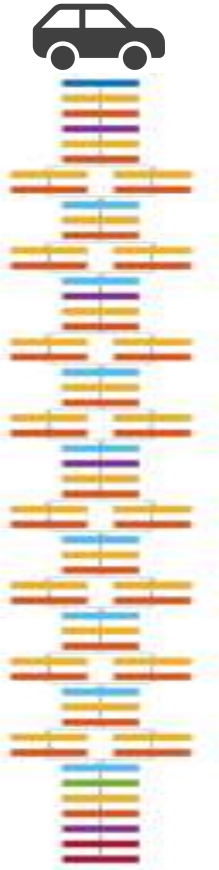


Network	Depth	Size MB	Params. (Millions)	Img. Size
alexnet	8	227	61.0	227x227
vgg16	16	515	138	224x224
vgg19	19	535	144	224x224
squeezenet	18	4.6	1.24	227x227
googlenet	22	27	7.0	224x224
inceptionv3	48	89	23.9	299x299
densenet201	201	77	20.0	224x224
resnet18	18	44	11.7	224x224
resnet50	50	96	25.6	224x224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

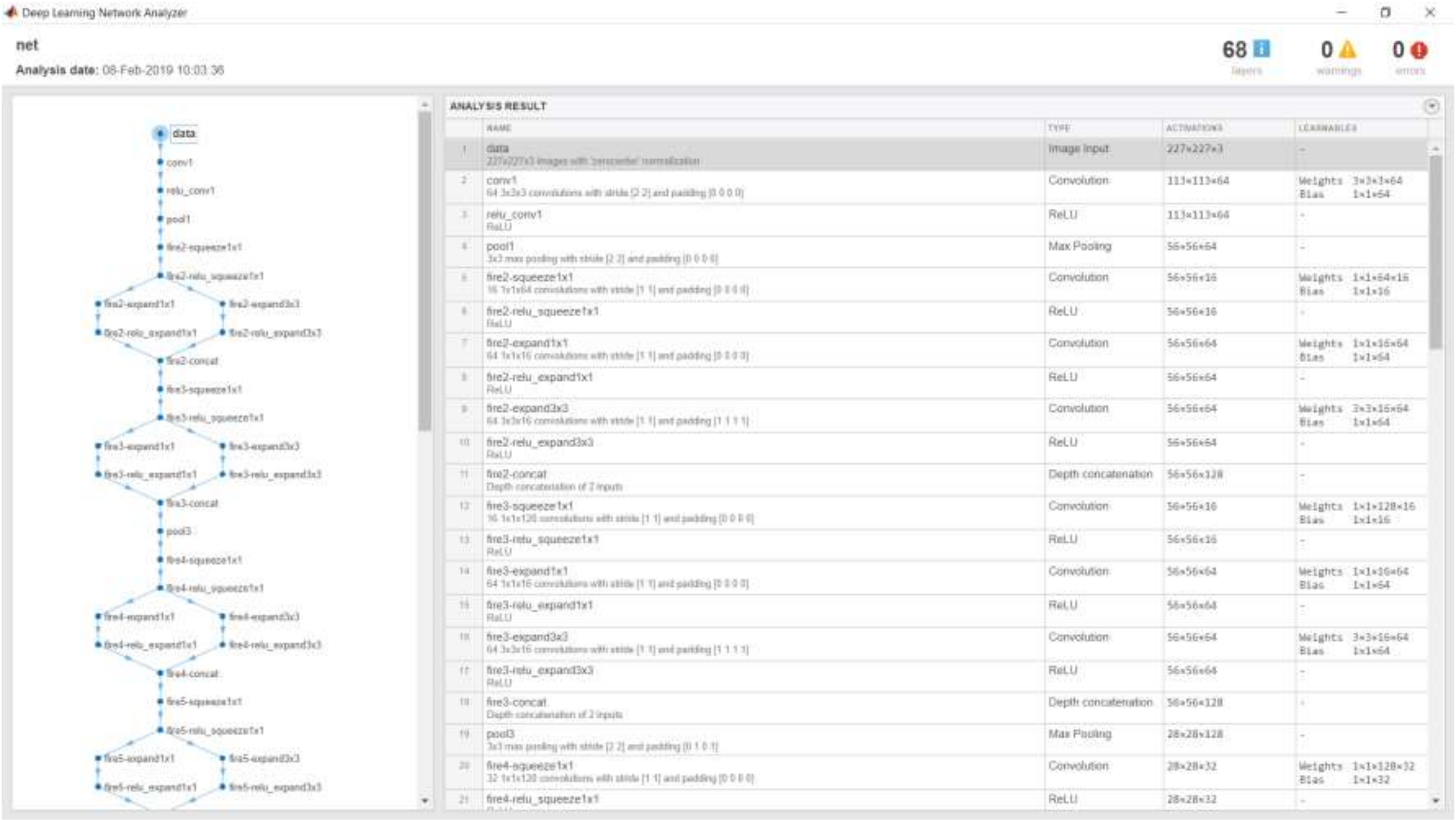
Source: MathWorks Website

SqueezeNet Model Used for Demo

Input image



Output prediction: CAR



CNN Object Prediction with S32V

TestMySqueezeNet.m

```
% Connect with S32V board and Create a camera object
s32 = nxpvt.s32v234('134.27.168.249', 123);
cam = nxpvt.cameraboard(s32, 1, 'Resolution', '720x1280');

% Process frames from S32V SonyCam and Predict
while (1)

    % Get video frames from S32V SonyCam
    img = cam.snapshot().data;

    % Resize image to fit SqueezeNet network
    [origwidth, origheight, ~] = size(img);
    img = imresize(img,[227,227]);

    % Predict with the SqueezeNet network
    [proc, classIdx] = mySqueezeNet(img);

    % Display results
    load classNames
    img = imresize(img,[origwidth, origheight]);
    img = insertText(img, [10, 10], [classNames{classIdx(1)} ' ' ...
        num2str(proc(1) * 100) '%'], ...
        'BoxOpacity', 0, 'TextColor','green');
    nxpvt.imshow(img);
end
```

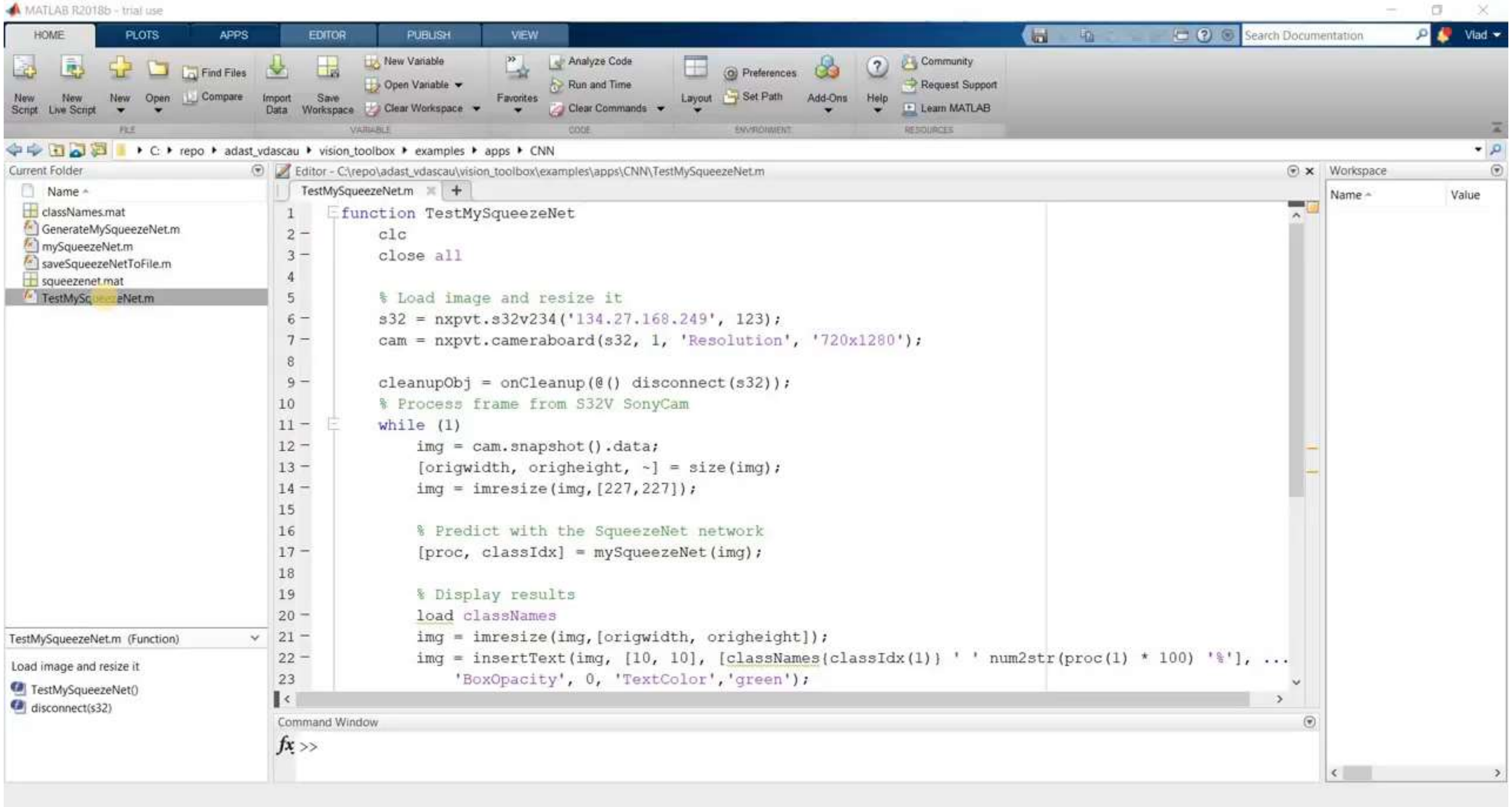
STEP 1: Create an object to connect with the NXP S32V from MATLAB.

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

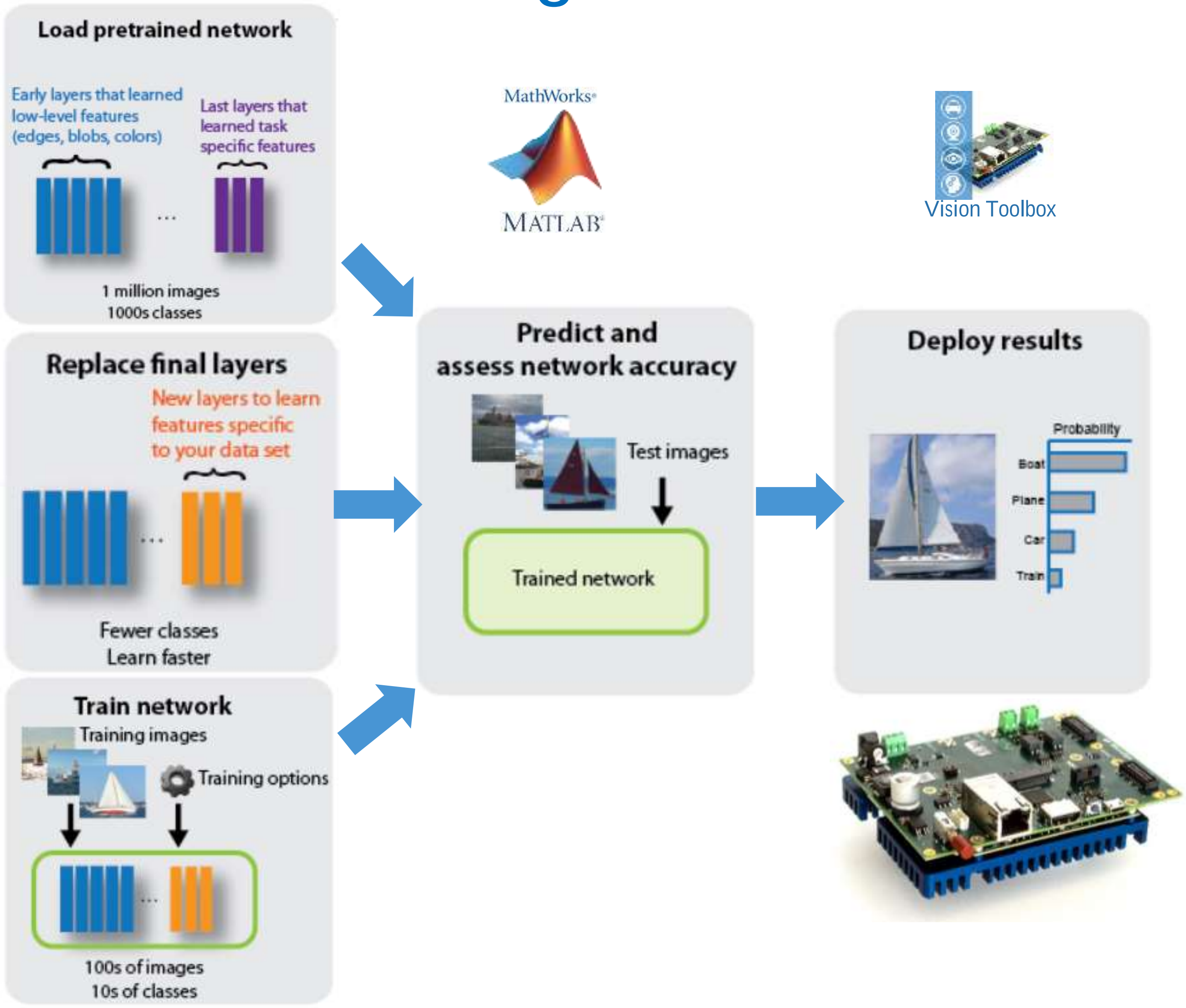
STEP 3: Run the SqueezeNet CNN to detect the objects within the video frames captured by S32V ISP

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

SqueezeNet CNN – Demo (Simulation & S32V HW)



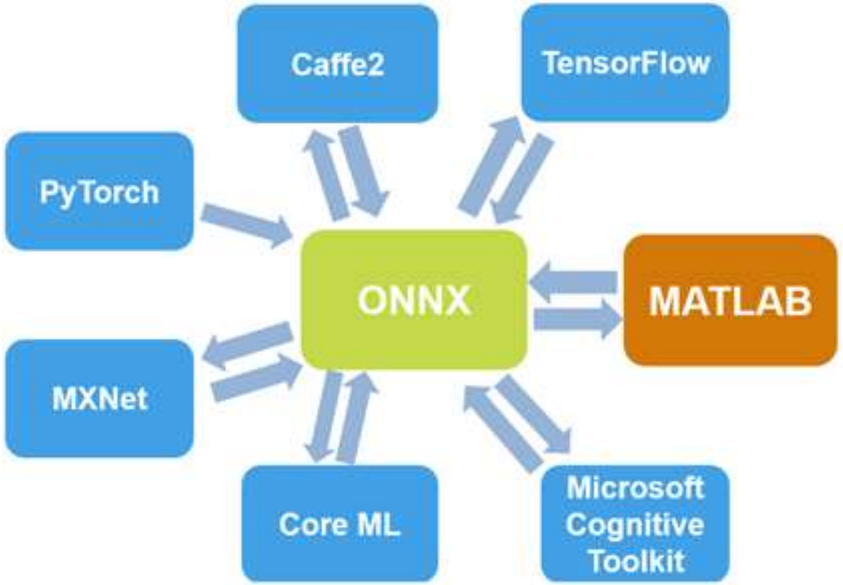
CNN Fine Tuning & Reuse



MathWorks
MATLAB

Vision Toolbox

Deep Learning Framework Interoperability



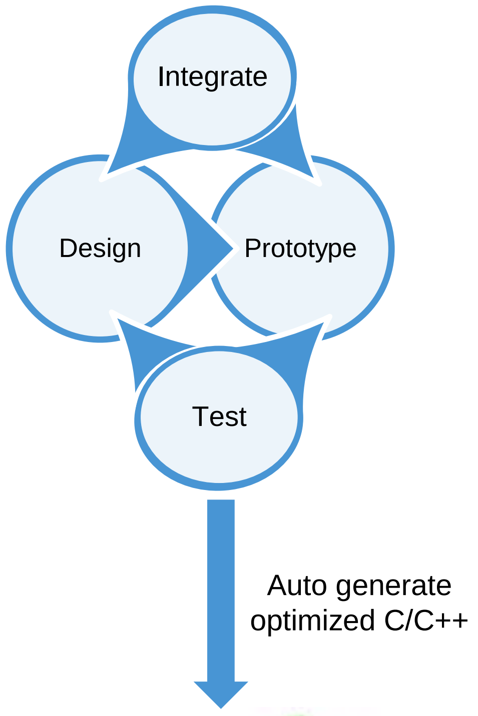
Source of Images: MathWorks Website

Conclusions

Achieve Efficient Development Workflow

Using MATLAB and various toolboxes, design engineers can:

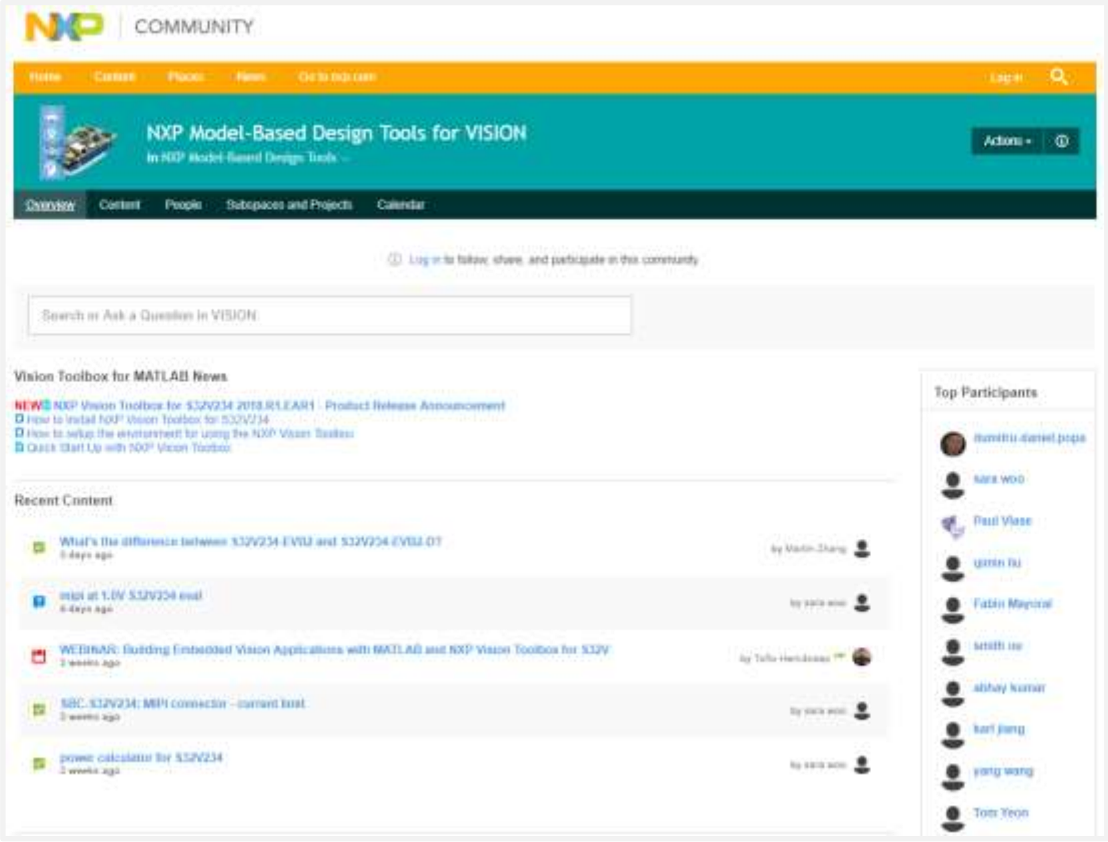
- Maintained one design in MATLAB
- Design faster and get to C quickly
- Test more systematic and frequently
- Focus on algorithm improvements

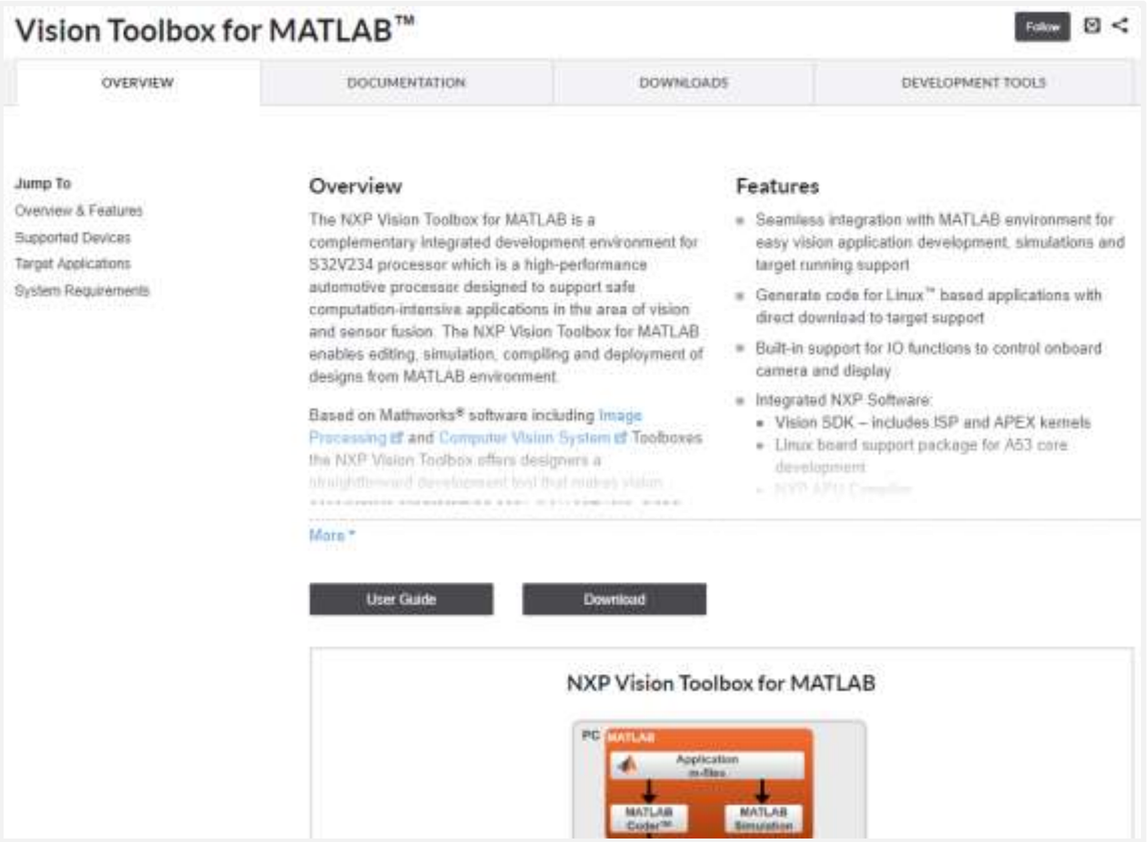


Getting Help

[Vision Toolbox Online Community Examples & Help]

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





SECURE CONNECTIONS
FOR A SMARTER WORLD