MathWorks
AUTOMOTIVE
CONFERENCE 2016

9월 2일 - 서울
Using Model-Based Design to develop high quality and reliable ADAS & Automated Driving Systems

김종현 차장
Senior Application Engineer
MathWorks Korea
Electronics & Active Safety: Helping to increase Road Safety

Evolution of Road Fatalities in EU (1991-2013)
Source: Website European Commission

How much were road fatalities reduced over the last 20 years?
Challenges in ADAS & AD Development

- **Algorithm**
  - Complicate algorithm to analyze the environment surrounding the car and make a decision in real-time
    - Perception
    - Localization
    - Situation analysis & Driving Behavior
    - Path Planning, Actuator Control

- **Process**
  - Vehicle prototype is difficult for testing in early engineering stage
  - Limited flexibility to vary and explore different design directions with real prototype

- **Test**
  - Vast amount of operating scenarios and data to test
    (Millions of test km required)
Agenda

- Challenges to automate the driving
- Example workflow for ADAS algorithm development
  - Managing & analyzing big data from vehicle fleet test
  - Algorithms MATLAB/Simulink provide to support ADAS development
  - Verifying and implementing ADAS algorithms using MBD
- ROS Interface for Automated Driving system development
- Q & A
MATLAB Gives Design Engineers the Environment for Algorithm Design, Exploration, and Verification

Interactive Exploration

Data

Algorithm

C Code

Regression Test
Developing models for ADAS Algorithms

Interactive Exploration

Data

Regression Test

Algorithm

C Code
Example of Analysis and Re-simulation Tool

1. Gather data
   - Instrument Toolox
   - Vehicle Network Toolbox
   - Image Acquisition Toolbox

2. Re-simulate
   - MATLAB/Simulink
   - Parallel Computing Toolbox
   - MDCS
Example of **Visualization and Algorithm development Tool**

- **Productivity**
  - Data visualization
  - Multi-modal data analysis

- **Allow Engineers Rapidly Build and Share Tools**
  - Easy-to-use plot functions
  - Build-and-deploy
  - Acquisition of Heterogeneous Data (CAN & Image)
Example of Analysis and Re-simulation Tool

1. Gather data
   - Instrument Toolox
   - Vehicle Network Toolbox
   - Image Acquisition Toolbox

2. Re-simulate
   - MATLAB/Simulink
   - Parallel Computing Toolbox
   - MDCS

3. Find situations
   - Various Data Analytics Tools

4. Classify and analyze
   - Various Data Analytics Tools
   - MATLAB Compiler
   - Report Generator

Model-Based Approach to Resource-Efficient Object Fusion for an Autonomous Braking System
Jonny Andersson, Scania,
MathWorks Automotive Conference 2015
MathWorks Tools Help Automate Ground Truth Labeling

- Tools for
  - Sensor input and visualization
  - Video annotation
  - Tracking and machine learning for automation

- Leverage parallel computing to speed up training

- Scale to big data architectures for processing and searching
function [frame, resultsThisFrame] = VGTGroundTruthingFunction(allInputs)

persistent pedestrianDetector stopSignDetector ROI

% INSTANTIATIONS (First-time only!)
if isempty(pedestrianDetector)
    pedestrianDetector = vision.PeopleDetector('UprightPeople_96x48',...
        'ClassificationThreshold', 2.5,...
        'ScaleFactor', 1.010,...
        'MinSize', [96,48], 'MaxSize', [384,192], ...%
        'UseSize', true,...
        'MergeROI', true);
end

if isempty(stopSignDetector)
    detectorRequested = 'C:\MathWorks DEMOS\TrainedDetectors\stopSignDetectorGoodest.xml';
    stopSignDetector = vision.CascadeObjectDetector(detectorRequested,...
        'MergeThreshold', 4);
end

% Parse inputs
frame = allInputs{1};
if isempty(ROI) ...
    [height,width,~] = size(frame);
    ROI = round([0.25*width 0.35*height 0.5*width 0.4*height]);
end

ROIsThisFrame = []; LabelsThisFrame = []; nROIsThisFrame = 0;
Continental built their tooling using MATLAB.
Developing models for ADAS Algorithms

Interactive Exploration

Data

Algorithm

C Code

Regression Test
# MathWorks Products for Vision Applications Development

<table>
<thead>
<tr>
<th><strong>Image Processing Toolbox™</strong></th>
<th><strong>Computer Vision System Toolbox™</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Contrast adjustment</td>
<td>- High-speed video I/O</td>
</tr>
<tr>
<td>- Geometric transformations</td>
<td>- Point Cloud processing</td>
</tr>
<tr>
<td>- Various filters</td>
<td>- Tracking</td>
</tr>
<tr>
<td>- Segmentation</td>
<td>- Stereovision</td>
</tr>
<tr>
<td>- Object analysis</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Image Acquisition Toolbox™</strong></th>
<th><strong>Statistics and Machine Learning Toolbox™</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Image capture from standard H/W</td>
<td>- Multivariate statistics</td>
</tr>
<tr>
<td>- Analog, Camera Link, DCAM, GigE Vision, USB camera, etc</td>
<td>- Probability distribution</td>
</tr>
<tr>
<td>- Microsoft Kinect Support</td>
<td>- Machine learning</td>
</tr>
<tr>
<td></td>
<td>- Experimental design</td>
</tr>
<tr>
<td></td>
<td>- Statistical process control</td>
</tr>
</tbody>
</table>
Cascade Detector

Computer Vision System Toolbox - 4 Easy Steps to Create a Car Detector
Example of 3D Vision Algorithm Development

Computer Vision System Toolbox

- Stereo Calibration App.
- Surface Modeling & Freespace Segmentation
- Building Stixel
- Feature Detection/Extraction
- Feature Matching
- Point Tracker
- Point Cloud Visualization
- Kd-tree based Point Cloud Object
- Point Cloud Matching

- Stereo Matching
- Surface Matching
- Road-Object segmentation
- Stixel World Building

- KLT based Image Tracking
- Ego Vehicle Status Estimator
- Delta X, Delta Y, Vx, Vy

- Inertial Signals

19
if(params.ptCloud.pcfitplane.enable)
    if(params.ptCloud.pcfitplane.lowerHeight>0)
        [~,outlierIndices] = pcfitplane(ptCloud, params.ptCloud.pcfitplane.lowerHeight, [0,0,1]);
        ptCloud = select(ptCloud, outlierIndices);
    end
    if(params.ptCloud.pcfitplane.upperHeight>0)
        [~,inlierIndices, ~] = pcfitplane(ptCloud, params.ptCloud.pcfitplane.upperHeight, [0,0,1]);
        ptCloud = select(ptCloud, inlierIndices);
    end
end
# Deep Learning for Object Recognition

## Neural Network Toolbox

## Addressing Challenges in Deep Learning for Computer Vision

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing large sets of labeled images</td>
<td><strong>imageSet</strong> or <strong>imageDataStore</strong> to handle large sets of images</td>
</tr>
<tr>
<td>Resizing, Data augmentation</td>
<td><strong>imresize, imcrop, imadjust, imageInputLayer</strong> etc.</td>
</tr>
<tr>
<td>Background in neural networks (deep learning)</td>
<td>Intuitive interfaces, well-documented architectures and examples</td>
</tr>
<tr>
<td>Computation intensive task (requires GPU)</td>
<td>Training supported on GPUs No GPU expertise is required</td>
</tr>
<tr>
<td></td>
<td>Automate. Offload computations to a cluster and test multiple architectures</td>
</tr>
</tbody>
</table>

**CNN Object Detector**
MathWorks Products for Radar Development

- **Channel modeling (interference, noise)**
  - Communications System Toolbox

- **Determination of waveforms (range, resolution)**
  - Phased Array System Toolbox
  - Signal Processing Toolbox
  - Instrument Control Toolbox

- **Modeling of antenna arrays (no of elements, position)**
  - Phased Array System Toolbox

- **Modeling of RF Impairments (noise, non-linearity, frequency dependency)**
  - SimRF

- **Algorithms for data analysis**
  - Phased Array System Toolbox
  - DSP System Toolbox
Phased Array System Toolbox

- Waveforms: Pulse, LFM, FMCW, etc.
- Transmitter: Monostatic and Bistatic
- Tx Antenna Arrays: ULA, URA, etc.
- Receiver: Monostatic and Bistatic
- Rx Antenna Arrays: ULA, URA, etc.

## Algorithms

- Beamforming, Matched Filtering, Detection, CFAR, STAP, etc.
- Environment effects, impairments, interference

## Simulation Tools
Phased Array System Toolbox

Algorithms

Root MUSIC DOA
Estimate the direction of arrival of a specified number of narrowband signals incident on a uniform linear array using the Root MUSIC algorithm.

Main

Direction of Arrival

Space-Time Adaptive Processing
Detection

Sensor Array

Signal propagation speed (m/s): \(v = \text{physconst}('\text{LightSpeed}')\)

Operating frequency (Hz):

Number of signals:

Spatial smoothing:

Simulate using: Interpreted execution
How about Sensor fusion algorithm for FCW

- Data Pre-processing
  - Calculate Ground Speed
  - Object classification

- Path Estimation
  - Zoning

- Sensor Fusion & Tracking
  - Sensor Fusion
  - Kalman Filter

- Threat Assessment
  - Maneuver Analysis
  - Risk Assessment
  - Find MIO

MIO: Most-Important Object
Sensor fusion algorithm for FCW

- **Data Pre-processing**:
  - Calculate Ground Speed
  - Object classification

- **Path Estimation**: Zoning

- **Sensor Fusion & Tracking**: Sensor Fusion → Kalman Filter

- **Threat Assessment**: Maneuver Analysis → Risk Assessment → Find MIO → Zoning

- **Radar Object**
- **Vision Object**
- **Vision LD**
- **Vehicle CAN**

**Legend**: FCW, MIO: Most-Important Object
Sensor Fusion Made Easy by MATLAB CVST

[assignments, unassignedVisions, unassignedRadars] = ...
assignDetectionsToTracks(costMatrix, param.costOfNonAssignment);
Kalman Filter

**Initial state & covariance**

\[
\begin{align*}
\hat{x}_0 & \quad \mathbf{P}_0 \\
\hat{x}_{k-1} & \quad \mathbf{P}_{k-1}
\end{align*}
\]

**Previous state & covariance**

**Current becomes previous**

\[
\begin{align*}
k \rightarrow k - 1
\end{align*}
\]

**Time Update ("Predict")**

1. Predict state based on physical model and previous state
   \[
   \hat{x}_k^- = A\hat{x}_{k-1} + Bu_k + w_k
   \]
2. Predict error covariance matrix
   \[
   P_k^- = AP_{k-1}A^T + Q
   \]

**Measurement Update ("Correct")**

1. Compute Kalman gain
   \[
   K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}
   \]
2. Update estimate state with measurement
   \[
   \hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-)
   \]
3. Update the error covariance matrix
   \[
   P_k = (I - K_k H) P_k^-
   \]

**Symbols and Matrices**

- **u**: Control variable matrix
- **w**: Process (state) noise
- **P_k^-**: Process (state) covariance matrix
- **e_k**: Measurement noise covariance matrix (estimation error)
- **Q**: Process noise covariance matrix
- **A**: State matrix relates the state at the previous, \(k-1\) to the state at the current, \(k\)
- **R**: Sensor noise covariance matrix (measurement error)
- **H**: Output matrix relates the state to the measurement
- **z_k**: From sensor spec or experiment
Kalman Filter Made Easy by **MATLAB CVST**

Initial state & covariance

\[
\hat{x}_0, P_0
\]

Previous state & covariance

\[
\hat{x}_{k-1}, P_{k-1}
\]

Time Update (“Predict”)

\[
[z_{\text{pred}}, x_{\text{pred}}, P_{\text{pred}}] = \text{predict}(\text{obj})
\]

- \(z_{\text{pred}}\): prediction of measurement
- \(x_{\text{pred}}\): prediction of state
- \(P_{\text{pred}}\): state estimation error covariance at the next time step

\(k \rightarrow k-1\)

Current becomes previous

Predicted state

\[x_{\text{pred}}\]

Measurement Update (“Correct”)

\[
[z_{\text{corr}}, x_{\text{corr}}, P_{\text{corr}}] = \text{correct}(\text{obj}, z)
\]

- \(z_{\text{corr}}\): correction of measurement
- \(x_{\text{corr}}\): correction of state
- \(P_{\text{corr}}\): state estimation error covariance

\[z\]

Output of updated state

\[x_{\text{corr}}, P_{\text{corr}}\]

```
```
Developing models for ADAS Algorithms

Interactive Exploration

Data

Algorithm

Regression Test

C Code
Create components in MATLAB and reuse them in Simulink

```matlab
[mostImportantObject,...
egoPath, ...
radarObjects, ...
visionObjects, ...
fusedObjects, ...
assessedThreats] = forwardCollisionWarning( ...
lane, inertialMeasurementUnit, ...
vision, radar, params, reset);
```
Automate regression testing with Simulink Test

Specify tests and Interact with results in the Test Manager

Generate a report to share results
Generate C code for your algorithm with MATLAB Coder

```matlab
function [mostImportantObject, ...
    egoPath, ...
    radarObjects, ...
    visionObjects, ...
    fusedObjects, ...
    assessedThreats] = forwardCollisionWarning( ...
    lane, ...
    inertialMeasurementUnit, ...
    visionSensor, ...
    radarSensor, ...
    params, ...
    reset)
```

```c
void forwardCollisionWarning(const laneStruct *lane, const
    inertialMeasurementUnitStruct *inertialMeasurementUnit, const
    visionObjectsStruct *visionSensor, const radarObjectsStruct *radarSensor,
    const fcwParamsStruct *params, boolean_T reset, mostImportantObjectStruct
    *mostImportantObject, egoPathStruct *egoPath, radarObjectsStruct *radarObjects,
    visionObjectsStruct *visionObjects, struct6_T *fusedObjects, struct8_T
    *assessedThreats)
```
Enable coverage of MATLAB and C code with Simulink Verification and Validation

Collects coverage on MATLAB code in “Normal” mode

Collects coverage on generated C code in “Software in the loop” mode
Forward Collision Warning
Test Bench

Ports Mapped To Data From Workspace

1. Lane
2. IMU
3. Vision
4. Radar

Lane
Inertial_Measurement_Unit
Vision
Radar

fcwC
Ego_Path
Vision_Objects
Fused_Objects
Assessed_Threats

lane
imu
egoPath
visionObjects
fusedObjects
assessedThreats

mostImportantObject
egoPath
radarObjects
visionObjects

StartFcn = BirdsEyeView.bringFigureToFront;
DeleteFcn = BirdsEyeView.close;
Prototype on hardware with Simulink Real-Time
as seen in today’s Test drive your ADAS algorithms presentation

- Algorithm Models
  - Forward Collision Warning
  - Autonomous Emergency Braking

- Vehicle and Environment Models
User Stories

Ground Detection using Backward Camera

Results - Performance

- Near real-time performance with ~10fps in SIL mode for “pure” algorithm in Coder generated C.
- Achieved the deployment target of 50ms.
- Performed well on all scenarios.
- Slow movements were also handled well.
- Excellent performance with respect to overall system.

Conclusion

- MATLAB & Coder allowed us to focus on algorithm design.
  - Quick turn around time was ideal for experimentation.
  - Challenges in finding replacement or equivalent code in C for functions that cannot be converted by Coder.
  - Overall, MATLAB & Coder is a powerful tool for quick prototyping.

For future improvements:

- Our existing MATLAB code could be refactored into modules for more flexible C-code generation.

Dynamic Alignment of Radar Sensor

Coder Code Performance

<table>
<thead>
<tr>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable. Coder code has been used in production code for half a year and no bug is found.</td>
</tr>
<tr>
<td>Efficient. Coder code is used for a variety of radar sensor alignment algorithms.</td>
</tr>
<tr>
<td>Easy to implement. MATLAB algorithms can be directly used as Coder code.</td>
</tr>
</tbody>
</table>

Summary

- Radar Sensors’ Alignment Algorithm design, simulation, data analysis, and implementation are done together within Matlab only.
- One algorithm engineer can completely responsible with one algorithm block. Save time to coordinate with software engineer.
- In the future, Matlab plus Coder will be popular in algorithm development area.
Agenda

- Challenges to automate the driving
- Example workflow for ADAS algorithm development
  - Managing & analyzing big data from vehicle fleet test
  - Algorithms MATLAB/Simulink provide to support ADAS development
  - Verifying and implementing ADAS algorithms using MBD

- ROS Interface for Automated Driving system development

- Q & A
Challenges of Doing It All by Yourself

You may need…

- Reliable distributed architecture
- Algorithms for autonomy
  - Path planning
  - Collision avoidance
  - Sensor processing
- Sensor data acquisition
- Simulation environments
Automated Driving at BMW

AUTOMATED DRIVING WITH ROS AT BMW.

MICHAEL AEBERHARD, THOMAS KÜHBECK, BERNHARD SEIDL, MARTIN FRIEDL, JULIAN THOMAS, OLIVER SCHEICHL.

BMW GROUP
Potential Frameworks @ BMW

1. In-house SW
   - BMW internally developed Framework for prototyping ADAS.
   - Shared memory transport mechanism.
   - Synchronized execution of software modules.
   - Internal development limited/complex.

2. Commercial SW
   - Commercial product popular within the automotive industry (OEMs/Suppliers).
   - Readily available toolboxes to hardware used in the automotive industry.
   - Easy to use GUI for manipulating various features and configuration a system.
   - URL: https://www.elektrobit.com/products/eb-assist/adtf/

3. Open source
   - Popular open source robotics framework.
   - Reliable distributed architecture.
   - Wide use in the robotics research community.
   - Huge selection of “off-the-shelf” software packages for hardware/algorithms/etc.
Potential Frameworks @ BMW

Why we ended up choosing ROS for the BMW research department:

- Autonomous driving benefits from robotics research and ROS has become very popular in the robotics community.
- Stability and reliability from a very large user-base.
- Quick tests and integration of already-available algorithms and software packages → saves development time.
- Open source.
- Easier cooperation with universities and other research institutes.
- Gain experience at BMW with using ROS and learn about its advantages /disadvantages with respect to other solutions → research department should try something new!
What is ROS (Robot Operating System)?

- An architecture for distributed inter-process communication
- Packages for common algorithms and drivers
- Multilanguage interface (C++, Python, Lua, Java and MATLAB)

Benefit for Automated Driving

- Sensor data acquisition
- Reliable distributed architecture
- Lots of “off the shelf” algorithms for autonomy
  Path planning / Collision avoidance /Sensor processing
- Simplified component compatibility through standalone interfaces
- Integration with simulation environments
ROS Trends in Robotics Development

• #1 middleware for robotics applications development

• Popular in research and gaining great momentum in industry

http://rosindustrial.org/ric-americas/
Autonomous Car as an Advanced Robotics System

Planning → Motion control

Localization → Motion control

Obstacle avoidance → Motion control

Global Map → Motion control

ROS: communication framework and stack of libraries

NODE

LIDAR
Camera
RADAR
GPS/IMU

NODE

Motion Controllers

NODE

Actuator ECUs
Steering Actuator
GAS/Brake Actuator
With MATLAB/Simulink?

- Early Idea
  - Custom C
  - MATLAB Code
  - Simulink Model

- Generate Code
  - C/C++ Code

- ROS
  - Need to learn ROS and Linux
  - Should be familiar with C++
  - Difficult to get started
Workflow with Robotics System Toolbox

Without RST

Early Idea

MATLAB Code

Simulink Model

Custom C

Generate Code

C/C++ Code

- Need to learn ROS and Linux
- Should be familiar with C++
- Difficult to get started

ROS

With RST

Early Idea

MATLAB Code

Simulink Model

Custom C

Merge to Simulink

Operation connecting with ROS

OR

Code Generation with Interface

ROS
What can be done with the Robotics System Toolbox?

MATLAB on PC

- MATLAB Code
- Built-in algorithms
- SM Models

Networking

ROS

Robot

Simulation environment

ROS node
Co-simulation with ROS

```matlab
%% Connect to ROS
rosinit '192.168.204.144';

%% Create subscribers
imSub = rossubscriber('/camera/rgb/image_raw');
scanSub = rossubscriber('/scan');

%% Create publisher
[velPub, velMsg] = rospublisher('/husky_velocity_controller/cmd_vel');
```
Co-simulation with ROS
What can be done with the Robotics System Toolbox?

- MATLAB on PC
- MATLAB Code
- Built-in algorithms
- SM Models
- Networking
- ROS
- Robot
- Simulation environment
- ROS node
Co-simulation with ROS

%% Connect to ROS
rosinit '192.168.204.144';

%% Create subscribers
imSub = rossubscriber('/camera/rgb/image_raw');
scanSub = rossubscriber('/scan');

%% Create publisher
[velPub, velMsg] = rspotpub('/cmd_vel');

Sensor
Communication Layer

Test Images

Image from ROS

Robot URI
Ex) 192.168.204.150

Simulator URI
Ex) 192.168.204.144

Actuator
Communication Layer

Send Velocity to ROS
Implementation
What can be done with the Robotics System Toolbox?

- MATLAB Code
- SM Models
- Built-in algorithms
- Networking
- Code Generation
- MATLAB on PC
- Robot
- Simulation environment

Generate standalone ROS node from Simulink
Automated Driving at BMW

Using MATLAB/SIMULINK with ROS.

- MathWorks released the Robotics System Toolbox this year for ROS integration with Matlab/Simulink.
- Easily read and analyze data from ROS Bags → useful for evaluating the system.
- Some of our software is implemented as a Simulink model.
  - Use the Toolbox to easily integrate this software into the ROS eco-system:

http://www.mathworks.com/products/robotics/
Automated Driving with ROS at BMW

** MICHAEL AEBERHARD, BMW, ROSCON 2015**
The MATLAB environment helps engineers …

- **Gain insight** by visualizing and analyzing data
  - Plotting helps you understand sensor behavior and “give life to the data”
  - Creating Apps helps you simplify the analysis process

- **Speed up algorithm design to implementation iterations** by generating code
  - Generating C code helps the algorithm and implementation stay synchronized
  - Automating regression testing helps algorithm and software engineers collaborate

- **Have a rapid interactive environment leveraging the ROS communication framework and stack of libraries.**
Thank you for your attention! Questions?