Trip Optimizer
Development of a Driver Assistance System for Locomotives Using MATLAB

8 May 2017
Agenda

1. Trip Optimizer Overview
2. Trip Optimizer and MATLAB
3. Integrating an external optimization library into MATLAB Code Generation Toolset
Trip Optimizer Overview
Train and driver variations result in:

- Less than optimal fuel use
- High emissions
- Trip variations
- Wear and tear

Trip Optimizer:

- Looks over the entire route for fuel savings opportunities
- Then controls the throttle to the plan
  - Saves fuel
  - Reduces emissions
  - Reduces equipment wear and tear
  - Consistent trips improve scheduling
Trip Optimizer Deployment/Operation

- 8,000 systems installed world wide
- 60,000 miles of mapped track
- 216M miles of auto operation
- 73M miles of auto in 2016
- 1.7M auto miles per week
- 142,000 gallons of diesel fuel saved per week
Trip Optimizer and MATLAB
Converging Technologies

• Research and development into Trip Optimizer began close to the time MathWorks began rolling out automatic code generation from Simulink
• Trip Optimizer team leveraged this technology to quickly produce proof of concept simulations
• Automatic code generation for embedded targets allowed accelerated transition from simulation environment to on locomotive demos
• Simulink and MATLAB now embedded in the core elements of the product
Trip Optimizer Block Diagram

- **Customer & GE IT Systems**
  - GE Offboard
  - External Interface & Track Database
  - Planner
  - GPS
  - Speed
  - Master Control

- **Auxiliary Processing Node**
  - Train & Trip Information
  - Route & Plan Info

- **Control System Node**
  - HMI
  - Core Loco Control
  - Supervisor
  - Location Determination
  - Speed Regulator, Estimation, & Train Handling

- **Throttle Trainlines**

---

Elements include:
- **HMI**
- **Core Loco Control**
- **Supervisor**
- **Location Determination**
- **Speed Regulator, Estimation, & Train Handling**
- **Throttle Trainlines**
- **GPS**
- **Speed**
- **Master Control**
- **Train & Trip Information**
- **Route & Plan Info**

Other notable elements:
- **Control System Node**
- **Customer & GE IT Systems**
- **GE Offboard**
- **External Interface & Track Database**
- **Planner**
- **Auxiliary Processing Node**
- **Throttle Trainlines**
- **GE Proprietary**

---

This diagram illustrates the flow of information and control processes within a trip optimizer system, highlighting the integration of various systems and nodes to manage train operations efficiently.
Integrating an External Optimization Library into the MATLAB Code Generation Toolset
Trip Optimizer Planner

- Fuel optimal plan generated for entire route at time of trip initialization
- Trip plan adjusted to account for changes in conditions along the route
- Plan speed is reference for speed regulator
- Plan throttle is 100% feed forward term on speed regulator output
Why Integration?

- m-code "Pre-Process" → AutoCode_EntryPoint_1()
- m-code "Initial Guess" → AutoCode_EntryPoint_2()
- m-code "Build Inputs" → Hand Code
- Invoke external executable
- m-code "Transform Output" → Hand Code
- m-code "Post-Process" → AutoCode_EntryPoint_3()

Single interface point with hand-code

m-code "Do It" → AutoCode_EntryPoint()

Desktop environment equivalent to embedded target
The Optimization Library C++ Interface

```cpp
SmartPtr<ProblemDefinition> my_problem = new MyProblemDefinition();
SmartPtr<OptimizationApplication> app = ApplicationFactory();
app->Initialize();
app->Optimize(my_problem);
```
How We Did It – ceval + minGW + addLinkObjects

```matlab
function [ProfileOut, errCode, cpuTime, nIter] = CallOpt_ceval(Optspec, Mesh, espec, ProfileOut)

    [OptData, Constants] = PopulateInitData_ceval(Optspec, Mesh, espec, ProfileOut);

    x = zeros(1, OptData.nvars);
    nIter = int32(0);
    errCode = int32(0);
    cpuTime = double(0);

    coder.ceval('ceval_optimizer.m');
    coder.ceval('ceval_optimizer', coder.rref(OptData), coder.vref(x), coder.vref(errCode), coder.vref(nIter), coder.vref(cpuTime));

    ProfileOut = Post_Process_ceval(Mesh, Constants, x, ProfileOut);

end
```

coder.updateBuildInfo('addLinkObjects', 'liboptimizer.a', ipoptLibPath, '', true, true);
coder.updateBuildInfo('addLinkObjects', 'libfortran.a', mingWLibPath, '', true, true);
coder.updateBuildInfo('addLinkObjects', 'libquadmath.a', mingWLibPath, '', true, true);
coder.updateBuildInfo('addLinkObjects', 'libstdc++.a', mingWLibPath, '', true, true);

Trip Optimizer: Development of a Driver Assistance System for Locomotives Using MATLAB | 8 May 2017 | GE Proprietary
Algorithm development

• Most robust debugging environment – breakpoints, figure, variable inspection

Logic errors
• Example: Air brake effort is negative, I should be using max() to limit it

Code generation
• Identify unsupported functions

Code flow errors
• Example: Can’t assign a field to a structure after it’s been accessed

Memory allocation errors
• Example: myvar = 1; myvar(2) = 2; Works in MATLAB, doesn’t work in generated code

Gross performance measures
• Example: I could do this interpolation one time and pass it around instead of doing it over and over

Compare code outputs from target and MATLAB environment

Defects not expected
• Each time the model is run on the target all inputs are written to a text file which can be read into MATLAB to recreate the scenario exactly
• Failures from field can be brought back to MATLAB environment for debugging/algorithm enhancement
Conclusions

• Integrating external code into code generation process can enable parity between MATLAB development environment and embedded execution target
• Increased productivity – Development and debugging
• Defects found earlier in life cycle