Bet & MathWorks

By Bet Herrera Sucarrat
Application Engineer
MathWorks
Researchers Test Control Algorithms for NASA SPHERES Satellites with a MATLAB Based Simulator

**Challenge**
Provide a platform for debugging and testing formation flight, autonomous rendezvous, and docking algorithms for satellites

**Solution**
Use MATLAB and Simulink simulation and 3D visualization to verify control algorithms and evaluate test results aboard the International Space Station

**Results**
- Algorithms verified via simulation
- Experimental results visualized in 3D
- Unique educational opportunity opened

For NASA, developing satellite trajectory optimization and control algorithms with MATLAB and related toolboxes is about twice as fast as developing them with languages that require everything to be coded from scratch.
**ESA’s First-Ever Lunar Mission Satellite Orbits Moon with Automatically Generated Flight Code**

**Challenge**
Develop ESA’s first lunar mission in a short time frame and at minimal cost

**Solution**
Use MathWorks tools for Model-Based Design to model, simulate, generate, and test the flight code

**Results**
- Reduced system development time by 50%
- Improved process efficiency
- Produced efficient code

“MathWorks tools for simulation and flight-code generation played a key role in this success and will serve as the foundation for future satellite programs, such as Prisma.”

Per Bodin
Swedish Space Corporation

**Link to user story**
JPL Tests, Tunes, and Implements Onboard Descent Systems for Spirit and Opportunity Mars Rovers

**Challenge**
Design an entry, descent, and landing (EDL) system for a Mars rover

**Solution**
Use MATLAB and Simulink for numerous phases of the mission, including navigation, data analysis, and EDL hardware and software design

**Results**
- Onboard descent systems tested under authentic atmospheric conditions
- Viable Mars landing sites easily identified
- Spacecraft landing flawlessly executed

Using MATLAB, engineers tested, tuned, and implemented the onboard descent systems that told the rover which rocket systems to fire and when. The landing of the rover happened precisely as the EDL team had predicted.

[Link to article](#)
NASA Uses Stateflow and Simulink Coder to Generate Fault-Protection Code for Deep Space 1

**Challenge**
Design and code the fault-protection system of a robotic spacecraft within strict time and budgetary limits

**Solution**
Use Stateflow and Simulink Coder to create the logic flow and automatically generate code for the fault-protection system

**Results**
- Prolific generation of reliable code
- Record-breaking operational lifespan
- Rock-solid performance

"Until Deep Space 1, statecharts and automatic code-generation technology had not been used on large systems for spacecraft avionics software. MathWorks tools made this approach possible."

Dr. Wesley Huntress
NASA

Deep Space 1 spacecraft. (Image courtesy of NASA/JPL/Caltech)
Lockheed Martin Space Systems Uses SimMechanics with a Real-Time Simulator to Automate Mars Reconnaissance Orbiter Development

Challenge
Develop the guidance, navigation, and control system for the Mars Reconnaissance Orbiter

Solution
Use MathWorks tools to accelerate control design and automate the development of accurate, real-time spacecraft simulations

Results
- Spacecraft pointing simulation modeled in days
- Interorganization communication improved
- Efficient code generated automatically

“Simulink, SimMechanics, and Simulink Coder enabled us to autonomously go from an accurate CAD model of the MRO vehicle into C code that runs in real time.”

Jim Chapel
Lockheed Martin Space Systems

Artist’s rendition of Mars Reconnaissance Orbiter. (Image courtesy of NASA.)

Link to user story
Lockheed Martin Space Systems develops GN&C system for IRIS satellite with Model-Based Design

Challenge
Develop the guidance, navigation, and control (GN&C) system for the Interface Region Imaging Spectrograph (IRIS) observatory satellite

Solution
Use Model-Based Design with MATLAB and Simulink to model components of the GN&C system and the IRIS satellite, run closed-loop and processor-in-the-loop simulations, and generate production code

Results
- Development efficiency doubled
- Efficient, defect-free code generated
- Design updates completed in a single day

“...designed, integrated, and tested the GN&C system in just 23 months. We were more efficient because we used the same tools for both analysis and code development, and generated 20,000 lines of defect-free code. For us, that makes a compelling case for Model-Based Design.”

Vincentz Knagenhjelm
Lockheed Martin Space Systems

Link to user story
A Case In Point: Developing Satellite Formation Flying Software With Model-Based Design
by SSC and MathWorks' Engineering Team

Today's satellites support a wide range of civilian and military applications, including televisions, cell phones and Global Position Systems (GPS). As demand for these applications increases, the satellite industry must continue to innovate to provide faster, more efficient, and more cost-effective service. Instead of deploying large, complex geosynchronous satellites, the industry is moving to constellations of smaller satellites flying in formation. These smaller satellites provide the same functionality as their bigger predecessors, but are less expensive to build and launch.

Challenge
While the cost of manufacture and launch scales down with satellite size, the engineering development cost for satellites does not – engineering development cost is mainly driven by the new technology incorporated into the satellite and operational complexity. For example, formation flying will require satellites to possess new technology to communicate with one another as well as complex algorithms to position themselves relative to each other in real time. To meet these new functional requirements, engineers need to explore a wider range of design alternatives, which further drives up development cost.

Tango and Mango, the two small satellites making up the PRISMA mission. Credit: CNES
What it means to be a developer at MathWorks

MATLAB: committed to excellent algorithms

- Setting up the problem – what are we trying to build?

- How are we going to do it
  - Design
  - Implementation
  - Testing
What do you want MATLAB to do?

MATLAB … Do the bits of my work I don’t really want to do – please!

MATLAB is a place where we build tools to help you do your work
Development trade-off

Time

Quality

Features
Attributes of good software

- Bug-free, Fast, Accurate, Usable
- Accurate
- Usable
- Bug-free
- Fast
The Main Software Engineering Principle

Frequently developers are surprisingly stupid
A software engineering methodology

- What and Why?
- How? (for the user)
- How? (for the software system)
- How? (no seriously, I do actually need to write some code!)
- Did I make a mistake? (in my own code)
- Did I make a mistake? (in my team)
- Did I make a mistake? (in any code from MathWorks)
- …
A software engineering methodology

- Requirements

- Functional Design
  - Function prototypes, object interface, API, etc.
  - Graphical User Interfaces

- Architecture
  - Modularity
  - Components
  - Reuse
Real situation: Make MATLAB use GPU’s

- **Requirements**
  - Things the GPU is good at – maths, what else?
  - All GPU’s? Some subset?

- **Functional Design**
  - The Nike Approach – Just Do It …
  - Explicit opt-in
“Simple to use” vs. “Lots of control”

- Solve $A\times x = b$ for unknown $x$
- Simple
  
  $x = A\backslash b$
- Lots of control
  
  DGETRS
  DGBTRS
  DGTTRS
  DPOTRS
  DPOTRS
  DPBTRS
  DPTTRS
  DSYTRS
  DSPTRS
  DSPTRRS
“Simple to use” vs. “Lots of control”

<table>
<thead>
<tr>
<th>Level of Control</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>gpuArray, maths, same syntax as MATLAB</td>
</tr>
<tr>
<td>Intermediate</td>
<td>arrayfun(@fun, ...)</td>
</tr>
<tr>
<td>Detailed</td>
<td>Direct integration with CUDA kernels</td>
</tr>
</tbody>
</table>
Implementation

- Remember The Main Software Engineering Principle
  - You can only remember 3 ± 4 things at any one time!
- Keep ALL code as simple as possible
- Always use standard patterns

- Don’t describe what you are doing, describe why

- Some details
  - McCabe complexity < 15
  - Compiler warning free, close to lint-free
  - Short functions (less than 2 screens)
  - Lots of green
Review Everything
Principle of Least Surprise

The only valid measurement of code quality: WTFs/minute

(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics
Performance

- Make sure the code works **before** tuning for performance
- Identify what, if anything, needs optimization
- Be careful, tuning for performance usually decreases the readability and maintainability of code
Testing

- Test at every possible level
  - Individual code units
  - Local systems
  - Full system integration
  - Stress tests for scalability
  - Interactive (both competent and new user)

- Automated testing
  - Aspire to full case coverage
  - Push for 100% code coverage

- Performance
Questions?