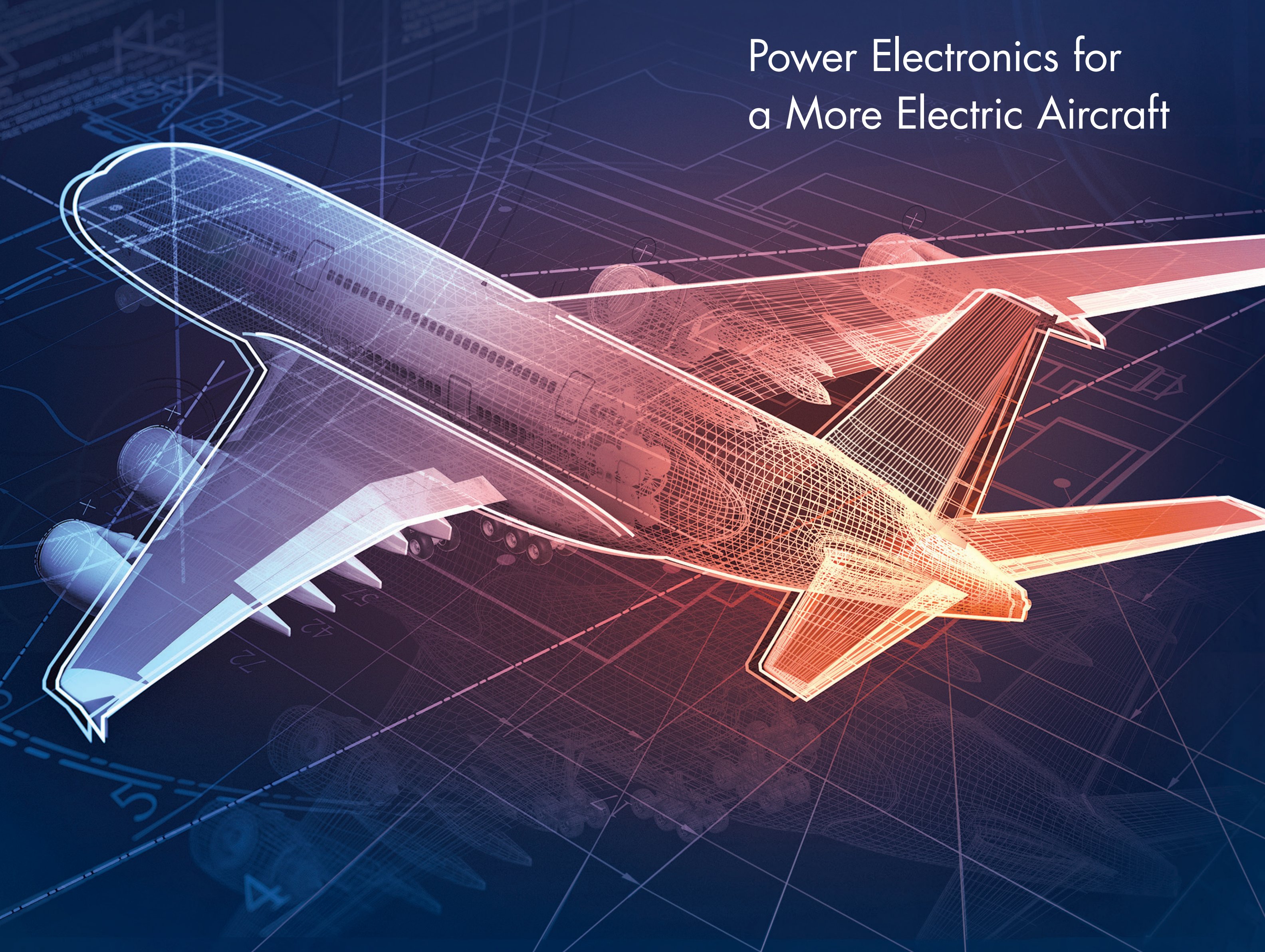
MathWorks News&Notes

The Magazine for the MATLAB® and Simulink® Community



Power Electronics for a More Electric Aircraft

MathWorks®

MATLAB SPEAKS WIRELESS DESIGN

You can simulate, prototype, and verify wireless systems right in MATLAB. Learn how today's MATLAB supports RF, LTE, WLAN, and 5G development and SDR hardware.

mathworks.com/wireless

MATLAB CENTRAL TURNS FIFTEEN!

Tap into the knowledge and experience of thousands of community members. Find or contribute downloadable code, technical tips, and answers.

2000
Problem sets and games on Cody

25,000
Downloadable files on File Exchange

120,000
Questions answered on MATLAB Answers

3000
Posts on MATLAB and Simulink blogs

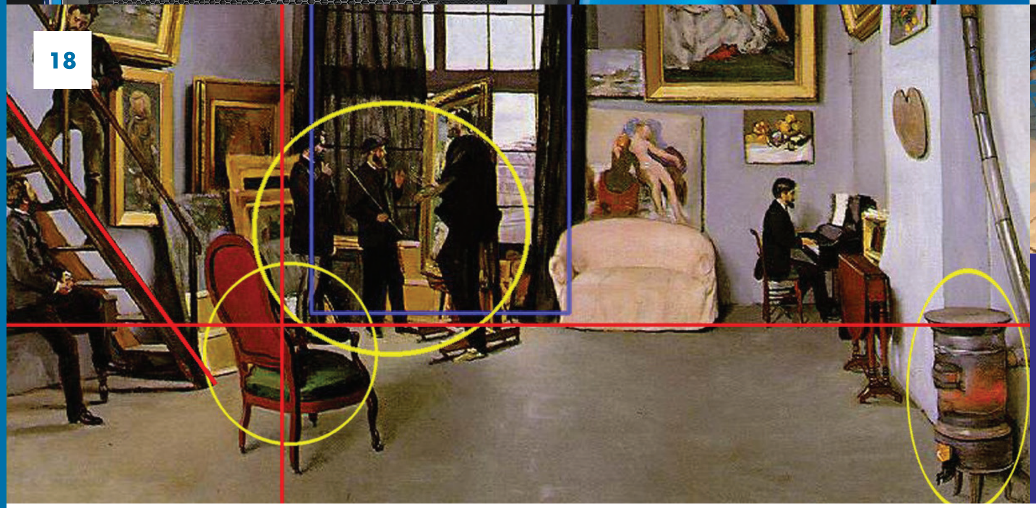
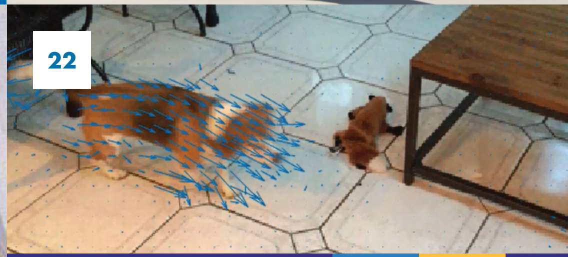
mathworks.com/matlabcentral

MathWorks News&Notes

>> FEATURES

10

14

18

22

30

>> ABOUT THE COVER

The cover shows a rendering of a commercial aircraft, highlighting the THSA (trimmable horizontal stabiliser actuator). On a more electric aircraft, mechanical actuators are replaced with all-electric actuators, reducing weight, improving reliability, and minimizing environmental impact. The article on page 6 describes how engineers in Microsemi's Aviation Center of Excellence developed a power core module (PCM) for a more electric aircraft. The PCM controls the electric motors used in primary flight control actuation and landing gear systems.

>> DEPARTMENTS

Managing Editor
Linda Webb

Editor
Rosemary Oxenford

Art Director
Robert Davison

Graphic Designer
Chris Roth

Technical Writer
Jack Wilber

Production Editor
Julie Cornell

Printer
DS Graphics

Production Staff
L. Bai, J. Mespelli, A. Pollack

Editorial Board
T. Andraczek, S. Gage, C. Hayhurst, M. Hirsch, S. Lehman, D. Lluch, M. Maher, A. May, C. Moler, M. Mulligan, L. Shure, J. Tung

Contributors and Reviewers
J. Abraham, J. Andersson, A. Bergmann, G. Bourdon, M. Carone, B. Chou, K. Cohan, S. DeLand, D. Doherty, A. Elgammal, T. Erkkinen, J. Friedman, T. Gerke, A. Himmeldorf, K. Hyman, W. Jin, T. Jones, J. Kelly, T. Kush, J. Lerche, V. Leung, C. Lin, K. Lorenc, R. Mawrey, T. McKay, S. Miller, S. Myschik, A. Nehemiah, S. O'Donnell, B. Patel, P. Pilotte, S. Prasanna, R. Purser, G. Reith, R. Rovner, G. Sandmann, H. Scharler, F. Smith, B. Tannenbaum, A. Thé, G. Thomas, S. Velilla, M. Vetsch, E. Wetjen

Subscribe
mathworks.com/subscribe

Comments
mathworks.com/contact

Find Us Online

MathWorks®

Physical Modeling

Engineers in the automotive, aerospace, manufacturing, and other industries use Simulink® and Simscape™ to assemble system-level models that include the control system and span mechanical, electrical, and other physical domains. By simulating the plant model and the controller in a single environment, they optimize system-level performance and test their designs under scenarios that would be difficult, expensive, or unsafe to test on physical prototypes.

AIRBUS

Developing a fuel management system for the A380 aircraft

The Airbus A380 is the largest commercial aircraft currently in operation. Its 11 fuel tanks enable nonstop flights of more than 8000 miles. Airbus engineers modeled the control logic for the A380's fuel management system in Simulink and Stateflow®. This model defines modes of operation on the ground (including refuel, defuel, and ground transfer) and in flight (including center of gravity control, load alleviation, and fuel jettison). The team developed a parameterized plant model of the tanks, pumps, and valves, and incorporated relays and other elements of the electrical power system with Simscape. Engineers can reconfigure this model to represent fuel systems for any Airbus aircraft.

mathworks.com/airbus

DCNS

Modeling and simulating a helicopter handling system

The SAMAHE® handling system can transfer a 10-ton helicopter between a hangar and a navy ship's flight deck in less than two minutes. It operates safely in waves up to 6 meters high, and can be configured for use on a variety of ships, from corvettes to frigates. Working in Simulink and Simscape, DCNS built a three-dimensional mechanical model of the system model comprising the handling system, helicopter, and ship. DCNS ran more than 1200 simulations for various helicopter mass configurations, center of gravity locations, and positions, as well as ship motion and wind conditions. The simulations enabled them to test the system under conditions that would be difficult or unsafe to test at sea.

mathworks.com/dcns

GasTOPS

Developing propulsion control algorithms for the USS Makin Island

The USS Makin Island is an 850-foot-long, twin-shaft amphibious assault ship in active service with the U.S. Navy. It uses a hybrid-electric propulsion system with gas turbines for high-speed travel and electric motors for low-speed operation. Using Simulink and Simscape, GasTOPS engineers developed models of the propulsion system, including submodels for the hull, propellers, shafting, gearboxes, motors, generators, and gas turbines. For the electrical plant, they modeled six diesel generators, eight transformers, numerous smart breakers, and the distribution system, as well as electric motors and other loads. They performed simulations to evaluate the system response to short circuits, generator failures, and various fault conditions.

mathworks.com/gastops

METSO

Developing a controller for an energy-saving digital hydraulic system for papermaking equipment

In industrial papermaking equipment, the calender rolls must be precisely controlled to ensure smoothness and glossiness in the paper. The pressure at the nip—the line of contact between rolls—must be within 0.2 bar of its set value. Metso developed a digital hydraulic system that consumes 98% less energy and is more reliable than the proportional hydraulic systems traditionally used for nip control. The team used Simulink and Simscape to model the control system, digital valves, the mechanical frame supporting the calender, and other elements of the equipment. By simulating the hydraulic systems and controls with the model, they could quickly explore fault conditions, valve configurations, and other design options.

mathworks.com/metso

SANDIA NATIONAL LABORATORIES

Simulating microgrid and photovoltaic systems

As part of Hawaii's Clean Energy Initiative, a 1.2 megawatt photovoltaic solar farm was installed on Lanai, an island served by Maui Electric Company. Maui Electric partnered with Sandia National Laboratories to evaluate the battery capacity and control systems required for reliable operation. Sandia developed a model of the Lanai microgrid using Simulink and Simscape Power Systems™, and conducted simulations to assess various configuration and control options. Initial estimates for the Lanai system included a 700 kilowatt-hour battery. The Simulink simulations demonstrated that a battery about half that size would be sufficient.

mathworks.com/sandia

VINTECC

PLC system development for a multi-axle harvesting machine

With a 780hp engine driving three independent rear axles and two independent wheels on the front axle, the JPS Mega Star harvester can collect and haul 100 tons of produce in a single load. Vintecc developed the harvester's complex control system. Using Simscape they modeled tire and vehicle body elements; hydraulic pumps, motors, and cylinders; powertrain components; and mechanical linkages. They developed algorithms for each controller using Stateflow charts to manage execution modes and Simulink PID Controller blocks to control the harvester's hydraulic and mechanical systems. To verify the traction control, axle alignment, cruise control, auto-reverse, and other functions, they ran model-in-the-loop simulations of the controller and plant models.

mathworks.com/vintecc

VOLVO CONSTRUCTION EQUIPMENT

Streamlining product development with a real-time, human-in-the-loop simulator

Volvo Construction Equipment's Virtual Machine Simulator (VMS) gives construction machine operators realistic visual, auditory, and motion feedback during simulation, enabling engineers to evaluate new designs before a prototype is built. The Volvo CE team used Simscape to model the physical system, with 3D mechanical models of the boom, arm, and bucket connected to hydraulic circuits containing control and relief valves, a swing motor, and other components. Integrating this model with models of the engine and control system in Simulink resulted in a multidomain model of the complete machine that supported control design tasks and real-time simulations in the VMS.

mathworks.com/volvo

LEARN MORE

Physical Modeling Solutions
mathworks.com/physical-modeling

User Stories
mathworks.com/user-stories

Real-Time Simulation and Testing of Power Electronics on a More Electric Aircraft

By Shane O'Donnell, Microsemi

>> IN TODAY'S AIRCRAFT, HYDRAULIC AND PNEUMATIC ACTUATION systems are increasingly being replaced by electrical systems. Actuators for primary flight control surfaces, as well as actuators in landing gear, braking systems, and fuel delivery systems, are now driven by power electronics. The electric motors that drive these actuators need to be small, light, and inexpensive. They also need to perform reliably for 50,000–150,000 hours of normal flight operation and under a wide range of failure conditions.

To meet these requirements, the Microsemi Aviation Center of Excellence is developing a line of Intelligent Power Solutions™ (IPS) based on a power core module (PCM) designed and tested with MATLAB® and Simulink®. Model-Based Design has enabled us to push our design to the limits because we can simulate failures, optimize performance, and lower risk by conducting real-time reliability tests of motor drive hardware and control software early in the development process.

Modeling the PCM and Running Closed-Loop Simulations

A complete power electrical control unit consists of functions for pulse width modulation (PWM) control, data conversion, and communications; filtering and protection; a three-phase permanent magnet synchronous motor (PMSM) drive; a control module; and a monitoring module (Figure 1). The motor current, motor speed, and actuator position are fed into the monitoring module, and the control module uses this information to direct the PCM to speed the motor up or slow it down. Because this was a new design, we had to develop the PCM without having working versions of the monitoring module or control module available to test it.

We modeled the PCM in Simulink, using Simscape Power Systems™ and Simscape Electronics™ to model the three-phase PMSM drive and electronic components and the control and monitoring modules. We then ran

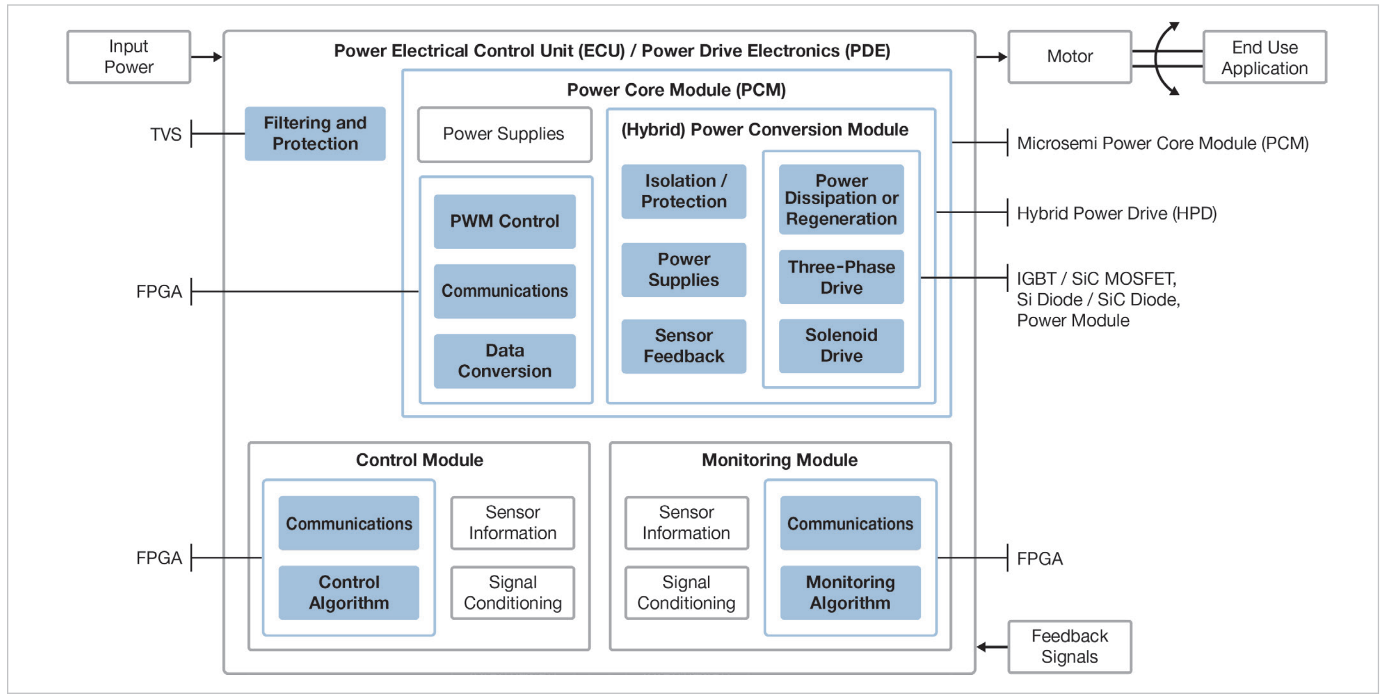


FIGURE 1. *Architectural diagram of the power core module within the larger power electrical control unit.*

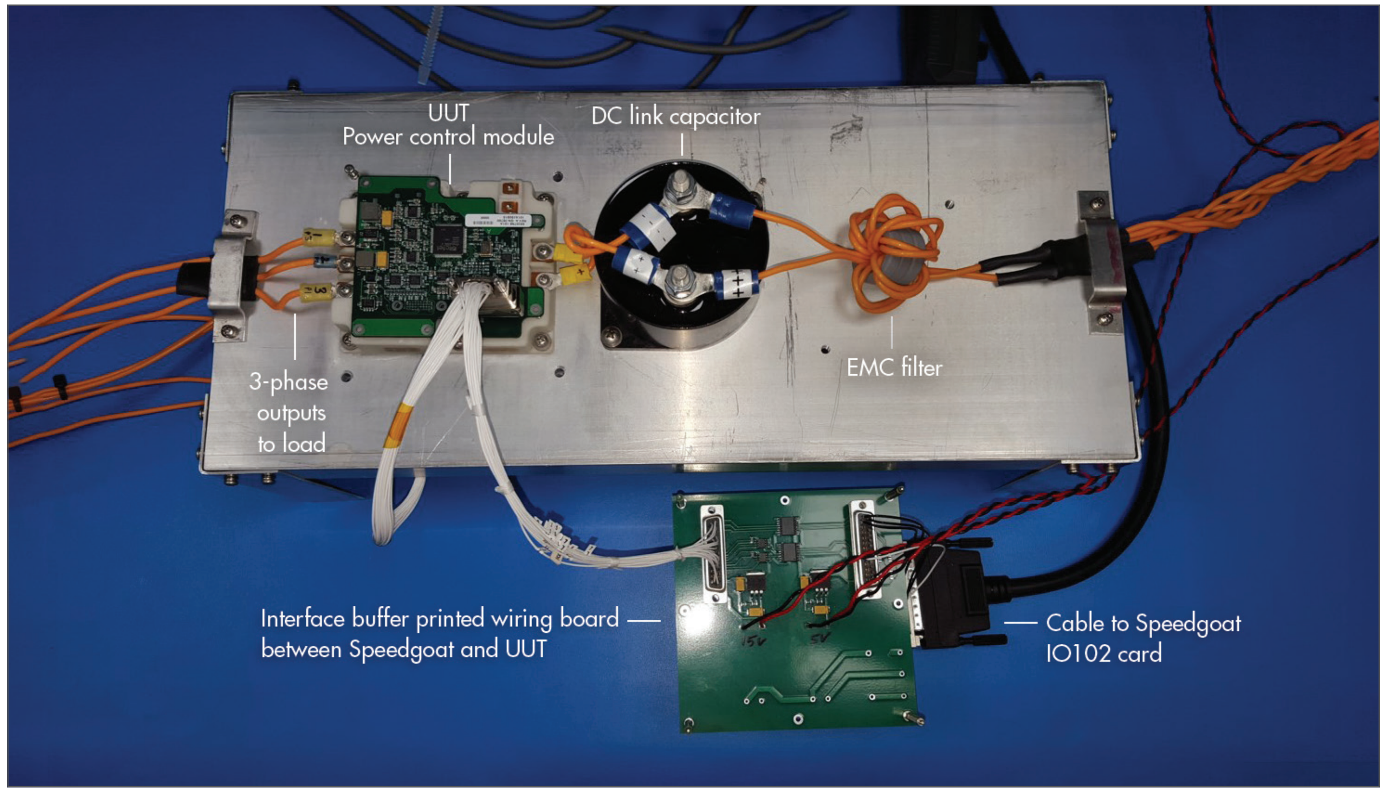

UUT
Power control module
DC link capacitor
3-phase outputs to load
EMC filter
Interface buffer printed wiring board between Speedgoat and UUT
Cable to Speedgoat IO102 card

FIGURE 2. *The Speedgoat setup with prototype PCM hardware.*

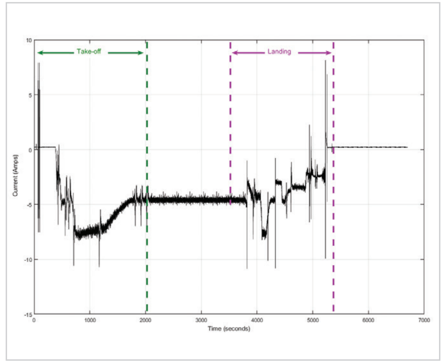


FIGURE 3. *A plot of motor current for a typical flight mission of a single-aisle aircraft.*

closed-loop simulations to characterize the system's electrical and mechanical behavior.

Next, we deployed the three models to a Spartan-6 FPGA in the Speedgoat target system using Simulink Coder™ and Simulink Real-Time™ (Figure 2). The modules communicate through a low-voltage differential signaling (LVDS) interface. In one test setup, both the PCM controller and the other modules were run on the target hardware for real-time tests. In a different setup, we deployed our controller to a production ProASIC3 FPGA on the PCM and ran hardware-in-the-loop tests with the target hardware system performing the functions of the control and monitoring modules. We tested normal operation using both test setups. We also tested the controller's response to several fault conditions to perform failure mode, effects, and criticality analysis.

Testing Real-World Flight Profiles Under Real-World Conditions

To demonstrate the PCM under realistic flight profiles, we developed Simulink and Stateflow® models that translate flight characteristics into electrical and mechanical requirements for an actuation system. As the aircraft proceeds through the typical phases of a flight—taxiing, taking off, climbing, cruising, descending, approaching, and landing—the motor current demands for an aileron actuator, for example, vary significantly. Simulations that we ran using our Simulink and Stateflow mission and flight profile models enabled us to accurately estimate motor current demands for ailerons and other components on specific aircraft (Figure 3).

For our reliability tests, we generated aircraft-specific motor current demands based on the flight profile simulation results. We use environmental chambers that vary the pressure and temperature. For example, the ambient temperature in Boston is much lower than that of Dubai in summer, and our tests must take that into account. With the environmental chambers, we can expose the systems to temperatures of -55° Celsius and pressures of less than 0.2 bar. Long-term reliability tests representative of 150,000 flight hours require careful monitoring and thorough analysis of the results. We conduct this monitoring and data analysis in MATLAB.

What We Learned

Through our extensive modeling and simulations, we established that units equipped with motor drives based on silicon carbide (SiC) MOSFETs operate at a temperature approximately 40° Celsius lower than similar units with IGBTs.

Because active cooling is not possible with today's smaller and lighter hardware designs, managing the temperature of the device while in operation is vital to ensuring that it will function reliably for 150,000 flight hours. Simulations also showed that power dissipation with IGBTs is considerably higher than with SiC MOSFETs (Figure 4). These insights informed our design decisions for the PCM and point to SiC MOSFETS as an enabling technology as the industry moves towards increased fly-by-wire controls in the more electric aircraft (MEA).

Simulink, Simulink Real-Time, and Speedgoat target hardware have enabled us to demonstrate the application-specific reliability of our early designs without installing the units on an actual aircraft. With Model-Based Design, we can do continuous validation and verification without waiting until all aspects of the power electrical control unit are developed.

The feedback we've received from our customers has been very positive. With our real-time simulation results, we are confident that we can meet the PCM's reliability targets as we continue to reduce the unit's size, weight, and cost. ▪

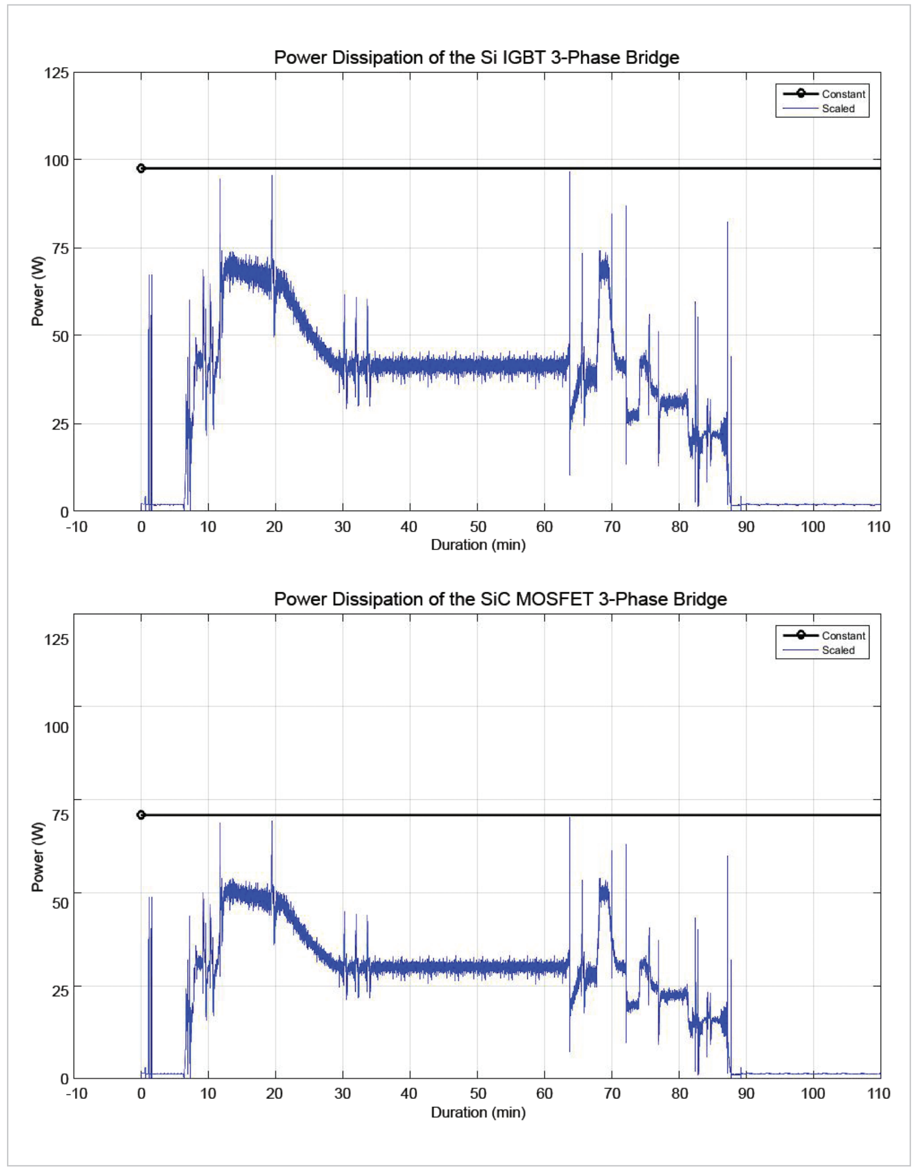


FIGURE 4. *Plots showing power dissipation over time for an IGBT 3-phase bridge (top) and SiC MOSFET 3-phase bridge (bottom).*

LEARN MORE

Real-Time Simulation and Testing with Simulink Real-Time *39:57*
mathworks.com/video-100956

Concurrent Execution with Simulink Real-Time and Multicore Target Hardware
mathworks.com/concurrent-execution

Developing and Verifying Sensor Fusion Methods for Advanced Emergency Braking Systems on Scania Trucks and Buses

By Jonny Andersson, Scania

>> REAR-END COLLISIONS ARE THE MOST COMMON TYPE OF ACCIDENT for freight-carrying trucks and other heavy vehicles. To reduce the risk of rear-end collisions, in 2015 the EU mandated advanced emergency braking systems (AEBS) for all new vehicles.

Like other advanced driver assistance systems (ADAS), an AEBS uses input from sensors to screen the environment. When a collision is imminent, the system warns the driver with an audio alarm. If the driver does not respond, it applies a warning brake. If the driver still does not respond, the system applies the brakes fully to avoid the collision (Figures 1a and 1b). The AEBS also provides "brake assist": When the driver brakes, but with insufficient force to avoid a collision, the system calculates and then applies the required extra braking force.

AEBS uses both radar and camera sensors mounted on the front of the vehicle to scan for objects in the area ahead. The system leverages the particular strengths of each sensor to gain a more precise environment model. Radar sensors excel at determining an object's range, relative velocity, and solidity but are less able to determine its shape or lateral position. A system using radar alone would find it difficult to distinguish a car parked at the side of the road from one in the driver's lane. Cameras, on the other hand, can pinpoint an object's size and lateral position but do not detect range well and are unable to assess density (a dense cloud may be perceived as a solid object).

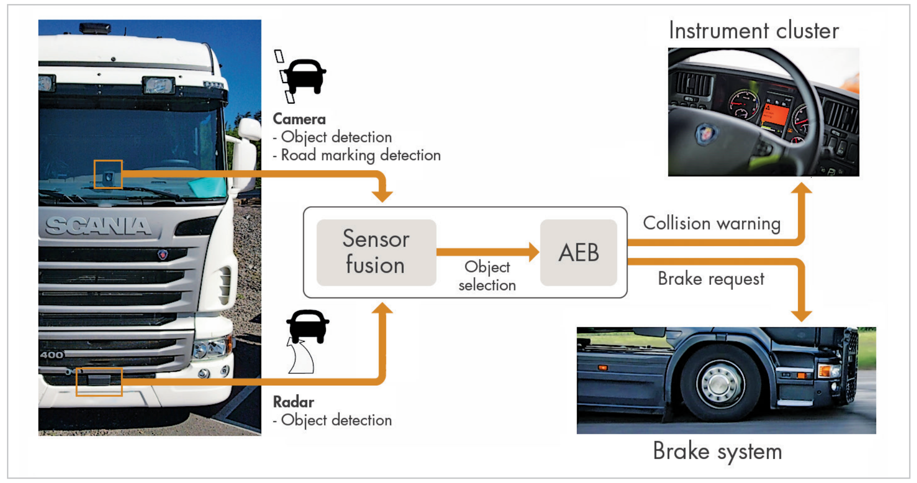


FIGURE 1A. *AEBS overview.*

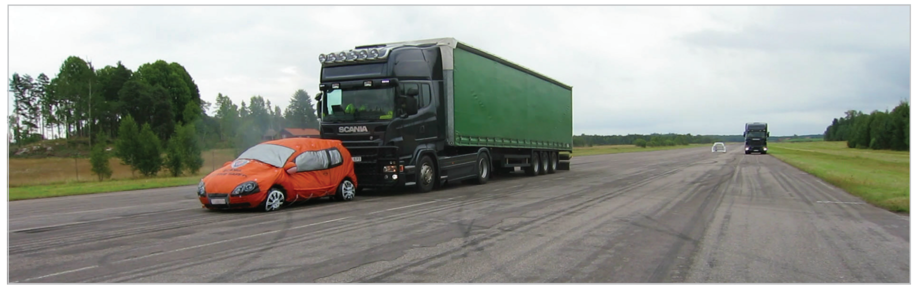


FIGURE 1B. *A typical AEBS scenario: truck with AEBS installed approaching a slow-moving vehicle.*

My colleagues and I built a sensor fusion system that matches and merges data from both sensors into a single object. The system uses four weighted properties—longitudinal speed and position and lateral speed and position—to calculate the probability that both sensors have detected the same object. Once the sensor fusion system has identified an object in the host vehicle's path, it passes the object's position and the vehicle's projected path to the AEBS, which determines when to alert the driver or engage the brakes.

Our group had previously used Model-Based Design to develop an adaptive cruise control system using radar technology, but we had never before developed a sensor fusion system. Because it was a new design, we knew we would need a readable, understandable architecture to visualize signal flow. We also anticipated many design iterations, so we wanted an easy way to visualize results and debug our designs. In addition, we wanted to save time by generating code, but the code had to be efficient, as the CPU load on our electronics control unit (ECU) was already about 60% when we started the sensor fusion project. Lastly, we needed to thoroughly verify our design—our plan was to run simulations based on more than 1.5 million kilometers' worth of sensor data. Model-Based Design met all these requirements.

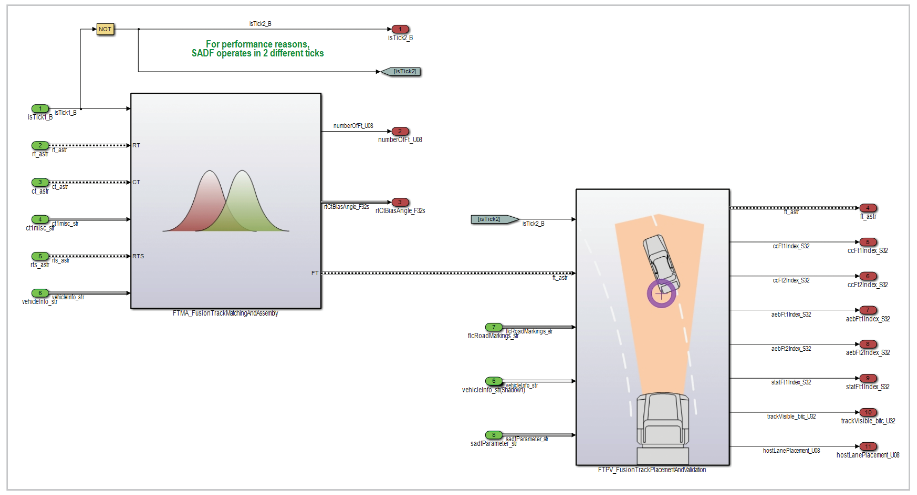


FIGURE 2. *Simulink model of the sensor fusion system showing independent functional blocks.*

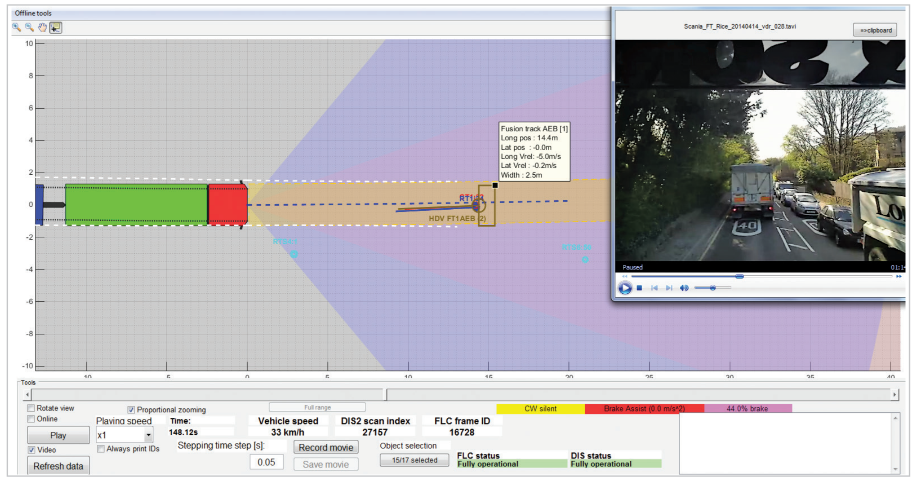


FIGURE 3. *Sensor visualization tool developed in MATLAB.*

Building the Sensor Fusion System

We began by partitioning the system design into functional units, such as object matching and projected path placement, and building a separate Simulink® block for each unit. The result was a clear software architecture with well-defined interfaces (Figure 2). We wrote MATLAB® code for the track association, to compute variances, calculate weighted probabilities, and perform other tasks that are easier to implement with a script than with blocks, and incorporated this code into our Simulink model with MATLAB Function blocks. These algorithm blocks made it easy for team members to merge their algorithms and integrate them with the control system.

To debug and refine our initial design, we ran simulations using recorded radar sensor data, corresponding camera images, and other vehicle sensor data. During debugging we found it useful to visualize the sensor data alongside a camera view from the front of the vehicle. We built a visualization tool in MATLAB that displays sensor fusion data synchronized with a web camera view of the surrounding traffic (Figure 3). Taking advantage of the object-oriented programming capabilities of MATLAB, the tool uses a MATLAB class to represent each object detected by any sensor and the unified object perceived by the sensor fusion system. These MATLAB objects enabled us to quickly step forward and backward in time as we visualized the data.

We used the same tool during road tests to visualize live data coming in from the vehicle network (Figure 4).

Implementing the System and Optimizing Performance

To deploy the sensor fusion system to the ECU, we generated C code from our Simulink model with Embedded Coder®. With code generation, we were able to get to an implementation quickly, as well as avoid coding errors. Most of the ECU processor's resources were allocated to maintenance functions—monitoring dashboard alerts, physical estimations, data gateway, adaptive cruise control, and so on. As a result, we needed to optimize our initial design to increase its efficiency.

In order to get the most performance out of the generated code, we worked with the MathWorks pilot team, who helped us optimize the code generated from MATLAB Coder. To further reduce the processing load we divided the model into separate parts that were executed on alternating cycles. For example, instead of running calculations for stationary and moving objects on every cycle, we ran them on alternating cycles. We realized that the processor was bogged down by the trigonometric functions our system was calling. To alleviate this problem, we wrote trigonometric approximation functions in C and called them from a MATLAB Function block. These modifications not only increased the efficiency of the sensor fusion code, they also enabled the AEBS software to react faster, which is vital when vehicles are traveling at highway speeds and every millisecond counts.

Verifying and Refining the Design

We tested the design in-vehicle on a closed course, but we needed to know how the system would react in real-world driving scenarios, such as different weather conditions, traffic patterns, and driver behaviors. It would be impractical as well as unsafe to test the AEBS directly under these conditions. Instead, we used a simulation-based workflow. We began by gathering data from a fleet of trucks. We decided to collect all data available on the ECU—not just data from the radar and camera used for sensor fusion—as well as images from a separate reference camera.

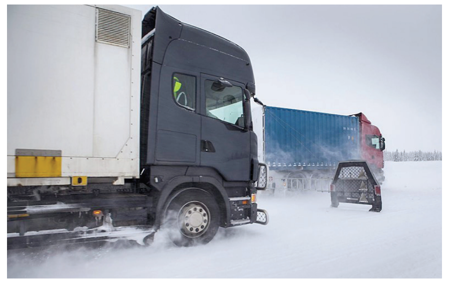


FIGURE 4. *A controlled road test of the AEBS software. The trapezoidal object between the two vehicles is a "soft target," designed to resemble a vehicle, that is used to "fool" the radar and the camera.*

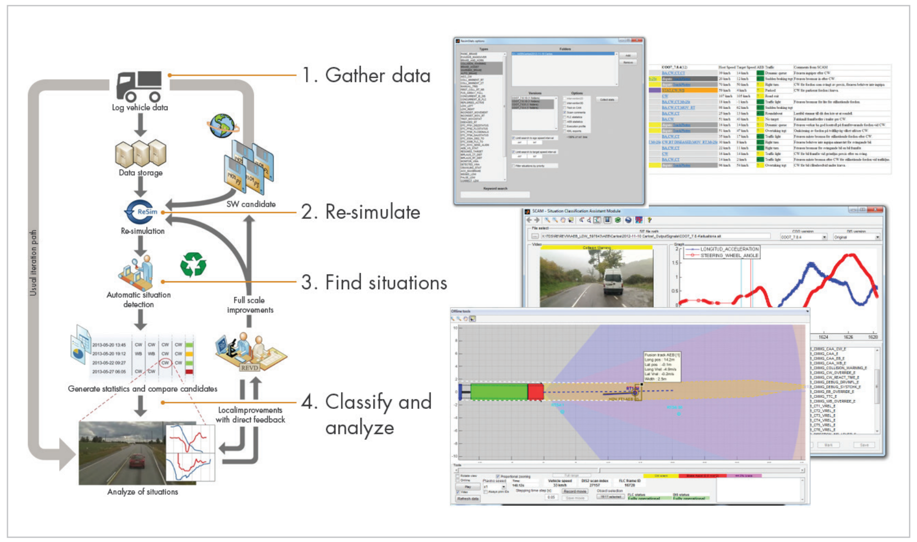


FIGURE 5. *The Situation Classification Assistant Module, a MATLAB based tool for processing logged ECU data and automatically identifying situations relevant to emergency braking.*

Using this fleet test data we ran simulations to identify interesting driving scenarios—scenarios in which the AEBS intervened to warn the driver or engage the brakes, and scenarios in which the system could have intervened but did not—for example, when the driver pressed the horn and braked simultaneously, swerved, or braked sharply. Focusing on these scenarios, we then analyzed the performance of the AEBS to identify areas in which we could improve the design.

We needed to resimulate every time we updated the AEBS software. However, with more than 80 terabytes of real traffic data logged over more than 1.5 million of kilometers of driving, it took several days to run a single simulation.

To accelerate the simulations, we built an emulator using code generated from our Simulink models with Embedded Coder. The emulator reads and writes the same MAT-files as our Simulink model but runs simulations 150 times faster. To further speed up simulations, we wrote MATLAB scripts that run simulations on multiple computers in our department as well as on dedicated multiprocessor servers, where we ran up to 300 simulations in parallel. With this setup, we cut the time needed to simulate all 1.5 million kilometers to just 12 hours. When we identified a new interesting scenario in the emulator, we reran the simulation in Simulink to analyze it in depth.

Identifying and classifying potentially interesting scenarios in terabytes of data was a tedious and time-consuming task, so we developed the Situation Classification Assistant Module, a MATLAB based tool that automates that part of the process (Figure 5). The tool generated a list of events from the simulations, such as collision warnings, warning brakes, and full brakes initiated by the system, as well as hard brakes and sharp turns initiated by the driver. We could then compare these lists for any two versions of our software.

The ability to perform extensive simulations enhanced the robustness and safety of the AEBS function and production code implementation for the ECU. It also enabled us to make changes more quickly. We had confidence in those changes because we were using all the available data in our simulations to test thousands of scenarios.

Deploying the Generated Code in Production ADAS

Most Scania trucks and buses are now equipped with AEBS running production code generated from Simulink models and verified via extensive simulations. We have reused our sensor fusion system design in Scania's adaptive cruise control system, and there are now more than 100,000 units on the road. ■

LEARN MORE

Radar System Modeling and Simulation for Automotive Advanced Driver Assistance Systems *26:00*
mathworks.com/video-107121

Cost and Benefit of Model-Based Development of Automotive Software—Results of a Global Study *30:30*
mathworks.com/video-92771

Model-Based Approach to Resource-Efficient Object Fusion for an Autonomous Braking System *21:21*
mathworks.com/video-108103

HS Bochum Students Design and Build a Motor Controller for an E-Longboard with Model-Based Design

By Dr. Arno Bergmann, Bochum University of Applied Sciences (HS Bochum)

>> WHEN I SAW MY GRADUATE STUDENTS COMPETING TO SEE WHO could maintain the highest average speed on the electric motor-powered skateboard they had built, I knew that I had achieved my two most important goals for their project. Not only had they gained a deep understanding of Model-Based Design, they also had a great deal of fun doing so.

Kevin Leiffels and Raphael-David Volmering designed and built the e-longboard as a final project for my course on electric drives and field-oriented control (vector control). Powered by two independent brushless DC motors (BLDCs), the longboard can carry a rider up to 25 km (15.5 miles) with a top speed of more than 40 km/hour (25 mph).

The Value of Hands-On Projects with Model-Based Design

Students acquire a much deeper understanding of engineering concepts by completing hands-on projects than by listening to a lecture. Even students who get excellent grades on tests don't truly understand concepts such as field-oriented control for electric motors until they have applied those concepts in the real world.

Model-Based Design enables students to tackle meaningful projects in the limited time available. For example, in a single semester Raphael completed the e-longboard's printed circuit board design while Kevin designed, implemented, and tested the controller. Kevin generated more than 15,000 lines of code—much more code than he could have written by hand in one semester.

HS Bochum's acquisition of a Total Academic Headcount (TAH) license was a major milestone for the university and a boon to my course. The students are free to use MATLAB® and Simulink® on assignments both inside and outside the lab. This flexibility is highly motivating. MATLAB and Simulink are industry standard tools, and the students know that to develop the skills required in industry they need more practice with the tools than they can get just by working in the lab. The TAH license also makes my job easier because I no longer have to keep track of individual licenses.

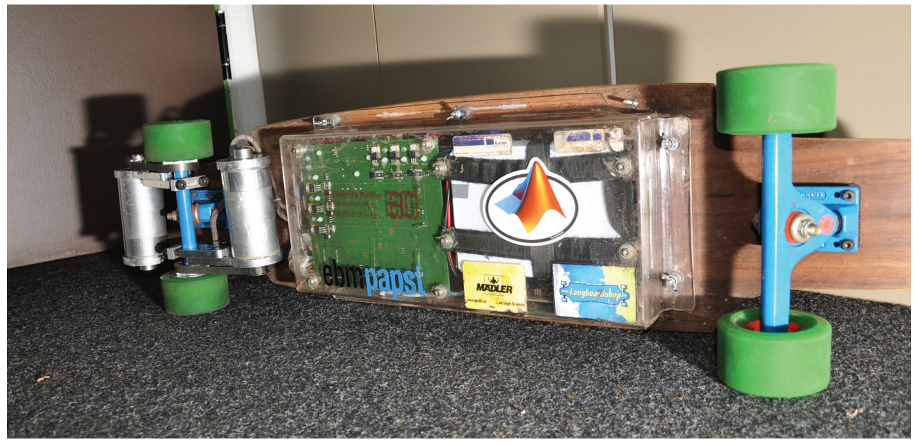


FIGURE 1. *The underside of the e-longboard showing two BLDC motors on the left.*

Establishing Project Requirements

For all student projects I establish a set of basic requirements and then let the students come up with their own ideas. The e-longboard project had to include a power device, two separate motors that are not mechanically connected, and a DSP on which the field-oriented control is implemented (Figure 1). The control part of the project must be challenging but simple enough for a student to complete in a single semester. Each student must use Model-Based Design. In my view, Model-Based Design is the state of the art for control system development because it enables early and thorough verification of the design, low implementation costs, portability to multiple hardware platforms, and short development times.

Before Model-Based Design became a requirement, the students did not learn as much about how real engineering projects are conducted. For example, they would often start with a prepackaged third-party motor controller, hack a few lines of C code together to get the motor spinning, and then move directly to constructing the rest of the system for trial-and-error testing. When they were done they would find out that the system did not meet their power demands or satisfy all the real-time requirements. Because they had not verified their designs via simulation, they only discovered these

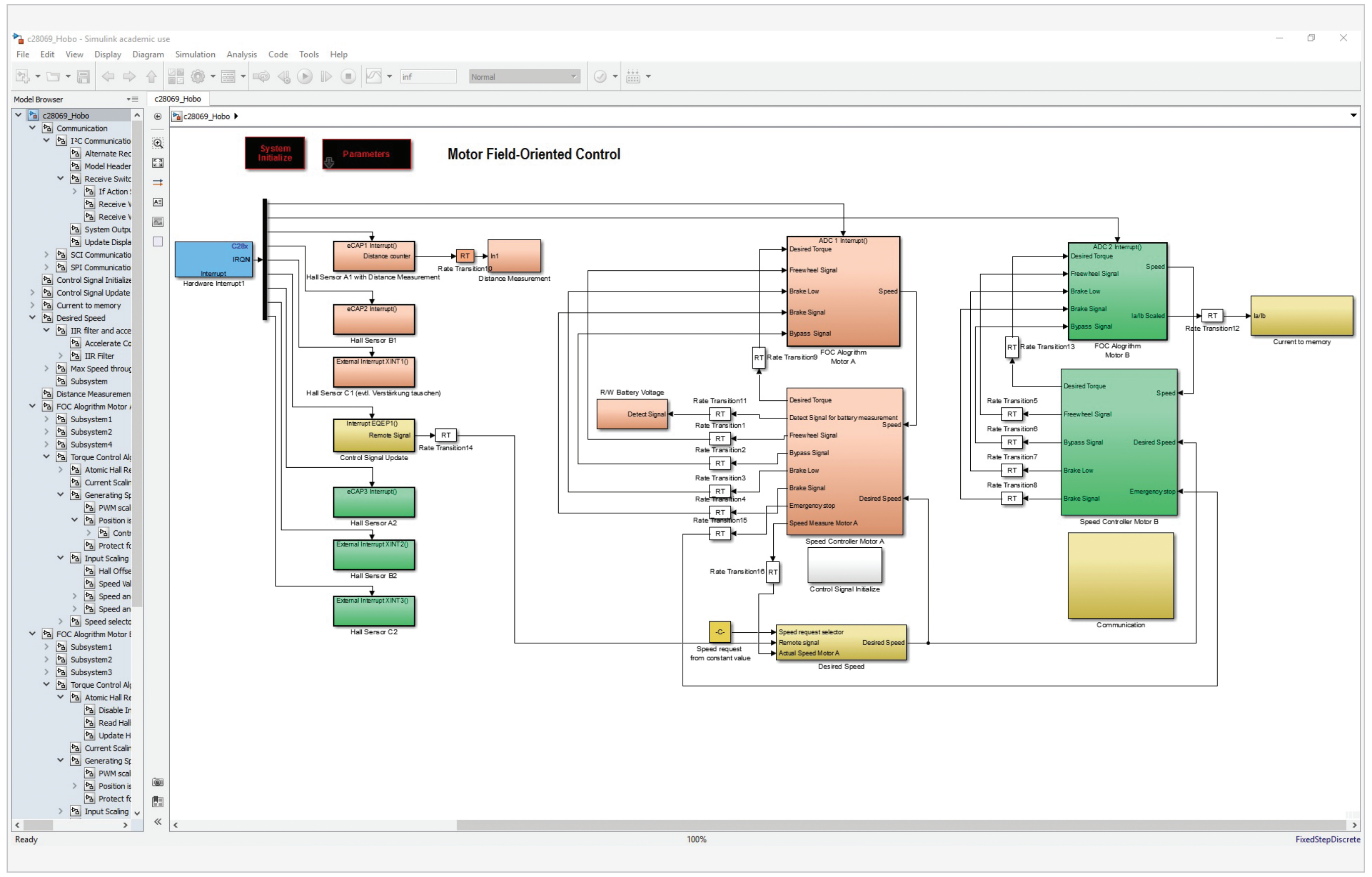


FIGURE 2. *Simulink model for simulating field-oriented control with a permanent magnet synchronous machine.*

problems when it was too late to do anything about them.

Supplementing the basic project requirements, the students included several requirements specific to the e-longboard. In addition to specifying a minimum range for a single battery charge, these requirements defined a maximum braking distance and a minimum hill grade that the e-longboard would need to be able to climb.

Designing and Implementing the Controller

A principal design challenge in field-oriented control is maintaining a 90° angle between the rotor and stator field in the motor. In addition to minimizing changes of the magnetic flux to enable fast transient responses, maintaining this angle maximizes motor torque for a given current. Kevin used Hall sensors to measure rotor position, which is a key input to both the field-oriented control and the board's speed control loop.

Kevin based his controller design on an example field-oriented control project from a MathWorks webinar. The example included a Simulink model for controlling the speed and torque of a three-phase permanent magnet synchronous machine (PMSM), which was modeled using Simscape Power Systems™ (Figure 2). We found the example to be a good implementation of field-oriented control.

After downloading the example project from mathworks.com, Kevin modified the parameters for the e-longboard, removed unneeded parts, and added features. After running simulations in Simulink, Kevin used Embedded Coder® to generate C code for the board's TI F28069 microcontroller. At that point, he began evaluating the real-time response of the system to see if it met the real-time requirements he had established.

Kevin and Raphael worked largely on their own, meeting me once a week so that I could monitor their progress. On this project, Kevin applied and expanded the basic knowledge of Model-Based Design with MATLAB and Simulink that he had acquired in earlier control design electives at HS Bochum. He relied on technical support from MathWorks to resolve any technical issues that he encountered, and that enabled him to work with surprisingly little assistance from me.

Once the printed circuit board was ready (Figure 3) and the rest of the e-longboard had been constructed, Kevin and his classmates began testing the board in and around the HS Bochum campus.

Kevin rode around a nearby lake—a distance of more than 25 km—to test the board's range. It wasn't long before the students were competing to see who could deplete the batteries fastest by maintaining the top average speed. Following test runs, Kevin post-processed metrics captured during the runs. To visualize the e-longboard's speed response, for example, he created a combined graph of desired speed and actual speed in MATLAB (Figure 4).

Next Steps for the Student and the Course

When Kevin completed his studies at HS Bochum he began working for an engineering company that uses Model-Based Design. When the company learned of Kevin's success with the e-longboard they hired him on the spot.

One of the most valuable lessons Kevin learned is the importance of verifying requirements as thoroughly as possible via modeling and simulation before the actual implementation. The company that he now works for will be supporting our upcoming use of Simulink Verification and Validation™ because they, too, are seeking to verify requirements at earlier stages in development.

I am planning a few changes for the next group of students taking the field-oriented controls course. I will still require hands-on projects with Model-Based Design, and I will be encouraging more students to build an enhanced version of the e-longboard. Next year's version of the course will place a stronger emphasis on requirements, logical modes, and physical modeling using Simulink Verification and Validation, Stateflow®, and Simscape Power Systems, respectively.

I plan to use the board to inspire next year's students. I will bring the board to class and let the students take a test drive outside to stir their interest before they head to the lab to begin their own projects using Model-Based Design and field-oriented control. ▪

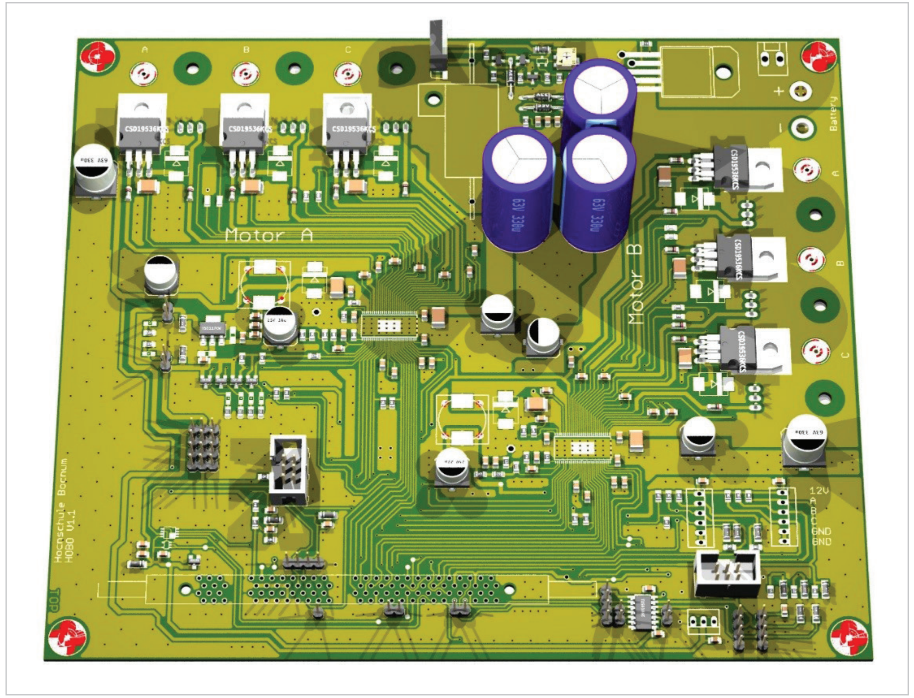


FIGURE 3. *The e-longboard printed circuit.*

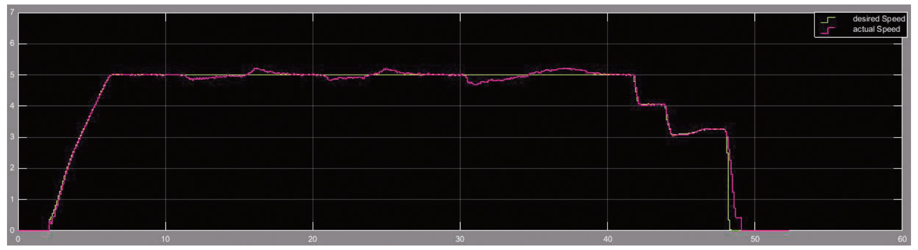


FIGURE 4. *Plot of desired speed (pink) and actual speed (yellow) showing how closely the two matched during a test run.*

LEARN MORE

Simulation of FOC Using PMSM Model
mathworks.com/foc-pmsm

MATLAB and Simulink in the World: The MATLAB Enabled Campus
mathworks.com/miw-tah

```
idl' * vis1;
d2' * vis2;
entum * deltaW + (eta / batch_size) * ((posprods - negprods)' -
/    = momentum * deltaBias_upW    + (eta / batch_size) * (sum(hid
vnW = momentum * deltaBias_downW + (eta / batch_size) * (sum(vis

% Compute probabilities for hidd
if sigmoid
    hid1 = 1 ./ (1 + exp(-(vis1
else
    hid1 = vis1 * W + repmat(bia
end
```

Creating Computer Vision and Machine Learning Algorithms That Can Analyze Works of Art

By Ahmed Elgammal, Rutgers University

>> WHEN YOU STUDY A PAINTING, CHANCES ARE THAT YOU CAN make several inferences about it. In addition to understanding the subject matter, for example, you may be able to classify it by period, style, and artist. Could a computer algorithm "understand" a painting well enough to perform these classification tasks as easily as a human being?

My colleagues and I at the Art and Artificial Intelligence Laboratory at Rutgers University explored this question using MATLAB®, Statistics and Machine Learning Toolbox™, and a database of thousands of paintings from the past six centuries. We also addressed two other intriguing questions about the capabilities and limitations of AI algorithms: whether they can identify which paintings have had the greatest influence on later artists, and whether they can measure a painting's creativity using only its visual features.

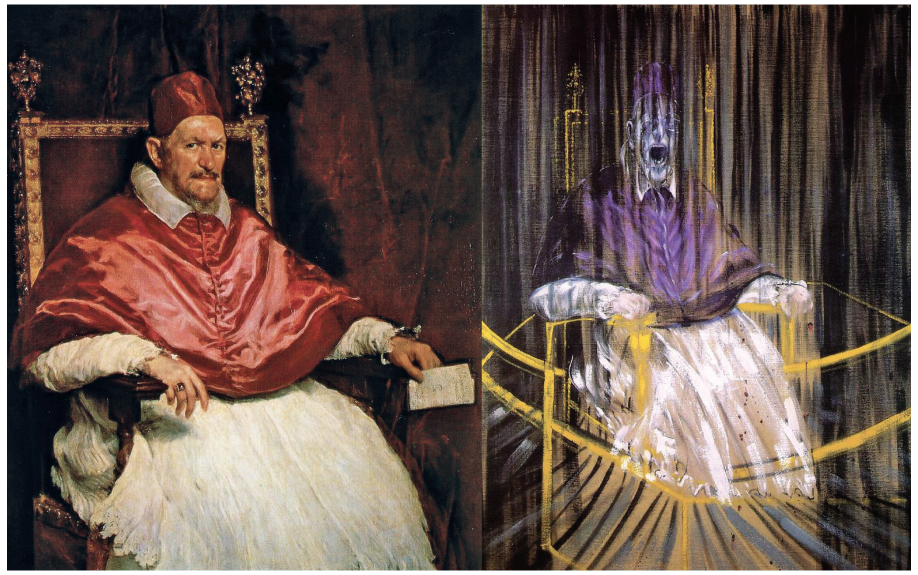


FIGURE 1. *Left: Diego Velázquez's "Portrait of Pope Innocent X." Right: Francis Bacon's "Study After Velázquez's Portrait of Pope Innocent X."*

Extracting Visual Features for Classifying Paintings

We wanted to develop algorithms capable of classifying large groups of paintings by style (for example, as Cubist, Impressionist, Abstract Expressionist, or Baroque), genre (for example, landscape, portrait, or still life), and artist. One requirement for this classification is the ability to recognize color, composition, texture, perspective, subject matter, and other visual features. A second is the ability to select those visual features that best indicate similarities between paintings.

Working with MATLAB and Image Processing Toolbox™, we developed algorithms to extract the visual features of a painting. The feature extraction algorithm is fairly common in computer vision, and straightforward to implement. The more challenging task was finding the best machine learning techniques. We began by testing support vector machines (SVMs) and other classification algorithms in Statistics and Machine Learning Toolbox to identify visual features that are useful in style classification. In MATLAB, we then applied distance metric learning techniques to weight the features and thereby improve the algorithm's ability to classify paintings.

The algorithms we developed classified the styles of paintings in our database with 60% accuracy, where chance performance would have been about 2%. While art historians can perform this task with much more than 60% accuracy, the algorithm outperforms typical non-expert humans.

Using Machine Learning to Uncover Artistic Influences

Once we had algorithms that could reliably identify similarities between pairs of paintings, we were ready to tackle our next challenge: using machine learning to reveal artistic influences. Our hypothesis was that visual features useful for style classification (a supervised learning problem) could also be used to determine influences (an unsupervised problem).

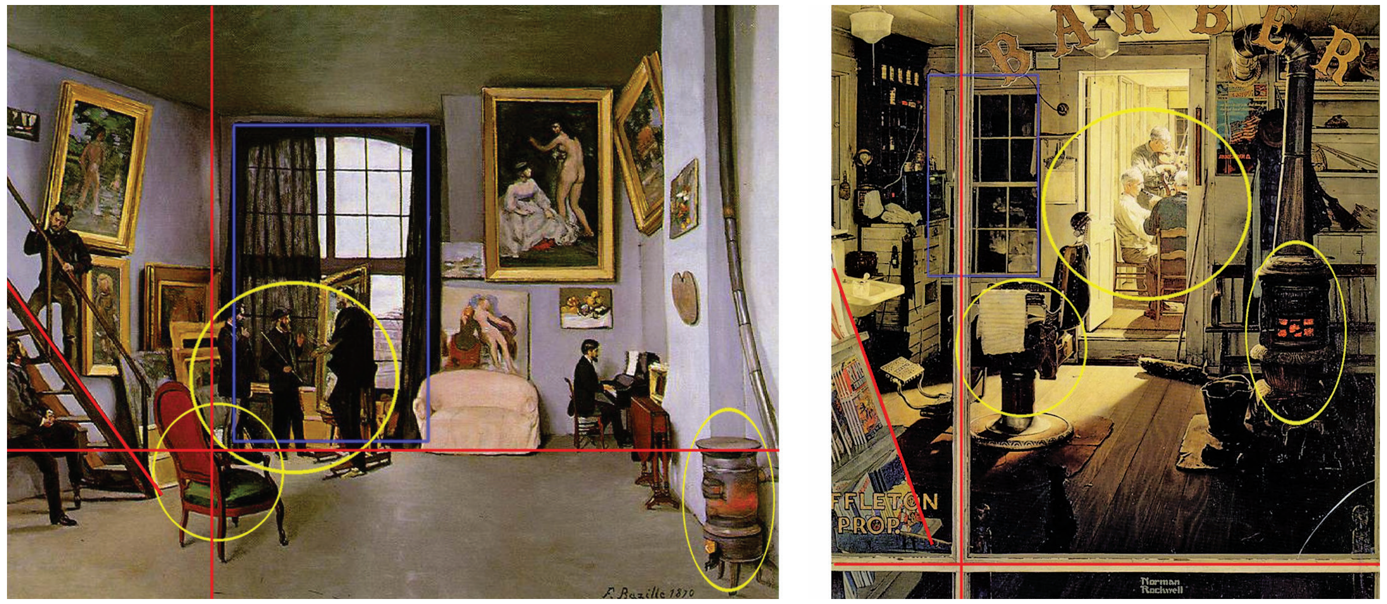


FIGURE 2. *Left: Frederic Bazille's "Bazille's Studio; 9 rue de la Condamine." Right: Norman Rockwell's "Shuffleton's Barbershop." Yellow circles indicate similar objects, red lines indicate similar composition, and the blue rectangle indicates a similar structural element.*

Art historians develop theories of artistic influence based on how the artists worked, traveled, or trained with contemporaries. Our MATLAB based machine learning algorithms used only visual elements and dates of composition. We hypothesized that an algorithm that took into account objects and symbols in the painting would be more effective than one that relied on low-level features such as color and texture. With this in mind, we used classification algorithms that were trained on Google images to identify specific objects.

We tested the algorithms on more than 1700 paintings from 66 different artists working over a span of 550 years. The algorithm readily identified the influence of Diego Velazquez's "Portrait of Pope Innocent X" on Francis Bacon's "Study After Velazquez's Portrait of Pope Innocent X" (Figure 1).

The similarities in composition and subject matter between these two paintings are easy even for a layman to spot, but the algorithm also produced results that surprised the art historians we worked with. For example, our algorithm identified "Bazille's Studio; 9 rue de la Condamine," painted by French Impressionist Frederic Bazille in 1870, as a possible influence on Norman Rockwell's "Shuffleton's Barbershop," completed 80 years later (Figure 2). Although the paintings might not look similar at first glance, a closer examination reveals similarities in composition and subject matter, including the heaters in the lower right of each work, the group of three men in the center, and the chairs and triangular spaces in the lower left.

In our data set, the algorithms correctly identified 60% of the 55 influences recognized by art historians, suggesting that visual similarity alone provides sufficient information for algorithms (and possibly for humans) to determine many influences.

Measuring Creativity by Solving a Network Centrality Problem

Recently, our research has focused on developing algorithms to measure creativity in art. We based this project on a widely used definition that identifies an object as creative if it is both novel and influential. In these terms, a creative painting will be unlike the paintings that came before it (novel), but similar to those that came after it (influential).

In addressing this problem, we once again saw an opportunity to apply our MATLAB algorithms for identifying similarities between paintings. In MATLAB we created a network in which the vertices are paintings and each edge represents the similarity between the two paintings at its vertices. Through a series of transformations on this network we saw that making inferences about creativity from such a graph is a network centrality problem, which can be solved efficiently using MATLAB.

We tested our creativity algorithms on two data sets containing more than 62,000 paintings. The algorithm gave high scores to several works recognized by art historians as both novel and influential, including some of the works shown in Figure 3. Ranking even higher than Pablo Picasso's "Young Ladies of Avignon" (1907) in the same period were several paintings by Kazimir Malevich. This result initially surprised me, as I knew little about Malevich's work. I have since learned that he was the founder of the Suprematism movement, one of the earliest developments in abstract art.

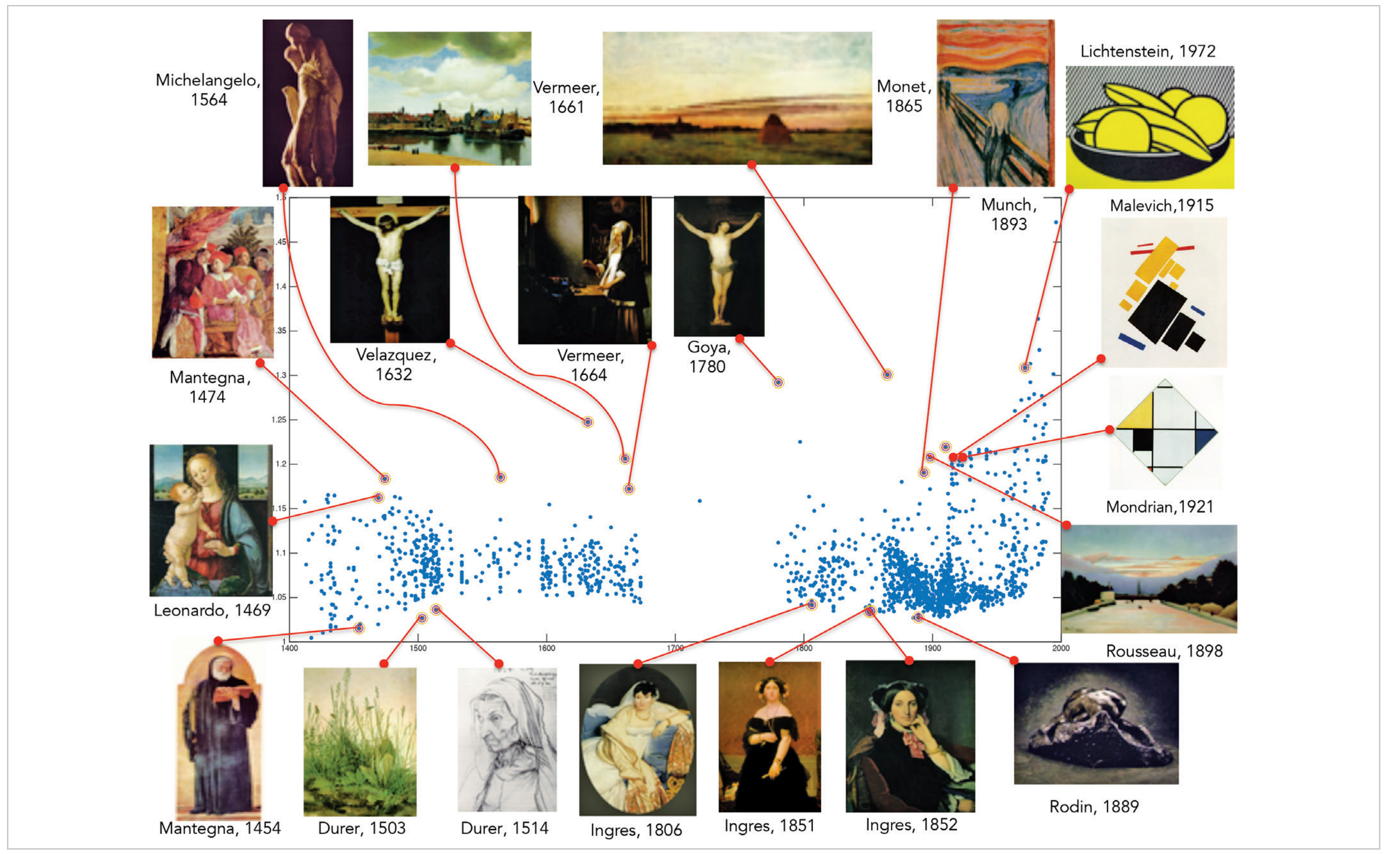


FIGURE 3. *Computed creativity scores (y-axis) for paintings from 1400 to 2000 (x-axis), showing selected highest scoring paintings for individual periods.*

To perform a basic validation of our algorithm, we changed the date on specific works of art, effectively shifting them backwards or forwards in time. In these "time machine" experiments, we saw significant creativity score increases for Impressionist art moved back to the 1600s and significant reductions for Baroque paintings moved forward to the 1900s. The algorithms correctly perceived that what was creative 300 years ago is not creative today, and that something that is creative now would have been much more creative if introduced far in the past.

A Scalable and Extensible Framework for Arts Research

Humans have the innate perceptual skills to classify art, and they excel at identifying similarities in pairs of paintings, but they lack the time and patience to apply these skills objectively to thousands or millions of paintings. Handling tasks at this scale is where computers come into their own. By developing machine learning algorithms that have perceptual capabilities similar to humans, our goal is to provide art historians with tools to navigate vast databases of images.

The framework we developed in MATLAB for identifying similarities and measuring creativity is not confined to art. It could be applied to literature, music, or virtually any other creative domain, as long as the individual works can be encoded in a way that is accessible to the algorithms.

For now, however, our focus remains on the visual arts. We are interested not only in ensuring that machine learning algorithms produce good results but also in how they arrive at those results. In this area, too, MATLAB is a tremendous advantage because it provides many ways to quickly and easily visualize results. These visualizations enable us to understand the results and use them to inform ongoing AI research. ▪

LEARN MORE

Machine Learning with MATLAB *41:26*
mathworks.com/video-81984

Woods Hole Oceanographic Institution Improves Underwater Imaging
mathworks.com/woods-hole

Applying Modern PDE Techniques to Digital Image Restoration
mathworks.com/image-restoration

Deep Learning for Computer Vision with MATLAB

By Avinash Nehemiah and Valerie Leung, MathWorks

>> COMPUTER VISION ENGINEERS HAVE USED MACHINE LEARNING techniques for decades to detect objects of interest in images and to classify or identify categories of objects. They extract features representing points, regions, or objects of interest and then use those features to train a model to classify or learn patterns in the image data.

In traditional machine learning, feature selection is a time-consuming manual process. Feature extraction usually involves processing each image with one or more image processing operations, such as calculating gradient to extract the discriminative information from each image.

Enter deep learning. Deep learning algorithms can learn features, representations, and tasks directly from images, text, and sound, eliminating the need for manual feature selection.

Using a simple object detection and recognition example, this article illustrates how easy it is to use MATLAB® for deep learning, even without extensive knowledge of advanced computer vision algorithms or neural networks.

Getting Started

The goal in this example is to train an algorithm to detect a pet in a video and correctly label the pet as a cat or a dog. We'll be using a convolutional neural network (CNN), a specific type of deep learning algorithm that can both perform classification and extract features from raw images.

To build the object detection and recognition algorithm in MATLAB, all we need is a pre-trained CNN and some dog and cat images. We'll use the CNN to extract discriminative features from the images, and then use a MATLAB app to train a machine learning algorithm to discriminate between cats and dogs.

Importing a CNN Classifier

We begin by downloading a CNN classifier pretrained on ImageNet, a database containing over 1.2 million labeled high-resolution images in 1000 categories. In this example we'll be using the AlexNet architecture.

```
websave('\networks\imagenet-caffe-alex.mat',...
    'http://www.vlfeat.org/matconvnet/models/beta16/
                         imagenet-caffe-alex.mat');
```

We import the network into MATLAB as a `SeriesNetwork` using Neural Network Toolbox, and display the architecture of the CNN. The `SeriesNetwork` object represents the CNN.

```
% Load MatConvNet network into a SeriesNetwork
convnet = helperImportMatConvNet(cnnFullMatFile);

% View the CNN architecture
convnet.Layers
```

We've stored the images in separate `cat` and `dog` folders under a parent called `pet_images`. The advantage of using this folder structure is that the MATLAB `imageDatastore` we create will be able to automatically read and manage image locations and class labels. (`imageDatastore` is a repository for collections of data that are too large to fit in memory.)

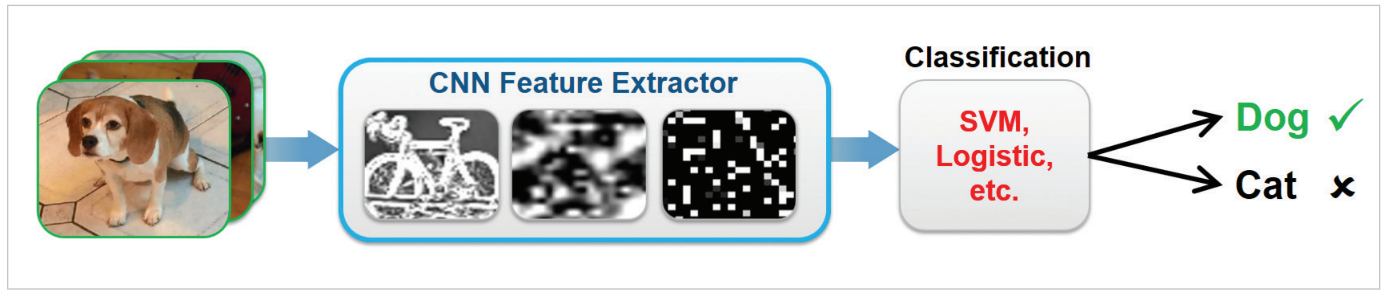


FIGURE 1. *Workflow for using a pre-trained CNN to extract features for a new task.*

We initialize an `imageDatastore` to access the images in MATLAB.

```
%% Set up image data
dataFolder = ' \data\PetImages';
categories = {'Cat', 'Dog'};
imds = imageDatastore(fullfile(dataFolder, ...
    categories), 'LabelSource', 'foldernames');
```

We then select a subset of the data that gives us an equal number of dog and cat images.

```
tbl = countEachLabel(imds)

%% Use the smallest overlap set
minSetCount = min(tbl{:,2});

% Use splitEachLabel method to trim the set.
imds = splitEachLabel(imds, minSetCount, ...
    'randomize');

% Notice that each set now has exactly the same
% number of images.
countEachLabel(imds)
```

Since the `AlexNet` network was trained on 227x227-pixel images, we have to resize all our training images to the same resolution. The following code allows us to read and process images from the `imageDatastore` at the same time.

```
%% Pre-process Images For CNN
% Set the ImageDatastore ReadFcn
imds.ReadFcn = @(filename)readAndPreprocessImage...
    (filename);

%% Divide data into training and testing sets
[trainingSet, testSet] = splitEachLabel(imds, ...
    0.3, 'randomize');
```

We use the `readAndPreprocessImage` function to resize the images to 227x227 pixels.

```
function Iout = readAndPreprocessImage(filename)

I = imread(filename);

% Some images may be grayscale. Replicate the image
% 3 times to create an RGB image.

      if  ismatrix(I)
          I = cat(3,I,I,I);
      end

% Resize the image as required for the CNN.
Iout = imresize(I, [227 227]);

end
```

Performing Feature Extraction

We want to use this new dataset with the pretrained `AlexNet` CNN. CNNs can learn to extract generic features that can be used to train a

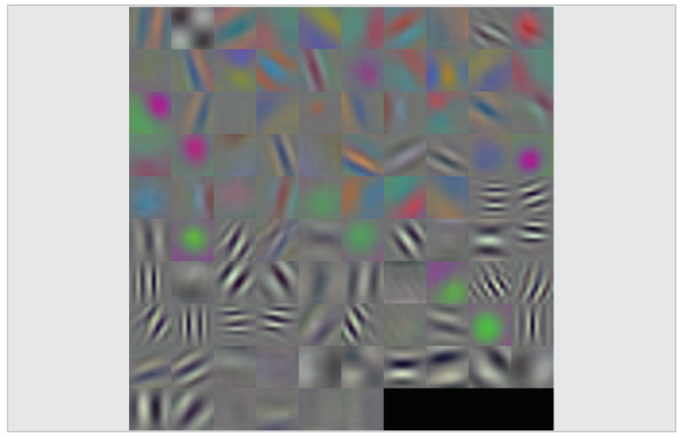


FIGURE 2. *Visualization of first layer filter weights.*

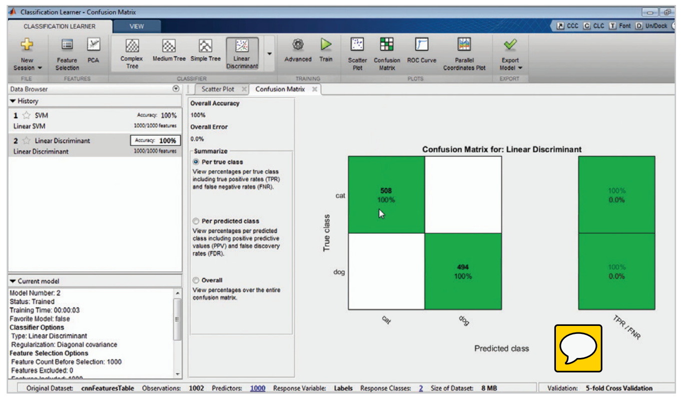


FIGURE 3. *Classification Learner app.*

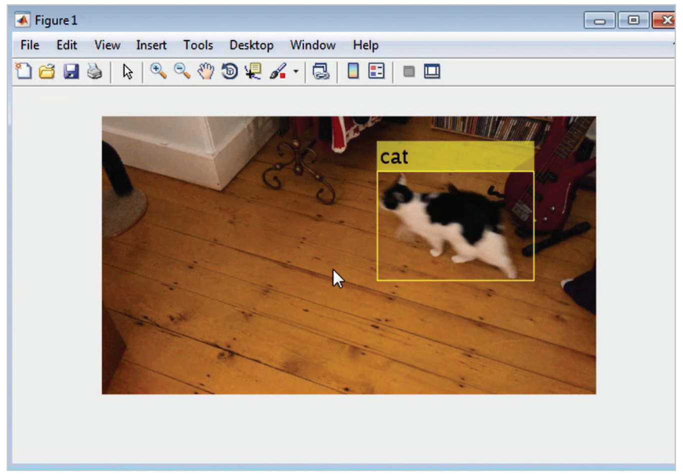


FIGURE 4. *Result of using the trained pet classifier on an image of a cat.*

new classifier to solve a different problem—in our case, classifying cats and dogs (Figure 1).

We pass the training data through the CNN and use the `activations` method to extract features at a particular *layer* in the network. Like other neural networks, CNNs are formed using interconnected layers of nonlinear processing elements, or neurons. Input and output layers connect to input and output signals, and hidden layers provide nonlinear complexity that gives a neural network its computational capacity.

While each layer of a CNN produces a response to an input image, only a few layers are suitable for image feature extraction. There is no exact formula for identifying these layers. The best approach is to simply try a few different layers and see how well they work.

The layers at the beginning of the network capture basic image features, such as edges and blobs. To see this, we visualize the network filter weights from the first convolutional layer (Figure 2).

```
% Get the network weights for the second
% convolutional layer
w1 = convnet.Layers(2).Weights;
% Scale and resize the weights for visualization
w1 = mat2gray(w1);
w1 = imresize(w1,5);
% Display a montage of network weights. There are 96
% individual sets of weights in the first layer.
figure
montage(w1)
title('First convolutional layer weights')
```

Notice that the first layer of the network has learned filters for capturing blob and edge features. These “primitive” features are then processed by deeper network layers, which combine the early features to form higher-level image features. These higher-level features are better suited for recognition tasks because they combine all the primitive features into a richer image representation. You can easily extract features from one of the deeper layers using the `activations` method.

The layer right before the classification layer fc7 is a good place to start. We extract training features using that layer.

```
featureLayer = 'fc7';
trainingFeatures = activations(convnet, ...
    trainingSet, featureLayer, 'MiniBatchSize', ...
    32, 'OutputAs', 'columns');
```

Training an SVM Classifier Using the Extracted Features

We're now ready to train a “shallow” classifier with the features extracted in the previous step. Note that the original network was trained to classify 1000 object categories. The “shallow” classifier will be trained to solve the specific dogs vs. cats problem.

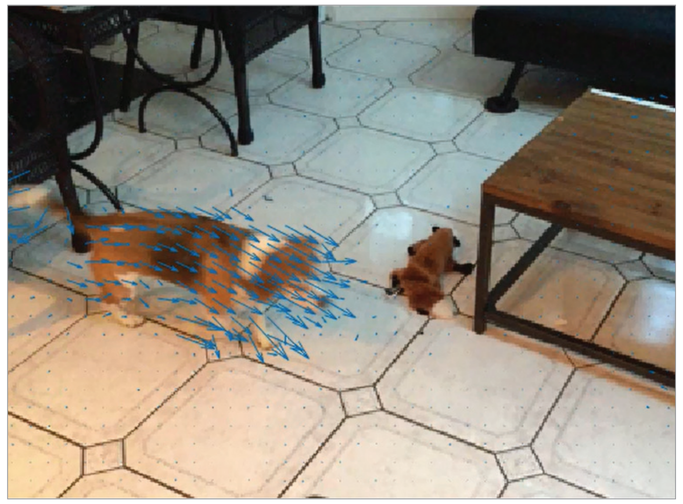


FIGURE 5. *A single frame of video showing the motion vectors overlaid.*

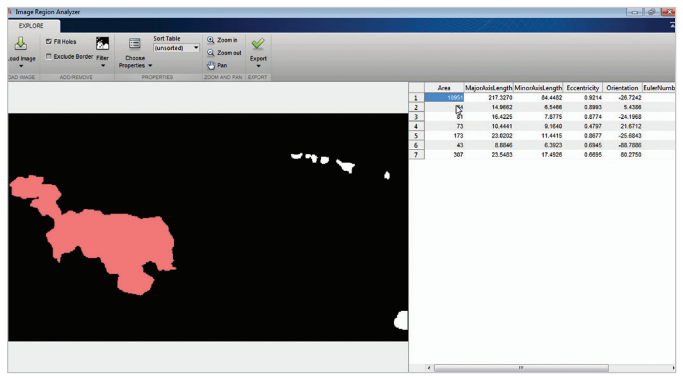


FIGURE 6. *Image Region Analyzer app.*

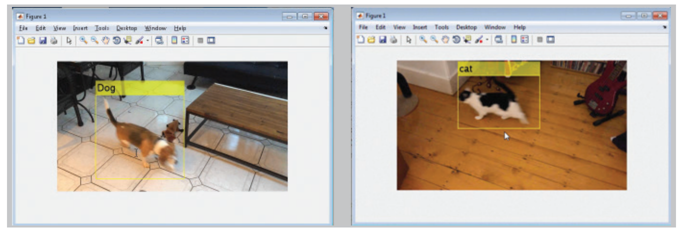


FIGURE 7. *Accurately classified cats and dogs.*

The Classification Learner app in Statistics and Machine Learning Toolbox™ lets us train and compare multiple models interactively (Figure 3). Alternatively, we could train the classifier in our MATLAB script.

We split the data into two sets, one for training and one for testing. Next, we train a support vector machine (SVM) classifier using the extracted features by calling the `fitcsvm` function using `trainingFeatures` as the input or predictors and `trainingLabels` as the output or response values. We will cross-validate the classifier on the test data to determine its *validation accuracy*, an unbiased estimate of how the classifier would perform on new data.

```
%% Train a classifier using extracted features
trainingLabels = trainingSet.Labels;

% Here I train a linear support vector machine
% (SVM) classifier.
svmmdl = fitcsvm(trainingFeatures ,trainingLabels);

% Perform cross-validation and check accuracy
cvmdl = crossval(svmmdl,'KFold',10);
fprintf('kFold CV accuracy: %2.2f\n',...
    1-cvmdl.kfoldLoss)
```

We can now use the `svmmdl` classifier to classify an image as a cat or a dog (Figure 4).

Performing Object Detection

In most images and video frames, there is a lot going on. For example, in addition to a dog, there could be a tree, or a flock of pigeons, or a raccoon chasing the dog. Even a reliable image classifier will only work well if we can locate the object of interest, crop the object, and then feed it to the classifier—in other words, if we can perform object detection.

For object detection we will use a technique called optical flow, which uses the motion of pixels in a video from frame to frame. Figure 5 shows a single frame of video with the motion vectors overlaid.

The next step in the detection process is to separate out pixels that are moving and then use the Image Region Analyzer app to analyze the connected components in the binary image to filter out noise caused by the motion of the camera. The output of the app is a MATLAB function that can locate the pet in the field of view (Figure 6).

We now have all the pieces we need to build a pet detection and recognition system (Figure 7). The system can:

- Detect the location of the pet in new images using optical flow
- Crop the pet from the image and extract features using a pretrained CNN
- Classify the features using the SVM classifier we trained to determine if the pet is a cat or a dog

In this article we used an existing deep learning network to solve a different task. You can use the same techniques to solve your own image classification problem—for example, classifying types of cars in videos for traffic flow analysis, identifying tumors in mass spectrometry data for cancer research, or identifying individuals by their facial features for security systems. ▪

LEARN MORE

Deep Learning for Computer Vision with MATLAB *36:50*
mathworks.com/video-116080

Object Detection Example Code (download)
mathworks.com/fx-57116

The Joy of Generating C Code from MATLAB

By Bill Chou, MathWorks

>> ENGINEERS HAVE TRANSLATED LOW-LEVEL LANGUAGES LIKE C into machine code for decades using compilers. But is it possible to translate a high-level language like MATLAB® to C using coders? Most engineers would agree that it's possible in theory—but does it work in practice? Is the generated code readable or spaghetti? Efficient or bloated? Fast or slow? And does it support industrial workflows, or just R&D?

This article addresses these concerns head-on. It provides tips and best practices for working with MATLAB Coder™, as well as industry examples of successful applications of generated code by companies such as Delphi, Baker Hughes, iSonea, and dorsaVi.

Comparing MATLAB and C Code: A Multiplication Example

The simple MATLAB function below multiplies two inputs.

```
function c = myMult(a, b)

% Multiply two inputs
c = a * b;
```

Given scalar inputs, MATLAB Coder generates the following C code:

```
#include "myMult.h"

double myMult(double a, double b)
{
  return a * b;
}
```

As you can see, the generated code maps clearly back to the MATLAB code.

The same piece of MATLAB code, when given two matrix inputs, generates three nested **for**-loops in C:

```
#include "myMult.h"

void myMult(const double a[12],
const double b[20], double c[15])
{
  int i0;
  int i1;
  int i2;
  for (i0 = 0; i0 < 3; i0++) {
    for (i1 = 0; i1 < 5; i1++)
    {
      c[i0 + 3 * i1] = 0.0;
      for (i2 = 0; i2 < 4;
      i2++) {
        c[i0 + 3 * i1] +=
        a[i0 + 3 * i2] * b[i2
        + (i1 << 2)];
      }
    }
  }
}
```

Recommended Three-Step Iterative Workflow

The simple function shown above can be implemented in a single step. But for more substantial projects, we recommend a structured approach using a three-step iterative workflow (Figure 1).

1. Prepare your algorithm for code generation. Examine and modify the MATLAB code to introduce implementation considerations needed for low-level C code, and use the MATLAB language and functions that support code generation.
2. Test the MATLAB code's readiness for code generation using default settings. Check for run-time errors by generating and executing a MEX file. If successful, move to the next step. If not, repeat step 1 until you can generate a MEX function.
3. Generate C code or keep the MEX function from step 2. You can iterate on the MATLAB code to optimize either the generated C code (for look and feel, memory, and speed) or the MEX function (for performance).

The MATLAB Coder app guides you through this iterative process while enabling you to stay within the MATLAB environment. It analyzes your MATLAB code to propose data types and sizes for your inputs. It tests whether your MATLAB code is ready for code generation by generating a MEX function, then executes the MEX function to check for run-time errors (Figure 2). Equivalent command-line functions provide the same functionality so you can generate code as part of a script or function.

Implementation Constraints

As you prepare your MATLAB algorithm for code generation, you need to take account of implementation constraints resulting from the differences between MATLAB and C code. These include:

- Memory allocation. In MATLAB, memory allocation is automatic. In C code, memory allocation is manual—it is either allocated statically (using `static`), dynamically (using `malloc`), or on the stack (using local variables).
- Array-based language. MATLAB provides a rich set of array operations that allow concise coding of numerical algorithms. C code requires explicit **for**-loops to express the same algorithms.
- Dynamic typing. MATLAB automatically determines the data types and sizes as your code runs. C requires explicit type

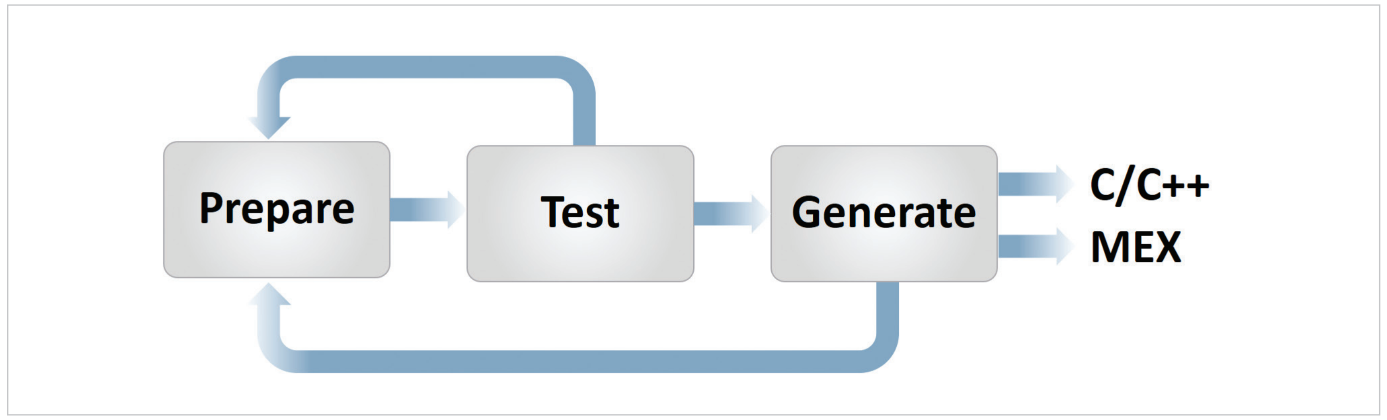
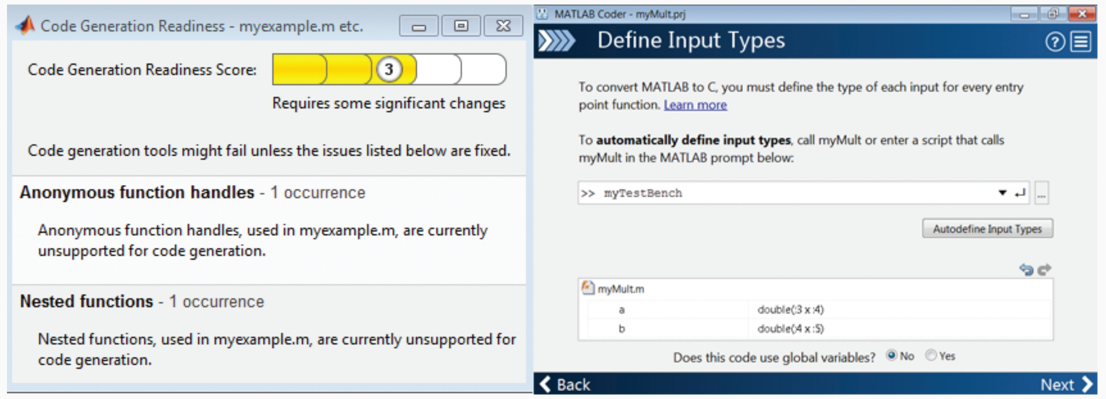


FIGURE 1. *Three-step iterative workflow for generating code.*


Code Generation Readiness - myexample.m etc.
Code Generation Readiness Score:
3
MATLAB Coder - myMult.prj
Define Input Types
Autodefine Input Types
Back
Next

Requires some significant changes

Code generation tools might fail unless the issues listed below are fixed.

Anonymous function handles - 1 occurrence

Anonymous function handles, used in myexample.m, are currently unsupported for code generation.

Nested functions - 1 occurrence

Nested functions, used in myexample.m, are currently unsupported for code generation.

To convert MATLAB to C, you must define the type of each input for every entry point function. Learn more

To **automatically define input types**, call myMult or enter a script that calls myMult in the MATLAB prompt below:

```
>> myTestBench
```

myMult.m	
a	double(:3 x :4)
b	double(:4 x :5)

Does this code use global variables? [✓ No] Yes

FIGURE 2. *Left: Automated checks for features and functions not supported for code generation. Right: Automated analysis and proposal for input data type and sizes.*

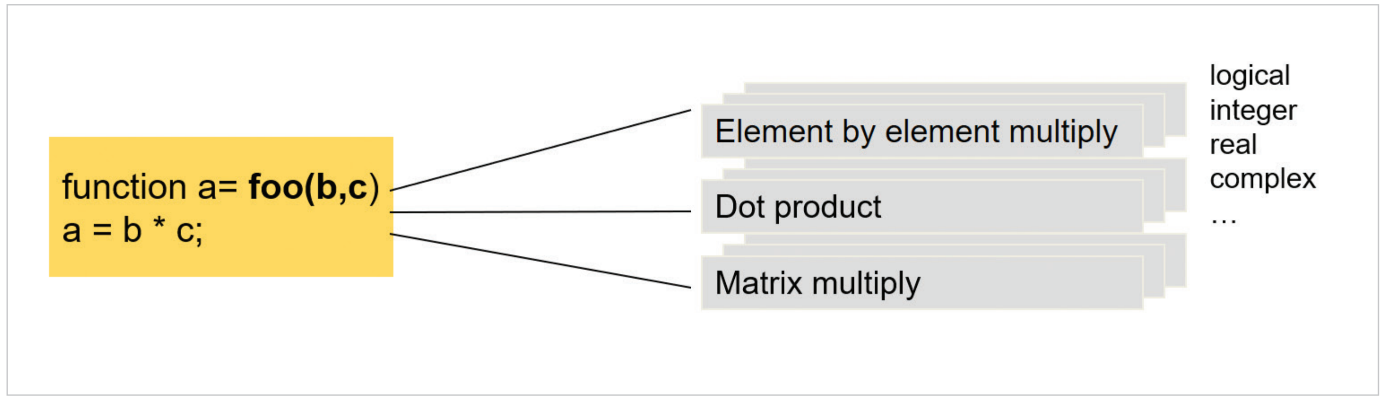


FIGURE 3. *Polymorphism example.*

declarations on all variables and functions.

- Polymorphism. MATLAB functions can support many different input types, while C requires fixed type declarations. At the top level, you must specify the intended C function declaration.

Let's take a closer look at polymorphism. Polymorphism can give a single line of MATLAB code different meanings depending on your inputs. For example, the function shown in Figure 3 could mean scalar multiplication, dot product, or matrix multiplication. In addition, your inputs could be of different data types (logical, integer, floating-point, fixed-point), and they could be real or complex numbers.

MATLAB is a powerful algorithm development environment precisely because you don't need to worry about implementation details as you create algorithms. However, for the equivalent C code, you have to specify what operations mean. For example, the line of MATLAB code shown above could be translated into this single line of C code that returns B*C:

```
double foo(double b, double c)
{
   return b * c;
}
```

Or, it could be translated into 11 lines of C code with three **for**-loops that multiply two matrices:

```
void myMult(const double a[12],
const double b[20], double c[15])
{
  int i0;
  int i1;
  int i2;
  for (i0 = 0; i0 < 3; i0++) {
    for (i1 = 0; i1 < 5; i1++) {
      c[i0 + 3 * i1] = 0.0;
      for (i2 = 0; i2 < 4; i2++) {
        c[i0 + 3 * i1] += a[i0 +
        3 * i2] * b[i2 + (i1 <<
        2)];
      }
    }
  }
}
```

Working with the Generated Code: Four Use Cases

Once you have generated readable and portable C/C++ code from MATLAB algorithms using MATLAB Coder, you have several options for using it. For example:

- Integrate your MATLAB algorithms as source code or libraries into a larger software project such as custom simulators or software packages running on PCs and servers.
- Implement and verify your MATLAB algorithms on embedded processors such as ARM® processors and mobile devices.
- Prototype your MATLAB algorithms as a standalone executable on PCs.
- Accelerate computationally intensive portions of your MATLAB code by generating a MEX function that calls the compiled C/C++ code.

Multicore-Capable Code Generation and Other Optimization Methods

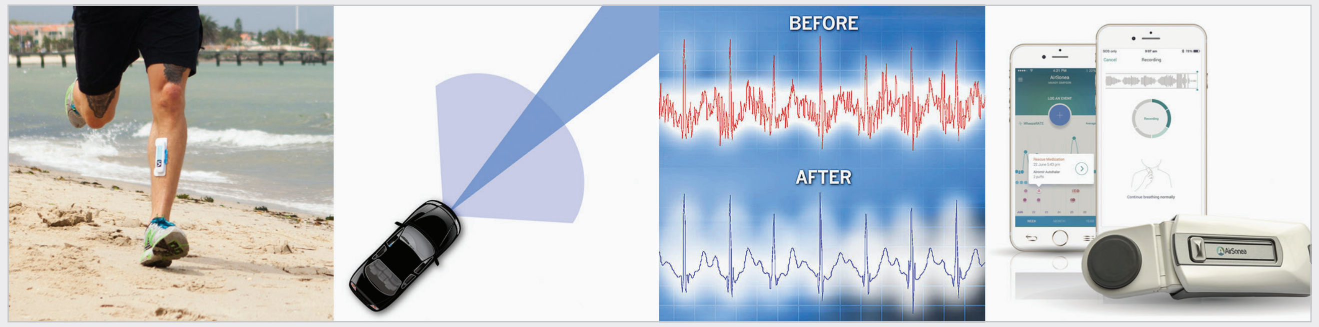
In MATLAB, **for**-loops whose iterations are independent of each other can be run in parallel simply by replacing **for** with **parfor**. MATLAB Coder uses the Open Multiprocessing (OpenMP) application interface to support shared-memory, multicore code generation from **parfor**-loops. OpenMP is supported by many C compilers (for example, Microsoft® Visual Studio® Professional).

```
  for (i1 = 0; i1 < 3; i1++) {
    sz[i1] = (unsigned int)
    originalImage->size[i1];
  }

  N = sz[0];
  M = sz[1];
  normalizer = (L - 1.0) /
  ((double)sz[0] * (double)sz[1]);

#pragma omp parallel for \
 num_threads(omp_get_max_threads()\
 ) private(s,r,planeHist_data,\
loop_ub,i3,y,x,j,d0,u0)

  for (plane = 0; plane < 3;
  plane++) {
    loop_ub = originalHist_size[1];
    for (i3 = 0; i3 < loop_ub;
    i3++) {
      planeHist_data[i3] =
      originalHist_data[plane +
      originalHist_size[0] * i3];
    }
```



Industry Success Stories

- dorsaVi generated C++ code from motion analysis algorithms and compiled it into a DLL, which was then integrated into their C# application running on a PC that analyzes the athlete's movements to diagnose injury.
- Baker Hughes' Dynamics & Telemetry group generated a DLL from sequence prediction algorithms and integrated it into surface decoding software running on a PC that enables downhole data to be decoded quickly and reliably during drilling operations.
- Delphi generated C code for an automotive radar sensor alignment algorithm and compiled it for an ARM10 processor.
- VivaQuant generated fixed-point C code from heart rhythm monitoring algorithms and compiled it for an ARM Cortex-M processor.
- Respiri generated C code from acoustic respiratory monitoring algorithms and compiled it for an iPhone app, an Android™ app, and cloud-based server software.

You can use MATLAB Coder with Embedded Coder® to further optimize code efficiency and customize the generated code. Embedded Coder provides optimizations for fine-grained control of the generated code's functions, files, and data. For example, you can use storage classes to control the declaration and definition of a global variable in the generated code, and use code generation templates to customize banners and comments in the generated code. Embedded Coder also improves code efficiency by using code replacement libraries, which replace certain operators and functions with implementations optimized for popular processors like ARM Cortex®-A and ARM Cortex-M.

Testing the Generated Code

As you develop your MATLAB algorithm, you can create unit tests to verify that the algorithm produces the results you expect. Tests written using the MATLAB unit testing framework can be reused to verify that the generated code behaves the same way as your MATLAB algorithm. With Embedded Coder you can reuse the unit tests in combination with software-in-the-loop (SIL) and processor-in-the-loop (PIL) tests on the generated standalone code or library.

An Automated Workflow

MATLAB Coder enables an automated workflow for translating MATLAB algorithms into C code. With this workflow you spend less time writing and debugging low-level C code and more time developing, testing, and tuning designs. By maintaining one golden reference in MATLAB, including the algorithm and test benches, you can propagate algorithmic changes to your C code more quickly. Automated tools like the MATLAB unit testing framework and the Embedded Coder SIL and PIL testing framework let you test both the MATLAB code and the C code thoroughly and systematically. Whether you are implementing designs running on traditional PCs, web servers, mobile devices, or embedded processors, MATLAB Coder will help you get from MATLAB to C code faster and with fewer manual translation errors. ▪

LEARN MORE

Generating C Code from MATLAB (download)
mathworks.com/fx-56270

MATLAB and C/C++ Resources
mathworks.com/matlab-c

Supported Language and Functions for MATLAB Coder
mathworks.com/coder-language

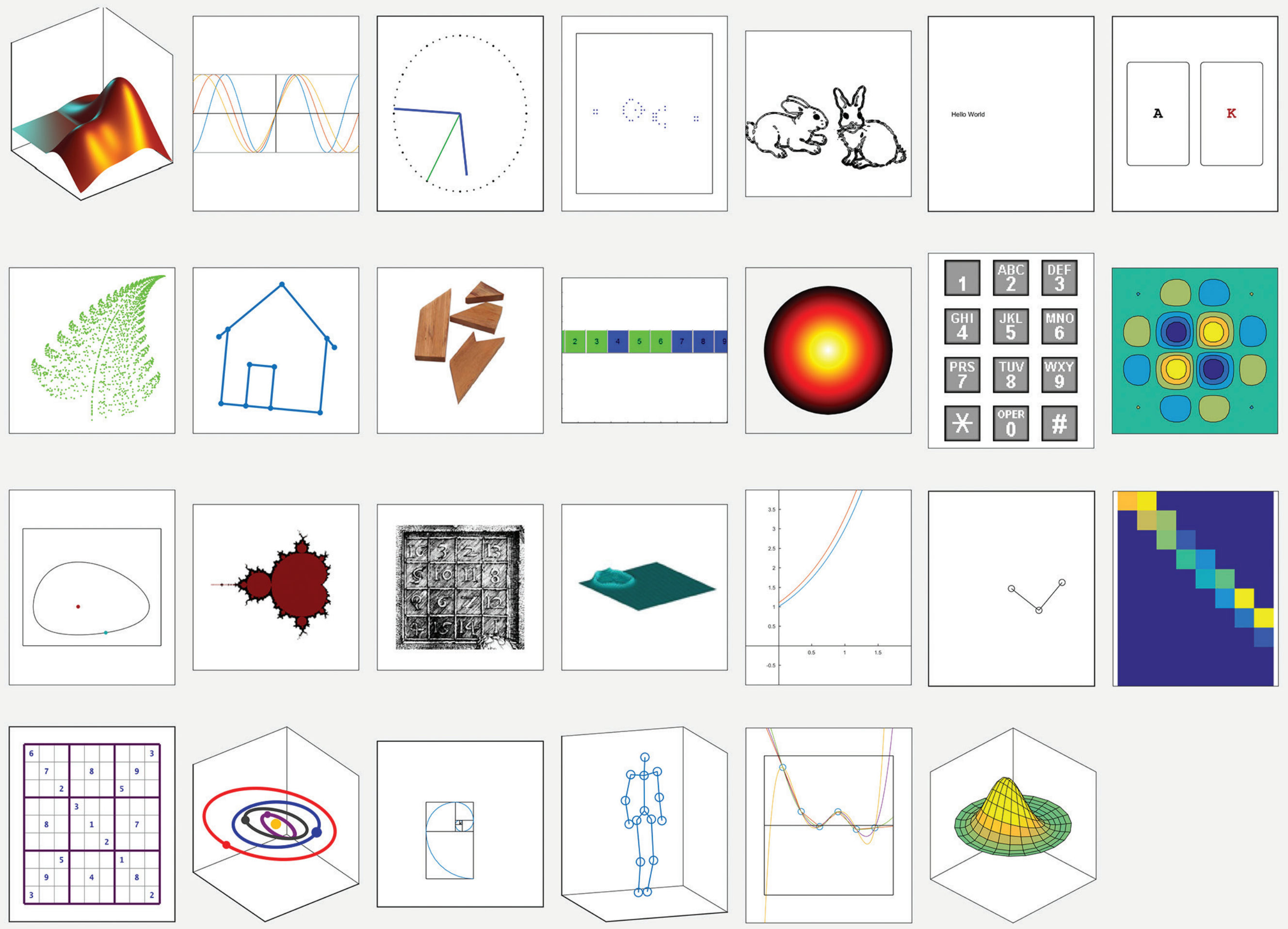
Introducing Cleve's Laboratory

By Cleve Moler, MathWorks

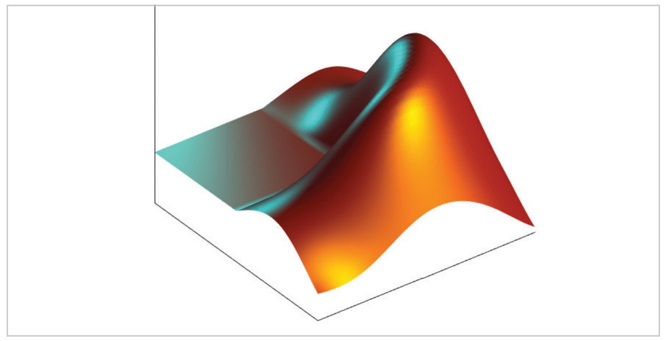
In this Cleve's Corner I want to give you a sneak preview of the experiments in "Cleve's Laboratory." I launched the laboratory to collect much of the work I have done over the last several years in one place. The experiments come from my two ebooks, my blog, Cleve's Corner columns in *MathWorks News & Notes*, and new work.

Each experiment centers on an interactive "app" that allows experimenters to try out the ideas for themselves. Several of the experiments, including `hello_world`, `klock`, and `biorhythms`, are designed to introduce newcomers to MATLAB®.

The figure below shows snapshots of the graphical entry pages. Even these are live pages driven by MATLAB. The wave in the first icon moves every time you open the page. The clock reads the correct time. The Sudoku puzzle changes. The double pendulum swings.



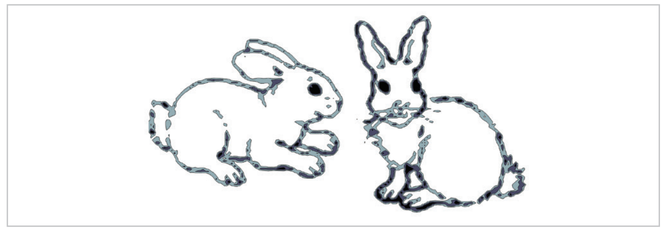
T puzzle image courtesy of Shop New Zealand *www.shopnewzealand.co.nz* © Shop New Zealand



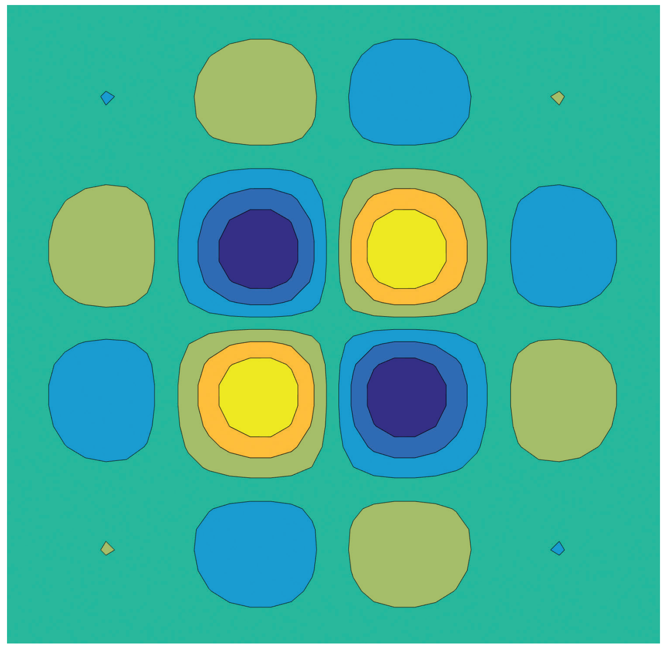
`logo_wave.` The MathWorks logo is the solution to the wave equation, a foundation of mathematical physics. MathWorks is the only company in the world whose logo is the solution to a partial differential equation. This program demonstrates the vibration of a membrane stretched over an L-shaped region.

`lifex.` This is a version of John Conway's Game of Life. "Life" is a cellular automaton that involves life and death in an infinite rectangular, 2D, cellular universe. The `lifex` program accesses the *Life Lexicon*, a historical collection of nearly 500 starting populations available online. It uses sparse matrix operations and an elegant, one-line implementation of Conway's rules for evolution in this universe. The icon shows Bill Gosper's glider gun, which emits a continuous stream of 5-element agents that move across the space.

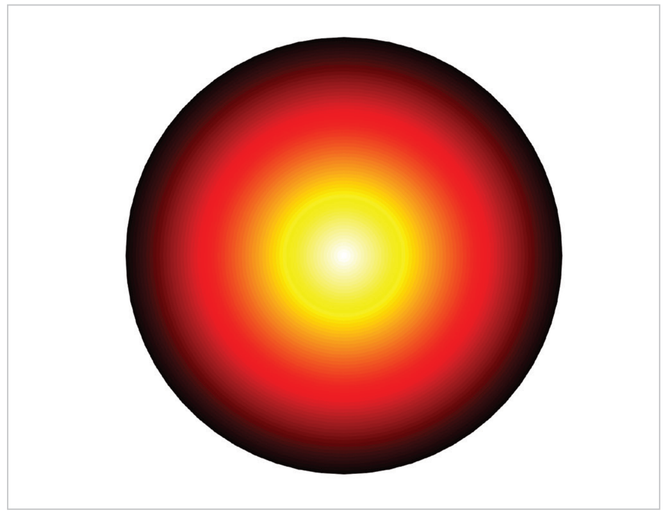
`fern.` The fern is a self-similar fractal. It was invented by Michael Barnsley, and is described in his book *Fractals Everywhere*. Each leaf is similar in structure to the larger fern and contains a miniature copy of yet another leaf. The `fern` program keeps running until the stop button is toggled.



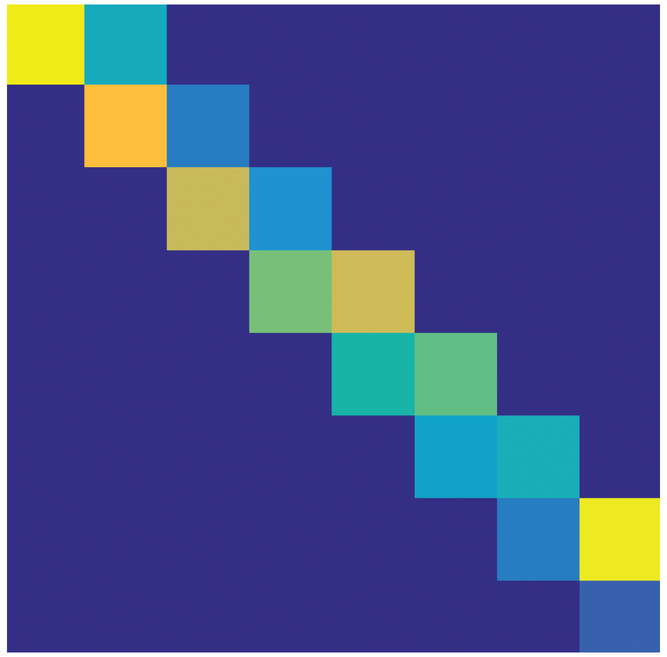
`fibonacci.` This program is based on Fibonacci's rabbit pen. A man puts a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair that from the second month on becomes productive? Today, the solution to this problem is known as the Fibonacci sequence, or Fibonacci numbers.



`pdeapp.` This program demonstrates finite difference methods for solving model problems for four partial differential equations involving Laplace's operator: the Poisson equation, the heat equation, the wave equation, and an eigenvalue equation. The regions are a square, an L-shape, an H-shape, a disc, an annulus, and a pair of isospectral drums.

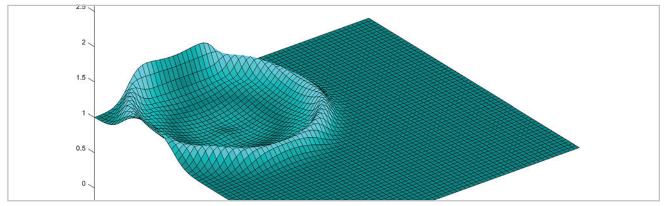


`flame.` This is an example of a stiff ordinary differential equation. `flame(r0)` specifies the initial radius is r0. Default r0 = .02. A ball of fire grows until its radius is just large enough for all the oxygen available through the surface to be consumed by combustion in the interior. The equation for the radius is rdot = r^2 - r^3. The problem becomes stiff as the radius approaches its limiting value.

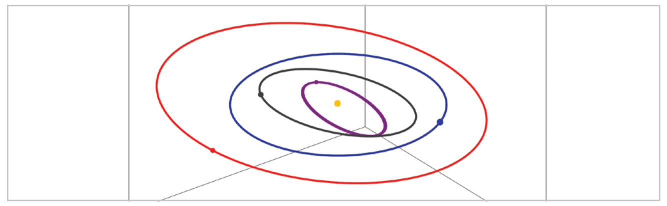


eigsvdapp. With this program you can watch how MATLAB handles three different matrix eigenvalue problems: the eigenvalues of a nonsymmetric matrix, the eigenvalues of a symmetric matrix, and the singular values of any matrix. Watch the reduction to Hessenberg, tridiagonal, or bidiagonal form, then the QR iteration to obtain Schur or diagonal form.

walker. This model, developed by Nikolaus Troje, is a five-term Fourier series with vector-valued coefficients that are the principal components for data obtained in motion-capture experiments involving subjects wearing reflective markers and walking on a treadmill. The components, also known as "postures" or "eigenwalkers," correspond to the subject's movements. The postures are also classified by gender.

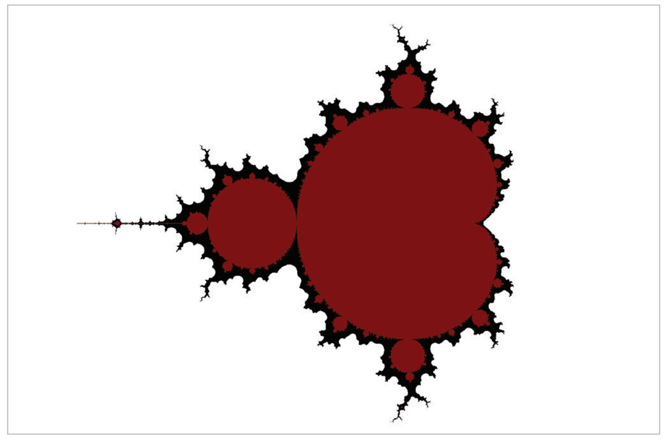


waterwave. A 2D shallow water model with reflexive boundary conditions. A random water drop initiates gravity waves. The surface plot displays height colored by momentum. The solution is computed by the Lax-Wendroff finite difference method. The plot title shows simulated time and total variation.



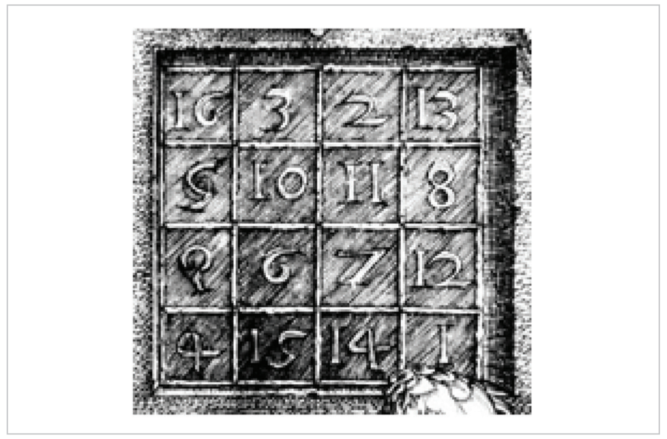
orbits. The orbits program solves Newton's equations governing the gravitational attraction among several bodies. When the number of bodies is equal to nine, this is the solar system with one sun and eight planets.

censusapp. This experiment is older than MATLAB—it started as an exercise in *Computer Methods for Mathematical Computations*, by Forsythe, Malcolm and Moler, published in 1977. The data comes from the decennial census of the U.S., 1900–2010. The task is to extrapolate population data beyond 2010. Today's MATLAB makes it easier to vary the parameters and see the results, but the underlying mathematical principle is unchanged: Using polynomials of even modest degree to predict the future by extrapolating data is a risky business.



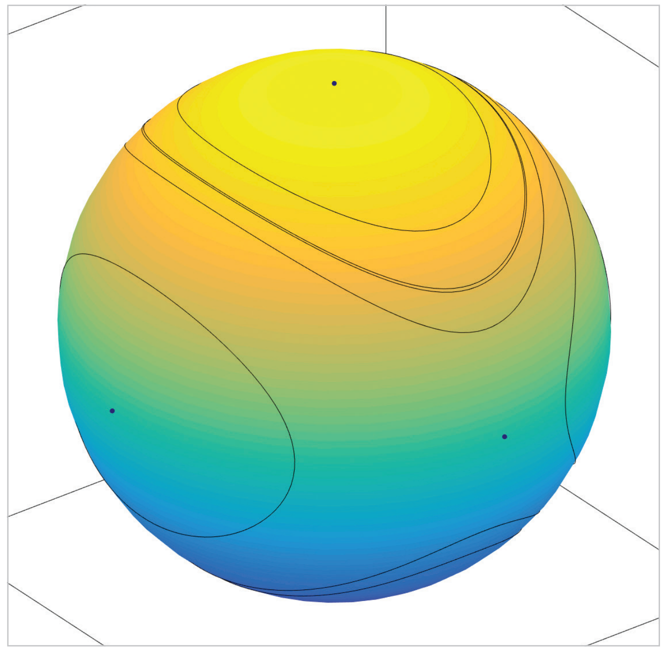
mandelbrot. This program invites you to explore the Mandelbrot fractal. You can use the mouse to select a region, zoom in on any region, increase the grid size, increase the iteration depth, and change the color map.

predprey. In this classic model, one species grows exponentially while the other decays exponentially in the absence of the other. The model is nonlinear, but the solutions are periodic. Dragging the red dot changes the equilibrium point. Dragging the blue-green dot changes the initial conditions.

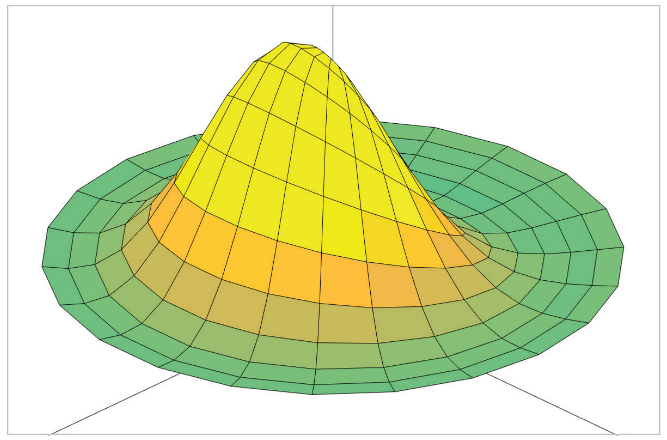


durerperm. The icon shows Durer's magic square. To permute the square, click on two different rows or columns. Is the result still a magic square?

golden_spiral. In this program you can see a continuously expanding sequence of golden rectangles and inscribed quarter circles.



tumbling_box. If you throw a rectangular box in the air with a twist, you can make it tumble stably about its longest or shortest axis. But if three sides of the box are of different lengths, you cannot make it tumble about its middle-sized axis. The Euler differential equations for the angular momenta about the three principal axes have two stable critical points and one unstable critical point.



waves. This program demonstrates the wave equation in 1D and 2D space dimensions. Solutions are expressed as time-varying weighted sums of the first four eigenfunctions. The 1D domain is an interval. The 2D domains include a square, a disc, a three-quarter circular sector, and the L-shaped union of three squares. With polar coordinates, the eigenfunctions of the disc and the sector involve Bessel functions.

tictactoe. This program combines three games that initially appear to be unrelated: Pick15, TicTacToe, and Magic3. In Pick15, the object is to generate a total of 15 using exactly three digits, where each digit can be chosen only once. TicTacToe follows the traditional game, but replaces X's and O's with blue and green. The object is to get three in a row, column, or diagonal. Magic3 superimposes a magic square of order three on TicTacToe to show that Pic15 is actually the same game.

t_puzzle. I first saw this wooden T puzzle at Puzzling World in Wanaka, New Zealand. The underlying mathematics involves geometry, trigonometry, and arithmetic with complex numbers. The `t_puzzle` program demonstrates some useful programming techniques. The four pieces all have the same width but different heights. It turns out that they can be arranged to form a capital "T" as well as an arrow and a rhombus. ▪

LEARN MORE

Cleve's Laboratory App (download)
mathworks.com/cleves-lab

Cleve's Corner Blog
blogs.mathworks.com/cleve

Cleve's Corner Collection
mathworks.com/cleves-corner

Smart Devices and Analytics Spur Innovation in the Internet of Things

By Eric Wetjen, MathWorks

The Internet of Things (IoT) is a rapidly evolving space in which virtually any smart hardware device—a mobile phone, a pacemaker, a wearable fitness sensor, even a refrigerator—can be connected to the internet to generate and receive data.

Combining internet-connected devices with cloud computing, machine learning, and other data analytics approaches is enabling products and solutions that are transforming the way we live and work. Today, for example, thanks to the Internet of Things:

- A doctor can remotely monitor how often a patient's pacemaker fires.
- Athletes can measure how many calories they burn during a run.
- Farmers can optimize irrigation of crops.
- Building managers can save electricity by optimizing controls for HVAC equipment.
- Asthma sufferers can manage their condition by using a mobile phone app to monitor their wheezing levels.
- Automakers are close to developing a driverless car capable of autonomously navigating through city streets.

Inside an IoT System

A typical IoT system works like this (Figure 1):

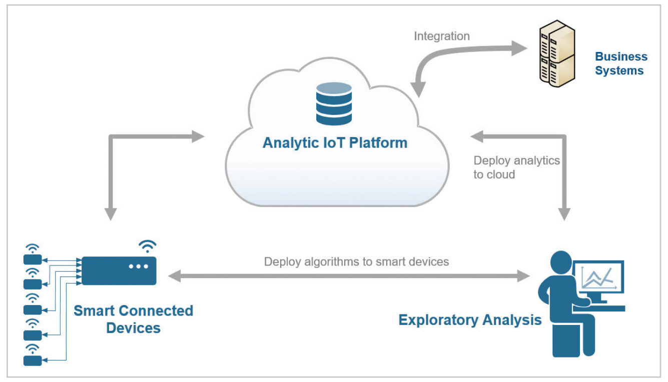


FIGURE 1. *A typical Internet of Things workflow.*

1. A smart connected device produces sensor data and ultimately sends data to the cloud. These devices are often smart enough to run data-reduction algorithms on their embedded processors.
2. An analytic IoT platform processes and stores the sensor data. It may integrate information from other sources, such as business systems. It analyzes and takes action on the incoming data.
3. A systems engineer or data scientist accesses the historical data from the cloud or the device and develops algorithms to preprocess and analyze it. These algorithms may involve machine learning techniques for predicting future values of a sensor quantity or for classifying the sensor data.
4. The algorithm is deployed in the cloud or on a smart device, where it operates on incoming live data.

Creating an IoT System with MATLAB and Simulink

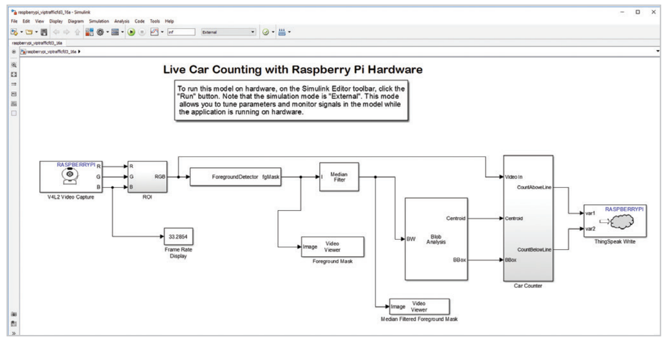
MATLAB® and Simulink® support IoT systems by helping you develop and test smart connected devices, access and collect data in the cloud, and analyze sensor data. Let's look at two examples.

Example 1. A Traffic Monitor

Tired of sitting in traffic every time they drove home from work, two engineers decided to study traffic flow trends on the busy highway outside their building. They installed a webcam in an office overlooking the highway and connected it to a Raspberry Pi™ board running a computer vision algorithm (Figure 2).

The data was collected in the cloud using ThingSpeak™, an analytic IoT platform that can run MATLAB code.

FIGURE 2. *Webcam connected to a Raspberry Pi 2.*


raspberrypi_viptrafficfd3_16a - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
inf
External
Live Car Counting with Raspberry Pi Hardware
RASPBERRYPI
V4L2 Video Capture
ROI
ForegroundDetector fgMask
Median Filter
Frame Rate Display
Video Viewer
Foreground Mask
Blob Analysis
Median Filtered Foreground Mask
Car Counter
ThingSpeak Write

To run this model on hardware, on the Simulink Editor toolbar, click the "Run" button. Note that the simulation mode is "External". This mode allows you to tune parameters and monitor signals in the model while the application is running on hardware.

FIGURE 3. *Simulink block diagram to find and count the cars.*

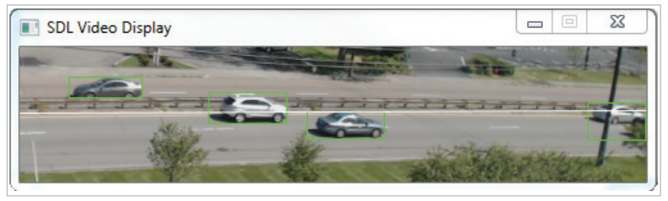


FIGURE 4. *Simulink external mode detection of cars during algorithm development phase.*

They built a Simulink block diagram that included a Median Filter block to clean up the image and a Blob Analysis block to identify cars in the image (Figure 3). They then wrote a custom block to count the cars, and sent the summary data to ThingSpeak.

Before deploying the algorithm to the Raspberry Pi, they verified it using the video display in Simulink external mode (Figure 4).

Once the data was in ThingSpeak, they used the MATLAB app integrated into ThingSpeak to create live visualizations of the traffic density in each direction for the previous 48 hours. These include a color-coded visualization (Figure 5) that categorizes the current state of the traffic (light = green, moderate = yellow, and heavy = red).

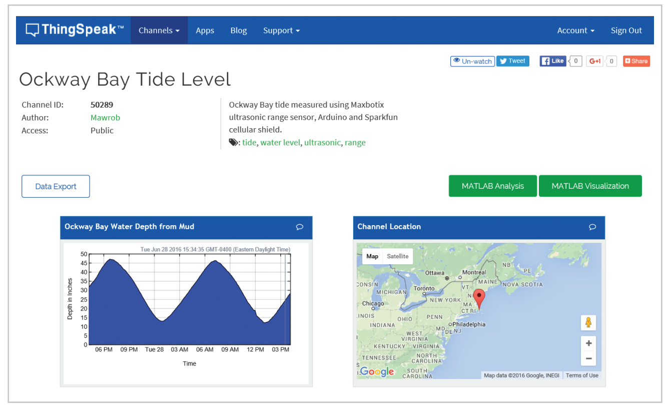
Example 2. A Tide Gauge and Alert System

Getting your boat stuck in the mud is a distinct possibility without an accurate measure of water depth. Because tide predictions and real-time water levels are not available for most bays and estuaries, one boater and MATLAB user built a low-cost, real-time tide gauge. He quickly learned that the timing and amplitude of tide levels is highly dependent on location.

The key hardware components he used were an Arduino® Mega board, a SparkFun Electronics® cellular shield, and an ultrasonic range finder. The cellular shield provided the connection from the Arduino to the internet. The Arduino was used to read the data from the ultrasonic sensor.

ThingSpeak was used to collect and process the data in the cloud. The ultrasonic sensor reports distance in mm. To convert from distance to water depth, the boater set up a ThingSpeak TimeControl that runs MATLAB code to read the range data, convert it into water

FIGURE 5. *Categorizing live traffic data in ThingSpeak.*



Ockway Bay Tide Level

Channel ID: 50289
Author: Mawrob
Access: Public

Ockway Bay tide measured using Maxbotix ultrasonic range sensor, Arduino and Sparkfun cellular shield.
: tide, water level, ultrasonic, range

FIGURE 6. *ThingSpeak data collection and tide level display.*

depth, and write the data to a new ThingSpeak channel. The result was an internet-connected tide gauge that can be viewed on a mobile phone or a web browser (Figure 6).

The boater set up additional MATLAB code to detect tidal thresholds. He then used the Twitter integration in ThingSpeak to send alerts when particular conditions were met.

As these examples show, analytics is a vital ingredient of the innovations occurring in IoT. With MATLAB and Simulink you can develop analytics that run on your smart devices or in the cloud. With ThingSpeak, you can easily collect data from your devices in the cloud. You can then use MATLAB to gain insight into the sensor data you have collected. ▪

LEARN MORE

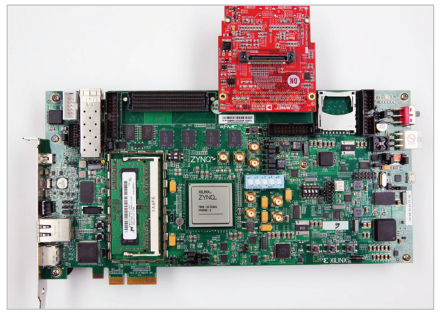
Developing Internet of Things Systems with MATLAB
mathworks.com/iot

ThingSpeak
thingspeak.com

Traffic Monitor Data Visualizations
thingspeak.com/channels/38629

Solutions for Image Acquisition and Computer Vision

MATLAB® and Simulink® provide a platform for engineers to explore images, develop computer vision algorithms, and evaluate implementation tradeoffs. Third-party imaging hardware enables them to acquire visible, thermal, depth, microscopy, and a range of other images. Engineers can analyze images, via live acquisition or by postprocessing in MATLAB, and then deploy computer vision applications such as object detection, tracking, and recognition onto embedded processors, FPGAs, multicore GPU systems, and other hardware.



FLIR Infrared Cameras

FLIR thermal and visible-light imaging systems are used in a wide variety of thermal imaging, situational awareness, and security applications. Using Image Acquisition Toolbox™, engineers can configure features of FLIR infrared cameras and stream fully temperature-calibrated data directly into MATLAB for analysis, visualization, and modeling.

flir.com

3i SlideBook

SlideBook microscopy software enables scientists to acquire and analyze image data across time, color, and specimen locations, with customizable experiment protocols and drivers for hundreds of research instruments. SlideBook supports data import/export with MATLAB, enabling users to call MATLAB to set variables and perform advanced image analysis, and extending SlideBook's built-in image processing functions. MATLAB scripts can drive live cell experiments by determining regions for capture and photomanipulation.

intelligent-imaging.com

Allied Vision FireWire and GigE Vision Cameras

Allied Vision offers high-performance IEEE-1394 FireWire, GigE Vision®, and USB3 Vision compliant digital cameras for machine vision, computer vision, and other industrial or medical applications. Cameras include CCD and CMOS sensors ranging from VGA to 29 megapixels. Image Acquisition Toolbox, with Allied Vision's Vimba Software Development Kit (SDK), enables direct access to the cameras from MATLAB and Simulink.

alliedvision.com

Microsoft Kinect

Microsoft® Kinect® for Windows includes a 3D depth sensor and integrated RGB camera for applications such as robotics, kinesiology, 3D scene reconstruction, point-cloud processing, and skeletal tracking. Engineers can acquire 3D data from Microsoft Kinect into MATLAB and Simulink using Image Acquisition Toolbox and Computer Vision System Toolbox™.

developer.microsoft.com/kinect

Xilinx Zynq-7000 All-Programmable SoC

The Xilinx® Zynq®-7000 All-Programmable SoC combines a dual-core ARM® Cortex®-A9 with Xilinx 7-series FPGA logic on a single chip. As a result, users can integrate camera control and image processing functions in a single device while enabling hardware acceleration of video analytics. Evaluation kits include hardware I/O, design tools, IP, and pre-verified reference designs. MATLAB, Simulink, HDL Coder®, Embedded Coder®, and Vision HDL Toolbox™ support the design and simulation of image processing applications and automated deployment to the Zynq device.

xilinx.com/zynq

LEARN MORE

Hardware Support
mathworks.com/hardware

Image Processing and Computer Vision
mathworks.com/solutions/image-video-processing

Third-Party Products and Services
mathworks.com/connections

WHAT IF **EVERYONE** ON CAMPUS HAD **MATLAB?**

More than **1 million students and 700 universities** around the world—including the top 10 ranked universities—have unlimited access to MATLAB and Simulink with a Total Academic Headcount (TAH) license.

"On multidisciplinary projects, students with quite different educational backgrounds can work together more easily because they are using the same tools."
Professor Jakob Stoustrup, Aalborg University

HANDS-ON LEARNING
42,000
Faculty and students using MATLAB to program hardware

JOB OPPORTUNITIES
82%
Fortune 100 companies with a MATLAB license

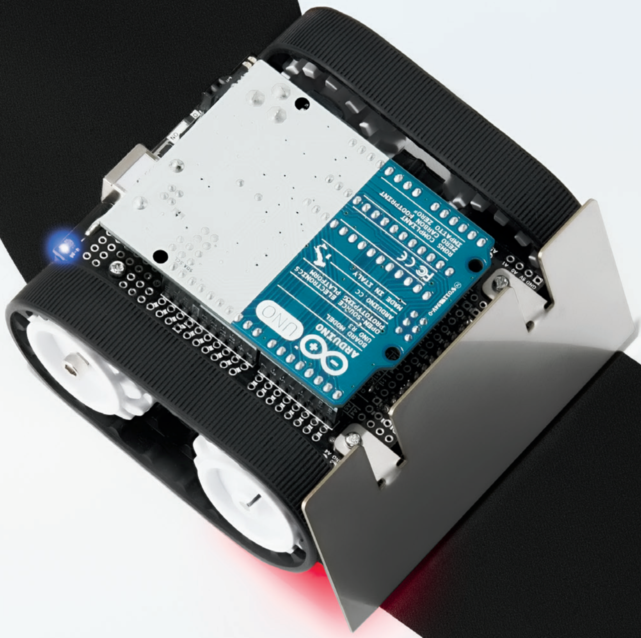
"If you want to work at Google, make sure you can use MATLAB."
Jonathan Rosenberg, Senior Vice President of Products, Google

"Our teams are here to do world-class research, and easy access to MATLAB enables them to be their most productive."
Shailesh Shenoy, Director of Research Computing, Albert Einstein College of Medicine of Yeshiva University

RESEARCH PRODUCTIVITY
1,970,000
Google Scholar results referencing MATLAB

mathworks.com/matlab-campus

MathWorks®



MATLAB SPEAKS ARDUINO so you don't have to

You can design, build, test, and run a system—on Arduino, Raspberry Pi, LEGO, and more—without writing traditional code.

Download free MATLAB and Simulink hardware support packages at *hardware.mathworks.com*

MATLAB® & SIMULINK®

92984v00 11/16