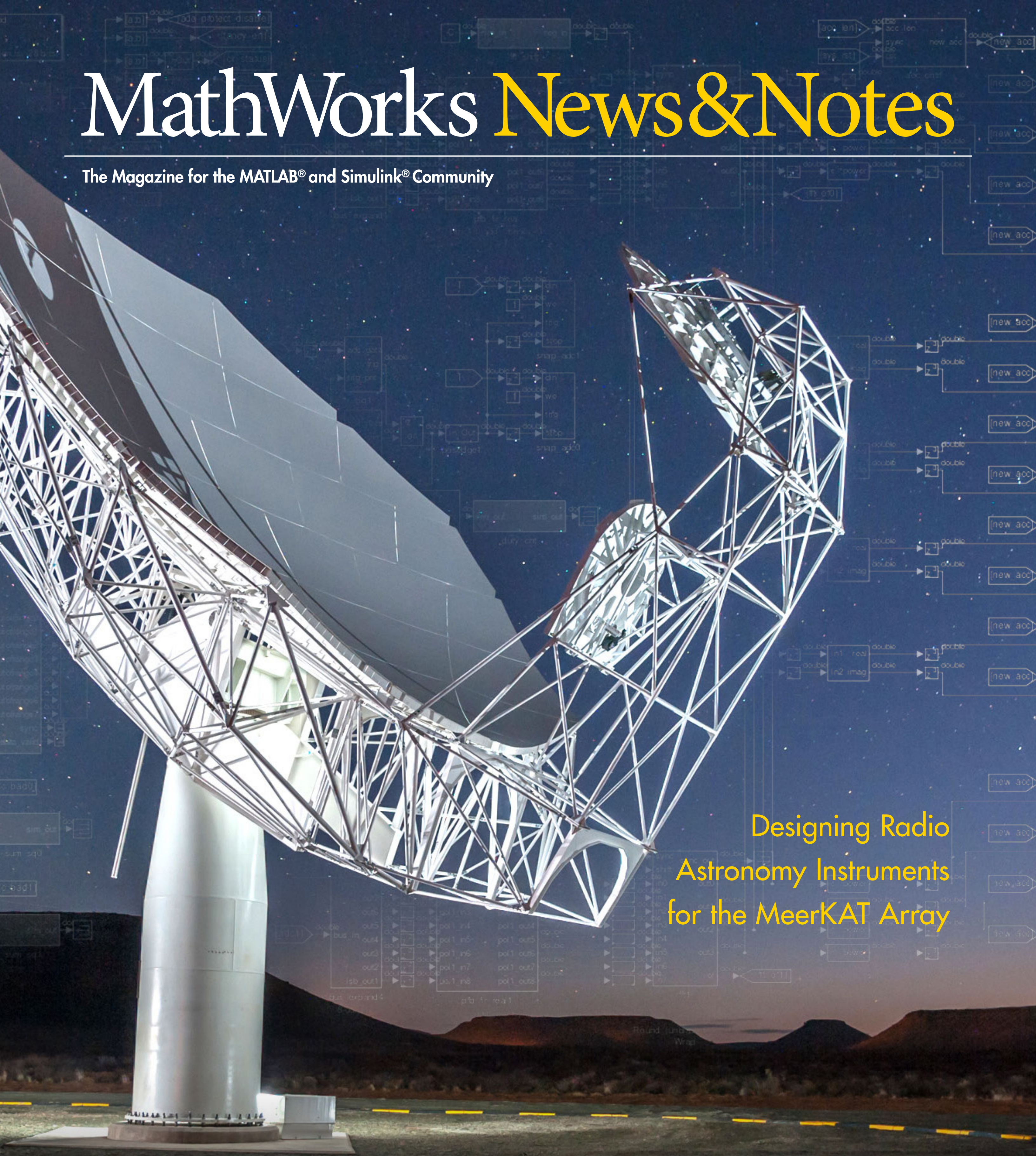
# MathWorks News&Notes

The Magazine for the MATLAB® and Simulink® Community



Designing Radio Astronomy Instruments for the MeerKAT Array

ALSO IN THIS ISSUE

MathWorks®

Why does `fft` show harmonics at different amplitude?

How do I use `randper` to produce a complete new sequence?

Can I pass arrays to and from a GUI?

How do I verify that a Support Package is installed?

**Got questions? Get answers.**

How do I calculate a loglikelihood value?

How do I pre-allocate memory when using MATLAB®?

What MATLAB Easter eggs do you

Is it possible to write several statements into an anonymous function?

How do I use multiple colormaps in a single figure?

How do I reverse the order of a vector?

Can I plot the spectrum diagram of a signal?

Does `dec2hex` or `hex2dec` work on negative numbers (twos complement)?

## MATLAB Answers

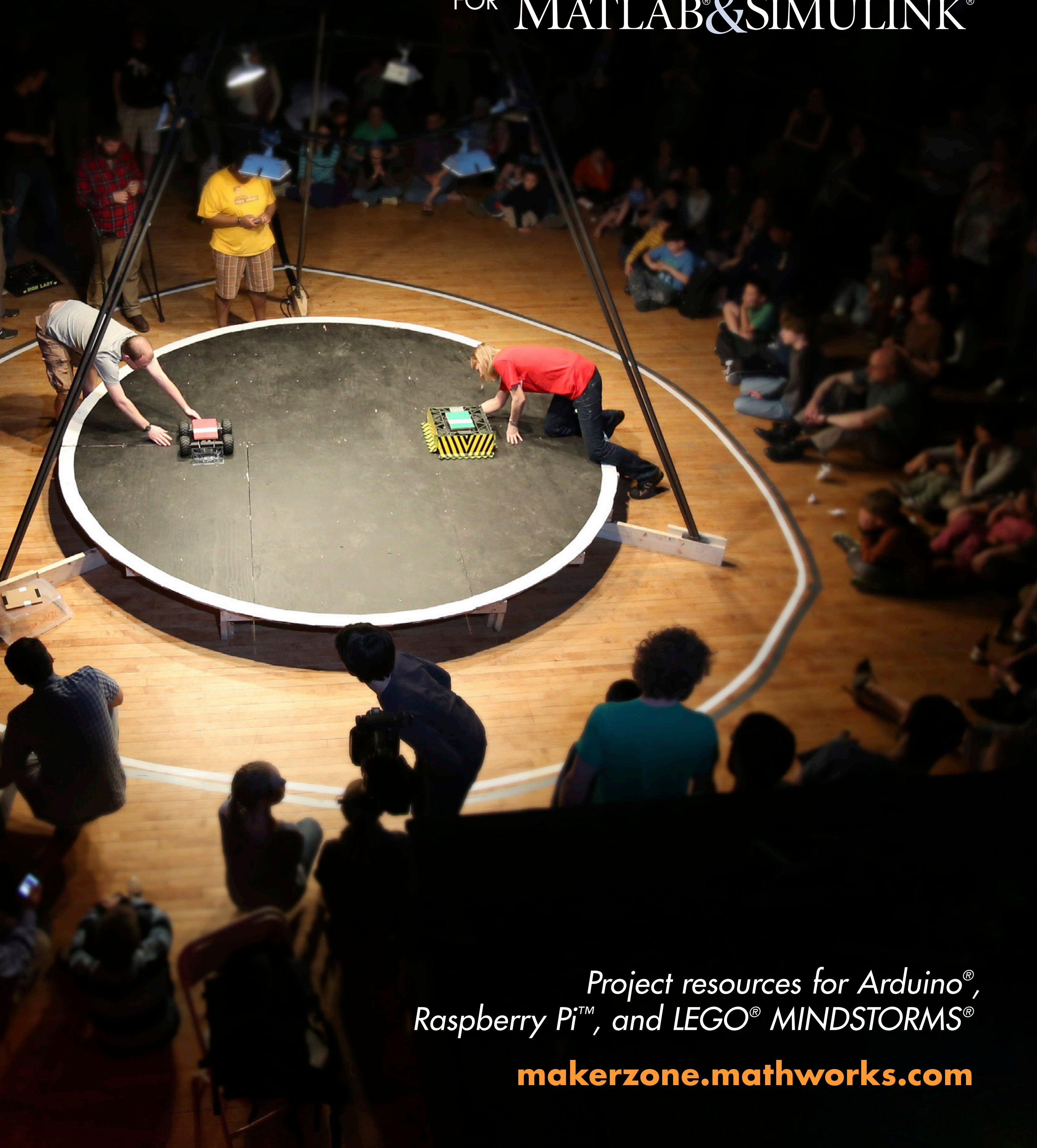
Tap into the knowledge and experience of over 100,000 MATLAB Central members.

mathworks.com/matlab-answers

Enter the

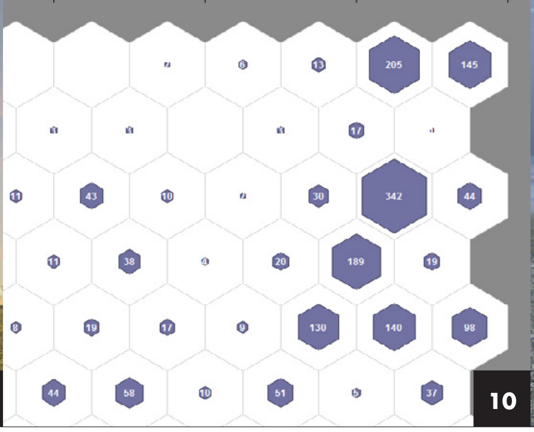
**MAKERZONE**

FOR MATLAB® & SIMULINK®

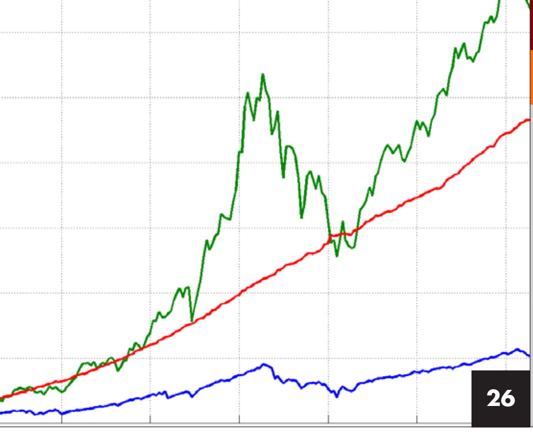


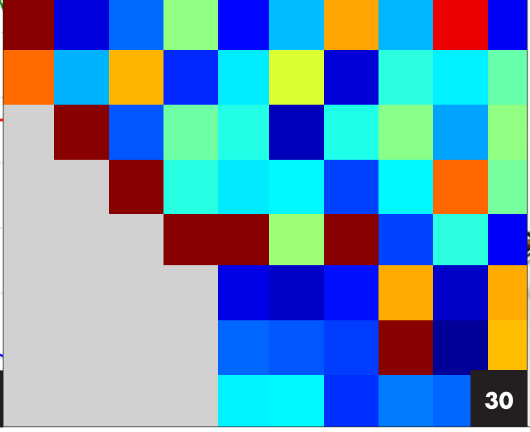
*Project resources for Arduino®, Raspberry Pi™, and LEGO® MINDSTORMS®*

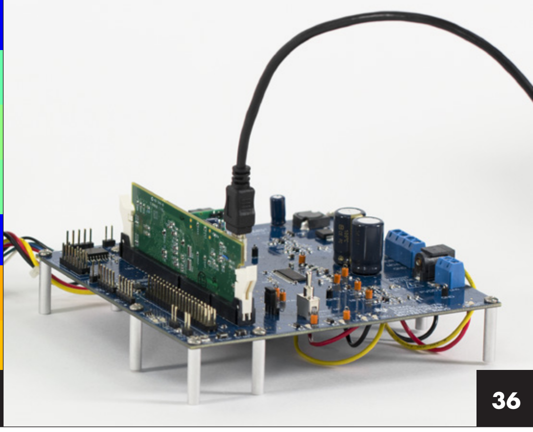
**makerzone.mathworks.com**



## FEATURES







## DEPARTMENTS

## ABOUT THE COVER

The cover shows the first of the 64 antennas that will make up the MeerKAT radio telescope in South Africa's Karoo region. As a forerunner to the Square Kilometre Array (SKA), slated for completion in 2024, MeerKAT will be used by scientists worldwide on a range of research projects, including observing neutral hydrogen gas and molecular hydrogen levels in the early universe, the physics of enigmatic neutron stars, and the nature and behavior of pulsars, dark matter, and the cosmic web.

**Managing Editor**
Linda Webb

**Editor**
Rosemary Oxenford

**Art Director**
Robert Davison

**Graphic Designer**
Katharine Motter

**Technical Writer**
Jack Wilber

**Production Editor**
Julie Cornell

**Printer**
DS Graphics

**Production Staff**
M. Arnold, D. Audette, K. Camerlin, L. Goodman, L. Heske, A. Hutker, K. Kevorkian, L. Macdonald, A. Pollack, M. Ruotolo, G. Quilop

**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**
G. Bourdon, G. Campa, M. Carone, K. Cohan, S. Cruickshank, J. Deinzer, M. DeMaio, A. Deoras, S. deSouza, S. Eddins, M. Egerstedt, J. Eggers, N. Fernandes, J. Friedman, W. Fullam, T. Gaudette, S. Hiremath, M. Judin, F. Kapp, R. Kause, S. Kodial, S. Kozola, T. Kush, T. Lennon, N. Levine, D. Mathis, M. McBroom, I. McKenna, M. Michanek, A. Mishra, S. Mosesson, S. Oliver, D. Oswill, B. Petteys, A. Poon, S. Prasanna, D. Ramaswamy, C. Rawal, M. Ricci, S. Sharma, R. Shenoy, M. Stakenborg, S. Tandon, S. Tiwari, T. Tohyama, L. Toms, A. Turevskiy, G. Vallabha, P. Wallner, D. West, E. Wetjen, S. Wilcockson, S. Zaranek, Y. Zhang

**Subscribe**
mathworks.com/subscribe

**Comments**
mathworks.com/contact

**Follow and Interact with Us Online**
/MATLAB
/+MATLAB
/the-mathworks_2
/MATLAB

MIX
From responsible sources
FSC® C084277
FSC www.fsc.org

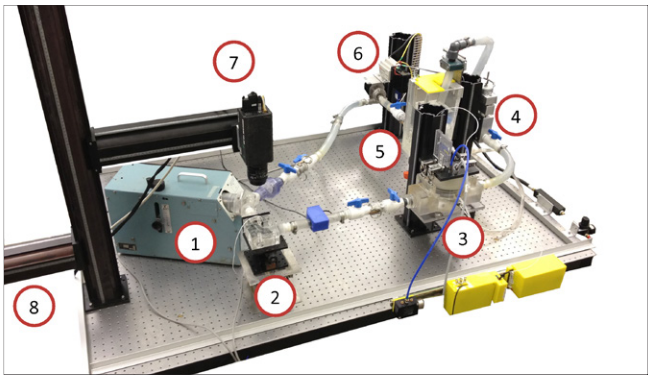
Printed on 30% post-consumer waste materials

Made in the U.S.

MathWorks®

# Medical Devices

In nearly 2000 medical device companies worldwide, engineers use MATLAB® and Simulink® to analyze image and signal data, design controls and signal processing algorithms, implement their designs in software after thorough verification and validation, and automatically generate reports for standards compliance.



### VIRGINIA COMMONWEALTH UNIVERSITY

**Simulating cardiovascular conditions using a fully automated mock circulatory loop**

Mock circulatory loops (MCLs) simulate the human circulatory system to enable testing of cardiac assist devices. When conducting tests using an MCL, investigators must adjust the settings on each component in the loop. VCU researchers modeled a fully automated MCL in which the mechanical pump, compliance chamber, and peripheral resistance valve settings are dynamically adjusted via microprocessors. The complete Simulink model of the system enables the user to apply parameter estimation routines to patient data to determine the settings needed to replicate a wide range of cardiac conditions and dynamics.

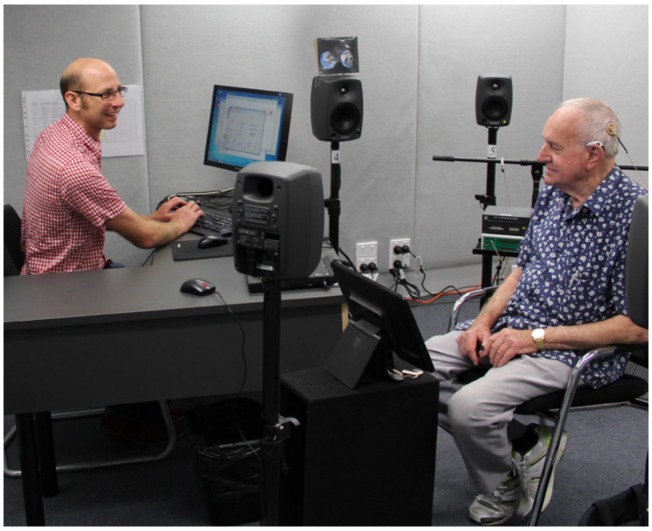
***mathworks.com/virginia-commonwealth***

### CLEVELAND FES CENTER AND CASE WESTERN RESERVE

**Restoring movement to paralyzed limbs with functional electrical stimulation**

For individuals with neurological impairments, functional electrical stimulation (FES) can help make real what was once only imagined: the restoration of movement to paralyzed arms and legs. FES devices send electrical impulses to electrodes—implanted in the body, worn on the skin, or operating through the skin—to produce and control movement. Researchers at Case Western Reserve University developed a flexible, configurable system technology platform that enables rehabilitation engineers to change an FES controller in the clinic and build FES control software. The Simulink based Universal External Control Unit enables clinicians to develop and refine their own FES applications up to 10 times faster than before.

***mathworks.com/fes***

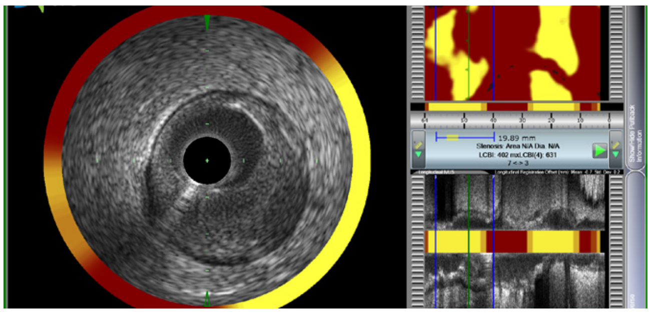


### COCHLEAR LIMITED

**Streamlining development of cochlear implant sound processing algorithms**

Noise reduction algorithms help cochlear implant recipients perceive speech in challenging acoustic settings. Engineers at Cochlear use Simulink to model and simulate candidate algorithms. After identifying a promising algorithm, the engineers generate C code from the model using Simulink Coder™. The code is compiled and deployed to a Speedgoat turnkey real-time system with Simulink Real-Time™. In clinical validations, the Simulink Real-Time system is linked via custom hardware to a recipient's implant, enabling the clinician to measure the performance of the algorithm using simulated real-life acoustic conditions, including live speech in quiet and noisy environments.

***mathworks.com/cochlear***



## INFRAREDX

### Accelerating FPGA development of an intravascular imaging system

A patient with lipid core plaques (LCP) is vulnerable to coronary artery disease, the number one killer in developed countries. Infraredx developed the only FDA-approved medical device for LCP detection. The TVC Imaging System™ combines near-infrared spectroscopy with intravascular ultrasound (IVUS) in a single coronary catheter to provide information about vessel composition and structure. Infraredx engineers modeled and simulated the IVUS signal and image processing algorithms in Simulink, and implemented the design on the Altera® Cyclone FPGA using VHDL® code generated from the Simulink model with HDL Coder™. The generated HDL used the same number of multipliers as their handwritten HDL while using 9% less logic and 3% less memory.

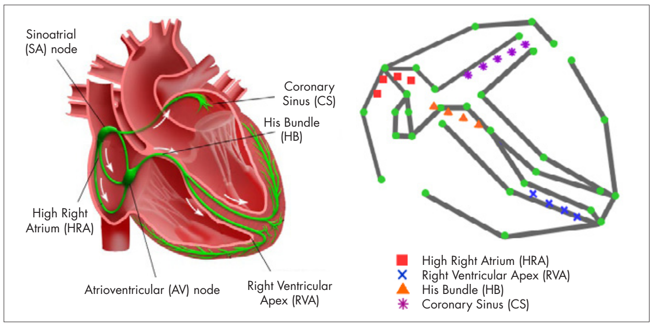
***mathworks.com/infraredx***

## ITK ENGINEERING

### Developing IEC 62304–compliant controller software for a dental drill motor

Sensorless brushless DC (BLDC) motors cause less abrasion than brushed motors, and are more reliable, quieter, and easier to maintain, but they require complex control algorithms. ITK Engineering developed a production BLDC motor controller that complies with the IEC 62304 medical device software standard. They modeled the motor, including its electrical and mechanical components, in Simulink. They developed a Simulink controller model, and used Stateflow® to model startup, shutdown, and error modes, as well as user-selectable operating modes. After running closed-loop simulations of the plant and the controller, they generated more than 5000 lines of C code from their controller model, and compiled the code for an ARM® Cortex®-M3 processor.

***mathworks.com/itk***



## UNIVERSITY OF PENNSYLVANIA

### Closed-loop real-time testing of pacemakers with virtual heart models

One-third of the more than 600,000 cardiac medical device recalls that occurred between 1990 and 2000 were due to software problems. To enable early verification of pacemaker software, University of Pennsylvania engineers developed a first-of-its-kind heart-on-a-chip that can be configured to match a patient's specific electrophysiological characteristics. The team built an electrophysiological model of the heart in Simulink and Stateflow, and deployed multiple versions of the heart model on an Altera FPGA using automatically generated HDL code. Using the virtual heart model running on the FPGA in real time they simulated several closed-loop scenarios, including pacemaker-mediated tachycardia and atrial flutter, as well as failure conditions, such as a displaced pacemaker lead.

***mathworks.com/upenn***

### Learn More

**User Stories**
*mathworks.com/user-stories*

**Technical Articles**
*mathworks.com/technical-articles*

# Accelerating the Design of Radio Astronomy Instruments in South Africa and Worldwide with Simulink

By Francois Kapp, SKA South Africa

When it is complete, the MeerKAT array in South Africa's Karoo region will include 64 antenna dishes, making it the largest and most sensitive radio telescope in the southern hemisphere. As a forerunner to the Square Kilometre Array (SKA), slated for completion in 2024, MeerKAT will be used by scientists worldwide on a range of research projects, including observing neutral hydrogen gas and molecular hydrogen levels in the early universe; the physics of enigmatic neutron stars; and the nature and behavior of pulsars, dark matter, and the cosmic web.

Each of these research initiatives depends on the ability to process the massive amounts of data collected by the radio antennas 24 hours a day. To meet this need, the Digital Back-End (DBE) team at SKA South Africa is developing a digital signal processing (DSP) system capable of handling 5 terabits of data per second. This system is designed with a Simulink® library that enables our team and fellow researchers to rapidly design and deploy radio astronomy instrumentation on modular, reusable hardware using Model-Based Design.

## Radio Astronomy Instrument Design Challenges

Early radio astronomy relied on large, single-dish antennas up to 300 meters in diameter that focused on relatively small areas of the sky. Modern antenna arrays, which are constructed using multiple small dishes, offer a wider field of view and greater flexibility. Moving the dishes farther apart increases the telescope's resolution and enables it to detect smaller objects. Moving the dishes closer together decreases the resolution but expands the field of view.

The challenge is to correlate data received from all the antennas in the array. To determine the position of an object in the sky, the DSP system must calculate the phase difference between radio signals arriving at each pair of antennas in the array. Because data from all the antennas must be cross-correlated (that is, each antenna's signal must be correlated with every other antenna's signal), the processing task is an $N^2$ problem: When we double the number of antennas, we have to quadruple the processing power of the DSP.

Compounding the challenge is the continuous nature of the data stream. Optical astronomy data can be collected at night, stored, and then processed the following day. Radio astronomy data, by contrast, is collected 24 hours a day, and must be processed as it is received.

## Moving from Specialized Hardware to Reusable Libraries and Hardware Components

Specialized hardware based on ASICs or FPGAs is needed to process the raw data received from the antenna array in real time. In the past, developing this hardware required a team of experienced hardware engineers and years of effort. Today, scientists with little or

*Artist's impression of the MeerKAT array in the Karoo region of South Africa. The SKA South Africa team had to develop a DSP system capable of correlating data as it is received from all the antennas in the array. Because data from all the antennas must be cross-correlated, when we double the number of antennas we have to quadruple the processing power of the DSP.*

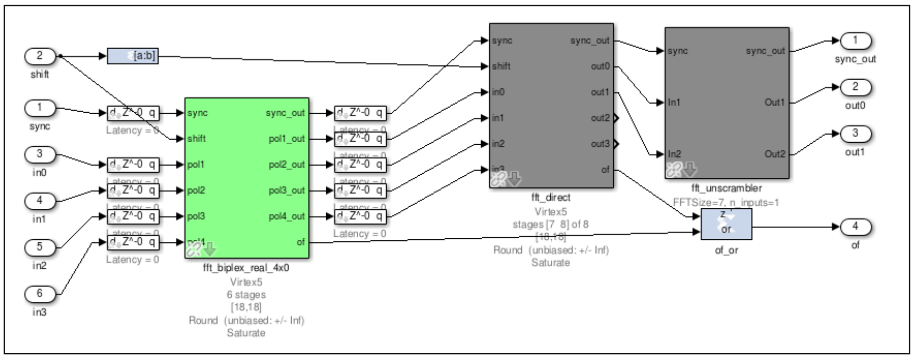
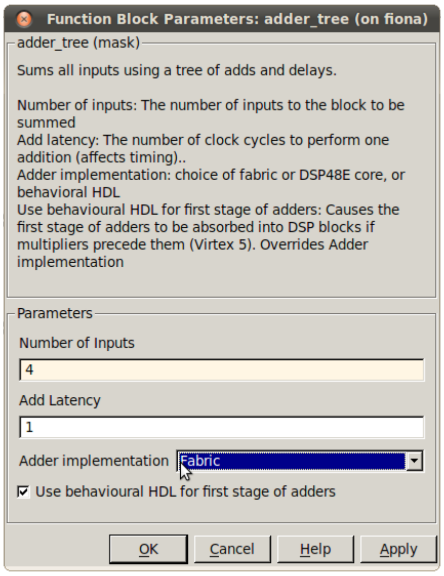


FIGURE 1. *A 4-input wideband FFT Simulink library block.*


Function Block Parameters: adder_tree (on fiona)

adder_tree (mask)

Sums all inputs using a tree of adds and delays.

Number of inputs: The number of inputs to the block to be summed
Add latency: The number of clock cycles to perform one addition (affects timing)..
Adder implementation: choice of fabric or DSP48E core, or behavioral HDL
Use behavioural HDL for first stage of adders: Causes the first stage of adders to be absorbed into DSP blocks if multipliers precede them (Virtex 5). Overrides Adder implementation

Parameters

Number of Inputs: 4

Add Latency: 1

Adder implementation: Fabric

☑ Use behavioural HDL for first stage of adders

FIGURE 2. *CASPER Library adder tree configuration interface. Users can specify the number of inputs, latency, and other implementation settings.*

no hardware design experience can develop their own instrumentation in months using Simulink, a library of drag-and-drop components, and standardized Reconfigurable Open Architecture Computing Hardware (ROACH) boards.

The Simulink library and ROACH boards form the basis of an open-source design environment for radio astronomy instrumentation maintained by the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER). SKA South Africa has been a key contributor to the CASPER library during the development of the DSP systems for MeerKAT.

The CASPER library consists of components commonly used in radio astronomy, including mixers, oscillators, down-converters, filters, matrix transforms, accumulators, adders, and wideband fast Fourier transform (FFT) blocks (Figure 1).

The blocks are parameterized and configured automatically by a MATLAB® script based on settings specified by the user (Figure 2). For example, to create a five-input parallel adder block, the script automatically inserts the required number of basic adders and delays, connects them, and packages them as a self-contained block for use in a larger design.

Scientists assemble signal processing systems from CASPER library blocks and simulate their design in Simulink, using recorded radio signal data as input. After verifying their design via simulation, they use HDL Coder™ to generate HDL or Verilog® code from the Simulink model for deployment on the ROACH board's Virtex® FPGA. In addition to the FPGA, the ROACH board includes analog-to-digital converters (ADCs), a 10-gigabit Ethernet interface, and other hardware components frequently required in radio astronomy instruments (Figure 3).

Initially developed with blocks from the Xilinx® System Generator for DSP™ blockset, the CASPER library made it easy for scientists to generate HDL code for their designs and target Xilinx FPGAs. At SKA South Africa, a project is under way to migrate the library to native Simulink blocks, which will enable the use of HDL Coder to generate the HDL code, and partition designs so that functional blocks can be targeted not only to FPGAs but also to ASICs or ARM® processors. HDL Coder also opens the possibility of expanding CASPER beyond radio astronomy and into other disciplines that require similar real-time signal processing capabilities.

## From KAT-7 to MeerKAT

The seven-dish KAT-7 array is an engineering prototype for the MeerKAT array. KAT-7 is the world's first radio telescope array with fiberglass dishes. It is located in the Karoo region of South Africa, a sparsely populated area with low levels of radio interference from human activity, where MeerKAT and parts of the Square Kilometre Array will also be built (Figure 4).

We developed the KAT-7 DSP system in Simulink using the CASPER library and deployed it to 16 ROACH boards. While its primary purpose was to serve as a proof-of-concept for MeerKAT, KAT-7 is a valuable telescope in its own right, and has already produced images of Centaurus A, a galaxy 14 million light years away.

We are currently using Simulink and the CASPER library to design the signal processing systems for the MeerKAT array, which is slated for completion in 2016. With 64 dishes, MeerKAT has almost 10 times as many antennas as KAT-7. This means that it will require roughly 100 times the signal processing to handle the data, which will be coming in at more than 5 terabits per second.

A key advantage of using Simulink and Model-Based Design for this effort is that designs will be easily retargeted to the next generation of ROACH boards when they become available. This retargeting enables us to take advantage of the more powerful

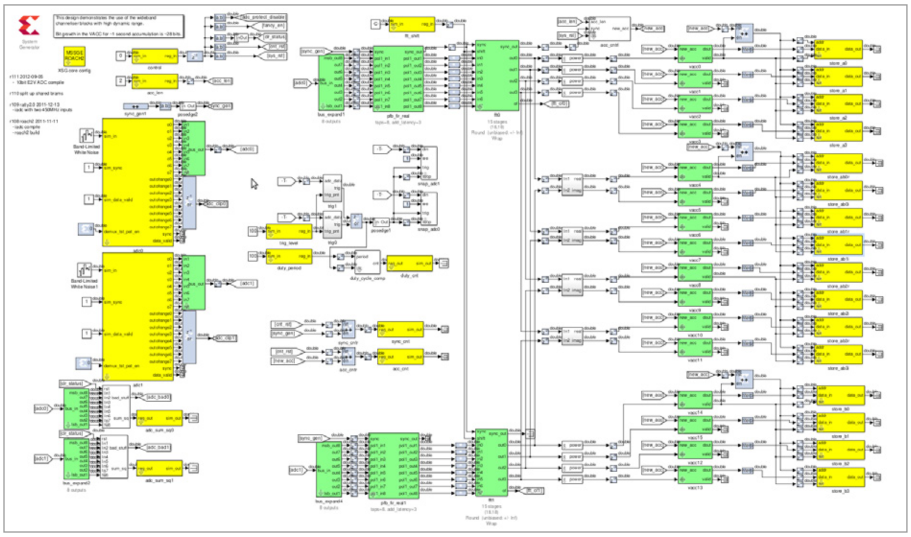


FIGURE 3. *A ROACH-II board.*

FIGURE 4. *Aerial view of the KAT-7 array.*

FPGAs that we expect to be available in the years to come. We anticipate using about 200 to 300 next-generation ROACH boards for MeerKAT.

## The Square Kilometre Array and Beyond

The Square Kilometre Array will have between 3000 and 4000 antennas, each about 15 meters in diameter. Combined, the dishes will have a square kilometer of surface area, making SKA the largest and most powerful radio astronomy telescope in the world—about 100 times more powerful than its closest competitor. SKA dishes will produce data at a rate equivalent to roughly 10 times the world's current Internet traffic, and the system will be so sensitive that it will be able to detect signals equivalent in strength to airport radar, on a planet 50 light-years away. Although construction will not begin until 2017, early signal processing design work has already begun, and the team has submitted a proposal to design the SKA's central processing system using Simulink and the CASPER library.

Our work with Simulink and the CASPER library is not only accelerating the development of radio astronomy instruments for MeerKAT and SKA; it is also helping researchers across the globe. Prior to CASPER, it was commonplace for the development of new instruments to take five years or more; with CASPER we have seen groups develop eight instruments in less than two years.

A principal benefit of using Simulink and CASPER is that beginners can become productive designers very quickly. We run workshops on the use of Simulink and CASPER. Attendees are typically hardware design novices, but after a week, they know enough to design their own instruments by dragging and dropping blocks in Simulink and deploying them to ROACH hardware with a single click. In fact, each year some of the trainees develop a new, working instrument during the weeklong training course. That kind of productivity is driving interest and enthusiasm in our field, and it is accelerating advances in radio astronomy not only in South Africa, but worldwide. ▪

## Learn More

**Square Kilometre Array**
*ska.ac.za*

**Developing Radio Astronomy Instruments with Simulink Libraries**
*mathworks.com/radio-astronomy*

# Deploying Standalone and Web-Based MATLAB Applications to Improve the Steel Manufacturing Process

By Mika Judin, Ruukki

The mass production of high-quality steel products requires a combination of time-tested manufacturing practices and modern technology. As steel coils pass through the rolling, galvanizing, color coating, and other lines in a steel manufacturing plant, operators must set oven temperatures, line speeds, and alignments based on the characteristics of each coil and the desired thickness and flatness profile. Failure to properly set up a line results in excess scrap. It can also damage the ovens and cause the line to be shut down for several days.

At Ruukki Metals, we built and deployed a web application with MATLAB® that enables operators to select and apply the proper settings throughout the steel manufacturing process. We built a second, standalone MATLAB application that our engineers use to aggregate and analyze production metrics from multiple databases, track individual coils, and refine our process.

Since deploying these applications, we have had a much more consistent and efficient process, with less scrap, improved flatness, and a shorter off-gauge length. For example, in the temper mill, the off-gauge length—the amount of a coil that does not achieve the target thickness—has been reduced from several meters to 50 centimeters or less. Key to these improvements were regular optimizations of setup calculation parameters, and data visualization in MATLAB. The power of these MATLAB visualizations for detecting deviant coils cannot be exaggerated.

## Identifying Potential Process Improvements

Line speeds and temperatures must be adjusted to keep the thickness and flatness of each coil within required tolerances. Before MATLAB based web applications were available, our operators relied on their own experience, personal notes, and judgment for this work. With multiple shifts running each day, this practice led to inconsistent results.

Once coils had been processed, it was difficult and time-consuming for engineers to determine what settings had been used for any particular coil or set of coils. For example, to check the thickness at the cold-rolling mill, the output after galvanizing, and the transverse thickness profile, they typically spent days collecting the necessary data, processing it, and producing the plots needed to understand the results.

## Analyzing Big Data with MATLAB and Neural Networks

At the heart of our new process is a set of data warehouses that we use to store and access information about the coils as they pass through the plant. A Microsoft® SQL based data warehouse stores thickness tolerances, dimensions, the raw material class for each coil, and the coil's intended customer. A Wonderware® Historian data warehouse stores time-series data for the coil's thickness and flatness and other process measurements. An Oracle® based data warehouse stores

*Cold-rolled steel coils. Ruukki Metals processes 60,000 coils annually. To ensure quality, engineers use a MATLAB application to aggregate and analyze up to 4000 different measurements for each coil.*

defects or anomalies detected and test results in a coil. All together, up to 4000 different measurements may be stored for each of the 60,000 coils processed annually.

With MATLAB and Database Toolbox™, we developed an application that retrieves data from each database, merges it in a separate Microsoft Access database, and creates documentation as needed. When a new coil is about to be processed in the line, this application analyzes the merged and stored data to calculate oven temperatures and other parameters used to set up equipment. In the galvanneal process, for example, the application uses a neural network created with Neural Network Toolbox™ to calculate setup values.

We relied on Neural Network Toolbox to implement a number of other key application features. We used self-organizing maps to classify coils by zinc mass, iron percentage, and flatness (Figure 1).

We also used Neural Network Toolbox to create a neural model. The model generates roll gap predictions based on finite element methods (FEM) (Figure 2).

In total, we created more than 30 different MATLAB applications for analyzing, visualizing, and exporting data for coils in the production chain. This data includes thickness, flatness, transverse profiles, raw material quality, and other characteristics (Figure 3).

## Deploying Applications

We used MATLAB Compiler™ to create a desktop application that provides easy access to the data analysis and visualization applications in MATLAB that we had developed. Thanks to our use of JDBC drivers, no ODBC database connections were needed, and we did not have to use database wizards to create a database connection for the applications. Engineers can install and run this application from any PC without having to install MATLAB. Currently, more than 20 engineers are using it to gain insights into how our production chain is working today and how it might be improved in the future. Using this

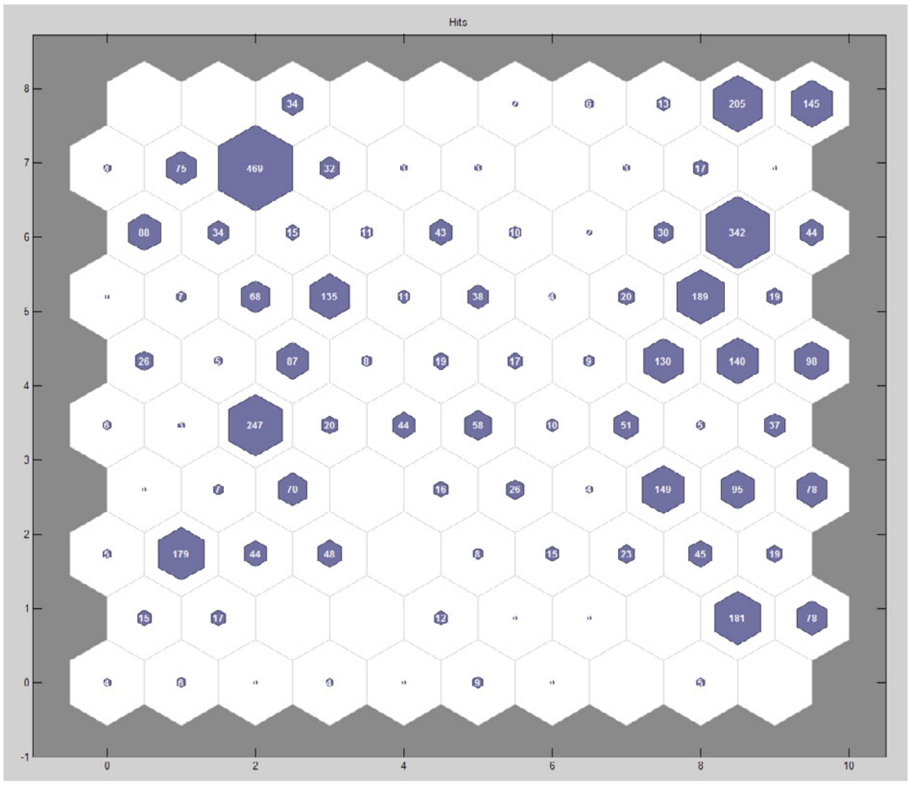


FIGURE 1. *A self-organizing map created with Neural Network Toolbox. Cluster 83 in the upper left shows that there were 469 coils with similar flatness characteristics.*

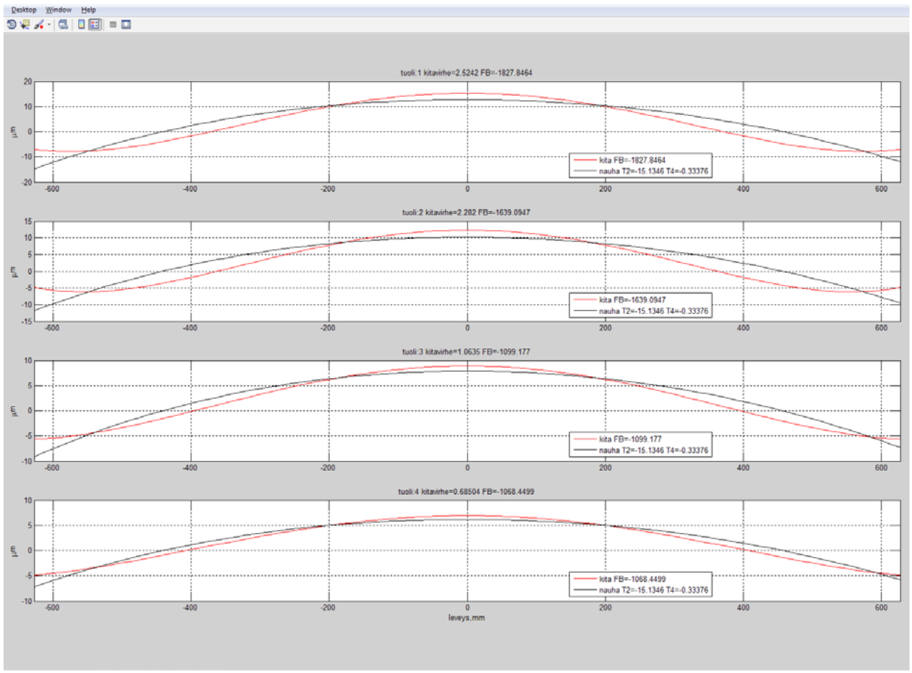

Desktop Window Help
tuoli:1 kitavirhe=2.5242 FB=-1827.8464
tuoli:2 kitavirhe=2.282 FB=-1639.0947
tuoli:3 kitavirhe=1.0635 FB=-1099.177
tuoli:4 kitavirhe=0.68504 FB=-1068.4499
kita FB=-1827.8464
nauha T2=-15.1346 T4=-0.33376
µm
leveys,mm

FIGURE 2. *Plots showing roll gap profile predictions.*

application, analyses that took days to complete manually can be completed in less than a minute.

We also created a web-based application, accessible via any web browser on our network, that line operators can use to view the data they need to monitor and set up the line (Figure 4). To build this .NET application, we packaged our MATLAB code as a DLL using MATLAB Builder™ NE. The DLL retrieves information from our database and creates plots that are sent as bitstreams to the main application, hosted on a Microsoft Internet Information Services (IIS) web server.

Using this application, our operators are prepared to make the necessary adjustments before each new coil enters a line. Since deploying this application and the standalone executable that we created with MATLAB and MATLAB Compiler, we have seen fewer misalignments, less scrap, and significant increases in efficiency and consistency at the plant.

Recently, we used MATLAB to model the bending process during cold rolling in the temper and tandem mill, and we performed simulations that have enabled us to improve the flatness of coils coming out of this process. MATLAB provides a powerful tool for accessing FEM models like COMSOL Multiphysics®. We have created several applications that use a COMSOL-MATLAB link that enables us to input values to an FEM model and visualize results from COMSOL. ▪

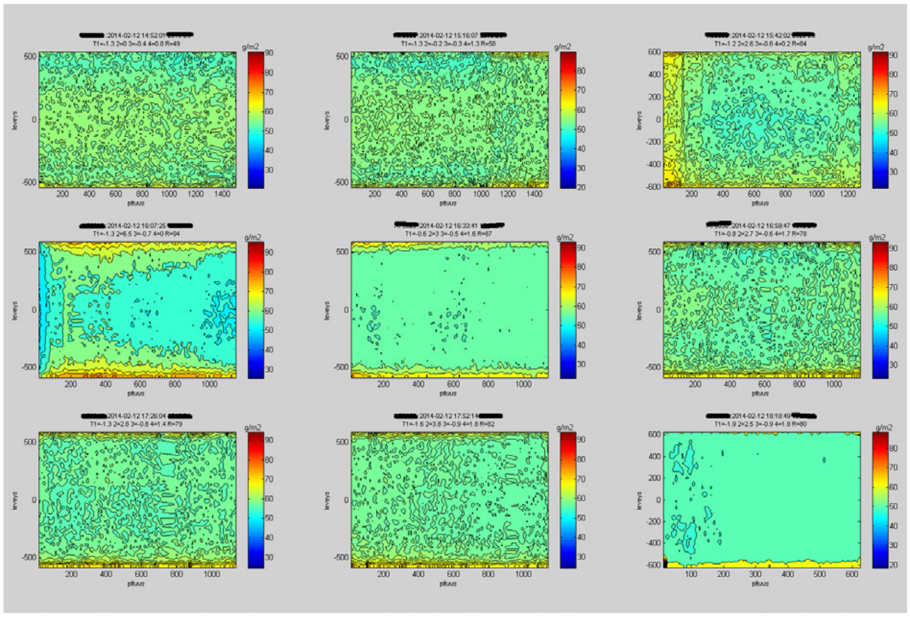
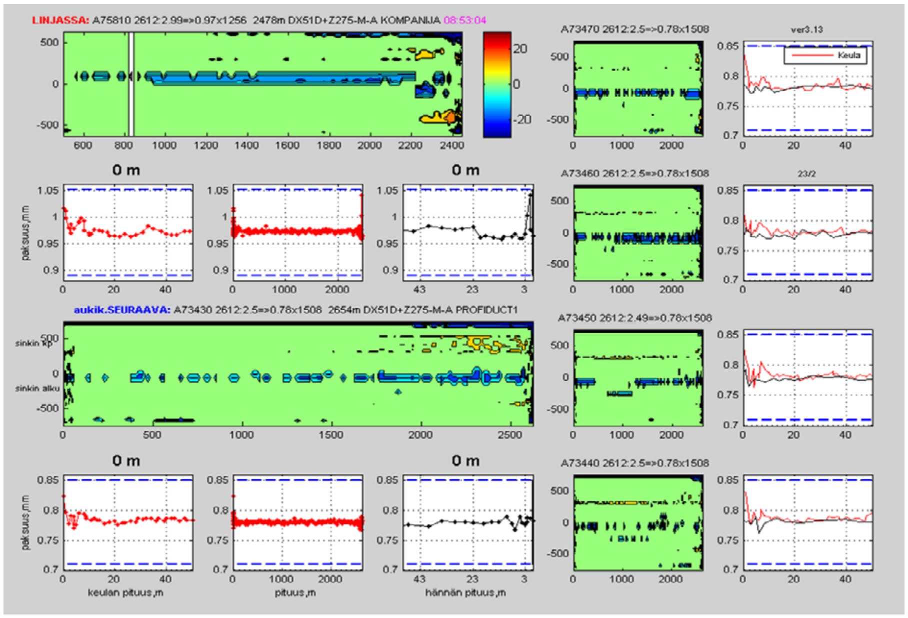


Figure 3. *Zinc mass visualization, created in MATLAB.*


LINJASSA: A75810 2612 2.99=>0.97x1256 2478m DX51D+Z275-M-A KOMPANIJA 08:53:04
A73470 2612 2.5=>0.78x1508
ver3.13
Keula
A73460 2612 2.5=>0.78x1508
aukik.SEURAAVA: A73430 2612 2.5=>0.78x1508 2654m DX51D+Z275-M-A PRODUCT1
A73450 2612 2.49=>0.78x1508
A73440 2612 2.5=>0.78x1508
0 m
paksuus,mm
keulan pituus,m
pituus,m
hännän pituus,m
sinkin alku
Figure 4. *The web application, built with components created in MATLAB Builder NE, running in a browser.*

## Learn More

**HKM Optimizes Just-in-Time Steel Manufacturing Schedule**
*mathworks.com/hkm*

**Example: Thickness Control for a Steel Beam**
*mathworks.com/thickness-control*

**Webinar: Application Deployment with MATLAB**
**22:58**
*mathworks.com/wbnr-81715*

# Inverting the Robotics Classroom with a Massive Open Online Course

By Magnus Egerstedt, Georgia Institute of Technology

Electrical and computer engineering students at Georgia Institute of Technology learn embedded control design in *ECE 4555: Embedded and Hybrid Control Systems*. While the course covers embedded control systems of all types, it places a heavy emphasis on mobile robots. Robotics helps bridge the gap between theory and practice in a way that engages and motivates students like little else.

In the most recent version of the course, students had more time to work with robots in class, thanks to an inverted (or flipped) classroom and a massive open online course (MOOC). More than 40,000 students participated in the MOOC worldwide, including the 30 third- and fourth-year students in *ECE 4555* at Georgia Tech. The videotaped lectures and online assignments in the MOOC enabled me to flip the on-campus classroom: Students learned theory from MOOC videos, and spent classroom time developing controllers in MATLAB® and working with actual robots.

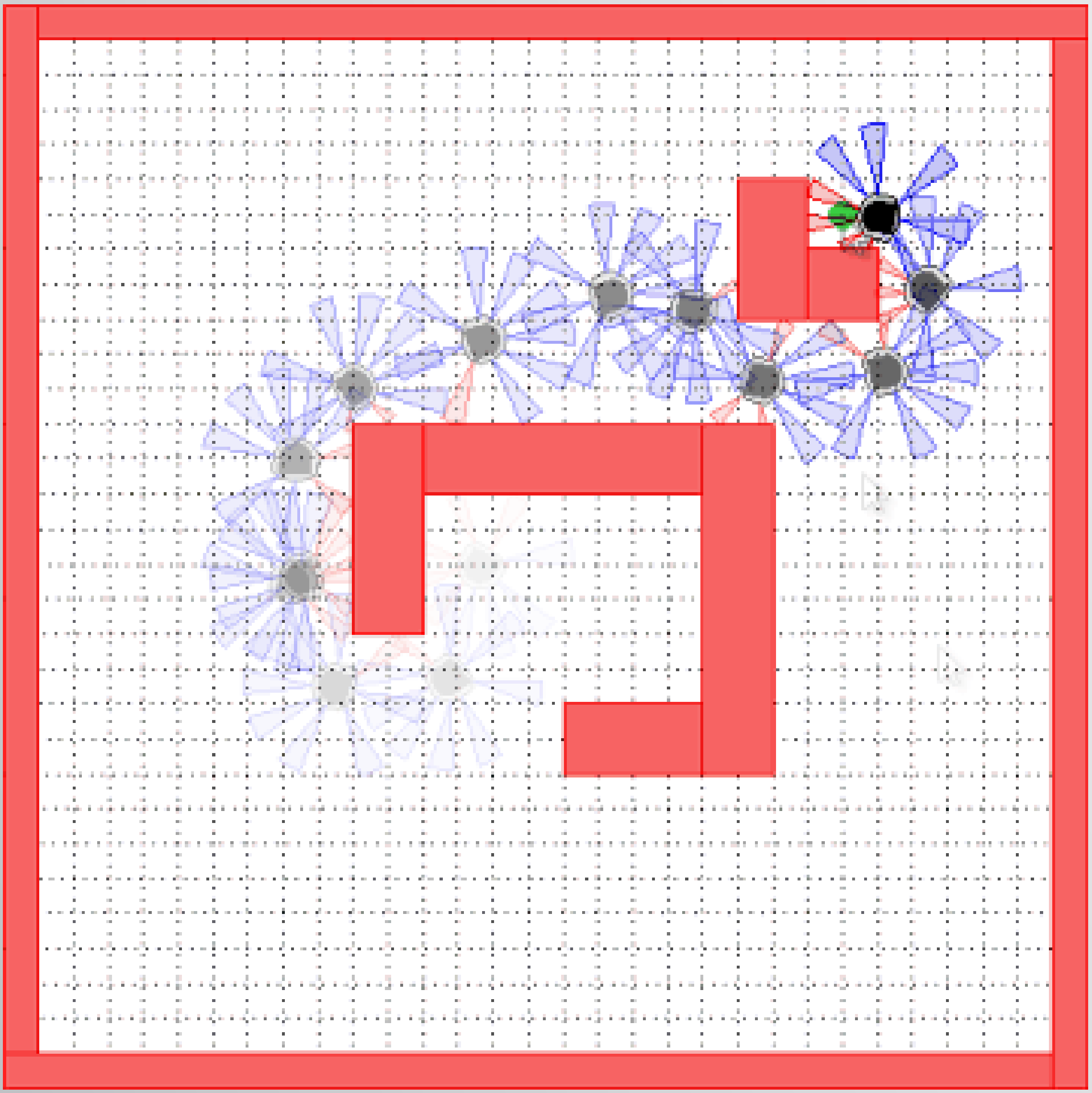
The results were outstanding. In all my years of teaching, I've never seen such a high level of energy and enthusiasm—even though the flipped classroom demanded more time than a typical three-credit class. In class surveys, students gave the course an average of 4.9 points out of 5, and although the MOOC was far more advanced than most online courses, more than 5000 students completed it, far exceeding the average retention rate.

## Using a MOOC to Invert the Classroom

When I started teaching *ECE 4555* I focused on general embedded control, but soon moved into robotics as the primary application domain. The first time that we taught the course with actual robots, I was so pleased with the results that I decided to add more robotics—and a much richer set of problems. I found, however, that as the assignments grew more complex, the students stopped being systematic in their design approach. I was faced with a dilemma. I wanted lots of robotics in the classroom, but I still needed time to teach the theory and make it both meaningful and accessible.

I flipped the classroom so that the students could learn the background material and theory outside class and spend class time working with me and my teaching assistant to develop control systems and test them on Khepera III robots (Figure 1).

Once I had decided to use an inverted classroom, the next question was, "Where will the material come from?" At that point, I opted to jump in with both feet and create a MOOC, *Control of Mobile Robots*. Since I would be depending on the MOOC to teach my on-campus students, I could not water down the material. I developed seven weekly modules, each with eight sublectures and a homework assignment covering topics such as linear



Students in ECE 455: Embedded and Hybrid Control Systems put theory into practice in their final control design projects. Among the most impressive designs this year was a simultaneous localization and mapping (SLAM) robot that automatically builds a map of its environment as it explores.

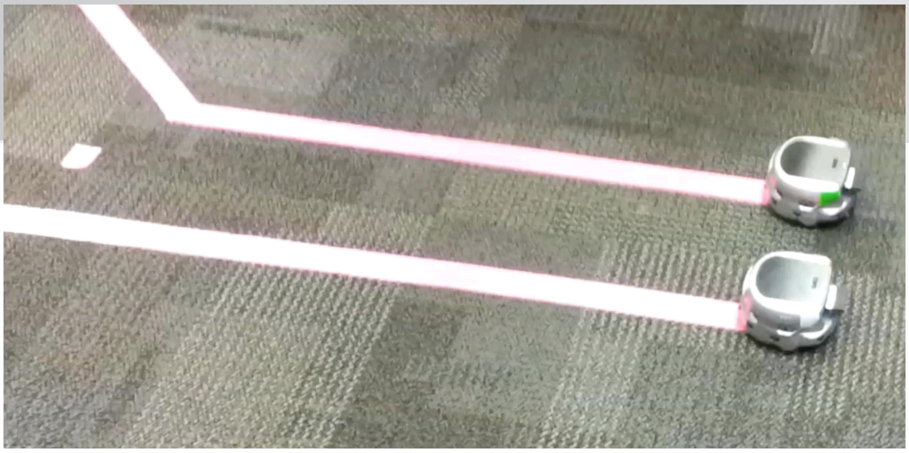


FIGURE 1. *Khepera robots, programmed by students in ECE 4555.*

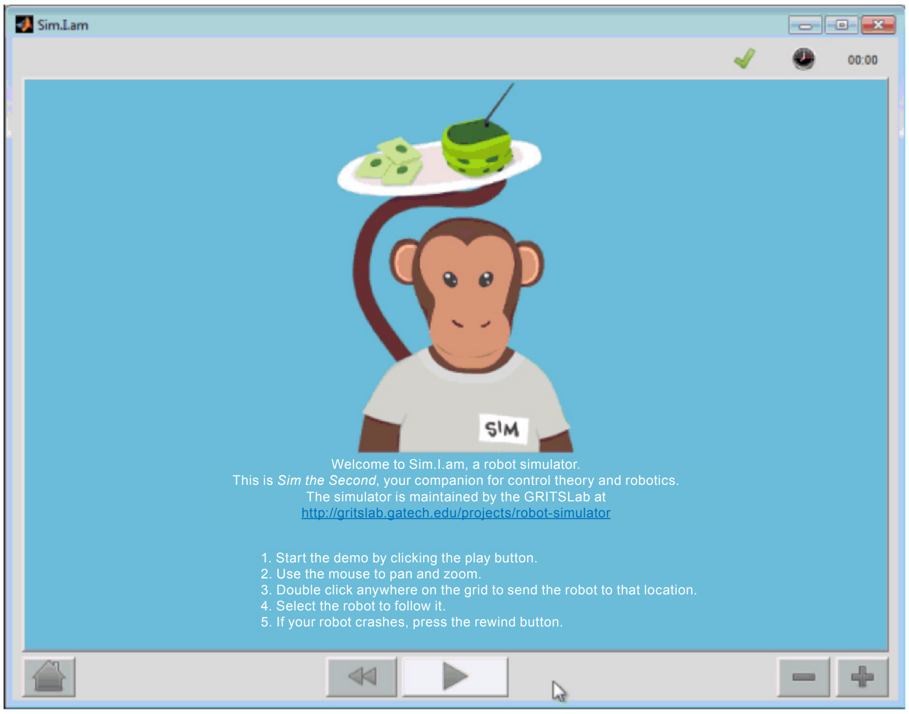
systems, control design, hybrid systems, and navigation. To make the MOOC broadly accessible, I used MATLAB to present control design examples in the lectures and as extra programming assignments, but did not require students to use MATLAB to complete assignments.

## Introducing the Sim.I.am Simulator

In *ECE 4555*, I faced a challenge common to many similar undergraduate courses: I had far more students than robots. Even working in groups, students had to spend much of their limited time with their robot debugging the control program instead of optimizing and enhancing it. The solution was to build a robot simulator in MATLAB. Developed by a teaching assistant, the Sim.I.am simulator enables students to write control programs in MATLAB and debug them in a virtual world (Figure 2). The same MATLAB code can then be used without modification to control a Khepera robot in the lab. The connection between simulation and actual robots is made possible by a thin TCP/IP server on the robots that enables the simulator to send the control signals from the controllers to the Khepera robots instead of their simulated counterparts.

Sim.I.am was instrumental to the success of the MOOC, as well. Online students who had MATLAB used the simulator to complete optional programming exercises. Students without access to MATLAB could download a standalone version of the simulator created with MATLAB Compiler™. While the students could not develop their own control programs using this version, they could experiment by setting controller gains to tune the system and see how their choices affected the behavior of the simulated robot.

Sim.I.am is designed to be adaptable to a variety of use cases. It can easily be modified to support other types of differential drive robots. It has a Simulink® interface that can be used as an alternative to programming in MATLAB. In addition to relying on the simulator as a teaching tool, I use it in my own research.



Welcome to Sim.I.am, a robot simulator.
This is *Sim the Second*, your companion for control theory and robotics.
The simulator is maintained by the GRITSLab at
http://gritslab.gatech.edu/projects/robot-simulator

1. Start the demo by clicking the play button.
2. Use the mouse to pan and zoom.
3. Double click anywhere on the grid to send the robot to that location.
4. Select the robot to follow it.
5. If your robot crashes, press the rewind button.

FIGURE 2. *The Sim.I.am robot simulator.*

## Programming Robots in MATLAB

After two weeks of in-class lessons on embedded controls, my on-campus students switched to the MOOC videos to learn new concepts. They came the first class session each week prepared to apply those concepts by developing MATLAB programs to run in Sim.I.am. In the second session, they tested and refined the programs on the Khepera robots. I did not have to spend class time teaching MATLAB because they had all learned programming with MATLAB in *CS 1371: Computing for Engineers*, a required course for all engineering majors at Georgia Tech.

For their first in-class assignment, students used MATLAB to program the robot to steer from a starting point to a goal using the movement of the robot's actuators to estimate its change in position. They created a single input, single output proportional integral derivative (PID) design, tuning it to ensure that the robot did not oscillate as it drove, and that the actuators did not saturate. While actuator saturation is simulated by Sim.I.am, effects such as skidding and slippage are not. As a result, students learned to cope with real-world effects as they worked to improve algorithms that performed well in the simulator but less so on the robot.

In the second assignment, they had to make the robot move and turn away from any obstacle that it encountered. This assignment required them to write a MATLAB program that controlled the robot based on data acquired from the robot's sensors.

Next, the students had to combine what they had learned in the first two assignments to design a controller that could get a robot to its goal while avoiding obstacles. For this objective, classic control design techniques are no longer sufficient; the system must switch between different controllers. In the following weeks, we added cul-de-sacs and multiple objects to the robots' environment. To enable the robots to navigate these more complex environments, students implemented sliding mode controls. They used Control System Toolbox™ to explore and analyze design ideas

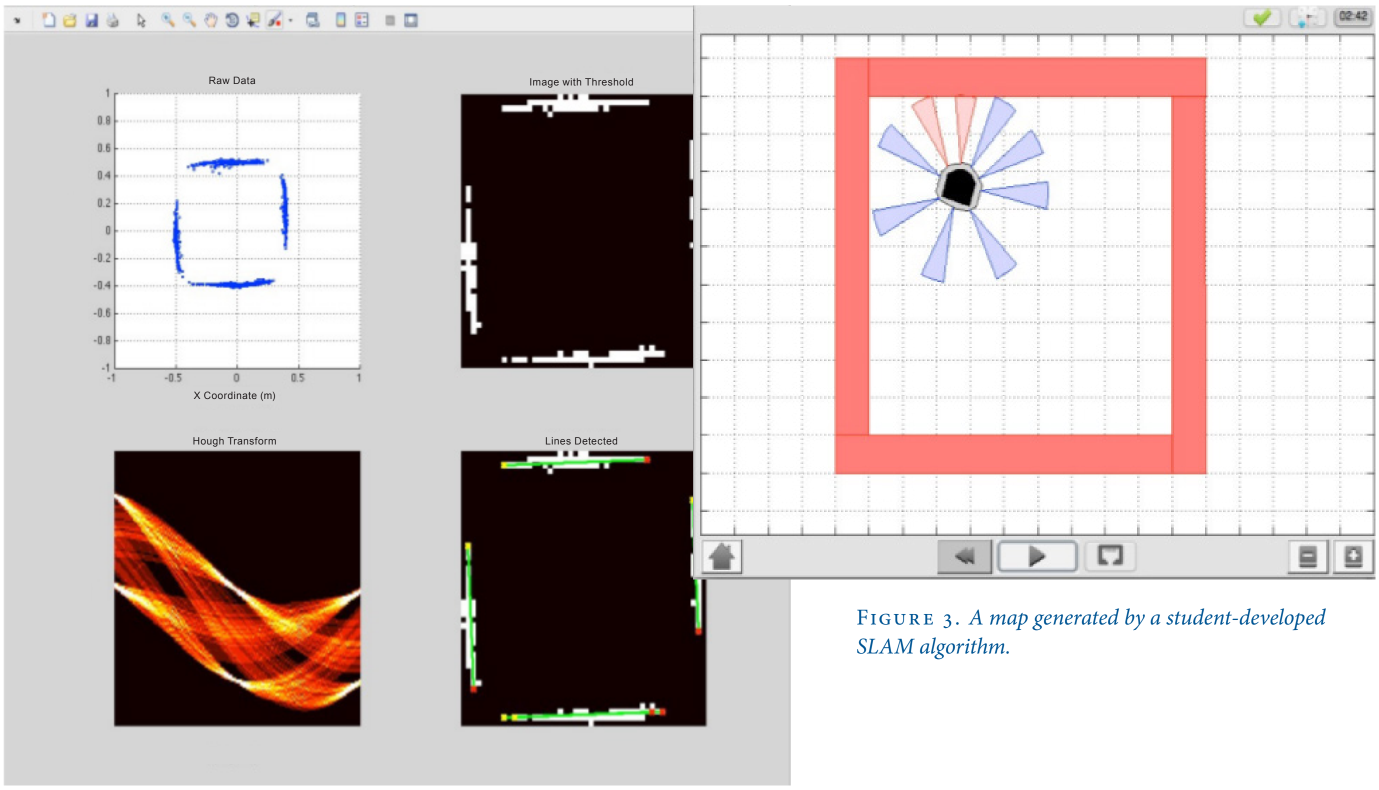


FIGURE 3. *A map generated by a student-developed SLAM algorithm.*

before implementing them in code.

When the seven-week MOOC ended, I had planned to spend the four weeks remaining in the Georgia Tech semester lecturing on supervisory control topics. However, the students were so engaged and excited to work with the robots that I opted instead to use that time to expand their final design projects. Each student group came up with their own project idea (subject to instructor approval), and implemented it using MATLAB, Sim.I.am, and the Khepera robots. Among the most impressive were a program that moved multiple robots in formation and a simultaneous localization and mapping (SLAM) design that automatically built a map of its environment as it explored (Figure 3).

## Gaps Bridged and Lessons Learned

Practicing engineers apply theory to make something happen in the real world. In ECE 4555, the students learned to do the same. They also learned several widely applicable engineering skills that transcend embedded controller design. Like practicing engineers, they argued about control designs and learned to identify the best ones.

In many group efforts, one student does all the work. This was not the case in *ECE 4555*. On many teams *every* student took the initiative to develop a solution, and then the team picked one to develop and optimize. Some teams were initially inclined to hack together a solution. They started fast, but soon discovered the value of slowing down to think through a design before rushing to implement it. They also learned how to assess and manage real-world effects. Such effects are easily dismissed when you are only doing simulations, and not actually smelling the burnt capacitor.

I'm now a firm believer in flipped classrooms, and the MOOC was invaluable to making ours a huge success. Other institutions have recognized the advantages as well; next year the University of Hawaii will flip an upper-level design course using *Control of Mobile Robots*.

On campus, I plan to incorporate the Simulink interface to Sim.I.am. Further improvements may include adapting it to work on a less expensive robotics platform. ▪

## Learn More

**Download: Sim.I.am app**
*mathworks.com/fx-40860*

**Sim.I.am Simulator**
*mathworks.com/blogs/sim-i-am*

**Motivating First-Year UC Berkeley Students to Learn Programming with a Virtual Robot Tournament**
*mathworks.com/learn-with-robot*

**Applied Project-Based Learning: Building Applications for Low-Cost Hardware**
*mathworks.com/building-applications*

# Adding Fun to First-Year Computer Programming Classes with MATLAB, Arduino Microcontrollers, and Model Trains

By Lowell Toms and Dustin West, Ohio State University

Ohio State University's First-Year Engineering course sequence is designed to give incoming students insight into most of the engineering disciplines offered at the university. The challenge is to keep the students interested and prevent that first year from turning into a painful slog through the core science and math courses. How do we solve that challenge? We let the students play with train sets while teaching them basic programming skills using MATLAB®.

The 1700 students who take *ENGR 1181: Fundamentals of Engineering I* each year not only learn engineering concepts and the basics of computer programming in MATLAB; they also complete a final project in which they develop a controller for an N scale train set using MATLAB and an Arduino® microcontroller. This project enables students to demonstrate what they've learned. It also gives them their first opportunity to experience the joy of engineering by seeing the results of their efforts come to life in the lab. We frequently see students jumping and cheering as they watch their control algorithms work for the first time.

## Learning Programming with MATLAB

Before they tackle the train project, students learn measurement, graphing, data analysis, and other core engineering concepts in parallel with the basics of programming. Lab assignments in MATLAB reinforce the programming lessons, which cover arrays, strings, loops, logical expressions, conditionals, and other constructs.

Most students come to the course with little or no programming experience. We have found that MATLAB is an excellent first language to learn. MATLAB is more interactive than C and similar languages, there is no need to compile, and there are far fewer low-level details for the students to master. Once they understand basic programming concepts in MATLAB, students learn to apply them in other languages quite rapidly. MATLAB is also ideal for *ENGR 1181* because it is used across all engineering disciplines at Ohio State.

## Controlling Trains with MATLAB and Arduino Hardware

Students begin the train project during the 10th week of the course. Their goal is to develop a MATLAB controller that works with an Arduino microcontroller to interact with a train as it travels around a circular track. Break-beam sensors separate the track into a "city" section and a "country" section (Figure 1). As the train enters the city, the MATLAB code must reduce the train speed, start the road crossing flashers, and then drop the crossing gate after a one-second delay. When the train leaves the city and enters the country, the MATLAB code must increase the train speed, turn off the flashers, and raise the gate.

Working in groups of four, the students use MATLAB and the MATLAB Support Package for Arduino Hardware to write a program that communicates with an Arduino Uno via a serial port. The program checks the status of the break-beam sensors via the Arduino's digital input, and uses the Arduino's digital

*Working in groups of four, students write a MATLAB program that reduces the train speed, starts the road crossing flashers, and drops or raises the crossing gate.*

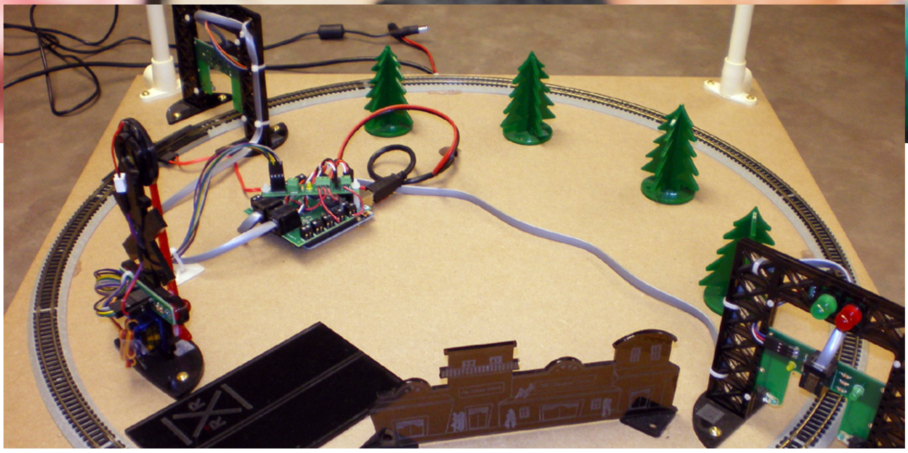


FIGURE 1. *N scale train setup in the lab.*

```
%Start a timer
    tic;

    %You know you are inside the approach gate so now you will
    %be checking the departure gate to see if you have passed
    dep = a.analogRead(departure);
    dep = a.analogRead(departure);
    dep = a.analogRead(departure);

    %This will run until you pass the gate.
    while dep < 300

        %Turn the crossing gate lights on to alternate
        %*****NOTE****
        %Extend this through about 6 or 7 seconds. If it takes
        %longer than 4.5 seconds for your train to pass between the
        %approach and departure gates, your lights won't blink after
        %4.5 seconds.

        if ((toc-floor(toc)) < 0.5)
            a.digitalWrite(rLED,1)
            a.digitalWrite(lLED,0)
        else
            a.digitalWrite(rLED,0)
            a.digitalWrite(lLED,1)
        end

        %Lower the crossing gate to horizontal 1.5 seconds
        %after the train goes through the approach sensor
        if (toc>1.5)
            a.servoWrite(1,170)
        end
```

FIGURE 2. *Student code using `tic()` and `toc()` to implement a non-blocking delay.*

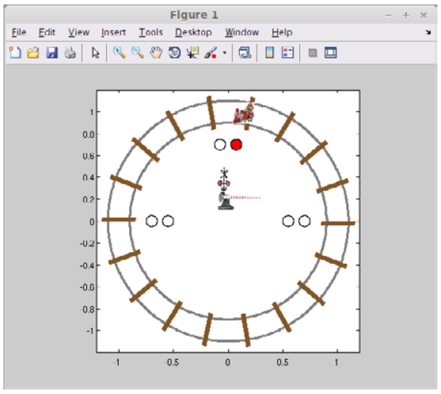


FIGURE 3. *The MATLAB train set simulator.*

output to flash the LEDs and operate a servo motor for raising and lowering the gate.

Over the course of three lab sessions, the students implement their MATLAB control algorithms. These typically consist of an endless polling loop with conditionals to check the status of sensors and logic to manage the servos and LEDs.

For many students, the most challenging aspect of the project is implementing a nonblocking delay, needed to flash the LEDs and lower the gate after one second. Students discover that they cannot use the MATLAB `pause()` function because the algorithm may miss sensor input while the pause is executing. Instead, they learn to use MATLAB `tic()` and `toc()` functions to time the delay without interrupting the polling loop (Figure 2).

## Building a Simulator for Debugging and Independent Work

Scheduling lab time for 1700 students was a significant challenge. Our lab has room for 18 tables, each capable of holding one train setup. With four-member teams, we could accommodate just 72 students at a time. Long waits for lab time were made even longer because students had to do all their debugging on an actual train set in the lab. Another concern was that one member of each team—usually someone with prior programming experience—tended to do most of the work while the other three watched.

To address these challenges, two graduate students developed a train simulator in MATLAB that emulates the train setups in the lab (Figure 3). The simulator uses a MATLAB figure window to draw the track, train, break-beam sensors, and gate. Students can use the simulator to debug their control algorithm code and then use the same code on the actual train set. Teams then spend lab time optimizing their algorithms instead of debugging, greatly reducing the amount of time each team needs with the actual trains. Further, we can now give assignments for students to complete independently using the simulator, ensuring that no single programmer does all the work.

## MATLAB in the Second Semester and Beyond

Engineering students continue to use MATLAB in *ENGR 1182: Fundamentals of Engineering 2*. In this course, the students must design and build an advanced energy vehicle that uses as little power as possible as it travels around a monorail track. Students analyze power consumption in MATLAB using voltage and

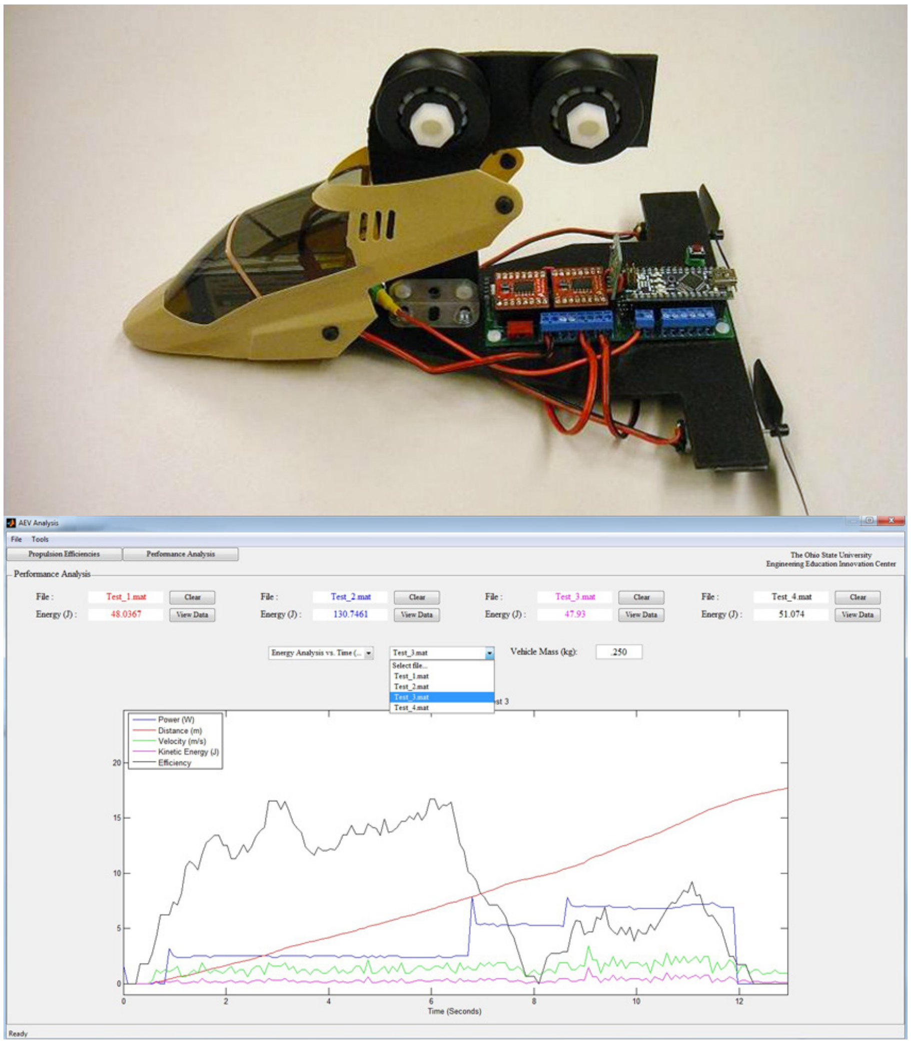


FIGURE 4. *A monorail vehicle and MATLAB interface used to plot power consumption, distance traveled, velocity, kinetic energy, and energy efficiency as a function of time.*

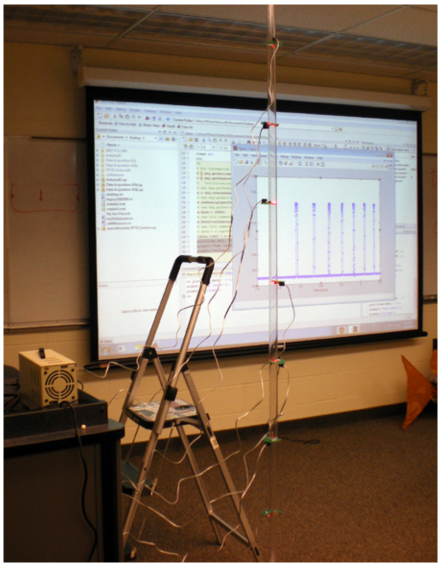


FIGURE 5. *DAQ setup for measuring the velocity of a dropped ball using MATLAB and Data Acquisition Toolbox.*

current measurements recorded by an Arduino Nano installed on the vehicles (Figure 4).

In upper-level courses and in their final-year capstone projects, students build upon the skills they've developed in MATLAB throughout the First-Year Engineering program. For example, in *ENGR 2167: Data Acquisition with MATLAB*, students explore the use of data acquisition (DAQ) devices, including the USB-6009 and myDAQ from National Instruments. During one exercise, a ball is dropped through a vertical tube lined with break-beam sensors (Figure 5). Using a DAQ device in session mode with Data Acquisition Toolbox™, we collect data from the sensors at 20,000 samples per second. We use MATLAB to compute the ball's velocity as it falls, and compare the result to the velocity of a ball dropping in a vacuum, calculated from Newton's laws.

## High-Fives and Other Positive Feedback

By the end of their first year, the students have a solid grasp of data analysis and programming with MATLAB and understand how vectors, indexing, loops, and conditionals work. The train project has led not only to high-fives in the lab, but also to very favorable course reviews from our students.

Since the adoption of the First-Year Engineering course sequence with MATLAB, a higher percentage of students are opting to continue in the engineering program. We believe that the increased retention rate is partly due to the students having experienced the satisfaction of facing and overcoming a genuine engineering challenge in their first year. ▪

## Learn More

**Teaching Computer Programming to First-Year Undergraduates with a MATLAB Based Robot Simulator**
*mathworks.com/robot-simulator*

**Classroom Resources: Programming and Computer Science**
*mathworks.com/programming-computer-science*

# Using Modeling and Simulation to Test Designs and Requirements

By Michael Carone, MathWorks

Modeling is an efficient and cost-effective way to represent a real-world system. A model can represent key aspects of the system, including the underlying requirements, the components, and how those components communicate with one another. The model can be simulated, enabling designers to test designs before hardware is available, or to test conditions that are either difficult or expensive to replicate in the real world. Iterating between modeling and simulation can improve the quality of the system design early, reducing the number of errors found later in the design process.

Despite these advantages, designers who heavily rely on hand coding do not always take full advantage of modeling and simulation. Setting up tests can be difficult and time-consuming, and when separate tools are used for each domain, it can be challenging to obtain a system-level view of the design. As a result, defects that could have been found in the modeling and simulation phase are often found during the implementation phase, when defects are more expensive to fix.

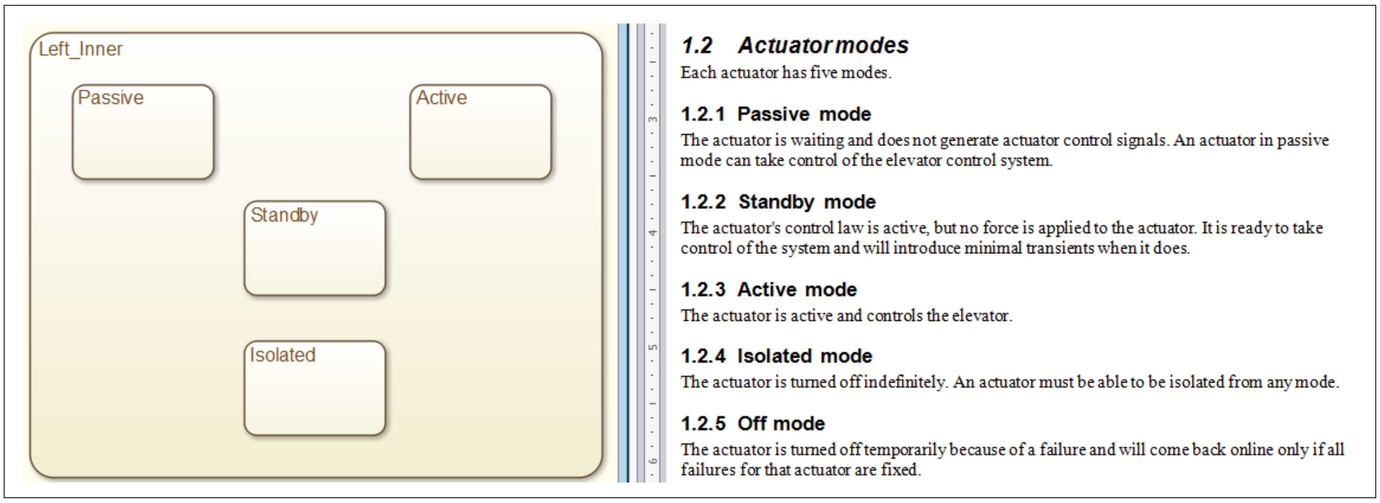
These issues are addressed in Simulink®, a platform for modeling and simulation. Simulink supports not only multidomain modeling but also simulation, with its own set of ordinary differential equation (ODE) solvers. A fundamental advantage of using Simulink is that you can represent different domains, including control systems, state machines, and environmental models, in one model, and then run simulations within Simulink to verify that the model is built correctly. As the simulation runs, you have access to simulation analysis capabilities, such as data displays, state animation, and conditional breakpoints. After the simulation is completed, you can analyze any logged data with MATLAB® scripts and visualization tools.

In this article, we describe a workflow for building a component model from requirements, simulating and testing that component model, and then connecting it to a system-level model for further simulation and testing. To illustrate this workflow we will build and test the fault detection, isolation, and recovery (FDIR) component of the HL-20, a re-entry vehicle designed by NASA to complement the Space Shuttle orbiter. We will connect our component to a system-level model that includes environmental models, flight controls, and guidance, navigation, and controls (GN&C) systems, and then simulate the system-level model to validate its behavior.

The model used in this example is available in Aerospace Blockset™.

## Building the Component Model from Requirements

The first step is to model the fault management logic of the actuator system. The requirements document specifies five possible modes for the actuator: passive, standby, active, isolated, and off. For simplicity, we will consider the first four modes only. We represent these modes by adding four states to a Stateflow® state diagram (Figure 1).



**1.2 Actuator modes**
Each actuator has five modes.

**1.2.1 Passive mode**
The actuator is waiting and does not generate actuator control signals. An actuator in passive mode can take control of the elevator control system.

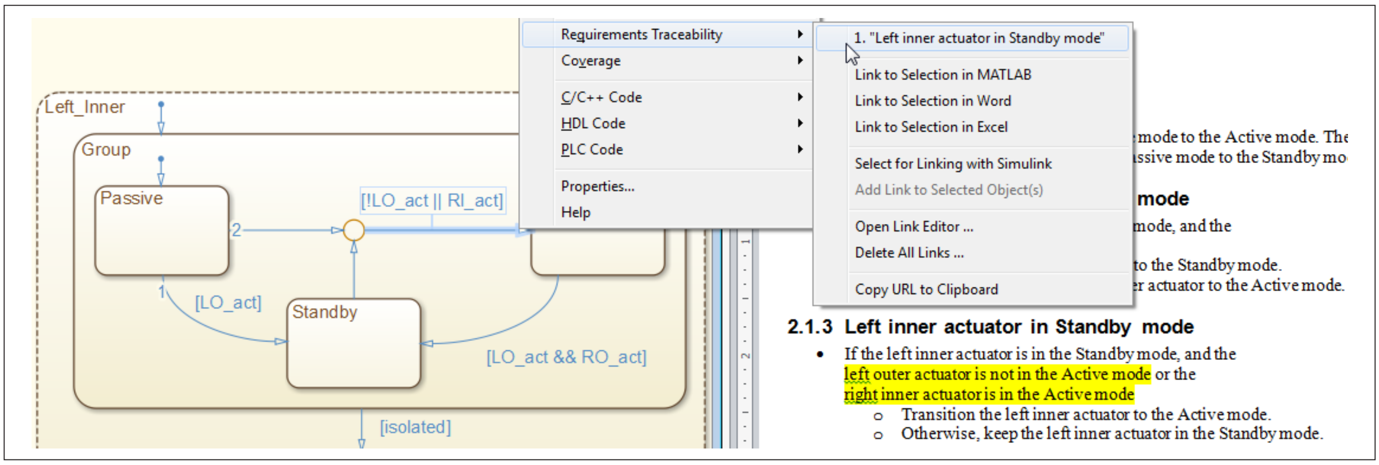
**1.2.2 Standby mode**
The actuator's control law is active, but no force is applied to the actuator. It is ready to take control of the system and will introduce minimal transients when it does.

**1.2.3 Active mode**
The actuator is active and controls the elevator.

**1.2.4 Isolated mode**
The actuator is turned off indefinitely. An actuator must be able to be isolated from any mode.

**1.2.5 Off mode**
The actuator is turned off temporarily because of a failure and will come back online only if all failures for that actuator are fixed.

FIGURE 1. *Stateflow diagram showing actuator modes represented by states.*


Left_Inner
Group
Passive
Standby
2
1
[!LO_act || RI_act]
[LO_act]
[LO_act && RO_act]
[isolated]
Requirements Traceability
Coverage
C/C++ Code
HDL Code
PLC Code
Properties...
Help
1. "Left inner actuator in Standby mode"
Link to Selection in MATLAB
Link to Selection in Word
Link to Selection in Excel
Select for Linking with Simulink
Add Link to Selected Object(s)
Open Link Editor ...
Delete All Links ...
Copy URL to Clipboard

mode to the Active mode. The
ssive mode to the Standby mo

mode
mode, and the

to the Standby mode.
r actuator to the Active mode.

**2.1.3 Left inner actuator in Standby mode**

- If the left inner actuator is in the Standby mode, and the left outer actuator is not in the Active mode or the right inner actuator is in the Active mode
  - Transition the left inner actuator to the Active mode.
  - Otherwise, keep the left inner actuator in the Standby mode.

FIGURE 2. *Requirements linked to specific parts of the state diagram.*

Next, we need to determine how the system will transition from one state (or mode) to another. Using the information provided in the requirements document, we add transitions connecting the states, and specify which conditions need to be satisfied for the system to switch states. We also group the Passive, Active, and Standby states together in a superstate, since they all transition to the Isolated state under the same condition. This hierarchical modeling technique helps us to model complex logic in a simple visual form.

We continue to build the model, connecting each element to a specific system requirement (Figure 2). Later we will be able to trace our model back to the requirements document to explain why a design choice was made.

Once we have built up the logic for the left inner actuator, we can reuse this design for the right inner actuator, since the structure is exactly the same. The only elements that need to be changed are the conditions that guard each transition, as described in the requirements document.

## Testing the Component Through Simulation

Now that the component is partially built, we are ready to run simulations to verify that it is behaving correctly. To do this, we set up a simple test harness that feeds input signals into the component using a combination of switch and constant blocks.

With Simulink and Stateflow we can start the simulation without having to manually define variables. When we press the **Play** button, a dialog box appears showing which variables need to be defined before the simulation

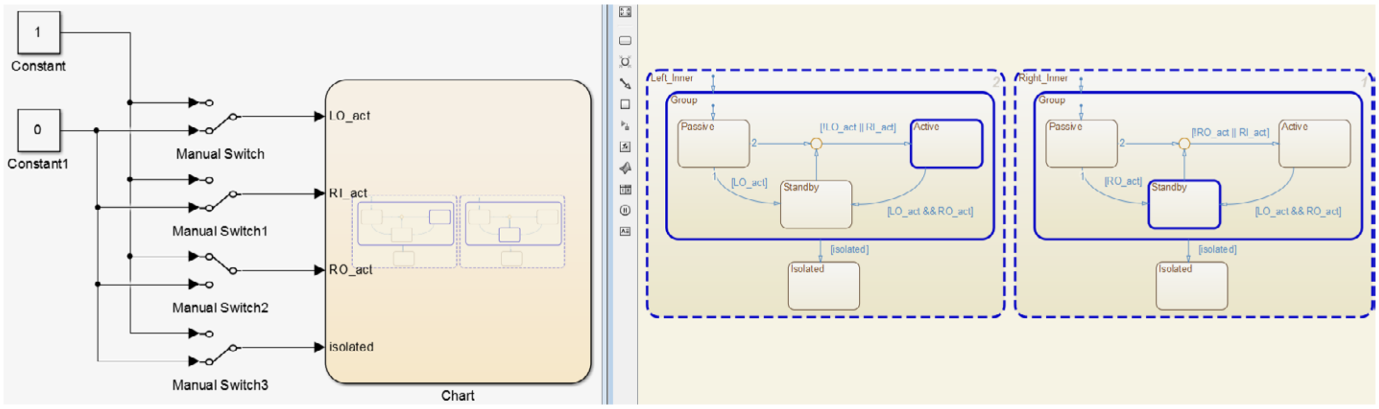


FIGURE 3. *State animation showing a defect in the design.*

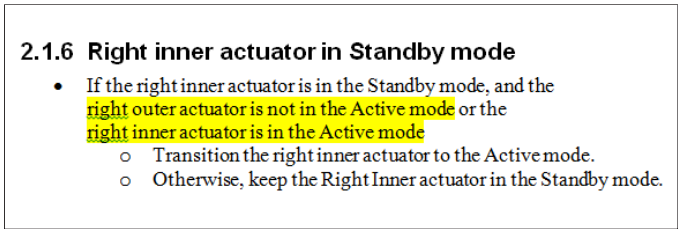
can run. When we press **OK**, those variables are automatically created.

As the simulation runs, the state diagram animates, letting us know which state is active at any given time and how the system is transitioning from state to state. Ad-hoc testing by switching input signals on or off reveals a flaw in the design (Figure 3). When the left inner actuator is activated, the right inner actuator should be activated, as well. The fact that we were able to set the input conditions so that this did not happen indicates that there is a flaw in our design.

It turns out that the condition on the transition from Active to Standby contains a defect. Since we have linked each condition to a requirement, we can trace that condition to the underlying requirement and verify that the defect originated from the requirements document and not from the design (Figure 4).

The last line should read “or the left inner actuator is in the Active mode.”

We fix the language in the requirements document, revise the condition, simulate the model again, and verify that the system now behaves correctly in response to the input signals.



**2.1.6 Right inner actuator in Standby mode**

- If the right inner actuator is in the Standby mode, and the right outer actuator is not in the Active mode or the right inner actuator is in the Active mode
  - Transition the right inner actuator to the Active mode.
  - Otherwise, keep the Right Inner actuator in the Standby mode.

FIGURE 4. *Design defect traced back to requirements document.*

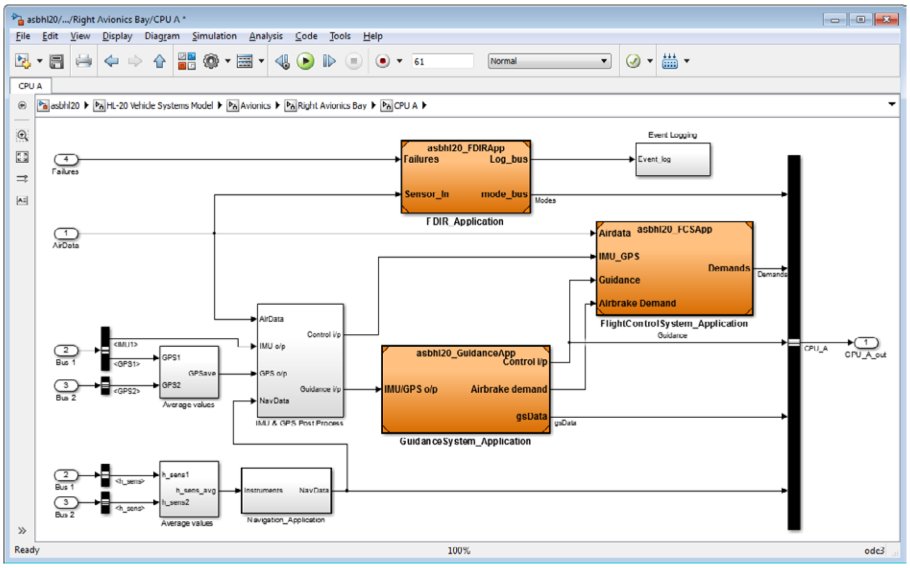


FIGURE 5. *System-level model showing three referenced component models: flight control system, FDIR logic, and guidance system.*

## Testing the System with the New Component

Now that the FDIR component has been tested on its own, we are ready to test it in the system-level model. We bring the component into the model as a Model block named FDIR_application. Once the block is integrated into the system model, we can continue to work on it independently from the rest of the system using the model referencing capability in Simulink (Figure 5).

We simulate the system-level model and visualize the behavior of the component in the state diagram, as well as the behavior

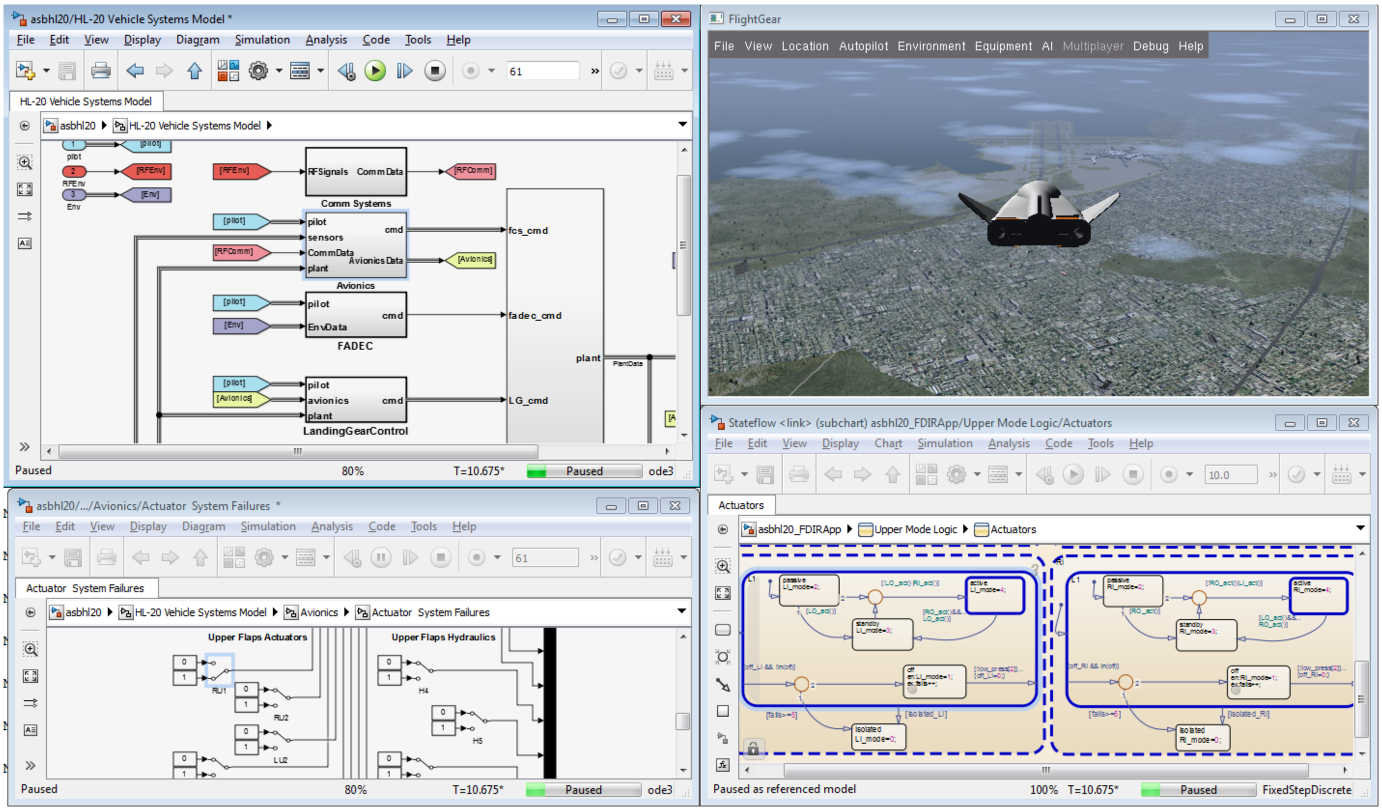

asbhl20/HL-20 Vehicle Systems Model *
FlightGear
asbhl20/.../Avionics/Actuator System Failures *
Stateflow <link> (subchart) asbhl20_FDIRApp/Upper Mode Logic/Actuators
Comm Systems
Avionics
FADEC
LandingGearControl
Upper Flaps Actuators
Upper Flaps Hydraulics
Paused
T=10.675*
ode3
Paused as referenced model
FixedStepDiscrete

FIGURE 6. *Simulation of the multidomain system-level model.*

of the overall system, using FlightGear, an open-source visualization tool. To test the system, we set up a harness that injects faults into the actuator system so that we can verify that both the component and overall system respond correctly (Figure 6).

## Next Steps

So far, we have built up a component from requirements, simulated and tested that component, and then connected it to a system-level model for additional simulation and testing. There are a number of additional steps that we can take to enhance the modeling and simulation workflow. For example, we can:

- Speed up simulation performance using the Performance Advisor in Simulink
- Implement a formal testing and verification process with design proofs, coverage analysis, and test-case generation
- Replace blocks with connections to hardware as hardware becomes available

Whichever step you choose next, the key is to model, simulate, and test the system as frequently and as early as possible to find and fix defects early to reduce overall system development cost. ▪

### Learn More

**Webinar: Simulink and Stateflow for Simulation Engineers** 48:04
*mathworks.com/wbnr-81993*

**Modeling and Simulation**
*mathworks.com/modeling-simulation*

# Systematic Fraud Detection Through Automated Data Analytics in MATLAB

By Jan Eggers, MathWorks

As the Madoff Ponzi scheme and recent high-profile rate-rigging scandals have shown, fraud is a significant threat to financial organizations, government institutions, and individual investors. Financial services and other organizations have responded by stepping up their efforts to detect fraud.

Systematic fraud detection presents several challenges. First, fraud detection methods require complex investigations that involve the processing of large amounts of heterogeneous data. The data is derived from multiple sources and crosses multiple knowledge domains, including finance, economics, business, and law. Gathering and processing this data manually is prohibitively time-consuming as well as error-prone. Second, fraud is "a needle in a haystack" problem because only a very small fraction of the data is likely to be coming from a fraudulent case. The vast quantity of regular data—that is, data produced from nonfraudulent sources—tends to blend out the cases of fraud. Third, fraudsters are continually changing their methods, which means that detection strategies are frequently several steps behind.

Using hedge fund data as an example, this article demonstrates how MATLAB® can be used to automate the process of acquiring and analyzing fraud detection data. It shows how to import and aggregate heterogeneous data, construct and test models to identify indicators for potential fraud, and train machine learning techniques to the calculated indicators to classify a fund as fraudulent or nonfraudulent.

The statistical techniques and workflow described are applicable to any area requiring detailed analysis of large amounts of heterogeneous data from multiple sources, including data mining and operational research tasks in retail and logistic analysis, defense intelligence, and medical informatics.

## The Hedge Fund Case Study

The number of hedge funds has grown exponentially in recent years: The Eurekahedge database indicates a total of approximately 20,000 active funds worldwide.[1] Hedge funds are minimally regulated investment vehicles and, therefore, prime targets of fraud. For example, hedge fund managers may fake return data to create the illusion of high profits and attract more investors.

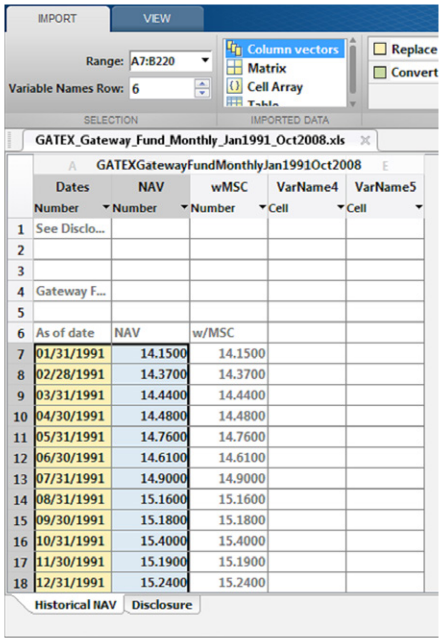
We will use monthly returns data from January 1991 to October 2008 from three hedge funds:

- Gateway Fund
- Growth Fund of America
- Fairfield Sentry Fund

The Fairfield Sentry Fund is a Madoff fund known to have reported fake data. As such, it offers a benchmark for verifying the efficacy of fraud detection mechanisms.

## Gathering Heterogeneous Data

Data for the Gateway Fund can be downloaded from the Natixis web site as a Microsoft® Excel® file containing the net asset value (NAV)



Range: A7:B220
Variable Names Row: 6

GATEX_Gateway_Fund_Monthly_Jan1991_Oct2008.xls

GATEXGatewayFundMonthlyJan1991Oct2008

| | Dates<br>Number | NAV<br>Number | wMSC<br>Number | VarName4<br>Cell | VarName5<br>Cell |
|---|---|---|---|---|---|
| 1 | See Disclo... | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | Gateway F... | | | | |
| 5 | | | | | |
| 6 | As of date | NAV | w/MSC | | |
| 7 | 01/31/1991 | 14.1500 | 14.1500 | | |
| 8 | 02/28/1991 | 14.3700 | 14.3700 | | |
| 9 | 03/31/1991 | 14.4400 | 14.4400 | | |
| 10 | 04/30/1991 | 14.4800 | 14.4800 | | |
| 11 | 05/31/1991 | 14.7600 | 14.7600 | | |
| 12 | 06/30/1991 | 14.6100 | 14.6100 | | |
| 13 | 07/31/1991 | 14.9000 | 14.9000 | | |
| 14 | 08/31/1991 | 15.1600 | 15.1600 | | |
| 15 | 09/30/1991 | 15.1800 | 15.1800 | | |
| 16 | 10/31/1991 | 15.4000 | 15.4000 | | |
| 17 | 11/30/1991 | 15.1900 | 15.1900 | | |
| 18 | 12/31/1991 | 15.2400 | 15.2400 | | |

FIGURE 1. *The MATLAB Data Import Tool for interactively importing data from files.*

of the fund on a monthly basis. Using the MATLAB Data Import Tool, we define how the data is to be imported (Figure 1). The Data Import Tool can automatically generate the MATLAB code to reproduce the defined import style.

After importing the NAV for the Gateway Fund, we calculate the monthly returns (code excerpt 1).

For the Growth Fund of America, we use Datafeed Toolbox™ to obtain data from Yahoo! Finance, specifying the ticker symbol for the fund (AGTHX), the name of the relevant field (adjusted close price), and the time period of interest (code excerpt 2).

Unfortunately, Yahoo does not provide data for the period from January 1991 to February 1993. For this time period, we have to collect the data manually.

Using the financial time series object in Financial Toolbox™, we convert the imported daily data to the desired monthly frequency (code excerpt 3).

Finally, we import reported data from the Fairfield Sentry fund. We use two freely available Java™ classes, PDFBox and FontBox, to read the text from the pdf version of the Fairfield Sentry fund fact sheet (code excerpt 4).

**Code Excerpt 1**

```
% Calculate monthly returns
gatewayReturns = tick2ret(gatewayNAV);
```

**Code Excerpt 2**

```
% Connect to yahoo and fetch data
c = yahoo;
data = fetch(c, 'AGTHX', 'Adj Close', startDate, endDate);
```

**Code Excerpt 3**

```
%Convert to monthly returns
tsobj = fints(dates, agthxClose);
tsobj = tomonthly(tsobj);
```

**Code Excerpt 4**

```
% Instantiate necessary classes
pdfdoc = org.apache.pdfbox.pdmodel.PDDocument;
reader = org.apache.pdfbox.util.PDFTextStripper;

% Read data
pdfdoc = pdfdoc.load(FilePath);
pdfstr = reader.getText(pdfdoc);
```

Having imported the text, we extract the parts containing the data of interest—that is, a table of monthly returns.

Some tests for fraudulent data require comparison of the funds' returns data to standard market data. We import the benchmark data for each fund using the techniques described above.

Once the data is imported and available, we can assess its consistency—for example, by comparing the normalized performance of all three funds (Figure 2).

Simply viewing the plot allows for a qualitative assessment. For example, the Madoff fund exhibits an unusually smooth growth, yielding a high profit. Furthermore, there are no obvious indications of inconsistency in the underlying data. This means that we will be able to use formal methods to detect fraudulent activities.

## Analyzing the Returns Data

Since misbehavior or fraud in hedge funds manifests itself mainly in misreported data, academic researchers have focused on devising methods to analyze and flag potentially manipulated fund returns. We compute metrics introduced by Bollen and Pool[2] and use them as potential indicators for fraud on the reported hedge fund returns. For example:

- Discontinuity at zero in the fund's returns distribution
- Low correlation with other assets, contradicting market trends
- Unconditional and conditional serial

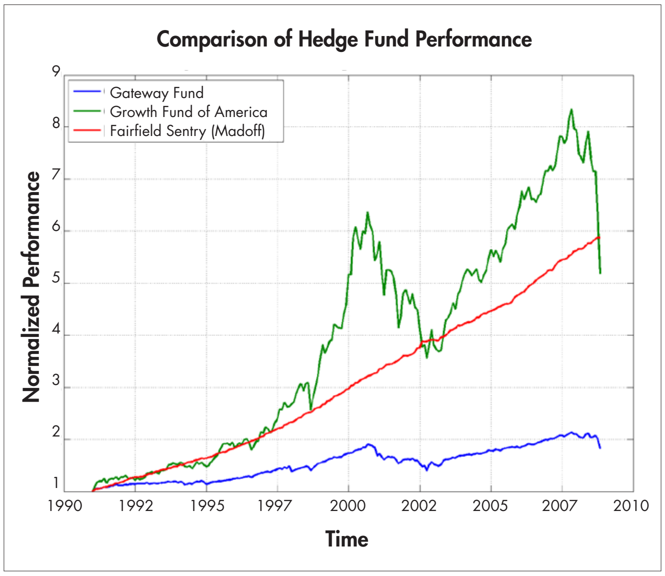


FIGURE 2. *Plot comparing the performance of the funds under consideration.*

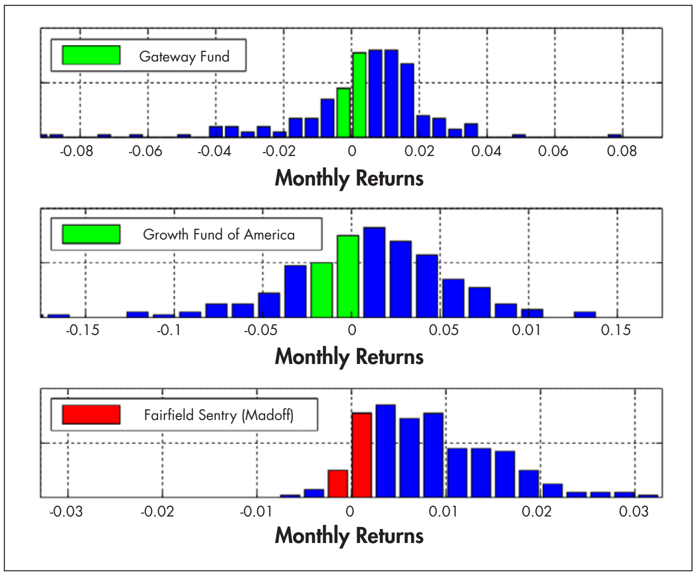


FIGURE 3. *Histograms of monthly returns for funds under consideration.*

correlation, indicating smoother than expected trends

- Number of returns equal to zero
- Number of negative, unique, and consecutive identical returns
- Distribution of the first digit (Does it follow Benford's law?) and the last digit (Is it uniform?) of reported returns

To illustrate the techniques, we will focus on discontinuity at zero.

## Testing for Discontinuity at Zero

Since funds with a higher number of positive returns attract more capital, fund managers have an incentive to misreport results to avoid negative returns. This means that a discontinuity at zero can be a potential indicator for fraud.

One test for such a discontinuity is counting the number of return observations that fall in three adjacent bins, two to the left of zero and one to the right. The number of observations in the middle bin should approximately equal the average of the surrounding two bins. A significant shortfall in the middle bin observations must be flagged.

Figure 3 shows the histograms of the funds' returns, with the two bins around zero highlighted. Green bars indicate no flag, and red bars indicate potential fraud. Only the Madoff fund did not pass this test.

## Results for Funds Under Consideration

Applying all the tests described above to the present data yields a table of indicators for each fund (Figure 4).

The Madoff fund raised a flag in nine out of ten tests, but the other two funds also raised flags. Positive test results do not prove that a given hedge fund was involved in fraudulent activities. However, a table like the one shown in Figure 4 indicates funds that merit further investigation.

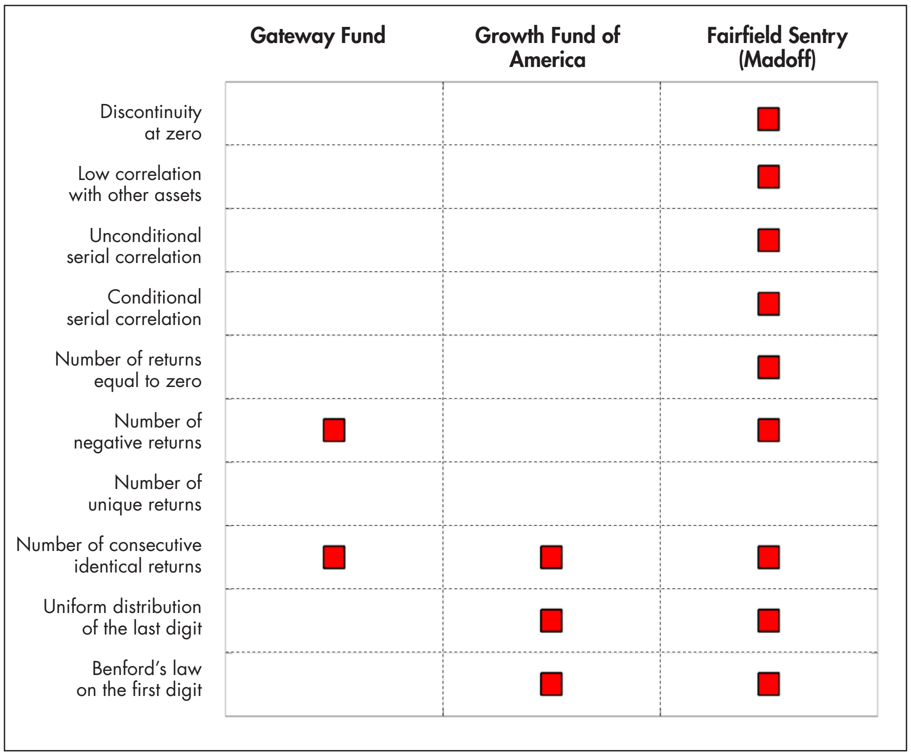
## Classifying Analysis Results with Machine Learning

We now have a set of flags that can be used as indicators for fraud. Automating the analytics enables us to review larger data sets and to use the computed flags to categorize funds as fraudulent or nonfraudulent. This classification problem can be addressed using machine learning methods—for example, bagged decision trees, using the TreeBagger algorithm in Statistics Toolbox™. The TreeBagger algorithm will require data for supervised learning to train the models. Note that our example uses data for only three funds. Applying bagged decision trees or other machine learning methods to an actual problem would require considerably more data than this small, illustrative set.

We want to build a model to classify funds as fraudulent or nonfraudulent, applying the indicators described in the section "Analyzing the Returns Data" as predictor variables. To create the model, we need a training set of data. Let us consider M hedge funds that are known as fraudulent or nonfraudulent. We store this information in the M-by-1-vector yTrain and compute the corresponding MxN-matrix xTrain of indicators. We can then create a bagged decision tree model (code excerpt 5), where nTrees is the number of decision trees created based on bootstrapped samples of the training data. The output of the nTrees decision trees is aggregated into a single classification.

Now, for a new fund, the classification can be performed (code excerpt 6).

We can use the fraud detection model to classify hedge funds based purely on their returns data. Since the model is automated,



| | Gateway Fund | Growth Fund of America | Fairfield Sentry (Madoff) |
|---|---|---|---|
| Discontinuity at zero | | | 🟥 |
| Low correlation with other assets | | | 🟥 |
| Unconditional serial correlation | | | 🟥 |
| Conditional serial correlation | | | 🟥 |
| Number of returns equal to zero | | | 🟥 |
| Number of negative returns | 🟥 | | 🟥 |
| Number of unique returns | | | |
| Number of consecutive identical returns | 🟥 | 🟥 | 🟥 |
| Uniform distribution of the last digit | | 🟥 | 🟥 |
| Benford's law on the first digit | | 🟥 | 🟥 |

FIGURE 4. *Test results for funds under consideration. Red boxes indicate results that raised a flag.*

**Code Excerpt 5**

```
% Create fraud detection model based on training data
fraudModel = TreeBagger(nTrees,xTrain,yTrain);
```

**Code Excerpt 6**

```
% Apply fraud detection model to new data
isFraud = predict(fraudModel, xNew);
```

it can be scaled to a large number of funds.

## The Bigger Picture

This article outlines the process of developing a fully automated algorithm for fraud detection based on hedge fund returns. The approach can be applied to a much larger data set using large-scale data processing solutions such as MATLAB Distributed Computing Server™ and Apache™ Hadoop®. Both technologies enable you to cope with data that exceeds the amount of memory available on a single machine.

The context in which the algorithm is deployed depends largely on the application use cases. Fund-of-funds managers working mostly with Excel might prefer to deploy the algorithm as an Excel add-in. They could use the module to investigate funds under consideration for future investments. Regulatory authorities could integrate a fraud detection scheme into their production systems, where it would periodically perform the analysis on new data, summarizing results in an automatically generated report.

We used advanced statistics to compute individual fraud indicators, and machine learning to create the classification model. In addition to the bagged decision trees discussed here, many other machine learning techniques are available in MATLAB, Statistics Toolbox, and Neural Network Toolbox™, enabling you to extend or alter the proposed solution according to the requirements of your project. ▪

[1] Eurekahedge. *www.eurekahedge.com*

[2] Bollen, Nicolas P. B., and Pool, Veronika K. "Suspicious Patterns in Hedge Fund Returns and the Risk of Fraud"(November 2011). *www2.owen.vanderbilt.edu/nick.bollen*

## Learn More

**Webinar: Machine Learning with MATLAB** 53:39
*mathworks.com/wbnr-81984*

**Example: Credit Rating by Bagged Decision Trees**
*mathworks.com/credit-rating*

# Variants of the QR Algorithm[1]

By Cleve Moler, MathWorks

The QR algorithm is one of the most successful and powerful tools we have in mathematical software.

The MATLAB® core library includes several variants of the QR algorithm. These variants compute the eigenvalues of real symmetric matrices, real nonsymmetric matrices, pairs of real matrices, complex matrices, pairs of complex matrices, and singular values of various types of matrices. These core library functions are used in various MATLAB toolboxes to find eigenvalues and singular values of sparse matrices and linear operators, find zeros of polynomials, solve special linear systems, assess numerical stability, and perform many other tasks.

Dozens of people have contributed to the development of the QR algorithm variants. The first papers on the subject came from J.G.F. Francis in 1961 and 1962 and Vera N. Kublanovskaya in 1963. But it was J.H. Wilkinson who developed the first complete implementation of the QR algorithm. Wilkinson also developed an important convergence analysis. Wilkinson's book *The Algebraic Eigenvalue Problem* and two of his papers were published in 1965. This means we'll be able to celebrate 2015 as the golden anniversary of the practical QR algorithm.

The variant of the QR algorithm used for the singular value decomposition (SVD) was published in 1965 by Gene Golub and Velvel Kahan and perfected in 1969 by Golub and Christian Reinsch.

## The Name "QR"

The name "QR" is derived from the letter Q, used to denote orthogonal matrices, and the letter R, used to denote right triangular matrices. There is a `qr` function in MATLAB, but it computes the QR factorization, not the QR algorithm. Any matrix, whether real or complex, square or rectangular, can be factored into the product of a matrix $Q$ with orthonormal columns and matrix $R$ that is nonzero only in its upper, or right, triangle. You might remember the Gram Schmidt process, which does pretty much the same thing, although in its original form it is numerically unstable.

## A One-Liner

Using the `qr` function, a simple variant of the QR algorithm can be expressed in one line of MATLAB code. Let `A` be a square, `n`-by-`n` matrix, and let `I = eye(n,n)`. Then one step of the QR iteration is given by

```
s = A(n,n); [Q,R] = qr(A - s*I); A = R*Q + s*I
```

The quantity `s` is the shift; it accelerates convergence. As `A` approaches an upper triangular matrix, `s` approaches an eigenvalue. If you enter these three statements on a single line, you can use the up-arrow key to iterate.

The QR factorization produces an upper triangular $R$.

$$A\text{-}sI = QR$$

Then the reverse order multiplication, $RQ$, restores the eigenvalues because

$$RQ + sI = Q'(A - sI)Q + sI = Q'AQ$$

So the new $A$ is similar to the original $A$. Each iteration effectively transfers some "mass" from the lower to the upper triangle while preserving the eigenvalues. As the iterations proceed, the matrix begins to approach an upper triangular matrix with the eigenvalues conveniently displayed on the diagonal.

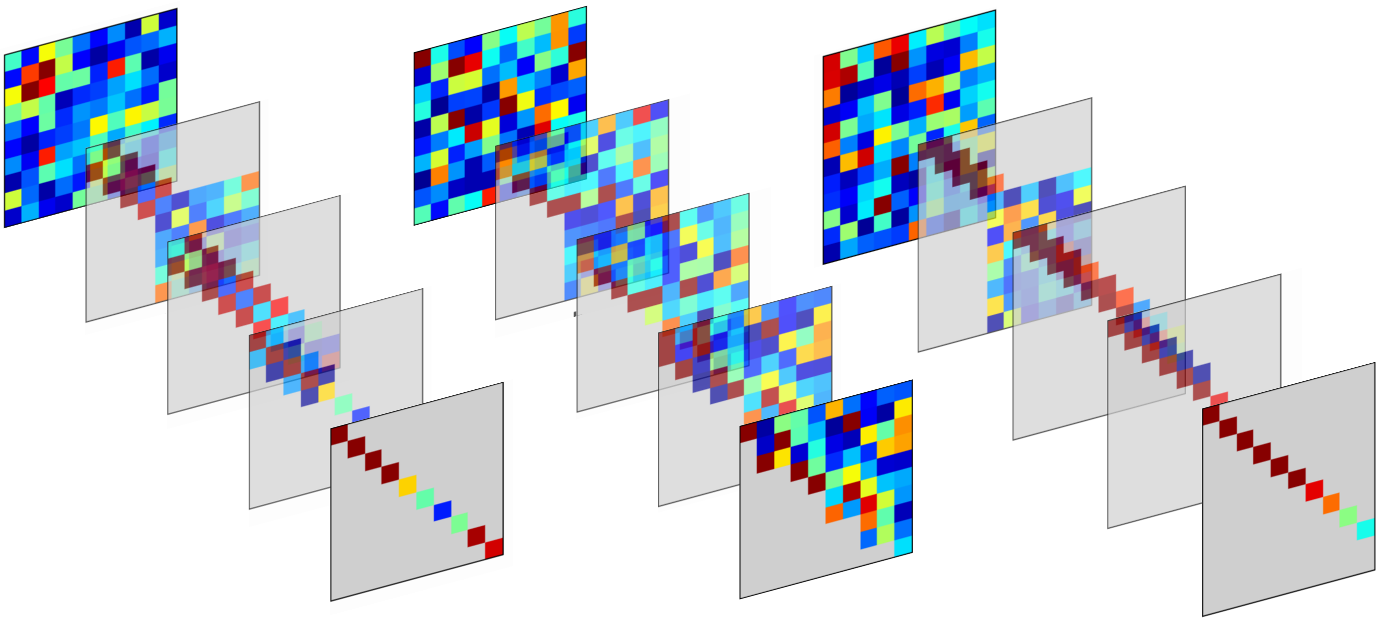
## An Example

To illustrate this process we'll use a matrix from the MATLAB Gallery collection.

```
A = gallery(3)
A =
    -149   -50   -154
     537   180    546
     -27    -9    -25
```

It is not at all obvious, but this matrix has been constructed to have eigenvalues 1, 2, and 3. The first iteration of our one-line QR code produces

```
A =
    28.8263   -259.8671   773.9292
     1.0353     -8.6686    33.1759
    -0.5973      5.5786   -14.1578
```



The matrix is now much nearer to being upper triangular, but the eigenvalues are still not evident. However, after five more iterations we have

```
A =
     3.0321   -8.0851  804.6651
     0.0017    0.9931  145.5046
    -0.0001    0.0005    1.9749
```

We begin to see the eigenvalues 3, 1, and 2 emerging on the diagonal. Eight more iterations give

```
A =
     3.0716   -7.6952  802.1201
     0.0193    0.9284  158.9556
          0         0    2.0000
```

The eigenvalue 2.0 has been computed to the displayed accuracy, and the below-diagonal element adjacent to it has become zero. At this point it is necessary to continue the iteration on the 2-by-2 upper left submatrix.

The QR algorithm is never carried out in this simple form. It is always preceded by a reduction to a compact form in which all the elements below the subdiagonal are zero. The iteration preserves this reduced form, and the factorizations can be done much more quickly. The shift strategy is more sophisticated, and is different for various forms of the algorithm. Additionally, the reduced form is of utmost importance for the convergence properties of the iteration.

## Symmetric Matrices

Figures 1–3 illustrate three of the most important variants of the QR algorithm. The figures are snapshots taken from the output generated by the program `eigsvdgui.m` from *Numerical Computing with MATLAB.*

The simplest variant involves real, symmetric matrices. An $n$-by-$n$ real, symmetric matrix can be reduced to tridiagonal form by means of $n$-2 Householder reflections, which are a sequence of similarity transformations preserving the eigenvalues. The QR iteration applies to the tridiagonal form. Wilkinson provided a shift strategy that allowed him to prove both global convergence and a local cubic convergence rate. Even in the presence of roundoff error, this algorithm is guaranteed to succeed.

Figure 1 shows an initial symmetric matrix, the situation halfway through the reduction to tridiagonal, the tridiagonal, the situation partway through the QR iteration, and finally, the eigenvalues. Actually, because the matrix is symmetric, the computation is only performed on one half of the array, but our figure reflects the results to show an entire matrix.

## Nonsymmetric Matrices

The situation for real, nonsymmetric matrices is much more complicated. The initial reduction uses $n$-2 Householder similarity transformations to create a Hessenberg matrix, which is upper triangular plus an “extra” subdiagonal. A QR iteration with a double shift strategy is then used. This preserves the Hessenberg form while attempting to create a real Schur form, which is upper triangular except for 2-by-2 blocks corresponding to pairs of complex conjugate

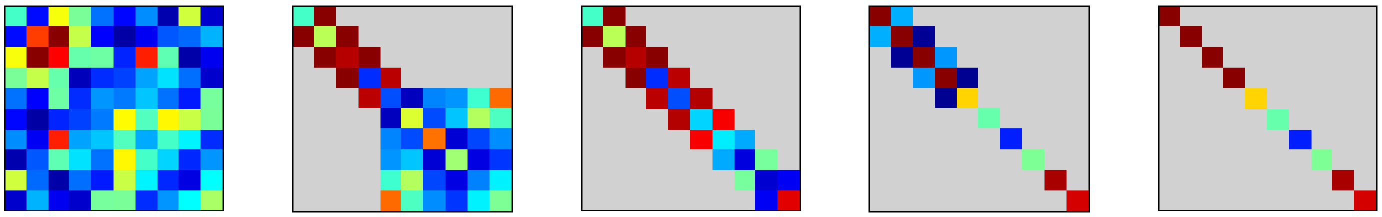


FIGURE 1. *Eigenvalues of a symmetric matrix. Left to right: The input—a random symmetric 10-by-10 matrix, halfway through the orthogonal reduction to tridiagonal form, symmetric tridiagonal form, partway through the symmetric tridiagonal QR iteration, and the final diagonal matrix of eigenvalues.*

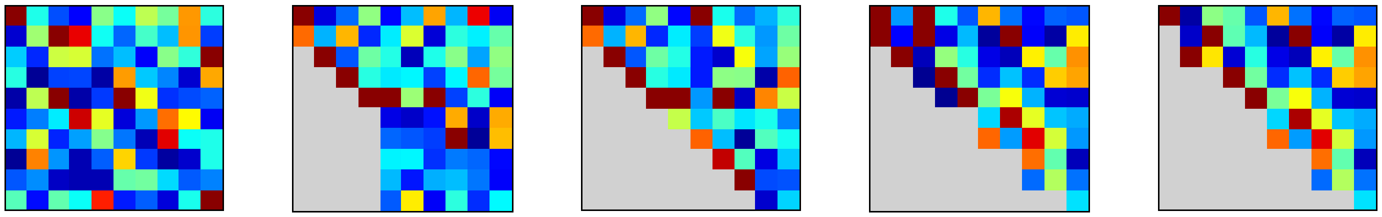


FIGURE 2. *Eigenvalues of a nonsymmetric matrix. Left to right: The input—a random 10-by-10 matrix; halfway through the orthogonal reduction to Hessenberg form; Hessenberg form, upper triangular plus one subdiagonal; partway through the nonsymmetric QR iteration; and the final real Schur form with real eigenvalues and 2-by-2 blocks with complex pairs of eigenvalues on the diagonal.*

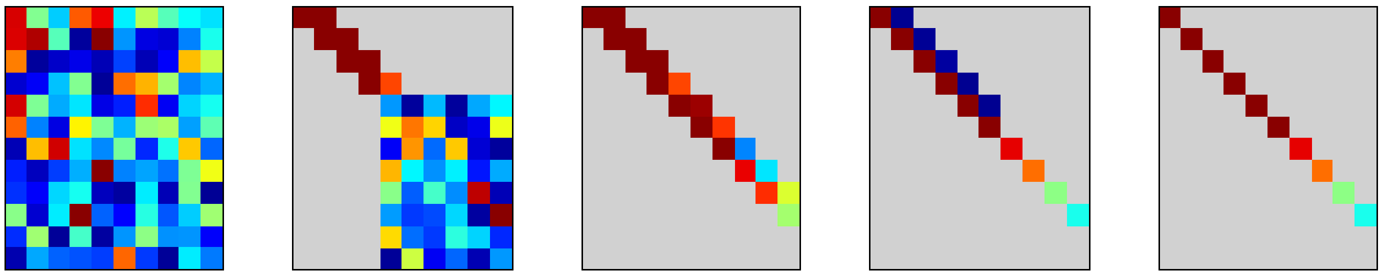


FIGURE 3. *Singular values of a rectangular matrix. Left to right: The input—a random rectangular 12-by-10 matrix, halfway through the orthogonal reduction to bidiagonal form, bidiagonal form, partway through the bidiagonal singular value QR iteration, and the final diagonal matrix of singular values.*

eigenvalues on the diagonal.

The nonsymmetric Hessenberg QR algorithm is not infallible. It is an iterative process that is not always guaranteed to converge. Even 30 years ago, counterexamples to the basic iteration were known. Wilkinson introduced an additional "ad hoc" shift to handle them, but no one has been able to prove a complete convergence theorem. So, on rare occasions, MATLAB users might see this message:

```
Error using ==> eig, Solution will not converge
```

Years ago, recipients of this message might have accepted it as unavoidable. But today, most people would be surprised or annoyed; they have come to expect infallibility.

We now know of a 4-by-4 example that may cause the real, nonsymmetric QR algorithm to fail on certain computers, even with Wilkinson's ad hoc shift. The matrix is

```
A =

     0   2   0  -1
     1   0   0   0
     0   1   0   0
     0   0   1   0
```

This is the companion matrix of the polynomial $p(x)=x^4-2x^2+1$, and the statement

```
roots([1 0 -2 0 1])
```

calls for the computation of `eig(A)`. The values $\lambda= 1$ and $\lambda= -1$ are both eigenvalues, or polynomial roots, with multiplicity two. For real $x$, the polynomial $p(x)$ is never negative. These double roots slow down the iteration so much that, on some computers, the vagaries of roundoff error interfere before convergence is detected. The iteration can wander forever, trying to converge but veering off when it gets close.

Similar behavior is shown by examples of the form

$$\begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & -\delta & 0 \\ 0 & \delta & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

where $\delta$ is small but not small enough to be neglected—say, $\delta = 10^{-8}$. The exact eigenvalues are close to a pair of double roots. The Wilkinson double shift iteration uses one eigenvalue from each pair. This iteration does change the matrix, but not enough to get rapid convergence. So we have to use a different double shift based on repeating one of the eigenvalues of the lower 2-by-2 blocks.

These different ad hoc shift strategies have been incorporated into the latest versions of LAPACK, and consequently, into MATLAB. We are now in the situation where we do not know of any matrices that cause `eig` or `roots` to display the "`will not converge`" message, but we have no proof that the nonsymmetric code with these various embellishments is infallible.

Figure 2 shows an initial nonsymmetric matrix, the situation halfway through the reduction to Hessenberg form, the Hessenberg form, the situation partway through the QR iteration, and the final real Schur form. For this particular matrix, it happens there are four real eigenvalues and three complex conjugate pairs, for a total of ten eigenvalues.

## Singular Values

The singular values of a possibly rectangular matrix $A$ are the square roots of the eigenvalues of the symmetric matrix $A'A$. This fact can be used to motivate and analyze an algorithm, but it should not be the basis for actual computation with finite precision arithmetic. The initial phase of the Golub-Kahan-Reinsch algorithm involves Householder reflections operating on both the left and the right to reduce a matrix to a bidiagonal form. This phase is followed by an SVD variant of the QR algorithm operating on the bidiagonal. Wilkinson's analysis of symmetric tridiagonal QR applies to this algorithm as well, so the process is guaranteed to be globally convergent.

Figure 3 shows an initial rectangular matrix, the situation halfway through the reduction to bidiagonal form, the bidiagonal form, the situation partway through the QR iteration, and the final diagonal form containing the singular values.

## QR Algorithm Applications

While the QR algorithms for computing eigenvalues and singular values are closely related, the applications of the results are usually very different. Eigenvalues are often employed to analyze systems of ordinary differential equations where behavior as a function of time is important. Singular values, on the other hand, are useful for analyzing static systems of simultaneous linear equations, where the number of equations is often not the same as the number of unknowns.

Control theory and control design automation make heavy use of eigenvalues. The classical state-space system of differential equations studied in control theory is

$$\dot{x}=Ax+Bu$$
$$y=Cx+Du$$

Using the QR algorithm to compute the eigenvalues of $A$ is essential to the investigation of issues like stability and controllability.

In statistics, the SVD is a numerically reliable way to obtain the principal components. Principal component analysis (PCA) is a technique for analyzing an overdetermined system of simultaneous linear equations

$$Ax=b$$

where $A$ has more rows than columns. Using the QR algorithm to compute the singular values and vectors of $A$ produces the principal components. ▪

## For Further Reading and Viewing

Golub, Gene H., and Charles F. Van Loan, *Matrix Computations, 4th Edition,* Johns Hopkins University Press, 1996, 697 pp. *mathworks.com/book-95399*.

Moler, Cleve, *Numerical Computing with MATLAB,* Chapter 10, "Eigenvalues and Singular Values", *mathworks.com/moler/eigs.pdf*.

Moler, Cleve, 1976 Matrix Singular Value Decomposition Film *youtube.com/watch?v=R9UoFyqJca8*.

Wilkinson, J.H., *The Algebraic Eigenvalue Problem,* Oxford University Press, 1965, 662 pp., *amazon.com/Algebraic-Eigenvalue-Monographs-Numerical-Analysis/dp/0198534183*.

[1] A portion of this article is based on "The QR Algorithm," *MATLAB News and Notes*, Summer 1995. *mathworks.com/qr-algorithm*.

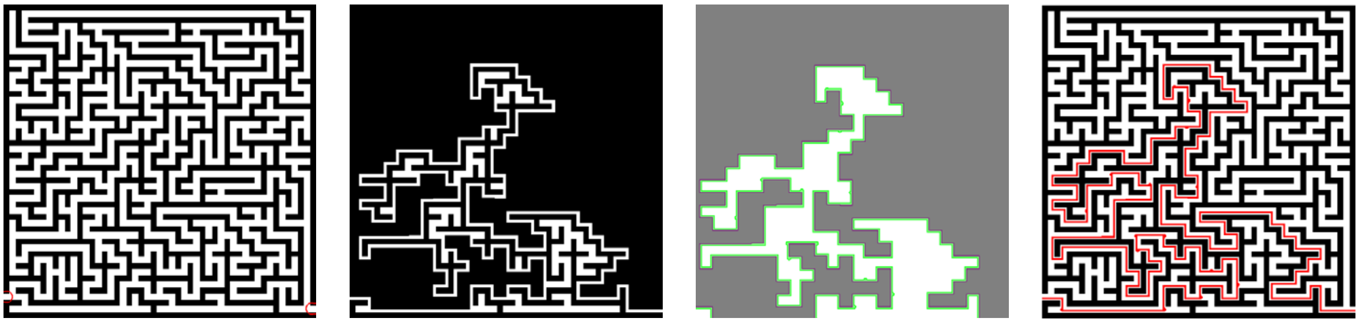
## Learn More

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

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

# Solving a Maze with the Watershed Transform

The watershed transform finds "catchment basins" and "watershed ridge lines" in an image by treating the image as a surface where light pixels are high and dark pixels are low. It is a powerful tool for solving segmentation problems in densely packed objects and other difficult images, such as an image of biological tissue or a container full of objects.



FIGURES 1–4. *Original maze, part of maze corresponding to a catchment basin, path (green) extracted by shrinking the catchment basin, and solution path overlaid on maze.*

Using a watershed transform to solve a maze—that is, to find a path between the entry and exit points[1]—is a straightforward process, but it involves several steps. All the steps must be completed to get the best results.

To illustrate this process, we will use the `watershed` function in Image Processing Toolbox™ to "solve" the maze (Figure 1).

The landscape (surface) of the maze image has two catchment basins. The solution path is the watershed between those catchment basins. Let's compute the catchment basins produced by `watershed`.

```
L = watershed(I);
```

We can visualize the catchment basins using `imshow` (figure not shown here). The interface (boundary) between the two basins is the solution (path) to the maze.

At this point we have already solved the bulk of the problem. All we need to do now is extract the interface between the two basins.

First we create a new image, retaining only the original image region from one of the catchment basins (Figure 2).

```
L1 = L == 2;
I1 = L1.*I;
imshow(I1)
```

Next we use `watershed` again to get catchment basins for this new image.

```
L2 = watershed(I1);
```

Using `imshowpair` we compare the new image to the original. We see in Figure 3 that the catchment basin in the new image has shrunk by roughly half the width of the maze paths (shown in green) compared with the same catchment basin in the original image.

```
imshowpair(L,L2)
```

Finally, we extract the path shown in green, which is the region where the two catchment basins differ.

```
img1 = L == 2;
img2 = L2 == 2;
path = img1 - img2;
```

Using the `imoverlay`[2] function we can visualize the path on the maze (Figure 4).

```
P = imoverlay(I, path, [1 0 0]);
imshow(P)
```

The watershed transform is also useful for image segmentation. It can be applied to many images where traditional techniques fall short—for example, images where objects are densely packed, such as cells in a tissue sample. Any image where you can visually identify high or low intensity areas between objects will be a good candidate for watershed segmentation. ■

[1]We are considering only a "standard" or "perfect" maze that has one path from entry to exit.

[2]Available on the File Exchange: *mathworks.com/fx-10502*

## Learn More

**Steve on Image Processing**
*blogs.mathworks.com/steve*

**Example: Marker-Controlled Watershed Segmentation**
*mathworks.com/marker*

**The Watershed Transform: Strategies for Image Segmentation**
*mathworks.com/watershed-transform*

# Hands-On Learning with MATLAB

MATLAB® supports a variety of hardware platforms for classroom laboratory use. With MATLAB and these platforms, your students can bring theory to life with hands-on projects in controls, mechatronics, robotics, signal processing, and circuit design.

## Analog Discovery

Digilent Analog Discovery Design Kit is a hardware development platform that enables students to design their own analog circuits. Teaching materials, reference designs, and lab projects are available for download.

The Data Acquisition Toolbox™ Support Package for Digilent Analog Discovery hardware lets you perform the following tasks in MATLAB:

- Read data from the two oscilloscope channels (analog input)
- Control and generate data from the two waveform generators (analog output)
- Characterize ICs and measure behavior of the circuit and IC components
- Find and display Digilent Analog Discovery device settings

**Digilent Analog Discovery Support from MATLAB**
*mathworks.com/digilent-analog-discovery*

**Hands-On Learning with MATLAB and Analog Discovery** 35:00
*mathworks.com/matlab-analog-discovery*

**EDN Network: Student Engineering Labs Revived with Hands-on Electronic Kits**
*edn.com/electronics-blogs/anablog/4427939/Student-engineering-labs-revived-with-hands-on-electronic-kits*

## Arduino

Arduino® is a microcontroller board for exploring concepts in electrical engineering, motor control, and mechatronics.

MATLAB support for Arduino enables you to use MATLAB to communicate with the Arduino board over a USB cable. With this package your students can:

- Interactively develop programs to acquire analog and digital data
- Control DC, servo, and stepper motors
- Run control loops at up to 25 Hz (not real time)

**Arduino Support from MATLAB**
*mathworks.com/arduino-support*

**Learning Basic Mechatronics Concepts Using the Arduino Board and MATLAB**
*mathworks.com/mechatronics-concepts*

**Arduino IO Package: Slides and Examples**
*mathworks.com/fx-27843*

## Raspberry Pi

Raspberry Pi™ is a tiny, low-cost, single-board computer specifically designed for teaching. It features a Broadcom® system-on-a-chip that includes an ARM11 processor running at 700 MHz, 256 or 512 MB RAM, and a VideoCore IV GPU.

MATLAB support for Raspberry Pi includes:

- A library of MATLAB functions that let you acquire data from sensors and imaging devices, including the Raspberry Pi Camera Board, I2C, SPI and serial interfaces, and GPIO pins
- Access to prebuilt audio and video algorithms in DSP System Toolbox™ and Computer Vision System Toolbox™

**Raspberry Pi Support from MATLAB**
*mathworks.com/raspberry-pi-matlab*

**Enabling Project-Based Learning with MATLAB, Simulink, and Target Hardware** 45:19
*mathworks.com/wbnr-81847*

**Working with the Raspberry Pi Camera Board**
*mathworks.com/raspberry-pi-camera-board*

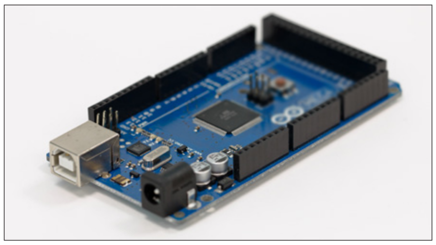
## Learn More

**Hardware for Project-Based Learning**
*mathworks.com/project-hardware*

**MakerZone**
*makerzone.mathworks.com*

# Connecting Low-Cost Hardware Platforms to MATLAB and Simulink

Low-cost hardware platforms from Arduino® to Zynq® have quickly developed vibrant user communities, and are now used in a range of applications, including sensing and control, computer vision, and robotics. With these platforms and their user communities, makers explore new ideas, students engage in hands-on projects, and developers deliver commercial products. Hardware support packages enable users to stream sensor data into MATLAB® and automatically generate code to run their Simulink® models on real hardware.



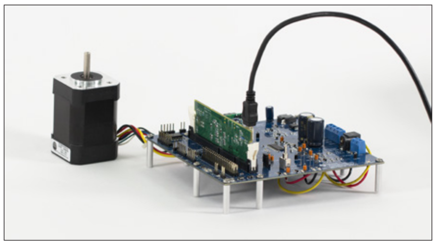
## Arduino, LEGO, and Raspberry Pi Development Platforms

Widely adopted by the maker and student communities, Arduino offers an open hardware architecture with a range of boards and accessories. The LEGO® MINDSTORMS® platform offers easy-to-assemble robots, and serves as a powerful teaching tool in many schools. Both platforms are frequently used in robotics and controls applications. Launched in 2012, Raspberry Pi™ is a pocket-sized, single-board computer designed to make computer science accessible for high school students. Kits and projects from the user community have helped broaden the range of applications to computer vision and communications. Support packages enable users to program any of these platforms with Simulink for standalone execution or configure them to communicate with a MATLAB session.

*arduino.cc*

*mindstorms.lego.com*

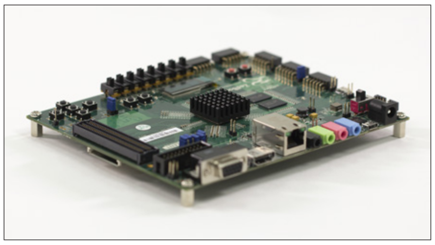
*raspberrypi.org*



## Texas Instruments: C2000 LaunchPad

The Texas Instruments C2000™ MCU family offers hardware for a variety of applications, from low-cost USB form factor controlSTICKs to full-featured developer platforms for solar, motor control, lighting, and digital power applications. Low-cost LaunchPad™ Evaluation Kits and BoosterPack™ plug-in modules make it easy to explore different applications enabled by the C2000 microcontroller. With Embedded Coder®, engineers can generate optimized C and C++ code directly from their algorithms in Simulink and compile it with TI's Code Composer Studio™ (CCS) IDE. Target I/O blocks for C2000 peripherals are provided with the C2000 hardware support package.

*ti.com/c2000*



## Xilinx: Zynq and ZedBoard

The Xilinx® Zynq-7000 All-Programmable SoC combines a dual-core ARM® Cortex®-A9 with Xilinx 7-series FPGA logic on a single chip. ZedBoard™ provides a low-cost, rapid-prototyping and development platform for a wide variety of Zynq applications, including motor control, software-defined radio, and video and image processing. MathWorks Zynq support enables you to use MATLAB, Simulink, HDL Coder™, and Embedded Coder to target both the ARM cores and the FPGA logic on Zynq.

*xilinx.com/zynq*

### Learn More

**Third-Party Products and Services**
*mathworks.com/connections*

**MakerZone**
*makerzone.mathworks.com*

**Hardware Support**
*mathworks.com/hardware*

# Explore MATLAB and Simulink Webinars

**Join a live session or browse the library of commercial and academic recordings.**

***mathworks.com/recordedwebinars***

Technical experts present practical tips and examples. Topics include:

- Code generation and verification
- Real-time simulation and testing
- Signal processing and communications
- Physical modeling
- Robotics
- Verification, validation, and test
- Image processing and computer vision
- Test and measurement
- Math, statistics, and optimization
- Machine learning
- GPU and parallel computing
- Control system design and analysis

*POWERED WITH ELECTRICITY, GAS,*
*AND AUTOMATICALLY-GENERATED CODE.*

*THAT'S MODEL-BASED DESIGN.*

***To create a two-mode hybrid powertrain, engineers at GM used models to continuously verify their design, test prototypes, and automatically generate the embedded code.***
***The result: a breakthrough HEV, delivered on time.***
***To learn more, visit mathworks.com/mbd***

MATLAB®
&SIMULINK®

MathWorks®
*Accelerating the pace of engineering and science*