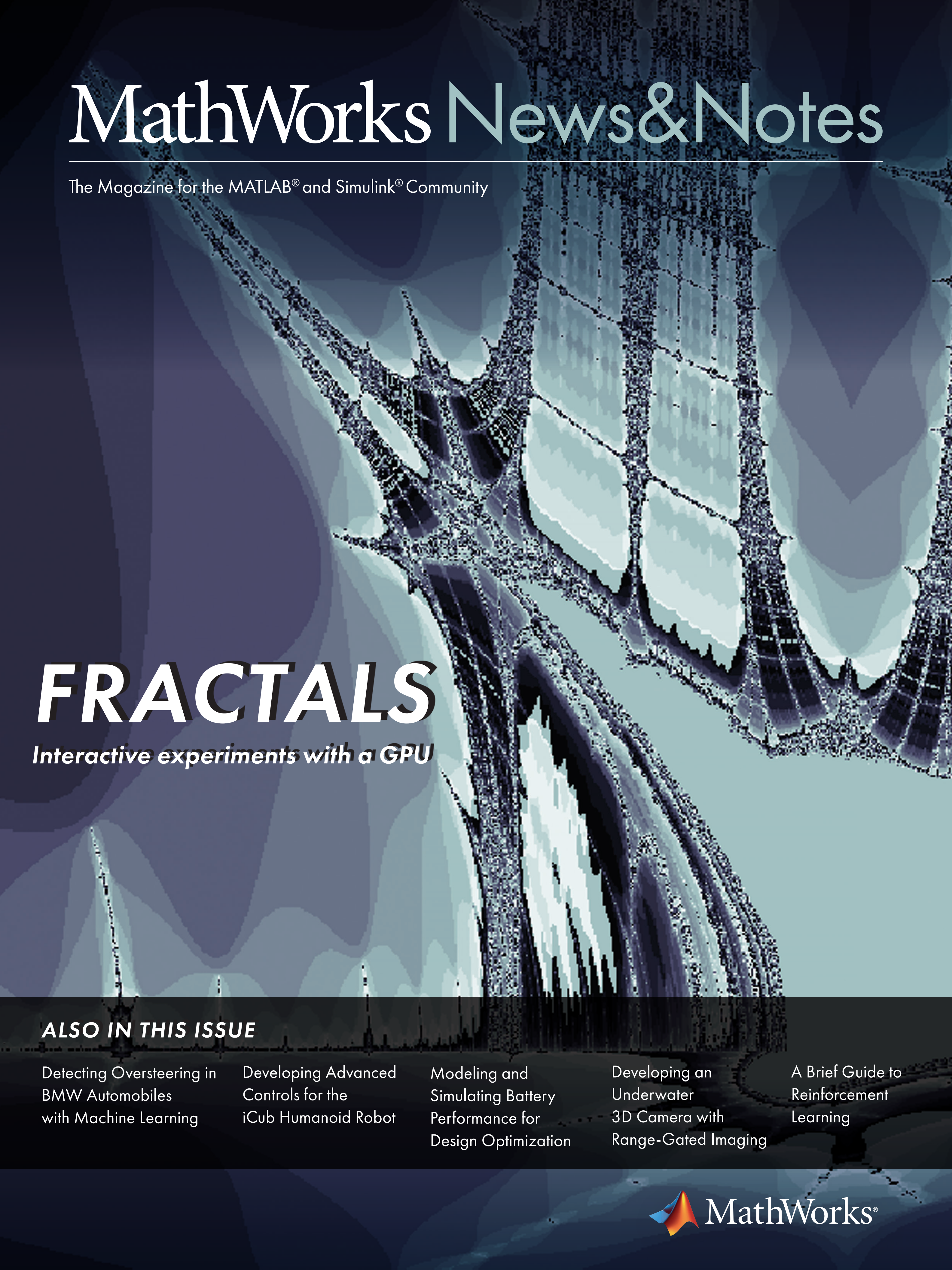
# MathWorks News&Notes

The Magazine for the MATLAB® and Simulink® Community



## FRACTALS

**Interactive experiments with a GPU**

MathWorks®

AI algorithms in this robotic arm let composer and drummer Jason Barnes, an amputee, drum at speeds that no human ever has before.

*"I'll bet a lot of metal drummers might be jealous of what I can do now. Speed is good. Faster is always better."*

**mathworks.com/drummer**

## FEATURES

## QUICK READS

*Frank Hurley's 1915 photograph of Ernest Shackleton's ship frozen in ice in Antarctica provides a jumping-off point for Cleve's fractal exploration (page 26).*

MANAGING EDITOR
Linda Webb

EDITOR
Rosemary Oxenford

ART DIRECTOR
Kevin Hart

GRAPHIC DESIGNER
Gabrielle Lydon

PRODUCTION EDITOR
Julie Cornell

TECHNICAL WRITER
Jack Wilber

DIGITAL PRODUCTION SPECIALIST
Nicole Mosher

PRINTER
DS Graphics

PRINT LIAISON
Jill Mespelli

EDITORIAL BOARD
Thomas Andraczek, P.J. Boardman, Michael Carone, Stacey Gage, Michelle Hirsch, Andy May, Cleve Moler, Sameer Prabhu, Richard Rovner, Loren Shure, John Stewart, Jim Tung

CONTRIBUTORS AND REVIEWERS
A. Balu, A. Bennett, S. Bhadravathi Patil, G. Bourdon, A. Caraceto, B. Chou, K. Cohan, S. DeLand, D. Ferigo, T. Freudling, J. Gazzarri, M. Glasser, T. Grassinger, C. Grytberg, L. Guthman, L. Harvey, K.H. Haugholt, A. Hoffman, R. Hyde, P. Kapur, L. Kempler, T. Lennon, J. Levelt, K. Lorenc, E. Martinho, P. Massano, A. Nehemiah, C. Patel, R. Pillat, P. Pilotte, J. Pingel, A. Poon, D. Pucci, O. Pujado, E. Putnam, P. Risholm, P. Sandberg, G. Sandmann, C. Santacruz-Rosero, A. Stothert, J. Thielemann, B. Tordoff, S. Traversaro, E. Tzorakoleftherakis, P. Vallauri, A. van Nieuwkerk, P. Wallner, C. Wang

SUBSCRIBE
mathworks.com/subscribe

CONTACT US
mathworks.com/contact

FOLLOW US ON SOCIAL MEDIA

READ ONLINE

MIX From responsible sources FSC® C084277

Printed on 30% post-consumer waste materials

MathWorks®

# OVERSTEERING

## Detecting Oversteering in BMW Automobiles with Machine Learning

By Tobias Freudling, BMW Group

*Today, a single vehicle can generate a terabyte of measured data in a day. Machine learning provides an opportunity to develop software that uses the available data to learn about a driver's behavior and improve the driving experience.*

Oversteering is an unsafe condition in which a vehicle's rear tires lose their grip while navigating a turn (Figure 1). It can be caused by worn tires, slippery road conditions, taking a turn too fast, braking abruptly while turning, or a combination of these factors.

Modern stability control systems are designed to automatically take corrective action when oversteer is detected. In theory, such systems can identify an oversteering condition by using mathematical models based on first principles. For example, when measurements from onboard sensors exceed established threshold values for parameters in the model, the system determines that the car is oversteering. In practice, however, this approach has proved difficult to implement because of the interplay of the many factors involved. A car with underinflated tires on an icy road might need vastly different threshold values than the same car operating with properly inflated tires on a dry surface.

At BMW, we are exploring a machine learning approach to detecting oversteering. Working in MATLAB®, we developed a supervised machine learning model as a proof of concept. Despite having little previous experience with machine learning, in just three weeks we completed a working ECU prototype capable of detecting oversteering with over 98% accuracy.

## Collecting Data and Extracting Features

We began by gathering real-world data from a vehicle before, during, and after oversteering. With the help of a professional driver, we conducted live driving tests in a BMW M4 at the BMW proving grounds in Miramas, France (Figure 2).

During the tests, we captured signals commonly used in oversteer detection algorithms: the vehicle's forward acceleration, lateral acceleration, steering angle, and yaw rate. In addition, we logged the driver's perception of oversteering: When the driver indicated the car was oversteering, my colleague, riding in the car as a passenger, pressed a button on her laptop. She released the button when the driver indicated the car had returned to handling normally. These button presses created the ground-truth labels we need to train a supervised learning model. Altogether, we captured about 259,000 data points in 43 minutes of recorded data.

Back in our Munich office, we loaded the data that we had collected into MATLAB and used the Classification Learner app in Statistics and Machine Learning Toolbox™ to train machine learning models using a variety of classifiers. The results produced by models trained on this raw data were not outstanding—the accuracy was between 75% and 80%. To achieve more accurate results, we cleaned and reduced the raw data. First, we applied filters to reduce noise on the signal data (Figure 3).

Next, we used peak analysis to identify the peaks (local maxima) on our filtered input signals (Figure 4).

**FIGURE 1.** *Oversteering a BMW M4 on a test track.*

## Evaluating Machine Learning Approaches

After filtering and reducing the collected data, we were in a better position to evaluate supervised machine learning approaches. Using the Classification Learner app, we tried k-nearest neighbor (KNN) classifiers, support vector machines (SVMs), quadratic discriminant analysis, and decision trees. We also used the app to see the effect of transforming features through principal component analysis (PCA), which helps prevent overfitting.

The results produced by the classifiers that we evaluated are summarized in Table 1. All the classifiers performed well in identifying oversteer, with three producing true positive rates above 98%. The deciding factor was the true negative rates: how accurately the classifier was able to determine when the vehicle was not oversteering. Here, decision trees outperformed the other classifiers, with a true negative rate of almost 96%.

**FIGURE 2.** *The BMW proving grounds in Miramas, France.*

## Generating Code for In-Vehicle Tests

The results produced by the decision tree were promising, but the true test would be how well the classifier performed on an ECU in a real car. We generated code from the model with MATLAB Coder™ and compiled the code for our target ECU, installed in a BMW 5 Series sedan. This time, we conducted the tests ourselves at a BMW facility near Aschheim, close to our office. As I drove, my colleague collected data, recording the precise times when I indicated that the car was oversteering.

Running in real time on the ECU, the classifier performed surprisingly well, with an accuracy rate of about 95%. Going into the tests, we had not known what to expect because we were using a different vehicle (a BMW 5 Series instead of an M4), a different driver, and a different track. A closer look at the data revealed that most of the mismatches between the model and the driver's perceived oversteering occurred near the beginning and end of the oversteering condition. This mismatch is understandable; it can be difficult even for a driver to determine exactly when oversteer has started and stopped.

Having successfully developed a machine learning model for oversteering detection and deployed it on a prototype ECU, we are now envisioning numerous other potential applications for machine learning at BMW. Vast amounts of data collected over decades are available to us, and today, a single vehicle can generate a terabyte of measured data in a day. Machine learning provides an opportunity to develop software that uses the available data to learn about a driver's behavior and improve the driving experience. ◆

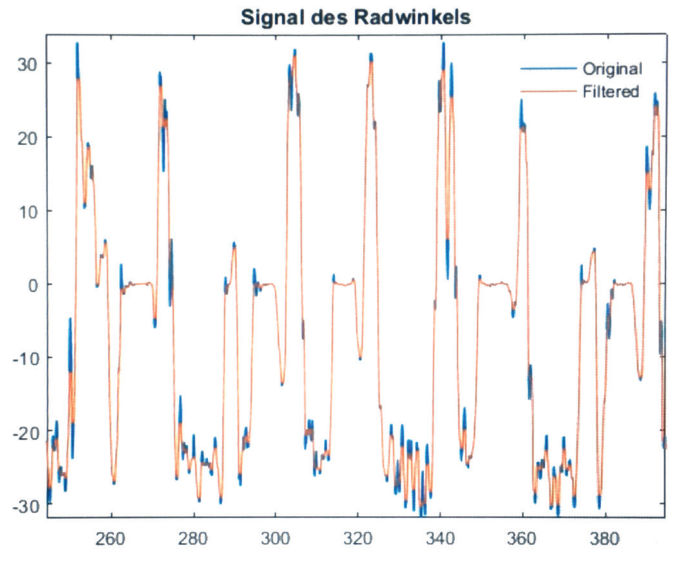


**FIGURE 3.** *The original steering angle signal (blue) and the same signal after filtering (orange).*

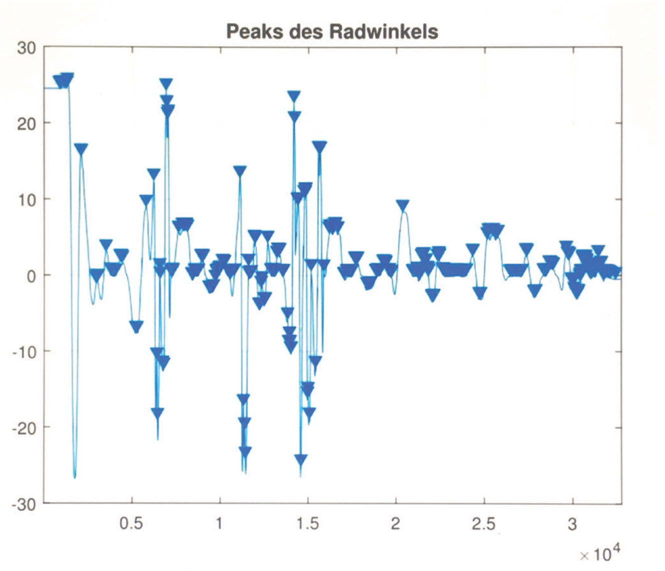
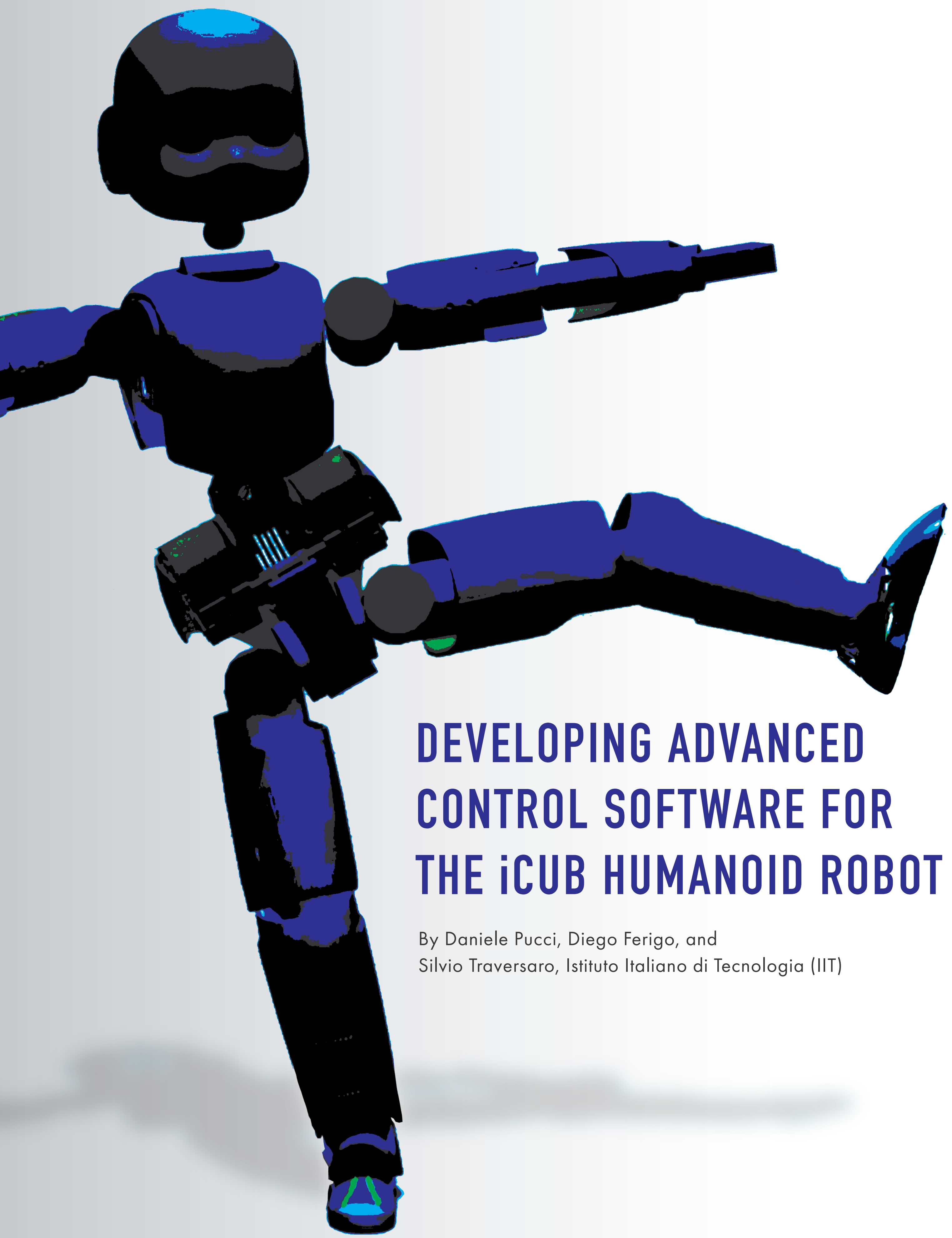


**FIGURE 4.** *The steering angle signal with peaks identified.*

| | True Positive (%) | True Negative (%) | False Positive (%) | False Negative (%) |
|---|---|---|---|---|
| K-Nearest Neighbor with PCA | 94.74 | 90.35 | 5.26 | 9.65 |
| Support Vector Machine | 98.92 | 73.07 | 1.08 | 26.93 |
| Quadratic Discriminant Analysis | 98.83 | 82.73 | 1.17 | 17.27 |
| Decision Trees | 98.16 | 95.86 | 1.84 | 4.14 |

**TABLE 1.** *Summary of results for four different supervised machine learning classifiers.*

*All the classifiers performed well in identifying oversteering, with three producing true positive rates above 98%. The deciding factor was the true negative rates ... here, decision trees outperformed the other classifiers, with a rate of almost 96%.*



# DEVELOPING ADVANCED CONTROL SOFTWARE FOR THE iCUB HUMANOID ROBOT

By Daniele Pucci, Diego Ferigo, and
Silvio Traversaro, Istituto Italiano di Tecnologia (IIT)

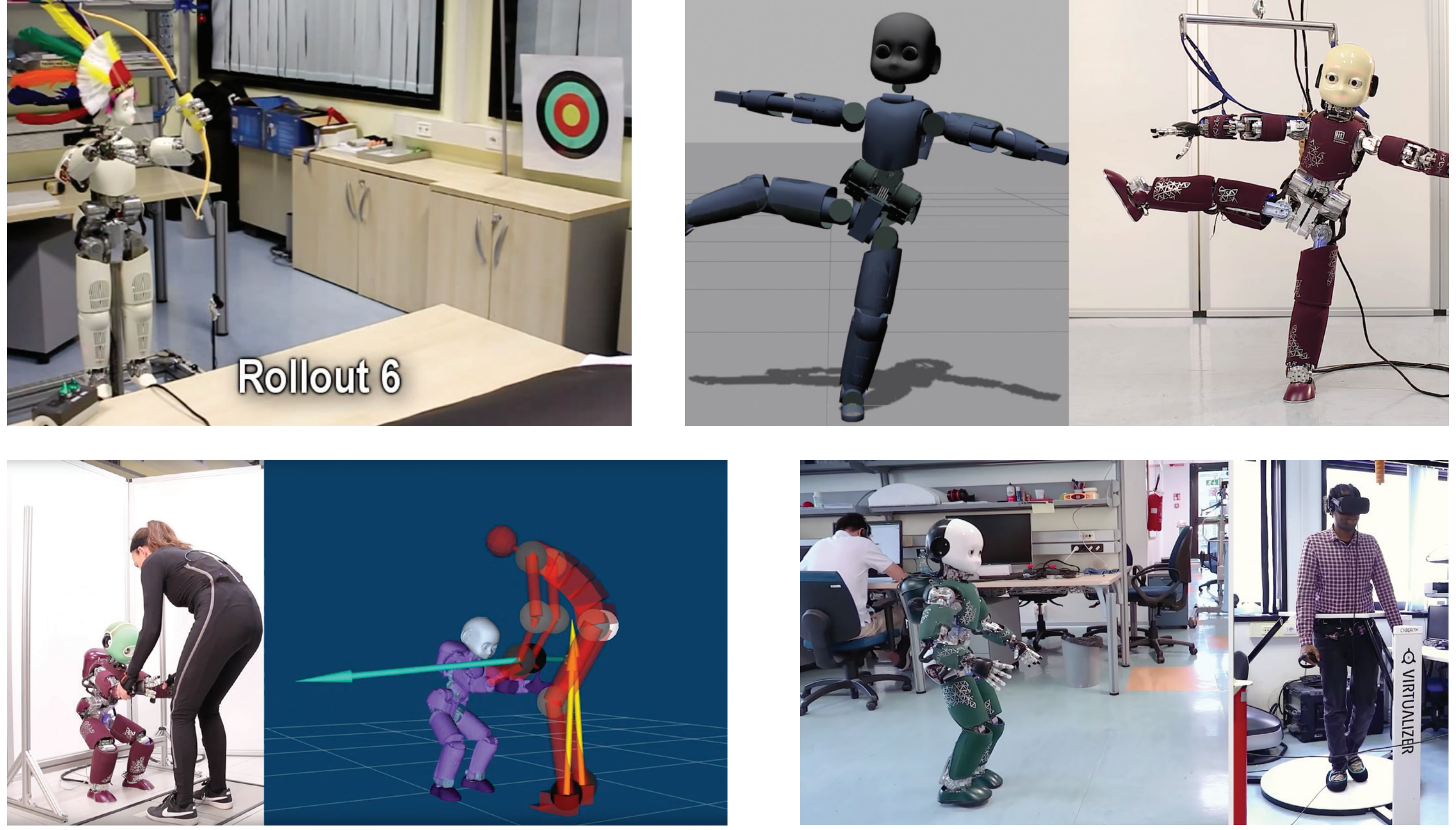


FIGURE 1. *Clockwise from top left: iCub shooting arrows, striking a yoga pose, walking while controlled via teleoperation, and working with a human to stand up.*

The iCub project was launched in 2004 as part of the RobotCub European Project, whose main aim was to study embodied cognition—the theory that an organism develops cognitive skills as it interacts with its environment. The main outcome of the iCub project is a 1-meter-tall, 53-degrees-of-freedom humanoid currently being developed at the Italian Institute of Technology (IIT). Over the years, the iCub robot has been used as a research platform for diverse fields of applied robotics, including balancing, teleoperated walking, and human-robot collaboration (Figure 1).

iCub is equipped with more than 50 motors, as well as force-torque sensors, inertial measurement units, and dozens of encoders and accelerometers. Developing control algorithms for a robot this complex is a difficult challenge. Our team at IIT—the Dynamic Interaction Control team—has created a development workflow based on Simulink® and Simulink Coder™ that makes it possible for even inexperienced team members to rapidly implement new control features, validate them through simulation, and run them on an iCub robot without writing any low-level code.

## PROTOTYPING iCUB CONTROLLERS

We prototype our control software using Simulink and the open-source Whole Body Toolbox, developed at IIT. Whole Body Toolbox is based on the BlockFactory dataflow framework, which we created to provide C++ interfaces for dataflow programming. We begin by modeling the controller in Simulink, incorporating sensors, actuators, and commonly used robotic algorithms. The Whole Body Toolbox then creates interfaces to either the real or the simulated iCub (Figure 2).

We use the Whole Body Toolbox most frequently for applications involving dynamic balancing. The robot controller regulates the contact forces between the robot and the environment, enabling the robot to maintain its balance even if pushed or otherwise perturbed by a human. As a secondary task, the robot tries to maintain a posture selected by the human: the robot automatically filters out any posture that alters its balance and stability.

We cosimulate the control model in Simulink with a model of iCub in the Gazebo robotics simulator (Figure 3). Cosimulation enables us to fix defects before testing on the actual robot, minimizing the risk of damaging the robot or endangering a human. For example, with our cosimulation setup we can determine whether a given set of gains creates unstable behavior in the robot in the event of unplanned fast movements.

iCub streams measurements coming from sensors at 100 Hz. These measurements are published over the network by our robotic middleware YARP.

Once we have validated the controller via simulations, we test it on a real iCub in the lab. In our test setup, Simulink runs on a PC with a standard x86 processor and communicates with the iCub via YARP middleware and TCP/UDP. The controller running in Simulink sends torque commands to the robot, which is able to follow a trajectory while maintaining its balance. A real-time synchronizer block, developed for the Whole Body Toolbox, synchronizes the robot moving in the real world with the control model.

## iCUB RESEARCH AT IIT

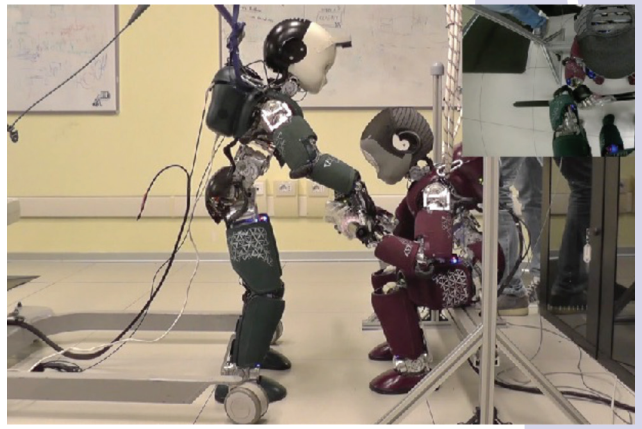
The IIT iCub project includes several lines of research. The Dynamic Interaction Control team focuses on three areas: telexistence, agent-robot collaboration (with the agent being either a human or another humanoid robot), and aerial humanoid robotics.

Telexistence enables a human to exist virtually in another location via a robotic avatar. In our experiments, the iCub walks and manipulates objects in the real world while the human walks and manipulates objects in a virtual environment. When the human takes a step, we process the human motion and a reference signal is sent to the iCub, causing it to take a step. Similarly, when the human closes a hand, a signal is sent to cause the iCub's hand to close.

Our research into agent-robot collaboration centers on ways that humans and robots can work together. In one of our experiments, a human helps the iCub to stand from a sitting position. The human wears a specially designed suit that provide real-time kinematics and dynamics data, enabling us to track and model the human's movements and muscular stresses while tracking the robot's movements in Simulink. We recently extended the human-robot collaboration experiment with a robot-robot collaboration version in which one iCub helps a second iCub to stand.

Aerial humanoid robotics is one of our most active research areas. We are working on a version of iCub that will be equipped with jetpacks. It will be able to fly to a specified location, land, and begin walking and interacting with the environment. A robot with these capabilities would be useful in high-risk disaster and search-and-rescue scenarios, such as earthquakes, floods, or wildfires. In these situations, the value of having robots able to fly from one building to another looking for survivors, open doors, close gas valves, and enter buildings is incalculable.

*Redesign of the iCub humanoid robot with jetpack.*


*Robot-robot collaboration.*

*A simulated iCub equipped with jetpacks during inspection maneuvers in a disaster-like scenario.*

## DEPLOYING THE CONTROLLER

Our over-the-network configuration is convenient for rapid design iterations, but it leaves iCub reliant on the TCP/UDP communication link. To break this reliance and enable iCub to operate more independently, we deploy our controller to an x86 processor inside the robot's head, eliminating the need for network communication, with the associated latency and risk of communication errors.

We generate C++ code from our control models with Simulink Coder, compile it, and validate it using the same over-the-network configuration that we used earlier when running the controller in Simulink. Then, we run the code on the x86 processor mounted inside iCub's head, enabling iCub to operate as a single unit rather than on a separate control PC.

## PORTABLE CONTROL DESIGN

Today, about 40 iCub robots are in use by research groups around the world. Although the hardware designs vary, every version can use the same controller design. This portability is made possible by a configuration block within our Simulink control model that loads a Unified Robot Description Format (URDF) file describing the kinematics and dynamics of each robot. Using this configuration file, we can run the same controller on IIT's 120-kilogram, 1.85-meter-tall simulated WALK-MAN as on a 33-kilogram, 1-meter-tall iCub.

## LOOKING AHEAD

We are confident that our research will result in many real-world applications. For example, in the future, telexistence could be used to help the physically disabled perform tasks requiring strength or dexterity. In these cases, a robotic avatar would move and act in the physical world, remotely controlled by the disabled human.

Our research on agent-robot collaboration is fundamental to creating robots to help humans at home and in the workplace—for example, assisting the elderly with activities of daily living and helping factory workers perform tasks involving muscular skeletal stress. Finally, the new branch of robotics that we are pioneering, aerial humanoid robotics, has a multitude of applications. Jet-powered, heavy-payload aerial platforms can be derived from flying humanoid robots for drug and food delivery during disaster response, heavy-payload last-mile delivery, and rescue platforms for firefighters, as well as platforms for humans performing high-voltage pylon inspection.

To translate the results of our research into real applications, we work closely with the iCub facility that develops, maintains, and continuously updates the iCub humanoid robot. ◆

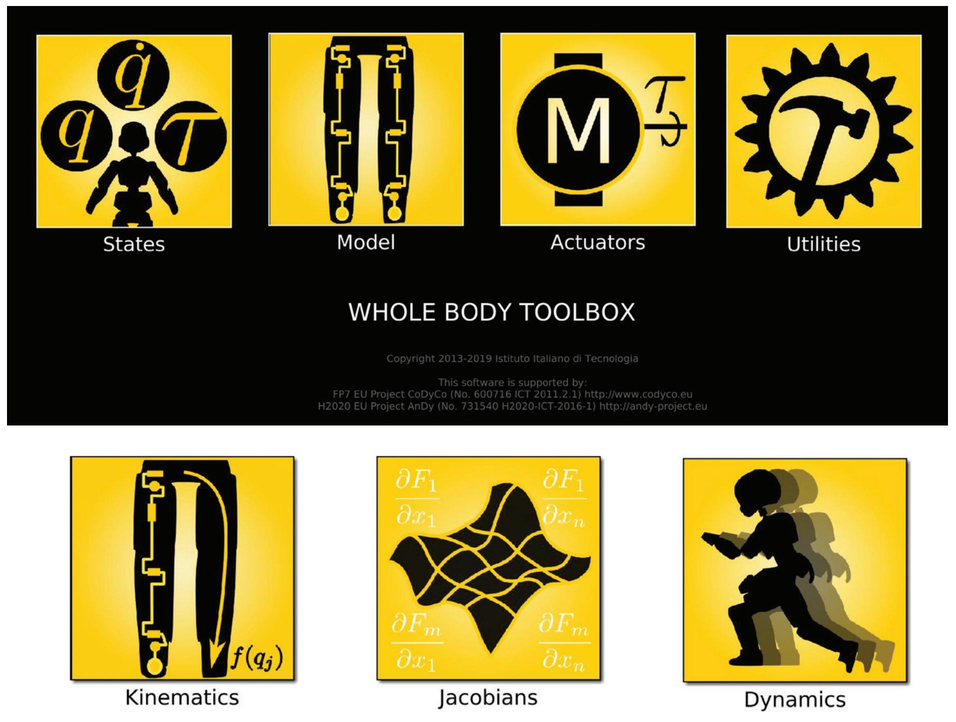

States
Model
Actuators
Utilities
WHOLE BODY TOOLBOX
Copyright 2013-2019 Istituto Italiano di Tecnologia
This software is supported by:
FP7 EU Project CoDyCo (No. 600716 ICT 2011.2.1) http://www.codyco.eu
H2020 EU Project AnDy (No. 731540 H2020-ICT-2016-1) http://andy-project.eu
Kinematics
Jacobians
Dynamics

**FIGURE 2.** *Blocks in the Whole Body Toolbox.*

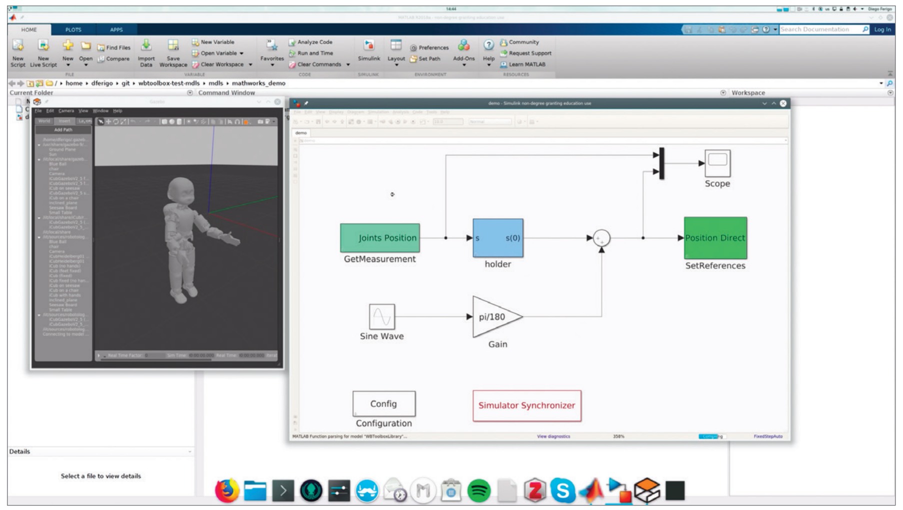


**FIGURE 3.** *Cosimulating a simple control model in Simulink with a physical model of iCub in Gazebo.*

# Beyond Algorithms and Optimization

## Helping Students to Work and Think Like Engineers

By Alwyn Hoffman, North-West University

Undergraduate engineering students are capable of solving much more complex problems than we give them credit for. I found this to be true in my *Algorithms and Optimization* course, where students solve optimization problems that require them to determine what tradeoffs have to be made to satisfy conflicting requirements and arrive at a viable solution.

While the problems I assign can be readily understood—for example, helping a farmer decide what crops to plant to maximize profits—they are too complex to solve analytically without an appropriate toolset. Students must integrate the concepts they've learned in first-year engineering courses with programming skills to develop a mathematical solution, translate it into code or a model, and then display and interpret the results.

MATLAB® and Simulink® make it possible for students to tackle problems of this complexity in one semester. Instead of focusing on low-level implementation details at the component level, they can use built-in functions and blocks to develop solutions that would take them up to 10 times longer in languages like Python, C, or C#.

### Introducing Programming in MATLAB

Students in *Algorithms and Optimization* have already taken introductory programming, but most are new to MATLAB. To familiarize them with the MATLAB language and development environment, the first assignment asks them to build a simple user interface (Figure 1). The interface incorporates animated, multicolor graphs of sample data, which engage the students more than static charts. The entire exercise gives them confidence in their ability to build an application—one they at first thought complicated—by following a straightforward process.

The first major assignment is a nonlinear optimization problem to be solved in MATLAB. In each of the three years that I've taught the course, I've presented a different problem. This year, it was the farming problem: The students had to determine how much land should be allocated to each of two crops to maximize profits, given an expected amount of rainfall and electricity costs to run irrigation pumps. Last year, the students had to optimize an energy management system to find the optimal mix of grid, solar, and wind power production. The year before that, I gave them a mining scenario in which they had to find the lowest-cost method of operating and ventilating a mine with a combination of diesel- and electric-powered load-haul-dump (LHD) equipment and ventilation fans.

Each of these scenarios is easy to understand conceptually, but none has an obvious mathematical solution. Students first translate the verbal description of the problem into a set of equations. They must then determine which nonlinear programming techniques to use and implement a solution in MATLAB. Finally, they generate a solution to the nonlinear optimization problem and pass this on to Simulink to visualize how this solution will perform in different practical scenarios.

### Dynamic Optimizations in Simulink

For the next assignments, the students learn how to solve dynamic optimization problems in Simulink, reworking their initial MATLAB solutions to handle parameters that vary over time. In the farming problem, for example, instead of assuming a constant rate of rainfall based on a seasonal average, the students must now consider rainfall amounts that vary day by day. They develop Simulink models in which rainfall, wind speed, cloud cover, and other environmental variables are simulated using randomly generated values within a predetermined range (Figure 2). They incorporate the MATLAB code from the earlier assignment as user-defined function blocks, and then refine the code and model until they find a solution that works for different sets of dynamic conditions.

To develop their models, students must first build individual components and then figure out how to combine them into a complete system. I encourage them to test early and frequently rather than waiting until the entire system is assembled. I also show them how they can use Simulink Scope blocks to plot signals at any point in their model and trace errors back through the system, just as an engineer would when debugging a real-world system.

### Working Like Engineers

While I encourage students to complete as much of the final assignment as they can by themselves, I also allow them to work in groups. Even if they choose to work together, however, every student must hand in their own assignment and be able to explain to me how each line of MATLAB code works. I tell my students that it is perfectly acceptable to incorporate partial solutions from others as long as you fully understand those solutions and how they operate. Some instructors do not agree with allowing students to include the work of their classmates, but I believe that my approach reflects the way engineers work in the real world—most engineers have neither the time nor the resources to solve every problem from scratch.

### MATLAB for Advanced Projects

I recently began teaching a fourth-year course on data analytics in which students use MATLAB to complete more advanced assignments. I based these assignments on my own research and on thesis papers written by my postgraduate students. I broke down the data analysis into three stages. In the first stage, students perform simple statistical explorations and visualizations of the data. Next, they apply statistical measures to see how various subsets of data differ from one another. Finally, they build regression, neural network, and decision tree models and use them to make predictions based on the data.

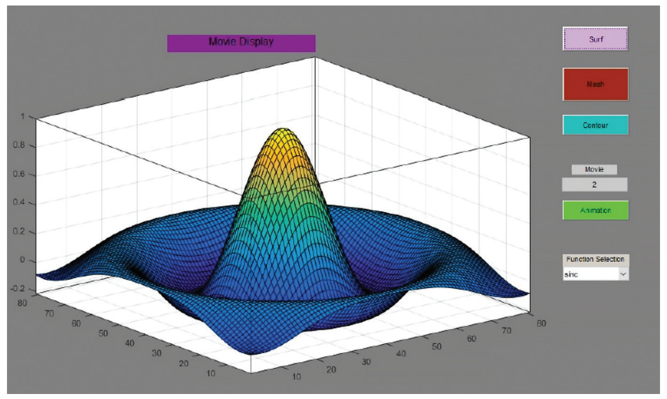


**FIGURE 1.** *GUI created by a student in MATLAB.*

The students had six weeks to complete their examination assignments. In the first few weeks, one student visited me several times during office hours with questions about how to proceed. I found out that this student shared what he had learned from me with his classmates—he was acting as a kind of senior engineer or lead engineer working with junior engineers on a real project. Almost every student in the class completed their assignments and did well.

These results confirmed to me, and proved to them, a core principle for today's engineers: You can think through and solve difficult engineering problems when you have the right tools and know how to use them. ◆

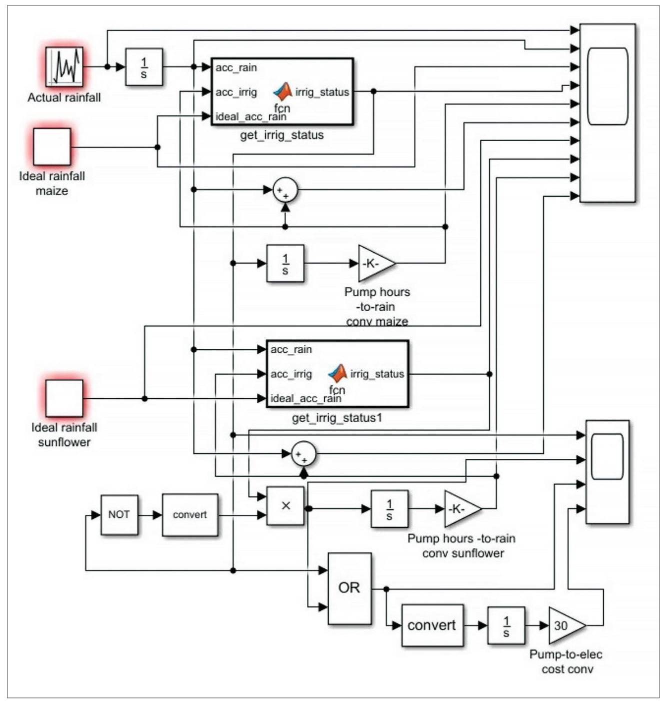
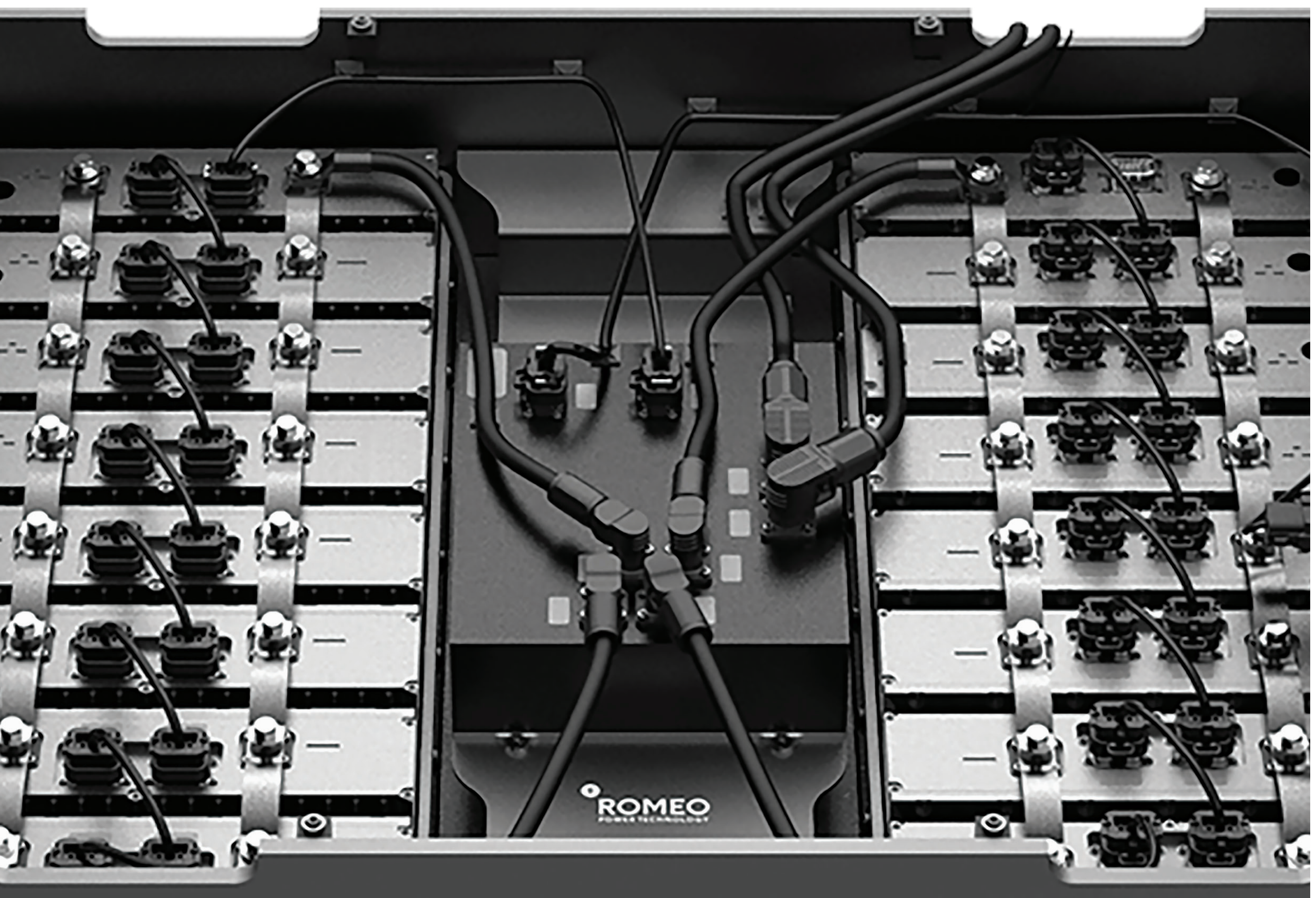


**FIGURE 2.** *Student-created Simulink model for optimizing farm crop selection.*



# BATTERY DESIGN

## Modeling and Simulating Battery Performance for Design Optimization

By Cecilia Wang, Romeo Power

At Romeo Power, we design our battery packs and battery technology to enable our customers to produce more efficient electric vehicles and implement scalable energy storage systems. Before they select one of our battery packs for their next product, our customers need to know how the pack will perform under the full range of expected operating conditions, including various temperatures and states of charge. Assessing battery pack performance using hardware prototypes can be both slow and costly, so we rely on simulation to ensure that we minimize hardware testing.

Modeling and simulation with MATLAB®, Simulink®, and Simscape™ is faster, safer, and less costly than building physical prototypes.

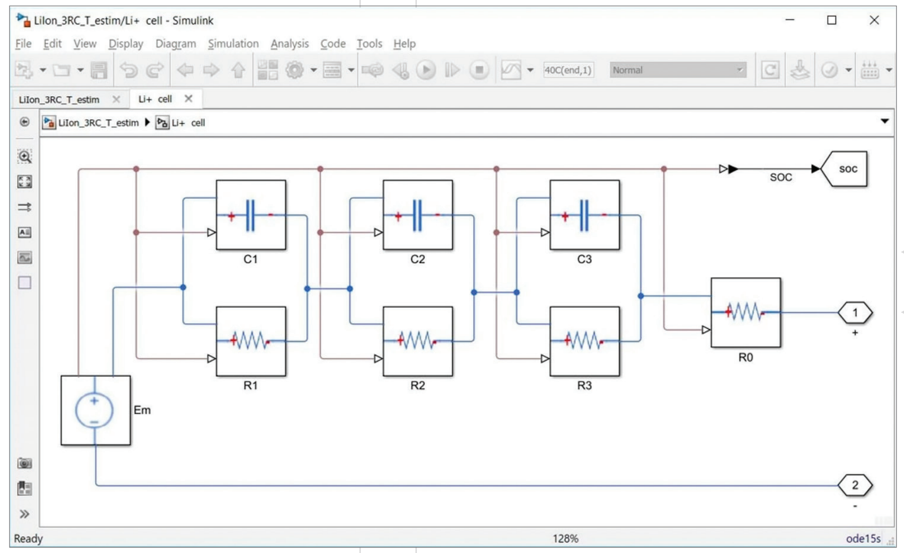

LiIon_3RC_T_estim/Li+ cell - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
LiIon_3RC_T_estim
Li+ cell
SOC
soc
C1
C2
C3
R1
R2
R3
R0
Em
1
2
Ready
128%
ode15s

FIGURE 1. *An isothermal 3-RC equivalent circuit developed for parameter estimation using Simscape blocks. Em = open-circuit voltage, R = resistance, and C = capacitance.*

We can identify algorithms or charging methods that will work for a particular design without running the whole system. We can test scenarios that would be difficult or hazardous to test on real batteries and optimize designs for specific applications and usage profiles. Simulation often reveals errors that are missed during system-level testing. In addition, our customers can use our models to evaluate battery packs and battery management systems for their electric vehicles or commercial and residential energy storage systems.

## Characterizing and Modeling Individual Cells Using Parameter Estimation

To model a battery cell, we need to characterize its properties—how it performs both initially and after multiple charge-discharge cycles, at various temperatures and states of charge. We run extensive tests, including open-circuit voltage (OCV) and hybrid pulse power characterization (HPPC) tests, using a thermal chamber to vary the cell temperature to cover the operating range of interest. We record changes in capacity and impedance at various states of charge after every aging milestone—for example, after every 200 charge-discharge cycles.

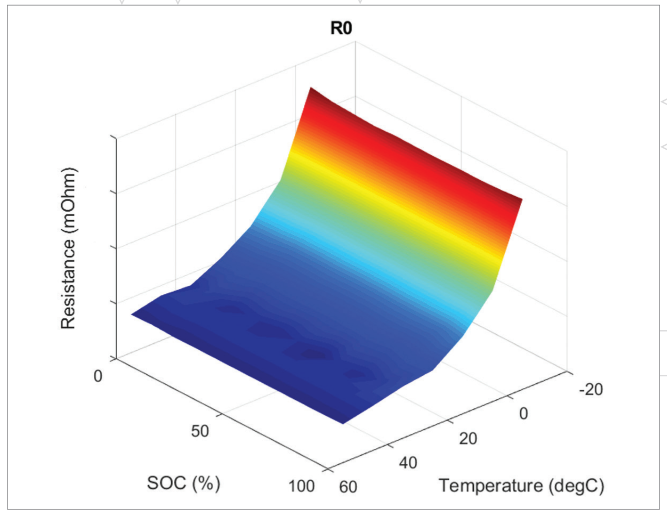


FIGURE 2. *Visualization of lookup table resulting from parameter estimation showing internal resistance as a function of state-of-charge and temperature.*

We import the measured data into MATLAB and perform parameter estimation to find open-circuit voltage, resistance, and capacitance values for an equivalent circuit model, which we build in Simulink using Simscape voltage source, resistor, and capacitor blocks (Figure 1).

Parameter estimation involves calculating the equivalent circuit parameters to match a simulation result to an experimental measurement. We start with a given equivalent circuit topology and a set of initial parameter guesses. MATLAB optimization functions calculate the parameter values that minimize the discrepancy between simulation and experiment. These steps are repeated at all the temperatures of interest to populate the lookup tables column-by-column. We repeat the parameter estimation using the data we collected as the battery aged, creating additional lookup tables for the battery at each age milestone.

As a result of the beginning of life (BOL) parameter estimation, each equivalent circuit component will have a two-dimensional lookup table with columns representing temperatures and rows representing states of charge. Figure 2 shows an example lookup table, where the internal resistance R0 is shown as a function of SOC and temperature.

To verify the parameterized model, we simulate it, plot the simulation results in MATLAB, and compare them with the battery test results (Figure 3).

## Creating Multicell Models

To create a complete battery pack or module, we link individual cell models in a series or parallel string and then connect the strings in parallel or series (Figure 4).

We insert convective heat transfer blocks between individual cells to account for thermal effects. During simulations we monitor the temperature, SOC, and voltage of individual cells as well as the temperature, voltage, and current of the complete module. By modifying the number of strings or the number of cells in each string, we can quickly evaluate different configurations and identify the best one for a specific application.

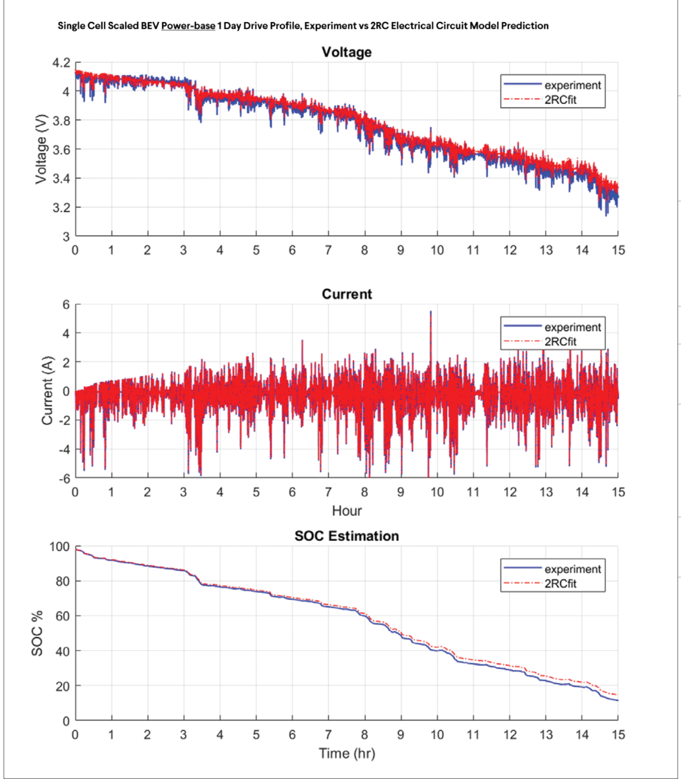


FIGURE 3. *One-day, power-driven simulation for an electric vehicle application (based on a single cell). Top to bottom: simulated and measured voltage, current, and state-of-charge.*

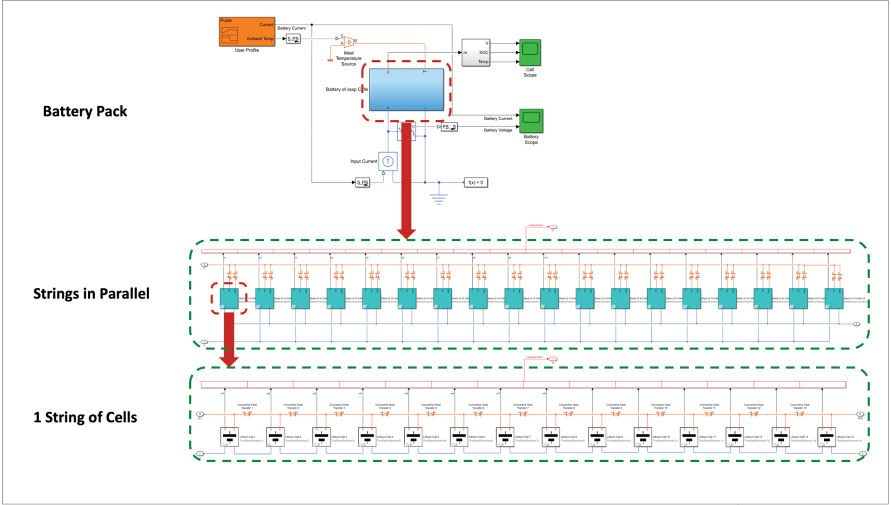


FIGURE 4. *From top to bottom: battery pack model, strings connected in parallel, and individual cells connected in series.*

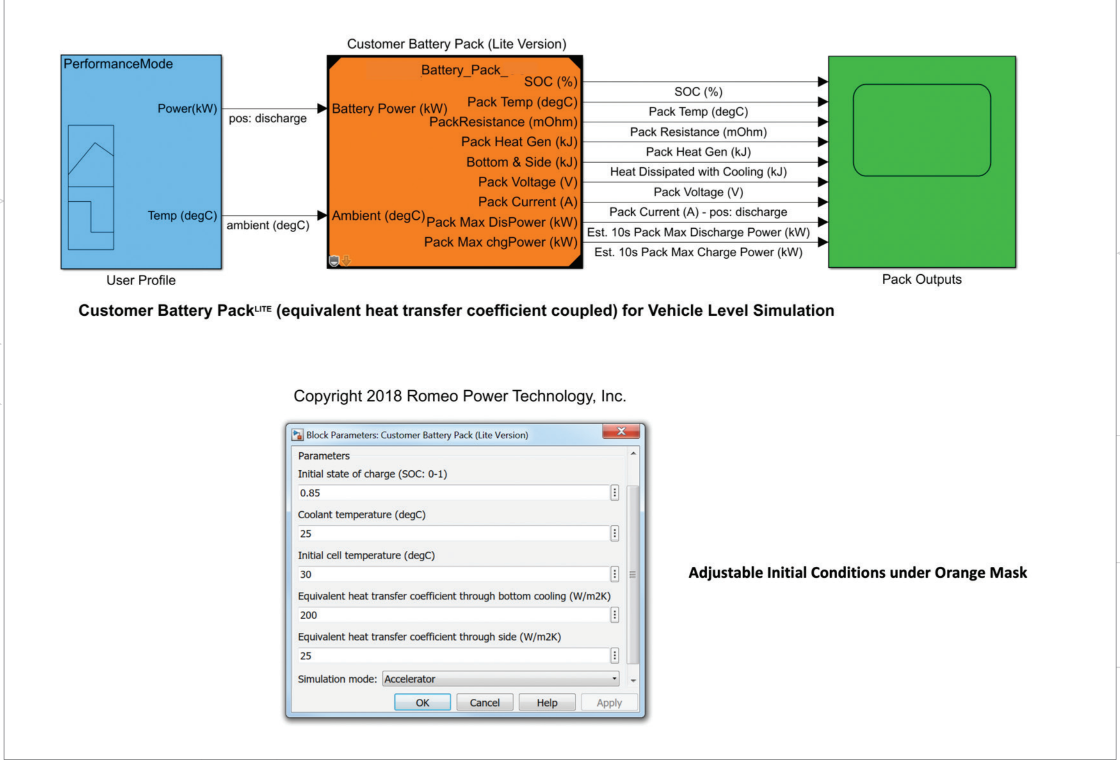
We adjust the fidelity of our models based on our own needs or the needs of our customer. We use a low-fidelity model to generate an initial design report for new customers who require a customized design, or when an existing product framework is not available for performing system sizing and preliminary analysis. We use a high-fidelity model for product validation, cell balancing, developing state estimation and charger control algorithms, hardware-in-the-loop testing, and integration into a vehicle platform.

## Sharing Models with Customers

Many of our customers run their own simulations to validate sizing or to see how a particular battery pack will work within one of their designs. A company developing electric vehicles, for example, may want to integrate a battery model with a model of the vehicle motor and run vehicle-level simulations for different drive profiles.

The vehicle model, and even the drive profiles, often contain proprietary information, as do our own battery models. To address this issue, we developed black-box versions of our battery pack models. We generated code from our original models and created new Simulink models based on the compiled code. Our customers have full control over setting up initial conditions, such as initial SOC, initial cell temperature, coolant temperature, and heat transfer coefficients (Figure 5).

We anticipate a growing demand for safe, cost-effective, and reliable batteries to meet the needs of the electric vehicle industry. By modeling and simulating in MATLAB and Simulink we can quickly explore a wide range of cell configurations and optimize the system architecture in terms of performance, weight, volume, or heat dissipation requirements. ◆



Block Parameters: Customer Battery Pack (Lite Version)

Parameters

Initial state of charge (SOC: 0-1): 0.85

Coolant temperature (degC): 25

Initial cell temperature (degC): 30

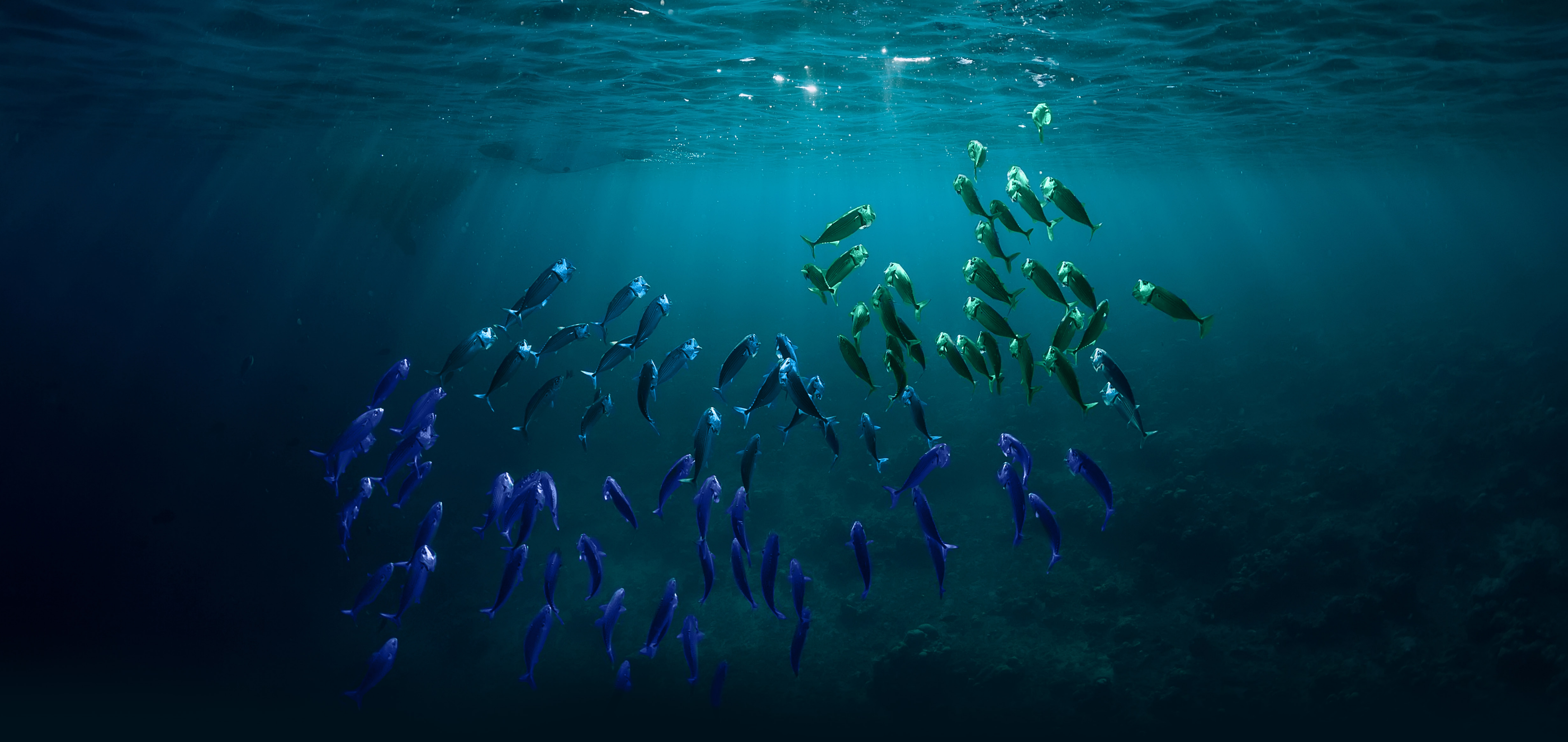
Equivalent heat transfer coefficient through bottom cooling (W/m2K): 200

Equivalent heat transfer coefficient through side (W/m2K): 25

Simulation mode: Accelerator

FIGURE 5. *Top: customer battery pack model. Bottom: interface for setting model parameters and initial conditions.*

*By modeling and simulating in MATLAB and Simulink we can quickly explore a wide range of cell configurations and optimize the system architecture in terms of performance, weight, volume, or heat dissipation requirements.*



# UNDERSEA IMAGING

## Developing an Underwater 3D Camera with Range-Gated Imaging

By Jens Thielemann, Petter Risholm, and Karl H. Haugholt, SINTEF

Underwater optical imaging has the potential to provide much higher resolution images than sonar. The clarity of these images, however, depends on the water quality. In turbid water, active illumination—used in low-light situations—causes *backscatter*, or the reflection of light from particles in the water back toward the camera (the same effect that makes it difficult to drive in fog).

To address this challenge, SINTEF worked with partners across the EU to develop UTOFIA, an imaging system for turbid environments.

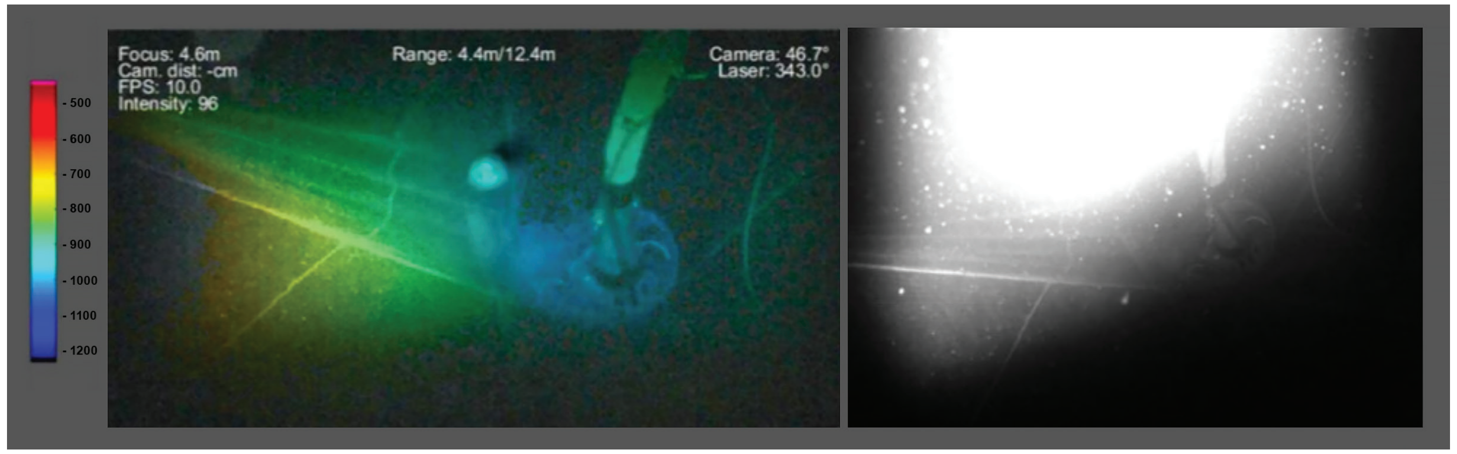


**FIGURE 2.** *Left: image taken with the UTOFIA camera and colored to show range information. Right: image of the same scene, taken with a regular camera, showing the effects of backscatter.*

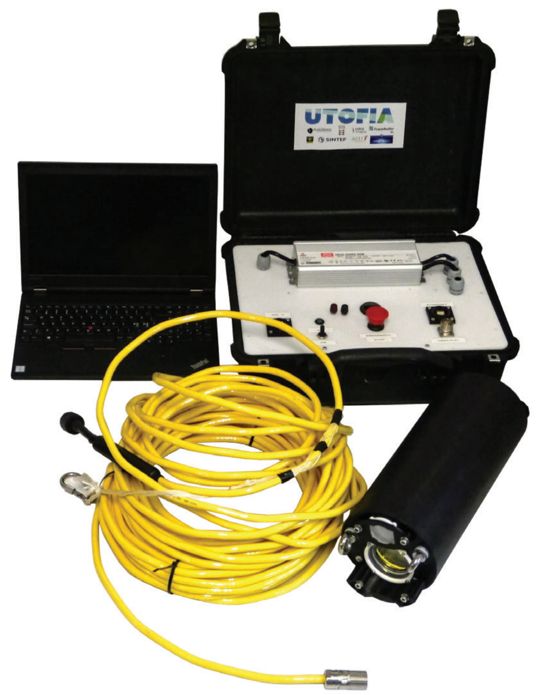


**FIGURE 1.** *The UTOFIA camera system.*

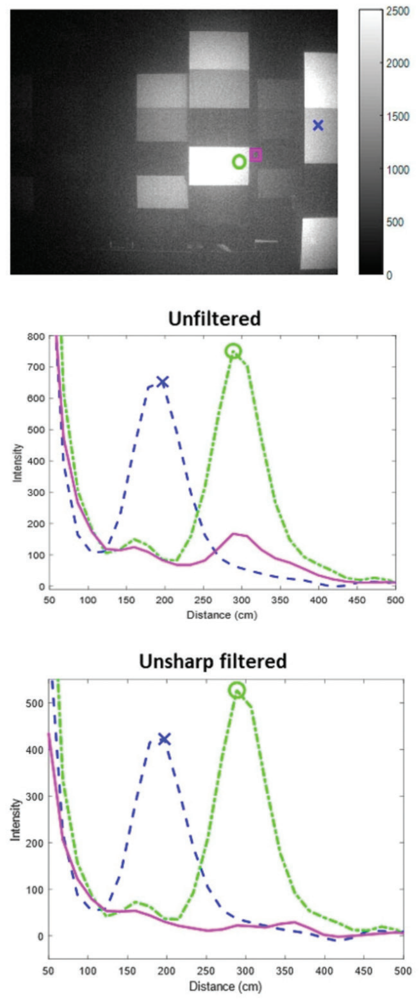


**FIGURE 3.** *Peaks in intensity as a function of distance (middle and bottom) for points in a captured image (top).*

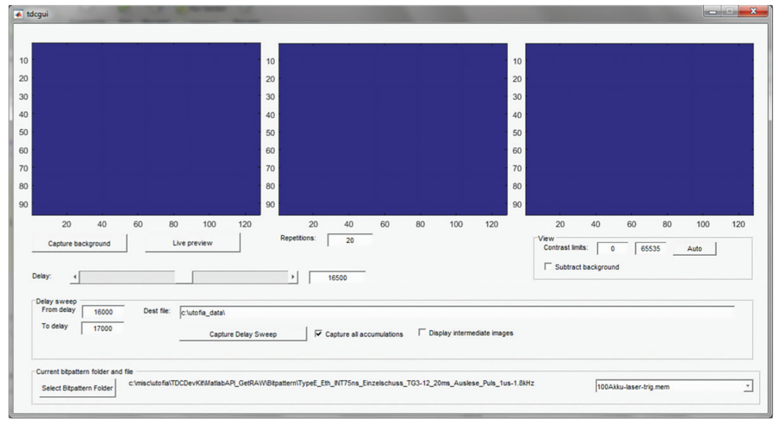


**FIGURE 4.** *MATLAB app used to automate data capture.*

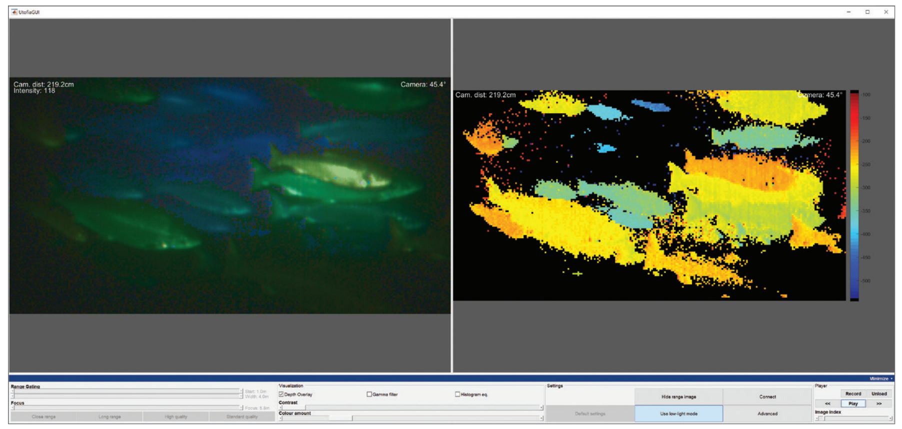


**FIGURE 5.** *MATLAB app for visualizing UTOFIA image data.*

The UTOFIA camera delivers 3D images at 10–20 frames per second with a range of up to 15 meters and a resolution of one centimeter at depths of up to 300 meters (Figure 1). It uses range-gated imaging (see sidebar) to minimize the effects of backscatter and obtain range information for objects in its field of view (Figure 2).

We developed algorithms to process raw data from the camera and produce 3D, backscatter-free images. We were working in a new domain, and we needed to quickly test new ideas. Thanks to the integrated environment of MATLAB®, with its strong visualization support, we were able to try more than 40 different approaches and techniques. In Python® or C++, each implementation and test would have taken much longer, and it is unlikely that we would have had time to test more than a handful.

## Initial Data Analysis and Peak Detection

Unlike a standard digital camera, which produces 2D arrays of pixels, our camera produces 3D arrays of cubes, with the recorded value at each cube representing the intensity of light reflected at a specific location in the field of view and at a specific distance from the camera. To extract useful images from the multigigabytes of data generated by the camera, our algorithms must identify the peaks in these intensity values (Figure 3). External factors influence the peak positions, and the scattering in the water will introduce false peaks. This reduces the clarity of the resulting image and the quality of the 3D reconstruction.

To understand the mechanism in action, we performed extensive statistical analyses on the data for various water turbidities and camera settings. These analyses involved building empirical models of backscatter, investigating the properties of forward scattering, and modeling detector response properties.

We also developed a MATLAB app to automate and control the data capture process (Figure 4). The app includes interface elements to control the pulse sweeps and a .NET interface that we used to configure capture settings and other camera components.

## Developing Algorithms for 3D Reconstruction

The camera hardware diminishes backscatter significantly, but we knew that we could reduce its effects still further in software. We developed a model of backscatter response across turbidities and implemented several algorithms for reducing backscatter effects. We explored many alternatives here, including homomorphic filtering and variations over histogram equalization, finally selecting unsharp filtering, which also improved our 3D performance. In addition, we developed algorithms for camera calibration, 3D estimation, peak detection, and peak fitting.

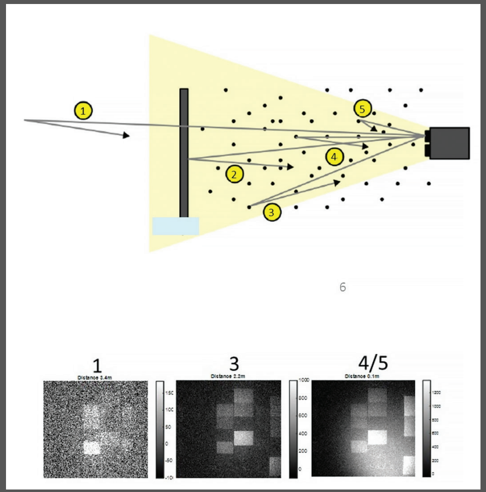
## Visualizing Image Data

Once we had analyzed the data and developed 3D reconstruction algorithms, we needed to share the results they produced with other organizations in the UTOFIA consortium. To do this, we built a second MATLAB app for visualizing UTOFIA image data (Figure 5). This app includes controls for adjusting options and algorithm parameters, including contrast, focus, noise removal, and histogram equalization. Users can set these parameters and immediately see the effects on screen.

We packaged a standalone version with MATLAB Compiler™ and distributed it to our partners, who provided us with feedback and enhancement requests. Using MATLAB and MATLAB Compiler, we could implement the changes they requested in a few days.

### Range-Gated Imaging

Instead of illuminating a target with a constant stream of light, range-gated imaging uses nanosecond-long pulses of light produced by a strobed laser. Light reflecting off particles in front of the target returns to the camera slightly earlier than light reflecting off the target itself. We can suppress backscatter by controlling the camera shutter to capture just the light reflected by the target and very little of the light reflected by particles in the water. In addition, we can accurately determine the distance to the target by measuring the time-of-flight of individual light pulses and dividing by the speed of light.



*Top: diagram showing a range-gated imaging camera. Bottom: images captured at various distances from the camera.*

Implementing these changes in C/C++ or a similar language would have taken weeks, if not months.

## Continued Development

We have completed the first phase of the UTOFIA project: the development of the camera and its core software. We are now performing additional processing on the image and 3D data for industry-specific applications and looking into the second phase of the project: applying machine learning and deep learning to the images to recognize objects and other phenomena.

The availability of real-time 3D data has opened new possibilities for improving processes in the fishery and aquaculture industries, particularly in the area of automated, quantitative analysis. For example, at an aquaculture facility in Spain, we used the camera to identify and measure the length of red tuna (Figure 6).

*We built a MATLAB app for visualizing UTOFIA image data. This app includes controls for adjusting options and algorithm parameters, including contrast, focus, noise removal, and histogram equalization. Users can set these parameters and immediately see the effects on screen.*

At a research facility in Norway, we used UTOFIA for behavior analysis, tracking individual fish over time to estimate swimming speed and patterns (Figure 7).

Meanwhile, in aquaculture trials of the camera, fish and other marine life are being observed in low light and high turbidity conditions for biomass estimation (Figure 8).

These conditions would have been impenetrable with a traditional underwater camera. ◆

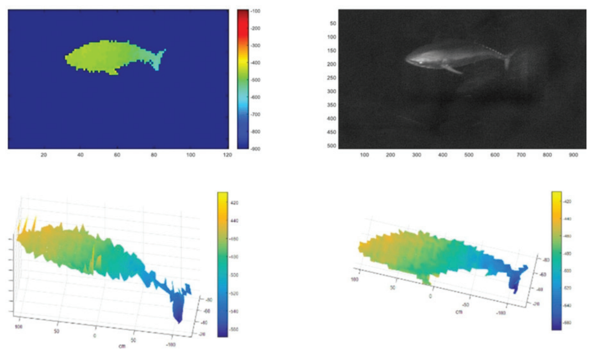


**FIGURE 6.** *Red tuna measurement results produced by the UTOFIA camera.*

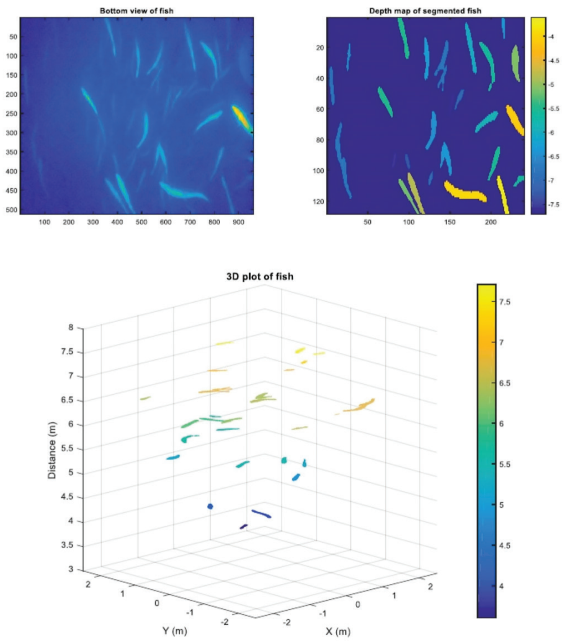


**FIGURE 7.** *Visualizations of fish biomass and behavior patterns.*

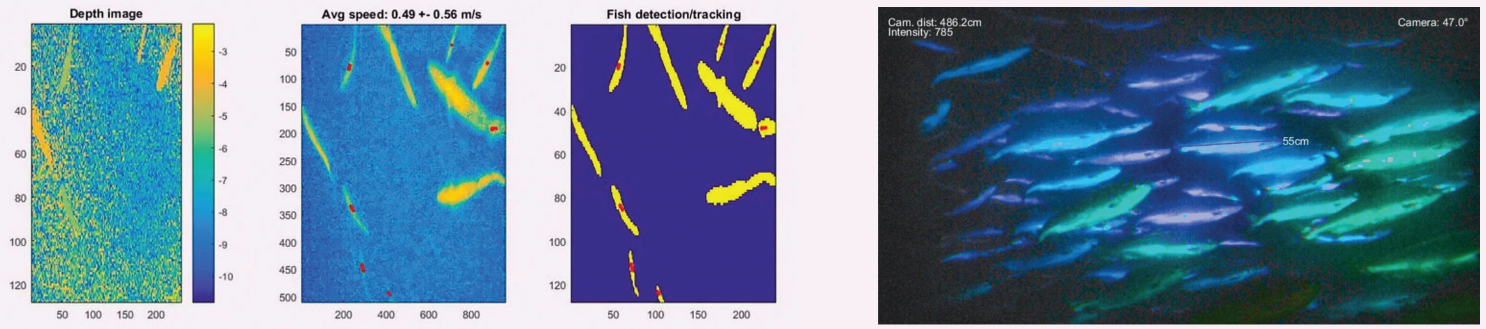


**FIGURE 8.** *Left: images used to track individual fish. Right: image overlaid with length measurement.*

# An Airplane That Flies Without Moving Parts

**Researchers at MIT have successfully demonstrated an airplane that has no propellers, rockets, or jet turbines. It's the first plane ever to fly using ionic wind.**

The plane is powered by a battery pack that applies a 20,000-volt charge to an array of wires that run the 5-meter length of the wings. Airfoils behind each wire are charged to 20,000 volts. The voltage strips electrons from the nitrogen molecules in the air, creating ions that flow from the wires and across the airfoils.

The flow of ions creates "ion drive," or ionic wind as the ions bump into air molecules.

Ionic propulsion opens new possibilities for developing aircraft that are quieter and emission-free. Meanwhile, conventional jets could improve their operation while reducing noise and pollution by incorporating the technology today.

**See the plane in flight and learn how MATLAB® was used to test it:**

***mathworks.com/solid-state-airplane***

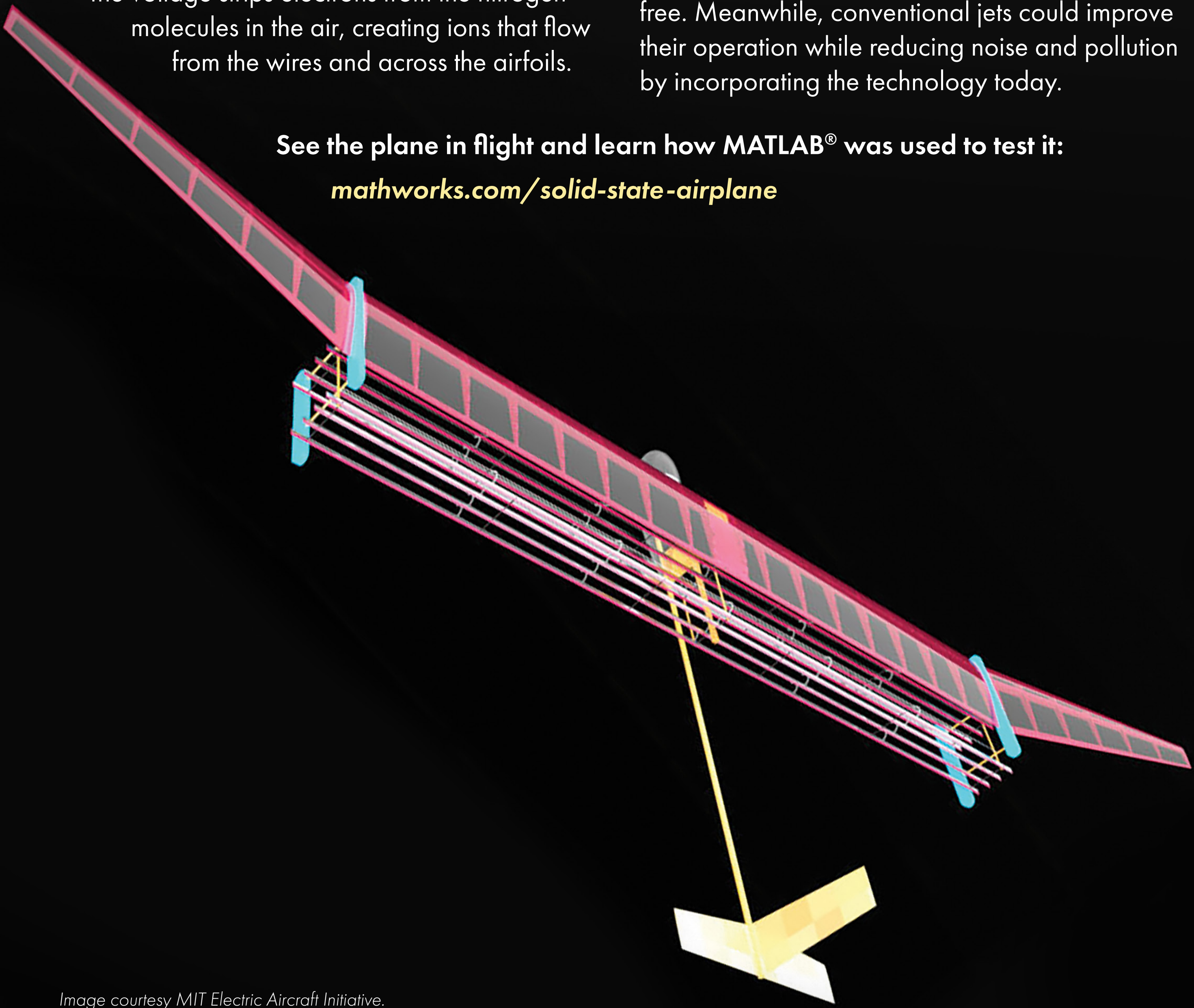


*Image courtesy MIT Electric Aircraft Initiative.*

# EMERGING COMPANIES ON A MISSION

Biology-guided radiotherapy, crop-protecting drones, broadband supertowers, intelligent electric scooters ... Combining novel technologies with bold vision, engineers at emerging companies are on a mission to improve the lives of individuals and entire communities.

## Riding the Wave of Industry 4.0

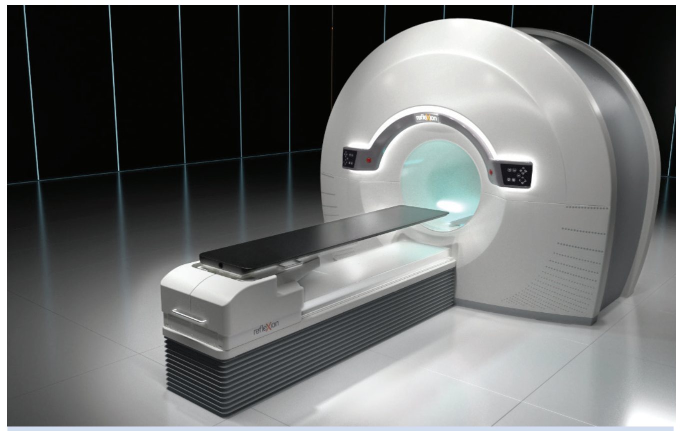
Industrial plants are gathering increasing amounts of data as they move toward full interconnectivity and automation. Until now, this data could only be accessed on a desktop or tablet. GlassUpF4 augmented reality (AR) visors enable machine operators to access the data on the shop floor—and without stopping work. The visors include a video camera, an LED lighting system, voice control, and Bluetooth®, as well as a gyroscope, compass, and accelerometer. A remote-control dashboard allows the user to share their point of view with any paired visor for on-the-job training and remote maintenance.

## Beaming Mobile Broadband to Isolated Communities

Altaeros plans to turn tethered blimps, or *aerostats*, into autonomous telecommunication SuperTowers that deliver high-speed broadband to the 4 billion people worldwide without access to the internet. One SuperTower could service an area that would require up to 20 conventional cell phone towers. Altaeros has completed pilot tests and is working with the FCC and the FAA to meet all US federal requirements to operate the SuperTower commercially.

*"Our hope is that by accelerating the rollout of rural telecom infrastructure, we can help bring underserved communities online for the first time."*

– Igor Braverman, Altaeros



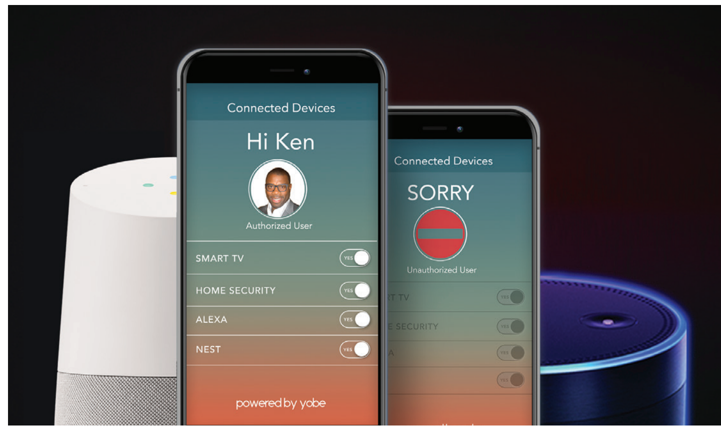
## Turning Cancer on Itself

RefleXion Medical is developing the first-ever biology-guided radiotherapy system for cancer treatment. The system combines a PET scanner and a radiotherapy/radiosurgery machine in a novel way. By means of PET tracers, tumors continuously signal their location in real time during treatment. This capability could one day revolutionize cancer care by enabling clinicians to treat multiple tumors in a single session.[1]

[1] The RefleXion machine requires 510(k) clearance and is not yet commercially available.

*"Using biology to guide radiotherapy, we hope to have the means to turn cancer on itself."*

– Sam Mazin, Ph.D., RefleXion



## Solving the "Cocktail Party Problem"

Using a hearing aid, smart home appliance, or other speech-recognition device in a noisy environment such as a party is challenging because the microphone has difficulty distinguishing one voice from all the others. Yobe's Voice Identification System for User Profile Retrieval (VISPR) combines artificial intelligence (AI) and signal processing to identify a signal's biometric characteristics, identify individual speakers, and distinguish speech from noise.

*"The ways we have been interacting with machines up until now have been artificial because these machines haven't been able to hear us. The natural way to communicate with something is to talk to it."*

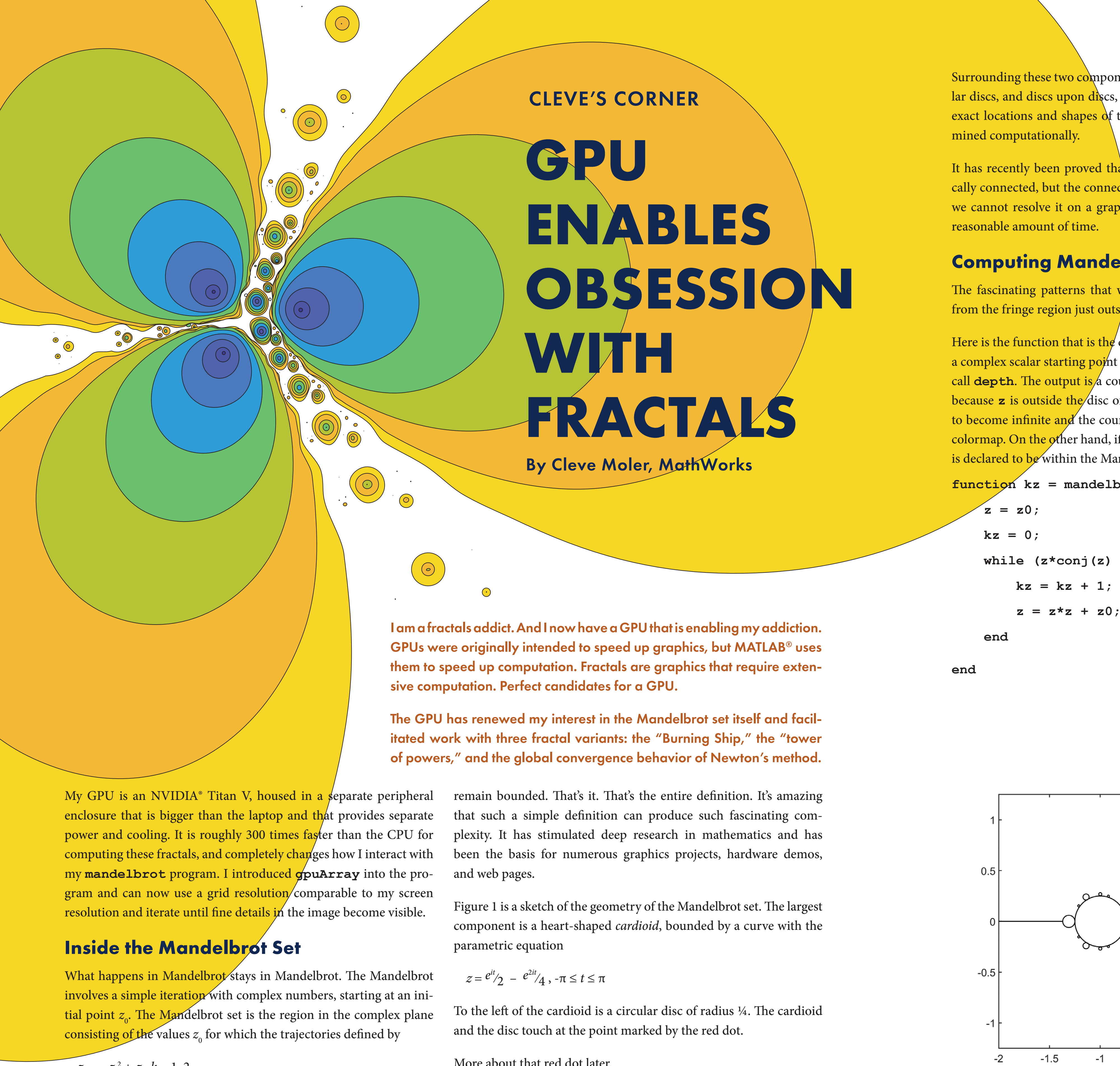
– Ken Sutton, Yobe

## Transforming the City Commute

Ather Energy's 450 electric scooter is the first of its kind in India. Designed for city driving, the Ather 450 can autonomously navigate tight parking spots, complicated street systems, and stop-and-go traffic. It is powered by a high-capacity li-ion battery pack, and has a charging range of 46 m and a top speed of 50 mph.

*"We set out to change the way people perceive electric vehicles ... In the last five years, we have not only built the scooter but an ecosystem for an electric vehicle future."*

– Tarun Mehta, Ather Energy



# GPU ENABLES OBSESSION WITH FRACTALS

By Cleve Moler, MathWorks

**I am a fractals addict. And I now have a GPU that is enabling my addiction. GPUs were originally intended to speed up graphics, but MATLAB® uses them to speed up computation. Fractals are graphics that require extensive computation. Perfect candidates for a GPU.**

**The GPU has renewed my interest in the Mandelbrot set itself and facilitated work with three fractal variants: the "Burning Ship," the "tower of powers," and the global convergence behavior of Newton's method.**

My GPU is an NVIDIA® Titan V, housed in a separate peripheral enclosure that is bigger than the laptop and that provides separate power and cooling. It is roughly 300 times faster than the CPU for computing these fractals, and completely changes how I interact with my **`mandelbrot`** program. I introduced **`gpuArray`** into the program and can now use a grid resolution comparable to my screen resolution and iterate until fine details in the image become visible.

## Inside the Mandelbrot Set

What happens in Mandelbrot stays in Mandelbrot. The Mandelbrot involves a simple iteration with complex numbers, starting at an initial point $z_0$. The Mandelbrot set is the region in the complex plane consisting of the values $z_0$ for which the trajectories defined by

$$z_{k+1} = z_k^2 + z_0,\ k = 1, 2, \ldots$$

remain bounded. That's it. That's the entire definition. It's amazing that such a simple definition can produce such fascinating complexity. It has stimulated deep research in mathematics and has been the basis for numerous graphics projects, hardware demos, and web pages.

Figure 1 is a sketch of the geometry of the Mandelbrot set. The largest component is a heart-shaped *cardioid*, bounded by a curve with the parametric equation

$$z = e^{it}/2 - e^{2it}/4,\ -\pi \le t \le \pi$$

To the left of the cardioid is a circular disc of radius ¼. The cardioid and the disc touch at the point marked by the red dot.

More about that red dot later.

Surrounding these two components are infinitely many nearly circular discs, and discs upon discs, with ever-decreasing diameters. The exact locations and shapes of the smallest discs can only be determined computationally.

It has recently been proved that the Mandelbrot set is mathematically connected, but the connected region is sometimes so thin that we cannot resolve it on a graphics screen or even compute it in a reasonable amount of time.

## Computing Mandelbrot

The fascinating patterns that we associate with Mandelbrot come from the fringe region just outside the set.

Here is the function that is the core of the computation. The input is a complex scalar starting point **`z0`** and a real scalar parameter that I call **`depth`**. The output is a count **`kz`**. If the iteration is terminated because **`z`** is outside the disc of radius of two, then **`z`** was destined to become infinite and the count **`kz`** can be used as an index into a colormap. On the other hand, if **`z`** survives **`depth`** iterations, then it is declared to be within the Mandelbrot set.

```
function kz = mandelbrot(z0,depth)
   z = z0;
   kz = 0;
   while (z*conj(z) <= 4) & (kz <= depth)
      kz = kz + 1;
      z = z*z + z0;
   end
end
```

Let's generate a grid of points in the complex plane covering a square of width **`w`**, centered at the complex point **`zc`**.

```
grid = 512;
s = w*(-1:1/grid:1);
[u,v] = meshgrid(s+real(zc),s+imag(zc));
```

Now comes the only difference between a program that runs on the CPU and one that runs on the GPU. To generate the grid on a CPU,

```
z0 = u + i*v;
```

For the GPU,

```
z0 = gpuArray(u + i*v);
```

Then, for either processor, the statement

```
kz = arrayfun(@mandelbrot,z0);
```

applies the scalar function **`mandelbrot`** to all the elements of the array **`z0`**. On the CPU, this is a vectorized double **`for`** loop over the grid. On the GPU, the statement is broken down into hundreds of individual tasks that run in parallel.

Now let's look at two extraordinary images derived from the Mandelbrot fringe.

## On the Fringe

The region known as the Valley of the Seahorses lies between the central cardioid and the disc to its left. There are actually two valleys, one above and one below the real axis. They meet at the red dot that we saw in Figure 1. With a little imagination, you can picture small marine creatures living in Figure 2. As we shall see later, the Valley also contains buried π.

Because the Mandelbrot set is self-similar, it contains an infinite

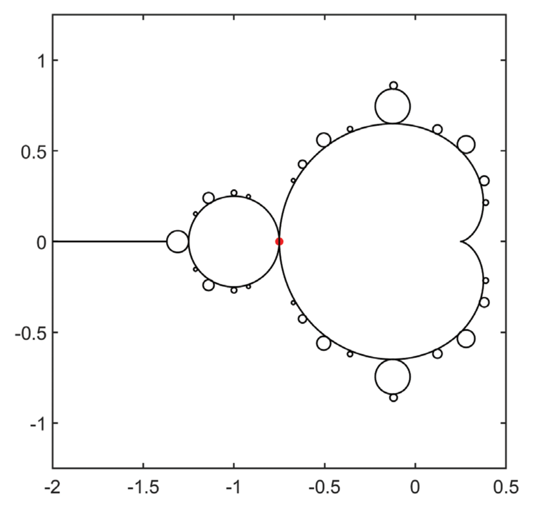


FIGURE 1. *Sketch of Mandelbrot set geometry.*

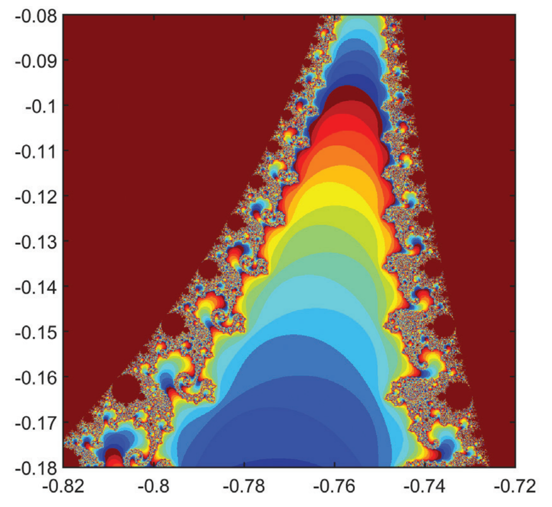


FIGURE 2. *Valley of the Seahorses.*

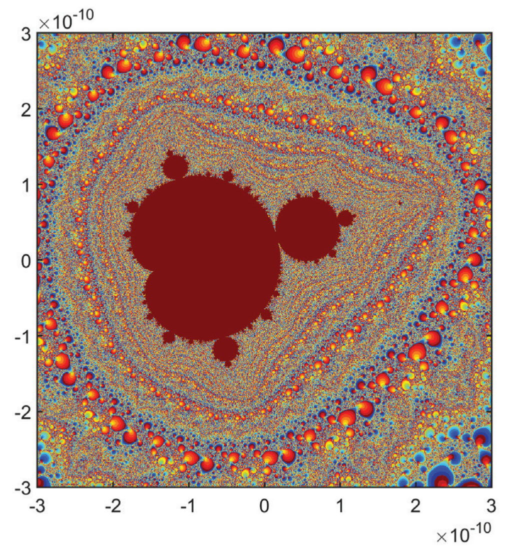


FIGURE 3. *Miniature Mandelbrot.*

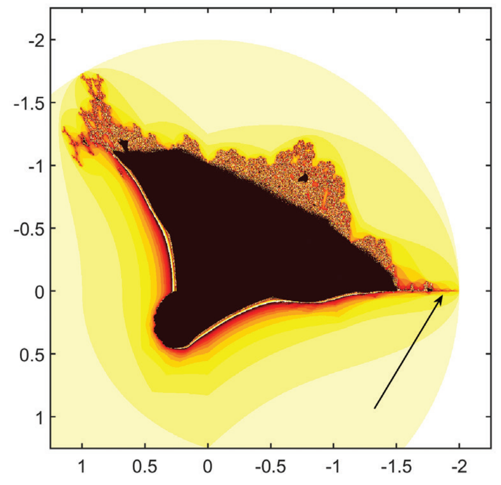


FIGURE 4. *Burning ship, initial domain.*

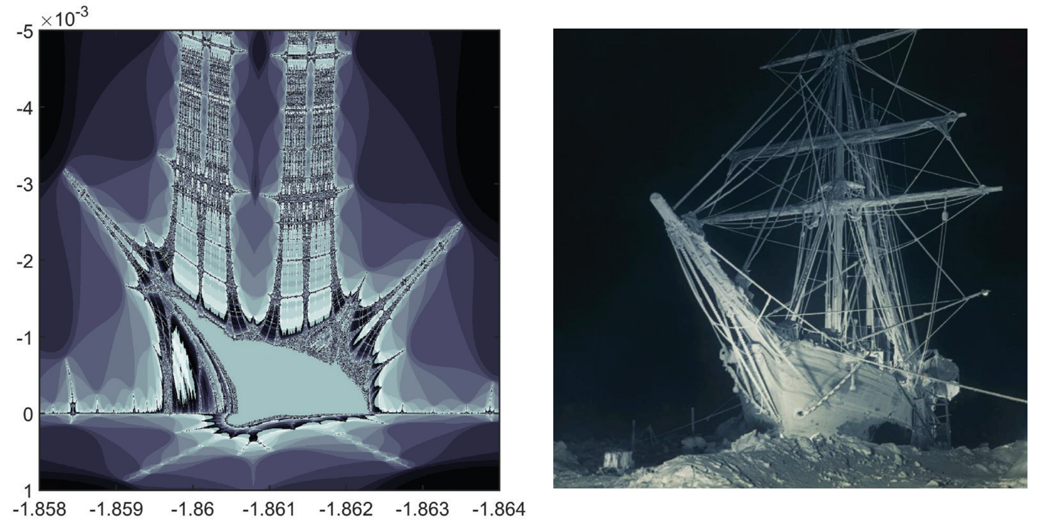


FIGURE 5. *Left: zoomed-in view of the Burning Ship's wake. Right: Hurley's 1915 photograph of Shackleton's ship frozen in ice in Antarctica.*[2] *Photo credit: Frank Hurley, National Library of Australia, nla.gov.au/nla.obj-158931586/view.*

number of miniature Mandelbrots, each with the same shape as the big one. The one shown in Figure 3 has a magnification factor of $10^{10}$.

## Buried π

A remarkable result was discovered in 1991 by Dave Boll, then a graduate student at Colorado State University. Boll was investigating the behavior of the Mandelbrot iteration in the Valley of the Seahorses. The valleys become narrower as they approach the axis, which they meet at (-3/4,0), the red dot in Figure 1.

We can repeat Boll's computation on a small grid centered just off the axis at the point -3/4 + *yi* for a tiny value of *y* and imaginary unit *i*.

```
    y = 1.0e-07

    zc = -3/4 + y*i
```

Make the grid just large enough to touch the axis.

```
    width = 2*y

    grid = 4
```

Choose **`depth`** to be inversely proportional to *y*, which makes it huge.

```
    depth = 4/y
```

With these parameters, run the code above. It produces

```
kz =
40000000    40000000    40000000    40000000    40000000
40000000    40000000    40000000    40000000    40000000
40000000    40000000    31415926    40000000    40000000
40000000    40000000    20943951    40000000    40000000
40000000    40000000    15707963    40000000    40000000
```

Look at the iteration count in the center of the grid. See a familiar value?

This isn't a fluke. A 2001 paper by Aaron Klebanoff, "π in the Mandelbrot Set," analyzes a similar computation in the cusp at the front of the cardioid.

Next, a curious variant of the Mandelbrot iteration.

## The Burning Ship

The Burning Ship comes from a strange iteration:

$$z_{k+1} = F(z_k) + z_0,\ k = 1, 2, \ldots$$

where

$$F(z) = (|\mathrm{Re}(z)| + i\,|\mathrm{Im}(z)|)^2$$

I say this is strange because the function F(*z*) is not analytic. I am interested in this iteration because of the uncanny similarities in the following pictures.

The initial domain, shown in Figure 4, is a square of width 3.5 centered at -0.5-0.5*i*. I've inserted an arrow pointing to the region of interest in the wake of the ship.

Zoom in on the ship's wake by a factor of 500 to the point -1.861-.002*i*. Apply the **`bone`** colormap to make it appear cold instead of burning. Figure 5 shows the resulting fractal next to a 1915 photograph of Antarctic explorer Ernest Shackleton's ship Endurance frozen in the ice in the Weddell Sea.

## Tower of Powers

Start with any complex number, *z*, and repeatedly exponentiate it.

$$z, z^z, z^{z^z}, z^{z^{z^z}}, \ldots$$

We can express this as an iteration. Start with $y_0 = 1$ and let

$$y_{k+1} = z^{y_k}$$

If *z* is too big, this iteration will blow up to infinity. For some *z*, it will converge to a finite limit. For example, if $z = \sqrt{2}$, the $y_k$ will converge to 2. The most interesting case is when $y_k$ approaches a cycle. For example, if *z* is near 2.5+4*i*, the cycle has length 7.

```
     2.4684 + 4.0754i
    -0.6216 + 0.3634i
     0.2603 - 0.0184i
     1.4868 + 0.3613i
    -3.4877 + 6.1054i
     7.7632e-06 - 2.6617e-06i
     1.0000 + 0.0000i
     2.4684 + 4.0755i
```

This cycle length is the basis for the "tower of powers" fractal. Figure 6 shows the overall fractal.

Zooming in by a factor of $10^5$ and changing the colormap produces the image shown in Figure 7.

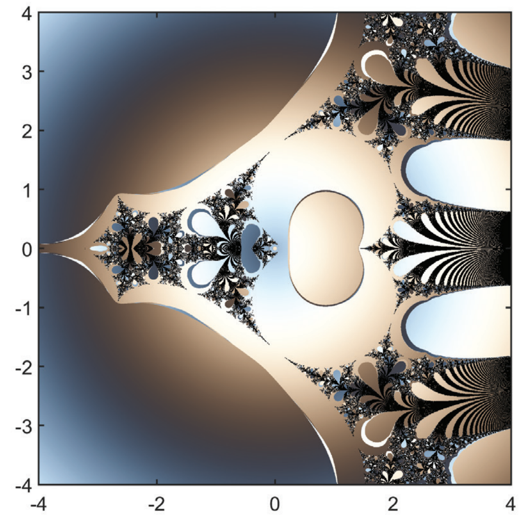


FIGURE 6. Tower of powers fractal.

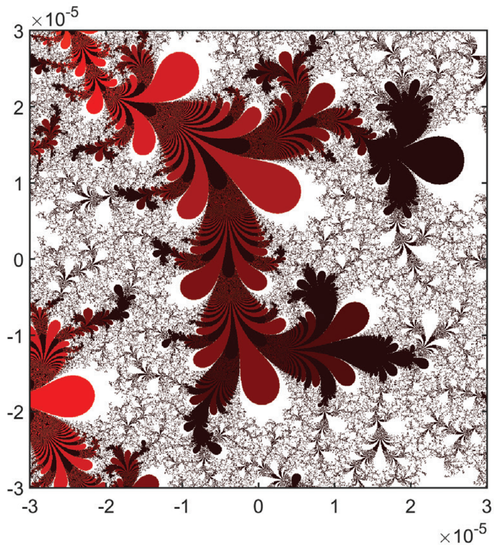


FIGURE 7. Detail of tower of powers fractal.

## Newton's Method

When the starting point of Newton's method is not close to a zero of the function, the global behavior can appear to be unpredictable. Contour plots of iteration counts to convergence from a region of starting points in the complex plane generate thought-provoking fractal images.

The iteration is familiar. Pick a function f($z$) with derivative f'($z$). Start at $z_0$ and let

$$z_{k+1} = z_k - f(z_k)/f'(z_k)$$

This will eventually converge to a zero of f. Count the number of iterations it takes to get close.

There are many functions (and colormaps) to choose from. My favorite cubic polynomial is

$$f(z) = z^3 - 2z - 5$$

Figure 8 shows the complex plane divided into three regions where the iteration converges to one of the three zeroes of the cubic. Between these regions are areas of intense fractal action, shown in black in the figure.

Figure 9 shows the global behavior of Newton's method seeking the zeroes of

$$f(z) = \tan(\sin(z)) - \sin(\tan(z))$$

The function has infinitely many zeroes and an unbounded first derivative.

The function for Figure 10 is

$$f(z) = z \sin(1/z)$$

The most prominent blue regions surround the zeroes at $\pm 1/\pi$.

Many of the images described here were new to me—I'd never seen them before. Interactive experiments with the GPU made them possible. ◆

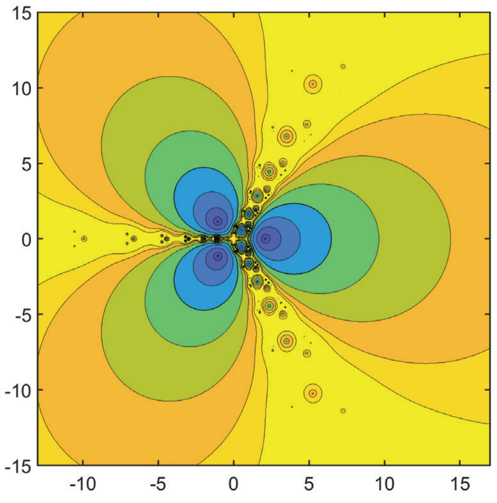


**FIGURE 8.** *Newton's iteration on $z^3 - 2z - 5$.*

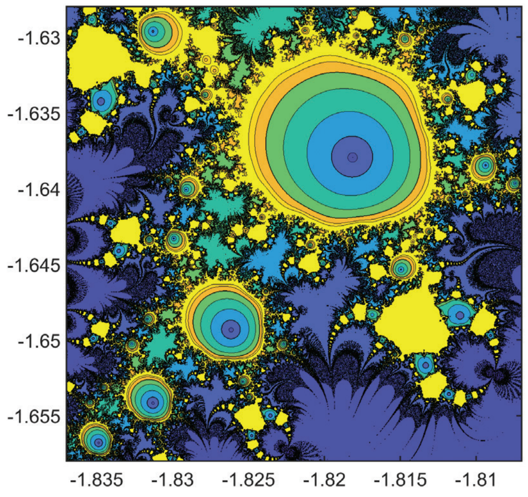


**FIGURE 9.** *Detail from Newton's iteration on tan(sin(z)) – sin(tan(z)).*

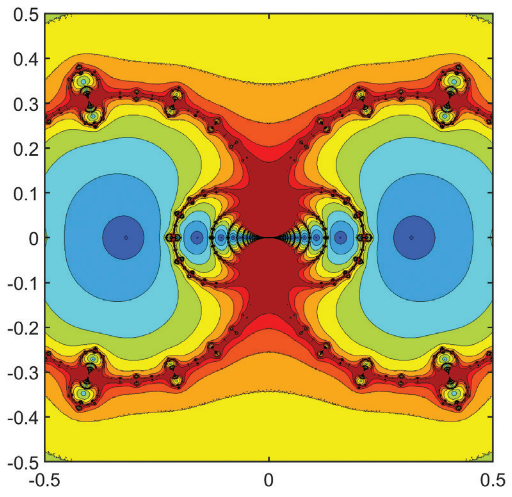


**FIGURE 10.** *Detail from Newton's iteration on z sin(1/z).*

## References

Aaron Klebanoff, "π in the Mandelbrot Set," *Fractals*, World Scientific Publishing Company, vol. 9, 2001. http://www.pi-e.de/PDF/mandel.pdf

Frank Hurley, 1915, *The long, long night [the Endurance in the Antarctic winter darkness, trapped in the Weddell Sea, Shackleton expedition, 27 August 1915],* National Library of Australia, http://nla.gov.au/nla.obj-158931586

THIRD-PARTY PRODUCTS

# Solutions for Developing Wireless Systems with MATLAB and Simulink

Wireless hardware platforms integrated with MATLAB® and Simulink® enable engineers, hobbyists, and students to move from theory to prototyping designs and deploying production systems. Radio frequency (RF) transceivers support a range of frequencies and modulation schemes. Programmable software-defined radio (SDR) hardware kits facilitate experimentation with new communication methods. New integrated system-on-chip (SoC) devices combine programmable controllers with RF converters, enabling engineers to design and deploy 5G, LTE, WLAN, and other wireless systems on a single chip.

## Analog Devices: PlutoSDR

The ADALM-PLUTO Active Learning Module (PlutoSDR) is a platform for engineers to learn wireless and SDR fundamentals and experiment with SDR communications schemes. PlutoSDR includes the Analog Devices® RF Agile Transceiver (AD9363) and the Xilinx Zynq Z-7010 all-programmable FPGA. The transceiver features an RF front end, a flexible mixed-signal baseband, and integrated frequency synthesizers to provide tunable operation (70 MHz to 6.0 GHz) with channel bandwidths from 200 kHz to 56 MHz. A Communications Toolbox™ add-on package enables engineers to prototype, verify, and test practical wireless systems such as FM radio and WLAN signals on PlutoSDR under real-world conditions.

*analog.com/plutosdr*

## Ettus Research/National Instruments: USRP Hardware

The USRP® product line includes tunable transceivers with frequencies from DC to 6 GHz for designing, prototyping, and deploying radio communication systems. The USRP Bus Series is best for low-cost experimentation, while the USRP Networked Series, USRP X Series, and standalone USRP devices are designed for prototyping more sophisticated systems. In addition, the USRP Embedded Series is ruggedized for field deployment. With Communications Toolbox and USRP hardware support packages, engineers can use USRP radios as peripherals for importing live RF data I/O into MATLAB. They can generate code with HDL Coder™ for deployment on select USRP Embedded Series devices.

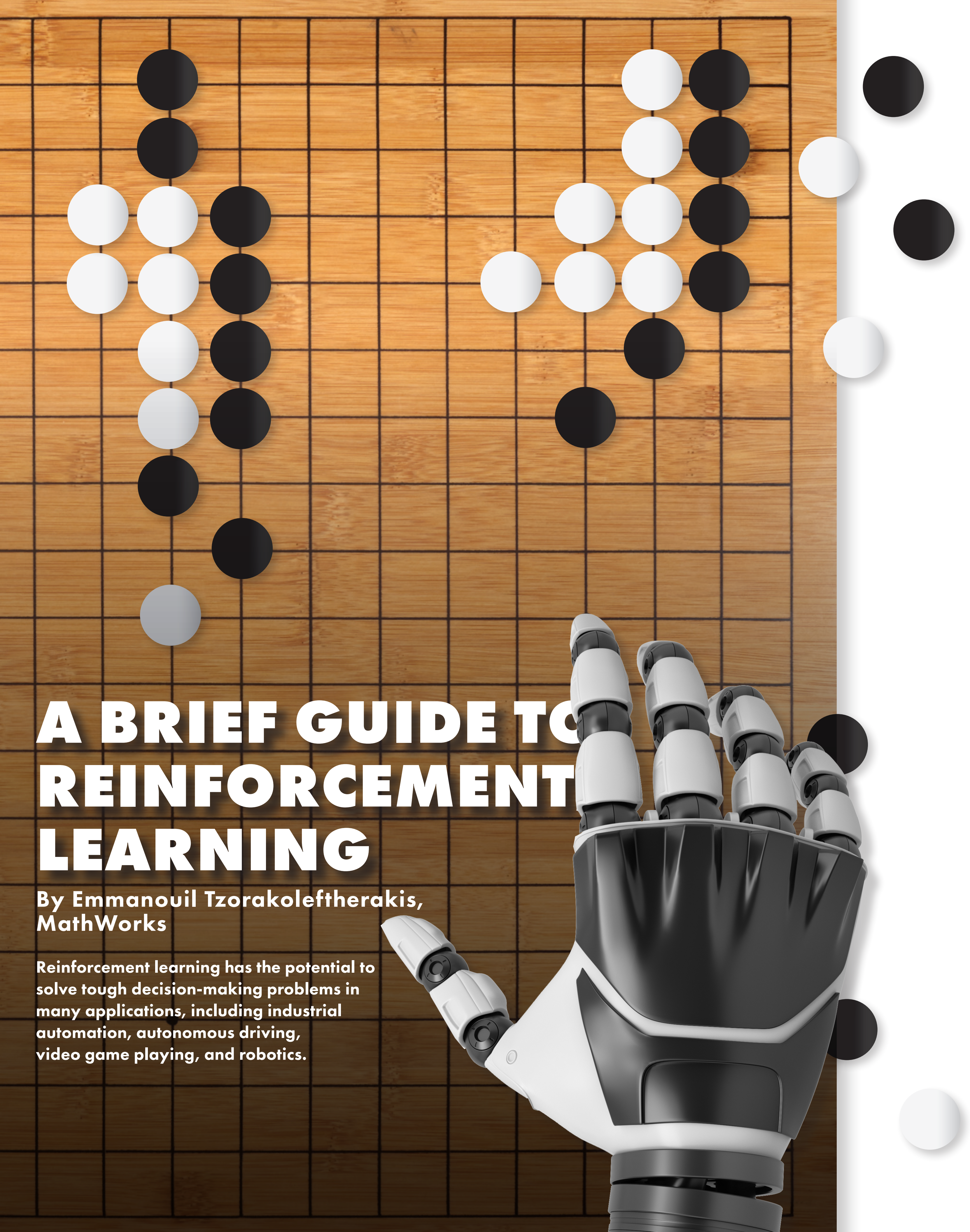
*ni.com/usrp*
*ettus.com*

## Xilinx: Zynq UltraScale+ RFSoC

Xilinx® Zynq® UltraScale+™ RFSoC integrates multi-gigasample RF data converters with the UltraScale architecture. With full support for sub-6GHz bands, it is designed for the latest wireless infrastructure standards and for a range of test and measurement, radar, and other high-performance, multichannel applications. Engineers can stream RF data into and out of MATLAB and Simulink to test designs under real-world conditions. To prototype and deploy custom designs, they can target Zynq UltraScale+ RFSoCs with HDL Coder and Embedded Coder®. With Avnet® RFSoC Explorer software they can characterize RF performance using waveforms from 5G Toolbox™, WLAN Toolbox™, and LTE Toolbox™ or custom waveforms.

*xilinx.com/rfsoc*

Software-Defined Radio Resources: *mathworks.com/sdr*



# A BRIEF GUIDE TO REINFORCEMENT LEARNING

By Emmanouil Tzorakoleftherakis, MathWorks

**Reinforcement learning has the potential to solve tough decision-making problems in many applications, including industrial automation, autonomous driving, video game playing, and robotics.**

In reinforcement learning, a type of machine learning, a computer learns to perform a task through repeated interactions with a dynamic environment. This trial-and-error learning approach enables the computer to make a series of decisions without human intervention and without being explicitly programmed to perform the task. One famous example of reinforcement learning in action is AlphaGo, the first computer program to defeat a world champion at the game of Go.

Reinforcement learning works with data from a dynamic environment—in other words, with data that changes based on external conditions, such as weather or traffic flow. The goal of a reinforcement learning algorithm is to find a strategy that will generate the optimal outcome. The way reinforcement learning achieves this goal is by allowing a piece of software called an *agent* to explore, interact with, and learn from the environment.

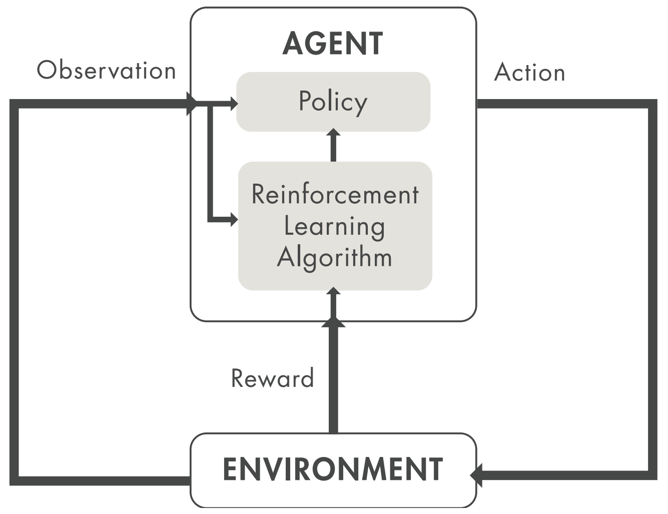


FIGURE 1. *Reinforcement learning overview.*

## An Automated Driving Example

One important aspect of automated driving is self-parking. The goal is for the vehicle computer (*agent*) to position the car in the correct parking spot and with the correct orientation. In this example, the *environment* is everything outside the agent—the dynamics of the vehicle, nearby vehicles, weather conditions, and so on. During training, the agent uses readings from cameras, GPS, lidar, and other sensors to generate steering, braking, and acceleration commands (*actions*). To learn how to generate the correct actions from the observations (*policy tuning*), the agent repeatedly tries to park the vehicle using trial and error. The correct action is rewarded (reinforced) with a numerical signal (Figure 1).

In this example, training is supervised by a *training algorithm*. The training algorithm is responsible for tuning the agent's policy based on the collected sensor readings, actions, and rewards. After training, the vehicle's computer should be able to park using only the tuned policy and the sensor readings.

## Algorithms for Reinforcement Learning

Many reinforcement learning training algorithms have been developed to date. Some of the most popular algorithms rely on deep neural networks. The biggest advantage of neural networks is that they can encode complex behaviors, making it possible to use reinforcement learning in applications that would be very challenging to tackle with traditional algorithms.

For example, in autonomous driving, a neural network can replace the driver and decide how to turn the steering wheel by simultaneously looking at input from multiple sensors, such as camera frames and lidar measurements (Figure 2). Without neural networks, the problem would be broken down into smaller pieces: a module that analyzes the camera input to identify useful features, another module that filters the lidar measurements, possibly one component that would aim to paint the full picture of the vehicle's surroundings by fusing the sensor outputs, a "driver" module, and so on.

*The goal of a reinforcement learning algorithm is to find a strategy that will generate the optimal outcome.*

## Reinforcement Learning Workflow

Training an agent using reinforcement learning involves five steps:

1. Create the environment. Define the environment within which the agent can learn, including the interface between agent and environment. The environment can be either a simulation model or a real physical system. Simulated environments are usually a good first step since they are safer and allow experimentation.

2. Define the reward. Specify the reward signal that the agent uses to measure its performance against the task goals and how this signal is calculated from the environment. Reward shaping may require a few iterations to get right.

3. Create the agent. The agent consists of the policy and the training algorithm, so you need to:

   - Choose a way to represent the policy (for example, using neural networks or lookup tables). Consider how you want to structure the parameters and logic that make up the decision-making part of the agent.

   - Select the appropriate training algorithm. Most modern reinforcement learning algorithms rely on neural networks because they are good candidates for large state/action spaces and complex problems.

4. Train and validate the agent. Set up training options (such as stopping criteria) and train the agent to tune the policy. The easiest way to validate a trained policy is through simulation.

5. Deploy the policy. Deploy the trained policy representation using, for example, generated C/C++ or CUDA code. No need to worry about agents and training algorithms at this point—the policy is a standalone decision-making system.

## An Iterative Process

Training an agent using reinforcement learning involves a fair amount of trial and error. Decisions and results in later stages can require you to return to an earlier stage in the learning workflow. For example, if the training process does not converge to an optimal policy within a reasonable amount of time, you may have to update any of the following before retraining the agent:

- Training settings
- Learning algorithm configuration
- Policy representation
- Reward signal definition
- Action and observation signals
- Environment dynamics

## When Is Reinforcement Learning the Right Approach?

While reinforcement learning is a major advance in machine learning, it is not always the best approach. Here are three issues to bear in mind if you are considering trying it:

- It is not sample-efficient. This means that a lot of training is required to reach acceptable performance. Even for relatively simple applications, training time can take anywhere from minutes to hours or days. AlphaGo was trained by playing millions of games nonstop for several days, accumulating thousands of years' worth of human knowledge.
- Setting up the problem correctly can be tricky; many design decisions need to be made, which may require several iterations to get right. These decisions include selecting the appropriate architecture for the neural network, tuning hyperparameters, and shaping the reward signal.
- A trained deep neural network policy is a "black box," meaning that the internal structure of the network is so complex (often consisting of millions of parameters) that it is almost impossible to understand, explain, and evaluate the decisions taken. This makes it difficult to establish formal performance guarantees with neural network policies.

If you are working on a time- or safety-critical project, you might want to try some alternative method. For example, for control design, using a traditional control method would be a good way to start.

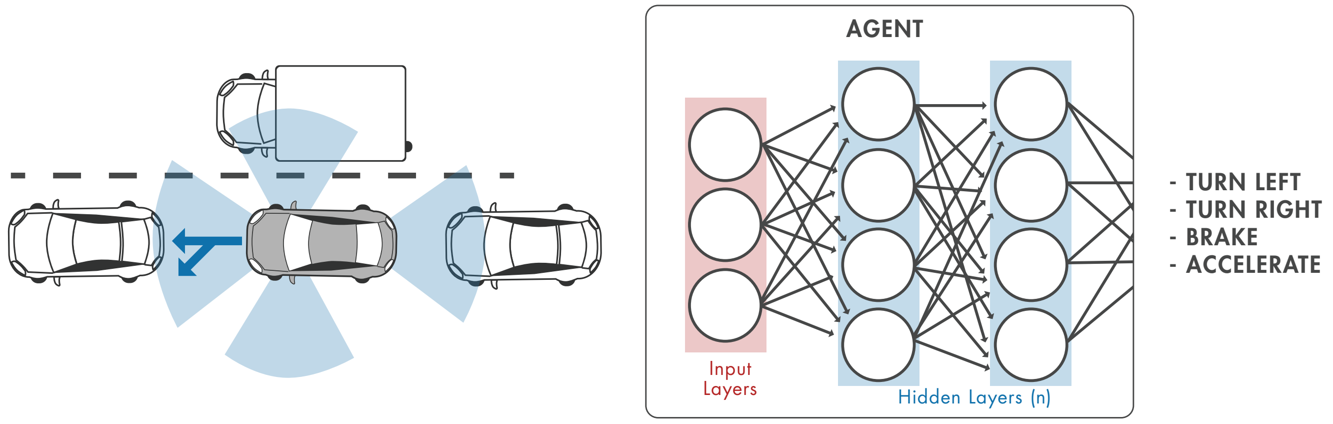
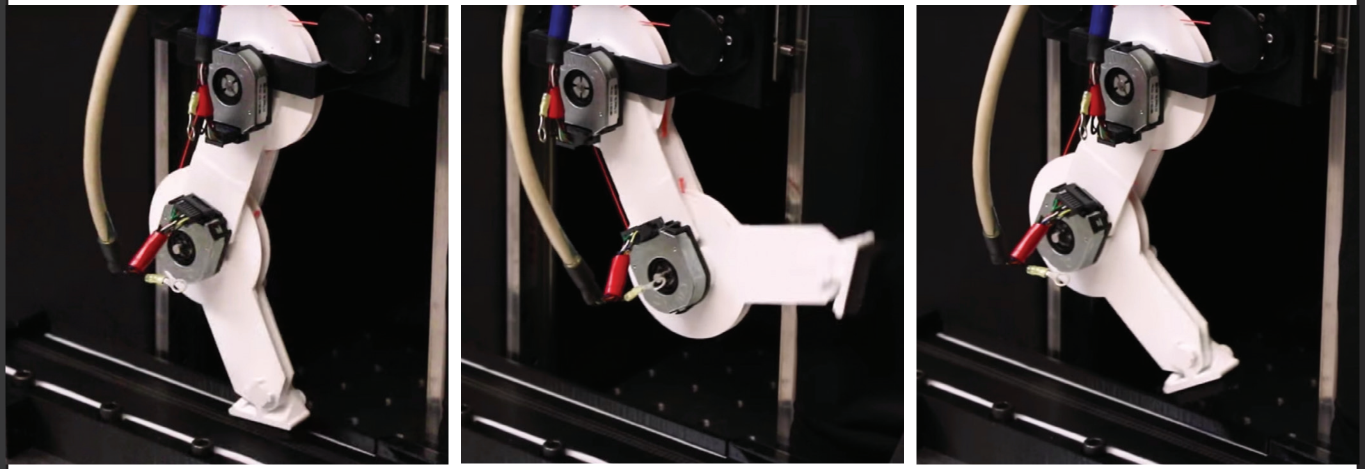


FIGURE 2. *Neural network for autonomous driving.*



## Real-World Example: Robot Teaches Itself to Walk

Researchers from the University of Southern California's Valero Lab built a simple robotic leg that taught itself how to move in just minutes using a reinforcement learning algorithm written in MATLAB® (Figure 3).

The three-tendon, two-joint limb learns autonomously, first by modeling its own dynamic properties and then by using reinforcement learning.

For the physical design, this robotic leg used a tendon architecture, much like the muscle and tendon structure that powers animals' movements. Reinforcement learning then used the understanding of the dynamics to accomplish the goal of walking on a treadmill.

### Reinforcement Learning and "Motor Babbling"

By combining motor babbling with reinforcement learning, the system attempts random motions and learns properties of its dynamics through the results of these motions. For this research, the team began by letting the system play at random, or *motor babble*. The researchers give the system a reward—in this case, moving the treadmill forward—every time it performs a given task correctly.

The resulting algorithm, called G2P (general to particular), replicates the general problem that biological nervous systems face when controlling limbs by learning from the movement that occurs when a tendon moves the limb (see page 36). It is followed by reinforcing (rewarding) the behavior that is particular to the task. In this case, the task is successfully moving the treadmill. The system creates a general understanding of its dynamics through motor babbling and then masters a desired "particular" task by learning from every experience, or G2P.

The neural network, built with MATLAB and Deep Learning Toolbox™, uses the results from the motor babbling to create an inverse map between inputs (movement kinematics) and outputs (motor activations). The network updates the model based on each attempt made during the reinforcement learning phase to home in on the desired results. It remembers the best result each time, and if a new input creates a better result, it overwrites the model with the new settings.

The G2P algorithm can learn a new walking task by itself after only 5 minutes of unstructured play. It can then adapt to other tasks without any additional programming.

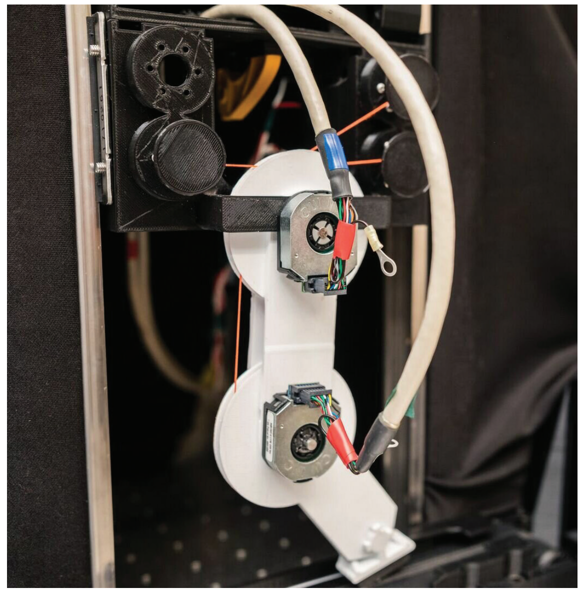


FIGURE 3. *Valero Lab's new robotic limb. Image credit: USC.*

## The G2P Algorithm

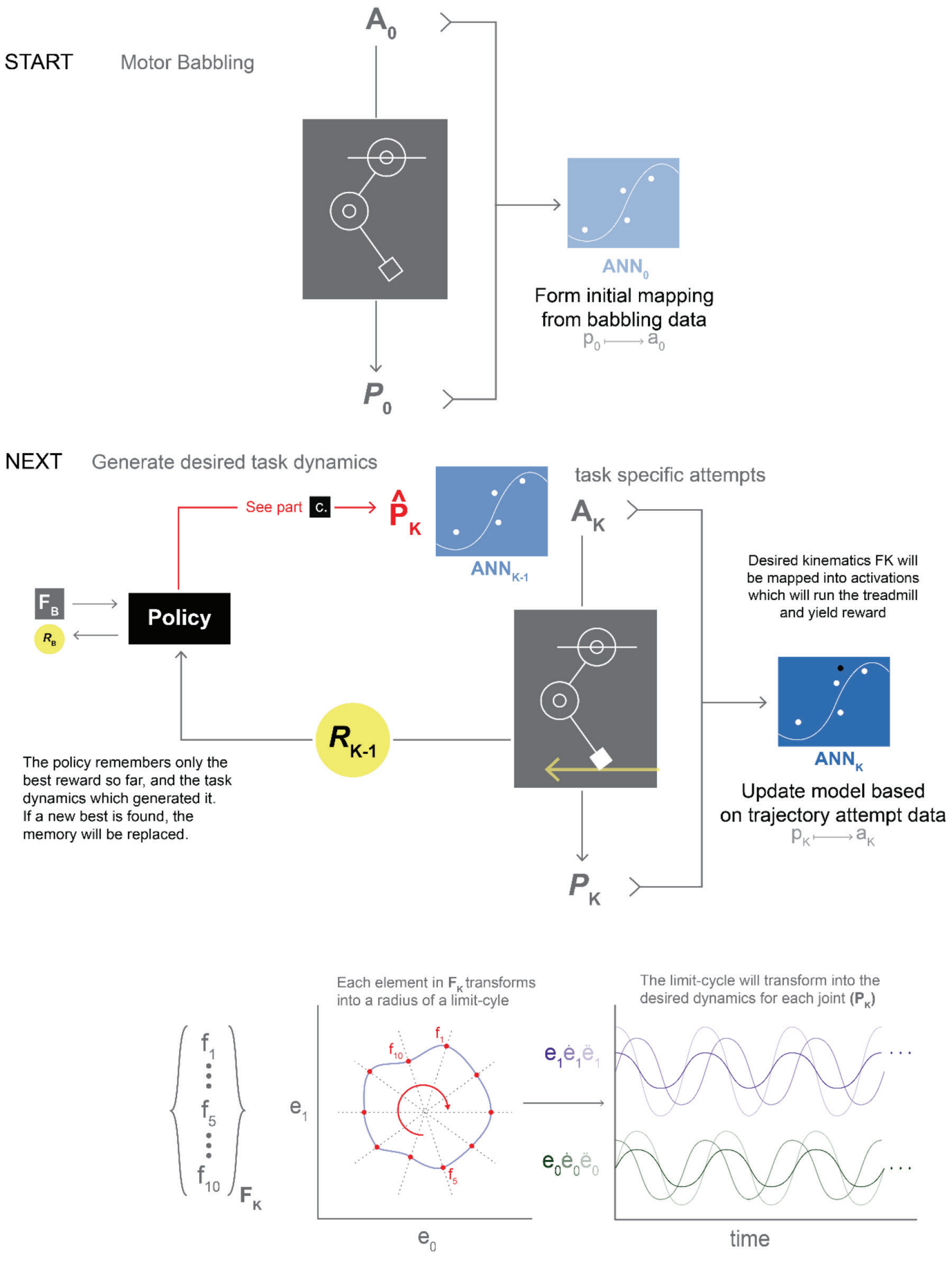

START
Motor Babbling
$A_0$
$P_0$
$ANN_0$
Form initial mapping from babbling data
$p_0 \longmapsto a_0$
NEXT
Generate desired task dynamics
See part C.
$\hat{P}_K$
$ANN_{K-1}$
task specific attempts
$A_K$
Desired kinematics FK will be mapped into activations which will run the treadmill and yield reward
$F_B$
$R_B$
Policy
$R_{K-1}$
The policy remembers only the best reward so far, and the task dynamics which generated it. If a new best is found, the memory will be replaced.
$ANN_K$
Update model based on trajectory attempt data
$p_K \longmapsto a_K$
$P_K$
Each element in $F_K$ transforms into a radius of a limit-cyle
The limit-cycle will transform into the desired dynamics for each joint ($P_K$)
$f_1$
$f_5$
$f_{10}$
$F_K$
$e_1$
$e_0$
$e_1 \dot{e}_1 \ddot{e}_1$
$e_0 \dot{e}_0 \ddot{e}_0$
time

*Image credit: Marjaninejad, et al.*

# Deep Learning Deciphers What Rats Are Saying

University of Washington researchers have developed software that uses deep learning to detect and analyze the ultrasonic vocalizations (USVs) of rats. Inaudible to the human ear, USVs are an invaluable source of insight into the rat's state of mind. By observing lab rats' responses to various stimuli, researchers hope to better understand how drugs change brain activity so as to devise more effective treatments for anxiety disorders, depression, and addiction in humans.

Rat calls are at such a high frequency that even with specialized microphones, recordings must be slowed down to make them audible. Manually tagging and categorizing the slowed-down USVs is labor-intensive and error-prone.

DeepSqueak software automates this laborious process by turning an audio problem into a visual problem. It translates raw audio files into filtered sonograms that are fed into a convolutional neural network (CNN) as labeled samples of vocalizations and noise. Biomimetic algorithms in DeepSqueak learn to classify the vocalizations and detect patterns, such as the order of squeaks and their duration.

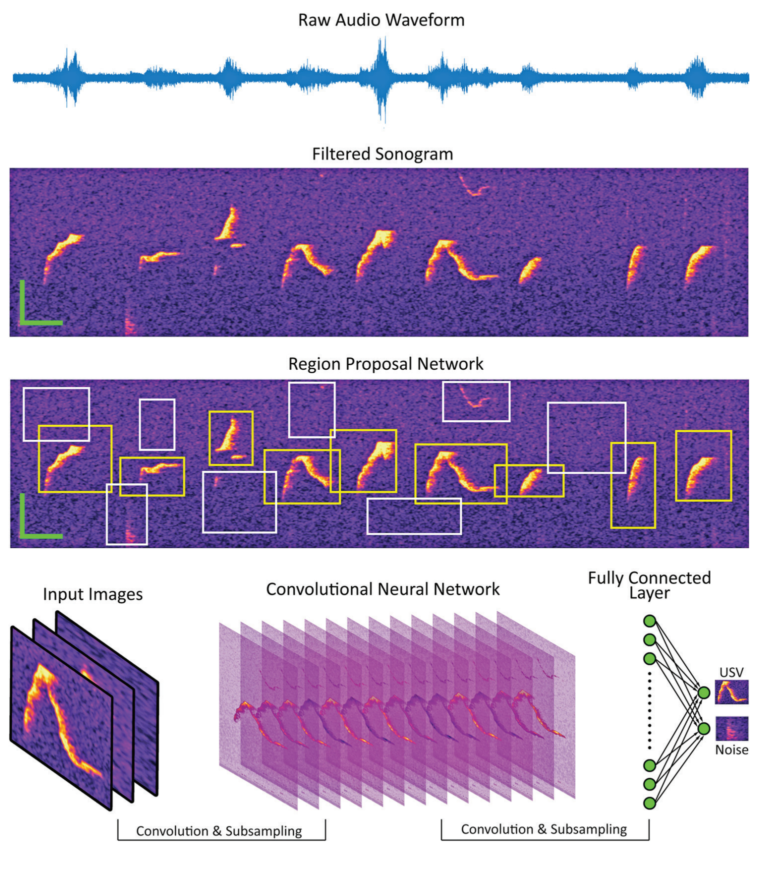


*Image courtesy Kevin R. Coffey, Russell G. Marx, and John F. Neumaier.*

## Power Electronics Control Design

# Start in Simulink

## GO ANYWHERE

Design power electronics controls, validate with simulation, and generate code for your target hardware:

- Model with hundreds of ready-to-use electrical modeling components
- Perform rapid prototyping and HIL simulation with multiple real-time hardware platforms
- **Get to code 50% faster than hand coding** by automatically generating readable, compact C and HDL code for any microcontroller, FPGA, or SoC

Get started in Simulink®

***mathworks.com/pec***

MathWorks®

93187v00 10/19