

The background features a dark blue field on the left and a grey field on the right, separated by a diagonal line. In the upper right, there are white, stylized waveforms. In the lower right, there is a 3D wireframe mesh with a color gradient from yellow to blue, and a faint blue circuit board pattern.

MATLAB EXPO 2017

Developing Autonomous Systems
with MATLAB and Simulink

Vivek Raju

Challenges in Developing Autonomous System

- **Challenge 1:** Understand the dynamics of the autonomous system and design control algorithm
- **Challenge 2:** Design vision, radar and perception algorithms
- **Challenge 3:** Verify and Implement the algorithm on to a real hardware

What are we doing today?

The image displays a ROS simulation environment with three main components:

- Top Left: Video Display** - A window titled "To Video Display" showing a 3D view of a stop sign in a simulated environment.
- Top Right: Gazebo** - The main 3D simulation window. It shows a robot on a grid floor with a stop sign, a barrier, and a house in the background. The left sidebar lists the scene hierarchy (World, GUI, Scene, Physics, Models) and the current selected object's properties (name: ground_plane, is_static: True, self_collide: False).
- Bottom Left: Simulink** - A Simulink workspace window titled "ImprocNode - Simulink". It shows a block diagram with the following layers:
 - Sensor Communication Layer:** Includes ROS Subscribe blocks for image data and a Set Pace block.
 - Application Layer:** Contains an ImageProc block, a stopSignData block, and a ConvertROSMag block.
 - Control UI:** Features buttons for "Add StopSign" and "Add Videoposts".
 - Command Communication Layer:** Includes ROS Publish blocks for Mag and StopSignBox.
 - Status Monitor:** Displays various sensor data including heading, altitude, and artificial horizon.

The bottom status bar shows simulation metrics: Real Time Factor: 0.63, Sim Time: 00:00:11.44945, Real Time: 00:00:01.12.076, Iterations: 32501, FPS: 29.9296.

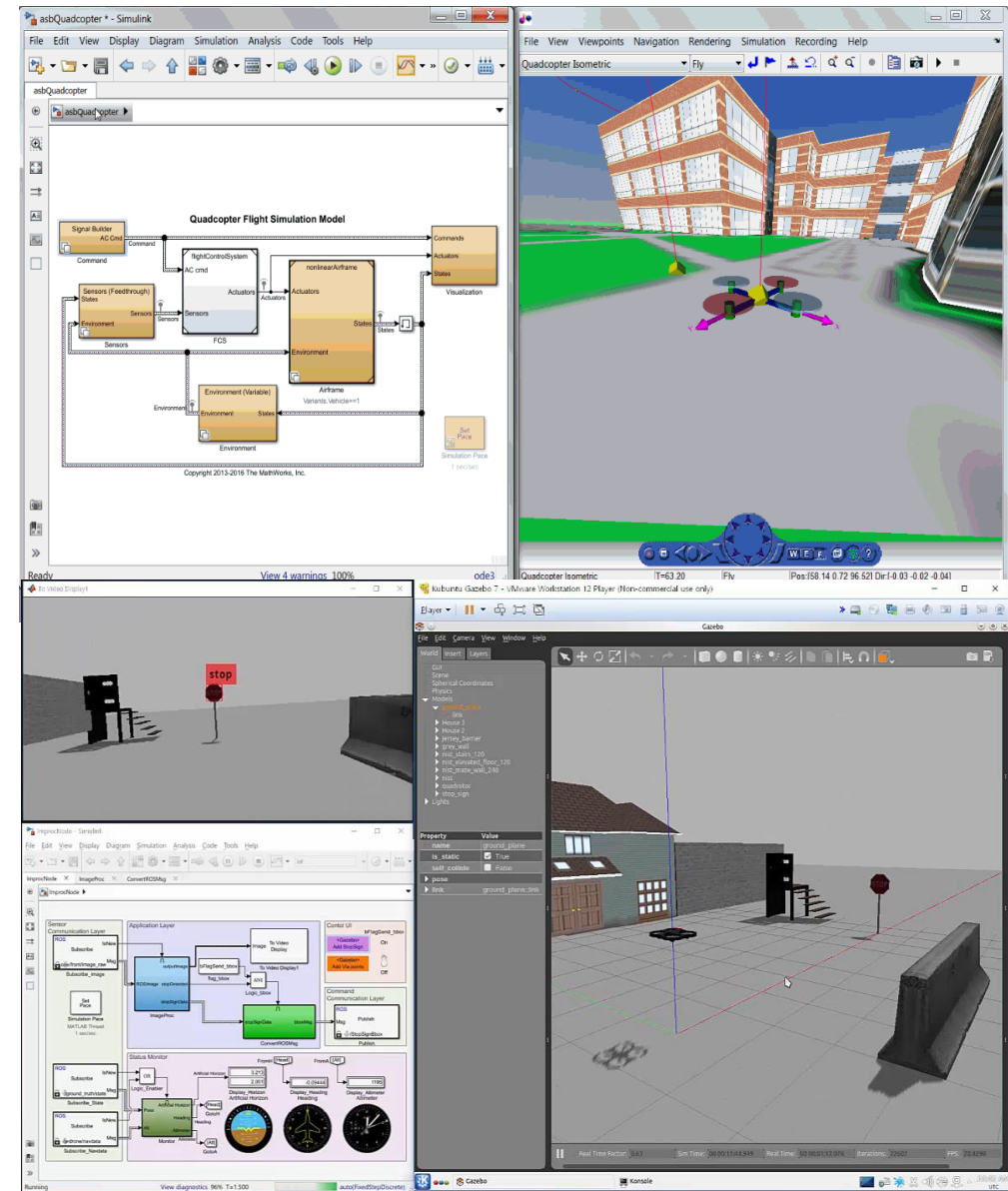
Key Takeaway

Autonomous system design using MATLAB and Simulink can help in :

- **Understanding the dynamics and develop the control algorithm**
 - Model aerodynamics, propulsion and motion
 - Design control algorithm in single environment

- **Design vision, radar, perception algorithms**
 - Visualizing different sensor data
 - Develop and test sensor fusion and tracking algorithm

- **Implementing the algorithm on actual hardware**
 - Test and verify algorithm on 3D simulators
 - Automatic C/C++ code generation on to actual hardware



Aerial Autonomous System Development Workflow

Aerodynamics and flight Control

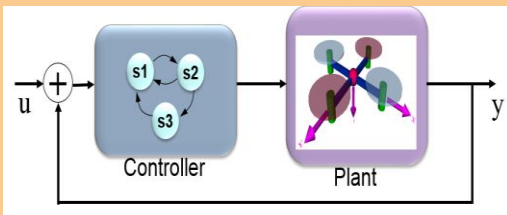
Develop perception and planning algorithm

Test and Refine in Simulation

Test and Refine on Real Robot

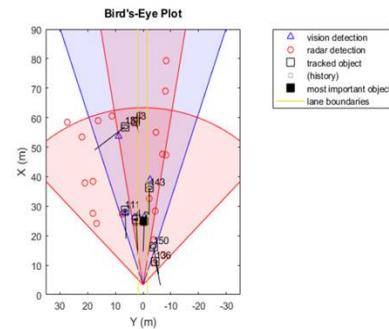
Challenge 1:

Understand the dynamics and design control algorithm



Challenge 2:

Design vision, radar and perception algorithms



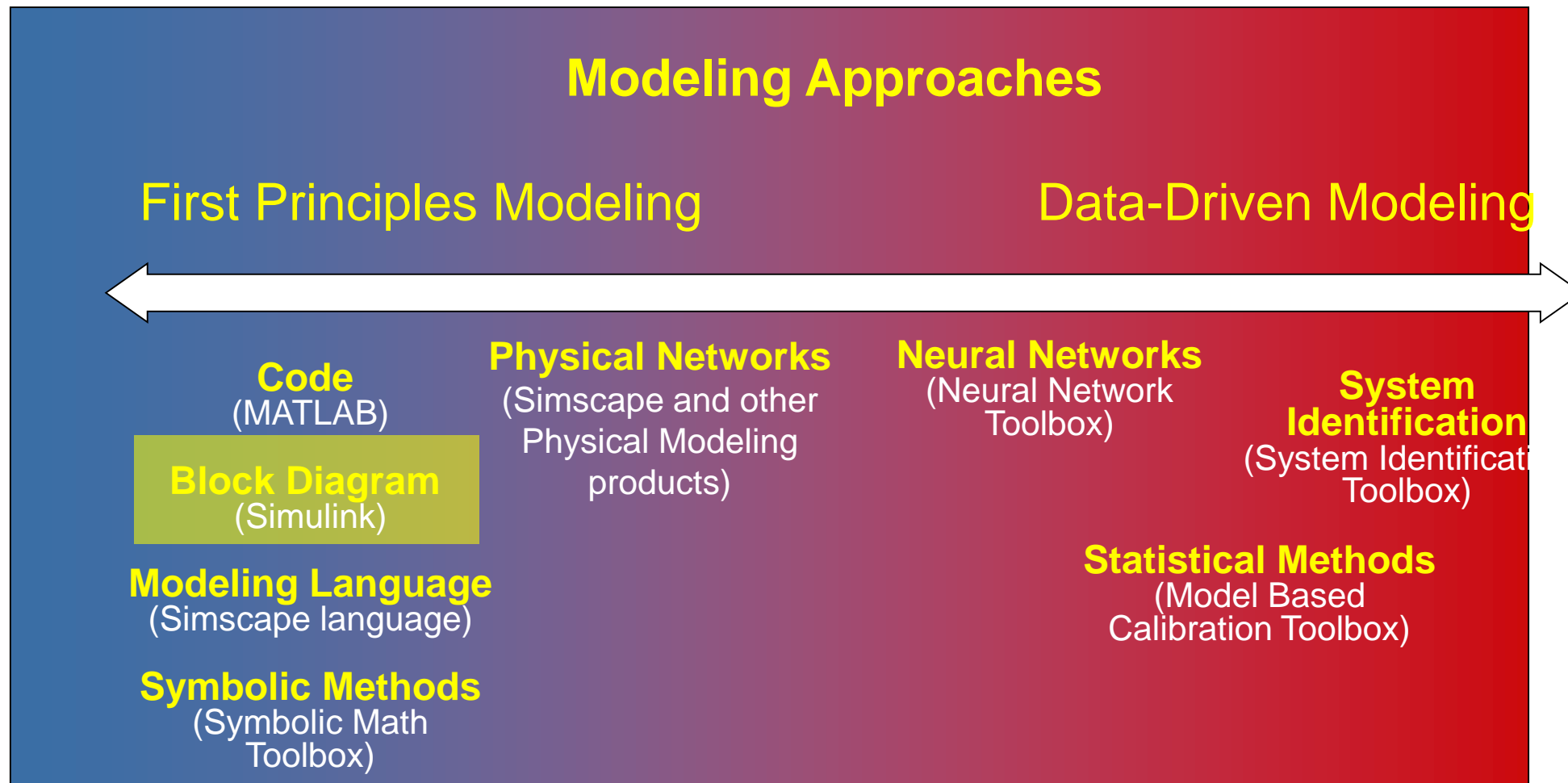
Birds Eye View

Challenge 3:

Verify and Implement the algorithm on to a real hardware



Different Approaches for Modeling Dynamic systems



Aerodynamics and control design

The image displays two software windows side-by-side. The left window is FlightGear, showing a 3D simulation of a white aircraft flying over a landscape. The right window is MATLAB/Simulink, showing a control system model for the aircraft.

The Simulink model, titled "asbh20", includes the following blocks and connections:

- AC (Aircraft Control) block:** Receives a 75-bit input signal and outputs a 24-bit "cmd" signal to the vehicle model.
- Pilot block:** Receives a 75-bit input signal and outputs a 75-bit signal to the vehicle model.
- Env RFEnv (RF Signals) block:** Receives a 75-bit input signal and outputs a 14-bit signal to the vehicle model.
- Env FG (Environment) block:** Receives a 75-bit input signal and outputs an 8-bit signal to the vehicle model.
- Vehicle Model:** Receives the 24-bit "cmd" signal, the 75-bit signal from the Pilot block, and the 14-bit signals from the RF and Environment blocks. It outputs a 75-bit signal back to the Pilot block and a 14-bit signal to the Environment block.

The vehicle model is labeled "HL-20 Vehicle Systems Model" and shows a 3D rendering of the aircraft's nose and cockpit area.

Supervisory control logic

The image displays two windows from a Simulink environment. The left window, titled "Stateflow (chart) sf_launchabort/launchAbortController - Simulink", shows a Stateflow chart for the launch abort controller. The chart is divided into two main sections: "ModeLogic" and "Abort".

ModeLogic Section:

- RTLS (entry.mode=1):** Transitions to **DRL (entry.mode=2)** when $[alt > 10000 \ \&\& \ !anomaly]$.
- DRL (entry.mode=2):** Transitions to **AOA (entry.mode=3)** when $[alt > 100000 \ \&\& \ !anomaly]$.
- AOA (entry.mode=3):** Transitions to **ATO (entry.mode=4)** when $[alt > 400000 \ \&\& \ !anomaly]$.
- ATO (entry.mode=4):** Transitions back to **RTLS (entry.mode=1)**.

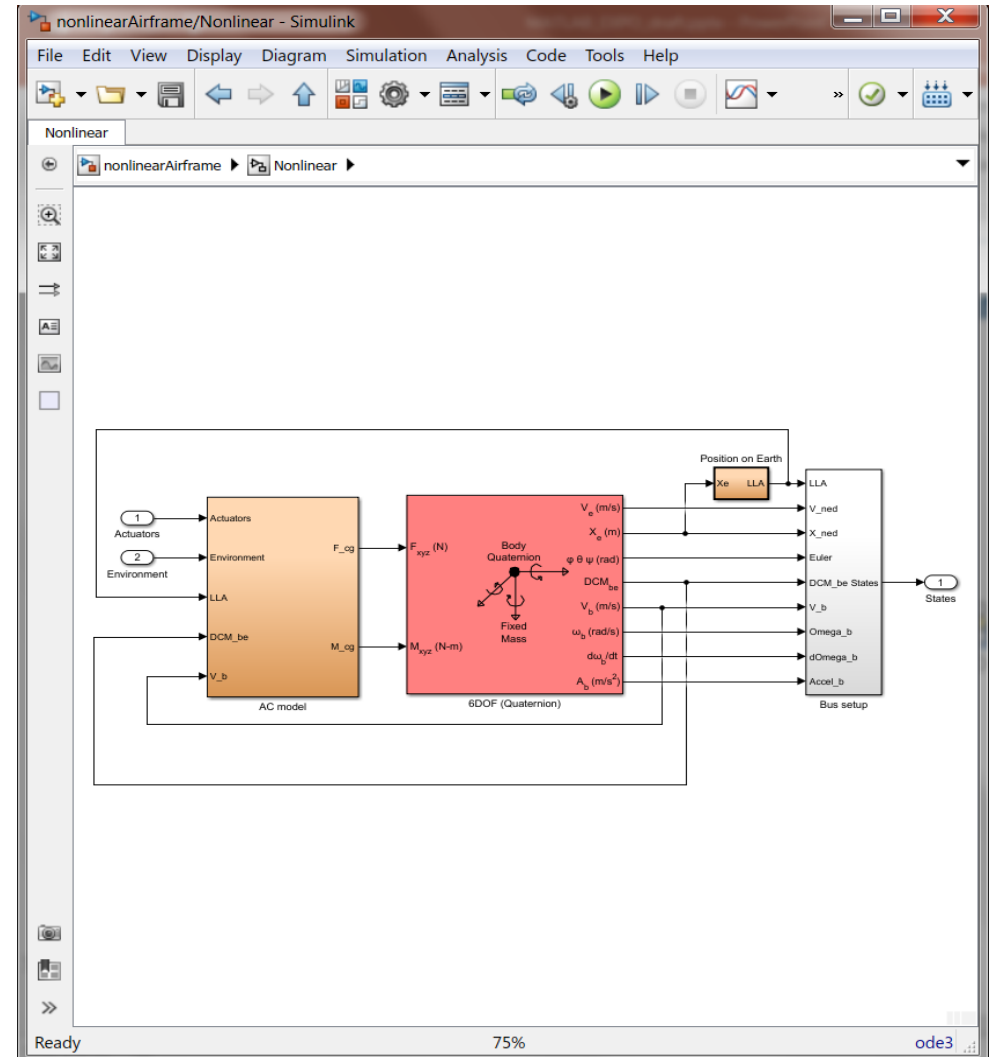
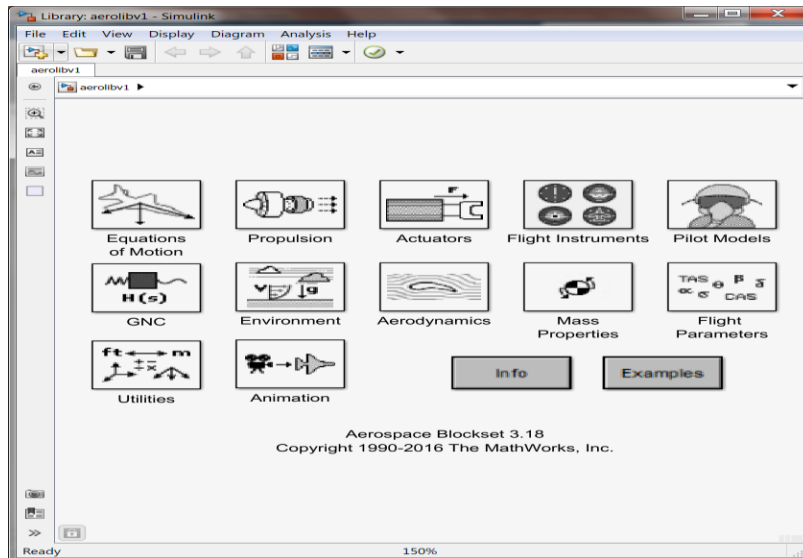
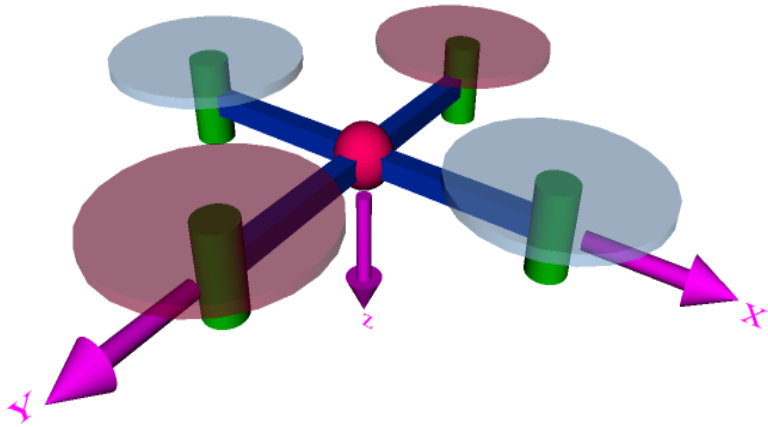
Abort Section:

- Normal (en: abort=NO_ABORT):** Transitions to **abortLogic** when $[anomaly]$.
- abortLogic:** Transitions to **abortComplete (en: abort=ABORT_COMPLETE)**.

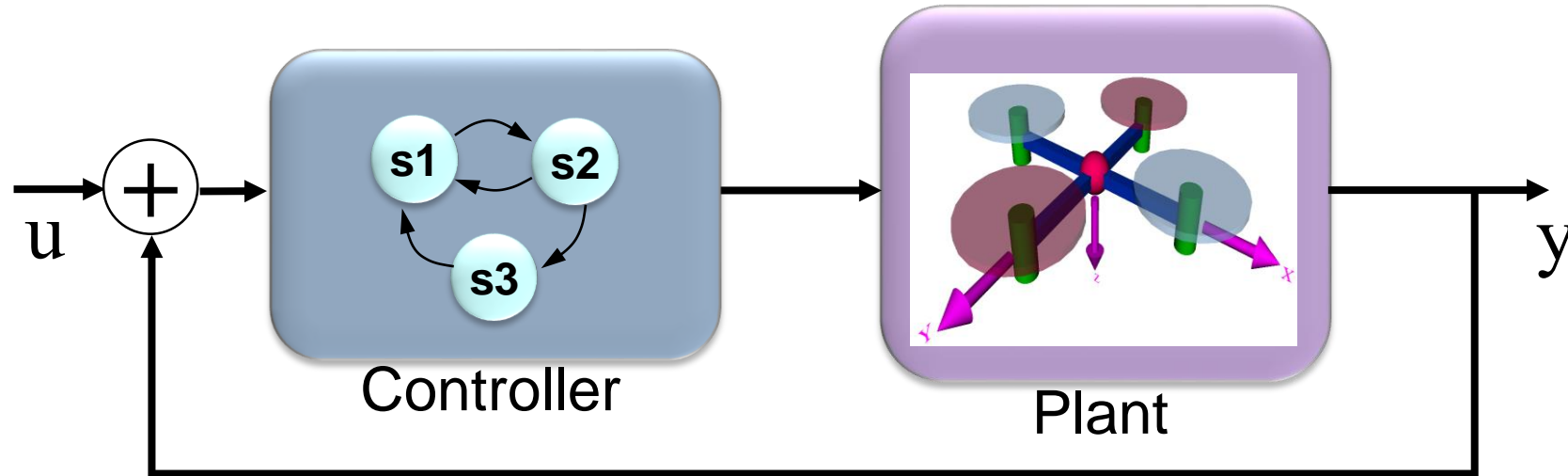
The right window, titled "Launch Abort - VR", shows a 3D visualization of the launch abort system. It features a yellow and black launch abort vehicle (LAV) positioned against a blue sky with white clouds. The interface includes a "Chase Plane" dropdown menu, a "Viewpoints" menu, and a "Navigation" toolbar. The status bar at the bottom of the VR window displays "Chase Plane", "T=69.30", "Examine", and "Pos:[183.26 502.14 -56.50] Dir:[-0.09 0.00 0.04]".

At the bottom of the Simulink window, the status bar shows "Update Stateflow/MATLAB Function block simula: 79%", a green "Compiling" indicator, and "FixedStepDiscrete". The Windows taskbar at the very bottom shows the system clock as 9:51 AM on 12-Apr-17.

Quickly model the Airframe using Aerospace blocksets

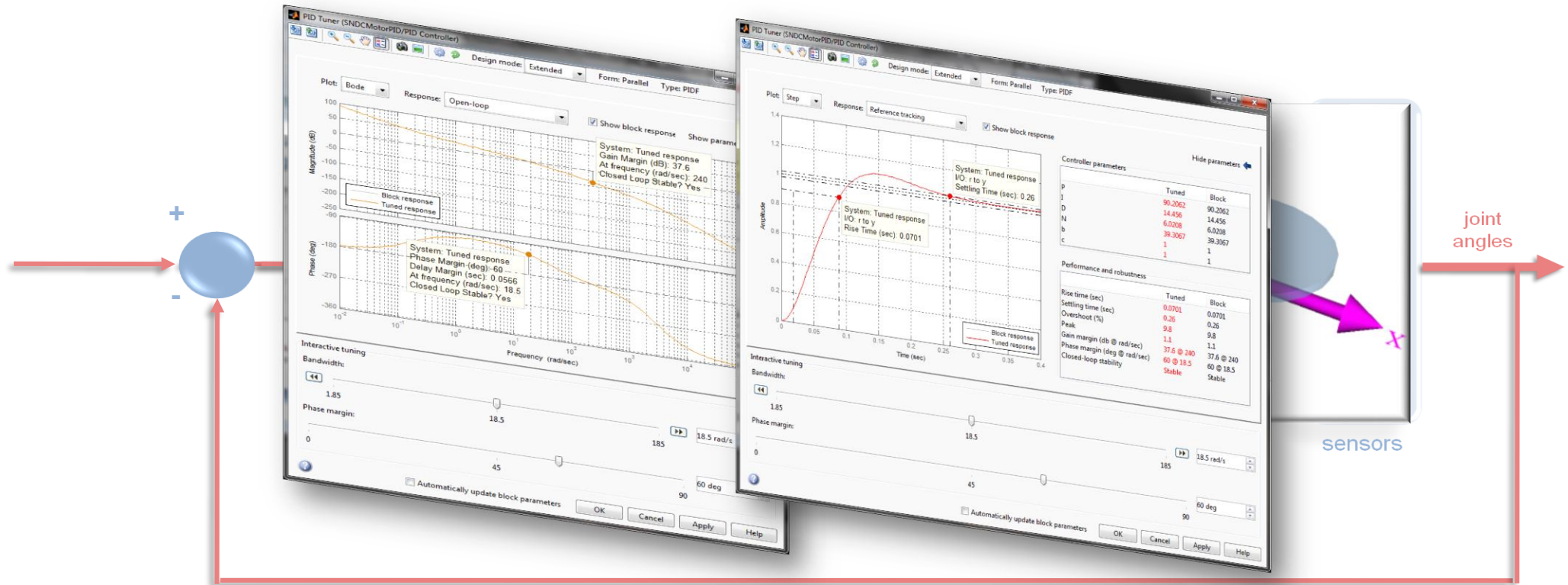


Design the flight control algorithm



- Simulating plant and controller **in one environment** allows you to **optimize system-level performance**

Automatic PID tuning



Use **Simulink Control Design** and the **Control System Toolbox** to automatically linearize the plant, design and tune your PID controllers

Quadcopter –Flight Simulation Model

The image displays a Simulink model and its corresponding 3D visualization of a quadcopter flight simulation.

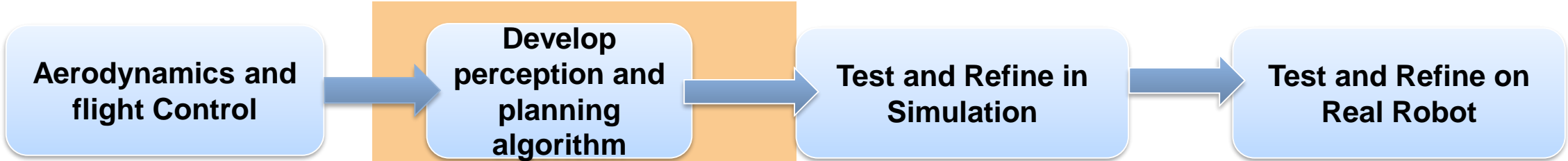
Simulink Model (Left Panel):

- Model Name:** asbQuadcopter
- Block Diagram:**
 - Signal Builder AC Cmd:** Provides the initial command.
 - FlightControlSystem (FCS):** Receives the command and sensor data, outputs actuators.
 - nonlinearAirframe:** Receives actuators and environmental data, outputs states.
 - Sensors (Feedthrough):** Provides sensor data to the FCS.
 - Environment:** Provides environmental data to both the FCS and the Airframe.
 - Environment (Variable):** Provides a variable environment to the Airframe.
 - Visualization:** Receives states and actuators for rendering.
- Simulation Pace:** Set to 1 second.
- Copyright:** 2013-2016 The MathWorks, Inc.

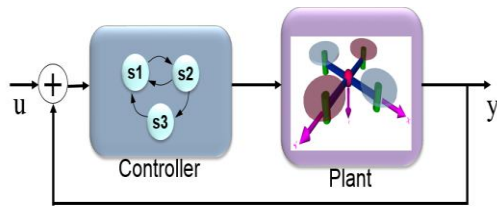
3D Visualization (Right Panel):

- View:** Quadcopter Isometric
- Mode:** Fly
- Environment:** A 3D scene with a green ground plane and a brick building.
- Quadcopter:** A yellow quadcopter with four rotors, positioned on the ground.
- Simulation Pace:** 1 second.
- Status Bar:**
 - Time: T=63.20
 - Mode: Fly
 - Position: Pos:[58.14 0.72 96.52]
 - Direction: Dir:[-0.03 -0.02 -0.04]

Aerial Autonomous System Development Workflow

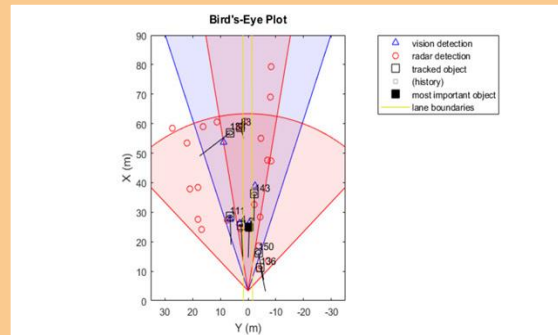


Challenge 1:
Understand the dynamics and design control algorithm



MATLAB EXPO 2017

Challenge 2:
Design vision, radar and perception algorithms



Birds Eye View

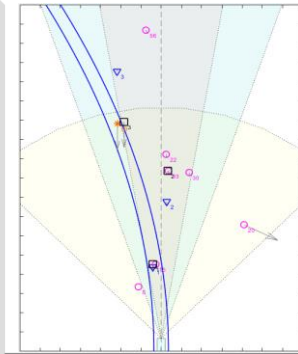
Challenge 3:
Verify and Implement the algorithm on to a real hardware



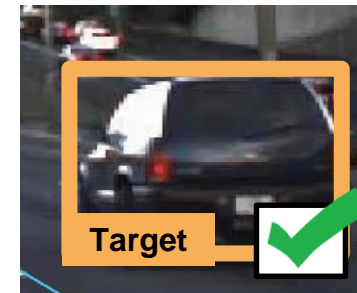
Common Questions from Engineers Integrating Autonomous Systems



How can I
*visualize my
sensor data?*



How can I develop &
*test sensor fusion and
tracking algorithms?*



How can I
*verify performance
and establish
requirements?*

Test vehicle equipped with sensors

Camera

Radar

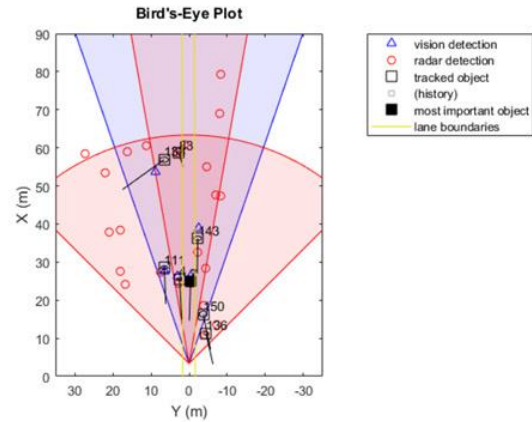


Lidar

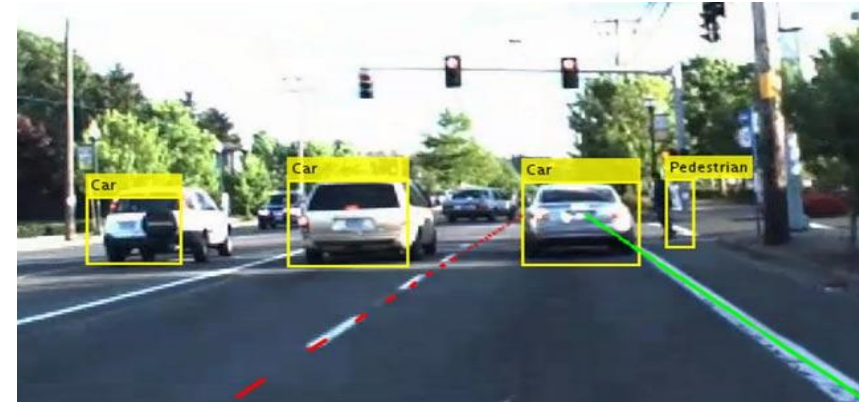
EO/IR

Inertial
measurement
unit

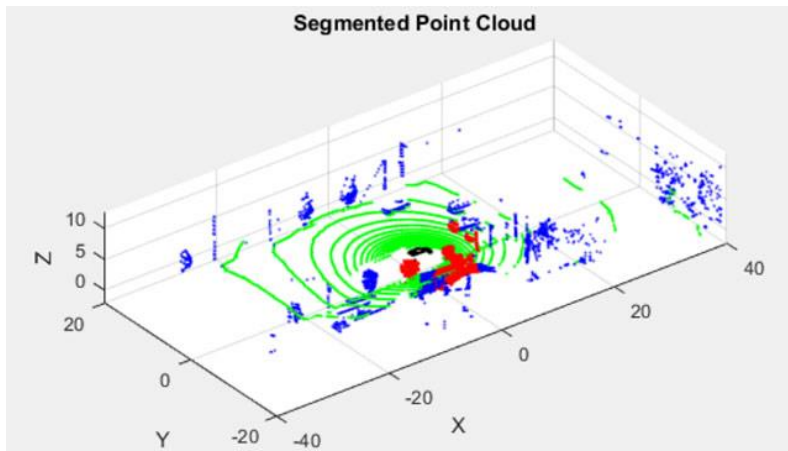
Tools to Visualize Multiple Types of Data



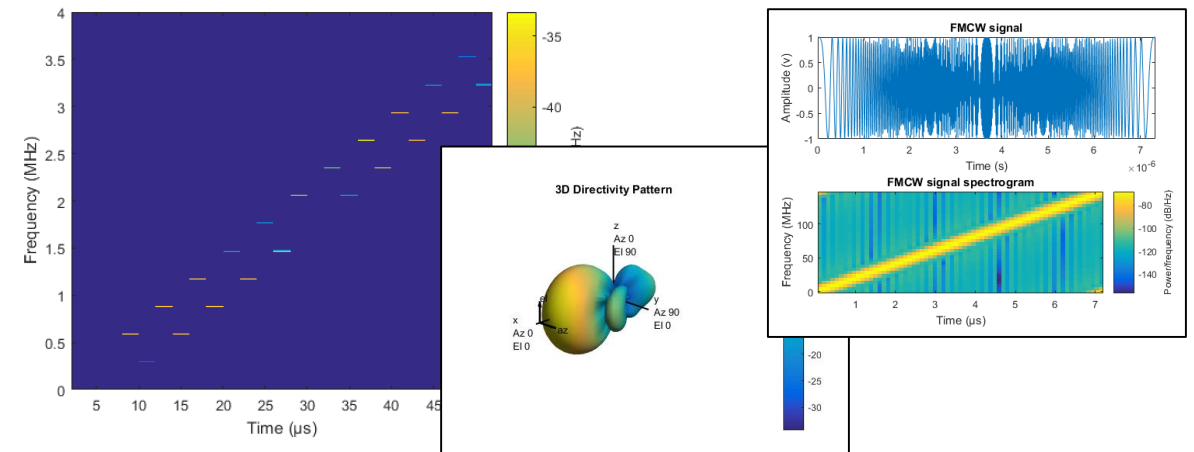
Birds Eye View



Video with Annotations



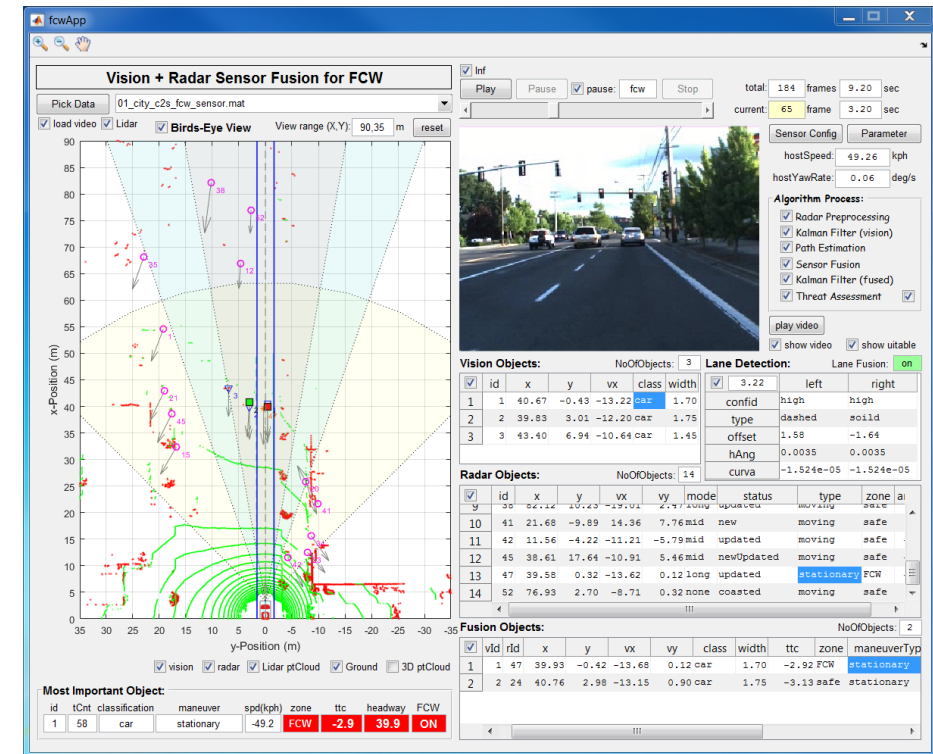
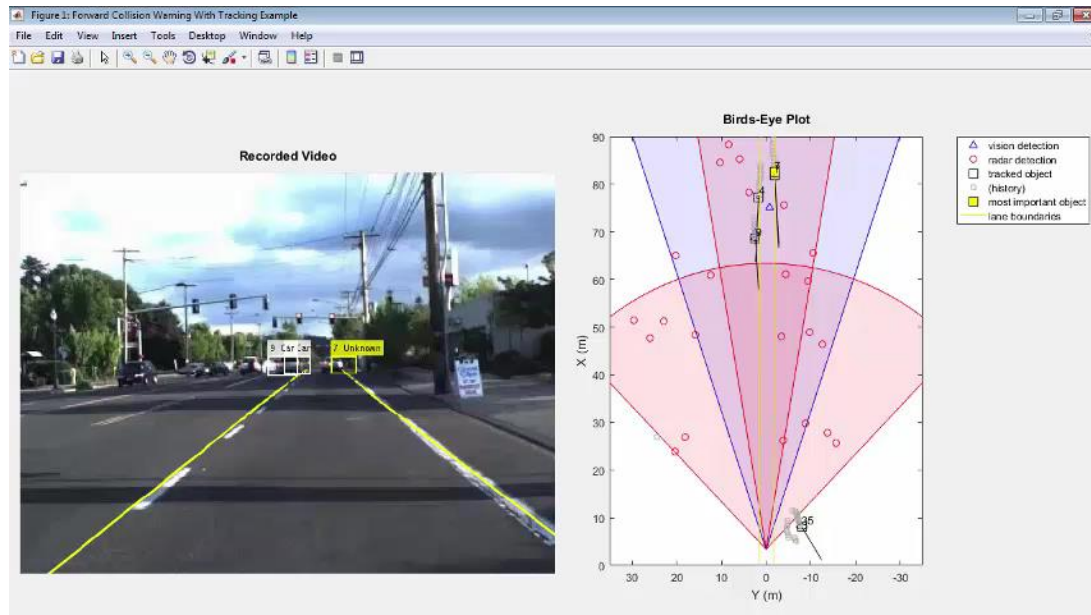
LiDAR & Point Clouds



RADAR

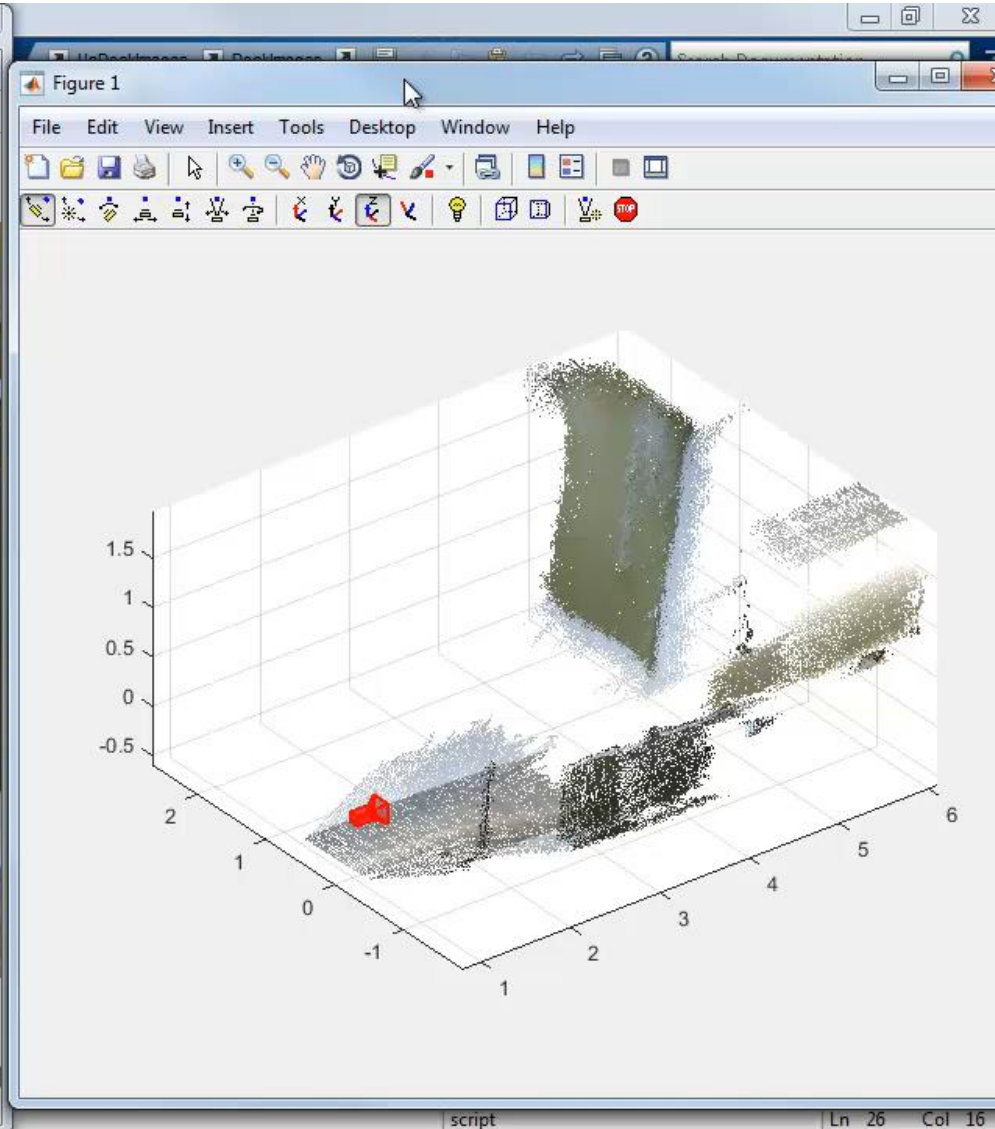
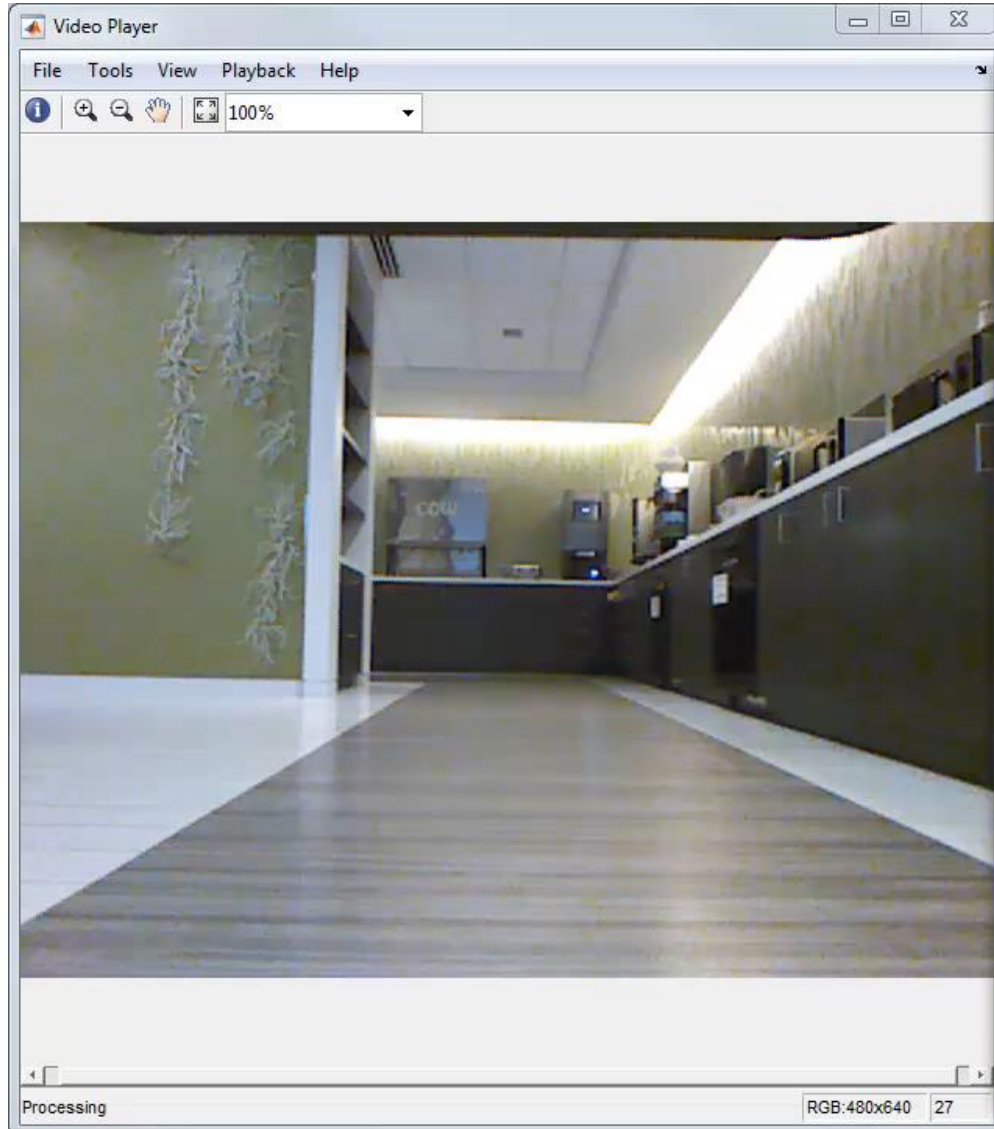
Custom Visualization & Apps

- Open and extensible framework
- Synchronize data from multiple sensor sources



Visual Odometry?

Manage Sensor data



Detailed Session on Image Processing and Computer Vision

Simplifying Image Processing and Computer Vision Application Development


16:45–17:30

Image processing and computer vision is an enabling technology that is driving the development of several of the smart systems today including self-driving cars, augmented reality, hyperspectral imaging, and medical imaging. Developers of modern image processing and computer vision applications face many challenges regarding handling large data sets and working with new computing paradigms, such as GPU computing. You can use MATLAB® to simplify your image processing and computer vision application development workflow.

Join this session to gain insight into:

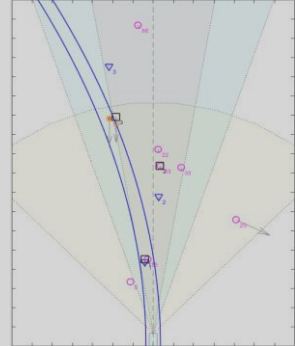
- Object detection and recognition using machine learning and deep learning
- Image processing on 3D data sets, including pixel operations, local filtering, and morphology

Common Questions from Engineers Integrating Autonomous Systems

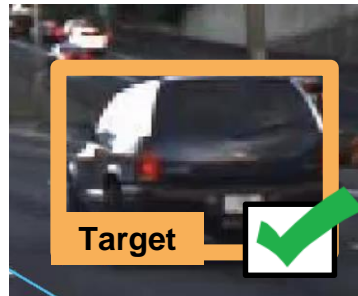


1011010101010100101001
 0101010001000000010101
 001101000101010010010
 01101010010101010101
 01000100000000000000
 00100100000000000000
 01001010101010101010
 01001010101010101010
 1010011101010100101010

How can I visualize my sensor data?

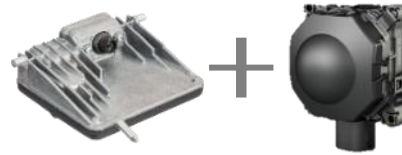


How can I develop & test sensor fusion and tracking algorithms?

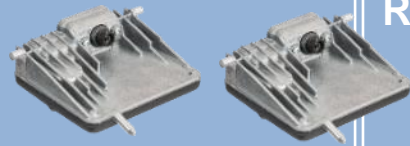



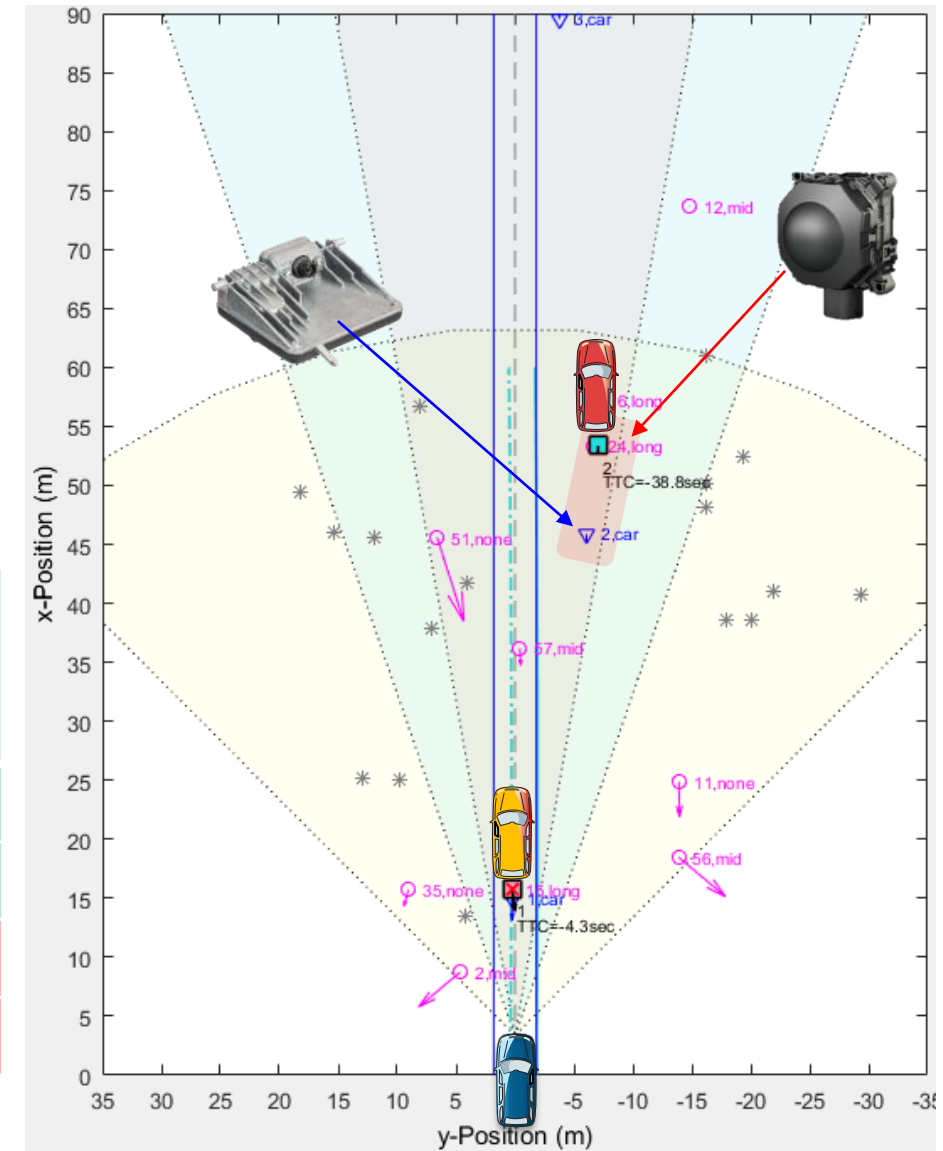
How can I verify performance and establish requirements?

Why Sensor Fusion?



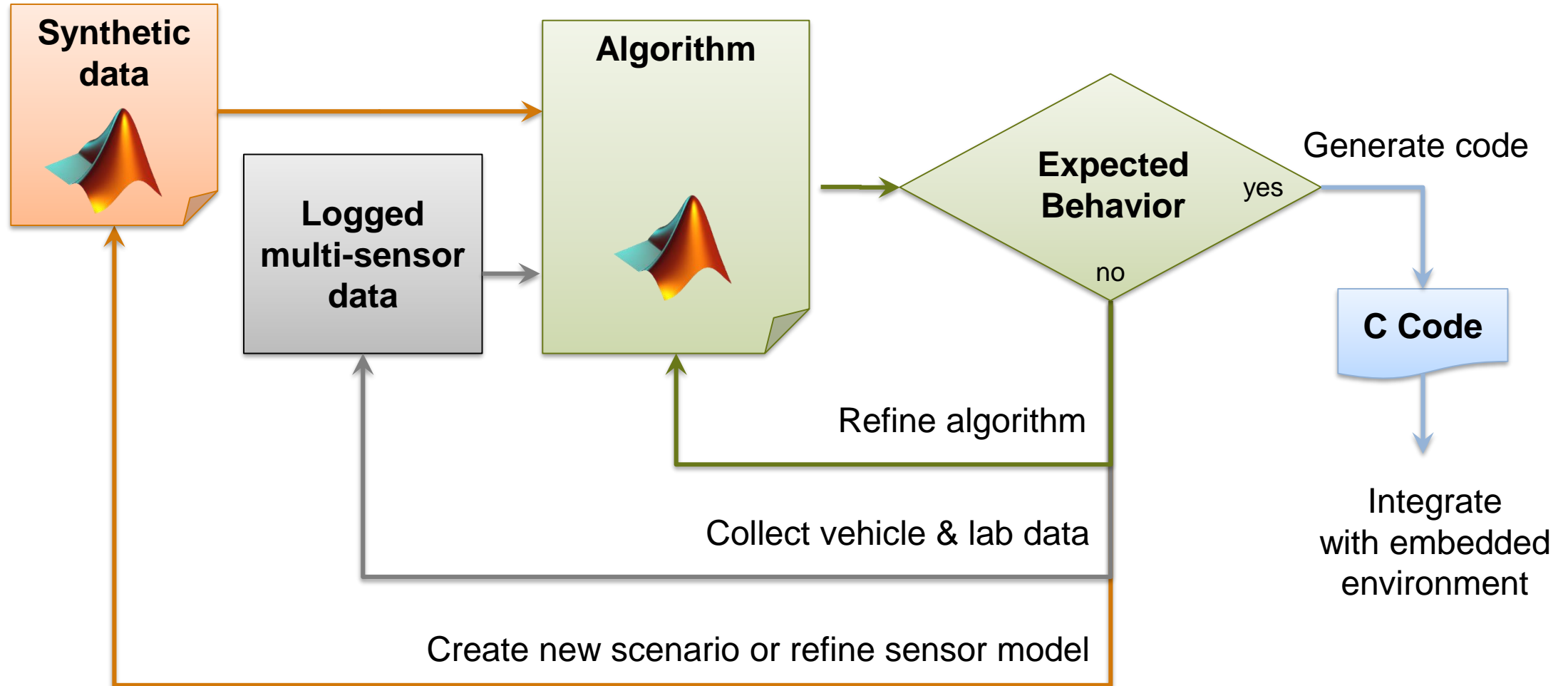
- Requires **robust detection** (low false positives) for obstacle avoidance
 - Needs **classification** of likely objects
 - Needs **accurate measurements** (range & speed)

	Camera 	Radar 
Range	Poor	Accurate
Speed	Estimated	Measureable directly
Angle Resolution	Relatively Good	Poor
Object Classification	Available	Estimated

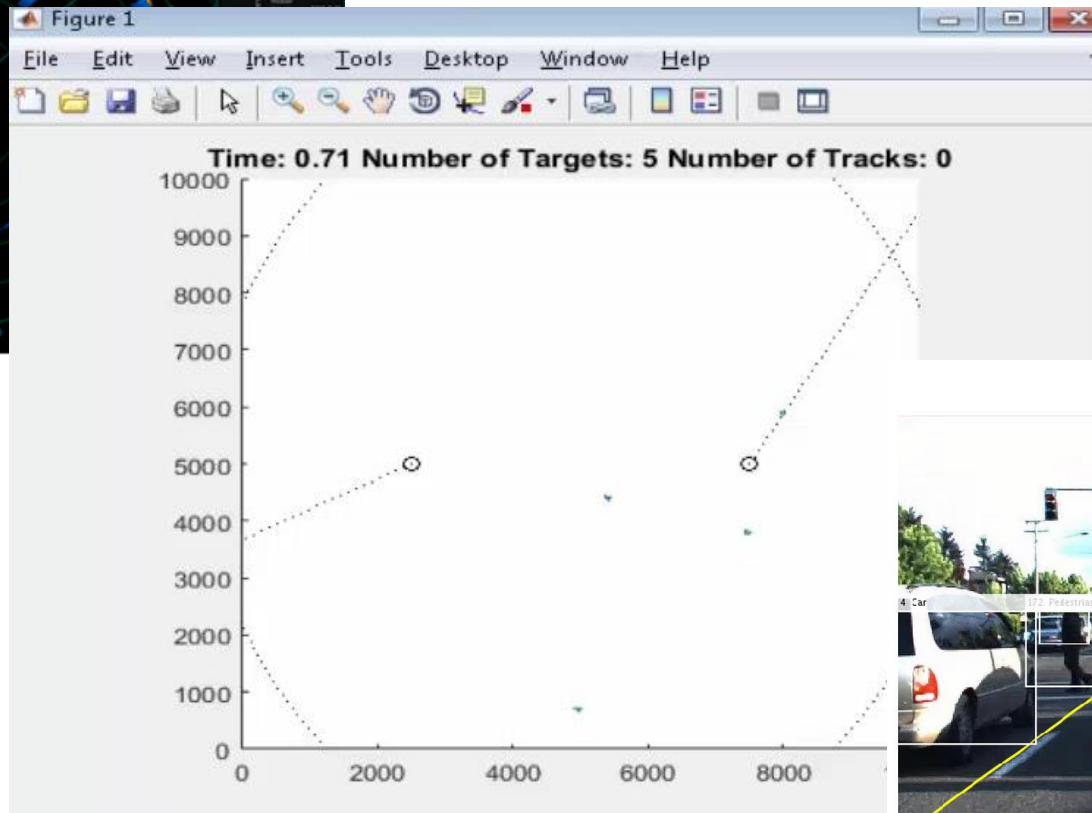
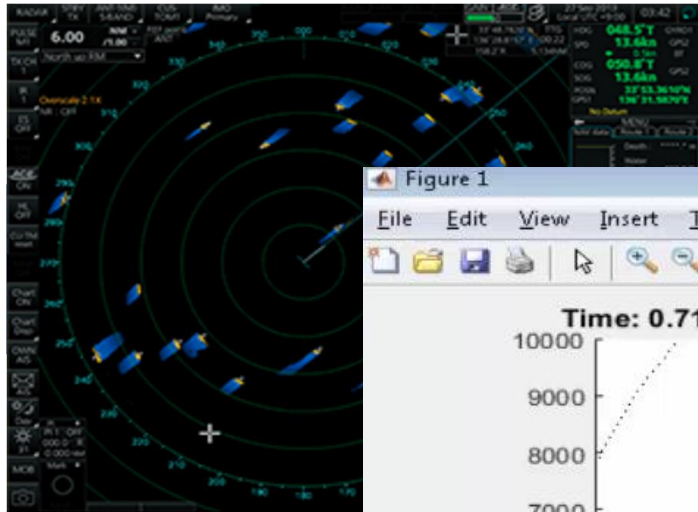


Example: Automated Ground Vehicle

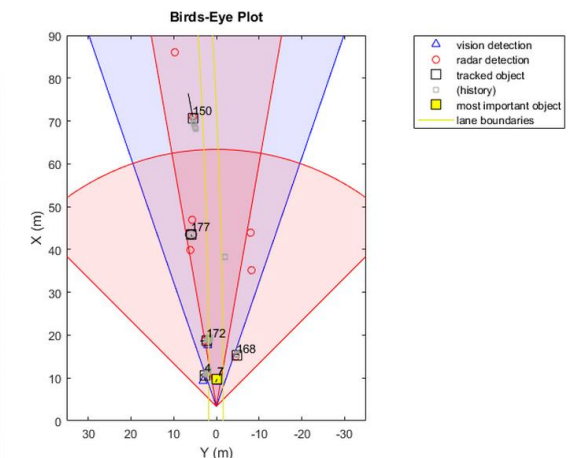
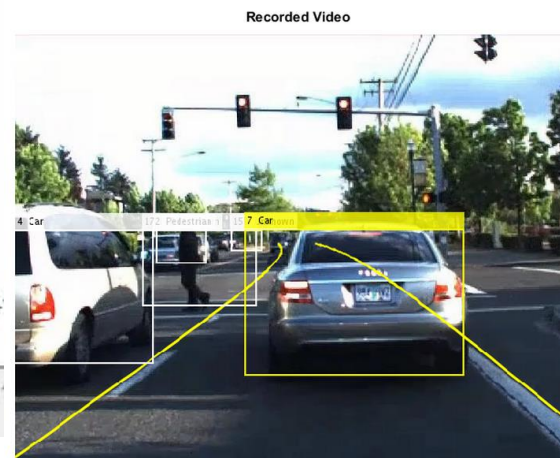
Fusion & Tracking Algorithm Development Workflow



Tracking of multiple objects (or “targets”) with one or multiple sensors (e.g. radar, EO/IR, Lidar)



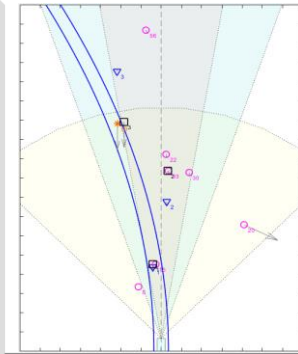
- Multi-object tracking
- Global nearest-neighbor assignment
- Kalman filtering
- Motion and measurement models
- Detection reporting



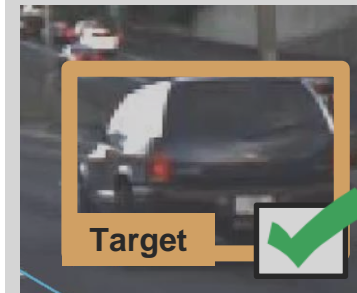
Common Questions from Engineers Integrating Autonomous Systems



How can I
*visualize my
sensor data?*

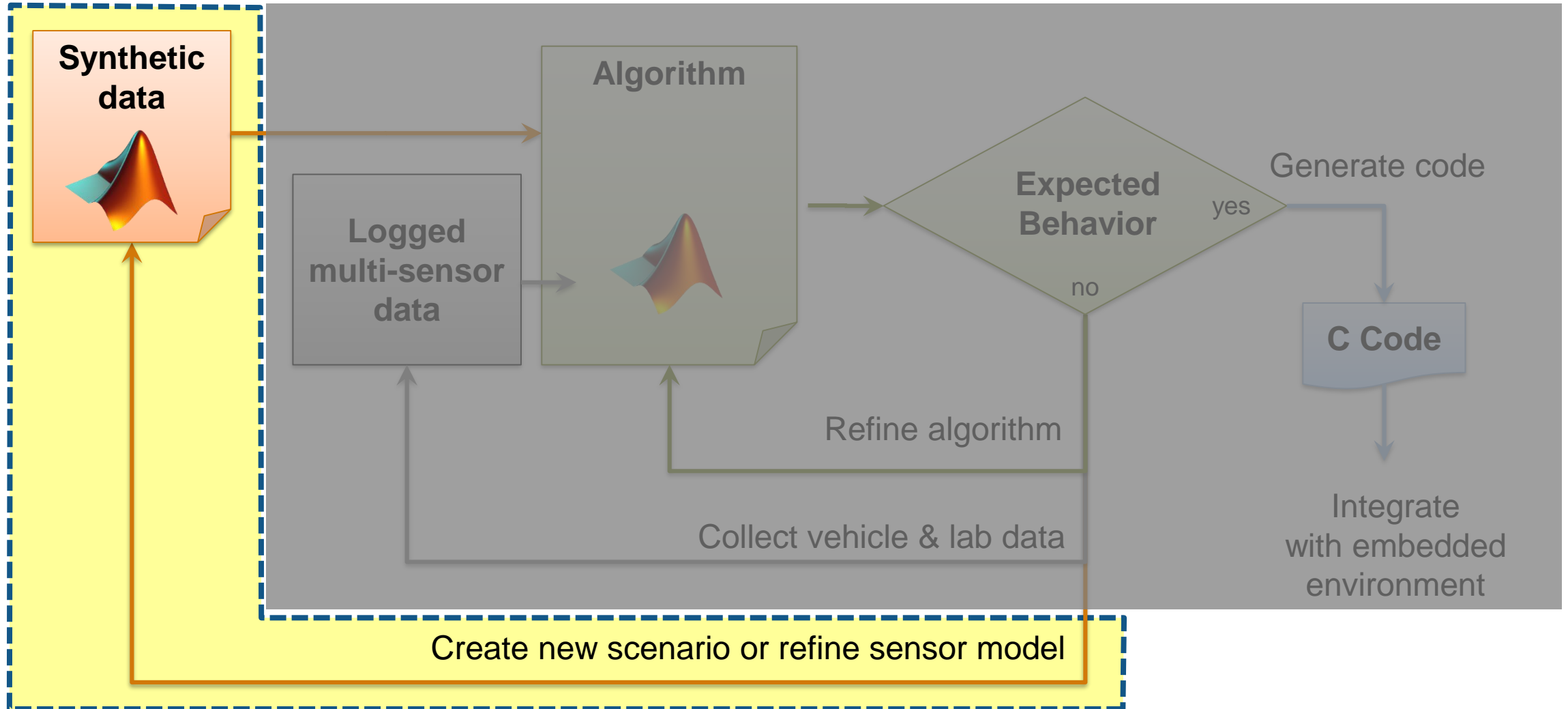


How can I develop &
*test sensor fusion and
tracking algorithms?*



How can I
*verify performance
and establish
requirements?*

Fusion & Tracking Algorithm Development Workflow



Common Approaches to Synthesizing Test Data

“Buy It”

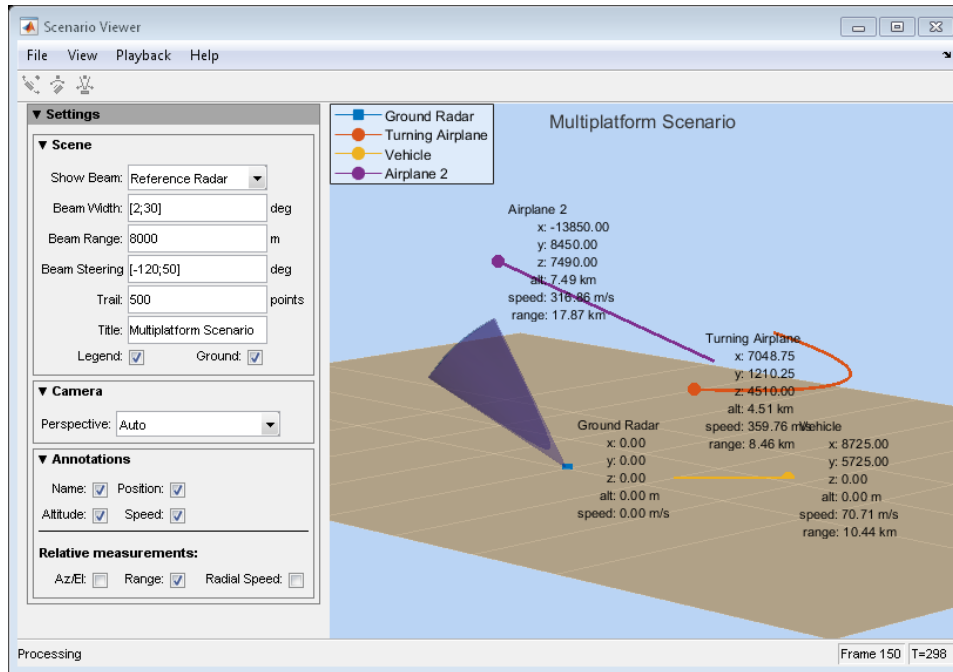
- Buying “Off the shelf” solutions like Unreal, Unity, or other simulators enable you to author scenarios and synthesize sensor data

“Build It”

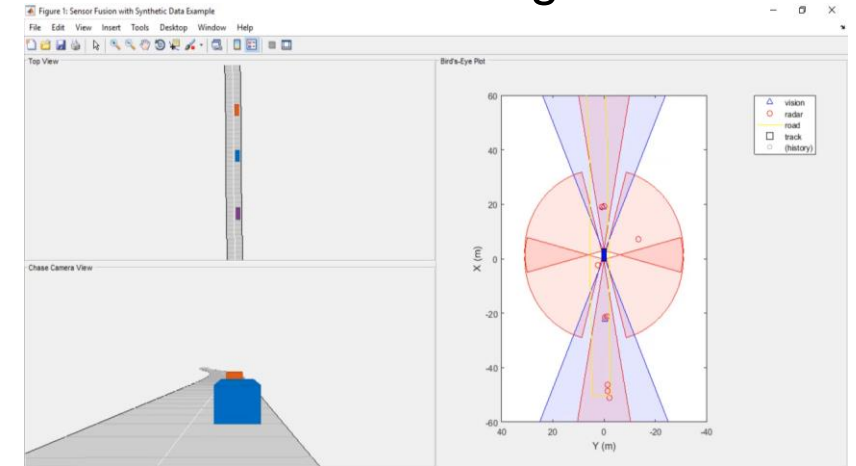
- Building it yourself enables you to control the level of fidelity/complexity appropriate for your application
- If building for multiple users, it is important to select an approach which is **scalable**, **maintainable**, and **testable**

Scenario Generation in MathWorks Tools

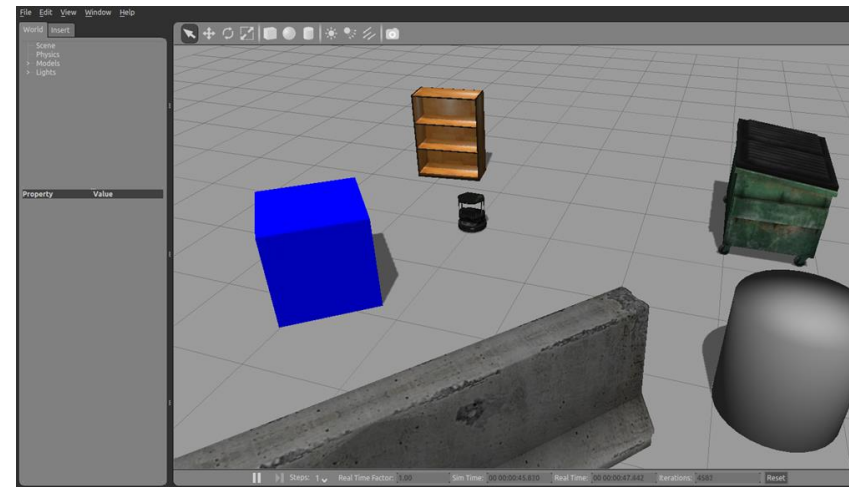
Phased Array Multi-Target Scenario



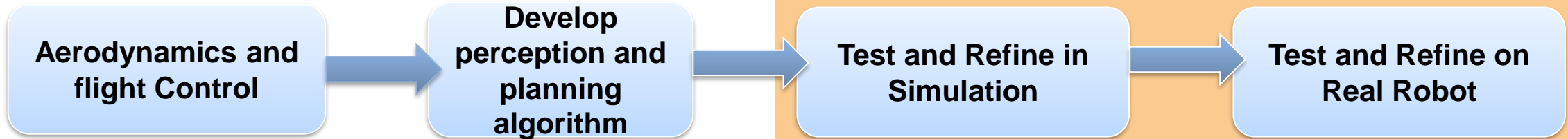
Autonomous Driving Scenario



Robotics Simulation Scenario

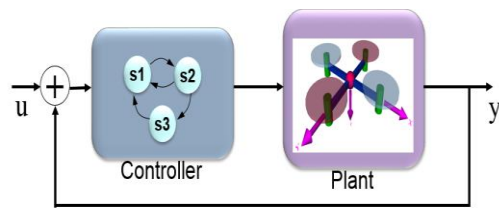


Aerial Autonomous System Development Workflow



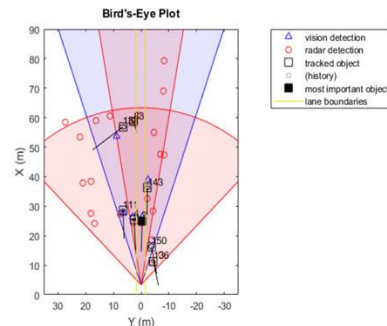
Challenge 1:

Understand the dynamics and design control algorithm



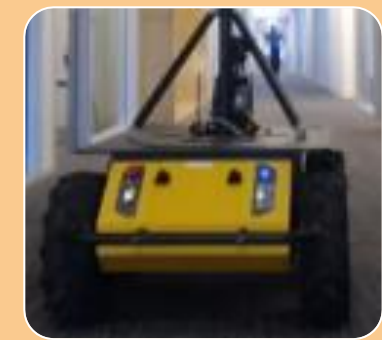
Challenge 2:

Design vision, radar and perception algorithms

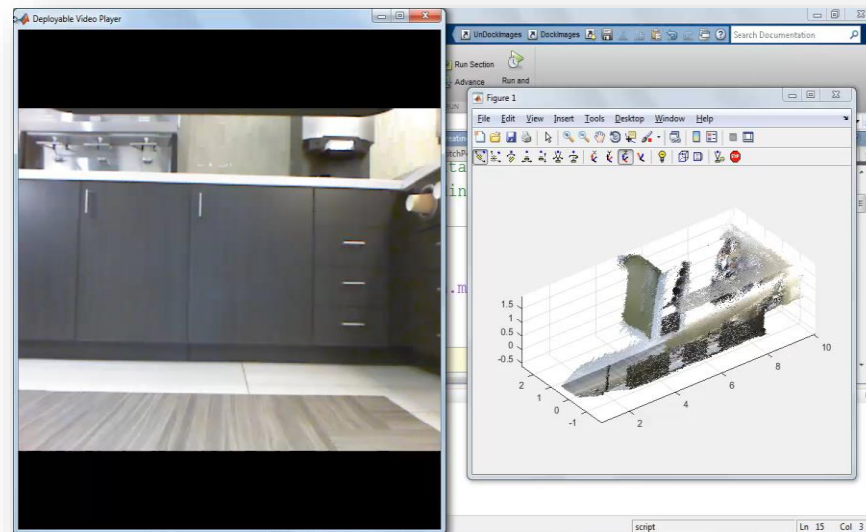
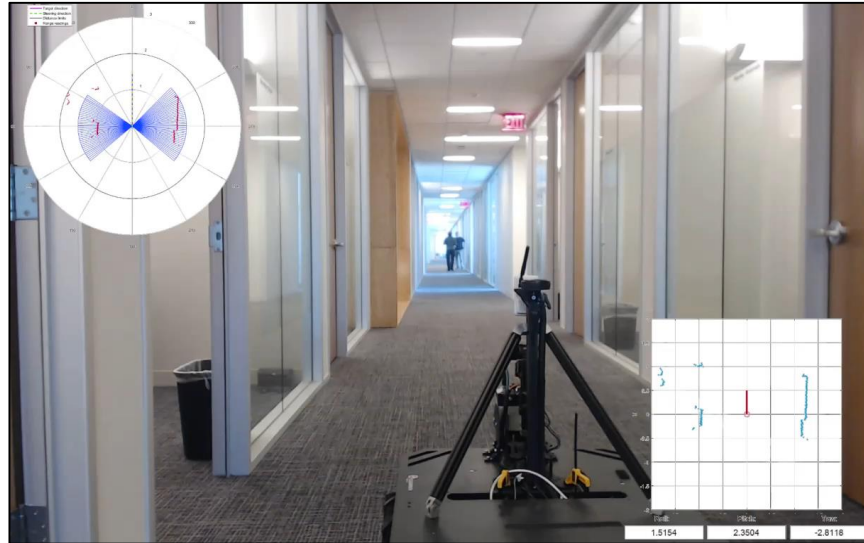


Birds Eye View

Challenge 3:
Verify and Implement the algorithm on to a real hardware

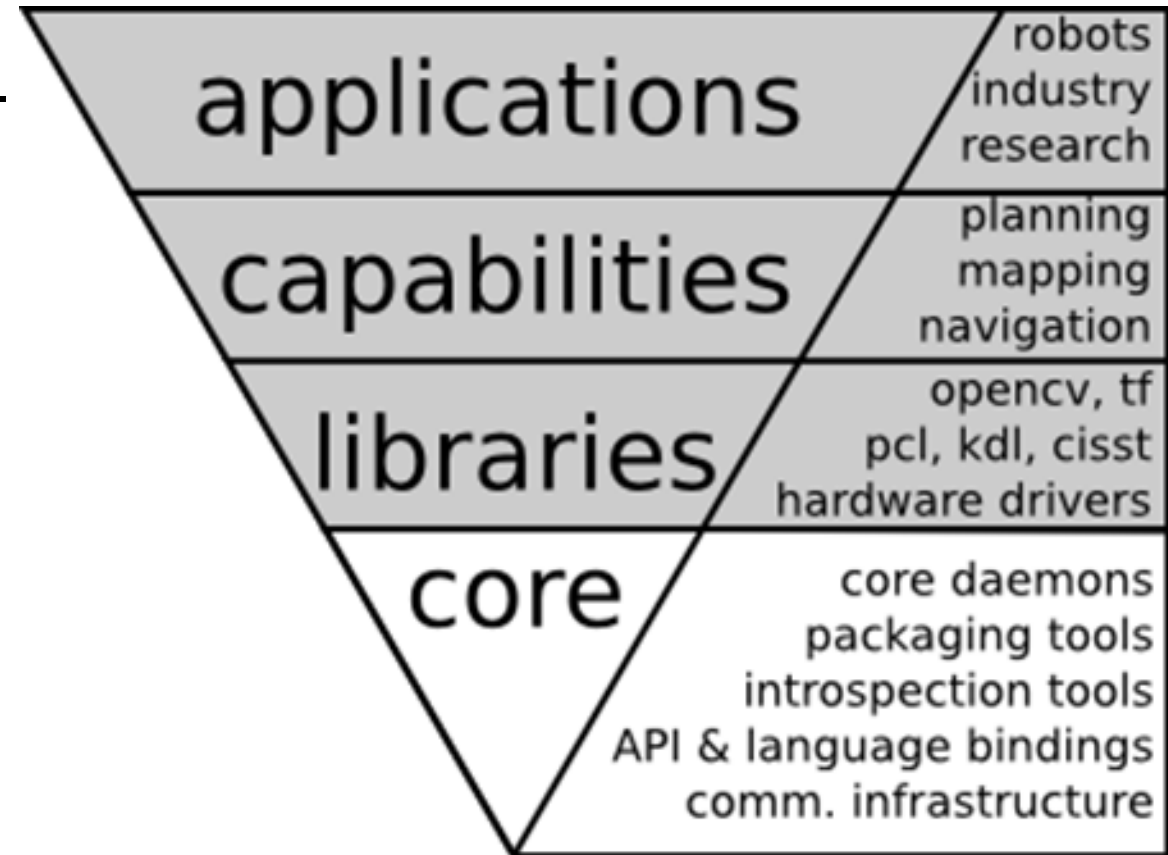


Examples of Hardware Implementation of Autonomous Systems

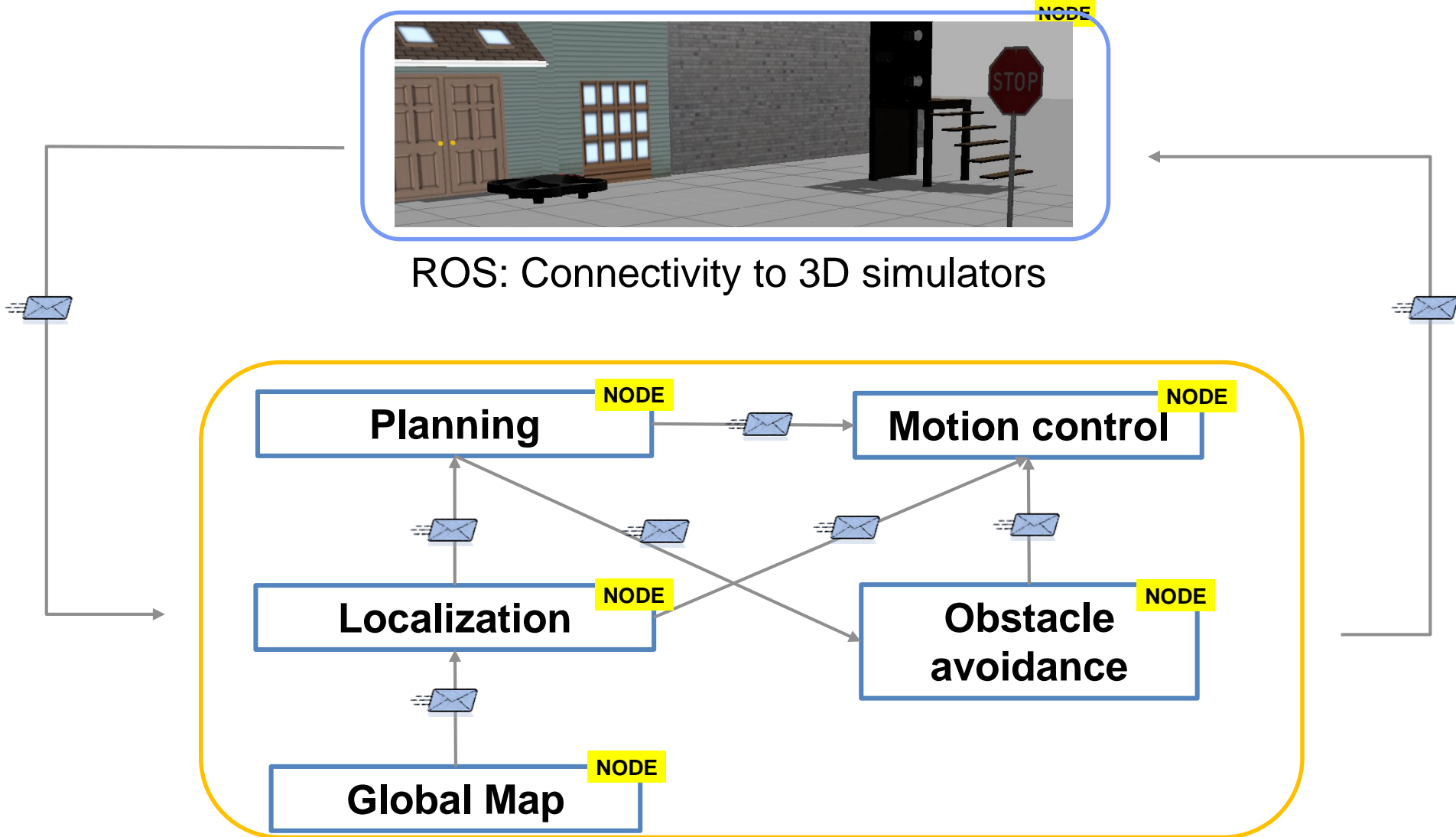


What is ROS (Robot Operating System)?

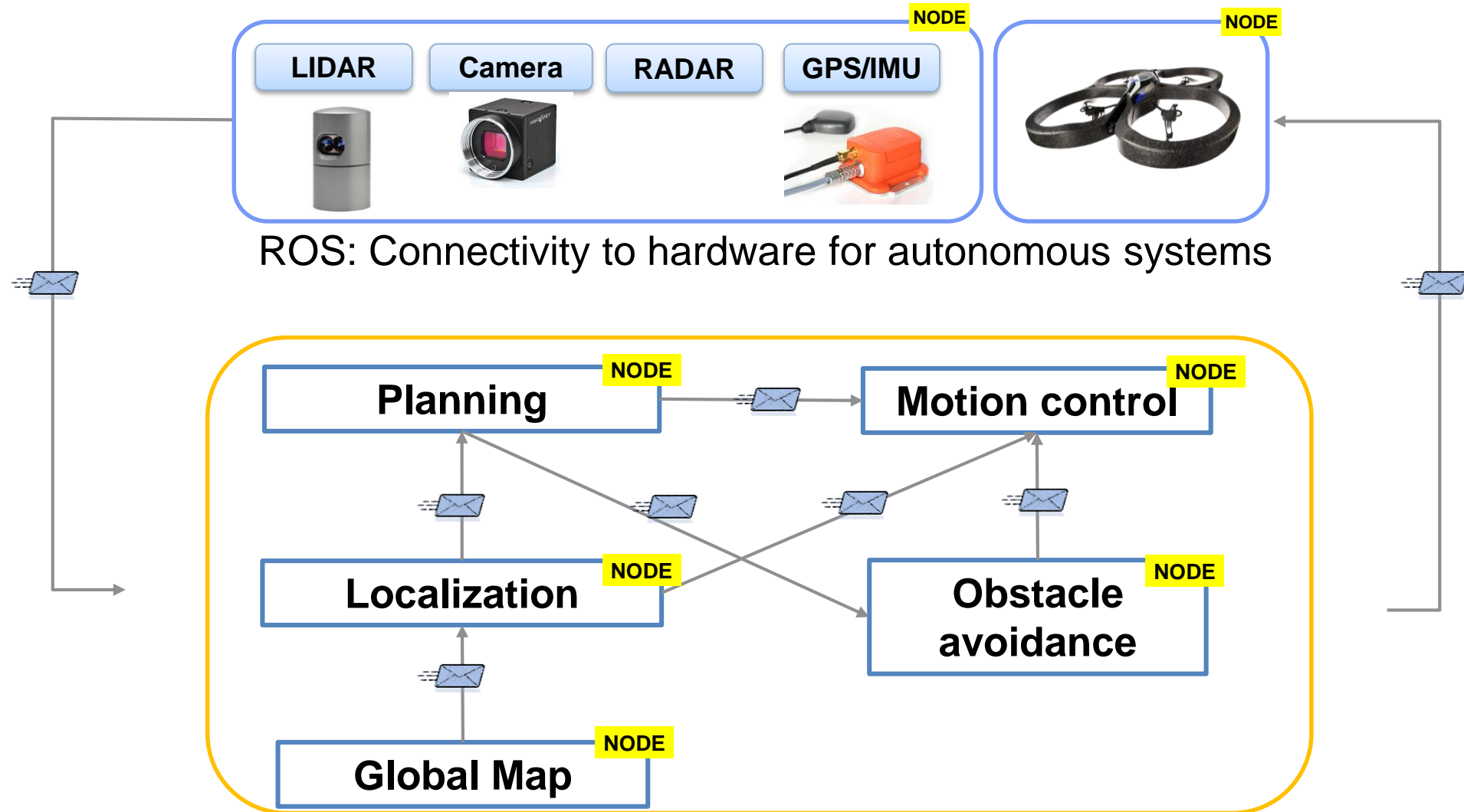
- An architecture for distributed inter-process communication
- Packages for common algorithms and drivers
- Multilanguage interface (C++, Python, Lua, Java and **MATLAB**)



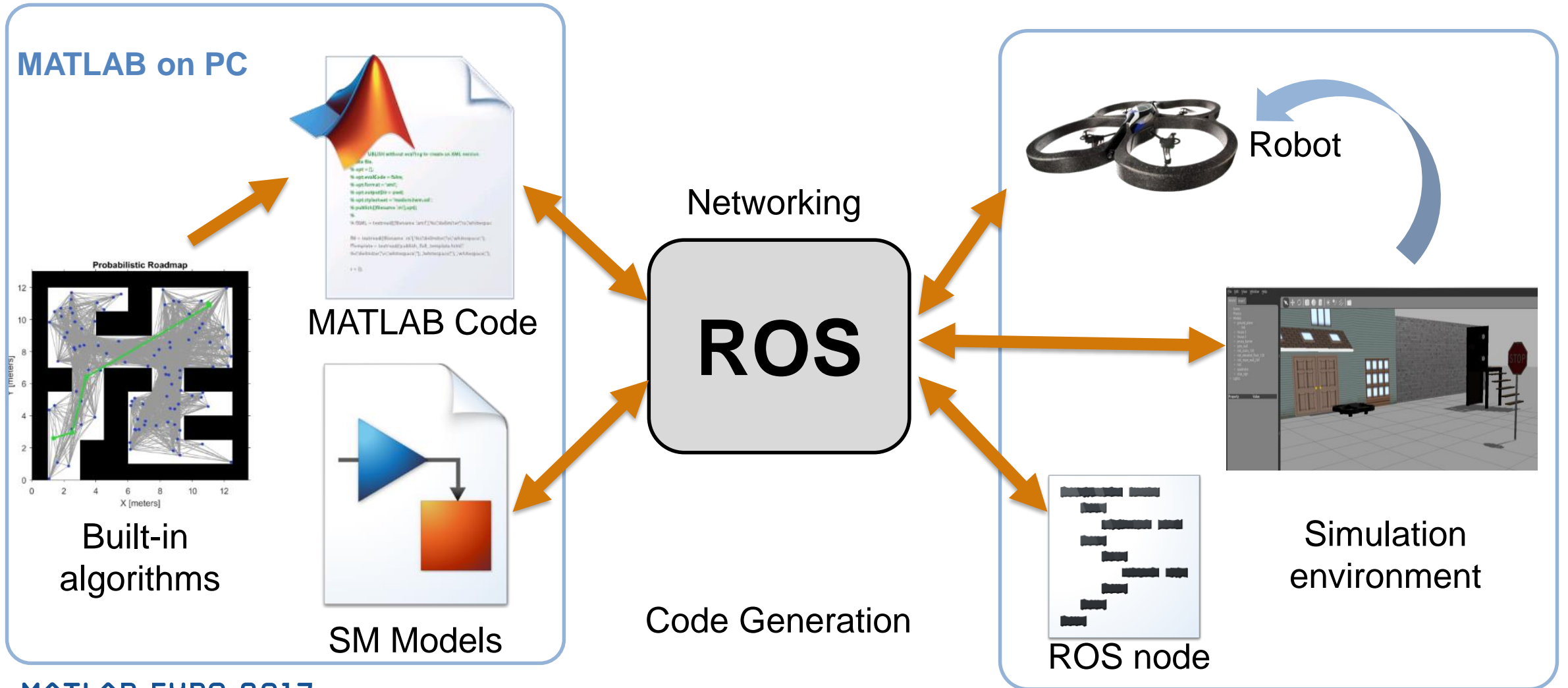
Advantages of ROS – Connectivity to 3D Simulators



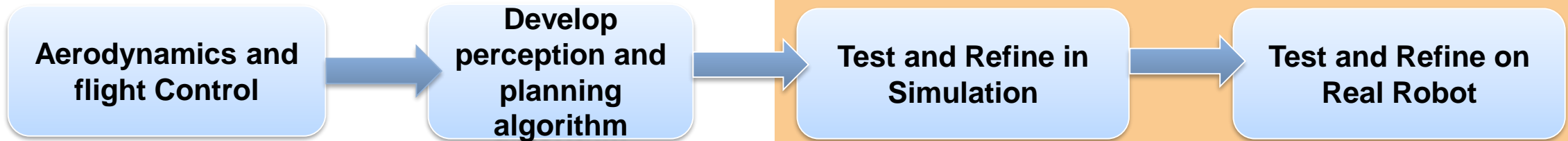
Advantages of ROS – Connectivity to Hardware



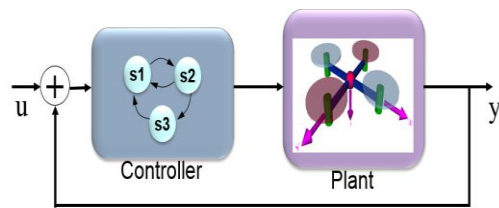
What can be done with the Robotics System Toolbox?



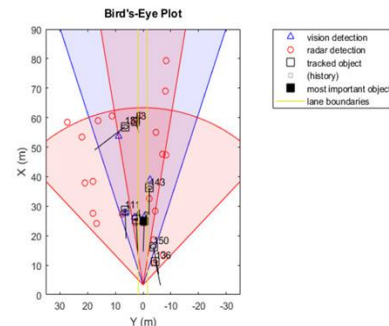
Aerial Autonomous System Development Workflow



Challenge 1:
Understand the dynamics and design control algorithm



Challenge 2:
Design vision, radar and perception algorithms

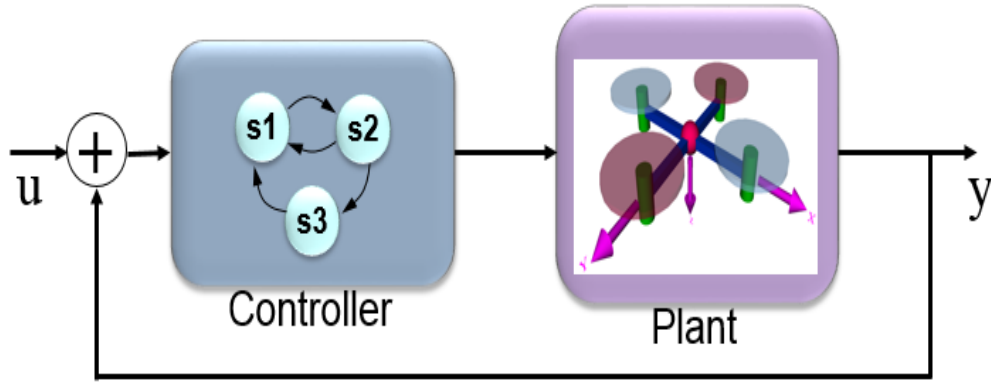


Birds Eye View

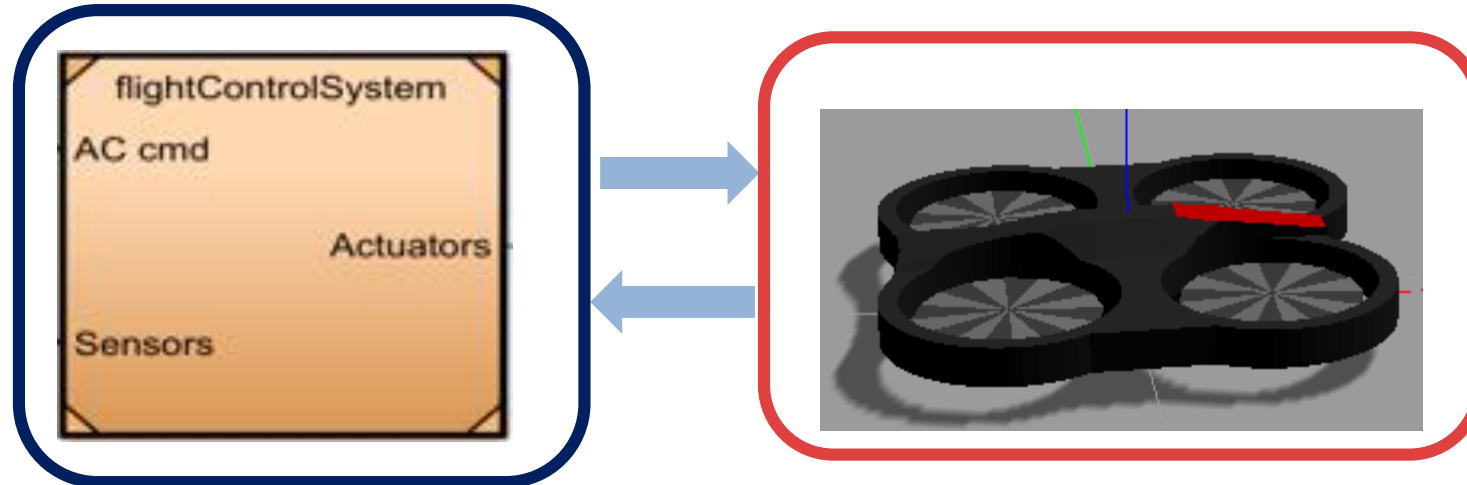
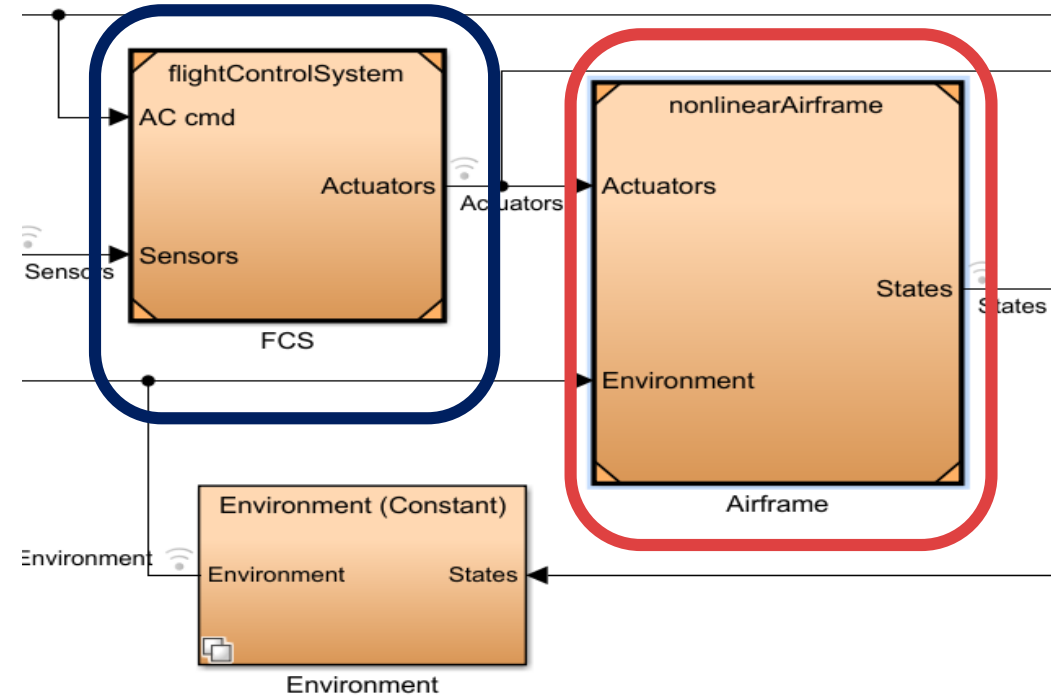
Challenge 3:
Verify and Implement the algorithm on to a real hardware



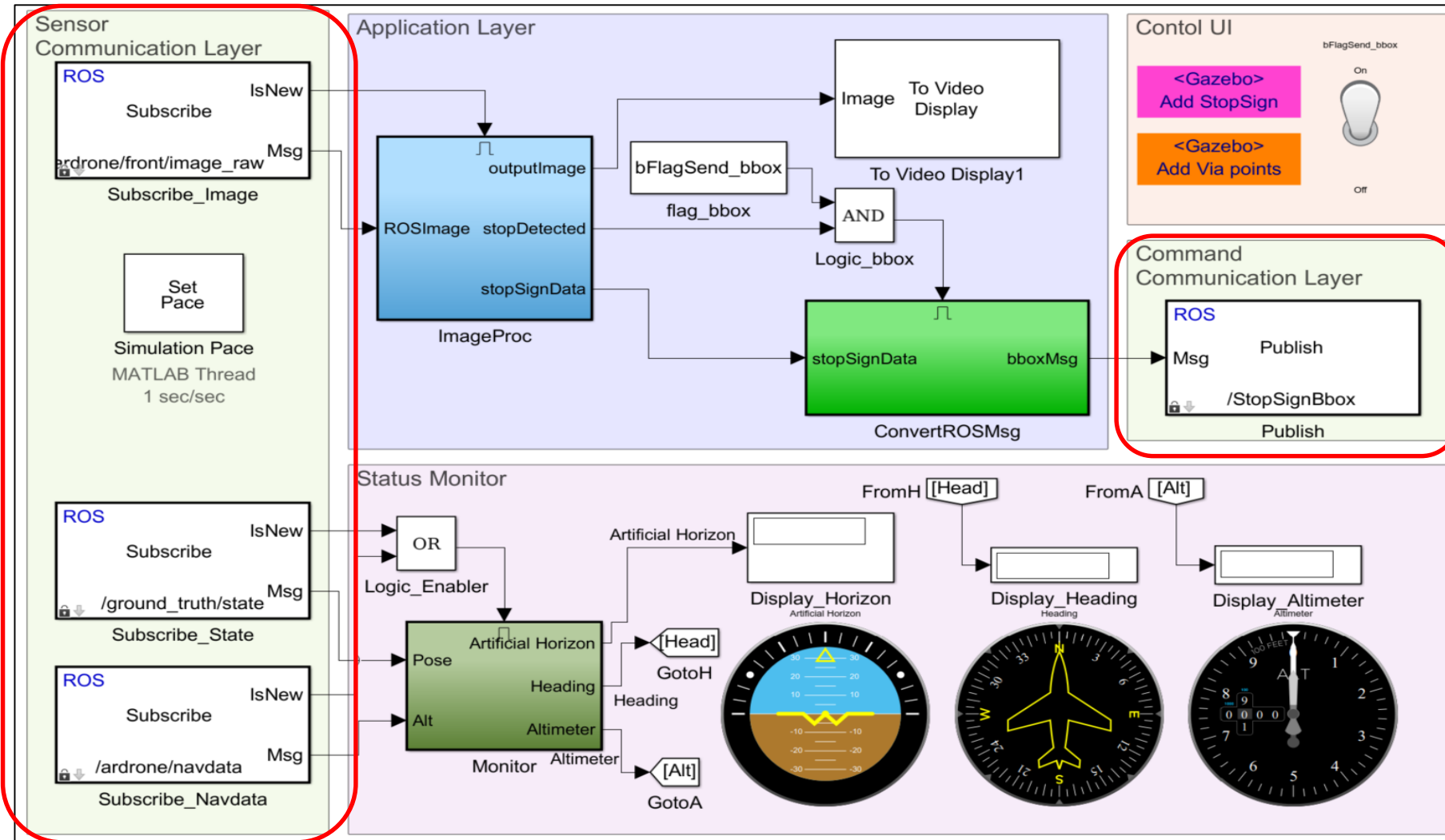
Quadrotor Motion Controller



Quadcopter Flight Simulation Model



System Level Design with MATLAB, Simulink and ROS



ROS as Communication Framework
 Algorithm Design Independent of ROS

Demo

The image displays a Simulink-Gazebo integration demo. It consists of three main windows:

- Top-Left Window (To Video Display):** Shows a 3D scene with a stop sign and a quadrotor.
- Bottom-Left Window (ImprocNode - Simulink):** Shows the Simulink model architecture. It includes:
 - Sensor Communication Layer:** Subscribes to ROS topics like `/frontImage_raw` and `/image`.
 - Application Layer:** Processes the image data using `ROSImage`, `stopDetector`, and `ImageProc` blocks. It outputs to `tfFlagSend_bbox` and `stopSignData`.
 - Status Monitor:** Monitors system health and displays various metrics like `Artificial Horizon`, `Heading`, and `Altitude`.
 - Control UI:** Contains buttons for `tfFlagSend_bbox` and `stopSignData`.
 - Command Communication Layer:** Publishes ROS messages like `StopSignBbox`.
- Right Window (Gazebo):** Shows the 3D environment with a quadrotor, a stop sign, and a barrier. The `World` panel lists models like `link`, `House 1`, `House 2`, `Jersey_barrier`, `grey_wall`, `rust_stairs_120`, `rust_elevated_floor_120`, `rust_mate_wall_240`, `rust`, `quadrotor`, `stop_sign`, and `Lights`. The `Property` panel shows details for the `ground_plane` model.

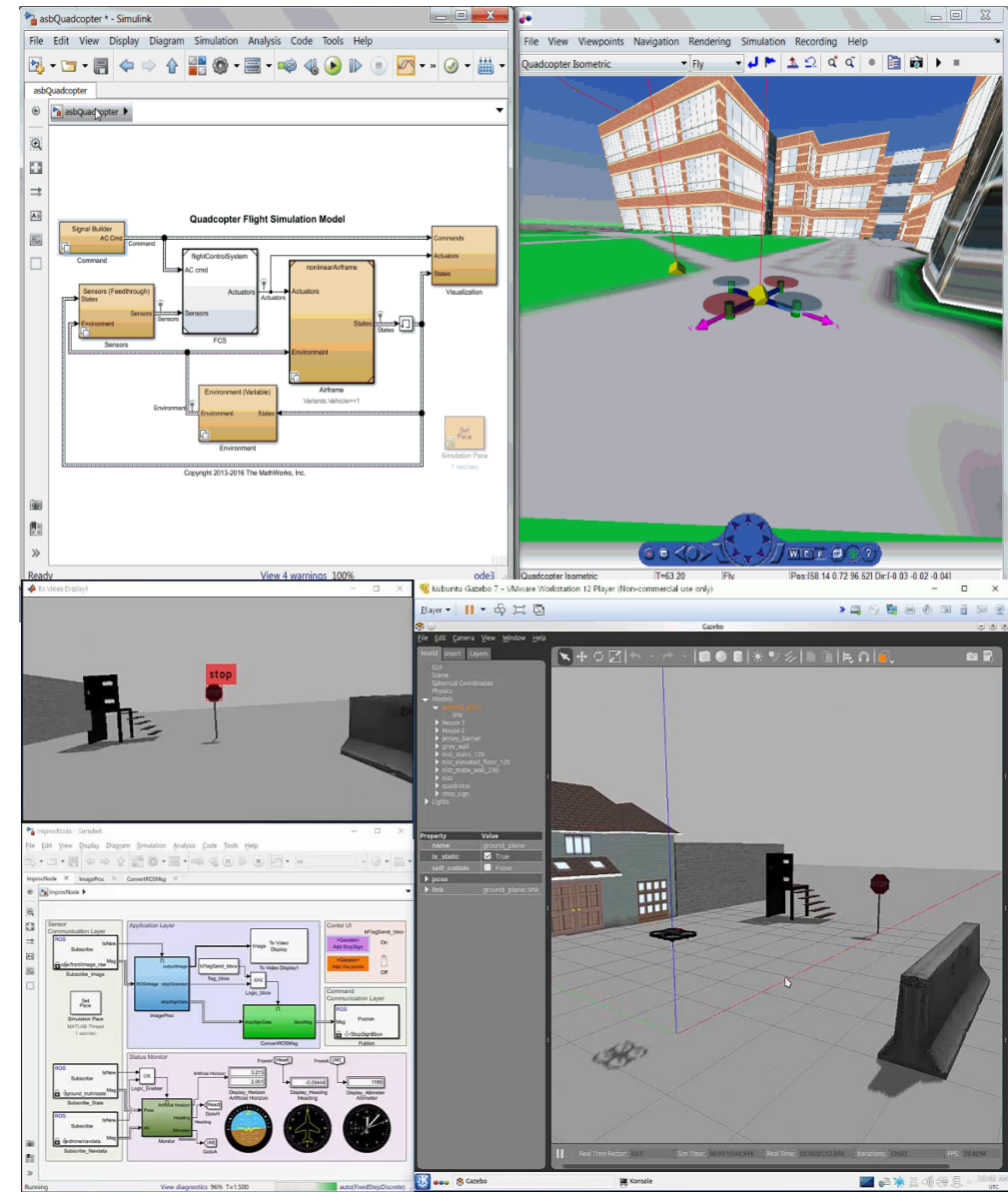
Key Takeaway

Autonomous system design using MATLAB and Simulink can help in :

- **Understanding the dynamics and develop the control algorithm**
 - Model aerodynamics, propulsion and motion
 - Design control algorithm in single environment

- **Design vision, radar, perception algorithms**
 - Visualizing different sensor data
 - Develop and test sensor fusion and tracking algorithm

- **Implementing the algorithm on actual hardware**
 - Test and verify algorithm on 3D simulators
 - Automatic C/C++ code generation on to actual hardware



MathWorks® | *Training Services*

Flexible delivery options:

- Public training available worldwide
- Onsite training with standard or customized courses
- Web-based training with live, interactive instructor-led courses
- Self-paced interactive online training



ENHANCE YOUR SKILLS

ADVANCE YOUR CAREER

More than 30 course offerings:

- Introductory and intermediate training on MATLAB, Simulink, Stateflow, code generation, and Polyspace products
- Specialized courses in control design, signal processing, parallel computing, code generation, communications, financial analysis, and other areas

Email: training@mathworks.in

MATLAB EXPO 2017



Control System Design with MATLAB and Simulink

This two-day course provides a general understanding of how to accelerate the design process for closed-loop control systems using MATLAB® and Simulink®.

Topics include:

- Control system design overview
- System modeling
- System analysis
- Control design
- Controller implementation



Designing Robotics Algorithms in MATLAB



This one-day course is for engineers designing mobile robotics algorithms for Robot Operating System (ROS) enabled simulators and robots.

Topics include:

- Listing the design workflows possible with Robotics System Toolbox™
- Communicating with ROS and Gazebo
- Building and testing mobile robotics algorithms
- Designing algorithms for execution and data sharing

MathWorks Training

 **Guaranteed to run**

Upcoming Public Trainings	Dates	Location
Image Processing with MATLAB	May 24 – 25 	Bangalore
Computer Vision with MATLAB	May 26 	Bangalore
Designing Robotics Algorithms in MATLAB	Sept 28	Pune

Email: training@mathworks.in

URL: <http://www.mathworks.in/services/training>

Phone: 080-6632-6000

Questions and Discussion



Accelerating the pace of engineering and science

Speaker Details

Email: Vivek.Raju@mathworks.in

LinkedIn: <https://www.linkedin.com/in/vivekraju87/>

Mobile: +91-8971669718

Contact MathWorks India

Products/Training Enquiry Booth

Call: 080-6632-6000

Email: info@mathworks.in

Your feedback is valued.

Please complete the feedback form provided to you.