

# MATLAB EXPO 2017

## KOREA

4월 27일, 서울

등록 하기 [matlabexpo.co.kr](http://matlabexpo.co.kr)

# 다중 센서 기반 자율시스템의 모델 설계 및 개발

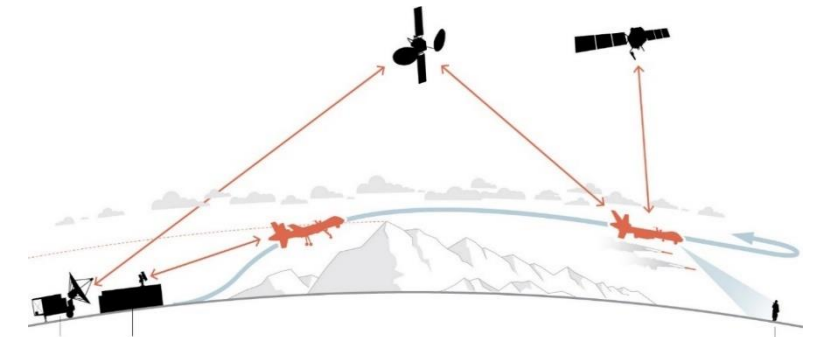
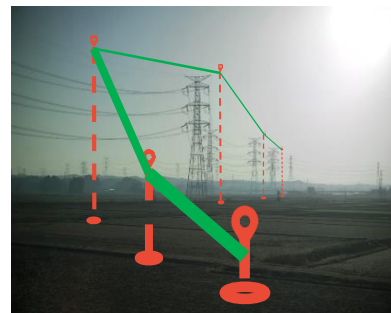
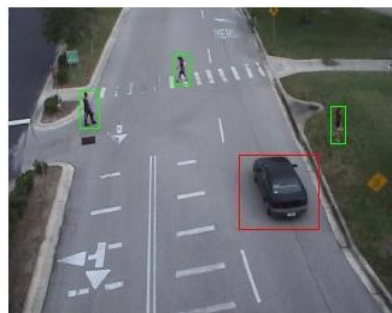
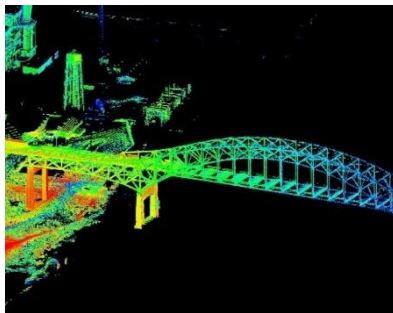
이제훈 차장

# What we will see today...



# Functional Segmentation of Autonomous System

Aircraft/ Platform	Sense	Perceive	Plan & Decide	Control	Connect/ Communicate
<ul style="list-style-type: none"> <li>• Control Surfaces, slats, flaps</li> <li>• Lifting Body</li> <li>• Landing Gear</li> <li>• Battery</li> <li>• Power Management</li> </ul>	<ul style="list-style-type: none"> <li>• Radar</li> <li>• Camera</li> <li>• Lidar</li> <li>• EO/IR</li> <li>• IMU</li> <li>• GPS-INS</li> <li>• HW Certification</li> </ul>	<ul style="list-style-type: none"> <li>• Environment mapping</li> <li>• Classification</li> <li>• Segmentation</li> <li>• Object Detection</li> <li>• Sensor Fusion</li> </ul>	<ul style="list-style-type: none"> <li>• Object Avoidance</li> <li>• Path &amp; motion planning</li> <li>• SLAM</li> </ul>	<ul style="list-style-type: none"> <li>• Guidance, Navigation &amp; Control</li> <li>• Flight SW certification</li> </ul>	<ul style="list-style-type: none"> <li>• Communication with ground operator</li> <li>• Multi-agent communication</li> <li>• Satellite data link</li> </ul>



# Functional Segmentation of Autonomous System

*Customer Groups*

Aircraft	Perception <i>“Autonomous Algorithms”</i>			Planning		Communication
Platform	Sense	Perceive	Plan & Decide	Control	Connect/ Communicate	
<ul style="list-style-type: none"> <li>Control Surfaces, slats, flaps</li> <li>Lifting Body</li> <li>Landing Gear</li> <li>Battery</li> <li>Power Management</li> </ul>	<ul style="list-style-type: none"> <li>Radar</li> <li>Camera</li> <li>Lidar</li> <li>EO/IR</li> <li>IMU</li> <li>GPS-INS</li> <li>HW Certification</li> </ul>	<ul style="list-style-type: none"> <li>Environment mapping</li> <li>Classification</li> <li>Segmentation</li> <li>Object Detection</li> <li>Sensor Fusion</li> </ul>	<ul style="list-style-type: none"> <li>Object Avoidance</li> <li>Path &amp; motion planning</li> <li>SLAM</li> </ul>	<ul style="list-style-type: none"> <li>Guidance, Navigation &amp; Control</li> <li>Flight SW certification</li> </ul>	<ul style="list-style-type: none"> <li>Communication with ground operator</li> <li>Multi-agent communication</li> <li>Satellite data link</li> </ul>	



# Autonomous System Development Workflow

**Aerodynamics and Flight Control**

**Autonomous algorithm**

**Test and Refine in Simulation**

**Test and Refine on Real Robot**

**Challenge 1:**

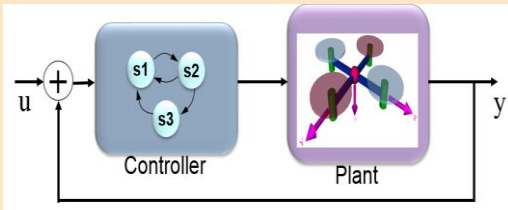
Understand the dynamics with control algorithm

**Challenge 2:**

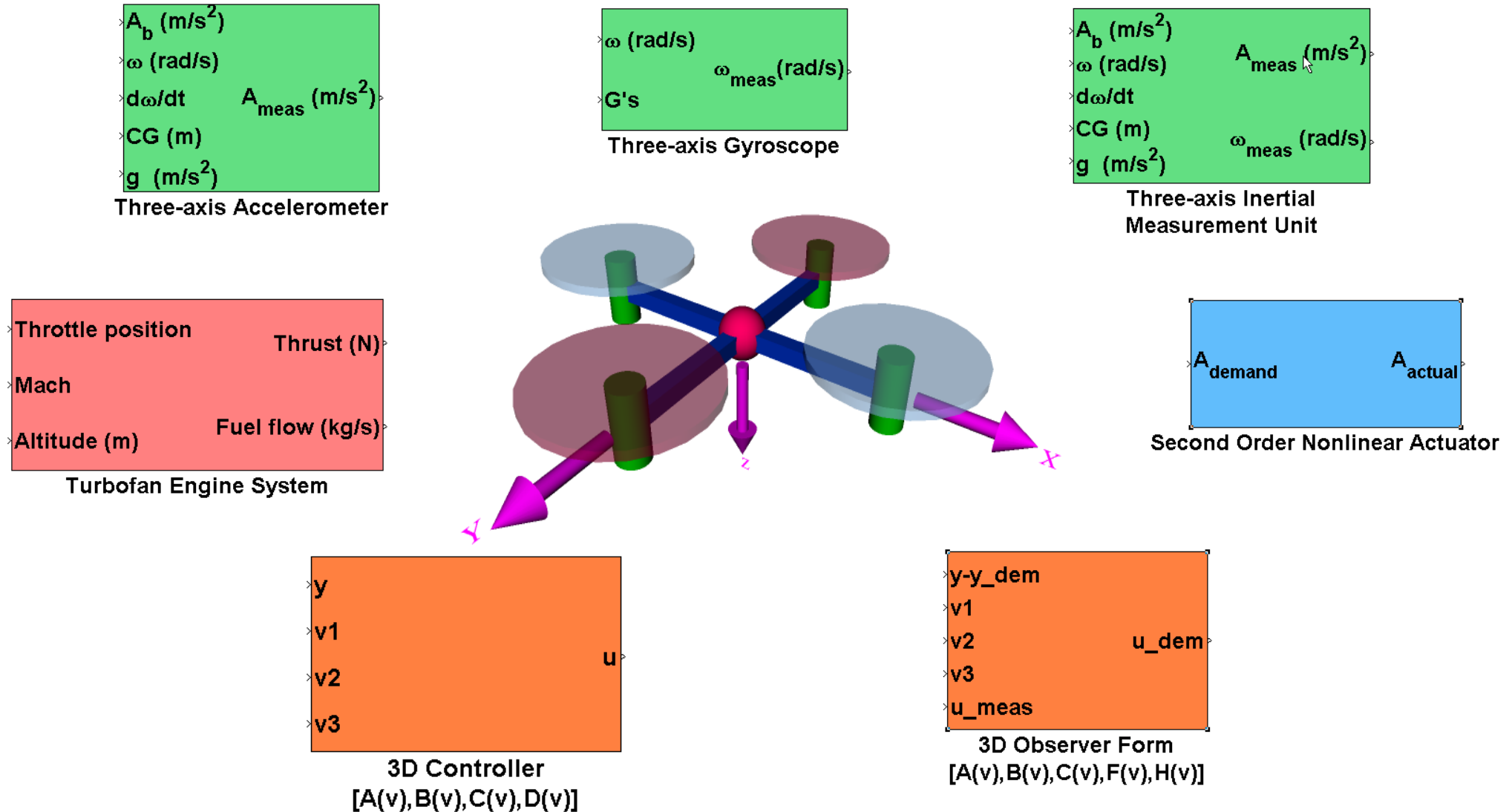
Design and Verify autonomous algorithms

**Challenge 3:**

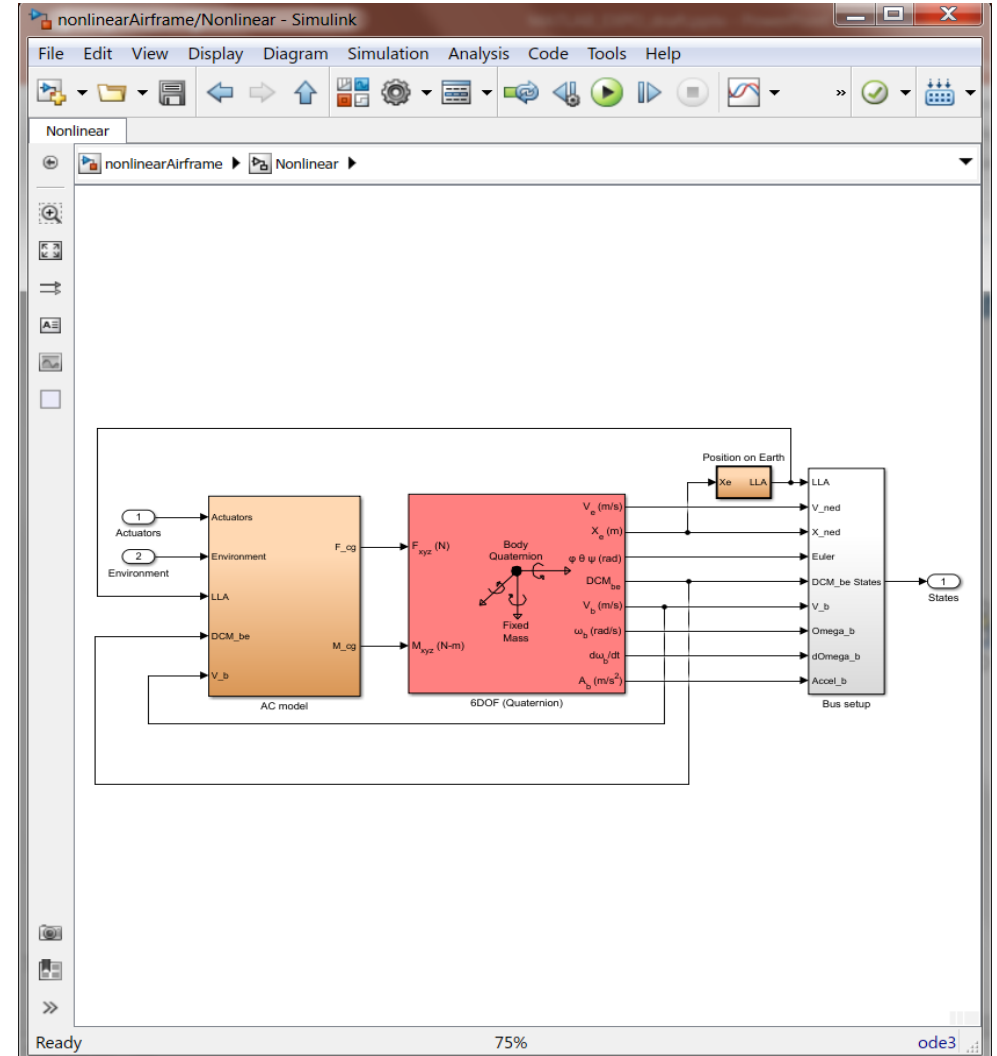
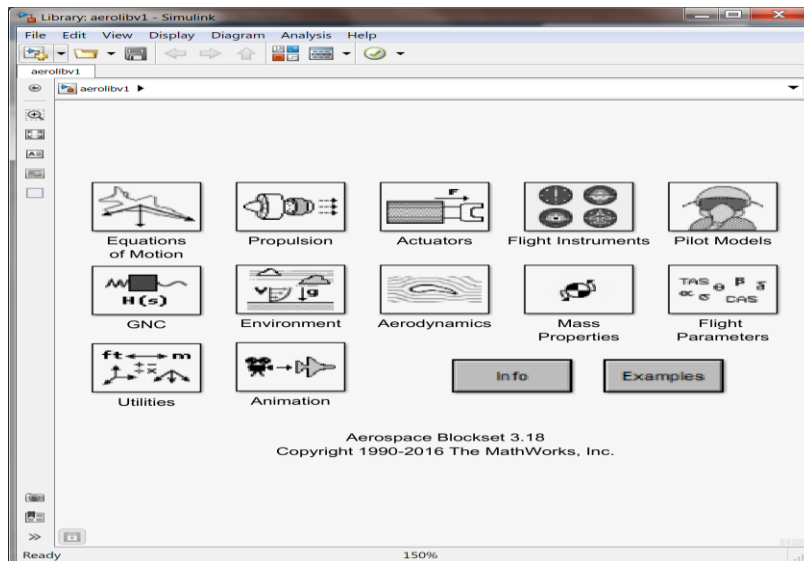
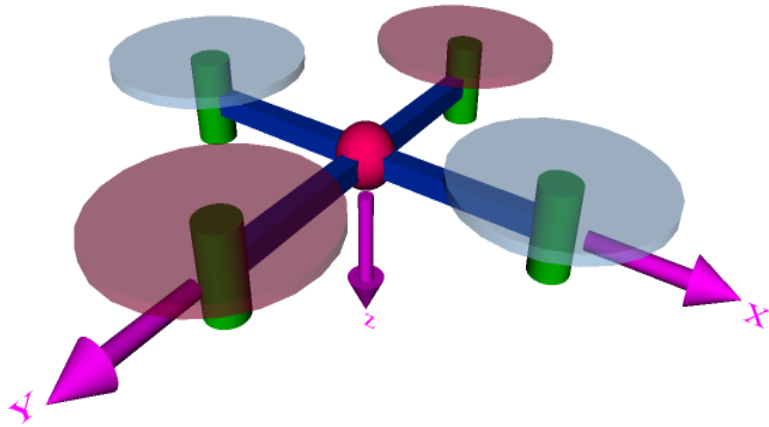
Verify and Implement the algorithm on to a real hardware



# Design Control algorithm with Dynamics



# Design Control algorithm with Dynamics

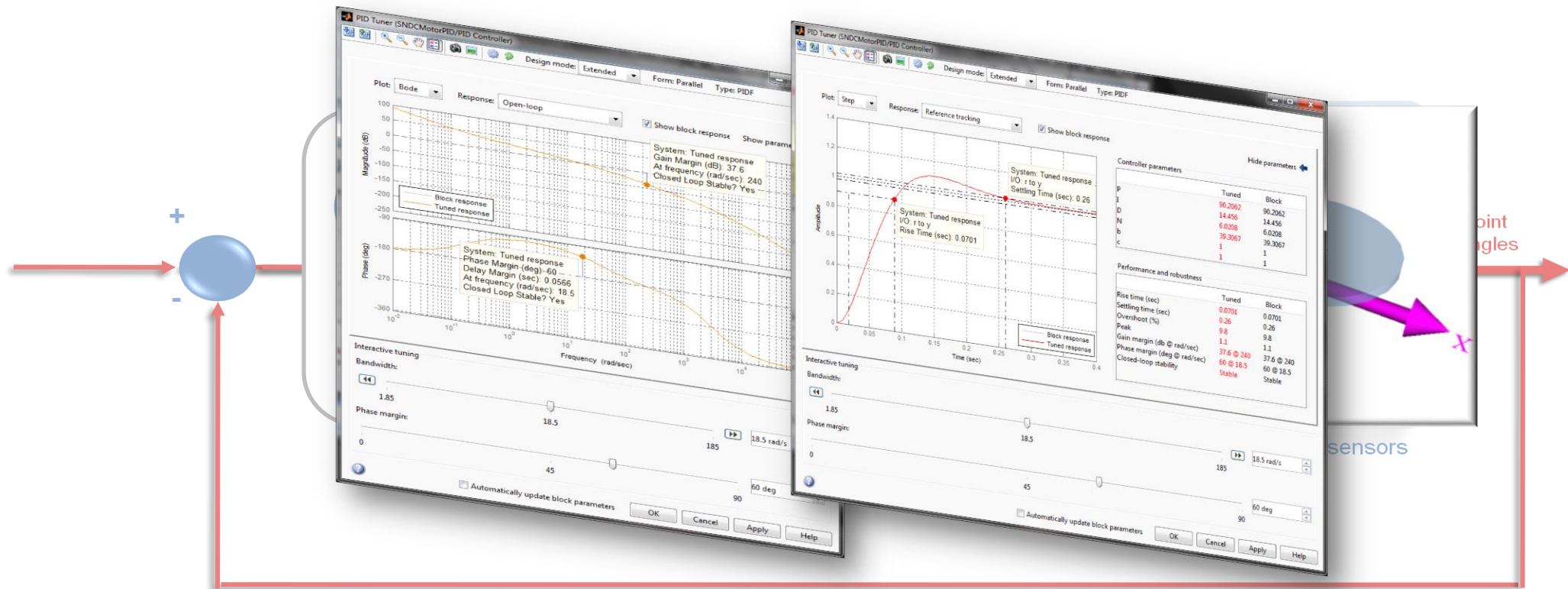




# Design Control algorithm with Dynamics

*Simulink Control Design & Control System Toolbox*

Simulating plant and controller ***in one environment***  
***Optimize system-level performance & Closed-loop simulation***



## ***Simulink Control Design & Control System Toolbox***

automatically linearize the plant, design and tune your PID controllers

# Design Control algorithm with Dynamics

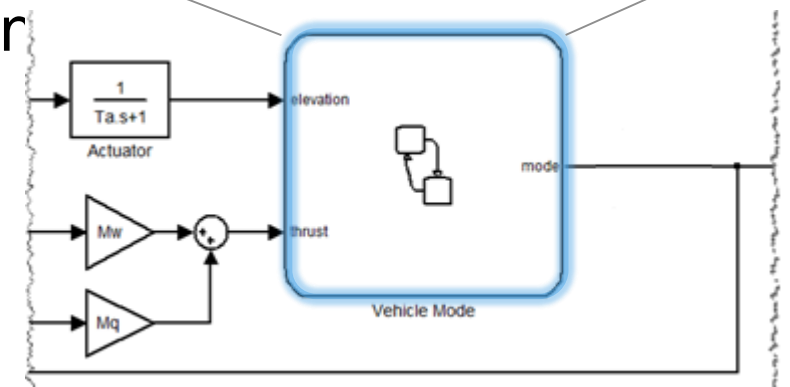
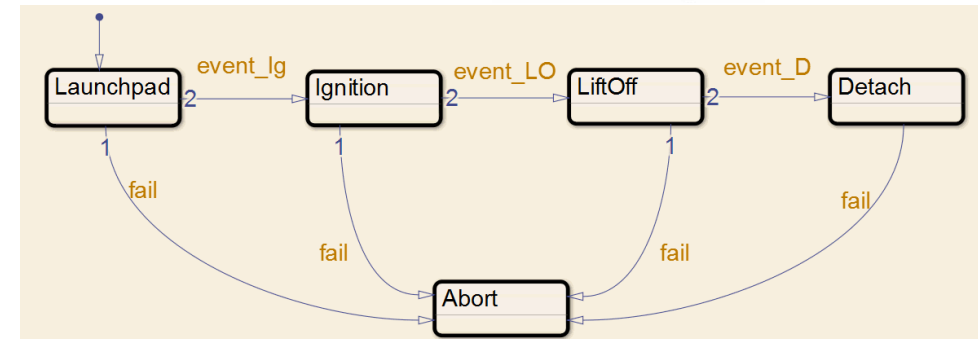
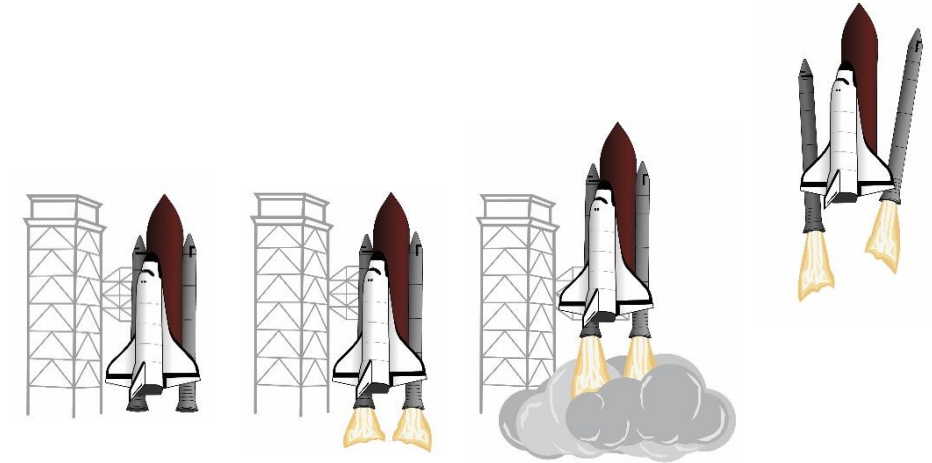
The image displays two windows from a MATLAB/Simulink environment. The left window, titled "asbQuadcopter \* - Simulink", shows a Simulink model titled "Quadcopter Flight Simulation Model". The model consists of several interconnected blocks: a "Signal Builder AC Cmd" block providing a "Command" input; a "FlightControlSystem" block (containing "AC cmd" and "Actuators" sub-blocks) that processes the command and outputs "Actuators"; a "nonlinearAirframe" block (containing "Actuators" and "States" sub-blocks) that receives the "Actuators" and outputs "States"; and an "Environment" block that provides "Sensors" to the "FlightControlSystem" and "nonlinearAirframe". A "Sensors (Feedthrough)" block also provides "Sensors" to the "FlightControlSystem". The "nonlinearAirframe" block is configured with "Variants.Vehicle==1". The "Environment" block also includes an "Environment (Variable)" sub-block. The model outputs "Commands", "Actuators", and "States" to a "Visualization" block. A "Set Pace" button is visible in the bottom right of the Simulink window. The status bar at the bottom of the Simulink window shows "Ready", "View 4 warnings 84%", and "ode3".

The right window, titled "Quadcopter Isometric", shows a 3D visualization of the quadcopter simulation. The quadcopter is shown in a "Fly" mode, hovering in a virtual environment with a brick building and a green field. The status bar at the bottom of this window displays "Quadcopter Isometric", "T=63.20", "Fly", and "Pos:[58.14 0.72 96.52] Dir:[-0.03 -0.02 -0.04]".

# Design Control algorithm with Dynamics

## Stateflow

- Model and simulate decision logic
  - supervisory control
  - task scheduling
  - fault management
- Develop mode-logic
  - using state machines and flow charts
- See how the logic behaves with diagram animation
  - and integrated debugger



# Design Control algorithm with Dynamics

The image displays two side-by-side windows from the MathWorks environment. The left window, titled "Stateflow (chart) sf\_launchabort/launchAbortController - Simulink", shows a Stateflow chart for the "launchAbortController". The chart is divided into two main sections: "ModeLogic" and "Abort".

**ModeLogic Section:**

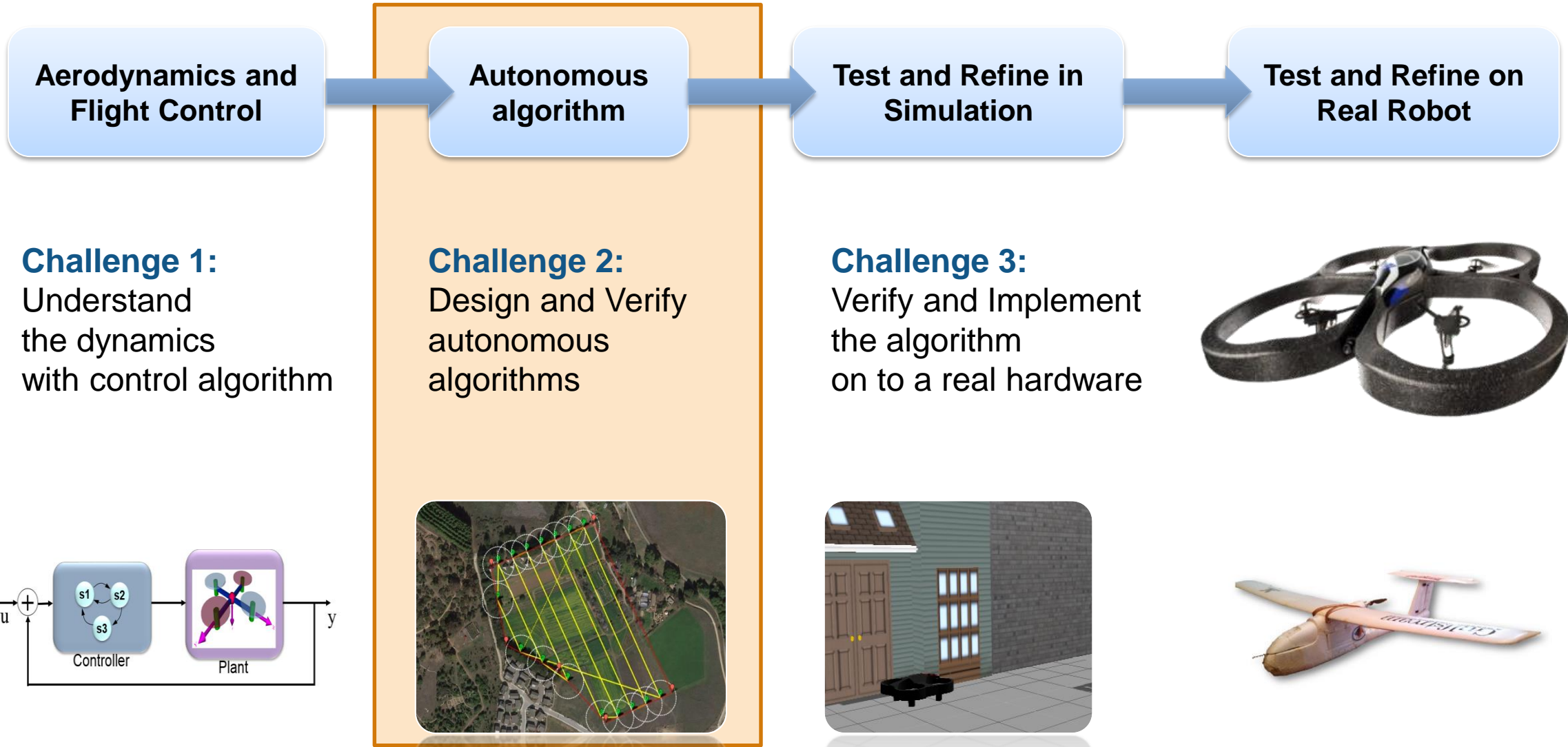
- RTLS** (entry: mode=1) transitions to **DRL** (entry: mode=2) on the condition `[alt>10000 && !anomaly]`.
- DRL** transitions to **AOA** (entry: mode=3) on the condition `[alt>100000 && !anomaly]`.
- AOA** transitions to **ATO** (entry: mode=4) on the condition `[alt>400000 && !anomaly]`.
- ATO** transitions back to **RTLS** on the condition `[alt>10000 && !anomaly]`.

**Abort Section:**

- Normal** (en: abort=NO\_ABORT) transitions to **abortLogic** on the condition `[anomaly]`.
- abortLogic** transitions to **abortComplete** (en: abort=ABORT\_COMPLETE).

The right window, titled "Launch Abort - VR", shows a 3D visualization of a 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" view, an "Examine" button, and a VR control panel at the bottom. The status bar at the bottom of the VR window shows: "Chase Plane", "T=69.30", "Examine", and "Pos:[183.26 502.14 -56.50] Dir:[-0.09 0.00 0.04]".

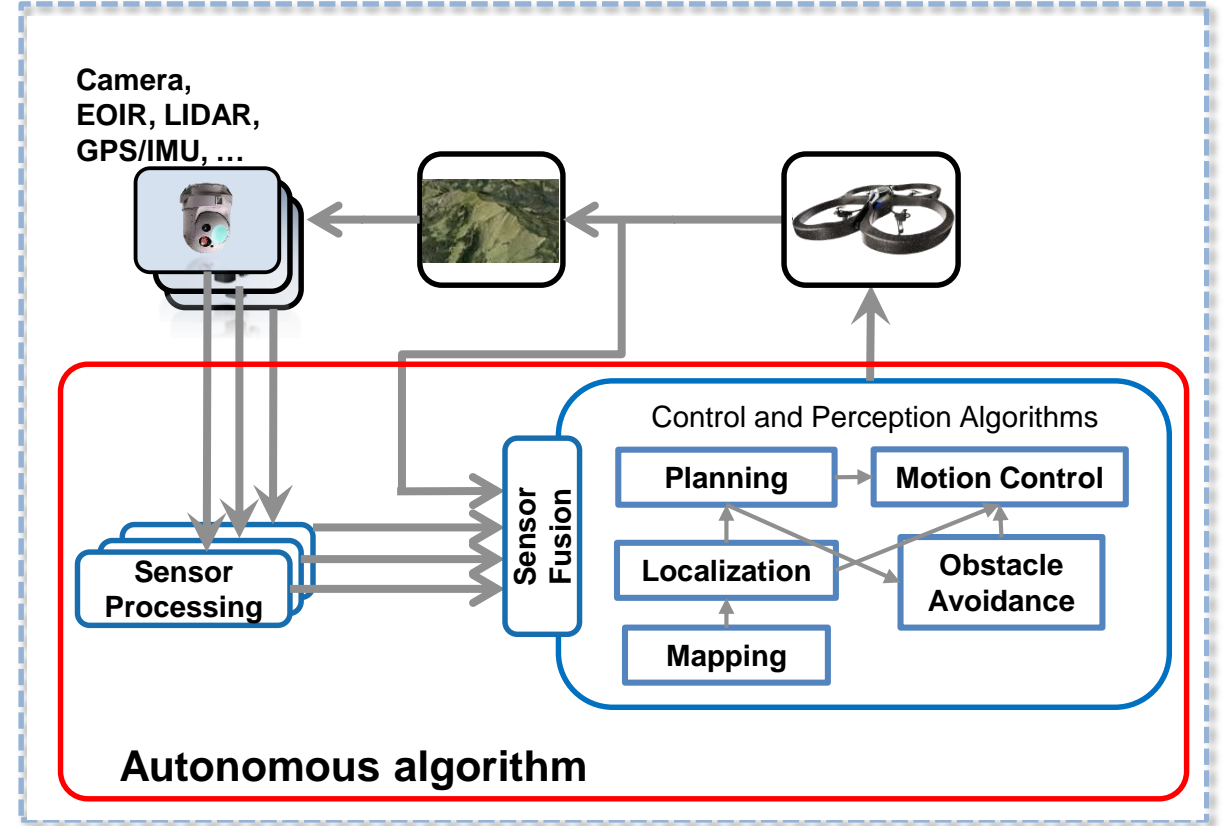
# Autonomous System Development Workflow



# How to develop Autonomous algorithms?



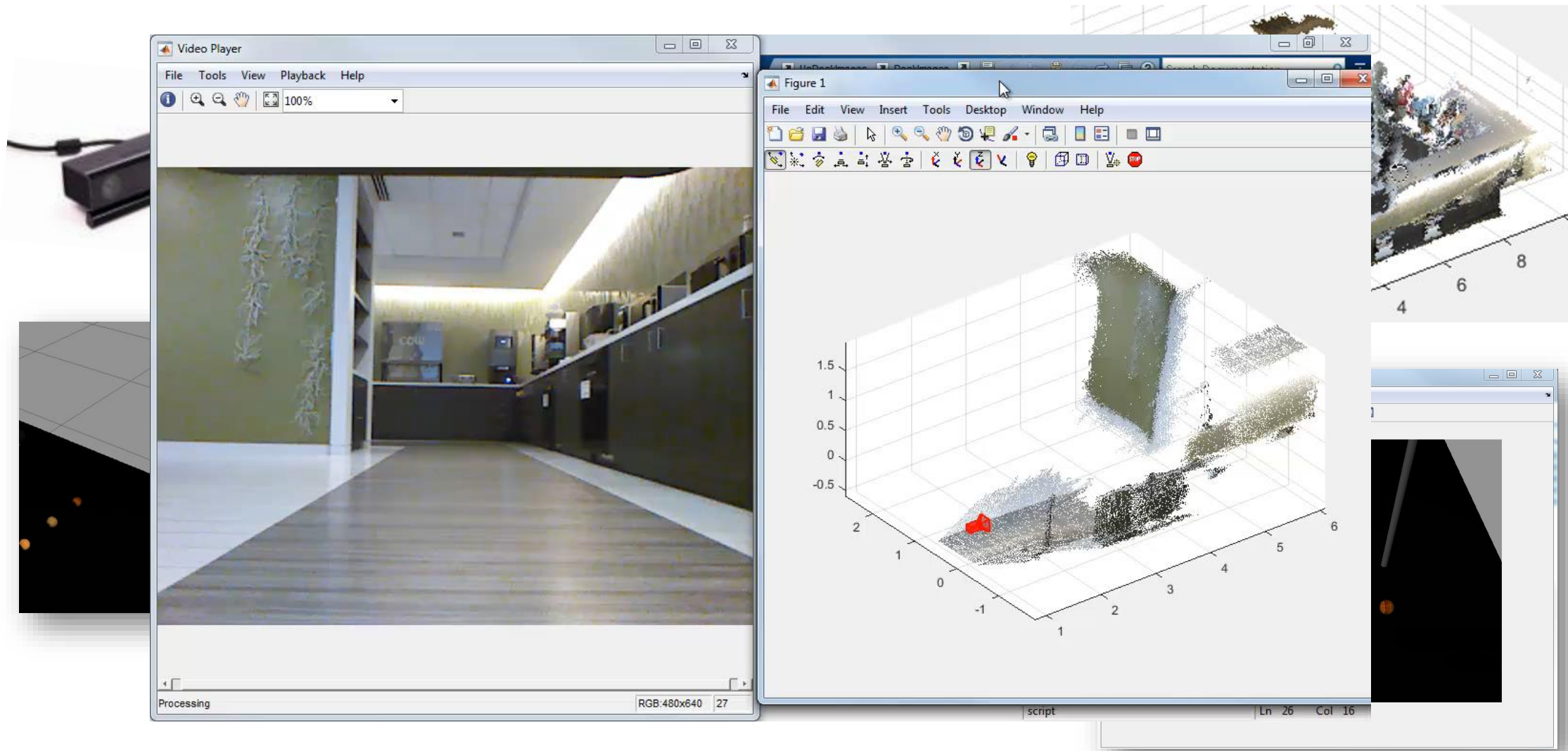
Autoroute, gated, post-process automatically



Efficient, speed, endurance, automation

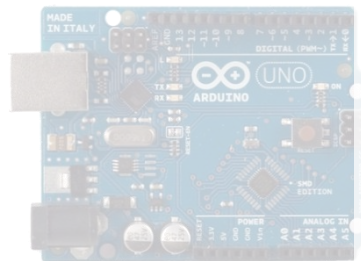
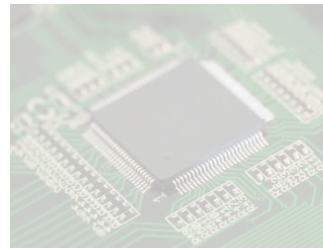
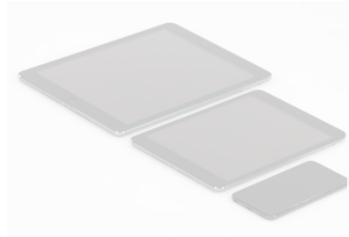
# How to develop Autonomous algorithms?

*Manage Sensor data*



# How to develop Autonomous algorithms?

*Manage Sensor data*



Data & RF

Embedded

Imaging

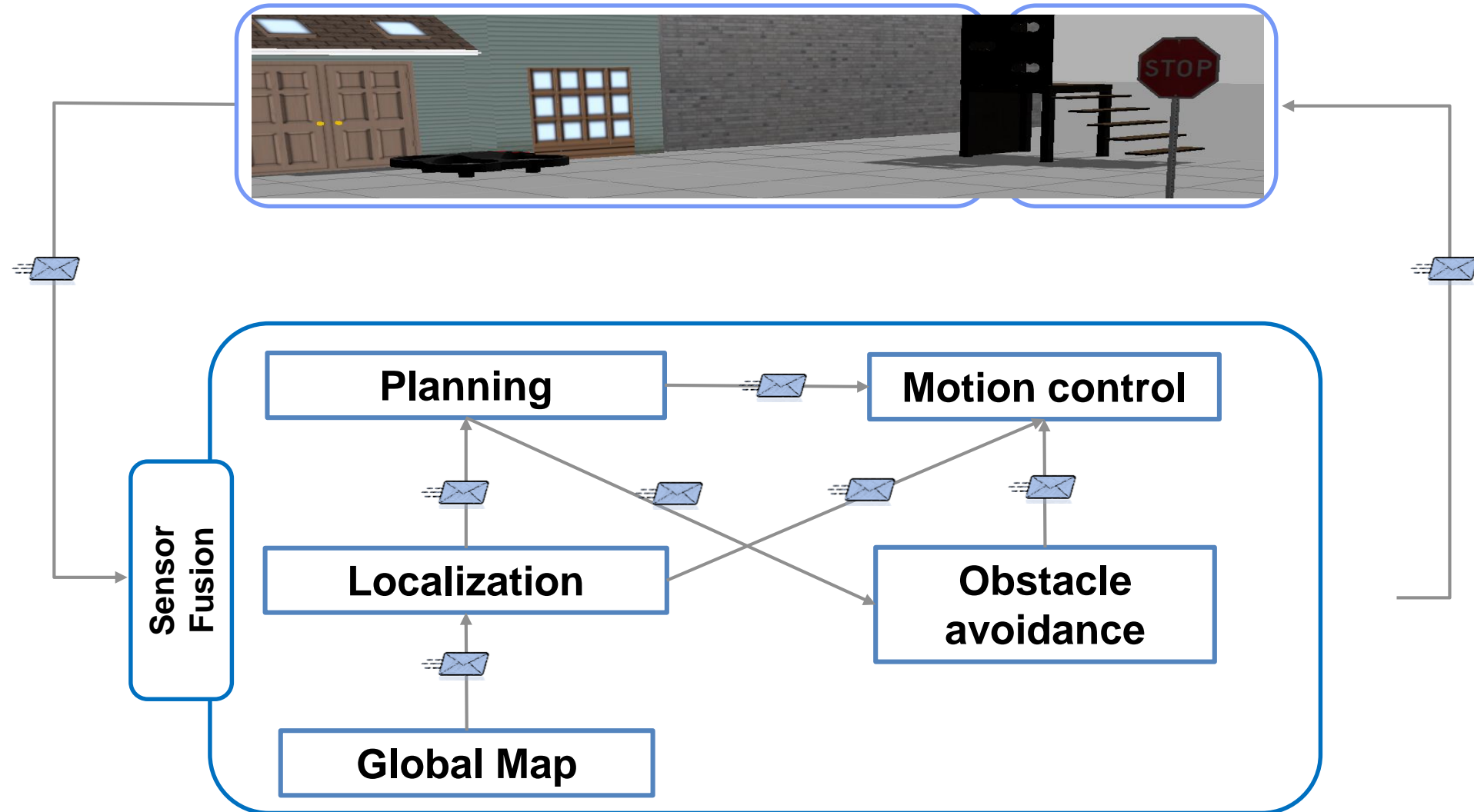
Specialty

Standards



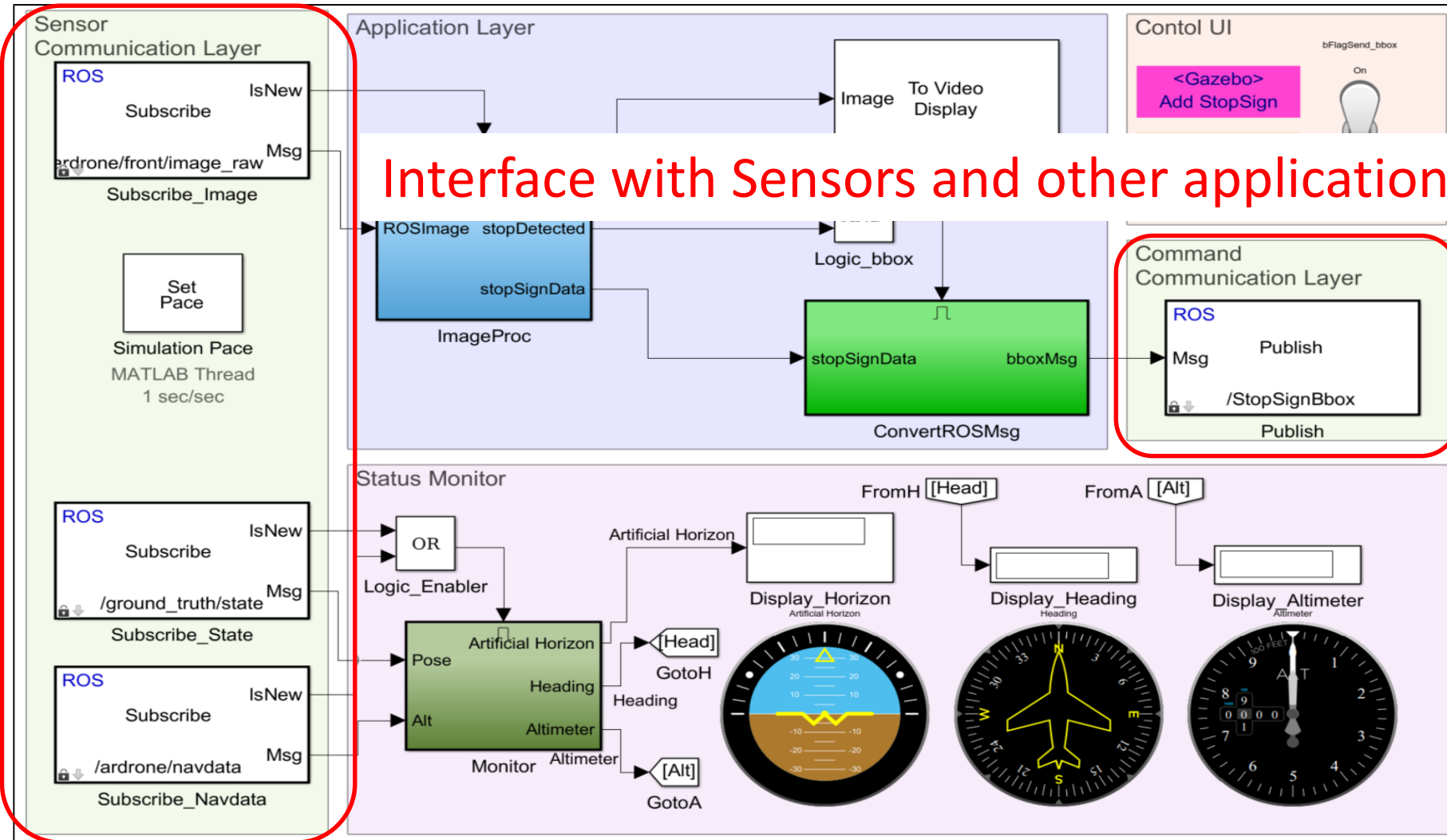
# How to develop Autonomous algorithms?

*Sensor data flow*



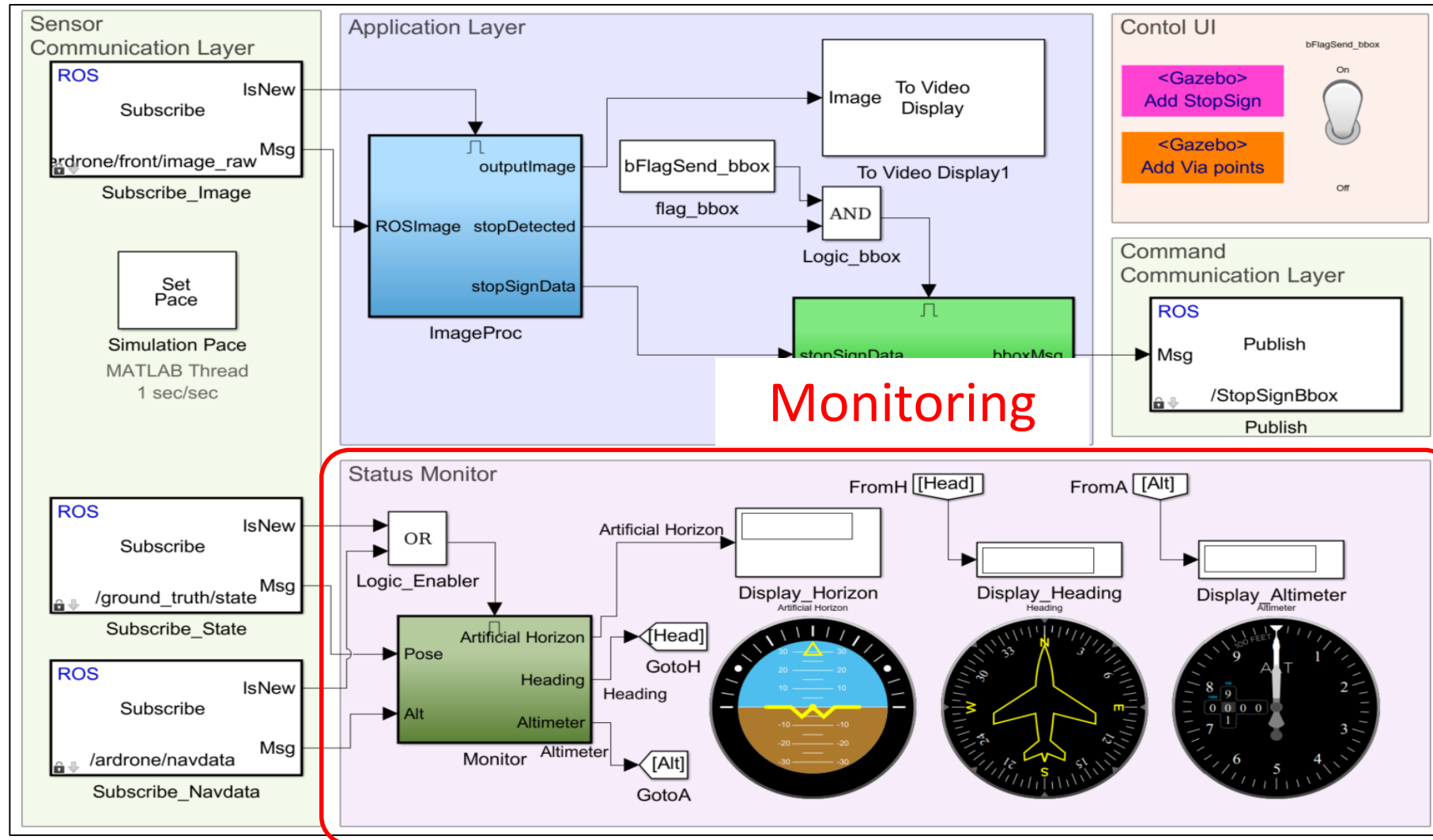
# How to develop Autonomous algorithms?

## System Level Design with MATLAB, Simulink



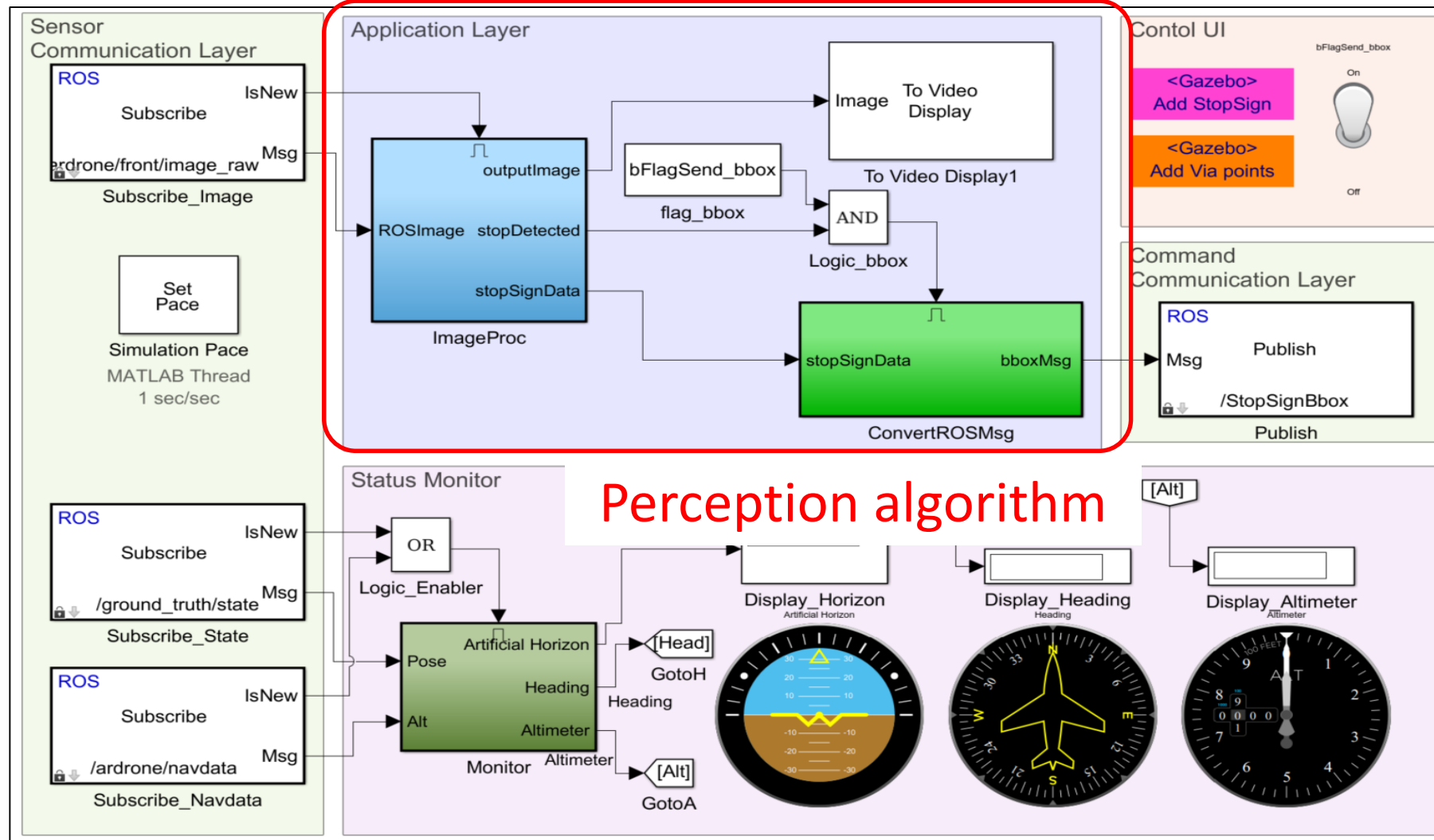
# How to develop Autonomous algorithms?

## System Level Design with MATLAB, Simulink



# How to develop Autonomous algorithms?

## System Level Design with MATLAB, Simulink

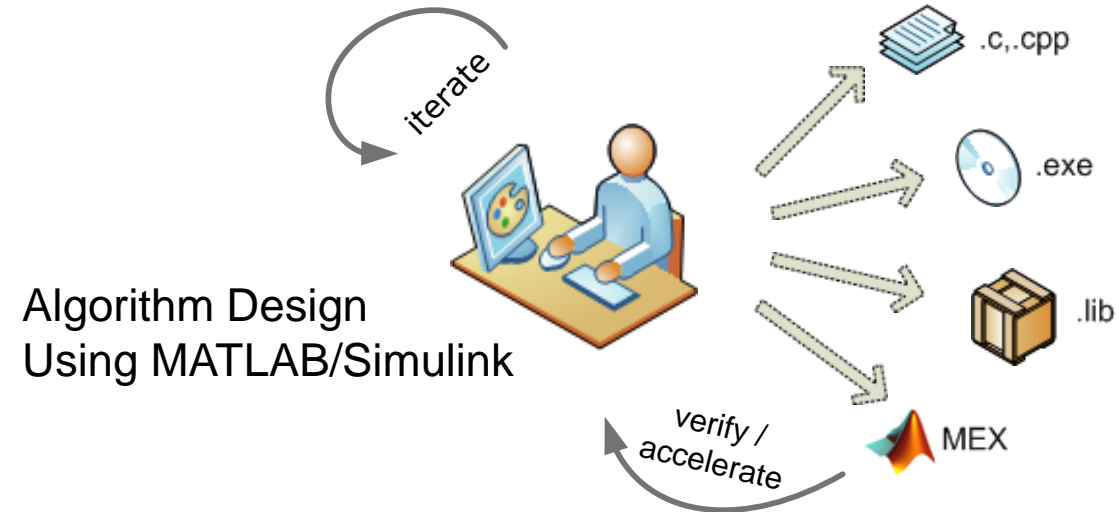


# How to develop **Autonomous algorithms?**

*System Level Design with MATLAB, Simulink*

- Design Perception algorithm

- Matlab function
- S-function
- System Object
- State & flow control



- Calling Libraries Written in Another Language From MATLAB



- C/C++, Python, Java
- Fortran
- COM components and ActiveX® controls
- RESTful, HTTP, and WSDL web services

# How to develop Autonomous algorithms?

## System objects

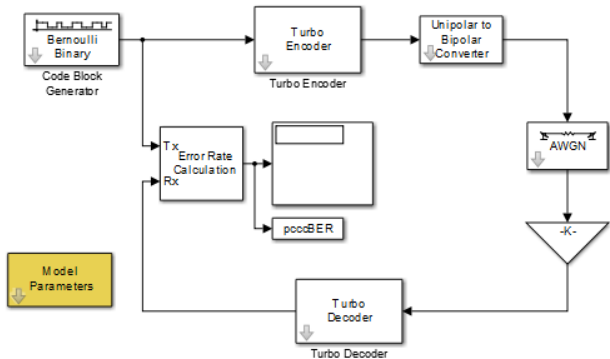
- Model dynamic systems using the MATLAB language
  - Algorithms implemented as MATLAB classes
  - API designed specifically for simulation
  - Can be used in MATLAB and Simulink

**System objects**



```

99
100 % Simulate the modified SUI-1 channel model.
101
102 Nsamp = 1.5e6; % Number of channel samples
103 Nsamp_f = 1e3; % Number of samples per frame
104 Nframes = Nsamp/Nsamp_f; % Number of frames
105 chanOut = zeros(Nsamp, Nr);
106 pathGains = zeros(Nsamp, length(tau), Nt);
107 for iFrames = 1:Nframes
108     inputSig = modulate(hModem, randi(S, [0 M-1], Nsamp_f, Nt));
109     idx = (1:Nsamp_f)+(iFrames-1)*Nsamp_f;
110     [chanOut(idx,:), pathGains(idx, :, :)] = step(h, inputSig);
111 end
    
```



# How to develop Autonomous algorithms?

## System objects

```
RemoveMean.m  x  +
1  classdef RemoveMean < matlab.System
2      %RemoveMean Remove estimated running mean
3
4  properties
5      % Memory weighting
6      Weight = 0.999
7  end
8
9  properties (DiscreteState)
10     Mean
11 end
12
13 methods (Access=protected)
14     function y = stepImpl(obj,u)
15         y = u - obj.Mean;
16         obj.Mean = u - y*obj.Weight;
17     end
18     function setupImpl(obj)
19         obj.Mean = 0;
20     end
21     function resetImpl(obj)
22         obj.Mean = 0;
23     end
24 end
25
26 end
```

Base Class is:  
matlab.System

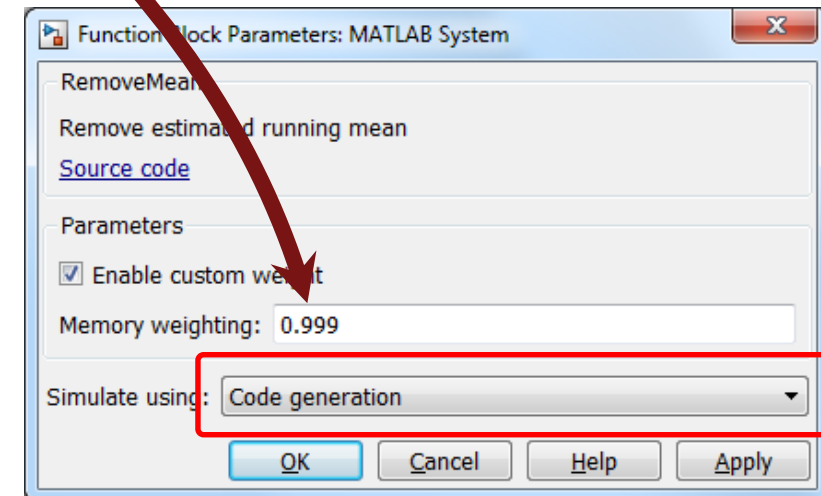
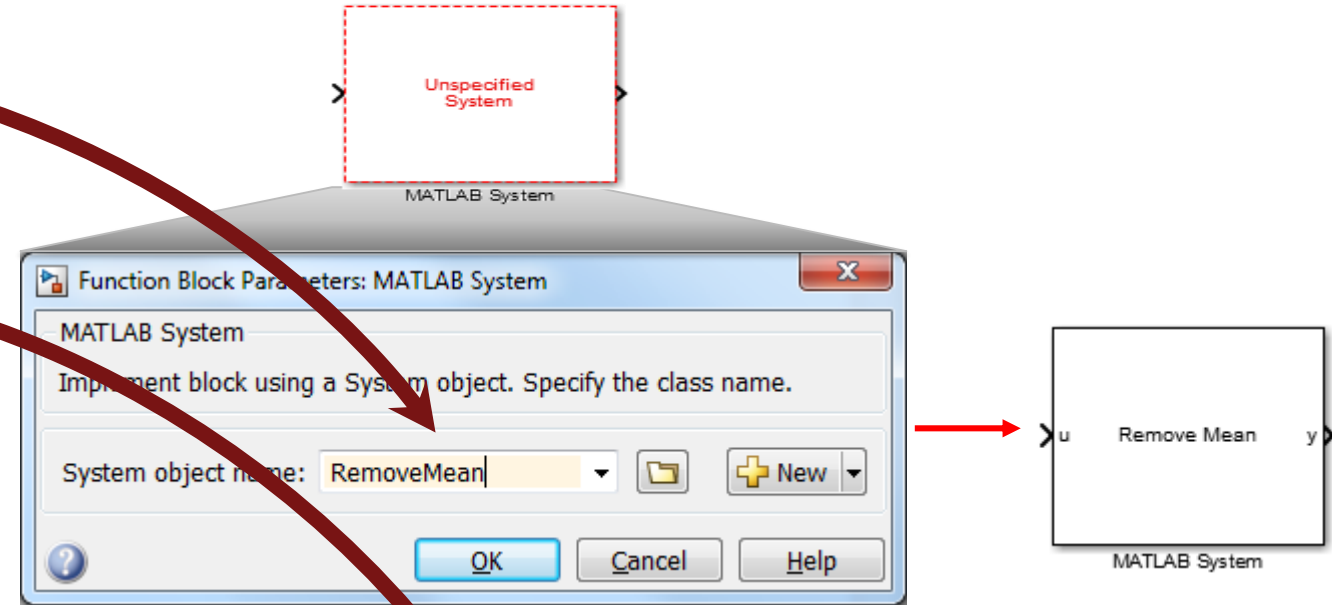
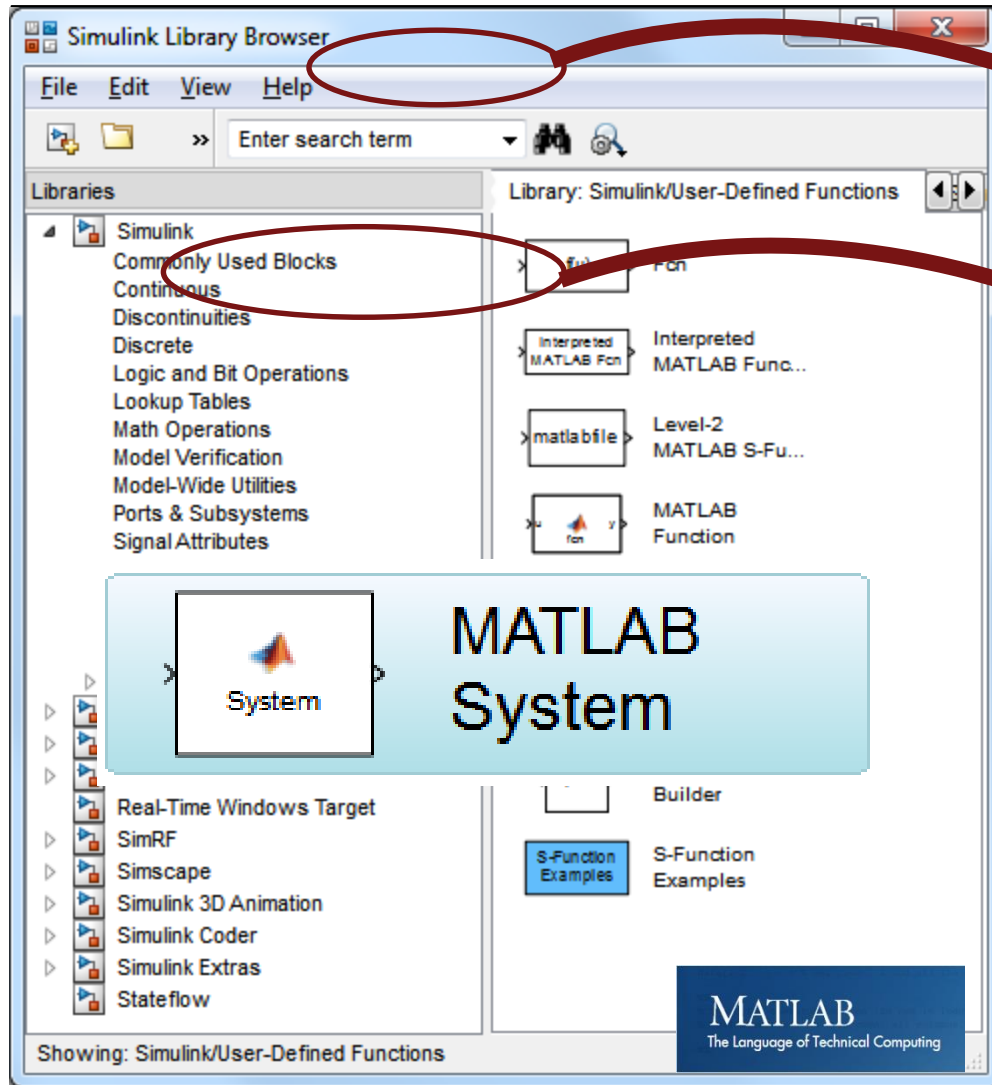
Properties provide persistent  
memory within the object

Functions for each phase of your  
system (validation, initialization,  
reset, step, ...)

- MATLAB automatically calls  
housekeeping functions,  
e.g., validation, initialization, etc.

# How to develop Autonomous algorithms?

## System objects

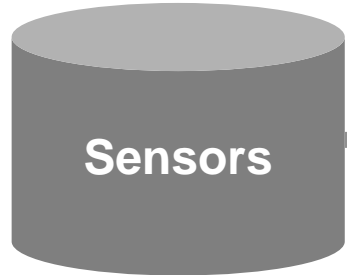




# How to develop Autonomous algorithms?

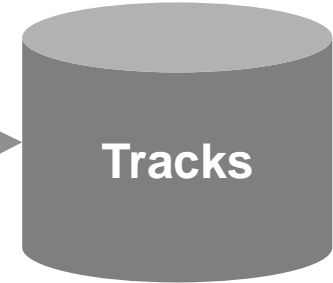
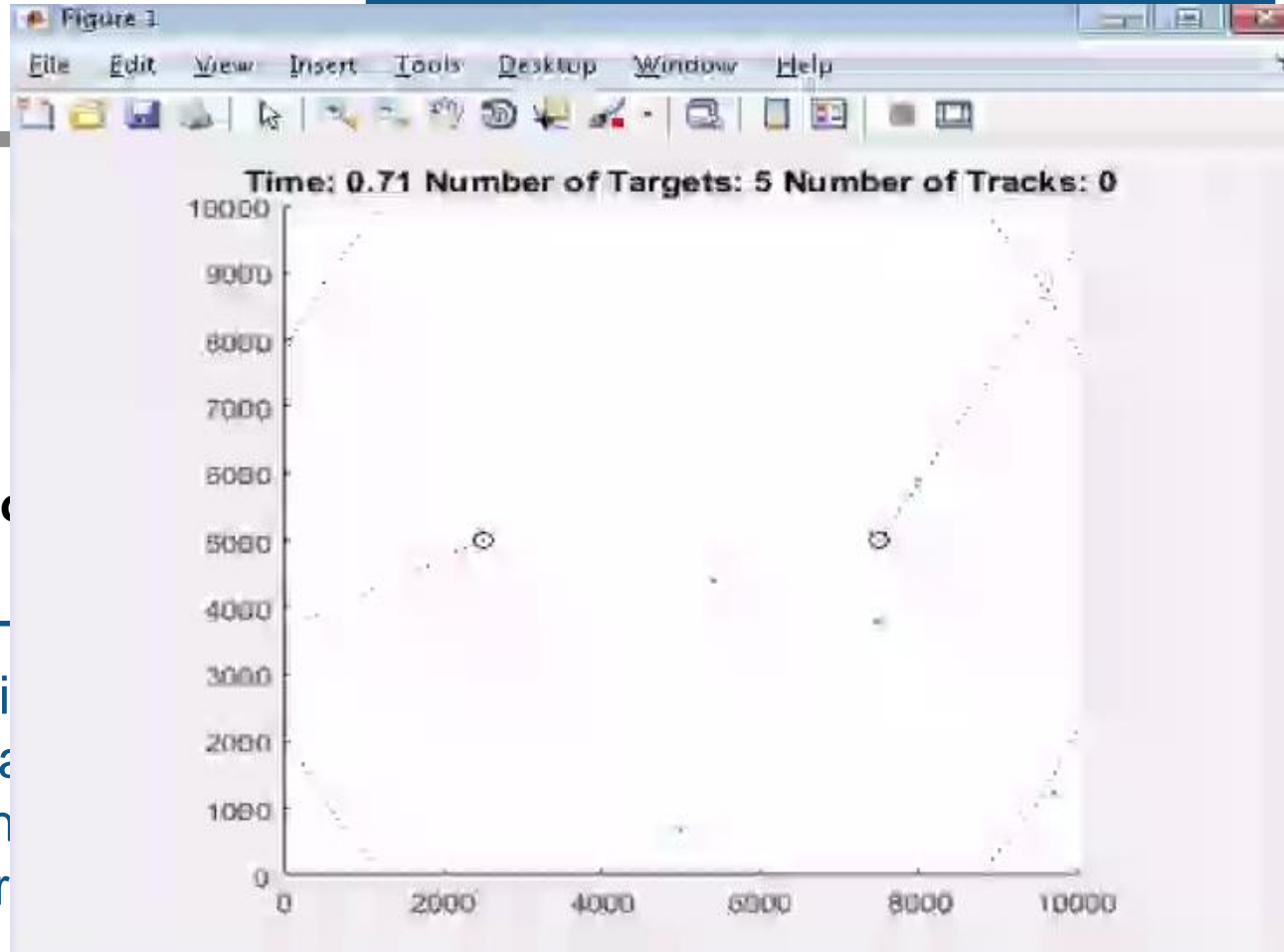
## Sensor fusion framework

### Sensor fusion Framework



Time  
Measurement  
Measurement No

- Assigns detection
- Creates new track
- Updates existing
- Removes old track

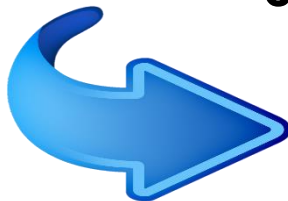
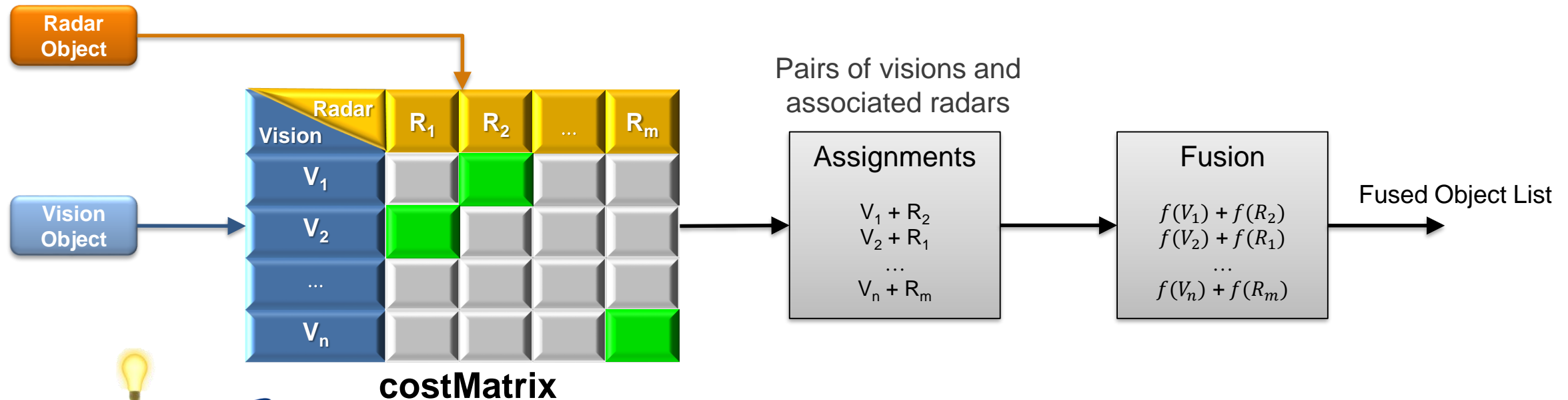


Time  
State  
State Covariance  
Track ID  
Age  
Is Confirmed  
Is Coasted

# How to develop Autonomous algorithms?

## Sensor fusion framework

- Track Manager

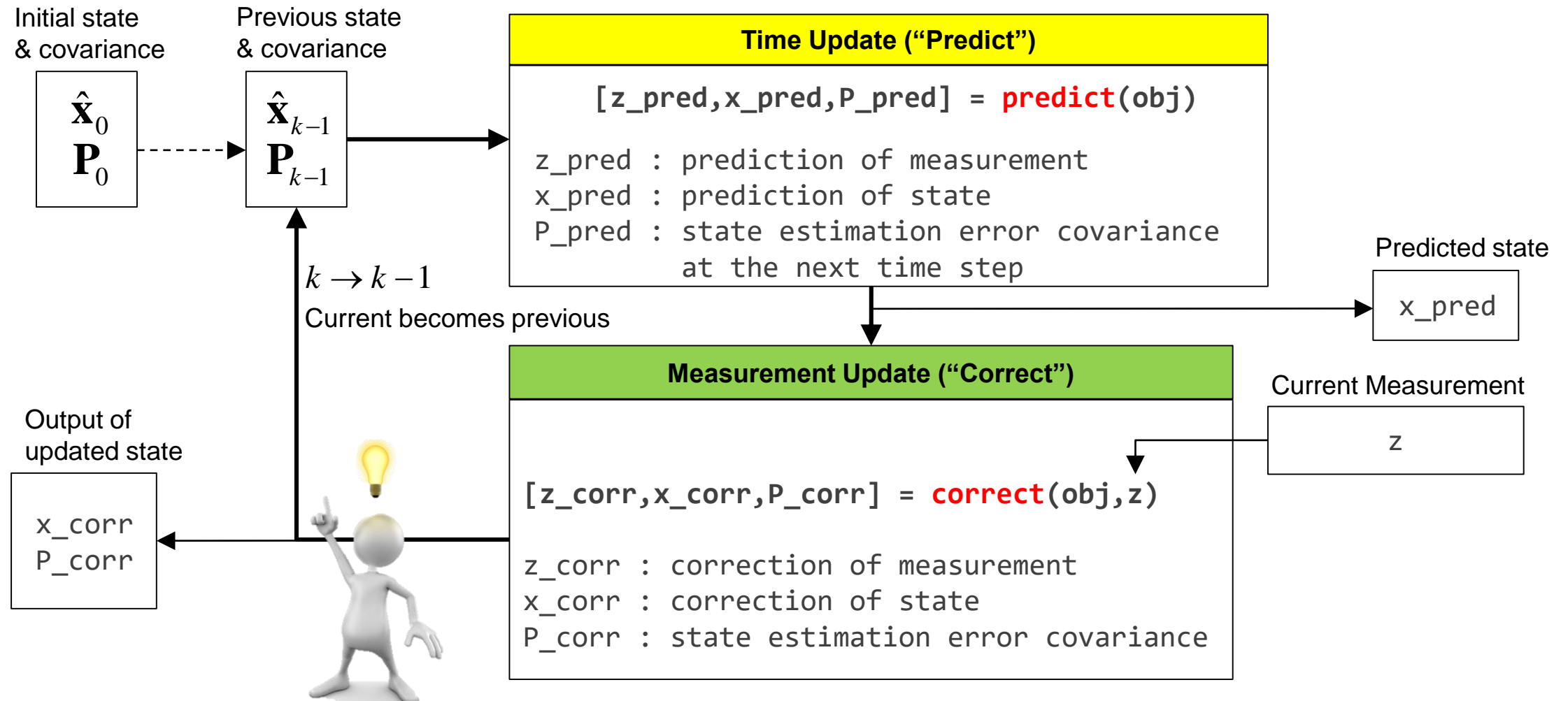


```
[assignments, unassignedVisions, unassignedRadars] = ...
    assignDetectionsToTracks(costMatrix, param.costOfNonAssignment);
```

# How to develop Autonomous algorithms?

## Sensor fusion framework

- Tracking Filter



# How to develop Autonomous algorithms?

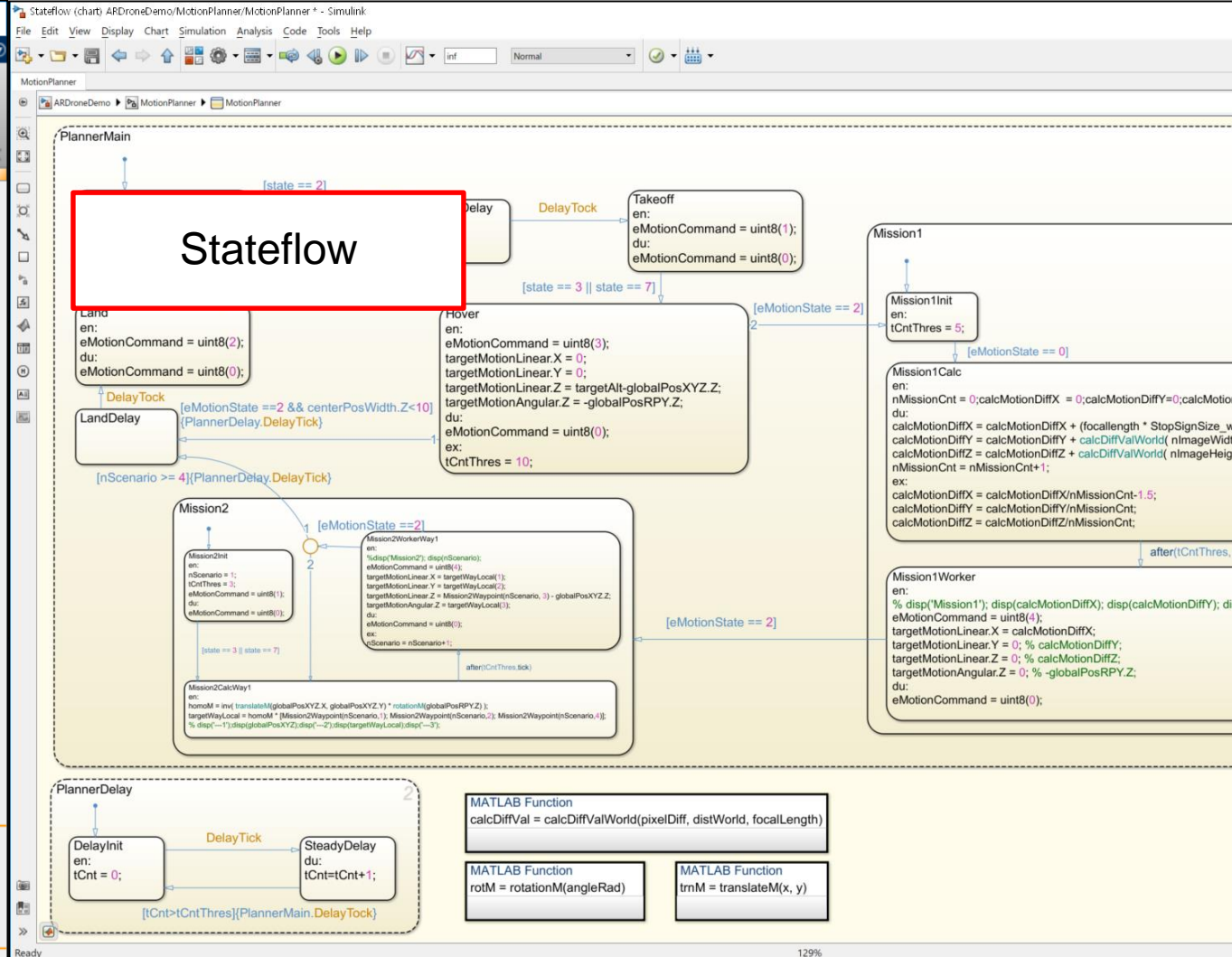
## System Level Design with MATLAB, Simulink

```

1 classdef SignDetector < matlab.System & matlab.system.mixin.Propagates ...
2   & matlab.system.mixin.CustomIcon
3   % Sign Detector System Object
4   %
5   % NOTE: When renaming the class name Untitled2, the file name
6   % and constructor name must be updated to use the class name.
7   %
8   % This template includes most, but not all, possible properties, attributes,
9   % and methods that you can implement for a System object in Simulink.
10  %
11  % Public, tunable properties
12  properties
13  end
14  end
15  % Public, non-tunable properties
16  properties(Nontunable)
17  imDir = 'unclassifiedData'; % Unclassified data directory name.
18  collectTrainingImages = false; % Save images for training
19  recordFlag = false; % Record video
20  classifierMATFile = 'trainedClassifier'; % Trained classifier MAT File Name.
21  end
22  end
23  properties(DiscreteState)
24  end
25  end
26  % Pre-computed constants
27  properties(Access = private)
28  outputBusName = 'stopSignBus_bbox';
29  dateTxt = []; % string to save current date and time
30  classifier = []; % trained classifier
31  outputImage = []; % image with results from object detection
32  BW = []; % black and white mask from image.
33  end
34  end
35  end
36  end
37  methods
38  % Constructor
39  function obj = SignDetector(varargin)

```

System Object



# Design (and verify) Autonomous algorithms

The image displays a multi-windowed software environment for autonomous system development and simulation. The main window is Gazebo, a 3D physics engine, showing a simulated environment with a grey ground plane, a black quadrotor, a red stop sign, and a concrete barrier. The Gazebo interface includes a menu bar (File, Edit, Camera, View, Window, Help), a toolbar, and a hierarchical tree on the left listing models like 'link', 'House 3', 'House 2', 'jersey\_barrier', 'grey\_wall', 'mist\_stairs\_120', 'mist\_elevated\_floor\_120', 'mist\_maze\_wall\_240', 'mist', 'quadrotor', and 'stop\_sign'. A 'Property' panel on the left shows details for the selected 'ground\_plane' model, including 'name', 'is\_static', 'self\_collide', and 'pose'.

Overlaid on the Gazebo window is a Simulink window titled 'reprocNode - Simulink'. The Simulink diagram is divided into several functional blocks:
 

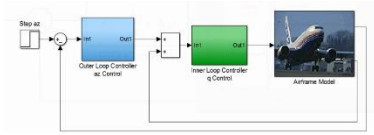
- Sensor Communication Layer:** Contains ROS 'Subscribe' blocks for 'image\_raw' and 'image', and a 'Set Pace' block.
- Application Layer:** Features an 'ImageProc' block that receives image data and outputs 'bFlagSend\_tbox' and 'flag\_tbox'. It also includes a 'ConvertROSMag' block.
- Status Monitor:** Includes 'Artificial Horizon' and 'Monitor' blocks that process sensor data into visual displays like 'Display\_Horizon' and 'Display\_Heading'.
- Control UI:** A panel with 'On/Off' buttons for 'Add StopSign' and 'Add Via points'.
- Command Communication Layer:** Contains ROS 'Publish' blocks for 'bFlagSend\_tbox' and 'StopSignBox'.

At the bottom of the Simulink window, a status bar shows 'Running', 'View diagnostics 96% T=1.500', and 'auto(FixedStepDiscrete)'. The Gazebo window's bottom status bar displays 'Real Time Factor: 0.63', 'Sim Time: 00:00:11.44.949', 'Real Time: 00:00:01.12.076', 'Iterations: 22602', and 'FPS: 20.8298'.

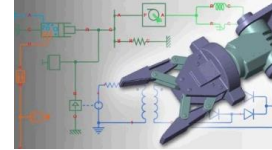
# How to develop Autonomous algorithms?

CONTROL  
 SENSE  
 PERCEIVE  
 PLAN  
 CONNECT

**Control System Tbx**



**Simscape Toolboxes**



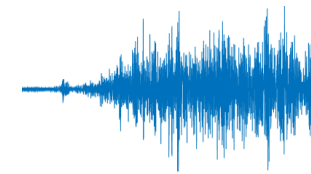
**Simulink Real-Time**



**HW Support Packages**



**Data Acquisition Tbx**



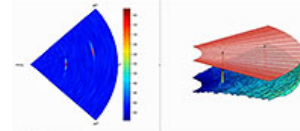
**Computer Vision**



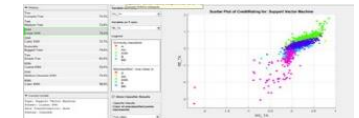
**Automated Driving System Tbx**



**Phased Array**



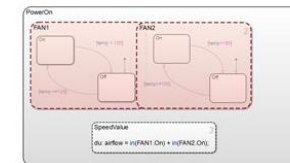
**Statistics & Machine Learning**



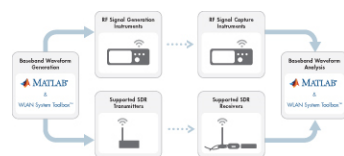
**Robotics System Tbx**



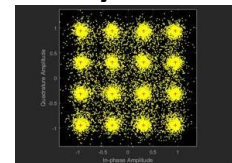
**Stateflow**



**Communications Tbx**



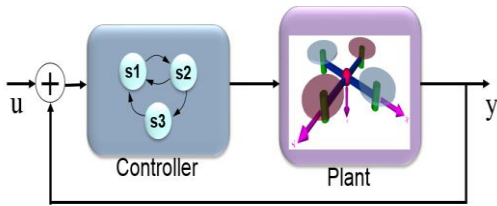
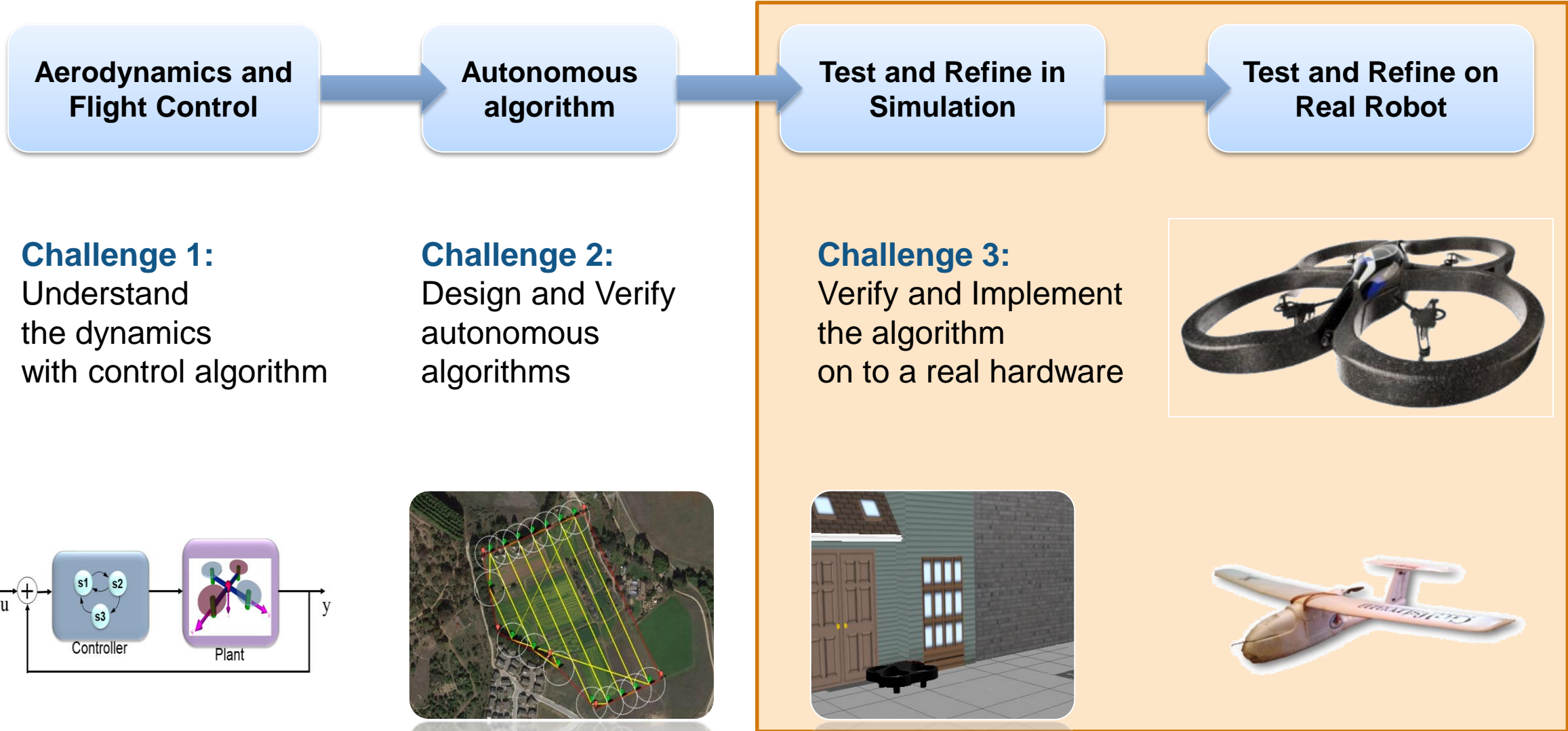
**WLAN System Toolbox**



**Robotics System Tbx**



# Autonomous System Development Workflow



# How to **Deploy** autonomous algorithm?

## **MATLAB Coder** - Code from MATLAB

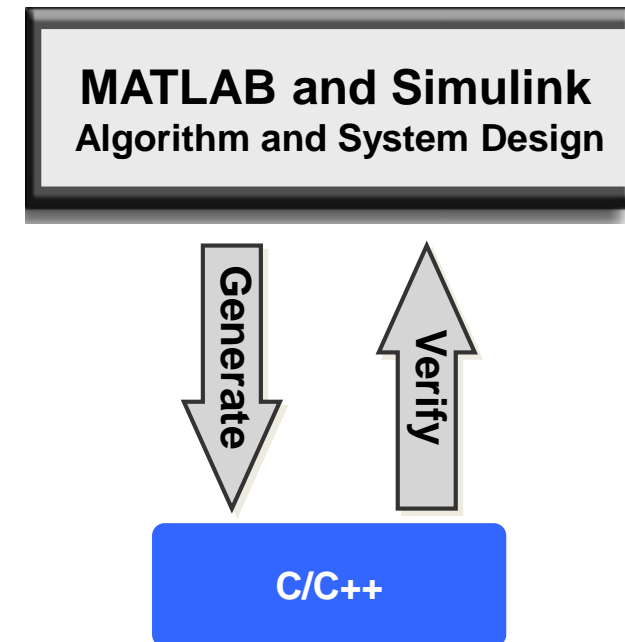
- Portable code for numerical algorithms
- Desktop applications (standalone, library)

## **Simulink Coder** - Code from Simulink

- Rapid prototyping or HIL applications
- Real-time machines

## **Embedded Coder** – Production optimized code

- Embedded applications
- MCU and DSP (fixed or float)
- Code verification (in-the-loop)
- Target-specific support (APIs and examples)



**All coders generate portable code (ANSI/ISO C) by default.**



# How to **Deploy** autonomous algorithm?



# Autonomous System with MATLAB/Simulink

## BAE Systems Controls Develops Autopilot for Unmanned Aerial Vehicle Using MathWorks Tools

### Challenge

Enable teams working in separate locations to design a sophisticated UAV autopilot system quickly and inexpensively

### Solution

Use MathWorks tools, modify existing software designs with Model-Based Design, and automatically generate embedded control code

### Results

- Design and rework costs substantially reduced
- Testing cycle time minimized
- Coding errors and manual documentation work minimized



**An Eagle 150 unmanned aerial vehicle flight.**

(Image courtesy of Composites Technology Research Malaysia.)

**“MATLAB and Simulink greatly reduced development cycle time and cut system software design and testing costs by 50%.”**

**Feng Liang  
BAE Systems Controls**

# Autonomous System with MATLAB/Simulink

## Airnamics Develops Unmanned Aerial System for Close-Range Filming with Model-Based Design

### Challenge

Design and develop an unmanned aerial camera motion system for close-range aerial filming

### Solution

Use Model-Based Design with MATLAB and Simulink to accelerate the design, debugging, and implementation of the vehicle's fly-by-wire and flight management system software

### Results

- Time-to-market shortened by up to an order of magnitude
- Test flight anomalies quickly resolved
- Debugging time reduced from weeks to hours



Airnamics co-founders Marko Thaler and Zoran Bjelić with the R5 MSN1 prototype after its first flight.

**“With Model-Based Design our three-engineer team found more than 95% of control software bugs before the first flight. We used the test flights to increase our Simulink models’ fidelity and isolate remaining bugs with high precision. The result is a safer, more reliable, and higher-quality product.”**

Marko Thaler  
Airnamics

# Summary of Aerial Autonomous System

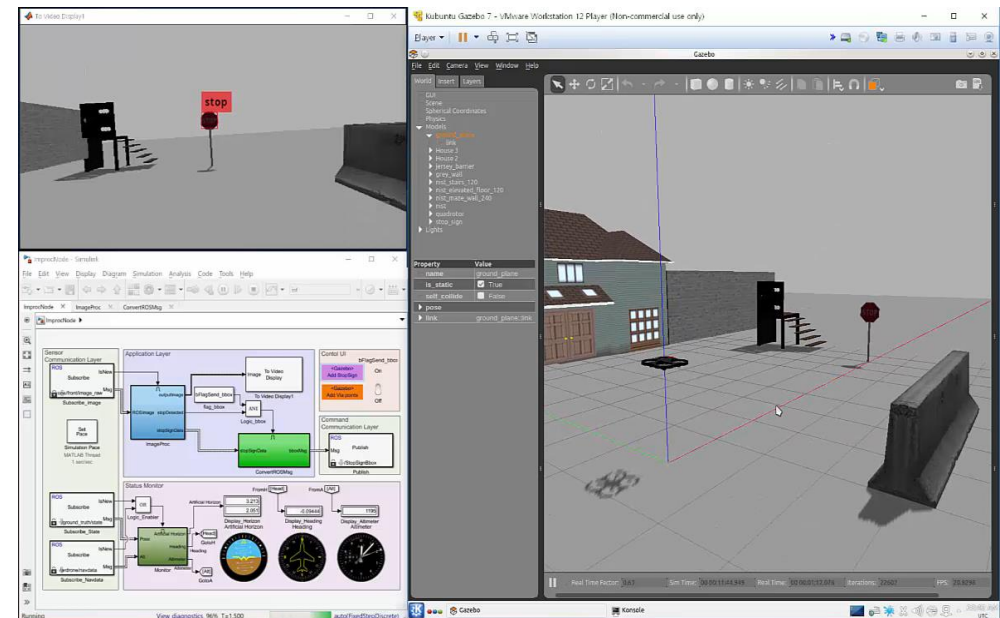
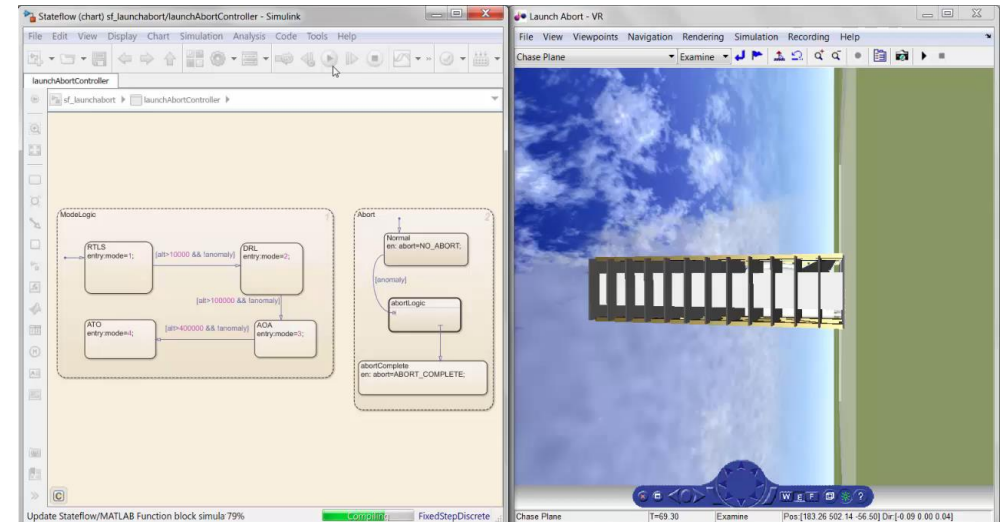
Aircraft	<i>“Autonomous Algorithms”</i>				Communication
	Perception	Perception	Planning	Planning	
Platform	Sense	Perceive	Plan & Decide	Control	Connect/ Communicate
<ul style="list-style-type: none"> <li>Control Surfaces, slats, flaps</li> <li>Lifting Body</li> <li>Landing Gear</li> <li>Battery</li> <li>Power Management</li> </ul>	<ul style="list-style-type: none"> <li>Radar</li> <li>Camera</li> <li>Lidar</li> <li>EO/IR</li> <li>IMU</li> <li>GPS-INS</li> <li>HW Certification</li> </ul>	<ul style="list-style-type: none"> <li>Environment mapping</li> <li>Classification</li> <li>Segmentation</li> <li>Object Detection</li> <li>Sensor Fusion</li> </ul>	<ul style="list-style-type: none"> <li>Object Avoidance</li> <li>Path &amp; motion planning</li> <li>SLAM</li> </ul>	<ul style="list-style-type: none"> <li>Guidance, Navigation &amp; Control</li> <li>Flight SW certification</li> </ul>	<ul style="list-style-type: none"> <li>Communication with ground operator</li> <li>Multi-agent communication</li> <li>Satellite data link</li> </ul>



# Key Takeaway

## Designing Autonomous system using MATLAB and Simulink can help in :

- **Understanding the dynamics with 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



% Thank you

# MATLAB

is the **easiest** and  
most **productive** environment  
for **engineers** and **scientists**