MATLAB EXPO 2018
Developing Algorithms for Robotics and Autonomous Systems

Jorik Caljouw
Key Takeaway of this Talk

Success in developing an autonomous robotics system requires:

1. Multi-domain simulation
2. Trusted tools which make complex workflows easy and integrate with other tools
3. Model-Based Design
Challenges with Autonomous Robotics Systems

- Applying Multidomain Expertise
- Complexity of Algorithms
- End-to-End workflows
- Technical Depth and System Stability
- IP Protection
What does success look like?
Another Example: Self-Driving Cars

Planning → Following

Localization → Obstacle avoidance

Global Map

Deep learning

Perceive → Communication → Plan

Sense

RADAR

Camera

LIDAR

GPS/IMU

Platform

Control

Steering

Accelerator Braking

Actuator ECUs

Motion Controllers
Today: Design Pick and Place Application

Platform → Sense

Sense → Perceive

Perceive → Plan & Decide

Plan & Decide → Control
Today: Design Pick and Place Application

- Platform
- Sense
- Perceive
- Plan & Decide
- Control

Diagram:
- User_Command
  - userCommand
  - image visionObjects
  - Kinect_2, Pose_Estimator
  - visionObjects
  - Command
  - robotState
  - Motion_Planner
  - \( \frac{1}{z} \)
Platform Design

How to create a model of my system that suits my needs?
Mechanics: Import models from common CAD Tools

SolidWorks Model

Simscape Multibody Model
Mechanics: One line import from URDF

```matlab
% Import robot from URDF
smimport('j2n6s300_standalone_stl.urdf');
```
Rigid Body Tree Dynamics

Compute rigid body tree dynamics quantities

- Specify rigid body inertial properties
- Compute for the rigid body tree
  - Forward dynamics
  - Inverse dynamics
  - Mass matrix
  - Velocity product
  - Gravity torque
  - Center of mass position and Jacobian

```matlab
» load exampleRobots.mat
» lbr.DataFormat = 'column';
» q = lbr.randomConfiguration;
» tau = inverseDynamics(lbr, q);
```
Actuators: Model other domains
Environment: Connect to an external robotics simulator
Environment: Connect MATLAB and Simulink with ROS

- MATLAB Code
- SM Models
- Built-in algorithms
- Networking
- Code Generation
- ROS Bag import
- ROS node
- Simulation environment
- Robot
Design Pick and Place Application

Platform

Sense

Perceive

Plan & Decide

Control
Design Pick and Place Application

- Support for Common Sensors
- Image analysis
- Apps
- Image enhancement
- Visualizing Point Clouds
Design Pick and Place Application

Platform

Sense

Perceive

Plan & Decide

Control
Object Classifier and Pose Estimator

Images

Pose Estimator

Labels and Poses

Object 1

Object 2

Object 3

Object 4
MATLAB makes machine learning easy and accessible

Traditional Machine Learning approach

Traditional Feature Extraction  Classification

Boy  
Dog  ✓
Bicycle  

Deep Learning approach

Convolutional Neural Network (CNN)

Learned features [95%]

End-to-end learning

Feature learning + Classification

Dog  ✓
Boy  
Bicycle  
% Detect regions
BW = createMask(videoFrame);

% Fill image regions
BW = imfill(BW,'holes');

% Get bounding boxes
stats = regionprops('table',BW,'BoundingBox','Area');

% Filter based on area size
targetIndex = stats.Area > 500;

% Get bounding boxes from detected regions
testFeatures(k,:) = extractHOGFeatures(Icr);
Design Pick and Place Application

Platform

Sense

Perceive

Plan & Decide

Control

- Kinect_2
- Pose_Estimator
- Image visionObjects
- User_Command
- userCommand
- userCommand
- visionObjects
- Command
- robotState
- Motion_Planner

\[ \frac{1}{z} \]
Planning: Find a path

Map
Initial Pose
Final Pose

Path Planner

Path

\[ [x_a \ y_a \ \theta_a] \]

\[ [x_b \ y_b \ \theta_b] \]
Plan with Stateflow

Gripper Control

Joint Control

Inverse Kinematics Control

Waypoint Control
Design Pick and Place Application

Platform

Sense

Perceive

Plan & Decide

Control

Platform

Sense

Perceive

Plan & Decide

Control

Design Pick and Place Application
Explore Built In Functions: Inverse Kinematics

% Create ik solver object
ik = robotics.InverseKinematics('RigidBodyTree',jaco2n6s300)
% Disable random restarts
ik.SolverParameters.AllowRandomRestart = false;
% Parameters to pass to the solver
weights = [1, 1, 1, 1, 1, 1, 1];
q_init = 0.1*ones(numel(q_home),1);
Key Takeaway of this Talk

Success in developing an autonomous robotics system requires:

1. Multi-domain simulation
2. Trusted tools which make complex workflows easy and integrate with other tools
3. Model-Based Design
German Aerospace Center (DLR) Robotics and Mechatronics Center Develops Autonomous Humanoid Robot with Model-Based Design

Challenge
Develop control systems for a two-armed mobile humanoid robot with 53 degrees of freedom

Solution
Use Model-Based Design with MATLAB and Simulink to model the controllers and plant, generate code for HIL testing and real-time operation, optimize trajectories, and automate sensor calibration

Results
- Programming defects eliminated
- Complex functionality implemented in hours
- Advanced control development by students enabled

“Model-Based Design and automatic code generation enable us to cope with the complexity of Agile Justin’s 53 degrees of freedom. Without Model-Based Design it would have been impossible to build the controllers for such a complex robotic system with hard real-time performance.”

- Berthold Bäuml, DLR
Clearpath Robotics Accelerates Algorithm Development for Industrial Robots

Challenge
Shorten development times for laser-based perception, computer vision, fleet management, and control algorithms used in industrial robots

Solution
Use MATLAB to analyze and visualize ROS data, prototype algorithms, and apply the latest advances in robotics research

Results
- Data analysis time cut by up to 50%
- Customer communication improved
- Cutting-edge SDV algorithms quickly incorporated

“ROS is good for robotics research and development, but not for data analysis. MATLAB, on the other hand, is not only a data analysis tool, it’s a data visualization and hardware interface tool as well, so it’s an excellent complement to ROS in many ways.”

- Ilia Baranov, Clearpath Robotics
Voyage develops longitudinal controls for self-driving taxis

Challenge
Develop a controller for a self-driving car to follow a target velocity and maintain a safe distance from obstacles

Solution
Use Simulink to design a longitudinal model predictive controller and tuned parameters based on experimental data imported into MATLAB using Robotics System Toolbox. Deploy the controller as a ROS node using Robotics System Toolbox. Generate source code using MATLAB Coder into a Docker Container.

Results
- Development speed tripled
- Easy integration with open-source software
- Simulink algorithms delivered as production software

“We were searching for a prototyping solution that was fast for development and robust for production. We decided to go with Simulink for controller development and code generation, while using MATLAB to automate development tasks.”
- Alan Mond, Voyage
Preceyes Accelerates Development of World’s First Eye-Surgery Robot Using Model-Based Design

Challenge
Develop a real-time control system for robot-assisted surgical procedures performed within the human eye

Solution
Use Model-Based Design with MATLAB and Simulink to model and simulate the control system and use Simulink Coder and Simulink Real-Time to deploy it to a real-time target

Results
- Development Core controller developed by one engineer
- Patient safety assured
- Road map to industrialization set

“MATLAB and Simulink provided a single platform that supported our complete workflow and all the components and protocols we needed for our robotic system. That enabled us to quickly develop a safe, real-time device, ready for clinical investigation.”
- Maarten Beelen, Preceyes