Autonomous Navigation using Model Based Design

MATLAB EXPO, San Jose
November 6, 2018

Pulkit Kapur
Industry Lead– Robotics and Autonomous Systems

Carlos Santacruz-Rosero
Application Lead– Robotics and Autonomous Systems
Components of Autonomous Systems

- Localization
- Perception
- Planning
- Control
Key Takeaway of this Talk

Success in developing an autonomous robotics system requires:

1. Multi-domain simulation with newer technologies
2. Trusted tools which make complex workflows easy and integrate with other tools
3. Model-based design provides the flexibility for changing development needs
Task: Autonomous Delivery Robot
Autonomous Delivery Workflow
Navigation Stack – Key Components

Sensing & Perception
Mapping
Planning & Decision making
Localization & Control

Physical System + Environment

Key Components for Enabling Full Autonomy
% Detect regions
BW = createMask(videoFrame);

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

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

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

% Get bounding boxes from detected regions
testFeatures(k,:) = extractHOGFeatures(Icr);
Deep Learning with MATLAB

“How do I label my data?”
New App for Ground Truth Labeling
Label pixels and regions for semantic segmentation

“How do I access the latest models?”
Caffe model importer
LSTM (time series, text)
DAG Networks
Library of pretrained models

“How do I make training and prediction faster?”
Multi-GPUs in parallel
Optimized GPU code
Training plots

“How do I deploy my new model?”
NEW PRODUCT:
GPU Coder-
Convert to NVIDIA CUDA code

Data
Models
Train / Predict
Deploy / Share
Navigation Stack – Key Components

Sensing & Perception

Mapping

Planning & Decision making

Localization & Control

Physical System + Environment

Key Components for Enabling Full Autonomy
Simultaneous Localization and Mapping (SLAM)

Robotics System Toolbox™

R2018a
Lidar SLAM Components

**Front-End**
- Sensor Data
  - Drifts over time
  - "Loop closure detection"
- Estimate robot motion locally
- Recognize previously visited places

**Sensor Specific**

**Back-End**
- Globally re-align all scans

**Sensor Agnostic**

**SLAM Estimate**

**Drifts over time**
SLAM Map Builder App

- Build 2D map of environment based on Lidar and odometry data

- Modify loop closures and incremental scan matches to improve map quality

- Export the resulting occupancy grid and use for path planning
Integrate your Own Sensors for Custom SLAM Implementation

- Use your own **custom sensor processing**
- **Re-use infrastructure** for building and optimizing maps
Navigation Stack – Key Components

- Sensing & Perception
- Mapping
- Planning & Decision making
- Localization & Control

Key Components for Enabling Full Autonomy
Path Planning

- Optimization or search problem

- Map representation is needed

- Data structures like graphs, heaps, and queues

- Post-processing is sometimes required: smoothing
Plan Paths for Nonholonomic Vehicles using RRT*
Navigation Stack – Key Components

Sensing & Perception

Mapping

Planning & Decision making

Localization & Control

Physical System + Environment

Key Components for Enabling Full Autonomy
Localization System using Model Based Design

Monte Carlo Localization (MCL)

ROS as Communication Framework
Control - Path Following

Map
Initial Pose
Final Pose

Path Planner

Path

Path Follower

Steering
Acceleration

$[x_a \ y_a \ \theta_a]$
Controls: Path Following

- Control Problem (feedback)
- Map representation (most of the time)
- Suited for graphical programming
- State machine for Supervisory logic

Angular velocity to go from current position to the lookahead
Path Controller

We need a vehicle model to design and test our algorithm
Pure Pursuit in Action
Complete System
### Table of Features for Autonomous Navigation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Application Area</th>
<th>MATLAB Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLAM</td>
<td>Ground Robots, ADAS, UAVs</td>
<td>robotics.LidarSLAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>robotics.PoseGraph</td>
</tr>
<tr>
<td></td>
<td></td>
<td>robotics.PoseGraph3D</td>
</tr>
<tr>
<td>Localization</td>
<td>All Autonomous Systems</td>
<td>robotics.MonteCarloLocalization</td>
</tr>
<tr>
<td>Scan Matching</td>
<td>ADAS, Ground Robots</td>
<td>matchScans</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matchScansGrid</td>
</tr>
<tr>
<td>Point Cloud Registration</td>
<td>ADAS, Computer Vision</td>
<td>pcregrigid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pcregistericp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pcregisterndt</td>
</tr>
<tr>
<td>Estimation Filters</td>
<td>All Autonomous Systems</td>
<td>trackingKF, trackingEKF, trackingUKF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>robotics.ParticleFilter</td>
</tr>
<tr>
<td>Path Planning</td>
<td>All Autonomous Systems</td>
<td>robotics.PRM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pathPlannerRRT(parkMap)</td>
</tr>
</tbody>
</table>
Third-Party Simulator Integration

- Use co-simulation with **third-party simulators** for rich sensor and environment simulation

Gazebo

Unreal
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

Link to user story
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 Simulink 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
Key Takeaway of this Talk

Success in developing an autonomous robotics system requires:

1. Multi-domain simulation with newer technologies
2. Trusted tools which make complex workflows easy and integrate with other tools
3. Model-based design provides the flexibility for changing development needs
Thank you

mathworks.com/robotics

Come talk to us at the Autonomous Navigation Booth!