ホワイトペーパー

MATLAB および Simulink による自律移動ロボット開発

MathWorks®

はじめに

自律移動ロボットが必要とされている背景

近年、人手不足が叫ばれている中、労働力不足の解消にロボットの自律化が求められています。特に、工場内の搬送作業や施設の見回り、清掃など移動作業を伴うタスクをロボットで代替させる動きが加速しています。

自動ロボットは従来、白線や磁気テープなどの誘導方式を用いた限定された空間での適用が主流でしたが、より汎用的な移動ロボットを構築するには人間同様に環境を認識し、自ら判断・意思決定を行うことが不可欠です。

本ホワイトペーパーではこのような自律移動ロボット開発の課題や使用されている技術について解説し、どのように MATLAB®および Simulink®が活用できるかをご紹介していきます。

自律移動ロボットの応用例

自律移動ロボットの応用例は多岐にわたり、自動車、産業機器からコンスーマー機器まで様々です。下記のような幅広い業界に渡って自律移動ロボットの技術の活用が期待されています。

- ADAS/自動運転
- お掃除ロボット
- 芝刈りロボットによる効率的な芝のメンテンナンス
- 工場内・生産ラインでの無人搬送車・無軌道型 AGV の自律運転
- 探査機・ローバーによる未知の惑星探査
- 介護サービスロボットによる訪問・見守り
- 警備サービスロボットによる巡回

自律移動ロボットシステムの概要

自律移動ロボットの概要

自律移動ロボットの実現には従来の制御開発だけでなく物体検出や状況認識、判断・意思決定、車両制御など複合的なアルゴリズムの組み合わせが必要となります。それらのアルゴリズムは閉ループのフィードバックシステムによって密に結合し、システムの挙動を複雑にしています。

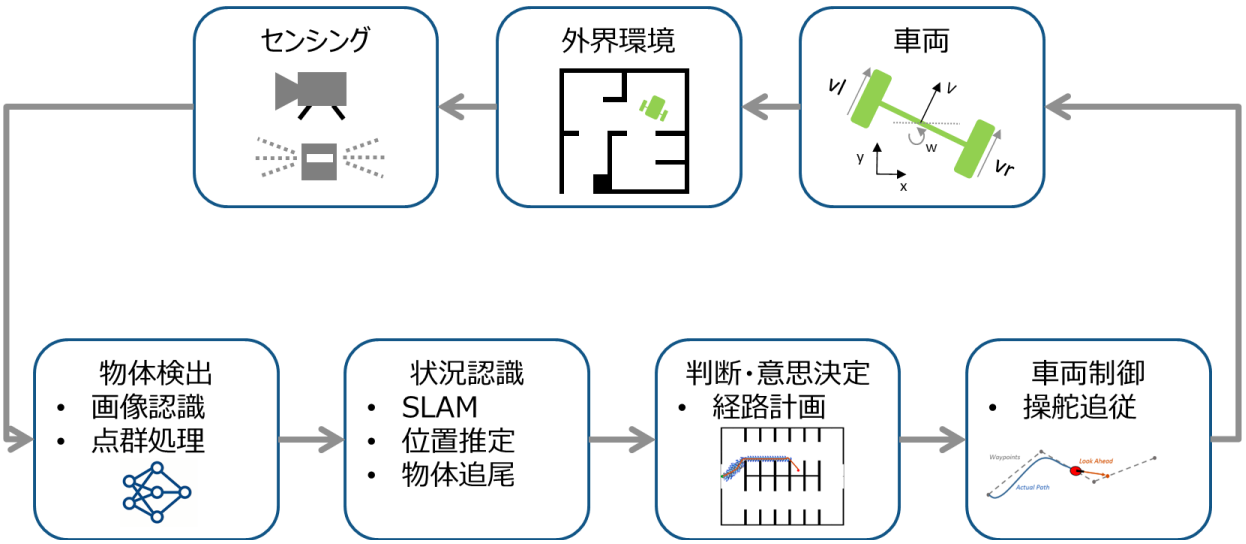

図 1 自律移動ロボットシステム

従来の移動ロボット

従来までの移動ロボットは人が目視でロボットの状態を確認しながら、経路を計画し、操作するという手動操縦が主でした。このような従来型の移動ロボットは下記のような特徴があります。

- 単一ループによる同期処理が基本
- 高いリアルタイム性が要求されるタスクが多い
- センサーは 1 次元の時系列信号であることが多い
- アクチュエーターは単一もしくは少数

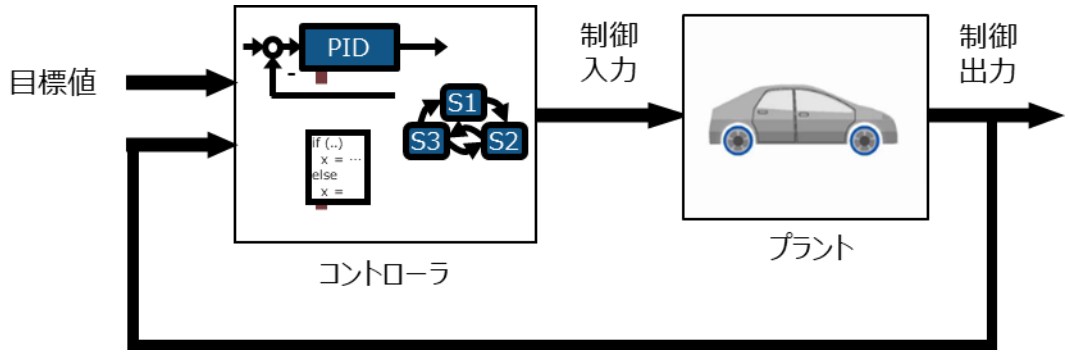

図 2 従来の移動ロボットのアーキテクチャ

自律移動ロボットのチャレンジ

一方、自律移動ロボットは従来の移動ロボットと比べると複雑度が増しています。従来、人間が行っていた高度な認知判断をロボット自身が行う必要があります。高度な認知判断の実現にはディープラーニングによる物体認識から Lidar 点群処理、センサーフュージョンによる障害物追尾、SLAM による地図生成、パスプランニング、経路追従制御など複合的な技術の組み合わせが不可欠です。

このような自律移動ロボットの実現には下記のようなチャレンジがあります。

- 画像処理、信号処理、制御など複数の領域の技術の組み合わせが必要
- 複数種類のセンサー、アクチュエーターの組み合わせが必要
- 高解像度、多次元のセンサー処理が必要(分散処理・並列処理が不可欠)
- センサー間の時刻同期やセンサーフュージョンが必要
- センサー、アクチュエーターなどの座標管理(フレーム管理)が煩雑
- センサー、アクチュエーターごとに複数のレートの管理が必要 (同期処理・非同期処理の混在)

これらのチャレンジを克服し、自律移動ロボットを効率よく開発していくことが求められています。

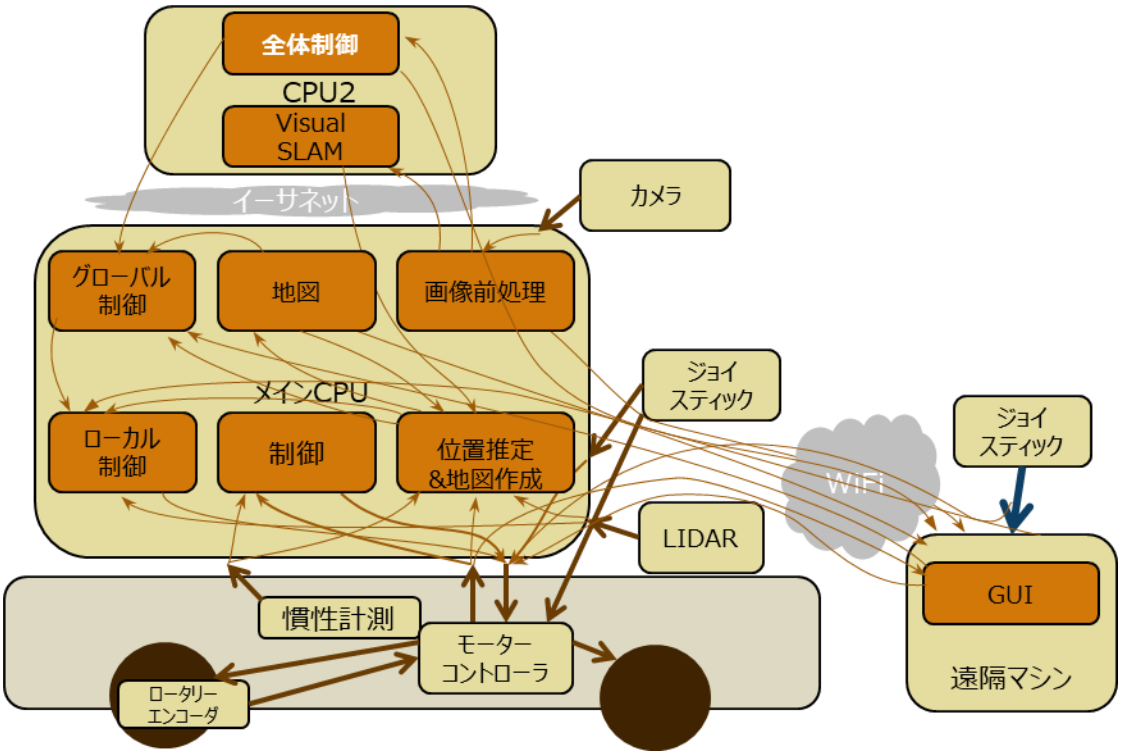

図 3 自律移動ロボットのアーキテクチャ

MATLAB および Simulink による自律移動ロボット開発

MATLAB および Simulink は自律移動ロボットシステムに活用可能な開発検証プラットフォームです。MATLAB および Simulink を有効活用することで下記のような利点があります。

- 複数領域の技術を単一プラットフォームで扱うことができる
- 複合的なセンサー処理、アクチュエーター処理を柔軟に記述可能
- 高解像度、多次元のセンサー処理のマルチコア CPU や GPU による並列化が可能

次章以降で自律移動ロボットのシステムに沿って MATLAB および Simulink および関連 Toolbox がどのように活用できるかを解説します。

図 4 MATLAB および Simulink プロダクトファミリー

関連リソース

- **武蔵精密工業株式会社、工場内の自律搬送ロボットを半年間で試作**

ハードウェア プラットフォームの設計

自律移動ロボットは下記 2 つのステップが必要となります。

1. ベースとなるハードウェアプラットフォームの構築
2. ハードウェアプラットフォームにセンサーやコンピューティングボードを追加し自律化

既存の移動ロボットを自律化する場合には 2 から始めることになりますが、自律移動ロボットのハードウェア自体をはじめから開発する場合には上記 1、2 の両方のステップが必要になります。

移動ロボットのハードウェアプラットフォーム構築

自律移動ロボットを開発するためにはセンサーやコンピューティングボードを搭載するためのベースとなるハードウェアプラットフォームが必要となります。ハードプラットフォームは組み込みボードなどの制御コントローラと電気回路、メカ部品などで複合的な領域の技術で構成されています。ハードウェアプラットフォームの開発にはモデルベースデザインが有効です。モデルベースデザインでは対象のふるまいを示す「モデル（シミュレーション）」を作成し、そのモデルを開発の様々なシチュエーションで活用することで、開発期間の短縮や、ソフトウェアの品質の向上などを実現させる開発手法です。Simulink はモデルベースデザイン環境のデファクトスタンダードとして長年、多数の実績があり、移動ロボットプラットフォームを効率的に開発することができます。

ハードウェアを一から開発する場合には、ハードウェア自体のメカ設計と並行してソフトウェア設計用のアーキテクチャおよび構成の定義、解析、仕様作成が必要になります。このような場合、System Composer™を活用することで要件を割り当てながら、アーキテクチャモデルを調整し、ハードウェア設計と減校してソフトウェアモデルを Simulink で設計、シミュレーションすることができます。システムはコンポーネントとインターフェイスとして記述するアーキテクチャモデルとし、それらの作成およびインポートが可能です。また、アーキテクチャモデルを Simulink の設計や C/C++ コードのアーキテクチャ要素から取り込むこともできます。これらのアーキテクチャモデルを使用して、要件の解析、ステレオタイプ化によるプロパティの取得、トレードオフスタディの実施、仕様の作成が可能です。

図 5 System Composer によるアーキテクチャの定義および解析

仕様の定義や分析ができたら、次に制御対象となる物理モデルを作成していきます。物理モデルは電気回路やメカ部品など複合的な領域の技術で構成されています。伝達関数などの数式や実験データに基づくモデリングのみならず、物理コンポーネントによる直感的なモデリングをサポートする Simscape™ファミリーがあります。メカ系の Simscape Multibody™、電気系の Simscape Electrical™などを提供しています。特にメカ設計では Simscape Multibody の CAD インポート機能を活用することで SolidWorks®, Autodesk Inventor®, PTC® Creo™などの各種 CAD ツールからの機構モデルの取り込みを可能とします。

制御モデルの開発では、伝達関数ベースの設計や PID 制御を行うための Control System Toolbox™や Simulink Control Design™などが提供されています。状態遷移やフローチャートのようなロジックの記述には Stateflow®が最適です。このように物理モデルから制御モデルまで統合環境の中でシミュレーションを活用しながら開発検証を加速することができます。

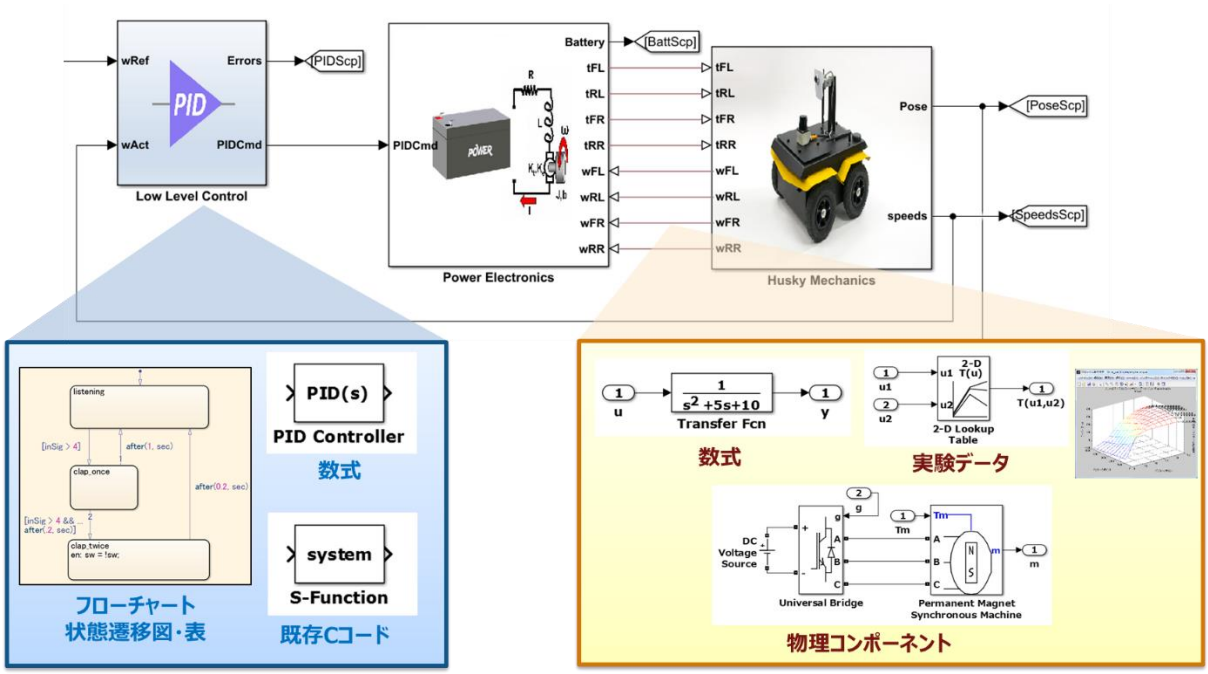

図 6 コントローラモデルと物理モデル

さらに、自動コード生成を活用して制御モデルから C コードを生成し、ラピッドコントロールプロトタイピング (RCP) を行うことも可能です。RCP を活用することで、組み込みボード向けに固定小数点化などのモデルの作りこみをすることなく、実機上で制御アルゴリズムの検証を行うことが可能です。また、製品開発を支援するための様々な検証系ツールも用意されています。

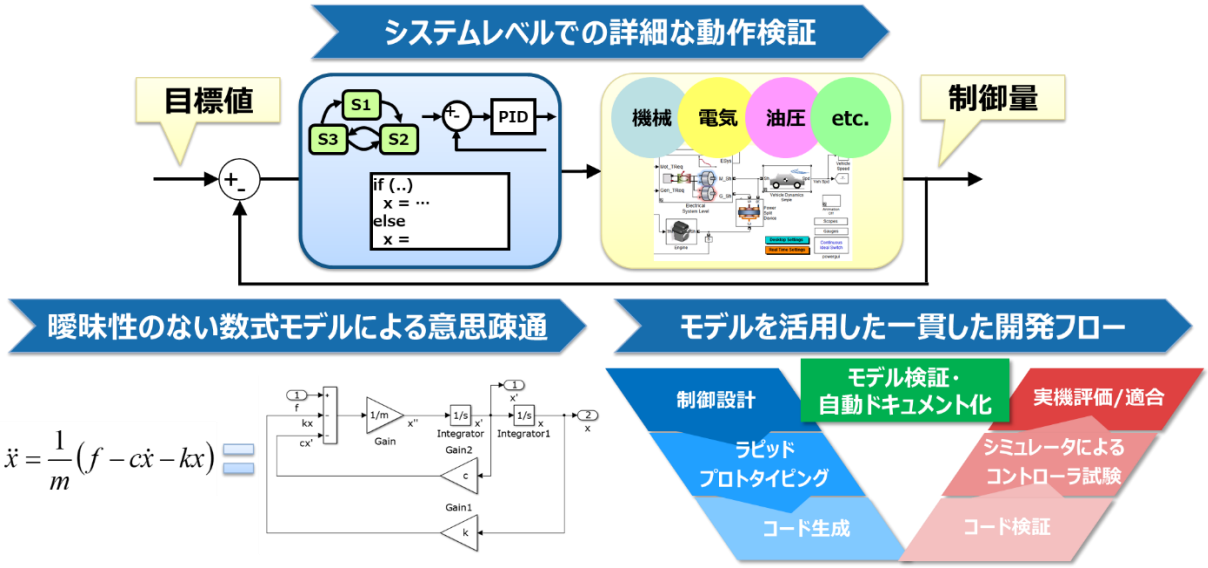

図 7 モデルベースデザインのベネフィット

Simulink をプラットフォームとしたモデルベースデザインにより、シミュレーションを活用することで開発検証を加速し、移動ロボットのプラットフォームを構築することが可能です。

関連リソース

- システムおよびソフトウェアアーキテクチャの設計と解析
- メカトロニクスシステムの設計

センサー データの収集

ここからは既存の移動プラットフォーム(車両)にセンサー群を配置し、各種センサーデータを収集することを考えます。センサー群としては下記のようなカメラ、Lidar、レーダー、超音波センサー、GNSS、IMU など複数のセンサーデバイスが考えられます。センサーの構成や配置を検討する初期段階では多数のセンサーを積載し、センサーデータを可視化しながら必要になるアルゴリズムを検討することになります。その際にセンサー間の座標系のキャリブレーションや時刻同期が必要です。

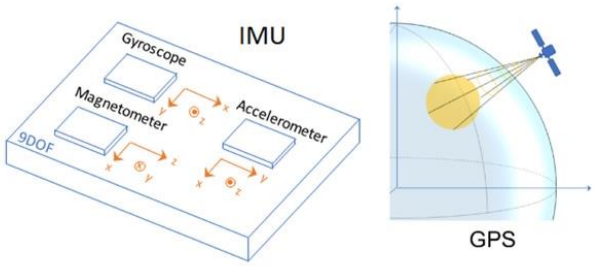

図 8 センサーのシミュレーションモデル

センサー間の座標系キャリブレーション

複数センサーを扱う場合にはそれぞれのセンサー位置姿勢を統合して環境を認識する必要があります。例えば、単眼カメラでこのような位置推定を行う場合、内部パラメーター(焦点距離、光学中心など)、外部パラメーター(位置姿勢)、歪係数が必要となります。このようなカメラのパラメーターを推定し、補正することをキャリブレーション(較正、校正)と呼びます。Computer Vision Toolbox™ではピンホールカメラや魚眼カメラをキャリブレーションするためのアプリケーションが提供されています。

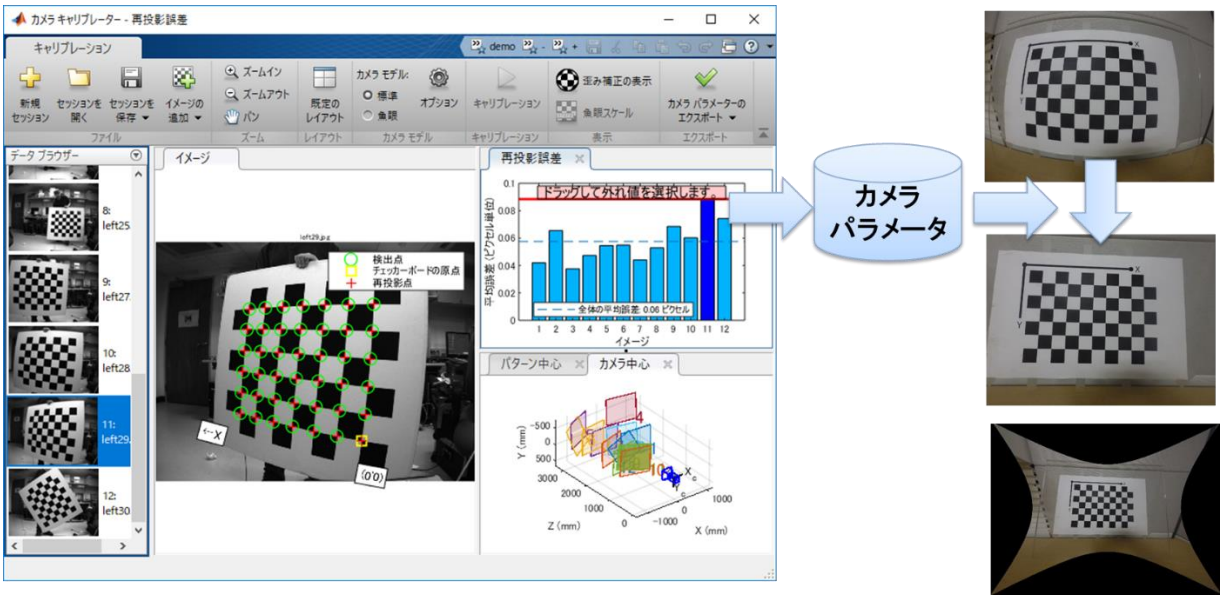

図 9 カメラキャリブレーション

さらに複数に取り付け位置のキャリブレーションを行う必要があります。Lidar や RGB-D カメラから取得される点群データと単眼カメラの座標を補正することで、物体検出は単眼カメラ、距離推定は点群といったように 2 つのセンサーの長所を組み合わせることができます。

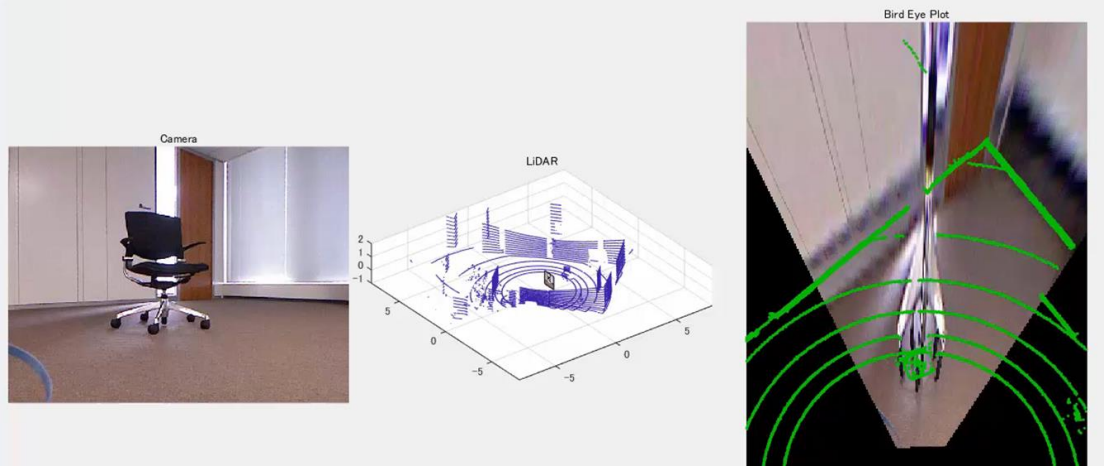

図 10 カメラと Lidar のキャリブレーション

センサー間の時刻同期

複数センサーを扱う上でもう一つ重要な作業が時刻の同期です。センサーの種類によってサンプリングレート、位相、ジッターなどが異なります。これらを正しく把握して時刻同期を行う必要があります。時刻同期が出来ていない場合、自己位置や対象物の位置を誤認識し、自車の発振や障害物への接触など予期せぬ問題が発生します。

このような複数のレートのセンサーデータの取り扱いはミドルウェアの活用が便利です。ROS (Robot Operating System) はロボット用のミドルウェア、ソフトウェアプラットフォームです。様々なセンサーベンダーが ROS 用のドライバを提供しており、複数センサーを接続して容易にセンサーデータを取得することが可能です。それぞれのノードが出力するトピックにはタイムスタンプがついており、時刻の一元管理に役立ちます。それらを保存し、再生解析する rosbag という機能も備えています。ROS Toolbox を活用することで rosbag に記録されたタイムスタンプ付きデータを MATLAB や Simulink で再生し、容易に時刻の同期を実現することができます。

関連リソース

- ***カメラ キャリブレーションと 3 次元ビジョン***
- ***timetable* による時刻同期**
- ***rosbag* ログ ファイルを開いて解析**

環境の認知

自律的にロボットが動作するために、各種センサーから出力されるセンサーデータを活用して周辺環境を認識していきます。認識結果は後段での行動計画や意思決定で活用されます。この章では画像認識および点群処理に焦点を当てて対象物体の検出について解説します。

画像処理・ディープラーニングによる物体検出

近年、AI 分野で最も注目されている技術がディープラーニング(深層学習)です。特に画像への応用が進んでおり、目覚ましい成果が発表されています。Deep Learning Toolbox™ではディープラーニングの画像、時系列信号など様々な適用例を提供しています。

画像のディープラーニングでは画像の分類や検出、セグメンテーションなど複数の粒度のアルゴリズムが提案されています。自律移動ロボットでは処理速度と検出精度の観点から物体検出アルゴリズムがよく用いられます。その中でも速度、精度で注目されているアルゴリズムの 1 つが YOLO です。

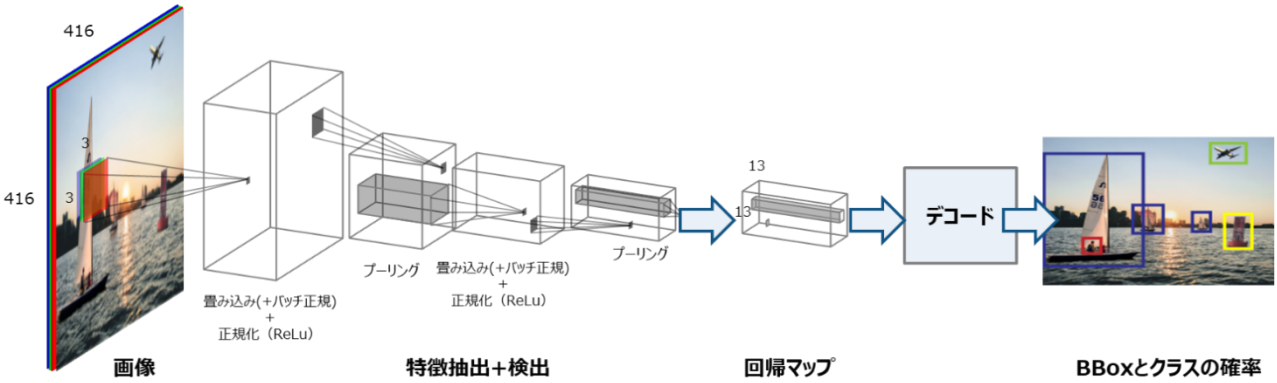

図 11 YOLO v2 による物体検出

YOLO はリアルタイム向けの物体検出アルゴリズムで、シングルステージ物体検出器に分類される手法です。候補領域の抽出と分類一度に実行するため、Faster R-CNN のような 2 ステージ物体検出器に比べて高速である点が特徴です。YOLO v2 および YOLO v3 が Deep Learning Toolbox および Computer Vision Toolbox でサポートされています。他に SSD(single shot multibox detector)もサポートされています。物体検出器を学習するためには通常、教師データが必要となります。このような教師データは自身で作成するか、公開されているデータセットを活用します。Computer Vision Toolbox には教師データをラベル付け(タグ付けやアノテーションとも呼ぶ)を行うためのラベラーアプリケーション (Image Labeler) も用意されています。

より高度かつ柔軟なディープニューラルネットワークを構築したい場合には Deep Learning Toolbox のローレベルインタフェースを活用することができます。`dlarray` と呼ばれる専用のデータクラスを活用することで、自動微分がサポートされ、逆伝播関数の実装など煩雑な手順を踏まずに複雑なネットワークを構築することが可能です。これにより、GAN (Generative Adversarial Network)や VAE (Variational Auto Encoder)といった先端的なネットワークを作成することも可能です。

点群処理による物体検出

自律移動ロボットを開発する上で不可欠なセンサーの 1 つが Lidar (Light Detection and Ranging)です。レーザー光の反射を使って対象物体までの距離を測定します。能動的な計測手法になるため、物体までの距離を高精度に計測することが可能です。さらに、レーザー素子を回転させながら距離計測

を行うこともできます。その際、1 本のレーザー素子を使うものを 2D Lidar と呼び、複数本のレーザー素子を使うものを 3D Lidar と呼びます。Lidar の計測結果は 2 次元もしくは 3 次元上の測距点の集合となるため点群(ポイントクラウド)と呼ばれます。このような点群を処理することで物体までの距離だけでなく、点群の分布から物体の種類や姿勢を推定することも可能です。

Computer Vision Toolbox は点群処理の様々な機能を提供しています。例えば、点群のダウンサンプリングや点群間のマッチング、形状への当てはめなどです。

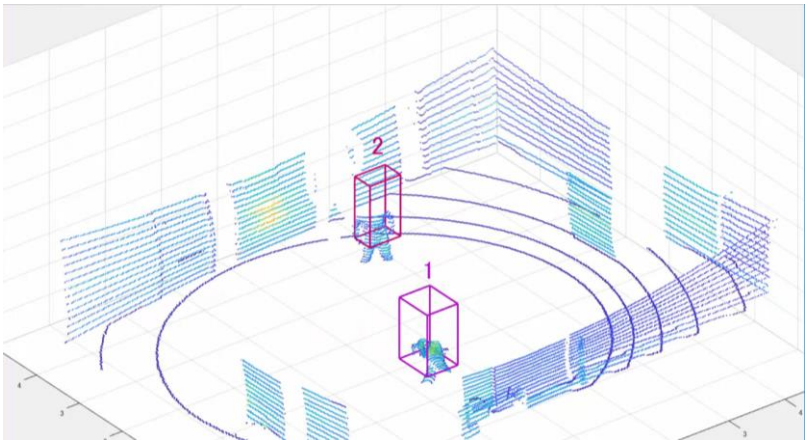

図 12 点群処理による物体検出

検出物体のトラッキング

次に周辺物体の動きを推定することを考えます。

周辺物体が移動する場合、カメラや Lidar などで検出されたそれぞれの検出結果をフュージョンする必要があります。このようなセンサーフュージョンではカルマンフィルターをはじめとした各種状態推定フィルタと複数のトラックの割り当て管理を行うためのアサインメントアルゴリズムが必要となります。

Sensor Fusion and Tracking Toolbox™を活用することで様々な状態推定フィルタおよびアサインメントアルゴリズムの検討を行うことができます。それぞれのセンサーの長所短所を補い、精度よく対象物体の位置を予測することは障害物回避などの安全な走行に対して不可欠です。

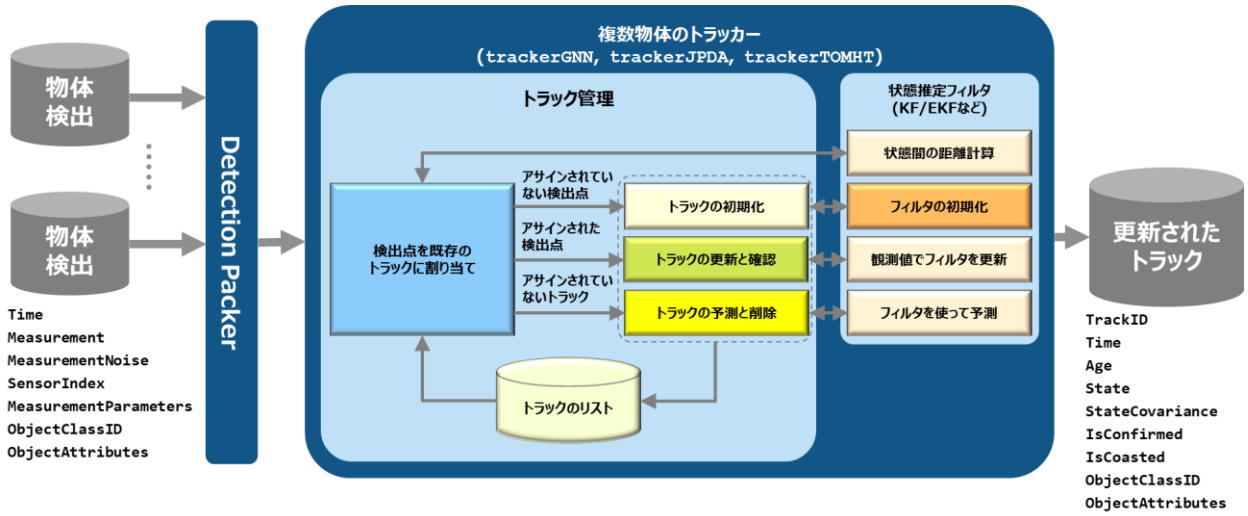

図 13 センサーフュージョンによる物体追尾

関連リソース

- *YOLO v2 深層学習を使用したオブジェクトの検出*
- *SSD 深層学習を使用したオブジェクトの検出*
- *Lidar および点群の処理*
- マルチセンサー追跡およびナビゲーション システムの設計およびシミュレーション

行動計画および意思決定

移動ロボットは環境を認知できたら、その情報を元に行動を計画し、タスクを遂行するための意思決定を行う必要があります。

SLAM と位置推定

2D Lidar (レーザーレンジファインダー) を使って未知の空間を探索することを考えます。Lidar から得られる点群を位置合わせしていくことで、徐々に地図を生成することができます。ところが逐次的に位置推定を行っていくと、どんどん誤差が蓄積していき、本来の位置から大きくずれてしまいます。このような位置ずれをドリフトと呼び、ドリフトの補正を考慮した地図生成が必要となります。

SLAM (Simultaneous Localization and Mapping, 自己位置推定と環境地図作成の同時実行) とは移動体の自己位置推定と環境地図作成を同時に行う技術の総称です。SLAM を活用することで、移動体が未知の環境下で環境地図を作成しながら、ドリフトを補正することが可能です。構築した地図情報を使って障害物などを回避しつつ、特定のタスクを遂行します。SLAM には Lidar を使った Lidar SLAM やカメラを使った Visual SLAM など複数の手法が存在します。

Lidar SLAM について見ていきます。Lidar(レーザーレーダー)は前述のとおり、カメラや ToF などの他の方式のセンサーに比べて格段に精度が高く、自動運転やドローンなど速度が速い移動体でも広く利

用されています。Lidar SLAM では一般に点群同士をマッチングすることで移動量を逐次推定し、積算することで自己位置を推定することができます。Lidar からの点群のマッチング では ICP (Iterative Closest Point)アルゴリズムや NDT (Normal Distributions Transform)アルゴリズムが用いられます。Lidar SLAM で扱われる地図表現は図のとおりです。縦軸が 2D/3D、横軸が点群とグリッドの違いになります。点群マップについては Lidar センサーの出力データをそのまま結合したものと考えることができます。細かな表現ができる一方で、障害物の衝突判定を行うには不向きです。点群を等間隔のグリッド上にサンプリングしたのがグリッドマップです。障害物検知や経路計画などに活用できます。

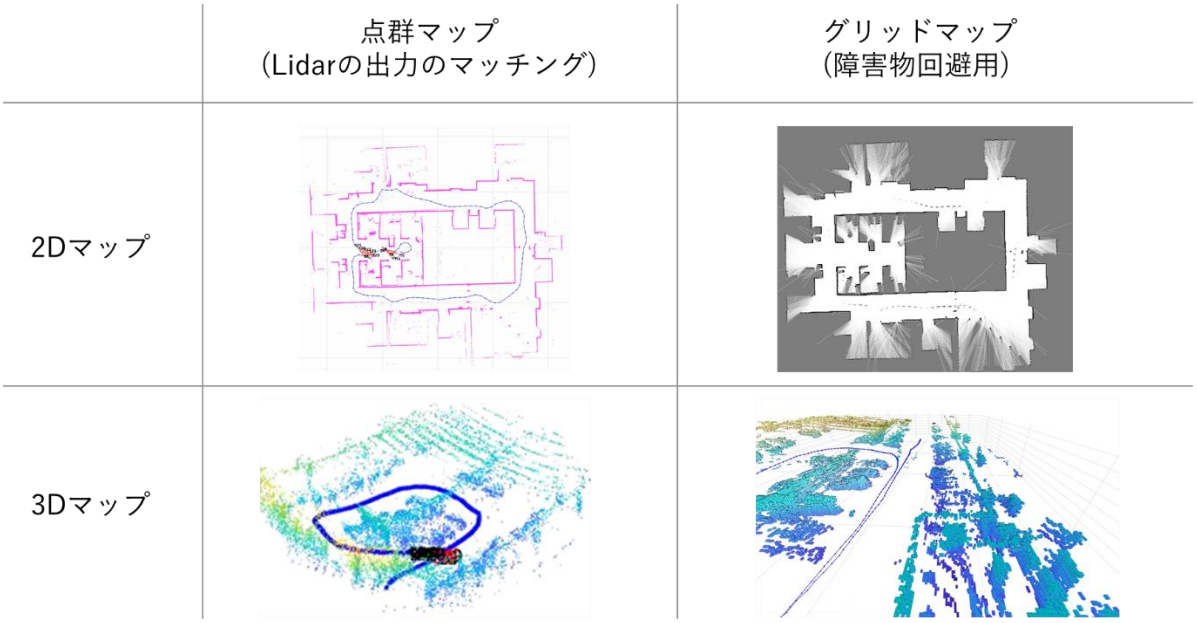

図 14 2D および 3D Lidar SLAM

一方、Lidar SLAM は密度の観点では画像に比べて粗く、点群同士のマッチングにおいて十分な特徴が得られない場合もあります。例えば、周囲に障害物が少ないような場所では点群間の位置合わせに失敗し、移動体の位置を見失ってしまうことがあります。また、点群のマッチングは一般に処理負荷が高いため、高速化の工夫が必要となります。

Visual SLAM は名称のとおり、カメラ・イメージセンサーからの画像を活用した SLAM 技術です。カメラの種類としては単眼カメラ (広角カメラ、魚眼カメラ、全天球カメラ)、複眼カメラ (ステレオカメラ、マルチカメラ)、RGB-D カメラ (深度カメラや ToF カメラ)などが含まれます。Visual SLAM のアルゴリズムは大きく 2 つに分類できます。画像特徴点のマッチングによる疎な手法と、画像全体の輝度を使った密な手法です。アルゴリズムとして、前者は PTAM や ORB-SLAM、後者は DTAM、LSD-SLAM、DSO、SVO などがよく知られています。

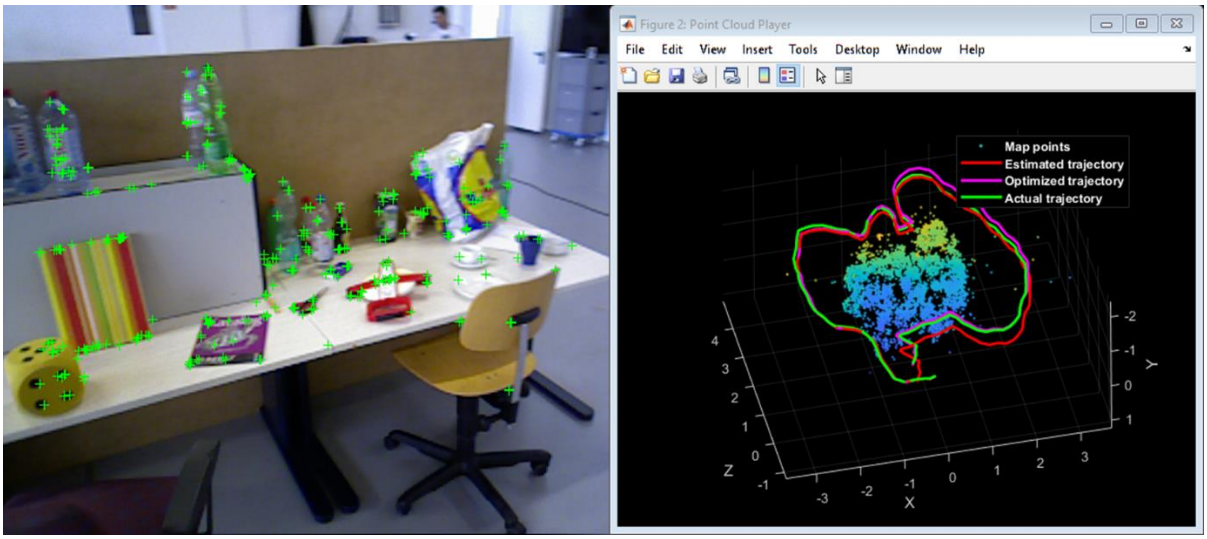

図 15 単眼カメラによる Visual SLAM (ORB-SLAM)の例

Visual SLAM は比較的安価に入手できるカメラが利用できるためコストが抑えられる利点があります。また、カメラは情報量が多いため様々なランドマーク(以前観測した場所)を検出することが可能です。一方、単眼カメラを使用する場合は距離の単位(スケール)が不定となるため、AR マーカーやチェッカーボードなど既知の物体を検出して画像上から物理量を知る、もしくは、IMU (Inertial Measurement Unit、慣性計測装置) などの物理量を直接計測できるセンサーとフュージョンする必要があります。Visual SLAM は Structure from Motion (SfM) や Visual Odometry (ビジュアルオドメトリー)、バンドル調整などの技術と強い結びつきがあります。

Lidar SLAM も Visual SLAM も逐次推定した姿勢を累積するためドリフトが発生します。このようなドリフトを低減に活用されるのがポーズグラフです。ポーズグラフは各姿勢がどの座標系をベースにしているか管理するためのグラフです。Navigation Toolbox™のポーズグラフは任意のセンサーデータ向けに拡張ができます。SLAM のフロントエンドはセンサーから移動量の推定や障害物の位置推定を行います。これらはセンサー依存の処理となります。カルマンフィルターなどを活用し、複数のセンサーを統合して位置推定することもできます。一方、バックエンドのポーズグラフの構築や最適化はセンサー非依存の処理です。バックエンドの部分はいろいろな SLAM に応用が可能です。ポーズグラフの最適化はバンドル調整と呼ばれる場合もあります。

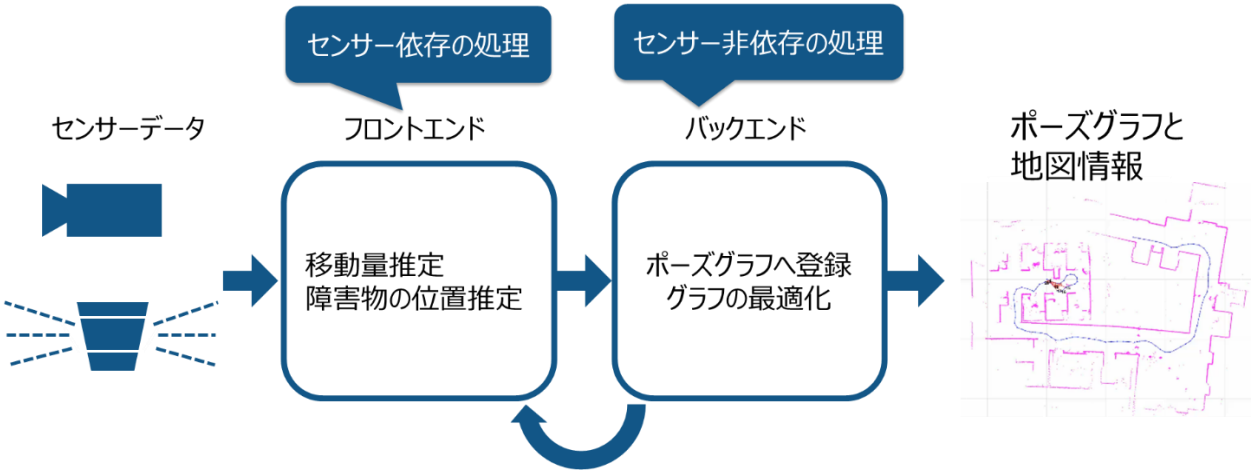

図 16 SLAM のフロントエンドおよびバックエンド

自車位置推定

ロボットが行動を計画する際、ロボット自身の位置と向きを推定する、自車位置推定(ローカリゼーション、Localization)が必要になります。ロボットの姿勢は既知の環境マップ、距離センサーやオドメトリなどのセンサーデータの組み合わせによって推定します。このような状態推定の一つの方法として粒子フィルター(パーティクルフィルター)を使用したモンテカルロローカリゼーション(逐次モンテカルロ法)があります。このとき各粒子はロボットが取り得る状態を表します。ロボットが環境内で動き、距離センサーを使用して環境内のさまざまな部分を検出する中で、粒子は 1 か所の周りに収束します。また、ロボットの動きは、オドメトリ センサーを使用して検出されます。このように確率的な分布でロボットの位置を表現することにより、環境の変化やセンサーの計測誤差を加味したロバストな推定を行うことが可能です。モンテカルロローカリゼーションは Navigation Toolbox に機能が含まれています。

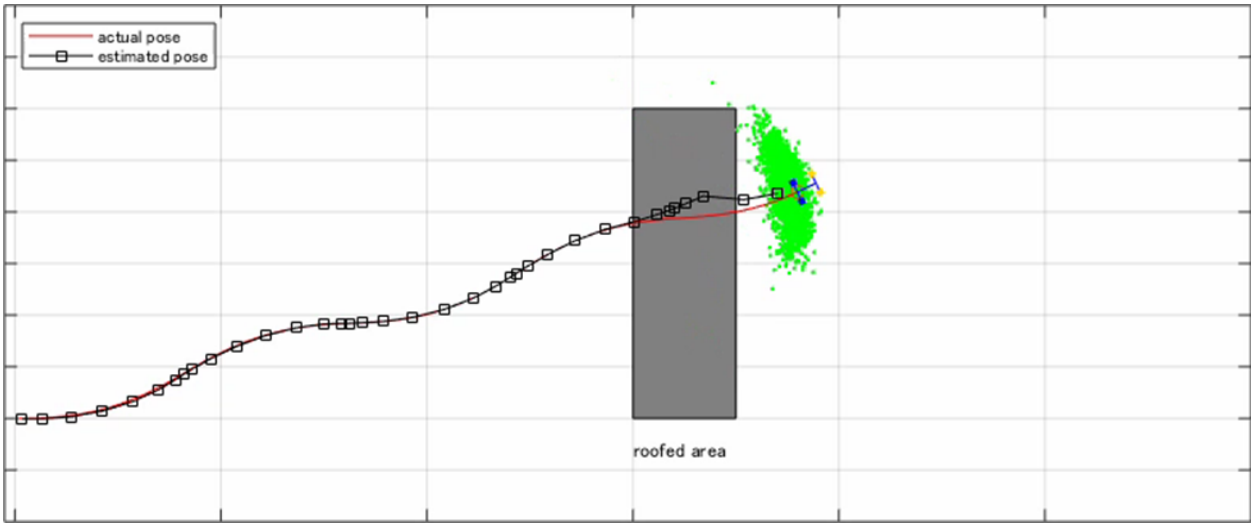

図 17 車両位置推定(ローカリゼーション)

行動計画および経路計画

行動計画(モーションプランニング)はパスプランナーを使用して、最適なパスを選択し、パス追従のためのステアリングコマンドを生成します。パスプランナーは経路計画(パスプランニング)のアルゴリズムを活用して、自車位置と目標位置の情報から地図上の障害物を回避する最適な経路を生成します。代表的な経路計画のアルゴリズムとして Rapidly-Exploring Random Tree (RRT) や RRT* などのサンプルベースの手法があります。これらは Navigation Toolbox でサポートされています。生成された経路は Dubins および Reeds-Shepp のモーションモデルを使用して、円滑かつ走行可能なパスに調整されます。最後に、経路のメトリクスを使用して、円滑性と障害の回避を検証します。数値比較および視覚的比較を用いて最良のパスを選択することが可能です。

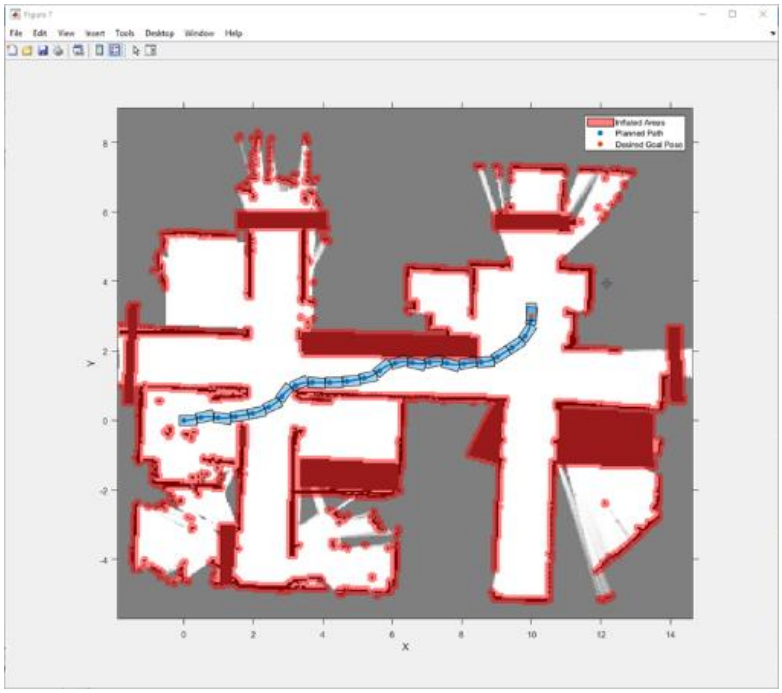

図 18 パスプランニング

状態遷移による意思決定

自律移動ロボットでは時々刻々と変わる状態に合わせて意思決定を行っていく必要があります。例えば、目の前に障害物があったら回避行動を取り、電池が消耗してきたら充電ステーションにモデルなどの判断・意思決定をモデリングする必要があります。このような状態の判断には Stateflow が活用できます。

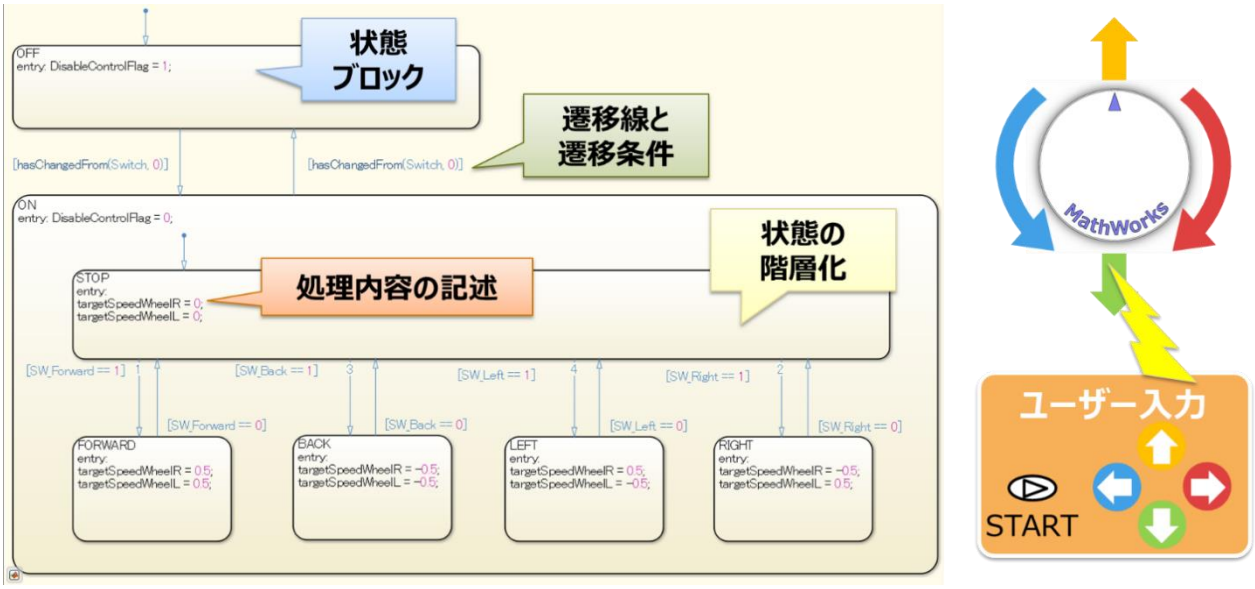

図 19 Stateflow による状態遷移表現

Stateflow は状態遷移図以外にもフローチャートや状態遷移表、真理値表現などの複数のロジックの記述表現を持っています。これらの表現方法を通じて設計者の意図が読み取れるような可読性の高い意思決定のロジックがモデリング可能となります。これよって複雑な状態遷移を抜け漏れなく記述することができ、開発の手戻りを減らすことが可能です。

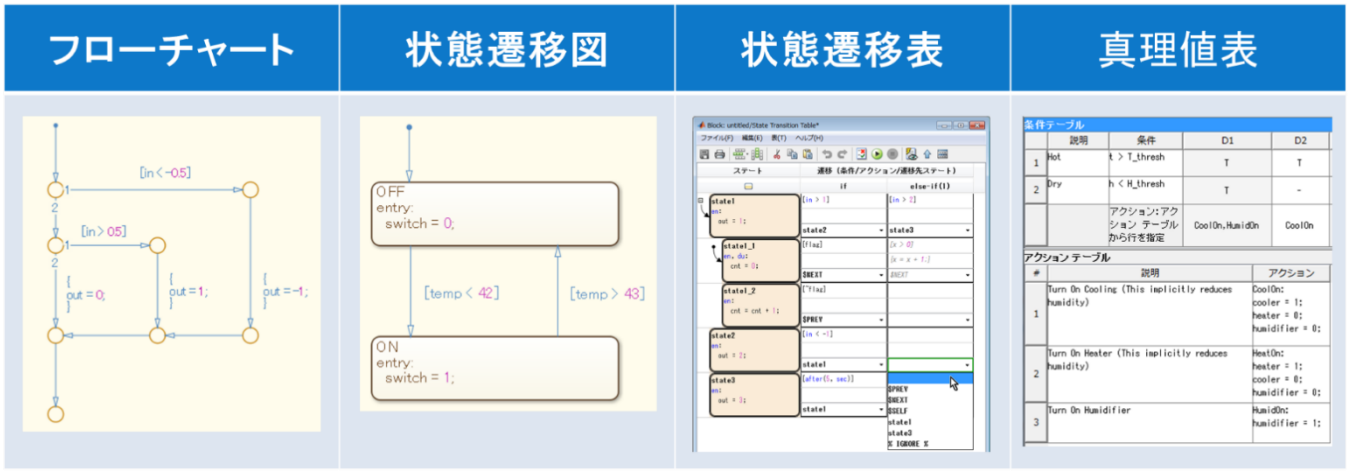

図 20 Stateflow によるロジックのモデリング

関連リソース

- *ロボティクス*: SLAM による自己位置推定と地図構築
- 単眼カメラの Visual SLAM

- モンテカルロ位置推定アルゴリズム
- 複雑度の異なる環境でのパス計画
- Stateflow チャートの作成と実行

追従制御

生成された経路に対して追従することを考えます。車両運動モデルを使用してステアリングと速度のコマンドを算出します。さらに、ベクトル場ヒストグラムなどのアルゴリズムにより障害物を回避します。

パス追従

パス追従アルゴリズムの 1 つとして Pure Pursuit があります。ロボットを現在位置から前方にある前方注視点に達するための角速度コマンドを計算します。直線速度は一定とするため、任意の点でロボットの直線速度を変更できます。ロボットが移動するたびに現在位置に基づいて、パス上の前方注視点をパスの最終点まで動かします。これによってロボットが常に自身の前にある点を追いかけているような動作となります。Navigation Toolbox では Pure Pursuit コントローラがサポートされています。

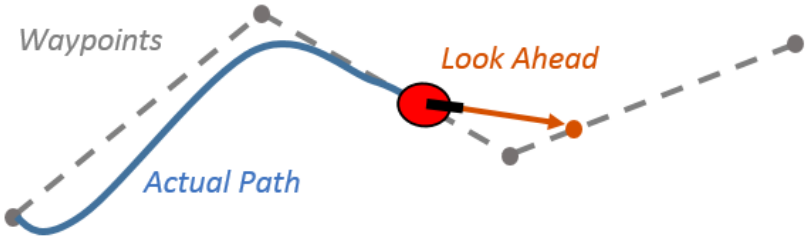

図 21 Pure Pursuit によるパス追従

モデル予測制御(MPC)

モデル予測制御 (MPC: Model Predictive Control) は、各時刻で未来の応答を予測しながら最適化を行う制御手法です。オンラインで高速に最適化問題を計算しながらフィードバック制御を行います。複雑な系に対して、より高性能な制御を実現することが期待されています。モデル予測制御は、実時間最適制御や Receding Horizon 制御という呼ばれ方もされます。モデル予測制御をパス追従に応用することで複合的な制約を満たす最適な追従制御を実現することが可能です。

モデル予測制御の設計とシミュレーションのステップでは、設計した予測モデルを取り込み、MPC コントローラに必要なパラメーターを設定します。Model Predictive Control Toolbox™では、MATLAB 言語によるプログラミング、あるいは、グラフィカルユーザーインターフェース（GUI）により必要なパラメーターを設定し、MPC コントローラを作成することが可能です。Simulink による閉ループ系の数値シミュレーションを通じて設計の妥当性を確認します。

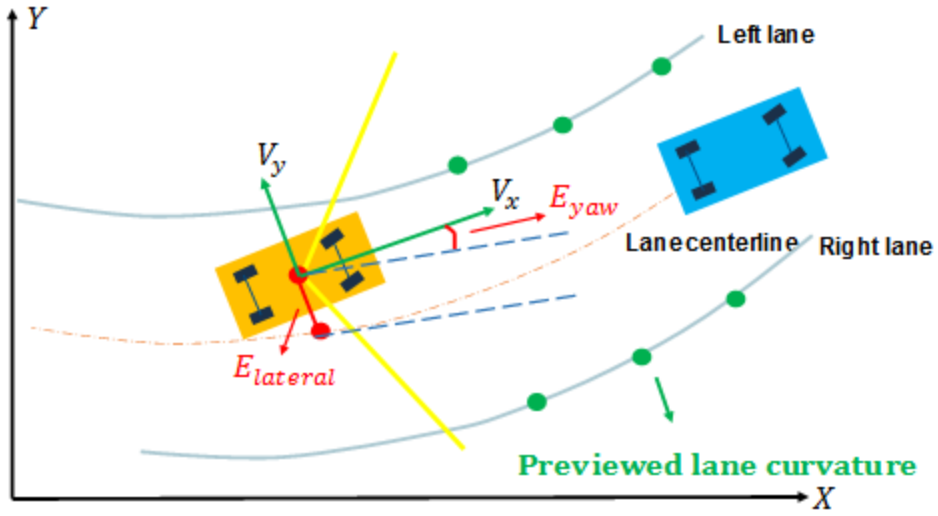

図 22 モデル予測制御によるパス追従

強化学習による障害物回避

経路追従や障害物回避には強化学習も応用が進んでいます。強化学習とは、環境中でエージェントが状態を観測し、取るべき行動を決定する機械学習の手法を指します。従来、強化学習では画像や点群などの高次元のセンサーデータを取り扱うことは困難でした。ディープラーニングの登場により、End-to-end の深層強化学習が注目を浴びています。深層強化学習を応用することで画像や点群から障害物を認識し、回避するような経路を試行錯誤で学習することが可能となりました。また、学習を継続することで環境の変化に対しても柔軟に対応することが可能です。

一方、深層強化学習のエージェントの学習には膨大な時間がかかるため、実機を使った学習は困難です。多くの深層強化学習ではシミュレーターを活用し、エージェントによる学習がある程度進んだら、実機上で追加学習、検証を行います。MATLAB および Simulink はシミュレーション環境の構築を強力にサポートします。さらに、Reinforcement Learning Toolbox™を活用することで DQN などの様々な強化学習エージェントを活用することが可能です。

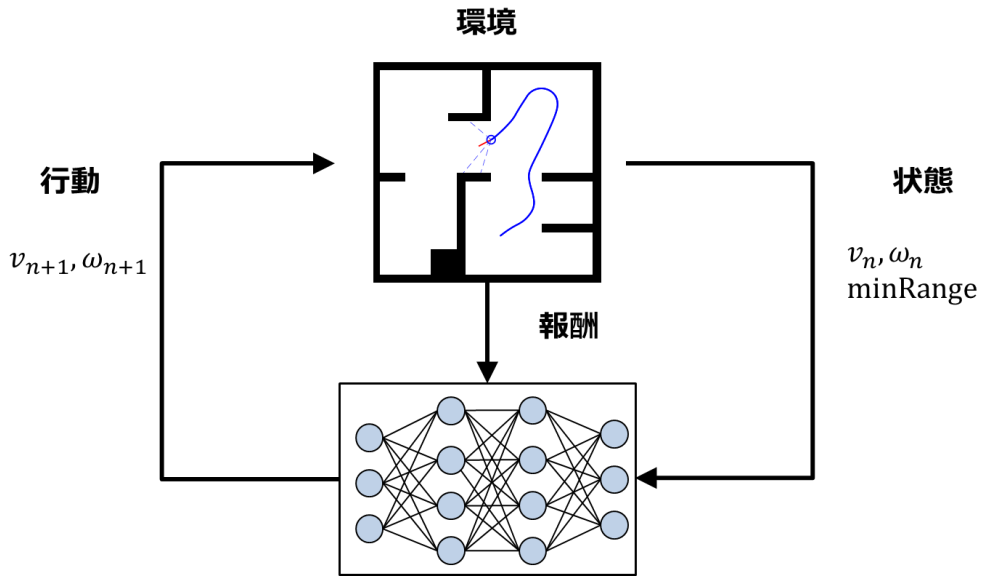

図 23 強化学習による障害物回避

関連リソース

- *差動駆動型ロボットのパス追従*
- *モデル予測制御（MPC）のビデオシリーズ*
- *強化学習による移動ロボットの障害物回避*

シミュレーション

自律移動ロボットのシミュレーションにはセンサーモデル含めた環境のモデル化が必要となります。センサーモデルも含めた 3 次元のシミュレーション環境として、Gazebo と呼ばれているシミュレーターがあります。Gazebo と協調シミュレーションする機能が Robotics System Toolbox™で提供されています。この機能を活用することで Gazebo と時刻同期しながら確定的なシミュレーションを実行することができます。Gazebo の様々なセンサーモデルと周辺環境モデルを活用することで、シミュレーションにて自律移動ロボットの動作を検証することが可能です。

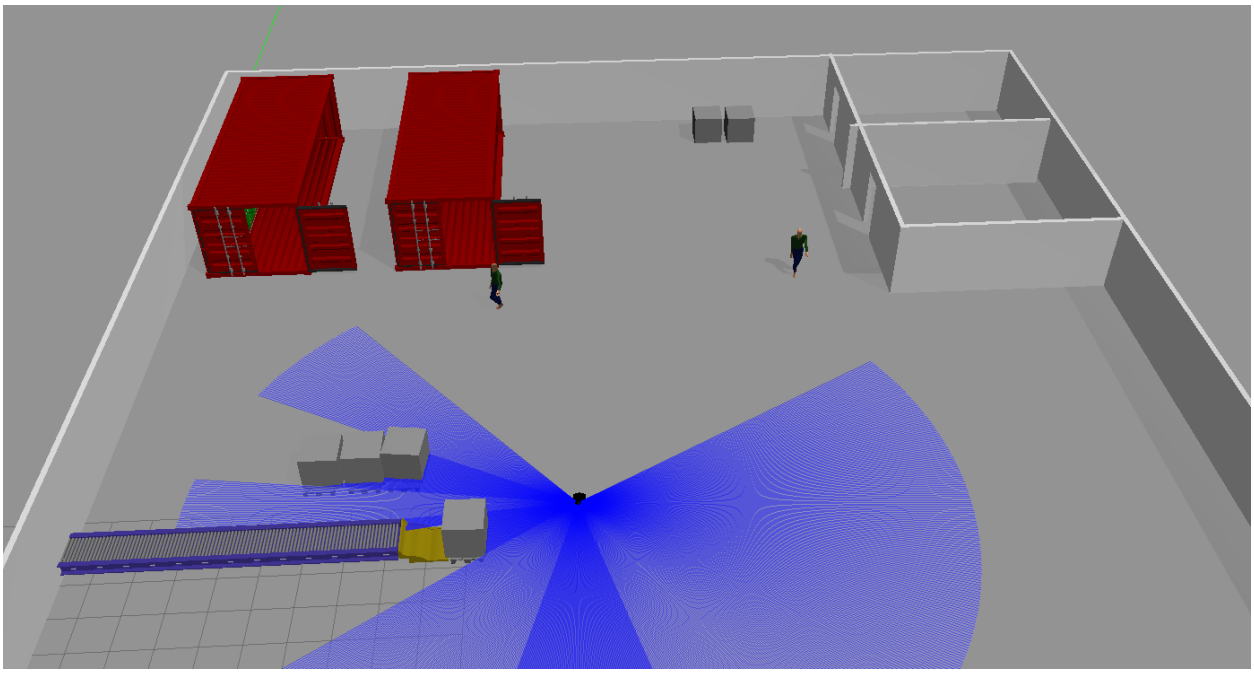

図 24 Simulink と Gazebo による協調シミュレーション

関連リソース

- *Simulink と Gazebo による協調シミュレーション*
- *Gazebo を使った倉庫内の移動ロボットのシミュレーション*

検証

Simulink でシステムを設計することでモデル化し、設計の早期段階で設計のテストを行えます。テストケース作成では Simulink Test™を活用することで複数のテストケースをテストハーネス作成してそれぞれのサブシステムに埋め込むことができます。また、テストマネージャーを活用することでシミュレーションの実行、テストの出力値と期待値の比較、等価性検証について、テストケースの管理と一括での実行を行います。これによりテスト全般を一元管理し、検証のトレーサビリティを確実にすることができます。

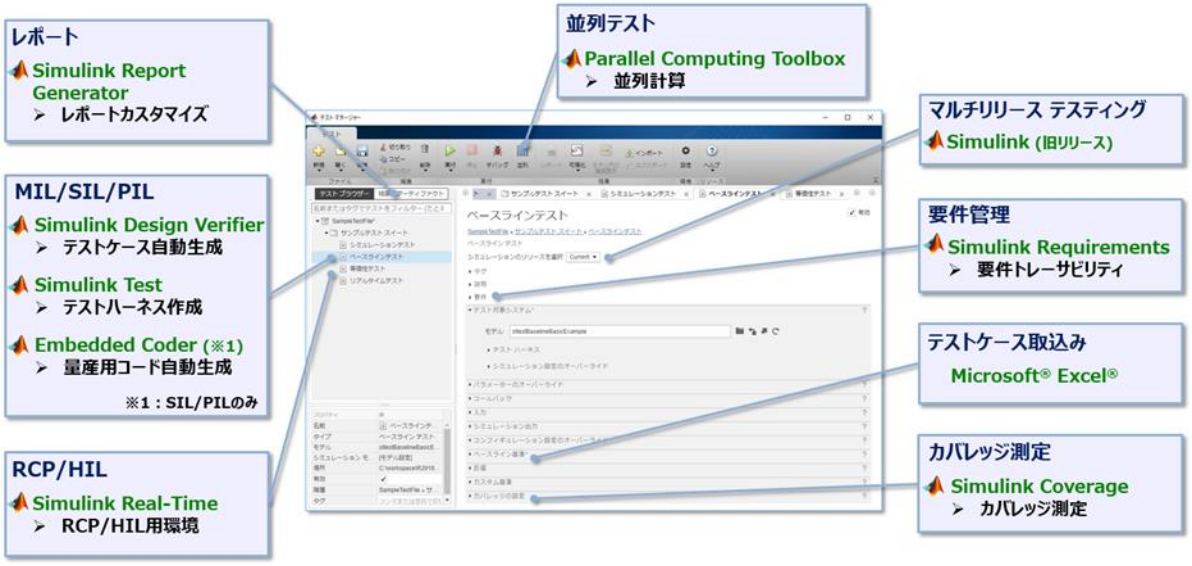

図 25 テストマネージャーによるテストケースの管理

関連リソース

- ***Simulink Test* でモデルのテスト実行・管理をより便利に**
- **モデルのデバッグ・品質保証の加速化に向けた処方箋**

ハードウェア実装

シミュレーションにて自律移動ロボットのアルゴリズムの確認ができたら、最終的にハードウェアに実装をしてプロトタイピングを行っていきます。自律移動ロボットのアルゴリズムは画像認識や点群処理などの並列性が要求される上位の処理から、位置推定、追従制御などのリアルタイム性が要求される下位の処理まで複合的な実装ターゲットが要求されます。MATLAB および Simulink では処理の内容に合わせて実装ターゲットを柔軟に選択いただくことができます。

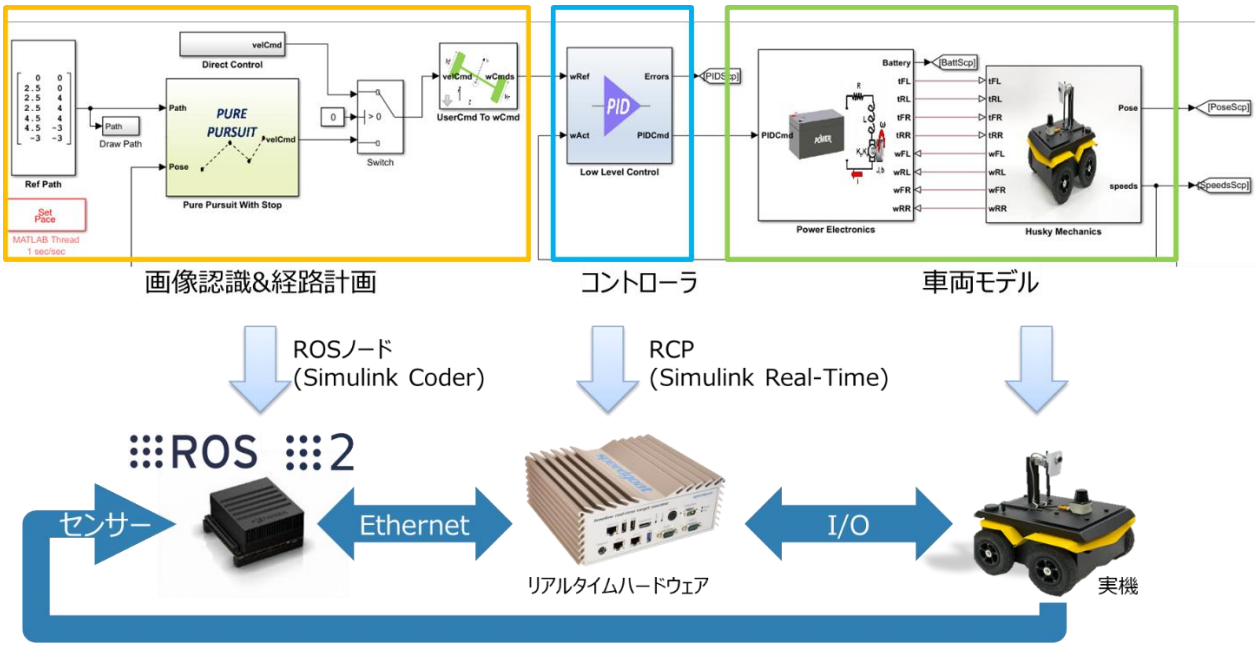

図 26 自律移動ロボットの実機実装概要

ROS ノードおよび ROS2 ノードの生成

画像処理や点群処理、モード制御など高度な並列性が要求されるようなアプリケーションには ROS が便利です。ROS は Robot Operating System の略称で、ロボット向けの分散並列処理のためのミドルウェアです。ROS を活用することで複雑なロボットシステムをコンポーネントベースで組み上げ、システムを構築することができます。

ROS は便利な一方、プラットフォームが Linux に限定されたり、リアルタイム性が欠けていたり、製品開発への適用が難しい側面があります。ROS に変わってマルチプラットフォーム対応でかつリアルタイム性を考慮できる ROS2 が注目を浴びています。ROS2 の登場によって従来のプロトタイピングのみならず量産開発への適用が期待されています。

ROS Toolbox は ROS および ROS2 との連携をサポートします。さらに構築した Simulink モデルから ROS および ROS2 ノードを生成することができます。これよって自律移動ロボット上のオンボードコンピューター上で生成したノードをスタンドアローンで動作させることができます。また、ROS から ROS2 に乗り換える際にも一部のブロックを置き替えるだけで移行することが可能です。

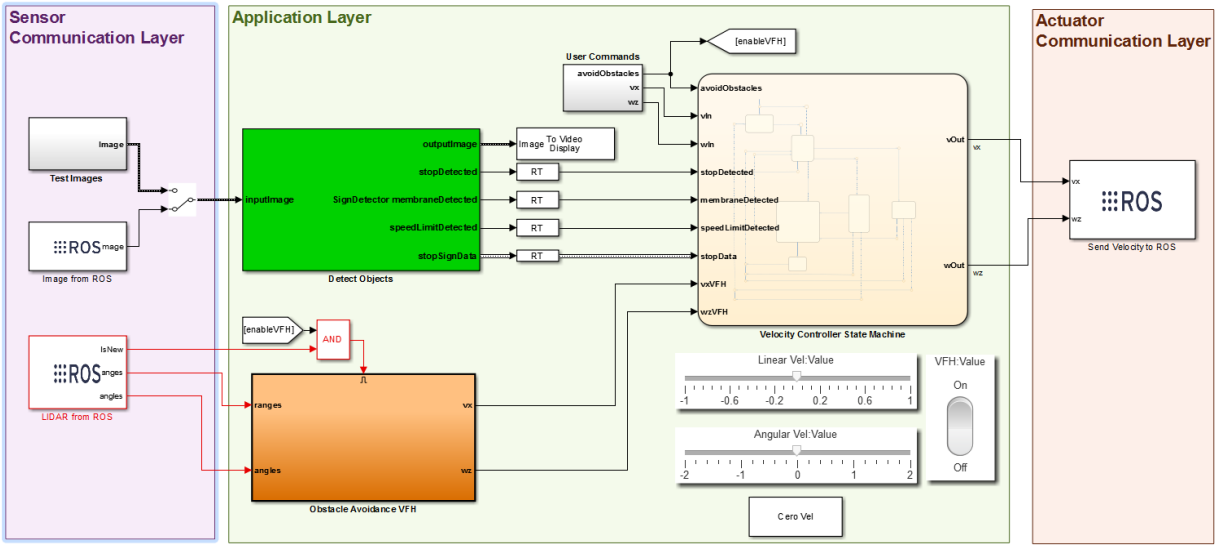

図 27　ROS の購読を含めた Simulink モデル

MATLAB Coder / GPU Coder による物体検出

画像処理やディープラーニングなど並列性が必要な処理に対しては MATLAB Coder™ や GPU Coder™ が活用できます。MATLAB Coder ではディープラーニングネットワークを高速実行するための Intel® MKL-DNN Library や ARM Compute Library をサポートしています。また、GPU Coder は NVIDIA® TensorRT™ や cuDNN などの高速化ライブラリをサポートしています。ターゲットに最適化されたライブラリを使用することで、学習したアルゴリズムを高速に実行することが可能です。

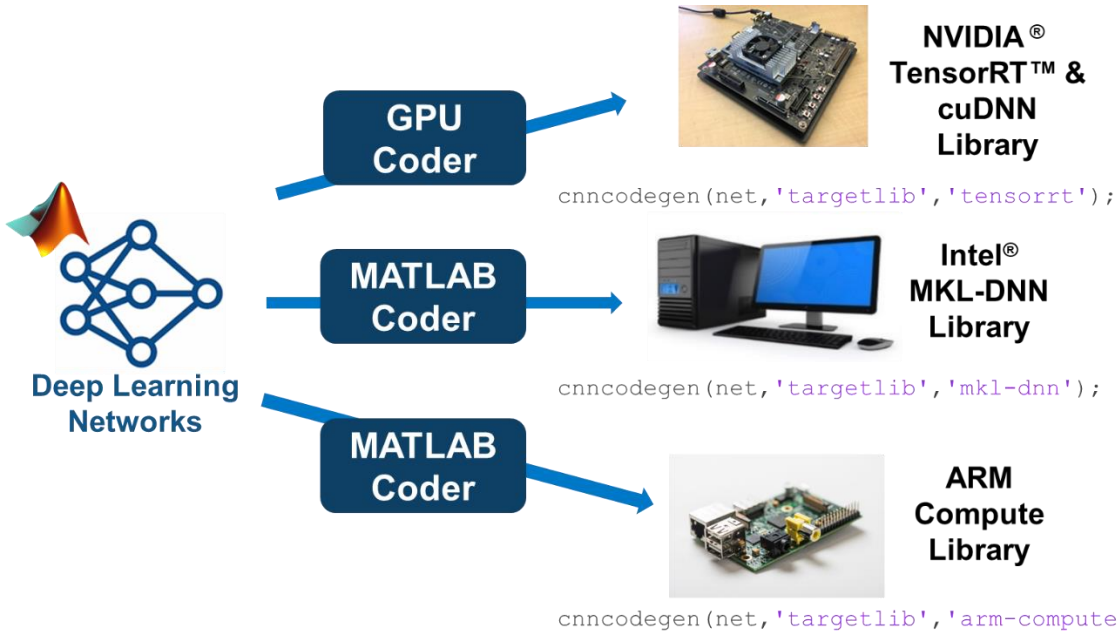

図 28 ディープラーニングネットワークの実装

リアルタイムハードウェアを連携

Simulink Real-Time™と Speedgoat を活用することでローレベルのリアルタイム制御を容易に実現することができます。制御モデルを迅速に実機テスト環境へと移行させる手法を特にラピッドコントロールプロトタイピング (RCP)と呼びます。この RCP を開発アプローチとして採用することで、Simulink モデルをハンドコーディングなしで素早く汎用ハードウェアに実装し、実機テストを行うことが可能となります。これにより、シミュレーションベースのアルゴリズム検討と実機試験ベースの機能検証・性能確認のプロセスを素早く回す事ができるようになり、業務効率の改善と製品品質の向上に寄与します。

この RCP の環境構築をサポートするのが、MathWorks が提供する Simulink Real-Time です。Simulink Real-Time を活用することで、RCP ハードウェアへのアルゴリズムの素早い実装や、試験中のパラメーター調整や信号のモニタリング/ロギングが可能となります。

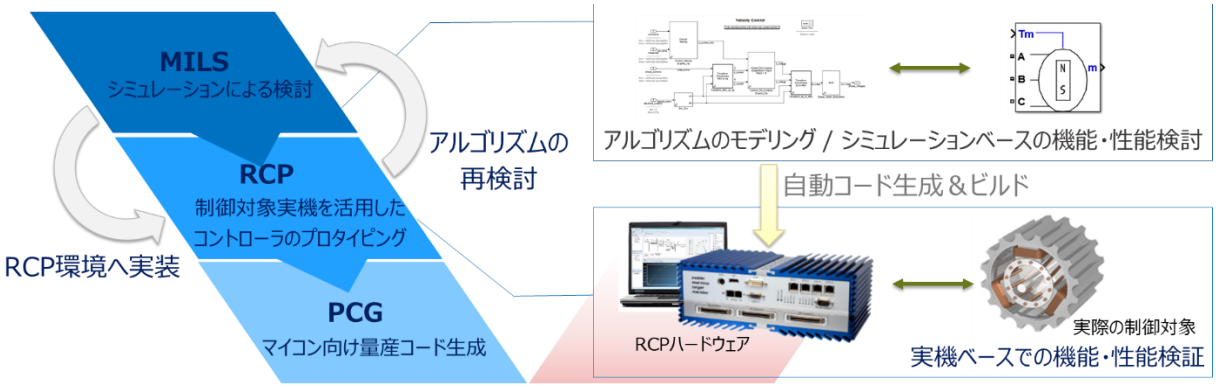

図 29　Simulink Real-Time と Speedgoat によるラピッドコントロールプロトタイピング

関連リソース

- *Simulink モデルからスタンドアローンの ROS2 ノード生成*
- *GPU Coder - NVIDIA GPU のための CUDA® コードを生成*
- ラピッドコントロールプロトタイピング*(RCP)*による制御アルゴリズムのテスト
- ハードウェアインザループシミュレーション*(HILS)*による量産/試作コントローラのテスト

まとめ

MATLAB および Simulink は自律移動ロボットシステムに活用可能な開発検証プラットフォームであり、MATLAB および Simulink および関連製品を活用することで複雑な自律移動ロボットの開発を加速することができます。

- 複数領域の技術を単一プラットフォームで取り扱いが可能
- 複合的なセンサー処理、アクチュエーター処理を柔軟に記述可能
- 高解像度、多次元のセンサー処理のマルチコア CPU や GPU による並列化が可能

マルチドメインシミュレーションを可能にする Simulink、自動運転に必要な要素技術をカバーする豊富なライブラリで、「認識・判断・制御」の各アルゴリズム開発から統合シミュレーションまで同一環境で実現できます。また、コード生成機能を利用すれば外部の環境へのアルゴリズムの配布が可能で、コードの書き換えの工数削減や書き換えによるバグ混入のリスクを低減できます。MATLAB および Simulink は自律移動ロボット開発においてもエンジニア・研究者の皆様を強力に支援します。

関連リソース

- ロボティクスおよび自律システム