PR2による物品片付け機能実現のための開発事例
–ROSのシステムとライブラリスタックに着目して–

○　花井　亮　　山崎　公俊　矢口　裕明　稲葉　雅幸 (東大)
Ryo HANAI, Kimitoshi YAMAZAKI, Hiroaki YAGUCHI and Masayuki INABA (The Univ. of Tokyo)
hanai@jsk.t.u-tokyo.ac.jp

ROS has been becoming popular as a distributed platform for robot applications. We present how we developed a tidying-up demonstration task including recognition of objects, grasping, handling of a tray and navigation in two-weeks stay at Willow Garage Inc. This paper focuses on the development using ROS framework, tools and packages.

Key words:　system integration，software architecture，ROS

1　背景

日常環境で片付け作業を行うロボットの実現のためには，多くの機能の統合が必須となる．したがって，デバイスや要素機能を部品として分割することで，個々のアルゴリズムの開発に専念することを可能にするミドルウェアが有効である．また，多くのロボットで共通に利用される機能であれば，それらをパッケージとして統一的なインタフェースでプログラマに提供することで，開発者はそこから開発を始めることができ，開発効率は大きく向上する．その意味で，ROS[1] や RT-middleware[2] をはじめとしたミドルウェアは共通の目的があるといえる．

しかし，実際にはそれらのミドルウェアを利用して上位のアプリケーションをどのように構築するかのノウハウが十分になければ実用的なタスクの実現は難しい．

そこで，本論文では著者らが willow garage 社における二週間の滞在で実現した片付け機能を例にとり，ROS を使って認識，把持，移動といった要素技術をどのように利用してシステムを構築したかを報告し，同様の開発を行う開発者に有益な情報を提供したい．

2　ROS システムの基本

ROS は分散システムであり，roscore ノードがネームサーバの役割を果たし，他のノードは名前解決後は直接通信する．通信形態は，トピックと呼ばれる 1 対他の非同期通信と，サービスと呼ばれる同期通信がある．センサデータなど継続的に送受信するものはトピックで流し，ナビゲーションの目的地など 1 回の要求が正しく受理されたか確認したいものなどはサービスで実行する．また，時間がかかる呼び出し時に実行状態の確認やキャンセルなどを可能とするサービスである actionlib がある．これは，通信の枠組みではなく，パッケージとして実装されている．

PR2 のシステムはこの ROS を設計に基づき，機能単位で細かくプロセスを分割し，それらの間のメッセージを定義することで構成されている．そうすることで，launch ファイルという起動用の xml ファイルを編集するだけで再コンパイルなしで，要素機能を組み合わせて別の機能やアプリケーションを構築することができる．これは大規模なアプリケーションの構築時には欠かせない．通信

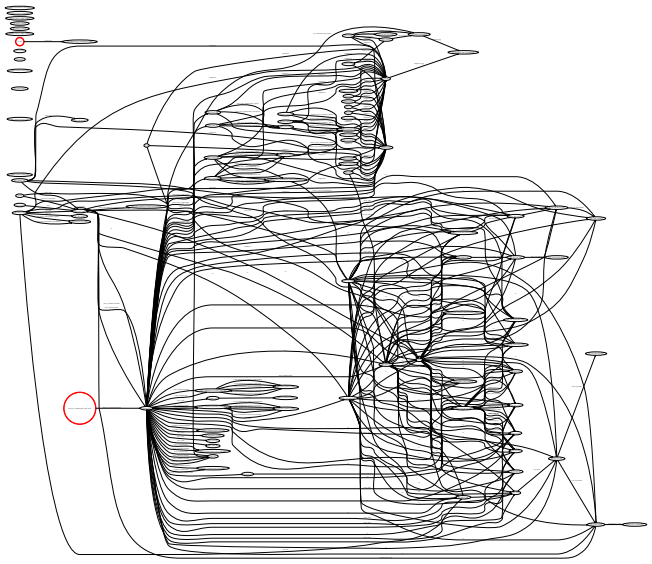

図 1: 96 nodes in the configuration of the tray demo (rxgraph)

相手はシステム全体で共有される名前空間における名前で指定する．これはソース中に記述されるが，ノードの起動時にポート名を別の名前に置き換える remapping 機能により，接続先を変更できる．これにより，プログラム部品の柔軟な再利用が可能となる．

ミドルウェアの場合，分散システムのフレームワークを提示するだけで，上位のロボットの機能をどのように分割し，それらの間の通信をどのように定義して実装すればよいかにまで立ち入ったものは多くない．その点において，ROS パッケージの実装は分散システムとしてロボットのシステムを実装する場合に多くの知見を与える．例えば，画像を含むセンサデータやカメラなどのパラメータ，座標変換もロボットの構造もすべてメッセージで流される完全な分散システムである．この仕組みにより，それぞれの処理を入力メッセージや呼び出し先のサービスにのみ依存する形で細かく分割実装することができる．集中管理のロボットステートを作ることはできるが，これ

は必須ではない．また，PR2 という具体的なハードウェアに対して，実用性が高い機能実装がパッケージとして公開されており，膨大なドキュメントやチュートリアル，そしてソースコードが用意されている点も ROS を用いてシステムを開発する利点である．

この設計方針に基づき細かく機能分割をするため，システムは非常に多くのプロセスで構成される．図 1 は片付けデモ実行時のノードとその間のトピック経路を図示したものである．システムの安定性は高く 96 個ものノードで構成されているにもかかわらず，安定して動作する．ただし，このシステムはかなり高速な計算機環境を想定していることを付け加えておく．実際，PR2 内部はクアッドコア，2 ノードであるがすべてのノードを起動しているときの負荷はかなり高い．これは ROS システムが分散システムで CPU コア数の恩恵を簡単に受けられる設計であるため，今後も予測される並列度の向上で解決すると見込んでのものだろう．計算能力が小さい場合には，使う機能を絞り込む必要がある．

ちなみに，各ノードを実行する場所は launch ファイルに machine タグで指定できる．また，外部の PC をシステムに加えるのも容易である．環境変数 ROS_MASTER_URI で接続先のネームサーバを指定すればシステムに参加し，透過的にメッセージの送受信やサービスの利用，提供が可能となる．ロボット外の PC で必要に応じてロボットの状態をモニタできる．実時間性に関しては，PR2 は Linux の実時間拡張を使用している．

3 開発プロセス

3.1 片付けタスク

実現したタスクは,「机の上の小物をトレイにのせて別のテーブルへ運ぶ」の他に，皿やシャツを発見して指定の場所へ片付けるものである．そのためには，把持すべき物品の認識，把持してトレイに置く動作の生成，位置を推定しながら安全に移動する機能，それらを統合したタスク記述が必要となる．以降の節ではこれらの開発の流れの中で，ROS のツールをどのように利用したかを説明する．

3.2 ナビゲーション

ROS には，自己位置推定，大域的経路計画，障害物回避を含む高機能なナビゲーションスタック [3] が用意されているため，これを利用した．PR2 はベースの水平 URG に加え，胸につけられた URG を上下に振ることで障害物検知を行っている．図 2 はデモの中で机に向かって移動しているところである．赤が三次元的な障害物情報，緑がベースの水平スキャン，青は障害物を一定距離だけ膨らませたもので経路計画に利用される．

ここでの問題は，PR2 のナビゲーションスタックは腕を畳んだ状態で安全を確保することを想定して設計されている点である．そこで，物を持った状態で移動をさせるために，把持物体に当たる点群および把持物体のエッジと後方の反射物の間に生じる shadow を取り除くフィルタを作成する必要があった．そのためには，レーザのデータがどのように処理されているかを知る必要がある．ROS ベースのシステムの解析は，トピックやサービスを調べることからはじまる．分散システムの動作確認は通常手間がかかるが，メッセージやノードの情報を提示したり，サービスやメッセージを操作するコマンドが充実していることにより，かなり見通しがよくなっている．rosnode や rostopic, rosservice といった軽量なコマンド群によりシステム内の個々のトピックやサービスの状況を素早く把握することができる．メッセージの周期を計測して，高負荷時に特定の処理が何 Hz で回っているかを確認したりもできる．また，全体の把握には rxgraph というコマンドを一行入力すれば，図 1 のように可視化できる．

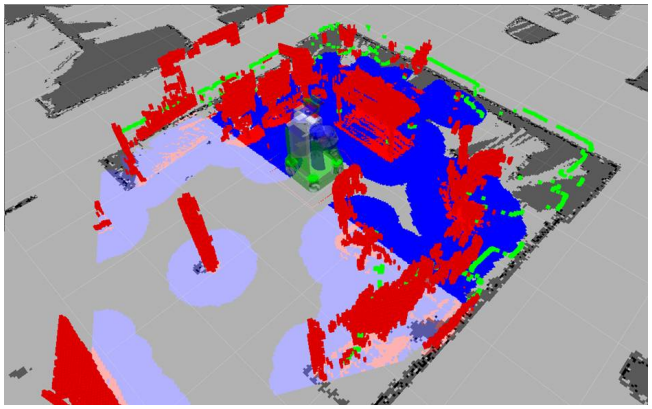

図 2: Navigation stack

利用するトピックとサービスがわかれば，処理の実装に移る．レーザのデータは base では位置推定や障害物検出，他にもマニピュレーション時の対象物の認識，障害物検出なの多くの処理に利用される．したがって，レーザのデータは多くの処理で加工される．データを加工する処理がノードに分割されているので，用途に応じて適当な処理を間に挟むことが容易である．そうすると次に処理結果を確認したくなる．組み込みの多くのメッセージは rviz というビューアでそのデータ型に適した形で可視化して確認することができる．図 3 は rviz でフィルタ処理の過程を可視化したものである．紫のスキャンがトレイの縁にあたり，shadow が出ていることがわかる．また，ここでは図示していないが，フィルタ後のスキャンの表示を追加することで，実装したフィルタが意図した通りに機能しているかどうかを確認できる．図 3 のナビゲーション処理も rviz に地図や障害物地図などのトピックを表示したものである．

フィルタは使いまわしたいことが多いため，ポートの remapping が有効に機能する．ロボット自体の反射はロボットのモデルを利用して取り除く機能がある．今回のデモでは時間も限られていたため，所持物を正確に除外することはせずに，腕とトレイを含む周辺を大雑把に取り除いた．また，ベストな解ではないが，把持姿勢で肘が外側に出た分とトレイの分だけロボットの衝突干渉のフットプリントを大きくした．これらのパラメータはファイルに記述されており，ナビゲーションスタック自体は他のロボットでも幅広く利用できるよう設計されている．

3.3 マニピュレーション

ROS には gazebo シミュレータ [4] があり，動作を確認しながら作成できる．このシミュレータは視野画像のシミュレートを利用した認識処理の開発など利用価値が高い，動力学シミュレーションには標準で ODE[5] を利用している．図 4 の左が gazebo シミュレータの画面であ

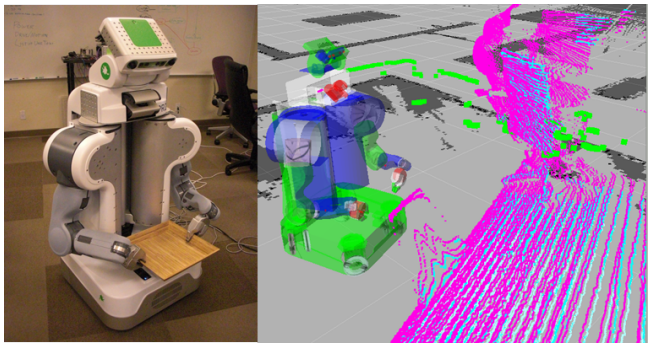

図 3: Visualization using Rviz (Laser pipeline)

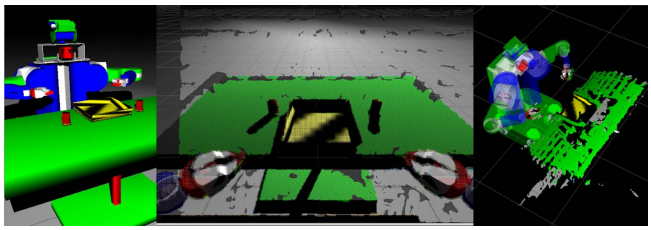

図 4: Motion development to grasp a tray using gazebo simulator

り，中央がシミュレートされ視野画像，右は rviz 上に 3 次元点群を表示したものである．センシングされたデータはメッセージとして流されるため，実ロボットの場合はデータ生成側が実センサノード，シミュレーションの場合はシミュレータからデータを生成するノードと切り替えるだけでそのデータを利用する側は変更が必要ない．また，後述のように生成されたメッセージをログとして保存する機能がシステムに用意されているため，センサデータ生成元をログの player に切り替えればログベースの開発になる．

ROS の場合，左右の腕や頭，グリッパなどの部位ごとに，また，機能レベルでも複数のサービス，トピックに分かれていて，必要なものを必要に応じて起動する仕組みになっている．これは柔軟な作りであるが，多くのタスクを実現するためには，ロボットの動作をプログラマが作成しやすい環境が必要であり，我々は EusLisp[6] で普段行っているように，Python スクリプト上で PR2 の全身の動作を統一的に記述できるインタフェースがあれば有用だと判断し，これを作成した．

実行速度の問題や，開発時間の関係で使いこなすに至らなかったことからデモでは利用しなかったが，tilting laser により検知した障害物を自動で回避して指定位置，姿勢へ手首リンクを移動させる機能もある．そのためのモーションプランナとして OMPL, CHOMP プランナが，また，プランナや IK が生成した軌道を滑らかなスプライン軌道に変換する trajectory filter がある．プランニング環境自体もサービスとして分離されているため，プランナの差し替えや新たなプランナの開発がしやすくなっている．

3.4 認識

PR2 は複数のカメラを持つが，元画像や歪み補正された画像，ステレオの三次元点群などがすべてトピックで発行されている．したがって，コマンドを使ってこれらのトピック名やその型を調べたり，rviz でどのようなデータが取得でできているかを表示することができる．また，画像から対応する世界座標での位置を知りたいときには，カメラのパラメータはトピックで，カメラの座標から世界座標への座標変換はサービスとして取得できるため，これらの情報を取得して，認識した物体の位置や姿勢の計算を行うノードを作る．認識プログラムは基本的にこの枠組みで作ることになる．

今回は，Tabletop-detector パッケージを改造した，投光型ステレオの三次元点群を利用して机上の複数の物品を検出する認識器，しわ情報を使った布認識器，輪郭情報を利用した皿認識器，箱や本などの物品を発見した後，SURF 特徴であらかじめ学習した物品とのマッチングを行う認識器などを作成した．計算結果をトピックとして発行したり，必要に応じて認識結果を返すサービスとすることで把持動作とつながる．

座標変換は，例えば手先からベースリンクというように 2 つの座標系を指定するとそれらの間の変換を計算してサービスとなっている点が使いやすい．また，ロボットにおいて多くの非同期センサデータ，動的に変化する座標変換の同期が重要になる．そのため，それらのデータには高精度のタイムスタンプがついているのに加え，時間がずれた座標変換を知らずに使ってしまわないようサービス側で時間の遅れた変換を検出する機構がある．

認識プログラムの開発にはログの記録・再生機能が便利である．機能が細かく分割されその間の情報がメッセージという形でやりとりされるため，メッセージの記録という単一機能でシステムの実行状態を再現するのに必要な情報の多くを記録できる．タイムスタンプつきで保存されるので，記録時のタイミング通りにメッセージを生じさせることができる．例えば，/narrow_stereo/points，/narrow_stereo/left/camera_info，/narrow_stereo/left/image_rect_color を記録しておけば画像からの特徴量の計算とステレオの点群を併用するプログラムの開発に使える．物体操作につなげるには，ロボットのベースリンクとの変換が残っているとよいので，座標変換である/tf も保存するといった具合に利用する．

3.5 デモタスクの記述

最終的には各要素機能を統合して，片付けタスクとして記述する必要がある．本デモではタスクプランナのようなものはなく，単純に動作を逐次的に記述した．また，エラーチェックに関しては，物品が見つからないという条件のみ検出可能であるため，その場合には次の動作へは移行しない．もしくは，位置を少し変えて再認識を試みた．

このようなタスク実行シーケンス記述を ROS で行う 1 つの方法は，各処理を actionlib として実装し，別にデモ実行ノードを作成し，そのノードから呼び出しをかける実装方法である．プラグ差し込みデモもこのように記述されている．今回のデモでは，我々が EusLisp で行っているように，オペレータが対話環境からコマンドで多くのデモシーケンスを制御できるようにした．つまり，複

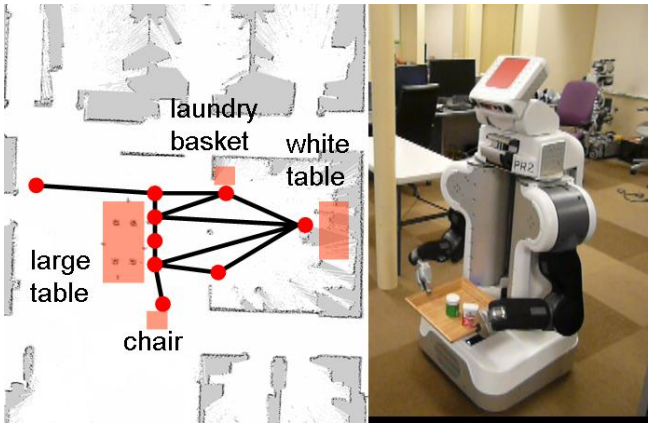

図 5: Demo map and a scene carrying the tray

数の認識器や把持，移動に必要なノードを launch ファイルに記述し起動させた後，Python シェルから対話的にコマンドを実行できるものを作成した．

開発言語の選択肢はいくつかあるが，サンプルコードやドキュメントの量などから，C++と Python を使うのがよいと判断した．性能より開発効率を優先する場合や，対話的部分は Python を使い，認識処理などは C++で実装するのが基本であり，移動，物体操作，認識の各要素機能を python シェルでラップしたものをさらにデモ用のクラスとしてまとめ，対話的に呼び出せるようにした．また，ロボットは片付けタスク実行時に部屋の中を動き回る．そこで，その記述を容易するため，部屋の各所に名前を定義し，その名前で移動を記述した (図 5)．

4 まとめ

本論文では，認識，把持，移動を含む片付け機能の開発を事例として，ROS の分散フレームワークやそのパッケージの特徴を開発の流れの中で説明した．ROS と PR2 のソフトウェアはフレームワークとしての重要性だけでなく，ロボットのソフトウェアを分散システムとして実装する場合に多くの知見を与えてくれる．

謝辞

本研究の遂行にあたり，PR2 ロボットと制御ソフトウェアおよび研究環境を提供していただいた Willow Garage 社の厚意に感謝いたします．

参考文献

[1] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *International Conference on Robotics and Automation*, Open-Source Software workshop, 2009.

[2] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W. Yoon. RT-Middleware: Distributed Component Middleware for RT (Robot Technology). In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005)*, p. 2005, 3555–3560.

[3] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*, 05/2010 2010.

[4] Y.Liu and S.Arimoto. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *International Journal of Robotics Research*, Vol. 11, No. 4, pp. 376–382, 1992.

[5] Open Dynamics Engine ODE. http://ode.org.

[6] T. Matsui and M. Inaba. EusLisp: An Object-Based Implementation of Lisp. *Journal of Information Processing*, Vol. 13, No. 3, pp. 327–338, 1990.