ROSを使って移動ロボットを実現するための マイコン用通信ライブラリの実装と移動ロボットの実現例

光永 法明，杉本 梓 (大阪教育大学)
Noriaki Mitsunaga and **Azusa Sugimoto (Osaka Kyoiku University)**
mitunaga@cc.osaka-kyoiku.ac.jp

概 要

本論文では ROS (Robot Operating System) を使って移動ロボットを実現する際に必要となる ROS パッケージと通信するための，ハードウェアを制御するマイコン用ライブラリを実装したので報告する．Yujin Robot 社の移動ロボット Kobuki の通信プロトコルを実装しているため，ROS の Kobuki 用パッケージが少しの改変で利用できる．実装したライブラリを移動台車を制御するマイコンに載せ移動ロボットを製作した．移動ロボットは Kobuki 用メタパッケージを少し改変して利用することで，本学内を約 50m 移動できた．

1 はじめに

最近ではロボットの開発用フレームワークとして ROS (Robot Operating System) [1]が広く使われている．ROS は研究用プラットホームとして開発されてきたことから，パッケージという単位で機能を分割することができ，研究成果をパッケージとして公開している研究者も多い．また複数のパッケージを組み合わせて複合的な機能を実現したメタパッケージも公開されており，市販のハードウェアプラットホーム用のメタパッケージの中には基本的な移動ロボットのナビゲーションを実現するものもある．ハードウェアにメタパッケージが対応していれば，基本的な機能の実現についてはメタパッケージに頼り，目的とする問題に素早く取り組むことが出来る．一方で，市販のハードウェアプラットホームを利用しない場合には，どのようにパッケージを組み合わせればよいのか ROS になじみがないと戸惑う面もある．

そこで，自作のハードウェアについて，既存のハードウェアプラットホーム用メタパッケージを利用できるようにすることを考える．ROS のパッケージは Ubuntu 等の PC 上の OS で動作し，ハードウェア側のアクチュエータやセンサとは何らかの通信を行うことが一般的である．この通信を模擬できれば，既存のハードウェアプラットホーム用メタパッケージが利用できる．

本論文では，Yujin Robot 社の移動ロボット Kobuki [2]用のメタパッケージが利用できるようにすることとし，その通信プロトコルを模擬するライブラリを実装する．そして実装したライブラリを用いたマイコンボードでモータ制御等をする移動ロボットを製作し，表らの著書 [3]内で紹介されている Kobuki 用のメタパッケージを利用して，本学内で試走させる．以下，まず Kobuki の通信プロトコルを説明し，ライブラリと移動ロボットの実装方法を述べる．そして，必要になった Kobuki 用パッケージの改変と試走結果を述べる．

2 Kobukiの通信プロトコル

Kobuki と ROS のパッケージ (kobuki_driver パッケージ，kobuki_core メタパッケージに含まれる) 間の通信プロトコルは公開 [4]されており，非同期シリアル (115.2kbps, 8bit, non parity, stop bit 1) で通信する．非同期シリアル上にはパケット (図 1) を載せる．パケットの長さは可変長で，プロトコル上の最短は，サブペイロード数が 1，サブペイロードの長さが 3 のときで 7 バイトである．最長はペイロードの長さが 255 バイトの時で 259 バイトである．

サブペイロードには ROS から Kobuki へ送るコマンドパケット (最大 14 種類で，現在規定されているのは 7 種類，表 1) と Kobuki から ROS へ送るフィードバックパケット（最大 21 種類で，現在規定されいてるのは 11 種類，表 2）がある．Kobuki は ID が 1,3,4,5,6,13,16 のフィードバックパケットを ROS からの要求がなくても 20ms 周期 (50Hz) で送信を続ける．ほかのフィードバックパケットは ROS からの要求があるときにのみ返す．

ROS から Kobuki へ送られるコマンドパケットの ID 1 である Base Control には台車の直進速度と回転速度が含まれ単位は mm/s である．したがってマイコン側で，モータの回転数（エンコーダパルス数）に変換する必要がある．一方で Kobuki から ROS へ送るフィード

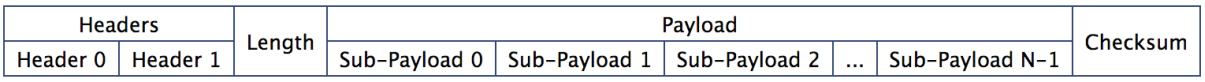

図 1: Kobuki と ROS のパッケージ間のパケット ([4] より引用)

表 1: ROS から Kobuki へ送るコマンドパケット

ID	Name
1	Base Control
3	Sound
4	Sound Sequence
9	Request Extara
12	General Purpose Output
13	Set Controller Gain
14	Get Controller Gain

表 2: Kobuki から ROS へ送るフィードバックパケット

ID	Name
1	Basic Sensor Data
3	Docking IR
4	Inertial Sensor
5	Cliff
6	Current
10	Hardware Version
11	Firmware Version
13	Raw data of 3-axis gyro
16	General Purpose Input
19	Unique Device Identifier (UDID)
21	Controller Info (PID gain)

バックパケットの ID 1 には左右のモータのエンコーダの値 (0〜65535) が含まれる．エンコーダの値から台車の移動量への変換は ROS のパッケージ kobuki_driver で行っているため，台車に合わせて kobuki_driver 内のパラメータを改変する必要がある．

3　ライブラリの実装

ライブラリは Arduino IDE で利用するものとして C++のクラスとして実装する．ライブラリは主に次のクラス内関数で構成されている．シリアルポートから受け取ったバイト列をパケットとして構築する関数 recv()，パケット内のコマンドパケット (サブペイロード) を解釈してクラス内変数に代入する関数 handle() と，シリアルポートへ送るフィードバックパケットをサブペイロードとして詰め込む関数 addBasicSensor(), addDocikngIR(), addInertialSensor() 等とチェックサム計算用関数 setCheckSum() である．ただし handle() 関数でコマンドパケットのうち ID 3, 4, 12 を無視している．

```
while (Serial1.available() > 0) {
  char c = Serial1.read();

  if (fbRecv.recv(c)) {
    // パケットを処理する
    fbRecv.handle();

    if (fbRecv.isSet(fbRecv.flagSetPID)) {
      // PID ゲインの更新
      kPIDs = fbRecv.kPIDs;
      fbRecv.clearFlag(fbRecv.flagSetPID);
    }
    if (fbRecv.isSet(fbRecv.flagUpdateVel)) {
      // 目標速度の更新
      setWheel(fbRecv.spd, fbRecv.radius);
      fbRecv.clearFlag(fbRecv.flagUpdateVel);
    }
    fbRecv.reset();
    break;
  }
}
```

図 2: パケットを受信するプログラム例

コマンドパケットを受信し処理する例を図 2 に示す．シリアルポートから 1 バイトずつ受信し，ステートマシンで構成された recv() 関数に渡す．recv() 関数の戻り値が真であればパケットを受信しているので handle 関数で処理している．

次にフィードバックパケットを送信する例を図 3 に示す．まず Basic Sensor Data パケットを組み立て，その後フィードバックパケットをリセットしてから順に ROS へ送信するサブペイロードとして詰め込んでいく．最後に setCheckSum() 関数でチェックサムを計算・追加し，シリアルポートからパケットを送信する．

```
bs.t = millis();
bs.leftEncoder = encL.read();
bs.rightEncoder= -encR.read();

// パケットの組み立て
fbPacket.reset();
fbPacket.addBasicSensor(bs);
fbPacket.addDockingIR(0, 0, 0);
fbPacket.addInertialSensor(0, 0);
fbPacket.addCliff(0, 0, 0);
fbPacket.addCurrent(0, 0);
fbPacket.addRawGyro2(gyroID,0,0,0,0,0,0);
         // Gyro のタイムスタンプをダミーで更新
gyroID += 2;
fbPacket.addGeneralPurposeInput(0,0,0,0,0);

// PC から要求されているパケットがあれば追加
if(fbRecv.isSet(fbRecv.flagHardwareVersion)){
  fbPacket.addHardwareVersion(1, 1, 0);
  fbRecv.clearFlag(fbRecv.flagHardwareVersion);
}
if(fbRecv.isSet(fbRecv.flagFirmwareVersion)){
  fbPacket.addFirmwareVersion(1, 2, 9);
  fbRecv.clearFlag(fbRecv.flagFirmwareVersion);
}
if (fbRecv.isSet(fbRecv.flagUDID)) {
  fbPacket.addUDID(1, 2, 3);
  fbRecv.clearFlag(fbRecv.flagUDID);
}
if (fbRecv.isSet(fbRecv.flagPID)) {
  fbPacket.addControllerInfo(1, kPIDs.Kp,
                          kPIDs.Ki, kPIDs.Kd);
  fbRecv.clearFlag(fbRecv.flagUDID);
}
// チェックサムの追加
fbPacket.setCheckSum();

// パケットの送信
Serial1.write(fbPacket.packet,
              fbPacket.len());
```

図 3: フィードバックパケットを送信するプログラム例

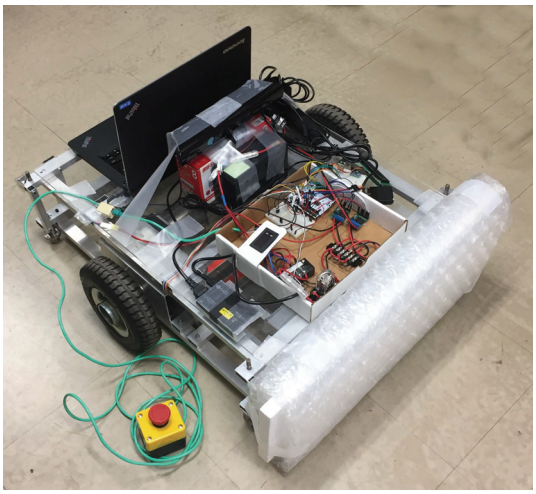

図 4: PC や Kinnect を載せた移動ロボット

4 移動ロボットの実装

メカニカルな部分 (モータ・エンコーダを含む) は著者等が以前に製作したロボット [5]の台車部分を利用している (図 4)．Kobuki との差異を表 3 に示す．これらの差異を制御マイコンもしくは，ROS のメタパッケージで吸収する必要がある．台車には ROS を載せた PC (Lenovo ThinkPad E420, Ubuntu 14.04 LTS, ROS Indigo)，Microsoft Kinect, モータと Kinect, PC 駆動用バッテリ (12V, 7.2Ah), 12V から AC100V をつくるインバータ，モータ制御用マイコンなどが載っている．

モータ制御用マイコンには STmicro の Nucleo-F103RBT6 マイコンボード (MCU: Coretex-M3, max 72MHz, ROM 128kB, RAM 20kB) を選択した．理由はハードウェアエンコーダを内蔵しており，3.3V 動作でありながら 5V トレラントピンが豊富であり，Arduino IDE で開発できて十分な速度で動作するからである．配線図を図 5 と図 6 に示す．配線図にはないがマイコンボード上の USB シリアルでは 115.2.kbps での連続通信時に不具合が生じたので FTDI の FT232 ベースの USB シリアル変換ケーブルを利用している．また ROS 非利用時にリモートコントロールできるようにヴィストンのゲームパッド VS-C1 用受信機を載せているが，衝突検出スイッチやジャイロなどは搭載していない．そのため，センサ類はダミーのデータを返す．マイコン上ではモータの PID 制御 (制御周期 50ms)，Kobuki の通信処理（周期 20ms）と VS-C1 の受信をすべてメインループ内に記述している．VS-C1 の受信にはライブラリ [6] を利用している．

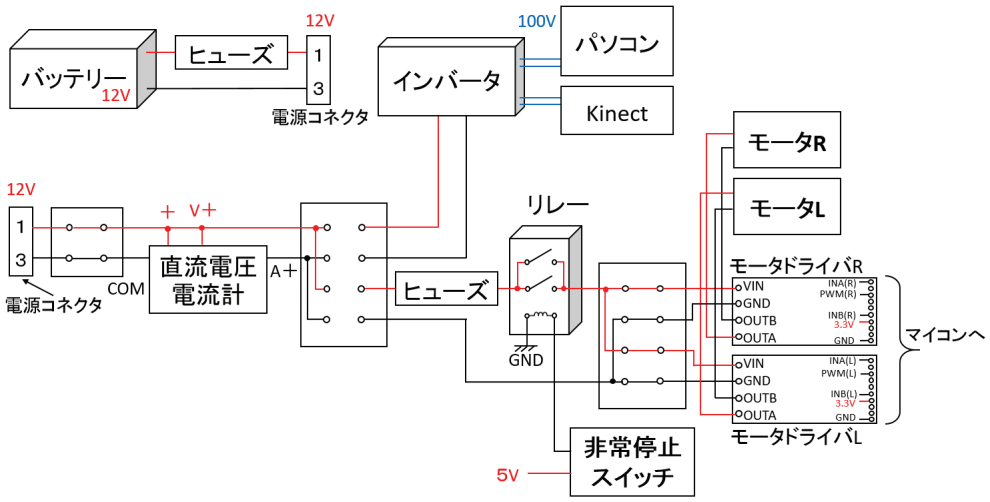

図 5: 電源とモータドライバ周りの配線図

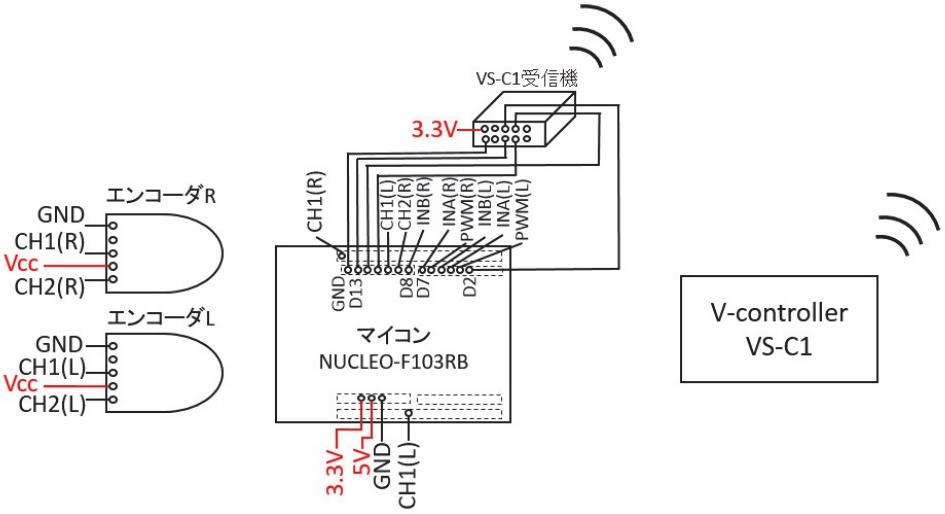

図 6: エンコーダとマイコン周りの配線図

5 Kobuki 用パッケージの改変

製作した移動ロボットを屋内でナビゲーションさせるために表らの著書 [3]にある kobuki_navigation メタパッケージを利用することにした．これは公開されている git リポジトリ [7]に含まれている．このメタパッケージのほかに kobuki_node メタパッケージ (kobuki_driver パッケージ含む) が必要である．また kobuki_navigation では Kobuki に北陽電機の LRF である UTM-30LX を載せることを前提としている．

表 3 の差異を吸収し LRF ではなく Microsoft Kinect を利用するために次の改変を必要とした．ロボットの形状と大きさについては形状は円形のままとしたが，経路計画時のパラメータを決める costmap_common_params.yaml にあるロボットの半径 (robot_radius) と衝突防止のための半径 (inflation_radius) を変更した．駆動輪直径，間隔，エンコーダ係数については kobuki_driver パッケージ内の diff_driver.cpp に記述があるので変更した．

Kinect を利用するためには，ロボット上での Kinect の位置を定義する tf を発行する必要がある．そのため kobuki_navigation メタパッケージに含まれる kobuki_tf パッケージ内の tf_broadcaster.cpp に追加した．また Kinect で得られるのは 3 次元距離画像 (point cloud とよばれる点群画像) で，ナビゲーションに必要とされる LRF 等で得られる 2 次元距離画像ではない．そこで depthimage_to_laserscan パッケージ [8]を利用し変換する．

ほかにシリアル通信部分（パケットの受信部分）に特定のタイミングで顕在化する不具合があり kobuki_driver パッケージの packet_finder.cpp を修正する必要があった．

表 3: Kobuki と製作したロボットの主要な差異

	Kobuki	製作したロボット
形状	円形	矩形
大きさ [mm]	直径 352	幅 600 ×全長 700
駆動輪直径 [mm]	70	210
駆動輪間隔 [mm]	230	640
エンコーダ係数	0.0024369...	0.00026329...
ジャイロ バンパーセンサ 脱輪センサ クリフセンサ 発音など	あり	なし

6　本学内での試走

製作した移動ロボットを図 7 の地図上の A 点と B 点間を試走させる．場所は本学内の C4 棟 2F の屋内である．A 点と B 点間の距離は約 50m である．ナビゲーション用の地図は環境を実測したものであり，図 7 はナビゲーション用地図に部屋名等を記入したものである．環境中に移動ロボットが乗り越えにくい段差があったため，そこに簡易スロープをとりつけた．またローカルプランナがグローバルプランナの計画した経路を追従するパラメータ等を事前に調整した．

A 点から B 点（往路）と，B 点から A 点（復路）のナビゲーションを各 2 回試した．その結果，往路は 2 回ともナビゲーションができたが，復路は 1 回のみであった．その原因は点 A のある廊下の幅がロボットの幅に対して狭く，廊下への進入角度が悪いため壁に近づきすぎてしまったためである．これはロボットの大きさを変えるか，ナビゲーションに使っているローカルプランナを改変する必要があると考える．一方で，通信ライブラリを含めロボットの動作に大きな不具合はみられなかった．

7　まとめ

本論文では既存のロボットプラットホーム Kobuki 用の ROS パッケージが利用出来る移動ロボットを実現するためのライブラリを実装を提案し，それを利用した移動ロボットがナビゲーションできることを示した．本稿が ROS パッケージを利用した移動ロボットの製作の参考になれば幸いである．ライブラリについては github 等で公開していきたいと考えている．

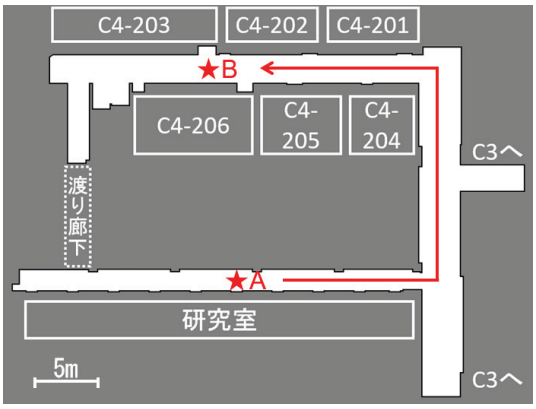

図 7: 試走した環境の地図．ナビゲーション用に用意したものに部屋名等を記載している．C4 は試走環境の棟を C3 は隣の棟を表す．

参考文献

[1] ROS: http://wiki.ros.org/ (2017/3/1 閲覧)

[2] Yujin robot: iClebo Kobuki, http://kobuki.yujinrobot.com/about2/ (2017/3/1 閲覧)

[3] 表 允哲, 倉爪 亮, 渡邊 裕太: 詳説 ROS ロボットプログラミング. Kurazume laboratory, 2015. (http://irvs.github.io/rosbook_jp/)(2017/3/1 閲覧)

[4] D. Stonier, Y. Ju, and J. S. Simon: kobuki_driver documentation, Appendix: Protocol specification, (https://yujinrobot.github.io/kobuki/doxygen/enAppendixProtocolSpecification.html) (2017/3/1 閲覧)

[5] 光永法明，江崎大樹，伊藤直也，按田翔悟: 人を乗せて案内をするロボットの提案と試作. ロボティクス・メカトロニクス講演会'10, 2P1-E30, 2010.

[6] Bill Porter: PlayStation 2 Controller Arduino Library v1.0, http://www.billporter.info/2010/06/05/playstation-2-controller-arduino-library-v1-0/ (2017/3/1 閲覧)

[7] rosbook_kobuki の git リポジトリ: https://github.com/irvs/rosbook_kobuki.git (2017/3/1 閲覧)

[8] depthimage_to_laserscan パッケージ: http://wiki.ros.org/depthimage_to_laserscan (2017/3/1 閲覧)