

汎用組込み OS 向けロボット制御ライブラリの設計

西川遼† 高橋淳二† Guillaume Lopez† 狐崎直文† 戸辺義人†

青山学院大学理工学部情報テクノロジー学科†

1. はじめに

制御 API を備えたロボットが身近になり、マイクロプロセッサ等で自動操縦するケースが多い中、自動操縦のためのアプリケーションのソースコードを再利用することが求められている。そこで我々は、ハードウェア抽象化を目的としたロボット制御ライブラリ MIKAN(Micro In-Kernel Assembled Native codes)を提案する。本ライブラリには下位 API と上位 API の 2 つの抽象度の API を定義し、アプリケーションを簡潔に記述する場合と細かく制御を記述する場合の両方に対応する。本稿では、 μ T-Kernel¹⁾用ライブラリの設計、実装、実験および評価結果について述べる。

2. 関連技術

本研究の関連技術として、ロボット制御用のメタオペレーティングシステムである ROS²⁾がある。ROS は、各デバイス毎のハードウェアの差異を吸収し、低レベルのデバイス制御、制御ライブラリ、そしてデバイス間のメッセージ通信機能を提供している。また、ROS は Ubuntu を始めとして OS X, Fedora, Windows 等の OS をサポートしており、これらの OS が動作している汎用機上で動かすことを想定している。一方、MIKAN は、ハードウェアの差異を吸収する点では ROS と共通であるが、リアルタイム OS 上で動作させる利点を生かし、精密な制御を行うことに重きを置いている。

3. 設計

本章では、MIKAN の設計方針について述べる。

3.1 設計方針

MIKAN は、ユーザがロボット制御のプログラムを容易に記述できることを目的としている。したがって、ロボット毎のインタフェースの差異、センサの種類、アクチュエータの差異を吸収し、抽象度の高い API を提供する。また、

API を下位 API と上位 API に分類し、さらに上位 API をその役割ごとに分類する。以下に API の体系図を示す。

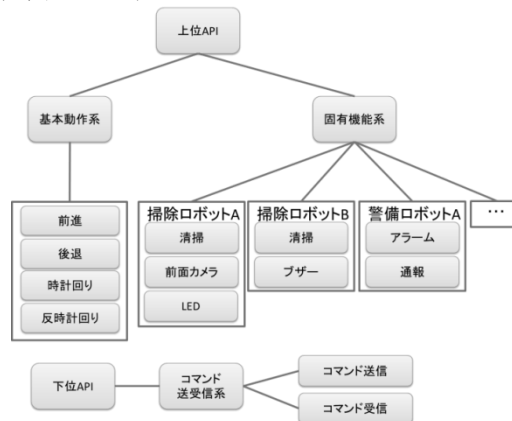


図 1 API 体系図

基本動作系 API は `move_forward()`, `move_back()`, `turn_right()`, `turn_left()` の 4 つの関数から構成され、アクチュエータの動作制御を行う。固有機能系 API はロボット固有の機能を使うための関数群である。下位 API はマイクロプロセッサと制御対象との通信のための API である。

3.2 Roomba Open Interface

iRobot Roomba^{®3)} は ROI (Roomba Open Interface⁴⁾)という制御 API を備えている。Roomba にはシリアルポートがあり、シリアルポートを通して PC やマイクロプロセッサから ROI コマンドを送信することにより Roomba を制御することができる。

3.3 μ T-Kernel

μ T-Kernel は小規模な組込みシステムを対象としたリアルタイムオペレーティングシステムである。

4. 実装

本章では、MIKAN を実装について述べる。

4.1 Roomba Open Interface 用実装

ROI には、ミリメートル毎秒で速度を指定し、指定した速度で移動する Drive コマンドが存在する。しかし、Drive コマンドで指定した速度は Roomba 内で 1 の位を四捨五入されるため実際

はセンチメートル単位での速度指定となる。また、Roomba が走行する路面や車輪の状態により、指定した速度に一定のバイアスがかかった状態で実際の速度が算出されると考えた。以上から、Roomba を高精度で移動制御するためにはマイクロプロセッサ側で補正を行う必要がある。以下に補正を組み込んだ上位 API である `move_forward()` の疑似コードを示す。

```

move_forward(UH n, UH v){
  /* n := 移動距離[mm] */
  /* v := 移動速度[mm/s] */
  send_cmd(drive(速度:v, 方向:前));
  tk_dly_tsk(((10000 * n) / ((10 * v) +
(CORRECTION))););
  send_cmd(stop);
}

```

図 2 `move_forward()` 用補正式

`tk_dly_tsk()` 内では、速度の補正係数として `CORRECTION` を定義した。

4.2 FM3 基板用実装

本研究で使用した FM3 基板から Roomba にコマンドを UART で送信するためには、以下の手順で初期化を行う必要がある。

- ① FM3 基板の I/O ポートは GPIO 機能と周辺機能を選択して使用する。そのため、UART 通信で使用する I/O ポートの端子を周辺機能の入力端子として使用するよう、PFR レジスタを設定する。
- ② EPFR レジスタで、シリアルデータ出力に使用する端子と入力に使用する端子を設定する。
- ③ ADE レジスタで、使用する端子の AD コンバータを無効にする。

次に、使用する端子における UART の設定を行う。

- ④ SCR レジスタのプログラマブルクリアビットを 1 とし、プログラマブルクリアを行う。
- ⑤ SMR レジスタで UART の詳細設定を行う。Roomba は Least Significant Bit ファーストであるため、SMR に書込む値は 0x01 であった。
- ⑥ SSR レジスタに 0x00 を書込み、各フラグを初期化する。
- ⑦ ESCR レジスタで UART のさらに詳細な設定を行う。Roomba は NRZ フォーマットであるため ESCR に書込む値は 0x00 であった。
- ⑧ BGR レジスタにシリアルクロックの分周比を書込む。
- ⑨ SCR レジスタの受信動作許可ビットと送信動作許可ビットを 1 とし、UART 通信を開始する。

以上により Roomba との UART 通信が可能となる。

5. 評価

本章では、`move_forward()` で Roomba の動作を計測し、本ライブラリの有用性を示す。

5.1 実験方法

Spansion 製 FM3 USB Stick Board を実験に用いた。`move_forward()` 関数で 1000mm の距離を 100mm/s で走行させる試行を、補正を行わない場合と行う場合で各 10 回ずつ行い、計測した。

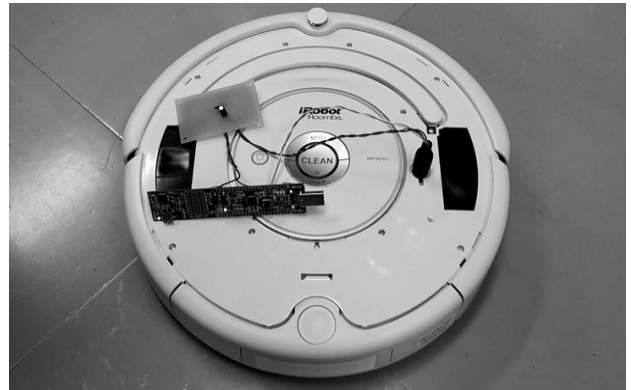


図 3 μ T-Kernel による Roomba 制御の様子

5.2 実験結果

実験を行った結果、補正を行わなかった場合、平均誤差は 75mm、標準偏差は 13mm となった。一方、補正を行った場合、平均誤差は 10mm、標準偏差は 14mm となった。以上から、補正を行うことで Roomba の走行速度にかかる一定のバイアスを軽減できた。

6. むすび

本稿では μ T-Kernel 上で動作するロボット制御ライブラリを設計、実装し、評価を行った。今後はフィードバック制御を用い、より汎用性が高いミドルウェアとしていく。

7. 参考文献

- [1] μ T-Kernel Specification http://www.t-engine.org/wp-content/themes/wp.vicuna/pdf/specifications/en_US/TEF020-S004-02.00.00_en.pdf
- [2] ROS <http://www.ros.org/>
- [3] iRobot Roomba® <http://www.irobot.com/us/learn/home/roomba.aspx>
- [4] iRobot® Roomba500OpenInterface (OI) Specification http://www.irobot.lv/uploaded_files/File/iRobot_Roomba_500_Open_Interface_Spec.pdf
- [5] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E. Wheeler, Rob. and Ng, A. Y. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* Vol. 3, no. 3.2, 2009