Spur 走行制御コマンドの基本

大島 章 OOS(2007), 斎藤 昌和 SIT(2008), 渡辺 敦志 AWD(2010), 識名 拓 STK(2011)

2012.12.5

1 山彦の走行系

1.1 山彦の動作と座標系

山彦は、独立に制御される２つの駆動輪によって動作（移動）します。この構成では、基本的に、平面な床や地面を走行することが想定されており、これに準じた走行制御コマンドが、標準で実装されています。

二次元平面上で動作する山彦では、ロボットを起動した時点の進行方向を X 軸、それに垂直な平面上の軸を Y 軸と定義します。さらに、X 軸に対するロボットの姿勢を θ とし、ロボットの位置・姿勢を (X, Y, θ) の 3 次元で表現します。

Spur を利用するために、複数の座標系が利用できます。GL 座標系、LC 座標系、FS 座標系です。（図 1）

GL(GLobal coordinate system) 座標系：プログラマが定義できる世界座標系。

LC(LoCal coordinate system) 座標系：プログラマにより設定・変更できるローカルな座標系。

FS(FrontSide coordinate system) 座標系：ロボットに張り付けられ、ロボットと共に移動する座標系。

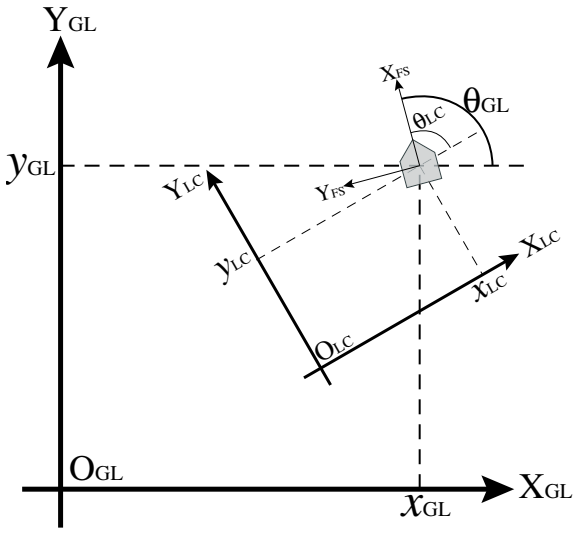

図 1: ロボットの座標系

2 Spur コマンド

山彦は、その構造から、直接的に移動できる方向が制限されます。普段私たちが利用している車と同じく、「真横に進む」ことができません。このような車体を「ノンホロノミックな性質を持つ車両」、と呼びます。これから紹介していくSpur は、２つの駆動輪の走行制御を行ったり、座標系や状態変数を管理・変更できるコマンド群です。

Spur では、ロボットの動作を 2 次元平面上の幾何図形（直線と円弧）で指示します。プログラマは、直線と円弧の組み合わせでロボットの走行軌跡を考えれば良く、実際に車輪をどのように回すかを考える必要はありません。（逆に言えば、Spur を利用する限り実際のタイヤの動きを直接制御することはできません）

それでは、具体的なプログラムを織り交ぜつつ、Spur について説明していきます。

2.1 Spur の概要

山彦の動作プログラムは、専用に用意された関数を利用しながら、C 言語の文法に従って書きます。Spur コマンドは、主に 3 つの範疇に分けられます。

走行制御コマンド：

直進走行、円弧追従、その場回転、停止などの走行制御をおこなう。

状態取得コマンド：

ロボットの現在位置・姿勢、速度、加速度などの状態を取得する。

パラメータ変更コマンド：

ロボットの位置・姿勢、速度、加速度、座標系などのパラメータを変更する。

Spur コマンドを利用するには、ソースの最初に ypspur.h というファイルをインクルードする必要があります。また、yp-spur のコマンドを使用する前に、Spur_init() という関数を実行し、ロボットの速度、角速度、加速度、角加速度を指定する必要があります。後は、C 言語の形式に合わせて、Spur を駆使したソースを作成しましょう。コンパイルするには、通常の C 言語をプログラムと同様に、gcc などを使用します。

2.2 Spur の使い方

プログラマが発行した Spur コマンドを解釈して、モータ制御を行っているマイクロコントローラに指示を出すために、PC 上のバックグラウンドで、ypspur-coordinator というプログラムを実行しておく必要があります。

```
1 $ ypspur-coordinator -p /usr/local/share/robot-params/beego.param -d /dev/ttyUSB0
```

beego.param の部分は使用するロボットの型に合わせて変更してください。パラメータファイルの指定を間違えるとロボットが暴走する可能性があるので注意して下さい。また、"/usr/local/share/robot-params/"の部分は省略することもできます。

"-d /dev/ttyUSB0"の部分は、通信ポートの指定です。デフォルト（指定しない場合）は"/dev/ttyUSB0"となっています。プログラムの途中で USB を引き抜いたときなどは、ポート番号が変わるときがあるので他のポートを試してみて下さい。

ypspur-coordinator を起動したら、ロボットのタイヤを手で少し回してみてください。制御がかかっていて、回しにくいはずです。(なっていなかったら、Ctrl+c で ypspur-coordinator を止めて、もう一度起動してみましょう。) これで、Spur を使ったプログラムを実行する準備ができました。

2.3 走行制御コマンドの基本

2.3.1 簡単なプログラムの例

それでは、具体的にプログラムを見ていきます。最初は、こんな動作を実行することを目標にしましょう。

「1m 直進してから左に 90 度回転し、さらに 1m 進んだあと、停止する」

「直進」「回転」「停止」のキーワードから、走行命令を羅列すると、以下のようなソースができそうです。コンパイルして実行してみてください。

```
#include <ypspur.h>

void main(void)
{
    // 初期化
    Spur_init();

    // ロボットの制御パラメータの設定
    Spur_set_vel( 0.3 );              // 速度を0.3m/に設定s
    Spur_set_accel( 1.0 );            // 加速度を1.0m/に設定ss
    Spur_set_angvel( 1.5 );           // 角速度を1.5rad/に設定s
    Spur_set_angaccel( 2.0 );         // 角加速度を2.0rad/に設定ss

    // 座標系の設定
    Spur_set_pos_GL( 0, 0, 0 );       // スタート地点を原点に設定

    // ロボットの行動
    Spur_line_GL( 0, 0, 0 );          // 直進
    Spur_spin_GL( 3.14/2 );           // ９０度回転
    Spur_line_GL( 1.0, 0, 3.14/2 ); // 直進
    Spur_stop();                      // ただちに停止
}
```

最初にある、Spur_init() という関数は、ypspur-coordinator との通信を確立するコマンドです。次はロボットの制御パラメータの設定です。ここではロボットを実際にどの様に動かしたいかを設定します。プログラムの目的に応じて適切に設定して下さい。また、これらの設定を行わないと意図した通りにロボットが動きませんので注意して下さい。

座標系は、ロボットがどこにいるかを考え、適切に設定します。座標系については、別の回で改めて説明します。この例では、プログラムを起動したときのロボットの位置を、GL 座標系の原点としています。この設定を行わないと、ypspur-coordinator が覚えている位置が GL 座標系の原点となります。

2.3.2 プログラムのコンパイル方法

例えば、test.c というファイルを、実行ファイル名 test としてコンパイルするには以下の操作をします。

```
$ gcc test.c -o test -lypspur
```

-lyspur の部分は、yp-spur のライブラリをリンクするという指示を意味しています。

しかし、**このプログラムは正常に動作しません!**

理由は、「Spur は、同時に 1 つの走行コマンドしか受け付けない」ことにあります。ここでは、「立て続けに異なる走行コマンドを出しているため、古い（実行中の）走行命令が、新しい走行コマンドによって上書きされている」といった事態が起こっているのです。つまり、一番最後に発行した停止コマンド (*Spur_stop*) が、ロボットの行動として現れるだけになります。

では、このプログラムを少し改良しましょう。sleep 関数を利用して、コマンドを発行する間隔を伸ばす処理を加えてみます。(ちなみに 1 秒以下の細かさで sleep するには、usleep 関数を使用して、マイクロ秒単位で待機する時間を指定します。ただし、usleep 関数は今後廃止される可能性があり、nanosleep 関数への移行する必要があるかもしれません。)

少し広い場所で実行してみましょう。

```
#include <ypspur.h>

void main(void)
```

```
{
    // 初期化
    Spur_init();

    // ロボットの制御パラメータの設定
    Spur_set_vel( 0.3 );              // 速度を0.3m/に設定s
    Spur_set_accel( 1.0 );            // 加速度を1.0m/に設定ss
    Spur_set_angvel( 1.5 );           // 角速度を1.5rad/に設定s
    Spur_set_angaccel( 2.0 );         // 角加速度を2.0rad/に設定ss

    // 座標系の設定
    Spur_set_pos_GL( 0, 0, 0 );       // スタート地点を原点に設定

    // ロボットの行動
    Spur_line_GL( 0, 0, 0 );          // 直進
    sleep( 2 );                       // ２秒間待機
    Spur_spin_GL( 3.14/2 );           // ９０度回転
    sleep( 2 );                       // ２秒間待機
    Spur_line_GL( 1.0, 0, 3.14/2 ); // 直進
    sleep( 2 );                       // ２秒間待機
    Spur_stop();                      // 直ちに停止
}
```

このプログラムでも、まだ十分に動作しません。

走行コマンドの間に sleep を挟み、処理が次の命令に移る前に、十分な時間を取るように設定しました。このプログラムでは、ロボットは直進、回転、直進を順に行おうとし、だいたいそのように動作します。が、まだ不十分です。

理由は、Spur の走行コマンドの ( ) 内に入れた数値が、処理の終了条件とは関係ないからです。直線追従コマンド *Spur_line_GL*(1,0,0) は、ロボットが走行するべき直進経路を指定するものです。ロボットの到達位置が $x = 1m$ に到達しようがしまいが、ただ直進を続けるだけなのです。つまり、ここまでに学んだコマンドだけでは、「いつ停止するか」などの行動の終了条件を、明確に与えることができないのです。

2.4 状態取得コマンドの基本

これまでに説明した通り、走行制御コマンドは、命令を出すところまでが仕事です。「○○まで走ったら停止しろ」といった類の動作を行わせるには、ロボットの状態を常に監視し、しかるべきときに新たな命令を出すようなプログラムを組む必要があります。

そこで、状態取得コマンドの登場です。Spur は、車輪の動きを監視しながら、ロボットがどの様に動き、今どこにどんな姿勢で存在しているかを推定しています（この推定手法をオドメトリと呼びます）。状態取得コマンドを使えば、この推定結果を、必要なタイミングで取得できるのです。

それでは、先ほどのプログラムを状態取得コマンドを交えて書き換えます。

```
#include <ypspur.h>

void main(void)
{
    // 初期化
    Spur_init();

    // ロボットの制御パラメータの設定
    Spur_set_vel( 0.3 );              // 速度を0.3m/に設定s
    Spur_set_accel( 1.0 );            // 加速度を1.0m/に設定ss
    Spur_set_angvel( 1.5 );           // 角速度を1.5rad/に設定s
    Spur_set_angaccel( 2.0 );         // 角加速度を2.0rad/に設定ss

    // 座標系の設定
    Spur_set_pos_GL( 0, 0, 0 );       // スタート地点を原点に設定

    // ロボットの行動
    Spur_line_GL( 0, 0, 0 );                        // 直進
    //  x = 1 mを越えたかどうかを判定
    while(1){
        if( Spur_over_line_GL( 1.0, 0, 0 ) )
```

```
            break;
        usleep(10000);
    }

    Spur_spin_GL( 3.14/2 );                          // ９０度回転
    // θ＝９０度になったかどうかを判定
    while(1){
        if( Spur_near_ang_GL( 3.14/2, 0.1 ) )    // 許容誤差は０．１ｒａｄ（約５．７度）
            break;
        usleep(10000);
    }

    Spur_line_GL( 1.0, 0, 3.14/2 );                  // 直進
    // ｙ＝１ｍを越えたかどうかを判定
    while(1){
        if( Spur_over_line_GL( 1.0, 1.0, 3.14/2 ) )
            break;
        usleep(10000);
    }

    Spur_stop();                                     // ただちに停止
}
```

今度はまともに動きます。

ここでは、while 文で無限ループを生成し、0.01sec 毎にロボットの状態を監視しています。ロボットが指定された位置・姿勢に到達したところで、ループを抜け、次の処理の移行しています。これでようやく、当初の目的である「1m 直進してから左に 90 度回転し、さらに 1m 進む」プログラムが実装できました。

ところでこの例では、ロボットは 1m をややオーバーシュートして停止し、旋回して少し戻りながら直進します。これは、ロボットの加速度に制限があり、急には止まれないために発生しています。もっとスムーズに走行するためにはどうすればよいか、後で考えてみましょう。(3 課題 2)

さて、上記のプログラムですが、ロボットの動作が単純な割りには行数が多いですね。少し書き方を工夫してみます。

```
#include <ypspur.h>

void main(void)
{
    Spur_init();
    Spur_set_vel( 0.3 );              // 速度を0.3m/に設定s
    Spur_set_accel( 1.0 );            // 加速度を1.0m/に設定ss
    Spur_set_angvel( 1.5 );           // 角速度を1.5rad/に設定s
    Spur_set_angaccel( 2.0 );         // 角加速度を2.0rad/に設定ss
    Spur_set_pos_GL( 0, 0, 0 );                      // スタート地点を原点に設定

    Spur_line_GL( 0, 0, 0 );                         // 直進
    // ｘ＝１ｍを越えたかどうかを判定
    while(!Spur_over_line_GL( 1.0, 0, 0 ) )
        usleep(10000);

    Spur_spin_GL( 3.14/2 );                          // ９０度回転
    // θ＝９０度になったかどうかを判定
    while( !Spur_near_ang_GL( 3.14/2, 0.1 ) )    // 許容誤差は０．１ｒａｄ（約５．７度）
        usleep(10000);

    Spur_line_GL( 1.0, 0, 3.14/2 );                  // 直進
    // ｙ＝１ｍを越えたかどうかを判定
    while( !Spur_over_line_GL( 1.0, 1.0, 3.14/2 ) )
        usleep(10000);

    Spur_stop();                                     // 直ちに停止
}
```

一つ前と全く同様に動きます。

状態取得コマンドを、while 文の条件式に入れてしまいました。余計な break などが不要になり、先ほどよりもソースコードがすっきりしたことが判ります。山彦セミナーの最終課題では、多少長いプログラムを作成する必要があるので、こういった工夫も知っておいて損はないでしょう。

2.5 パラメータ変更コマンドの基本

Spur コマンドを用いて、ロボットの動きを制御できるようになりました。本節では、ロボットの動作中に、以下のようなパラメータを変更する Spur コマンドについて説明します。

- ロボットの速度・加速度
- ロボットの位置・姿勢
- 座標系

さて、以下のような目標動作を設定します。
「秒速 300mm で 1m 直進してから左に 90 度回転し、速度を秒速 500mm に変更してさらに 1m 進む」
このソース例は以下の通りです。

```
#include <ypspur.h>

void main(void)
{
    // 初期化
    Spur_init();

    // ロボットの制御パラメータの設定
    Spur_set_vel( 0.3 );              // 速度を0.3m/sに設定
    Spur_set_accel( 1.0 );            // 加速度を1.0m/ss に設定
    Spur_set_angvel( 1.5 );           // 角速度を1.5rad/sに設定
    Spur_set_angaccel( 2.0 );         // 角加速度を2.0rad/ss に設定

    // 座標系の設定
    Spur_set_pos_GL( 0, 0, 0 );       // スタート地点を原点に設定

    // ロボットの行動
    Spur_line_GL( 0, 0, 0 );                          // 直進
    // ｘ＝１ｍを越えたかどうかを判定
    while( !Spur_over_line_GL( 1.0, 0, 0 ) )
        usleep(10000);

    Spur_spin_GL( 3.14/2 );                           // ９０度回転
    // θ＝９０度になったかどうかを判定
    while( !Spur_near_ang_GL( 3.14/2, 0.1 ) )         // 許容誤差は０．１ｒａｄ（約５．７度）
        usleep(10000);

    Spur_set_vel( 0.5 );                              // 並進速度の再設定

    Spur_line_GL( 1.0, 0, 3.14/2 );                   // 直進
    // ｙ＝１ｍを越えたかどうかを判定
    while( !Spur_over_line_GL( 1.0, 1.0, 3.14/2 ) )
        usleep(10000);

    Spur_stop();                                      // 直ちに停止
}
```

Spur_set_vel() コマンドを利用することで、ロボットの走行速度を随時変更することができます。
他にも、座標系を張りかえるコマンドを使用することで、複雑な動作を簡単に書くことや、センサの情報を使った結果を動作にうまく反映することができるようになります。これについては山セミの別の回で説明します。

3 課題

課題 0：YPSpur を B-Loco ボードに書き込んでみよう！ YP-Spur は PC とマイコン両方を合わせて動く構成となっています。まだ皆さんのロボットに搭載されているマイコン上にはまだプログラムが書かれていませんので、internal ページ -> Robot Platform Project の wiki -> YPSpur/インストール手順 -> SH_vel の書き込み (初回のみ) に書かれている手法に従ってファームウェアをインストールして下さい。

課題 1：今回、紹介されたソースを一通り試してみてください。

課題 2：今回紹介しなかった関数も使ってみましょう。上記のソースで利用した関数やアルゴリズムは、目標動作を実現するための一例に過ぎません。次のような条件のもとでは、どんなソースになるでしょうか。考えて実験してみましょう。

- *Spur_over_line* の代わりに、*Spur_get_pos* や *Spur_near_pos* を利用する。
- *Spur_spin* を利用せず、直線の乗り換えを円弧追従コマンド *Spur_circle* で行う。
- *Spur_spin* も *Spur_circle* も利用せず、直線をオーバーシュートせずに滑らかに乗り換える。

課題 3：1m 四方の四角形を描くように、走行するプログラムを作ってみてください。

4 おまけ

これまでのプログラムでも動きますが、プログラムを Ctrl-C で途中で停止させる場合、最後のコマンドが残ってしまい、動き続けてしまいます。おまけで、ロボットを Ctrl-C で止めても正しく停止するプログラムのテンプレートです。詳しくは"linux signal"とかで調べてみてください。

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <ypspur.h>

// Ctrl-が押されたときに呼び出す関数C
void ctrlC( int aStatus )
{
    Spur_stop();                        // 直ちに停止
    signal( SIGINT, NULL );
    exit(aStatus);
}

// Ctrl-が押されたときにCctrlC()を呼ぶようにする関数
void setSigInt(  )
{
    signal( SIGINT, ctrlC );
}

// 関数main
int main(void)
{
    // 初期化
    if( !Spur_init(  ) )
    {
        fprintf( stderr, "ERROR␣:␣cannot␣open␣spur\n");
        return -1;
    }
    setSigInt(  );

    // ロボットの制御パラメータの設定
    Spur_set_vel( 0.3 );                // 速度を0.3m/に設定s
    Spur_set_accel( 1.0 );              // 加速度を1.0m/に設定ss
    Spur_set_angvel( 1.5 );             // 角速度を1.5rad/に設定s
    Spur_set_angaccel( 2.0 );           // 角加速度を2.0rad/に設定ss

    // 座標系の設定
    Spur_set_pos_GL( 0, 0, 0 );         // スタート地点を原点に設定

    // ロボットの行動
    //-----------------------------------------


    // 終了処理
    //-----------------------------------------
    Spur_stop();                        // 直ちに停止
    return 0;
}
```