Ver 1.0

ライブラリ説明

谷川郁太
tanigawa@f.ait.kyushu-u.ac.jp

ロボット関連

移動コマンド (関数)

- void Create2.GoForward(double velocity)
  - 指定した速度[mm/s]でまっすぐ前進
- void Create2.GoBackward(double velocity)
  - 指定した速度[mm/s]でまっすぐ後進

- void Create2.TurnLeft(double angularVelocity)
  - 指定した角速度[度/s]でその場で左に曲がる
- void Create2.TurnRight(double angularVelocity)
  - 指定した角速度[度/s]でその場で右に曲がる

- void Create2.Stop()
  - その場に停止する

移動コマンド (関数)

- void Create2.OI.Drive(short velocity, short radius)
  - iRobot Create 2の基本的な移動コマンド
  - velocity: 速度[mm/s]
    - 正の値で直進、負の値で後進
  - radius: 半径(曲がる量) [mm]
    - ここで指定した半径の円に沿うように、ロボットを移動させる
    - 与える値が0に近いほどよく旋回する
    - 正の値で正面から見て左方向、負の値で右方向に曲がる
    - 0が与えられた場合は直進する

- void Create2.OI.DriveDirect(short rightVelocity, short leftVelocity)
  - 左右の車輪にそれぞれ速度[mm/s]を指定することで移動するコマンド

センサ情報 (変数)

- bool Create2.OI.LeftBumper
- bool Create2.OI.RightBumper
  - 左右のバンパの衝突検知
  - 何かにぶつかっていれば真、そうでなければ偽となる
- bool Create2.OI.LeftWheelDrop
- bool Create2.OI.RightWheelDrop
  - 左右の車輪が下がっているかどうか
  - 下がっていれば真、そうでなければ偽となる
- int Create2.OI.LeftEncoderCounts
- int Create2.OI.RightEncoderCounts
  - 左右の車輪のモータの回転数
  - この数値から、ロボットの移動距離や角度が分かる

位置情報 (変数)

- double Create2.X
- double Create2.Y
  - ロボットが開始位置からどれだけ移動したかをX,Y座標[mm]で表す
  - X座標はアプリケーション開始時のロボットの横方向の移動距離[mm]
    - 右を正、左を負とする
  - Y座標はアプリケーション開始時のロボットの前後方向の移動距離[mm]
    - 正面を正、後ろを負とする
- double Create2.Angle
  - ロボットが開始時から曲がった角度[度]を表す
  - 初期値は90度であり、左方向が正、右方向を負とする

- その他、コマンド・センサ情報について
  - 今回の演習には、恐らく必要ない
  - 気になる方は、「iRobot Create 2 Open Interface」で調べてください
  - 「Create2.OI.コマンドorセンサ名」と書くと利用できます

カラーセンサ

RGB (変数)

- string ColorSensorIP
  - カラーセンサのIPアドレスを設定するための変数
  - 初期化関数内で、IPアドレスを代入すると、IPアドレスで指定したカラーセンサと通信を開始する
  - この変数の値を変更しなかった場合は、カラーセンサとの通信は行わない
- string ColorSensor.Color
  - カラーセンサから読み取った色を以下の文字列で表す
  - black red green blue aqua magenta yellow white
- byte ColorSensor.Red
- byte ColorSensor.Green
- byte ColorSensor.Blue
  - カラーセンサから読み取ったRGBの値を取得
  - それぞれ符号無しの1バイトのデータ(0～255の値)

TCP通信

TCP (関数・変数)

- string TCPCommunicationIP
  - TCP通信のIPアドレスを設定するための変数
  - 初期化関数内で、IPアドレスを代入すると、IPアドレスで指定したPythonプログラムと通信を開始する
  - この変数の値を変更しなかった場合は、Pythonとの通信は行わない
- void SendTCPMessage(string text)
  - 文字列をサーバに送る
- void TCPCommunication.Write(byte[] buf, int offset, int length)
  - バイト列をサーバに送る
  - bufが送りたいバイト列を入れた配列、offsetが配列の開始位置、lengthが送りたいデータの長さ
- void TCPCommunication.Read(byte[] buf, int offset, int length)
  - バッファにサーバから送信されたデータを読み込む

システム関連

文字列の入出力(関数)

- void AppConsole.Write(string text)
- void AppConsole.WriteLine(string text)
  - 引数として与えた文字列を画面に出力する
  - 後者の関数では、末尾に改行が付く
  - 改行を明示的に入れたい場合は、「¥r¥n」と書くこと
- string AppConsole.ReadLine()
  - ユーザが画面から入力した文字列を戻り値として受け取る
  - ユーザが何かを入力するまでは、プログラムは次のステップに移らない
  - 数値なども全て文字列として返ってくるので、数値を扱いたい場合は別途変換する必要がある
  - 数値変換にはC#の標準ライブラリを用いると良い (以下変換例)

```
string text = AppConsole.ReadLine();    // ユーザの入力した文字列を受け取る
int value = 0;                          // ユーザが入力した数値を格納するための変数
int.TryParse(text, out value);          // textを数値変換した結果をvalueに格納(失敗時0)
```

時間(関数・変数)

- void KobuTimer.Sleep(int sleepingTime)
  - 指定した時間[ミリ秒]プログラムを停止させる
  - -1を入れた場合、プログラムを終了させるまで、永遠に止まる
- int KobuTimer.ExecutionTime
  - プログラムを実行してから現在まで経過した時間[ミリ秒]を示す変数

プログラムの終了

- void StopApp()
  - プログラムを終了させる
  - ユーザインタフェースで停止ボタンを押したときと同じ働きをする

C#について

C言語との違い

- Cの標準ライブラリは使えない (printfなど)
- staticなローカル変数が使えない (代わりにメンバ変数を使う)
- #include, #defineが使えない
- 関数を作る際、プロトタイプ宣言が不要
- 型名が異なる (後述)
- enumの使い方が異なる (後述)
- 配列の作り方が異なる (後述)
- 無限ループの書き方が異なる (後述)

変数の型名一覧

型名	C言語の型	概要
sbyte	char	符号付き1バイト
byte	unsigned char	符号なし1バイト
short	short	符号付き2バイト
ushort	unsigned short	符号なし2バイト
int	int	符号付き4バイト
uint	unsigned int	符号なし4バイト
long		符号付き8バイト
ulong		符号なし8バイト
bool		1bitの真偽値
string	char[ ]	文字列
char		Unicode 2 バイト文字

enumの違い

- C言語の場合

```
enum State{
    Stop,
    Run,
};
…
enum State state = Stop;
switch(state){
    case Stop:
        …
    case Run:
        …
}
```

- C#の場合

```
enum State{
    Stop,
    Run,
}
…
State state = State.Stop;
switch(state){
    case State.Stop:
        …
    case State.Run:
        …
}
```

enumの要素を用いる際は型名が必要となる

← セミコロン不要

↓ enum不要

↑ 型名が必要

↑ 型名が必要

↑ 型名が必要

配列の作り方

- 要素数5のint型配列arrayを作る際の違い

- C言語の場合
  - int array[5]; // 初期化無し
  - int array[5] = {1, 2, 3, 4, 5}; // 初期化あり

- C#の場合
  - int[] array = new int[5]; // 初期化無し
  - int[] array = new int[5] {1, 2, 3, 4, 5}; // 初期化あり

無限ループの書き方

- C言語の場合

```
while(1) {
    // 処理
}
```

- C#の場合

```
While(true) {
    // 処理
}
```