

Choreonoid講習会 ～ ロボットモデルの制御 ～

中村啓太（会津大学）
穴澤剛士（株式会社FSK）

講習会内容

- ❖ Choreonoidとは?
- ❖ Choreonoidの基本操作
- ❖ プロジェクトの作成
- ❖ ボディモデルとは?
- ❖ コントローラとは?
- ❖ TurtleBot2の自律走行
- ❖ カメラ画像を用いたTurtleBot2の自律走行

コントローラとは？

コントローラ

- ❖ ロボットの制御を行うためのプログラム
- ❖ Choreonoidで使用可能なコントローラ
 - ❖ **シンプルコントローラ**
 - ❖ Choreonoid独自のコントローラ実装形式
 - ❖ **BodyIoRTC**
 - ❖ ロボット用ミドルウェア『OpenRTM』との連携ができるコントローラ

コントローラの実装

- ❖ コントローラでは基本的に以下のことができる
 1. ロボットの状態を入力
 2. 制御計算
 3. ロボットへ指令を出力

コントローラの作成場所

- ❖ シンプルコントローラは、基本的に以下の場所に作成
~/choreonoid/sample/SimpleController/
- ❖ ただし、今回は分かりやすいように以下の場所に作成
~/choreonoid/ext

コントローラの作成場所

❖ ターミナルで以下のコマンドを実行

❖ ディレクトリの作成

```
$ cd ~/choreonoid/ext
```

```
$ mkdir TurtleBot2
```

```
$ cd TurtleBot2
```

❖ コントローラの作成

```
$ gedit TB2JoystickSampleController.cpp &
```

コントローラの実装例

- ❖ シンプルコントローラ
 - ❖ SimpleControllerクラスを継承し, 実装
 - ❖ `cnoide/SimpleController`をインクルード
- ❖ ゲームパッドを使用
 - ❖ `cnoide/Joystick`をインクルード
- ❖ 『wheelNames』 配列にホイール名を格納
 - ❖ 『Kobuki.body』内の『name』で指定したホイール名を指定

コントローラの実装例

```
#include <cnoid/SimpleController>
#include <cnoid/Joystick>
#include <fmt/format.h>

using namespace std;
using namespace cnoid;
using fmt::format;

class TB2JoystickSampleController : public SimpleController
{
    static const int WHEEL_NUM = 2;
    const string wheelNames[WHEEL_NUM] = { "wheel_left", "wheel_right" };
    Link::ActuationMode actuationMode;
    Link* wheels[2];
    Joystick joystick;
    Body* body;
```

コントローラの実装例

- ❖ initialize関数でコントローラの初期化
 - ❖ 引数のioを通して, コントローラとロボット間の入出力に必要な情報を取得
 - ❖ 『body = io->body()』
 - ❖ bodyモデルの情報を取得
 - ❖ 『actuationMode = Link::JOINT_TORQUE』
 - ❖ 関節トルクを指令値に設定
 - ❖ 『option = io->optionString()』
 - ❖ コントローラオプションを取得

コントローラの実装例

```
class TB2JoystickSampleController : public SimpleController
{
// 変数の宣言は省略
public:
    virtual bool initialize(SimpleControllerIO* io) override
    {
        ostream& os = io->os();

        // Body情報取得
        body = io->body();

        // リンクのアクチュエーションモード設定
        actuationMode = Link::JOINT_TORQUE;

        // コントローラオプションの取得
        string option = io->optionString();
    }
};
```

コントローラの実装例

- ❖ コントローラオプションの設定値で、アクチュエーションモードを変更
 - ❖ 『velocity』もしくは『position』 : Link::JOINT_VELOCITY
 - ❖ 『torque』 : Link::JOINT_TORQUE
 - ❖ それ以外: Warning を出力
 - ❖ 『¥"』と表示されている場合,
"¥"をバックスラッシュ("\")に変更

コントローラの実装例

```
if(!option.empty()){
    // コントローラオプションが空の場合
    if(option == "velocity" || option == "position"){
        // velocityかpositionが指定されている場合
        actuationMode = Link::JOINT_VELOCITY;
    } else if(option == "torque"){
        // torqueが指定されている場合
        actuationMode = Link::JOINT_TORQUE;
    } else {
        // 上記以外の場合
        os << format("Warning: Unknown option \\"{}\\".", option) << endl;
    }
}
```

アクチュエーションモード

❖ 関節駆動時にどの状態変数を指令値として使うか決める

シンボル	内容	状態変数
NO_ACTUATION	駆動なし（関節はフリーの状態）	—
JOINT_EFFORT	関節に与える力やトルクが指令値	Link::u()
JOINT_FORCE	JOINT_EFFORTと同様（直動関節用）	Link::u()
JOINT_TORQUE	JOINT_EFFORTと同様（回転関節用）	Link::u()
JOINT_DISPLACEMENT	関節変位（関節角度, 関節並進位置）が指令値	Link::q_target()
JOINT_ANGLE	JOINT_DISPLACEMENTと同様（回転関節用）	Link::q_target()
JOINT_VELOCITY	関節の角速度やオフセット速度が指令値	Link::dq_target()
JOINT_SURFACE_VELOCITY	リンク表面と環境との接触における相対速度が指令値（簡易クローラやベルトコンベアで使用）	Link::dq_target()

コントローラの実装例

- ❖ 『wheels[i] = body->link(リンク)』
 - ❖ 指令値を与えるリンクの取得
- ❖ 『wheels[i]->setActuationMode(actuationMode)』
 - ❖ リンクのアクチュエーションモードを設定
- ❖ 『io->enableOutput(リンク)』
 - ❖ リンクに対するコントローラからの出力を有効化
 - ❖ 有効化していないとリンクが動かない
 - ❖ 入力, 入出力の有効化の場合,
 - ❖ io->enableInput(リンク)
 - ❖ io->enableIO(リンク)

コントローラの実装例

```
for(int i = 0; i < WHEEL_NUM; ++i){
    // ホイールリンクを取得
    wheels[i] = body->link(wheelNames[i]);
    if(!wheels[i]){
        // リンクが取得できない場合
        os << format("{0} of {1} is not found.", wheelNames[i], body->name()) << endl;
        return false;
    }

    // ホイールのアクチュエーションモードを設定
    wheels[i]->setActuationMode(actuationMode);
    // ホイールに対する出力を有効化
    io->enableOutput(wheels[i]);
}

return true;
}
```


コントローラの実装例

- ❖ 『`joystick.readCurrentState()`』
 - ❖ ジョイスティックの状態を取得
- ❖ 左ジョイスティックの変化量を設定
 - ❖ `pos[0]` : 横方向
 - ❖ `pos[1]` : 縦方向
- ❖ 『`if(fabs(pos[i]) < 0.2)`』
 - ❖ ジョイスティックを操作していない場合でも、若干の傾きがあり動くことがあるため、動かないように制御

コントローラの実装例

```
virtual bool control() override
{
    // ジョイスティックの状態取得
    joystick.readCurrentState();
    double pos[2]; // ジョイスティックの変化量

    for(int i = 0; i < 2; ++i){
        // ジョイスティックの値取得
        pos[i] = joystick.getPosition(
            i == 0 ? Joystick::L_STICK_H_AXIS : Joystick::L_STICK_V_AXIS);

        if(fabs(pos[i]) < 0.2){
            // 変化量が0.2未満の場合
            pos[i] = 0.0;
        }
    }
}
```

コントローラの実装例

- ❖ 『if(actuationMode == Link::JOINT_VELOCITY)』
 - ❖ アクチュエーションモードが,
『Link::JOINT_VELOCITY』かどうかを判定
 - ❖ 『リンク->dq_target()』に指令値を与える
- ❖ 『static const double K = 20.0』
 - ❖ ジョイスティックの感度
- ❖ 『CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(クラス名)』
 - ❖ シンプルコントローラとして利用可能

コントローラの実装例

```
if(actuationMode == Link::JOINT_VELOCITY){
    // アクチュエーションモードがvelocityの場合
    static const double K = 20.0;
    wheels[0]->dq_target() = K * (-pos[1] + pos[0]);
    wheels[1]->dq_target() = K * (-pos[1] - pos[0]);
}

return true;
}
};

CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(TB2JoystickSampleController)
```

シンプルコントローラのビルド

- ❖ 作成したソースファイルと同じディレクトリの『CMakeLists.txt』に以下を追加
 - ❖ 1行で, 『add_cnooid_simple_controller (コントローラ名 ファイル名)』

```
add_cnooid_simple_controller  
(TB2JoystickSampleController  
TB2JoystickSampleController.cpp)
```
 - ❖ 『CMakeLists.txt』がない場合は, 新規作成を行う

```
$ gedit CMakeLists.txt &
```

シンプルコントローラのビルド

- ❖ 修正ができたなら, Choreonoidのビルドを行う

```
$ cd ~/choreonoid/build
```

```
$ cmake ..
```

```
$ make
```

- ❖ 『~/choreonoid/build/lib/choreonoid-1.8/simplecontroller』
ディレクトリに作成した 『TB2JoystickSampleController.so』
ファイルがあるか確認

```
$ cd ~/choreonoid/build/lib/choreonoid-1.8/  
simplecontroller
```

```
$ ls TB2JoystickSampleController.so
```

TurtleBot2のシミュレーション

- ❖ PS4コントローラをUSBでPCに接続
- ❖ TurtleBot2プロジェクトを開く
 - `$ cd ~/choreonoid/`
 - `$ choreonoid ext/Education/ChoreonoidWorkshop/TurtleBot2/project/TurtleBot2.cnoid`
- ❖ 作成したコントローラを読み込む
 - ❖ シンプルコントローラアイテムを追加
 - ❖ 『コントローラモジュール』から『TB2JoystickSampleController.so』を選択

TurtleBot2の自律走行

TurtleBot2の自律走行 | 直進後停止

- ❖ 『SimpleControllerIO* io』
 - ❖ シンプルコントローラ入出力情報取得変数
- ❖ 『const double d = 0.115』
 - ❖ トレッド幅の半分の値を格納する定数
- ❖ 『const double Kp = 48.0』
 - ❖ 比例定数を格納する定数

TurtleBot2の自律走行 | 直進後停止

```
#include <cnoid/SimpleController>
#include <fmt/format.h>

using namespace std;
using namespace cnoid;
using fmt::format;

class TB2StraightSampleController : public SimpleController
{
    static const int WHEEL_NUM = 2;
    const string wheelNames[WHEEL_NUM] = { "wheel_left", "wheel_right" };
    Link::ActuationMode actuationMode;
    Link* wheels[2];
    Body* body;
    SimpleControllerIO* io;
    double startTime = 0.0;
    const double d = 0.115;
    const double Kp = 48.0;
```

TurtleBot2の自律走行 | 直進後停止

- ❖ 『ostream& os = io->os()』 から
『io->enableOutput(wheels[i])』 まで,
『TB2JoystickSampleController』 と同様
- ❖ 『startTime = 0.0』
- ❖ シミュレーション開始時間の初期化

```
public:
    virtual bool initialize(SimpleControllerIO* io) override
    {
        // ioオブジェクトの取得
        this->io = io;

        // TB2JoystickSampleControllerと同様

        // 開始時間の初期化
        startTime = 0.0;

        return true;
    }
```

TurtleBot2の自律走行 | 直進後停止

- ❖ 車体の中心の速度を v_x , 旋回角速度を v_a とする
- ❖ 『`startTime = io->currentTime()`』
- ❖ 現在のシミュレーション時間を設定

```
virtual bool control() override
{
    // 車体の中心の速度 $v_x$ (m/s), 旋回角速度 $v_a$ (rad/s)
    double vx, va;
    va = 0.0;
    vx = 0.3;

    if(actuationMode == Link::JOINT_VELOCITY){
        // アクチュエーションモードがvelocityの場合
        // 関節速度の指令値格納変数
        double dq_target[2];

        if(startTime == 0.0){
            // 開始時間が0.0の場合
            // 開始時間に現在のシミュレーション時間を設定
            startTime = io->currentTime();
        }
    }
}
```

TurtleBot2の自律走行 | 直進後停止

- ❖ 『`if(io->currentTime() - startTime > 2.0)`』
- ❖ シミュレーション時間で2秒経過したかチェック
 - ❖ 2秒経過していれば, 停止
 - ❖ 2秒経過していなければ, 直進

TurtleBot2の自律走行 | 直進後停止

```
    if(io->currentTime() - startTime > 2.0){
        // 現在時間 - 開始時間が2.0より大きい場合
        // 左右のホイールの指令値を0.0に設定
        wheels[0]->dq_target() = 0.0;
        wheels[1]->dq_target() = 0.0;
    }else{
        dq_target[0] = Kp * (vx - va * d);
        dq_target[1] = Kp * (vx + va * d);
        // 左右のホイールに指令値を与える
        wheels[0]->dq_target() = dq_target[0];
        wheels[1]->dq_target() = dq_target[1];
    }
}

return true;
}
};
```

```
CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(TB2StraightSampleController)
```

対向2輪ロボットの操作量

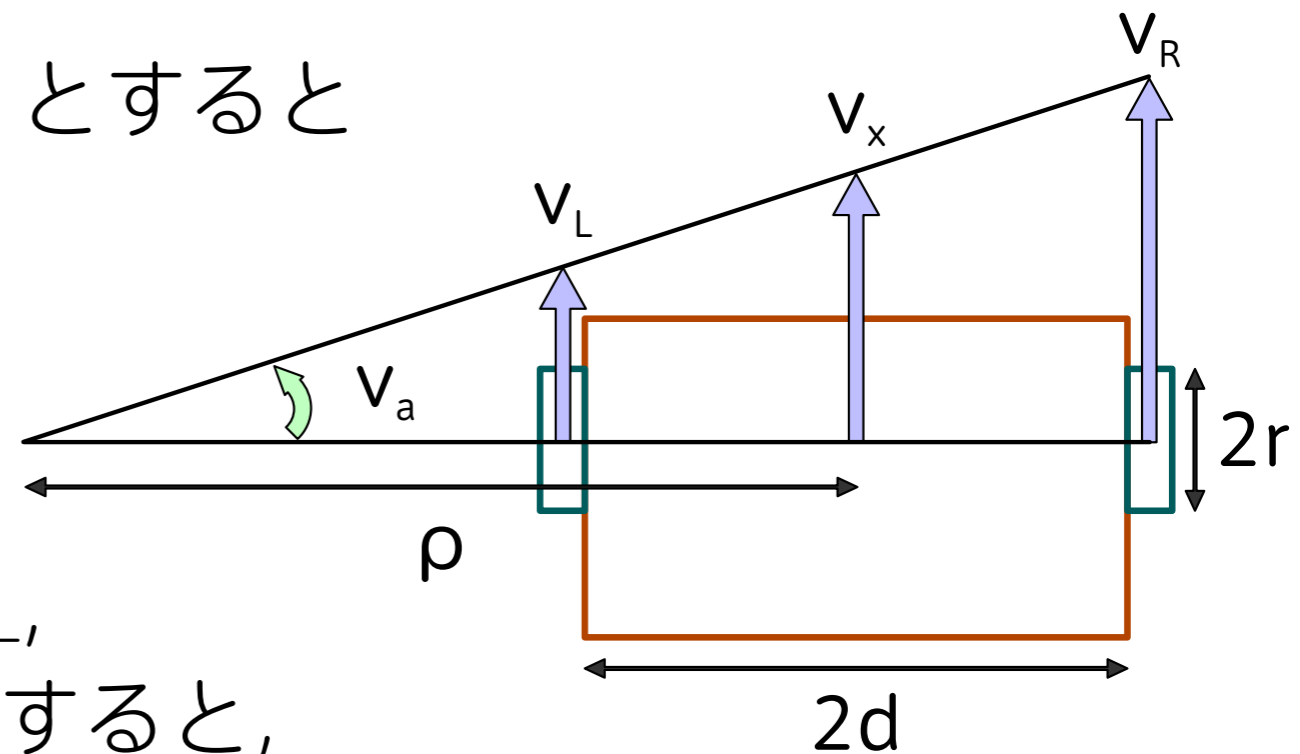
- ❖ ロボットの旋回角速度を v_a , 中心の速度を v_x , 旋回半径を ρ とすると 旋回中心回りの運動は以下の式で表すことができる

- ① $v_x = \rho v_a$

- ❖ 車輪の接地点での速度を v_R, v_L , 中心から車輪までの距離を d とすると, それぞれの車輪の旋回半径は d だけ増減する

- ② $v_R = (\rho + d)v_a$

- ③ $v_L = (\rho - d)v_a$



対向2輪ロボットの操作量

① $v_x = \rho v_a$

② $v_R = (\rho + d)v_a$

③ $v_L = (\rho - d)v_a$

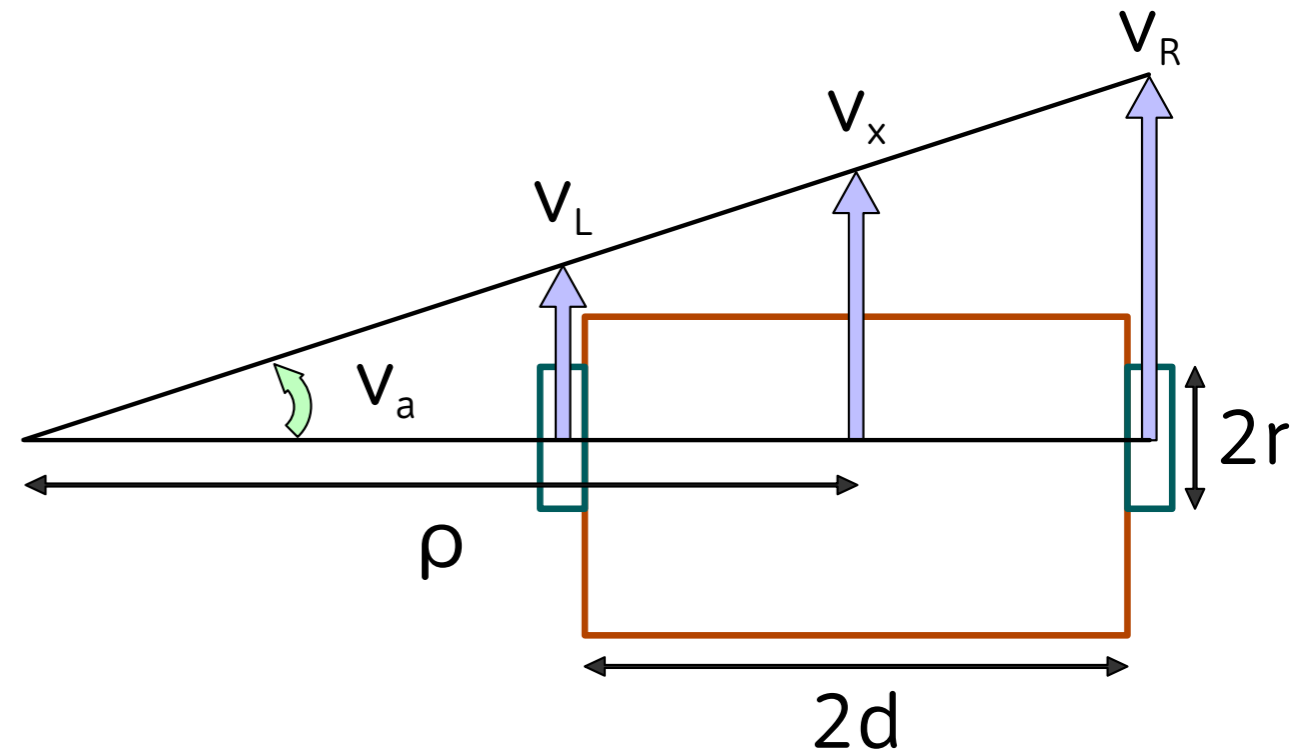
❖ ①を ρ について解くと

④ $\rho = v_x / v_a$

❖ ②, ③に, ④を代入すると, 以下の式が得られる

1. $v_R = (v_x + d * v_a)$

2. $v_L = (v_x - d * v_a)$



TurtleBot2の自律走行 | 直進後停止

- ❖ TB2StraightSampleController.cppをビルドしたら、
ターミナルで以下のコマンドを実行

```
$ cd ~/choreonoid/  
  
$ choreonoid ext/Education/ChoreonoidWorkshop/TurtleBot2/  
project/TurtleBot2_Straight.cnoid
```

TurtleBot2の自律走行 I 課題

- ❖ 2秒直進後停止を行ったプログラムを参考に、以下の動作をするコントローラを作成してください
- ❖ ファイル名: `TB2TurningSampleController.cpp`
- ❖ ①～⑥の順に動作
 - ① 2秒間直進
 - ② 90° 右旋回
 - ③ 2秒間直進
 - ④ 90° 左旋回
 - ⑤ 2秒間直進
 - ⑥ 停止

TurtleBot2の自律走行 | 課題

- ❖ TB2TurningSampleController.cppをビルドしたら、
ターミナルで以下のコマンドを実行

```
$ cd ~/choreonoid/  
  
$ choreonoid ext/Education/ChoreonoidWorkshop/TurtleBot2/  
project/TurtleBot2_Turning.cnoid
```

TurtleBot2の自律走行 | 課題解答例

- ❖ 『control()』関数の『double vx, va』の宣言までは、TB2StraightSampleControllerと同様

```
#include <cnoid/SimpleController>
#include <fmt/format.h>

using namespace std;
using namespace cnoid;
using fmt::format;

class TB2TurningSampleController : public SimpleController
{
    // TB2StraightSampleControllerと同様
public:
    virtual bool initialize(SimpleControllerIO* io) override
    {
        // TB2StraightSampleControllerと同様
        return true;
    }

    virtual bool control() override
    {
        // 車体の中心の速度vx(m/s), 旋回角速度va(rad/s)
        double vx, va;
        va = 0.0;
        vx = 0.3;
    }
};
```

TurtleBot2の自律走行 | 課題解答例

❖ 『control()』関数の
『double vx, va』の
宣言の後に追加

```
if(io->currentTime() - startTime < 2.0){  
    // シミュレーション時間が2s未満の場合、直進  
    va = 0.0;  
    vx = 0.3;  
}else if(io->currentTime() - startTime < 2.2){  
    va = vx = 0.0;  
}else if(io->currentTime() - startTime < 2.7){  
    // 90°右旋回  
    va = -2.22;  
    vx = 0.0;  
}else if(io->currentTime() - startTime < 2.9){  
    va = vx = 0.0;  
}else if(io->currentTime() - startTime < 4.9){  
    // 2s間直進  
    va = 0.0;  
    vx = 0.3;  
}else if(io->currentTime() - startTime < 5.1){  
    va = vx = 0.0;  
}else if(io->currentTime() - startTime < 5.6){  
    // 90°左旋回  
    va = 2.22;  
    vx = 0.0;  
}else if(io->currentTime() - startTime < 5.8){  
    va = vx = 0.0;  
}else if(io->currentTime() - startTime < 7.8){  
    // 2s間直進  
    va = 0.0;  
    vx = 0.3;  
}else{  
    // 停止  
    va = vx = 0.0;  
}
```

TurtleBot2の自律走行 | 課題解答例

- ❖ 『TB2StraightSampleController』の
if(actuatinMode == Link::JOINT_VELOCITY)ブロック内の
停止制御を削除

```
if(actuationMode == Link::JOINT_VELOCITY){
    // アクチュエーションモードがvelocityの場合
    // 関節速度の指令値格納変数
    double dq_target[2];
    if(startTime == 0.0){
        // 開始時間が0.0の場合
        // 開始時間に現在のシミュレーション時間を設定
        startTime = io->currentTime();
    }

    dq_target[0] = Kp * (vx - va * d);
    dq_target[1] = Kp * (vx + va * d);
    // 左右のホイールに指令値を与える
    wheels[0]->dq_target() = dq_target[0];
    wheels[1]->dq_target() = dq_target[1];
}
return true;
};

CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(TB2TurningSampleController)
```

カメラ画像を用いた TurtleBot2の自律走行

カメラ画像による色の判定

- ❖ Choreonoidにカラーセンサの機能はない
 - ❖ カメラ画像の取得は可能
 - ❖ カメラ画像からRGB値を取得し、色の判定を行う
 - ❖ 画像内で取得できる色の目標値を決め、目標値に達したら停止するなどの処理を行う

TurtleBot2の自律走行 | ライン上で停止

- ❖ cnoide/Cameraをインクルード
 - ❖ カメラ情報を取得
- ❖ 『CameraPtr camera』
 - ❖ カメラ情報格納変数
- ❖ 『std::shared_ptr<const Image> prevImage』
 - ❖ 過去の画像格納変数

```
#include <cnoide/SimpleController>
#include <cnoide/Camera>
#include <fmt/format.h>

using namespace std;
using namespace cnoide;
using fmt::format;

class TB2StopLineSampleController : public SimpleController{
    static const int WHEEL_NUM = 2;
    const string wheelNames[WHEEL_NUM] = {"wheel_left", "wheel_right"};
    Link::ActuationMode actuationMode;
    Link* wheels[2];
    Body* body;
    // カメラデバイス情報格納変数
    CameraPtr camera;
    // 前回の画像格納変数
    std::shared_ptr<const Image> prevImage;
    // 画像内のグレー、白、黄色の数の格納変数
    int cnt[3] = { 0, 0, 0 };
    const double d = 0.115;
    // PID制御の係数
    const double Kp = 48.0;
    // 画像内の白線の目標値
    static const int TARGET = 15000;
    ScopedConnection cameraConnection;
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ 『int cnt[3]』
 - ❖ 画像内の灰色, 白色, 黄色の数を格納する配列
- ❖ 『static const int TARGET = 15000』
 - ❖ 停止する白線上での白色の目標値
- ❖ 『ScopedConnection cameraConnection』
 - ❖ シグナル設定状態を取得する変数

```
#include <cnoid/SimpleController>
#include <cnoid/Camera>
#include <fmt/format.h>

using namespace std;
using namespace cnoid;
using fmt::format;

class TB2StopLineSampleController : public SimpleController{
    static const int WHEEL_NUM = 2;
    const string wheelNames[WHEEL_NUM] = {"wheel_left", "wheel_right"};
    Link::ActuationMode actuationMode;
    Link* wheels[2];
    Body* body;
    // カメラデバイス情報格納変数
    CameraPtr camera;
    // 前回の画像格納変数
    std::shared_ptr<const Image> prevImage;
    // 画像内のグレー、白、黄色の数の格納変数
    int cnt[3] = { 0, 0, 0 };
    const double d = 0.115;
    // PID制御の係数
    const double Kp = 48.0;
    // 画像内の白線の目標値
    static const int TARGET = 15000;
    ScopedConnection cameraConnection;
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ 『camera = body->findDevice <Camera>("カメラ名)』
- ❖ LineTraceカメラを取得
- ❖ TurtleBot2.body内のカメラ名を指定
- ❖ 『io->enableInput(camera)』
- ❖ カメラからのコントローラへの入力を有効化

```
public:
    virtual bool initialize(SimpleControllerIO* io) override
    {
        // TB2JoystickSampleControllerと同様

        // LineTraceカメラを取得
        camera = body->findDevice<Camera>("LineTrace");
        // カメラのコントローラへの入力を有効化
        io->enableInput(camera);
        cameraConnection.disconnect();
        cameraConnection = camera->sigStateChanged().connect([&]() { onCameraStateChanged(); });

        return true;
    }
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ 『cameraConnection = camera->sigStateChanged().connect([&]() { onCameraStateChanged(); });』
- ❖ センサーの状態が変わった場合, 色の割合を取得する onCameraStateChanged() 関数を呼び出す

```
public:
    virtual bool initialize(SimpleControllerIO* io) override
    {
        // TB2JoystickSampleControllerと同様

        // LineTraceカメラを取得
        camera = body->findDevice<Camera>("LineTrace");
        // カメラのコントローラへの入力を有効化
        io->enableInput(camera);
        cameraConnection.disconnect();
        cameraConnection = camera->sigStateChanged().connect([&]() { onCameraStateChanged(); });

        return true;
    }
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ 『if(cnt[1] > TARGET)』
- ❖ 画像内の白色の数が目標値より大きいかどうか判定
- ❖ 大きければ停止

```
virtual bool control() override
{
    double vx, va;
    va = 0.0;
    vx = 0.3;

    if(actuationMode == Link::JOINT_VELOCITY){
        double dq_target[2];

        dq_target[0] = Kp * (vx - va * d);
        dq_target[1] = Kp * (vx + va * d);

        wheels[0]->dq_target() = dq_target[0];
        wheels[1]->dq_target() = dq_target[1];

        if(cnt[1] > TARGET){
            // 白線が近くなったら停止
            wheels[0]->dq_target() = 0.0;
            wheels[1]->dq_target() = 0.0;
        }
    }
    return true;
}
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ 『`if(camera->sharedImage() != prevImage)`』
 - ❖ カメラ画像が更新されたか確認
- ❖ 『`const Image& image = camera->constImage()`』
 - ❖ カメラ画像を取得
- ❖ 『`unsigned char* src = (unsigned char*)image.pixels()`』
 - ❖ 画像の1ピクセルごとのデータ取得

TurtleBot2の自律走行 | ライン上で停止

```
void onCameraStateChanged()
{
    size_t length = 0;
    if(camera->sharedImage() != prevImage){
        // カメラ画像が更新されたか確認
        const Image& image = camera->constImage();
        if(!image.empty()){
            // カメラ画像が取得できた場合
            int width, height;
            // 画像のサイズを取得
            height = image.height();
            width = image.width();
            length = width * height * image.numComponents() * sizeof(unsigned char);
        }

        // 画像の1ピクセルごとのデータを取得
        unsigned char* src = (unsigned char*)image.pixels();

        // グレー、白、黄色のカウント用配列の初期化
        cnt[0] = cnt[1] = cnt[2] = 0;
        // RGB値格納配列
        int rgb[3];
    }
}
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ rgb配列にRGBの値を格納
 - ❖ 画像内の灰色, 白色, 黄色の数を取得
 - ❖ cnt[0] : 灰色の数
 - ❖ cnt[1] : 白色の数
 - ❖ cnt[2] : 黄色の数

TurtleBot2の自律走行 | ライン上で停止

```
// データ数分ループ
for(int i = 0; i < length / 3; ++i){
    // RGBの値を格納
    rgb[0] = (int)src[i * 3];
    rgb[1] = (int)src[i * 3 + 1];
    rgb[2] = (int)src[i * 3 + 2];

    if((rgb[0] >= 100 && rgb[0] < 180)
        && (rgb[1] >= 100 && rgb[1] < 180)
        && (rgb[2] >= 100 && rgb[2] < 180)
        && abs(rgb[0] - rgb[1]) <= 10
        && abs(rgb[1] - rgb[2]) <= 10
        && abs(rgb[2] - rgb[0]) <= 10){
        cnt[0]++; // グレーの個数をカウント
    }else if(rgb[0] >= 180 && rgb[1] >= 180 && rgb[2] >= 180){
        cnt[1]++; // 白の個数をカウント
    }else if(rgb[0] >= 170 && rgb[1] >= 170 && rgb[2] <= 100){
        cnt[2]++; // 黄色の個数をカウント
    }
}
// 前回値の更新
prevImage = camera->sharedImage();
}
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ シミュレーション停止時に、画像から取得した灰色、白色、黄色の数を初期化

```
virtual void stop() override
{
    for(int i = 0; i < 3; ++i){
        cnt[i] = 0;
    }
    cameraConnection.disconnect();
}
};
```

```
CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(TB2StopLineSampleController)
```

TurtleBot2の自律走行 | ライン上で停止

- ❖ TB2StopLineSampleController.cppをビルドしたら、ターミナルで以下のコマンドを実行

```
$ cd ~/choreonoid/  
  
$ choreonoid ext/Education/ChoreonoidWorkshop/TurtleBot2/  
project/TurtleBot2_StopLine.cnoid
```

ライントレース | 白線上を走行

- ❖ PID制御の定数を宣言
 - ❖ Kp: 比例定数
 - ❖ Ki: 積分定数
 - ❖ Kd: 微分定数
- ❖ PID制御で使用する変数を宣言
 - ❖ diff_R, diff_L: 偏差値
 - ❖ integral[2]: 積分値
 - ❖ derivation[2]: 微分値

```
#include <cnoid/SimpleController>
#include <cnoid/Camera>
#include <fmt/format.h>

using namespace std;
using namespace cnoid;
using fmt::format;

class TB2TurnLineSampleController : public SimpleController
{
    // TB2StopLineSampleControllerと同様
    // タイムステップ格納変数
    double dt;
    // PID制御の定数
    const double Kp = 48.0;
    const double Ki = 0.0092;
    const double Kd = 0.00015;
    // 画像内の白線の目標値
    static const int TARGET = 10000;
    // 前回と現在の偏差値格納変数
    double diff_R[2] = { 0, 0 };
    double diff_L[2] = { 0, 0 };
    // 偏差の積分値格納変数
    double integral[2] = { 0, 0 };
    // 偏差の微分値格納変数
    double derivation[2] = { 0, 0 };
    ScopedConnection cameraConnection;
```

PID制御

- ❖ 入力値の制御を出力値と目標値との偏差, 偏差の積分, 偏差の微分の3つの要素で行う制御
 - ❖ **P(Proportional): 比例制御**
 - ❖ 現在発生している誤差を修正
 - ❖ **I(Integral): 積分制御**
 - ❖ 現在までに蓄積された誤差を修正
 - ❖ **D(Differential): 微分制御**
 - ❖ これから発生する誤差を修正

ライントレース | 白線上を走行

- ❖ `dt = io->timeStep()`
- ❖ シミュレータのタイムステップを取得

```
public:
    virtual bool initialize(SimpleControllerIO* io) override
    {
        // TB2JoystickSampleControllerと同様
        // LineTraceカメラを取得
        camera = body->findDevice<Camera>("LineTrace");
        // カメラのコントローラへの入力を有効化
        io->enableInput(camera);
        cameraConnection.disconnect();
        cameraConnection = camera->sigStateChanged().connect([&]() { onCameraStateChanged(); });
        // タイムステップの設定
        dt = io->timeStep();
        return true;
    }
```

ライントレース | 白線上を走行

- ❖ $\text{diff_L}[0] = \text{diff_L}[1]$
 - ❖ 前回の偏差値を設定
- ❖ $\text{diff_L}[1] = (\text{TARGET} - \text{cnt}[1]) / 2500$
 - ❖ 偏差値を取得 (目標値 - 白色の割合)
 - ❖ 偏差が大きいため, 2500で割っている
- ❖ $\text{integral}[0] += (\text{diff_L}[1] + \text{diff_L}[0]) / 2.0 * dt$
 - ❖ 偏差の積分値を取得
 - ❖ $(\{(\text{最新の偏差} - \text{前回の偏差}) / 2\} \times \text{時間})$
- ❖ $\text{derivation}[0] = (\text{diff_L}[1] - \text{diff_L}[0]) / dt$
 - ❖ 偏差の微分値を取得: $(\{(\text{最新の偏差} - \text{現在の偏差}) / \text{時間}\})$

ライントレース | 白線上を走行

```
virtual bool control() override
{
    double vx, va;
    va = 0.5;
    vx = 0.3;

    // 前回の偏差値を設定
    diff_L[0] = diff_L[1];
    // 現在の偏差値(目標値 - センサ値)を取得
    diff_L[1] = (TARGET - cnt[1]) / 2500;
    // 偏差の積分値を取得。偏差の積分値 = ((最新の偏差 + 前回の偏差) / 2) * 時間
    integral[0] += (diff_L[1] + diff_L[0]) / 2.0 * dt;
    // 偏差の微分値を取得。偏差の微分値 = (最新の偏差 - 前回の偏差) / 時間
    derivation[0] = (diff_L[1] - diff_L[0]) / dt;
    diff_R[0] = diff_R[1];
    // 現在の偏差値(目標値 - センサ値)を取得
    diff_R[1] = (TARGET - cnt[1]) / 2500;
    integral[1] += (diff_R[1] + diff_R[0]) / 2.0 * dt;
    derivation[1] = (diff_R[1] - diff_R[0]) / dt;
```


ライントレース | 白線上を走行

- ❖ PID制御により指令値を算出
 - ❖ 比例定数 \times (速度 - 旋回速度)
 - ❖ 積分定数 \times 偏差の積分値 \times 旋回速度
 - ❖ 微分定数 \times 偏差の微分値 \times 旋回速度

```
if(actuationMode == Link::JOINT_VELOCITY){  
    // 関節速度の指令値格納変数  
    double dq_target[2];  
  
    // PID制御  
    dq_target[0] = Kp * (vx - (va * d * diff_L[1])) + Ki * (integral[0] * va * d) + Kd * (derivation[0] * va * d);  
    dq_target[1] = Kp * (vx + (va * d * diff_R[1])) + Ki * (integral[1] * va * d) + Kd * (derivation[1] * va * d);  
  
    // 左右のホイールに指令値を与える  
    wheels[0]->dq_target() = dq_target[0];  
    wheels[1]->dq_target() = dq_target[1];  
}  
return true;  
}
```

ライントレース | 白線上を走行

- ❖ $\text{diff_R}[i] = \text{diff_L}[i] = 0$
 - ❖ 偏差の初期化
- ❖ $\text{integral}[i] = \text{derivation}[i] = 0$
 - ❖ 偏差の積分値, 偏差の微分値の初期化

```
virtual void stop() override
{
    for(int i = 0; i < WHEEL_NUM; ++i){
        diff_R[i] = diff_L[i] = 0;
        integral[i] = derivation[i] = 0;
    }

    for(int i = 0; i < 3; ++i){
        cnt[i] = 0;
    }
    cameraConnection.disconnect();
}

void onCameraStateChanged()
{
    // TB2StopLineSampleControllerと同様
}
};

CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(TB2TurnLineSampleController)
```

ライントレース | 白線上を走行

- ❖ TB2TurnLineSampleController.cppをビルドしたら、
ターミナルで以下のコマンドを実行

```
$ cd ~/choreonoid/  
  
$ choreonoid ext/Education/ChoreonoidWorkshop/TurtleBot2/  
project/TurtleBot2_TurnLine.cnoid
```

ライントレース | 課題

- ❖ TB2TurnLineSampleController.cppを参考に、白と黄色の線を認識し、その間を走行するコントローラを作成してください
 - ❖ ファイル名: TB2AutoSampleController.cpp
 - ❖ 目標値: 1700
- ❖ TB2AutoSampleController.cppをビルドしたら、ライントレースを行ってみてください

```
$ cd ~/choreonoid/  
  
$ choreonoid ext/Education/ChoreonoidWorkshop/TurtleBot2/  
project/TurtleBot2_Autorace.cnoid
```

ライントレースのグラフ化

- ❖ matplotlib-cppを使用したグラフ作成
 - ❖ matplotlib-cppをダウンロード
 - ❖ 『matplotlib-cppのインストール方法.docx』を参照
 - ❖ グラフ作成プログラム『plot.cpp』を作成
 - ❖ X軸を時間T, Y軸をX, Y座標とし, グラフをプロット
 - ❖ 読み込むファイルは, 0.1秒ごとにX, Y座標を取得
 - ❖ 『plot.cpp』をコンパイル・実行するシェルスクリプト『plot.sh』を作成

グラフ作成プログラム

- ❖ 『vector<double> t, x, y』
 - ❖ ファイルから読み込んだ値を格納するため、動的配列として宣言
- ❖ 『string cmd = "pwd | tr '\n ' '/'"』
 - ❖ カレントディレクトリ取得コマンド
 - ❖ pwdコマンド実行後の標準出力は改行が入るため、trコマンドで改行コードを / に置換
 - ❖ 『¥n』と表示されている場合、
“¥”をバックスラッシュ(\)に変更

グラフ作成プログラム

```
#include <fstream>
#include <matplotlib-cpp/matplotlibcpp.h>

using namespace std;
namespace plt = matplotlibcpp;

int main(){
    cout << "matplotlib start" << endl;

    // ファイル名
    string fileName = "plot.tsv";
    // 配列の定義
    vector<double> t, x, y;
    // タブ区切りのデータ格納変数
    string tmp;
    // ファイル入出力の宣言
    FILE* fp;
    // バッファサイズ
    static const int BUF_SIZE = 100;
    char str[BUF_SIZE];
    // カレントディレクトリの取得コマンド
    string cmd = "pwd | tr '\n' '/'";
    int cnt = 0;
```

グラフ作成プログラム

- ❖ 『fp = popen(コマンド, タイプ)』
 - ❖ タイプには以下を指定
 - ❖ r : 標準出力を読み込む, w : 標準入力に書き込む
 - ❖ プロセスをオープンし,
カレントディレクトリ取得コマンドを実行
- ❖ 『fgets(読込データ格納変数, バイト数, ファイルポインタ)』
 - ❖ コマンド結果を1行ずつ読み込む
- ❖ 『pclose(fp)』
 - ❖ プロセスをクローズ

グラフ作成プログラム

- ❖ 『`ifstream ifs(filePath)`』 : ファイルを読み込む
- ❖ 『`ifs.fail()`』 : ファイル読み込みに失敗したか判定
- ❖ 『`plt::xlabel(ラベル名)`』 : X軸のラベルを設定
- ❖ 『`plt::ylabel(ラベル名)`』 : Y軸のラベル設定

グラフ作成プログラム

```
// ファイルパス
string filePath = "";

if((fp = popen(cmd.c_str(), "r")) != NULL){
    // プロセスをオープンしコマンドを実行
    while(fgets(str, sizeof(str), fp) != NULL){
        // コマンド結果を1行ずつ読み込む
        // カレントディレクトリの取得
        filePath += str;
    }
    // プロセスをクローズ
    pclose(fp);
}

// ディレクトリ名とファイル名を連結
filePath = filePath + fileName;
// ファイルを読み込む
ifstream ifs(filePath);
if(ifs.fail()){
    cerr << "Failed to open file." << endl;
    return false;
}

plt::xlabel("T");
plt::ylabel("X, Y");
```

グラフ作成プログラム

- ❖ 『getline(データ取得ストリーム, データ格納文字列, 区切り文字)』
 - ❖ 入力ストリームから文字を読み込み, 文字列に格納
 - ❖ 『¥t』と表示されている場合, “¥”をバックスラッシュ(\)に変更
- ❖ 『t.push_back(格納値)』 : 動的配列t, x, yに値を格納
 - ❖ t(time) : 1列目, x(X軸) : 2列目, y(Y軸) : 3列目
- ❖ 『plt::plot(X軸, Y軸, 線の色)』
 - ❖ 色を指定してグラフをプロット
- ❖ 『plt::show()』 : プロットしたグラフの表示

グラフ作成プログラム

```
while(getline(ifs, tmp, '\t')){
    if(ifs.eof()){
        ifs.close();
        break;
    } else {
        if(cnt == 0){
            t.push_back(stod(tmp));
            cnt++;
        } else if(cnt == 1){
            x.push_back(stod(tmp));
            cnt++;
        } else if(cnt == 2){
            y.push_back(stod(tmp));
            cnt = 0;
        }
    }
}

plt::plot(t, x, "b");
plt::plot(t, y, "r");
plt::show();

return 0;
}
```

グラフ作成プログラムの実行

❖ 『plot.sh』ファイルを作成

```
#!/bin/sh
# カレントディレクトリの取得
cwd=`dirname "${0}"`
# 相対パスが取得された場合, 絶対パスを取得
expr "${0}" : "/.*" > /dev/null || cwd=`(cd "${cwd}" && pwd)`
# 作成したplot.cppファイルをコンパイル・実行
g++ ${cwd}/plot.cpp -I/usr/include/python3.6m -lstdc++ -lpython3.6m -std=c++11 && ${cwd}/a.out
```

❖ 『plot.sh』の実行

```
$ cd ~/choreonoid/ext/TurtleBot2/
```

```
$ sudo chmod 755 plot.sh
```

```
$ ./plot.sh
```

グラフ作成プログラムの実行

- ❖ 『`cwd=`dirname "${0}"``』
 - ❖ 指定したファイルのディレクトリパスを取得
 - ❖ `${0}` : 実行ファイル名

グラフ作成プログラムの実行

- ❖ 『`expr "${0}" : "/.*" > /dev/null || cwd=`(cd "${cwd}" && pwd)``』
- ❖ 『`expr 文字列 : 正規表現`』で文字列を評価
 - ❖ `/` : 開始文字
 - ❖ `.` : 任意の1文字
 - ❖ `*` : 0回以上の繰り返し
- ❖ 『`> /dev/null`』 : 標準出力結果の書き込みを破棄
- ❖ 『`cd ディレクトリパス && pwd`』
 - ❖ ファイルが存在するディレクトリに移動し、絶対パスを取得

グラフ作成プログラムの実行

- ❖ 『`g++ ${cwd}/plot.cpp -I/usr/include/python3.6m -lstdc++ -lpython3.6m -std=c++11 && ${cwd}/a.out`』
- ❖ `g++` : C++コンパイラコマンド
- ❖ `-I` : ヘッダを追加
- ❖ `-l` : ライブラリをリンク
- ❖ `-std=c++11` : C++11の機能を有効化
- ❖ `${cwd}/a.out` : 作成されたファイルを実行

ライントレース | 課題解答例

- ❖ TB2TurnLineSampleControllerと同様の記述を行う
 - ❖ PID制御の定数のみ変更
- ❖ 以下のインクルードを追加
 - ❖ `fstream`
- ❖ PID制御の定数の変更
- ❖ ファイル出力に必要なとなる変数の定義

ライントレース | 課題解答例

```
#include <fstream>
// TB2TurnLineSampleControllerと同様

class TB2AutoSampleController : public SimpleController
{
    // TB2TurnLineSampleControllerと同様
    Link* rootLink;
    // PID制御の係数
    const double Kp = 48.0;
    const double Ki = 0.002;
    const double Kd = 0.0007;
    // シミュレーション時間取得変数
    SimpleControllerIO* io;
    double startTime, waitTime;
    const double INTERVAL = 0.1;
    // ファイル出力ストリーム
    ofstream ofs;
    FILE* fp;
    // バッファサイズ
    static const int BUF_SIZE = 100;
    char str[BUF_SIZE];
    // カレントディレクトリ取得コマンド
    string cmd = "cd ~/choreonoid/ext/TurtleBot2/; pwd | tr '\n' '/'";
};
```

ライントレース | 課題解答例

- ❖ 『rootLink->setActuationMode(Link::LINK_POSITION)』
 - ❖ アクチュエーションモードを LINK_POSITION に設定
 - ❖ リンク位置を取得するために必要
- ❖ 『fp = popen(cmd.c_str(), "r")』
 - ❖ プロセスをオープンしコマンドを実行
- ❖ 『fgets(str, sizeof(str), fp)』
 - ❖ コマンド結果を1行ずつ読み込む
- ❖ 『pclose(fp)』
 - ❖ プロセスをクローズ

ライントレース | 課題解答例

```
// ファイルパス
string filePath = "";
const string FILENAME = "plot.tsv";

public:
virtual bool initialize(SimpleControllerIO* io) override
{
    this->io = io;
    // TB2TurnLineSampleControllerと同様
    // TurtleBot2のコントローラへの入力を有効化
    rootLink = body->link("kobuki");
    rootLink->setActuationMode(Link::LINK_POSITION);
    io->enableInput(rootLink);

    if((fp = popen(cmd.c_str(), "r")) != NULL){
        // プロセスをオープンしコマンドを実行
        while(fgets(str, sizeof(str), fp) != NULL){
            // コマンド結果を1行ずつ読み込む
            filePath += str;
        }
        // プロセスをクローズ
        pclose(fp);
    }
}
```

ライントレース | 課題解答例

- ❖ 『`ofs.open(ファイルパス, モード)`』
 - ❖ ファイルを開く
 - ❖ モードは以下を指定可能
 - ❖ `ios::out` : 書き込みモード
 - ❖ `ios::in` : 読み取りモード
 - ❖ `ios::app` : 追記モード
 - ❖ `ios::trunc` : 上書きモード
 - ❖ `ios::binary` : バイナリモード

ライントレース | 課題解答例

```
// ディレクトリ名とファイル名を連結  
filePath = filePath + FILENAME;  
  
ofs.open(filePath, ios::out);  
startTime = 0.0;  
  
return true;  
}
```

ライントレース | 課題解答例

- ❖ $\text{diff_L}[1] = (\text{TARGET} - \text{cnt}[2]) / 500$
偏差値を取得 (目標値 - 黄色の割合)
 - ❖ 偏差が大きいため, 500で割っている
- ❖ $\text{diff_R}[1] = -(\text{TARGET} - \text{cnt}[1]) / 500$
 - ❖ 偏差値を取得 (目標値 - 白色の割合)
 - ❖ 符号を反転させないと反対方向に旋回
- ❖ PID制御により指令値を算出
 - ❖ 比例定数 \times (速度 - 旋回速度)
 - ❖ 積分定数 \times 偏差の積分值 \times 旋回速度
 - ❖ 微分定数 \times 偏差の微分值 \times 旋回速度

ライントレース | 課題解答例

```
virtual bool control() override
{
    // TB2TurnLineSampleControllerと同様

    diff_L[0] = diff_L[1];
    diff_L[1] = (TARGET - cnt[2]) / 500;
    integral[0] += (diff_L[1] + diff_L[0]) / 2.0 * dt;
    derivation[0] = (diff_L[1] - diff_L[0]) / dt;

    diff_R[0] = diff_R[1];
    diff_R[1] = -(TARGET - cnt[1]) / 500;
    integral[1] += (diff_R[1] + diff_R[0]) / 2.0 * dt;
    derivation[1] = (diff_R[1] - diff_R[0]) / dt;

    if(actuationMode == Link::JOINT_VELOCITY){
        double dq_target[2];
        dq_target[0] = Kp * (vx - (va * d * diff_L[1])) + Ki * (integral[0] * va * d) + Kd * (derivation[0] * va * d);
        dq_target[1] = Kp * (vx + (va * d * diff_R[1])) + Ki * (integral[1] * va * d) + Kd * (derivation[1] * va * d);
    }
}
```


ライントレース | 課題解答例

- ❖ `if(cnt[1] == 0 && cnt[2] == 0)`
 - ❖ 黄色と白色の線があるか判定
 - ❖ 存在しない場合は停止：指令値に0を設定
- ❖ 以降の処理は、`TB2TurnLineSampleController`と同様
- ❖ 0.1秒毎に時間、X座標、Y座標をファイルに出力
 - ❖ 『`%t`』と表示されている場合、
“`%`”をバックスラッシュ(`\`)に変更

ライントレース | 課題解答例

```
// 左右のホイールに指令値を与える
wheels[0]->dq_target() = dq_target[0];
wheels[1]->dq_target() = dq_target[1];

if(cnt[1] == 0 and cnt[2] == 0){
    // 黄色と白の線がなくなったら停止
    wheels[0]->dq_target() = 0.0;
    wheels[1]->dq_target() = 0.0;
}
}

waitTime = io->currentTime() - startTime;
// 0.1s毎にファイル出力を行う
if(waitTime >= INTERVAL){
    ofs << io->currentTime() << "\t" << rootLink->position().translation().x() << "\t" << rootLink->position().translation().y() << "\t" << endl;
    startTime = io->currentTime();
}

return true;
}
```

ライントレース | 課題解答例

❖ 『ofs.close()』

❖ シミュレーションが停止したら、ファイルをクローズ

```
virtual void stop() override
{
    for(int i = 0; i < WHEEL_NUM; ++i){
        diff_R[i] = diff_L[i] = 0;
        integral[i] = derivation[i] = 0;
    }

    for(int i = 0; i < 3; ++i){
        cnt[i] = 0;
    }
    cameraConnection.disconnect();
    ofs.close();
}

void onCameraStateChanged()
{
    // TB2StopLineSampleControllerと同様
}
};

CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(TB2AutoSampleController)
```