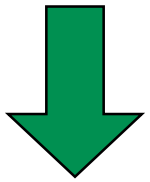
10-11コマ目
車輪移動ロボットの制御

移動ロボットの運動学

対向２輪型（２輪駆動１キャスタ）操作

- ロボット操作の考え方：
  2次元速度指令を各車輪の角速度に変換し，モータに指令する

```
struct Velocity2D {
        double va; // 角速度 [rad/s]
        double vx; // 並進速度(前方) [m/s]
        double vy; // 並進速度(横方向) [m/s]
};
```



目標速度になるように
車輪の角速度を求める

左車輪の角速度：ω_L，右車輪の角速度：ω_R

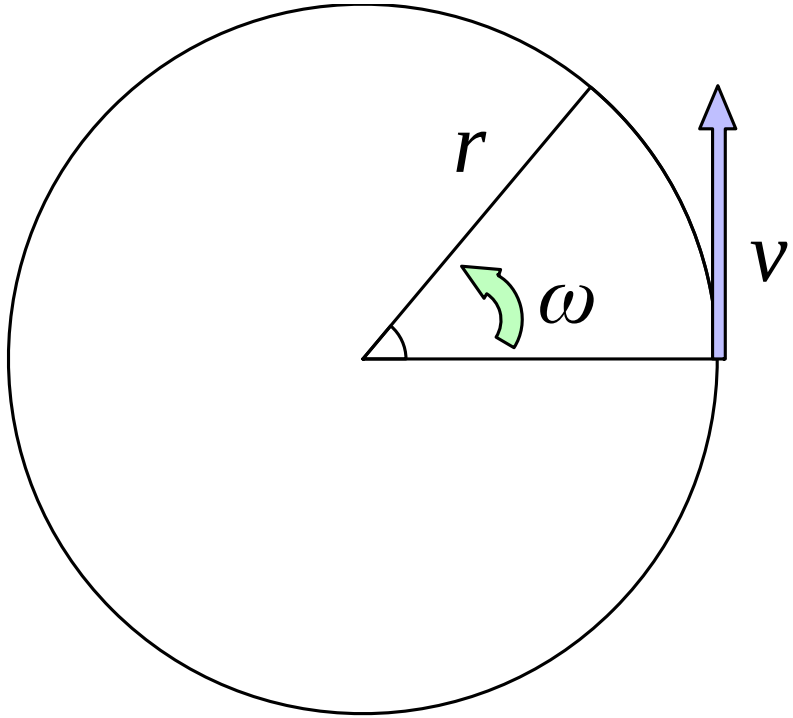
車輪移動ロボットの基礎

- 前提条件：車輪は滑らない

- この場合，以下のようにみなすことができる
  - 車輪は車軸に垂直方向にのみ移動する
  - 車両が運動しているとき，各瞬間ごとに
    「旋回中心」という点が存在し，すべて車輪の回転軸はこの点を通る
  - その瞬間，各車輪とロボットは旋回中心を中心とした円運動を行う

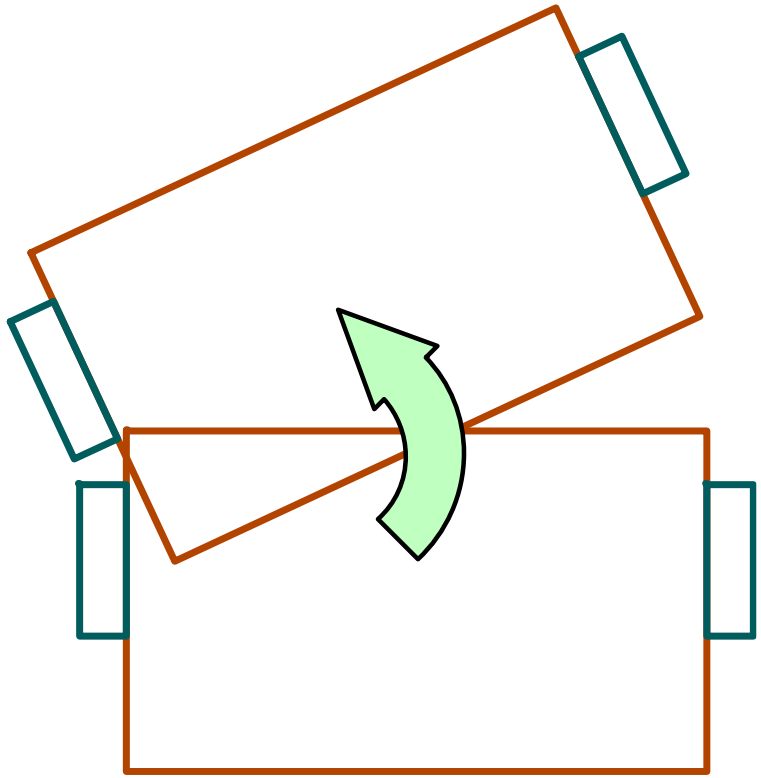
周速度

- 周速度：円周上の点の単位時間あたりの移動距離
  - 公式：速度をv，半径をr，角速度をωとすると 『$v = r\omega$』



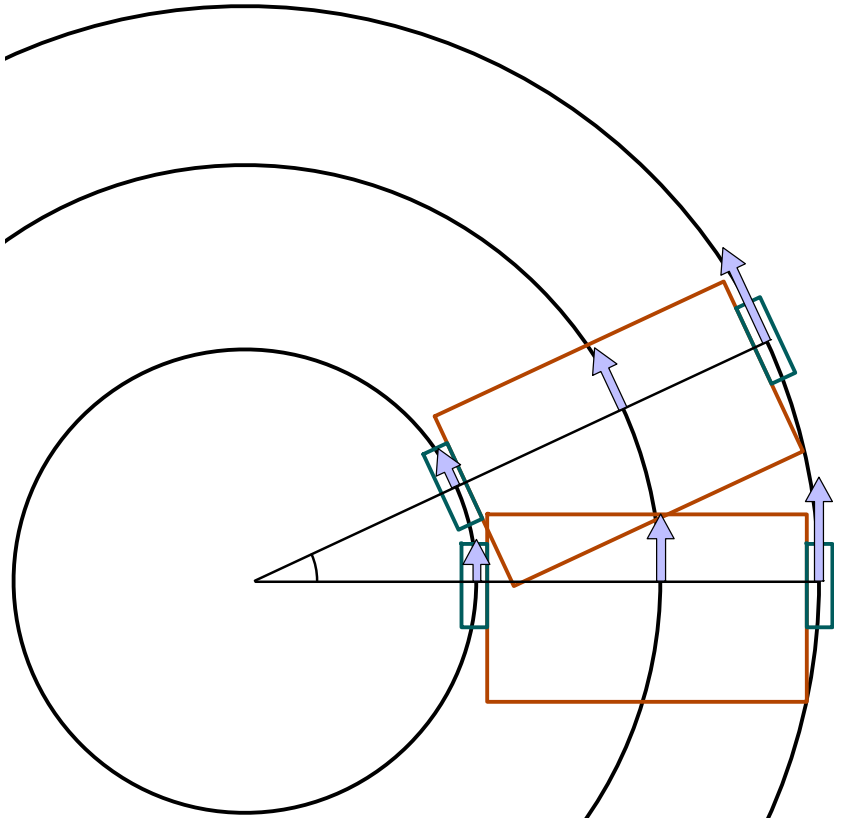
2輪ロボット左旋回の例

- 以下のように左旋回をする2輪ロボットを例に考える



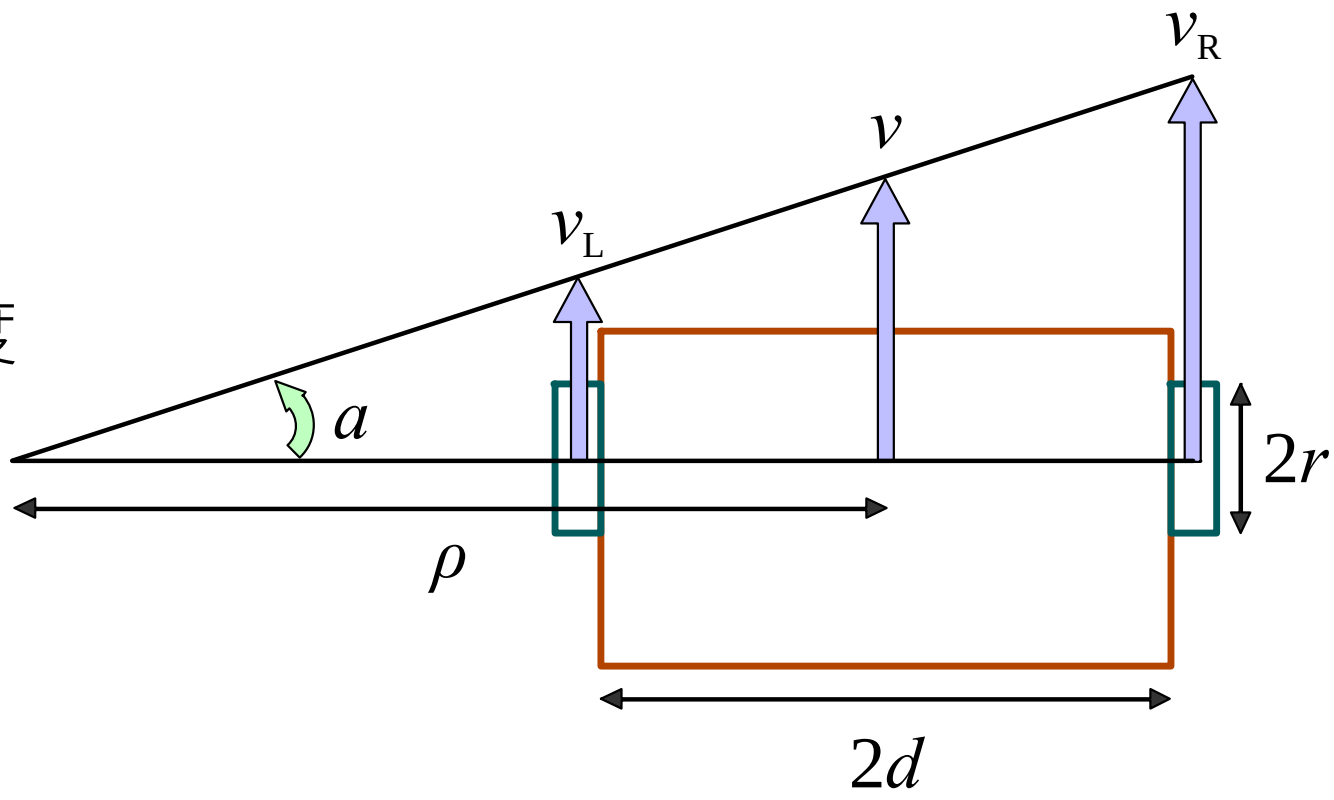
考え方

- 旋回の中心となる点から旋回半径とする円運動について考える
  1. 右車輪
  2. ロボット中心
  3. 左車輪



各パラメータについて

- v：ロボットの中心の速度
- a：角速度
- v_L：左車輪の接地点での速度
- v_R：右車輪の接地点での速度
- ρ：旋回半径
- d：中心から車輪までの距離
- r：車輪の半径



2輪ロボットの旋回速度

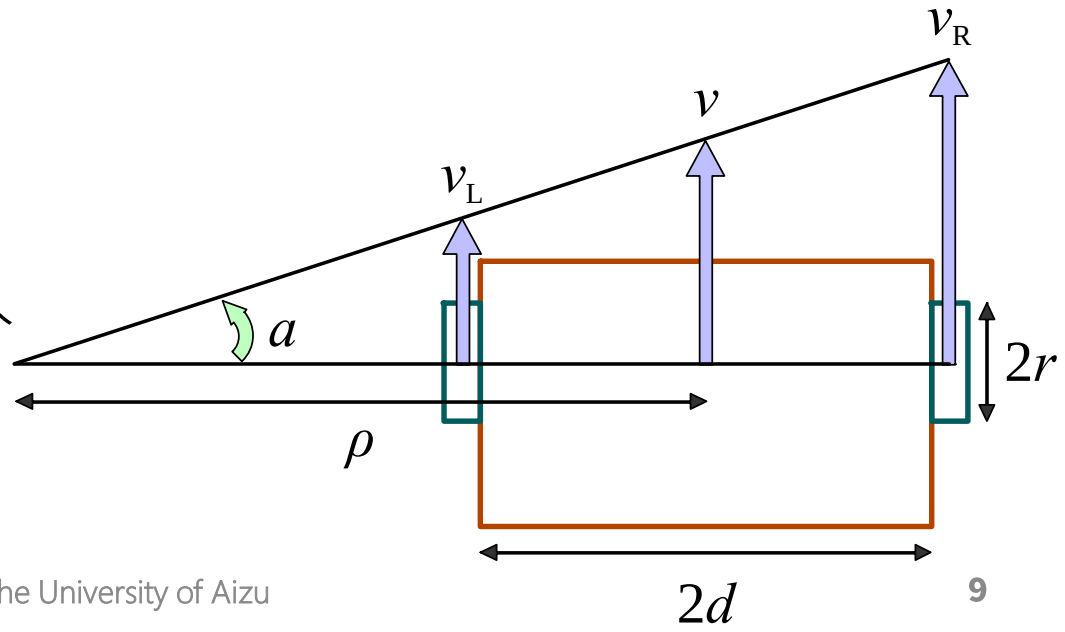
- 左旋回の場合のロボット中心の速度
  - 旋回半径ρ，角速度a，ロボット中心の速度vから，$v = \rho a$ ・・・①
- 左右の車輪の速度
  - 旋回半径ρに対して中心から車輪までの，距離dの分だけ増減する

    $v_L = (\rho - d)a$ ・・・②

    $v_R = (\rho + d)a$ ・・・③
- 式の整理
  - ①をρについて整理し，②， ③に代入

    $v_L = v - da$ ・・・④

    $v_R = v + da$ ・・・⑤

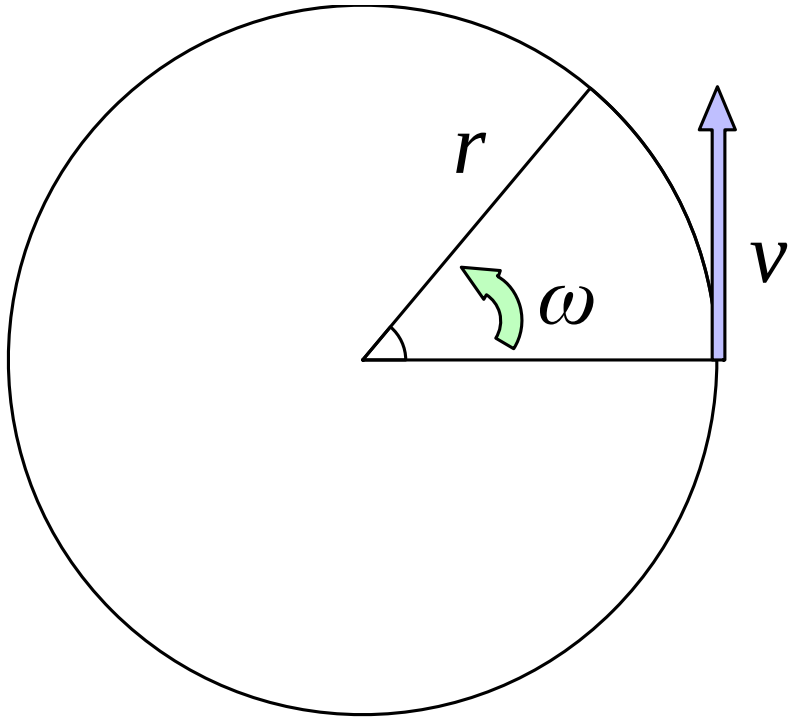


車輪の速度（v_L, v_R）

- 二輪ロボットの左右の車輪の速度：
  周速度の公式に当てはめると下記式になる

$$v_L = r\omega_L \quad \cdots ⑥$$

$$v_R = r\omega_R \quad \cdots ⑦$$



車輪の角速度

- これまでの導出した式④～⑦を整理して角速度を求める

$$v_L = v - da \quad \cdots ④$$

$$v_R = v + da \quad \cdots ⑤$$

$$v_L = r\omega_L \quad \cdots ⑥$$

$$v_R = r\omega_R \quad \cdots ⑦$$

- 式④と⑥から，$v - da = r\omega_L$ より，$\omega_L = (v - da) / r$
- 式⑤と⑦から，$v + da = r\omega_R$ より，$\omega_R = (v + da) / r$

モータ制御について

モータを制御

- 公式から，各車輪の角速度を求めることが可能

- モータを制御するには，最終的に目的の電圧を出力するモータドライバの値を計算する
  - つまり，角速度から回転数に変換後，モータドライバの値に変換

変換式

- 回転数から目的の電圧を出力するモータシールドの値へ変換する計算式は以下となる

```
値 = (value-in_min) * (out_max-out_min) // (in_max-in_min) + out_min
```

value：現在の回転数
in_min：回転数の範囲の下限
in_max：回転数の範囲の上限
out_min：モータドライバの範囲の下限
out_max：モータドライバの範囲の上限

これらの変数の値を知ることができれば，モータを制御することができる

モータとギヤボックス

- 回転数と電圧の関係
  - モータの回転数はかけられた電圧と比例するとここでは仮定する
  - 1.5V：6400r/min の場合，電圧を2倍にすると，回転数も2倍になる
  - 3.0V：12800r/min となる

- 回転数とギヤ比
  - モータの回転数が6400r/min,ギヤ比が344.2:1の場合，
    6400r/min÷344.2=18.5 となるため，
    回転数は18.5r/minとなる

回転数の上限下限

- 最大電圧
  - モータの電圧は限界電圧が3.0Vなので，最大電圧を3.0Vとする

- 回転数の最大最初値
  - ギア比が117.7なので0V,3V時の回転数は以下になります。
    最大回転数(3V)： (2×6400/117.7)= 108.75
    最小回転数(0V)： (0×6400/117.7)= 0

モータドライバーの上限下限値

- 回転数の上限下限を3V，0Vと分かったので，その電圧を出すモータドライバーの値を求める
- 上限値：
  3Vに近い電圧時の値から，
  $0x26_{(16)} \rightarrow 38_{(10)}$
- 下限値：
  0Vに近い電圧時の値から
  $0x01_{(16)} \rightarrow 1_{(10)}$

電圧表

VSET[5..0]	出力電圧	VSET[5..0]	出力電圧
0x00h	予約済み	0x20h	2.57
0x01h	予約済み	0x21h	2.65
0x02h	予約済み	0x22h	2.73
0x03h	予約済み	0x23h	2.81
0x04h	予約済み	0x24h	2.89
0x05h	予約済み	0x25h	2.97
0x06h	0.48	0x26h	3.05
0x07h	0.56	0x27h	3.13
0x08h	0.64	0x28h	3.21
0x09h	0.72	0x29h	3.29
0x0Ah	0.80	0x2Ah	3.37
0x0Bh	0.88	0x2Bh	3.45
0x0Ch	0.96	0x2Ch	3.53
0x0Dh	1.04	0x2Dh	3.61
0x0Eh	1.12	0x2Eh	3.69
0x0Fh	1.20	0x2Fh	3.77
0x10h	1.29	0x30h	3.86
0x11h	1.37	0x31h	3.94
0x12h	1.45	0x32h	4.02
0x13h	1.53	0x33h	4.10
0x14h	1.61	0x34h	4.18
0x15h	1.69	0x35h	4.26
0x16h	1.77	0x36h	4.34
0x17h	1.85	0x37h	4.42
0x18h	1.93	0x38h	4.50
0x19h	2.01	0x39h	4.58
0x1Ah	2.09	0x3Ah	4.66
0x1Bh	2.17	0x3Bh	4.74
0x1Ch	2.25	0x3Ch	4.82
0x1Dh	2.33	0x3Dh	4.90
0x1Eh	2.41	0x3Eh	4.98
0x1Fh	2.49	0x3Fh	5.06

データシート：http://akizukidenshi.com/download/ds/akizuki/AE-MOTOR8830_manual.pdf

現在の回転数

- 角速度から回転数に変換するには以下の式を使用する

$$\text{左車輪の回転数} = (\omega_L \times 60) \div (2 \times \pi)$$
$$\text{右車輪の回転数} = (\omega_R \times 60) \div (2 \times \pi)$$

- これにより，現在の各車輪の回転数が分かる

モータの向き

- モータの向きは角速度の向きで判断

  角速度 > 0：モータは正転

  角速度 < 0：モータは逆転

  角速度 = 0：モータは停止

DCモータサンプルプログラム

- 必要なimportについて

```
import smbus # PythonでI2Cを使用するために必要

import time  # sleep関数を使用するため

from time import sleep

import math  # piを計算するため
```

DCモータサンプルプログラム

- モータドライバのアドレス

```
bus = smbus.SMBus(1)          # I2Cバス番号
SLAVE_ADDRESS_LEFT = 0x64   # 左モータのアドレス
SLAVE_ADDRESS_RIGHT = 0x63 # 右モータのアドレス
```

DCモータサンプルプログラム

- 命令レジスタのアドレス

```
CONTROL = 0x00
```

- 出力の状態（モータの回転の向きの制御に使用する）

```
FORWARD = 0x01 # 正の回転
BACK = 0x02    # 負の回転
STOP = 0x00    # 停止
TOWARD =0x00   # モータの向き
```

DCモータサンプルプログラム

- 各パラメータの初期化

```
radius = 0.0275                    # タイヤの半径
tread = 0.046                      # タイヤ間の距離の半分
rpm = 6400                         # モータの回転数
gear = 114.7                       # ギア比
Maxrpm = int(2 * (rpm / gear))     # 3.0V時の回転数
Minrpm = 0 * (rpm / gear)          # 0V時の回転数
in_min = Minrpm                    # 回転数最小値
in_max = Maxrpm                    # 回転数最大値
out_min = 1                        # 電圧設定最小値
out_max = 38                       # 電圧設定最大値(3V)
Vx = 0.1                           # 速度
va = 0                             # 角速度
```

DCモータサンプルプログラム

- モータへ与える電圧への計算式

```
# 左タイヤの計算
# 左モータの角速度
omega_l = (vx + tread * va) / radius

#モータの回転向き確認
if omega_l > 0:
        TOWARD = FORWARD # 角速度がプラスならモータの回転は正
elif omega_l < 0:
        TOWARD = BACK    # 角速度がマイナスならモータの回転は負
else:
        TOWARD = STOP    # 角速度が0ならモータは停止

omega_l = abs(omega_l)  # 角速度がマイナスにならないようにする
```

DCモータサンプルプログラム

- モータへ与える電圧への計算式

```
# 左タイヤの計算
# 左モータの回転数
leftrpm = omega_l * 60 / (2 * math.pi)

# 左モータの回転数から電圧への変換
left_VSET = (leftrpm - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

# 電圧にモータの回転向きを加える ( ブリッジ制御を加える )
left_sval = TOWARD | ((int(left_VSET) + 5) << 2)
```

DCモータサンプルプログラム

- モータへ与える電圧への計算式

```
# 右タイヤの計算
# 右モータの角速度
omega_r = (vx + tread * va) / radius

#モータの回転向き確認
if omega_r > 0:
        TOWARD = FORWARD # 角速度がプラスならモータの回転は正
elif omega_r < 0:
        TOWARD = BACK    # 角速度がマイナスならモータの回転は負
else:
        TOWARD = STOP    # 角速度が0ならモータは停止

omega_r = abs(omega_r)  # 角速度がマイナスにならないようにする
```

DCモータサンプルプログラム

- モータへ与える電圧への計算式

```
# 右タイヤの計算
# 右モータの回転数
rightrpm = omega_r * 60 / (2 * math.pi)

# 右モータの回転数から電圧への変換
right_VSET  = (leftrpm - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

# 電圧にモータの回転向きを加える ( ブリッジ制御を加える )
right_sval = TOWARD |((int(right_VSET) + 5) << 2)
```

DCモータサンプルプログラム

- モータに電圧を与える

```
bus.write_i2c_block_data(SLAVE_ADDRESS_LEFT,CONTROL,[left_sval])   # 左モータ正回転
bus.write_i2c_block_data(SLAVE_ADDRESS_RIGHT,CONTROL,[right_sval]) # 右モータ正回転
sleep(2.5)

left_sval = BACK | ((int(left_VSET) + 5) << 2)
right_sval = BACK | ((int(right_VSET) + 5) << 2)

bus.write_i2c_block_data(SLAVE_ADDRESS_LEFT,CONTROL,[left_sval])   # 左モータ負回転
bus.write_i2c_block_data(SLAVE_ADDRESS_RIGHT,CONTROL,[right_sval]) # 右モータ負回転
sleep(2.5)

bus.write_i2c_block_data(SLAVE_ADDRESS_LEFT,CONTROL,[STOP])  # 左モータ停止
bus.write_i2c_block_data(SLAVE_ADDRESS_RIGHT,CONTROL,[STOP]) # 右モータ停止
```

移動ロボット用コンポーネント作成

- DCモータサンプルプログラムを参考に，
  移動用ロボットコンポーネントを作成する

- Pythonの場合，関数内で宣言した変数をその関数の範囲外で使用する場合，変数の先頭に`self.`を付ける

- コンポーネントのテストは9コマ目で作成したConsoleVelInを使用する
  - テスト時の値はVx=0.2，Va=0.1でお願いします

移動ロボット用コンポーネント仕様

コンポーネント名			
RobotCar			
概要			
マウス型ロボットカーの制御コンポーネント			
ポート名	フローポート	変数型	意味
VelIn	InPort	TimedVelocity2D	2次元速度ベクトルを入力

移動ロボット用コンポーネント仕様

- RTCBuilderで以下のように設定

基本
- モジュール名：RobotCar
- モジュール概要：任意(RobotCar component)
- バージョン：1.0.0
- ベンダ名：任意
- モジュールカテゴリ：任意(Category)
- コンポーネント型：STATIC
- アクティビティ型：PERIODIC
- コンポーネントの種類：DataFlow
- 最大インスタンス数：1
- 実行型：PeriodicExecutionContext
- 実行周期：1000.0

選択アクションコールバック
- onInitialize
- onAcivated
- onExcute
- onDeactivated

InPort
- ポート名：Vel
- データ型：RTC::TimedVelocity2D
- 変数名：VelValue
- 表示位置：LEFT

言語・環境
- Python

プログラム作成に関するヒント

- Pythonプログラムの編集手順
  - RobotCar.pyでモジュールを追加し，各関数を編集する
    - import：必要なモジュールをimportする
    - __init__：ポートの初期化を行う
    - onInitialize，onActivated：各パラメータの初期化を行う
    - onDeactivated：コンポーネント終了後、モータの停止の処理を行う
    - onExecute：Inportから値を受け取りモータを動かす処理を行う

RobotCar.py 解答例

コピー用テキスト

- 以下URLからコピー用のテキストが配置されている
  https://rtc-fukushima.jp/wp/wp-content/uploads/2018/11/10-11Program.zip
- 以下のファイルがあることを確認する
  - RobotCar.txt
- このテキストからコピー＆ペーストがしづらい方は，
  zipファイル内のテキストからコピー＆ペーストが可能

プログラムの流れ

1. InPortから値を読み込む
2. 左車輪の値を求めてモータドライバに出力
    1. 速度から左右の車輪の角速度を求める
    2. モータの向きを求める
    3. 角速度から回転数を求める
    4. 回転数から電圧を求める
    5. モータドライバに値を送る

プログラムの流れ

3. 右車輪の値を求めてモータドライバに出力
    1. 速度から左右の車輪の角速度を求める
    2. モータの向きを求める
    3. 角速度から回転数を求める
    4. 回転数から電圧を求める
    5. モータドライバに値を送る

Python

- Pythonプログラムの編集手順
  - RobotCar.pyの各関数を編集
    - import
      - モジュールをimportする
    - __init__
      - 起動時によばれ，ポートの初期化を行う
    - onInitialize
      - コンポーネント起動時によばれるときに1度だけ初期化を行う

Python

- Pythonプログラムの編集手順
  - RobotCar.pyの各関数を編集
    - `onActivated`
      - 非アクティブ状態からアクティブ化されるとき1度だけよばれる
    - `onDeactivated`
      - アクティブ状態から非アクティブ化されるとき1度だけよばれる
    - `onExecute`
      - アクティブ状態時に周期的によばれる

モジュールのimport

- 以下のモジュールを読み込む

```
import smbus
import math
```

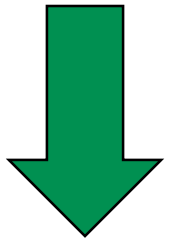
- importするモジュールが記載されている場所に以下のように追加

```
import RTC              # Open-RTM module
import OpenRTM_aist     # Open-RTM module
import smbus            # PythonでI2Cを使用するために必要
import math             # piを計算するため
```

__init__関数の変更

- __init__関数内のInPort，OutPortの初期化を以下のように変更する

```
self._d_VelValue = RTC.TimedVelocity2D(*VelValue_arg)
```



```
self._d_VelValue = RTC.TimedVelocity2D(RTC.Time(0, 0), RTC.Velocity2D(0.0, 0.0, 0.0))
```

onInitialize関数｜前半

- onInitialize関数に以下の変数を追加する

```
self.radius = 0.0275 # タイヤの半径
self.tread = 0.046   # タイヤ間の距離の半分
self.rpm = 6400      # モータの回転数
self.gear = 114.7    # ギア比
self.Maxrpm = int(2 * (self.rpm / self.gear)) # 3.0V時の回転数
self.Minrpm = 0 * (self.rpm / self.gear)      # 0V時の回転数
self.out_min = 1                              # 電圧設定最小値
self.out_max = 38                             # 電圧設定最大値(3V)
self.in_min = self.Minrpm                     # 回転数最小値
self.in_max = self.Maxrpm                     # 回転数最大値
self.right_sval = 0
self.left_sval = 0
```

onInitialize関数｜後半

- onInitialize関数に以下の変数を追加する

```
self.bus = smbus.SMBus(1)          # I2Cバス番号
self.SLAVE_ADDRESS_LEFT = 0x64   # 左モータのアドレス
self.SLAVE_ADDRESS_RIGHT = 0x63 # 右モータのアドレス
self.CONTROL = 0x00 # 命令レジスタのアドレス
self.FORWARD = 0x01 # 正の回転
self.BACK = 0x02     # 負の回転
self.STOP = 0x00     # 停止
self.TOWARD = 0x00   # モータの向き
```

onInitialize関数｜まとめ

- onInitialize関数を以下のように変更する

```
def onInitialize(self):
        self.radius = 0.0275 # タイヤの半径
        self.tread = 0.046   # タイヤ間の距離の半分
        self.rpm = 6400      # モータの回転数
              ～
        self.STOP = 0x00     # 停止
        self.TOWARD = 0x00   # モータの向き

        return RTC.RTC_OK
```

onActivated関数全文

- onActivated関数を以下のように編集する

```
def onActivated(self, ec_id):
      print "onActivated”
      return RTC.RTC_OK
```

onDeactivated関数全文

- onDeactivated関数を以下のように編集する

```
def onDeactivated(self, ec_id):
        print "onDeactivated”
        #タイヤ停止
        self.bus.write_i2c_block_data(self.SLAVE_ADDRESS_LEFT,self.CONTROL,[0x00])
        self.bus.write_i2c_block_data(self.SLAVE_ADDRESS_RIGHT,self.CONTROL,[0x00])

        return RTC.RTC_OK
```

onExecute関数 | InPortからの値の読み込み

- InPortから値の読み込み

```
if self._VelIn.isNew():
        # 値を読み込む
        readdata = self._VelIn.read()
        # 前回の値を保存
        old_left_sval = self.left_sval
        old_right_sval = self.right_sval
```

onExecute関数 | 左車輪値計算

- 速度から左車輪の角速度を求める

```
# 左タイヤの計算
# 左モータの角速度
omega_l = (readdata.data.vx-self.tread*readdata.data.va) / self.radius
```

onExecute関数 | 左車輪値計算

- モータの向きを求める

```
# モータの回転向き確認
if omega_l > 0:
        self.TOWARD = self.FORWARD # 角速度がプラスならモータの回転は正
elif omega_l < 0:
        self.TOWARD = self.BACK    # 角速度がマイナスならモータの回転は負
else:
        self.TOWARD = self.STOP    # 角速度が0ならモータは停止
omega_l = abs(omega_l)             # 角速度がマイナスにならないようにする
```

onExecute関数 | 左車輪値計算

- 角速度から回転数を求める

```
# 左モータの回転数
leftrpm = omega_l * 60 / (2 * math.pi)
```

- 回転数から電圧を求める

```
# 左モータの回転数から電圧への変換
left_VSET  = (leftrpm-self.in_min) * (self.out_max-self.out_min) // (self.in_max-self.in_min) + self.out_min

# 電圧にモータの回転向きを加える ( ブリッジ制御を加える )
self.left_sval = self.TOWARD | ((int(left_VSET) + 5) << 2)
```

onExecute関数 | 左車輪値計算

- I2Cで値をモータドライバーに送る

```
# 値がおかしくないか確認
if self.left_sval >= 1 and self.left_sval <= 255 and old_left_sval != self.left_sval:
        self.bus.write_i2c_block_data(self.SLAVE_ADDRESS_LEFT, self.CONTROL, [self.left_sval])
else:
        print "DCmotor1 value limite "+str(self.left_sval)
```

onExecute関数 | 右車輪値計算

- 速度から右車輪の角速度を求める

```
# 右タイヤの計算
# 右モータの角速度
omega_r = (readdata.data.vx+self.tread * readdata.data.va) / self.radius
```

onExecute関数 | 右車輪値計算

- モータの向きを求める

```
# モータの回転向き確認
if omega_r > 0:
        self.TOWARD = self.FORWARD # 角速度がプラスならモータの回転は正
elif omega_r < 0:
        self.TOWARD = self.BACK    # 角速度がマイナスならモータの回転は負
else :
        self.TOWARD = self.STOP    # 角速度が0ならモータは停止
omega_r = abs(omega_r)             # 角速度がマイナスにならないようにする
```

onExecute関数 | 右車輪値計算

- 角速度から回転数を求める

```
#右モータの回転数
rightrpm = omega_r * 60 / (2 * math.pi)
```

- 回転数から電圧を求める

```
#右モータの回転数から電圧への変換
right_VSET =  (rightrpm-self.in_min) * (self.out_max-self.out_min) // (self.in_max-self.in_min) + self.out_min

# 電圧にモータの回転向きを加える（ブリッジ制御を加える）
self.right_sval = self.TOWARD | ((int(right_VSET) + 5) << 2)
```

onExecute関数 | 右車輪値計算

- I2Cで値をモータドライバーに送る

```
# 値がおかしくないか確認
if self.right_sval >= 1 and self.right_sval <= 255 and old_right_sval != self.right_sval:
        self.bus.write_i2c_block_data(self.SLAVE_ADDRESS_RIGHT,self.CONTROL,[self.right_sval])
else:
        print "DCmotor2 value limite "+str(self.right_sval)
```

onExecute関数全文

- onExecute関数を以下のように編集する

```
def onExecute(self, ec_id):
        # InPort部分を書く
        # 左車輪設定内容を書く
        # 右車輪設定内容を書く

        return RTC.RTC_OK
```

制御コンポーネント作成

制御コンポーネント作成

- 作成した移動ロボット用コンポーネントに速度を与える

制御コンポーネント作成

- 前進→停止→後退→旋回を順番に行うコンポーネントを作成する

- 作成する際は一気にすべて作成するのではなく，段階的に作成する
  1. 前進の処理を行うコンポーネント作成
  2. 1.のプログラムに停止の処理を追加したコンポーネント作成
  3. 2.のプログラムに後退の処理を追加したコンポーネント作成
  4. 3.のプログラムに旋回の処理を追加したコンポーネント作成

コンポーネントの仕様

- RTCBuilderで以下のように設定

基本
- モジュール名：RobotControl
- モジュール概要：任意(Control component)
- バージョン：1.0.0
- ベンダ名：任意
- モジュールカテゴリ：任意(Category)
- コンポーネント型：STATIC
- アクティビティ型：PERIODIC
- コンポーネントの種類：DataFlow
- 最大インスタンス数：1
- 実行型：PeriodicExecutionContext
- 実行周期：1000.0

選択アクションコールバック
- onInitialize
- onAcivated
- onExcute
- onDeactivated

OutPort
- ポート名：Vel
- データ型：RTC::TimedVelocity2D
- 変数名：Vel
- 表示位置：RIGHT

言語・環境
- Python

前進のコンポーネント仕様

- プログラム流れ
  1. 速度ベクトル（**RTC::TimedVelocity2D**）に前進の値を入れる
  2. 出力（**Write**）する
  3. 1秒間コンポーネントを停止（**Sleep**）する

前進コンポーネントプログラム例

```
# 変数に前進の値を代入
self._d_Vel.data = RTC.Velocity2D(0.04, 0.0, 0.0)

# アウトポート(vel)に書き込み
self._VelOut.write()

# 1秒間コンポーネントを停止
time.sleep(1)
```

停止のコンポーネント仕様

- プログラム流れ
  1. 速度ベクトル（**RTC::TimedVelocity2D**）に前進の値を入れる
  2. 出力（**Write**）する
  3. 1秒間コンポーネントを停止（**Sleep**）する
  4. 速度ベクトル（**RTC::TimedVelocity2D**）に停止の値を入れる
  5. 出力（**Write**）する
  6. 1秒間コンポーネントを停止（**Sleep**）する

- 前進するプログラムを参考にプログラムを追記してください

後退のコンポーネント仕様

- プログラム流れ
  1. 速度ベクトル（**RTC::TimedVelocity2D**）に前進の値を入れる
  2. 出力（**Write**）する
  3. 1秒間コンポーネントを停止（**Sleep**）する
  4. 速度ベクトル（**RTC::TimedVelocity2D**）に停止の値を入れる
  5. 出力（**Write**）する
  6. 1秒間コンポーネントを停止（**Sleep**）する
  7. 速度ベクトル（**RTC::TimedVelocity2D**）に後退の値を入れる
  8. 出力（**Write**）する
  9. 1秒間コンポーネントを停止（**Sleep**）する
- 停止するプログラムを参考にプログラムを追記してください

旋回のコンポーネント仕様

- プログラム流れ
  1. 速度ベクトル（**RTC::TimedVelocity2D**）に前進の値を入れる
  2. 出力（**Write**）する
  3. 1秒間コンポーネントを停止（**Sleep**）する
  4. 速度ベクトル（**RTC::TimedVelocity2D**）に停止の値を入れる
  5. 出力（**Write**）する
  6. 1秒間コンポーネントを停止（**Sleep**）する
  7. 速度ベクトル（**RTC::TimedVelocity2D**）に後退の値を入れる
  8. 出力（**Write**）する
  9. 1秒間コンポーネントを停止（**Sleep**）する
  10. 速度ベクトル（**RTC::TimedVelocity2D**）に旋回の値を入れる
  11. 出力（**Write**）する
  12. 1秒間コンポーネントを停止（**Sleep**）する
  - 後退するプログラムを参考にプログラムを追記してください

RobotControl.py解答例

コピー用テキスト

- 以下URLからコピー用のテキストが配置されている
  https://rtc-fukushima.jp/wp/wp-content/uploads/2018/11/10-11Program.zip
- 以下のファイルがあることを確認する
  - RobotControl.txt
- このテキストからコピー＆ペーストがしづらい方は，
  zipファイル内のテキストからコピー＆ペーストが可能

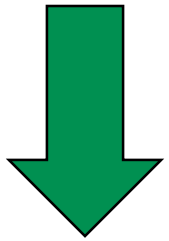
Python

- Pythonプログラムの編集手順
  - RobotControl.pyの各関数を編集
    - __init__
      - 起動時によばれ，ポートの初期化を行う
    - onActivated
      - 非アクティブ状態からアクティブ化されるとき1度だけよばれる
    - onDeactivated
      - アクティブ状態から非アクティブ化されるとき1度だけよばれる
    - onExecute
      - アクティブ状態時に周期的によばれる

__init__関数の変更

- __init__関数内のInPort，OutPortの初期化を以下のように変更する

```
self._d_Vel= RTC.TimedVelocity2D(*OutValue_arg)
```



```
self._d_Vel = RTC.TimedVelocity2D(RTC.Time(0, 0), RTC.Velocity2D(0.0, 0.0, 0.0))
```

onActivated関数全文

- onActivated関数を以下のように編集する

```
def onActivated(self, ec_id):
        # 値の初期化
        self._d_Vel.data.va = 0
        self._d_Vel.data.vx = 0
        self._d_Vel.data.vy = 0
        return RTC.RTC_OK
```

onDeactivated関数全文

- onDeactivated関数を以下のように編集する

```
def onDeactivated(self, ec_id):
      print "onDeactivated"
      return RTC.RTC_OK
```

onExecute関数 | 前進

- 前進の値を出力

```
# 前進指定
print "Front"
self._d_Vel.data = RTC.Velocity2D(0.2, 0.0, 0.0)
self._VelOut.write()
time.sleep(1)
```

onExecute関数 | 停止

- 停止の値を出力

```
# 停止指定
print "Stop"
self._d_Vel.data = RTC.Velocity2D(0.0, 0.0, 0.0)
self._VelOut.write()
time.sleep(1)
```

onExecute関数 | 後退

- 後退の値を出力

```
# 後退指定
print "Back"
self._d_Vel.data = RTC.Velocity2D(-0.2, 0.0, 0.0)
self._VelOut.write()
time.sleep(1)
```

onExecute関数 | 右旋回

- 右旋回の値を出力

```
# 右旋回指定
print "roll"
self._d_Vel.data = RTC.Velocity2D(0.0, 0.0, -5)
self._VelOut.write()
time.sleep(1)
```

onExecute関数全文

- onExecute関数を以下のように編集する

```
def onExecute(self, ec_id):
        # 前進指定内容を書く
        # 停止指定内容を書く
        # 後退指定内容を書く
        # 右旋回指定内容を書く

        return RTC.RTC_OK
```