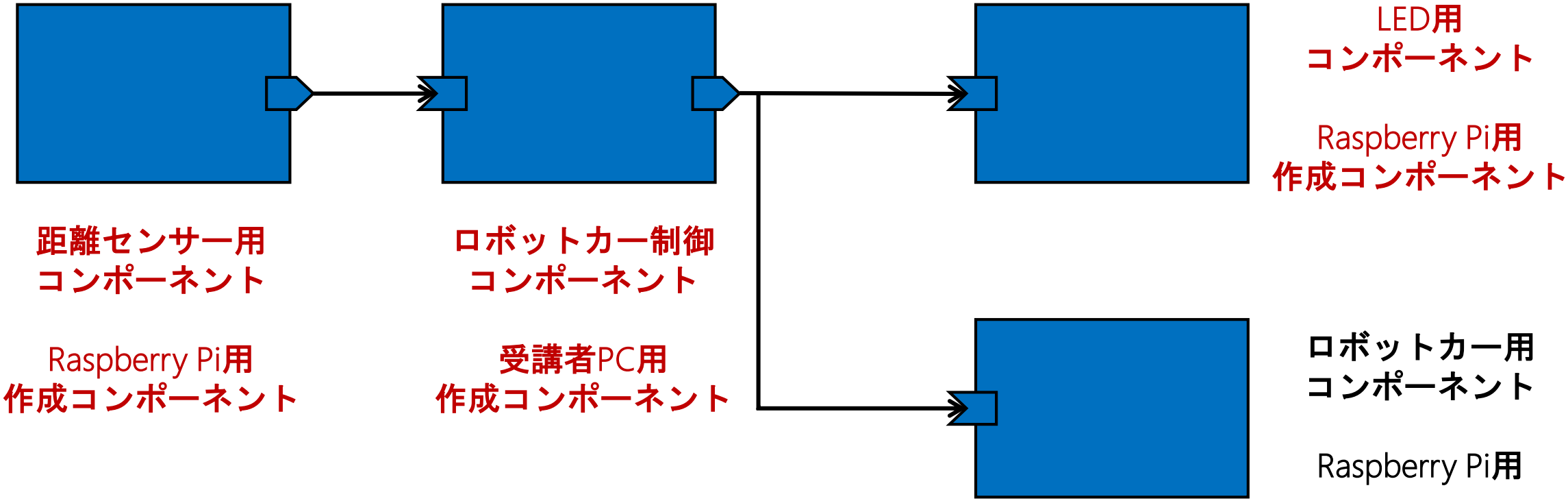
14コマ目，15コマ目，16コマ目
車輪移動ロボットの自律制御

距離センサなどを利用した障害物回避

実習内容

- 障害物との距離が一定以下になった場合，
  LEDを点灯させて，障害物を回避するコンポーネントを作成



LED用作成コンポーネント概要

コンポーネント名			
LED			
概要			
入力を受け取り，TrueならLEDを点灯，FalseならLEDを消灯する			
ポート名	フローポート	変数	意味
LED	InPort	TimedBoolean	LEDの状態を決める値を受け取る

RTC.TimedBoolean型

- RTC.TimedBooleanは構造体で以下のように定義されている

データ型	変数名	意味
Boolean	data	Ture か False
RTC.Time	tm	タイムスタンプ

- 使用例

```
self._d_LEDValue.data = True # dataにTrueを代入
```

距離センサ用作成コンポーネント概要

コンポーネント名			
Distance			
概要			
距離センサの値を出力する			
ポート名	フローポート	変数	意味
Distance	OutPort	TimedULong	距離センサ値を出力

RTC.TimedULong型

- RTC.TimedULongは構造体で以下のように定義されている

データ型	変数名	意味
int	data	整数の値を格納
RTC.Time	tm	タイムスタンプ

- 使用例

```
self._d_distanceValue.data = 1 # dataに1を代入
```

2輪用ロボット制御用コンポーネント概要

コンポーネント名			
RobotCarControl			
概要			
距離センサの値を受け取り，障害物を検知したら，LEDを光らせる値と 旋回の速度ベクトルを出力する			
ポート名	フローポート	変数	意味
Vel	OutPort	TimedVelocity2D	2次元速度ベクトル出力
Distance	InPort	TimedUlong	距離センサ値を受け取る
LED	OutPort	TimedBoolean	LEDの状態を出力

実習解答例

LEDセンサ用コンポーネント概要

コンポーネント名			
LED			
概要			
入力を受け取り，TrueならLEDを点灯，FalseならLEDを消灯			
ポート名	フローポート	変数	意味
LED	InPort	TimedBoolean	LEDの状態を決める値を受け取る

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：LED
- モジュール概要：任意(LED 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：LED
- データ型：RTC::TimedBoolean
- 変数名：LEDValue
- 表示位置：LEFT

言語・環境
- Python

プログラムの編集

プログラムの流れ

1. InPortから値を読み込む
2. 読み込んだ値がTrueならLED点灯
3. 読み込んだ値がFalseならLED消灯

Python

- Pythonプログラムの編集手順
  - LED.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
  - LED.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

- GPIOを使用するために以下のモジュールを読み込む

```
import RPi.GPIO as GPIO
```

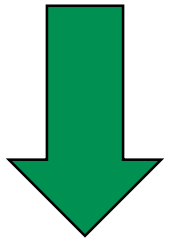
- importするモジュールが記載されている場所に以下のように追加

```
import RTC                  # Import RTM module
import OpenRTM_aist         # Import RTM module
import RPi.GPIO as GPIO # GPOIを使用するためのモジュール
```

__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_LEDValue = RTC.TimedBoolean(*LEDValue_arg)
```



```
self._d_LEDValue = RTC.TimedBoolean(RTC.Time(0, 0), False)
```

onInitialize関数全文

- onInitialize関数に以下の変数を追加する

```
self.LEDPIN = 4 # 使用するGPIOピン
```

- 以下のように追加する

```
def onInitialize(self):
        self.LEDPIN = 4 # 使用するGPIOピン
        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):
        # GPIOピンの設定の仕方(GPIOの数字で指定)
        GPIO.setmode(GPIO.BCM)
        # GPIO4を出力として使用
        GPIO.setup(self.LEDPIN, GPIO.OUT)

        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):
      GPIO.cleanup() # GPIOの終了
      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._LEDIn.isNew():
      # LEDの状態を読み込む
      self._d_LEDValue = self._LEDIn.read()
```

onExecute関数 | LED点灯

- 値を判定してLEDを点灯させる

```
# LEDを点灯
if self._d_LEDValue.data == True:
      GPIO.output(self.LEDPIN, True)
```

onExecute関数 | LED消灯

- 値を判定してLEDを消灯させる

```
# LEDを消灯
else:
        GPIO.output(self.LEDPIN, False)
```

onExecute関数全文

- onExecute関数を以下のように編集する

```
def onExecute(self, ec_id):
        if self._LEDIn.isNew():
                self._d_LEDValue = self._LEDIn.read() # LEDの状態を読み込む
                if self._d_LEDValue.data == True:
                        GPIO.output(self.LEDPIN, True)  # LEDを点灯
                else:
                        GPIO.output(self.LEDPIN, False) # LEDを消灯
        return RTC.RTC_OK
```

距離センサ用コンポーネント概要

コンポーネント名			
Distance			
概要			
距離センサの値を出力			
ポート名	フローポート	変数	意味
Distance	OutPort	TimedULong	距離センサ値を出力

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：Distance
- モジュール概要：任意(Distance 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：Distance
- データ型：RTC::TimedULong
- 変数名：DistanceValue
- 表示位置：RIGHT

言語・環境
- Python

プログラムの編集

プログラムの流れ

1. 距離センサからアナログ値を取得
2. アナログ値を距離に変換
3. 距離の値を出力

Python

- Pythonプログラムの編集手順
  - `Distance.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
  - Distance.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

- spiを使用するために以下のモジュールを読み込む

```
import spidev
```

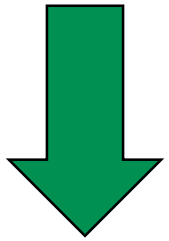
- importするモジュールが記載されている場所に以下のように追加

```
import RTC             # Import RTM module
import OpenRTM_aist # Import RTM module
import spidev          # spiを使用するためのモジュール
```

__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_DistanceValue = RTC.TimedULong(*OutValue_arg)
```



```
self._d_DistanceValue = RTC.TimedULong(RTC.Time(0, 0), 0)
```

onInitialize関数全文

- onInitialize関数に以下の変数を追加する

```
self.DISTANCE_PIN = 0       # A0コネクタにDistanceを接続
self.spi = spidev.SpiDev() # spiの初期宣言
```

- 以下のように追加する

```
def onInitialize(self):
        self.DISTANCE_PIN = 0       # A0コネクタにDistanceを接続
        self.spi = spidev.SpiDev() # spiの初期宣言
        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):
        self.spi.open(0, 0) # デバイスを接続
        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):
        self.spi.close() # デバイスを切断
        return RTC.RTC_OK
```

onExecute関数 | 距離センサから値を取得

- 距離センサから値を取得するプログラム

```
# 指定したチャンネルから値を受け取る
adc = self.spi.xfer2([1, (8 + self.DISTANCE_PIN) << 4, 0])

# 受け取った値をアナログ値に変換
data = ((adc[1] & 3) << 8) + adc[2]
```

onExecute関数 | 距離センサ値を距離に変換

- 距離センサ値を距離に変換するプログラム

```
if data <= 17: # 値が17だと値がおかしくなるため，18に変更する
      data = 18
# 距離変換式は http://appnote.tumblr.com/ を参考
distance = 5461 / (data - 17) - 2 # 距離変換式
print("data : {:3},distance : {:3}".format(data, distance))

if distance < 0: # 距離が0以下の場合は0にする
      distance = 0
```

onExecute関数 | OutPortから値の出力

- OutPortから値を出力するプログラム

```
self._d_DistanceValue.data = int(distance)

# 値を出力
self._DistanceOut.write()
```

onExecute関数全文

- onExecute関数を以下のように編集する

```
def onExecute(self, ec_id):
        # 指定したチャンネルから値を受け取る
        adc = self.spi.xfer2([1, (8 + self.DISTANCE_PIN) << 4, 0])
        # 受け取った値をアナログ値に変換
        data = ((adc[1] & 3) << 8) + adc[2]
        # アナログ値を距離に変換
        if data <= 17:
                data = 18
        # 距離変換式は http://appnote.tumblr.com/ を参考
        distance = 5461 / (data - 17) - 2 # 距離変換式
        print("data : {:3},distance : {:3} ".format(data, distance))
        if distance < 0:
                distance = 0
        self._d_DistanceValue.data = int(distance)
        # 値を出力
        self._DistanceOut.write()
        return RTC.RTC_OK
```

2輪用ロボット制御用コンポーネント概要

コンポーネント名			
RobotCarControl			
概要			
距離センサの値を受け取り，障害物を検知したら，LEDを光らせる値と 旋回の速度ベクトルを出力			
ポート名	フローポート	変数	意味
Vel	OutPort	TimedVelcity2D	2次元速度ベクトル出力
Distance	InPort	TimedUlong	距離センサ値を受け取る
LED	OutPort	TimedBoolean	LEDの状態を出力

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： RobotCarControl
- モジュール概要：任意(RobotCarControl 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：Distance
- データ型：RTC::TimedUlong
- 変数名：DistanceValue
- 表示位置：LEFT

OutPort
- ポート名：LED
- データ型：RTC::TimedBoolean
- 変数名：LEDValue
- 表示位置：RIGHT

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

言語・環境
- Python

プログラムの編集

プログラムの流れ

1. 距離の値を入力

2. 距離の値が規定値以上なら以下の処理
    1. LEDコンポーネントにTrueを出力
    2. 停止の速度を出力
    3. 後退の速度出力
    4. 旋回の速度出力

3. LEDコンポーネントにFalseを出力

4. 前進の速度出力

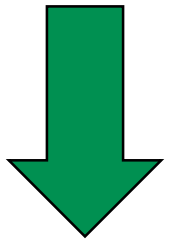
Python

- Pythonプログラムの編集手順
  - RobotCarControl.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_DistanceValue = RTC.TimedULong(*DistanceValue_arg)
self._d_LEDValue = RTC.TimedBoolean(*LEDValue_arg)
self._d_VelValue = RTC.TimedVelocity2D(*VelValue_arg)
```



```
self._d_DistanceValue = RTC.TimedULong(RTC.Time(0, 0), 0)
self._d_LEDValue = RTC.TimedBoolean(RTC.Time(0, 0), False)
self._d_VelValue = 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):
      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"
      return RTC.RTC_OK
```

onExecute関数 | InPortからの読み込み

- InPortからの読み込み

```
if self._DistanceIn.isNew():
        # 距離の値を読み込む
        self._d_DistanceValue = self._DistanceIn.read()
```

onExecute関数 | LEDコンポーネントに値出力

- 距離の値を規定値と比較しLEDコンポーネントに値を出力する

```
if self._d_DistanceValue.data < 10: # 距離の値を規定値と比較
    # LEDの状態の値(True)を出力
    self._d_LEDValue.data = True
    self._LEDOut.write()
```

onExecute関数 | 停止後退旋回の処理

- コンポーネントから値を出力する

```
# 停止指定
self._d_VelValue.data = RTC.Velocity2D(0.0, 0.0, 0.0)
self._VelOut.write()
time.sleep(0.5)
# 後退指定
self._d_VelValue.data = RTC.Velocity2D(-0.2, 0.0, 0.0)
self._VelOut.write()
time.sleep(1)
# 右旋回指定
self._d_VelValue.data = RTC.Velocity2D(0.0, 0.0, -5)
self._VelOut.write()
time.sleep(1)
```

onExecute関数| LEDコンポーネントに値出力

- LEDにFalseの値を出力

```
# LEDの状態の値(False)を出力
self._d_LEDValue.data = False
self._LEDOut.write()
```

onExecute関数 | 前進

- コンポーネントから値を出力する

```
# 前進指定
self._d_VelValue.data = RTC.Velocity2D(0.2, 0.0, 0.0)
self._VelOut.write()
```

onExecute関数全文 | 前半部分

- onExecute関数を以下のように編集する

```
def onExecute(self, ec_id):
        if self._DistanceIn.isNew():
                self._d_DistanceValue = self._DistanceIn.read() # 距離の値を読み込む

                if self._d_DistanceValue.data < 10: # 距離の値を規定値と比較
                        # LEDの状態の値(True)を出力
                        self._d_LEDValue.data = True
                        self._LEDOut.write()
                        # ロボットの旋回処理
                        # 停止指定
                        self._d_VelValue.data = RTC.Velocity2D(0.0, 0.0, 0.0)
                        self._VelOut.write()
                        time.sleep(0.5)
                        # 後退指定
```

onExecute関数全文 | 後半部分

- onExecute関数を以下のように編集する

```
                    # 後退指定（インデントに注意）
                    self._d_VelValue.data = RTC.Velocity2D(-0.2, 0.0, 0.0)
                    self._VelOut.write()
                    time.sleep(1)
                    # 右旋回指定
                    self._d_VelValue.data = RTC.Velocity2D(0.0, 0.0, -5)
                    self._VelOut.write()
                    time.sleep(1)
            # LEDの状態の値(False)を出力
            self._d_LEDValue.data = False
            self._LEDOut.write()
            # 前進指定
            self._d_VelValue.data = RTC.Velocity2D(0.2, 0.0, 0.0)
            self._VelOut.write()
        return RTC.RTC_OK
```

プログラムコピー用ZIP

- 以下URLからコピー用のZIPファイルをダウンロード可能
  https://rtc-fukushima.jp/wp/wp-content/uploads/2018/11/14-15-16Program.zip

- 以下ファイルがあることを確認
  - Distance.txt
  - LED.txt
  - RobotCarControl.txt

- プログラムのコピーは上記ファイルのから，
  コピーするようにしてください