

mruby-on-ev3rt+tecs_package

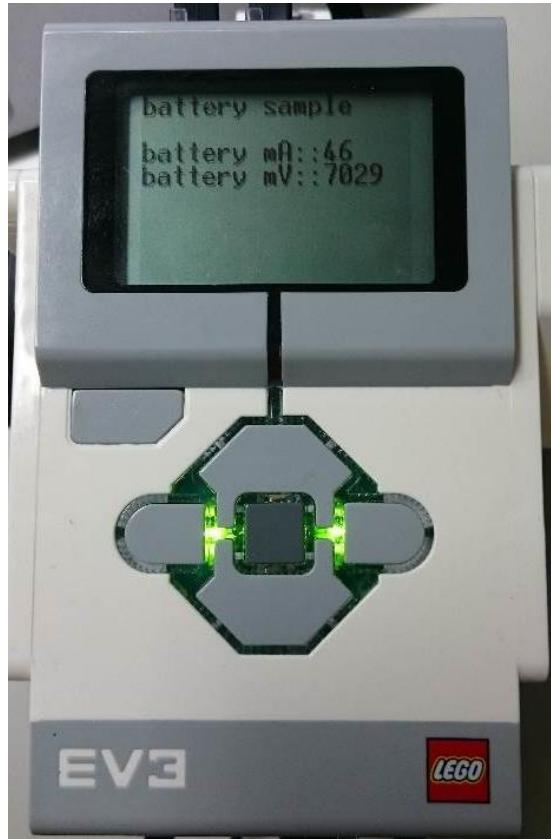
サンプルプログラムの説明

安積卓也 (大阪大学)
長谷川涼 (大阪大学)

最終更新日 : 2015/ 8/ 30

battery_sample

- バッテリの電流値と電圧値を測定し、LCDに表示する



battery_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font=:medium
```

```
    LCD.draw("battery sample", 0, 0)
```

```
loop{
```

```
    LCD.draw("battery mA::#{Battery_mA}", 0, 2)
```

```
    LCD.draw("battery mV::#{Battery_mV}", 0, 3)
```

```
}
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

Battery

はクラスインスタンス化 (new)
せずに利用する

button_sample

- EV3の各ボタン（上、下、左、右、エンター（中央））が押されるとどのボタンが押されたかをLCDに表示する

中央 (Enter) ボタンが
押された場合



右ボタンが押された場合



button_sample.rb

```
include EV3RT_TECS
begin
LCD.font=:medium
LCD.draw("button sample", 0, 0)
loop{
LCD.draw("left button ", 0, 2) if Button[:left].pressed?
LCD.draw("right button ", 0, 2) if Button[:right].pressed?
LCD.draw("up button ", 0, 2) if Button[:up].pressed?
LCD.draw("down button ", 0, 2) if Button[:down].pressed?
LCD.draw("enter button ", 0, 2) if Button[:enter].pressed?
break if Button[:back].pressed?
}
rescue => e
LCD.error_puts e
end
```

:left, :right, :up, :down, :enter
: backから選択
※:back電源OFFで利用

Button[button].pressed?は、
はクラスインスタンス化 (new)
せずに利用する

color_sample

- カラーセンサの値をLCDに表示するプログラム
- カラーセンサ (:port_4)



color_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font=:medium
```

```
    color_port = :port_4
```

ColorSensorはポートを指定して
インスタンス化 (new) する
:port_1、:port_2、:port_3、:port_4から選択
※シリアルを利用する場合は、:port_1を利用しない

```
    LCD.draw("color sample", 0, 0)
```

```
    LCD.draw("port #{color_port}", 0, 2)
```

```
    $color_sensor = ColorSensor.new(color_port)
```

オブジェクト (\$color_sensor) を指定して
メソッドを呼び出す

```
loop{
```

```
    color = $color_sensor.reflect
```

```
    LCD.draw("color reflect = #{color} ", 0, 5)
```

```
}
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

color_sample2

- カラーセンサで色を識別（黒、青、緑、黄色、赤、白、茶）しLCDで表示するプログラム
- カラーセンサ (:port_4)

黒色を認識した場合



白色を認識した場合



color_sample2.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font=:medium
```

```
    color_port = :port_4
```

```
    LCD.draw("color sample2", 0, 0)
```

```
    LCD.draw("port #{color_port}", 0, 2)
```

```
    $color_sensor = ColorSensor.new(color_port)
```

LCD.draw("color = #{\$color_sensor.color} ", 0, 5)
でも同様の出力が得られる

```
loop{
```

```
    LCD.draw("color = black ", 0, 5) if $color_sensor.black?
```

```
    LCD.draw("color = blue ", 0, 5) if $color_sensor.blue?
```

```
    LCD.draw("color = green ", 0, 5) if $color_sensor.green?
```

```
    LCD.draw("color = yellow", 0, 5) if $color_sensor.yellow?
```

```
    LCD.draw("color = red   ", 0, 5) if $color_sensor.red?
```

```
    LCD.draw("color = white ", 0, 5) if $color_sensor.white?
```

```
    LCD.draw("color = brown ", 0, 5) if $color_sensor.brown?
```

```
}
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

ev3way_sample

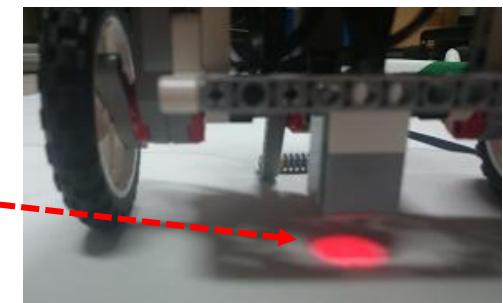
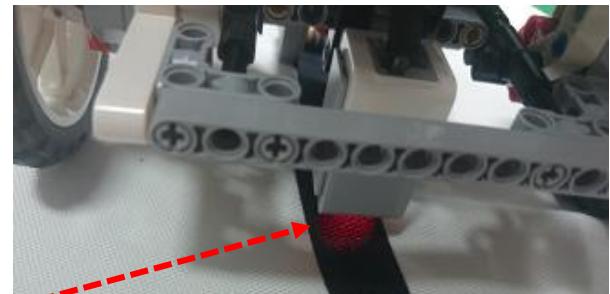
- ET口ボコン用のサンプル
 - 倒立制御しながら、ライントレースを行う
- タッチセンサ (:port_1)
- カラーセンサ (:port_2)
- ジャイロセンサ (:port_3)
- 超音波センサ (:port_4)
- しっぽモータ (:port_a)
- 右モータ (:port_b)
- 左モータ (:port_c)



ev3way_sample

- 操作手順

- 電源を入れる
- 黒色のライン上にカラーセンサを移動
- タッチセンサを押す：黒色の値を取得
- 白色の上にカラーセンサを移動
- タッチセンサを押す：白色の値を取得
：しっぽを下ろす
- ライン上移動
- タッチセンサを押す：ライントレーススタート



ev3way_sample.rb : 初期化

```
begin
```

```
LCD.puts "ev3way_sample.rb"  
LCD.puts "--- mruby version -"  
Speaker.volume = 1  
forward = turn = 0
```

ひとつしかないもの（ポート番号指定不要）は、
クラスメソッドとして直接呼び出す

```
#各オブジェクトを生成・初期化する
```

```
$sonar = UltrasonicSensor.new(SONAR_SENSOR)  
$color = ColorSensor.new(COLOR_SENSOR)  
$color.reflect  
$touch = TouchSensor.new(TOUCH_SENSOR)  
$gyro = GyroSensor.new(GYRO_SENSOR)
```

ポート番号を指定して初期化
(インスタンス化)

```
$motor_l = Motor.new(LEFT_MOTOR)  
$motor_r = Motor.new(RIGHT_MOTOR)  
$motor_t = Motor.new(TAIL_MOTOR)  
$motor_t.reset_count
```

```
#各オブジェクトを生成・初期化する
```

```
LED.color = :orange
```

```
...
```

ev3way_sample.rb : 黒色、白色の取得

```
#黒・白色のキャリブレーション
```

```
$black_value = color_calibration -----
```

```
LCD.puts "black::#$black_value"
```

```
$white_value = color_calibration -----
```

```
LCD.puts "white::#$white_value"
```

ライントレースの
基準値を計算

```
threshold = ((black_value + white_value) / 2).round
```

```
# スタート待機
```

```
LCD.puts "Ready to start"
```

タッチセンサが押されるまで待つ

カラーセンサn回取得し、
平均値を取得

```
def color_calibration(n=10)
  loop {
    break if $touch.pressed?
    RTOS.delay(10)
  }
  col = 0
  n.times { col += $color.reflect}
  col = (col / n).round
  Speaker.tone(:a4, 200)
  RTOS.delay(500)
  col
end
```

ev3way_sample.rb : スタート準備

```
# スタート待機
LCD.puts "Ready to start"
loop {
    # 完全停止用角度に制御
    tail_control(TAIL_ANGLE_STAND_UP)
    RTOS.delay(10)
    # タッチセンサが押されるまで待つ
    break if $touch.pressed?
}
#走行モータエンコーダリセット
$motor_l.reset_count
$motor_r.reset_count
# ジャイロセンサリセット
$gyro.reset

# LED:緑 走行状態
LED.color = :green
```

しっぽの位置を指定された角度に保つ
(フィードバック制御)

```
def tail_control(angle)
    目標値           現在の値
    ↓               ↓
    pwm = ((angle - $motor_t.count) * P_GAIN).to_i
    pwm = (pwm > PWM_ABS_MAX) ? PWM_ABS_MAX :
    (pwm < -PWM_ABS_MAX) ? -PWM_ABS_MAX : pwm
    $motor_t.power = pwm
    $motor_t.stop(true) if pwm == 0
end
```

ev3way_sample.rb : ライントレース

障害物まで一定の距離以下に
なると止まる

```
# main loop
forward = turn = 0
loop {
    start = RTOS.msec
    # バランス走行用角度に制御
    tail_control(TAIL_ANGLE_DRIVE)
    # 障害物検知
    if sonar_alert
        forward = turn = 0
    else
        # Line trace
        turn = $color.reflect >= threshold ? 20 : -20
        forward = 30
    end
    ...
}
```

サンプルでは、30に固定

```
def sonar_alert
    $sonar_counter += 1
    if $sonar_counter == 10
        distance = $sonar.distance
        $sonar_alert = distance <=
            SONAR_ALERT_DISTANCE
        && distance >= 0
        $sonar_counter = 0
    end
    $sonar_alert
end
```

カラーセンサと閾値と比較し
どちらかに曲がる
ここを変更すると、
自前のライントレースが可能

ev3way_sample.rb : 倒立制御

```
# main loop
loop {
    start = RTOS.msec
    ...
    # 倒立振子制御APIを呼び出し、倒立走行するための
    # 左右モータ出力値を得る */
    pwm_l, pwm_r = Balancer.control(
        forward.to_f,
        turn.to_f,
        -$gyro.rate.to_f,
        GYRO_OFFSET,
        $motor_l.count.to_f,
        $motor_r.count.to_f,
        Battery.mV.to_f)
    $motor_l.stop(true) if pwm_l == 0
    $motor_l.power = pwm_l
    $motor_r.stop(true) if pwm_r == 0
    $motor_r.power = pwm_r
    wait = 4 - (RTOS.msec - start)
    RTOS.delay(wait) if wait > 0
}
```

バランサの返り値が2つ

C言語で実装されたバランサを呼び出す

4ミリ秒周期で実行
現状1ミリ秒程度で処理完了
mrubyでも十分制御可能

gyro_sample

- ジャイロセンサの値（角度）をLCDに表示するプログラム
- ジャイロセンサ (:port_4)



gyro_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font=:medium
```

```
    gyro_port = :port_4
```

```
    $gyro_sensor = GyroSensor.new(gyro_port)
```

GyroSensorはポートを指定して
インスタンス化 (new) する

:port_1、:port_2、:port_3、:port_4から選択
※シリアルを利用する場合は、:port_1を利用しない

```
    LCD.draw("gyro sample", 0, 0)
```

```
    LCD.draw("port #{gyro_port}", 0,2)
```

```
    LCD.draw("gyro reset #{$gyro_sensor.reset}", 0,3)
```

```
loop{
```

```
    gyro = $gyro_sensor.angle
```

```
    LCD.draw("gyro = #{gyro} ", 0, 4)
```

```
    RTOS.delay(10)
```

```
}
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

lcd_sample

- LCDコンソールに出力(LCD.puts)をするプログラム
：コンソールモード
- 左、右、中央ボタンで押されたボタンをコンソールに表示
- 上、下のボタンでコンソールを移動
- 戻るボタン長押しでプログラム終了



lcd_sample.rb

```
include EV3RT_TECS

begin
    LCD.font=:medium
    LCD.puts "lcd sample"
    loop{
        LCD.puts "left button " if Button[:left].pressed?
        LCD.puts "right button " if Button[:right].pressed?
        LCD.puts "enter button " if Button[:enter].pressed?
        break if Button[:back].pressed?

        RTOS.delay(100)
    }

    rescue => e
        LCD.error_puts e
end
```

LCD

はクラスインスタンス化 (new)
せずに利用する

lcd_sample2

- LCDの指定した座標（フォント）に表示（LCD.draw）するプログラム：drawモード



lcd_sample2.rb

```
include EV3RT_TECS
```

```
begin
```

```
LCD.font=:medium  
#LCD.font=:small
```

:mediumまたは、:smallを選択

左上から、178（横幅）128（高さ）の長方形を黒色(:black)で塗りつぶす

```
LCD.fill_rect(0, 0, 178, 128, :black)
```

```
LCD.draw("lcd sample2", 0, 0)
```

```
15.times{|i|
```

```
    LCD.draw("#{i}", i, i+1)
```

```
}
```

```
rescue => e
```

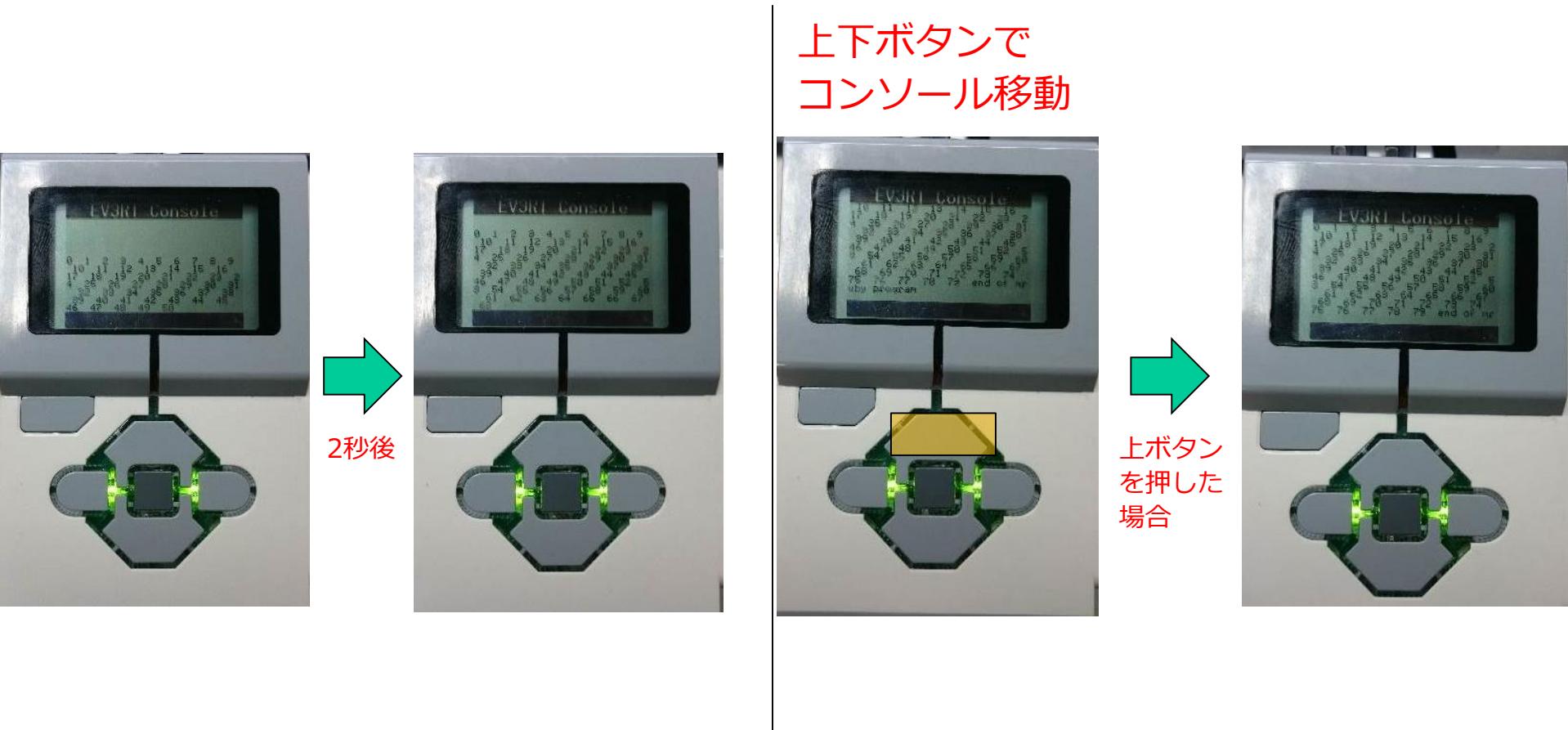
```
    LCD.error_puts e
```

```
end
```

i (x:横), i+1 (y:縦)の
フォントサイズ分ずらしたところに
iを表示

lcd_sample3

- LCD.printを利用したのプログラム：コンソールモード
- 200ミリごとに、数字を順番に表示する。



lcd_sample3.rb

```
include EV3RT_TECS
```

```
begin
```

```
  80.times{|i|  
    LCD.print "#{i} "  
    RTOS.delay(200)  
  }
```

```
rescue => e
```

```
  LCD.error_puts e
```

```
end
```

led_sample

- 押されたボタンに応じた、本体のLEDの色を変更するプログラム
- 左ボタン：緑
- 右ボタン：オレンジ
- 上ボタン：赤
- 下ボタン：消灯
- 中央ボタン：オレンジ



led_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font=:medium
```

```
    LCD.draw("led sample", 0, 0)
```

```
loop{
```

```
    LED.color=:green if Button[:left].pressed?
```

```
    LED.color=:orange if Button[:right].pressed?
```

```
    LED.color=:red   if Button[:up].pressed?
```

```
    LED.off         if Button[:down].pressed?
```

```
    LED.color=:orange if Button[:enter].pressed?
```

```
    break if Button[:back].pressed?
```

```
}
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

LED

はクラスインスタンス化 (new)
せずに利用する

motor_sample

- 超音波センサの値（距離）が15cm以上のとき前進し、15cm未満の時は停止する
- 左モータ (:port_a)
- 右モータ (:port_b)
- 超音波センサ (:port_3)



motor_sample.rb : 前半

```
include EV3RT_TECS
begin
LCD.font=:medium
LCD.draw("motor sample", 0, 0)
# Sensors and Actuators
left_port = :port_a
right_port = :port_b
ultrasonic_port = :port_3
LCD.draw("left motor:#{left_port} ", 0, 2)
LCD.draw("right motor:#{right_port} ", 0, 3)
LCD.draw("ultrasonic :#{ultrasonic_port}", 0, 4)
$left_motor = Motor.new(left_port)
$right_motor = Motor.new(right_port)
$ultrasonic_sensor = UltrasonicSensor.new(ultrasonic_port)
```

Motorはポートを指定して
インスタンス化 (new) する
`:port_a`、`:port_b`、`:port_c`、`:port_d`から選択

motor_sample.rb:後半

```
loop{
    distance = $ultrasonic_sensor.distance
    LCD.draw("distance = #{distance} ", 0, 6)

    if distance < 15 then
        $left_motor.stop
        $right_motor.stop
    else
        $left_motor.power=30
        $right_motor.power=30
    end
}
rescue => e
LCD.error_puts e
end
```

motor_sample2

- モータ(rotate, reset, counts)サンプルプログラム



rtos_sample

- 性能測定をするサンプルプログラム



rtos_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    n = 10
```

```
    #測定オーバヘッドの測定  
    startu = RTOS.usec
```

```
    n.times{
```

```
        RTOS.usec
```

```
}
```

```
    endu = RTOS.usec
```

```
    overhead = (endu - startu) / (n + 1)
```

```
    a=[*1..10]
```

```
    sum = 0
```

```
    #測定開始
```

```
    startu= RTOS.usec
```

```
    a.each{|num|
```

```
        sum = sum + num
```

```
}
```

```
    #測定終了
```

```
    endu = RTOS.usec
```

```
    LCD.puts "sum=#{sum}"
```

```
    LCD.puts "#{endu - startu - overhead}usec"
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

RTOS

はクラスインスタンス化 (new)
せずに利用する

測定オーバヘッドの測定

測定したい処理

例では、 1 ~ 1 0 の合計を求める

計算結果の表示

測定結果の表示

speaker_sample

- 配列で定義された音を順番に鳴らすプログラム



speaker_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font=:medium
```

```
    LCD.puts "speaker sample"
```

```
    Speaker.volume = 1
```

Speaker
はクラスインスタンス化 (new)
せずに利用する

トーンと長さをArrayで定義

```
[[{:f5, 150}, {:f5, 150}, {:f5, 150}, {:f5, 200}, {:e5, 300}, {:g5, 300}, {:f5, 400}].each do |tone_duration|
```

```
    LCD.puts tone_duration
```

```
    Speaker.tone(tone_duration[0], tone_duration[1])
```

トーンと長さを取り出し、
音を鳴らす

```
    RTOS.delay(tone_duration[1])
```

```
end
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

speaker_sample2

- 押されたボタンに応じた、音を鳴らすプログラム
- 左ボタン : C4
- 右ボタン : C5
- 上ボタン : C6
- 下ボタン : F4
- 中央ボタン : F5



speaker_sample2.rb

```
include EV3RT_TECS

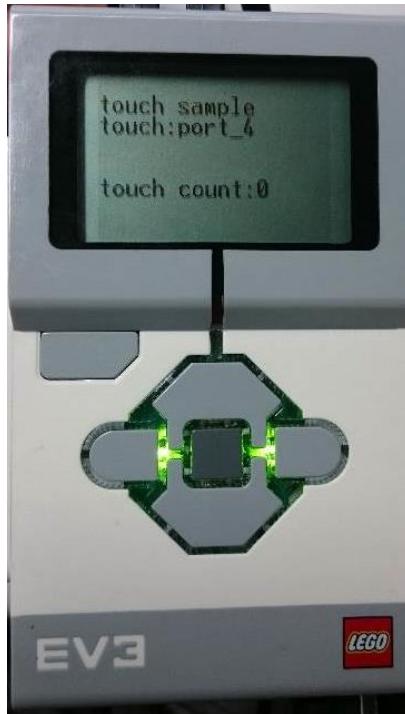
begin
  LCD.font=:medium
  LCD.draw("speaker sample", 0,1)

  Speaker.volume= 1
  loop{
    Speaker.tone(:c4, 30)      if(Button[:left].pressed?)
    Speaker.tone(:c5, 30)      if(Button[:right].pressed?)
    Speaker.tone(:c6, 30)      if(Button[:up].pressed?)
    Speaker.tone(:f4, 30)      if(Button[:down].pressed?)
    Speaker.tone(:f5, 30)      if(Button[:enter].pressed?)
    RTOS.delay(30)
  }
  rescue => e
    LCD.error_puts e
  end
```

touch_sample

- タッチセンサの押された回数をLCDに表示するプログラム
- タッチセンサ (:port_4)

起動時



タッチセンサを2回押した後



touch_sample.rb

```
include EV3RT_TECS
```

```
begin
```

```
    LCD.font= :medium
```

```
    LCD.draw("touch sample", 0,1)
```

```
    touch_port = :port_4
```

TouchSensorはポートを指定して
インスタンス化 (new) する
:port_1、:port_2、:port_3、:port_4から選択
※シリアルを利用する場合は、:port_1を利用しない

```
    LCD.draw("touch:#{touch_port}", 0, 2)
```

```
    $touch_sensor = TouchSensor.new(touch_port)
```

```
    count = 0
```

```
    loop{
```

```
        LCD.draw("touch count:#{count}", 0, 5)
```

```
        while !$touch_sensor.pressed? do end
```

```
        while $touch_sensor.pressed? do end
```

```
        count = count + 1
```

```
}
```

```
    rescue => e
```

```
        LCD.error_puts e
```

```
end
```

ultrasonic_sample

- 超音波信号の検知結果を表示 (true,false)
- 超音波センサの値（距離）を表示
- 超音波センサ (:port_3)

検知できなかった場合



検知でできた場合



ultrasonic_sample.rb

```
include EV3RT_TECS
begin
LCD.font=:medium

LCD.draw("ultrasonic sample", 0, 0)
# Sensors and Actuators

ultrasonic_port = :port_3


UltrasonicSensorはポートを指定して  
インスタンス化 (new) する  
:port_1、:port_2、:port_3、:port_4から選択  
※シリアルを利用する場合は、:port_1を利用しない



LCD.draw("ultrasonic:#{$ultrasonic_sensor.port}", 0, 2)
$ultrasonic_sensor = UltrasonicSensor.new(ultrasonic_port)

LCD.draw("listen = #{$ultrasonic_sensor.listen}", 0, 3)
loop{
  distance = $ultrasonic_sensor.distance
  LCD.draw("distance = #{distance}", 0, 4)
}
rescue => e
LCD.error_puts e
end
```