

# mruby-on-ev3rt+tecs\_package

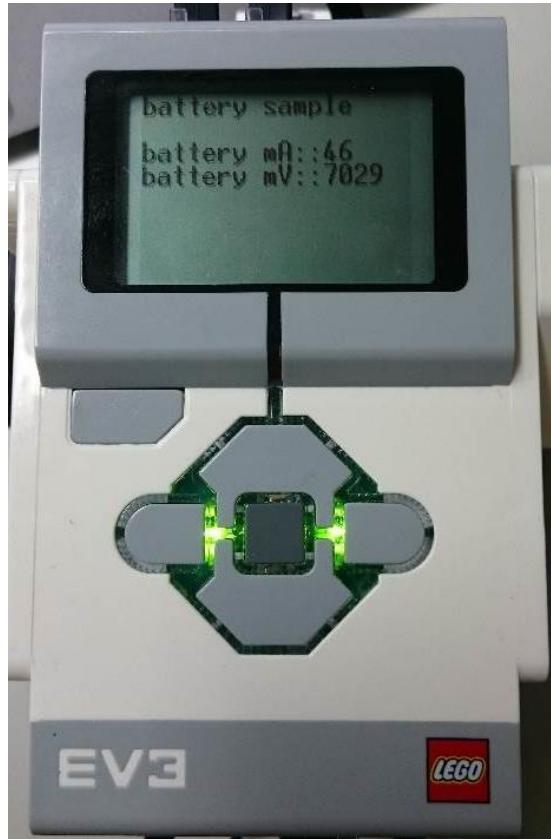
## サンプルプログラムの説明

安積卓也 (大阪大学)  
長谷川涼 (大阪大学)  
小南靖雄 (TOPPERS個人会員)

最終更新日 : 2018/ 5/ 16

# 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)
```

Battery

はクラスインスタンス化 (new)  
せずに利用する

```
    while true
```

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

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

```
    end
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

# 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)
while true
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?
end
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 (Parameter.color\_sensor\_port)



# color\_sample.rb

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

color_port = Parameter color_senso
LCD.draw("color sample", 0, 0)
LCD.draw("port #{color_port}", 0, 2)

color_sensor = ColorSensor.new(color_port)
while true
  color = color_sensor.reflect
  LCD.draw("color reflect = #{color} ", 0, 5)
end
rescue => e
LCD.error_puts e
end
```

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

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

# 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 (Parameter.color\_sensor\_port)

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 = Parameter.color_sensor_port
```

```
    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

```
while true
```

```
    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?
```

```
end
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

## color\_sample3

- カラーセンサで色をRGB形式で取得しLCDで表示するプログラム
- カラーセンサ (Parameter.color\_sensor\_port)  
RGB形式で表示



# color\_sample3.rb

```
include EV3RT_TECS
```

```
begin
```

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

```
    color_port = Parameter.color_sensor_port
```

```
    LCD.draw("color sample3", 0, 0)
```

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

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

```
    while true
```

```
        r, g, b = color_sensor.rgb
```

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

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

```
        LCD.draw("b = #{b}", 0, 6)
```

```
        RTOS.delay(500)
```

```
    end
```

```
    rescue => e
```

```
        LCD.error_puts e
```

```
    end
```

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

# ev3way\_sample

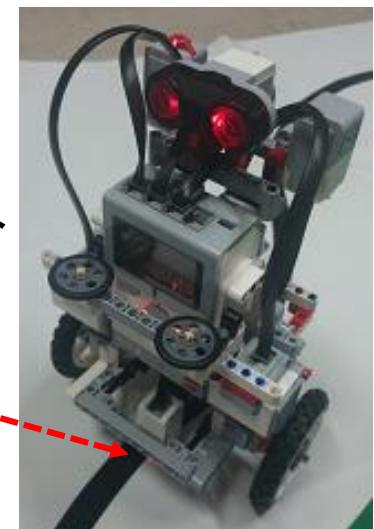
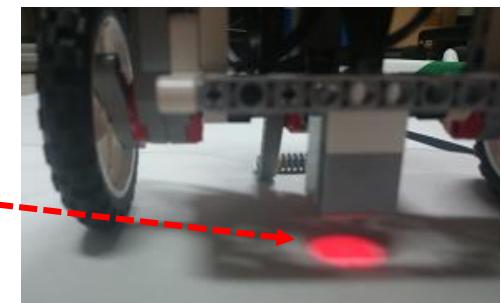
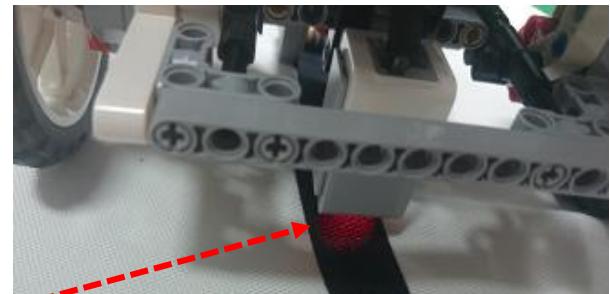
- ET口ボコン用のサンプル
  - 倒立制御しながら、ライントレースを行う
- タッチセンサ (Parameter.touch\_sensor\_port)
- カラーセンサ (Parameter.color\_sensor\_port)
- ジャイロセンサ (Parameter.gyro\_sensor\_port)
- 超音波センサ (Parameter.sonar\_sensor\_port)
- しっぽモータ (Parameter.tail\_motor\_port)
- 右モータ (Parameter.right\_motor\_port)
- 左モータ (Parameter.left\_motor\_port)



# 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)
  while true
    break if @touch.pressed?
    RTOS.delay(10)
  end
  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"
while true
    # 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?
end
#走行モータエンコーダーリセット
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)
    pwm = ((angle - @motor_t.count) * P_GAIN).to_i
    if pwm > PWM_ABS_MAX
        pwm = PWM_ABS_MAX
    elsif pwm < -PWM_ABS_MAX
        pwm = -PWM_ABS_MAX
    end
    if pwm == 0
        @motor_t.stop(true)
    else
        @motor_t.power = pwm
    end
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
while true
    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
        forward = 30
        turn = @color.reflect >= threshold ? 20 : -20
    end
    ...
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_var = distance <=
            SONAR_ALERT_DISTANCE
        && distance >= 0
        @sonar_counter = 0
    end
    @sonar_alert_var
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
while true
    ...
    # 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, turn,
        @gyro.rate, GYRO_OFFSET,
        motor_l.count, motor_r.count,
        Battery.mV, BACKLASHHALF)
    if pwm_l == 0
        motor_l.stop(true)
    else
        motor_l.power = pwm_l
    end
    if pwm_r == 0
        motor_r.stop(true)
    else
        motor_r.power = pwm_r
    end
    RTOS.delay(wait)
```

バランサの返り値が2つ

pwm\_l, pwm\_r = Balancer.control(  
forward, turn,  
@gyro.rate, GYRO\_OFFSET,  
motor\_l.count, motor\_r.count,  
Battery.mV, BACKLASHHALF)

C言語で実装されたバランサを呼び出す  
**ETロボコン2018で採用されたデカタイヤ**  
を用いる場合は**BACKLASHHALF=4**  
が適当であろう。ただし、走行体毎に調整  
する必要もあるだろう。  
ノーマルなタイヤの場合は0にする。

ループ内を4ミリ秒周期で実行  
wait= 3(ミリ秒) でdelayする  
(1ミリ秒+ 3ミリ秒=4ミリ秒)  
**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（Parameter.gyro\_sensor\_port）



# gyro\_sample.rb

```
include EV3RT_TECS
```

```
begin
```

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

```
    gyro_port = Parameter.gyro_sensor_port  
    gyro_sensor = GyroSensor.new(gyro_port)
```

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

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

```
    while true
```

```
        gyro = gyro_sensor.angle  
        LCD.draw("gyro = #{gyro} ", 0, 4)  
        RTOS.delay(10)
```

```
    end
```

```
    rescue => e
```

```
        LCD.error_puts e
```

```
    end
```

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\_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
- 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"  
    while true  
        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)  
    end  
  
    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
```

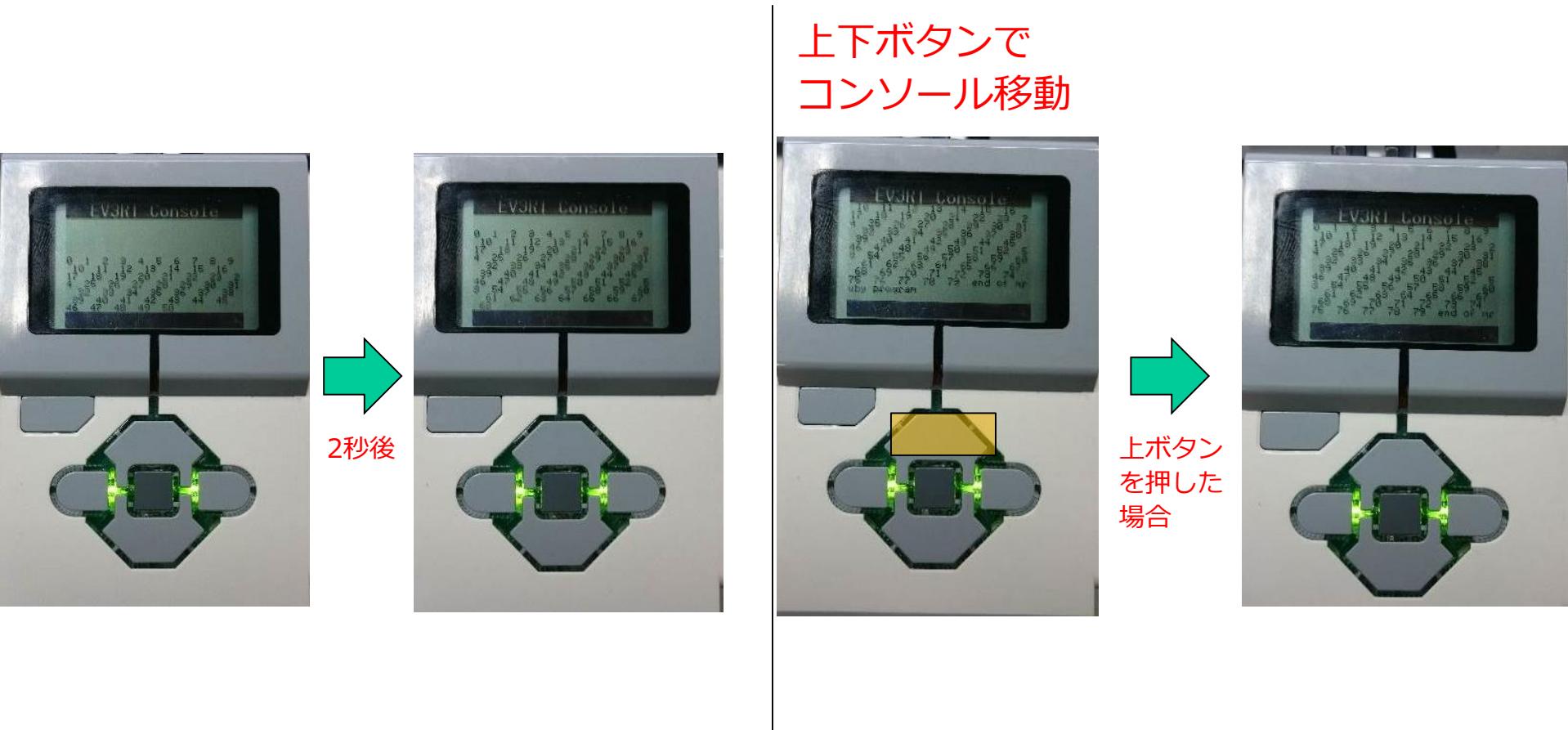
```
    LCD.error_puts e
```

```
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)
```

```
    while true
```

```
        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?
```

```
    end
```

```
    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（Parameter.left\_motor）
- 右モータ（Parameter.right\_motor）
- 超音波センサ（Parameter.ultrasonic）



# motor\_sample.rb : 前半

```
include EV3RT_TECS
```

```
begin
```

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

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

```
    # Sensors and Actuators
```

```
    left_port = Parameter.left_motor_port
```

```
    right_port = Parameter.right_motor_port
```

```
    ultrasonic_port = Parameter.sonar_sensor_port
```

```
    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

# motor\_sample.rb:後半

```
while true
    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
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



## motor\_sample2.rb

```
count = motor.count
LCD.puts "motor count = #{count}"    target = 20
if count > 0
    angle = -target - count
    モータを反対向きに動かす
else if count <= 0
    angle = target - count
end
motor.rotate( angle, 10, false)
```

```
count = motor.count
LCD.puts "motor angle=#{angle} count = #{count}"
puts "motor angle=#{angle} count = #{count}"
```

# 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

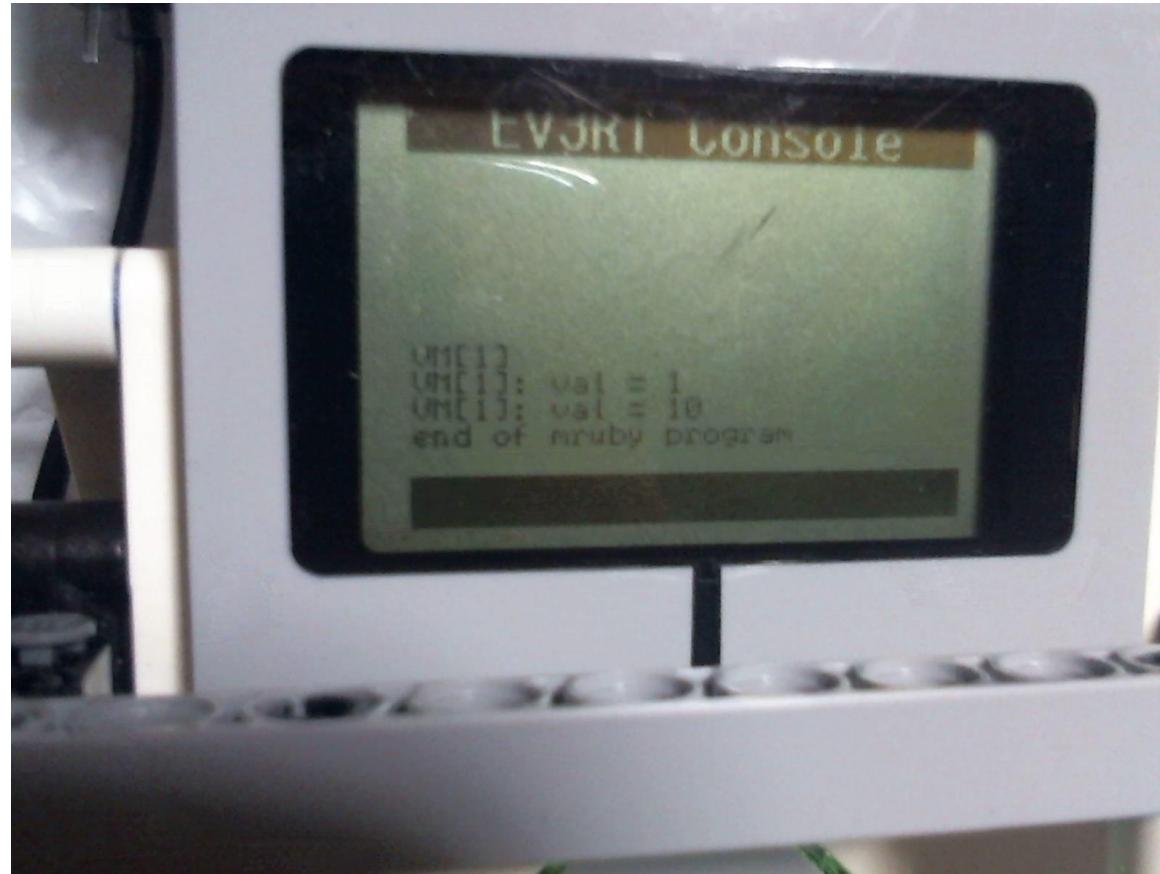
例では、 1 ~ 1 0 の合計を求める

計算結果の表示

測定結果の表示

# sharedmemory\_sample

- 共有メモリへの値の書き込みと読み込みのサンプルプログラム



# sharedmemory\_sample.rb

```
include EV3RT_TECS
```

```
begin
```

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

```
    LCD.puts "VM[1]"
```

```
    BT_CMD = 1 # 共有メモリの index = 1 を bt_cmd として使用する
```

```
    sm = SharedMemory.new()
```

```
    sm.putVal( BT_CMD, 1 )
```

SharedMemoryはインスタンス化 (new) する

共有メモリのindex=1に1を書き込み

```
    val = sm.getVal( BT_CMD )
```

共有メモリのindex=1から読み込み

```
    LCD.puts "VM[1]: val = #{val}"
```

```
    RTOS.delay(1000)
```

```
    sm.putVal( BT_CMD, 10 )
```

共有メモリのindex=1に10を書き込み

```
    val = sm.getVal( BT_CMD )
```

共有メモリのindex=1から読み込み

```
    LCD.puts "VM[1]: val = #{val}"
```

共有メモリのindex=1から読み込み

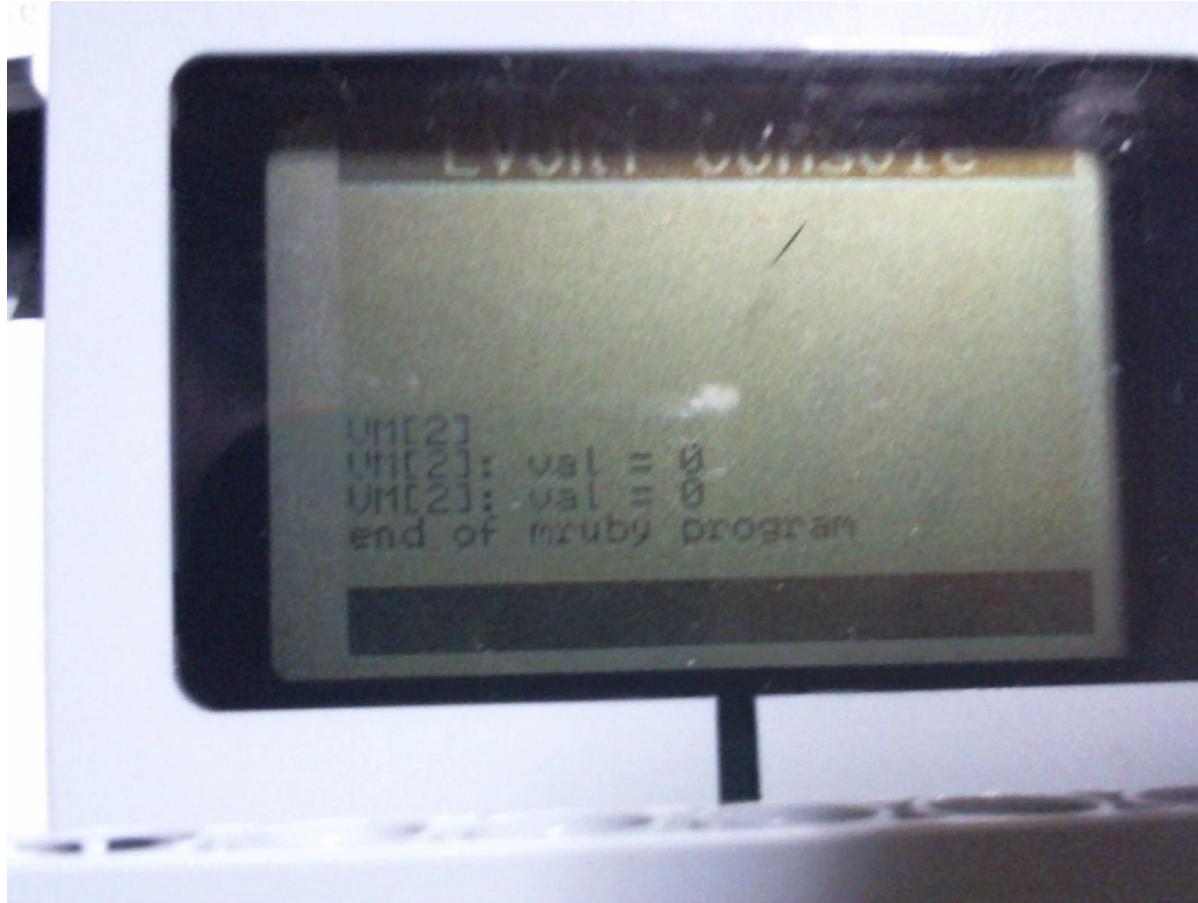
```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

## sharedmemory\_sample2

- 共有メモリから値を読み込むサンプルプログラム



# sharedmemory\_sample2.rb

```
include EV3RT_TECS
```

```
begin
```

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

```
    LCD.puts "VM[2]"
```

```
    RTOS.delay(1000)
```

```
    BT_CMD = 1 # 共有メモリの index = 1 を bt_cmd として使用する
```

```
    sm = SharedMemory.new()
```

```
    val = sm.getVal( BT_CMD )
```

```
    LCD.puts "VM[2]: val = #{val}"
```

```
    RTOS.delay(1000)
```

```
    val = sm.getVal( BT_CMD )
```

```
    LCD.puts "VM[2]: val = #{val}"
```

```
rescue => e
```

```
    LCD.error_puts e
```

```
end
```

SharedMemoryはインスタンス化 (new) する

共有メモリのindex=1から読み込み

共有メモリのindex=1から読み込み

# 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 sample2", 0,1)

  Speaker.volume= 1
  while true
    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)
  end
rescue => e
  LCD.error_puts e
end
```

押されたボタンに対応するトーン  
と長さで音を鳴らす



# 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 (Parameter.touch\_sensor\_port)

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 = Parameter.touch_sensor_port

    LCD.draw("touch:#{touch_port}", 0, 2)
    $touch_sensor = TouchSensor.new(touch_port)

    count = 0
    while true
        LCD.draw("touch count:#{count}", 0, 5)
        while !$touch_sensor.pressed? do end
        while $touch_sensor.pressed? do end
        count = count + 1
    end
rescue => e
    LCD.error_puts e
end
```

TouchSensorはポートを指定して  
インスタンス化 (new) する  
※シリアルを利用する場合は、:port\_1 を利用しない

# 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 (Parameter.sonar\_sensor\_port)

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 = Parameter.sonar_sensor_port

    LCD.draw("ultrasonic:#{$ultrasonic_port}", 0, 2)
    ultrasonic_sensor = UltrasonicSensor.new(ultrasonic_port)
    LCD.draw("listen = #{$ultrasonic_sensor.listen} ", 0, 3)

    while true
        distance = ultrasonic_sensor.distance
        LCD.draw("distance = #{distance} ", 0, 4)
    end
    rescue => e
        LCD.error_puts e
    end
```

UltrasonicSensorはポートを指定して  
インスタンス化 (new) する  
※シリアルを利用する場合は、:port\_1 を利用しない

# parameter.rb

- parameter.rbはサンプルプログラムから利用されるライブラリであり、ev3way用、またはetrobo用のポート番号の割り当てを記録している。
- Makefileでの設定にしたがい、mrbcによるMrubyスクリプトのコンパイル時に、Parameterクラスのsetメソッドを記述したMrubyスクリプト（ポート番号の割り当てを記録する）がコンパイル対象に含まれた状態で、コンパイルされる。
- サンプルプログラムは、Parameterクラスが記録しているポート番号を取得して、設定する。

# parameter\_ev3way.rb

```
module EV3RT_TECS
  Parameter.set(:port_1, :port_2, :port_3, :port_4,
                :port_a, :port_b, :port_c)
end
```



EV3WAY用のポート番号の割り当てを記録する

# parameter\_etrobo.rb

```
module EV3RT_TECS
  Parameter.set(:port_1, :port_3, :port_4, :port_2,
                :port_a, :port_b, :port_c)
end
```



ETロボコン用のポート番号の割り当てを記録する

# 改版履歴

- 2015年6月8日 alpha 1.0.0
  - 安積卓也 (大阪大学)
  - 長谷川涼 (大阪大学)
- 2015年7月9日 alpha 1.0.2
  - 安積卓也 (大阪大学)
  - 長谷川涼 (大阪大学)
- 2016年8月30日 beta 1.0.0
  - 安積卓也 (大阪大学)
  - 長谷川涼 (大阪大学)
- 2018年5月16日 beta 2.1.0
  - 安積卓也 (大阪大学)
  - 長谷川涼 (大阪大学)
  - 小南靖雄 (TOPPERS個人会員)