Manuale d’utilizzo

simpleWaitLib & simplemotorLib

Luca Di Bello, Fadil smajilbasic

2019

|  |  |  |
| --- | --- | --- |
| **Autore** | **Data** | **Versione** |
| Luca Di Bello e Fadil Smajilbasic | 08.02.19 | 1.0 |

# Indice

[Indice 2](#_Toc522614)

[SimpleMotorLib 3](#_Toc522615)

[Sommario della classe 3](#_Toc522616)

[Descrizione della classe 3](#_Toc522617)

[goMotorStandard(short port, byte power = 127) 3](#_Toc522618)

[goMotorSeconds(short port, float seconds , byte power = 127) 3](#_Toc522619)

[goMotorDegrees(short port, float degrees, byte power = 127) 4](#_Toc522620)

[goMotorRotations(short port, int rotations, byte power = 127) 4](#_Toc522621)

[stopMotor(short port) 4](#_Toc522622)

[SimpleWaitLib 5](#_Toc522623)

[Sommario della classe 5](#_Toc522624)

[Descrizione della classe 5](#_Toc522625)

[wait(float seconds) 5](#_Toc522626)

[waitNxtButtons(short button) 5](#_Toc522627)

[waitDistance(short port,int distance,short mode) 5](#_Toc522628)

[waitMicrophone(short port,int db,short mode) 6](#_Toc522629)

[waitLight(short port, int light\_value, short mode) 6](#_Toc522630)

[waitTouch(short port, short mode) 7](#_Toc522631)

[Contatti 7](#_Toc522632)

# SimpleMotorLib

## Sommario della classe

|  |  |
| --- | --- |
| goMotorStandard(short port, byte power = 127) | Permette di avviare un motore con una determinata velocità |
| goMotorSeconds(short port, float seconds , byte power = 127) | Permette di avviare un motore con una determinata velocità per un lasso di tempo definito in secondi |
| goMotorDegrees(short port, float degrees, byte power = 127) | Permette di avviare un motore con una determinata velocità facendolo girare per un determinato numero di gradi |
| goMotorRotations(short port, int rotations, byte power = 127) | Permette di avviare un motore con una determinata velocità facendolo girare per un determinato numero di rotazioni |
| stopMotor(short port) | Permette di fermare un motore istantaneamente |

## Descrizione della classe

|  |
| --- |
| goMotorStandard(short port, byte power = 127) |
| Questo metodo permette di avviare un motore con una determinata velocità.  Parametri:   * *port*   + Riferimento all’oggetto che descrive il motore * *power*   + Valore intero che descrive la velocità da dare al motore, il range dei valori accettati è da **-127** a **+127**. Se viene passato un valore negativo il motore girerà al contrario (marcia indietro). Di default la velocità del motore è di 127 (velocità massima in avanti) |

|  |
| --- |
| goMotorSeconds(short port, float seconds , byte power = 127) |
| Questo metodo permette di avviare un motore con una determinata velocità per un lasso di tempo definito in secondi.  Parametri:   * *port*   + Riferimento all’oggetto che descrive il motore * *seconds*   + Valore decimale che descrive il lasso di tempo nel quale il motore girerà. Non sono ammessi valori negativi. * *power*   + Valore intero che descrive la velocità da dare al motore, il range dei valori accettati è da **-127** a **+127**. Se viene passato un valore negativo il motore girerà al contrario (marcia indietro). Di default la velocità del motore è di 127 (velocità massima in avanti) |

|  |
| --- |
| goMotorDegrees(short port, float degrees, byte power = 127) |
| Questo metodo permette di avviare un motore con una determinata velocità facendolo girare per un determinato numero di gradi.  Parametri:   * *port*   + Riferimento all’oggetto che descrive il motore * *degrees*   + Valore decimale che descrive il numero di gradi che il motore girerà * *power*   + Valore intero che descrive la velocità da dare al motore, il range dei valori accettati è da **-127** a **+127**. Se viene passato un valore negativo il motore girerà al contrario (marcia indietro). Di default la velocità del motore è di 127 (velocità massima in avanti) |

|  |
| --- |
| goMotorRotations(short port, int rotations, byte power = 127) |
| Questo metodo permette di avviare un motore con una determinata velocità facendolo girare per un determinato numero di rotazioni.  Parametri:   * *port*   + Riferimento all’oggetto che descrive il motore * *rotations*   + Valore intero che descrive il numero di rotazioni che effettuerà il motore * *power*   + Valore intero che descrive la velocità da dare al motore, il range dei valori accettati è da **-127** a **+127**. Se viene passato un valore negativo il motore girerà al contrario (marcia indietro). Di default la velocità del motore è di 127 (velocità massima in avanti) |

|  |
| --- |
| stopMotor(short port) |
| Permette di fermare un motore istantaneamente impostandogli la sua velocità a **0**.  Parametri:   * *port*   + Riferimento all’oggetto che descrive il motore |

# SimpleWaitLib

## Sommario della classe

|  |  |
| --- | --- |
| wait(float seconds) | Permette di fermare l’esecuzione del programma per un lasso di tempo specificato in secondi |
| waitNxtButtons(short button) | Permette di fermare l’esecuzione del programma finché uno dei pulsanti del blocchetto viene premuto |
| waitDistance(short port,int distance,short mode) | Permette di fermare l’esecuzione del programma finché il sensore ultrasuoni (di distanza) non rileva una determinata distanza |
| waitMicrophone(short port,int db,short mode) | Permette di fermare l’esecuzione del programma finché il microfono non rileva un determinato valore di Decibel |
| waitLight(short port, int light\_value, short mode) | Permette di fermare l’esecuzione del programma finché il sensore di luce non rileva una determinata percentuale di luce riflessa |
| waitTouch(short port, short mode) | Permette di fermare l’esecuzione del programma finché il sensore touch (pulsante) non viene premuto, rilasciato oppure cliccato (ovvero premuto e rilasciato). |

## Descrizione della classe

|  |
| --- |
| wait(float seconds) |
| Questo metodo permette di fermare l’esecuzione del programma per un lasso di tempo specificato in secondi.  Parametri:   * *seconds*   + Valore decimale che descrive la durata del wait in secondi |

|  |
| --- |
| waitNxtButtons(short button) |
| Questo metodo permette di fermare l’esecuzione del programma finché uno dei pulsanti del blocchetto viene premuto.  Parametri:   * *button*   + Specifica il pulsante da pigiare, questi sono i codici identificativi per i bottoni:     - 0: Rettangolo grigio     - 1: Freccia destra     - 2: Freccia sinistra     - 3: Quadrato arancione |

|  |
| --- |
| waitDistance(short port,int distance,short mode) |
| Questo metodo permette di fermare l’esecuzione del programma finché il sensore ultrasuoni (di distanza) non rileva una determinata distanza (maggiore o minore di *distance*).  Parametri:   * *port*   + Riferimento all’oggetto che descrive il sensore ultrasuoni (di distanza) * *distance*   + Valore intero che descrive la distanza target sulla quale si baserà il metodo, il valore massimo accettato è **255** * *mode*   + Valore intero che descrive la modalità utilizzata dal programma. Il parametro accetta i valori 0 ed 1:     - Con il valore 0: Il valore letto dal sensore deve essere minore di quello passato come target (*distance*)     - Con il valore 1: Il valore letto dal sensore deve essere maggiore di quello passato come target (*distance*) |

|  |
| --- |
| waitMicrophone(short port,int db,short mode) |
| Questo metodo permette di fermare l’esecuzione del programma finché il microfono non rileva un determinato valore di Decibel (maggiore o minore di *db*).  Parametri:   * *port*   + Riferimento all’oggetto che descrive il microfono * *db*   + Valore intero che descrive il valore di Decibel desiderato (target), non sono ammessi valori negativi * *mode*   + Valore intero che descrive la modalità utilizzata dal programma. Il parametro accetta i valori 0 ed 1:     - Con il valore 0: Il valore letto dal sensore deve essere minore di quello passato come target (*db*)     - Con il valore 1: Il valore letto dal sensore deve essere maggiore di quello passato come target (*db*) |

|  |
| --- |
| waitLight(short port, int light\_value, short mode) |
| Questo metodo permette di fermare l’esecuzione del programma finché il sensore di luce non rileva una determinata percentuale di luce riflessa (maggiore o minore di *light\_value*).  Parametri:   * *port*   + Riferimento all’oggetto che descrive il sensore di luce * *light\_value*   + Valore intero che descrive la percentuale di luce riflessa desiderata (target), non sono ammessi valori negativi * *mode*   + Valore intero che descrive la modalità utilizzata dal programma. Il parametro accetta i valori 0 ed 1:     - Con il valore 0: Il valore letto dal sensore deve essere minore di quello passato come target (light\_value)     - Con il valore 1: Il valore letto dal sensore deve essere maggiore di quello passato come target (*light\_value*) |

|  |
| --- |
| waitTouch(short port, short mode) |
| Questo metodo permette di fermare l’esecuzione del programma finché il sensore touch (pulsante) non viene premuto (*pressed*), rilasciato (*released*) oppure cliccato (ovvero premuto e rilasciato, *clicked*).  Parametri:   * *port*   + Riferimento all’oggetto che descrive il sensore touch (pulsante) * *mode*   + Valore intero che descrive la modalità utilizzata dal programma. Il parametro accetta i valori 0,1 e 2:     - Con il valore 0: Il pulsante deve venire premuto (*pressed*)     - Con il valore 1: il pulsante deve venire rilasciato (*released*)     - Con il valore 2: il pulsante deve venire cliccato (premuto e rilasciato, *clicked*) |

# Contatti

Se il manuale d’utilizzo risulta incompleto oppure di difficile comprensione potete contattare **Luca Di Bello** ([luca.dibello@samtrevano.ch](mailto:luca.dibello@samtrevano.ch)) oppure **Fadil Smajilbasic** ([fadil.smajilbasic@samtrevano.ch](mailto:fadil.smajilbasic@samtrevano.ch)).