EBG Roboter Library

Library Funktionen

Allgemeiner Teil

roboter_init() *Init*

Initialisiert alle Module der Library: WICHTIG: Diese Funktion muss aufgerufen werden, bevor Libraryfunktionen genutzt werden können!

pinMode(pin_num, mode) *Input/Output*

Schaltet einen Pin als Ausgang oder Eingang: pin_num ist die Nummer auf dem Arduino Board. *mode* ist entweder OUTPUT oder INPUT. Diese Funktion ist von der Syntax identisch zum Arduino. Diese Funktion kann nur mit fixen Werten genutzt werden. Es ist nicht möglich Variablen zu verwenden. Intern wird ein Makro verwendet, das zur Kompilierzeit aufgelöst wird.

digitalWrite(pin_num, value) *Input/Output*

Schaltet einen Ausgang auf LOW oder HIGH: pin_num ist die Nummer auf dem Arduino Board. *value* ist entweder LOW oder HIGH. (Zusatz: wenn der Pin als Eingang geschalten ist, kann man mit einem HIGH einen PullUp aktivieren) Diese Funktion ist von der Syntax identisch zum Arduino. Diese Funktion kann nur mit fixen Werten genutzt werden. Es ist nicht möglich Variablen zu verwenden. Intern wird ein Makro verwendet, das zur Kompilierzeit aufgelöst wird.

digitalRead(pin_num) *Input/Output*

List den digitalen Wert am Eingang aus: pin_num ist die Nummer auf dem Arduino Board. *return:* Der Rückgabewert ist der Zustand des Eingangs. Diese Funktion ist von der Syntax identisch zum Arduino. Diese Funktion kann nur mit fixen Werten genutzt werden. Es ist nicht möglich Variablen zu verwenden. Intern wird ein Makro verwendet, das zur Kompilierzeit aufgelöst wird.

analogRead(pin_num) *Input/Output*

List den analogen Wert an einem analogen Eingang aus: pin_num ist die Nummer auf dem Arduino Board. *return:* Der Rückgabewert ist der Analogwert. Die Zahl kann sich von 0 bis 1024 erstrecken, was einer Spannung von 0 Volt bis 5 Volt am Eingang entspricht.
Diese Funktion ist von der Syntax identisch zum Arduino.

delay_ms(time) *Time*

Wartet x Millisekunden: *time* gibt die Anzahl der Millisekunden an, die gewartet werden. Diese Funktion ist von der Syntax identisch zum Arduino.

delay_us(time) *Time*

Wartet x Mikrosekunden: *time* gibt die Anzahl der Mikrosekunden an, die gewartet werden. Diese Funktion ist von der Syntax identisch zum Arduino.

serial_print(string) *Serial*

Gibt einen Text auf der seriellen Konsole aus: Bei dieser Funktion kann die printf-Syntax angewandt werden. Somit können beliebige Variablen direkt in den Funktionsaufruf integriert werden. Beispiel: `funcname("Das ist die Zahl i:  %d", i)`

serial_println(string) *Serial*

Gibt einen Text auf der seriellen Konsole aus und fügt eine neue Zeile an: Bei dieser Funktion kann die printf-Syntax angewandt werden. Somit können beliebige Variablen direkt in den Funktionsaufruf integriert werden. Beispiel: `funcname("Das ist die Zahl i:  %d", i)`

serial_puts(string) *Serial*

Gibt einen Text auf der seriellen Konsole aus: Dies ist eine Low-Level Funktion und sollte nur mit Erfahrung genutzt werden.

serial_putc(char) *Serial*

Gibt einen Character auf der seriellen Konsole aus: Dies ist eine Low-Level Funktion und sollte nur mit Erfahrung genutzt werden.

serial_available() *Serial*

Prüft ob ein neuer Char auf der seriellen Konsole verfügbar ist: *return:* Gibt eine Zahl größer Null zurück, fall Daten verfügbar sind.

serial_getchar() *Serial*

Liest einen Char von der seriellen Konsole ein: *return:* Gibt entweder den Char zurück oder die Konstante UART_NO_DATA. Die Funktion ist Non-Blocking.

Funktionsspezifischer Teil

lcd_print(string) *LCD*

Schreibt einen String auf das LCD Display: Bei dieser Funktion kann die printf-Syntax angewandt werden. Somit können beliebige Variablen direkt in den Funktionsaufruf integriert werden. Beispiel: `funcname("Das ist die Zahl i:  %d", i)`

lcd_clear() *LCD*

Löscht den Inhalt des LCDs und setzt den Cursor auf die Ausgangsposition:

lcd_cursor_set(spalte, zeile) *LCD*

Setzt den Cursor auf eine beliebige Position: *spalte* gibt dabei die horizontale Position an. Falls man in eine neue Zeile springen möchte, setzt man *spalte* gleich null.
zeile gibt die vertikale Position an. Dabei beginnt man bei 0 für die erste Zeile.

led(leds, state) *LED*

Schaltet die LEDs des Roboters ein oder aus: *leds* ist dabei eine oder mehrere der LED-Konstanten. Beispiel `led(LED_SCHEINWERFER | LED_RUECKLICHT, ON);` *state* ist entweder LED_ON oder LED_OFF. Es können folgende LED-Konstanten verwendet werden:

- LED_SCHEINWERFER
- LED_SCHEINWERFER_R
- LED_SCHEINWERFER_L
- LED_RUECKLICHT
- LED_RUECKLICHT_R
- LED_RUECKLICHT_L
- LED_BLINKER_RECHTS
- LED_BLINKER_LINKS
- LED_BLINKER_VORN_R
- LED_BLINKER_VORN_L
- LED_BLINKER_HINTEN_R
- LED_BLINKER_HINTEN_L

drive(motor, direction, speed) *MOTOR*

Setzt die Richtung und Geschwindigkeit für einen oder mehrere Motoren: *motor* wählt die Motoren aus. Es können die Konstanten MOTOR_LEFT und MOTOR_RIGHT gewählt werden. *direction* ist die Drehrichtung der Motoren. Folgende Konstanten sind möglich FORWARD, BACKWARD, BRAKE und RELEASE. *speed* ist eine Zahl von 0 bis 255. Es erlaubt die Motoren mit mehr oder weniger viel Kraft anzusteuern.

ir_read() *IR-Sensor*

Liest alle Infrarotsensoren aus: Diese Funktion muss vor `ir_get(sensor_id)` ausgeführt werden.

ir_get(sensor_id) *IR-Sensor*

Gibt die ausgelesen Werte für einen bestimmten Sensor zurück: *sensor_id* kann eine der folgenden Konstanten sein: IR_LINKS, IR_MITTE und IR_RECHTS. *return:* Gibt den Helligkeitswert zum Zeitpunkt des Aufrufs von `ir_read()` an. Mögliche Werte sind im Bereich 0 bis 255. Bevor diese Funktion genutzt werden kann muss `ir_read()` ausgeführt werden.

us_read() *US-Sensor*

Liest alle Ultraschallsensoren aus: Diese Funktion muss vor `us_get(sensor_id)` ausgeführt werden.

us_get(sensor_id) *US-Sensor*

Gibt die ausgelesen Werte für einen bestimmten Sensor zurück: *sensor_id* kann eine der folgenden Konstanten sein: US_LINKS, US_MITTE und US_RECHTS. *return:* Gibt die Entfernung des Objektes vor dem Ultraschallsensor in Zentimeter zum Zeitpunkt des Aufrufs von `us_read()` an. Bevor diese Funktion genutzt werden kann muss `us_read()` ausgeführt werden.

taster_read(taster_id) *Taster*

Liest den aktuellen Status eines Tasters aus: *taster_id* kann eine der folgenden Konstanten sein: TASTER_LINKS, TASTER_MITTE und TASTER_RECHTS. *return:* Gibt eine 1 zurück, wenn der Taster gedrückt ist.

voltage_get() *Voltage*

Misst die Batterie-Spannung des Roboters in Volt: *return:* gibt die Spannung als float zurück.