An autonomous crawling robot

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

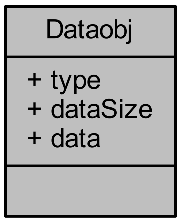
Data Structure Documentation

3.1 Dataobj Struct Reference

This struct contains the parameters of the data object that can be used to transmit data.

```
#include <protocol.h>
```

Collaboration diagram for Dataobj:



Data Fields

- unsigned char **type**
- unsigned char **dataSize**
- int **data**

3.1.1 Detailed Description

This struct contains the parameters of the data object that can be used to transmit data.

Parameters

type	this parameter contains the representation of the data
DataSize	contains the amount of bytes of the data

The documentation for this struct was generated from the following file:

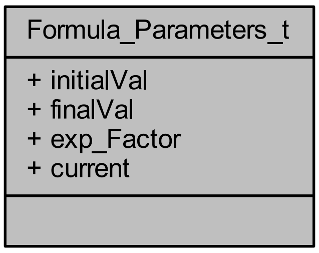
- lib/protocol/protocol.h

3.2 Formula_Parameters_t Struct Reference

This struct contains the parameters of the Q-learning formula. Some other functions are specificly designed to change those parameters by using an exponential function.

```
#include <q_learning.h>
```

Collaboration diagram for Formula_Parameters_t:



Data Fields

- unsigned char **initialVal**
- unsigned char **finalVal**
- float **exp_Factor**
- float **current**

3.2.1 Detailed Description

This struct contains the parameters of the Q-learning formula. Some other functions are specificly designed to change those parameters by using an exponential function.

The documentation for this struct was generated from the following file:

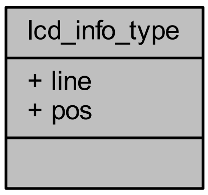
- lib/q_learning/q_learning.h

3.3 lcd_info_type Struct Reference

Stores the current line and position of the cursor.

```
#include <DwenguinoLCD.h>
```

Collaboration diagram for lcd_info_type:



Data Fields

- unsigned char line
- unsigned char pos

3.3.1 Detailed Description

Stores the current line and position of the cursor.

3.3.2 Field Documentation

3.3.2.1 line

```
unsigned char line
```

line number lcd_info_type::a.

3.3.2.2 pos

```
unsigned char pos
```

position in the line lcd_info_type::b.

The documentation for this struct was generated from the following file:

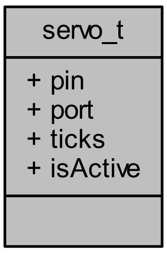
- lib/dwenguinoLCD/DwenguinoLCD.h

3.4 servo_t Struct Reference

This struct contains the parameters of a servo. Some other functions are specificly designed to change those parameters.

```
#include <servo.h>
```

Collaboration diagram for servo_t:



Data Fields

- unsigned char pin
  pointer to the pin
- volatile unsigned char ∗ port
  pointer to the port
- unsigned int ticks
  contains amount of prescaled ticks
- unsigned char isActive
  0 = inactive

3.4.1 Detailed Description

This struct contains the parameters of a servo. Some other functions are specificly designed to change those parameters.

The documentation for this struct was generated from the following file:

- lib/servo/servo.h

Chapter 4

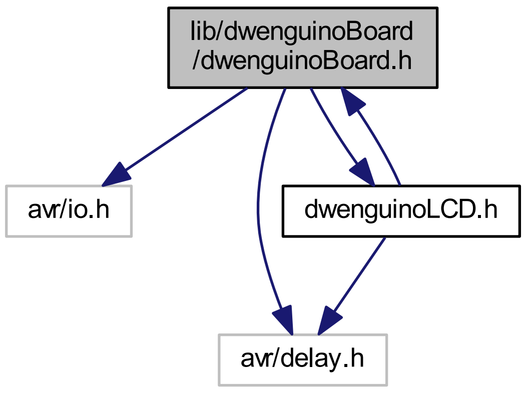
File Documentation

4.1 lib/dwenguinoBoard/dwenguinoBoard.h File Reference

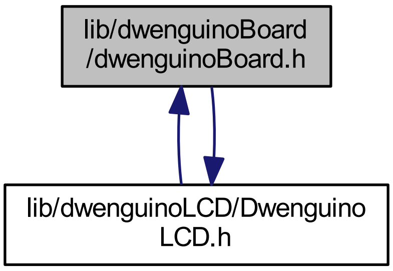
This library contains pin assingments and basic macros for Dwenguino boards.

```
#include <avr/io.h>
#include <avr/delay.h>
#include "dwenguinoLCD.h"
```

Include dependency graph for dwenguinoBoard.h:



This graph shows which files directly or indirectly include this file:


lib/dwenguinoBoard /dwenguinoBoard.h
lib/dwenguinoLCD/Dwenguino LCD.h

Macros

- #define **TRUE** 1
- #define **FALSE** 0
- #define **HIGH** 1
- #define **LOW** 0
- #define **PORT_HIGH** 0xFF
- #define **PORT_LOW** 0x00
- #define **INPUT** 0
- #define **OUTPUT** 1
- #define **SET_PIN_HIGH**(PORT, PIN) PORT |= (1 << PIN)
- #define **SET_PIN_LOW**(PORT, PIN) PORT &= ∼(1 << PIN)
- #define **SET_BIT_HIGH**(REG, BIT) REG |= (1 << BIT)
- #define **SET_BIT_LOW**(REG, BIT) REG &= ∼(1 << BIT)
- #define **BYTE** unsigned char
- #define **LEDS_DIR** DDRA
- #define **LEDS** PORTA
- #define **LED_ON**(LED) SET_PIN_HIGH(PORTA, LED)
- #define **LED_OFF**(LED) (SET_PIN_LOW(PORTA, LED)
- #define **SW_C_HIGH** SET_PIN_HIGH(PORTC, 6)
- #define **SW_C_LOW** SET_PIN_LOW(PORTC, 6)
- #define **SW_C_IN** SET_PIN_LOW(DDRC, 6)
- #define **SW_C_OUT** SET_PIN_HIGH(DDRC, 6)
- #define **SW_W_HIGH** SET_PIN_HIGH(PORTE, 4)
- #define **SW_W_LOW** SET_PIN_LOW(PORTE, 4)
- #define **SW_W_IN** SET_PIN_LOW(DDRE, 4)
- #define **SW_W_OUT** SET_PIN_HIGH(DDRE, 4)
- #define **SW_S_HIGH** SET_PIN_HIGH(PORTE, 5)
- #define **SW_S_LOW** SET_PIN_LOW(PORTE, 5)
- #define **SW_S_IN** SET_PIN_LOW(DDRE, 5)
- #define **SW_S_OUT** SET_PIN_HIGH(DDRE, 5)
- #define **SW_E_HIGH** SET_PIN_HIGH(PORTE, 6)
- #define **SW_E_LOW** SET_PIN_LOW(PORTE, 6)
- #define **SW_E_IN** SET_PIN_LOW(DDRE, 6)
- #define **SW_E_OUT** SET_PIN_HIGH(DDRE, 6)
- #define **SW_N_HIGH** SET_PIN_HIGH(PORTE, 7)

- #define **SW_N_LOW** SET_PIN_LOW(PORTE, 7)
- #define **SW_N_IN** SET_PIN_LOW(DDRE, 6)
- #define **SW_N_OUT** SET_PIN_HIGH(DDRE, 6)
- #define **LCD_DATA** PORTA
- #define **LCD_DATA_DIR** DDRA
- #define **LCD_BACKLIGHT_ON** SET_PIN_HIGH(PORTE, 3)
- #define **LCD_BACKLIGHT_OFF** SET_PIN_LOW(PORTE, 3)
- #define **LCD_BACKLIGHT_OUT** SET_PIN_HIGH(DDRE, 3)
- #define **LCD_BACKLIGHT_IN** SET_PIN_LOW(DDRE, 3)
- #define **LCD_RW_HIGH** SET_PIN_HIGH(PORTE, 1)
- #define **LCD_RW_LOW** SET_PIN_LOW(PORTE, 1)
- #define **LCD_RW_OUT** SET_PIN_HIGH(DDRE, 1)
- #define **LCD_RS_HIGH** SET_PIN_HIGH(PORTE, 0)
- #define **LCD_RS_LOW** SET_PIN_LOW(PORTE, 0)
- #define **LCD_RS_OUT** SET_PIN_HIGH(DDRE, 0)
- #define **LCD_EN_HIGH** SET_PIN_HIGH(PORTE, 2)
- #define **LCD_EN_LOW** SET_PIN_LOW(PORTE, 2)
- #define **LCD_EN_OUT** SET_PIN_HIGH(DDRE, 2)
- #define **SERVO1** PORTC0
- #define **SERVO2** PORTC1
- #define **MOTOR1_0_HIGH** SET_PIN_HIGH(PORTC, 3)
- #define **MOTOR1_0_LOW** SET_PIN_LOW(PORTC, 3)
- #define **MOTOR1_1_HIGH** SET_PIN_HIGH(PORTC, 4)
- #define **MOTOR1_1_LOW** SET_PIN_LOW(PORTC, 4)
- #define **MOTOR2_0_HIGH** SET_PIN_HIGH(PORTC, 2)
- #define **MOTOR2_0_LOW** SET_PIN_LOW(PORTC, 2)
- #define **MOTOR2_1_HIGH** SET_PIN_HIGH(PORTC, 5)
- #define **MOTOR2_1_LOW** SET_PIN_LOW(PORTC, 5)

Functions

- void initBoard (void)

4.1.1 Detailed Description

This library contains pin assingments and basic macros for Dwenguino boards.

Author

Tom Neutens

Date

Jan 19, 2016

See also

http://www.dwengo.org/tutorials

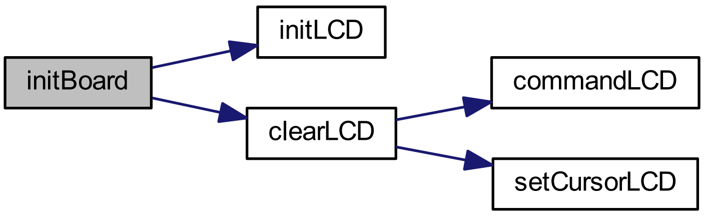
4.1.2 Function Documentation

4.1.2.1 initBoard()

```
void initBoard (
            void )
```

dwenguinoBoard.c

Created on: Jan 19, 2016 Author: Tom Here is the call graph for this function:

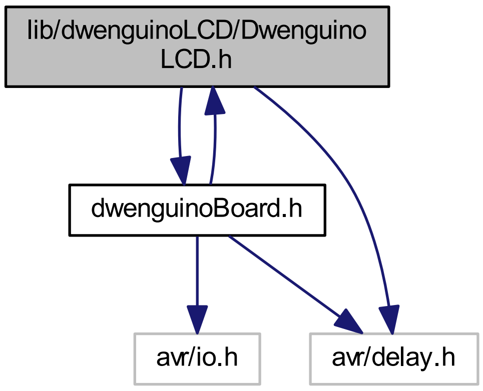


4.2 lib/dwenguinoLCD/DwenguinoLCD.h File Reference

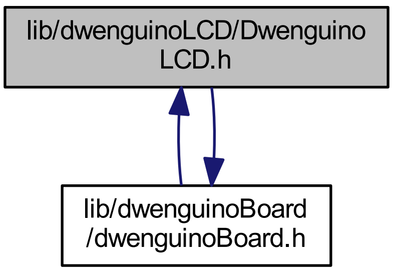
This library contains function declarations which allow you to communicate with the lcd screen on the Dwenguino board.

```
#include "dwenguinoBoard.h"
#include <avr/delay.h>
```

Include dependency graph for DwenguinoLCD.h:



This graph shows which files directly or indirectly include this file:


lib/dwenguinoLCD/Dwenguino LCD.h
lib/dwenguinoBoard /dwenguinoBoard.h

Data Structures

- struct lcd_info_type

  Stores the current line and position of the cursor.

Macros

- #define **LCD_WIDTH** 16
- #define **LCD_HEIGHT** 2
- #define **LCD_LASTLINE** (LCD_HEIGHT - 1)
- #define **LCD_LASTPOS** (LCD_WIDTH - 1)
- #define **backlightOn**() (LCD_BACKLIGHT_ON)
- #define **backlightOff**() (LCD_BACKLIGHT_OFF)
- #define **appendStringToLCD**(message) appendStringToLCD_((const char∗)(message));

Functions

- void initLCD (void)

  initializes the LCD screen This function sets up the lcd for displaying the data we will send
- void clearLCD (void)

  clears the LCD screen This function removes all the content from the LCD screen
- void commandLCD (const BYTE c)

  sends a command to the LCD This function sends a low level command to the LCD. The command is represented as a byte and is transferred to the LCD screen through the PORTA register.
- void setCursorLCD (BYTE l, BYTE p)

  Move the cursor on the screen Sets the cursor to a specified line on a certain position.
- void appendCharToLCD (const char c)

  Append a character to the LCD Adds a character at the current cursor positon then moves the cursor to the next position.
- void printCharToLCD (const char s, BYTE l, BYTE p)

  Print character to LCD Prints a character to a specified position.
- void appendIntToLCD (int i)

  Append an integer to the lcd screen Prints the integer at the current cursor position and moves the cursor to the position affther the printed integer.
- void printIntToLCD (int i, BYTE l, BYTE p)

  Prints an integer to the LCD screen Prints an integer to a specified line and position on the screen.

Variables

- struct lcd_info_type lcd_info

4.2.1 Detailed Description

This library contains function declarations which allow you to communicate with the lcd screen on the Dwenguino board.

See also

∗ For more information on the dwenguino board visit: `http://www.dwengo.org/tutorials`

Author

Tom Neutens

Date

11/01/2017 Before you can use the lcd you should call the initLCD() function. Afterwards you can either append or print characters or integers to the screen.

4.2.2 Function Documentation

4.2.2.1 appendCharToLCD()

```
void appendCharToLCD (
             const char c )
```

Append a character to the LCD Adds a character at the current cursor positon then moves the cursor to the next position.

Parameters

c	the character to append

4.2.2.2 appendIntToLCD()

```
void appendIntToLCD (
             int i )
```

Append an integer to the lcd screen Prints the integer at the current cursor position and moves the cursor to the position afther the printed integer.

Parameters

i	the integer to print

4.2.2.3 commandLCD()

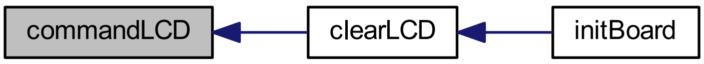
```
void commandLCD (
             const BYTE c )
```

sends a command to the LCD This function sends a low level command to the LCD. The command is represented as a byte and is transferred to the LCD screen through the PORTA register.

Parameters

c	command to be transfered to the LCD

Here is the caller graph for this function:



4.2.2.4 printCharToLCD()

```
void printCharToLCD (
             const char s,
             BYTE l,
             BYTE p )
```

Print character to LCD Prints a character to a specified position.

Parameters

s	the character to print
l	the line
p	the position in the line

4.2.2.5 printIntToLCD()

```
void printIntToLCD (
             int i,
             BYTE l,
             BYTE p )
```

Prints an integer to the LCD screen Prints an integer to a specified line and position on the screen.

Parameters

i	the integer to print
l	the line
p	the position in the line

4.2.2.6 setCursorLCD()

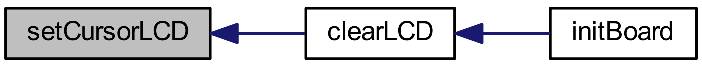
```
void setCursorLCD (
             BYTE l,
             BYTE p )
```

Move the cursor on the screen Sets the cursor to a specified line on a certain position.

Parameters

l	line number
p	position in line

Here is the caller graph for this function:



4.2.3 Variable Documentation

4.2.3.1 lcd_info

```
struct lcd_info_type lcd_info
```

dwenguinoLCD.c

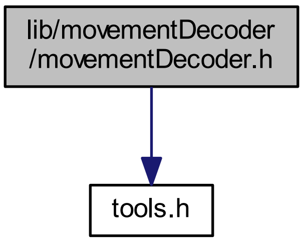
Created on: Jan 19, 2016 Author: Tom

4.3 lib/movementDecoder/movementDecoder.h File Reference

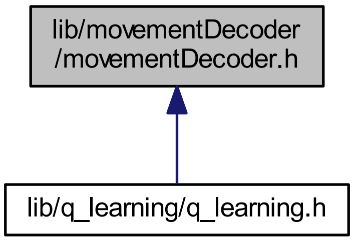
This library contains functions to determine the direction by using 2 optosensores. Before using the movement↩
Decoder's methods, call the initMovementDecoder() function.

```
#include "tools.h"
```

Include dependency graph for movementDecoder.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **DECODER1_PORT** PORTD
- #define **DECODER1_DIR** DDRD
- #define **DECODER1_DATA** PIND
- #define DECODER1 PORTD0

  connect the DO pin of optosensor 1 to this port
- #define DECODER1_INT INT0_vect

  the interrupt used by optosensor 1
- #define **DECODER2_PORT** PORTD
- #define **DECODER2_DIR** DDRD
- #define **DECODER2_DATA** PIND

- #define DECODER2 PORTD1

  connect the DO pin of optosensor 1 to this port
- #define DECODER2_INT INT1_vect

  the interrupt used by optosensor2
- #define **GRAY_TO_BIN**(GRAY) (GRAY & (1 << 1)) ? (TOGGLE_BIT(GRAY, 0)) : (0)

Functions

- void initMovementDecoder ()

  Initializes the optosensor. This function sets up the optosensores to generate interrupts and using other functions implemented in this library to retrieve the movement of the agent.
- void checkMovementDir ()

  Determines the movement direction of the agent. This function uses the current and previous state of the optosensors to determine the direction.
- int getMovementDirCount ()

  Returns the dirCount value.
- void clearDirCount ()

  Clears the dirCount value.

4.3.1 Detailed Description

This library contains functions to determine the direction by using 2 optosensores. Before using the movement↵ Decoder's methods, call the initMovementDecoder() function.

Author

Jonas Van Der Donckt

Date

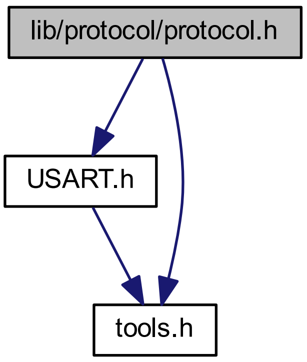
13/05/2017

4.4 lib/protocol/protocol.h File Reference

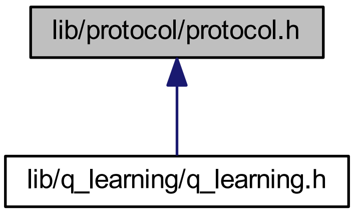
This library contains the custom protocol used to send data to a peripheral device which uses the same protocol to receive them.

```
#include "USART.h"
#include "tools.h"
```

Include dependency graph for protocol.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Dataobj

    This struct contains the parameters of the data object that can be used to transmit data.

Macros

- #define **Q_VALUE_TYPE** 'q'
- #define **POSITION_TYPE** 'p'
- #define **TABLE_TYPE** 't'
- #define **START_TYPE** 's'
- #define **END_TYPE** 'e'
- #define **UCHAR_SIZE** (sizeof(unsigned char))
- #define **CHAR_SIZE** (sizeof(char))
- #define **INT_SIZE** (sizeof(int))
- #define **FLOAT_SIZE** (sizeof(float))

Functions

- void transmit_data (struct Dataobj ∗dObj)

  Transmits the datastruct.
- void init_QvalProtocol (int QVal, struct Dataobj ∗Q_Data)

  Initializes a Q_value data object.
- void init_PositionProtocol (unsigned char position[2], struct Dataobj ∗posObj)

  Initializes a position data object.
- void init_SizeProtocol (unsigned char size[2], struct Dataobj ∗sizeObj)

  Initializes a size data object.

4.4.1 Detailed Description

This library contains the custom protocol used to send data to a peripheral device which uses the same protocol to receive them.

Author

Jonas Van Der Donckt

Date

14/05/2017

4.4.2 Function Documentation

4.4.2.1 init_PositionProtocol()

```
void init_PositionProtocol (
            unsigned char position[2],
            struct Dataobj * posObj )
```

Initializes a position data object.

Parameters

position	the indices of the position (row, column)
dataObj1	passes areference to the Dataobj that needs to be initialized

4.4.2.2 init_QvalProtocol()

```
void init_QvalProtocol (
            int QVal,
            struct Dataobj * Q_Data )
```

Initializes a Q_value data object.

Parameters

Qval	the Q value
Q_Data	passes a reference to the Dataobj that needs to be initialized

4.4.2.3 init_SizeProtocol()

```
void init_SizeProtocol (
            unsigned char size[2],
            struct Dataobj * sizeObj )
```

Initializes a size data object.

Parameters

size	the indices of the Q_table's size (row, column)
sizeObj	passes a reference to the Dataobj that needs to be initialized

4.4.2.4 transmit_data()

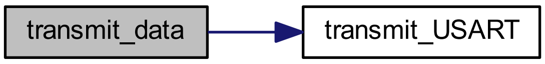
```
void transmit_data (
            struct Dataobj * dObj )
```

Transmits the datastruct.

Parameters

dObj	passes a reference to the Dataobj that needs to be transmitted

Here is the call graph for this function:

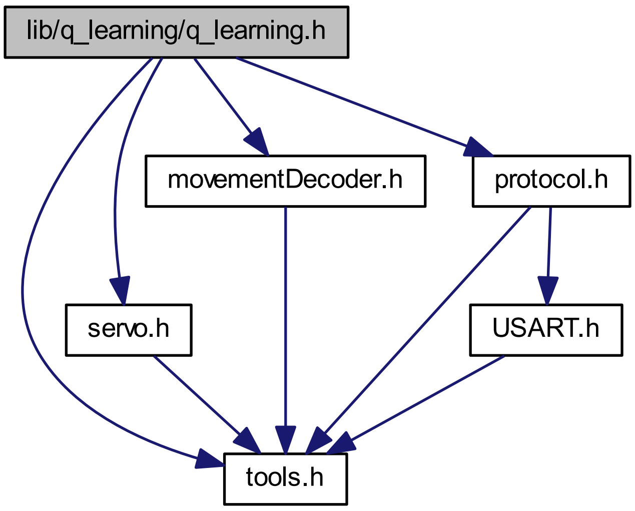


4.5 lib/q_learning/q_learning.h File Reference

This library contains the Q-learning algorithm. before you can use the learn method function you should call the initQ_Learning() function. Afterwards you can call the learn() function.

```
#include "tools.h"
#include "servo.h"
#include "movementDecoder.h"
#include "protocol.h"
```

Include dependency graph for q_learning.h:



Data Structures

- struct Formula_Parameters_t

    This struct contains the parameters of the Q-learning formula. Some other functions are specificly designed to change those parameters by using an exponential function.

Macros

- #define **NUMB_ACTIONS** (2∗NUMB_SERVOS)
- #define OPT_PER_SERVO 2

    options of actions per servo
- #define **NUMB_STATES** ((unsigned char)pow(NUMB_SERVO_STATES,NUMB_SERVOS))
- #define RAND_INITQ_B 1

    0 if the Q table doesn't need a random initialisation
- #define EXP_ENABLED 1

    bool; 1 if the
- #define **EULER** (double)2.71828
- #define **NUMB_PARAMETERS** 3

- #define RADIX 100

  fixed point notation
- #define MAX_FACTOR 100

  fixed point notation is used –> this is maximum
- #define EXP_REWARD_FACTOR (double)0.1

  the exponential reward factor
- #define **MAX_COUNT** 80
- #define **LEARNING_RATE_INDEX** 0
- #define INIT_LEARNING_RATE 90

  alpha 0.5
- #define **EXP_FACTOR_LEARNING** (float)1
- #define **FINAL_LEARING_RATE** 40
- #define **GREEDY_FACTOR_INDEX** 1
- #define INIT_GREEDY_FACTOR 90

  epsillon 0.1
- #define **EXP_GREEDY_FACTOR** (float)1
- #define **FINAL_GREEDY_FACTOR** 0
- #define **DISCOUNT_FACTOR_INDEX** 2
- #define INIT_DISCOUNT_FACTOR 99

  gamma 0.99
- #define **EXP_FACTOR_DISCOUNT** (float)1
- #define **FINAL_DISCOUNT_FACTOR** 50
- #define STARTSTATES1 0

  this value should be < than NUMB_STATES
- #define **STARTSTATES2** 3
- #define **STARTSTATES3** 0
- #define TRESHMOVFW 2

  threshold value for moving forward
- #define DELAYVAL 700

  value in ms that is used to wait until the robot stopped moving
- #define **BTN_S** PORTE5
- #define **BTN_S_PORT** PORTE
- #define **BTN_S_DIR** DDRE
- #define **BTN_S_INT** INT5
- #define **PREVACTION_INDEX_NIBBLE** 0
- #define **BTNPRESSED_INDEX** 7
- #define **CALC_STATE_INDEX**(STATES1, STATES2) (STATES1∗4 + STATES2)
- #define **GETSIZE**(QTABLE) (sizeof(QTABLE[0])/ sizeof(QTABLE[0][0]))
- #define **ENABLE_BTN** (SET_BIT_HIGH(EIMSK, BTN_S_INT))

Functions

- void initQ_Learning ()

  Initializes the Q-learning algorithm. This function sets up the pins and the interrupt for the button. It also initialzes the needed components for the Q-learning algorithm (servo.h, movementDecoder.h)
- void initQ_Table ()

  Initializes the Q_Table with random values.
- void learn ()

  The Q-learning algorithm itself. This function executes the Q_learning algorithm itself.
- int calcQVal ()

  This function calculates the Q-value. It should be only called by the learn() method.
- unsigned char calcNextAction ()

This function calculates the next action. It should be only called by the learn() method.
- unsigned char CalcStateIndex (unsigned char stateArr[ ])

  Calculates the state index.
- int calcMaxQval (unsigned char state)

  This function calculates the maximum Q-value for a certain state. It should be only called by the learn() method.
- unsigned char calcMaxActionIndex (unsigned char state)

  This function calculates the maximum state index. It should be only called by the learn() method.
- unsigned char wait_BTN_S ()

  This function waits until button south is pressed.
- void adjustParameters ()

  Adjusts the parameters in the Q-learning formule.
- void calcParameter (struct Formula_Parameters_t ∗paramVal)

  Calculates the new value of the parameters used in the Q-learning formula.

4.5.1 Detailed Description

This library contains the Q-learning algorithm. before you can use the learn method function you should call the initQ_Learning() function. Afterwards you can call the learn() function.

Author

Jonas Van Der Donckt, Jules Noppe

Date

6/05/2017

4.5.2 Macro Definition Documentation

4.5.2.1 RAND_INITQ_B

```
#define RAND_INITQ_B 1
```

0 if the Q table doesn't need a random initialisation

bool; 1 if Q table needs to get randomly initialized

4.5.3 Function Documentation

4.5.3.1 calcMaxActionIndex()

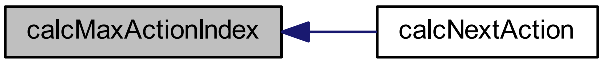
```
unsigned char calcMaxActionIndex (
            unsigned char state )
```

This function calculates the maximum state index. It should be only called by the learn() method.

Parameters

state	this is the state of the robot

Here is the caller graph for this function:



4.5.3.2 calcMaxQval()

```
int calcMaxQval (
             unsigned char state )
```

This function calculates the maximum Q-value for a certain state. It should be only called by the learn() method.

Parameters

state	this is the state of the robot

4.5.3.3 calcParameter()

```
void calcParameter (
             struct Formula_Parameters_t * paramVal )
```

Calculates the new value of the parameters used in the Q-learning formula.

Parameters

∗paramVal	this is the parameter that will be updated

4.5.3.4 CalcStateIndex()

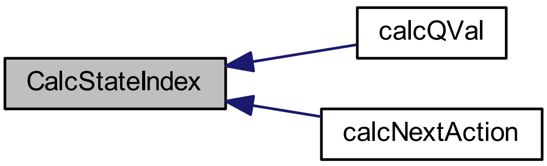
```
unsigned char CalcStateIndex (
             unsigned char stateArr[ ] )
```

Calculates the state index.

Parameters

stateArr[ ]	this array contains the stateval of each servo

Here is the caller graph for this function:

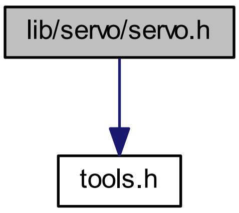


4.6 lib/servo/servo.h File Reference

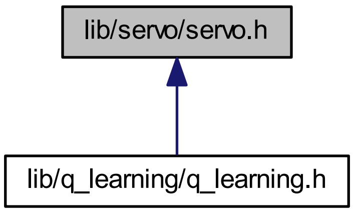
This library contains functions whom help to control multiple servos. Before using any of the the servo methods, call the initServo() function.

```
#include "tools.h"
```

Include dependency graph for servo.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct servo_t

  This struct contains the parameters of a servo. Some other functions are specificly designed to change those parameters.

Macros

- #define CLK_TICKS_US 16

  the unprescaled clock uses 16 ticks per µs
- #define PRESCALER 8

  the prescaler of the clock
- #define MAXANGLE 180

  just informative
- #define MIN_STATE_ANGLE 10

  must be smaller than MAX_STATE_ANGLE
- #define MAX_STATE_ANGLE 110

  must be smaller than MAXANGLE
- #define MAX_NUMB_SERVOS 8

  this is the maximum amount of servos
- #define MIN_PULSE_WIDTH 600

  the shortest pulse sent to a servo in µs
- #define MAX_PULSE_WIDTH 2400

  the longest pulse sent to a servo
- #define DEFAULT_PULSE_WIDTH 1500

  default pulse width when servo is attached
- #define PERIOD 20000

  this is the refresh interval
- #define **NUMB_SERVOS** 2
- #define **NUMB_SERVO_STATES** 4
- #define INTERPOLLATEINT 1500

  refresh inte rval for interpollation
- #define INTERPOLLATEVAL 1

  angle change wich each interpollation
- #define **NUMB_SWITCH_STATES** 2

- #define **PULSE** 0
- #define **WAIT** 1
- #define ISMOVING_INDEX 7

    index of the bool isMoving in the data byte
- #define SERVO_INDEX_NIBBLE 0

    start index of the nibble that contains the servo index
- #define STATE_INDEX_POS 4

    single bit value that contains the switch state index
- #define **SERVO1** PORTC0
- #define SERVO1_PORT PORTC

    will be used as a pointer
- #define SERVO1_DIR DDRC

    will be used as a pointer
- #define **SERVO2** PORTC1
- #define SERVO2_PORT PORTC

    will be used as a pointer
- #define SERVO2_DIR DDRC

    will be used as a pointer
- #define **SERVO3** PORTB1
- #define SERVO3_PORT PORTB

    will be used as a pointer
- #define SERVO3_DIR DDRB

    will be used as a pointer
- #define **SERVO4** 0
- #define SERVO4_PORT PORTC

    will be used as a pointer
- #define SERVO4_DIR DDRC

    will be used as a pointer
- #define **SERVO5** 0
- #define SERVO5_PORT PORTC

    will be used as a pointer
- #define SERVO5_DIR DDRC

    will be used as a pointer
- #define **SERVO6** 0
- #define SERVO6_PORT PORTC

    will be used as a pointer
- #define SERVO6_DIR DDRC

    will be used as a pointer
- #define **SERVO7** 0
- #define SERVO7_PORT PORTC

    will be used as a pointer
- #define SERVO7_DIR DDRC

    will be used as a pointer
- #define **SERVO8** 0
- #define SERVO8_PORT PORTC

    will be used as a pointer
- #define SERVO8_DIR DDRC

    will be used as a pointer
- #define **STATE_ANGLE_WIDTH** ((MAX_STATE_ANGLE - MIN_STATE_ANGLE)/(NUMB_SERVO_STA↩
TES - 1))
- #define BASE_TIME_WIDTH (PERIOD/NUMB_SERVOS)

val in µs
- #define **ANGLE_TO_TICKS_INTERP**(ANG) (((US_TO_TICKS(MAX_PULSE_WIDTH - MIN_PULSE_WI↩ DTH)/180)∗ANG))
- #define **US_TO_TICKS**(US) (US∗(CLK_TICKS_US/PRESCALER))
- #define **ANGLE_TO_US**(ANG) (MIN_PULSE_WIDTH + ((MAX_PULSE_WIDTH - MIN_PULSE_WID↩ TH)/180)∗ANG)
- #define **ANGLE_TO_TICKS**(ANG) (US_TO_TICKS(ANGLE_TO_US(ANG)))
- #define **TICKS_TO_ANGLE**(TCK) ((TCK - US_TO_TICKS(MIN_PULSE_WIDTH))/(US_TO_TICKS((MAX↩ _PULSE_WIDTH - MIN_PULSE_WIDTH)/180)))

Functions

- void initServo (unsigned char restStateArr[NUMB_SERVOS])

  Initializes the servo motors. This function sets up the pins and interrupts for the servos. It should be called before you want to use the servos.
- void servoWriteStates (unsigned char stateArr[ ])

  This function makes the servos go to a given state.
- void servoWriteAngles (unsigned char posArr[ ])

  This function makes the servo's go to a given angle.
- void servoWriteState (unsigned char state, BYTE servoIndex)

  This function makes a specific servo go to a given state.
- void servoWriteAngle (unsigned char angle, BYTE servoIndex)

  This function makes the servo's go to a given angle.
- void interpollate ()

  Interpollates the movement of the servo's. This function should only be used within this library.
- void setServoState ()

  This function helps to generate a PWM-signal. This function should only be used by servo timer interrupt (TIMER1↩ _COMPA_vect)
- void enableServo (int index)

  This function enables the servo. It should be called before writing a position to the servos.
- void disableServo (int index)

  This function disables the servo. Writing a position to the servo will have no effect. When you enable the servo, it will go to the last written position.
- void waitServoMovement ()

  This function waits until the servo reached the goal position.

4.6.1 Detailed Description

This library contains functions whom help to control multiple servos. Before using any of the the servo methods, call the initServo() function.

Author

Jonas Van Der Donckt

Date

16/03/2017

4.6.2 Function Documentation

4.6.2.1 disableServo()

```
void disableServo (
            int index )
```

This function disables the servo. Writing a position to the servo will have no effect. When you enable the servo, it will go to the last written position.

Parameters

index	the index of the servo that should be disabled

4.6.2.2 enableServo()

```
void enableServo (
            int index )
```

This function enables the servo. It should be called before writing a position to the servos.

Parameters

index	the index of the servo that should be enabled

4.6.2.3 initServo()

```
void initServo (
            unsigned char restStateArr[NUMB_SERVOS] )
```

Initializes the servo motors. This function sets up the pins and interrupts for the servos. It should be called before you want to use the servos.

Parameters

restStateArr	these are the restStates of the servos

4.6.2.4 servoWriteAngle()

```
void servoWriteAngle (
            unsigned char angle,
            BYTE servoIndex )
```

This function makes the servo's go to a given angle.

Parameters

angle	this contains the angle the servo needs to go to
servoIndex	this contains the index of the servo

4.6.2.5 servoWriteAngles()

```
void servoWriteAngles (
            unsigned char posArr[] )
```

This function makes the servo's go to a given angle.

Parameters

posArr[]	this contains the angle values of the servo's

4.6.2.6 servoWriteState()

```
void servoWriteState (
            unsigned char state,
            BYTE servoIndex )
```

This function makes a specific servo go to a given state.

Parameters

state	this contains the state index corresponding servo
servoIndex	this contains the index of the servo

4.6.2.7 servoWriteStates()

```
void servoWriteStates (
            unsigned char stateArr[] )
```

This function makes the servos go to a given state.

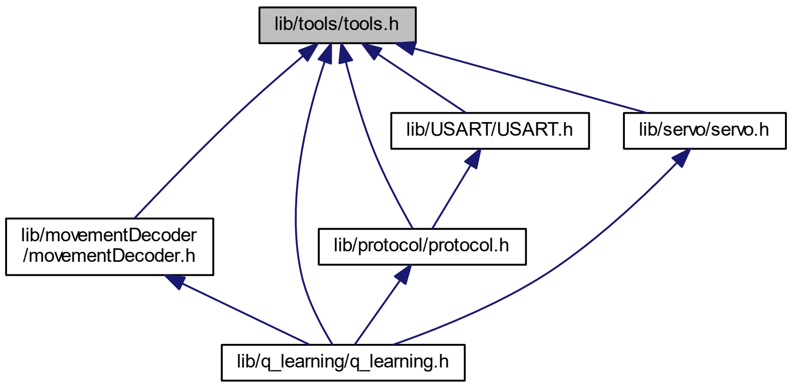
Parameters

stateArr[ ]	this contains the state index corresponding servo

4.7 lib/tools/tools.h File Reference

This library contains useful MACROS which can be used in other libraries.

This graph shows which files directly or indirectly include this file:



Macros

- #define F_CPU 16000000UL

  adapt this if the CPU has a different frequency
- #define **BYTE** unsigned char
- #define **SET_PIN_HIGH**(PORT, PIN) PORT |= (1 << PIN)
- #define **SET_PIN_LOW**(PORT, PIN) PORT &= ∼(1 << PIN)
- #define **SET_BIT_HIGH**(REG, BIT) REG |= (1 << BIT)
- #define **SET_BIT_LOW**(REG, BIT) REG &= ∼(1 << BIT)
- #define **SET_PIN_OUTPUT**(DDR, PIN) SET_PIN_HIGH(DDR, PIN)
- #define **SET_PIN_INPUT**(DDR, PIN) SET_PIN_LOW(DDR, PIN)
- #define **TOGGLE_BIT**(REG, POS) (REG ^= (1<< POS))
- #define **GET_BIT**(REG, POS) ((REG & (1<< POS)) > (0) ? (1) : (0))
- #define **GET_NIBBLE**(REG, LSB_INDEX, D1) (D1 = (REG & (0x0F << LSB_INDEX)))
- #define **SET_NIBBLE**(REG, LSB_INDEX, D1) (REG |= (D1 << LSB_INDEX))
- #define **CLEAR_NIBBLE**(REG, LSB_INDEX) (REG &=∼(0x0F << LSB_INDEX))

4.7.1 Detailed Description

This library contains useful MACROS which can be used in other libraries.

Author

Jonas Van Der Donckt

Date

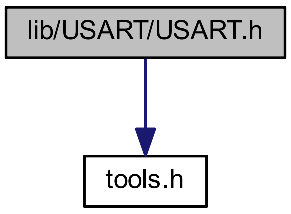
14/05/2017

4.8 lib/USART/USART.h File Reference

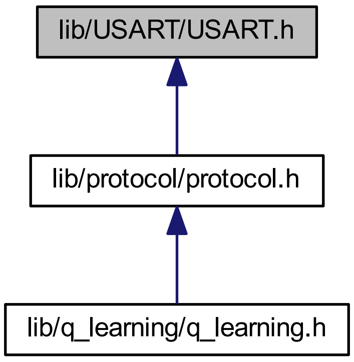
This library contains functions to transmit data with the USART protocol. Before using the USART's methods, call the initUSART() function.

```
#include "tools.h"
```

Include dependency graph for USART.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define BAUDRATE (long)9600

    adapt to change BAUDRATE
- #define **UBBRVAL** (F_CPU/(16∗BAUDRATE) - 1)

Functions

- void initUSART ()

  Initializes the USART protocol and configures the frame format. This function should be called before using any of the other methods within this library.
- void transmit_USART (BYTE data)

  Transmits data with the USART protocol.
- unsigned char Receive_USART ()

  Receives data with the USART protocol.

4.8.1 Detailed Description

This library contains functions to transmit data with the USART protocol. Before using the USART's methods, call the initUSART() function.

Author

Jonas Van Der Donckt

Date

14/05/2017

4.8.2 Function Documentation

4.8.2.1 initUSART()

```
void initUSART ( )
```

Initializes the USART protocol and configures the frame format. This function should be called before using any of the other methods within this library.

USART.c

Created on: May 14, 2017 Author: Jonas Van Der Donckt

4.8.2.2 transmit_USART()

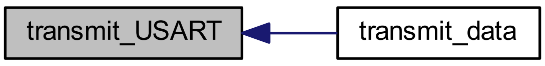
```
void transmit_USART (
             BYTE data )
```

Transmits data with the USART protocol.

Parameters

data	this is the data that will be transmitted.

Here is the caller graph for this function:


transmit_USART
transmit_data