

An autonomous crawling robot

Generated by Doxygen 1.8.13

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	Dataobj Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Field Documentation	6
3.1.2.1	data	6
3.1.2.2	dataSize	6
3.1.2.3	type	6
3.2	Formula_Parameters_t Struct Reference	6
3.2.1	Detailed Description	7
3.2.2	Field Documentation	7
3.2.2.1	current	7
3.2.2.2	exp_Factor	7
3.2.2.3	finalVal	7
3.2.2.4	initialVal	7
3.3	Icd_info_type Struct Reference	8
3.3.1	Detailed Description	8
3.3.2	Field Documentation	8
3.3.2.1	line	8
3.3.2.2	pos	8
3.4	servo_t Struct Reference	9
3.4.1	Detailed Description	9
3.4.2	Field Documentation	9
3.4.2.1	isActive	9
3.4.2.2	pin	10
3.4.2.3	port	10
3.4.2.4	ticks	10

4 File Documentation	11
4.1 lib/dwenguinoBoard/dwenguinoBoard.c File Reference	11
4.1.1 Function Documentation	11
4.1.1.1 initBoard()	12
4.2 lib/dwenguinoBoard/dwenguinoBoard.h File Reference	12
4.2.1 Detailed Description	14
4.2.2 Macro Definition Documentation	15
4.2.2.1 BYTE	15
4.2.2.2 FALSE	15
4.2.2.3 HIGH	15
4.2.2.4 INPUT	15
4.2.2.5 LCD_BACKLIGHT_IN	15
4.2.2.6 LCD_BACKLIGHT_OFF	15
4.2.2.7 LCD_BACKLIGHT_ON	15
4.2.2.8 LCD_BACKLIGHT_OUT	16
4.2.2.9 LCD_DATA	16
4.2.2.10 LCD_DATA_DIR	16
4.2.2.11 LCD_EN_HIGH	16
4.2.2.12 LCD_EN_LOW	16
4.2.2.13 LCD_EN_OUT	16
4.2.2.14 LCD_RS_HIGH	16
4.2.2.15 LCD_RS_LOW	16
4.2.2.16 LCD_RS_OUT	17
4.2.2.17 LCD_RW_HIGH	17
4.2.2.18 LCD_RW_LOW	17
4.2.2.19 LCD_RW_OUT	17
4.2.2.20 LED_OFF	17
4.2.2.21 LED_ON	17
4.2.2.22 LEDS	17
4.2.2.23 LEDS_DIR	18

4.2.2.24	LOW	18
4.2.2.25	MOTOR1_0_HIGH	18
4.2.2.26	MOTOR1_0_LOW	18
4.2.2.27	MOTOR1_1_HIGH	18
4.2.2.28	MOTOR1_1_LOW	18
4.2.2.29	MOTOR2_0_HIGH	18
4.2.2.30	MOTOR2_0_LOW	18
4.2.2.31	MOTOR2_1_HIGH	19
4.2.2.32	MOTOR2_1_LOW	19
4.2.2.33	OUTPUT	19
4.2.2.34	PORT_HIGH	19
4.2.2.35	PORT_LOW	19
4.2.2.36	SERVO1	19
4.2.2.37	SERVO2	19
4.2.2.38	SET_BIT_HIGH	20
4.2.2.39	SET_BIT_LOW	20
4.2.2.40	SET_PIN_HIGH	20
4.2.2.41	SET_PIN_LOW	20
4.2.2.42	SW_C_HIGH	20
4.2.2.43	SW_C_IN	20
4.2.2.44	SW_C_LOW	20
4.2.2.45	SW_C_OUT	21
4.2.2.46	SW_E_HIGH	21
4.2.2.47	SW_E_IN	21
4.2.2.48	SW_E_LOW	21
4.2.2.49	SW_E_OUT	21
4.2.2.50	SW_N_HIGH	21
4.2.2.51	SW_N_IN	21
4.2.2.52	SW_N_LOW	21
4.2.2.53	SW_N_OUT	22

4.2.2.54	SW_S_HIGH	22
4.2.2.55	SW_S_IN	22
4.2.2.56	SW_S_LOW	22
4.2.2.57	SW_S_OUT	22
4.2.2.58	SW_W_HIGH	22
4.2.2.59	SW_W_IN	22
4.2.2.60	SW_W_LOW	22
4.2.2.61	SW_W_OUT	23
4.2.2.62	TRUE	23
4.2.3	Function Documentation	23
4.2.3.1	initBoard()	23
4.3	lib/dwenguinoLCD/dwenguinoLCD.c File Reference	23
4.3.1	Function Documentation	25
4.3.1.1	appendCharToLCD()	25
4.3.1.2	appendIntToLCD()	25
4.3.1.3	appendStringToLCD_()	26
4.3.1.4	appendStringToLCDcharptr()	27
4.3.1.5	clearLCD()	27
4.3.1.6	commandLCD()	28
4.3.1.7	initLCD()	28
4.3.1.8	printCharToLCD()	29
4.3.1.9	printIntToLCD()	30
4.3.1.10	printStringToLCD()	30
4.3.1.11	setCursorLCD()	31
4.3.2	Variable Documentation	32
4.3.2.1	lcd_info	32
4.4	lib/dwenguinoLCD/DwenguinoLCD.h File Reference	32
4.4.1	Detailed Description	34
4.4.2	Macro Definition Documentation	34
4.4.2.1	appendStringToLCD	34

4.4.2.2	backlightOff	34
4.4.2.3	backlightOn	34
4.4.2.4	LCD_HEIGHT	35
4.4.2.5	LCD_LASTLINE	35
4.4.2.6	LCD_LASTPOS	35
4.4.2.7	LCD_WIDTH	35
4.4.3	Function Documentation	35
4.4.3.1	appendCharToLCD()	35
4.4.3.2	appendIntToLCD()	36
4.4.3.3	clearLCD()	37
4.4.3.4	commandLCD()	37
4.4.3.5	initLCD()	38
4.4.3.6	printCharToLCD()	39
4.4.3.7	printIntToLCD()	39
4.4.3.8	setCursorLCD()	40
4.4.4	Variable Documentation	40
4.4.4.1	lcd_info	40
4.5	lib/movementDecoder/movementDecoder.c File Reference	41
4.5.1	Function Documentation	41
4.5.1.1	checkMovementDir()	42
4.5.1.2	clearDirCount()	42
4.5.1.3	getMovementDirCount()	42
4.5.1.4	initMovementDecoder()	43
4.5.1.5	ISR() [1/2]	43
4.5.1.6	ISR() [2/2]	43
4.5.2	Variable Documentation	44
4.5.2.1	dirCount	44
4.5.2.2	prevAngleState	44
4.6	lib/movementDecoder/movementDecoder.h File Reference	44
4.6.1	Detailed Description	46

4.6.2	Macro Definition Documentation	46
4.6.2.1	DECODER1	46
4.6.2.2	DECODER1_DATA	46
4.6.2.3	DECODER1_DIR	46
4.6.2.4	DECODER1_INT	46
4.6.2.5	DECODER1_PORT	47
4.6.2.6	DECODER2	47
4.6.2.7	DECODER2_DATA	47
4.6.2.8	DECODER2_DIR	47
4.6.2.9	DECODER2_INT	47
4.6.2.10	DECODER2_PORT	47
4.6.2.11	GRAY_TO_BIN	47
4.6.3	Function Documentation	48
4.6.3.1	checkMovementDir()	48
4.6.3.2	clearDirCount()	48
4.6.3.3	getMovementDirCount()	49
4.6.3.4	initMovementDecoder()	49
4.7	lib/protocol/protocol.c File Reference	49
4.7.1	Function Documentation	50
4.7.1.1	init_PositionProtocol()	50
4.7.1.2	init_QvalProtocol()	51
4.7.1.3	init_SizeProtocol()	51
4.7.1.4	transmit_data()	53
4.8	lib/protocol/protocol.h File Reference	54
4.8.1	Detailed Description	55
4.8.2	Macro Definition Documentation	55
4.8.2.1	CHAR_SIZE	55
4.8.2.2	END_TYPE	56
4.8.2.3	FLOAT_SIZE	56
4.8.2.4	INT_SIZE	56

4.8.2.5	POSITION_TYPE	56
4.8.2.6	Q_VALUE_TYPE	56
4.8.2.7	START_TYPE	56
4.8.2.8	TABLE_TYPE	56
4.8.2.9	UCHAR_SIZE	56
4.8.3	Function Documentation	56
4.8.3.1	init_PositionProtocol()	56
4.8.3.2	init_QvalProtocol()	57
4.8.3.3	init_SizeProtocol()	57
4.8.3.4	transmit_data()	59
4.9	lib/q_learning/q_learning.c File Reference	60
4.9.1	Function Documentation	61
4.9.1.1	adjustParameters()	61
4.9.1.2	calcMaxActionIndex()	61
4.9.1.3	calcMaxQval()	62
4.9.1.4	calcNextAction()	62
4.9.1.5	calcParameter()	63
4.9.1.6	calcQVal()	64
4.9.1.7	CalcStateIndex()	64
4.9.1.8	initQ_Learning()	65
4.9.1.9	initQ_Table()	66
4.9.1.10	ISR()	67
4.9.1.11	Learn()	67
4.9.1.12	waitBTN()	68
4.9.2	Variable Documentation	68
4.9.2.1	dataByte	68
4.9.2.2	nextServoStateArr	68
4.9.2.3	numblterations	68
4.9.2.4	parameters	68
4.9.2.5	prevServoStateArr	69

4.9.2.6	Q_Table	69
4.9.2.7	temp	69
4.10	lib/q_learning/q_learning.h File Reference	69
4.10.1	Detailed Description	72
4.10.2	Macro Definition Documentation	72
4.10.2.1	BTN_S	72
4.10.2.2	BTN_S_DIR	72
4.10.2.3	BTN_S_INT	72
4.10.2.4	BTN_S_PORT	72
4.10.2.5	BTNPRESSED_INDEX	72
4.10.2.6	CALC_STATE_INDEX	73
4.10.2.7	DELAYVAL	73
4.10.2.8	DISCOUNT_FACTOR_INDEX	73
4.10.2.9	ENABLE_BTN	73
4.10.2.10	EULER	73
4.10.2.11	EXP_ENABLED	73
4.10.2.12	EXP_FACTOR_DISCOUNT	73
4.10.2.13	EXP_FACTOR_LEARNING	74
4.10.2.14	EXP_GREEDY_FACTOR	74
4.10.2.15	EXP_REWARD_FACTOR	74
4.10.2.16	FINAL_DISCOUNT_FACTOR	74
4.10.2.17	FINAL_GREEDY_FACTOR	74
4.10.2.18	FINAL_LEARNING_RATE	74
4.10.2.19	GETSIZE	74
4.10.2.20	GREEDY_FACTOR_INDEX	75
4.10.2.21	INIT_DISCOUNT_FACTOR	75
4.10.2.22	INIT_GREEDY_FACTOR	75
4.10.2.23	INIT_LEARNING_RATE	75
4.10.2.24	LEARNING_RATE_INDEX	75
4.10.2.25	MAX_COUNT	75

4.10.2.26 MAX_FACTOR	75
4.10.2.27 NUMB_ACTIONS	76
4.10.2.28 NUMB_PARAMETERS	76
4.10.2.29 NUMB_STATES	76
4.10.2.30 OPT_PER_SERVO	76
4.10.2.31 PREVACTION_INDEX_NIBBLE	76
4.10.2.32 RADIX	76
4.10.2.33 RAND_INITQ_B	76
4.10.2.34 STARTSTATES1	77
4.10.2.35 STARTSTATES2	77
4.10.2.36 STARTSTATES3	77
4.10.2.37 TRESHMOVFW	77
4.10.3 Function Documentation	77
4.10.3.1 adjustParameters()	77
4.10.3.2 calcMaxActionIndex()	78
4.10.3.3 calcMaxQval()	78
4.10.3.4 calcNextAction()	79
4.10.3.5 calcParameter()	79
4.10.3.6 calcQVal()	80
4.10.3.7 CalcStateIndex()	81
4.10.3.8 initQ_Learning()	81
4.10.3.9 initQ_Table()	82
4.10.3.10 learn()	83
4.10.3.11 wait_BTN_S()	83
4.11 lib/servo/servo.c File Reference	84
4.11.1 Function Documentation	85
4.11.1.1 disableServo()	85
4.11.1.2 enableServo()	85
4.11.1.3 initServo()	86
4.11.1.4 interpolate()	86

4.11.1.5	ISR()	87
4.11.1.6	servoWriteAngle()	87
4.11.1.7	servoWriteAngles()	88
4.11.1.8	servoWriteState()	88
4.11.1.9	servoWriteStates()	89
4.11.1.10	setServoState()	90
4.11.1.11	waitServoMovement()	90
4.11.2	Variable Documentation	91
4.11.2.1	CurrentTicks	91
4.11.2.2	dataB	91
4.11.2.3	interpolateTime	91
4.11.2.4	servos	91
4.11.2.5	stateAngles	91
4.12	lib/servo/servo.h File Reference	92
4.12.1	Detailed Description	95
4.12.2	Macro Definition Documentation	95
4.12.2.1	ANGLE_TO_TICKS	95
4.12.2.2	ANGLE_TO_TICKS_INTERP	95
4.12.2.3	ANGLE_TO_US	96
4.12.2.4	BASE_TIME_WIDTH	96
4.12.2.5	CLK_TICKS_US	96
4.12.2.6	DEFAULT_PULSE_WIDTH	96
4.12.2.7	INTERPOLLATEINT	96
4.12.2.8	INTERPOLLATEVAL	96
4.12.2.9	ISMOVING_INDEX	97
4.12.2.10	MAX_NUMB_SERVOS	97
4.12.2.11	MAX_PULSE_WIDTH	97
4.12.2.12	MAX_STATE_ANGLE	97
4.12.2.13	MAXANGLE	97
4.12.2.14	MIN_PULSE_WIDTH	97

4.12.2.15 MIN_STATE_ANGLE	98
4.12.2.16 NUMB_SERVO_STATES	98
4.12.2.17 NUMB_SERVOS	98
4.12.2.18 NUMB_SWITCH_STATES	98
4.12.2.19 PERIOD	98
4.12.2.20 PRESCALER	98
4.12.2.21 PULSE	98
4.12.2.22 SERVO1	99
4.12.2.23 SERVO1_DIR	99
4.12.2.24 SERVO1_PORT	99
4.12.2.25 SERVO2	99
4.12.2.26 SERVO2_DIR	99
4.12.2.27 SERVO2_PORT	99
4.12.2.28 SERVO3	99
4.12.2.29 SERVO3_DIR	100
4.12.2.30 SERVO3_PORT	100
4.12.2.31 SERVO4	100
4.12.2.32 SERVO4_DIR	100
4.12.2.33 SERVO4_PORT	100
4.12.2.34 SERVO5	100
4.12.2.35 SERVO5_DIR	101
4.12.2.36 SERVO5_PORT	101
4.12.2.37 SERVO6	101
4.12.2.38 SERVO6_DIR	101
4.12.2.39 SERVO6_PORT	101
4.12.2.40 SERVO7	101
4.12.2.41 SERVO7_DIR	102
4.12.2.42 SERVO7_PORT	102
4.12.2.43 SERVO8	102
4.12.2.44 SERVO8_DIR	102

4.12.2.45	SERVO8_PORT	102
4.12.2.46	SERVO_INDEX_NIBBLE	102
4.12.2.47	STATE_ANGLE_WIDTH	103
4.12.2.48	STATE_INDEX_POS	103
4.12.2.49	TICKS_TO_ANGLE	103
4.12.2.50	US_TO_TICKS	103
4.12.2.51	WAIT	103
4.12.3	Function Documentation	103
4.12.3.1	disableServo()	103
4.12.3.2	enableServo()	104
4.12.3.3	initServo()	104
4.12.3.4	interpollate()	105
4.12.3.5	servoWriteAngle()	105
4.12.3.6	servoWriteAngles()	106
4.12.3.7	servoWriteState()	106
4.12.3.8	servoWriteStates()	107
4.12.3.9	setServoState()	108
4.12.3.10	waitServoMovement()	108
4.13	lib/tools/tools.c File Reference	109
4.14	lib/tools/tools.h File Reference	109
4.14.1	Detailed Description	110
4.14.2	Macro Definition Documentation	110
4.14.2.1	BYTE	110
4.14.2.2	CLEAR_NIBBLE	110
4.14.2.3	F_CPU	110
4.14.2.4	GET_BIT	111
4.14.2.5	GET_NIBBLE	111
4.14.2.6	SET_BIT_HIGH	111
4.14.2.7	SET_BIT_LOW	111
4.14.2.8	SET_NIBBLE	111

4.14.2.9	SET_PIN_HIGH	111
4.14.2.10	SET_PIN_INPUT	112
4.14.2.11	SET_PIN_LOW	112
4.14.2.12	SET_PIN_OUTPUT	112
4.14.2.13	TOGGLE_BIT	112
4.15	lib/USART/USART.c File Reference	112
4.15.1	Function Documentation	113
4.15.1.1	initUSART()	113
4.15.1.2	Receive_USART()	113
4.15.1.3	transmit_USART()	113
4.16	lib/USART/USART.h File Reference	114
4.16.1	Detailed Description	115
4.16.2	Macro Definition Documentation	116
4.16.2.1	BAUDRATE	116
4.16.2.2	UBBRVAL	116
4.16.3	Function Documentation	116
4.16.3.1	initUSART()	116
4.16.3.2	Receive_USART()	116
4.16.3.3	transmit_USART()	116
4.17	src/main.c File Reference	117
4.17.1	Function Documentation	117
4.17.1.1	main()	118
4.17.1.2	waitForButton()	118

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Dataobj	This struct contains the parameters of the data object that can be used to transmit data	5
Formula_Parameters_t	This struct contains the parameters of the Q-learning formula. Some other functions are specifically designed to change those parameters by using an exponential function	6
lcd_info_type	Stores the current line and position of the cursor	8
servo_t	This struct contains the parameters of a servo. Some other functions are specifically designed to change those parameters	9

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

lib/dwenguinoBoard/ dwenguinoBoard.c	11
lib/dwenguinoBoard/ dwenguinoBoard.h	
This library contains pin assignments and basic macros for Dwenguino boards	12
lib/dwenguinoLCD/ dwenguinoLCD.c	23
lib/dwenguinoLCD/ DwenguinoLCD.h	
This library contains function declarations which allow you to communicate with the lcd screen on the Dwenguino board	32
lib/movementDecoder/ movementDecoder.c	41
lib/movementDecoder/ movementDecoder.h	
This library contains functions to determine the direction by using 2 optosensores. Before using the movementDecoder's methods, call the initMovementDecoder() function	44
lib/protocol/ protocol.c	49
lib/protocol/ protocol.h	
This library contains the custom protocol used to send data to a peripheral device which uses the same protocol to receive them	54
lib/q_learning/ q_learning.c	60
lib/q_learning/ q_learning.h	
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	69
lib/servo/ servo.c	84
lib/servo/ servo.h	
This library contains functions whom help to control multiple servos. Before using any of the the servo methods, call the initServo() function	92
lib/tools/ tools.c	109
lib/tools/ tools.h	
This library contains useful MACROS which can be used in other libraries	109
lib/USART/ USART.c	112
lib/USART/ USART.h	
This library contains functions to transmit data with the USART protocol. Before using the USART's methods, call the initUSART() function	114
src/ main.c	117

Chapter 3

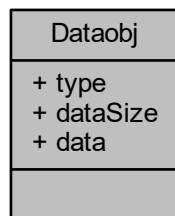
Data Structure Documentation

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

<i>type</i>	this parameter contains the representation of the data
<i>DataSize</i>	contains the amount of bytes of the data

3.1.2 Field Documentation**3.1.2.1 data**

```
int data
```

3.1.2.2 dataSize

```
unsigned char dataSize
```

3.1.2.3 type

```
unsigned char type
```

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

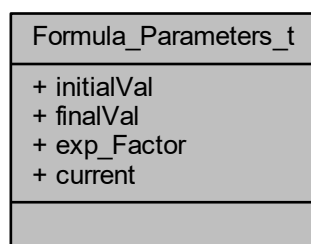
- [lib/protocol/protocol.h](#)

3.2 Formula_Parameters_t Struct Reference

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

3.2.2 Field Documentation

3.2.2.1 [current](#)

```
float current
```

3.2.2.2 [exp_Factor](#)

```
float exp_Factor
```

3.2.2.3 [finalVal](#)

```
unsigned char finalVal
```

3.2.2.4 [initialVal](#)

```
unsigned char initialVal
```

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

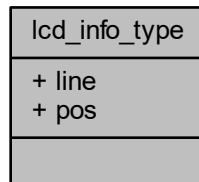
- [lib/q_learning/q_learning.h](#)

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:

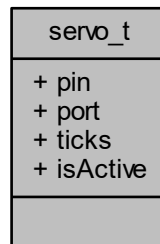
- lib/dwenguinoLCD/[DwenguinoLCD.h](#)

3.4 servo_t Struct Reference

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

3.4.2 Field Documentation

3.4.2.1 isActive

```
unsigned char isActive
```

0 = inactive

3.4.2.2 pin

`unsigned char pin`

pointer to the pin

3.4.2.3 port

`volatile unsigned char* port`

pointer to the port

3.4.2.4 ticks

`unsigned int ticks`

contains amount of prescaled ticks

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

- `lib/servo/servo.h`

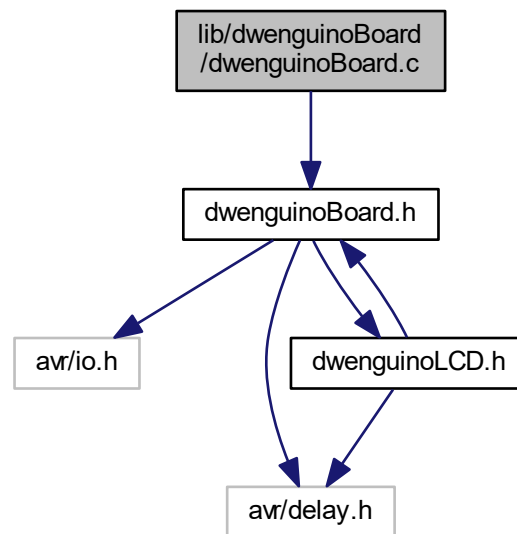
Chapter 4

File Documentation

4.1 lib/dwenguinoBoard/dwenguinoBoard.c File Reference

```
#include "dwenguinoBoard.h"
```

Include dependency graph for dwenguinoBoard.c:



Functions

- void [initBoard](#) (void)

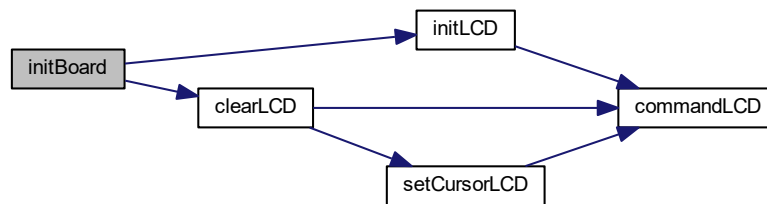
4.1.1 Function Documentation

4.1.1.1 initBoard()

```
void initBoard (
    void )
```

[dwenguinoBoard.c](#)

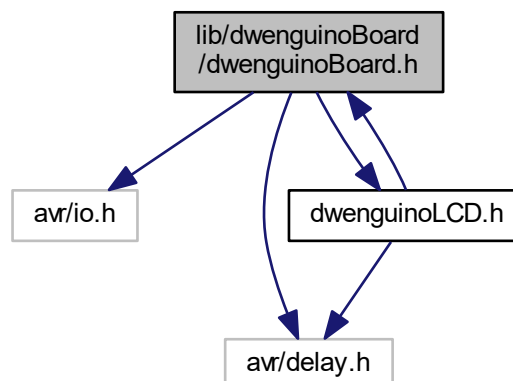
Created on: Jan 19, 2016 Author: Tom Here is the call graph for this function:



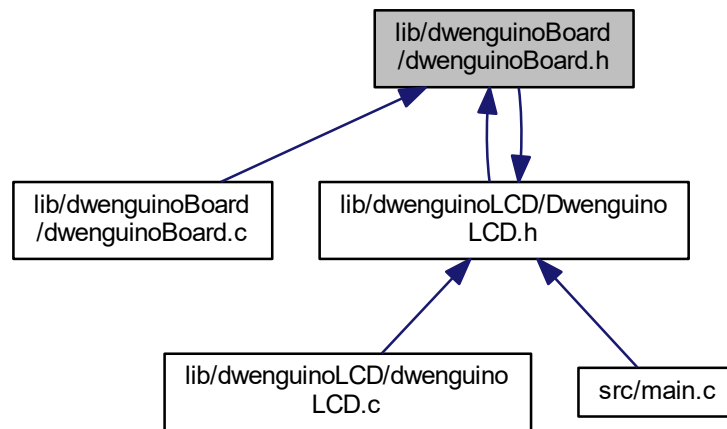
4.2 lib/dwenguinoBoard/dwenguinoBoard.h File Reference

This library contains pin assignments 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:



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)

4.2.1 Detailed Description

This library contains pin assignments and basic macros for Dwengino boards.

Author

Tom Neutens

Date

Jan 19, 2016

See also

<http://www.dwengo.org/tutorials>

4.2.2 Macro Definition Documentation

4.2.2.1 BYTE

```
#define BYTE unsigned char
```

4.2.2.2 FALSE

```
#define FALSE 0
```

4.2.2.3 HIGH

```
#define HIGH 1
```

4.2.2.4 INPUT

```
#define INPUT 0
```

4.2.2.5 LCD_BACKLIGHT_IN

```
#define LCD_BACKLIGHT_IN SET_PIN_LOW(DDRE, 3)
```

4.2.2.6 LCD_BACKLIGHT_OFF

```
#define LCD_BACKLIGHT_OFF SET_PIN_LOW(PORTE, 3)
```

4.2.2.7 LCD_BACKLIGHT_ON

```
#define LCD_BACKLIGHT_ON SET_PIN_HIGH(PORTE, 3)
```

4.2.2.8 LCD_BACKLIGHT_OUT

```
#define LCD_BACKLIGHT_OUT SET_PIN_HIGH(DDRE, 3)
```

4.2.2.9 LCD_DATA

```
#define LCD_DATA PORTA
```

4.2.2.10 LCD_DATA_DIR

```
#define LCD_DATA_DIR DDRA
```

4.2.2.11 LCD_EN_HIGH

```
#define LCD_EN_HIGH SET_PIN_HIGH(PORTE, 2)
```

4.2.2.12 LCD_EN_LOW

```
#define LCD_EN_LOW SET_PIN_LOW(PORTE, 2)
```

4.2.2.13 LCD_EN_OUT

```
#define LCD_EN_OUT SET_PIN_HIGH(DDRE, 2)
```

4.2.2.14 LCD_RS_HIGH

```
#define LCD_RS_HIGH SET_PIN_HIGH(PORTE, 0)
```

4.2.2.15 LCD_RS_LOW

```
#define LCD_RS_LOW SET_PIN_LOW(PORTE, 0)
```


4.2.2.16 LCD_RS_OUT

```
#define LCD_RS_OUT SET_PIN_HIGH(DDRE, 0)
```

4.2.2.17 LCD_RW_HIGH

```
#define LCD_RW_HIGH SET_PIN_HIGH(PORTE, 1)
```

4.2.2.18 LCD_RW_LOW

```
#define LCD_RW_LOW SET_PIN_LOW(PORTE, 1)
```

4.2.2.19 LCD_RW_OUT

```
#define LCD_RW_OUT SET_PIN_HIGH(DDRE, 1)
```

4.2.2.20 LED_OFF

```
#define LED_OFF(  
    LED ) (SET_PIN_LOW(PORTA, LED)
```

4.2.2.21 LED_ON

```
#define LED_ON(  
    LED ) SET_PIN_HIGH(PORTA, LED)
```

4.2.2.22 LEDS

```
#define LEDS PORTA
```

4.2.2.23 Leds_Dir

```
#define LEDS_DIR DDRA
```

4.2.2.24 LOW

```
#define LOW 0
```

4.2.2.25 MOTOR1_0_HIGH

```
#define MOTOR1_0_HIGH SET_PIN_HIGH(PORTC, 3)
```

4.2.2.26 MOTOR1_0_LOW

```
#define MOTOR1_0_LOW SET_PIN_LOW(PORTC, 3)
```

4.2.2.27 MOTOR1_1_HIGH

```
#define MOTOR1_1_HIGH SET_PIN_HIGH(PORTC, 4)
```

4.2.2.28 MOTOR1_1_LOW

```
#define MOTOR1_1_LOW SET_PIN_LOW(PORTC, 4)
```

4.2.2.29 MOTOR2_0_HIGH

```
#define MOTOR2_0_HIGH SET_PIN_HIGH(PORTC, 2)
```

4.2.2.30 MOTOR2_0_LOW

```
#define MOTOR2_0_LOW SET_PIN_LOW(PORTC, 2)
```

4.2.2.31 MOTOR2_1_HIGH

```
#define MOTOR2_1_HIGH SET_PIN_HIGH(PORTC, 5)
```

4.2.2.32 MOTOR2_1_LOW

```
#define MOTOR2_1_LOW SET_PIN_LOW(PORTC, 5)
```

4.2.2.33 OUTPUT

```
#define OUTPUT 1
```

4.2.2.34 PORT_HIGH

```
#define PORT_HIGH 0xFF
```

4.2.2.35 PORT_LOW

```
#define PORT_LOW 0x00
```

4.2.2.36 SERV01

```
#define SERV01 PORTC0
```

4.2.2.37 SERV02

```
#define SERV02 PORTC1
```

4.2.2.38 SET_BIT_HIGH

```
#define SET_BIT_HIGH(  
    REG,  
    BIT ) REG |= (1 << BIT)
```

4.2.2.39 SET_BIT_LOW

```
#define SET_BIT_LOW(  
    REG,  
    BIT ) REG &= ~(1 << BIT)
```

4.2.2.40 SET_PIN_HIGH

```
#define SET_PIN_HIGH(  
    PORT,  
    PIN ) PORT |= (1 << PIN)
```

4.2.2.41 SET_PIN_LOW

```
#define SET_PIN_LOW(  
    PORT,  
    PIN ) PORT &= ~(1 << PIN)
```

4.2.2.42 SW_C_HIGH

```
#define SW_C_HIGH SET_PIN_HIGH(PORTC, 6)
```

4.2.2.43 SW_C_IN

```
#define SW_C_IN SET_PIN_LOW(DDRC, 6)
```

4.2.2.44 SW_C_LOW

```
#define SW_C_LOW SET_PIN_LOW(PORTC, 6)
```

4.2.2.45 SW_C_OUT

```
#define SW_C_OUT SET_PIN_HIGH(DDRC, 6)
```

4.2.2.46 SW_E_HIGH

```
#define SW_E_HIGH SET_PIN_HIGH(PORTE, 6)
```

4.2.2.47 SW_E_IN

```
#define SW_E_IN SET_PIN_LOW(DDRE, 6)
```

4.2.2.48 SW_E_LOW

```
#define SW_E_LOW SET_PIN_LOW(PORTE, 6)
```

4.2.2.49 SW_E_OUT

```
#define SW_E_OUT SET_PIN_HIGH(DDRE, 6)
```

4.2.2.50 SW_N_HIGH

```
#define SW_N_HIGH SET_PIN_HIGH(PORTE, 7)
```

4.2.2.51 SW_N_IN

```
#define SW_N_IN SET_PIN_LOW(DDRE, 6)
```

4.2.2.52 SW_N_LOW

```
#define SW_N_LOW SET_PIN_LOW(PORTE, 7)
```

4.2.2.53 SW_N_OUT

```
#define SW_N_OUT SET_PIN_HIGH(DDRE, 6)
```

4.2.2.54 SW_S_HIGH

```
#define SW_S_HIGH SET_PIN_HIGH(PORTE, 5)
```

4.2.2.55 SW_S_IN

```
#define SW_S_IN SET_PIN_LOW(DDRE, 5)
```

4.2.2.56 SW_S_LOW

```
#define SW_S_LOW SET_PIN_LOW(PORTE, 5)
```

4.2.2.57 SW_S_OUT

```
#define SW_S_OUT SET_PIN_HIGH(DDRE, 5)
```

4.2.2.58 SW_W_HIGH

```
#define SW_W_HIGH SET_PIN_HIGH(PORTE, 4)
```

4.2.2.59 SW_W_IN

```
#define SW_W_IN SET_PIN_LOW(DDRE, 4)
```

4.2.2.60 SW_W_LOW

```
#define SW_W_LOW SET_PIN_LOW(PORTE, 4)
```

4.2.2.61 SW_W_OUT

```
#define SW_W_OUT SET_PIN_HIGH(DDRE, 4)
```

4.2.2.62 TRUE

```
#define TRUE 1
```

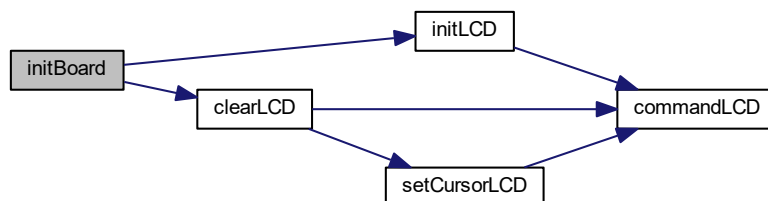
4.2.3 Function Documentation

4.2.3.1 initBoard()

```
void initBoard (  
    void )
```

[dwenguinoBoard.c](#)

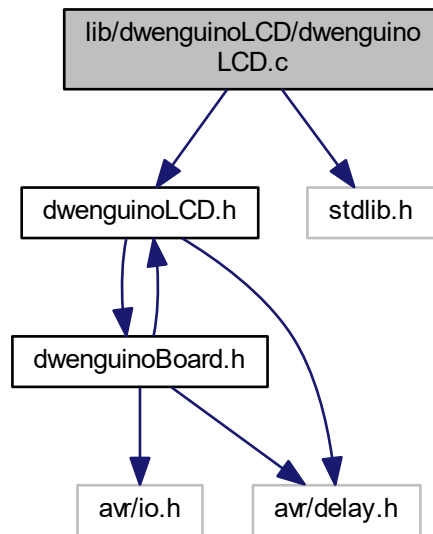
Created on: Jan 19, 2016 Author: Tom Here is the call graph for this function:



4.3 lib/dwenguinoLCD/dwenguinoLCD.c File Reference

```
#include "dwenguinoLCD.h"  
#include <stdlib.h>
```

Include dependency graph for dwenguinoLCD.c:



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 position then moves the cursor to the next position.
- void [printCharToLCD](#) (const char c, [BYTE](#) l, [BYTE](#) p)
Print character to LCD Prints a character to a specified position.
- void [appendStringToLCD_](#) (const char *message)
- void [printStringToLCD](#) (char *message, [BYTE](#) l, [BYTE](#) p)
- void [appendStringToLCDcharptr](#) (char *message)
- 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 after 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](#)

4.3.1 Function Documentation

4.3.1.1 appendCharToLCD()

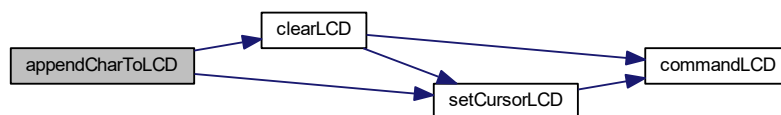
```
void appendCharToLCD (
    const char c )
```

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

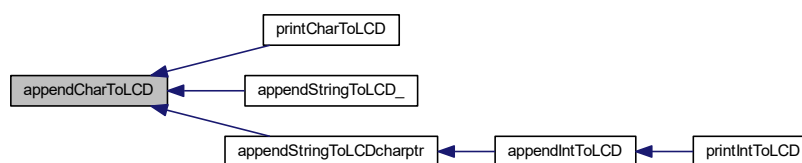
Parameters

<code>c</code>	the character to append
----------------	-------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.2 appendIntToLCD()

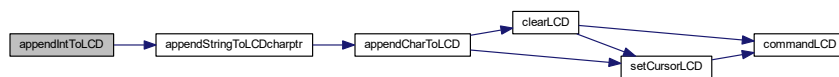
```
void appendIntToLCD (
    int i )
```

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

Parameters

<i>i</i>	the integer to print
----------	----------------------

Here is the call graph for this function:



Here is the caller graph for this function:

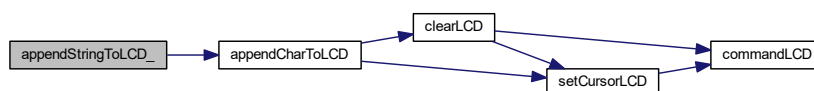


4.3.1.3 appendStringToLCD_()

```

void appendStringToLCD_ (
    const char * message )
  
```

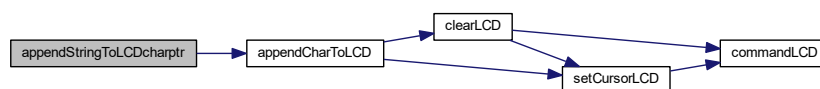
Here is the call graph for this function:



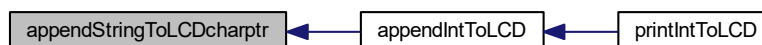
4.3.1.4 appendStringToLCDcharptr()

```
void appendStringToLCDcharptr (  
    char * message )
```

Here is the call graph for this function:



Here is the caller graph for this function:

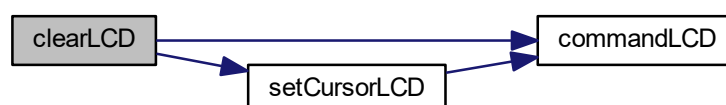


4.3.1.5 clearLCD()

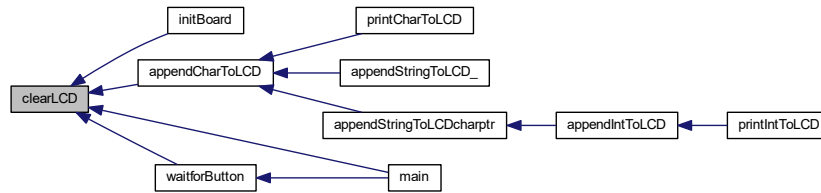
```
void clearLCD (  
    void )
```

clears the LCD screen This function removes all the content from the LCD screen

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.6 commandLCD()

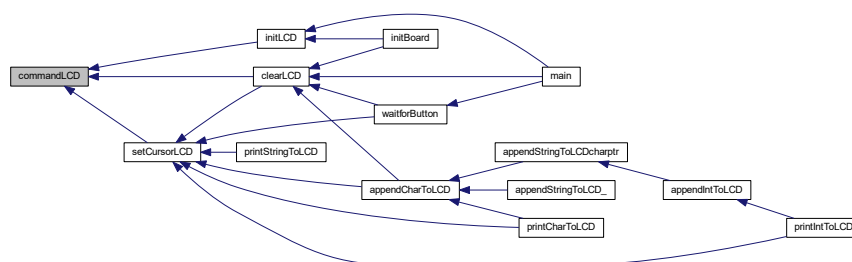
```
void commandLCD (
    const BYTE c )
```

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

<code>c</code>	command to be transferred to the LCD
----------------	--------------------------------------

Here is the caller graph for this function:

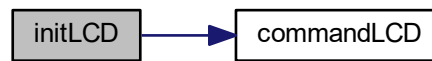


4.3.1.7 initLCD()

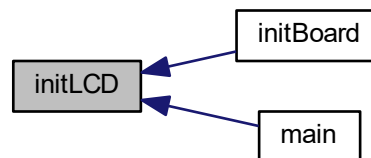
```
void initLCD (
    void )
```

initializes the LCD screen This function sets up the lcd for displaying the data we will send

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.8 printCharToLCD()

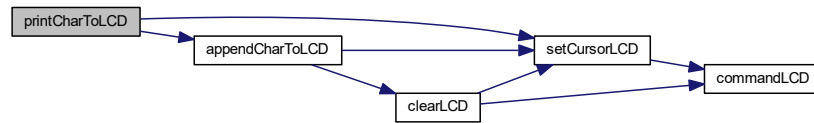
```
void printCharToLCD (
    const char s,
    BYTE l,
    BYTE p )
```

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

Parameters

<i>s</i>	the character to print
<i>l</i>	the line
<i>p</i>	the position in the line

Here is the call graph for this function:



4.3.1.9 printIntToLCD()

```

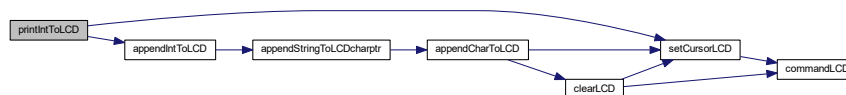
void printIntToLCD (
    int i,
    BYTE l,
    BYTE p )
  
```

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

Parameters

<i>i</i>	the integer to print
<i>l</i>	the line
<i>p</i>	the position in the line

Here is the call graph for this function:

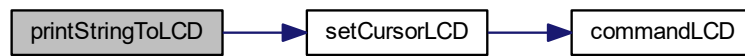


4.3.1.10 printStringToLCD()

```

void printStringToLCD (
    char * message,
    BYTE l,
    BYTE p )
  
```

Here is the call graph for this function:



4.3.1.11 setCursorLCD()

```

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

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

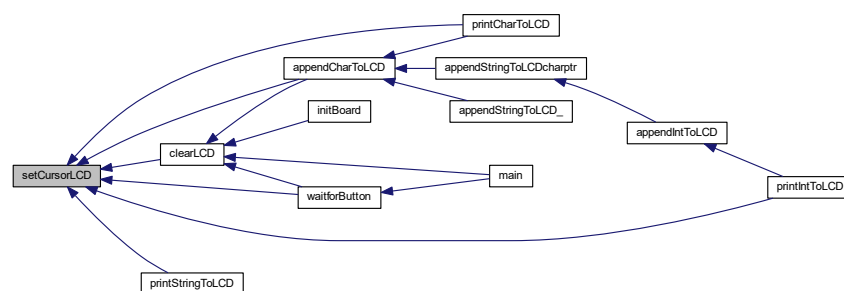
Parameters

<i>l</i>	line number
<i>p</i>	position in line

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2 Variable Documentation

4.3.2.1 lcd_info

```
struct lcd_info_type lcd_info
```

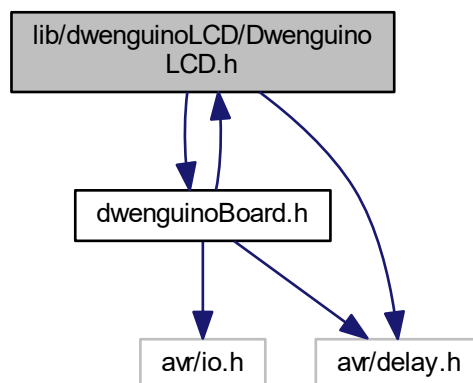
[dwenguinoLCD.c](#)

Created on: Jan 19, 2016 Author: Tom

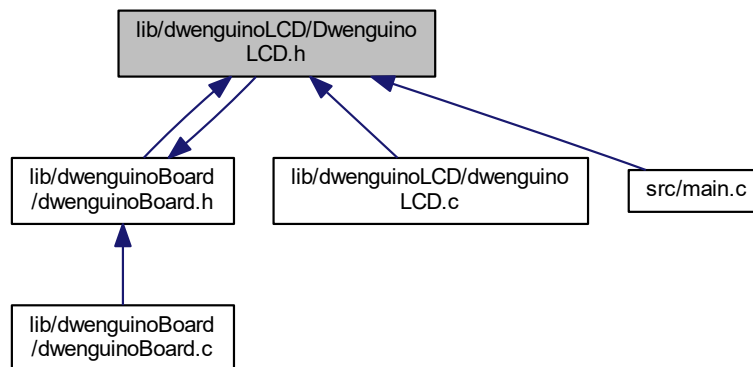
4.4 lib/dwenguinoLCD/DwenguinoLCD.h File Reference

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:



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 position 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 after 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`

4.4.1 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.

4.4.2 Macro Definition Documentation

4.4.2.1 appendStringToLCD

```
#define appendStringToLCD(  
    message ) appendStringToLCD_((const char*)(message));
```

4.4.2.2 backlightOff

```
#define backlightOff( ) (LCD_BACKLIGHT_OFF)
```

4.4.2.3 backlightOn

```
#define backlightOn( ) (LCD_BACKLIGHT_ON)
```

4.4.2.4 LCD_HEIGHT

```
#define LCD_HEIGHT 2
```

4.4.2.5 LCD_LASTLINE

```
#define LCD_LASTLINE (LCD_HEIGHT - 1)
```

4.4.2.6 LCD_LASTPOS

```
#define LCD_LASTPOS (LCD_WIDTH - 1)
```

4.4.2.7 LCD_WIDTH

```
#define LCD_WIDTH 16
```

4.4.3 Function Documentation

4.4.3.1 appendCharToLCD()

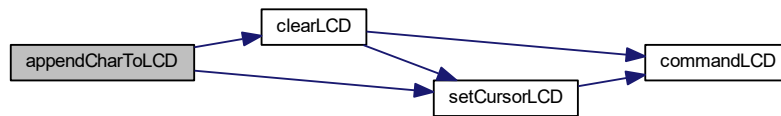
```
void appendCharToLCD (  
    const char c )
```

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

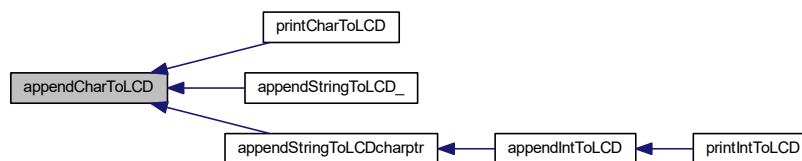
Parameters

<i>c</i>	the character to append
----------	-------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.2 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 after the printed integer.

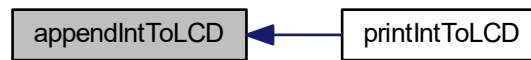
Parameters

<i>i</i>	the integer to print
----------	----------------------

Here is the call graph for this function:



Here is the caller graph for this function:

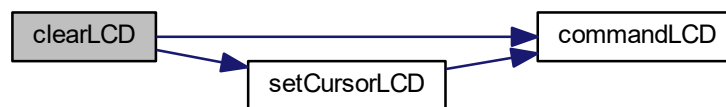


4.4.3.3 clearLCD()

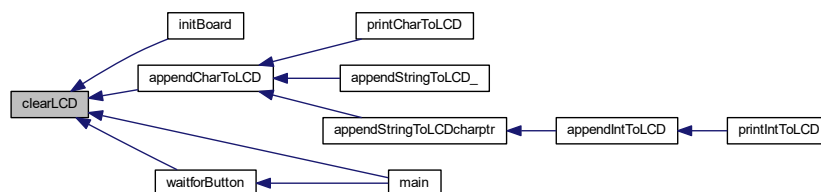
```
void clearLCD (
    void )
```

clears the LCD screen This function removes all the content from the LCD screen

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.4 commandLCD()

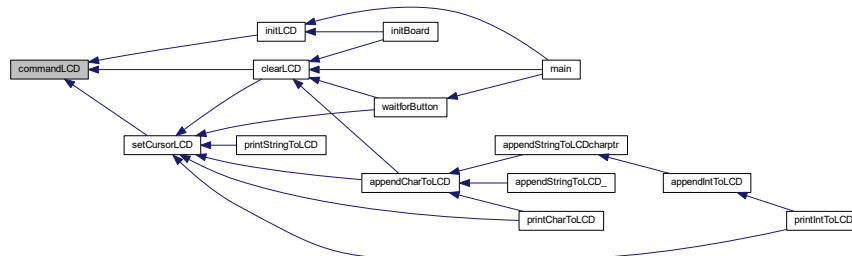
```
void commandLCD (
    const BYTE c )
```

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

<code>c</code>	command to be transfered to the LCD
----------------	-------------------------------------

Here is the caller graph for this function:

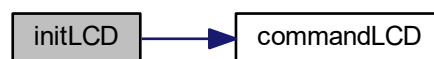


4.4.3.5 initLCD()

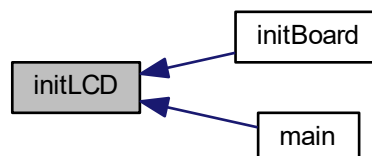
```
void initLCD (
    void )
```

initializes the LCD screen This function sets up the lcd for displaying the data we will send

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.6 printCharToLCD()

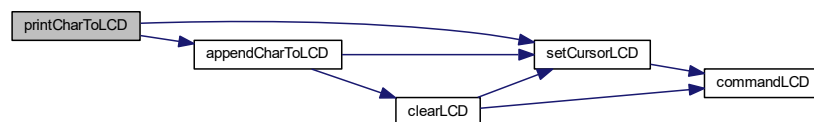
```
void printCharToLCD (
    const char s,
    BYTE l,
    BYTE p )
```

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

Parameters

<i>s</i>	the character to print
<i>l</i>	the line
<i>p</i>	the position in the line

Here is the call graph for this function:



4.4.3.7 printIntToLCD()

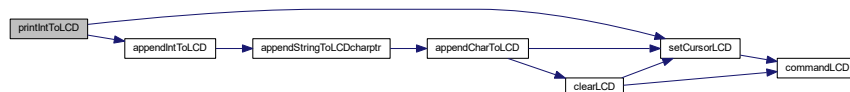
```
void printIntToLCD (
    int i,
    BYTE l,
    BYTE p )
```

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

Parameters

<i>i</i>	the integer to print
<i>l</i>	the line
<i>p</i>	the position in the line

Here is the call graph for this function:



4.4.3.8 setCursorLCD()

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

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

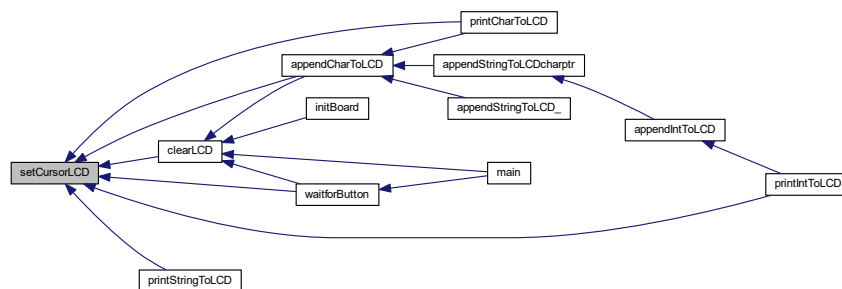
Parameters

<i>l</i>	line number
<i>p</i>	position in line

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.4 Variable Documentation

4.4.4.1 lcd_info

```
struct lcd_info_type lcd_info
```

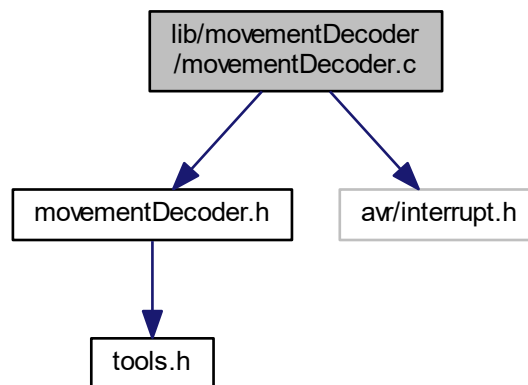
[dwenguinoLCD.c](#)

Created on: Jan 19, 2016 Author: Tom

4.5 lib/movementDecoder/movementDecoder.c File Reference

```
#include "movementDecoder.h"
#include <avr/interrupt.h>
```

Include dependency graph for movementDecoder.c:



Functions

- void `initMovementDecoder ()`
Initializes the optosensor. This function sets up the optosensors 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.
- `ISR (DECODER1_INT)`
- `ISR (DECODER2_INT)`

Variables

- `BYTE prevAngleState = 0`
- int `dirCount = 0`
Value = movement, sign = direction.

4.5.1 Function Documentation

4.5.1.1 checkMovementDir()

```
void checkMovementDir ( ) [inline]
```

Determines the movement direction of the agent. This function uses the current and previous state of the optosensors to determine the direction.

Here is the caller graph for this function:

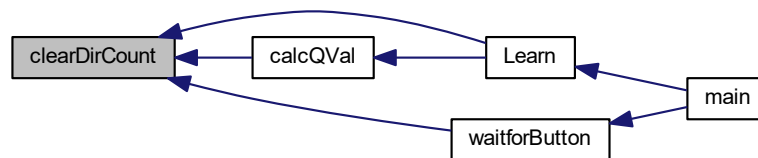


4.5.1.2 clearDirCount()

```
void clearDirCount ( )
```

Clears the dirCount value.

Here is the caller graph for this function:

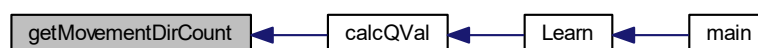


4.5.1.3 getMovementDirCount()

```
int getMovementDirCount ( )
```

Returns the dirCount value.

Here is the caller graph for this function:



4.5.1.4 initMovementDecoder()

```
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.

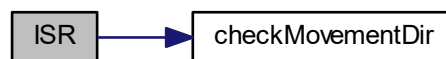
Here is the caller graph for this function:



4.5.1.5 ISR() [1/2]

```
ISR (
    DECODER1_INT )
```

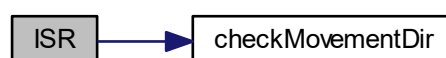
Here is the call graph for this function:



4.5.1.6 ISR() [2/2]

```
ISR (
    DECODER2_INT )
```

Here is the call graph for this function:



4.5.2 Variable Documentation

4.5.2.1 dirCount

```
int dirCount = 0
```

Value = movement, sign = direction.

4.5.2.2 prevAngleState

```
BYTE prevAngleState = 0
```

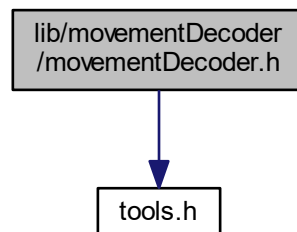
The first time prevAngleState is unknown, represented by the MSB of prevAngleState BIT 7: (MSB) bool prev↔AngleStateKnown BIT 0-1: the binary value of prevAngleState

4.6 lib/movementDecoder/movementDecoder.h File Reference

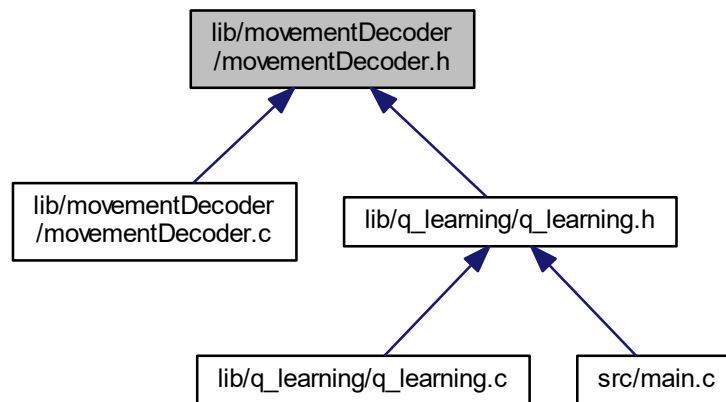
This library contains functions to determine the direction by using 2 optosensores. Before using the movement↔Decoder's methods, call the [initMovementDecoder\(\)](#) function.

```
#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.

4.6.1 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

4.6.2 Macro Definition Documentation

4.6.2.1 DECODER1

```
#define DECODER1 PORTD0
```

connect the DO pin of optosensor 1 to this port

4.6.2.2 DECODER1_DATA

```
#define DECODER1_DATA PIND
```

4.6.2.3 DECODER1_DIR

```
#define DECODER1_DIR DDRD
```

4.6.2.4 DECODER1_INT

```
#define DECODER1_INT INT0_vect
```

the interrupt used by optosensor 1

4.6.2.5 DECODER1_PORT

```
#define DECODER1_PORT PORTD
```

4.6.2.6 DECODER2

```
#define DECODER2 PORTD1
```

connect the DO pin of optosensor 1 to this port

4.6.2.7 DECODER2_DATA

```
#define DECODER2_DATA PIND
```

4.6.2.8 DECODER2_DIR

```
#define DECODER2_DIR DDRD
```

4.6.2.9 DECODER2_INT

```
#define DECODER2_INT INT1_vect
```

the interrupt used by optosensor2

4.6.2.10 DECODER2_PORT

```
#define DECODER2_PORT PORTD
```

4.6.2.11 GRAY_TO_BIN

```
#define GRAY_TO_BIN(  
    GRAY ) (GRAY & (1 << 1)) ? (TOGGLE_BIT(GRAY, 0)) : (0)
```

4.6.3 Function Documentation

4.6.3.1 checkMovementDir()

```
void checkMovementDir ( ) [inline]
```

Determines the movement direction of the agent. This function uses the current and previous state of the optosensors to determine the direction.

Here is the caller graph for this function:

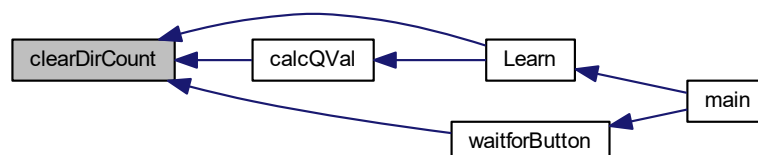


4.6.3.2 clearDirCount()

```
void clearDirCount ( )
```

Clears the `dirCount` value.

Here is the caller graph for this function:

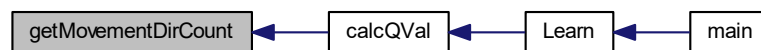


4.6.3.3 getMovementDirCount()

```
int getMovementDirCount ( )
```

Returns the dirCount value.

Here is the caller graph for this function:

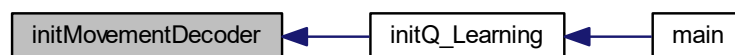


4.6.3.4 initMovementDecoder()

```
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.

Here is the caller graph for this function:

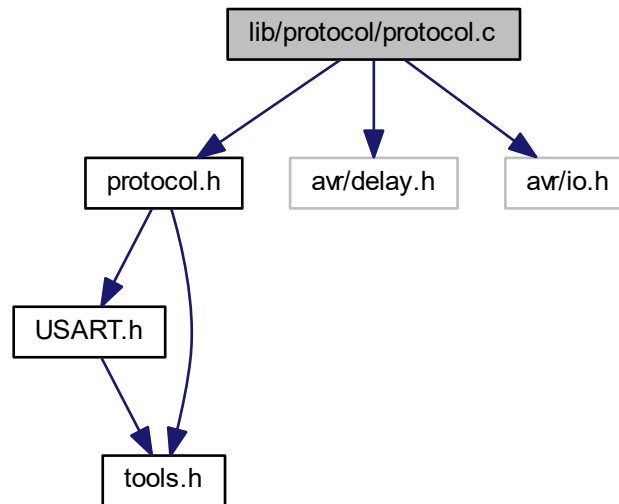


4.7 lib/protocol/protocol.c File Reference

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

```
#include <avr/io.h>
```

Include dependency graph for protocol.c:



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.

4.7.1 Function Documentation

4.7.1.1 `init_PositionProtocol()`

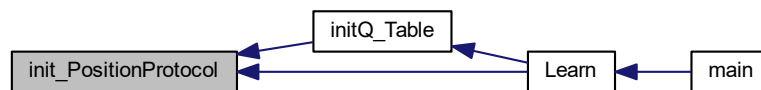
```
void init_PositionProtocol (
    unsigned char position[2],
    struct Dataobj * posObj )
```

Initializes a position data object.

Parameters

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

Here is the caller graph for this function:

4.7.1.2 `init_QvalProtocol()`

```

void init_QvalProtocol (
    int QVal,
    struct Dataobj * Q_Data )

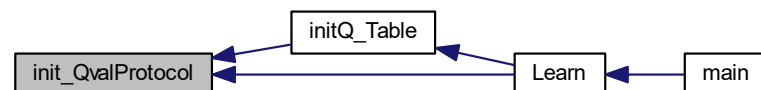
```

Initializes a Q_value data object.

Parameters

<i>Qval</i>	the Q value
<i>Q_Data</i>	passes a reference to the Dataobj that needs to be initialized

Here is the caller graph for this function:

4.7.1.3 `init_SizeProtocol()`

```

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

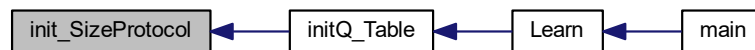
```

Initializes a size data object.

Parameters

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

Here is the caller graph for this function:



4.7.1.4 transmit_data()

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

Transmits the datastruct.

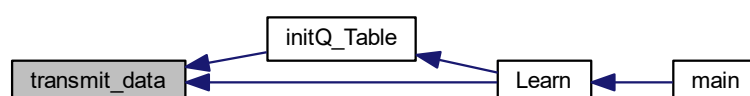
Parameters

<i>dObj</i>	passes a reference to the Dataobj that needs to be transmitted
-------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



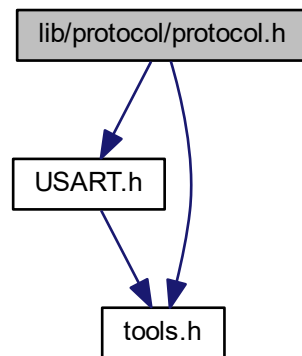
4.8 lib/protocol/protocol.h File Reference

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

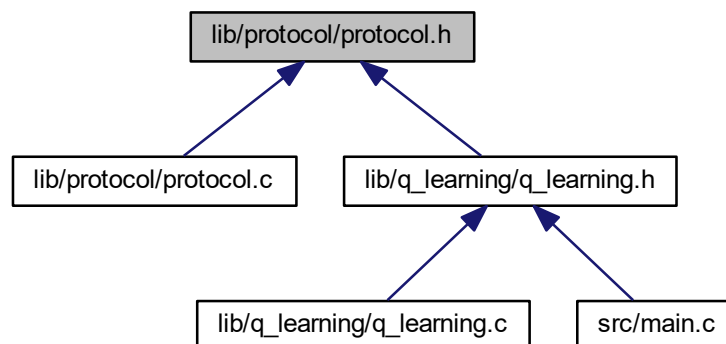
```
#include "USART.h"
```

```
#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.

4.8.1 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

4.8.2 Macro Definition Documentation

4.8.2.1 CHAR_SIZE

```
#define CHAR_SIZE (sizeof(char))
```

4.8.2.2 END_TYPE

```
#define END_TYPE 'e'
```

4.8.2.3 FLOAT_SIZE

```
#define FLOAT_SIZE (sizeof(float))
```

4.8.2.4 INT_SIZE

```
#define INT_SIZE (sizeof(int))
```

4.8.2.5 POSITION_TYPE

```
#define POSITION_TYPE 'p'
```

4.8.2.6 Q_VALUE_TYPE

```
#define Q_VALUE_TYPE 'q'
```

4.8.2.7 START_TYPE

```
#define START_TYPE 's'
```

4.8.2.8 TABLE_TYPE

```
#define TABLE_TYPE 't'
```

4.8.2.9 UCHAR_SIZE

```
#define UCHAR_SIZE (sizeof(unsigned char))
```

4.8.3 Function Documentation

4.8.3.1 init_PositionProtocol()

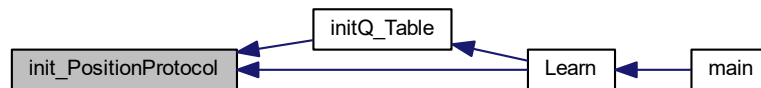
```
void init_PositionProtocol (
    unsigned char position[2],
    struct Dataobj * posObj )
```

Initializes a position data object.

Parameters

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

Here is the caller graph for this function:

4.8.3.2 `init_QvalProtocol()`

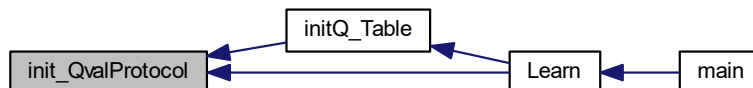
```
void init_QvalProtocol (
    int QVal,
    struct Dataobj * Q_Data )
```

Initializes a Q_value data object.

Parameters

<i>Qval</i>	the Q value
<i>Q_Data</i>	passes a reference to the Dataobj that needs to be initialized

Here is the caller graph for this function:

4.8.3.3 `init_SizeProtocol()`

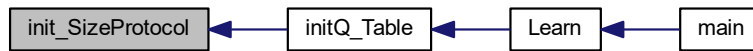
```
void init_SizeProtocol (
    unsigned char size[2],
    struct Dataobj * sizeObj )
```

Initializes a size data object.

Parameters

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

Here is the caller graph for this function:



4.8.3.4 transmit_data()

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

Transmits the datastruct.

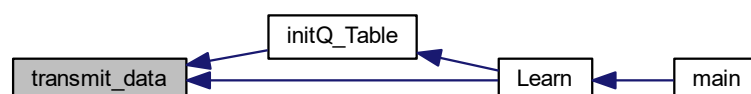
Parameters

<i>dObj</i>	passes a reference to the Dataobj that needs to be transmitted
-------------	--

Here is the call graph for this function:



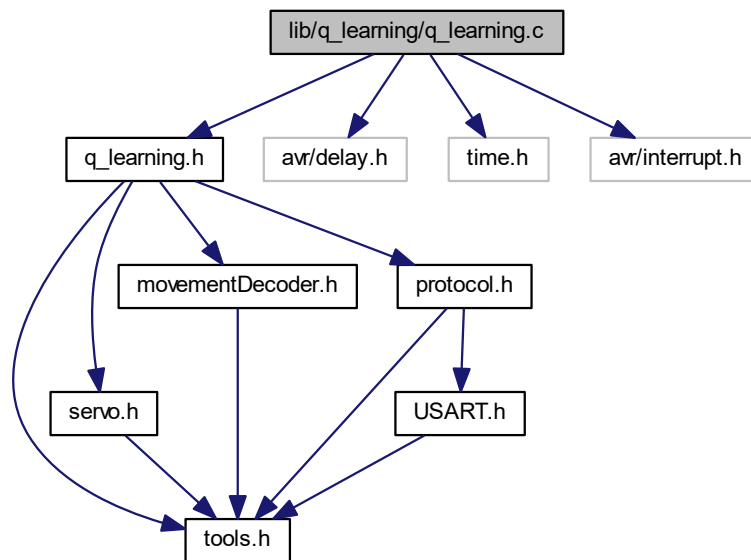
Here is the caller graph for this function:



4.9 lib/q_learning/q_learning.c File Reference

```
#include "q_learning.h"
#include <avr/delay.h>
#include <time.h>
#include <avr/interrupt.h>
```

Include dependency graph for q_learning.c:



Functions

- void [initQ_Learning](#) ()
Initializes the Q-learning algorithm. This function sets up the pins and the interrupt for the button. It also initializes the needed components for the Q-learning algorithm ([servo.h](#), [movementDecoder.h](#))
- void [initQ_Table](#) ()
Initializes the Q_Table with random values.
- void [Learn](#) ()
- 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.
- void [waitBTN](#) ()
- 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.
- [ISR](#) (INT5_vect)

Variables

- unsigned char `dataByte`
- int ** `Q_Table`
dynamically allocated Q_Table
- int * `temp`
pointer used to allocate Q_Table
- unsigned char `prevServoStateArr` [NUMB_SERVOS] = {STARTSTATES1, STARTSTATES2}
- unsigned char `nextServoStateArr` [NUMB_SERVOS] = {STARTSTATES1, STARTSTATES2}
- int `numbliterations` = 0
the amount of iterations the agent has done through the learn method
- struct `Formula_Parameters_t` `parameters` [NUMB_PARAMETERS]

4.9.1 Function Documentation

4.9.1.1 `adjustParameters()`

```
void adjustParameters ( )
```

Adjusts the parameters in the Q-learning formule.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.1.2 `calcMaxActionIndex()`

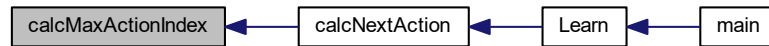
```
unsigned char calcMaxActionIndex (
    unsigned char state )
```

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

Parameters

<i>state</i>	this is the state of the robot
--------------	--------------------------------

Here is the caller graph for this function:

**4.9.1.3 calcMaxQval()**

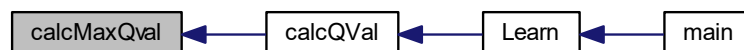
```
int calcMaxQval (
    unsigned char state )
```

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

Parameters

<i>state</i>	this is the state of the robot
--------------	--------------------------------

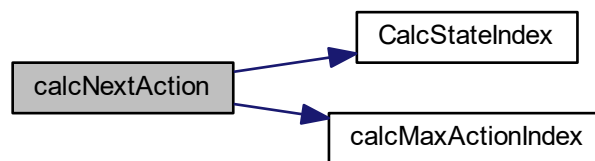
Here is the caller graph for this function:

**4.9.1.4 calcNextAction()**

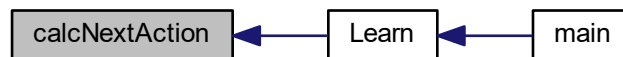
```
unsigned char calcNextAction ( )
```

This function calculates the next action. It should be only called by the [learn\(\)](#) method.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.1.5 calcParameter()

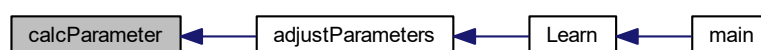
```
void calcParameter (
    struct Formula_Parameters_t * paramVal )
```

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

Parameters

<i>*paramVal</i>	this is the parameter that will be updated
------------------	--

Here is the caller graph for this function:

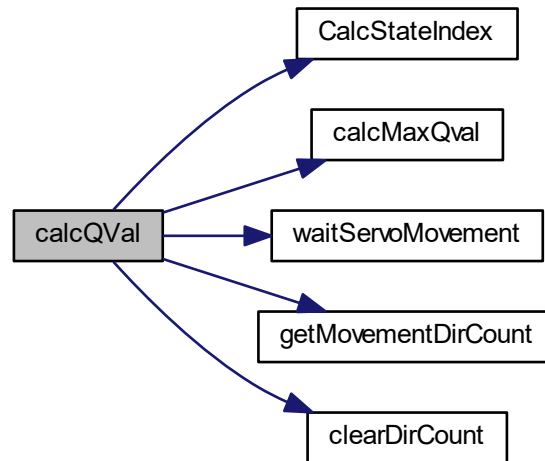


4.9.1.6 calcQVal()

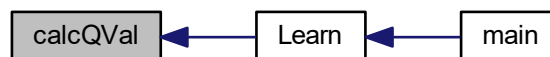
```
int calcQVal ( )
```

This function calculates the Q-value. It should be only called by the [learn\(\)](#) method.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.1.7 CalcStateIndex()

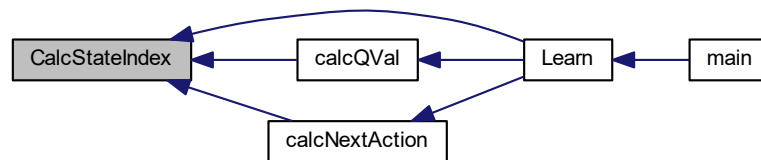
```
unsigned char CalcStateIndex (
    unsigned char stateArr[ ] )
```

Calculates the state index.

Parameters

<code>stateArr[]</code>	this array contains the stateval of each servo
-------------------------	--

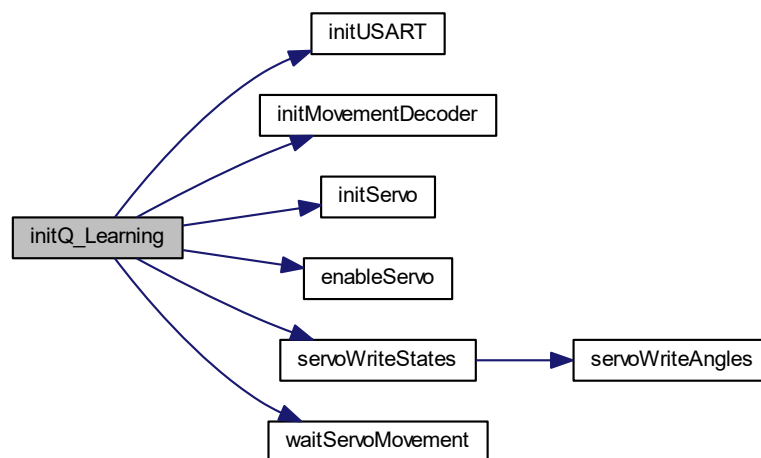
Here is the caller graph for this function:

4.9.1.8 `initQ_Learning()`

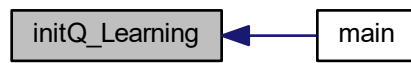
```
void initQ_Learning ( )
```

Initializes the Q-learning algorithm. This function sets up the pins and the interrupt for the button. It also initializes the needed components for the Q-learning algorithm ([servo.h](#), [movementDecoder.h](#))

Here is the call graph for this function:



Here is the caller graph for this function:

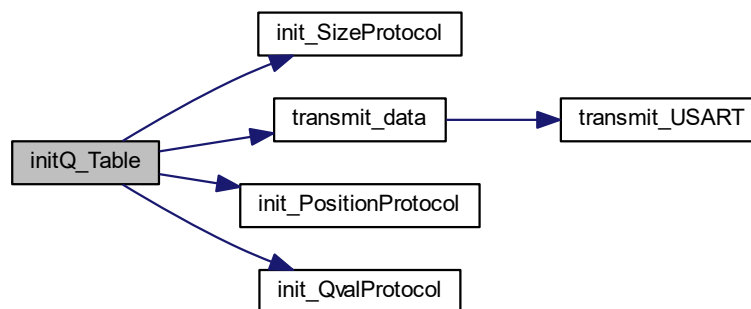


4.9.1.9 initQ_Table()

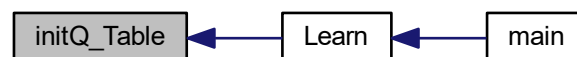
```
void initQ_Table ( )
```

Initializes the Q_Table with random values.

Here is the call graph for this function:



Here is the caller graph for this function:



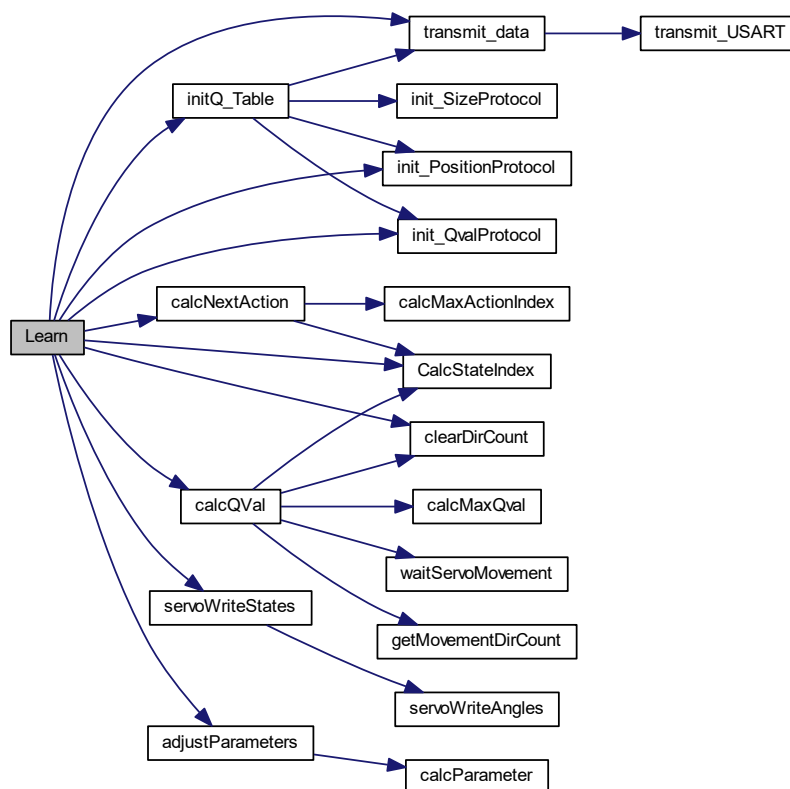
4.9.1.10 ISR()

```
ISR (
    INT5_vect )
```

4.9.1.11 Learn()

```
void Learn ( )
```

Here is the call graph for this function:



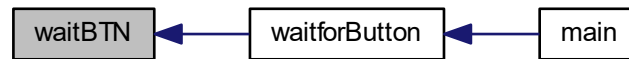
Here is the caller graph for this function:



4.9.1.12 waitBTN()

```
void waitBTN ( )
```

Here is the caller graph for this function:



4.9.2 Variable Documentation

4.9.2.1 dataByte

```
unsigned char dataByte
```

this dataByte contains data of multiple parameters BIT 7: Bool btnPressed (true when the button is pressed) BIT 0-3: Nibble prevAction (max amount of servos is 8 \rightarrow 2 actions per servo = 16 = 2^4 = Nibble)

4.9.2.2 nextServoStateArr

```
unsigned char nextServoStateArr[NUMB_SERVOS] = {STARTSTATES1, STARTSTATES2}
```

4.9.2.3 numblterations

```
int numblterations = 0
```

the amount of iterations the agent has done through the learn method

4.9.2.4 parameters

```
struct Formula_Parameters_t parameters[NUMB_PARAMETERS]
```

4.9.2.5 prevServoStateArr

```
unsigned char prevServoStateArr[NUMB_SERVOS] = {STARTSTATES1, STARTSTATES2}
```

4.9.2.6 Q_Table

```
int** Q_Table
```

dynamically allocated Q_Table

4.9.2.7 temp

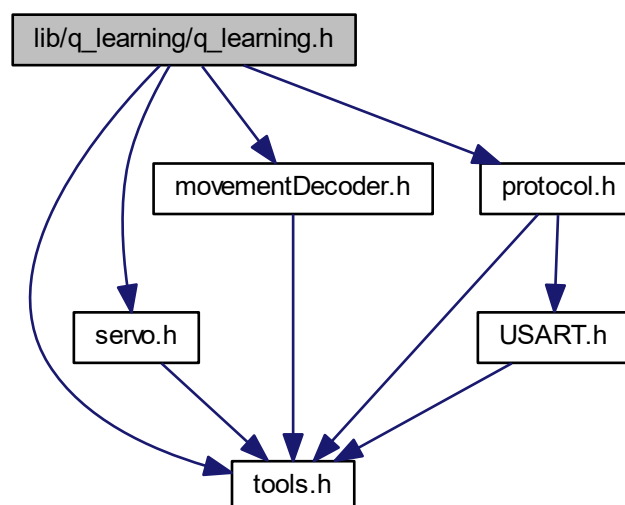
```
int* temp
```

pointer used to allocate Q_Table

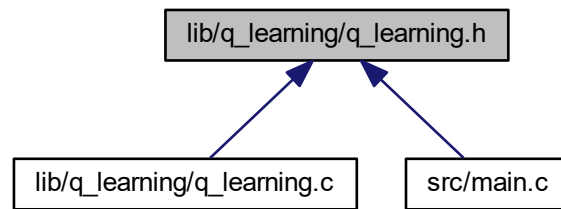
4.10 lib/q_learning/q_learning.h File Reference

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:
```



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



Data Structures

- struct [Formula_Parameters_t](#)

This struct contains the parameters of the Q-learning formula. Some other functions are specifically 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_LEARNING_RATE 40`
- `#define GREEDY_FACTOR_INDEX 1`
- `#define INIT_GREEDY_FACTOR 90`
epsilon 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 PREVACION_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 initializes 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.

4.10.1 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

4.10.2 Macro Definition Documentation

4.10.2.1 BTN_S

```
#define BTN_S PORTE5
```

4.10.2.2 BTN_S_DIR

```
#define BTN_S_DIR DDRE
```

4.10.2.3 BTN_S_INT

```
#define BTN_S_INT INT5
```

4.10.2.4 BTN_S_PORT

```
#define BTN_S_PORT PORTE
```

4.10.2.5 BTPRESSED_INDEX

```
#define BTPRESSED_INDEX 7
```


4.10.2.6 CALC_STATE_INDEX

```
#define CALC_STATE_INDEX(  
    STATES1,  
    STATES2 ) (STATES1*4 + STATES2)
```

4.10.2.7 DELAYVAL

```
#define DELAYVAL 700
```

value in ms that is used to wait until the robot stopped moving

4.10.2.8 DISCOUNT_FACTOR_INDEX

```
#define DISCOUNT_FACTOR_INDEX 2
```

4.10.2.9 ENABLE_BTN

```
#define ENABLE_BTN (SET_BIT_HIGH(EIMSK, BTN_S_INT))
```

4.10.2.10 EULER

```
#define EULER (double)2.71828
```

4.10.2.11 EXP_ENABLED

```
#define EXP_ENABLED 1
```

bool; 1 if the

4.10.2.12 EXP_FACTOR_DISCOUNT

```
#define EXP_FACTOR_DISCOUNT (float)1
```

4.10.2.13 EXP_FACTOR_LEARNING

```
#define EXP_FACTOR_LEARNING (float)1
```

4.10.2.14 EXP_GREEDY_FACTOR

```
#define EXP_GREEDY_FACTOR (float)1
```

4.10.2.15 EXP_REWARD_FACTOR

```
#define EXP_REWARD_FACTOR (double)0.1
```

the exponential reward factor

4.10.2.16 FINAL_DISCOUNT_FACTOR

```
#define FINAL_DISCOUNT_FACTOR 50
```

4.10.2.17 FINAL_GREEDY_FACTOR

```
#define FINAL_GREEDY_FACTOR 0
```

4.10.2.18 FINAL_LEARNING_RATE

```
#define FINAL_LEARNING_RATE 40
```

4.10.2.19 GETSIZE

```
#define GETSIZE(  
    QTABLE ) (sizeof(QTABLE[0])/ sizeof(QTABLE[0][0]))
```

4.10.2.20 GREEDY_FACTOR_INDEX

```
#define GREEDY_FACTOR_INDEX 1
```

4.10.2.21 INIT_DISCOUNT_FACTOR

```
#define INIT_DISCOUNT_FACTOR 99
```

gamma 0.99

4.10.2.22 INIT_GREEDY_FACTOR

```
#define INIT_GREEDY_FACTOR 90
```

epsilon 0.1

4.10.2.23 INIT_LEARNING_RATE

```
#define INIT_LEARNING_RATE 90
```

alpha 0.5

4.10.2.24 LEARNING_RATE_INDEX

```
#define LEARNING_RATE_INDEX 0
```

4.10.2.25 MAX_COUNT

```
#define MAX_COUNT 80
```

4.10.2.26 MAX_FACTOR

```
#define MAX_FACTOR 100
```

fixed point notation is used → this is maximum

4.10.2.27 NUMB_ACTIONS

```
#define NUMB_ACTIONS (2*NUMB_SERVOS)
```

4.10.2.28 NUMB_PARAMETERS

```
#define NUMB_PARAMETERS 3
```

4.10.2.29 NUMB_STATES

```
#define NUMB_STATES ((unsigned char)pow(NUMB_SERVO_STATES, NUMB_SERVOS))
```

4.10.2.30 OPT_PER_SERVO

```
#define OPT_PER_SERVO 2
```

options of actions per servo

4.10.2.31 PREVACTION_INDEX_NIBBLE

```
#define PREVACTION_INDEX_NIBBLE 0
```

4.10.2.32 RADIX

```
#define RADIX 100
```

fixed point notation

4.10.2.33 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

4.10.2.34 STARTSTATES1

```
#define STARTSTATES1 0
```

this value should be < than NUMB_STATES

4.10.2.35 STARTSTATES2

```
#define STARTSTATES2 3
```

4.10.2.36 STARTSTATES3

```
#define STARTSTATES3 0
```

4.10.2.37 TRESHMOVFW

```
#define TRESHMOVFW 2
```

threshold value for moving forward

4.10.3 Function Documentation

4.10.3.1 adjustParameters()

```
void adjustParameters ( )
```

Adjusts the parameters in the Q-learning formule.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.2 `calcMaxActionIndex()`

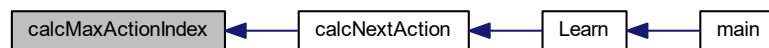
```
unsigned char calcMaxActionIndex (  
    unsigned char state )
```

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

Parameters

<code>state</code>	this is the state of the robot
--------------------	--------------------------------

Here is the caller graph for this function:



4.10.3.3 `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

<code>state</code>	this is the state of the robot
--------------------	--------------------------------

Here is the caller graph for this function:

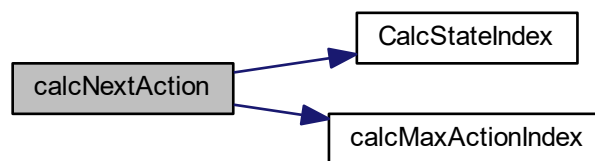


4.10.3.4 calcNextAction()

```
unsigned char calcNextAction ( )
```

This function calculates the next action. It should be only called by the [learn\(\)](#) method.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.5 calcParameter()

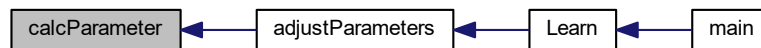
```
void calcParameter (
    struct Formula_Parameters_t * paramVal )
```

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

Parameters

<i>*paramVal</i>	this is the parameter that will be updated
------------------	--

Here is the caller graph for this function:

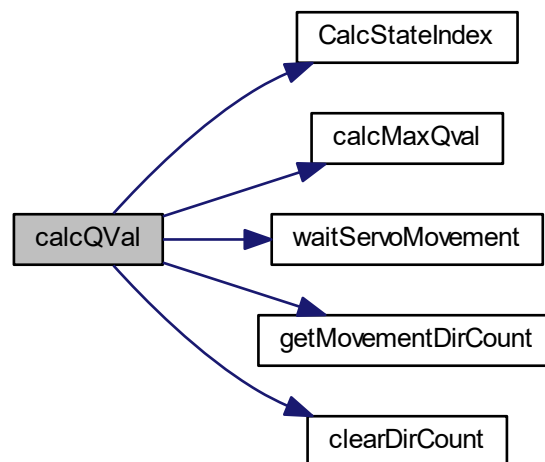


4.10.3.6 calcQVal()

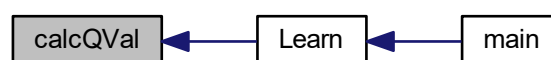
```
int calcQVal ( )
```

This function calculates the Q-value. It should be only called by the [learn\(\)](#) method.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.7 CalcStateIndex()

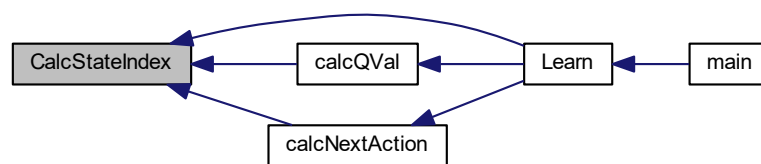
```
unsigned char CalcStateIndex (  
    unsigned char stateArr[] )
```

Calculates the state index.

Parameters

<code>stateArr[]</code>	this array contains the stateval of each servo
-------------------------	--

Here is the caller graph for this function:

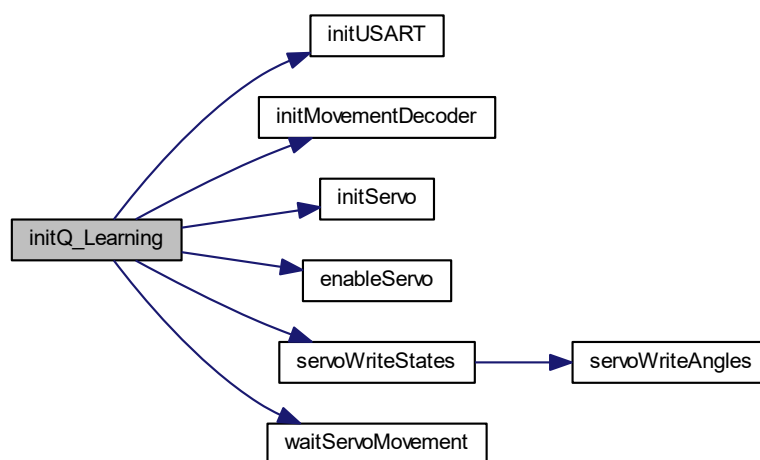


4.10.3.8 initQ_Learning()

```
void initQ_Learning ( )
```

Initializes the Q-learning algorithm. This function sets up the pins and the interrupt for the button. It also initializes the needed components for the Q-learning algorithm ([servo.h](#), [movementDecoder.h](#))

Here is the call graph for this function:



Here is the caller graph for this function:

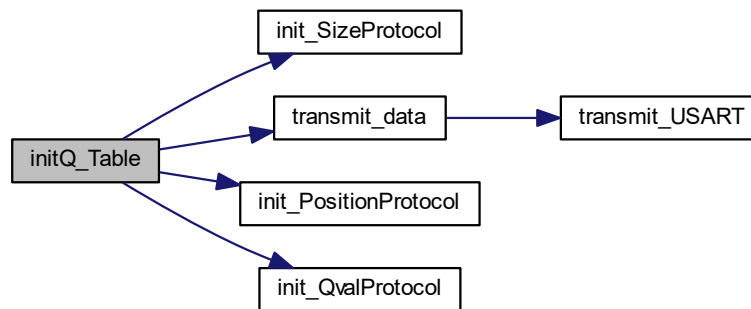


4.10.3.9 initQ_Table()

```
void initQ_Table ( )
```

Initializes the Q_Table with random values.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.10 learn()

```
void learn ( )
```

The Q-learning algorithm itself. This function executes the Q_learning algorithm itself.

4.10.3.11 wait_BTN_S()

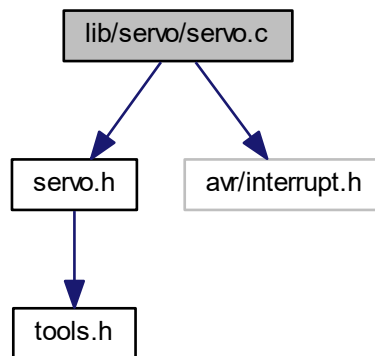
```
unsigned char wait_BTN_S ( )
```

This function waits until button south is pressed.

4.11 lib/servo/servo.c File Reference

```
#include "servo.h"
#include <avr/interrupt.h>
```

Include dependency graph for servo.c:



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.
- `ISR` (`TIMER1_COMPA_vect`)

Variables

- [BYTE dataB](#)
- struct [servo_t servos](#) [[NUMB_SERVOS](#)]
Contains all the data of the servos.
- unsigned char [stateAngles](#) [[NUMB_SERVO_STATES](#)]
Contains the angles of the predefined states.
- unsigned int [CurrentTicks](#) [[NUMB_SERVOS](#)]
Contains the amount of ticks that the servo is high.
- int [interpollateTime](#) = 0
Contains the amount of ticks before we interpollate the servo.

4.11.1 Function Documentation

4.11.1.1 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

<i>index</i>	the index of the servo that should be disabled
--------------	--

4.11.1.2 enableServo()

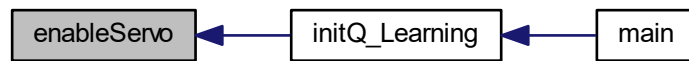
```
void enableServo (  
    int index )
```

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

Parameters

<i>index</i>	the index of the servo that should be enabled
--------------	---

Here is the caller graph for this function:



4.11.1.3 `initServo()`

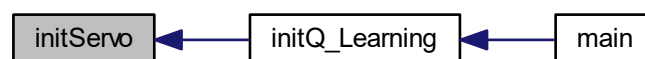
```
void initServo (
    unsigned char restStateArr[NUMB_SERVOS] )
```

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

<i>restStateArr</i>	these are the restStates of the servos
---------------------	--

Here is the caller graph for this function:

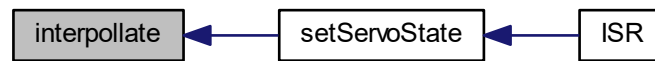


4.11.1.4 `interpollate()`

```
void interpollate ( ) [inline]
```

Interpollates the movement of the servo's. This function should only be used within this library.

Here is the caller graph for this function:



4.11.1.5 ISR()

```
ISR (
    TIMER1_COMPA_vect )
```

Here is the call graph for this function:



4.11.1.6 servoWriteAngle()

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

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

Parameters

<i>angle</i>	this contains the angle the servo needs to go to
<i>servoIndex</i>	this contains the index of the servo

Here is the caller graph for this function:



4.11.1.7 servoWriteAngles()

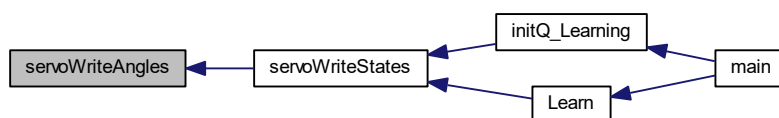
```
void servoWriteAngles (
    unsigned char posArr[] )
```

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

Parameters

<i>posArr[]</i>	this contains the angle values of the servo's
-----------------	---

Here is the caller graph for this function:



4.11.1.8 servoWriteState()

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

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

Parameters

<i>state</i>	this contains the state index corresponding servo
<i>servoIndex</i>	this contains the index of the servo

Here is the call graph for this function:



4.11.1.9 servoWriteStates()

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

This function makes the servos go to a given state.

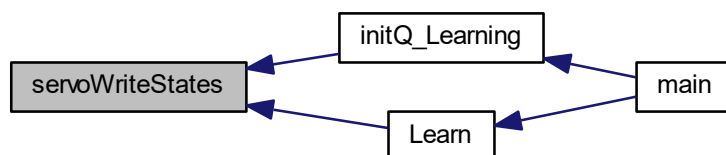
Parameters

<code>stateArr[]</code>	this contains the state index corresponding servo
-------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.1.10 setServoState()

```
void setServoState ( ) [inline]
```

This function helps to generate a PWM-signal. This function should only be used by servo timer interrupt (TIME←R1_COMP_A_vect)

Here is the call graph for this function:



Here is the caller graph for this function:

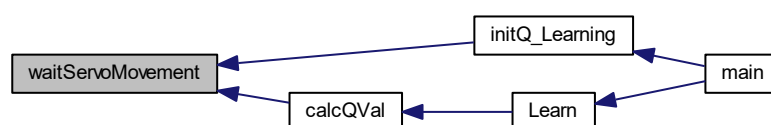


4.11.1.11 waitServoMovement()

```
void waitServoMovement ( )
```

This function waits until the servo reached the goal position.

Here is the caller graph for this function:



4.11.2 Variable Documentation

4.11.2.1 CurrentTicks

```
unsigned int CurrentTicks[NUMB_SERVOS]
```

Contains the amount of ticks that the servo is high.

4.11.2.2 dataB

```
BYTE dataB
```

[servo.c](#)

Created on: Mar 16, 2017 Author: Jonas Van Der Donck
this dataByte contains data of multiple parameters
BIT7: Bool isMoving (true if servos are moving)
BIT4: stateIndex (of switch state)
BIT0-3: Nibble servoIndex (max amount of servos is 1)

4.11.2.3 interpollateTime

```
int interpollateTime = 0
```

Contains the amount of ticks before we interpollate the servo.

4.11.2.4 servos

```
struct servo_t servos[NUMB_SERVOS]
```

Contains all the data of the servos.

4.11.2.5 stateAngles

```
unsigned char stateAngles[NUMB_SERVO_STATES]
```

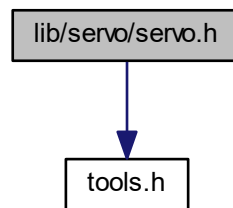
Contains the angles of the predefined states.

4.12 lib/servo/servo.h File Reference

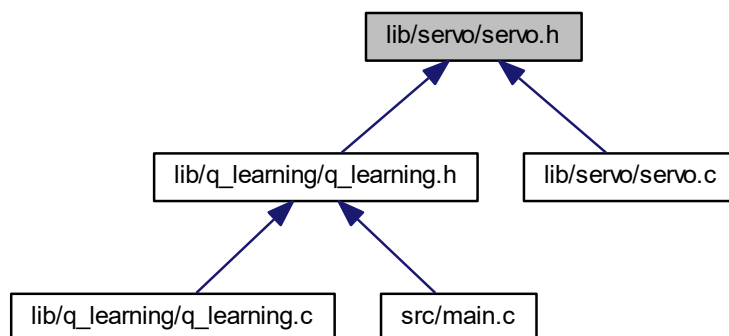
This library contains functions whom help to control multiple servos. Before using any of the the servo methods, call the [initServo\(\)](#) function.

```
#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 specifically designed to change those parameters.

Macros

- #define `CLK_TICKS_US` 16
the unprescaled clock uses 16 ticks per μ s
- #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 μ s
- #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 interval for interpolation
- #define `INTERPOLLATEVAL` 1
angle change with each interpolation
- #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_STATES - 1))
• #define BASE_TIME_WIDTH (PERIOD/NUMB_SERVOS)
    val in  $\mu$ s
• #define ANGLE_TO_TICKS_INTERP(ANG) (((US_TO_TICKS(MAX_PULSE_WIDTH - MIN_PULSE_WIDTH)/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_WIDTH)/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↔_COMP_A_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.

4.12.1 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

4.12.2 Macro Definition Documentation

4.12.2.1 ANGLE_TO_TICKS

```
#define ANGLE_TO_TICKS(  
    ANG ) ((US_TO_TICKS (ANGLE_TO_US (ANG)) )
```

4.12.2.2 ANGLE_TO_TICKS_INTERP

```
#define ANGLE_TO_TICKS_INTERP(  
    ANG ) (((US_TO_TICKS (MAX_PULSE_WIDTH - MIN_PULSE_WIDTH) / 180) * ANG))
```

4.12.2.3 ANGLE_TO_US

```
#define ANGLE_TO_US(  
    ANG ) (MIN_PULSE_WIDTH + ((MAX_PULSE_WIDTH - MIN_PULSE_WIDTH)/180)*ANG)
```

4.12.2.4 BASE_TIME_WIDTH

```
#define BASE_TIME_WIDTH (PERIOD/NUMB_SERVOS)
```

val in μ s

4.12.2.5 CLK_TICKS_US

```
#define CLK_TICKS_US 16
```

the unprescaled clock uses 16 ticks per μ s

4.12.2.6 DEFAULT_PULSE_WIDTH

```
#define DEFAULT_PULSE_WIDTH 1500
```

default pulse width when servo is attached

4.12.2.7 INTERPOLLATEINT

```
#define INTERPOLLATEINT 1500
```

refresh inte rval for interpollation

4.12.2.8 INTERPOLLATEVAL

```
#define INTERPOLLATEVAL 1
```

angle change wich each interpollation

4.12.2.9 ISMOVING_INDEX

```
#define ISMOVING_INDEX 7
```

index of the bool isMoving in the data byte

4.12.2.10 MAX_NUMB_SERVOS

```
#define MAX_NUMB_SERVOS 8
```

this is the maximum amount of servos

4.12.2.11 MAX_PULSE_WIDTH

```
#define MAX_PULSE_WIDTH 2400
```

the longest pulse sent to a servo

4.12.2.12 MAX_STATE_ANGLE

```
#define MAX_STATE_ANGLE 110
```

must be smaller than MAXANGLE

4.12.2.13 MAXANGLE

```
#define MAXANGLE 180
```

just informative

4.12.2.14 MIN_PULSE_WIDTH

```
#define MIN_PULSE_WIDTH 600
```

the shortest pulse sent to a servo in μ s

4.12.2.15 MIN_STATE_ANGLE

```
#define MIN_STATE_ANGLE 10
```

must be smaller than MAX_STATE_ANGLE

4.12.2.16 NUMB_SERVO_STATES

```
#define NUMB_SERVO_STATES 4
```

4.12.2.17 NUMB_SERVOS

```
#define NUMB_SERVOS 2
```

4.12.2.18 NUMB_SWITCH_STATES

```
#define NUMB_SWITCH_STATES 2
```

4.12.2.19 PERIOD

```
#define PERIOD 20000
```

this is the refresh interval

4.12.2.20 PRESCALER

```
#define PRESCALER 8
```

the prescaler of the clock

4.12.2.21 PULSE

```
#define PULSE 0
```

4.12.2.22 SERVO1

```
#define SERVO1 PORTC0
```

4.12.2.23 SERVO1_DIR

```
#define SERVO1_DIR DDRC
```

will be used as a pointer

4.12.2.24 SERVO1_PORT

```
#define SERVO1_PORT PORTC
```

will be used as a pointer

4.12.2.25 SERVO2

```
#define SERVO2 PORTC1
```

4.12.2.26 SERVO2_DIR

```
#define SERVO2_DIR DDRC
```

will be used as a pointer

4.12.2.27 SERVO2_PORT

```
#define SERVO2_PORT PORTC
```

will be used as a pointer

4.12.2.28 SERVO3

```
#define SERVO3 PORTB1
```

4.12.2.29 SERVO3_DIR

```
#define SERVO3_DIR DDRB
```

will be used as a pointer

4.12.2.30 SERVO3_PORT

```
#define SERVO3_PORT PORTB
```

will be used as a pointer

4.12.2.31 SERVO4

```
#define SERVO4 0
```

4.12.2.32 SERVO4_DIR

```
#define SERVO4_DIR DDRC
```

will be used as a pointer

4.12.2.33 SERVO4_PORT

```
#define SERVO4_PORT PORTC
```

will be used as a pointer

4.12.2.34 SERVO5

```
#define SERVO5 0
```

4.12.2.35 SERVO5_DIR

```
#define SERVO5_DIR DDRC
```

will be used as a pointer

4.12.2.36 SERVO5_PORT

```
#define SERVO5_PORT PORTC
```

will be used as a pointer

4.12.2.37 SERVO6

```
#define SERVO6 0
```

4.12.2.38 SERVO6_DIR

```
#define SERVO6_DIR DDRC
```

will be used as a pointer

4.12.2.39 SERVO6_PORT

```
#define SERVO6_PORT PORTC
```

will be used as a pointer

4.12.2.40 SERVO7

```
#define SERVO7 0
```

4.12.2.41 SERV07_DIR

```
#define SERV07_DIR DDRC
```

will be used as a pointer

4.12.2.42 SERV07_PORT

```
#define SERV07_PORT PORTC
```

will be used as a pointer

4.12.2.43 SERV08

```
#define SERV08 0
```

4.12.2.44 SERV08_DIR

```
#define SERV08_DIR DDRC
```

will be used as a pointer

4.12.2.45 SERV08_PORT

```
#define SERV08_PORT PORTC
```

will be used as a pointer

4.12.2.46 SERVO_INDEX_NIBBLE

```
#define SERVO_INDEX_NIBBLE 0
```

start index of the nibble that contains the servo index

4.12.2.47 STATE_ANGLE_WIDTH

```
#define STATE_ANGLE_WIDTH ((MAX_STATE_ANGLE - MIN_STATE_ANGLE) / (NUMB_SERVO_STATES - 1))
```

4.12.2.48 STATE_INDEX_POS

```
#define STATE_INDEX_POS 4
```

single bit value that contains the switch state index

4.12.2.49 TICKS_TO_ANGLE

```
#define TICKS_TO_ANGLE(  
    TCK ) ((TCK - US_TO_TICKS(MIN_PULSE_WIDTH)) / (US_TO_TICKS((MAX_PULSE_WIDTH - MIN_↵  
_PULSE_WIDTH) / 180)))
```

4.12.2.50 US_TO_TICKS

```
#define US_TO_TICKS(  
    US ) (US * (CLK_TICKS_US / PRESCALER))
```

4.12.2.51 WAIT

```
#define WAIT 1
```

4.12.3 Function Documentation

4.12.3.1 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

<i>index</i>	the index of the servo that should be disabled
--------------	--

4.12.3.2 enableServo()

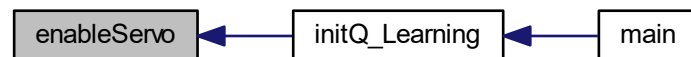
```
void enableServo (
    int index )
```

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

Parameters

<i>index</i>	the index of the servo that should be enabled
--------------	---

Here is the caller graph for this function:

**4.12.3.3 initServo()**

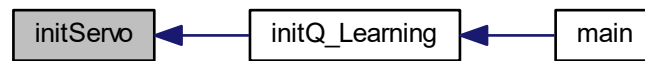
```
void initServo (
    unsigned char restStateArr[NUMB_SERVOS] )
```

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

<i>restStateArr</i>	these are the restStates of the servos
---------------------	--

Here is the caller graph for this function:

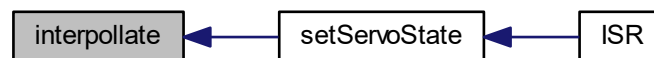


4.12.3.4 `interpollate()`

```
void interpollate ( ) [inline]
```

Interpollates the movement of the servo's. This function should only be used within this library.

Here is the caller graph for this function:



4.12.3.5 `servoWriteAngle()`

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

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

Parameters

<i>angle</i>	this contains the angle the servo needs to go to
<i>servoIndex</i>	this contains the index of the servo

Here is the caller graph for this function:



4.12.3.6 servoWriteAngles()

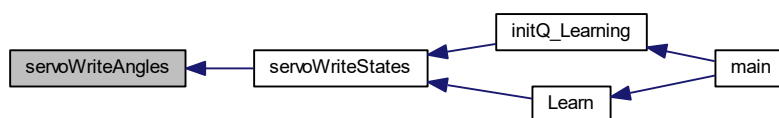
```
void servoWriteAngles (
    unsigned char posArr[] )
```

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

Parameters

<i>posArr[]</i>	this contains the angle values of the servo's
-----------------	---

Here is the caller graph for this function:



4.12.3.7 servoWriteState()

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

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

Parameters

<i>state</i>	this contains the state index corresponding servo
<i>servoIndex</i>	this contains the index of the servo

Here is the call graph for this function:



4.12.3.8 `servoWriteStates()`

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

This function makes the servos go to a given state.

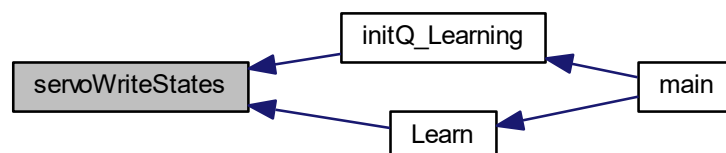
Parameters

<code>stateArr[]</code>	this contains the state index corresponding servo
-------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.9 setServoState()

```
void setServoState ( ) [inline]
```

This function helps to generate a PWM-signal. This function should only be used by servo timer interrupt (TIME←R1_COMPA_vect)

Here is the call graph for this function:



Here is the caller graph for this function:

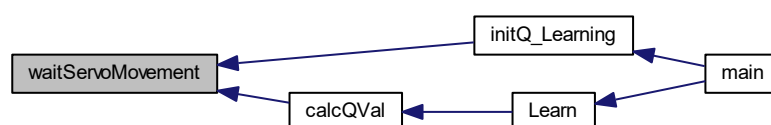


4.12.3.10 waitServoMovement()

```
void waitServoMovement ( )
```

This function waits until the servo reached the goal position.

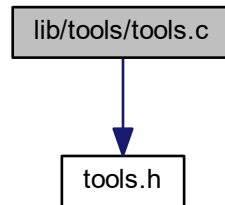
Here is the caller graph for this function:



4.13 lib/tools/tools.c File Reference

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

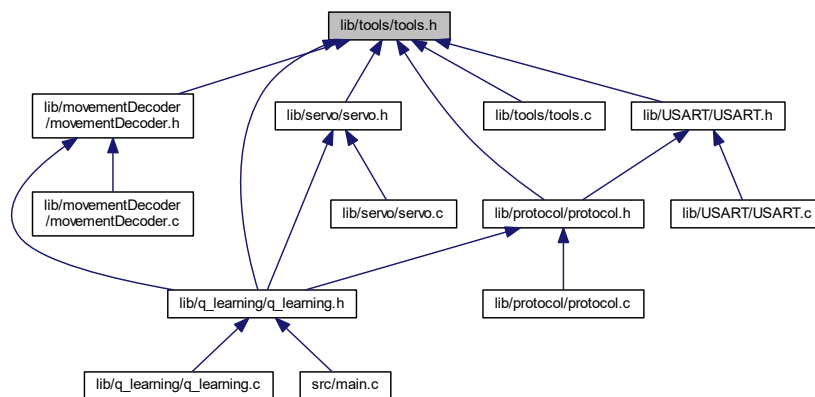
Include dependency graph for tools.c:



4.14 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))`

4.14.1 Detailed Description

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

Author

Jonas Van Der Donckt

Date

14/05/2017

4.14.2 Macro Definition Documentation

4.14.2.1 BYTE

```
#define BYTE unsigned char
```

4.14.2.2 CLEAR_NIBBLE

```
#define CLEAR_NIBBLE(  
    REG,  
    LSB_INDEX ) (REG &= ~(0x0F << LSB_INDEX) )
```

4.14.2.3 F_CPU

```
#define F_CPU 16000000UL
```

adapt this if the CPU has a different frequency

4.14.2.4 GET_BIT

```
#define GET_BIT(  
    REG,  
    POS ) ((REG & (1<< POS)) > (0) ? (1) : (0))
```

4.14.2.5 GET_NIBBLE

```
#define GET_NIBBLE(  
    REG,  
    LSB_INDEX,  
    D1 ) (D1 = (REG & (0x0F << LSB_INDEX)))
```

4.14.2.6 SET_BIT_HIGH

```
#define SET_BIT_HIGH(  
    REG,  
    BIT ) REG |= (1 << BIT)
```

4.14.2.7 SET_BIT_LOW

```
#define SET_BIT_LOW(  
    REG,  
    BIT ) REG &= ~(1 << BIT)
```

4.14.2.8 SET_NIBBLE

```
#define SET_NIBBLE(  
    REG,  
    LSB_INDEX,  
    D1 ) (REG |= (D1 << LSB_INDEX))
```

4.14.2.9 SET_PIN_HIGH

```
#define SET_PIN_HIGH(  
    PORT,  
    PIN ) PORT |= (1 << PIN)
```

4.14.2.10 SET_PIN_INPUT

```
#define SET_PIN_INPUT(  
    DDR,  
    PIN ) SET_PIN_LOW(DDR, PIN)
```

4.14.2.11 SET_PIN_LOW

```
#define SET_PIN_LOW(  
    PORT,  
    PIN ) PORT &= ~(1 << PIN)
```

4.14.2.12 SET_PIN_OUTPUT

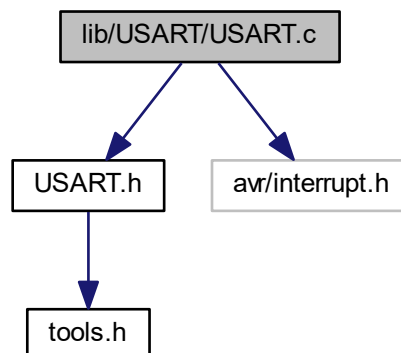
```
#define SET_PIN_OUTPUT(  
    DDR,  
    PIN ) SET_PIN_HIGH(DDR, PIN)
```

4.14.2.13 TOGGLE_BIT

```
#define TOGGLE_BIT(  
    REG,  
    POS ) (REG ^= (1<< POS))
```

4.15 lib/USART/USART.c File Reference

```
#include "USART.h"  
#include <avr/interrupt.h>  
Include dependency graph for USART.c:
```



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.

4.15.1 Function Documentation

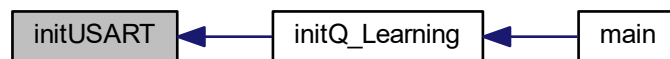
4.15.1.1 `initUSART()`

```
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 Here is the caller graph for this function:



4.15.1.2 `Receive_USART()`

```
unsigned char Receive_USART ( )
```

Receives data with the USART protocol.

4.15.1.3 `transmit_USART()`

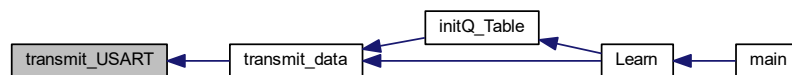
```
void transmit_USART (
    BYTE data )
```

Transmits data with the USART protocol.

Parameters

<i>data</i>	this is the data that will be transmitted.
-------------	--

Here is the caller graph for this function:

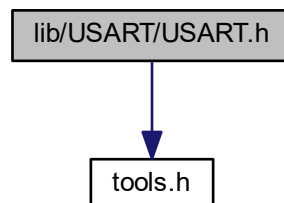


4.16 lib/USART/USART.h File Reference

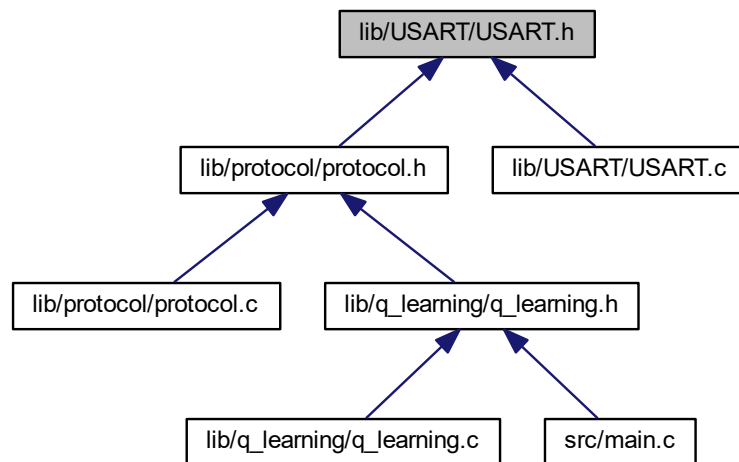
This library contains functions to transmit data with the USART protocol. Before using the USART's methods, call the [initUSART\(\)](#) function.

```
#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.

4.16.1 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

4.16.2 Macro Definition Documentation

4.16.2.1 BAUDRATE

```
#define BAUDRATE (long)9600
```

adapt to change BAUDRATE

4.16.2.2 UBBRVAL

```
#define UBBRVAL (F_CPU/(16*BAUDRATE) - 1)
```

4.16.3 Function Documentation

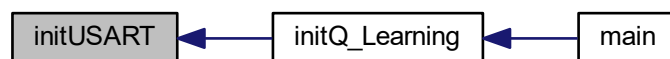
4.16.3.1 initUSART()

```
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 Here is the caller graph for this function:



4.16.3.2 Receive_USART()

```
unsigned char Receive_USART ( )
```

Receives data with the USART protocol.

4.16.3.3 transmit_USART()

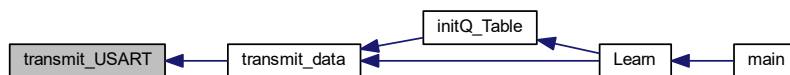
```
void transmit_USART (
    BYTE data )
```

Transmits data with the USART protocol.

Parameters

<i>data</i>	this is the data that will be transmitted.
-------------	--

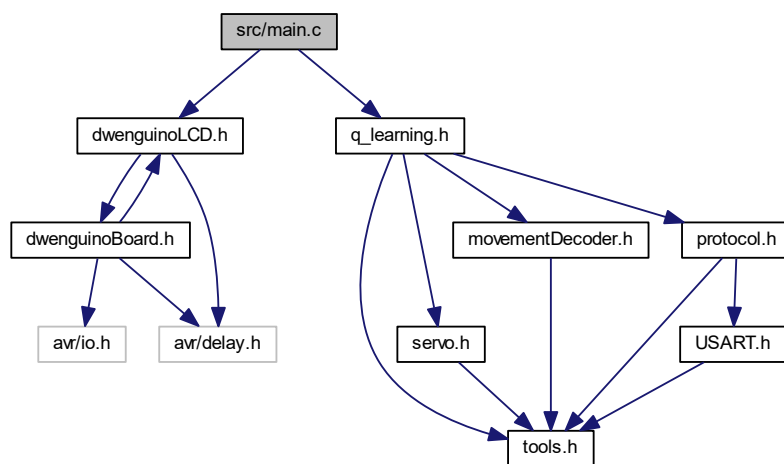
Here is the caller graph for this function:



4.17 src/main.c File Reference

```
#include "dwenguinoLCD.h"
#include "q_learning.h"
```

Include dependency graph for `main.c`:



Functions

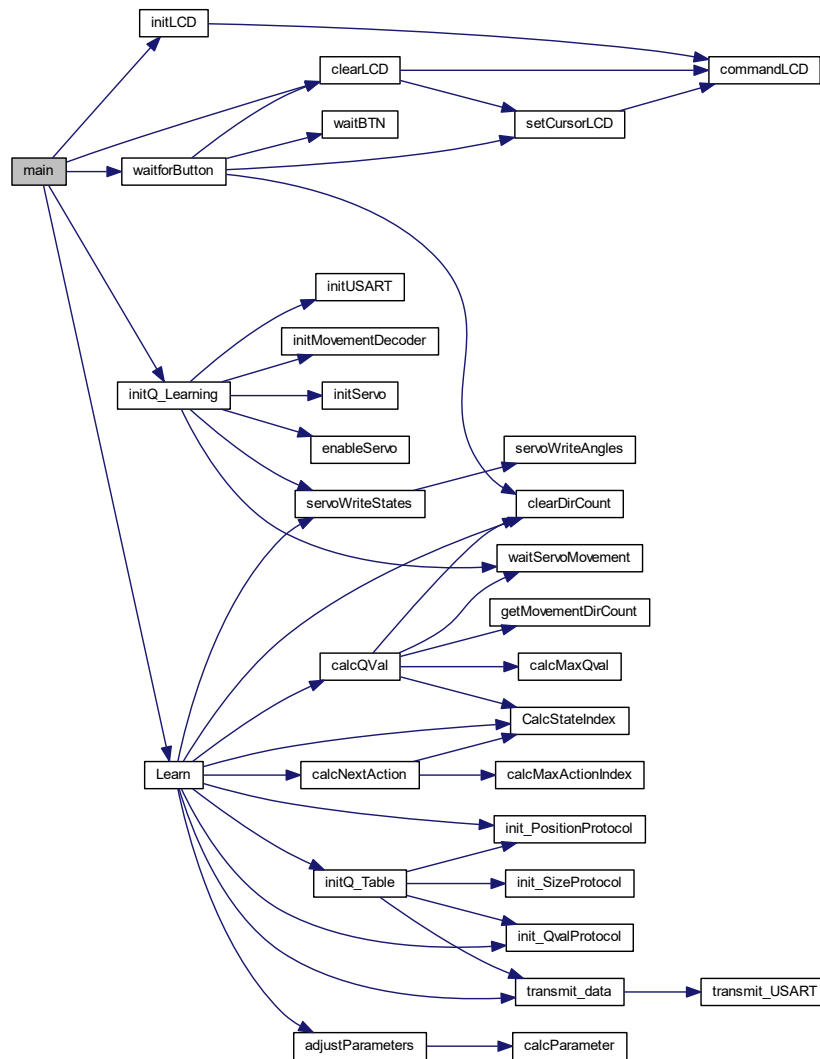
- void `waitforButton` ()
- int `main` ()

4.17.1 Function Documentation

4.17.1.1 main()

```
int main ( )
```

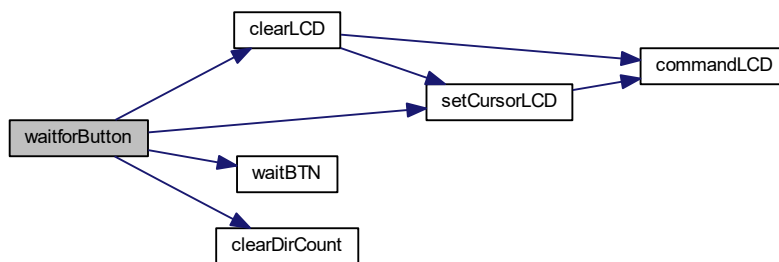
< waits until the user has pressed BTN_S Here is the call graph for this function:



4.17.1.2 waitForButton()

```
void waitForButton ( )
```

Here is the call graph for this function:



Here is the caller graph for this function:



