ESP Project Software API Documentation

Beta4

Group 3

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 SensorArray Class Reference

Class representing a sensor array.

Public Member Functions

- SensorArray (PinName s1, PinName s2, PinName s3, PinName s4, PinName s5, PinName s6, PinName l1, PinName l2, PinName l3, PinName l4, PinName l5, PinName l6)

  Constructor for SensorArray class.
- double getSensorVolts (int i)

  Get voltage reading from a sensor.
- void sLEDOn (int i)

  Turn on sensor LEDs.
- void sLEDOff (int i)

  Turn off sensor LEDs.
- void **toggle3and3Read** (void)

  Toggle reading of sensors 3 at a time.
- void **toggleReadSingle** (void)

  Toggle reading of sensors one at a time.
- void **read3and3** (void)

  Reads the line displacement 3 sensors at once, called by the getValue Ticker.
- double getErrorValue (void)

  Returns the value of displacement from the line.
- void **singleRead** (void)

  Reads the line displacement 1 sensor at a time, called by the getValue Ticker.

Protected Attributes

- bool **readOff**

    Flag indicating whether reading is turned off.
- int **read3and3State**

    State for error value reading.
- int **singleReadState**

    State for single reading.
- double **sensor1Val**
- double **sensor2Val**
- double **sensor3Val**
- double **sensor4Val**
- double **sensor5Val**
- double **sensor6Val**

    Current sensor values.
- double **sample1Val**
- double **sample2Val**
- double **sample3Val**
- double **sample4Val**
- double **sample5Val**
- double **sample6Val**

    Sample sensor values.
- Ticker **getValue**

    Timer for triggering readings.
- AnalogIn **sensor1In**
- AnalogIn **sensor2In**
- AnalogIn **sensor3In**
- AnalogIn **sensor4In**
- AnalogIn **sensor5In**
- AnalogIn **sensor6In**

    Analog input pins for sensors.
- DigitalOut **LED1Out**
- DigitalOut **LED2Out**
- DigitalOut **LED3Out**
- DigitalOut **LED4Out**
- DigitalOut **LED5Out**
- DigitalOut **LED6Out**

    Digital output pins for LEDs.

3.1.1 Detailed Description

Class representing a sensor array.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 SensorArray()

```
SensorArray::SensorArray (
             PinName s1,
             PinName s2,
             PinName s3,
             PinName s4,
             PinName s5,
             PinName s6,
             PinName l1,
             PinName l2,
             PinName l3,
             PinName l4,
             PinName l5,
             PinName l6 )  [inline]
```

Constructor for SensorArray class.

Parameters

s1-s6	Pin names for sensors.
l1-l6	Pin names for LEDs.

3.1.3 Member Function Documentation

3.1.3.1 getErrorValue()

```
double SensorArray::getErrorValue (
             void  )  [inline]
```

Returns the value of displacement from the line.

Returns

Error value.

3.1.3.2 getSensorVolts()

```
double SensorArray::getSensorVolts (
             int i )  [inline]
```

Get voltage reading from a sensor.

Parameters

i	Sensor index.

Returns

Voltage value.

3.1.3.3 sLEDOff()

```
void SensorArray::sLEDOff (
             int i )  [inline]
```

Turn off sensor LEDs.

Parameters

i	LED index.

3.1.3.4 sLEDOn()

```
void SensorArray::sLEDOn (
             int i )  [inline]
```

Turn on sensor LEDs.

Parameters

i	LED index.

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

- main.cpp

3.2 unipolarmotor Class Reference

Represents a unipolar motor with PWM control.

Public Member Functions

- unipolarmotor (PinName pwmPin, PinName dirPin)

  Constructor to initialize the motor with PWM and direction pins.
- void **changeDirection** ()

  Change the direction of the motor.
- void **motorStop** ()

Public Attributes

- PwmOut **pwm**

  PWM output for motor speed control.
- DigitalOut **dir**

  Direction control for motor rotation.
- float **pwmValue**

  PWM value for motor speed control.

3.2.1 Detailed Description

Represents a unipolar motor with PWM control.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 unipolarmotor()

```
unipolarmotor::unipolarmotor (
            PinName pwmPin,
            PinName dirPin )  [inline]
```

Constructor to initialize the motor with PWM and direction pins.

Parameters

pwmPin	Pin for PWM control
dirPin	Pin for direction control

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

- main.cpp

Chapter 4

File Documentation

4.1 main.cpp File Reference

Autonomous Line Follower Program.

```
#include "C12832.h"
#include "QEI.h"
#include "mbed.h"
```

Classes

- class SensorArray

  Class representing a sensor array.
- class unipolarmotor

  Represents a unipolar motor with PWM control.

Functions

- C12832 **lcd** (D11, D13, D12, D7, D10)

  LCD display object.
- InterruptIn **FirePressed** (D4)

  Interrupt for motor driver board enable.
- DigitalOut **EnableMDB** (D8)

  Motor driver board enable.
- DigitalOut **Bipolar1** (PB_13)

  Bipolar control 1.
- DigitalOut **Bipolar2** (PB_14)

  Bipolar control 2.
- Serial **hm10** (PA_11, PA_12)
- Serial **pc** (USBTX, USBRX)
- QEI **leftEncoder** (PC_8, PC_6, NC, 256, QEI::X4_ENCODING)

  Left motor encoder.
- QEI **rightEncoder** (PC_2, PC_3, NC, 256, QEI::X4_ENCODING)

  Right motor encoder.

- void **updateEncoders** ()

    Update encoder pulse counts and calculate speed.
- void **enableMotorDriverBoard** ()

    Enable the motor driver board.
- void **motorStop** ()

    Stop both motors.
- void **motorStart** ()
- void **readEncoders** ()
- void **turnAround** ()
- float PID_controller (float error, float &integral, float &previous_error, float dt, float Kp, float Ki, float Kd)

    PID controller to compute motor control output.
- void **PID_reset** ()
- void **serial_config** ()
- int main ()

    Main function.

Variables

- unipolarmotor **leftMotor** (D9, D2)

    Left motor object.
- unipolarmotor **rightMotor** (D5, PA_13)

    Right motor object.
- int **LeftPulse** = 0

    Left encoder pulse count.
- int **RightPulse** = 0

    Right encoder pulse count.
- int **Left_lastPulse** = 0

    Last recorded left encoder pulse.
- int **Right_lastPulse** = 0

    Last recorded right encoder pulse.
- float **L_defaultPPS** = 3500

    Target PPS for left motor.
- float **L_targetPPS**
- float **R_defaultPPS** = 3500

    Target PPS for right motor.
- float **R_targetPPS**
- float **L_defaultPWM** = 0.70

    Default PWM for left motor.
- float **R_defaultPWM** = 0.70

    Default PWM for right motor.
- float **L_PWM** = 0.70

    Default PWM for left motor.
- float **R_PWM** = 0.70

    Default PWM for right motor.
- float **vs1**
- float **vs2**
- float **vs3**
- float **vs4**
- float **vs5**
- float **vs6**

    Voltage readings from line sensors.

- float **lineSensorError** = 0.0

  Error calculated from line sensor values.
- float **positiveTurnLimit** = 0.5
- float **negativeTurnLimit** = -0.5
- char **c**
- char **w**
- int **_180degree** = 1350
- Ticker **encoderTicker**

  Ticker for encoder update.
- float **integral_left** = 0

  Integral term for left motor PID controller.
- float **previous_error_left** = 0

  Previous error for left motor PID controller.
- float **integral_right** = 0

  Integral term for right motor PID controller.
- float **previous_error_right** = 0

  Previous error for right motor PID controller.

4.1.1 Detailed Description

Autonomous Line Follower Program.

4.1.2 Function Documentation

4.1.2.1 main()

```
int main ( )
```

Main function.

Returns

Program exit status

4.1.2.2 PID_controller()

```
float PID_controller (
             float error,
             float & integral,
             float & previous_error,
             float dt,
             float Kp,
             float Ki,
             float Kd )
```

PID controller to compute motor control output.

Parameters

error	Error value to control
integral	Integral component of PID
previous_error	Previous error for derivative component
dt	Time interval
Kp	Proportional gain
Ki	Integral gain
Kd	Derivative gain

Returns

PID output