

# Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Wed Aug 19 2015 01:07:04

## Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>1</b>
2.1	Class List . . . . .	1
<b>3</b>	<b>File Index</b>	<b>1</b>
3.1	File List . . . . .	1
<b>4</b>	<b>Class Documentation</b>	<b>2</b>
4.1	ColorRecognition Class Reference . . . . .	2
4.1.1	Detailed Description . . . . .	2
4.1.2	Member Function Documentation . . . . .	2
4.2	ColorRecognitionTCS230 Class Reference . . . . .	3
4.2.1	Detailed Description . . . . .	5
4.2.2	Member Enumeration Documentation . . . . .	5
4.2.3	Constructor & Destructor Documentation . . . . .	5
4.2.4	Member Function Documentation . . . . .	5
4.2.5	Member Data Documentation . . . . .	7
4.3	ColorRecognitionTCS230PI Class Reference . . . . .	8
4.3.1	Detailed Description . . . . .	9
4.3.2	Member Enumeration Documentation . . . . .	9
4.3.3	Constructor & Destructor Documentation . . . . .	10
4.3.4	Member Function Documentation . . . . .	10
4.3.5	Member Data Documentation . . . . .	11
<b>5</b>	<b>File Documentation</b>	<b>12</b>
5.1	ColorRecognition.cpp File Reference . . . . .	12
5.1.1	Macro Definition Documentation . . . . .	12
5.2	ColorRecognition.cpp . . . . .	13
5.3	ColorRecognition.h File Reference . . . . .	13
5.4	ColorRecognition.h . . . . .	13
5.5	ColorRecognitionTCS230.cpp File Reference . . . . .	13
5.5.1	Macro Definition Documentation . . . . .	14
5.6	ColorRecognitionTCS230.cpp . . . . .	14
5.7	ColorRecognitionTCS230.h File Reference . . . . .	16
5.7.1	Macro Definition Documentation . . . . .	16
5.8	ColorRecognitionTCS230.h . . . . .	17
5.9	ColorRecognitionTCS230PI.cpp File Reference . . . . .	18
5.9.1	Macro Definition Documentation . . . . .	19

5.10	<a href="#">ColorRecognitionTCS230PI.cpp</a>	19
5.11	<a href="#">ColorRecognitionTCS230PI.h File Reference</a>	20
5.11.1	<a href="#">Macro Definition Documentation</a>	21
5.12	<a href="#">ColorRecognitionTCS230PI.h</a>	22
5.13	<a href="#">simple_read.c File Reference</a>	23
5.13.1	<a href="#">Function Documentation</a>	23
5.14	<a href="#">simple_read.c</a>	24
	<a href="#">Index</a>	25

## 1 Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<b>ColorRecognition</b>	<a href="#">2</a>
<b>ColorRecognitionTCS230</b>	<a href="#">3</a>
<b>ColorRecognitionTCS230PI</b>	<a href="#">8</a>

## 2 Class Index

### 2.1 Class List

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

<b>ColorRecognition</b>	
<b>Arduino - Color Recognition Sensor</b>	<a href="#">2</a>
<b>ColorRecognitionTCS230</b>	<a href="#">3</a>
<b>ColorRecognitionTCS230PI</b>	<a href="#">8</a>

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<b>ColorRecognition.cpp</b>	<a href="#">12</a>
<b>ColorRecognition.h</b>	<a href="#">13</a>
<b>ColorRecognitionTCS230.cpp</b>	<a href="#">13</a>
<b>ColorRecognitionTCS230.h</b>	<a href="#">16</a>
<b>ColorRecognitionTCS230PI.cpp</b>	<a href="#">18</a>

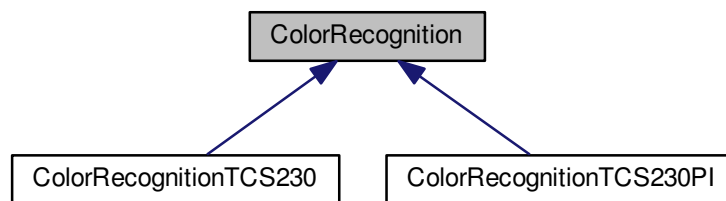
<a href="#">ColorRecognitionTCS230PI.h</a>	20
<a href="#">simple_read.c</a>	23

## 4 Class Documentation

### 4.1 ColorRecognition Class Reference

```
#include <ColorRecognition.h>
```

Inheritance diagram for ColorRecognition:



#### Public Member Functions

- virtual unsigned char [getRed](#) ()=0
- virtual unsigned char [getGreen](#) ()=0
- virtual unsigned char [getBlue](#) ()=0
- virtual bool [fillRGB](#) (unsigned char buf[3])=0

#### 4.1.1 Detailed Description

Arduino - Color Recognition Sensor.

[ColorRecognition.h](#)

The abstract class for the color recognition sensors.

#### Author

Dalmir da Silva [dalmirdasilva@gmail.com](mailto:dalmirdasilva@gmail.com)

Definition at line 14 of file [ColorRecognition.h](#).

#### 4.1.2 Member Function Documentation

##### 4.1.2.1 virtual bool ColorRecognition::fillRGB ( unsigned char buf[3] ) [pure virtual]

Returns the blue color intensity.

The blue color intensity.

Implemented in [ColorRecognitionTCS230](#), and [ColorRecognitionTCS230PI](#).

4.1.2.2 `virtual unsigned char ColorRecognition::getBlue ( ) [pure virtual]`

Returns the blue color intensity.

The blue color intensity.

Implemented in [ColorRecognitionTCS230](#), and [ColorRecognitionTCS230PI](#).

4.1.2.3 `virtual unsigned char ColorRecognition::getGreen ( ) [pure virtual]`

Returns the green color intensity.

The green color intensity.

Implemented in [ColorRecognitionTCS230](#), and [ColorRecognitionTCS230PI](#).

4.1.2.4 `virtual unsigned char ColorRecognition::getRed ( ) [pure virtual]`

Returns the red color intensity.

The red color intensity.

Implemented in [ColorRecognitionTCS230](#), and [ColorRecognitionTCS230PI](#).

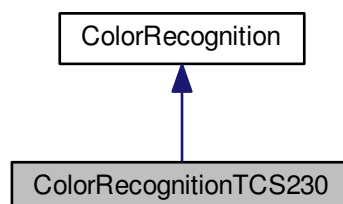
The documentation for this class was generated from the following file:

- [ColorRecognition.h](#)

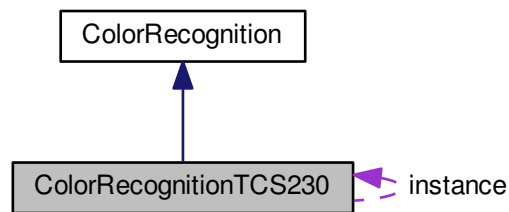
## 4.2 ColorRecognitionTCS230 Class Reference

```
#include <ColorRecognitionTCS230.h>
```

Inheritance diagram for ColorRecognitionTCS230:



Collaboration diagram for ColorRecognitionTCS230:



### Public Types

- enum `Filter` { `RED_FILTER`, `GREEN_FILTER`, `BLUE_FILTER`, `CLEAR_FILTER` }

### Public Member Functions

- virtual `~ColorRecognitionTCS230` ()
- void `initialize` (unsigned char `outPin`, unsigned char `s2Pin`, unsigned char `s3Pin`)
- void `adjustWhiteBalance` ()
- unsigned char `getRed` ()
- unsigned char `getGreen` ()
- unsigned char `getBlue` ()
- bool `fillRGB` (unsigned char `buf[3]`)

### Static Public Member Functions

- static `ColorRecognitionTCS230 * getInstance` ()
- static void `setFilter` (`Filter` `filter`)

### Public Attributes

- `Filter currentFilter`

### Private Member Functions

- `ColorRecognitionTCS230` ()

### Static Private Member Functions

- static void `externalInterruptHandler` ()
- static void `timerInterruptHandler` ()

### Private Attributes

- unsigned char [s2Pin](#)
- unsigned char [s3Pin](#)
- unsigned char [outPin](#)
- int [count](#)
- int [lastFrequencies](#) [3]
- int [whiteBalanceFrequencies](#) [3]

### Static Private Attributes

- static [ColorRecognitionTCS230](#) instance

#### 4.2.1 Detailed Description

Definition at line 94 of file [ColorRecognitionTCS230.h](#).

#### 4.2.2 Member Enumeration Documentation

##### 4.2.2.1 enum ColorRecognitionTCS230::Filter

Filter color enumeration.

### Enumerator

***RED\_FILTER***  
***GREEN\_FILTER***  
***BLUE\_FILTER***  
***CLEAR\_FILTER***

Definition at line 139 of file [ColorRecognitionTCS230.h](#).

#### 4.2.3 Constructor & Destructor Documentation

##### 4.2.3.1 virtual ColorRecognitionTCS230::~ColorRecognitionTCS230 ( ) [inline], [virtual]

Definition at line 160 of file [ColorRecognitionTCS230.h](#).

##### 4.2.3.2 ColorRecognitionTCS230::ColorRecognitionTCS230 ( ) [inline], [private]

Private constructor.

Definition at line 230 of file [ColorRecognitionTCS230.h](#).

#### 4.2.4 Member Function Documentation

##### 4.2.4.1 void ColorRecognitionTCS230::adjustWhiteBalance ( )

Store the current read as the maximum frequency for each color.

It tells what is considered white.

Definition at line 33 of file [ColorRecognitionTCS230.cpp](#).

**4.2.4.2** `void ColorRecognitionTCS230::externalInterruptHandler ( ) [static],[private]`

Device output interruption handler.

Definition at line 40 of file [ColorRecognitionTCS230.cpp](#).

**4.2.4.3** `bool ColorRecognitionTCS230::fillRGB ( unsigned char buff[3] ) [virtual]`

Returns the blue color intensity.

The blue color intensity.

Implements [ColorRecognition](#).

Definition at line 87 of file [ColorRecognitionTCS230.cpp](#).

**4.2.4.4** `unsigned char ColorRecognitionTCS230::getBlue ( ) [virtual]`

Returns the blue color intensity.

The blue color intensity.

Implements [ColorRecognition](#).

Definition at line 80 of file [ColorRecognitionTCS230.cpp](#).

**4.2.4.5** `unsigned char ColorRecognitionTCS230::getGreen ( ) [virtual]`

Returns the green color intensity.

The green color intensity.

Implements [ColorRecognition](#).

Definition at line 73 of file [ColorRecognitionTCS230.cpp](#).

**4.2.4.6** `static ColorRecognitionTCS230* ColorRecognitionTCS230::getInstance ( ) [inline],[static]`

Singleton.

Gets the instance of the driver.

Returns

Definition at line 156 of file [ColorRecognitionTCS230.h](#).

**4.2.4.7** `unsigned char ColorRecognitionTCS230::getRed ( ) [virtual]`

Returns the red color intensity.

The red color intensity.

Implements [ColorRecognition](#).

Definition at line 66 of file [ColorRecognitionTCS230.cpp](#).

**4.2.4.8** `void ColorRecognitionTCS230::initialize ( unsigned char outPin, unsigned char s2Pin, unsigned char s3Pin )`

Initializes the IO and timers.

Parameters

<i>outPin</i>	The out pin. (NOTE: It must be the 2 or 3 pin to support external interrupts).
---------------	--



<i>s2Pin</i>	The s2 pin.
<i>s3Pin</i>	The s3 pin.

### Returns

Definition at line 20 of file [ColorRecognitionTCS230.cpp](#).

#### 4.2.4.9 void ColorRecognitionTCS230::setFilter ( Filter filter ) [static]

Sets the s2 and s3 pins according of the color passed as filter.

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

### Parameters

<i>filter</i>	The next filter.
---------------	------------------

Definition at line 94 of file [ColorRecognitionTCS230.cpp](#).

#### 4.2.4.10 void ColorRecognitionTCS230::timerInterruptHandler ( ) [static],[private]

TimerOne interrupt handler.

Definition at line 44 of file [ColorRecognitionTCS230.cpp](#).

### 4.2.5 Member Data Documentation

#### 4.2.5.1 int ColorRecognitionTCS230::count [private]

Holds the number of interrupts of the current filter.

Definition at line 117 of file [ColorRecognitionTCS230.h](#).

#### 4.2.5.2 Filter ColorRecognitionTCS230::currentFilter

Current filter.

Definition at line 149 of file [ColorRecognitionTCS230.h](#).

#### 4.2.5.3 ColorRecognitionTCS230 ColorRecognitionTCS230::instance [static],[private]

Singleton.

The instance.

Definition at line 132 of file [ColorRecognitionTCS230.h](#).

#### 4.2.5.4 int ColorRecognitionTCS230::lastFrequencies[3] [private]

Holds the last count for each filter.

Definition at line 122 of file [ColorRecognitionTCS230.h](#).

#### 4.2.5.5 unsigned char ColorRecognitionTCS230::outPin [private]

The out pin.

NOTE: It must be the 2 or 3 pin to support external interrupts.

Definition at line 112 of file [ColorRecognitionTCS230.h](#).

4.2.5.6 unsigned char ColorRecognitionTCS230::s2Pin [private]

The s2 pin.

Definition at line 100 of file [ColorRecognitionTCS230.h](#).

4.2.5.7 unsigned char ColorRecognitionTCS230::s3Pin [private]

The s3 pin.

Definition at line 105 of file [ColorRecognitionTCS230.h](#).

4.2.5.8 int ColorRecognitionTCS230::whiteBalanceFrequencies[3] [private]

Holds the maximum frequencies.

Definition at line 127 of file [ColorRecognitionTCS230.h](#).

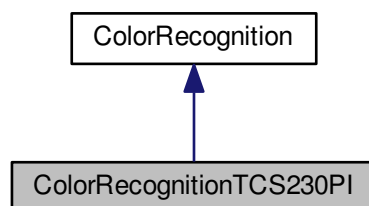
The documentation for this class was generated from the following files:

- [ColorRecognitionTCS230.h](#)
- [ColorRecognitionTCS230.cpp](#)

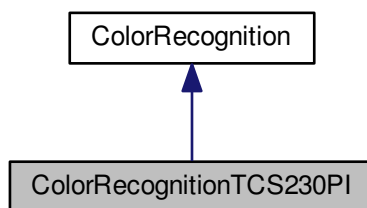
### 4.3 ColorRecognitionTCS230PI Class Reference

```
#include <ColorRecognitionTCS230PI.h>
```

Inheritance diagram for ColorRecognitionTCS230PI:



Collaboration diagram for ColorRecognitionTCS230PI:



#### Public Types

- enum `Filter` { `RED_FILTER`, `GREEN_FILTER`, `BLUE_FILTER`, `CLEAR_FILTER` }

#### Public Member Functions

- `ColorRecognitionTCS230PI` (unsigned char `outPin`, unsigned char `s2Pin`, unsigned char `s3Pin`)
- void `adjustWhiteBalance` ()
- void `adjustBlackBalance` ()
- unsigned char `getRed` ()
- unsigned char `getGreen` ()
- unsigned char `getBlue` ()
- bool `fillRGB` (unsigned char buf[3])
- long `getFrequency` (unsigned int samples)
- void `setFilter` (`Filter` filter)

#### Private Attributes

- unsigned char `s2Pin`
- unsigned char `s3Pin`
- unsigned char `outPin`
- long `minFrequency` [3]
- long `maxFrequency` [3]

#### 4.3.1 Detailed Description

Definition at line 81 of file `ColorRecognitionTCS230PI.h`.

#### 4.3.2 Member Enumeration Documentation

##### 4.3.2.1 enum `ColorRecognitionTCS230PI::Filter`

Filter color enumeration.

#### Enumerator

**`RED_FILTER`**

**GREEN\_FILTER**

**BLUE\_FILTER**

**CLEAR\_FILTER**

Definition at line 114 of file [ColorRecognitionTCS230PI.h](#).

#### 4.3.3 Constructor & Destructor Documentation

4.3.3.1 **ColorRecognitionTCS230PI::ColorRecognitionTCS230PI ( unsigned char *outPin*, unsigned char *s2Pin*, unsigned char *s3Pin* )**

Private constructor.

Definition at line 16 of file [ColorRecognitionTCS230PI.cpp](#).

#### 4.3.4 Member Function Documentation

4.3.4.1 **void ColorRecognitionTCS230PI::adjustBlackBalance ( )**

Store the current read as the maximum frequency for each color.

It tells what is considered white.

Definition at line 37 of file [ColorRecognitionTCS230PI.cpp](#).

4.3.4.2 **void ColorRecognitionTCS230PI::adjustWhiteBalance ( )**

Store the current read as the minimum frequency for each color.

It tells what is considered black.

Definition at line 30 of file [ColorRecognitionTCS230PI.cpp](#).

4.3.4.3 **bool ColorRecognitionTCS230PI::fillIRGB ( unsigned char *buf*[3] ) [virtual]**

Returns the blue color intensity.

The blue color intensity.

Implements [ColorRecognition](#).

Definition at line 59 of file [ColorRecognitionTCS230PI.cpp](#).

4.3.4.4 **unsigned char ColorRecognitionTCS230PI::getBlue ( ) [virtual]**

Returns the blue color intensity.

The blue color intensity.

Implements [ColorRecognition](#).

Definition at line 54 of file [ColorRecognitionTCS230PI.cpp](#).

4.3.4.5 **long ColorRecognitionTCS230PI::getFrequency ( unsigned int *samples* )**

Gets the frequency from the out pin.

NOTE: It uses `pulseIn`, collects some samples and calculate the frequency.

The out pin generates a square wave, we sum the times between the raise edge and divide by the number of samples.

```

      1      2      3
-----

```

```

|   |   |   |   |
-----

```

**Returns**

The pin frequency.

Definition at line 78 of file [ColorRecognitionTCS230PI.cpp](#).

**4.3.4.6 unsigned char ColorRecognitionTCS230PI::getGreen ( ) [virtual]**

Returns the green color intensity.

The green color intensity.

Implements [ColorRecognition](#).

Definition at line 49 of file [ColorRecognitionTCS230PI.cpp](#).

**4.3.4.7 unsigned char ColorRecognitionTCS230PI::getRed ( ) [virtual]**

Returns the red color intensity.

The red color intensity.

Implements [ColorRecognition](#).

Definition at line 44 of file [ColorRecognitionTCS230PI.cpp](#).

**4.3.4.8 void ColorRecognitionTCS230PI::setFilter ( Filter filter )**

Sets the s2 and s3 pins according of the color passed as filter.

```

S2   S3   PHOTODIODE TYPE
L    L    Red
L    H    Blue
H    L    Clear (no filter)
H    H    Green

```

**Parameters**

<i>filter</i>	The next filter.
---------------	------------------

Definition at line 66 of file [ColorRecognitionTCS230PI.cpp](#).

**4.3.5 Member Data Documentation****4.3.5.1 long ColorRecognitionTCS230PI::maxFrequency[3] [private]**

The maximum frequency.

Definition at line 107 of file [ColorRecognitionTCS230PI.h](#).

**4.3.5.2 long ColorRecognitionTCS230PI::minFrequency[3] [private]**

The minimum frequency.

Definition at line 102 of file [ColorRecognitionTCS230PI.h](#).

**4.3.5.3 unsigned char ColorRecognitionTCS230PI::outPin [private]**

The out pin.

Definition at line 97 of file [ColorRecognitionTCS230PI.h](#).

#### 4.3.5.4 unsigned char ColorRecognitionTCS230PI::s2Pin [private]

The s2 pin.

Definition at line 87 of file [ColorRecognitionTCS230PI.h](#).

#### 4.3.5.5 unsigned char ColorRecognitionTCS230PI::s3Pin [private]

The s3 pin.

Definition at line 92 of file [ColorRecognitionTCS230PI.h](#).

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

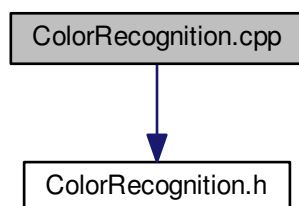
- [ColorRecognitionTCS230PI.h](#)
- [ColorRecognitionTCS230PI.cpp](#)

## 5 File Documentation

### 5.1 ColorRecognition.cpp File Reference

```
#include "ColorRecognition.h"
```

Include dependency graph for ColorRecognition.cpp:



#### Macros

- `#define __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ 1`

#### 5.1.1 Macro Definition Documentation

##### 5.1.1.1 `#define __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ 1`

Arduino - Color Recognition Sensor.

[ColorRecognition.h](#)

The abstract class for the color recognition sensors.

#### Author

Dalmir da Silva [dalmirdasilva@gmail.com](mailto:dalmirdasilva@gmail.com)

Definition at line 12 of file [ColorRecognition.cpp](#).

## 5.2 ColorRecognition.cpp

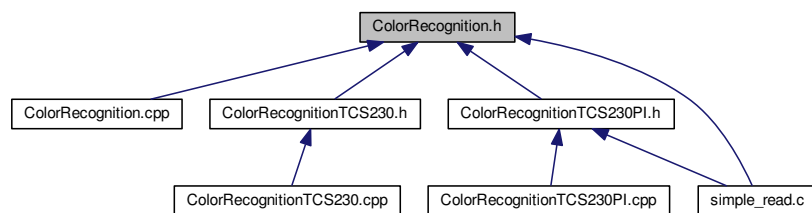
```

00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ 1
00013
00014 #include "ColorRecognition.h"
00015
00016 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ */

```

## 5.3 ColorRecognition.h File Reference

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



### Classes

- class [ColorRecognition](#)

## 5.4 ColorRecognition.h

```

00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_H__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_H__ 1
00013
00014 class ColorRecognition {
00015
00016 public:
00017
00023     virtual unsigned char getRed() = 0;
00024
00030     virtual unsigned char getGreen() = 0;
00031
00037     virtual unsigned char getBlue() = 0;
00038
00044     virtual bool fillRGB(unsigned char buf[3]) = 0;
00045 };
00046
00047 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_H__ */

```

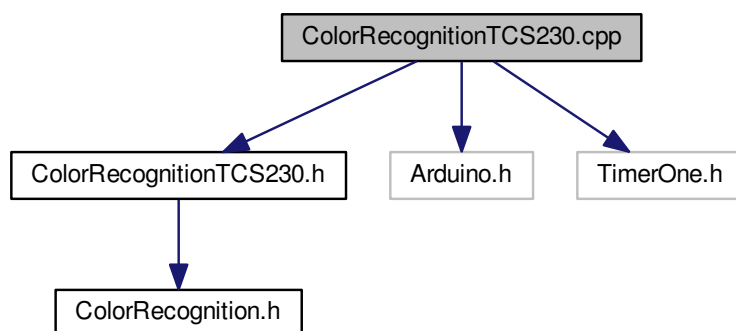
## 5.5 ColorRecognitionTCS230.cpp File Reference

```

#include "ColorRecognitionTCS230.h"
#include <Arduino.h>
#include <TimerOne.h>

```

Include dependency graph for ColorRecognitionTCS230.cpp:



## Macros

- `#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ 1`

### 5.5.1 Macro Definition Documentation

#### 5.5.1.1 `#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ 1`

Arduino - Color Recognition Sensor.

[ColorRecognitionTCS230.h](#)

The abstract class for the Color Recognition TCS230 sensor.

## Author

Dalmir da Silva [dalmirdasilva@gmail.com](mailto:dalmirdasilva@gmail.com)

Definition at line 12 of file [ColorRecognitionTCS230.cpp](#).

## 5.6 ColorRecognitionTCS230.cpp

```

00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ 1
00013
00014 #include "ColorRecognitionTCS230.h"
00015 #include <Arduino.h>
00016 #include <TimerOne.h>
00017
00018 ColorRecognitionTCS230 ColorRecognitionTCS230::instance
00019 ;
00020 void ColorRecognitionTCS230::initialize(unsigned char outPin, unsigned
char s2Pin, unsigned char s3Pin) {
00021     this->s2Pin = s2Pin;
00022     this->s3Pin = s3Pin;
00023     this->outPin = outPin;
00024     this->currentFilter = CLEAR_FILTER;
00025     pinMode(s2Pin, OUTPUT);
00026     pinMode(s3Pin, OUTPUT);
00027     pinMode(outPin, INPUT);
00028     Timer1.initialize();
00029     Timer1.attachInterrupt(ColorRecognitionTCS230::timerInterruptHandler
);
  
```



```

00030     attachInterrupt((outPin - 2),
00031                     ColorRecognitionTCS230::externalInterruptHandler, RISING);
00032 }
00033 void ColorRecognitionTCS230::adjustWhiteBalance() {
00034     delay(4000);
00035     instance.whiteBalanceFrequencies[0] =
00036     instance.lastFrequencies[0];
00037     instance.whiteBalanceFrequencies[1] =
00038     instance.lastFrequencies[1];
00039     instance.whiteBalanceFrequencies[2] =
00040     instance.lastFrequencies[2];
00041 }
00042 void ColorRecognitionTCS230::externalInterruptHandler() {
00043     instance.count++;
00044 }
00045 void ColorRecognitionTCS230::timerInterruptHandler() {
00046     switch (instance.currentFilter) {
00047     case CLEAR_FILTER:
00048         setFilter(RED_FILTER);
00049         break;
00050     case RED_FILTER:
00051         instance.lastFrequencies[0] = instance.
00052         count;
00053         setFilter(GREEN_FILTER);
00054         break;
00055     case GREEN_FILTER:
00056         instance.lastFrequencies[1] = instance.
00057         count;
00058         setFilter(BLUE_FILTER);
00059         break;
00060     case BLUE_FILTER:
00061         instance.lastFrequencies[2] = instance.
00062         count;
00063         setFilter(RED_FILTER);
00064         break;
00065     }
00066     instance.count = 0;
00067     Timer1.setPeriod(1000000);
00068 }
00069 unsigned char ColorRecognitionTCS230::getRed() {
00070     if (lastFrequencies[0] > whiteBalanceFrequencies[0]) {
00071         return 255;
00072     }
00073     return (unsigned char) map(lastFrequencies[0], 0,
00074                               whiteBalanceFrequencies[0], 0, 255);
00075 }
00076 unsigned char ColorRecognitionTCS230::getGreen() {
00077     if (lastFrequencies[1] > whiteBalanceFrequencies[1]) {
00078         return 255;
00079     }
00080     return (unsigned char) map(lastFrequencies[1], 0,
00081                               whiteBalanceFrequencies[1], 0, 255);
00082 }
00083 unsigned char ColorRecognitionTCS230::getBlue() {
00084     if (lastFrequencies[2] > whiteBalanceFrequencies[2]) {
00085         return 255;
00086     }
00087     return (unsigned char) map(lastFrequencies[2], 0,
00088                               whiteBalanceFrequencies[2], 0, 255);
00089 }
00090 bool ColorRecognitionTCS230::fillRGB(unsigned char buf[3]) {
00091     buf[0] = getRed();
00092     buf[1] = getGreen();
00093     buf[2] = getBlue();
00094     return true;
00095 }
00096 void ColorRecognitionTCS230::setFilter(Filter filter) {
00097     unsigned char s2 = LOW, s3 = LOW;
00098     instance.currentFilter = filter;
00099     if (filter == CLEAR_FILTER || filter == GREEN_FILTER) {
00100         s2 = HIGH;
00101     }
00102     if (filter == BLUE_FILTER || filter == GREEN_FILTER) {
00103         s3 = HIGH;
00104     }
00105     digitalWrite(instance.s2Pin, s2);
00106     digitalWrite(instance.s3Pin, s3);
00107 }

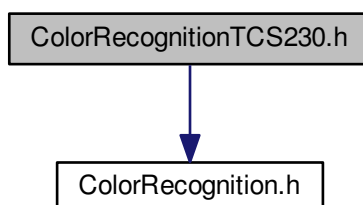
```

```
00107 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ */
```

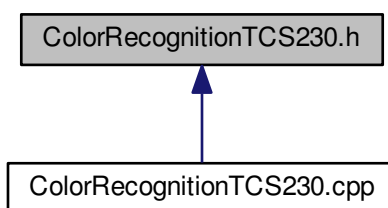
## 5.7 ColorRecognitionTCS230.h File Reference

```
#include <ColorRecognition.h>
```

Include dependency graph for ColorRecognitionTCS230.h:



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



### Classes

- class [ColorRecognitionTCS230](#)

### Macros

- #define [MAX\\_FREQUENCY\\_IN\\_HZ](#) 1000

#### 5.7.1 Macro Definition Documentation

##### 5.7.1.1 #define MAX\_FREQUENCY\_IN\_HZ 1000

Arduino - Color Recognition Sensor.

[ColorRecognitionTCS230.h](#)

The abstract class for the Color Recognition TCS230 sensor.

**Author**

Dalmir da Silva [dalmirdasilva@gmail.com](mailto:dalmirdasilva@gmail.com) In this driver we are assuming the S0 pin is LOW and S1 pin is HIGH. With output frequency at 2%. It saves arduino pins also.

S0	S1	OUTPUT FREQUENCY
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

Also we are assuming the OE pin is LOW, this pin controls the device activation. If OE is LOW the device is enable.

**Output frequency scaling:**

Output-frequency scaling is controlled by two logic inputs, S0 and S1. The internal light-to-frequency converter generates a fixed-pulsewidth pulse train. Scaling is accomplished by internally connecting the pulse-train output of the converter to a series of frequency dividers. Divided outputs are 50%-duty cycle square waves with relative frequency values of 100%, 20%, and 2%. Because division of the output frequency is accomplished by counting pulses of the principal internal frequency, the final-output period represents an average of the multiple periods of the principle frequency.

The output-scaling counter registers are cleared upon the next pulse of the principal frequency after any transition of the S0, S1, S2, S3, and OE lines. The output goes high upon the next subsequent pulse of the principal frequency, beginning a new valid period. This minimizes the time delay between a change on the input lines and the resulting new output period. The response time to an input programming change or to an irradiance step change is one period of new frequency plus 1  $\mu$ S. The scaled output changes both the full-scale frequency and the dark frequency by the selected scale factor. The frequency-scaling function allows the output range to be optimized for a variety of measurement techniques. The scaled-down outputs may be used where only a slower frequency counter is available, such as low-cost microcontroller, or where period measurement techniques are used.

**Measuring the frequency:**

The choice of interface and measurement technique depends on the desired resolution and data acquisition rate. For maximum data-acquisition rate, period-measurement techniques are used. Output data can be collected at a rate of twice the output frequency or one data point every microsecond for full-scale output. Period measurement requires the use of a fast reference clock with available resolution directly related to reference clock rate. Output scaling can be used to increase the resolution for a given clock rate or to maximize resolution as the light input changes. Period measurement is used to measure rapidly varying light levels or to make a very fast measurement of a constant light source. Maximum resolution and accuracy may be obtained using frequency-measurement, pulse-accumulation, or integration techniques. Frequency measurements provide the added benefit of averaging out random- or high-frequency variations (jitter) resulting from noise in the light signal. Resolution is limited mainly by available counter registers and allowable measurement time. Frequency measurement is well suited for slowly varying or constant light levels and for reading average light levels over short periods of time. Integration (the accumulation of pulses over a very long period of time) can be used to measure exposure, the amount of light present in an area over a given time period. When used with a BASIC Stamp, the TCS230's output frequency can be read using the Stamp's statement, as shown in the example code on the front side of this sheet. In this example, and were both pulled "high", enabling the TCS230's fastest output rate. However, this rate can be as much as 600KHz or more at maximum light intensity

MAX: 600KHz (I don't belive) we are usin: 2% of such frequency we are usin: 1 second between the interrupts.

$(600 * 2 / 100) * 1000$  it is too much.

Definition at line 92 of file [ColorRecognitionTCS230.h](#).

**5.8 ColorRecognitionTCS230.h**

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__ 1
00013
00014 #include <ColorRecognition.h>
00015
00092 #define MAX_FRQUENCY_IN_HZ 1000
```

```

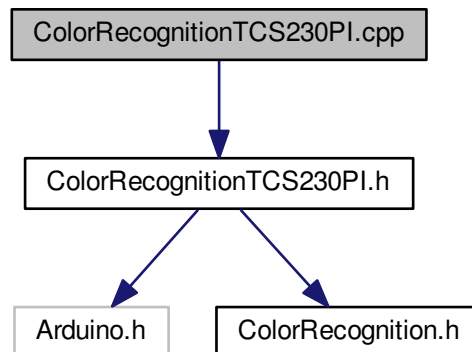
00093
00094 class ColorRecognitionTCS230: public ColorRecognition {
00095 private:
00096
00100     unsigned char s2Pin;
00101
00105     unsigned char s3Pin;
00106
00112     unsigned char outPin;
00113
00117     int count;
00118
00122     int lastFrequencies[3];
00123
00127     int whiteBalanceFrequencies[3];
00128
00132     static ColorRecognitionTCS230 instance;
00133
00134 public:
00135
00139     enum Filter {
00140         RED_FILTER,
00141         GREEN_FILTER,
00142         BLUE_FILTER,
00143         CLEAR_FILTER
00144     };
00145
00149     Filter currentFilter;
00150
00156     static ColorRecognitionTCS230* getInstance() {
00157         return &ColorRecognitionTCS230::instance;
00158     }
00159
00160     virtual ~ColorRecognitionTCS230() {
00161     }
00162
00173     void initialize(unsigned char outPin, unsigned char s2Pin, unsigned char s3Pin);
00174
00180     void adjustWhiteBalance();
00181
00187     unsigned char getRed();
00188
00194     unsigned char getGreen();
00195
00201     unsigned char getBlue();
00202
00208     bool fillRGB(unsigned char buf[3]);
00209
00223     static void setFilter(Filter filter);
00224
00225 private:
00226
00230     ColorRecognitionTCS230()
00231         : s2Pin(0), s3Pin(0), outPin(0), count(0), currentFilter(
00232         CLEAR_FILTER) {
00233         whiteBalanceFrequencies[0] = MAX_FREQUENCY_IN_HZ;
00234         whiteBalanceFrequencies[1] = MAX_FREQUENCY_IN_HZ;
00235         whiteBalanceFrequencies[2] = MAX_FREQUENCY_IN_HZ;
00236     }
00240     static void externalInterruptHandler();
00241
00245     static void timerInterruptHandler();
00246 };
00247
00248 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__ */

```

## 5.9 ColorRecognitionTCS230PI.cpp File Reference

```
#include "ColorRecognitionTCS230PI.h"
```

Include dependency graph for ColorRecognitionTCS230PI.cpp:



#### Macros

- `#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ 1`

#### 5.9.1 Macro Definition Documentation

##### 5.9.1.1 `#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ 1`

Arduino - Color Recognition Sensor.

[ColorRecognitionTCS230PI.h](#)

The abstract class for the Color Recognition TCS230 sensor.

#### Author

Dalmir da Silva [dalmirdasilva@gmail.com](mailto:dalmirdasilva@gmail.com)

Definition at line 12 of file [ColorRecognitionTCS230PI.cpp](#).

#### 5.10 ColorRecognitionTCS230PI.cpp

```

00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ 1
00013
00014 #include "ColorRecognitionTCS230PI.h"
00015
00016 ColorRecognitionTCS230PI::ColorRecognitionTCS230PI(
    unsigned char outPin,
00017     unsigned char s2Pin, unsigned char s3Pin) {
00018     this->s2Pin = s2Pin;
00019     this->s3Pin = s3Pin;
00020     this->outPin = outPin;
00021     pinMode(s2Pin, OUTPUT);
00022     pinMode(s3Pin, OUTPUT);
00023     pinMode(outPin, INPUT);
00024     for (unsigned char i = 0; i < 3; i++) {
00025         minFrequency[i] = 0;
00026         maxFrequency[i] = 1000;
00027     }
00028 }
00029
  
```

```

00030 void ColorRecognitionTCS230PI::adjustWhiteBalance() {
00031     for (unsigned char i = 0; i < 3; i++) {
00032         setFilter((Filter) i);
00033         maxFrequency[i] = getFrequency(255);
00034     }
00035 }
00036
00037 void ColorRecognitionTCS230PI::adjustBlackBalance() {
00038     for (unsigned char i = 0; i < 3; i++) {
00039         setFilter((Filter) i);
00040         minFrequency[i] = getFrequency(255);
00041     }
00042 }
00043
00044 unsigned char ColorRecognitionTCS230PI::getRed() {
00045     setFilter(RED_FILTER);
00046     return (unsigned char) map(getFrequency(SAMPLES),
minFrequency[0], maxFrequency[0], 0, 255);
00047 }
00048
00049 unsigned char ColorRecognitionTCS230PI::getGreen() {
00050     setFilter(GREEN_FILTER);
00051     return (unsigned char) map(getFrequency(SAMPLES),
minFrequency[1], maxFrequency[1], 0, 255);
00052 }
00053
00054 unsigned char ColorRecognitionTCS230PI::getBlue() {
00055     setFilter(BLUE_FILTER);
00056     return (unsigned char) map(getFrequency(SAMPLES),
minFrequency[2], maxFrequency[2], 0, 255);
00057 }
00058
00059 bool ColorRecognitionTCS230PI::fillRGB(unsigned char buf[3]) {
00060     buf[0] = getRed();
00061     buf[1] = getGreen();
00062     buf[2] = getBlue();
00063     return true;
00064 }
00065
00066 void ColorRecognitionTCS230PI::setFilter(
Filter filter) {
00067     unsigned char s2 = LOW, s3 = LOW;
00068     if (filter == CLEAR_FILTER || filter == GREEN_FILTER) {
00069         s2 = HIGH;
00070     }
00071     if (filter == BLUE_FILTER || filter == GREEN_FILTER) {
00072         s3 = HIGH;
00073     }
00074     digitalWrite(s2Pin, s2);
00075     digitalWrite(s3Pin, s3);
00076 }
00077
00078 long ColorRecognitionTCS230PI::getFrequency(unsigned int samples) {
00079     long frequency = 0;
00080     for (unsigned int i = 0; i < samples; i++) {
00081         frequency += 500000 / pulseIn(outPin, HIGH, 250000);
00082     }
00083     return frequency / samples;
00084 }
00085
00086 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ */

```

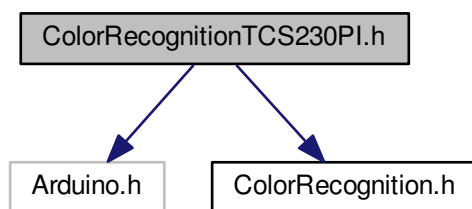
## 5.11 ColorRecognitionTCS230PI.h File Reference

```

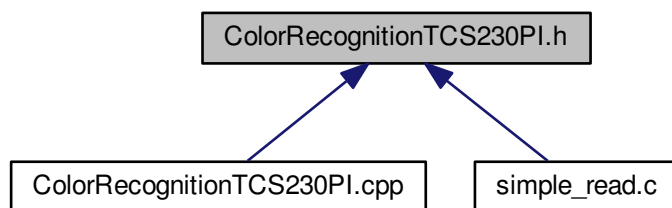
#include <Arduino.h>
#include <ColorRecognition.h>

```

Include dependency graph for ColorRecognitionTCS230PI.h:



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



#### Classes

- class [ColorRecognitionTCS230PI](#)

#### Macros

- `#define` [SAMPLES](#) 32

#### 5.11.1 Macro Definition Documentation

##### 5.11.1.1 `#define` [SAMPLES](#) 32

Arduino - Color Recognition Sensor.

[ColorRecognitionTCS230PI.h](#)

The abstract class for the Color Recognition TCS230 sensor.

#### Author

Dalmir da Silva [dalmirdasilva@gmail.com](mailto:dalmirdasilva@gmail.com) In this driver we are assuming the S0 pin is LOW and S1 pin is HIGH. With output frequency at 2%. It saves arduino pins also.

S0	S1	OUTPUT FREQUENCY
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

Also we are assuming the OE pin is LOW, this pin controls the device activation. If OE is LOW the device is enable.

Output frequency scaling:

Output-frequency scaling is controlled by two logic inputs, S0 and S1. The internal light-to-frequency converter generates a fixed-pulsewidth pulse train. Scaling is accomplished by internally connecting the pulse-train output of the converter to a series of frequency dividers. Divided outputs are 50%-duty cycle square waves with relative frequency values of 100%, 20%, and 2%. Because division of the output frequency is accomplished by counting pulses of the principal internal frequency, the final-output period represents an average of the multiple periods of the principle frequency.

The output-scaling counter registers are cleared upon the next pulse of the principal frequency after any transition of the S0, S1, S2, S3, and OE lines. The output goes high upon the next subsequent pulse of the principal frequency, beginning a new valid period. This minimizes the time delay between a change on the input lines and the resulting new output period. The response time to an input programming change or to an irradiance step change is one period of new frequency plus 1  $\mu$ S. The scaled output changes both the full-scale frequency and the dark frequency by the selected scale factor. The frequency-scaling function allows the output range to be optimized for a variety of measurement techniques. The scaled-down outputs may be used where only a slower frequency counter is available, such as low-cost microcontroller, or where period measurement techniques are used.

Measuring the frequency:

The choice of interface and measurement technique depends on the desired resolution and data acquisition rate. For maximum data-acquisition rate, period-measurement techniques are used. Output data can be collected at a rate of twice the output frequency or one data point every microsecond for full-scale output. Period measurement requires the use of a fast reference clock with available resolution directly related to reference clock rate. Output scaling can be used to increase the resolution for a given clock rate or to maximize resolution as the light input changes. Period measurement is used to measure rapidly varying light levels or to make a very fast measurement of a constant light source. Maximum resolution and accuracy may be obtained using frequency-measurement, pulse-accumulation, or integration techniques. Frequency measurements provide the added benefit of averaging out random- or high-frequency variations (jitter) resulting from noise in the light signal. Resolution is limited mainly by available counter registers and allowable measurement time. Frequency measurement is well suited for slowly varying or constant light levels and for reading average light levels over short periods of time. Integration (the accumulation of pulses over a very long period of time) can be used to measure exposure, the amount of light present in an area over a given time period.

Definition at line 79 of file [ColorRecognitionTCS230PI.h](#).

## 5.12 ColorRecognitionTCS230PI.h

```

00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H__ 1
00013
00014 #include <Arduino.h>
00015 #include <ColorRecognition.h>
00016
00079 #define SAMPLES 32
00080
00081 class ColorRecognitionTCS230PI : public ColorRecognition {
00082 private:
00083
00087     unsigned char s2Pin;
00088
00092     unsigned char s3Pin;
00093
00097     unsigned char outPin;
00098
00102     long minFrequency[3];
00103
00107     long maxFrequency[3];
00108

```



```

00109 public:
00110
00114     enum Filter {
00115         RED_FILTER, GREEN_FILTER, BLUE_FILTER,
00116         CLEAR_FILTER
00117     };
00121     ColorRecognitionTCS230PI(unsigned char outPin, unsigned char s2Pin,
00122                             unsigned char s3Pin);
00123
00129     void adjustWhiteBalance();
00130
00136     void adjustBlackBalance();
00137
00143     unsigned char getRed();
00144
00150     unsigned char getGreen();
00151
00157     unsigned char getBlue();
00158
00164     bool fillRGB(unsigned char buf[3]);
00165
00184     long getFrequency(unsigned int samples);
00185
00199     void setFilter(Filter filter);
00200
00201 };
00202
00203 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H__ */

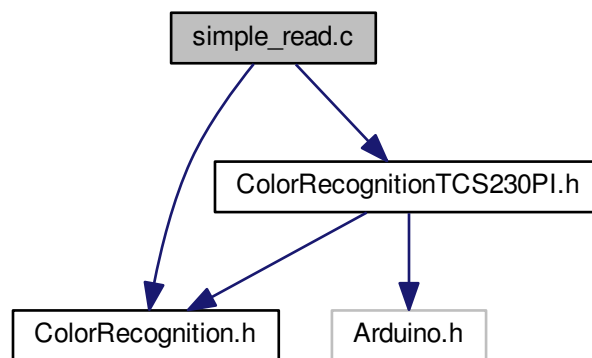
```

## 5.13 simple\_read.c File Reference

```

#include <ColorRecognition.h>
#include <ColorRecognitionTCS230PI.h>
Include dependency graph for simple_read.c:

```



### Functions

- void [setup](#) ()
- void [loop](#) ()

#### 5.13.1 Function Documentation

##### 5.13.1.1 void loop ( )

Definition at line 33 of file [simple\\_read.c](#).

### 5.13.1.2 void setup ( )

Definition at line 4 of file [simple\\_read.c](#).

## 5.14 simple\_read.c

```
00001 #include <ColorRecognition.h>
00002 #include <ColorRecognitionTCS230PI.h>
00003
00004 void setup() {
00005
00006     Serial.begin(9600);
00007
00008     ColorRecognitionTCS230PI tcs230(2, 3, 4);
00009
00010     Serial.println("Adjust white color, show something white to the sensor and press y.");
00011     while(!Serial.available() && Serial.read() != 'y');
00012     Serial.read();
00013     Serial.println("Adjusting...");
00014     tcs230.adjustWhiteBalance();
00015
00016     Serial.println("Adjust black color, show something black to the sensor and press y.");
00017     while(!Serial.available() && Serial.read() != 'y');
00018     Serial.read();
00019     Serial.println("Adjusting...");
00020     tcs230.adjustBlackBalance();
00021
00022     while (1) {
00023         Serial.print("Read: ");
00024         Serial.println(tcs230.getRed());
00025         Serial.print("Green: ");
00026         Serial.println(tcs230.getGreen());
00027         Serial.print("Blue ");
00028         Serial.println(tcs230.getBlue());
00029         delay(3000);
00030     }
00031 }
00032
00033 void loop() {
00034 }
```

## Index

- `__ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__`
  - `ColorRecognition.cpp`, 12
- `__ARDUINO_DRIVER_COLOR_RECOGNITION_TC230PI_CPP__`
  - `ColorRecognitionTCS230PI.cpp`, 19
- `__ARDUINO_DRIVER_COLOR_RECOGNITION_TC230_CPP__`
  - `ColorRecognitionTCS230.cpp`, 14
- `~ColorRecognitionTCS230`
  - `ColorRecognitionTCS230`, 5
- `adjustBlackBalance`
  - `ColorRecognitionTCS230PI`, 10
- `adjustWhiteBalance`
  - `ColorRecognitionTCS230`, 5
  - `ColorRecognitionTCS230PI`, 10
- `BLUE_FILTER`
  - `ColorRecognitionTCS230`, 5
  - `ColorRecognitionTCS230PI`, 10
- `CLEAR_FILTER`
  - `ColorRecognitionTCS230`, 5
  - `ColorRecognitionTCS230PI`, 10
- `ColorRecognition`, 2
  - `fillRGB`, 2
  - `getBlue`, 2
  - `getGreen`, 3
  - `getRed`, 3
- `ColorRecognition.cpp`, 12, 13
  - `__ARDUINO_DRIVER_COLOR_RECOGNITION_TC230PI_CPP__`, 12
- `ColorRecognition.h`, 13
- `ColorRecognitionTCS230`, 3
  - `~ColorRecognitionTCS230`, 5
  - `adjustWhiteBalance`, 5
  - `BLUE_FILTER`, 5
  - `CLEAR_FILTER`, 5
  - `ColorRecognitionTCS230`, 5
  - `count`, 7
  - `currentFilter`, 7
  - `externalInterruptHandler`, 5
  - `fillRGB`, 6
  - `Filter`, 5
  - `GREEN_FILTER`, 5
  - `getBlue`, 6
  - `getGreen`, 6
  - `getInstance`, 6
  - `getRed`, 6
  - `initialize`, 6
  - `instance`, 7
  - `lastFrequencies`, 7
  - `outPin`, 7
  - `RED_FILTER`, 5
  - `s2Pin`, 8
  - `s3Pin`, 8
  - `setFilter`, 7
  - `timerInterruptHandler`, 7
  - `whiteBalanceFrequencies`, 8
- `ColorRecognitionTCS230.cpp`, 13, 14
  - `__ARDUINO_DRIVER_COLOR_RECOGNITION_TC230PI_CPP__`, 14
- `ColorRecognitionTCS230.h`, 16, 17
  - `MAX_FREQUENCY_IN_HZ`, 16
- `ColorRecognitionTCS230PI`, 8
  - `adjustBlackBalance`, 10
  - `adjustWhiteBalance`, 10
  - `BLUE_FILTER`, 10
  - `CLEAR_FILTER`, 10
  - `ColorRecognitionTCS230PI`, 10
  - `fillRGB`, 10
  - `Filter`, 9
  - `GREEN_FILTER`, 9
  - `getBlue`, 10
  - `getFrequency`, 10
  - `getGreen`, 11
  - `getRed`, 11
  - `maxFrequency`, 11
  - `minFrequency`, 11
  - `outPin`, 11
  - `RED_FILTER`, 9
  - `s2Pin`, 11
  - `s3Pin`, 12
  - `setFilter`, 11
- `ColorRecognitionTCS230PI.cpp`, 18, 19
  - `__ARDUINO_DRIVER_COLOR_RECOGNITION_TC230PI_CPP__`, 19
- `ColorRecognitionTCS230PI.h`, 20, 22
  - `SAMPLES`, 21
- `count`
  - `ColorRecognitionTCS230`, 7
- `currentFilter`
  - `ColorRecognitionTCS230`, 7
- `externalInterruptHandler`
  - `ColorRecognitionTCS230`, 5
- `fillRGB`
  - `ColorRecognition`, 2
  - `ColorRecognitionTCS230`, 6
  - `ColorRecognitionTCS230PI`, 10
- `Filter`
  - `ColorRecognitionTCS230`, 5
  - `ColorRecognitionTCS230PI`, 9
- `GREEN_FILTER`
  - `ColorRecognitionTCS230`, 5
  - `ColorRecognitionTCS230PI`, 9
- `getBlue`
  - `ColorRecognition`, 2
  - `ColorRecognitionTCS230`, 6

- ColorRecognitionTCS230PI, [10](#)
- getFrequency
  - ColorRecognitionTCS230PI, [10](#)
- getGreen
  - ColorRecognition, [3](#)
  - ColorRecognitionTCS230, [6](#)
  - ColorRecognitionTCS230PI, [11](#)
- getInstance
  - ColorRecognitionTCS230, [6](#)
- getRed
  - ColorRecognition, [3](#)
  - ColorRecognitionTCS230, [6](#)
  - ColorRecognitionTCS230PI, [11](#)
- initialize
  - ColorRecognitionTCS230, [6](#)
- instance
  - ColorRecognitionTCS230, [7](#)
- lastFrequencies
  - ColorRecognitionTCS230, [7](#)
- loop
  - simple\_read.c, [23](#)
- MAX\_FREQUENCY\_IN\_HZ
  - ColorRecognitionTCS230.h, [16](#)
- maxFrequency
  - ColorRecognitionTCS230PI, [11](#)
- minFrequency
  - ColorRecognitionTCS230PI, [11](#)
- outPin
  - ColorRecognitionTCS230, [7](#)
  - ColorRecognitionTCS230PI, [11](#)
- RED\_FILTER
  - ColorRecognitionTCS230, [5](#)
  - ColorRecognitionTCS230PI, [9](#)
- s2Pin
  - ColorRecognitionTCS230, [8](#)
  - ColorRecognitionTCS230PI, [11](#)
- s3Pin
  - ColorRecognitionTCS230, [8](#)
  - ColorRecognitionTCS230PI, [12](#)
- SAMPLES
  - ColorRecognitionTCS230PI.h, [21](#)
- setFilter
  - ColorRecognitionTCS230, [7](#)
  - ColorRecognitionTCS230PI, [11](#)
- setup
  - simple\_read.c, [23](#)
- simple\_read.c, [23](#), [24](#)
  - loop, [23](#)
  - setup, [23](#)
- timerInterruptHandler
  - ColorRecognitionTCS230, [7](#)
- whiteBalanceFrequencies
  - ColorRecognitionTCS230, [8](#)