

LabRoboticsProject

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	Angle Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Member Enumeration Documentation . . . . .	6
3.1.2.1	ANGLE_TYPE . . . . .	6
3.1.3	Constructor & Destructor Documentation . . . . .	6
3.1.3.1	Angle() [1/2] . . . . .	7
3.1.3.2	Angle() [2/2] . . . . .	7
3.1.4	Member Function Documentation . . . . .	7
3.1.4.1	add() . . . . .	7
3.1.4.2	copy() . . . . .	7
3.1.4.3	degToRad() . . . . .	8
3.1.4.4	get() . . . . .	8
3.1.4.5	getType() . . . . .	8
3.1.4.6	getTypeName() . . . . .	8
3.1.4.7	normalize() . . . . .	8
3.1.4.8	operator+() . . . . .	8
3.1.4.9	operator+=() . . . . .	9

3.1.4.10	<a href="#">operator-()</a>	9
3.1.4.11	<a href="#">operator==()</a>	9
3.1.4.12	<a href="#">operator=()</a>	11
3.1.4.13	<a href="#">radToDeg()</a>	11
3.1.4.14	<a href="#">set()</a>	11
3.1.4.15	<a href="#">setType()</a>	12
3.1.4.16	<a href="#">sub()</a>	12
3.1.4.17	<a href="#">toDeg()</a>	12
3.1.4.18	<a href="#">toRad()</a>	13
3.1.5	<a href="#">Friends And Related Function Documentation</a>	13
3.1.5.1	<a href="#">operator&lt;&lt;</a>	13
3.2	<a href="#">Configuration2&lt; T1 &gt; Class Template Reference</a>	13
3.2.1	<a href="#">Detailed Description</a>	14
3.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	15
3.2.2.1	<a href="#">Configuration2() [1/3]</a>	15
3.2.2.2	<a href="#">Configuration2() [2/3]</a>	15
3.2.2.3	<a href="#">Configuration2() [3/3]</a>	15
3.2.3	<a href="#">Member Function Documentation</a>	16
3.2.3.1	<a href="#">angle() [1/2]</a>	16
3.2.3.2	<a href="#">angle() [2/2]</a>	16
3.2.3.3	<a href="#">distance()</a>	16
3.2.3.4	<a href="#">EuDistance()</a>	17
3.2.3.5	<a href="#">MaDistance()</a>	17
3.2.3.6	<a href="#">offset() [1/3]</a>	18
3.2.3.7	<a href="#">offset() [2/3]</a>	18
3.2.3.8	<a href="#">offset() [3/3]</a>	18
3.2.3.9	<a href="#">offset_angle()</a>	19
3.2.3.10	<a href="#">offset_x()</a>	19
3.2.3.11	<a href="#">offset_y()</a>	20
3.2.3.12	<a href="#">x() [1/2]</a>	20

3.2.3.13	<a href="#">x()</a> [2/2]	20
3.2.3.14	<a href="#">y()</a> [1/2]	21
3.2.3.15	<a href="#">y()</a> [2/2]	21
3.2.4	<a href="#">Friends And Related Function Documentation</a>	21
3.2.4.1	<a href="#">operator&lt;&lt;</a>	21
3.3	<a href="#">dubins Class Reference</a>	22
3.3.1	<a href="#">Constructor &amp; Destructor Documentation</a>	22
3.3.1.1	<a href="#">dubins()</a> [1/3]	22
3.3.1.2	<a href="#">dubins()</a> [2/3]	22
3.3.1.3	<a href="#">dubins()</a> [3/3]	23
3.3.1.4	<a href="#">~dubins()</a>	23
3.4	<a href="#">Point2&lt; T &gt; Class Template Reference</a>	23
3.4.1	<a href="#">Detailed Description</a>	24
3.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	24
3.4.2.1	<a href="#">Point2()</a> [1/2]	24
3.4.2.2	<a href="#">Point2()</a> [2/2]	24
3.4.3	<a href="#">Member Function Documentation</a>	25
3.4.3.1	<a href="#">distance()</a>	25
3.4.3.2	<a href="#">EuDistance()</a>	25
3.4.3.3	<a href="#">MaDistance()</a>	26
3.4.3.4	<a href="#">offset()</a> [1/3]	26
3.4.3.5	<a href="#">offset()</a> [2/3]	26
3.4.3.6	<a href="#">offset()</a> [3/3]	27
3.4.3.7	<a href="#">offset_x()</a>	27
3.4.3.8	<a href="#">offset_y()</a>	28
3.4.3.9	<a href="#">x()</a> [1/2]	28
3.4.3.10	<a href="#">x()</a> [2/2]	28
3.4.3.11	<a href="#">y()</a> [1/2]	29
3.4.3.12	<a href="#">y()</a> [2/2]	29
3.4.4	<a href="#">Friends And Related Function Documentation</a>	29

3.4.4.1	operator<<	29
3.5	Settings Class Reference	30
3.5.1	Member Enumeration Documentation	32
3.5.1.1	InputType	32
3.5.1.2	Pattern	32
3.5.2	Constructor & Destructor Documentation	32
3.5.2.1	Settings()	32
3.5.3	Member Function Documentation	32
3.5.3.1	isListOfImages()	32
3.5.3.2	nextImage()	33
3.5.3.3	read()	33
3.5.3.4	readStringList()	33
3.5.3.5	validate()	34
3.5.3.6	write()	34
3.5.4	Member Data Documentation	35
3.5.4.1	aspectRatio	35
3.5.4.2	atImageList	35
3.5.4.3	boardSize	35
3.5.4.4	calibFixPrincipalPoint	35
3.5.4.5	calibrationPattern	35
3.5.4.6	calibZeroTangentDist	36
3.5.4.7	cameraID	36
3.5.4.8	delay	36
3.5.4.9	fixK1	36
3.5.4.10	fixK2	36
3.5.4.11	fixK3	36
3.5.4.12	fixK4	37
3.5.4.13	fixK5	37
3.5.4.14	flag	37
3.5.4.15	flipVertical	37

3.5.4.16	<a href="#">goodInput</a>	37
3.5.4.17	<a href="#">imageList</a>	37
3.5.4.18	<a href="#">input</a>	37
3.5.4.19	<a href="#">inputCapture</a>	38
3.5.4.20	<a href="#">inputType</a>	38
3.5.4.21	<a href="#">nrFrames</a>	38
3.5.4.22	<a href="#">outputFileName</a>	38
3.5.4.23	<a href="#">showUndistorsed</a>	38
3.5.4.24	<a href="#">squareSize</a>	38
3.5.4.25	<a href="#">useFisheye</a>	39
3.5.4.26	<a href="#">writeExtrinsics</a>	39
3.5.4.27	<a href="#">writePoints</a>	39
3.6	<a href="#">Tuple&lt; T &gt; Class Template Reference</a>	39
3.6.1	<a href="#">Detailed Description</a>	40
3.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	40
3.6.2.1	<a href="#">Tuple() [1/2]</a>	40
3.6.2.2	<a href="#">Tuple() [2/2]</a>	40
3.6.3	<a href="#">Member Function Documentation</a>	41
3.6.3.1	<a href="#">add()</a>	41
3.6.3.2	<a href="#">distance()</a>	41
3.6.3.3	<a href="#">EuDistance()</a>	41
3.6.3.4	<a href="#">get()</a>	42
3.6.3.5	<a href="#">MaDistance()</a>	42
3.6.3.6	<a href="#">remove()</a>	43
3.6.3.7	<a href="#">set()</a>	43
3.6.3.8	<a href="#">size()</a>	43
3.6.4	<a href="#">Friends And Related Function Documentation</a>	43
3.6.4.1	<a href="#">operator&lt;&lt;</a>	44

<b>4 File Documentation</b>	<b>45</b>
4.1 src/calibration.cc File Reference	45
4.1.1 Function Documentation	46
4.1.1.1 calcBoardCornerPositions()	46
4.1.1.2 calibration()	46
4.1.1.3 computeReprojectionErrors()	46
4.1.1.4 read()	47
4.1.1.5 runCalibration()	47
4.1.1.6 runCalibrationAndSave()	48
4.1.1.7 saveCameraParams()	49
4.2 src/calibration.hh File Reference	49
4.2.1 Detailed Description	50
4.2.2 Enumeration Type Documentation	50
4.2.2.1 anonymous enum	50
4.2.3 Function Documentation	51
4.2.3.1 calibration()	51
4.2.3.2 runCalibrationAndSave()	51
4.3 src/calibration_run.cc File Reference	52
4.3.1 Function Documentation	52
4.3.1.1 main()	52
4.4 src/create_xml.cc File Reference	52
4.4.1 Function Documentation	53
4.4.1.1 main()	53
4.5 src/detection.cc File Reference	53
4.5.1 Macro Definition Documentation	54
4.5.1.1 WAIT	54
4.5.2 Function Documentation	54
4.5.2.1 crop_number_section()	54
4.5.2.2 detection()	54
4.5.2.3 erode_dilation()	55



4.5.2.4	<a href="#">find_contours()</a> . . . . .	55
4.5.2.5	<a href="#">load_number_template()</a> . . . . .	55
4.5.2.6	<a href="#">number_recognition()</a> . . . . .	56
4.5.2.7	<a href="#">save_convex_hull()</a> . . . . .	56
4.5.2.8	<a href="#">shape_detection()</a> . . . . .	56
4.5.3	<a href="#">Variable Documentation</a> . . . . .	57
4.5.3.1	<a href="#">fs_xml</a> . . . . .	57
4.5.3.2	<a href="#">templates</a> . . . . .	57
4.5.3.3	<a href="#">xml_settings</a> . . . . .	57
4.6	<a href="#">src/detection.hh File Reference</a> . . . . .	57
4.6.1	<a href="#">Function Documentation</a> . . . . .	58
4.6.1.1	<a href="#">crop_number_section()</a> . . . . .	58
4.6.1.2	<a href="#">detection()</a> . . . . .	59
4.6.1.3	<a href="#">erode_dilation()</a> . . . . .	59
4.6.1.4	<a href="#">find_contours()</a> . . . . .	59
4.6.1.5	<a href="#">load_number_template()</a> . . . . .	60
4.6.1.6	<a href="#">number_recognition()</a> . . . . .	60
4.6.1.7	<a href="#">save_convex_hull()</a> . . . . .	60
4.6.1.8	<a href="#">shape_detection()</a> . . . . .	61
4.7	<a href="#">src/detection_run.cc File Reference</a> . . . . .	61
4.7.1	<a href="#">Function Documentation</a> . . . . .	61
4.7.1.1	<a href="#">main()</a> . . . . .	61
4.8	<a href="#">src/dubins.cc File Reference</a> . . . . .	62
4.9	<a href="#">src/dubins.hh File Reference</a> . . . . .	62
4.10	<a href="#">src/main.cc File Reference</a> . . . . .	63
4.10.1	<a href="#">Function Documentation</a> . . . . .	63
4.10.1.1	<a href="#">main()</a> . . . . .	63
4.11	<a href="#">src/math.cc File Reference</a> . . . . .	64
4.12	<a href="#">src/math.hh File Reference</a> . . . . .	64
4.12.1	<a href="#">Macro Definition Documentation</a> . . . . .	65

4.12.1.1	DEGTORAD	66
4.12.1.2	RADTODEG	66
4.12.2	Enumeration Type Documentation	66
4.12.2.1	DISTANCE_TYPE	66
4.12.3	Function Documentation	66
4.12.3.1	pow2()	66
4.13	src/unwrapping.cc File Reference	66
4.13.1	Macro Definition Documentation	67
4.13.1.1	area_ratio	67
4.13.1.2	WAIT	67
4.13.2	Function Documentation	67
4.13.2.1	distance()	67
4.13.2.2	loadCoefficients()	68
4.13.2.3	swap()	68
4.13.2.4	unwrapping()	68
4.13.3	Variable Documentation	69
4.13.3.1	xml_settings	69
4.14	src/unwrapping.hh File Reference	69
4.14.1	Function Documentation	70
4.14.1.1	loadCoefficients()	70
4.14.1.2	unwrapping()	70
4.15	src/unwrapping_run.cc File Reference	71
4.15.1	Function Documentation	71
4.15.1.1	main()	71
4.16	src/utls.cc File Reference	71
4.16.1	Function Documentation	71
4.16.1.1	my_imshow()	72
4.17	src/utls.hh File Reference	72
4.17.1	Macro Definition Documentation	72
4.17.1.1	INFO	73
4.17.2	Function Documentation	73
4.17.2.1	my_imshow()	73

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Angle</a>	This class allows to save and handle angles. It supports DEG and RAD, operations such as addition and subtraction with operators overloading, conversion from RAD to DEG and viceversa and normalization of the angle . . . . .	5
<a href="#">Configuration2&lt; T1 &gt;</a>	This class stores a configuration, that is a point and an angle . . . . .	13
<a href="#">dubins</a>	. . . . .	22
<a href="#">Point2&lt; T &gt;</a>	Class that stores two value to construct a point in 2D. The value is saved in a <a href="#">Tuple</a> . . . . .	23
<a href="#">Settings</a>	. . . . .	30
<a href="#">Tuple&lt; T &gt;</a>	. . . . .	39



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

src/calibration.cc . . . . .	45
src/calibration.hh . . . . .	
Library for calibration . . . . .	49
src/calibration_run.cc . . . . .	52
src/create_xml.cc . . . . .	52
src/detection.cc . . . . .	53
src/detection.hh . . . . .	57
src/detection_run.cc . . . . .	61
src/dubins.cc . . . . .	62
src/dubins.hh . . . . .	62
src/main.cc . . . . .	63
src/math.cc . . . . .	64
src/math.hh . . . . .	64
src/unwrapping.cc . . . . .	66
src/unwrapping.hh . . . . .	69
src/unwrapping_run.cc . . . . .	71
src/utls.cc . . . . .	71
src/utls.hh . . . . .	72



## Chapter 3

# Class Documentation

### 3.1 Angle Class Reference

This class allows to save and handle angles. It supports DEG and RAD, operations such as addition and subtraction with operators overloading, conversion from RAD to DEG and viceversa and normalization of the angle.

```
#include <maths.hh>
```

#### Public Types

- enum `ANGLE_TYPE` { `DEG`, `RAD`, `INVALID` }

#### Public Member Functions

- `Angle ()`  
*A void constructor to create an angle.*
- `Angle (double _th, ANGLE_TYPE _type)`  
*This constructor takes the angle value and the type of angle and stores them. It also normalize the angle in case is above 2pi (360°) or below 0.*
- `double get () const`  
*Returns the dimension of the angle.*
- `ANGLE_TYPE getType () const`  
*Returns the type of the angle.*
- `string getTypeName () const`
- `template<class T >`  
`void set (const T _th)`  
*Set the value of the angle.*
- `void setType (ANGLE_TYPE _type)`  
*Set the type of the angle.*
- `double degToRad ()`  
*Convert and store the angle from DEG to RAD.*
- `double radToDeg ()`  
*Converts and stores the angle from RAD to DEG.*
- `double toRad () const`  
*Converts but does not store the value of the angle from DEG to RAD.*

- double `toDeg ()` const  
*Converts but does not store the value of the angle from RAD to DEG.*
- void `normalize ()`  
*Normalize the angle, that is to set it in  $[0, 2\pi]$  or  $[0, 360]$ .*
- `Angle add` (const `Angle` phi)  
*Sums and angle to this one. In the process a new angle is created so `normalize ()` is also called.*
- `Angle sub` (const `Angle` phi)  
*Subtracts and angle to this one. In the process a new angle is created so `normalize ()` is also called.*
- `Angle copy` (const `Angle` phi)  
*Copies an angle to this one. In the process a new angle is created so `normalize ()` is also called.*
- `Angle operator+` (const `Angle` phi)
- `Angle operator-` (const `Angle` phi)
- `Angle operator=` (const `Angle` phi)
- `Angle & operator+=` (const `Angle` phi)
- `Angle & operator-=` (const `Angle` phi)

## Friends

- ostream & `operator<<` (ostream &out, const `Angle` &data)

### 3.1.1 Detailed Description

This class allows to save and handle angles. It supports DEG and RAD, operations such as addition and subtraction with operators overloading, conversion from RAD to DEG and viceversa and normalization of the angle.

### 3.1.2 Member Enumeration Documentation

#### 3.1.2.1 ANGLE\_TYPE

```
enum Angle::ANGLE_TYPE
```

#### Enumerator

DEG	
RAD	
INVALID	

### 3.1.3 Constructor & Destructor Documentation



### 3.1.3.1 Angle() [1/2]

```
Angle::Angle ( ) [inline]
```

A void constructor to create an angle.

### 3.1.3.2 Angle() [2/2]

```
Angle::Angle (
    double _th,
    ANGLE_TYPE _type ) [inline]
```

This constructor takes the angle value and the type of angle and stores them. It also normalize the angle in case is above 2pi (360°) or below 0.

#### Parameters

in	<i>_th</i>	The dimension of the angle.
in	<i>_type</i>	The type of the angle.

## 3.1.4 Member Function Documentation

### 3.1.4.1 add()

```
Angle Angle::add (
    const Angle phi ) [inline]
```

Sums and angle to this one. In the process a new angle is created so `normalize()` is also called.

#### Parameters

in	<i>phi</i>	The angle to be summed.
----	------------	-------------------------

#### Returns

The angle summed.

### 3.1.4.2 copy()

```
Angle Angle::copy (
    const Angle phi ) [inline]
```

Copies an angle to this one. In the process a new angle is created so `normalize()` is also called.

**Parameters**

in	<i>phi</i>	The angle to be copied.
----	------------	-------------------------

**Returns**

The new angle.

**3.1.4.3 degToRad()**

```
double Angle::degToRad ( ) [inline]
```

Convert and store the angle from DEG to RAD.

**Returns**

The value of the angle.

**3.1.4.4 get()**

```
double Angle::get ( ) const [inline]
```

Returns the dimension of the angle.

**3.1.4.5 getType()**

```
ANGLE_TYPE Angle::getType ( ) const [inline]
```

Returns the type of the angle.

**3.1.4.6 getTypeName()**

```
string Angle::getTypeName ( ) const [inline]
```

<Returns a string that tells the type of angle.

**3.1.4.7 normalize()**

```
void Angle::normalize ( ) [inline]
```

Normalize the angle, that is to set it in  $[0, 2\pi]$  or  $[0, 360]$ .

**3.1.4.8 operator+()**

```
Angle Angle::operator+ (
    const Angle phi ) [inline]
```

This function overload the operator +. It simply calls the [add\(\)](#) function.

**Parameters**

in	<i>phi</i>	The angle to be summed.
----	------------	-------------------------

**Returns**

The angle summed.

**3.1.4.9 operator+=()**

```
Angle& Angle::operator+= (
    const Angle phi ) [inline]
```

This function overload the operator +. It simply calls the `add()` function and then assign the result to this.

**Parameters**

in	<i>phi</i>	The angle to be summed.
----	------------	-------------------------

**Returns**

`this.`

**3.1.4.10 operator-()**

```
Angle Angle::operator- (
    const Angle phi ) [inline]
```

This function overload the operator -. It simply calls the `sub()` function.

**Parameters**

in	<i>phi</i>	The angle to be subtracted.
----	------------	-----------------------------

**Returns**

The angle subtracted.

**3.1.4.11 operator-=()**

```
Angle& Angle::operator-= (
    const Angle phi ) [inline]
```

This function overload the operator -. It simply calls the `sub()` function and then assign the result to this.

**Parameters**

in	<i>phi</i>	The angle to be subtracted.
----	------------	-----------------------------

**Returns**

this.

**3.1.4.12 operator=()**

```
Angle Angle::operator= (
    const Angle phi ) [inline]
```

This function overload the operator -. It simply calls the `sub()` function.

**Parameters**

in	<i>phi</i>	The angle to be copied.
----	------------	-------------------------

**Returns**

The new angle.

**3.1.4.13 radToDeg()**

```
double Angle::radToDeg ( ) [inline]
```

Converts and stores the angle from RAD to DEG.

**Returns**

The value of the angle.

**3.1.4.14 set()**

```
template<class T >
void Angle::set (
    const T _th ) [inline]
```

Set the value of the angle.

## Template Parameters

<i>T</i>	The programming type for the value to be stored. It's then cast to <code>double</code> .
----------	--

## Parameters

in	↔	The dimension of the angle to be stored.
	↔ <i>th</i>	

## 3.1.4.15 setType()

```
void Angle::setType (
    ANGLE_TYPE _type ) [inline]
```

Set the type of the angle.

## Parameters

in	↔	The type of the angle to be stored.
	↔ <i>th</i>	

## 3.1.4.16 sub()

```
Angle Angle::sub (
    const Angle phi ) [inline]
```

Subtracts and angle to this one. In the process a new angle is created so `normalize()` is also called.

## Parameters

in	<i>phi</i>	The angle to be subtracted.
----	------------	-----------------------------

## Returns

The angle subtracted.

## 3.1.4.17 toDeg()

```
double Angle::toDeg ( ) const [inline]
```

Converts but does not store the value of the angle from RAD to DEG.

**Returns**

The value of the angle

**3.1.4.18 toRad()**

```
double Angle::toRad ( ) const [inline]
```

Converts but does not store the value of the angle from DEG to RAD.

**Returns**

The value of the angle

**3.1.5 Friends And Related Function Documentation****3.1.5.1 operator<<**

```
ostream& operator<< (
    ostream & out,
    const Angle & data ) [friend]
```

This function overload the << operator so to print with `std::cout` the most essential info, that is the dimension and the type of angle.

**Parameters**

in	<i>out</i>	The out stream.
in	<i>data</i>	The angle to print.

**Returns**

An output stream to be printed.

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

- [src/maths.hh](#)

**3.2 Configuration2< T1 > Class Template Reference**

This class stores a configuration, that is a point and an angle.

```
#include <maths.hh>
```

## Public Member Functions

- [Configuration2](#) ()  
*Default constructor that use as point (0,0) and as angle 0 RAD.*
- [Configuration2](#) (const T1 \_x, const T1 \_y, const [Angle](#) \_th)  
*Default constructor that takes the coordinates, the angle, and stores them.*
- [Configuration2](#) (const [Point2](#)< T1 > P, const [Angle](#) \_th)  
*Default constructor that takes the point, the angle, and stores them.*
- T1 [x](#) () const
- T1 [y](#) () const
- [Angle](#) [angle](#) () const
- int [x](#) (const T1 \_x)  
*This function stores a new value for the abscissa.*
- int [y](#) (const T1 \_y)  
*This function stores a new value for the ordinate.*
- void [angle](#) (const [Angle](#) \_th)  
*This function stores a new value for the angle.*
- template<class T2 >  
int [offset](#) (const T2 \_offset, const [Angle](#) phi, const [Angle](#) \_th)  
*This function compute the offset of the point given a vector, that is the lenght of the vector and its angle. The angle must be an [Angle](#) variable. It takes also another [Angle](#) to change the [Angle](#) in the configuration.*
- int [offset](#) ([Configuration2](#)< T1 > p)  
*This function compute the offset of the point given another [Configuration2](#).*
- int [offset](#) ([Point2](#)< T1 > p, const [Angle](#) \_th=[Angle](#)())  
*This function compute the offset of the point given a [Point2](#) containing the offsets for the abscissa and the ordiante and an [Angle](#) to change the [Angle](#) in the configuration.*
- int [offset\\_x](#) (const T1 \_offset)  
*Function to add an offset to the abscissa.*
- int [offset\\_y](#) (const [Angle](#) \_offset)  
*Function to add an offset to the ordinate.*
- void [offset\\_angle](#) (const [Angle](#) \_th)  
*Function to add an offset to the angle.*
- template<class T2 >  
[Tuple](#)< double > [distance](#) ([Configuration2](#)< T2 > B, [DISTANCE\\_TYPE](#) dist\_type=[EUCLIDEAN](#))  
*Wrapper to compute different distances. T2 The type of the elements in the second [Configuration2](#).*
- template<class T2 >  
[Tuple](#)< double > [EuDistance](#) ([Configuration2](#)< T2 > B)  
*Function that compute the Euclidean Distance between two configurations. T2 The type of the elements in the second [Configuration2](#).*
- template<class T2 >  
[Tuple](#)< double > [MaDistance](#) ([Configuration2](#)< T2 > B)  
*Function that compute the Manhattan Distance between two configurations. T2 The type of the elements in the second [Configuration2](#).*

## Friends

- ostream & [operator](#)<< (ostream &out, const [Configuration2](#) &data)  
*Overload of operator << to output the content of a [Configuration2](#).*

### 3.2.1 Detailed Description

```
template<class T1>
class Configuration2< T1 >
```

This class stores a configuration, that is a point and an angle.



## Template Parameters

<i>T1</i>	The type of the coordinates.
-----------	------------------------------

## 3.2.2 Constructor &amp; Destructor Documentation

## 3.2.2.1 Configuration2() [1/3]

```
template<class T1>
Configuration2< T1 >::Configuration2 ( ) [inline]
```

Default constructor that use as point (0,0) and as angle 0 RAD.

## 3.2.2.2 Configuration2() [2/3]

```
template<class T1>
Configuration2< T1 >::Configuration2 (
    const T1 _x,
    const T1 _y,
    const Angle _th ) [inline]
```

Default constructor that takes the coordinates, the angle, and stores them.

## Parameters

in	↔ _x	The abscissa coordinate.
in	↔ _y	The ordinate coordinate.
in	↔ _th	The angle.

## 3.2.2.3 Configuration2() [3/3]

```
template<class T1>
Configuration2< T1 >::Configuration2 (
    const Point2< T1 > P,
    const Angle _th ) [inline]
```

Default constructor that takes the point, the angle, and stores them.

**Parameters**

in	$P$	The coordinates.
in	$\leftarrow$ $\leftarrow$ $th$	The angle.

**3.2.3 Member Function Documentation****3.2.3.1 angle()** [1/2]

```
template<class T1>
Angle Configuration2< T1 >::angle ( ) const [inline]
```

**Returns**

The angle.

**3.2.3.2 angle()** [2/2]

```
template<class T1>
void Configuration2< T1 >::angle (
    const Angle _th ) [inline]
```

This function stores a new value for the angle.

**Parameters**

in	$\leftarrow$ $\leftarrow$ $th$	The value to be stored.
----	--------------------------------------	-------------------------

**Returns**

1 if everything went ok, 0 otherwise.

**3.2.3.3 distance()**

```
template<class T1>
template<class T2 >
```

```

Tuple<double> Configuration2< T1 >::distance (
    Configuration2< T2 > B,
    DISTANCE_TYPE dist_type = EUCLIDEAN ) [inline]

```

Wrapper to compute different distances. T2 The type of the elements in the second [Configuration2](#).

#### Parameters

in	<i>B</i>	The second <a href="#">Configuration2</a> to use for computing the distance.
in	<i>dist</i>	The type of distance to be computed.

#### Returns

The distance between the two configurations.

#### 3.2.3.4 EuDistance()

```

template<class T1>
template<class T2 >
Tuple<double> Configuration2< T1 >::EuDistance (
    Configuration2< T2 > B ) [inline]

```

Function that compute the Euclidean Distance between two configurations. T2 The type of the elements in the second [Configuration2](#).

#### Parameters

in	<i>B</i>	the second <a href="#">Configuration2</a> to use for computing the distance.
----	----------	--

#### Returns

The Euclidean distance between the two configurations.

#### 3.2.3.5 MaDistance()

```

template<class T1>
template<class T2 >
Tuple<double> Configuration2< T1 >::MaDistance (
    Configuration2< T2 > B ) [inline]

```

Function that compute the Manhattan Distance between two configurations. T2 The type of the elements in the second [Configuration2](#).

#### Parameters

in	<i>B</i>	the second <a href="#">Configuration2</a> to use for computing the distance.
----	----------	--

**Returns**

The Manhattan distance between the two configurations.

**3.2.3.6 offset()** [1/3]

```
template<class T1>
template<class T2 >
int Configuration2< T1 >::offset (
    const T2 _offset,
    const Angle phi,
    const Angle _th ) [inline]
```

This function compute the offset of the point given a vector, that is the lenght of the vector and its angle. The angle must be an [Angle](#) variable. It takes also another [Angle](#) to change the [Angle](#) in the configuration.

**Template Parameters**

--	--

**3.2.3.7 offset()** [2/3]

```
template<class T1>
int Configuration2< T1 >::offset (
    Configuration2< T1 > p ) [inline]
```

This function compute the offset of the point given another [Configuration2](#).

**Parameters**

in	<i>p</i>	The configuration containing the offsets.
----	----------	---

**Returns**

1 if everything went fine, 0 otherwise.

**3.2.3.8 offset()** [3/3]

```
template<class T1>
int Configuration2< T1 >::offset (
    Point2< T1 > p,
    const Angle _th = Angle() ) [inline]
```

This function compute the offset of the point given a [Point2](#) containing the offsets for the abscissa and the ordiante and an [Angle](#) to change the [Angle](#) in the configuration.

## Parameters

in	$p$	The point containing the offsets.
in	$\leftarrow$ $\leftarrow$ $th$	The offset for the <a href="#">Angle</a> in the configuration. It's set to 0 as default so to easily change just the coordinates.

## Returns

1 if everything went fine, 0 otherwise.

## 3.2.3.9 offset\_angle()

```
template<class T1>
void Configuration2< T1 >::offset_angle (
    const Angle _th ) [inline]
```

Function to add an offset to the angle.

## Parameters

in	<code>_offset</code>	The offset.
----	----------------------	-------------

## Returns

1 if everything went fine, 0 otherwise.

## 3.2.3.10 offset\_x()

```
template<class T1>
int Configuration2< T1 >::offset_x (
    const T1 _offset ) [inline]
```

Function to add an offset to the abscissa.

## Parameters

in	<code>_offset</code>	The offset.
----	----------------------	-------------

## Returns

1 if everything went fine, 0 otherwise.

**3.2.3.11 offset\_y()**

```
template<class T1>
int Configuration2< T1 >::offset_y (
    const Angle _offset ) [inline]
```

Function to add an offset to the ordinate.

**Parameters**

in	<code>_offset</code>	The offset.
----	----------------------	-------------

**Returns**

1 if everything went fine, 0 otherwise.

**3.2.3.12 x()** [1/2]

```
template<class T1>
T1 Configuration2< T1 >::x ( ) const [inline]
```

**Returns**

The abscissa coordinate.

**3.2.3.13 x()** [2/2]

```
template<class T1>
int Configuration2< T1 >::x (
    const T1 _x ) [inline]
```

This function stores a new value for the abscissa.

**Parameters**

in	↩	The value to be stored.
	↩	
	<code>x</code>	

**Returns**

1 if everything went ok, 0 otherwise.

**3.2.3.14** `y()` [1/2]

```
template<class T1>
T1 Configuration2< T1 >::y ( ) const [inline]
```

**Returns**

The ordinate coordinate.

**3.2.3.15** `y()` [2/2]

```
template<class T1>
int Configuration2< T1 >::y (
    const T1 _y ) [inline]
```

This function stores a new value for the ordinate.

**Parameters**

in	↔	The value to be stored.
	↔	
	y	

**Returns**

1 if everything went ok, 0 otherwise.

**3.2.4 Friends And Related Function Documentation****3.2.4.1** `operator<<`

```
template<class T1>
ostream& operator<< (
    ostream & out,
    const Configuration2< T1 > & data ) [friend]
```

Overload of operator << to output the content of a `Configuration2`.

**Parameters**

in	out	The output stream.
in	data	The <code>Configuration2</code> to print.

#### Returns

An output stream to be printed.

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

- [src/math.h](#)

## 3.3 dubins Class Reference

```
#include <dubins.h>
```

### Public Member Functions

- [dubins](#) (double x0, double y0, double th0, double x1, double y1, double th1, double max=1.0)
- [dubins](#) ([Point2](#) \_P0, [Point2](#) \_P1, [Angle](#) \_th1, [Angle](#) \_th2, Kmax=1.0)
- [dubins](#) ([Configuration2](#) \_P0, [Configuration2](#) \_P1, Kmax=1.0)
- [~dubins](#) ()

### 3.3.1 Constructor & Destructor Documentation

#### 3.3.1.1 dubins() [1/3]

```
dubins::dubins (
    double x0,
    double y0,
    double th0,
    double x1,
    double y1,
    double th1,
    double max = 1.0 )
```

#### 3.3.1.2 dubins() [2/3]

```
dubins::dubins (
    Point2 _P0,
    Point2 _P1,
    Angle _th1,
    Angle _th2,
    Kmax = 1.0 )
```



## 3.3.1.3 dubins() [3/3]

```
dubins::dubins (
    Configuration2 _P0,
    Configuration2 _P1,
    Kmax = 1.0 )
```

## 3.3.1.4 ~dubins()

```
dubins::~~dubins ( )
```

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

- src/[dubins.hh](#)

## 3.4 Point2&lt; T &gt; Class Template Reference

Class that stores two value to construct a point in 2D. The value is saved in a [Tuple](#).

```
#include <maths.hh>
```

## Public Member Functions

- [Point2](#) ()  
*Default constructor to build an empty [Tuple](#).*
- [Point2](#) (const T \_x, const T \_y)  
*Constructor that taked to elements and builds a point.*
- T x () const
- T y () const
- int x (const T \_x)  
*Set the abscissa value.*
- int y (const T \_y)  
*Set the ordinate value.*
- template<class T1 >  
int [offset](#) (const T1 \_offset, const [Angle](#) th)  
*This function compute the offset of the point given a vector, that is the lenght of the vector and its angle. The angle must be an [Angle](#) variable.*
- int [offset](#) (const [Point2](#)< T > p)  
*This function compute an offset given another point made of the abscissa offset and the ordinate offset.*
- int [offset](#) (const [Tuple](#)< T > p)  
*This function compute an offset given a [Tuple](#) made of the abscissa offset and the ordinate offset.*
- int [offset\\_x](#) (const T \_offset)  
*This function compute an offset for the abscissa.*
- int [offset\\_y](#) (const T \_offset)  
*This function compute an offset for the ordinate.*
- template<class T1 >  
double [distance](#) ([Point2](#)< T1 > B, [DISTANCE\\_TYPE](#) dist=[EUCLIDEAN](#))

*Wrapper to compute different distances. T1 The type of the elements in the second [Point2](#).*

- `template<class T1 >`  
`double MaDistance (Point2< T1 > B)`

*Function that compute the Manhattan Distance between two points. T1 The type of the elements in the second [Point2](#).*

- `template<class T1 >`  
`double EuDistance (Point2< T1 > B)`

*Function that compute the Euclidean Distance between two points. T1 The type of the elements in the second [Point2](#).*

## Friends

- `ostream & operator<< (ostream &out, const Point2< T > &data)`  
*Overload of operator << to output the content of a [Point2](#).*

### 3.4.1 Detailed Description

```
template<class T>
class Point2< T >
```

Class that stores two value to construct a point in 2D. The value is saved in a [Tuple](#).

#### Template Parameters

<code>T</code>	The type of the coordinates to be stored.
----------------	---

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 [Point2\(\)](#) [1/2]

```
template<class T>
Point2< T >::Point2 ( ) [inline]
```

Default constructor to build an empty [Tuple](#).

#### 3.4.2.2 [Point2\(\)](#) [2/2]

```
template<class T>
Point2< T >::Point2 (
    const T _x,
    const T _y ) [inline]
```

Constructor that taked to elements and builds a point.

## Parameters

in	$\leftrightarrow$ $\overleftarrow{x}$	The abscissa coordinate.
in	$\leftrightarrow$ $\overleftarrow{y}$	The ordinate coordinate.

## 3.4.3 Member Function Documentation

## 3.4.3.1 distance()

```
template<class T>
template<class T1 >
double Point2< T >::distance (
    Point2< T1 > B,
    DISTANCE_TYPE dist = EUCLIDEAN ) [inline]
```

Wrapper to compute different distances. T1 The type of the elements in the second [Point2](#).

## Parameters

in	<i>B</i>	The second <a href="#">Point2</a> to use for computing the distance.
in	<i>dist</i>	The type of distance to be computed.

## Returns

The distance between the two points.

## 3.4.3.2 EuDistance()

```
template<class T>
template<class T1 >
double Point2< T >::EuDistance (
    Point2< T1 > B ) [inline]
```

Function that compute the Euclidean Distance between two points. T1 The type of the elements in the second [Point2](#).

## Parameters

in	<i>B</i>	the second <a href="#">Point2</a> to use for computing the distance.
----	----------	--

**Returns**

The Euclidean distance between the two points.

**3.4.3.3 MaDistance()**

```
template<class T>
template<class T1 >
double Point2< T >::MaDistance (
    Point2< T1 > B ) [inline]
```

Function that compute the Manhattan Distance between two points. T1 The type of the elements in the second [Point2](#).

**Parameters**

in	<i>B</i>	the second <a href="#">Point2</a> to use for computing the distance.
----	----------	--

**Returns**

The Manhattan distance between the two points.

**3.4.3.4 offset()** [1/3]

```
template<class T>
template<class T1 >
int Point2< T >::offset (
    const T1 _offset,
    const Angle th ) [inline]
```

This function compute the offset of the point given a vector, that is the lenght of the vector and its angle. The angle must be an [Angle](#) variable.

**Template Parameters**

--	--

**3.4.3.5 offset()** [2/3]

```
template<class T>
int Point2< T >::offset (
    const Point2< T > p ) [inline]
```

This function compute an offset given another point made of the abscissa offset and the ordinate offset.

**Parameters**

in	<i>p</i>	The point with the offsets.
----	----------	-----------------------------

**Returns**

1 if everything went fine, 0 otherwise.

**3.4.3.6 offset()** [3/3]

```
template<class T>
int Point2< T >::offset (
    const Tuple< T > p ) [inline]
```

This function compute an offset given a [Tuple](#) made of the abscissa offset and the ordinate offset.

**Parameters**

in	<i>p</i>	The <a href="#">Tuple</a> with the offsets. Its dimension must be 2.
----	----------	--

**Returns**

1 if everything went fine, 0 otherwise.

**3.4.3.7 offset\_x()**

```
template<class T>
int Point2< T >::offset_x (
    const T _offset ) [inline]
```

This function compute an offset for the abscissa.

**Parameters**

in	<i>_offset</i>	The offset.
----	----------------	-------------

**Returns**

1 if everything went fine, 0 otherwise.

### 3.4.3.8 offset\_y()

```
template<class T>
int Point2< T >::offset_y (
    const T _offset ) [inline]
```

This function compute an offset for the ordinate.

#### Parameters

in	<code>_offset</code>	The offset.
----	----------------------	-------------

#### Returns

1 if everything went fine, 0 otherwise.

### 3.4.3.9 x() [1/2]

```
template<class T>
T Point2< T >::x ( ) const [inline]
```

#### Returns

The abscissa coordinate

### 3.4.3.10 x() [2/2]

```
template<class T>
int Point2< T >::x (
    const T _x ) [inline]
```

Set the abscissa value.

#### Parameters

in	↩	The new abscissa value
	↩	
	<code>x</code>	

#### Returns

1 if it was successful, 0 otherwise.

**3.4.3.11** `y()` [1/2]

```
template<class T>
T Point2< T >::y ( ) const [inline]
```

**Returns**

The ordinate coordinate

**3.4.3.12** `y()` [2/2]

```
template<class T>
int Point2< T >::y (
    const T _y ) [inline]
```

Set the ordinate value.

**Parameters**

in	↔	The new ordinate value
	↔	
	x	

**Returns**

1 if it was successful, 0 otherwise.

**3.4.4 Friends And Related Function Documentation****3.4.4.1** `operator<<`

```
template<class T>
ostream& operator<< (
    ostream & out,
    const Point2< T > & data ) [friend]
```

Overload of operator << to output the content of a `Point2`.

**Parameters**

in	<i>out</i>	The output stream.
in	<i>data</i>	The <code>Point2</code> to print.

#### Returns

An output stream to be printed.

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

- [src/maths.hh](#)

## 3.5 Settings Class Reference

```
#include <calibration.hh>
```

#### Public Types

- enum [Pattern](#) { [NOT\\_EXISTING](#) =0, [CHESSBOARD](#) =1 }
- enum [InputType](#) { [INVALID](#) =0, [IMAGE\\_LIST](#) =3 }

#### Public Member Functions

- [Settings](#) ()  
*Constructor that sets `goodInput` to false.*
- void [write](#) (FileStorage &fs) const  
*Write serialization.*
- void [read](#) (const FileNode &node)  
*Read serialization.*
- void [validate](#) ()  
*This function validate the content of the file.*
- Mat [nextImage](#) ()  
*Get next image from list.*

#### Static Public Member Functions

- static bool [readStringList](#) (const string &filename, vector< string > &l)  
*Read from file a list of images.*
- static bool [isListOfImages](#) (const string &filename)  
*Check if the file from which is trying to retrieve a list is a valid format (xml or yaml).*



## Public Attributes

- Size `boardSize`  
*The size of the board -> Number of items by width and height.*
- Pattern calibrationPattern = `CHESSBOARD`  
*One of the Chessboard, circles, or asymmetric circle pattern.*
- float `squareSize`  
*The size of a square in your defined unit (point, millimeter, etc).*
- int `nrFrames`  
*The number of frames to use from the input for calibration.*
- float `aspectRatio`  
*The aspect ratio.*
- int `delay`  
*In case of a video input.*
- bool `writePoints`  
*Write detected feature points.*
- bool `writeExtrinsics`  
*Write extrinsic parameters.*
- bool `calibZeroTangentDist`  
*Assume zero tangential distortion.*
- bool `calibFixPrincipalPoint`  
*Fix the principal point at the center.*
- bool `flipVertical`  
*Flip the captured images around the horizontal axis.*
- string `outputFileName`  
*The name of the file where to write.*
- bool `showUndistorted`  
*Show undistorted images after calibration.*
- string `input`  
*The input.*
- bool `useFisheye` = false  
*use fisheye camera model for calibration*
- bool `fixK1`  
*fix K1 distortion coefficient*
- bool `fixK2`  
*fix K2 distortion coefficient*
- bool `fixK3`  
*fix K3 distortion coefficient*
- bool `fixK4`  
*fix K4 distortion coefficient*
- bool `fixK5`  
*fix K5 distortion coefficient*
- int `cameraID`
- vector< string > `imageList`
- size\_t `atImageList`
- VideoCapture `inputCapture`
- InputType `inputType` = `IMAGE_LIST`
- bool `goodInput`
- int `flag`

### 3.5.1 Member Enumeration Documentation

#### 3.5.1.1 InputType

enum `Settings::InputType`

Enumerator

INVALID	
IMAGE_LIST	

#### 3.5.1.2 Pattern

enum `Settings::Pattern`

Enumerator

NOT_EXISTING	
CHESSBOARD	

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 Settings()

```
Settings::Settings ( ) [inline]
```

Constructor that sets `goodInput` to false.

### 3.5.3 Member Function Documentation

#### 3.5.3.1 isListOfImages()

```
bool Settings::isListOfImages (
    const string & filename ) [static]
```

Check if the file from which is trying to retrieve a list is a valid format (xml or yaml).

**Parameters**

in	<i>filename</i>	The name of the file to check for validity.
----	-----------------	---

**Returns**

`false` is the file is not xml or yaml  
`true` otherwise.

**3.5.3.2 nextImage()**

```
Mat Settings::nextImage ( )
```

Get next image from list.

**Returns**

A matrix containing the next image to consider.

**3.5.3.3 read()**

```
void Settings::read (
    const FileNode & node )
```

Read serialization.

This function read data from a file and stores each node in their corresponding variables.

**Parameters**

in	<i>node</i>	The node of the file to consider.
----	-------------	-----------------------------------

**3.5.3.4 readStringList()**

```
bool Settings::readStringList (
    const string & filename,
    vector< string > & l ) [static]
```

Read from file a list of images.

**Parameters**

in	<i>filename</i>	The name of the file from which to read.
out	<i>l</i>	A vector which will contain the names of the file from the list.

**Returns**

`false` if the file could not be opened or if the file doesn't contain a list  
`true` otherwise.

**3.5.3.5 validate()**

```
void Settings::validate ( )
```

This function validate the content of the file.

Even though this function doesn't return anything nor has any parameters for output, it sets a variable of the `Settings` class, that is `googInput`, to `false` if some infos were wrong. `true` otherwise. The options it takes in consideration are the following:

- Size must be positive.
- Cells must be greater than  $10^{-6}$ .
- The number of frames considered, that is images, must be greater than 0.
- Check for valid input, that is a valid list of images.
- Else a list of image is being used.
- Check the field pattern: if it doesn't correspond to a known one than it's invalid.

**3.5.3.6 write()**

```
void Settings::write (
    FileStorage & fs ) const
```

Write serialization.

This function write data to a file.

## Parameters

<code>in</code>	<code>fs</code>	The filename where to write.
-----------------	-----------------	------------------------------

### 3.5.4 Member Data Documentation

#### 3.5.4.1 aspectRatio

```
float Settings::aspectRatio
```

The aspect ratio.

#### 3.5.4.2 atImageList

```
size_t Settings::atImageList
```

#### 3.5.4.3 boardSize

```
Size Settings::boardSize
```

The size of the board -> Number of items by width and height.

#### 3.5.4.4 calibFixPrincipalPoint

```
bool Settings::calibFixPrincipalPoint
```

Fix the principal point at the center.

#### 3.5.4.5 calibrationPattern

```
Pattern Settings::calibrationPattern = CHESSBOARD
```

One of the Chessboard, circles, or asymmetric circle pattern.

#### 3.5.4.6 calibZeroTangentDist

```
bool Settings::calibZeroTangentDist
```

Assume zero tangential distortion.

#### 3.5.4.7 cameraID

```
int Settings::cameraID
```

#### 3.5.4.8 delay

```
int Settings::delay
```

In case of a video input.

#### 3.5.4.9 fixK1

```
bool Settings::fixK1
```

fix K1 distortion coefficient

#### 3.5.4.10 fixK2

```
bool Settings::fixK2
```

fix K2 distortion coefficient

#### 3.5.4.11 fixK3

```
bool Settings::fixK3
```

fix K3 distortion coefficient

#### 3.5.4.12 fixK4

```
bool Settings::fixK4
```

fix K4 distortion coefficient

#### 3.5.4.13 fixK5

```
bool Settings::fixK5
```

fix K5 distortion coefficient

#### 3.5.4.14 flag

```
int Settings::flag
```

#### 3.5.4.15 flipVertical

```
bool Settings::flipVertical
```

Flip the captured images around the horizontal axis.

#### 3.5.4.16 goodInput

```
bool Settings::goodInput
```

#### 3.5.4.17 imageList

```
vector<string> Settings::imageList
```

#### 3.5.4.18 input

```
string Settings::input
```

The input.

#### 3.5.4.19 inputCapture

```
VideoCapture Settings::inputCapture
```

#### 3.5.4.20 inputType

```
InputType Settings::inputType = IMAGE_LIST
```

#### 3.5.4.21 nrFrames

```
int Settings::nrFrames
```

The number of frames to use from the input for calibration.

#### 3.5.4.22 outputFileName

```
string Settings::outputFileName
```

The name of the file where to write.

#### 3.5.4.23 showUndistorted

```
bool Settings::showUndistorted
```

Show undistorted images after calibration.

#### 3.5.4.24 squareSize

```
float Settings::squareSize
```

The size of a square in your defined unit (point, millimeter,etc).



## 3.5.4.25 useFisheye

```
bool Settings::useFisheye = false
```

use fisheye camera model for calibration

## 3.5.4.26 writeExtrinsics

```
bool Settings::writeExtrinsics
```

Write extrinsic parameters.

## 3.5.4.27 writePoints

```
bool Settings::writePoints
```

Write detected feature points.

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

- [src/calibration.hh](#)
- [src/calibration.cc](#)

## 3.6 Tuple&lt; T &gt; Class Template Reference

```
#include <maths.hh>
```

## Public Member Functions

- [Tuple](#) ()  
*Default constructor.*
- [Tuple](#) (int \_n,...)  
*Constructors that takes the number of objectes to be stored, the objects and then stores them.*
- int [size](#) () const
- T [get](#) (const int \_n) const  
*Gets the n-th element.*
- void [add](#) (const T \_new)  
*Adds a value at the end of the list.*
- int [remove](#) (const T pos)  
*Removes a value from the list.*
- int [set](#) (const int pos, const T \_new)  
*Set a value in a certain position, or adds the element if the position equals the number of elements.*
- template<class T1 >  
double [EuDistance](#) (const [Tuple](#)< T1 > B)  
*Function that compute the Euclidean Distance between two tuples. They must have the same number of elements.  
T1 The type of the elements in the second [Tuple](#).*
- template<class T1 >  
double [MaDistance](#) (const [Tuple](#)< T1 > B)  
*Function that compute the Manhattan Distance between two tuples. They must have the same number of elements.  
T1 The type of the elements in the second [Tuple](#).*
- template<class T1 >  
double [distance](#) (const [Tuple](#)< T1 > B, const [DISTANCE\\_TYPE](#) dist=[EUCLIDEAN](#))  
*Wrapper to compute different distances. They must have the same number of elements. T1 The type of the elements in the second [Tuple](#).*

## Friends

- ostream & [operator<<](#) (ostream &out, const [Tuple](#) &data)  
*Overload of operator << to output the content of the tuple.*

### 3.6.1 Detailed Description

```
template<class T>
class Tuple< T >
```

This class allows the definition and storage of tuples of different dimensions. Functions to compute distance between tuples are also available.

#### Template Parameters

<i>T</i>	The type of elements to be stored.
----------	------------------------------------

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 [Tuple\(\)](#) [1/2]

```
template<class T>
Tuple< T >::Tuple ( ) [inline]
```

Default constructor.

#### 3.6.2.2 [Tuple\(\)](#) [2/2]

```
template<class T>
Tuple< T >::Tuple (
    int _n,
    ... ) [inline]
```

Constructors that takes the number of objects to be stored, the objects and then stores them.

#### Parameters

in	$\leftrightarrow$ $\_ \leftarrow$ <i>n</i>	Number of objects to store.
in	...	Objects to store.

### 3.6.3 Member Function Documentation

#### 3.6.3.1 add()

```
template<class T>
void Tuple< T >::add (
    const T _new ) [inline]
```

Adds a value at the end of the list.

##### Parameters

in	<code>_new</code>	The new value to be added.
----	-------------------	----------------------------

#### 3.6.3.2 distance()

```
template<class T>
template<class T1 >
double Tuple< T >::distance (
    const Tuple< T1 > B,
    const DISTANCE_TYPE dist = EUCLIDEAN ) [inline]
```

Wrapper to compute different distances. They must have the same number of elements. T1 The type of the elements in the second [Tuple](#).

##### Parameters

in	<code>B</code>	The second <a href="#">Tuple</a> to use for computing the distance.
in	<code>dist</code>	The type of distance to be computed.

##### Returns

The distance between the two [Tuple](#).

#### 3.6.3.3 EuDistance()

```
template<class T>
template<class T1 >
double Tuple< T >::EuDistance (
    const Tuple< T1 > B ) [inline]
```

Function that compute the Euclidean Distance between two tuples. They must have the same number of elements. T1 The type of the elements in the second [Tuple](#).

**Parameters**

in	<i>B</i>	the second <a href="#">Tuple</a> to use for computing the distance.
----	----------	---

**Returns**

The Euclidean distance between the two [Tuple](#).

**3.6.3.4 get()**

```
template<class T>
T Tuple< T >::get (
    const int _n ) const [inline]
```

Gets the n-th element.

**Parameters**

in	$\leftrightarrow$	The position of the element to retrieve.
	$\leftarrow$ <i>n</i>	

**Returns**

The element in the n-th position.

**3.6.3.5 MaDistance()**

```
template<class T>
template<class T1 >
double Tuple< T >::MaDistance (
    const Tuple< T1 > B ) [inline]
```

Function that compute the Manhattan Distance between two tuples. They must have the same number of elements.  
T1 The type of the elements in the second [Tuple](#).

**Parameters**

in	<i>B</i>	the second <a href="#">Tuple</a> to use for computing the distance.
----	----------	---

**Returns**

The Manhattan distance between the two [Tuple](#).

## 3.6.3.6 remove()

```
template<class T>
int Tuple< T >::remove (
    const T pos ) [inline]
```

Removes a value from the list.

## Parameters

in	<i>pos</i>	The position of the value to be removed.
----	------------	--

## 3.6.3.7 set()

```
template<class T>
int Tuple< T >::set (
    const int pos,
    const T _new ) [inline]
```

Set a value in a certain position, or adds the element if the position equals the number of elements.

## Parameters

in	<i>pos</i>	Must be in $[0, n - 1]$ . If $pos = n$ then the element is added at the end of the vector.
in	<i>_new</i>	The new element to be set.

## Returns

1 if everything went right, 0 if the position was greater than  $n$  or less the 0.

## 3.6.3.8 size()

```
template<class T>
int Tuple< T >::size ( ) const [inline]
```

## Returns

The number of stored elements.

## 3.6.4 Friends And Related Function Documentation

### 3.6.4.1 operator<<

```
template<class T>
ostream& operator<< (
    ostream & out,
    const Tuple< T > & data ) [friend]
```

Overload of operator << to output the content of the tuple.

#### Parameters

in	<i>out</i>	The output stream.
in	<i>data</i>	The <a href="#">Tuple</a> to print.

#### Returns

An output stream to be printed.

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

- [src/maths.hh](#)

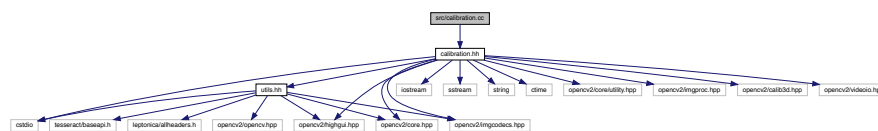
## Chapter 4

# File Documentation

### 4.1 src/calibration.cc File Reference

```
#include "calibration.hh"
```

Include dependency graph for calibration.cc:



### Functions

- int [calibration](#) (const string inputFile)  
*Function to run the complete calibration.*
- static void [read](#) (const FileNode &node, [Settings](#) &x, const [Settings](#) &default\_value)  
*Reads settings from file. If there is none then initiate a new [Settings](#).*
- static double [computeReprojectionErrors](#) (const vector< vector< Point3f > > &objectPoints, const vector< vector< Point2f > > &imagePoints, const vector< Mat > &rvecs, const vector< Mat > &tvecs, const Mat &cameraMatrix, const Mat &distCoeffs, vector< float > &perViewErrors, bool fisheye)  
*Compute the errors of the projection.*
- void [calcBoardCornerPositions](#) (Size boardSize, float squareSize, vector< Point3f > &corners)  
*This function compute the position of the upper corners of every cell.*
- static bool [runCalibration](#) ([Settings](#) &s, Size &imageSize, Mat &cameraMatrix, Mat &distCoeffs, vector< vector< Point2f > > imagePoints, vector< Mat > &rvecs, vector< Mat > &tvecs, vector< float > &reprojErrs, double &totalAvgErr)  
*This function run the calibration creating the matrixed for the camera and the distortion coefficients.*
- static void [saveCameraParams](#) (const [Settings](#) &s, const Size &imageSize, const Mat &cameraMatrix, const Mat &distCoeffs, const vector< Mat > &rvecs, const vector< Mat > &tvecs, const vector< float > &reprojErrs, const vector< vector< Point2f > > &imagePoints, const double totalAvgErr)  
*Function to save the computed parameters to a file.*
- bool [runCalibrationAndSave](#) ([Settings](#) &s, Size imageSize, Mat &cameraMatrix, Mat &distCoeffs, vector< vector< Point2f > > imagePoints)  
*Reads settings from file. If there is none then initiate a new [Settings](#).*

## 4.1.1 Function Documentation

### 4.1.1.1 calcBoardCornerPositions()

```
void calcBoardCornerPositions (
    Size boardSize,
    float squareSize,
    vector< Point3f > & corners )
```

This function compute the position of the upper corners of every cell.

#### Parameters

in	<i>boardSiz</i>	The dimension of the chess board.
in	<i>squareSize</i>	The dimension of the edge of a cell.
out	<i>corners</i>	A vector of Point3fs which equals to the corners of the cells.

### 4.1.1.2 calibration()

```
int calibration (
    const string inputFile )
```

Function to run the complete calibration.

#### Parameters

in	<i>inputFile</i>	Name of the setting.xml file. It's set to default to default.xml
----	------------------	--

#### Returns

- 2 if the settings file could be load but the input was not well-formed
- 1 if the settings file could not be opened.
- 0 if everything went fine.

### 4.1.1.3 computeReprojectionErrors()

```
static double computeReprojectionErrors (
    const vector< vector< Point3f > > & objectPoints,
    const vector< vector< Point2f > > & imagePoints,
    const vector< Mat > & rvecs,
    const vector< Mat > & tvecs,
    const Mat & cameraMatrix,
```



```

    const Mat & distCoeffs,
    vector< float > & perViewErrors,
    bool fisheye ) [static]

```

Compute the errors of the projection.

#### Parameters

in	<i>objectPoints</i>	The real image points which will be projected
in	<i>rvecs</i>	Input vector of rotation vectors estimated for each pattern view.
in	<i>tvecs</i>	Input vector of translation vectors estimated for each pattern view.
in	<i>cameraMatrix</i>	The matrix containing the parameters for the camera
in	<i>distCoeffs</i>	The matrix containing the distortion coefficients.
in	<i>fisheye</i>	A variable which says if a fish eye correction should be applied or no.
out	<i>perViewErrors</i>	A vector containing the error for each image.
out	<i>imagePoints</i>	The projected points for each image.

#### Returns

The total error.

#### 4.1.1.4 read()

```

static void read (
    const FileNode & node,
    Settings & x,
    const Settings & default_value ) [inline], [static]

```

Reads settings from file. If there is none then initiate a new [Settings](#).

#### Parameters

in	<i>node</i>	node to consider for getting settings;
in	<i>x</i>	<a href="#">Settings</a> to configure;
in	<i>default_value</i>	<a href="#">Settings</a> default value. Setted to <a href="#">Settings()</a> .

#### 4.1.1.5 runCalibration()

```

static bool runCalibration (
    Settings & s,
    Size & imageSize,
    Mat & cameraMatrix,
    Mat & distCoeffs,
    vector< vector< Point2f > > & imagePoints,
    vector< Mat > & rvecs,

```

```
vector< Mat > & tvecs,
vector< float > & reprojErrs,
double & totalAvgErr ) [static]
```

This function run the calibration creating the matrixed for the camera and the distorsion coefficients.

#### Parameters

in	<i>s</i>	The <a href="#">Settings</a> read from the file and memorized.
in	<i>imageSize</i>	The size of the image used in <code>calibrateCamera()</code> to initialize the camera matrix.
in	<i>imagePoints</i>	The projected points for each image.
in	<i>reprojErrs</i>	The re-projection error, that is a geometric error corresponding to the image distance between a projected point and a measured one.
out	<i>cameraMatrix</i>	The matrix of the camera parameters
out	<i>distCoeffs</i>	The matrix of the distorsion coefficients.
out	<i>rvecs</i>	Output vector of rotation vectors estimated for each pattern view.
out	<i>tvecs</i>	Output vector of translation vectors estimated for each pattern view.
out	<i>totalAvgErr</i>	The total avarage error given from distorsion.

#### Returns

`false` if one or more elements in the `cameraMatrix` and `distCoeffs` are invalid.  
`true` if all the elements are valid.

#### 4.1.1.6 runCalibrationAndSave()

```
bool runCalibrationAndSave (
    Settings & s,
    Size imageSize,
    Mat & cameraMatrix,
    Mat & distCoeffs,
    vector< vector< Point2f > > imagePoints )
```

Reads settings from file. If there is none then initiate a new [Settings](#).

#### Parameters

in	<i>s</i>	The <a href="#">Settings</a> being used during the execution.
in	<i>imageSize</i>	The dimensions of the images.
in	<i>imagePoints</i>	The projected points for each image.
out	<i>cameraMatrix</i>	The matrix which is used to store the values for the camera parameters.
out	<i>distCoeffs</i>	The matrix which is used to store the distortion coefficients.

#### Returns

`true` if the calibration succeded.  
`false` otherwise.

## 4.1.1.7 saveCameraParams()

```
static void saveCameraParams (
    const Settings & s,
    const Size & imageSize,
    const Mat & cameraMatrix,
    const Mat & distCoeffs,
    const vector< Mat > & rvecs,
    const vector< Mat > & tvecs,
    const vector< float > & reprojErrs,
    const vector< vector< Point2f > > & imagePoints,
    const double totalAvgErr ) [static]
```

Function to save the computed parameters to a file.

## Parameters

in	<i>s</i>	Use the <a href="#">Settings</a> got at the beginning for information as the output file name, image and board size.
in	<i>imageSize</i>	The size of the image.
in	<i>cameraMatrix</i>	The camera matrix.
in	<i>distCoeffs</i>	The distortion coefficient matrix.
	<i>[int]</i>	rvecs Vector of rotation vectors estimated for each pattern view.
in	<i>tvecs</i>	Vector of translation vectors estimated for each pattern view.
in	<i>reprojErrs</i>	The re-projection error, that is a geometric error corresponding to the image distance between a projected point and a measured one.
in	<i>imagePoints</i>	The projected points for each image.
in	<i>totalAvgErr</i>	The total average error given from distortion.

Open file for writing

Stores time of calibration

Store infos about the images

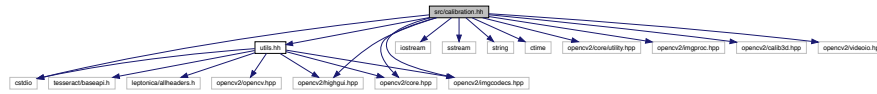
## 4.2 src/calibration.hh File Reference

Library for calibration.

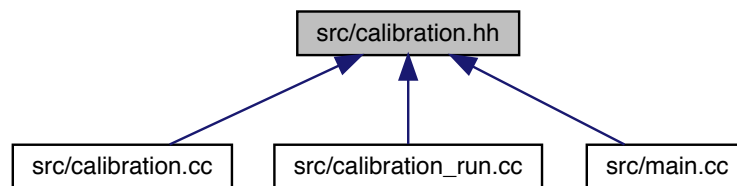
```
#include "utils.hh"
#include <iostream>
#include <sstream>
#include <string>
#include <ctime>
#include <cstdio>
#include <opencv2/core.hpp>
#include <opencv2/core/utility.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/videoio.hpp>
```

```
#include <opencv2/highgui.hpp>
```

Include dependency graph for calibration.hh:



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



## Classes

- class [Settings](#)

## Enumerations

- enum { [DETECTION](#) = 0, [CAPTURING](#) = 1, [CALIBRATED](#) = 2 }

## Functions

- int [calibration](#) (const string inputFile="data/calib\_config.xml")  
*Function to run the complete calibration.*
- bool [runCalibrationAndSave](#) ([Settings](#) &s, Size imageSize, Mat &cameraMatrix, Mat &distCoeffs, vector< vector< Point2f > > imagePoints)  
*Reads settings from file. If there is none then initiate a new [Settings](#).*

### 4.2.1 Detailed Description

Library for calibration.

### 4.2.2 Enumeration Type Documentation

#### 4.2.2.1 anonymous enum

anonymous enum

## Enumerator

DETECTION	
CAPTURING	
CALIBRATED	

## 4.2.3 Function Documentation

## 4.2.3.1 calibration()

```
int calibration (
    const string inputFile )
```

Function to run the complete calibration.

## Parameters

in	<i>inputFile</i>	Name of the setting.xml file. It's set to default to default.xml
----	------------------	--

## Returns

-2 if the settings file could be load but the input was not well-formed  
 -1 if the settings file could not be opened.  
 0 if everything went fine.

## 4.2.3.2 runCalibrationAndSave()

```
bool runCalibrationAndSave (
    Settings & s,
    Size imageSize,
    Mat & cameraMatrix,
    Mat & distCoeffs,
    vector< vector< Point2f > > imagePoints )
```

Reads settings from file. If there is none then initiate a new [Settings](#).

## Parameters

in	<i>s</i>	The <a href="#">Settings</a> being used during the execution.
in	<i>imageSize</i>	The dimensions of the images.
in	<i>imagePoints</i>	The projected points for each image.
out	<i>cameraMatrix</i>	The matrix which is used to store the values for the camera parameters.
out	<i>distCoeffs</i>	The matrix which is used to store the distortion coefficients.

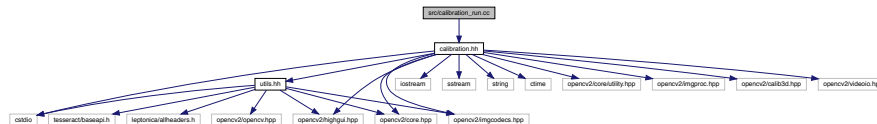
**Returns**

`true` if the calibration succeeded.  
`false` otherwise.

**4.3 src/calibration\_run.cc File Reference**

```
#include "calibration.hh"
```

Include dependency graph for calibration\_run.cc:

**Functions**

- int `main` ()

**4.3.1 Function Documentation****4.3.1.1 main()**

```
int main ( )
```

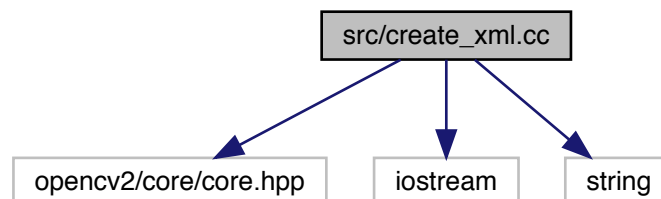
**4.4 src/create\_xml.cc File Reference**

```
#include <opencv2/core/core.hpp>
```

```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for create\_xml.cc:



## Functions

- int [main](#) ()

### 4.4.1 Function Documentation

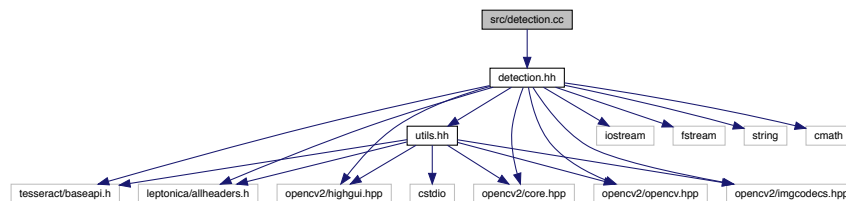
#### 4.4.1.1 main()

```
int main ( )
```

## 4.5 src/detection.cc File Reference

```
#include "detection.hh"
```

Include dependency graph for detection.cc:



## Macros

- #define [WAIT](#)

## Functions

- int [detection](#) ()  
*Loads some images and detects shapes according to different colors.*
- void [load\\_number\\_template](#) ()  
*Load some templates and save them in the global variable 'templates'.*
- void [shape\\_detection](#) (const Mat &img, const int color)  
*Detect shapes inside the image according to the variable 'color'.*
- void [erode\\_dilation](#) (Mat &img, const int color)  
*It apply some filtering function for isolate the subject and remove the noise.*
- void [find\\_contours](#) (const Mat &img, Mat original, const int color)  
*Given an image, in black/white format, identify all the borders that delimit the shapes.*
- void [save\\_convex\\_hull](#) (const vector< vector< Point >> &contours, const int color, const vector< int > &victims)  
*Given some vector save it in a xml file.*
- int [number\\_recognition](#) (Rect blob, const Mat &base)  
*Detect a number on an image inside a region of interest.*
- void [crop\\_number\\_section](#) (Mat &ROI)  
*Given an image identify the region of interest(ROI) and crop it out.*

## Variables

- const string `xml_settings` = "data/settings.xml"
- FileStorage `fs_xml`
- vector< Mat > `templates`

## 4.5.1 Macro Definition Documentation

### 4.5.1.1 WAIT

```
#define WAIT
```

## 4.5.2 Function Documentation

### 4.5.2.1 crop\_number\_section()

```
void crop_number_section (
    Mat & ROI )
```

Given an image identify the region of interest(ROI) and crop it out.

#### Parameters

<code>in, out</code>	<i>ROI</i>	Is the image that the function will going to elaborate.
----------------------	------------	---

### 4.5.2.2 detection()

```
int detection ( )
```

Loads some images and detects shapes according to different colors.

#### Returns

Return 0 if the function reach the end.



## 4.5.2.3 erode\_dilation()

```
void erode_dilation (
    Mat & img,
    const int color )
```

It apply some filtering function for isolate the subject and remove the noise.

An example of the sub functions called are: GaussianBlur, Erosion, Dilation and Threshold.

## Parameters

<i>in, out</i>	<i>img</i>	Is the image on which the function apply the filtering.
<i>in</i>	<i>color</i>	Can has 4 value: 0 -> Red 1 -> Green 2 -> Blue 3 -> Black According to the color the filtering functions apply can change in the type and in the order.

## 4.5.2.4 find\_contours()

```
void find_contours (
    const Mat & img,
    Mat original,
    const int color )
```

Given an image, in black/white format, identify all the borders that delimit the shapes.

## Parameters

<i>in</i>	<i>img</i>	Is an image in HSV format at the base of the elaboration process.
<i>out</i>	<i>original</i>	Is the original source of 'img', it is used for showing the detected contours.
<i>in</i>	<i>color</i>	Can has 3 value: 0 -> Red 1 -> Green 2 -> Blue Is used for decid which procedure apply to the image.

## 4.5.2.5 load\_number\_template()

```
void load_number_template ( )
```

Load some templates and save them in the global variable 'templates'.

#### 4.5.2.6 number\_recognition()

```
int number_recognition (
    Rect blob,
    const Mat & base )
```

Detect a number on an image inside a region of interest.

##### Parameters

in	<i>blob</i>	Identify the region of interest inside the image 'base'.
in	<i>base</i>	Is the image where the function will going to search the number.

##### Returns

The number recognise, '-1' otherwise.

#### 4.5.2.7 save\_convex\_hull()

```
void save_convex_hull (
    const vector< vector< Point >> & contours,
    const int color,
    const vector< int > & victims )
```

Given some vector save it in a xml file.

##### Parameters

in	<i>contours</i>	Is a vector that is saved in a xml file.
in	<i>color</i>	Is the parameter according to which the function decide if saved ('color==1') or not ('otherwise') the vector 'victims'.
in	<i>victims</i>	Is a vector that is saved in a xml file.

#### 4.5.2.8 shape\_detection()

```
void shape_detection (
    const Mat & img,
    const int color )
```

Detect shapes inside the image according to the variable 'color'.

##### Parameters

in	<i>img</i>	Image on which the research will done.
in	<i>color</i>	Can has 3 value: 0 -> Red 1 -> Green
		2 -> Blue These color identify the possible spectrum that the function search on the image.

### 4.5.3 Variable Documentation

#### 4.5.3.1 fs\_xml

```
FileStorage fs_xml
```

#### 4.5.3.2 templates

```
vector<Mat> templates
```

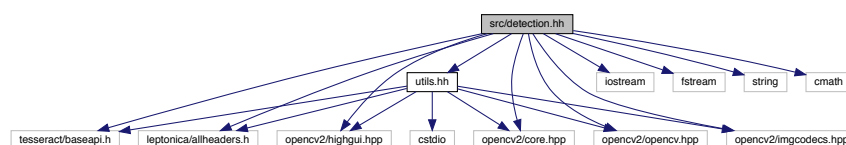
#### 4.5.3.3 xml\_settings

```
const string xml_settings = "data/settings.xml"
```

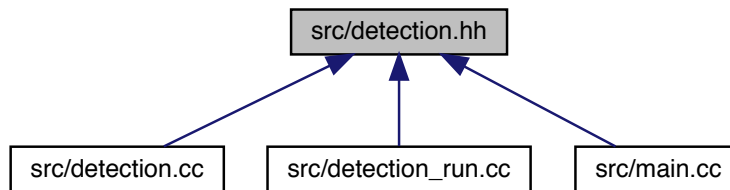
## 4.6 src/detection.hh File Reference

```
#include <tesseract/baseapi.h>
#include <leptonica/allheaders.h>
#include "utils.hh"
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
```

Include dependency graph for detection.hh:



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



## Functions

- int [detection](#) ()  
*Loads some images and detects shapes according to different colors.*
- void [shape\\_detection](#) (const Mat &img, const int color)  
*Detect shapes inside the image according to the variable 'color'.*
- void [erode\\_dilation](#) (Mat &img, const int color)  
*It apply some filtering function for isolate the subject and remove the noise.*
- void [find\\_contours](#) (const Mat &img, Mat original, const int color)  
*Given an image, in black/white format, identify all the borders that delimit the shapes.*
- int [number\\_recognition](#) (Rect blob, const Mat &base)  
*Detect a number on an image inside a region of interest.*
- void [save\\_convex\\_hull](#) (const vector< vector< Point >> &contours, const int color, const vector< int > &victims)  
*Given some vector save it in a xml file.*
- void [load\\_number\\_template](#) ()  
*Load some templates and save them in the global variable 'templates'.*
- void [crop\\_number\\_section](#) (Mat &processROI)  
*Given an image identify the region of interest(ROI) and crop it out.*

### 4.6.1 Function Documentation

#### 4.6.1.1 crop\_number\_section()

```
void crop_number_section (
    Mat & ROI )
```

Given an image identify the region of interest(ROI) and crop it out.

#### Parameters

in, out	ROI	Is the image that the function will going to elaborate.
---------	-----	---

#### 4.6.1.2 detection()

```
int detection ( )
```

Loads some images and detects shapes according to different colors.

##### Returns

Return 0 if the function reach the end.

#### 4.6.1.3 erode\_dilation()

```
void erode_dilation (
    Mat & img,
    const int color )
```

It apply some filtering function for isolate the subject and remove the noise.

An example of the sub functions called are: GaussianBlur, Erosion, Dilation and Threshold.

##### Parameters

in, out	<i>img</i>	Is the image on which the function apply the filtering.
in	<i>color</i>	Can has 4 value: 0 -> Red 1 -> Green 2 -> Blue 3 -> Black According to the color the filtering functions apply can change in the type and in the order.

#### 4.6.1.4 find\_contours()

```
void find_contours (
    const Mat & img,
    Mat original,
    const int color )
```

Given an image, in black/white format, identify all the borders that delimit the shapes.

##### Parameters

in	<i>img</i>	Is an image in HSV format at the base of the elaboration process.
out	<i>original</i>	Is the original source of 'img', it is used for showing the detected contours.

**Parameters**

in	<i>color</i>	Can has 3 value: 0 -> Red 1 -> Green 2 -> Blue Is used for decid which procedure apply to the image.
----	--------------	--

**4.6.1.5 load\_number\_template()**

```
void load_number_template ( )
```

Load some templates and save them in the global variable 'templates'.

**4.6.1.6 number\_recognition()**

```
int number_recognition (
    Rect blob,
    const Mat & base )
```

Detect a number on an image inside a region of interest.

**Parameters**

in	<i>blob</i>	Identify the region of interest inside the image 'base'.
in	<i>base</i>	Is the image where the function will going to search the number.

**Returns**

The number recognise, '-1' otherwise.

**4.6.1.7 save\_convex\_hull()**

```
void save_convex_hull (
    const vector< vector< Point >> & contours,
    const int color,
    const vector< int > & victims )
```

Given some vector save it in a xml file.

**Parameters**

in	<i>contours</i>	Is a vector that is saved in a xml file.
in	<i>color</i>	Is the parameter according to which the function decide if saved ('color==1') or not ('otherwise') the vector 'victims'.
in	<i>victims</i>	Is a vector that is saved in a xml file.

## 4.6.1.8 shape\_detection()

```
void shape_detection (
    const Mat & img,
    const int color )
```

Detect shapes inside the image according to the variable 'color'.

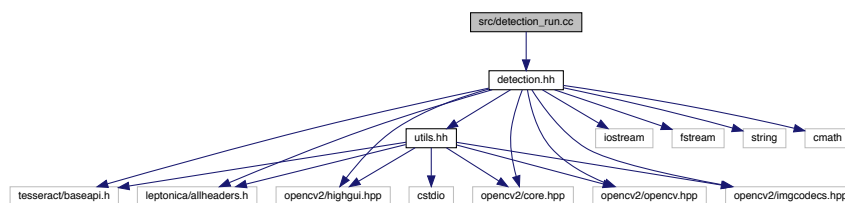
## Parameters

in	<i>img</i>	Image on which the research will done.
in	<i>color</i>	Can has 3 value: 0 -> Red 1 -> Green 2 -> Blue These color identify the possible spectrum that the function search on the image.

## 4.7 src/detection\_run.cc File Reference

```
#include "detection.hh"
```

Include dependency graph for detection\_run.cc:



## Functions

- int [main](#) ()

## 4.7.1 Function Documentation

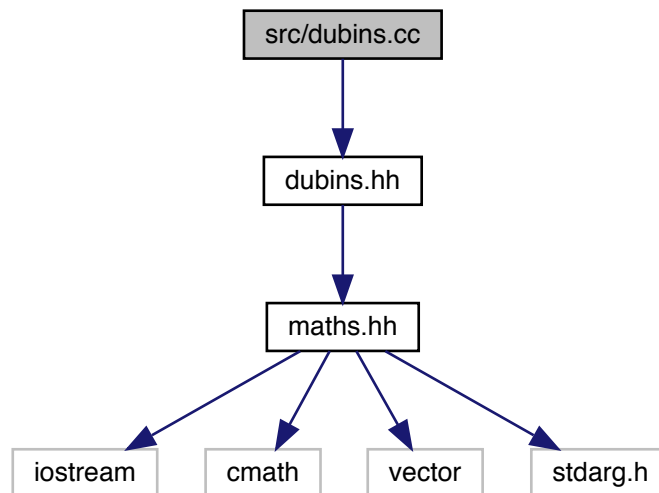
## 4.7.1.1 main()

```
int main ( )
```

## 4.8 src/dubins.cc File Reference

```
#include "dubins.hh"
```

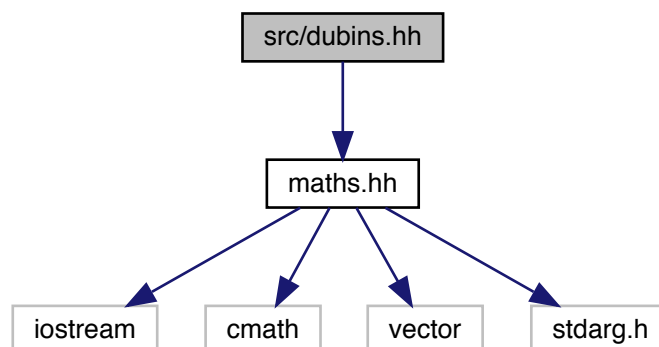
Include dependency graph for dubins.cc:



## 4.9 src/dubins.hh File Reference

```
#include "maths.hh"
```

Include dependency graph for dubins.hh:



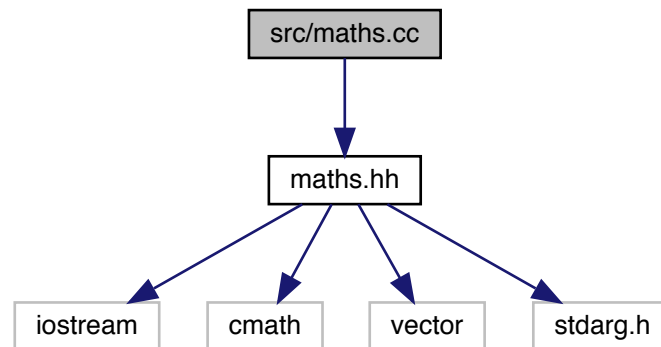




## 4.11 src/math.cc File Reference

```
#include "maths.hh"
```

Include dependency graph for math.cc:



## 4.12 src/math.h File Reference

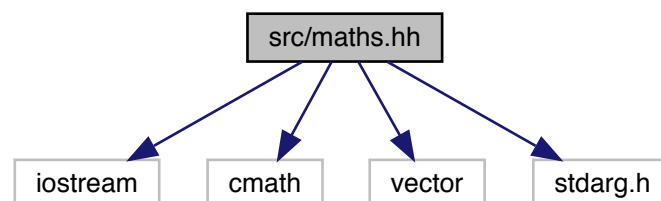
```
#include <iostream>
```

```
#include <cmath>
```

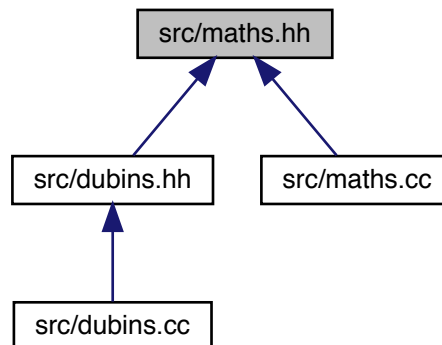
```
#include <vector>
```

```
#include <stdarg.h>
```

Include dependency graph for math.h:



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



## Classes

- class [Angle](#)  
*This class allows to save and handle angles. It supports DEG and RAD, operations such as addition and subtraction with operators overloading, conversion from RAD to DEG and viceversa and normalization of the angle.*
- class [Tuple< T >](#)
- class [Point2< T >](#)  
*Class that stores two value to construct a point in 2D. The value is saved in a [Tuple](#).*
- class [Configuration2< T1 >](#)  
*This class stores a configuration, that is a point and an angle.*

## Macros

- `#define DEGTORAD M_PI/180`
- `#define RADTODEG 180/M_PI`

## Enumerations

- enum [DISTANCE\\_TYPE](#) { [EUCLIDEAN](#), [MANHATTAN](#) }

## Functions

- `template<class T >`  
`T pow2 (const T x)`

### 4.12.1 Macro Definition Documentation

#### 4.12.1.1 DEGTORAD

```
#define DEGTORAD M_PI/180
```

#### 4.12.1.2 RADTO DEG

```
#define RADTO DEG 180/M_PI
```

### 4.12.2 Enumeration Type Documentation

#### 4.12.2.1 DISTANCE\_TYPE

```
enum DISTANCE_TYPE
```

##### Enumerator

EUCLIDEAN	
MANHATTAN	

### 4.12.3 Function Documentation

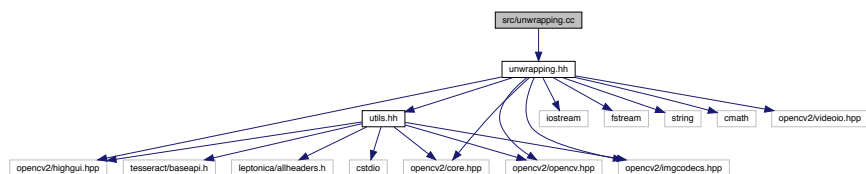
#### 4.12.3.1 pow2()

```
template<class T >
T pow2 (
    const T x ) [inline]
```

### 4.13 src/unwrapping.cc File Reference

```
#include "unwrapping.hh"
```

Include dependency graph for unwrapping.cc:



## Macros

- `#define WAIT`
- `#define area_ratio 0.7`

## Functions

- static float `distance` (Point c1, Point c2)  
*Compute the euclidean distance.*
- static void `swap` (int &a, int &b)  
*Swap the two integers passed.*
- int `unwrapping` ()  
*Take some images according to a xml and unwrap the black rectangle inside the image after applying undistortion trasformation.*
- void `loadCoefficients` (const string filename, Mat &camera\_matrix, Mat &dist\_coeffs)  
*Load coefficients from a file.*

## Variables

- const string `xml_settings` = "data/settings.xml"

### 4.13.1 Macro Definition Documentation

#### 4.13.1.1 area\_ratio

```
#define area_ratio 0.7
```

#### 4.13.1.2 WAIT

```
#define WAIT
```

### 4.13.2 Function Documentation

#### 4.13.2.1 distance()

```
static float distance (  
    Point c1,  
    Point c2 ) [static]
```

Compute the euclidean distance.

**Parameters**

in, out	<i>c1</i>	The first point.
in, out	<i>c2</i>	The second point.

**Returns**

The euclidean distance.

**4.13.2.2 loadCoefficients()**

```
void loadCoefficients (
    const string filename,
    Mat & camera_matrix,
    Mat & dist_coeffs )
```

Load coefficients from a file.

Load two matrix 'camera\_matrix' and 'distortion\_coefficients' from the xml file passed.

**Parameters**

in	<i>filename</i>	The string that identify the location of the xml file.
out	<i>camera_matrix</i>	Where the 'camera_matrix' matrix is saved.
out	<i>dist_coeffs</i>	Where the 'distortion_coefficients' matrix is saved.

**4.13.2.3 swap()**

```
static void swap (
    int & a,
    int & b ) [static]
```

Swap the two integers passed.

**Parameters**

in, out	<i>a</i>	First parameter.
in, out	<i>b</i>	Second parameter.

**4.13.2.4 unwrapping()**

```
int unwrapping ( )
```

Take some images according to a xml and unwrap the black rectangle inside the image after applying undistortion transformation.

Load from the xml file 'data/settings.xml' the name of some images, load the images from the file, apply the calibration (undistortion transformation) thanks to the matrices load with the 'loadCoefficients' function. Then, with the use of a filter for the black the region of interest (a rectangle) is identified and all the perspective is rotated for reach a top view of the rectangle. Finally, the images are saved on some files.

#### Returns

A 0 is return if the function reach the end.

### 4.13.3 Variable Documentation

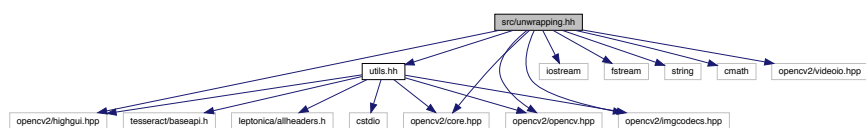
#### 4.13.3.1 xml\_settings

```
const string xml_settings = "data/settings.xml"
```

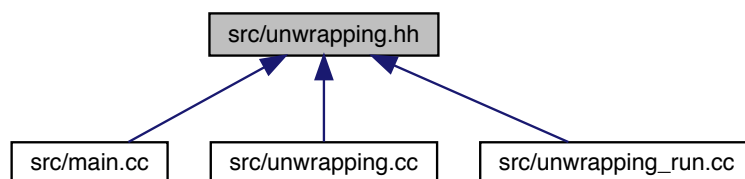
## 4.14 src/unwrapping.hh File Reference

```
#include "utils.hh"
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
```

Include dependency graph for unwrapping.hh:



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



## Functions

- int `unwrapping` ()  
Take some images according to a xml and unwrap the black rectangle inside the image after applying undistortion trasformation.
- void `loadCoefficients` (const string filename, Mat &camera\_matrix, Mat &dist\_coeffs)  
Load coefficients from a file.

### 4.14.1 Function Documentation

#### 4.14.1.1 loadCoefficients()

```
void loadCoefficients (
    const string filename,
    Mat & camera_matrix,
    Mat & dist_coeffs )
```

Load coefficients from a file.

Load two matrix 'camera\_matrix' and 'distortion\_coefficients' from the xml file passed.

##### Parameters

in	<i>filename</i>	The string that identify the location of the xml file.
out	<i>camera_matrix</i>	Where the 'camera_matrix' matrix is saved.
out	<i>dist_coeffs</i>	Where the 'distortion_coefficients' matrix is saved.

#### 4.14.1.2 unwrapping()

```
int unwrapping ( )
```

Take some images according to a xml and unwrap the black rectangle inside the image after applying undistortion trasformation.

Load from the xml file 'data/settings.xml' the name of some images, load the images from the file, apply the calibration (undistortion trasformation) thanks to the matrices load with the 'loadCoefficients' function. Then, with the use of a filter for the black the region of interest (a rectangle) is identified and all the perspective is rotated for reach a top view of the rectangle. Finally, the images are saved on some files.

##### Returns

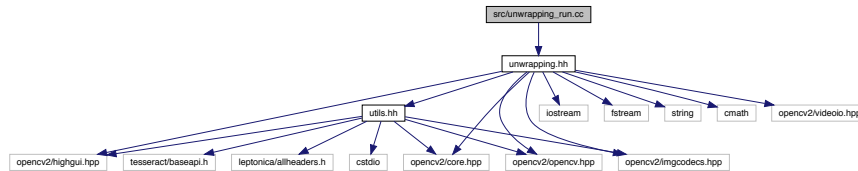
A 0 is return if the function reach the end.



## 4.15 src/unwrapping\_run.cc File Reference

```
#include "unwrapping.hh"
```

Include dependency graph for unwrapping\_run.cc:



### Functions

- int [main](#) ()

#### 4.15.1 Function Documentation

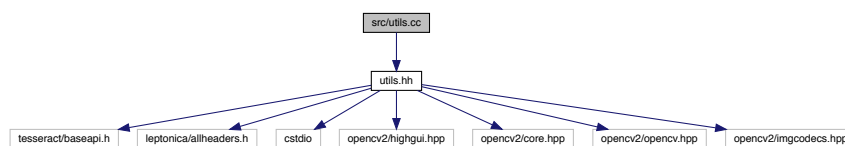
##### 4.15.1.1 main()

```
int main ( )
```

## 4.16 src/utils.cc File Reference

```
#include "utils.hh"
```

Include dependency graph for utils.cc:



### Functions

- void [my\\_imshow](#) (const char \*win\_name, Mat img, bool reset)

#### 4.16.1 Function Documentation

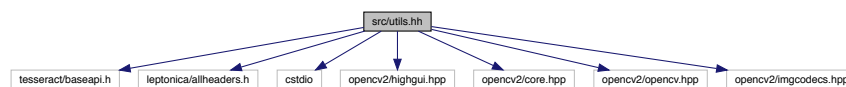
#### 4.16.1.1 my\_imshow()

```
void my_imshow (
    const char * win_name,
    Mat img,
    bool reset )
```

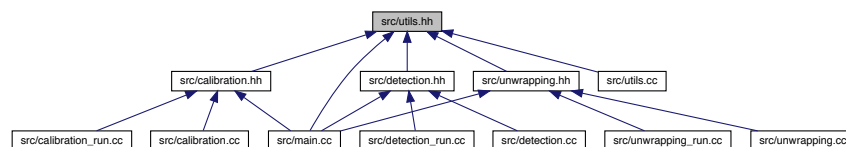
### 4.17 src/utils.hh File Reference

```
#include <tesseract/baseapi.h>
#include <leptonica/allheaders.h>
#include <cstdio>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
```

Include dependency graph for utils.hh:



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



### Macros

- `#define INFO(msg)`

### Functions

- `void my_imshow (const char *win_name, Mat img, bool reset=false)`

#### 4.17.1 Macro Definition Documentation

#### 4.17.1.1 INFO

```
#define INFO(  
    msg )
```

### 4.17.2 Function Documentation

#### 4.17.2.1 my\_imshow()

```
void my_imshow (  
    const char * win_name,  
    Mat img,  
    bool reset = false )
```



# Index

- ~dubins
  - dubins, [23](#)
- ANGLE\_TYPE
  - Angle, [6](#)
- add
  - Angle, [7](#)
  - Tuple, [41](#)
- Angle, [5](#)
  - ANGLE\_TYPE, [6](#)
  - add, [7](#)
  - Angle, [6](#), [7](#)
  - copy, [7](#)
  - degToRad, [8](#)
  - get, [8](#)
  - getType, [8](#)
  - getTypeName, [8](#)
  - normalize, [8](#)
  - operator<<, [13](#)
  - operator+, [8](#)
  - operator+=, [9](#)
  - operator-, [9](#)
  - operator-=, [9](#)
  - operator=, [11](#)
  - radToDeg, [11](#)
  - set, [11](#)
  - setType, [12](#)
  - sub, [12](#)
  - toDeg, [12](#)
  - toRad, [13](#)
- angle
  - Configuration2, [16](#)
- area\_ratio
  - unwrapping.cc, [67](#)
- aspectRatio
  - Settings, [35](#)
- atImageList
  - Settings, [35](#)
- boardSize
  - Settings, [35](#)
- calcBoardCornerPositions
  - calibration.cc, [46](#)
- calibFixPrincipalPoint
  - Settings, [35](#)
- calibZeroTangentDist
  - Settings, [35](#)
- calibration
  - calibration.cc, [46](#)
  - calibration.hh, [51](#)
- calibration.cc
  - calcBoardCornerPositions, [46](#)
  - calibration, [46](#)
  - computeReprojectionErrors, [46](#)
  - read, [47](#)
  - runCalibration, [47](#)
  - runCalibrationAndSave, [48](#)
  - saveCameraParams, [48](#)
- calibration.hh
  - calibration, [51](#)
  - runCalibrationAndSave, [51](#)
- calibration\_run.cc
  - main, [52](#)
- calibrationPattern
  - Settings, [35](#)
- cameraID
  - Settings, [36](#)
- computeReprojectionErrors
  - calibration.cc, [46](#)
- Configuration2
  - angle, [16](#)
  - Configuration2, [15](#)
  - distance, [16](#)
  - EuDistance, [17](#)
  - MaDistance, [17](#)
  - offset, [18](#)
  - offset\_angle, [19](#)
  - offset\_x, [19](#)
  - offset\_y, [19](#)
  - operator<<, [21](#)
  - x, [20](#)
  - y, [20](#), [21](#)
- Configuration2< T1 >, [13](#)
- copy
  - Angle, [7](#)
- create\_xml.cc
  - main, [53](#)
- crop\_number\_section
  - detection.cc, [54](#)
  - detection.hh, [58](#)
- DEGTORAD
  - maths.hh, [65](#)
- DISTANCE\_TYPE
  - maths.hh, [66](#)
- degToRad
  - Angle, [8](#)
- delay
  - Settings, [36](#)

- detection
  - detection.cc, [54](#)
  - detection.hh, [59](#)
- detection.cc
  - crop\_number\_section, [54](#)
  - detection, [54](#)
  - erode\_dilation, [54](#)
  - find\_contours, [55](#)
  - fs\_xml, [57](#)
  - load\_number\_template, [55](#)
  - number\_recognition, [55](#)
  - save\_convex\_hull, [56](#)
  - shape\_detection, [56](#)
  - templates, [57](#)
  - WAIT, [54](#)
  - xml\_settings, [57](#)
- detection.hh
  - crop\_number\_section, [58](#)
  - detection, [59](#)
  - erode\_dilation, [59](#)
  - find\_contours, [59](#)
  - load\_number\_template, [60](#)
  - number\_recognition, [60](#)
  - save\_convex\_hull, [60](#)
  - shape\_detection, [61](#)
- detection\_run.cc
  - main, [61](#)
- distance
  - Configuration2, [16](#)
  - Point2, [25](#)
  - Tuple, [41](#)
  - unwrapping.cc, [67](#)
- dubins, [22](#)
  - ~dubins, [23](#)
  - dubins, [22](#)
- erode\_dilation
  - detection.cc, [54](#)
  - detection.hh, [59](#)
- EuDistance
  - Configuration2, [17](#)
  - Point2, [25](#)
  - Tuple, [41](#)
- find\_contours
  - detection.cc, [55](#)
  - detection.hh, [59](#)
- fixK1
  - Settings, [36](#)
- fixK2
  - Settings, [36](#)
- fixK3
  - Settings, [36](#)
- fixK4
  - Settings, [36](#)
- fixK5
  - Settings, [37](#)
- flag
  - Settings, [37](#)
- flipVertical
  - Settings, [37](#)
- fs\_xml
  - detection.cc, [57](#)
- get
  - Angle, [8](#)
  - Tuple, [42](#)
- getType
  - Angle, [8](#)
- getTypeName
  - Angle, [8](#)
- goodInput
  - Settings, [37](#)
- INFO
  - utils.hh, [72](#)
- imageList
  - Settings, [37](#)
- input
  - Settings, [37](#)
- inputCapture
  - Settings, [37](#)
- inputType
  - Settings, [38](#)
- InputType
  - Settings, [32](#)
- isListOfImages
  - Settings, [32](#)
- load\_number\_template
  - detection.cc, [55](#)
  - detection.hh, [60](#)
- loadCoefficients
  - unwrapping.cc, [68](#)
  - unwrapping.hh, [70](#)
- MaDistance
  - Configuration2, [17](#)
  - Point2, [26](#)
  - Tuple, [42](#)
- main
  - calibration\_run.cc, [52](#)
  - create\_xml.cc, [53](#)
  - detection\_run.cc, [61](#)
  - main.cc, [63](#)
  - unwrapping\_run.cc, [71](#)
- main.cc
  - main, [63](#)
- maths.hh
  - DEGTORAD, [65](#)
  - DISTANCE\_TYPE, [66](#)
  - pow2, [66](#)
  - RADTODEG, [66](#)
- my\_imshow
  - utils.cc, [71](#)
  - utils.hh, [73](#)
- nextImage

- Settings, 33
- normalize
  - Angle, 8
- nrFrames
  - Settings, 38
- number\_recognition
  - detection.cc, 55
  - detection.hh, 60
- offset
  - Configuration2, 18
  - Point2, 26, 27
- offset\_angle
  - Configuration2, 19
- offset\_x
  - Configuration2, 19
  - Point2, 27
- offset\_y
  - Configuration2, 19
  - Point2, 27
- operator<<
  - Angle, 13
  - Configuration2, 21
  - Point2, 29
  - Tuple, 43
- operator+
  - Angle, 8
- operator+=
  - Angle, 9
- operator-
  - Angle, 9
- operator-=
  - Angle, 9
- operator=
  - Angle, 11
- outputFileName
  - Settings, 38
- Pattern
  - Settings, 32
- Point2
  - distance, 25
  - EuDistance, 25
  - MaDistance, 26
  - offset, 26, 27
  - offset\_x, 27
  - offset\_y, 27
  - operator<<, 29
  - Point2, 24
  - x, 28
  - y, 28, 29
- Point2< T >, 23
- pow2
  - maths.hh, 66
- RADTODEG
  - maths.hh, 66
- radToDeg
  - Angle, 11
- read
  - calibration.cc, 47
  - Settings, 33
- readStringList
  - Settings, 33
- remove
  - Tuple, 42
- runCalibration
  - calibration.cc, 47
- runCalibrationAndSave
  - calibration.cc, 48
  - calibration.hh, 51
- save\_convex\_hull
  - detection.cc, 56
  - detection.hh, 60
- saveCameraParams
  - calibration.cc, 48
- set
  - Angle, 11
  - Tuple, 43
- setType
  - Angle, 12
- Settings, 30
  - aspectRatio, 35
  - atImageList, 35
  - boardSize, 35
  - calibFixPrincipalPoint, 35
  - calibZeroTangentDist, 35
  - calibrationPattern, 35
  - cameraID, 36
  - delay, 36
  - fixK1, 36
  - fixK2, 36
  - fixK3, 36
  - fixK4, 36
  - fixK5, 37
  - flag, 37
  - flipVertical, 37
  - goodInput, 37
  - imageList, 37
  - input, 37
  - inputCapture, 37
  - inputType, 38
  - InputType , 32
  - isListOfImages, 32
  - nextImage, 33
  - nrFrames, 38
  - outputFileName, 38
  - Pattern, 32
  - read, 33
  - readStringList, 33
  - Settings, 32
  - showUndistorsed, 38
  - squareSize, 38
  - useFisheye, 38
  - validate, 34
  - write, 34
  - writeExtrinsics, 39

- writePoints, 39
- shape\_detection
  - detection.cc, 56
  - detection.hh, 61
- showUndistorsed
  - Settings, 38
- size
  - Tuple, 43
- squareSize
  - Settings, 38
- src/calibration.cc, 45
- src/calibration.hh, 49
- src/calibration\_run.cc, 52
- src/create\_xml.cc, 52
- src/detection.cc, 53
- src/detection.hh, 57
- src/detection\_run.cc, 61
- src/dubins.cc, 62
- src/dubins.hh, 62
- src/main.cc, 63
- src/math.cc, 64
- src/math.hh, 64
- src/unwrapping.cc, 66
- src/unwrapping.hh, 69
- src/unwrapping\_run.cc, 71
- src/Utils.cc, 71
- src/Utils.hh, 72
- sub
  - Angle, 12
- swap
  - unwrapping.cc, 68
- templates
  - detection.cc, 57
- toDeg
  - Angle, 12
- toRad
  - Angle, 13
- Tuple
  - add, 41
  - distance, 41
  - EuDistance, 41
  - get, 42
  - MaDistance, 42
  - operator<<, 43
  - remove, 42
  - set, 43
  - size, 43
  - Tuple, 40
- Tuple< T >, 39
- unwrapping
  - unwrapping.cc, 68
  - unwrapping.hh, 70
- unwrapping.cc
  - area\_ratio, 67
  - distance, 67
  - loadCoefficients, 68
  - swap, 68
  - unwrapping, 68
  - WAIT, 67
  - xml\_settings, 69
- unwrapping.hh
  - loadCoefficients, 70
  - unwrapping, 70
- unwrapping\_run.cc
  - main, 71
- useFisheye
  - Settings, 38
- utils.cc
  - my\_imshow, 71
- utils.hh
  - INFO, 72
  - my\_imshow, 73
- validate
  - Settings, 34
- WAIT
  - detection.cc, 54
  - unwrapping.cc, 67
- write
  - Settings, 34
- writeExtrinsics
  - Settings, 39
- writePoints
  - Settings, 39
- x
  - Configuration2, 20
  - Point2, 28
- xml\_settings
  - detection.cc, 57
  - unwrapping.cc, 69
- y
  - Configuration2, 20, 21
  - Point2, 28, 29