LabRoboticsProject

Generated by Doxygen 1.8.15

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 CHRONO Namespace Reference

Enumerations

- enum TIME_TYPE { SEC, MSEC, MUSEC, NSEC }

Functions

- string getType (TIME_TYPE type, string ret="")
- double getElapsed (Clock::time_point start, Clock::time_point stop, TIME_TYPE type=MUSEC)
- string getElapsed (Clock::time_point start, Clock::time_point stop, string ret, TIME_TYPE type=MUSEC)

5.1.1 Enumeration Type Documentation

5.1.1.1 TIME_TYPE

`enum CHRONO::TIME_TYPE`

Enumerator

SEC	
MSEC	
MUSEC	
NSEC	

5.1.2 Function Documentation

5.1.2.1 getElapsed() [1/2]

```
double CHRONO::getElapsed (
            Clock::time_point start,
            Clock::time_point stop,
            TIME_TYPE type = MUSEC )  [inline]
```

5.1.2.2 getElapsed() [2/2]

```
string CHRONO::getElapsed (
            Clock::time_point start,
            Clock::time_point stop,
            string ret,
            TIME_TYPE type = MUSEC )  [inline]
```

5.1.2.3 getType()

```
string CHRONO::getType (
            TIME_TYPE type,
            string ret = "" )  [inline]
```

5.2 ClipperLib Namespace Reference

Classes

- class Clipper
- class ClipperBase
- class clipperException
- class ClipperOffset
- struct DoublePoint
- class Int128
- struct IntersectNode
- struct IntPoint
- struct IntRect
- struct Join
- struct LocalMinimum
- struct LocMinSorter
- struct OutPt
- struct OutRec
- class PolyNode
- class PolyTree
- struct TEdge

Typedefs

- typedef signed long long cInt
- typedef signed long long long64
- typedef unsigned long long ulong64
- typedef std::vector< IntPoint > Path
- typedef std::vector< Path > Paths
- typedef std::vector< PolyNode ∗ > PolyNodes
- typedef std::vector< OutRec ∗ > PolyOutList
- typedef std::vector< TEdge ∗ > EdgeList
- typedef std::vector< Join ∗ > JoinList
- typedef std::vector< IntersectNode ∗ > IntersectList

Enumerations

- enum Direction { dRightToLeft, dLeftToRight }
- enum NodeType { ntAny, ntOpen, ntClosed }
- enum ClipType { ctIntersection, ctUnion, ctDifference, ctXor }
- enum PolyType { ptSubject, ptClip }
- enum PolyFillType { pftEvenOdd, pftNonZero, pftPositive, pftNegative }
- enum InitOptions { ioReverseSolution = 1, ioStrictlySimple = 2, ioPreserveCollinear = 4 }
- enum JoinType { jtSquare, jtRound, jtMiter }
- enum EndType {
  etClosedPolygon, etClosedLine, etOpenButt, etOpenSquare,
  etOpenRound }
- enum EdgeSide { esLeft = 1, esRight = 2 }

Functions

- cInt Round (double val)
- cInt Abs (cInt val)
- Int128 Int128Mul (long64 lhs, long64 rhs)
- bool Orientation (const Path &poly)
- double Area (const Path &poly)
- double Area (const OutPt ∗op)
- double Area (const OutRec &outRec)
- bool PointIsVertex (const IntPoint &Pt, OutPt ∗pp)
- int PointInPolygon (const IntPoint &pt, const Path &path)
- int PointInPolygon (const IntPoint &pt, OutPt ∗op)
- bool Poly2ContainsPoly1 (OutPt ∗OutPt1, OutPt ∗OutPt2)
- bool SlopesEqual (const TEdge &e1, const TEdge &e2, bool UseFullInt64Range)
- bool SlopesEqual (const IntPoint pt1, const IntPoint pt2, const IntPoint pt3, bool UseFullInt64Range)
- bool SlopesEqual (const IntPoint pt1, const IntPoint pt2, const IntPoint pt3, const IntPoint pt4, bool UseFull←Int64Range)
- bool IsHorizontal (TEdge &e)
- double GetDx (const IntPoint pt1, const IntPoint pt2)
- void SetDx (TEdge &e)
- void SwapSides (TEdge &Edge1, TEdge &Edge2)
- void SwapPolyIndexes (TEdge &Edge1, TEdge &Edge2)
- cInt TopX (TEdge &edge, const cInt currentY)
- void IntersectPoint (TEdge &Edge1, TEdge &Edge2, IntPoint &ip)
- void ReversePolyPtLinks (OutPt ∗pp)
- void DisposeOutPts (OutPt ∗&pp)

- void InitEdge (TEdge ∗e, TEdge ∗eNext, TEdge ∗ePrev, const IntPoint &Pt)
- void InitEdge2 (TEdge &e, PolyType Pt)
- TEdge ∗ RemoveEdge (TEdge ∗e)
- void ReverseHorizontal (TEdge &e)
- void SwapPoints (IntPoint &pt1, IntPoint &pt2)
- bool GetOverlapSegment (IntPoint pt1a, IntPoint pt1b, IntPoint pt2a, IntPoint pt2b, IntPoint &pt1, IntPoint &pt2)
- bool FirstIsBottomPt (const OutPt ∗btmPt1, const OutPt ∗btmPt2)
- OutPt ∗ GetBottomPt (OutPt ∗pp)
- bool Pt2IsBetweenPt1AndPt3 (const IntPoint pt1, const IntPoint pt2, const IntPoint pt3)
- bool HorzSegmentsOverlap (cInt seg1a, cInt seg1b, cInt seg2a, cInt seg2b)
- void RangeTest (const IntPoint &Pt, bool &useFullRange)
- TEdge ∗ FindNextLocMin (TEdge ∗E)
- OutRec ∗ GetLowermostRec (OutRec ∗outRec1, OutRec ∗outRec2)
- bool OutRec1RightOfOutRec2 (OutRec ∗outRec1, OutRec ∗outRec2)
- bool IsMinima (TEdge ∗e)
- bool IsMaxima (TEdge ∗e, const cInt Y)
- bool IsIntermediate (TEdge ∗e, const cInt Y)
- TEdge ∗ GetMaximaPair (TEdge ∗e)
- TEdge ∗ GetMaximaPairEx (TEdge ∗e)
- TEdge ∗ GetNextInAEL (TEdge ∗e, Direction dir)
- void GetHorzDirection (TEdge &HorzEdge, Direction &Dir, cInt &Left, cInt &Right)
- bool IntersectListSort (IntersectNode ∗node1, IntersectNode ∗node2)
- bool EdgesAdjacent (const IntersectNode &inode)
- int PointCount (OutPt ∗Pts)
- void SwapIntersectNodes (IntersectNode &int1, IntersectNode &int2)
- bool E2InsertsBeforeE1 (TEdge &e1, TEdge &e2)
- bool GetOverlap (const cInt a1, const cInt a2, const cInt b1, const cInt b2, cInt &Left, cInt &Right)
- void UpdateOutPtIdxs (OutRec &outrec)
- OutPt ∗ DupOutPt (OutPt ∗outPt, bool InsertAfter)
- bool JoinHorz (OutPt ∗op1, OutPt ∗op1b, OutPt ∗op2, OutPt ∗op2b, const IntPoint Pt, bool DiscardLeft)
- static OutRec ∗ ParseFirstLeft (OutRec ∗FirstLeft)
- DoublePoint GetUnitNormal (const IntPoint &pt1, const IntPoint &pt2)
- void ReversePath (Path &p)
- void ReversePaths (Paths &p)
- void SimplifyPolygon (const Path &in_poly, Paths &out_polys, PolyFillType fillType)
- void SimplifyPolygons (const Paths &in_polys, Paths &out_polys, PolyFillType fillType)
- void SimplifyPolygons (Paths &polys, PolyFillType fillType)
- double DistanceSqrd (const IntPoint &pt1, const IntPoint &pt2)
- double DistanceFromLineSqrd (const IntPoint &pt, const IntPoint &ln1, const IntPoint &ln2)
- bool SlopesNearCollinear (const IntPoint &pt1, const IntPoint &pt2, const IntPoint &pt3, double distSqrd)
- bool PointsAreClose (IntPoint pt1, IntPoint pt2, double distSqrd)
- OutPt ∗ ExcludeOp (OutPt ∗op)
- void CleanPolygon (const Path &in_poly, Path &out_poly, double distance)
- void CleanPolygon (Path &poly, double distance)
- void CleanPolygons (const Paths &in_polys, Paths &out_polys, double distance)
- void CleanPolygons (Paths &polys, double distance)
- void Minkowski (const Path &poly, const Path &path, Paths &solution, bool isSum, bool isClosed)
- void MinkowskiSum (const Path &pattern, const Path &path, Paths &solution, bool pathIsClosed)
- void TranslatePath (const Path &input, Path &output, const IntPoint delta)
- void MinkowskiSum (const Path &pattern, const Paths &paths, Paths &solution, bool pathIsClosed)
- void MinkowskiDiff (const Path &poly1, const Path &poly2, Paths &solution)
- void AddPolyNodeToPaths (const PolyNode &polynode, NodeType nodetype, Paths &paths)
- void PolyTreeToPaths (const PolyTree &polytree, Paths &paths)
- void ClosedPathsFromPolyTree (const PolyTree &polytree, Paths &paths)

- void OpenPathsFromPolyTree (PolyTree &polytree, Paths &paths)
- std::ostream & operator<< (std::ostream &s, const IntPoint &p)
- std::ostream & operator<< (std::ostream &s, const Path &p)
- std::ostream & operator<< (std::ostream &s, const Paths &p)
- Path & operator<< (Path &poly, const IntPoint &p)
- Paths & operator<< (Paths &polys, const Path &p)

Variables

- static double const pi = 3.141592653589793238
- static double const two_pi = pi *2
- static double const def_arc_tolerance = 0.25
- static int const Unassigned = -1
- static int const Skip = -2
- static cInt const loRange = 0x3FFFFFFF
- static cInt const hiRange = 0x3FFFFFFFFFFFFFFFLL

5.2.1 Typedef Documentation

5.2.1.1 cInt

```
typedef signed long long ClipperLib::cInt
```

5.2.1.2 EdgeList

```
typedef std::vector< TEdge* > ClipperLib::EdgeList
```

5.2.1.3 IntersectList

```
typedef std::vector< IntersectNode* > ClipperLib::IntersectList
```

5.2.1.4 JoinList

```
typedef std::vector< Join* > ClipperLib::JoinList
```

5.2.1.5 long64

```
typedef signed long long ClipperLib::long64
```

5.2.1.6 Path

```
typedef std::vector< IntPoint > ClipperLib::Path
```

5.2.1.7 Paths

```
typedef std::vector< Path > ClipperLib::Paths
```

5.2.1.8 PolyNodes

```
typedef std::vector< PolyNode* > ClipperLib::PolyNodes
```

5.2.1.9 PolyOutList

```
typedef std::vector< OutRec* > ClipperLib::PolyOutList
```

5.2.1.10 ulong64

```
typedef unsigned long long ClipperLib::ulong64
```

5.2.2 Enumeration Type Documentation

5.2.2.1 ClipType

```
enum ClipperLib::ClipType
```

Enumerator

ctIntersection	
ctUnion	
ctDifference	
ctXor	

5.2.2.2 Direction

enum ClipperLib::Direction

Enumerator

dRightToLeft	
dLeftToRight	

5.2.2.3 EdgeSide

enum ClipperLib::EdgeSide

Enumerator

esLeft	
esRight	

5.2.2.4 EndType

enum ClipperLib::EndType

Enumerator

etClosedPolygon	
etClosedLine	
etOpenButt	
etOpenSquare	
etOpenRound	

5.2.2.5 InitOptions

`enum ClipperLib::InitOptions`

Enumerator

ioReverseSolution	
ioStrictlySimple	
ioPreserveCollinear	

5.2.2.6 JoinType

`enum ClipperLib::JoinType`

Enumerator

jtSquare	
jtRound	
jtMiter	

5.2.2.7 NodeType

`enum ClipperLib::NodeType`

Enumerator

ntAny	
ntOpen	
ntClosed	

5.2.2.8 PolyFillType

`enum ClipperLib::PolyFillType`

Enumerator

pftEvenOdd	
pftNonZero	
pftPositive	
pftNegative	

5.2.2.9 PolyType

```
enum ClipperLib::PolyType
```

Enumerator

ptSubject	
ptClip	

5.2.3 Function Documentation

5.2.3.1 Abs()

```
cInt ClipperLib::Abs (
            cInt val ) [inline]
```

5.2.3.2 AddPolyNodeToPaths()

```
void ClipperLib::AddPolyNodeToPaths (
            const PolyNode & polynode,
            NodeType nodetype,
            Paths & paths )
```

5.2.3.3 Area() [1/3]

```
double ClipperLib::Area (
            const Path & poly )
```

5.2.3.4 Area() [2/3]

```
double ClipperLib::Area (
            const OutPt * op )
```

5.2.3.5 Area() [3/3]

```
double ClipperLib::Area (
            const OutRec & outRec )
```

5.2.3.6 CleanPolygon() [1/2]

```
void ClipperLib::CleanPolygon (
            const Path & in_poly,
            Path & out_poly,
            double distance )
```

5.2.3.7 CleanPolygon() [2/2]

```
void ClipperLib::CleanPolygon (
            Path & poly,
            double distance )
```

5.2.3.8 CleanPolygons() [1/2]

```
void ClipperLib::CleanPolygons (
            const Paths & in_polys,
            Paths & out_polys,
            double distance )
```

5.2.3.9 CleanPolygons() [2/2]

```
void ClipperLib::CleanPolygons (
            Paths & polys,
            double distance )
```

5.2.3.10 ClosedPathsFromPolyTree()

```
void ClipperLib::ClosedPathsFromPolyTree (
            const PolyTree & polytree,
            Paths & paths )
```

5.2.3.11 DisposeOutPts()

```
void ClipperLib::DisposeOutPts (
            OutPt *& pp )
```

5.2.3.12 DistanceFromLineSqrd()

```
double ClipperLib::DistanceFromLineSqrd (
            const IntPoint & pt,
            const IntPoint & ln1,
            const IntPoint & ln2 )
```

5.2.3.13 DistanceSqrd()

```
double ClipperLib::DistanceSqrd (
            const IntPoint & pt1,
            const IntPoint & pt2 ) [inline]
```

5.2.3.14 DupOutPt()

```
OutPt* ClipperLib::DupOutPt (
            OutPt * outPt,
            bool InsertAfter )
```

5.2.3.15 E2InsertsBeforeE1()

```
bool ClipperLib::E2InsertsBeforeE1 (
            TEdge & e1,
            TEdge & e2 ) [inline]
```

5.2.3.16 EdgesAdjacent()

```
bool ClipperLib::EdgesAdjacent (
            const IntersectNode & inode ) [inline]
```

5.2.3.17 ExcludeOp()

```
OutPt* ClipperLib::ExcludeOp (
            OutPt * op )
```

5.2.3.18 FindNextLocMin()

```
TEdge* ClipperLib::FindNextLocMin (
            TEdge * E )
```

5.2.3.19 FirstIsBottomPt()

```
bool ClipperLib::FirstIsBottomPt (
            const OutPt * btmPt1,
            const OutPt * btmPt2 )
```

5.2.3.20 GetBottomPt()

```
OutPt* ClipperLib::GetBottomPt (
            OutPt * pp )
```

5.2.3.21 GetDx()

```
double ClipperLib::GetDx (
            const IntPoint pt1,
            const IntPoint pt2 )  [inline]
```

5.2.3.22 GetHorzDirection()

```
void ClipperLib::GetHorzDirection (
            TEdge & HorzEdge,
            Direction & Dir,
            cInt & Left,
            cInt & Right )
```

5.2.3.23 GetLowermostRec()

```
OutRec* ClipperLib::GetLowermostRec (
            OutRec * outRec1,
            OutRec * outRec2 )
```

5.2.3.24 GetMaximaPair()

```
TEdge* ClipperLib::GetMaximaPair (
            TEdge * e )
```

5.2.3.25 GetMaximaPairEx()

```
TEdge* ClipperLib::GetMaximaPairEx (
            TEdge * e )
```

5.2.3.26 GetNextInAEL()

```
TEdge* ClipperLib::GetNextInAEL (
            TEdge * e,
            Direction dir )
```

5.2.3.27 GetOverlap()

```
bool ClipperLib::GetOverlap (
            const cInt a1,
            const cInt a2,
            const cInt b1,
            const cInt b2,
            cInt & Left,
            cInt & Right )
```

5.2.3.28 GetOverlapSegment()

```
bool ClipperLib::GetOverlapSegment (
            IntPoint pt1a,
            IntPoint pt1b,
            IntPoint pt2a,
            IntPoint pt2b,
            IntPoint & pt1,
            IntPoint & pt2 )
```

5.2.3.29 GetUnitNormal()

```
DoublePoint ClipperLib::GetUnitNormal (
             const IntPoint & pt1,
             const IntPoint & pt2 )
```

5.2.3.30 HorzSegmentsOverlap()

```
bool ClipperLib::HorzSegmentsOverlap (
             cInt seg1a,
             cInt seg1b,
             cInt seg2a,
             cInt seg2b )
```

5.2.3.31 InitEdge()

```
void ClipperLib::InitEdge (
             TEdge * e,
             TEdge * eNext,
             TEdge * ePrev,
             const IntPoint & Pt )  [inline]
```

5.2.3.32 InitEdge2()

```
void ClipperLib::InitEdge2 (
             TEdge & e,
             PolyType Pt )
```

5.2.3.33 Int128Mul()

```
Int128 ClipperLib::Int128Mul (
             long64 lhs,
             long64 rhs )
```

5.2.3.34 IntersectListSort()

```
bool ClipperLib::IntersectListSort (
             IntersectNode * node1,
             IntersectNode * node2 )
```

5.2.3.35 IntersectPoint()

```
void ClipperLib::IntersectPoint (
            TEdge & Edge1,
            TEdge & Edge2,
            IntPoint & ip )
```

5.2.3.36 IsHorizontal()

```
bool ClipperLib::IsHorizontal (
            TEdge & e )  [inline]
```

5.2.3.37 IsIntermediate()

```
bool ClipperLib::IsIntermediate (
            TEdge * e,
            const cInt Y )  [inline]
```

5.2.3.38 IsMaxima()

```
bool ClipperLib::IsMaxima (
            TEdge * e,
            const cInt Y )  [inline]
```

5.2.3.39 IsMinima()

```
bool ClipperLib::IsMinima (
            TEdge * e )  [inline]
```

5.2.3.40 JoinHorz()

```
bool ClipperLib::JoinHorz (
            OutPt * op1,
            OutPt * op1b,
            OutPt * op2,
            OutPt * op2b,
            const IntPoint Pt,
            bool DiscardLeft )
```

5.2.3.41 Minkowski()

```
void ClipperLib::Minkowski (
            const Path & poly,
            const Path & path,
            Paths & solution,
            bool isSum,
            bool isClosed )
```

5.2.3.42 MinkowskiDiff()

```
void ClipperLib::MinkowskiDiff (
            const Path & poly1,
            const Path & poly2,
            Paths & solution )
```

5.2.3.43 MinkowskiSum() [1/2]

```
void ClipperLib::MinkowskiSum (
            const Path & pattern,
            const Path & path,
            Paths & solution,
            bool pathIsClosed )
```

5.2.3.44 MinkowskiSum() [2/2]

```
void ClipperLib::MinkowskiSum (
            const Path & pattern,
            const Paths & paths,
            Paths & solution,
            bool pathIsClosed )
```

5.2.3.45 OpenPathsFromPolyTree()

```
void ClipperLib::OpenPathsFromPolyTree (
            PolyTree & polytree,
            Paths & paths )
```

5.2.3.46 operator<<() [1/5]

```
Path& ClipperLib::operator<< (
            Path & poly,
            const IntPoint & p )  [inline]
```

5.2.3.47 operator<<() [2/5]

```
Paths& ClipperLib::operator<< (
            Paths & polys,
            const Path & p )  [inline]
```

5.2.3.48 operator<<() [3/5]

```
std::ostream & ClipperLib::operator<< (
            std::ostream & s,
            const IntPoint & p )
```

5.2.3.49 operator<<() [4/5]

```
std::ostream & ClipperLib::operator<< (
            std::ostream & s,
            const Path & p )
```

5.2.3.50 operator<<() [5/5]

```
std::ostream & ClipperLib::operator<< (
            std::ostream & s,
            const Paths & p )
```

5.2.3.51 Orientation()

```
bool ClipperLib::Orientation (
            const Path & poly )
```

5.2.3.52 OutRec1RightOfOutRec2()

```
bool ClipperLib::OutRec1RightOfOutRec2 (
            OutRec * outRec1,
            OutRec * outRec2 )
```

5.2.3.53 ParseFirstLeft()

```
static OutRec* ClipperLib::ParseFirstLeft (
            OutRec * FirstLeft )  [static]
```

5.2.3.54 PointCount()

```
int ClipperLib::PointCount (
            OutPt * Pts )
```

5.2.3.55 PointInPolygon() [1/2]

```
int ClipperLib::PointInPolygon (
            const IntPoint & pt,
            const Path & path )
```

5.2.3.56 PointInPolygon() [2/2]

```
int ClipperLib::PointInPolygon (
            const IntPoint & pt,
            OutPt * op )
```

5.2.3.57 PointIsVertex()

```
bool ClipperLib::PointIsVertex (
            const IntPoint & Pt,
            OutPt * pp )
```

5.2.3.58 PointsAreClose()

```
bool ClipperLib::PointsAreClose (
            IntPoint pt1,
            IntPoint pt2,
            double distSqrd )
```

5.2.3.59 Poly2ContainsPoly1()

```
bool ClipperLib::Poly2ContainsPoly1 (
            OutPt * OutPt1,
            OutPt * OutPt2 )
```

5.2.3.60 PolyTreeToPaths()

```
void ClipperLib::PolyTreeToPaths (
            const PolyTree & polytree,
            Paths & paths )
```

5.2.3.61 Pt2IsBetweenPt1AndPt3()

```
bool ClipperLib::Pt2IsBetweenPt1AndPt3 (
            const IntPoint pt1,
            const IntPoint pt2,
            const IntPoint pt3 )
```

5.2.3.62 RangeTest()

```
void ClipperLib::RangeTest (
            const IntPoint & Pt,
            bool & useFullRange )
```

5.2.3.63 RemoveEdge()

```
TEdge* ClipperLib::RemoveEdge (
            TEdge * e )
```

5.2.3.64 ReverseHorizontal()

```
void ClipperLib::ReverseHorizontal (
            TEdge & e )  [inline]
```

5.2.3.65 ReversePath()

```
void ClipperLib::ReversePath (
            Path & p )
```

5.2.3.66 ReversePaths()

```
void ClipperLib::ReversePaths (
            Paths & p )
```

5.2.3.67 ReversePolyPtLinks()

```
void ClipperLib::ReversePolyPtLinks (
            OutPt * pp )
```

5.2.3.68 Round()

```
cInt ClipperLib::Round (
            double val )  [inline]
```

5.2.3.69 SetDx()

```
void ClipperLib::SetDx (
            TEdge & e )  [inline]
```

5.2.3.70 SimplifyPolygon()

```
void ClipperLib::SimplifyPolygon (
            const Path & in_poly,
            Paths & out_polys,
            PolyFillType fillType )
```

5.2.3.71 SimplifyPolygons() [1/2]

```
void ClipperLib::SimplifyPolygons (
            const Paths & in_polys,
            Paths & out_polys,
            PolyFillType fillType )
```

5.2.3.72 SimplifyPolygons() [2/2]

```
void ClipperLib::SimplifyPolygons (
            Paths & polys,
            PolyFillType fillType )
```

5.2.3.73 SlopesEqual() [1/3]

```
bool ClipperLib::SlopesEqual (
            const TEdge & e1,
            const TEdge & e2,
            bool UseFullInt64Range )
```

5.2.3.74 SlopesEqual() [2/3]

```
bool ClipperLib::SlopesEqual (
            const IntPoint pt1,
            const IntPoint pt2,
            const IntPoint pt3,
            bool UseFullInt64Range )
```

5.2.3.75 SlopesEqual() [3/3]

```
bool ClipperLib::SlopesEqual (
            const IntPoint pt1,
            const IntPoint pt2,
            const IntPoint pt3,
            const IntPoint pt4,
            bool UseFullInt64Range )
```

5.2.3.76 SlopesNearCollinear()

```
bool ClipperLib::SlopesNearCollinear (
            const IntPoint & pt1,
            const IntPoint & pt2,
            const IntPoint & pt3,
            double distSqrd )
```

5.2.3.77 SwapIntersectNodes()

```
void ClipperLib::SwapIntersectNodes (
            IntersectNode & int1,
            IntersectNode & int2 )
```

5.2.3.78 SwapPoints()

```
void ClipperLib::SwapPoints (
            IntPoint & pt1,
            IntPoint & pt2 )
```

5.2.3.79 SwapPolyIndexes()

```
void ClipperLib::SwapPolyIndexes (
            TEdge & Edge1,
            TEdge & Edge2 )  [inline]
```

5.2.3.80 SwapSides()

```
void ClipperLib::SwapSides (
            TEdge & Edge1,
            TEdge & Edge2 )  [inline]
```

5.2.3.81 TopX()

```
cInt ClipperLib::TopX (
            TEdge & edge,
            const cInt currentY )  [inline]
```

5.2.3.82 TranslatePath()

```
void ClipperLib::TranslatePath (
            const Path & input,
            Path & output,
            const IntPoint delta )
```

5.2.3.83 UpdateOutPtIdxs()

```
void ClipperLib::UpdateOutPtIdxs (
            OutRec & outrec )  [inline]
```

5.2.4 Variable Documentation

5.2.4.1 def_arc_tolerance

```
double const ClipperLib::def_arc_tolerance = 0.25  [static]
```

5.2.4.2 hiRange

```
cInt const ClipperLib::hiRange = 0x3FFFFFFFFFFFFFFFLL  [static]
```

5.2.4.3 loRange

```
cInt const ClipperLib::loRange = 0x3FFFFFFF  [static]
```

5.2.4.4 pi

```
double const ClipperLib::pi = 3.141592653589793238  [static]
```

5.2.4.5 Skip

```
int const ClipperLib::Skip = -2  [static]
```

5.2.4.6 two_pi

```
double const ClipperLib::two_pi = pi *2  [static]
```

5.2.4.7 Unassigned

```
int const ClipperLib::Unassigned = -1  [static]
```

5.3 DW Namespace Reference

Functions

- void init (x, y, GLfloat *vertices_buffer={0.0f})
- void changeBuffer (GLfloat *vertices_buffer, uint dim)

Variables

- GLFWwindow * window
- GLuint map_buffer

5.3.1 Function Documentation

5.3.1.1 changeBuffer()

```
void DW::changeBuffer (
            GLfloat * vertices_buffer,
            uint dim )
```

5.3.1.2 init()

```
void DW::init (
            x ,
            y ,
            GLfloat * vertices_buffer = {0.0f} )
```

5.3.2 Variable Documentation

5.3.2.1 map_buffer

```
GLuint DW::map_buffer
```

5.3.2.2 window

```
GLFWwindow* DW::window
```

5.4 Planning Namespace Reference

Functions

- vector< Point2< int > > convertToVP (const vector< vector< Point2< int > > > &arr)

  Convert a vector of vector of points into a vector of points (AKA collapse everything).
- vector< Point2< int > > convertToVP (const vector< vector< Configuration2< double > > > &arr)

  Convert a vector of vector of configurations into a vector of points (AKA collapse everything).
- vector< Configuration2< double > > convertToVC (const vector< vector< Configuration2< double > > > &arr)

  Convert a vector of vector of configurations into a vector of configurations (AKA collapse everything).
- vector< Configuration2< double > > convertToVC (const vector< vector< Point2< int > > > &arr)

  Convert a vector of vector of points into a vector of configurations (AKA collapse everything).
- void draw (const vector< vector< Point2< int > > > &vv, string name)

  Show in a window the representation of the map with the addition of the points and segment taken from the parameters.
- void draw (const vector< vector< Configuration2< double > > > &vv, string name)

  Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters.
- void draw (const vector< vector< Configuration2< double > > > &vv, const vector< Configuration2< double > > &left, const vector< Configuration2< double > > &right, string name)

  Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters. Plus a set of grey points (left vector) and black points (right vector).
- vector< Configuration2< double > > planning (const Mat &img)

  The function plan a route from the actual position of the robot up to the final gate through all the victims.
- void createMapp ()

  The goal is to load, all the neccessary data, from files and create a Mapp that store everything.
- void loadVVP (vector< vector< Point2< int > > > &vvp, FileNode fn)

  The function load from the given fileNode a vector of vectors of Point2<int>.
- void loadVP (vector< Point2< int > > &vp, FileNode fn)

  The function load from the given fileNode a vector of Point2<int>.
- int getNPoints ()

  Get the numper of points needed for the function sampleNpoints.
- int ** allocateAAInt (const int a, const int b)

  Allocate a dynamic 2D array of int.
- int *** allocateAAAInt (const int a, const int b, const int c)

  Allocate a dynamic 3D array of int.
- int **** allocateAAAAInt (const int a, const int b, const int c, const int d)

  Allocate a dynamic 4D array of int.

- double ** allocateAADouble (const int a, const int b)

    Allocate a dynamic 2D array of int.
- double *** allocateAAADouble (const int a, const int b, const int c)

    Allocate a dynamic 3D array of double.
- double **** allocateAAAADouble (const int a, const int b, const int c, const int d)

    Allocate a dynamic 4D array of double.
- Point2< int > ** allocateAAPointInt (const int a, const int b)

    Allocate a dynamic 2D array of Points.
- vector< vector< Point2< int > > > minPathNPointsWithChoice (const vector< Point2< int > > &vp, const double bonus, const bool angle)

    Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< vector< Point2< int > > > minPathNPoints (const vector< Point2< int > > &vp, const bool angle)

    Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > minPathTwoPoints (const Point2< int > &p0, const Point2< int > &p1, const bool angle)

    Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > minPathTwoPointsInternal (const Point2< int > &startP, const Point2< int > &endP, double **distances, Point2< int > **parents)

    Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- int angleSector (const double &d)

    Compute the sector of an angle.
- vector< Point2< int > > minPathTwoPointsInternalAngles (const Point2< int > &startP, const Point2< int > &endP, double ***distances, int ****parents, const double initialDir)
- void intToVect (int c, vector< int > &v)

    Converts an integer into the vector of its digits. The result is inverse respect to the given integer.
- void resetDistanceMap (double **distances, const double value)

    It reset, to the given value, the matrix of distances given, to compute again the minPath search.
- void resetDistanceMap (double ***distances, const double value)

    It reset, to the given value, the matrix of distances given, to compute again the minPath search.
- vector< Point2< int > > sampleNPoints (const vector< vector< Point2< int > > > &vvp, const int n)

    It extracts from the given vector of vector of points, a subset of points that always contains the first one and the last one of each vector.
- vector< Point2< int > > sampleNPoints (const vector< Point2< int > > &points, const int n)

    It extracts from the given vector of points, a subset of points that always contains the first one and the last one.
- vector< Point2< int > > samplePointsEachNCells (const vector< Point2< int > > &points, const int step)

    It extracts from the given vector of points, a subset of points that always contains the first one and the last one.
- void fromVcToPath (vector< Configuration2< double > > &vc, Path &path)

    Convert a vector of point to a path, from Enrico's notation to Paolo's notation.
- template<class T >
  bool check_dubins_D (Dubins< T > &D)
- template<class T >
  bool check_dubins_DS (DubinsSet< T > &DS)
- bool compute_roundabout_dubins (DubinsSet< double > &new_DS, Configuration2< double > _start, const vector< Configuration2< double > > &vC, uint &vC_id, bool gate=false)
- template<class T >
  DubinsSet< double > victims_dubins (const vector< Configuration2< T > > &vC1, const vector< Configuration2< T > > &vC2, uint &vC1_pos, uint &vC2_pos)
- template<class T >
  DubinsSet< double > start_end_dubins (const Configuration2< double > &anchorPoint, const vector< Configuration2< T > > &vConfs, uint &pos, const bool start)

- vector< Configuration2< double > > vvCtovC (Tuple< Tuple< Configuration2< double > > > vv)
- vector< Configuration2< double > > vvvCtovC (Tuple< Tuple< Tuple< Configuration2< double > > > > vvv)
- Angle compute_final_angle (Configuration2< double > gate)
- void inter_victims (vector< vector< Configuration2< double > > > &vvConfs, vector< int > &vI, DubinsSet< double > &path, vector< DubinsSet< double > > &victimV)
- void plan_dubins (const Configuration2< double > &_start, vector< vector< Configuration2< double > > > &vvConfs)
- template<class T >
  void deleteAA (T **arr, const int a)
- template<class T >
  void deleteAAA (T ***arr, const int a, const int b)
- template<class T >
  void deleteAAAA (T ****arr, const int a, const int b, const int c)

Variables

- Mapp * map
- Configuration2< double > conf
- const double angleRange = 12*M_PI/180
- const int nAngles = 90
- const int range = 3
- const double DEGTORAD
- static constexpr double baseDistance = -1.0
- static constexpr double baseDir = -1.0
- const int foundLimit = 20
- const int foundLimitAngles = 40
- static const int nPoints = 50
- constexpr double initialDistAllowed = 20.0

5.4.1 Function Documentation

5.4.1.1 allocateAAAADouble()

```
double **** Planning::allocateAAAADouble (
            const int a,
            const int b,
            const int c,
            const int d )
```

Allocate a dynamic 4D array of double.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.
in	*c*	The third dimension.
in	*d*	The fourth dimension.

Returns

The allocated array.

5.4.1.2 allocateAAAAInt()

```
int **** Planning::allocateAAAAInt (
            const int a,
            const int b,
            const int c,
            const int d )
```

Allocate a dynamic 4D array of int.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.
in	*c*	The third dimension.
in	*d*	The fourth dimension.

Returns

The allocated array.

5.4.1.3 allocateAAADouble()

```
double *** Planning::allocateAAADouble (
            const int a,
            const int b,
            const int c )
```

Allocate a dynamic 3D array of double.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.
in	*c*	The third dimension.

Returns

The allocated array.

5.4.1.4 allocateAAAInt()

```
int *** Planning::allocateAAAInt (
            const int a,
            const int b,
            const int c )
```

Allocate a dynamic 3D array of int.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.
in	*c*	The third dimension.

Returns

The allocated array.

5.4.1.5 allocateAADouble()

```
double ** Planning::allocateAADouble (
            const int a,
            const int b )
```

Allocate a dynamic 2D array of int.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.

Returns

The allocated array.

5.4.1.6 allocateAAInt()

```
int ** Planning::allocateAAInt (
            const int a,
            const int b )
```

Allocate a dynamic 2D array of int.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.

Returns

The allocated array.

5.4.1.7 allocateAAPointInt()

```
Point2< int > ** Planning::allocateAAPointInt (
            const int a,
            const int b )
```

Allocate a dynamic 2D array of Points.

Parameters

in	*a*	The first dimension.
in	*b*	The second dimension.

Returns

The allocated array.

5.4.1.8 angleSector()

```
int Planning::angleSector (
            const double & d )
```

Compute the sector of an angle.

Parameters

in	*d*	The initial angle in radiants. \rreturn The sector of the angle

5.4.1.9 check_dubins_D()

```
template<class T >
bool Planning::check_dubins_D (
            Dubins< T > & D )
```

5.4.1.10 check_dubins_DS()

```
template<class T >
bool Planning::check_dubins_DS (
            DubinsSet< T > & DS )
```

5.4.1.11 compute_final_angle()

```
Angle Planning::compute_final_angle (
            Configuration2< double > gate )
```

5.4.1.12 compute_roundabout_dubins()

```
bool Planning::compute_roundabout_dubins (
            DubinsSet< double > & new_DS,
            Configuration2< double > _start,
            const vector< Configuration2< double > > & vC,
            uint & vC_id,
            bool gate = false )
```

5.4.1.13 convertToVC() [1/2]

```
vector< Configuration2< double > > Planning::convertToVC (
            const vector< vector< Configuration2< double > > > & arr )
```

Convert a vector of vector of configurations into a vector of configurations (AKA collapse everything).

Parameters

in	*The*	vector of vector of configurations that needs to be collapsed

Returns

The new vector of configurations.

5.4.1.14 convertToVC() [2/2]

```
vector< Configuration2< double > > Planning::convertToVC (
            const vector< vector< Point2< int > > > & arr )
```

Convert a vector of vector of points into a vector of configurations (AKA collapse everything).

Parameters

in	*The*	vector of vector of points that needs to be collapsed

Returns

The new vector of configurations.

5.4.1.15 convertToVP() [1/2]

```
vector< Point2< int > > Planning::convertToVP (
            const vector< vector< Point2< int > > > & arr )
```

Convert a vector of vector of points into a vector of points (AKA collapse everything).

Parameters

in	*The*	vector of vector of points that needs to be collapsed

Returns

The new vector of points.

5.4.1.16 convertToVP() [2/2]

```
vector< Point2< int > > Planning::convertToVP (
            const vector< vector< Configuration2< double > > > & arr )
```

Convert a vector of vector of configurations into a vector of points (AKA collapse everything).

Parameters

in	*The*	vector of vector of configurations that needs to be collapsed

Returns

The new vector of points.

5.4.1.17 createMapp()

```
void Planning::createMapp ( )
```

The goal is to load, all the neccessary data, from files and create a Mapp that store everything.

Returns

The created mapp.

5.4.1.18 deleteAA()

```
template<class T >
void Planning::deleteAA (
            T ** arr,
            const int a )
```

5.4.1.19 deleteAAA()

```
template<class T >
void Planning::deleteAAA (
            T *** arr,
            const int a,
            const int b )
```

5.4.1.20 deleteAAAA()

```
template<class T >
void Planning::deleteAAAA (
            T **** arr,
            const int a,
            const int b,
            const int c )
```

5.4.1.21 draw() [1/3]

```
void Planning::draw (
            const vector< vector< Point2< int > > > & vv,
            string name )
```

Show in a window the representation of the map with the addition of the points and segment taken from the parameters.

Parameters

in	*vv*	A vector of vector of points that will be added to the map.
in	*name*	The name of the window that will be created.

5.4.1.22 draw() [2/3]

```
void Planning::draw (
            const vector< vector< Configuration2< double > > > & vv,
            string name )
```

Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters.

Parameters

in	*vv*	A vector of vector of configurations that will be added to the map.
in	*name*	The name of the window that will be created.

5.4.1.23 draw() [3/3]

```
void Planning::draw (
            const vector< vector< Configuration2< double > > > & vv,
            const vector< Configuration2< double > > & left,
            const vector< Configuration2< double > > & right,
            string name )
```

Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters. Plus a set of grey points (left vector) and black points (right vector).

Parameters

in	*vv*	A vector of vector of configurations that will be added to the map.
in	*left*	A set of grey points will be added to the map.
in	*right*	A set of bleck points will be added to the map.
in	*name*	The name of the window that will be created.

5.4.1.24 fromVcToPath()

```
void Planning::fromVcToPath (
            vector< Configuration2< double > > & vc,
            Path & path )
```

Convert a vector of point to a path, from Enrico's notation to Paolo's notation.

Parameters

in	*vp*	The sorce vector.
out	*path*	The destination path.

5.4.1.25 getNPoints()

```
int Planning::getNPoints ( )
```

Get the numper of points needed for the function sampleNpoints.

Returns

The number of points.

5.4.1.26 inter_victims()

```
void Planning::inter_victims (
            vector< vector< Configuration2< double > > > & vvConfs,
            vector< int > & vI,
            DubinsSet< double > & path,
            vector< DubinsSet< double > > & victimV )
```

5.4.1.27 intToVect()

```
void Planning::intToVect (
            int c,
            vector< int > & v )
```

Converts an integer into the vector of its digits. The result is inverse respect to the given integer.

Parameters

in	*c*	The to split into the vector.
out	*v*	The vector where the split will be saved.

5.4.1.28 loadVP()

```
void Planning::loadVP (
```

```
vector< Point2< int > > & vp,
FileNode fn )
```

The function load from the given fileNode a vector of Point2<int>.

Parameters

out	*vp*	The location where to save the loaded vector.
in	*fn*	The fileNode from which to load the vector.

5.4.1.29 loadVVP()

```
void Planning::loadVVP (
            vector< vector< Point2< int > > > & vvp,
            FileNode fn )
```

The function load from the given fileNode a vector of vectors of Point2<int>.

Parameters

out	*vvp*	The location where to save the loaded vector of vectors.
in	*fn*	The fileNode from which to load the vector of vectors.

5.4.1.30 minPathNPoints()

```
vector< vector< Point2< int > > > Planning::minPathNPoints (
            const vector< Point2< int > > & vp,
            const bool angle )
```

Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.

The function is based on a Breadth-first search (BFS).

Parameters

in	*p0*	The source point.
in	*p1*	The destination point.
in	*angle*	It is a boolean flag that says if (for the first segment) call the angle version of the minPath or not.

Returns

A vector of vector of points along the path (one for each cell of the grid of the map). Each vector is the best path for one connection, given n points there are n-1 connecctions.

5.4.1.31 minPathNPointsWithChoice()

```
vector< vector< Point2< int > > > Planning::minPathNPointsWithChoice (
            const vector< Point2< int > > & vp,
            const double bonus,
            const bool angle )
```

Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.

The function is based on a Breadth-first search (BFS). In addittion, the function considered the bonus choose if it is convinient to collect all the victims or only some of them, the bonus is given for each saved victim.

Parameters

in	*vp*	The n points that need to be connected.
in	*bonus*	It is the time in second as reward for each victim saved.
in	*angle*	It is a boolean flag that says if (for the segment starting from the robot) call the angle version of the minPath or not.

Returns

A vector of vector of points along the path (one for each cell of the grid of the map). Each vector is the best path for one connection, given n points there are n-1 connecctions.

5.4.1.32 minPathTwoPoints()

```
vector< Point2< int > > Planning::minPathTwoPoints (
            const Point2< int > & p0,
            const Point2< int > & p1,
            const bool angle )
```

Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.

The function is based on a Breadth-first search (BFS).

Parameters

in	*p0*	The source point.
in	*p1*	The destination point.
in	*angle*	It is a boolean flag that says if call the angle version of the minPath or not.

Returns

A vector of points along the path (one for each cell of the grid of the map).

5.4.1.33 minPathTwoPointsInternal()

```
vector< Point2< int > > Planning::minPathTwoPointsInternal (
            const Point2< int > & startP,
            const Point2< int > & endP,
            double ** distances,
            Point2< int > ** parents )
```

Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.

The function is based on a Breadth-first search (BFS).

Parameters

in	*startP*	The source point.
in	*endP*	The destination point.
in	*distances*	A matrix that is needed to store the distances of the visited cells.
in	*parents*	A matrix that is needed to store the parent of each cell (AKA the one that have discovered that cell with the minimum distance).

Returns

A vector of points along the path (one for each cell of the grid of the map).

5.4.1.34 minPathTwoPointsInternalAngles()

```
vector< Point2< int > > Planning::minPathTwoPointsInternalAngles (
            const Point2< int > & startP,
            const Point2< int > & endP,
            double *** distances,
            int **** parents,
            const double initialDir )
```

5.4.1.35 plan_dubins()

```
void Planning::plan_dubins (
            const Configuration2< double > & _start,
            vector< vector< Configuration2< double > > > & vvConfs )
```

Function to compute `Dubins` between the various points in the path.

Parameters

in	*start*	`Configuration2` that is the starting configuration.
	[in/out]	vvConfs Vector of vectors of `Configuration2` that are the points of the path.

5.4.1.36 planning()

```
vector< Configuration2< double > > Planning::planning (
            const Mat & img )
```

The function plan a route from the actual position of the robot up to the final gate through all the victims.

All the data about the objects are loaded from the files previously saved. Then a Mapp is created and on that structure, thanks to a minPath function and a lot of dubin curves, the best route is computed.

Parameters

in	*img*	It is a raw image of the scene that will be used from the localize function to find the starting state of the robot.

Returns

Two elements are returned: a pointer to the Mapp where all data are stored and a vector of points placed on the computed route.

5.4.1.37 resetDistanceMap() [1/2]

```
void Planning::resetDistanceMap (
            double ** distances,
            const double value )
```

It reset, to the given value, the matrix of distances given, to compute again the minPath search.

Parameters

out	*distances*	It is the array that need to be initialized.
in	*value*	The value to be set.

5.4.1.38 resetDistanceMap() [2/2]

```
void Planning::resetDistanceMap (
            double *** distances,
            const double value )
```

It reset, to the given value, the matrix of distances given, to compute again the minPath search.

Parameters

out	*distances*	It is the array that need to be initialized.
in	*value*	The value to be set.

5.4.1.39 sampleNPoints() [1/2]

```
vector< Point2< int > > Planning::sampleNPoints (
            const vector< vector< Point2< int > > > & vvp,
            const int n )
```

It extracts from the given vector of vector of points, a subset of points that always contains the first one and the last one of each vector.

Parameters

in	*n*	The n number of points to sample.
in	*points*	The vector of vector of points to be selected.

Returns

The vector containing the subset of n points.

5.4.1.40 sampleNPoints() [2/2]

```
vector< Point2< int > > Planning::sampleNPoints (
            const vector< Point2< int > > & points,
            const int n )
```

It extracts from the given vector of points, a subset of points that always contains the first one and the last one.

Parameters

in	*n*	The number of points to select exept the extremes, it must be greater or equal than 2.
in	*points*	The vector of points to be selected.

Returns

The vector containing the subset of n points.

5.4.1.41 samplePointsEachNCells()

```
vector< Point2< int > > Planning::samplePointsEachNCells (
            const vector< Point2< int > > & points,
            const int step )
```

It extracts from the given vector of points, a subset of points that always contains the first one and the last one.

Parameters

in	*step*	The distance (counted as cells) from the previous to the next cell, it must but >=2 to have a reason.
in	*points*	The vector of points to be selected.

Returns

The vector containing the subset of points, each step cells.

5.4.1.42 start_end_dubins()

```
template<class T >
DubinsSet<double> Planning::start_end_dubins (
            const Configuration2< double > & anchorPoint,
            const vector< Configuration2< T > > & vConfs,
            uint & pos,
            const bool start )
```

5.4.1.43 victims_dubins()

```
template<class T >
DubinsSet<double> Planning::victims_dubins (
            const vector< Configuration2< T > > & vC1,
            const vector< Configuration2< T > > & vC2,
            uint & vC1_pos,
            uint & vC2_pos )
```

5.4.1.44 vvCtovC()

```
vector<Configuration2<double> > Planning::vvCtovC (
            Tuple< Tuple< Configuration2< double > > > vv )
```

5.4.1.45 vvvCtovC()

```
vector<Configuration2<double> > Planning::vvvCtovC (
            Tuple< Tuple< Tuple< Configuration2< double > > > > vvv )
```

5.4.2 Variable Documentation

5.4.2.1 angleRange

```
const double Planning::angleRange = 12*M_PI/180
```

5.4.2.2 baseDir

```
constexpr double Planning::baseDir = -1.0 [static]
```

5.4.2.3 baseDistance

```
constexpr double Planning::baseDistance = -1.0 [static]
```

5.4.2.4 conf

```
Configuration2< double > Planning::conf
```

5.4.2.5 DEGTORAD

```
const double Planning::DEGTORAD
```

5.4.2.6 foundLimit

```
const int Planning::foundLimit = 20
```

5.4.2.7 foundLimitAngles

```
const int Planning::foundLimitAngles = 40
```

5.4.2.8 initialDistAllowed

```
constexpr double Planning::initialDistAllowed = 20.0
```

5.4.2.9 map

```
Mapp * Planning::map
```

5.4.2.10 nAngles

```
const int Planning::nAngles = 90
```

5.4.2.11 nPoints

```
const int Planning::nPoints = 50 [static]
```

5.4.2.12 range

```
const int Planning::range = 3
```

5.5 timeutils Namespace Reference

Functions

- int64_t timespecDiff (struct timespec *timeA_p, struct timespec *timeB_p)
- double getTimeS ()

5.5.1 Function Documentation

5.5.1.1 getTimeS()

```
double timeutils::getTimeS ( )
```

5.5.1.2 timespecDiff()

```
int64_t timeutils::timespecDiff (
            struct timespec * timeA_p,
            struct timespec * timeB_p )
```

Chapter 6

Class Documentation

6.1 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=RAD)

  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 $[0, 360)$. Moreover it check if the value is infinite or NaN. In this case the `type` is set to `INVALID`.
- 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.
- template<class T1 >
  Angle mul (const T1 A)

  Multiply and angle by a costant. In the process a new angle is created so `normalize()` is also called.
- template<class T1 >
  Angle div (const T1 A)

  Divide and angle by a costant. 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)
- template<class T1 >
  Angle operator * (const T1 A)
- template<class T1 >
  Angle operator/ (const T1 A)
- Angle operator= (const Angle phi)
- Angle operator= (const double phi)
- Angle & operator+= (const Angle phi)
- Angle & operator-= (const Angle phi)
- template<class T >
  Angle & operator *= (const T A)
- template<class T >
  Angle & operator/= (const T A)
- bool equal (const Angle &phi)
- bool less (const Angle &phi)
- bool greater (const Angle &phi)
- bool operator== (const Angle &phi)
- bool operator!= (const Angle &phi)
- bool operator< (const Angle &phi)
- bool operator> (const Angle &phi)
- bool operator<= (const Angle &phi)
- bool operator>= (const Angle &phi)
- double cos () const

  Compute the cosine of the angle. \retunrs A `double` that is the cosine of the angle.
- double sin () const

  Compute the sine of the angle. \retunrs A `double` that is the sine of the angle.
- double tan () const

  Compute the tangent of the angle. \retunrs A `double` that is the tangent of the angle.
- operator int () const

  Cast to int.
- operator double () const

  Cast to double.
- operator float () const

  Cast to float.
- operator long () const

  Cast to long.
- stringstream to_string (ANGLE_TYPE _type=INVALID) const

Static Public Member Functions

- static bool checkValue (const double th)

Friends

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

6.1.1 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.

6.1.2 Member Enumeration Documentation

6.1.2.1 ANGLE_TYPE

```
enum Angle::ANGLE_TYPE
```

Enumerator

DEG	
RAD	
INVALID	

6.1.3 Constructor & Destructor Documentation

6.1.3.1 Angle() [1/2]

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

A void constructor to create an angle.

6.1.3.2 Angle() [2/2]

```
Angle::Angle (
            double _th,
            ANGLE_TYPE _type = RAD )  [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	*_th*	The dimension of the angle.
in	*_type*	The type of the angle.

6.1.4 Member Function Documentation

6.1.4.1 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	*phi*	The angle to be summed.

Returns

The angle summed.

6.1.4.2 checkValue()

```
static bool Angle::checkValue (
            const double th )  [inline], [static]
```

6.1.4.3 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	*A*	The angle to be copied.

Returns

The new angle.

6.1.4.4 cos()

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

Compute the cosine of the angle. \retunrs A `double` that is the cosine of the angle.

6.1.4.5 degToRad()

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

Convert and store the angle from DEG to RAD.

Returns

The value of the angle.

6.1.4.6 div()

```
template<class T1 >
Angle Angle::div (
            const T1 A )  [inline]
```

Divide and angle by a costant. In the process a new angle is created so `normalize()` is also called.

Template Parameters

The	type of the dividend.

Parameters

in	*A*	The costant to use to divide.

Returns

The angle divided.

6.1.4.7 equal()

```
bool Angle::equal (
            const Angle & phi ) [inline]
```

This function takes an angle to copare, an using the `equal` function for `double`s calculates if it is equal or not to `this`.

Parameters

in	*phi*	The angle to compare.

Returns

`true` if the two angle are equal, `false` otherwise.

6.1.4.8 get()

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

Returns the dimension of the angle.

6.1.4.9 getType()

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

Returns the type of the angle.

6.1.4.10 getTypeName()

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

<Returns a string that tells the type of angle.

6.1.4.11 greater()

```
bool Angle::greater (
            const Angle & phi ) [inline]
```

This function takes the value in radiants of an angle and compares it with this.

Parameters

in	*phi*	The angle to compare.

Returns

`true` if this is more than phi, `false` otherwise.

6.1.4.12 less()

```
bool Angle::less (
            const Angle & phi )  [inline]
```

This function takes the value in radiants of an angle and compares it with this.

Parameters

in	*phi*	The angle to compare.

Returns

`true` if this is less than phi, `false` otherwise.

6.1.4.13 mul()

```
template<class T1 >
Angle Angle::mul (
            const T1 A )  [inline]
```

Multiply and angle by a costant. In the process a new angle is created so `normalize()` is also called.

Template Parameters

The	type of the coefficient.

Parameters

in	*phi*	The costant to use to multiply.

Returns

The angle multiplied.

6.1.4.14 normalize()

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

Normalize the angle, that is to set it in $[0, 2\pi)$ or $[0, 360)$. Moreover it check if the value is infinite or NaN. In this case the `type` is set to `INVALID`.

6.1.4.15 operator *()

```
template<class T1 >
Angle Angle::operator * (
            const T1 A ) [inline]
```

This function overload the operator $*$. It simply calls the `mul()` function.

Template Parameters

The	type of the coefficient.

Parameters

in	*A*	The coefficient.

Returns

The angle multiplied.

6.1.4.16 operator *=()

```
template<class T >
Angle& Angle::operator *= (
            const T A ) [inline]
```

This function overload the operator $*$=. It simply calls the `mul()` function and then assign the result to this.

Parameters

in	*A*	The coefficient.

Returns

`this`.

6.1.4.17 operator double()

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

Cast to double.

Returns

The value in RAD of the angle casted to double

6.1.4.18 operator float()

```
Angle::operator float ( ) const [inline]
```

Cast to float.

Returns

The value in RAD of the angle casted to float

6.1.4.19 operator int()

```
Angle::operator int ( ) const [inline]
```

Cast to int.

Returns

The value in RAD of the angle casted to int

6.1.4.20 operator long()

```
Angle::operator long ( ) const [inline]
```

Cast to long.

Returns

The value in RAD of the angle casted to long

6.1.4.21 operator"!=()

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

This function overload the operator !=. It simply calls the `equal()` function and negates it.

Parameters

in	*phi*	The second angle.

Returns

`false` if the two angle are equal, `true` otherwise.

6.1.4.22 operator+()

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

This function overload the operator +. It simply calls the `add()` function.

Parameters

in	*phi*	The angle to be summed.

Returns

The angle summed.

6.1.4.23 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	*phi*	The angle to be summed.

Returns

`this`.

6.1.4.24 operator-()

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

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

Parameters

in	*phi*	The angle to be subtracted.

Returns

The angle subtracted.

6.1.4.25 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	*phi*	The angle to be subtracted.

Returns

`this`.

6.1.4.26 operator/()

```
template<class T1 >
Angle Angle::operator/ (
            const T1 A )  [inline]
```

This function overload the operator /. It simply calls the `div()` function.

Template Parameters

The	type of the dividend.

Parameters

in	*A*	The dividend.

Returns

The angle divided.

6.1.4.27 operator/=()

```
template<class T >
Angle& Angle::operator/= (
              const T A )  [inline]
```

This function overload the operator /=. It simply calls the `div()` function and then assign the result to this.

Parameters

in	*A*	The dividend.

Returns

`this`.

6.1.4.28 operator<()

```
bool Angle::operator< (
              const Angle & phi )  [inline]
```

This function overload the operator $<$. It simply calls the `less()` function.

Parameters

in	*phi*	The second angle.

Returns

`true` if the first angle (this) is less than the second one, `false` otherwise.

6.1.4.29 operator<=()

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

This function overload the operator $<$. It simply calls the `less()` function and `equal()` function.

Parameters

in	*phi*	The second angle.

Returns

`true` if the first angle (this) is less or equal than the second one, `false` otherwise.

6.1.4.30 operator=() [1/2]

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

This function overload the operator =. It simply calls the `copy()` function.

Parameters

in	*phi*	The angle to be copied.

Returns

The new angle.

6.1.4.31 operator=() [2/2]

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

6.1.4.32 operator==()

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

This function overload the operator ==. It simply calls the `equal()` function.

Parameters

in	*phi*	The second angle.

Returns

`true` if the two angle are equal, `false` otherwise.

6.1.4.33 operator>()

```
bool Angle::operator> (
             const Angle & phi )  [inline]
```

This function overload the operator $>$. It simply calls the `greater()` function.

Parameters

in	*phi*	The second angle.

Returns

`true` if the first angle (this) is greater than the second one, `false` otherwise.

6.1.4.34 operator>=()

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

This function overload the operator $<$. It simply calls the `greater()` function and `equal()` function.

Parameters

in	*phi*	The second angle.

Returns

`true` if the first angle (this) is greater or equal than the second one, `false` otherwise.

6.1.4.35 radToDeg()

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

Converts and stores the angle from RAD to DEG.

Returns

The value of the angle.

6.1.4.36 set()

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

Set the value of the angle.

Template Parameters

T	The programming type for the value to be stored. It's then cast to `double`.

Parameters

in	↩ _↩ *th*	The dimension of the angle to be stored.

6.1.4.37 setType()

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

Set the type of the angle.

Parameters

in	↩ _↩ *th*	The type of the angle to be stored.

6.1.4.38 sin()

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

Compute the sine of the angle. \retunrs A `double` that is the sine of the angle.

6.1.4.39 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	*phi*	The angle to be subtracted.

Returns

The angle subtracted.

6.1.4.40 tan()

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

Compute the tangent of the angle. \retunrs A `double` that is the tangent of the angle.

6.1.4.41 to_string()

```
stringstream Angle::to_string (
            ANGLE_TYPE _type = INVALID ) const  [inline]
```

This function create a strinstream object containing the most essential info, that is the dimension and the type of angle.

Parameters

in	*The*	type of values to be printed. Default is set to INVALID and it'll print the data of the `Angle` as it was saved.

Returns

A string stream.

6.1.4.42 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

6.1.4.43 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

6.1.5 Friends And Related Function Documentation

6.1.5.1 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	*out*	The out stream.
in	*data*	The angle to print.

Returns

An output stream to be printed.

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

- src/include/maths.hh

6.2 CalSettings Class Reference

`#include <calibration.hh>`

Public Types

- enum Pattern { NOT_EXISTING =0, CHESSBOARD =1 }
- enum InputType { INVALID =0, IMAGE_LIST =3 }

Public Member Functions

- CalSettings ()

  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 retrive 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 showUndistorsed

  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

6.2.1 Member Enumeration Documentation

6.2.1.1 InputType

`enum CalSettings::InputType`

Enumerator

INVALID	
IMAGE_LIST	

6.2.1.2 Pattern

`enum CalSettings::Pattern`

Enumerator

NOT_EXISTING	
CHESSBOARD	

6.2.2 Constructor & Destructor Documentation

6.2.2.1 CalSettings()

`CalSettings::CalSettings ( ) [inline]`

Constructor that sets `goodInput` to false.

6.2.3 Member Function Documentation

6.2.3.1 isListOfImages()

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

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

Parameters

in	*filename*	The name of the file to check for validity.

Returns

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

6.2.3.2 nextImage()

```
Mat CalSettings::nextImage ( )
```

Get next image from list.

Returns

A matrix containing the next image to consider.

6.2.3.3 read()

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

Read serialization.

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

Parameters

in	*node*	The node of the file to consider.

6.2.3.4 readStringList()

```
bool CalSettings::readStringList (
```

```
const string & filename,
vector< string > & l )  [static]
```

Read from file a list of images.

Parameters

in	*filename*	The name of the file from which to read.
out	*l*	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.

6.2.3.5 validate()

```
void CalSettings::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 `CalSettings` 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.

6.2.3.6 write()

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

Write serialization.

This function write data to a file.

Parameters

in	*fs*	The filename where to write.

6.2.4 Member Data Documentation

6.2.4.1 aspectRatio

```
float CalSettings::aspectRatio
```

The aspect ratio.

6.2.4.2 atImageList

```
size_t CalSettings::atImageList
```

6.2.4.3 boardSize

```
Size CalSettings::boardSize
```

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

6.2.4.4 calibFixPrincipalPoint

```
bool CalSettings::calibFixPrincipalPoint
```

Fix the principal point at the center.

6.2.4.5 calibrationPattern

```
Pattern CalSettings::calibrationPattern = CHESSBOARD
```

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

6.2.4.6 calibZeroTangentDist

```
bool CalSettings::calibZeroTangentDist
```

Assume zero tangential distortion.

6.2.4.7 cameraID

```
int CalSettings::cameraID
```

6.2.4.8 delay

```
int CalSettings::delay
```

In case of a video input.

6.2.4.9 fixK1

```
bool CalSettings::fixK1
```

fix K1 distortion coefficient

6.2.4.10 fixK2

```
bool CalSettings::fixK2
```

fix K2 distortion coefficient

6.2.4.11 fixK3

```
bool CalSettings::fixK3
```

fix K3 distortion coefficient

6.2.4.12 fixK4

```
bool CalSettings::fixK4
```

fix K4 distortion coefficient

6.2.4.13 fixK5

```
bool CalSettings::fixK5
```

fix K5 distortion coefficient

6.2.4.14 flag

```
int CalSettings::flag
```

6.2.4.15 flipVertical

```
bool CalSettings::flipVertical
```

Flip the captured images around the horizontal axis.

6.2.4.16 goodInput

```
bool CalSettings::goodInput
```

6.2.4.17 imageList

```
vector<string> CalSettings::imageList
```

6.2.4.18 input

```
string CalSettings::input
```

The input.

6.2.4.19 inputCapture

```
VideoCapture CalSettings::inputCapture
```

6.2.4.20 inputType

```
InputType CalSettings::inputType = IMAGE_LIST
```

6.2.4.21 nrFrames

```
int CalSettings::nrFrames
```

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

6.2.4.22 outputFileName

```
string CalSettings::outputFileName
```

The name of the file where to write.

6.2.4.23 showUndistorsed

```
bool CalSettings::showUndistorsed
```

Show undistorted images after calibration.

6.2.4.24 squareSize

```
float CalSettings::squareSize
```

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

6.2.4.25 useFisheye

```
bool CalSettings::useFisheye = false
```

use fisheye camera model for calibration

6.2.4.26 writeExtrinsics

```
bool CalSettings::writeExtrinsics
```

Write extrinsic parameters.

6.2.4.27 writePoints

```
bool CalSettings::writePoints
```

Write detected feature points.

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

- src/include/calibration.hh
- src/calibration.cc

6.3 CameraCapture Class Reference

```
#include <camera_capture.hh>
```

Inherits VideoCapture.

Classes

- struct input_options_t
  Structure for store the input option for the class CameraCapture.

Public Member Functions

- CameraCapture (input_options_t options)
- bool grab (cv::Mat &img, double &timestamp)
- bool isOpened ()
- bool isAlive ()
- ∼CameraCapture ()
- bool startCamera ()
- bool loadCoefficients (std::string const &filename)

6.3.1 Constructor & Destructor Documentation

6.3.1.1 CameraCapture()

```
CameraCapture::CameraCapture (
            input_options_t options )
```

Initializer of the camera capture class

Parameters

options	for the class

Returns

6.3.1.2 ∼CameraCapture()

```
CameraCapture::∼CameraCapture ( )
```

release the resource

6.3.2 Member Function Documentation

6.3.2.1 grab()

```
bool CameraCapture::grab (
            cv::Mat & img,
            double & timestamp )
```

Grab the first frame available and store it in frame variable

Returns

success if a frame is grabbed, false if not

6.3.2.2 isAlive()

```
bool CameraCapture::isAlive ( )
```

Check if the videostream is alive

Returns

true if open, false if not

6.3.2.3 isOpened()

```
bool CameraCapture::isOpened ( )
```

Check if the videostream is opened

Returns

true if open, false if not

6.3.2.4 loadCoefficients()

```
bool CameraCapture::loadCoefficients (
            std::string const & filename )
```

6.3.2.5 startCamera()

```
bool CameraCapture::startCamera ( )
```

get time in ns

Returns

time in ns

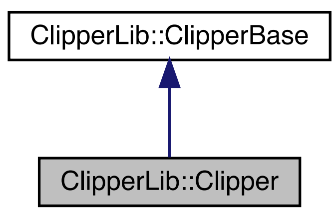
The documentation for this class was generated from the following files:

- src/include/camera_capture.hh
- src/camera_capture.cc

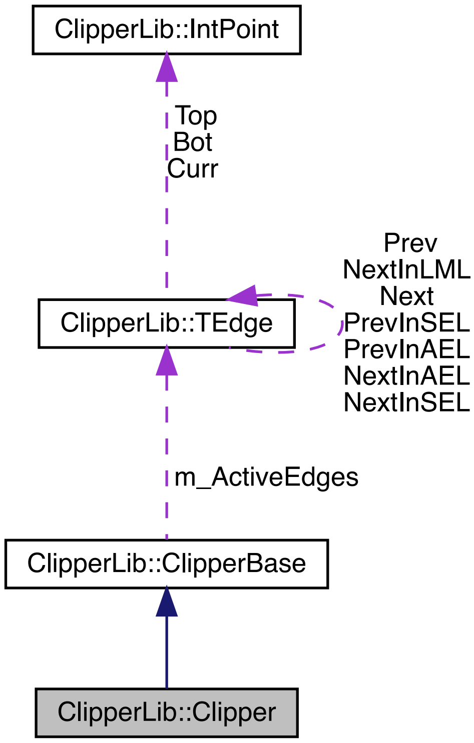
6.4 ClipperLib::Clipper Class Reference

```
#include <clipper.hh>
```

Inheritance diagram for ClipperLib::Clipper:



Collaboration diagram for ClipperLib::Clipper:



Public Member Functions

- Clipper (int initOptions=0)
- bool Execute (ClipType clipType, Paths &solution, PolyFillType fillType=pftEvenOdd)
- bool Execute (ClipType clipType, Paths &solution, PolyFillType subjFillType, PolyFillType clipFillType)
- bool Execute (ClipType clipType, PolyTree &polytree, PolyFillType fillType=pftEvenOdd)
- bool Execute (ClipType clipType, PolyTree &polytree, PolyFillType subjFillType, PolyFillType clipFillType)
- bool ReverseSolution ()
- void ReverseSolution (bool value)
- bool StrictlySimple ()
- void StrictlySimple (bool value)

Protected Member Functions

- virtual bool ExecuteInternal ()

Additional Inherited Members

6.4.1 Constructor & Destructor Documentation

6.4.1.1 Clipper()

```
ClipperLib::Clipper::Clipper (
            int initOptions = 0 )
```

6.4.2 Member Function Documentation

6.4.2.1 Execute() [1/4]

```
bool ClipperLib::Clipper::Execute (
            ClipType clipType,
            Paths & solution,
            PolyFillType fillType = pftEvenOdd )
```

6.4.2.2 Execute() [2/4]

```
bool ClipperLib::Clipper::Execute (
            ClipType clipType,
            Paths & solution,
            PolyFillType subjFillType,
            PolyFillType clipFillType )
```

6.4.2.3 Execute() [3/4]

```
bool ClipperLib::Clipper::Execute (
            ClipType clipType,
            PolyTree & polytree,
            PolyFillType fillType = pftEvenOdd )
```

6.4.2.4 Execute() [4/4]

```
bool ClipperLib::Clipper::Execute (
            ClipType clipType,
            PolyTree & polytree,
            PolyFillType subjFillType,
            PolyFillType clipFillType )
```

6.4.2.5 ExecuteInternal()

```
bool ClipperLib::Clipper::ExecuteInternal ( )  [protected], [virtual]
```

6.4.2.6 ReverseSolution() [1/2]

```
bool ClipperLib::Clipper::ReverseSolution ( )  [inline]
```

6.4.2.7 ReverseSolution() [2/2]

```
void ClipperLib::Clipper::ReverseSolution (
             bool value )  [inline]
```

6.4.2.8 StrictlySimple() [1/2]

```
bool ClipperLib::Clipper::StrictlySimple ( )  [inline]
```

6.4.2.9 StrictlySimple() [2/2]

```
void ClipperLib::Clipper::StrictlySimple (
             bool value )  [inline]
```

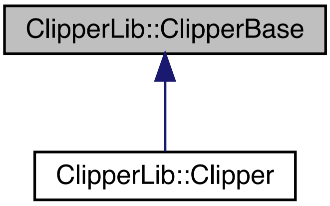
The documentation for this class was generated from the following files:

- src/include/clipper.hh
- src/clipper.cc

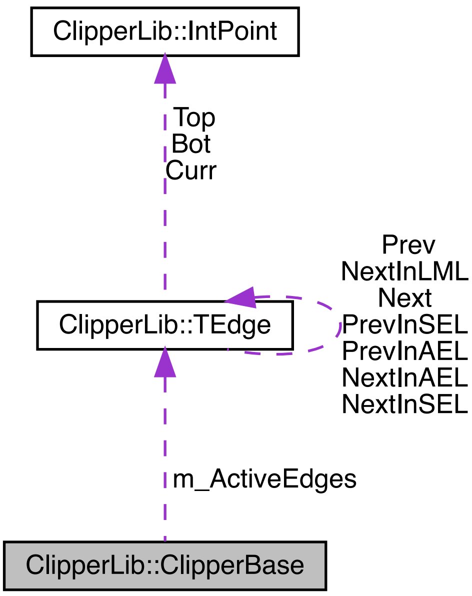
6.5 ClipperLib::ClipperBase Class Reference

```
#include <clipper.hh>
```

Inheritance diagram for ClipperLib::ClipperBase:



Collaboration diagram for ClipperLib::ClipperBase:



Public Member Functions

- ClipperBase ()
- virtual ~ClipperBase ()
- virtual bool AddPath (const Path &pg, PolyType PolyTyp, bool Closed)

- bool AddPaths (const Paths &ppg, PolyType PolyTyp, bool Closed)
- virtual void Clear ()
- IntRect GetBounds ()
- bool PreserveCollinear ()
- void PreserveCollinear (bool value)

Protected Types

- typedef std::vector< LocalMinimum > MinimaList
- typedef std::priority_queue< cInt > ScanbeamList

Protected Member Functions

- void DisposeLocalMinimaList ()
- TEdge * AddBoundsToLML (TEdge *e, bool IsClosed)
- virtual void Reset ()
- TEdge * ProcessBound (TEdge *E, bool IsClockwise)
- void InsertScanbeam (const cInt Y)
- bool PopScanbeam (cInt &Y)
- bool LocalMinimaPending ()
- bool PopLocalMinima (cInt Y, const LocalMinimum *&locMin)
- OutRec * CreateOutRec ()
- void DisposeAllOutRecs ()
- void DisposeOutRec (PolyOutList::size_type index)
- void SwapPositionsInAEL (TEdge *edge1, TEdge *edge2)
- void DeleteFromAEL (TEdge *e)
- void UpdateEdgeIntoAEL (TEdge *&e)

Protected Attributes

- MinimaList::iterator m_CurrentLM
- MinimaList m_MinimaList
- bool m_UseFullRange
- EdgeList m_edges
- bool m_PreserveCollinear
- bool m_HasOpenPaths
- PolyOutList m_PolyOuts
- TEdge * m_ActiveEdges
- ScanbeamList m_Scanbeam

6.5.1 Member Typedef Documentation

6.5.1.1 MinimaList

```
typedef std::vector<LocalMinimum> ClipperLib::ClipperBase::MinimaList  [protected]
```

6.5.1.2 ScanbeamList

```
typedef std::priority_queue<cInt> ClipperLib::ClipperBase::ScanbeamList  [protected]
```

6.5.2 Constructor & Destructor Documentation

6.5.2.1 ClipperBase()

```
ClipperLib::ClipperBase::ClipperBase ( )
```

6.5.2.2 ∼ClipperBase()

```
ClipperLib::ClipperBase::∼ClipperBase ( )  [virtual]
```

6.5.3 Member Function Documentation

6.5.3.1 AddBoundsToLML()

```
TEdge* ClipperLib::ClipperBase::AddBoundsToLML (
            TEdge * e,
            bool IsClosed )  [protected]
```

6.5.3.2 AddPath()

```
bool ClipperLib::ClipperBase::AddPath (
            const Path & pg,
            PolyType PolyTyp,
            bool Closed )  [virtual]
```

6.5.3.3 AddPaths()

```
bool ClipperLib::ClipperBase::AddPaths (
            const Paths & ppg,
            PolyType PolyTyp,
            bool Closed )
```

6.5.3.4 Clear()

```
void ClipperLib::ClipperBase::Clear ( ) [virtual]
```

6.5.3.5 CreateOutRec()

```
OutRec * ClipperLib::ClipperBase::CreateOutRec ( ) [protected]
```

6.5.3.6 DeleteFromAEL()

```
void ClipperLib::ClipperBase::DeleteFromAEL (
            TEdge * e ) [protected]
```

6.5.3.7 DisposeAllOutRecs()

```
void ClipperLib::ClipperBase::DisposeAllOutRecs ( ) [protected]
```

6.5.3.8 DisposeLocalMinimaList()

```
void ClipperLib::ClipperBase::DisposeLocalMinimaList ( ) [protected]
```

6.5.3.9 DisposeOutRec()

```
void ClipperLib::ClipperBase::DisposeOutRec (
            PolyOutList::size_type index ) [protected]
```

6.5.3.10 GetBounds()

```
IntRect ClipperLib::ClipperBase::GetBounds ( )
```

6.5.3.11 InsertScanbeam()

```
void ClipperLib::ClipperBase::InsertScanbeam (
            const cInt Y )  [protected]
```

6.5.3.12 LocalMinimaPending()

```
bool ClipperLib::ClipperBase::LocalMinimaPending ( )  [protected]
```

6.5.3.13 PopLocalMinima()

```
bool ClipperLib::ClipperBase::PopLocalMinima (
            cInt Y,
            const LocalMinimum *& locMin )  [protected]
```

6.5.3.14 PopScanbeam()

```
bool ClipperLib::ClipperBase::PopScanbeam (
            cInt & Y )  [protected]
```

6.5.3.15 PreserveCollinear() [1/2]

```
bool ClipperLib::ClipperBase::PreserveCollinear ( )  [inline]
```

6.5.3.16 PreserveCollinear() [2/2]

```
void ClipperLib::ClipperBase::PreserveCollinear (
            bool value )  [inline]
```

6.5.3.17 ProcessBound()

```
TEdge * ClipperLib::ClipperBase::ProcessBound (
            TEdge * E,
            bool IsClockwise )  [protected]
```

6.5.3.18 Reset()

```
void ClipperLib::ClipperBase::Reset ( ) [protected], [virtual]
```

6.5.3.19 SwapPositionsInAEL()

```
void ClipperLib::ClipperBase::SwapPositionsInAEL (
            TEdge * edge1,
            TEdge * edge2 ) [protected]
```

6.5.3.20 UpdateEdgeIntoAEL()

```
void ClipperLib::ClipperBase::UpdateEdgeIntoAEL (
            TEdge *& e ) [protected]
```

6.5.4 Member Data Documentation

6.5.4.1 m_ActiveEdges

```
TEdge* ClipperLib::ClipperBase::m_ActiveEdges [protected]
```

6.5.4.2 m_CurrentLM

```
MinimaList::iterator ClipperLib::ClipperBase::m_CurrentLM [protected]
```

6.5.4.3 m_edges

```
EdgeList ClipperLib::ClipperBase::m_edges [protected]
```

6.5.4.4 m_HasOpenPaths

```
bool ClipperLib::ClipperBase::m_HasOpenPaths [protected]
```

6.5.4.5 m_MinimaList

```
MinimaList ClipperLib::ClipperBase::m_MinimaList [protected]
```

6.5.4.6 m_PolyOuts

```
PolyOutList ClipperLib::ClipperBase::m_PolyOuts [protected]
```

6.5.4.7 m_PreserveCollinear

```
bool ClipperLib::ClipperBase::m_PreserveCollinear [protected]
```

6.5.4.8 m_Scanbeam

```
ScanbeamList ClipperLib::ClipperBase::m_Scanbeam [protected]
```

6.5.4.9 m_UseFullRange

```
bool ClipperLib::ClipperBase::m_UseFullRange [protected]
```

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

- src/include/clipper.hh
- src/clipper.cc

6.6 ClipperLib::clipperException Class Reference

```
#include <clipper.hh>
```

Inherits exception.

Public Member Functions

- clipperException (const char ∗description)
- virtual ∼clipperException () throw ()
- virtual const char ∗ what () const throw ()

6.6.1 Constructor & Destructor Documentation

6.6.1.1 clipperException()

```
ClipperLib::clipperException::clipperException (
            const char * description )  [inline]
```

6.6.1.2 ∼clipperException()

```
virtual ClipperLib::clipperException::∼clipperException ( ) throw ( )   [inline], [virtual]
```

6.6.2 Member Function Documentation

6.6.2.1 what()

```
virtual const char* ClipperLib::clipperException::what ( ) const throw ( )   [inline], [virtual]
```

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

- src/include/clipper.hh

6.7 ClipperLib::ClipperOffset Class Reference

```
#include <clipper.hh>
```

Public Member Functions

- ClipperOffset (double miterLimit=2.0, double roundPrecision=0.25)
- ∼ClipperOffset ()
- void AddPath (const Path &path, JoinType joinType, EndType endType)
- void AddPaths (const Paths &paths, JoinType joinType, EndType endType)
- void Execute (Paths &solution, double delta)
- void Execute (PolyTree &solution, double delta)
- void Clear ()

Public Attributes

- double MiterLimit
- double ArcTolerance

6.7.1 Constructor & Destructor Documentation

6.7.1.1 ClipperOffset()

```
ClipperLib::ClipperOffset::ClipperOffset (
            double miterLimit = 2.0,
            double roundPrecision = 0.25 )
```

6.7.1.2 ∼ClipperOffset()

```
ClipperLib::ClipperOffset::∼ClipperOffset ( )
```

6.7.2 Member Function Documentation

6.7.2.1 AddPath()

```
void ClipperLib::ClipperOffset::AddPath (
            const Path & path,
            JoinType joinType,
            EndType endType )
```

6.7.2.2 AddPaths()

```
void ClipperLib::ClipperOffset::AddPaths (
            const Paths & paths,
            JoinType joinType,
            EndType endType )
```

6.7.2.3 Clear()

```
void ClipperLib::ClipperOffset::Clear ( )
```

6.7.2.4 Execute() [1/2]

```
void ClipperLib::ClipperOffset::Execute (
            Paths & solution,
            double delta )
```

6.7.2.5 Execute() [2/2]

```
void ClipperLib::ClipperOffset::Execute (
            PolyTree & solution,
            double delta )
```

6.7.3 Member Data Documentation

6.7.3.1 ArcTolerance

```
double ClipperLib::ClipperOffset::ArcTolerance
```

6.7.3.2 MiterLimit

```
double ClipperLib::ClipperOffset::MiterLimit
```

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

- src/include/clipper.hh
- src/clipper.cc

6.8 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.
- template<class T2 >
  Configuration2 (const Point2< T2 > P, const Angle _th)

  Default constructor that takes the point, the angle, and stores them.
- Point2< T1 > point () const
- 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. \tparan 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. \tparan 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. \tparan T2 The type of the elements in the second `Configuration2`.
- stringstream to_string () const

  Function to create a stringstream containing the detail of the configuration.
- template<class T2 >
  operator Point2< T2 > () const

  Cast of Configuration to Point2.
- Configuration2< T1 > copy (const Configuration2< T1 > &A)

Copy a configuration into another one.
- Configuration2< T1 > operator= (const Configuration2< T1 > &A)

  Overload of the = operatore. Just calls `copy`.
- bool equal (const Configuration2< T1 > &A)

  Equalize two configurations.
- bool operator== (const Configuration2< T1 > &A)

  Overload of the == operator. Just calls `equal`.
- bool operator!= (const Configuration2< T1 > &A)

  Overload of the != operator. Just calls `equal` *and negates it.*
- operator Point2< T1 > ()

  Cast a `Configuration2` *to a* `Point2` *of the same type.*
- template<class T2 >
  operator Configuration2< T2 > () const

  Cast a `Configuration2` *to a* `Configuration2` *of a different type.*
- void invert ()

  Invert the x and y of the point, and even the angle of the configuration.

Friends

- ostream & operator<< (ostream &out, const Configuration2< T1 > &data)

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

6.8.1 Detailed Description

template<class T1>
class Configuration2< T1 >

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

Template Parameters

T1	The type of the coordinates.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Configuration2() [1/4]

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

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

6.8.2.2 Configuration2() [2/4]

```
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.

6.8.2.3 Configuration2() [3/4]

```
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	↩ _↩ th	The angle.

6.8.2.4 Configuration2() [4/4]

```
template<class T1>
template<class T2 >
Configuration2< T1 >::Configuration2 (
            const Point2< T2 > P,
            const Angle _th )  [inline]
```

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

Template Parameters

Type	of point in input.

Parameters

in	*P*	The coordinates.
in	↩ _↩ *th*	The angle.

6.8.3 Member Function Documentation

6.8.3.1 angle() [1/2]

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

Returns

The angle.

6.8.3.2 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	↩ _↩ *th*	The value to be stored.

Returns

1 if everything went ok, 0 otherwise.

6.8.3.3 copy()

```
template<class T1>
Configuration2<T1> Configuration2< T1 >::copy (
            const Configuration2< T1 > & A ) [inline]
```

Copy a configuration into another one.

Parameters

in	*A*	Configuration to be coppied.

Returns

this.

6.8.3.4 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. \tparan T2 The type of the elements in the second `Configuration2`.

Parameters

in	*B*	The second `Configuration2` to use for computing the distance.
in	*dist*	The type of distance to be computed.

Returns

The distance between the two configurations.

6.8.3.5 equal()

```
template<class T1>
bool Configuration2< T1 >::equal (
            const Configuration2< T1 > & A ) [inline]
```

Equalize two configurations.

Parameters

in	*A*	Configuration to be equalized.

Returns

true if the two configurations are equal.

6.8.3.6 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. \tparan T2 The type of the elements in the second `Configuration2`.

Parameters

in	*B*	the second `Configuration2` to use for computing the distance.

Returns

The Euclidean distance between the two configurations.

6.8.3.7 invert()

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

Invert the x and y of the point, and even the angle of the configuration.

6.8.3.8 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. \tparan T2 The type of the elements in the second `Configuration2`.

Parameters

in	*B*	the second `Configuration2` to use for computing the distance.

Returns

The Manhattan distance between the two configurations.

6.8.3.9 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

6.8.3.10 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	*p*	The configuration containing the offsets.

Returns

1 if everything went fine, 0 otherwise.

6.8.3.11 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	*_th*	The offset for the `Angle` 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.

6.8.3.12 offset_angle()

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

Function to add an offset to the angle.

Parameters

in	*_offset*	The offset.

Returns

1 if everything went fine, 0 otherwise.

6.8.3.13 offset_x()

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

Function to add an offset to the abscissa.

Parameters

in	*_offset*	The offset.

Returns

1 if everything went fine, 0 otherwise.

6.8.3.14 offset_y()

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

Function to add an offset to the ordinate.

Parameters

in	*_offset*	The offset.

Returns

1 if everything went fine, 0 otherwise.

6.8.3.15 operator Configuration2< T2 >()

```
template<class T1>
template<class T2 >
Configuration2< T1 >::operator Configuration2< T2 > ( ) const  [inline]
```

Cast a `Configuration2` to a `Configuration2` of a different type.

Template Parameters

T2	The type of the `Configuration2` to be casted to.

6.8.3.16 operator Point2< T1 >()

```
template<class T1>
Configuration2< T1 >::operator Point2< T1 > ( )  [inline]
```

Cast a `Configuration2` to a `Point2` of the same type.

6.8.3.17 operator Point2< T2 >()

```
template<class T1>
template<class T2 >
Configuration2< T1 >::operator Point2< T2 > ( ) const  [inline]
```

Cast of Configuration to Point2.

Template Parameters

T2	Type of Point2 to be casted to.

Returns

A Point2 of type T2.

6.8.3.18 operator"!=()

```
template<class T1>
bool Configuration2< T1 >::operator!= (
              const Configuration2< T1 > & A )  [inline]
```

Overload of the != operator. Just calls `equal` and negates it.

Parameters

`in`	*A*	Configuration to be equalized.

Returns

true if the two configurations are different, false otherwise.

6.8.3.19 operator=()

```
template<class T1>
Configuration2<T1> Configuration2< T1 >::operator= (
              const Configuration2< T1 > & A )  [inline]
```

Overload of the = operatore. Just calls `copy`.

Parameters

`in`	*A*	Configuration to be coppied.

Returns

this.

6.8.3.20 operator==()

```
template<class T1>
bool Configuration2< T1 >::operator== (
            const Configuration2< T1 > & A )  [inline]
```

Overload of the == operator. Just calls `equal`.

Parameters

in	*A*	Configuration to be equalized.

Returns

true if the two configurations are equal.

6.8.3.21 point()

```
template<class T1>
Point2<T1> Configuration2< T1 >::point ( ) const  [inline]
```

Returns

A `Point2` variable containing the coordinates.

6.8.3.22 to_string()

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

Function to create a stringstream containing the detail of the configuration.

Returns

A stringstream.

6.8.3.23 x() [1/2]

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

Returns

The abscissa coordinate.

6.8.3.24 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	_x	The value to be stored.

Returns

1 if everything went ok, 0 otherwise.

6.8.3.25 y() [1/2]

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

Returns

The ordinate coordinate.

6.8.3.26 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	↩ _↩ y	The value to be stored.

Returns

1 if everything went ok, 0 otherwise.

6.8.4 Friends And Related Function Documentation

6.8.4.1 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 `Configuration2` to print.

Returns

An output stream to be printed.

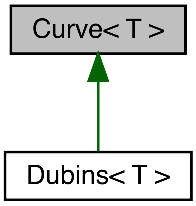
The documentation for this class was generated from the following file:

- src/include/maths.hh

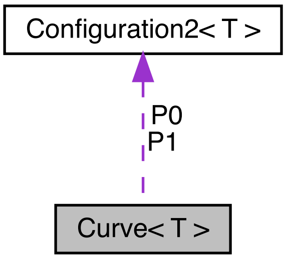
6.9 Curve< T > Class Template Reference

`#include <dubins.hh>`

Inheritance diagram for Curve< T >:



Collaboration diagram for Curve< T >:



Public Member Functions

- Curve ()

    Default plain constructor which creates two plain `Configuration2`s.
- Curve (const Configuration2< T > _P0, const Configuration2< T > _P1)
- Curve (const Point2< T > _P0, const Point2< T > _P1, const Angle _th0, const Angle _th1)
- Curve (const T x0, const T y0, const Angle _th0, const T x1, const T y1, const Angle _th1)
- Configuration2< T > begin () const

    Returns the starting `Configuration2` of the `Curve`.
- Configuration2< T > end () const

    Returns the ending `Configuration2` of the `Curve`.
- void begin (Configuration2< T > _P0)
- void end (Configuration2< T > _P1)
- stringstream to_string () const

Protected Attributes

- Configuration2< T > P0

    Start `Configuration2`.
- Configuration2< T > P1

    End `Configuration2`.

Friends

- ostream & operator<< (ostream &out, const Curve &data)

6.9.1 Detailed Description

template<class T>
class Curve< T >

Class that defines a general curve. It just containes a start `Configuration2` and an end `Configuration2`.

Template Parameters

T	The type of the `Configuration2`s

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Curve() [1/4]

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

Default plain constructor which creates two plain `Configuration2`s.

6.9.2.2 Curve() [2/4]

```
template<class T>
Curve< T >::Curve (
            const Configuration2< T > _P0,
            const Configuration2< T > _P1 ) [inline]
```

Constructor that takes two `Configuration2`s and stores them.

Parameters

in	*_P0*	Start `Configuration2`.
in	*_P1*	End `Configuration2`.

6.9.2.3 Curve() [3/4]

```
template<class T>
Curve< T >::Curve (
            const Point2< T > _P0,
            const Point2< T > _P1,
            const Angle _th0,
            const Angle _th1 )  [inline]
```

Constructor that takes two `Point2`s and two `Angle`s and stores them as `Configuration2`s.

Parameters

in	*_P0*	Start `Point2`.
in	*_P1*	End `Point2`.
in	*_th0*	Starting `Angle`
in	*_th1*	Ending `Angle`

6.9.2.4 Curve() [4/4]

```
template<class T>
Curve< T >::Curve (
            const T x0,
            const T y0,
            const Angle _th0,
            const T x1,
            const T y1,
            const Angle _th1 )  [inline]
```

Constructor that takes the bare coordinates of two points and their `Angle`s and stores them as `Configuration2`s.

Parameters

in	*x0*	Start abscissa coordinate.
in	*y0*	Start ordinate coordinate.
in	*_th0*	Start `Angle`.
in	*x1*	End abscissa coordinate.
in	*y1*	End ordinate coordinate.
in	*_th1*	End `Angle`.

6.9.3 Member Function Documentation

6.9.3.1 begin() [1/2]

```
template<class T>
Configuration2<T> Curve< T >::begin ( ) const  [inline]
```

Returns the starting Configuration2 of the Curve.

6.9.3.2 begin() [2/2]

```
template<class T>
void Curve< T >::begin (
            Configuration2< T > _P0 )  [inline]
```

Function that stores the starting Configuration2.

Parameters

in	*_P0*	Starting Configuration2.

6.9.3.3 end() [1/2]

```
template<class T>
Configuration2<T> Curve< T >::end ( ) const  [inline]
```

Returns the ending Configuration2 of the Curve.

6.9.3.4 end() [2/2]

```
template<class T>
void Curve< T >::end (
            Configuration2< T > _P1 )  [inline]
```

Function that stores the ending Configuration2.

Parameters

in	*_P0*	Ending Configuration2.

6.9.3.5 to_string()

```
template<class T>
stringstream Curve< T >::to_string ( ) const  [inline]
```

This function create a strinstream object containing infos about the Curve.

Returns

A string stream.

6.9.4 Friends And Related Function Documentation

6.9.4.1 operator<<

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

This function overload the << operator so to print with `std::cout` the values of the `Curve`.

Parameters

in	*out*	The out stream.
in	*data*	The `Curve` to print.

Returns

An output stream to be printed.

6.9.5 Member Data Documentation

6.9.5.1 P0

```
template<class T>
Configuration2<T> Curve< T >::P0  [protected]
```

Start `Configuration2`.

6.9.5.2 P1

```
template<class T>
Configuration2<T> Curve< T >::P1  [protected]
```

End `Configuration2`.

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

- src/include/dubins.hh

6.10 ClipperLib::DoublePoint Struct Reference

```
#include <clipper.hh>
```

Public Member Functions

- DoublePoint (double x=0, double y=0)
- DoublePoint (IntPoint ip)

Public Attributes

- double X
- double Y

6.10.1 Constructor & Destructor Documentation

6.10.1.1 DoublePoint() [1/2]

```
ClipperLib::DoublePoint::DoublePoint (
            double x = 0,
            double y = 0 )  [inline]
```

6.10.1.2 DoublePoint() [2/2]

```
ClipperLib::DoublePoint::DoublePoint (
            IntPoint ip )  [inline]
```

6.10.2 Member Data Documentation

6.10.2.1 X

```
double ClipperLib::DoublePoint::X
```

6.10.2.2 Y

```
double ClipperLib::DoublePoint::Y
```

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

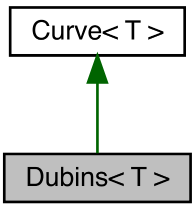
- src/include/clipper.hh

6.11 Dubins< T > Class Template Reference

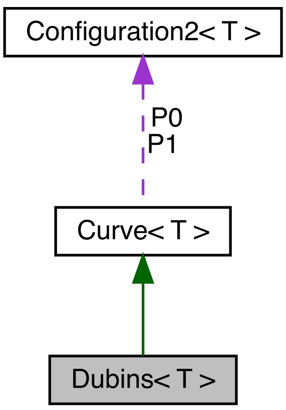
Class to store a Dubins curve. This class inherits from `Curve` and is composed of three `DubinsArc`.

```
#include <dubins.hh>
```

Inheritance diagram for Dubins< T >:



Collaboration diagram for Dubins< T >:



Public Member Functions

- Dubins ()
- Dubins (const Configuration2< T > _P0, const Configuration2< T > _P1, const double _K=KMAX)
- Dubins (const Point2< T > _P0, const Point2< T > _P1, const Angle _th0, const Angle _th1, const double _K=KMAX)
- Dubins (const T x0, const T y0, const Angle _th0, const T x1, const T y1, const Angle _th1, const double _K=KMAX)
- double getKmax () const

  Returns the maximum curvature of the Dubins.
- double length () const

  Returns the length of the Dubins.
- double getId ()

  Returns the id of the Dubins, that is the set of three maneuvers that creates the curve.
- Configuration2< T > begin () const
- Configuration2< T > end () const
- DubinsArc< T > getA1 () const

  Returns the first `DubinsArc`.
- DubinsArc< T > getA2 () const

  Returns the second `DubinsArc`.
- DubinsArc< T > getA3 () const

  Returns the third `DubinsArc`.
- double ∗ LSL (double th0, double th1, double _kmax)
- double ∗ RSR (double th0, double th1, double _kmax)
- double ∗ LSR (double th0, double th1, double _kmax)
- double ∗ RSL (double th0, double th1, double _kmax)
- double ∗ RLR (double th0, double th1, double _kmax)
- double ∗ LRL (double th0, double th1, double _kmax)
- Tuple< double > scaleToStandard ()

  Function to compute standardize the parameters. This function computes the initial and final angles as if the reference system is P0(-1,0), P1(0,1). This allows to simplify the calculations to find the best set of maneuvers.
- Tuple< double > scaleFromStandard (double lambda, double sc_s1, double sc_s2, double sc_s3)
- int shortest_path ()

  This function computes the shortest path for the Dubins constructed. First the values are scaled. Then the six sets of maneuvers are computed and their lengths are stored. Once the set that gives the Dubins with the minimum length is found, the lengths are rescaled and the `DubinsArc` are created. In the process length is also computed.
- bool check (double s1, double k0, double s2, double k1, double s3, double k2, Angle th0, Angle th1) const
- Tuple< Tuple< Configuration2< double > > > splitIt (double _L=PIECE_LENGTH, int _arch=0)
- stringstream to_string () const
- void draw (double dimX, double dimY, double inc, Scalar scl, Mat &image, double SHIFT=0)
- bool is_on_dubins (Configuration2< T > C)

Static Public Member Functions

- static double rangeSymm (double ang)

Friends

- ostream & operator<< (ostream &out, const Dubins &data)

Additional Inherited Members

6.11.1 Detailed Description

template<class T>
class Dubins< T >

Class to store a Dubins curve. This class inherits from `Curve` and is composed of three `DubinsArc`.

Template Parameters

T	The type of the classes `Curve` and `DubinsArc`.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Dubins() [1/4]

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

Plain constructor for Dubins that calls the plain constructor of `Curve` and `DubinsArc`.

6.11.2.2 Dubins() [2/4]

```
template<class T>
Dubins< T >::Dubins (
            const Configuration2< T > _P0,
            const Configuration2< T > _P1,
            const double _K = KMAX )  [inline]
```

Constructor that takes an initial and a final `Configuration2`, a curvature and compute the Dubins that connect the two configurations.

Parameters

in	*_P0*	Initial `Configuration2`.
in	*_P1*	Final `Configuration2`.
in	*_K*	Curvature.

6.11.2.3 Dubins() [3/4]

```
template<class T>
Dubins< T >::Dubins (
```

```
const Point2< T > _P0,
const Point2< T > _P1,
const Angle _th0,
const Angle _th1,
const double _K = KMAX )   [inline]
```

Constructor that takes an initial and a final `Point2`, the two respectively `Angle`s and the curvature and computes the Dubins.

Parameters

in	*_P0*	Initial `Point2`.
in	*_P1*	Final `Point2`.
in	*_th0*	Initial `Angle`
in	*_th1*	Final `Angle`
in	*_K*	Curvature.

6.11.2.4 Dubins() [4/4]

```
template<class T>
Dubins< T >::Dubins (
    const T x0,
    const T y0,
    const Angle _th0,
    const T x1,
    const T y1,
    const Angle _th1,
    const double _K = KMAX )   [inline]
```

Constructor that takes the initial and final coordinates, the respective `Angle`s and the curvature and compute a Dubins.

Parameters

in	*x0*	Initial abscissa coordinate.
in	*y0*	Initial ordinate coordinate.
in	*_th0*	Initial `Angle`.
in	*x1*	Final abscissa coordinate.
in	*y1*	Final ordinate coordinate.
in	*_th1*	Final `Angle`.
in	*_K*	Curvature of the curve.

6.11.3 Member Function Documentation

6.11.3.1 begin()

```
template<class T>
Configuration2<T> Dubins< T >::begin ( ) const [inline]
```

6.11.3.2 check()

```
template<class T>
bool Dubins< T >::check (
            double s1,
            double k0,
            double s2,
            double k1,
            double s3,
            double k2,
            Angle th0,
            Angle th1 ) const [inline]
```

Function that checks that the values got in `shortest_path()` are right.

Parameters

in	*s1*	Length for the first `DubinsArc`.
in	*k0*	Curvature for the first `DubinsArc`.
in	*s2*	Length for the second `DubinsArc`.
in	*k1*	Curvature for the second `DubinsArc`.
in	*s3*	Length for the third `DubinsArc`.
in	*k2*	Curvature for the third `DubinsArc`.
in	*th0*	Initial angles (standardised).
in	*th1*	Final angles (standardised).

Returns

`true` if the values where correct, `false` otherwise.

6.11.3.3 draw()

```
template<class T>
void Dubins< T >::draw (
            double dimX,
            double dimY,
            double inc,
            Scalar scl,
            Mat & image,
            double SHIFT = 0 ) [inline]
```

Function to draw the `Dubins`.

Parameters

in	*dimX*	The dimension X of the Mat.
in	*dimY*	The dimension Y of the Mat.
in	*inc*	The value to scale each point.
in	*scl*	The Scalar that defines the color to use.
in	*image*	The Mat where to draw the points.
in	*SHIFT*	The value to use to shift the points to make them stay inside the matrix.

6.11.3.4 end()

```
template<class T>
Configuration2<T> Dubins< T >::end ( ) const [inline]
```

6.11.3.5 getA1()

```
template<class T>
DubinsArc<T> Dubins< T >::getA1 ( ) const [inline]
```

Returns the first `DubinsArc`.

6.11.3.6 getA2()

```
template<class T>
DubinsArc<T> Dubins< T >::getA2 ( ) const [inline]
```

Returns the second `DubinsArc`.

6.11.3.7 getA3()

```
template<class T>
DubinsArc<T> Dubins< T >::getA3 ( ) const [inline]
```

Returns the third `DubinsArc`.

6.11.3.8 getId()

```
template<class T>
double Dubins< T >::getId ( )  [inline]
```

Returns the id of the Dubins, that is the set of three maneuvers that creates the curve.

6.11.3.9 getKmax()

```
template<class T>
double Dubins< T >::getKmax ( ) const  [inline]
```

Returns the maximum curvature of the Dubins.

6.11.3.10 is_on_dubins()

```
template<class T>
bool Dubins< T >::is_on_dubins (
             Configuration2< T > C )  [inline]
```

Function to check if a `Configuration2` is on a `Dubins`.

Returns

`true` if the `Configuration2` is on the `Dubins`, `false` otherwise.

6.11.3.11 length()

```
template<class T>
double Dubins< T >::length ( ) const  [inline]
```

Returns the length of the Dubins.

6.11.3.12 LRL()

```
template<class T>
double* Dubins< T >::LRL (
             double th0,
             double th1,
             double _kmax )  [inline]
```

Function to compute the set of maneuvers Left Right Left.

Parameters

in	*th0*	The initial angle standardized.
in	*th1*	The final angle standardized.
in	*_kmax*	The maximum curvature.

Returns

An array of dimension 3 containing the length of the 3 maneuvers.

6.11.3.13 LSL()

```
template<class T>
double* Dubins< T >::LSL (
            double th0,
            double th1,
            double _kmax )  [inline]
```

Function to compute the set of maneuvers Left Straight Left.

Parameters

in	*th0*	The initial angle standardized.
in	*th1*	The final angle standardized.
in	*_kmax*	The maximum curvature.

Returns

An array of dimension 3 containing the length of the 3 maneuvers.

6.11.3.14 LSR()

```
template<class T>
double* Dubins< T >::LSR (
            double th0,
            double th1,
            double _kmax )  [inline]
```

Function to compute the set of maneuvers Left Straight Right.

Parameters

in	*th0*	The initial angle standardized.
in	*th1*	The final angle standardized.
in	*_kmax*	The maximum curvature.

Returns

An array of dimension 3 containing the length of the 3 maneuvers.

6.11.3.15 rangeSymm()

```
template<class T>
static double Dubins< T >::rangeSymm (
              double ang )  [inline], [static]
```

Normalize an angular difference $(-\pi, \pi]$.

Parameters

in	*ang*	The value of the angle to be normalized.

Returns

The normalized angle.

6.11.3.16 RLR()

```
template<class T>
double* Dubins< T >::RLR (
              double th0,
              double th1,
              double _kmax )  [inline]
```

Function to compute the set of maneuvers Right Left Right.

Parameters

in	*th0*	The initial angle standardized.
in	*th1*	The final angle standardized.
in	*_kmax*	The maximum curvature.

Returns

An array of dimension 3 containing the length of the 3 maneuvers.

6.11.3.17 RSL()

```
template<class T>
double* Dubins< T >::RSL (
```

```
            double th0,
            double th1,
            double _kmax )  [inline]
```

Function to compute the set of maneuvers Right Straight Left.

Parameters

in	*th0*	The initial angle standardized.
in	*th1*	The final angle standardized.
in	*_kmax*	The maximum curvature.

Returns

An array of dimension 3 containing the length of the 3 maneuvers.

6.11.3.18 RSR()

```
template<class T>
double* Dubins< T >::RSR (
            double th0,
            double th1,
            double _kmax )  [inline]
```

Function to compute the set of maneuvers Right Straight Right.

Parameters

in	*th0*	The initial angle standardized.
in	*th1*	The final angle standardized.
in	*_kmax*	The maximum curvature.

Returns

An array of dimension 3 containing the length of the 3 maneuvers.

6.11.3.19 scaleFromStandard()

```
template<class T>
Tuple<double> Dubins< T >::scaleFromStandard (
            double lambda,
            double sc_s1,
            double sc_s2,
            double sc_s3 )  [inline]
```

Function that scales from a given system to another through a parameter lambda.

Parameters

in	*lambda*	Coefficient to be applied to restore original system.
in	*sc_s1*	Angle or value to be scaled.
in	*sc_s1*	Angle or value to be scaled.
in	*sc_s1*	Angle or value to be scaled.

Returns

a `Tuple` containing the value scaled.

6.11.3.20 scaleToStandard()

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

Function to compute standardize the parameters. This function computes the initial and final angles as if the reference system is P0(-1,0), P1(0,1). This allows to simplify the calculations to find the best set of maneuvers.

Returns

A `Tuple` of `duoble` containing the standardised initial and final angle, the new curvature and the parameter lambda that allows to compute the real dimension lengths.

6.11.3.21 shortest_path()

```
template<class T>
int Dubins< T >::shortest_path ( )  [inline]
```

This function computes the shortest path for the Dubins constructed. First the values are scaled. Then the six sets of maneuvers are computed and their lengths are stored. Once the set that gives the Dubins with the minimum length is found, the lengths are rescaled and the `DubinsArc` are created. In the process length is also computed.

Returns

The id of the set of maneuvers.

6.11.3.22 splitIt()

```
template<class T>
Tuple<Tuple<Configuration2<double> > > Dubins< T >::splitIt (
              double _L = PIECE_LENGTH,
              int _arch = 0 )  [inline]
```

Function to split a Dubins in points.

Parameters

in	*_arch*	If defined returns only the points for a single DubinsArc.
in	*_L*	The distance from one point to another.

Returns

A Tuple containing three Tuple of Point2 (one for each arc) containing the computed points.

6.11.3.23 to_string()

```
template<class T>
stringstream Dubins< T >::to_string ( ) const [inline]
```

This function create a strinstream object containing infos about the Dubins.

Returns

A string stream.

6.11.4 Friends And Related Function Documentation

6.11.4.1 operator<<

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

This function overload the << operator so to print with std::cout the values of the Dubins, that is printing the 3 DubinsArcs.

Parameters

in	*out*	The out stream.
in	*data*	The Dubins to print.

Returns

An output stream to be printed.

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

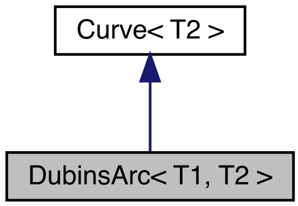
- src/include/dubins.hh

6.12 DubinsArc< T1, T2 > Class Template Reference

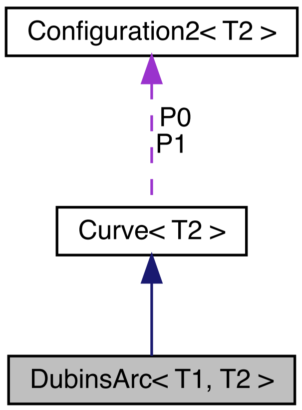
Class to store a maneuver of Dubins. It inherits from `Curve`. Since each Dubins is formed of atmost 3 maneuvers, this class is meant to store one of this maneuver, which can be L, R or S respectively Left, Right, Straight.

`#include <dubins.hh>`

Inheritance diagram for DubinsArc< T1, T2 >:



Collaboration diagram for DubinsArc< T1, T2 >:



Public Member Functions

- DubinsArc ()
- DubinsArc (const Configuration2< T2 > _P0, const T1 _k, const T1 _l)
- T1 getK () const

  Returns the curvature of the arc.
- T1 length () const

Returns the length of the arc.

- Tuple< Configuration2< T2 > > splitIt (double _L=PIECE_LENGTH)

  Splits the `DubinsArc` in pieces of _L length. This function starts from the begining of the arc and computes n new arcs through the `circline()` function using the curvature of the `DubinsArc` and _L as the length.
- stringstream to_string () const
- void draw (double dimX, double dimY, double inc, Scalar scl, Mat &image, double SHIFT=D_SHIFT)
- bool is_on_dubinsArc (Configuration2< T2 > C)

Friends

- ostream & operator<< (ostream &out, const DubinsArc &data)

Additional Inherited Members

6.12.1 Detailed Description

template<class T1 = double, class T2 = double>
class DubinsArc< T1, T2 >

Class to store a maneuver of Dubins. It inherits from `Curve`. Since each Dubins is formed of atmost 3 maneuvers, this class is meant to store one of this maneuver, which can be L, R or S respectively Left, Right, Straight.

Template Parameters

T1	The type of Length and Curvature.
T2	The type of the class `Curve`.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 DubinsArc() [1/2]

```
template<class T1 = double, class T2 = double>
DubinsArc< T1, T2 >::DubinsArc ( )  [inline]
```

Plain constructor of `DubinsArc` that sets L and K to 0 and creates a plain `Curve`.

6.12.2.2 DubinsArc() [2/2]

```
template<class T1 = double, class T2 = double>
DubinsArc< T1, T2 >::DubinsArc (
            const Configuration2< T2 > _P0,
            const T1 _k,
            const T1 _l )  [inline]
```

Creates a new `DubinsArc` given a start `Configuration2`, the curvature and the length of the arc calling `circline()`.

Parameters

in	*_P0*	The starting Configuration2.
in	*_k*	The curvature of the DubinsArc.
in	*_l*	The length of the DubinsArc.

6.12.3 Member Function Documentation

6.12.3.1 draw()

```
template<class T1 = double, class T2 = double>
void DubinsArc< T1, T2 >::draw (
            double dimX,
            double dimY,
            double inc,
            Scalar scl,
            Mat & image,
            double SHIFT = D_SHIFT )  [inline]
```

This function draws the DubinsArc.

Parameters

in	*dimX*	The dimension X of the Mat.
in	*dimY*	The dimension Y of the Mat.
in	*inc*	The value to scale each point.
in	*scl*	The Scalar that defines the color to use.
in	*image*	The Mat where to draw the points.
in	*SHIFT*	The value to use to shift the points to make them stay inside the matrix.

6.12.3.2 getK()

```
template<class T1 = double, class T2 = double>
T1 DubinsArc< T1, T2 >::getK ( ) const  [inline]
```

Returns the curvature of the arc.

6.12.3.3 is_on_dubinsArc()

```
template<class T1 = double, class T2 = double>
bool DubinsArc< T1, T2 >::is_on_dubinsArc (
            Configuration2< T2 > C )  [inline]
```

Function that given a `Configuration2` says if the point is on the `DubinsArc`, or not.

If the arc has 0 curvature, than the arc is a line and so the configuration needs to have the same direction as the start configuration, have the same angle from start than the angle from start to end and be inside the extremities of the segment. If the curvature is not 0, then we are on a circle and the function `is_on_circarc` is called.

Parameters

in	*C*	The `Configuration2` to be checked.

Returns

`true` if the configuration is on the arc, `false` otherwise.

6.12.3.4 length()

```
template<class T1 = double, class T2 = double>
T1 DubinsArc< T1, T2 >::length ( ) const  [inline]
```

Returns the length of the arc.

6.12.3.5 splitIt()

```
template<class T1 = double, class T2 = double>
Tuple<Configuration2<T2> > DubinsArc< T1, T2 >::splitIt (
            double _L = PIECE_LENGTH )  [inline]
```

Splits the `DubinsArc` in pieces of _L length. This function starts from the begining of the arc and computes n new arcs through the `circline()` function using the curvature of the `DubinsArc` and _L as the length.

Parameters

in	↩ _↩ *L*	The length that each points should have.

Returns

A `Tuple` of `Configuration2`s representing the points along the arc.

6.12.3.6 to_string()

```
template<class T1 = double, class T2 = double>
stringstream DubinsArc< T1, T2 >::to_string ( ) const  [inline]
```

This function create a strinstream object containing infos about the `DubinsArc`.

Returns

A string stream.

6.12.4 Friends And Related Function Documentation

6.12.4.1 operator<<

```
template<class T1 = double, class T2 = double>
ostream& operator<< (
            ostream & out,
            const DubinsArc< T1, T2 > & data )  [friend]
```

This function overload the << operator so to print with `std::cout` the values of the `DubinsArc`, that is `Curve` values more the length and the curvature.

Parameters

in	*out*	The out stream.
in	*data*	The `DubinsArc` to print.

Returns

An output stream to be printed.

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

- src/include/dubins.hh

6.13 DubinsSet< T > Class Template Reference

Given a set of point, compute the shortest set of Dubins that allows to go from start to end through all points.

`#include <dubins.hh>`

Public Member Functions

- DubinsSet ()
- DubinsSet (Tuple< Dubins< T > > _dubinses, double _kmax=KMAX)
- DubinsSet (Tuple< Configuration2< T > > _confs, double _kmax=KMAX)
- DubinsSet (Configuration2< T > start, Configuration2< T > end, Tuple< Point2< T > > _points, double _kmax=KMAX)

  Constructor that given a start `Configuration2`, an end `Configuration2` and a `Tuple` of `Point2`, computes the best path from start to end through all points by brute forcing all possible angles. Since this approach is based on a brute force algorithm, it's best not to use this on too many points.
- DubinsSet (Tuple< Point2< T > > _points, double _kmax=KMAX)

Constructor that computes a series of `Dubins` given only `Point2` points via brute force. Since this approach is based on a brute force algorithm, it's best not to use this on too many points.
- void find_best (Tuple< Point2< T > > _points, Tuple< Angle > &_angles, Angle area=A_2PI, double tries=4.0, double _kmax=KMAX)
- double getLength ()

  Returns the Length of the set of `Dubins`.
- double getKmax ()

  Returns the maximum curvature.
- double getSize ()

  Returns the number of `Dubins` stored.
- Tuple< Dubins< T > > getDubinses ()

  Returns a `Tuple` containing all the `Dubins`.
- Configuration2< T > getBegin ()

  Returns the starting `Configuration2` of the `DubinsSet`.
- Configuration2< T > getEnd ()

  Returns the ending `Configuration2` of the `DubinsSet`
- Dubins< T > getDubins (int id)
- Tuple< Dubins< T > > getDubinsFrom (int id)
- Dubins< T > * getDubinsPtr (int id)
- void clean ()
- int is_on_dubinsSet (Configuration2< T > C)

  Function that checks whether a `Configuration2` is on the `DubinsSet` or not.
- bool addDubins (Dubins< T > *D)

  Function to add a `Dubins` at the end of the `DubinsSet`.
- void removeDubins ()

  Remove the last `Dubins` from the set.
- DubinsSet< T > copy (DubinsSet< T > *DS)
- DubinsSet< T > operator= (DubinsSet< T > *DS)
- DubinsSet< T > join (DubinsSet< T > *DS, int startPos=-1)

  Function to join two DubinsSet.
- Tuple< Tuple< Tuple< Configuration2< T > > > > splitIt (double _length=PIECE_LENGTH)
- stringstream to_string ()

Friends

- ostream & operator<< (ostream &out, DubinsSet &data)

6.13.1 Detailed Description

template<class T>
class DubinsSet< T >

Given a set of point, compute the shortest set of Dubins that allows to go from start to end through all points.

Template Parameters

T	Type for class `Dubins`.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 DubinsSet() [1/5]

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

Plain constructor for `DubinsSet`.

6.13.2.2 DubinsSet() [2/5]

```
template<class T>
DubinsSet< T >::DubinsSet (
            Tuple< Dubins< T > > _dubinses,
            double _kmax = KMAX ) [inline]
```

Constructor that given a `Tuple` of `Dubins` computes stores all of them.

Parameters

in	*_dubinses*	The `Tuple` of `Dubins`.
in	*_kmax*	The maximum curvature.

6.13.2.3 DubinsSet() [3/5]

```
template<class T>
DubinsSet< T >::DubinsSet (
            Tuple< Configuration2< T > > _confs,
            double _kmax = KMAX ) [inline]
```

Constructor that takes a `Tuple` of `Configuration2`s and computes the `Dubins` between them.

Parameters

in	*_confs*	The `Tuple` of `Configuration2`s.
in	*_kmax*	The maximum curvature to be used.

6.13.2.4 DubinsSet() [4/5]

```
template<class T>
DubinsSet< T >::DubinsSet (
```

```
            Configuration2< T > start,
            Configuration2< T > end,
            Tuple< Point2< T > > _points,
            double _kmax = KMAX )  [inline]
```

Constructor that given a start `Configuration2`, an end `Configuration2` and a `Tuple` of `Point2`, computes the best path from start to end through all points by brute forcing all possible angles. Since this approach is based on a brute force algorithm, it's best not to use this on too many points.

Parameters

in	*start*	`Configuration2` of start.
in	*end*	`Configuration2` of end.
in	*_points*	`Tuple` of `Point2` containing all the intermediate points.
in	*_kmax*	The maximum curvature of the system.

6.13.2.5 DubinsSet() [5/5]

```
template<class T>
DubinsSet< T >::DubinsSet (
            Tuple< Point2< T > > _points,
            double _kmax = KMAX )  [inline]
```

Constructor that computes a series of `Dubins` given only `Point2` points via brute force. Since this approach is based on a brute force algorithm, it's best not to use this on too many points.

Parameters

in	*_points*	A `Tuple` containing all points.
in	*_kmax*	The maximum curvature to be used for all `Dubins`.

6.13.3 Member Function Documentation

6.13.3.1 addDubins()

```
template<class T>
bool DubinsSet< T >::addDubins (
            Dubins< T > * D )  [inline]
```

Function to add a `Dubins` at the end of the `DubinsSet`.

The `Dubins` to be added must respect some conditions such as the same curvature as the `DubinsSet`, the initial `Configuration2` must be on the path of the `DubinsSet`.

Parameters

D	The Dubins to add.

Returns

`true` if the Dubins could be added, `false` otherwise.

6.13.3.2 clean()

```
template<class T>
void DubinsSet< T >::clean ( ) [inline]
```

Function to remove all Dubins, set curvature to 0 and L to ∞

6.13.3.3 copy()

```
template<class T>
DubinsSet<T> DubinsSet< T >::copy (
            DubinsSet< T > * DS ) [inline]
```

Function to copy a DubinsSet on to `this`.

Parameters

DS	The DubinsSet to be copied on `this`

Returns

`this`, that is DS.

6.13.3.4 find_best()

```
template<class T>
void DubinsSet< T >::find_best (
            Tuple< Point2< T > > _points,
            Tuple< Angle > & _angles,
            Angle area = A_2PI,
            double tries = 4.0,
            double _kmax = KMAX ) [inline]
```

Function to compute the best path. This function calls `disp()` in order to calculate all possible angles, and then creates a Dubins for each possibility choosing the one with the minimum length.

Parameters

in	*_points*	A `Tuple` of `Point2` through which the path should flow.
in	*_angles*	A `Tuple` of `Angle` containing all base `Angle`.
in	*area*	This is the angle around each angle to be "scanned".
in	*tries*	The number of discretizations that should be made.
in	*_kmax*	The maximum curvature to be used.

6.13.3.5 getBegin()

```
template<class T>
Configuration2<T> DubinsSet< T >::getBegin ( ) [inline]
```

Returns the starting `Configuration2` of the `DubinsSet`.

6.13.3.6 getDubins()

```
template<class T>
Dubins<T> DubinsSet< T >::getDubins (
            int id ) [inline]
```

This functions returns a specific `Dubins` from the set.

Parameters

in	*id*	The position of the `Dubins` in the set.

Returns

The id-th `Dubins`.

6.13.3.7 getDubinses()

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

Returns a `Tuple` containing all the `Dubins`.

6.13.3.8 getDubinsFrom()

```
template<class T>
Tuple<Dubins<T> > DubinsSet< T >::getDubinsFrom (
             int id ) [inline]
```

6.13.3.9 getDubinsPtr()

```
template<class T>
Dubins<T>* DubinsSet< T >::getDubinsPtr (
             int id ) [inline]
```

This functions returns a specific `Dubins` from the set.

Parameters

in	*id*	The position of the `Dubins` in the set.

Returns

The id-th `Dubins`.

6.13.3.10 getEnd()

```
template<class T>
Configuration2<T> DubinsSet< T >::getEnd ( ) [inline]
```

Returns the ending `Configuration2` of the `DubinsSet`

6.13.3.11 getKmax()

```
template<class T>
double DubinsSet< T >::getKmax ( ) [inline]
```

Returns the maximum curvature.

6.13.3.12 getLength()

```
template<class T>
double DubinsSet< T >::getLength ( ) [inline]
```

Returns the Length of the set of `Dubins`.

6.13.3.13 getSize()

```
template<class T>
double DubinsSet< T >::getSize ( ) [inline]
```

Returns the number of Dubins stored.

6.13.3.14 is_on_dubinsSet()

```
template<class T>
int DubinsSet< T >::is_on_dubinsSet (
            Configuration2< T > C ) [inline]
```

Function that checks whether a Configuration2 is on the DubinsSet or not.

Parameters

C	The Configuration2 to be checked.

Returns

The id of the Dubins on which the point is.

6.13.3.15 join()

```
template<class T>
DubinsSet<T> DubinsSet< T >::join (
            DubinsSet< T > * DS,
            int startPos = -1 ) [inline]
```

Function to join two DubinsSet.

This function joins two DubinsSet. If `this` is empty, than stores the Dubins from the DubinsSet to join. If it is not, then checks whether the Configuration2 of the ending Dubins coincides with the starting Configuration2 of the Dubins to join. If they do then the two sets can be merged, otherwise they cannot.

Parameters

DS	The `DubinSet` to join to `this`.
startPos	If this value is negative, then all Dubins in DS are going to be merged, otherwise only the Dubins from this position onwards.

Returns

`this`, that is the merged DubinsSet.

6.13.3.16 operator=()

```
template<class T>
DubinsSet<T> DubinsSet< T >::operator= (
             DubinsSet< T > * DS ) [inline]
```

Overload of operator =. It calls the function copy.

Parameters

DS	The `DubinsSet` to copy to `this`.

Returns

`this`, that is DS.

6.13.3.17 removeDubins()

```
template<class T>
void DubinsSet< T >::removeDubins ( ) [inline]
```

Remove the last `Dubins` from the set.

6.13.3.18 splitIt()

```
template<class T>
Tuple<Tuple<Tuple<Configuration2<T> > > > DubinsSet< T >::splitIt (
             double _length = PIECE_LENGTH ) [inline]
```

Function to split a Dubins in points.

Parameters

in	*_length*	The distance from one point to another.

Returns

A `Tuple` containing a `Tuple` containing three `Tuple` of `Configuration2` for each Dubins in the DubinsSet.

6.13.3.19 to_string()

```
template<class T>
stringstream DubinsSet< T >::to_string ( ) [inline]
```

This function create a strinstream object containing infos about the `DubinsSet`.

Returns

A string stream.

6.13.4 Friends And Related Function Documentation

6.13.4.1 operator<<

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

This function overload the << operator so to print with `std::cout` the values of the `DubinsSet`, that is printing all the `Dubins` stored.

Parameters

in	*out*	The out stream.
in	*data*	The `DubinsSet` to print.

Returns

An output stream to be printed.

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

- src/include/dubins.hh

6.14 Filter Class Reference

`#include <filter.hh>`

Public Member Functions

- Filter ()
  Default constructor: it set all values to 0.
- Filter (int _low_h, int _low_s, int _low_v, int _high_h, int _high_s, int _high_v)
  Constructor that sets all the values.
- Filter (vector< int > v)
  Constructor from a vector.
- Scalar Low ()

Returns a Scalar containing the lower boudary.
- Scalar High ()

  Returns a Scalar containing the lower boudary.
- stringstream to_string () const

  Save value in a stringstream.
- Filter copy (const Filter &fil)

  A function to copy a filter to this.
- Filter operator= (const Filter &filt)

  Overload of operator =. It just calls the copy function.
- operator vector< int > () const

  Overload of operator cast to vector<int>.

Public Attributes

- int low_h

  Lower value for hue.
- int low_s

  Lower value for saturation.
- int low_v

  Lower value for value.
- int high_h

  Higher value for hue.
- int high_s

  Higher value for saturation.
- int high_v

  Higher value for value.

Friends

- ostream & operator<< (ostream &out, const Filter &data)

6.14.1 Detailed Description

A class to store the values for an HSV filter with lower and higher boundary.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Filter() [1/3]

`Filter::Filter ( ) [inline]`

Default constructor: it set all values to 0.

6.14.2.2 Filter() [2/3]

```
Filter::Filter (
            int _low_h,
            int _low_s,
            int _low_v,
            int _high_h,
            int _high_s,
            int _high_v ) [inline]
```

Constructor that sets all the values.

Parameters

_low_h	Lower value for hue
_low_s	Lower value for saturation
_low_v	Lower value for value
_high_h	Higher value for hue
_high_s	Higher value for saturation
_high_v	Higher value for value

6.14.2.3 Filter() [3/3]

```
Filter::Filter (
            vector< int > v ) [inline]
```

Constructor from a vector.

Parameters

v	The vector containing the 6 values. Mind that they must be 6.

6.14.3 Member Function Documentation

6.14.3.1 copy()

```
Filter Filter::copy (
            const Filter & fil ) [inline]
```

A function to copy a filter to this.

Parameters

fil	The filter to be copied.

Returns

this filter with the new values copied.

6.14.3.2 High()

```
Scalar Filter::High ( )  [inline]
```

Returns a Scalar containing the lower boudary.

6.14.3.3 Low()

```
Scalar Filter::Low ( )  [inline]
```

Returns a Scalar containing the lower boudary.

6.14.3.4 operator vector< int >()

```
Filter::operator vector< int > ( ) const  [inline]
```

Overload of operator cast to vector<int>.

Returns

A vector containing the 6 values.

6.14.3.5 operator=()

```
Filter Filter::operator= (
            const Filter & filt )  [inline]
```

Overload of operator =. It just calls the copy function.

Parameters

filt	The filter to be copied.

Returns

this filter with the new values copied.

6.14.3.6 to_string()

```
stringstream Filter::to_string ( ) const  [inline]
```

Save value in a stringstream.

Returns

A stringstream containing the values of both boundaries.

6.14.4 Friends And Related Function Documentation

6.14.4.1 operator<<

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

This function overload the << operator so to print with `std::cout` .

Parameters

in	*out*	The out stream.
in	*data*	The filter to print.

Returns

An output stream to be printed.

6.14.5 Member Data Documentation

6.14.5.1 high_h

```
int Filter::high_h
```

Higher value for hue.

6.14.5.2 high_s

```
int Filter::high_s
```

Higher value for saturation.

6.14.5.3 high_v

```
int Filter::high_v
```

Higher value for value.

6.14.5.4 low_h

```
int Filter::low_h
```

Lower value for hue.

6.14.5.5 low_s

```
int Filter::low_s
```

Lower value for saturation.

6.14.5.6 low_v

```
int Filter::low_v
```

Lower value for value.

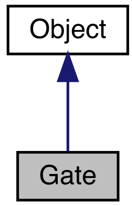
The documentation for this class was generated from the following file:

- src/include/filter.hh

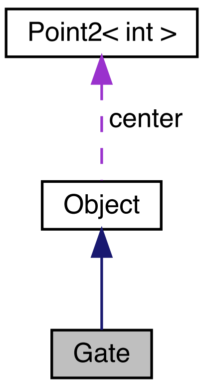
6.15 Gate Class Reference

`#include <objects.hh>`

Inheritance diagram for Gate:



Collaboration diagram for Gate:



Public Member Functions

- Gate (vector< Point2< int > > &vp)

  Constructor of the gate class and automatically compute center and radius.
- string toString ()

  Generate a string that describe the gate.
- void print ()

  Print the describing string of the gate.

Additional Inherited Members

6.15.1 Constructor & Destructor Documentation

6.15.1.1 Gate()

```
Gate::Gate (
            vector< Point2< int > > & vp )
```

Constructor of the gate class and automatically compute center and radius.

Parameters

in	*vp*	Vector of points that is the convex hull of the gate.

Returns

Return the created gate.

6.15.2 Member Function Documentation

6.15.2.1 print()

```
void Gate::print ( )
```

Print the describing string of the gate.

6.15.2.2 toString()

```
string Gate::toString ( )
```

Generate a string that describe the gate.

Returns

The generated string.

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

- src/include/objects.hh
- src/objects.cc

6.16 CameraCapture::input_options_t Struct Reference

Structure for store the input option for the class CameraCapture.

```
#include <camera_capture.hh>
```

Public Member Functions

- input_options_t ()
- input_options_t (const uint32_t frameHeight_px_, const uint32_t frameWidth_px_, const uint32_t cameraF←PS_, const uint32_t cameraId_)
- input_options_t (const input_options_t &inpOpt)

Public Attributes

- uint32_t frameHeight_px
- uint32_t frameWidth_px
- uint32_t cameraFPS
- char nameCamera [20]

6.16.1 Detailed Description

Structure for store the input option for the class CameraCapture.

frameHeight_px desidered height of the camera

frameWidth_px desidered width of the frame of the camera

cameraFPS desidered FPS of the camera

nameCamera is the camera filedescriptor (max 20 char)

6.16.2 Constructor & Destructor Documentation

6.16.2.1 input_options_t() [1/3]

```
CameraCapture::input_options_t::input_options_t ( )
```

6.16.2.2 input_options_t() [2/3]

```
CameraCapture::input_options_t::input_options_t (
            const uint32_t frameHeight_px_,
            const uint32_t frameWidth_px_,
            const uint32_t cameraFPS_,
            const uint32_t cameraId_ )
```

6.16.2.3 input_options_t() [3/3]

```
CameraCapture::input_options_t::input_options_t (
            const input_options_t & inpOpt )
```

6.16.3 Member Data Documentation

6.16.3.1 cameraFPS

```
uint32_t CameraCapture::input_options_t::cameraFPS
```

6.16.3.2 frameHeight_px

```
uint32_t CameraCapture::input_options_t::frameHeight_px
```

6.16.3.3 frameWidth_px

```
uint32_t CameraCapture::input_options_t::frameWidth_px
```

6.16.3.4 nameCamera

```
char CameraCapture::input_options_t::nameCamera[20]
```

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

- src/include/camera_capture.hh
- src/camera_capture.cc

6.17 ClipperLib::Int128 Class Reference

Public Member Functions

- Int128 (long64 _lo=0)
- Int128 (const Int128 &val)
- Int128 (const long64 &_hi, const ulong64 &_lo)
- Int128 & operator= (const long64 &val)
- bool operator== (const Int128 &val) const
- bool operator != (const Int128 &val) const
- bool operator > (const Int128 &val) const
- bool operator< (const Int128 &val) const
- bool operator >= (const Int128 &val) const
- bool operator<= (const Int128 &val) const
- Int128 & operator+= (const Int128 &rhs)
- Int128 operator+ (const Int128 &rhs) const
- Int128 & operator -= (const Int128 &rhs)
- Int128 operator - (const Int128 &rhs) const
- Int128 operator- () const
- operator double () const

Public Attributes

- ulong64 lo
- long64 hi

6.17.1 Constructor & Destructor Documentation

6.17.1.1 Int128() [1/3]

```
ClipperLib::Int128::Int128 (
            long64 _lo = 0 )  [inline]
```

6.17.1.2 Int128() [2/3]

```
ClipperLib::Int128::Int128 (
            const Int128 & val )  [inline]
```

6.17.1.3 Int128() [3/3]

```
ClipperLib::Int128::Int128 (
            const long64 & _hi,
            const ulong64 & _lo )  [inline]
```

6.17.2 Member Function Documentation

6.17.2.1 operator "!=()

```
bool ClipperLib::Int128::operator != (
             const Int128 & val ) const  [inline]
```

6.17.2.2 operator -()

```
Int128 ClipperLib::Int128::operator - (
             const Int128 & rhs ) const  [inline]
```

6.17.2.3 operator -=()

```
Int128& ClipperLib::Int128::operator -= (
             const Int128 & rhs )  [inline]
```

6.17.2.4 operator >()

```
bool ClipperLib::Int128::operator > (
             const Int128 & val ) const  [inline]
```

6.17.2.5 operator >=()

```
bool ClipperLib::Int128::operator >= (
             const Int128 & val ) const  [inline]
```

6.17.2.6 operator double()

```
ClipperLib::Int128::operator double ( ) const  [inline]
```

6.17.2.7 operator+()

```
Int128 ClipperLib::Int128::operator+ (
            const Int128 & rhs ) const [inline]
```

6.17.2.8 operator+=()

```
Int128& ClipperLib::Int128::operator+= (
            const Int128 & rhs ) [inline]
```

6.17.2.9 operator-()

```
Int128 ClipperLib::Int128::operator- ( ) const [inline]
```

6.17.2.10 operator<()

```
bool ClipperLib::Int128::operator< (
            const Int128 & val ) const [inline]
```

6.17.2.11 operator<=()

```
bool ClipperLib::Int128::operator<= (
            const Int128 & val ) const [inline]
```

6.17.2.12 operator=()

```
Int128& ClipperLib::Int128::operator= (
            const long64 & val ) [inline]
```

6.17.2.13 operator==()

```
bool ClipperLib::Int128::operator== (
            const Int128 & val ) const [inline]
```

6.17.3 Member Data Documentation

6.17.3.1 hi

`long64 ClipperLib::Int128::hi`

6.17.3.2 lo

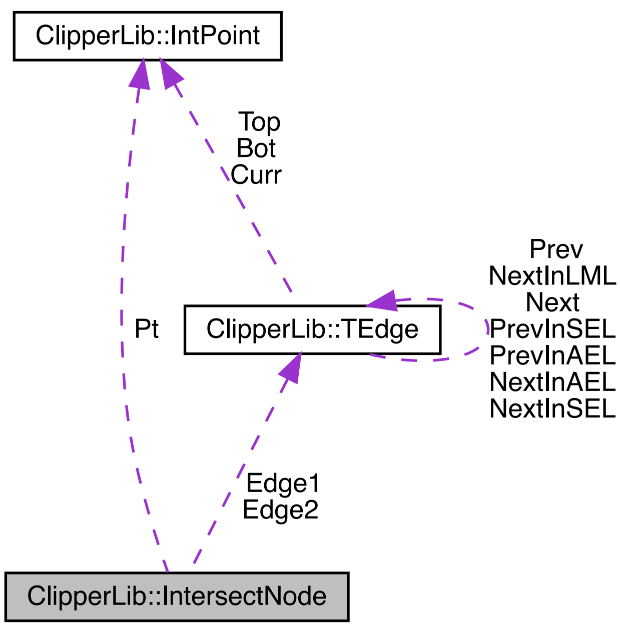
`ulong64 ClipperLib::Int128::lo`

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

- src/clipper.cc

6.18 ClipperLib::IntersectNode Struct Reference

Collaboration diagram for ClipperLib::IntersectNode:



Public Attributes

- TEdge ∗ Edge1
- TEdge ∗ Edge2
- IntPoint Pt

6.18.1 Member Data Documentation

6.18.1.1 Edge1

```
TEdge* ClipperLib::IntersectNode::Edge1
```

6.18.1.2 Edge2

```
TEdge* ClipperLib::IntersectNode::Edge2
```

6.18.1.3 Pt

```
IntPoint ClipperLib::IntersectNode::Pt
```

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

- src/clipper.cc

6.19 ClipperLib::IntPoint Struct Reference

```
#include <clipper.hh>
```

Public Member Functions

- IntPoint (cInt x=0, cInt y=0)

Public Attributes

- cInt X
- cInt Y

Friends

- bool operator== (const IntPoint &a, const IntPoint &b)
- bool operator!= (const IntPoint &a, const IntPoint &b)

6.19.1 Constructor & Destructor Documentation

6.19.1.1 IntPoint()

```
ClipperLib::IntPoint::IntPoint (
            cInt x = 0,
            cInt y = 0 )  [inline]
```

6.19.2 Friends And Related Function Documentation

6.19.2.1 operator"!=

```
bool operator!= (
            const IntPoint & a,
            const IntPoint & b )  [friend]
```

6.19.2.2 operator==

```
bool operator== (
            const IntPoint & a,
            const IntPoint & b )  [friend]
```

6.19.3 Member Data Documentation

6.19.3.1 X

```
cInt ClipperLib::IntPoint::X
```

6.19.3.2 Y

```
cInt ClipperLib::IntPoint::Y
```

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

- src/include/clipper.hh

6.20 ClipperLib::IntRect Struct Reference

```
#include <clipper.hh>
```

Public Attributes

- cInt left
- cInt top
- cInt right
- cInt bottom

6.20.1 Member Data Documentation

6.20.1.1 bottom

```
cInt ClipperLib::IntRect::bottom
```

6.20.1.2 left

```
cInt ClipperLib::IntRect::left
```

6.20.1.3 right

```
cInt ClipperLib::IntRect::right
```

6.20.1.4 top

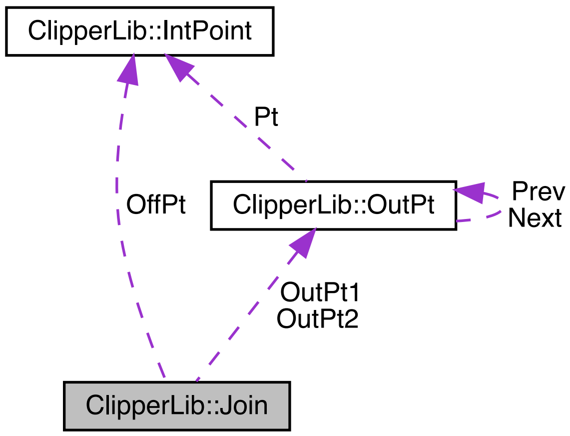
`cInt ClipperLib::IntRect::top`

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

- src/include/clipper.hh

6.21 ClipperLib::Join Struct Reference

Collaboration diagram for ClipperLib::Join:



Public Attributes

- OutPt ∗ OutPt1
- OutPt ∗ OutPt2
- IntPoint OffPt

6.21.1 Member Data Documentation

6.21.1.1 OffPt

`IntPoint ClipperLib::Join::OffPt`

6.21.1.2 OutPt1

```
OutPt* ClipperLib::Join::OutPt1
```

6.21.1.3 OutPt2

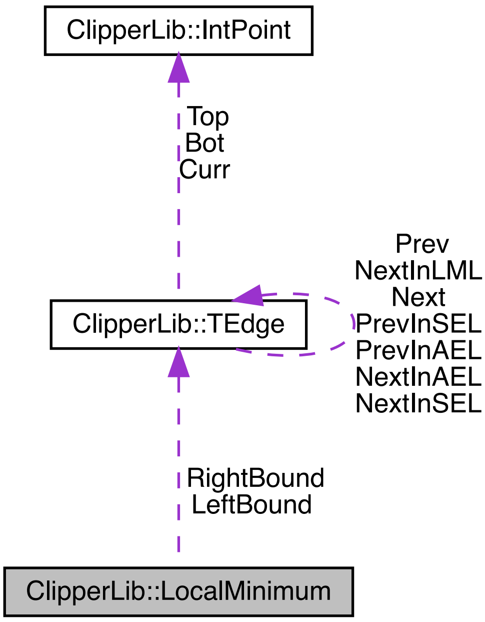
```
OutPt* ClipperLib::Join::OutPt2
```

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

- src/clipper.cc

6.22 ClipperLib::LocalMinimum Struct Reference

Collaboration diagram for ClipperLib::LocalMinimum:



Public Attributes

- cInt Y
- TEdge ∗ LeftBound
- TEdge ∗ RightBound

6.22.1 Member Data Documentation

6.22.1.1 LeftBound

```
TEdge* ClipperLib::LocalMinimum::LeftBound
```

6.22.1.2 RightBound

```
TEdge* ClipperLib::LocalMinimum::RightBound
```

6.22.1.3 Y

```
cInt ClipperLib::LocalMinimum::Y
```

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

- src/clipper.cc

6.23 ClipperLib::LocMinSorter Struct Reference

Public Member Functions

- bool operator() (const LocalMinimum &locMin1, const LocalMinimum &locMin2)

6.23.1 Member Function Documentation

6.23.1.1 operator()()

```
bool ClipperLib::LocMinSorter::operator() (
            const LocalMinimum & locMin1,
            const LocalMinimum & locMin2 )  [inline]
```

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

- src/clipper.cc

6.24 Mapp Class Reference

`#include <map.hh>`

Public Member Functions

- Mapp (const int _lengthX=1000, const int _lengthY=1500, const int _pixX=cellSize, const int _pixY=cellSize, const int _borderSize=borderSizeDefault, const vector< vector< Point2< int > > > &vvp=vector< vector< Point2< int > > >())

    Constructor of the class.
- ∼Mapp ()

    Destructor of the class.
- void addObject (const vector< Point2< int > > &vp, const OBJ_TYPE type)

    Given an object it is added to the map.
- void addObjects (const vector< vector< Point2< int > > > &vvp, const OBJ_TYPE type)

    Given a vector objects it is added them to the map.
- void addObjects (const vector< Obstacle > &objs)

    Given a vector of obstacles adds them to the map.
- void addObjects (const vector< Victim > &objs)

    Given a vector of victims adds them to the map.
- void addObjects (const vector< Gate > &objs)

    Given a vector of gates (tipically this vector contain only one element) adds it to the map.
- void getVictimCenters (vector< Point2< int > > &vp)

    Add to the given vector the set of centers of the victims of the map.
- void getGateCenter (vector< Point2< int > > &vp)

    Add to the given vector the center of the gate of the map.
- OBJ_TYPE getPointType (const Point2< int > &p)

    Given a point return the type (status) of the cell in the map that contain it.
- OBJ_TYPE getCellType (const int i, const int j)

    Given a cell return its type.
- bool checkSegment (const Point2< int > &p0, const Point2< int > &p1)

    Given a segment, the function answer if that segment cross a cell with obstacles.
- bool checkSegmentCollisionWithType (const Point2< int > &p0, const Point2< int > &p1, const OBJ_TYPE type)

    Given a segment and a type, the function answer if that segment cross a cell with the given type.
- bool checkPointInMap (Point2< int > P)

    Given a point, the function answer if that point is inside the map.
- bool checkPointInActualMap (Point2< int > P)

    Given a point, the function answer if that point is inside the actual map. This means that the border of the map also consider the offsetting due to the robot and not only the point.
- bool checkCellInMap (const int i, const int j)

    Given a cell(defined with its row and column), the function answer if that cell is inside the cell representation of the map.
- Mat createMapRepresentation ()

    The function create an image (Mat) with the dimensions of the Mapp and all its objects inside.
- void imageAddSegments (Mat &image, const vector< Point2< int > > &v, const int thickness=3)

    It add to the given image a set of (n-1) segments specified by the n points given.
- void imageAddSegments (Mat &image, const vector< Configuration2< double > > &v, const int thickness=3)

    It add to the given image a set of (n-1) segments specified by the n points given.

- void imageAddSegment (Mat &image, const Point2< int > &p0, const Point2< int > &p1, const int thickness=3, const Scalar color=Scalar(0, 255, 255))

    It add to the given image the segment defined from p0 to p1.
- void imageAddPoints (Mat &image, const vector< Configuration2< double > > &v, const int radius=7)

    It add to the given image a vector of points.
- void imageAddPoints (Mat &image, const vector< Point2< int > > &vp, const int radius=7)

    It add to the given image a vector of points.
- void imageAddPoint (Mat &image, const Point2< int > &p, const int radius=7, const Scalar color=Scalar(0, 255, 255))

    It add to the given image a point.
- void printMap ()

    Print to the terminal the main informations of the Map, and its grid representation.
- string matrixToString ()

    Generate a string (a grid of pixels) that represent the matrix.
- void printDimensions ()

    Print to the terminal the main informations of the Map.
- int getOffsetValue ()
- int getBorderSizeDefault ()
- int getCellSize ()
- int getLengthX ()
- int getLengthY ()
- int getActualLengthX ()
- int getActualLengthY ()
- int getDimX ()
- int getDimY ()
- int getPixX ()
- int getPixY ()
- int getBorderSize ()

Protected Member Functions

- set< pair< int, int > > cellsFromSegment (const Point2< int > &p0, const Point2< int > &p1)

    Given a segment (from p0 to p1) it return a set of all the cells that are partly cover from that segment.

Protected Attributes

- OBJ_TYPE ** map
- const int offsetValue = 75
- int lengthX
- int lengthY
- int dimX
- int dimY
- int pixX
- int pixY
- int borderSize
- vector< Obstacle > vObstacles
- vector< Victim > vVictims
- vector< Gate > vGates

Static Protected Attributes

- static const int borderSizeDefault = 10
- static const int cellSize = 5

6.24.1 Constructor & Destructor Documentation

6.24.1.1 Mapp()

```
Mapp::Mapp (
            const int _lengthX = 1000,
            const int _lengthY = 1500,
            const int _pixX = cellSize,
            const int _pixY = cellSize,
            const int _borderSize = borderSizeDefault,
            const vector< vector< Point2< int > > > & vvp = vector< vector<Point2<int> > >()
)
```

Constructor of the class.

Parameters

in	*_lengthX*	It is the size in pixel of the horizontal dimension.
in	*_lengthY*	It is the size in pixel of the vertical dimension.
in	*_pixX*	It is the horizontal granularity of a cell (how many pixels for each cell).
in	*_pixY*	It is the vertical granularity of a cell (how many pixels for each cell).
in	*_borderSize*	It is the dimension (defined based on cells of the map) of the border of each obstascles.
in	*vvp*	It is a vector, of vector, of point that delimit, as a convex hull, a set of obstacles in the map.

6.24.1.2 ∼Mapp()

```
Mapp::∼Mapp ( )
```

Destructor of the class.

6.24.2 Member Function Documentation

6.24.2.1 addObject()

```
void Mapp::addObject (
            const vector< Point2< int > > & vp,
            const OBJ_TYPE type )
```

Given an object it is added to the map.

This means that all the cells of the map that are partly cover from this obstacle will be set to its type.

Parameters

in	*vp*	It is the vector of points (convex hull) that delimit the object of interest.
in	*type*	It is the type of the given object. Defined as a OBJ_TYPE.

6.24.2.2 addObjects() [1/4]

```
void Mapp::addObjects (
            const vector< vector< Point2< int > > > & vvp,
            const OBJ_TYPE type )
```

Given a vector objects it is added them to the map.

This means that all the cells of the map that are partly cover from these obstacles will be set to its type. It is a wrapper function of addObject.

Parameters

in	*vvp*	It is the vector of vector of points (set of convex hull) that delimit the objects of interest.
in	*type*	It is the type of the given object. Defined as a OBJ_TYPE.

6.24.2.3 addObjects() [2/4]

```
void Mapp::addObjects (
            const vector< Obstacle > & objs )
```

Given a vector of obstacles adds them to the map.

This means that all the cells of the map that are partly cover from these obstacles will be set to its type. It is a wrapper function of addObject.

Parameters

in	*objs*	It is the vector of obstacles to be loaded in the map structure.

6.24.2.4 addObjects() [3/4]

```
void Mapp::addObjects (
            const vector< Victim > & objs )
```

Given a vector of victims adds them to the map.

This means that all the cells of the map that are partly cover from these victims will be set to its type. It is a wrapper function of addObject.

Parameters

in	*objs*	It is the vector of victims to be loaded in the map structure.

6.24.2.5 addObjects() [4/4]

```
void Mapp::addObjects (
            const vector< Gate > & objs )
```

Given a vector of gates (tipically this vector contain only one element) adds it to the map.

This means that all the cells of the map that are partly cover from this gate will be set to its type. It is a wrapper function of addObject.

Parameters

in	*objs*	It is the vector of gates to be loaded in the map structure.

6.24.2.6 cellsFromSegment()

```
set< pair< int, int > > Mapp::cellsFromSegment (
            const Point2< int > & p0,
            const Point2< int > & p1 ) [protected]
```

Given a segment (from p0 to p1) it return a set of all the cells that are partly cover from that segment.

Parameters

in	*p0*	First point of the segment.
in	*p1*	Second point of the segment.

Returns

A set containing all the cells, identified by their row(i or y) and column(j or x).

6.24.2.7 checkCellInMap()

```
bool Mapp::checkCellInMap (
            const int i,
            const int j )
```

Given a cell(defined with its row and column), the function answer if that cell is inside the cell representation of the map.

Parameters

in	*i*	The i=row of the cell.
in	*j*	The j=column of the cell.

Returns

True if the cell is inside the cell representation of the map, false otherwise.

6.24.2.8 checkPointInActualMap()

```
bool Mapp::checkPointInActualMap (
            Point2< int > P )
```

Given a point, the function answer if that point is inside the actual map. This means that the border of the map also consider the offsetting due to the robot and not only the point.

Parameters

in	*p*	The point that need to be checked.

Returns

True if the point is inside the actual map, false otherwise.

6.24.2.9 checkPointInMap()

```
bool Mapp::checkPointInMap (
            Point2< int > P )
```

Given a point, the function answer if that point is inside the map.

Parameters

in	*p*	The point that need to be checked.

Returns

True if the point is inside the map, false otherwise.

6.24.2.10 checkSegment()

```
bool Mapp::checkSegment (
            const Point2< int > & p0,
            const Point2< int > & p1 )
```

Given a segment, the function answer if that segment cross a cell with obstacles.

It is a wrapper for the function 'checkSegmentCollisionWithType'.

Parameters

in	*p0*	First point of the segment.
in	*p1*	Second point of the segment.

Returns

True if the obstacles were crossed, false otherwise.

6.24.2.11 checkSegmentCollisionWithType()

```
bool Mapp::checkSegmentCollisionWithType (
            const Point2< int > & p0,
            const Point2< int > & p1,
            const OBJ_TYPE type )
```

Given a segment and a type, the function answer if that segment cross a cell with the given type.

Parameters

in	*p0*	First point of the segment.
in	*p1*	Second point of the segment.
in	*type*	The type to be detected.

Returns

True if the type was found, false otherwise.

6.24.2.12 createMapRepresentation()

```
Mat Mapp::createMapRepresentation ( )
```

The function create an image (Mat) with the dimensions of the Mapp and all its objects inside.

Returns

The generated image is returned.

6.24.2.13 getActualLengthX()

```
int Mapp::getActualLengthX ( ) [inline]
```

6.24.2.14 getActualLengthY()

```
int Mapp::getActualLengthY ( ) [inline]
```

6.24.2.15 getBorderSize()

```
int Mapp::getBorderSize ( ) [inline]
```

6.24.2.16 getBorderSizeDefault()

```
int Mapp::getBorderSizeDefault ( ) [inline]
```

6.24.2.17 getCellSize()

```
int Mapp::getCellSize ( ) [inline]
```

6.24.2.18 getCellType()

```
OBJ_TYPE Mapp::getCellType (
            const int i,
            const int j )
```

Given a cell return its type.

Parameters

in	*i*	The row of the cell.
in	*j*	The column of the cell.

Returns

The type (OBJ_TYPE) of the requested cell.

6.24.2.19 getDimX()

```
int Mapp::getDimX ( ) [inline]
```

6.24.2.20 getDimY()

```
int Mapp::getDimY ( ) [inline]
```

6.24.2.21 getGateCenter()

```
void Mapp::getGateCenter (
        vector< Point2< int > > & vp )
```

Add to the given vector the center of the gate of the map.

Parameters

out	*vp*	A vector where the requested center will be added.

6.24.2.22 getLengthX()

```
int Mapp::getLengthX ( ) [inline]
```

6.24.2.23 getLengthY()

```
int Mapp::getLengthY ( ) [inline]
```

6.24.2.24 getOffsetValue()

```
int Mapp::getOffsetValue ( ) [inline]
```

6.24.2.25 getPixX()

```
int Mapp::getPixX ( ) [inline]
```

6.24.2.26 getPixY()

```
int Mapp::getPixY ( ) [inline]
```

6.24.2.27 getPointType()

```
OBJ_TYPE Mapp::getPointType (
            const Point2< int > & p )
```

Given a point return the type (status) of the cell in the map that contain it.

Parameters

in	*p*	The point of which we want to know the informations.

Returns

The type (OBJ_TYPE) of the cell.

6.24.2.28 getVictimCenters()

```
void Mapp::getVictimCenters (
            vector< Point2< int > > & vp )
```

Add to the given vector the set of centers of the victims of the map.

Parameters

out	*vp*	A vector where the requested centers will be added.

6.24.2.29 imageAddPoint()

```
void Mapp::imageAddPoint (
            Mat & image,
```

```
            const Point2< int > & p,
            const int radius = 7,
            const Scalar color = Scalar(0, 255, 255) )
```

It add to the given image a point.

Parameters

	[in/out]	map The image where the points will be added.
in	*p*	The point to add.
in	*radius*	The radius of the point to be drawn.
in	*color*	The color of the point to be drawn.

6.24.2.30 imageAddPoints() [1/2]

```
void Mapp::imageAddPoints (
            Mat & image,
            const vector< Configuration2< double > > & v,
            const int radius = 7 )
```

It add to the given image a vector of points.

Parameters

	[in/out]	map The image where the point will be added.
in	*v*	The vecotor of points to add.
in	*radius*	The radius of the points to be drawn.

6.24.2.31 imageAddPoints() [2/2]

```
void Mapp::imageAddPoints (
            Mat & image,
            const vector< Point2< int > > & v,
            const int radius = 7 )
```

It add to the given image a vector of points.

Parameters

	[in/out]	map The image where the point will be added.
in	*v*	The vecotor of points to add.
in	*radius*	The radius of the points to be drawn.

6.24.2.32 imageAddSegment()

```
void Mapp::imageAddSegment (
            Mat & image,
            const Point2< int > & p0,
            const Point2< int > & p1,
            const int thickness = 3,
            const Scalar color = Scalar(0, 255, 255) )
```

It add to the given image the segment defined from p0 to p1.

Parameters

	[in/out]	map The image where the segment will be added.
in	*p0*	The first point of the segment.
in	*p1*	The end point of the segment.
in	*thickness*	The thickness of the line to be drawn.
in	*colot*	The color of the line to be drawn.

6.24.2.33 imageAddSegments() [1/2]

```
void Mapp::imageAddSegments (
            Mat & image,
            const vector< Point2< int > > & v,
            const int thickness = 3 )
```

It add to the given image a set of (n-1) segments specified by the n points given.

Parameters

	[in/out]	map The image where the segments will be added.
in	*v*	The vector of points that identify the segments.
in	*thickness*	The thickness of the lines to be drawn.

6.24.2.34 imageAddSegments() [2/2]

```
void Mapp::imageAddSegments (
            Mat & image,
            const vector< Configuration2< double > > & v,
            const int thickness = 3 )
```

It add to the given image a set of (n-1) segments specified by the n points given.

Parameters

	[in/out]	map The image where the segments will be added.
in	*v*	The vector of points that identify the segments.
in	*thickness*	The thickness of the lines to be drawn.

6.24.2.35 matrixToString()

```
string Mapp::matrixToString ( )
```

Generate a string (a grid of pixels) that represent the matrix.

Returns

The generated string.

6.24.2.36 printDimensions()

```
void Mapp::printDimensions ( )
```

Print to the terminal the main informations of the Map.

6.24.2.37 printMap()

```
void Mapp::printMap ( )
```

Print to the terminal the main informations of the Map, and its grid representation.

6.24.3 Member Data Documentation

6.24.3.1 borderSize

```
int Mapp::borderSize [protected]
```

6.24.3.2 borderSizeDefault

```
const int Mapp::borderSizeDefault = 10 [static], [protected]
```

6.24.3.3 cellSize

```
const int Mapp::cellSize = 5  [static], [protected]
```

6.24.3.4 dimX

```
int Mapp::dimX  [protected]
```

6.24.3.5 dimY

```
int Mapp::dimY  [protected]
```

6.24.3.6 lengthX

```
int Mapp::lengthX  [protected]
```

6.24.3.7 lengthY

```
int Mapp::lengthY  [protected]
```

6.24.3.8 map

```
OBJ_TYPE** Mapp::map  [protected]
```

6.24.3.9 offsetValue

```
const int Mapp::offsetValue = 75  [protected]
```

6.24.3.10 pixX

```
int Mapp::pixX  [protected]
```

6.24.3.11 pixY

```
int Mapp::pixY [protected]
```

6.24.3.12 vGates

```
vector<Gate> Mapp::vGates [protected]
```

6.24.3.13 vObstacles

```
vector<Obstacle> Mapp::vObstacles [protected]
```

6.24.3.14 vVictims

```
vector<Victim> Mapp::vVictims [protected]
```

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

- src/include/map.hh
- src/map.cc

6.25 MyException< T > Class Template Reference

```
#include <utils.hh>
```

Inherits exception.

Public Member Functions

- MyException (EXCEPTION_TYPE _type, T _a, int _b, string _s="???")

  Plain constructor for the object.
- const char ∗ what () const throw ()

  Function to call to get the exception meaning.

Public Attributes

- EXCEPTION_TYPE type
- T a
- int b
- string s

6.25.1 Detailed Description

template<class T>
class MyException< T >

This class allows to throw personalized exceptions.

Template Parameters

T	The type of a variable.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 MyException()

```
template<class T >
MyException< T >::MyException (
              EXCEPTION_TYPE _type,
              T _a,
              int _b,
              string _s = "???" )  [inline]
```

Plain constructor for the object.

Parameters

in	*_type*	The type of the exception
in	*_a*	Variable meaning.
in	*_b*	Variable meaning.
in	*_s*	Variable meaning.

6.25.3 Member Function Documentation

6.25.3.1 what()

```
template<class T >
const char* MyException< T >::what ( ) const throw ( )   [inline]
```

Function to call to get the exception meaning.

Returns

A string containing why the exception was thrown.

6.25.4 Member Data Documentation

6.25.4.1 a

```
template<class T >
T MyException< T >::a
```

6.25.4.2 b

```
template<class T >
int MyException< T >::b
```

6.25.4.3 s

```
template<class T >
string MyException< T >::s
```

6.25.4.4 type

```
template<class T >
EXCEPTION_TYPE MyException< T >::type
```

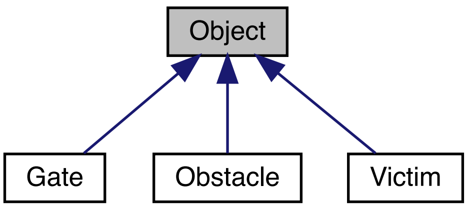
The documentation for this class was generated from the following file:

- src/include/utils.hh

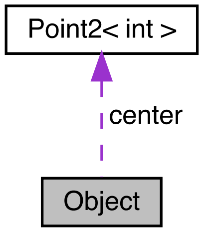
6.26 Object Class Reference

```
#include <objects.hh>
```

Inheritance diagram for Object:



Collaboration diagram for Object:



Public Member Functions

- string toString ()

  Generate a string that describe the object.
- unsigned int size ()

  Return the number of points of the object.
- unsigned int nPoints ()

  Return the number of points of the object.
- vector< Point2< int > > getPoints ()

  Return the of points of the object.
- Point2< int > getCenter ()

  Retrieve the center of the object.
- double getRadius ()

  Retrieve the radius of the object.
- void computeCenter ()

  Find the representative center of the object.
- void computeRadius ()

  Compute the radius of the object.
- void offsetting (const int offset, const int limitX, const int limitY)

  Enlarge the object of the given offset (defined as pixels=mm in our scenario).
- bool insidePolyApprox (Point2< int > pt)

  Check if the given point is inside the approximation shape of the object (a circle).
- bool insidePoly (Point2< int > pt)

  Exact check if a point is inside the object (no approximation).

Protected Attributes

- vector< Point2< int > > points
- Point2< int > center
- double radius

6.26.1 Member Function Documentation

6.26.1.1 computeCenter()

```
void Object::computeCenter ( )
```

Find the representative center of the object.

The center is computed as the mean of the minimum and maximum x and y.

6.26.1.2 computeRadius()

```
void Object::computeRadius ( )
```

Compute the radius of the object.

This function assume that the center of the object is already computed and consistent.

6.26.1.3 getCenter()

```
Point2< int > Object::getCenter ( )
```

Retrieve the center of the object.

Returns

The center.

6.26.1.4 getPoints()

```
vector< Point2< int > > Object::getPoints ( )
```

Return the of points of the object.

Returns

The vector of points.

6.26.1.5 getRadius()

```
double Object::getRadius ( )
```

Retrieve the radius of the object.

Returns

The radius.

6.26.1.6 insidePoly()

```
bool Object::insidePoly (
            Point2< int > pt )
```

Exact check if a point is inside the object (no approximation).

Parameters

in	*pt*	The point to be checked.

Returns

True if the point is inside the object, false otherwise.

6.26.1.7 insidePolyApprox()

```
bool Object::insidePolyApprox (
            Point2< int > pt )
```

Check if the given point is inside the approximation shape of the object (a circle).

Parameters

in	*pt*	The point to be checked.

Returns

True if the point is inside the object, false otherwise.

6.26.1.8 nPoints()

```
unsigned int Object::nPoints ( )
```

Return the number of points of the object.

Returns

The number of points.

6.26.1.9 offsetting()

```
void Object::offsetting (
            const int offset,
            const int limitX,
            const int limitY )
```

Enlarge the object of the given offset (defined as pixels=mm in our scenario).

The function automatically update even the center and the radius.

Parameters

in	*offset*	The size of the offset.
in	*limitX*	The the maximum x that the point can have.
in	*limitY*	The the maximum y that the point can have.

6.26.1.10 size()

```
unsigned int Object::size ( )
```

Return the number of points of the object.

Returns

The number of points.

6.26.1.11 toString()

```
string Object::toString ( )
```

Generate a string that describe the object.

Returns

The generated string.

6.26.2 Member Data Documentation

6.26.2.1 center

```
Point2<int> Object::center [protected]
```

6.26.2.2 points

```
vector<Point2<int> > Object::points [protected]
```

6.26.2.3 radius

```
double Object::radius [protected]
```

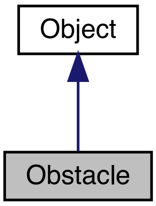
The documentation for this class was generated from the following files:

- src/include/objects.hh
- src/objects.cc

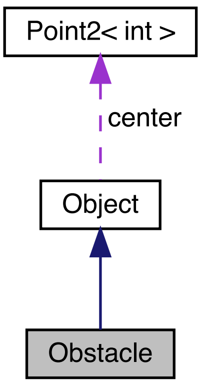
6.27 Obstacle Class Reference

```
#include <objects.hh>
```

Inheritance diagram for Obstacle:



Collaboration diagram for Obstacle:



Public Member Functions

- Obstacle (vector< Point2< int > > &vp)

  Constructor of the obstacle class and automatically compute center and radius.
- string toString ()

  Generate a string that describe the obstacle.
- void print ()

  Print the describing string of the obstacle.

Additional Inherited Members

6.27.1 Constructor & Destructor Documentation

6.27.1.1 Obstacle()

```
Obstacle::Obstacle (
            vector< Point2< int > > & vp )
```

Constructor of the obstacle class and automatically compute center and radius.

Parameters

in	*vp*	Vector of points that is the convex hull of the obstacle.

Returns

Return the created obstacle.

6.27.2 Member Function Documentation

6.27.2.1 print()

```
void Obstacle::print ( )
```

Print the describing string of the obstacle.

6.27.2.2 toString()

```
string Obstacle::toString ( )
```

Generate a string that describe the obstacle.

Returns

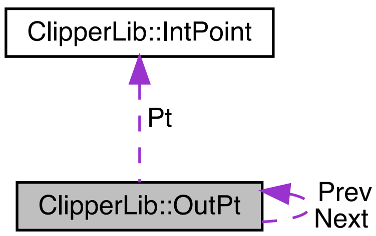
The generated string.

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

- src/include/objects.hh
- src/objects.cc

6.28 ClipperLib::OutPt Struct Reference

Collaboration diagram for ClipperLib::OutPt:



Public Attributes

- int Idx
- IntPoint Pt
- OutPt ∗ Next
- OutPt ∗ Prev

6.28.1 Member Data Documentation

6.28.1.1 Idx

```
int ClipperLib::OutPt::Idx
```

6.28.1.2 Next

```
OutPt* ClipperLib::OutPt::Next
```

6.28.1.3 Prev

```
OutPt* ClipperLib::OutPt::Prev
```

6.28.1.4 Pt

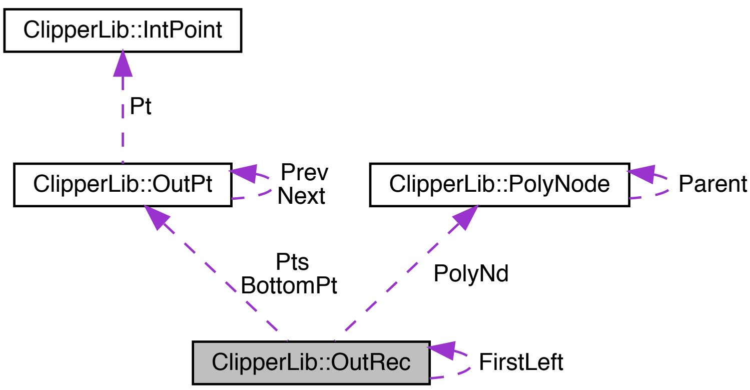
```
IntPoint ClipperLib::OutPt::Pt
```

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

- src/clipper.cc

6.29 ClipperLib::OutRec Struct Reference

Collaboration diagram for ClipperLib::OutRec:



Public Attributes

- int Idx
- bool IsHole
- bool IsOpen
- OutRec ∗ FirstLeft
- PolyNode ∗ PolyNd
- OutPt ∗ Pts
- OutPt ∗ BottomPt

6.29.1 Member Data Documentation

6.29.1.1 BottomPt

`OutPt* ClipperLib::OutRec::BottomPt`

6.29.1.2 FirstLeft

`OutRec* ClipperLib::OutRec::FirstLeft`

6.29.1.3 Idx

`int ClipperLib::OutRec::Idx`

6.29.1.4 IsHole

`bool ClipperLib::OutRec::IsHole`

6.29.1.5 IsOpen

`bool ClipperLib::OutRec::IsOpen`

6.29.1.6 PolyNd

`PolyNode* ClipperLib::OutRec::PolyNd`

6.29.1.7 Pts

`OutPt* ClipperLib::OutRec::Pts`

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

- src/clipper.cc

6.30 Point2< T > 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.
- Point2 (const cv::Point p)
  Constructor that takes a cv::Point and returns a Point2.
- T x () const
- T y () const
- void x (const T _x)
  Set the abscissa value.
- void y (const T _y)
  Set the ordinate value.
- template<class T1 >
  Point2< T > 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.
- Point2< T > offset (const Point2< T > p)
  This function compute an offset given another point made of the abscissa offset and the ordinate offset.
- Point2< T > offset (const Tuple< T > p)
  This function compute an offset given a `Tuple` made of the abscissa offset and the ordinate offset.
- Point2< T > offset_x (const T _offset)
  This function compute an offset for the abscissa.
- Point2< T > 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. \tparan 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. \tparan 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. \tparan T1 The type of the elements in the second `Point2`.
- stringstream to_string () const
  Returns a string representation of the object.
- Point2< T > copy (const Point2< T > &A)
  Copy a point into another one.
- Point2< T > operator= (const Point2< T > &A)
  Overload of the = operatore. Just calls `copy`.
- bool equal (const Point2< T > &A)
  Equalize two points.

- bool operator== (const Point2< T > &A)

  Overload of the == operator. Just calls `equal`.
- bool operator!= (const Point2< T > &A)

  Overload of the != operator. Just calls `equal` and negates it.
- operator cv::Point () const

  Cast to cv::Point.
- bool operator< (const Point2< T > &A)

  Overloading of operator <. Since no roght implementetion can be used, then it returns only `true`
- template<class T1 >
  Angle th (Point2< T1 > P1, Angle::ANGLE_TYPE type=Angle::RAD) const

  Computes the angle between two points, that is the atan of the angular coeficcient of the line joining the two points.
- template<class T1 >
  operator Point2< T1 > () const

  Cast to a Point2 of different type.
- void invert ()

  Invert the x and y of the point.

Friends

- ostream & operator<< (ostream &out, const Point2< T > &data)

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

6.30.1 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

T	The type of the coordinates to be stored.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 Point2() [1/3]

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

Default constructor to build an empty Tuple.

6.30.2.2 Point2() [2/3]

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

Constructor that taked to elements and builds a point.

Parameters

in	↩ _↩ x	The abscissa coordinate.
in	↩ _↩ y	The ordinate coordinate.

6.30.2.3 Point2() [3/3]

```
template<class T>
Point2< T >::Point2 (
            const cv::Point p )  [inline]
```

Constructor that takes a cv::Point and returns a Point2.

Parameters

in	*p*	The cv::Point to be copied.

6.30.3 Member Function Documentation

6.30.3.1 copy()

```
template<class T>
Point2<T> Point2< T >::copy (
            const Point2< T > & A )  [inline]
```

Copy a point into another one.

Parameters

in	*A*	point to be coppied.

Returns

this.

6.30.3.2 distance()

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

Wrapper to compute different distances. \tparan T1 The type of the elements in the second Point2.

Parameters

in	*B*	The second Point2 to use for computing the distance.
in	*dist*	The type of distance to be computed.

Returns

The distance between the two points. If something went wrong the return is -1.0.

6.30.3.3 equal()

```
template<class T>
bool Point2< T >::equal (
            const Point2< T > & A )  [inline]
```

Equalize two points.

Parameters

in	*A*	point to be compared to.

Returns

true if the two points are equal.

6.30.3.4 EuDistance()

```
template<class T>
template<class T1 >
```

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

Function that compute the Euclidean Distance between two points. \tparan T1 The type of the elements in the second Point2.

Parameters

in	*B*	the second Point2 to use for computing the distance.

Returns

The Euclidean distance between the two points.

6.30.3.5 invert()

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

Invert the x and y of the point.

6.30.3.6 MaDistance()

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

Function that compute the Manhattan Distance between two points. \tparan T1 The type of the elements in the second Point2.

Parameters

in	*B*	the second Point2 to use for computing the distance.

Returns

The Manhattan distance between the two points.

6.30.3.7 offset() [1/3]

```
template<class T>
template<class T1 >
```

```
Point2<T> 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

6.30.3.8 offset() [2/3]

```
template<class T>
Point2<T> 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	*p*	The point with the offsets.

Returns

1 if everything went fine, 0 otherwise.

6.30.3.9 offset() [3/3]

```
template<class T>
Point2<T> 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	*p*	The `Tuple` with the offsets. Its dimension must be 2.

Returns

1 if everything went fine, 0 otherwise.

6.30.3.10 offset_x()

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

This function compute an offset for the abscissa.

Parameters

in	*_offset*	The offset.

Returns

1 if everything went fine, 0 otherwise.

6.30.3.11 offset_y()

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

This function compute an offset for the ordinate.

Parameters

in	*_offset*	The offset.

Returns

1 if everything went fine, 0 otherwise.

6.30.3.12 operator cv::Point()

```
template<class T>
Point2< T >::operator cv::Point ( ) const [inline]
```

Cast to cv::Point.

Returns

The value casted to point

6.30.3.13 operator Point2< T1 >()

```
template<class T>
template<class T1 >
Point2< T >::operator Point2< T1 > ( ) const [inline]
```

Cast to a Point2 of different type.

Template Parameters

T1	The type of `Point2` to be casted to.

6.30.3.14 operator"!=()

```
template<class T>
bool Point2< T >::operator!= (
            const Point2< T > & A ) [inline]
```

Overload of the != operator. Just calls `equal` and negates it.

Parameters

in	*A*	point to be compared to.

Returns

true if the two configurations are different.

6.30.3.15 operator<()

```
template<class T>
bool Point2< T >::operator< (
            const Point2< T > & A ) [inline]
```

Overloading of operator $<$. Since no roght implementetion can be used, then it returns only `true`

Parameters

in	*A*	The second `Point2` to be compared to.

Returns

`true`.

6.30.3.16 operator=()

```
template<class T>
Point2<T> Point2< T >::operator= (
            const Point2< T > & A ) [inline]
```

Overload of the = operatore. Just calls `copy`.

Parameters

in	*A*	point to be coppied.

Returns

this.

6.30.3.17 operator==()

```
template<class T>
bool Point2< T >::operator== (
            const Point2< T > & A ) [inline]
```

Overload of the == operator. Just calls `equal`.

Parameters

in	*A*	point to be compared to.

Returns

true if the two configurations are equal.

6.30.3.18 th()

```
template<class T>
template<class T1 >
Angle Point2< T >::th (
            Point2< T1 > P1,
            Angle::ANGLE_TYPE type = Angle::RAD ) const [inline]
```

Computes the angle between two points, that is the atan of the angular coeficcient of the line joining the two points.

Parameters

in	*P1*	The point towards which the line is going.
in	*type*	The type of the `Angle` to be returned.

Template Parameters

T1	The type of the point.

Returns

The Angle.

6.30.3.19 to_string()

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

Returns a string representation of the object.

Returns

String representation of the object.

6.30.3.20 x() [1/2]

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

Returns

The abscissa coordinate

6.30.3.21 x() [2/2]

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

Set the abscissa value.

Parameters

in	↩ _↩ *x*	The new abscissa value

Returns

1 if it was successful, 0 otherwise.

6.30.3.22 y() [1/2]

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

Returns

The ordinate coordinate

6.30.3.23 y() [2/2]

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

Set the ordinate value.

Parameters

in	↩ _↩ x	The new ordinate value

Returns

1 if it was successful, 0 otherwise.

6.30.4 Friends And Related Function Documentation

6.30.4.1 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	*out*	The output stream.
in	*data*	The `Point2` to print.

Returns

An output stream to be printed.

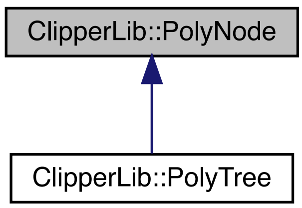
The documentation for this class was generated from the following file:

- src/include/maths.hh

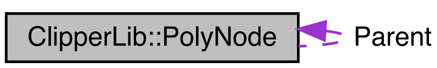
6.31 ClipperLib::PolyNode Class Reference

```
#include <clipper.hh>
```

Inheritance diagram for ClipperLib::PolyNode:



Collaboration diagram for ClipperLib::PolyNode:



Public Member Functions

- PolyNode ()
- virtual ∼PolyNode ()
- PolyNode ∗ GetNext () const
- bool IsHole () const
- bool IsOpen () const
- int ChildCount () const

Public Attributes

- Path Contour
- PolyNodes Childs
- PolyNode * Parent

Friends

- class Clipper
- class ClipperOffset

6.31.1 Constructor & Destructor Documentation

6.31.1.1 PolyNode()

`ClipperLib::PolyNode::PolyNode ( )`

6.31.1.2 ∼PolyNode()

`virtual ClipperLib::PolyNode::∼PolyNode ( ) [inline], [virtual]`

6.31.2 Member Function Documentation

6.31.2.1 ChildCount()

`int ClipperLib::PolyNode::ChildCount ( ) const`

6.31.2.2 GetNext()

`PolyNode * ClipperLib::PolyNode::GetNext ( ) const`

6.31.2.3 IsHole()

`bool ClipperLib::PolyNode::IsHole ( ) const`

6.31.2.4 IsOpen()

```
bool ClipperLib::PolyNode::IsOpen ( ) const
```

6.31.3 Friends And Related Function Documentation

6.31.3.1 Clipper

```
friend class Clipper [friend]
```

6.31.3.2 ClipperOffset

```
friend class ClipperOffset [friend]
```

6.31.4 Member Data Documentation

6.31.4.1 Childs

```
PolyNodes ClipperLib::PolyNode::Childs
```

6.31.4.2 Contour

```
Path ClipperLib::PolyNode::Contour
```

6.31.4.3 Parent

```
PolyNode* ClipperLib::PolyNode::Parent
```

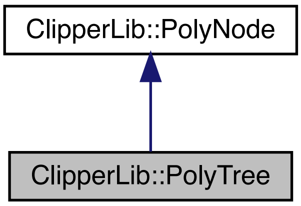
The documentation for this class was generated from the following files:

- src/include/clipper.hh
- src/clipper.cc

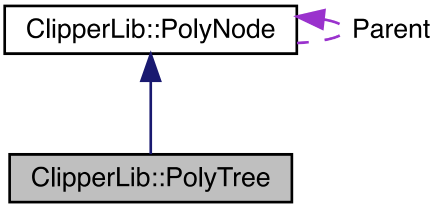
6.32 ClipperLib::PolyTree Class Reference

`#include <clipper.hh>`

Inheritance diagram for ClipperLib::PolyTree:



Collaboration diagram for ClipperLib::PolyTree:



Public Member Functions

- ∼PolyTree ()
- PolyNode ∗ GetFirst () const
- void Clear ()
- int Total () const

Friends

- class Clipper

Additional Inherited Members

6.32.1 Constructor & Destructor Documentation

6.32.1.1 ∼PolyTree()

```
ClipperLib::PolyTree::∼PolyTree ( ) [inline]
```

6.32.2 Member Function Documentation

6.32.2.1 Clear()

```
void ClipperLib::PolyTree::Clear ( )
```

6.32.2.2 GetFirst()

```
PolyNode * ClipperLib::PolyTree::GetFirst ( ) const
```

6.32.2.3 Total()

```
int ClipperLib::PolyTree::Total ( ) const
```

6.32.3 Friends And Related Function Documentation

6.32.3.1 Clipper

```
friend class Clipper [friend]
```

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

- src/include/clipper.hh
- src/clipper.cc

6.33 RobotProject Class Reference

```
#include <robotProject.hh>
```

Public Member Functions

- RobotProject (CameraCapture ∗camera, double &frame_time)

    The main constructor of the class.
- ∼RobotProject ()

    The destructor of the class.
- bool preprocessMap (const Mat &img)

    Taken an image this function elaborate it in order to detect the foundamental elements and store them on files.
- bool planPath (const Mat &img, Path &path)

    Taken an image this function try to compute (and return) a path on it, that will bring the robot from its actual position through all the victims and in the end up to the gate.
- bool localize (const Mat &img, vector< double > &state)

    Taken an image this function try to localize the position and the orientation of the robot. It also apply the neccessary transformation matrix to solve the problem of the different planes.

6.33.1 Constructor & Destructor Documentation

6.33.1.1 RobotProject()

```
RobotProject::RobotProject (
            CameraCapture ∗ camera,
            double & frame_time )
```

The main constructor of the class.

Parameters

in	*camera*	It is the camera from which the image will be loaded.
in	*frame_time*	The index of the frame as last one from the camera.

Returns

6.33.1.2 ∼RobotProject()

```
RobotProject::∼RobotProject ( )
```

The destructor of the class.

6.33.2 Member Function Documentation

6.33.2.1 localize()

```
bool RobotProject::localize (
            const Mat & img,
            vector< double > & state )
```

Taken an image this function try to localize the position and the orientation of the robot. It also apply the neccessary transformation matrix to solve the problem of the different planes.

Parameters

in	*img*	The immage that will be processed.
out	*state*	The state that acts as the return value of the function.

Returns

A true value if everything goes well. False otherwise.

6.33.2.2 planPath()

```
bool RobotProject::planPath (
            const Mat & img,
            Path & path )
```

Taken an image this function try to compute (and return) a path on it, that will bring the robot from its actual position through all the victims and in the end up to the gate.

Parameters

in	*img*	The immage that will be processed.
out	*path*	The path that acts as the return value of the function.

Returns

A true value if everything goes well. False otherwise.

6.33.2.3 preprocessMap()

```
bool RobotProject::preprocessMap (
            const Mat & img )
```

Taken an image this function elaborate it in order to detect the foundamental elements and store them on files.

Parameters

in	*img*	The immage that will be processed.

Returns

A true value if everything goes well. False otherwise.

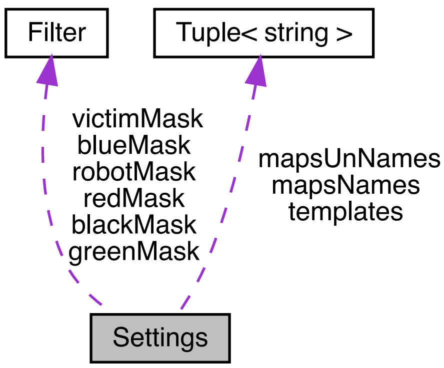
The documentation for this class was generated from the following files:

- src/include/robotProject.hh
- src/robotProject.cc

6.34 Settings Class Reference

`#include <settings.hh>`

Collaboration diagram for Settings:



Public Types

- enum COLOR {
  BLACK, RED, GREEN, VICTIMS,
  BLUE, ROBOT }

Public Member Functions

- Settings (string _baseFolder="data/", string _mapsFolder="map", string _templatesFolder="num_template/", vector< string > _mapsNames={}, vector< string > _mapsUnNames={}, string _intrinsicCalibrationFile="intrinsic_calibration.xml", string _calibrationFile="calib_config.xml", Filter _blackMask=Filter(0, 0, 0, 179, 255, 70), Filter _redMask=Filter(15, 100, 140, 160, 255, 255), Filter _greenMask=Filter(54, 74, 25, 119, 255, 88), Filter _victimMask=Filter(0, 0, 0, 179, 255, 80), Filter _blueMask=Filter(100, 100, 40, 140, 200, 170), Filter _roboteMask=Filter(100, 100, 40, 140, 200, 170), int _kernelSide=9, string _convexHullFile="convexHull.xml", vector< string > _templates={})

  Constructor of class Settings. The value are all set by default. The constructor does NOT read from or write to file.
- ~Settings ()

  Destructor.
- void save (string _baseFolder="data/", string _mapsFolder="map/", string _templatesFolder="num_template/", vector< string > _mapsNames={}, vector< string > _mapsUnNames={}, string _intrinsicCalibrationFile="intrinsic_calibration.xml", string _calibrationFile="calib_config.xml", Filter _blackMask=Filter(0, 0, 0, 179, 255, 70), Filter _redMask=Filter(15, 100, 140, 160, 255, 255), Filter _greenMask=Filter(54, 74, 25, 119, 255, 88), Filter _victimMask=Filter(0, 0, 0, 179, 255, 80), Filter _blueMask=Filter(100, 100, 40, 140, 200, 170), Filter _roboteMask=Filter(100, 100, 40, 140, 200, 170), int _kernelSide=9, string _convexHullFile="convexHull.xml", vector< string > _templates={})

  Function to change values. The value are all set by default. This function does NOT read from or write to file.
- void writeToFile (string _path="")

  Function to write settings to file. Default is data/settings.xml.
- void readFromFile (string _path="")

  Function to read from file. The data found is going to be added to the settings. Default file is data/settings.xml.
- void clean ()

  Function to clean all settings: number types are set to 0, string are set to "", Tuples are set to Tuple<>() and Filter are set to all 0s.
- void cleanAndRead (string _path="")

  Function to clean all settings and then read from file. If no path is given the baseFolder is used.
- Tuple< string > maps (Tuple< int > ids=Tuple< int >())

  Function to return the paths of maps. If ids are not specified all maps are returned.
- Tuple< string > maps (int id=-1)

  Function to return the path of a map. If id is negative all maps are returned.
- string maps (string _mapName)

  A function to return the path of a given map.
- Tuple< string > maps (Tuple< string > _mapNames)

  A function to return the paths of a given Tuple of maps.
- bool addUnMap (string unMap)

  Adds the name of an undistorted map to the list.
- Tuple< string > unMaps (Tuple< int > ids=Tuple< int >())

  Function to return the paths of undistorted maps. If ids are not specified all undistorted maps are returned.
- Tuple< string > unMaps (int id=-1)

  Function to return the path of an undistorted map. If id is negative all undistorted maps are returned.
- string unMaps (string _unMapName)

  A function to return the path of a given undistorted map.
- Tuple< string > unMaps (Tuple< string > _unMapNames)

  A function to return the paths of a given Tuple of undistorted maps.
- Tuple< string > getTemplates (int id=-1)

  Function to return the path of a template. If id is negative all templates are returned.
- string getTemplates (string _template)

  A function to return the path of a given template.
- Tuple< string > getTemplates (Tuple< string > _templates)

  A function to return the paths of a given Tuple of templates.

- void changeMask (Tuple< COLOR > color, Tuple< Filter > fil)

  Change the values of Tuple of filters. Mind that no write function is called.
- void changeMask (COLOR color, Filter fil)

  Change the values of a filter. Mind that no write function is called.
- stringstream to_string () const

  A function that creates a stringstream to print the values stored in settings.

Public Attributes

- string baseFolder

  A string containing the path for the base dir of data.
- string mapsFolder

  A string containing the name for maps folder. No certainty is given about the form of this string.
- Tuple< string > mapsNames

  A Tuple containing the names of the maps. These are not paths but just names.
- Tuple< string > mapsUnNames

  A Tuple containing the names of the undistorted maps. These are not paths but just names.
- string intrinsicCalibrationFile

  A string containing the name to the file containing the values of the matrix for the calibration.
- string calibrationFile

  A string containing the name to the file containing the data for the calibration.
- Filter blackMask

  Filter for black.
- Filter redMask

  Filter for red.
- Filter greenMask

  Filter for green.
- Filter victimMask

  Filter for the victims.
- Filter blueMask

  Filter for blue.
- Filter robotMask

  Filter for the triangle above the robot.
- int kernelSide
- string convexHullFile

  AString containing the name to file containing the points of the elements in the arena.
- string templatesFolder

  A String containing the name of the folder containing the number templates.
- Tuple< string > templates

  A Tuple containing the names of the templates. These are not paths but just names.

Friends

- ostream & operator<< (ostream &out, const Settings &data)

6.34.1 Detailed Description

Class that stores settings for the projects such as location of files, name of maps and filters to use. Mind that when created it does not read from file by default but the function must be invoked.

6.34.2 Member Enumeration Documentation

6.34.2.1 COLOR

```
enum Settings::COLOR
```

Enumerator

BLACK	
RED	
GREEN	
VICTIMS	
BLUE	
ROBOT	

6.34.3 Constructor & Destructor Documentation

6.34.3.1 Settings()

```
Settings::Settings (
            string _baseFolder = "data/",
            string _mapsFolder = "map",
            string _templatesFolder = "num_template/",
            vector< string > _mapsNames = {},
            vector< string > _mapsUnNames = {},
            string _intrinsicCalibrationFile = "intrinsic_calibration.xml",
            string _calibrationFile = "calib_config.xml",
            Filter _blackMask = Filter(0, 0, 0, 179, 255, 70),
            Filter _redMask = Filter(15, 100, 140, 160, 255, 255),
            Filter _greenMask = Filter(54, 74, 25, 119, 255, 88),
            Filter _victimMask = Filter(0, 0, 0, 179, 255, 80),
            Filter _blueMask = Filter(100, 100, 40, 140, 200, 170),
            Filter _robotMask = Filter(100, 100, 40, 140, 200, 170),
            int _kernelSide = 9,
            string _convexHullFile = "convexHull.xml",
            vector< string > _templates = {} )
```

Constructor of class Settings. The value are all set by default. The constructor does NOT read from or write to file.

Parameters

in	*baseFolder*	A string containing the path for the base dir of data.
in	*mapsFolder*	A string containing the name for maps folder. No certainty is given about the form of this string.
in	*_templatesFolder*	A String containing the name of the folder containing the number templates.

Parameters

in	*_mapsNames*	A Tuple containing the names of the maps. These are not paths but just names.
in	*_mapsUnNames*	A Tuple containing the names of the undistorted maps. These are not paths but just names.
in	*_calibrationFile*	A string containing the name to the file containing the data for the calibration.
in	*_intrinsicCalibrationFile*	A string containing the name to the file containing the values of the matrix for the calibration.
in	*_blackMask*	Filter for black.
in	*_redMask*	Filter for red.
in	*_greenMask*	Filter for green.
in	*_victimMask*	Filter for the victims.
in	*_blueMask*	Filter for blue.
in	*_robotMask*	Filter for the triangle above the robot.
in	*_kernelSide*	
in	*_convexHullFile*	A String containing the name to file containing the points of the elements in the arena.
in	*_templates*	A Tuple containing the names of the templates. These are not paths but just names.
	mapsFolder	A string containing the path for mapsFolder. No certainty is given about the form of this string
	_templatesFolder	A String containing the path of the folder containing the number templates.
	_mapsNames	A Tuple containing the names of the maps. These are not paths but just names.
	_mapsUnNames	A Tuple containing the names of the undistorted maps. These are not paths but just names.
	_calibrationFile	A string containing the path to the file containing the data for the calibration.
	_intrinsicCalibrationFile	A string containing the path to the file containing the values of the matrix for the calibration.
	_blackMask	Filter for black.
	_redMask	Filter for red.
	_greenMask	Filter for green.
	_victimMask	Filter for the victims.
	_blueMask	Filter for blue.
	_robotMask	Filter for the triangle above the robot.
	_kernelSide	
	_convexHullFile	A String containing the path to file containing the points of the elements in the arena.
	_templates	A Tuple containing the names of the templates. These are not paths but just names.

6.34.3.2 ∼Settings()

```
Settings::∼Settings ( )
```

Destructor.

6.34.4 Member Function Documentation

6.34.4.1 addUnMap()

```
bool Settings::addUnMap (
            string _unMap )
```

Adds the name of an undistorted map to the list.

Adds the name of an undistorted map.

Parameters

in	*unMap*	The undistorted map

Returns

`true` of the name of the map could be added, `false` otherwise.

Parameters

in	*_unMap*	The name of the undistorted map

Returns

`true` if it succeded, `false` otherwise.

6.34.4.2 changeMask() [1/2]

```
void Settings::changeMask (
            Tuple< COLOR > color,
            Tuple< Filter > fil )
```

Change the values of Tuple of filters. Mind that no write function is called.

Parameters

color	A Tuple containing the colors of the filters to change.
fil	The new filters to be stored.

6.34.4.3 changeMask() [2/2]

```
void Settings::changeMask (
            COLOR color,
            Filter fil )
```

Change the values of a filter. Mind that no write function is called.

Parameters

color	The filter to change.
fil	The new filter to be stored.

6.34.4.4 clean()

```
void Settings::clean ( )
```

Function to clean all settings: number types are set to 0, string are set to "", Tuples are set to Tuple<>() and Filter are set to all 0s.

6.34.4.5 cleanAndRead()

```
void Settings::cleanAndRead (
            string _path = "" )
```

Function to clean all settings and then read from file. If no path is given the baseFolder is used.

Function to clean all settings and then read from file. Default is data/settings.xml.

Parameters

in	*_path*	Path to the file. Mind that it doesn't require the name of the file.

6.34.4.6 getTemplates() [1/3]

```
Tuple< string > Settings::getTemplates (
            int id = -1 )
```

Function to return the path of a template. If id is negative all templates are returned.

Function to return the path of a template. If id is not specified all templates are returned.

Parameters

id	The positions in this.templates of the template to be retrieved

Returns

A Tuple containing the paths of the templates.

6.34.4.7 getTemplates() [2/3]

```
string Settings::getTemplates (
            string _template )
```

A function to return the path of a given template.

Parameters

_templateName	The name of the template to check in the Tuple.

Returns

The path to the template if it is found, an empty string otherwise.

6.34.4.8 getTemplates() [3/3]

```
Tuple< string > Settings::getTemplates (
            Tuple< string > _templates )
```

A function to return the paths of a given Tuple of templates.

Parameters

_template	A Tuple containing the names of the templates to check in the Tuple.

Returns

The paths to the templates if they are found, an empty Tuple otherwise.

6.34.4.9 maps() [1/4]

```
Tuple< string > Settings::maps (
            Tuple< int > ids = Tuple<int>() )
```

Function to return the paths of maps. If ids are not specified all maps are returned.

Parameters

ids	A Tuple containing the ids (that is the positions in this.mapsNames) of the maps to be retrieved.

Returns

A Tuple containing the paths of the maps.

6.34.4.10 maps() [2/4]

```
Tuple< string > Settings::maps (
            int id = -1 )
```

Function to return the path of a map. If id is negative all maps are returned.

Function to return the path of a map. If id is not specified all maps are returned.

Parameters

id	The positions in this.mapsNames of the map to be retrieved

Returns

A Tuple containing the paths of the maps.

Parameters

id	A the positions in this.mapsNames of the map to be retrieved

Returns

A Tuple containing the paths of the maps.

6.34.4.11 maps() [3/4]

```
string Settings::maps (
            string _mapName )
```

A function to return the path of a given map.

Parameters

_mapName	The name of the map to check in the Tuple.

Returns

The path to the map if the map is found, an empty string otherwise.

6.34.4.12 maps() [4/4]

```
Tuple< string > Settings::maps (
            Tuple< string > _mapNames )
```

A function to return the paths of a given Tuple of maps.

Parameters

_mapNames	A Tuple containing the names of the maps to check in the Tuple.

Returns

The paths to the maps if they are found, an empty Tuple otherwise.

6.34.4.13 readFromFile()

```
void Settings::readFromFile (
            string _path = "" )
```

Function to read from file. The data found is going to be added to the settings. Default file is data/settings.xml.

Parameters

in	*_path*	Path to the file. Mind that it doesn't require the name of the file.
	_path	The path of file to read from.

6.34.4.14 save()

```
void Settings::save (
            string _baseFolder = "data/",
            string _mapsFolder = "map/",
            string _templatesFolder = "num_template/",
            vector< string > _mapsNames = {},
            vector< string > _mapsUnNames = {},
            string _intrinsicCalibrationFile = "intrinsic_calibration.xml",
            string _calibrationFile = "calib_config.xml",
            Filter _blackMask = Filter(0, 0, 0, 179, 255, 70),
```

```
Filter _redMask = Filter(15, 100, 140, 160, 255, 255),
Filter _greenMask = Filter(54, 74, 25, 119, 255, 88),
Filter _victimMask = Filter(0, 0, 0, 179, 255, 80),
Filter _blueMask = Filter(100, 100, 40, 140, 200, 170),
Filter _robotMask = Filter(100, 100, 40, 140, 200, 170),
int _kernelSide = 9,
string _convexHullFile = "convexHull.xml",
vector< string > _templates = {} )
```

Function to change values. The value are all set by default. This function does NOT read from or write to file.

Parameters

in	*baseFolder*	A string containing the path for the base dir of data.
in	*mapsFolder*	A string containing the name for mapsFolder. No certainty is given about the form of this string
in	*_templatesFolder*	A String containing the name of the folder containing the number templates.
in	*_mapsNames*	A Tuple containing the names of the maps. These are not paths but just names.
in	*_mapsUnNames*	A Tuple containing the names of the undistorted maps. These are not paths but just names.
in	*_calibrationFile*	A string containing the name to the file containing the data for the calibration.
in	*_intrinsicCalibrationFile*	A string containing the name to the file containing the values of the matrix for the calibration.
in	*_blackMask*	Filter for black.
in	*_redMask*	Filter for red.
in	*_greenMask*	Filter for green.
in	*_victimMask*	Filter for the victims.
in	*_blueMask*	Filter for blue.
in	*_robotMask*	Filter for the triangle above the robot.
in	*_kernelSide*	
in	*_convexHullFile*	A String containing the name to file containing the points of the elements in the arena.
in	*_templates*	A Tuple containing the names of the templates. These are not paths but just names.
	mapsFolder	A string containing the path for mapsFolder. No certainty is given about the form of this string
	_templatesFolder	A String containing the path of the folder containing the number templates.
	_mapsNames	A Tuple containing the names of the maps. These are not paths but just names.
	_mapsUnNames	A Tuple containing the names of the undistorted maps. These are not paths but just names.
	_intrinsicCalibrationFile	A string containing the path to the file containing the values of the matrix for the calibration.
	_calibrationFile	A string containing the path to the file containing the data for the calibration.
	_blackMask	Filter for black.
	_redMask	Filter for red.
	_greenMask	Filter for green.
	_victimMask	Filter for the victims.
	_blueMask	Filter for blue.
	_robotMask	Filter for the triangle above the robot.
	_kernelSide	
	_convexHullFile	A String containing the path to file containing the points of the elements in the arena.

Parameters

	_templates	A Tuple containing the names of the templates. These are not paths but just names.

6.34.4.15 to_string()

```
stringstream Settings::to_string ( ) const  [inline]
```

A function that creates a stringstream to print the values stored in settings.

Returns

A strinstream containing the settings values.

6.34.4.16 unMaps() [1/4]

```
Tuple< string > Settings::unMaps (
            Tuple< int > ids = Tuple<int>() )
```

Function to return the paths of undistorted maps. If ids are not specified all undistorted maps are returned.

Parameters

ids	A Tuple containing the ids (that is the positions in this.mapsUnNames) of the undistorted maps to be retrieved.

Returns

A Tuple containing the paths of the undistorted maps.

6.34.4.17 unMaps() [2/4]

```
Tuple< string > Settings::unMaps (
            int id = -1 )
```

Function to return the path of an undistorted map. If id is negative all undistorted maps are returned.

Function to return the path of an undistorted map. If id is not specified all undistorted maps are returned.

Parameters

id	The positions in this.mapsUnNames of the undistorted map to be retrieved

Returns

A Tuple containing the paths of the undistorted maps.

Parameters

id	A the positions in this.mapsUnNames of the undistorted map to be retrieved

Returns

A Tuple containing the paths of the undistorted maps.

6.34.4.18 unMaps() [3/4]

```
string Settings::unMaps (
            string _unMapName )
```

A function to return the path of a given undistorted map.

Parameters

_unMapName	The name of the undistorted map to check in the Tuple.

Returns

The path to the undistorted map if it is found, an empty string otherwise.

6.34.4.19 unMaps() [4/4]

```
Tuple< string > Settings::unMaps (
            Tuple< string > _unMapNames )
```

A function to return the paths of a given Tuple of undistorted maps.

Parameters

_unMapNames	A Tuple containing the names of the undistorted maps to check in the Tuple.

Returns

The paths to the undistorted maps if they are found, an empty Tuple otherwise.

6.34.4.20 writeToFile()

```
void Settings::writeToFile (
            string _path = "" )
```

Function to write settings to file. Default is data/settings.xml.

Parameters

in	*_path*	Path to the file. Mind that it doesn't require the name of the file.
	_path	The path of the file to write to.

6.34.5 Friends And Related Function Documentation

6.34.5.1 operator<<

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

This function overload the << operator so to print with `std::cout`.

Parameters

in	*out*	The out stream.
in	*datThe*	settings to print.

Returns

An output stream to be printed.

6.34.6 Member Data Documentation

6.34.6.1 baseFolder

```
string Settings::baseFolder
```

A string containing the path for the base dir of data.

6.34.6.2 blackMask

```
Filter Settings::blackMask
```

Filter for black.

6.34.6.3 blueMask

```
Filter Settings::blueMask
```

Filter for blue.

6.34.6.4 calibrationFile

```
string Settings::calibrationFile
```

A string containing the name to the file containing the data for the calibration.

6.34.6.5 convexHullFile

```
string Settings::convexHullFile
```

AString containing the name to file containing the points of the elements in the arena.

6.34.6.6 greenMask

```
Filter Settings::greenMask
```

Filter for green.

6.34.6.7 intrinsicCalibrationFile

```
string Settings::intrinsicCalibrationFile
```

A string containing the name to the file containing the values of the matrix for the calibration.

6.34.6.8 kernelSide

```
int Settings::kernelSide
```

6.34.6.9 mapsFolder

```
string Settings::mapsFolder
```

A string containing the name for maps folder. No certainty is given about the form of this string.

6.34.6.10 mapsNames

```
Tuple<string> Settings::mapsNames
```

A Tuple containing the names of the maps. These are not paths but just names.

6.34.6.11 mapsUnNames

```
Tuple<string> Settings::mapsUnNames
```

A Tuple containing the names of the undistorted maps. These are not paths but just names.

6.34.6.12 redMask

```
Filter Settings::redMask
```

Filter for red.

6.34.6.13 robotMask

```
Filter Settings::robotMask
```

Filter for the triangle above the robot.

6.34.6.14 templates

```
Tuple<string> Settings::templates
```

A Tuple containing the names of the templates. These are not paths but just names.

6.34.6.15 templatesFolder

```
string Settings::templatesFolder
```

A String containing the name of the folder containing the number templates.

6.34.6.16 victimMask

```
Filter Settings::victimMask
```

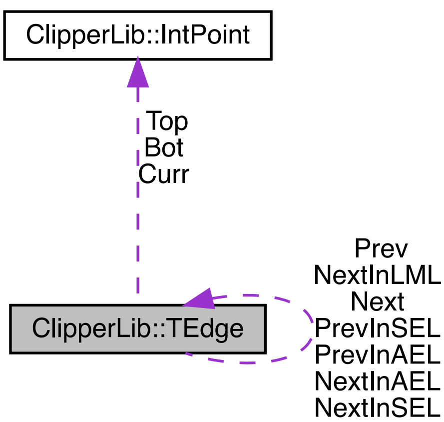
Filter for the victims.

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

- src/include/settings.hh
- src/settings.cc

6.35 ClipperLib::TEdge Struct Reference

Collaboration diagram for ClipperLib::TEdge:



Public Attributes

- IntPoint Bot
- IntPoint Curr
- IntPoint Top
- double Dx
- PolyType PolyTyp
- EdgeSide Side
- int WindDelta
- int WindCnt
- int WindCnt2
- int OutIdx
- TEdge * Next
- TEdge * Prev
- TEdge * NextInLML
- TEdge * NextInAEL
- TEdge * PrevInAEL
- TEdge * NextInSEL
- TEdge * PrevInSEL

6.35.1 Member Data Documentation

6.35.1.1 Bot

`IntPoint ClipperLib::TEdge::Bot`

6.35.1.2 Curr

`IntPoint ClipperLib::TEdge::Curr`

6.35.1.3 Dx

`double ClipperLib::TEdge::Dx`

6.35.1.4 Next

`TEdge* ClipperLib::TEdge::Next`

6.35.1.5 NextInAEL

`TEdge* ClipperLib::TEdge::NextInAEL`

6.35.1.6 NextInLML

`TEdge* ClipperLib::TEdge::NextInLML`

6.35.1.7 NextInSEL

`TEdge* ClipperLib::TEdge::NextInSEL`

6.35.1.8 OutIdx

`int ClipperLib::TEdge::OutIdx`

6.35.1.9 PolyTyp

`PolyType ClipperLib::TEdge::PolyTyp`

6.35.1.10 Prev

`TEdge* ClipperLib::TEdge::Prev`

6.35.1.11 PrevInAEL

`TEdge* ClipperLib::TEdge::PrevInAEL`

6.35.1.12 PrevInSEL

`TEdge* ClipperLib::TEdge::PrevInSEL`

6.35.1.13 Side

```
EdgeSide ClipperLib::TEdge::Side
```

6.35.1.14 Top

```
IntPoint ClipperLib::TEdge::Top
```

6.35.1.15 WindCnt

```
int ClipperLib::TEdge::WindCnt
```

6.35.1.16 WindCnt2

```
int ClipperLib::TEdge::WindCnt2
```

6.35.1.17 WindDelta

```
int ClipperLib::TEdge::WindDelta
```

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

- src/clipper.cc

6.36 Tuple< T > Class Template Reference

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

Public Member Functions

- Tuple ()

  Defualt constructor.
- Tuple (int _n,...)

  Constructors that takes the number of objectes to be stored, the objects and then stores them. For compatibility problem we strongly suggest to use this constructor only with standard types or types that can be promotted to one of the standard ones. For any other type we suggest to use an empty constructor and then use the `add()` function.
- Tuple (std::vector< T > v)

  Constructor that takes a vector with elements and stores it.
- int size () const
- T get (const int _n) const

  Gets the n-th element.
- Tuple< T > get (const uint start, const uint end)

  Function that returns a `Tuple` with elements.
- T front ()
- T back ()
- int find (T _el)

  A function that search for an element in the `Tuple` and returns it.
- void add (const T _new)

  Adds a value at the end of the list.
- void addIfNot (T _el, bool _throw=false)

  Adds a value. but only if it is not already present.
- bool remove (const uint pos)

  Removes a value from the list.
- bool remove_from (const uint pos)

  Removes all the element from a position onwards.
- void eraseAll ()

  Removes all values from the `Tuple`.
- 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.
- void ahead (const T _new)

  Function that adds an element at the head of the vector.
- Tuple< T > copy (const Tuple< T > &A)

  Copy a Tuple into another one.
- Tuple< T > operator= (const Tuple< T > &A)

  Overload of the = operator. Just calls `copy`.
- bool equal (Tuple< T > _t)

  Function that takes two `Tuple`s and verifies if they contain the same values.
- bool operator== (Tuple< T > _t)
- Tuple< T > sum (Tuple< T > t)
- Tuple< T > sum (T inc)

  Function to sum a value to all the elements in the `Tuple`.
- Tuple< T > operator+ (T inc)
- Tuple< T > & operator+= (T inc)
- Tuple< T > mul (Tuple< T > t)

  Function to multiply one by one the values from this to the values of a `Tuple`.
- Tuple< T > mul (T inc)

  Function to multiply a value to all the elements in the `Tuple`.
- Tuple< T > operator * (T inc)
- Tuple< T > & operator *= (T inc)

- 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. \tparan 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. \tparan 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. \tparan T1 The type of the elements in the second Tuple.
- stringstream to_string (string _prefix="") const
- string to_std_string () const

  Returns a standard string of the object.
- operator std::string () const

  Overload of operator std::string(). It simply calls the function `to_std_string()`.
- operator vector< T > () const

  Overload of cast to vector of same type.
- template<class T1 >
  operator vector< T1 > () const

  Overload of cast to vector of different type.
- T & operator[ ] (int index)

  Overloading [] operator to access elements in array style.
- tupleIter begin ()

  Iterator.
- tupleConstIter begin () const

  Const iterator.
- tupleIter end ()

  Iterator.
- tupleConstIter end () const

  Const iterator.

Friends

- ostream & operator<< (ostream &out, const Tuple< T > &data)

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

6.36.1 Detailed Description

template<class T>
class Tuple< T >

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

Template Parameters

T	The type of elements to be stored.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 Tuple() [1/3]

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

Defualt constructor.

6.36.2.2 Tuple() [2/3]

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

Constructors that takes the number of objectes to be stored, the objects and then stores them. For compatibility problem we strongly suggest to use this constructor only with standard types or types that can be promotted to one of the standard ones. For any other type we suggest to use an empty constructor and then use the `add()` function.

Parameters

in	_n	Number of obejctes to store.
in	...	Objects to store.

6.36.2.3 Tuple() [3/3]

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

Constructor that takes a vector with elements and stores it.

Parameters

in	*v*	The vector to store.

6.36.3 Member Function Documentation

6.36.3.1 add()

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

Adds a value at the end of the list.

Parameters

in	*_new*	The new value to be added.

6.36.3.2 addIfNot()

```
template<class T>
void Tuple< T >::addIfNot (
             T _el,
             bool _throw = false )  [inline]
```

Adds a value. but only if it is not already present.

Parameters

in	*_el*	The element to add.
in	*_throw*	If an exception can be thrown.

6.36.3.3 ahead()

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

Function that adds an element at the head of the vector.

Parameters

in	*_new*	The element to be added.

6.36.3.4 back()

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

Returns

The last element in the Tuple.

6.36.3.5 begin() [1/2]

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

Iterator.

Returns

the elements.begin() iterator.

6.36.3.6 begin() [2/2]

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

Const iterator.

Returns

the elements.begin() iterator.

6.36.3.7 copy()

```
template<class T>
Tuple<T> Tuple< T >::copy (
            const Tuple< T > & A )  [inline]
```

Copy a Tuple into another one.

Parameters

in	*A*	Tuple to be coppied.

Returns

this.

6.36.3.8 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. \tparan T1 The type of the elements in the second Tuple.

Parameters

in	*B*	The second Tuple to use for computing the distance.
in	*dist*	The type of distance to be computed.

Returns

The distance between the two Tuple.

6.36.3.9 end() [1/2]

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

Iterator.

Returns

the elements.end() iterator.

6.36.3.10 end() [2/2]

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

Const iterator.

Returns

the elements.begin() iterator.

6.36.3.11 equal()

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

Function that takes two `Tuple`s and verifies if they contain the same values.

Parameters

in	← _← t	The `Tuple` to compare.

Returns

`true` if the two `Tuple`s have the same element, `false` otherwise.

6.36.3.12 eraseAll()

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

Removes all values from the `Tuple`.

6.36.3.13 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. \tparan T1 The type of the elements in the second Tuple.

Parameters

in	*B*	the second Tuple to use for computing the distance.

Returns

The Euclidean distance between the two Tuple.

6.36.3.14 find()

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

A function that search for an element in the `Tuple` and returns it.

Parameters

in	↩ _↩ *el*	The element you are looking for.

Returns

The position of the element. -1 if no such element was found.

6.36.3.15 front()

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

Returns

The first element in the `Tuple`.

6.36.3.16 get() [1/2]

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

Gets the n-th element.

Parameters

in	↩ _↩ *n*	The position of the element to retrieve.

Returns

The element in the n-th position or an empty costructor if _n is greater then n or less than 0.

6.36.3.17 get() [2/2]

```
template<class T>
Tuple<T> Tuple< T >::get (
            const uint start,
            const uint end )  [inline]
```

Function that returns a `Tuple` with elements.

Parameters

in	*start*	The starting position
in	*end*	The ending position

Returns

A `Tuple` containing the element from the start-th position to the end-th.

6.36.3.18 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. \tparan T1 The type of the elements in the second Tuple.

Parameters

in	*B*	the second Tuple to use for computing the distance.

Returns

The Manhattan distance between the two Tuple.

6.36.3.19 mul() [1/2]

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

Function to multiply one by one the values from this to the values of a `Tuple`.

Parameters

in	*inc*	The multiplier `Tuple`

Returns

A `Tuple` (this) containing the new values.

6.36.3.20 mul() [2/2]

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

Function to multiply a value to all the elements in the Tuple.

Parameters

in	*inc*	The multiplier

Returns

A Tuple (this) containing the new values.

6.36.3.21 operator *()

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

This function overload the operator *. It simply calls the mul() function with only a multiplier and not a Tuple.

Parameters

in	↩ _↩ *t*	The increment.

Returns

A Tuple (this) containing the new values.

6.36.3.22 operator *=()

```
template<class T>
Tuple<T>& Tuple< T >::operator *= (
            T inc )  [inline]
```

This function overload the operator *=. It simply calls the mul() function with only a multiplier and not a Tuple.

Parameters

in	← _← *t*	The increment.

Returns

A `Tuple` (this) containing the new values.

6.36.3.23 operator std::string()

```
template<class T>
Tuple< T >::operator std::string ( ) const  [inline]
```

Overload of operator std::string(). It simply calls the function `to_std_string()`.

6.36.3.24 operator vector< T >()

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

Overload of cast to vector of same type.

Returns

A vector containing the values of elements.

6.36.3.25 operator vector< T1 >()

```
template<class T>
template<class T1 >
Tuple< T >::operator vector< T1 > ( ) const  [inline]
```

Overload of cast to vector of different type.

Template Parameters

Type	of vector to cast to.

Returns

A vector containing the values of elements.

6.36.3.26 operator+()

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

This function overload the operator +. It simply calls the `sum()` function.

Parameters

in	↩ _↩ *t*	The increment.

Returns

A `Tuple` (this) containing the new values.

6.36.3.27 operator+=()

```
template<class T>
Tuple<T>& Tuple< T >::operator+= (
            T inc )  [inline]
```

This function overload the operator +. It simply calls the `sum()` function.

Parameters

in	↩ _↩ *t*	The increment.

Returns

A `Tuple` (this) containing the new values.

6.36.3.28 operator=()

```
template<class T>
Tuple<T> Tuple< T >::operator= (
            const Tuple< T > & A )  [inline]
```

Overload of the = operator. Just calls `copy`.

Parameters

in	*A*	Tuple to be coppied.

Returns

this.

6.36.3.29 operator==()

```
template<class T>
bool Tuple< T >::operator== (
            Tuple< T > _t )  [inline]
```

This function overload the operator ==. It simply calls the `equal()` function.

Parameters

in	*_t*	The second `Tuple`.

Returns

`true` if the first `Tuple` (this) is equal to the second one, `false` otherwise.

6.36.3.30 operator[]()

```
template<class T>
T& Tuple< T >::operator[] (
            int index )  [inline]
```

Overloading [] operator to access elements in array style.

Parameters

in	*index*	Id of value to get.

Returns

Value at id position.

6.36.3.31 remove()

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

Removes a value from the list.

Parameters

in	*pos*	The position of the value to be removed.

Returns

`true` if verything went fine, `false` otherwise.

6.36.3.32 remove_from()

```
template<class T>
bool Tuple< T >::remove_from (
            const uint pos )  [inline]
```

Removes all the element from a position onwards.

Parameters

in	*pos*	The position from which to remove the elements.

Returns

`true` if the elements could be removed, `false` otherwise.

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

Returns

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

6.36.3.34 size()

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

Returns

The number of stored elements. -1 if the Tuple has a different number of elements.

6.36.3.35 sum() [1/2]

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

6.36.3.36 sum() [2/2]

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

Function to sum a value to all the elements in the `Tuple`.

Parameters

in	*inc*	The increment

Returns

A `Tuple` (this) containing the new values.

6.36.3.37 to_std_string()

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

Returns a standard string of the object.

Returns

Standard string of the object.

6.36.3.38 to_string()

```
template<class T>
stringstream Tuple< T >::to_string (
            string _prefix = "" ) const  [inline]
```

This function create a strinstream object containing the values of the Tuple.

Returns

A string stream.

6.36.4 Friends And Related Function Documentation

6.36.4.1 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	*out*	The output stream.
in	*data*	The Tuple to print.

Returns

An output stream to be printed.

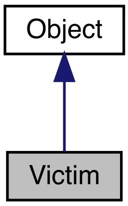
The documentation for this class was generated from the following file:

- src/include/maths.hh

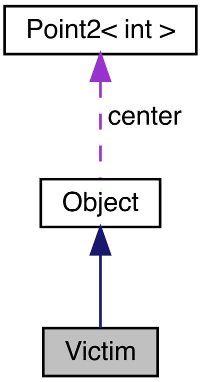
6.37 Victim Class Reference

```
#include <objects.hh>
```

Inheritance diagram for Victim:



Collaboration diagram for Victim:



Public Member Functions

- Victim (vector< Point2< int > > &vp, int _value)

  Constructor of the victim class and automatically compute center and radius.
- string toString ()

  Generate a string that describe the victim.
- void print ()

  Print the describing string of the victim.
- int getValue ()
- void setValue (int v)

Protected Attributes

- int value

6.37.1 Constructor & Destructor Documentation

6.37.1.1 Victim()

```
Victim::Victim (
            vector< Point2< int > > & vp,
            int _value )
```

Constructor of the victim class and automatically compute center and radius.

Parameters

in	*vp*	Vector of points that is the convex hull of the victim.
in	*_value*	The representative number of the victim.

Returns

Return the created victim.

6.37.2 Member Function Documentation

6.37.2.1 getValue()

```
int Victim::getValue ( ) [inline]
```

6.37.2.2 print()

```
void Victim::print ( )
```

Print the describing string of the victim.

6.37.2.3 setValue()

```
void Victim::setValue (
            int v ) [inline]
```

6.37.2.4 toString()

```
string Victim::toString ( )
```

Generate a string that describe the victim.

Returns

The generated string.

6.37.3 Member Data Documentation

6.37.3.1 value

```
int Victim::value [protected]
```

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

- src/include/objects.hh
- src/objects.cc

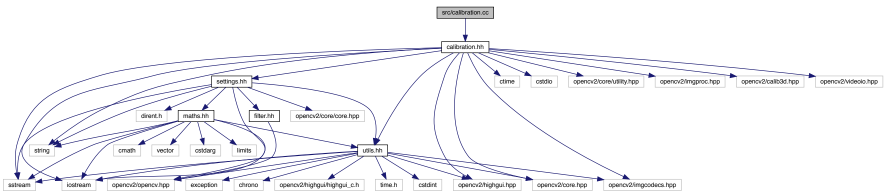
Chapter 7

File Documentation

7.1 src/calibration.cc File Reference

`#include "calibration.hh"`

Include dependency graph for calibration.cc:



Functions

- int calibration (string inputFile)

    Function to run the complete calibration.

- static void read (const FileNode &node, CalSettings &x, const CalSettings &default_value)

    Reads CalSettings from file. If there is none then initiate a new `CalSettings`.

- 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 (CalSettings &s, Size &imageSize, Mat &cameraMatrix, Mat &distCoeffs, vector< vector< Point2f > > imagePoints, vector< Mat > &rvecs, vector< Mat > &tvecs, vector< float > &reproj↩ Errs, double &totalAvgErr)

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

- static void saveCameraParams (const CalSettings &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 (CalSettings &s, Size imageSize, Mat &cameraMatrix, Mat &distCoeffs, vector< vector< Point2f > > imagePoints)

    Reads CalSettings from file. If there is none then initiate a new `CalSettings`.

7.1.1 Function Documentation

7.1.1.1 calcBoardCornerPositions()

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

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

Parameters

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

7.1.1.2 calibration()

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

Function to run the complete calibration.

Parameters

in	*inputFile*	Name of the setting.xml file. It's set to default to default.xml

Returns

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

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

Returns

The total error.

7.1.1.4 read()

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

Reads CalSettings from file. If there is none then initiate a new `CalSettings`.

Parameters

in	*node*	node to consider for getting CalSettings;
in	*x*	`CalSettings` to configure;
in	*default_value*	`CalSettings` default value. Setted to `CalSettings()`.

7.1.1.5 runCalibration()

```
static bool runCalibration (
        CalSettings & 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	*s*	The `CalSettings` read from the file and memorized.
in	*imageSize*	The size of the image used in `calibrateCamera()` to initialize the camera matrix.
in	*imagePoints*	The projected points for each image.
in	*reprojErrs*	The re-projection error, that is a geometric error corresponding to the image distance between a projected point and a measured one.
out	*cameraMatrix*	The matrix of the camera parameters
out	*distCoeffs*	The matrix of the distorsion coefficients.
out	*rvecs*	Output vector of rotation vectors estimated for each pattern view.
out	*tvecs*	Output vector of translation vectors estimated for each pattern view.
out	*totalAvgErr*	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.

7.1.1.6 runCalibrationAndSave()

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

Reads CalSettings from file. If there is none then initiate a new `CalSettings`.

Parameters

in	*s*	The `CalSettings` being used during the execution.
in	*imageSize*	The dimensions of the images.
in	*imagePoints*	The projected points for each image.
out	*cameraMatrix*	The matrix which is used to store the values for the camera parameters.
out	*distCoeffs*	The matrix which is used to store the distortion coefficients.

Returns

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

7.1.1.7 saveCameraParams()

```
static void saveCameraParams (
            const CalSettings & 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	*s*	Use the `CalSettings` got at the beginning for information as the output file name, image and board size.
in	*imageSize*	The size of the imgage.
in	*cameraMatrix*	The camera matrix.
in	*distCoeffs*	The distorsion coefficient matrix.
	[int]	rvecs Vector of rotation vectors estimated for each pattern view.
in	*tvecs*	Vector of translation vectors estimated for each pattern view.
in	*reprojErrs*	The re-projection error, that is a geometric error corresponding to the image distance between a projected point and a measured one.
in	*imagePoints*	The projected points for each image.
in	*totalAvgErr*	The total avarage error given from distorsion.

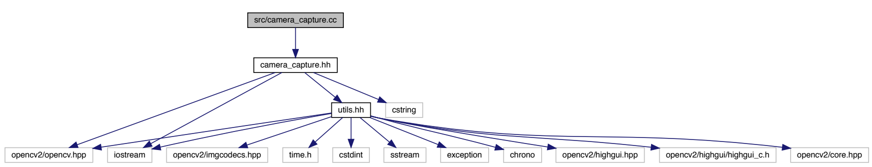
Open file for writing

Stores time of calibration

Store infos about the images

7.2 src/camera_capture.cc File Reference

`#include <camera_capture.hh>`
Include dependency graph for camera_capture.cc:



Macros

- #define SDEBUG(X) {}

7.2.1 Macro Definition Documentation

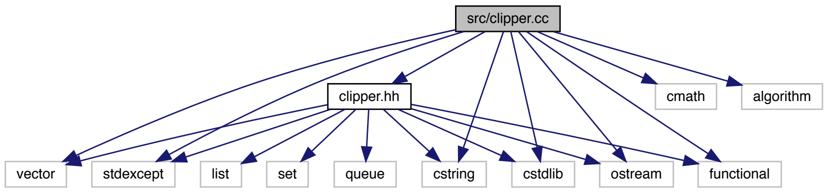
7.2.1.1 SDEBUG

```
#define SDEBUG(
            X ) {}
```

7.3 src/clipper.cc File Reference

```
#include "clipper.hh"
#include <cmath>
#include <vector>
#include <algorithm>
#include <stdexcept>
#include <cstring>
#include <cstdlib>
#include <ostream>
#include <functional>
```

Include dependency graph for clipper.cc:



Classes

- struct ClipperLib::TEdge
- struct ClipperLib::IntersectNode
- struct ClipperLib::LocalMinimum
- struct ClipperLib::OutRec
- struct ClipperLib::OutPt
- struct ClipperLib::Join
- struct ClipperLib::LocMinSorter
- class ClipperLib::Int128

Namespaces

- ClipperLib

Macros

- #define HORIZONTAL (-1.0E+40)
- #define TOLERANCE (1.0e-20)
- #define NEAR_ZERO(val) (((val) > -TOLERANCE) && ((val) < TOLERANCE))

Enumerations

- enum ClipperLib::Direction { ClipperLib::dRightToLeft, ClipperLib::dLeftToRight }
- enum ClipperLib::NodeType { ClipperLib::ntAny, ClipperLib::ntOpen, ClipperLib::ntClosed }

Functions

- cInt ClipperLib::Round (double val)
- cInt ClipperLib::Abs (cInt val)
- Int128 ClipperLib::Int128Mul (long64 lhs, long64 rhs)
- bool ClipperLib::Orientation (const Path &poly)
- double ClipperLib::Area (const Path &poly)
- double ClipperLib::Area (const OutPt ∗op)
- double ClipperLib::Area (const OutRec &outRec)
- bool ClipperLib::PointIsVertex (const IntPoint &Pt, OutPt ∗pp)
- int ClipperLib::PointInPolygon (const IntPoint &pt, const Path &path)
- int ClipperLib::PointInPolygon (const IntPoint &pt, OutPt ∗op)
- bool ClipperLib::Poly2ContainsPoly1 (OutPt ∗OutPt1, OutPt ∗OutPt2)
- bool ClipperLib::SlopesEqual (const TEdge &e1, const TEdge &e2, bool UseFullInt64Range)
- bool ClipperLib::SlopesEqual (const IntPoint pt1, const IntPoint pt2, const IntPoint pt3, bool UseFullInt64↩ Range)
- bool ClipperLib::SlopesEqual (const IntPoint pt1, const IntPoint pt2, const IntPoint pt3, const IntPoint pt4, bool UseFullInt64Range)
- bool ClipperLib::IsHorizontal (TEdge &e)
- double ClipperLib::GetDx (const IntPoint pt1, const IntPoint pt2)
- void ClipperLib::SetDx (TEdge &e)
- void ClipperLib::SwapSides (TEdge &Edge1, TEdge &Edge2)
- void ClipperLib::SwapPolyIndexes (TEdge &Edge1, TEdge &Edge2)
- cInt ClipperLib::TopX (TEdge &edge, const cInt currentY)
- void ClipperLib::IntersectPoint (TEdge &Edge1, TEdge &Edge2, IntPoint &ip)
- void ClipperLib::ReversePolyPtLinks (OutPt ∗pp)
- void ClipperLib::DisposeOutPts (OutPt ∗&pp)
- void ClipperLib::InitEdge (TEdge ∗e, TEdge ∗eNext, TEdge ∗ePrev, const IntPoint &Pt)
- void ClipperLib::InitEdge2 (TEdge &e, PolyType Pt)
- TEdge ∗ ClipperLib::RemoveEdge (TEdge ∗e)
- void ClipperLib::ReverseHorizontal (TEdge &e)
- void ClipperLib::SwapPoints (IntPoint &pt1, IntPoint &pt2)
- bool ClipperLib::GetOverlapSegment (IntPoint pt1a, IntPoint pt1b, IntPoint pt2a, IntPoint pt2b, IntPoint &pt1, IntPoint &pt2)
- bool ClipperLib::FirstIsBottomPt (const OutPt ∗btmPt1, const OutPt ∗btmPt2)
- OutPt ∗ ClipperLib::GetBottomPt (OutPt ∗pp)
- bool ClipperLib::Pt2IsBetweenPt1AndPt3 (const IntPoint pt1, const IntPoint pt2, const IntPoint pt3)
- bool ClipperLib::HorzSegmentsOverlap (cInt seg1a, cInt seg1b, cInt seg2a, cInt seg2b)
- void ClipperLib::RangeTest (const IntPoint &Pt, bool &useFullRange)
- TEdge ∗ ClipperLib::FindNextLocMin (TEdge ∗E)
- OutRec ∗ ClipperLib::GetLowermostRec (OutRec ∗outRec1, OutRec ∗outRec2)
- bool ClipperLib::OutRec1RightOfOutRec2 (OutRec ∗outRec1, OutRec ∗outRec2)

- bool ClipperLib::IsMinima (TEdge ∗e)
- bool ClipperLib::IsMaxima (TEdge ∗e, const cInt Y)
- bool ClipperLib::IsIntermediate (TEdge ∗e, const cInt Y)
- TEdge ∗ ClipperLib::GetMaximaPair (TEdge ∗e)
- TEdge ∗ ClipperLib::GetMaximaPairEx (TEdge ∗e)
- TEdge ∗ ClipperLib::GetNextInAEL (TEdge ∗e, Direction dir)
- void ClipperLib::GetHorzDirection (TEdge &HorzEdge, Direction &Dir, cInt &Left, cInt &Right)
- bool ClipperLib::IntersectListSort (IntersectNode ∗node1, IntersectNode ∗node2)
- bool ClipperLib::EdgesAdjacent (const IntersectNode &inode)
- int ClipperLib::PointCount (OutPt ∗Pts)
- void ClipperLib::SwapIntersectNodes (IntersectNode &int1, IntersectNode &int2)
- bool ClipperLib::E2InsertsBeforeE1 (TEdge &e1, TEdge &e2)
- bool ClipperLib::GetOverlap (const cInt a1, const cInt a2, const cInt b1, const cInt b2, cInt &Left, cInt &Right)
- void ClipperLib::UpdateOutPtIdxs (OutRec &outrec)
- OutPt ∗ ClipperLib::DupOutPt (OutPt ∗outPt, bool InsertAfter)
- bool ClipperLib::JoinHorz (OutPt ∗op1, OutPt ∗op1b, OutPt ∗op2, OutPt ∗op2b, const IntPoint Pt, bool DiscardLeft)
- static OutRec ∗ ClipperLib::ParseFirstLeft (OutRec ∗FirstLeft)
- DoublePoint ClipperLib::GetUnitNormal (const IntPoint &pt1, const IntPoint &pt2)
- void ClipperLib::ReversePath (Path &p)
- void ClipperLib::ReversePaths (Paths &p)
- void ClipperLib::SimplifyPolygon (const Path &in_poly, Paths &out_polys, PolyFillType fillType)
- void ClipperLib::SimplifyPolygons (const Paths &in_polys, Paths &out_polys, PolyFillType fillType)
- void ClipperLib::SimplifyPolygons (Paths &polys, PolyFillType fillType)
- double ClipperLib::DistanceSqrd (const IntPoint &pt1, const IntPoint &pt2)
- double ClipperLib::DistanceFromLineSqrd (const IntPoint &pt, const IntPoint &ln1, const IntPoint &ln2)
- bool ClipperLib::SlopesNearCollinear (const IntPoint &pt1, const IntPoint &pt2, const IntPoint &pt3, double distSqrd)
- bool ClipperLib::PointsAreClose (IntPoint pt1, IntPoint pt2, double distSqrd)
- OutPt ∗ ClipperLib::ExcludeOp (OutPt ∗op)
- void ClipperLib::CleanPolygon (const Path &in_poly, Path &out_poly, double distance)
- void ClipperLib::CleanPolygon (Path &poly, double distance)
- void ClipperLib::CleanPolygons (const Paths &in_polys, Paths &out_polys, double distance)
- void ClipperLib::CleanPolygons (Paths &polys, double distance)
- void ClipperLib::Minkowski (const Path &poly, const Path &path, Paths &solution, bool isSum, bool isClosed)
- void ClipperLib::MinkowskiSum (const Path &pattern, const Path &path, Paths &solution, bool pathIsClosed)
- void ClipperLib::TranslatePath (const Path &input, Path &output, const IntPoint delta)
- void ClipperLib::MinkowskiSum (const Path &pattern, const Paths &paths, Paths &solution, bool pathIs←Closed)
- void ClipperLib::MinkowskiDiff (const Path &poly1, const Path &poly2, Paths &solution)
- void ClipperLib::AddPolyNodeToPaths (const PolyNode &polynode, NodeType nodetype, Paths &paths)
- void ClipperLib::PolyTreeToPaths (const PolyTree &polytree, Paths &paths)
- void ClipperLib::ClosedPathsFromPolyTree (const PolyTree &polytree, Paths &paths)
- void ClipperLib::OpenPathsFromPolyTree (PolyTree &polytree, Paths &paths)
- std::ostream & ClipperLib::operator<< (std::ostream &s, const IntPoint &p)
- std::ostream & ClipperLib::operator<< (std::ostream &s, const Path &p)
- std::ostream & ClipperLib::operator<< (std::ostream &s, const Paths &p)

Variables

- static double const ClipperLib::pi = 3.141592653589793238
- static double const ClipperLib::two_pi = pi ∗2
- static double const ClipperLib::def_arc_tolerance = 0.25
- static int const ClipperLib::Unassigned = -1
- static int const ClipperLib::Skip = -2

7.3.1 Macro Definition Documentation

7.3.1.1 HORIZONTAL

```
#define HORIZONTAL (-1.0E+40)
```

7.3.1.2 NEAR_ZERO

```
#define NEAR_ZERO(
            val ) (((val) > -TOLERANCE) && ((val) < TOLERANCE))
```

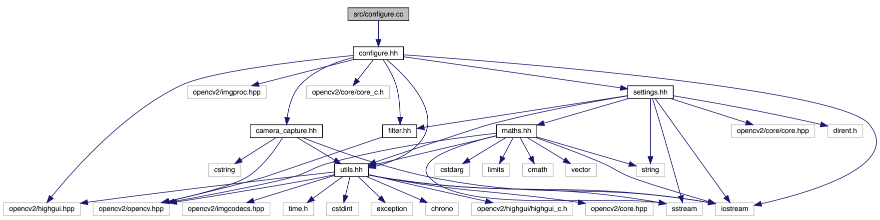
7.3.1.3 TOLERANCE

```
#define TOLERANCE (1.0e-20)
```

7.4 src/configure.cc File Reference

```
#include <configure.hh>
```

Include dependency graph for configure.cc:



Functions

- void on_low_h_thresh_trackbar (int, void ∗)
- void on_high_h_thresh_trackbar (int, void ∗)
- void on_low_s_thresh_trackbar (int, void ∗)
- void on_high_s_thresh_trackbar (int, void ∗)
- void on_low_v_thresh_trackbar (int, void ∗)
- void on_high_v_thresh_trackbar (int, void ∗)
- void update_trackers ()
- void configure (Mat &img, bool deploy, int img_id)

    It acqire a frame from the default camera of the pc.
- bool show_all_conditions (const Mat &frame)

Variables

- Filter filter = Filter(30, 30, 30, 100, 100, 100)

7.4.1 Function Documentation

7.4.1.1 configure()

```
void configure (
            Mat & img,
            bool deploy,
            int img_id )
```

It acqire a frame from the default camera of the pc.

Parameters

in	*save*	If save, or not, the acquired image to a file.

Returns

The Mat of the acquired frame.

If DEPLOY is defined then takes a photo from the camera, shows tha various filters and asks if they are visually correct. If not then it allows to set the various filters through trackbars. If DEPLOY is not defined then it takes a map from the folder set in Settings and ask for visual confirmation.

7.4.1.2 on_high_h_thresh_trackbar()

```
void on_high_h_thresh_trackbar (
            int ,
            void * )
```

@function on_high_h_thresh_trackbar

7.4.1.3 on_high_s_thresh_trackbar()

```
void on_high_s_thresh_trackbar (
            int ,
            void * )
```

@function on_high_s_thresh_trackbar

7.4.1.4 on_high_v_thresh_trackbar()

```
void on_high_v_thresh_trackbar (
            int ,
            void * )
```

@function on_high_v_thresh_trackbar

7.4.1.5 on_low_h_thresh_trackbar()

```
void on_low_h_thresh_trackbar (
            int ,
            void * )
```

@function on_low_h_thresh_trackbar

7.4.1.6 on_low_s_thresh_trackbar()

```
void on_low_s_thresh_trackbar (
            int ,
            void * )
```

@function on_low_s_thresh_trackbar

7.4.1.7 on_low_v_thresh_trackbar()

```
void on_low_v_thresh_trackbar (
            int ,
            void * )
```

@function on_low_v_thresh_trackbar

7.4.1.8 show_all_conditions()

```
bool show_all_conditions (
            const Mat & frame )
```

Function to show a picture with various filters taken from Settings. It then asks for visual confirmation.

Parameters

frame	The image to show.

Returns

True if the filters are okay, false otherwise.

7.4.1.9 update_trackers()

```
void update_trackers ( )
```

Function to update trackers with filter

7.4.2 Variable Documentation

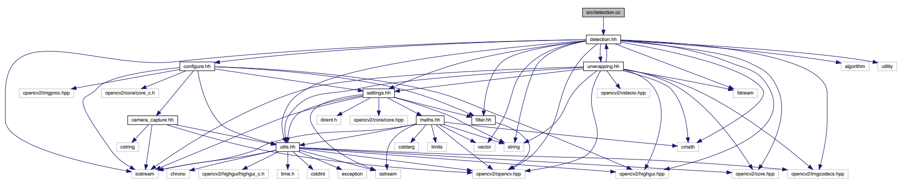
7.4.2.1 filter

```
Filter filter = Filter(30, 30, 30, 100, 100, 100)
```

7.5 src/detection.cc File Reference

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

Include dependency graph for detection.cc:



Macros

- #define EPS_CURVE 5

  Given an image, in black/white format, identify all the borders that delimit the shapes.
- #define MIN_AREA_SIZE 1000

Functions

- int detection (const bool _imgRead, const Mat ∗img)

  Loads some images and detects shapes according to different colors.
- void getConversionParameters (Mat &transf, const bool get)

  The function simply store the value of the given matrix and allow the access to it from different function location.
- Configuration2< double > localize (const Mat &img, const bool raw)

  Identify the location of the robot respect to the given image.
- void load_number_template ()

  Load some templates and save them in the global variable 'templates'.
- void shape_detection (const Mat &img, const COLOR_TYPE color)

  Detect shapes inside the image according to the variable 'color'.
- void erode_dilation (Mat &img, const COLOR_TYPE color)

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

- bool _compare (const pair< int, int > &a, const pair< int, int > &b)
- void find_contours (const Mat &img, const Mat &original, const COLOR_TYPE 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 COLOR_TYPE color)

  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

- vector< Mat > templates
- vector< Point > robotShape

  Identify the loation of the robot by acquiring the image from the default camera of the environment.

7.5.1 Macro Definition Documentation

7.5.1.1 EPS_CURVE

```
#define EPS_CURVE 5
```

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

Parameters

in	*img*	It is an image in HSV format at the base of the elaboration process.
out	*original*	It is the original source of 'img', it is used for showing the detected contours, in the victim number recognition.
in	*color*	It is the type of reference color.

7.5.1.2 MIN_AREA_SIZE

```
#define MIN_AREA_SIZE 1000
```

7.5.2 Function Documentation

7.5.2.1 _compare()

```
bool _compare (
            const pair< int, int > & a,
            const pair< int, int > & b )
```

7.5.2.2 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.

7.5.2.3 detection()

```
int detection (
            const bool _imgRead,
            const Mat * img )
```

Loads some images and detects shapes according to different colors.

Parameters

in	*_imgRead*	Boolean flag that says if load or not the image from file or as a function parameter. True=load from file.
in	*img*	The imgage that eventually is loaded from the function.

Returns

Return 0 if the function reach the end.

7.5.2.4 erode_dilation()

```
void erode_dilation (
            Mat & img,
            const COLOR_TYPE 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	*img*	Is the image on which the function apply the filtering.
in	*color*	It is the type of reference color. According to the color the filtering functions apply can change in the type and in the order.

7.5.2.5 find_contours()

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

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

Parameters

in	*img*	Is an image in HSV format at the base of the elaboration process.
out	*original*	It is the original source of 'img', it is used for showing the detected contours.
in	*color*	It is the type of reference color.

7.5.2.6 getConversionParameters()

```
void getConversionParameters (
            Mat & transf,
            const bool get )
```

The function simply store the value of the given matrix and allow the access to it from different function location.

The transformation matrix are computed in the unwrapping phase and taken from the localization.

Parameters

in	*transf*	It is the matrix that can be stored but also retrieved.
in	*get*	It is the flag that says if the given matrix need to be stored or retrieved.

7.5.2.7 load_number_template()

```
void load_number_template ( )
```

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

7.5.2.8 localize()

```
Configuration2<double> localize (
            const Mat & img,
            const bool raw )
```

Identify the location of the robot respect to the given image.

Identify the loation of the robot by acquiring the image from the default camera of the environment.

Parameters

in	*img*	It is the image where the robot need to be located.
in	*raw*	It is a boolean flag that says if the img is raw and need filters or not.

Returns

Configuration of the robot in this exactly moment, according to the image.

7.5.2.9 number_recognition()

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

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

Parameters

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

Returns

The number recognise, '-1' otherwise.

7.5.2.10 save_convex_hull()

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

Given some vector save it in a xml file.

Parameters

in	*contours*	Is a vector that is saved in a xml file.
in	*color*	It is the type of reference color, according to which the function decide if saved ('color==GREEN') or not ('otherwise') the vector 'victims'.

7.5.2.11 shape_detection()

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

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

Parameters

in	*img*	Image on which the research will done.
in	*color*	It is the type of reference color. These color identify the possible spectrum that the function search on the image.

7.5.3 Variable Documentation

7.5.3.1 robotShape

```
vector<Point> robotShape
```

Identify the loation of the robot by acquiring the image from the default camera of the environment.

Returns

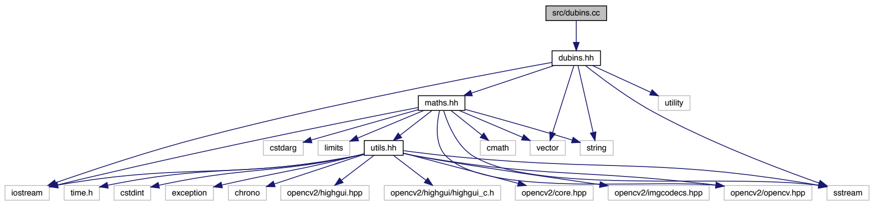
The configuration of the robot in this exactly moment.

7.5.3.2 templates

```
vector<Mat> templates
```

7.6 src/dubins.cc File Reference

`#include "dubins.hh"`
Include dependency graph for dubins.cc:



Functions

- Configuration2< double > circline (double _L, Configuration2< double > _P0, double _K)
- Tuple< Angle > toBase (Tuple< Angle > z, int n, int base, const Angle &inc, int startPos, int endPos)

    Convert a value in base 10 to base `base` in a `Tuple`. To each value an inc is muiltiplied and the initial `Angle` is added.
- void disp (Tuple< Tuple< Angle > > &t, Tuple< Angle > &z, int N, const Angle &inc, int startPos, int endPos)

    Compute the arrangements. Since each arrangement can be computed as n_{parts}, where each values is then multiplied for the increment and is added to the initial values.

7.6.1 Function Documentation

7.6.1.1 circline()

```
Configuration2<double> circline (
            double _L,
            Configuration2< double > _P0,
            double _K )
```

Computes an arrival point from an initial configuration through an arc of length _L and curvature _K.

Parameters

in	*_L*	The length of the arch.
in	*_P0*	The starting `Configuration2` of the arc.
in	*_K*	The curvature of the arc.

Returns

The ending `Configuration2` of the arc.

7.6.1.2 disp()

```
void disp (
            Tuple< Tuple< Angle > > & t,
            Tuple< Angle > & z,
            int N,
            const Angle & inc,
            int startPos = 0,
            int endPos = 0 )
```

Compute the arrangements. Since each arrangement can be computed as n_{parts}, where each values is then multiplied for the increment and is added to the initial values.

Parameters

out	*t*	A Tuple containing all the Tuples containing the Angles.
in	*z*	A Tuple containing all the initial Angles.
in	*N*	The number of iterations. Each iteration is going to be converted in base parts.
in	*inc*	The increment to give each initial Angle.
in	*startPos*	The initial position to consider in Tuple.
in	*endPos*	The final position to consider in Tuple.

7.6.1.3 toBase()

```
Tuple<Angle> toBase (
            Tuple< Angle > z,
            int n,
            int base,
            const Angle & inc,
            int startPos,
            int endPos )
```

Convert a value in base 10 to base `base` in a Tuple. To each value an inc is muiltiplied and the initial Angle is added.

Parameters

in	*z*	A Tuple containing all the initial Angles.
in	*n*	The value to be converted.
in	*base*	The base.
in	*inc*	The increment.
in	*startPos*	The starting position of the Tuple of Angles.
in	*endPos*	The ending position of the Tuple of Angles.

Returns

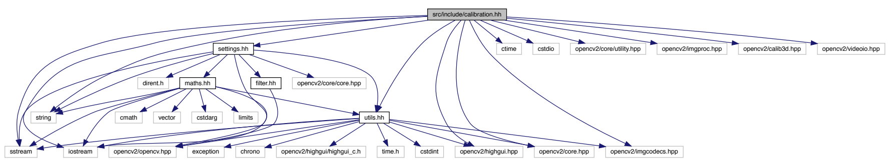
A vector containing the digits of the number converted to the specified base.

7.7 src/include/calibration.hh File Reference

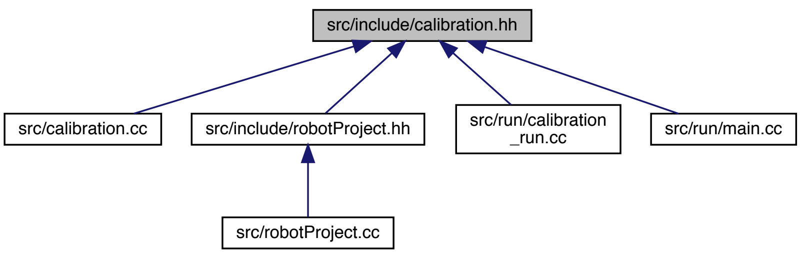
Library for calibration.

```
#include <utils.hh>
#include <settings.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 CalSettings

Enumerations

- enum { DETECTION = 0, CAPTURING = 1, CALIBRATED = 2 }

Functions

- int calibration (string inputFile="")

    Function to run the complete calibration.
- bool runCalibrationAndSave (CalSettings &s, Size imageSize, Mat &cameraMatrix, Mat &distCoeffs, vector< vector< Point2f > > imagePoints)

    Reads CalSettings from file. If there is none then initiate a new `CalSettings`.

Variables

- Settings * sett

7.7.1 Detailed Description

Library for calibration.

7.7.2 Enumeration Type Documentation

7.7.2.1 anonymous enum

`anonymous enum`

Enumerator

DETECTION	
CAPTURING	
CALIBRATED	

7.7.3 Function Documentation

7.7.3.1 calibration()

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

Function to run the complete calibration.

Parameters

in	*inputFile*	Name of the setting.xml file. It's set to default to default.xml

Returns

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

7.7.3.2 runCalibrationAndSave()

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

Reads CalSettings from file. If there is none then initiate a new `CalSettings`.

Parameters

in	*s*	The `CalSettings` being used during the execution.
in	*imageSize*	The dimensions of the images.
in	*imagePoints*	The projected points for each image.
out	*cameraMatrix*	The matrix which is used to store the values for the camera parameters.
out	*distCoeffs*	The matrix which is used to store the distortion coefficients.

Returns

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

7.7.4 Variable Documentation

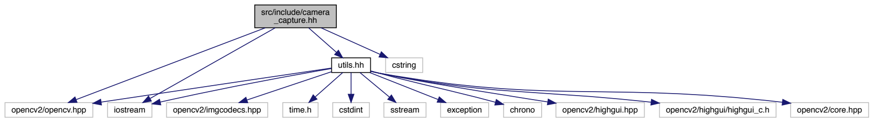
7.7.4.1 sett

```
Settings* sett
```

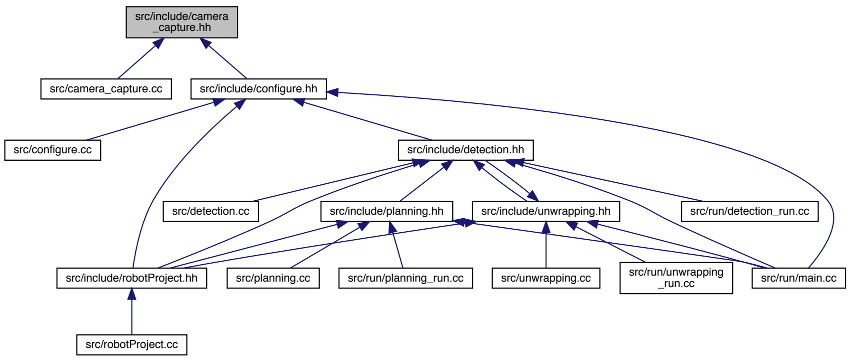
7.8 src/include/camera_capture.hh File Reference

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <cstring>
```

```
#include <utils.hh>
```
Include dependency graph for camera_capture.hh:



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



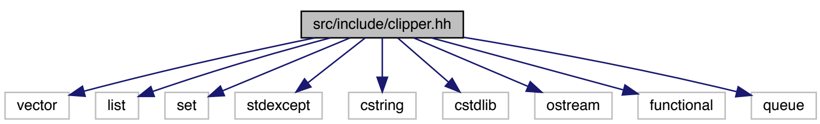
Classes

- class CameraCapture
- struct CameraCapture::input_options_t

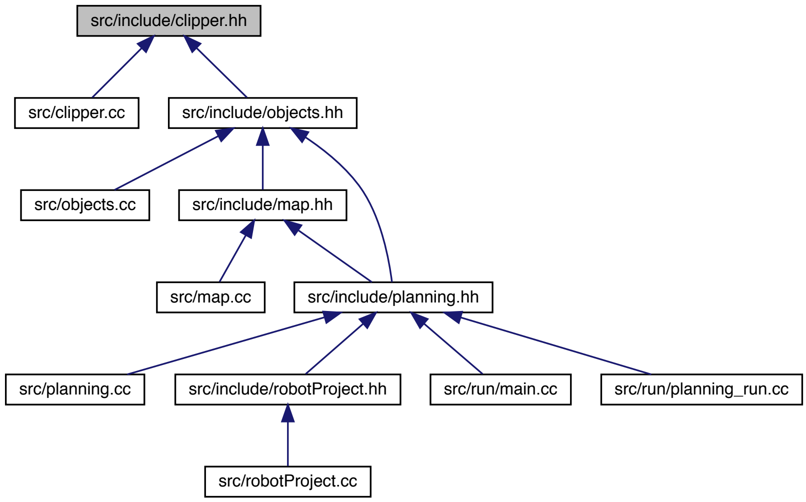
    Structure for store the input option for the class CameraCapture.

7.9 src/include/clipper.hh File Reference

```
#include <vector>
#include <list>
#include <set>
#include <stdexcept>
#include <cstring>
#include <cstdlib>
#include <ostream>
#include <functional>
#include <queue>
```
Include dependency graph for clipper.hh:



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



Classes

- struct ClipperLib::IntPoint
- struct ClipperLib::DoublePoint
- class ClipperLib::PolyNode
- class ClipperLib::PolyTree
- struct ClipperLib::IntRect
- class ClipperLib::ClipperBase
- class ClipperLib::Clipper
- class ClipperLib::ClipperOffset
- class ClipperLib::clipperException

Namespaces

- ClipperLib

Macros

- #define CLIPPER_VERSION "6.4.2"
- #define use_lines

Typedefs

- typedef signed long long ClipperLib::cInt
- typedef signed long long ClipperLib::long64
- typedef unsigned long long ClipperLib::ulong64
- typedef std::vector< IntPoint > ClipperLib::Path
- typedef std::vector< Path > ClipperLib::Paths
- typedef std::vector< PolyNode * > ClipperLib::PolyNodes
- typedef std::vector< OutRec * > ClipperLib::PolyOutList
- typedef std::vector< TEdge * > ClipperLib::EdgeList
- typedef std::vector< Join * > ClipperLib::JoinList
- typedef std::vector< IntersectNode * > ClipperLib::IntersectList

Enumerations

- enum ClipperLib::ClipType { ClipperLib::ctIntersection, ClipperLib::ctUnion, ClipperLib::ctDifference, ClipperLib::ctXor }
- enum ClipperLib::PolyType { ClipperLib::ptSubject, ClipperLib::ptClip }
- enum ClipperLib::PolyFillType { ClipperLib::pftEvenOdd, ClipperLib::pftNonZero, ClipperLib::pftPositive, ClipperLib::pftNegative }
- enum ClipperLib::InitOptions { ClipperLib::ioReverseSolution = 1, ClipperLib::ioStrictlySimple = 2, ClipperLib::ioPreserveCollinear = 4 }
- enum ClipperLib::JoinType { ClipperLib::jtSquare, ClipperLib::jtRound, ClipperLib::jtMiter }
- enum ClipperLib::EndType {
  ClipperLib::etClosedPolygon, ClipperLib::etClosedLine, ClipperLib::etOpenButt, ClipperLib::etOpenSquare, ClipperLib::etOpenRound }
- enum ClipperLib::EdgeSide { ClipperLib::esLeft = 1, ClipperLib::esRight = 2 }

Functions

- Path & ClipperLib::operator<< (Path &poly, const IntPoint &p)
- Paths & ClipperLib::operator<< (Paths &polys, const Path &p)
- std::ostream & ClipperLib::operator<< (std::ostream &s, const IntPoint &p)
- std::ostream & ClipperLib::operator<< (std::ostream &s, const Path &p)
- std::ostream & ClipperLib::operator<< (std::ostream &s, const Paths &p)
- bool ClipperLib::Orientation (const Path &poly)
- double ClipperLib::Area (const Path &poly)
- int ClipperLib::PointInPolygon (const IntPoint &pt, const Path &path)
- void ClipperLib::SimplifyPolygon (const Path &in_poly, Paths &out_polys, PolyFillType fillType)
- void ClipperLib::SimplifyPolygons (const Paths &in_polys, Paths &out_polys, PolyFillType fillType)
- void ClipperLib::SimplifyPolygons (Paths &polys, PolyFillType fillType)
- void ClipperLib::CleanPolygon (const Path &in_poly, Path &out_poly, double distance)
- void ClipperLib::CleanPolygon (Path &poly, double distance)
- void ClipperLib::CleanPolygons (const Paths &in_polys, Paths &out_polys, double distance)
- void ClipperLib::CleanPolygons (Paths &polys, double distance)
- void ClipperLib::MinkowskiSum (const Path &pattern, const Path &path, Paths &solution, bool pathIsClosed)
- void ClipperLib::MinkowskiSum (const Path &pattern, const Paths &paths, Paths &solution, bool pathIs↵ Closed)
- void ClipperLib::MinkowskiDiff (const Path &poly1, const Path &poly2, Paths &solution)
- void ClipperLib::PolyTreeToPaths (const PolyTree &polytree, Paths &paths)
- void ClipperLib::ClosedPathsFromPolyTree (const PolyTree &polytree, Paths &paths)
- void ClipperLib::OpenPathsFromPolyTree (PolyTree &polytree, Paths &paths)
- void ClipperLib::ReversePath (Path &p)
- void ClipperLib::ReversePaths (Paths &p)

Variables

- static cInt const ClipperLib::loRange = 0x3FFFFFFF
- static cInt const ClipperLib::hiRange = 0x3FFFFFFFFFFFFFFFLL

7.9.1 Macro Definition Documentation

7.9.1.1 CLIPPER_VERSION

```
#define CLIPPER_VERSION "6.4.2"
```

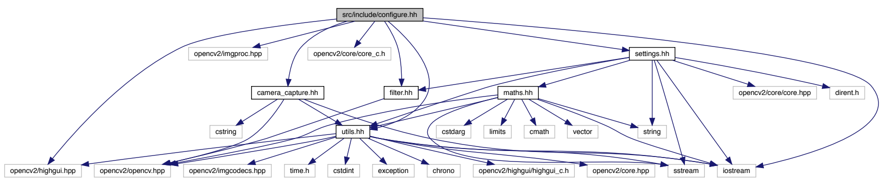
7.9.1.2 use_lines

```
#define use_lines
```

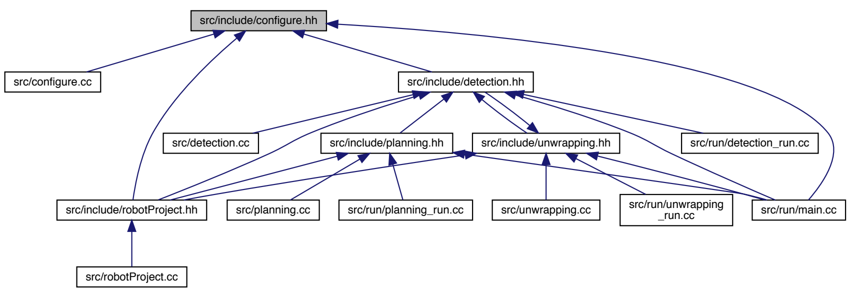
7.10 src/include/configure.hh File Reference

```
#include <iostream>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core/core_c.h>
#include <utils.hh>
#include <filter.hh>
#include <camera_capture.hh>
#include <settings.hh>
```

Include dependency graph for configure.hh:



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



Functions

- void configure (Mat &img, bool deploy=true, int img_id=0)

  It acqire a frame from the default camera of the pc.
- bool show_all_conditions (const Mat &frame)

Variables

- Settings ∗ sett

7.10.1 Function Documentation

7.10.1.1 configure()

```
void configure (
            Mat & img,
            bool deploy,
            int img_id )
```

It acqire a frame from the default camera of the pc.

Parameters

in	*save*	If save, or not, the acquired image to a file.

Returns

The Mat of the acquired frame.

If deploy is true then takes a photo from the camera, shows tha various filters and asks if they are visually correct. If not then it allows to set the various filters through trackbars. If deploy is false then it takes the imd_id-th maps from the folder set in Settings and ask for visual confirmation.

Parameters

in	*save*	If save, or not, the acquired image to a file.

Returns

The Mat of the acquired frame.

If DEPLOY is defined then takes a photo from the camera, shows tha various filters and asks if they are visually correct. If not then it allows to set the various filters through trackbars. If DEPLOY is not defined then it takes a map from the folder set in Settings and ask for visual confirmation.

7.10.1.2 show_all_conditions()

```
bool show_all_conditions (
            const Mat & frame )
```

Function to show a picture with various filters taken from Settings. It then asks for visual confirmation.

Parameters

frame	The image to show.

Returns

True if the filters are okay, false otherwise.

Function to show a picture with various filters taken from Settings. It then asks for visual confirmation.

Parameters

frame	The image to show.

Returns

True if the filters are okay, false otherwise.

7.10.2 Variable Documentation

7.10.2.1 sett

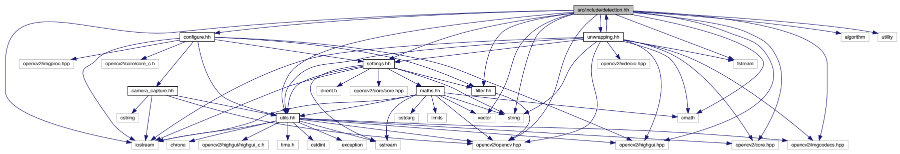
```
Settings* sett
```

7.11 src/include/detection.hh File Reference

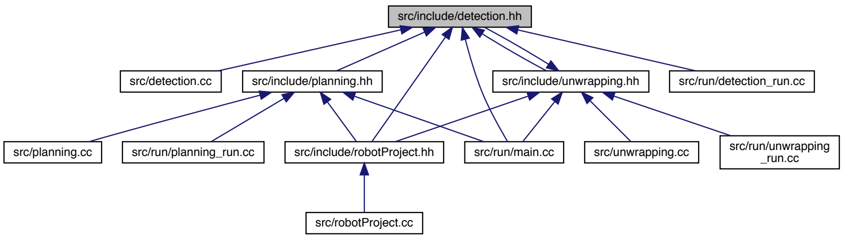
```
#include <utils.hh>
#include <settings.hh>
#include <filter.hh>
#include <configure.hh>
#include <unwrapping.hh>
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <algorithm>
#include <vector>
#include <utility>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#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:



Enumerations

- enum COLOR_TYPE {
  RED, GREEN, BLUE, CYAN,
  BLACK }

Functions

- int detection (const bool _imgRead=true, const Mat ∗img=nullptr)

  Loads some images and detects shapes according to different colors.
- void getConversionParameters (Mat &transf, const bool get=true)

  The function simply store the value of the given matrix and allow the access to it from different function location.
- Configuration2< double > localize (const Mat &img, const bool raw=true)

  Identify the loation of the robot by acquiring the image from the default camera of the environment.
- void shape_detection (const Mat &img, const COLOR_TYPE color)

  Detect shapes inside the image according to the variable 'color'.
- void erode_dilation (Mat &img, const COLOR_TYPE color)

  It apply some filtering function for isolate the subject and remove the noise.
- void find_contours (const Mat &img, const Mat &original, const COLOR_TYPE 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 COLOR_TYPE color)

  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.

7.11.1 Enumeration Type Documentation

7.11.1.1 COLOR_TYPE

```
enum COLOR_TYPE
```

Enumerator

RED	
GREEN	
BLUE	
CYAN	
BLACK	

7.11.2 Function Documentation

7.11.2.1 crop_number_section()

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

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

Parameters

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

7.11.2.2 detection()

```
int detection (
            const bool _imgRead,
            const Mat * img )
```

Loads some images and detects shapes according to different colors.

Parameters

in	*_imgRead*	Boolean flag that says if load or not the image from file or as a function parameter. True=load from file.
in	*img*	The imgage that eventually is loaded from the function.

Returns

Return 0 if the function reach the end.

7.11.2.3 erode_dilation()

```
void erode_dilation (
            Mat & img,
            const COLOR_TYPE 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	*img*	Is the image on which the function apply the filtering.
in	*color*	It is the type of reference color. According to the color the filtering functions apply can change in the type and in the order.

7.11.2.4 find_contours()

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

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

Parameters

in	*img*	Is an image in HSV format at the base of the elaboration process.
out	*original*	It is the original source of 'img', it is used for showing the detected contours.
in	*color*	It is the type of reference color.

7.11.2.5 getConversionParameters()

```
void getConversionParameters (
            Mat & transf,
            const bool get )
```

The function simply store the value of the given matrix and allow the access to it from different function location.

The transformation matrix are computed in the unwrapping phase and taken from the localization.

Parameters

in	*transf*	It is the matrix that can be stored but also retrieved.
in	*get*	It is the flag that says if the given matrix need to be stored or retrieved.

7.11.2.6 load_number_template()

```
void load_number_template ( )
```

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

7.11.2.7 localize()

```
Configuration2<double> localize (
            const Mat & img,
            const bool raw )
```

Identify the loation of the robot by acquiring the image from the default camera of the environment.

Returns

The configuration of the robot in this exactly moment.

Identify the location of the robot respect to the given image.

Parameters

in	*img*	It is the image where the robot need to be located.
in	*raw*	It is a boolean flag that says if the img is raw and need filters or not.

Returns

Configuration of the robot in this exactly moment, according to the image.

Identify the loation of the robot by acquiring the image from the default camera of the environment.

Parameters

in	*img*	It is the image where the robot need to be located.
in	*raw*	It is a boolean flag that says if the img is raw and need filters or not.

Returns

Configuration of the robot in this exactly moment, according to the image.

7.11.2.8 number_recognition()

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

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

Parameters

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

Returns

The number recognise, '-1' otherwise.

7.11.2.9 save_convex_hull()

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

Given some vector save it in a xml file.

Parameters

in	*contours*	Is a vector that is saved in a xml file.
in	*color*	It is the type of reference color, according to which the function decide if saved ('color==GREEN') or not ('otherwise') the vector 'victims'.

7.11.2.10 shape_detection()

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

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

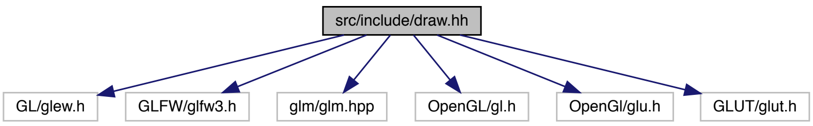
Parameters

in	*img*	Image on which the research will done.
in	*color*	It is the type of reference color. These color identify the possible spectrum that the function search on the image.

7.12 src/include/draw.hh File Reference

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <OpenGL/gl.h>
#include <OpenGl/glu.h>
#include <GLUT/glut.h>
```

Include dependency graph for draw.hh:



Namespaces

- DW

Typedefs

- typedef uint unsigned int

Functions

- void DW::init (x, y, GLfloat ∗vertices_buffer={0.0f})
- void DW::changeBuffer (GLfloat ∗vertices_buffer, uint dim)

Variables

- GLFWwindow ∗ DW::window
- GLuint DW::map_buffer

7.12.1 Typedef Documentation

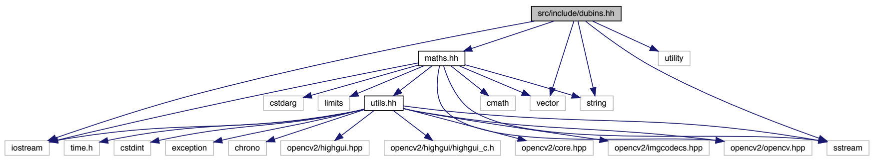
7.12.1.1 int

```
typedef uint unsigned int
```

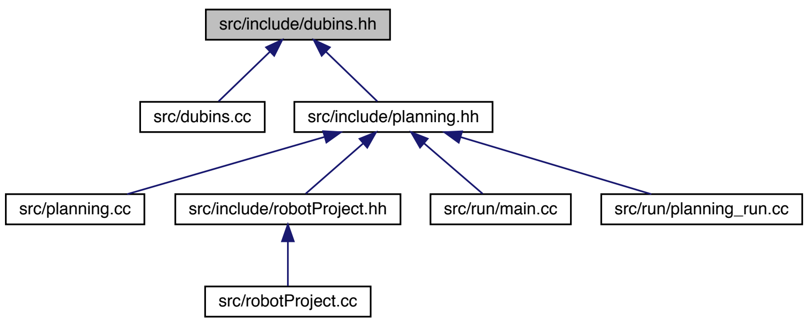
7.13 src/include/dubins.hh File Reference

```
#include <maths.hh>
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
#include <utility>
```

Include dependency graph for dubins.hh:



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



Classes

- class Curve< T >
- class DubinsArc< T1, T2 >

  Class to store a maneuver of Dubins. It inherits from `Curve`. Since each Dubins is formed of atmost 3 maneuvers, this class is meant to store one of this maneuver, which can be L, R or S respectively Left, Right, Straight.
- class Dubins< T >

  Class to store a Dubins curve. This class inherits from `Curve` and is composed of three `DubinsArc`.
- class DubinsSet< T >

  Given a set of point, compute the shortest set of Dubins that allows to go from start to end through all points.

Macros

- #define D_SHIFT 100
- #define PIECE_LENGTH 2
- #define PREC 100000
- #define KMAX 0.01

Functions

- double sinc (double t)
- Configuration2< double > circline (double _L, Configuration2< double > _P0, double _K)
- template<class T >
  bool is_on_circarc (Point2< T > p0, Point2< T > pi, Point2< T > pf, Point2< T > p)

  Function that computes a circle given 3 points and check if a point is on the circle arc.
- Tuple< Angle > toBase (Tuple< Angle > z, int n, int base, const Angle &inc, int startPos, int endPos)

  Convert a value in base 10 to base `base` in a `Tuple`. To each value an inc is muiltiplied and the initial `Angle` is added.
- void disp (Tuple< Tuple< Angle > > &t, Tuple< Angle > &z, int N, const Angle &inc, int startPos=0, int endPos=0)

  Compute the arrangements. Since each arrangement can be computed as n_{parts}, where each values is then multiplied for the increment and is added to the initial values.

7.13.1 Macro Definition Documentation

7.13.1.1 D_SHIFT

```
#define D_SHIFT 100
```

7.13.1.2 KMAX

```
#define KMAX 0.01
```

7.13.1.3 PIECE_LENGTH

```
#define PIECE_LENGTH 2
```

7.13.1.4 PREC

```
#define PREC 100000
```

7.13.2 Function Documentation

7.13.2.1 circline()

```
Configuration2<double> circline (
            double _L,
            Configuration2< double > _P0,
            double _K )
```

Computes an arrival point from an initial configuration through an arc of length _L and curvature _K.

Parameters

in	*_L*	The length of the arch.
in	*_P0*	The starting `Configuration2` of the arc.
in	*_K*	The curvature of the arc.

Returns

The ending `Configuration2` of the arc.

7.13.2.2 disp()

```
void disp (
            Tuple< Tuple< Angle > > & t,
            Tuple< Angle > & z,
            int N,
            const Angle & inc,
            int startPos = 0,
            int endPos = 0 )
```

Compute the arrangements. Since each arrangement can be computed as n_{parts}, where each values is then multiplied for the increment and is added to the initial values.

Parameters

out	*t*	A `Tuple` containing all the `Tuple`s containing the `Angle`s.
in	*z*	A `Tuple` containing all the initial `Angle`s.
in	*N*	The number of iterations. Each iteration is going to be converted in base parts.
in	*inc*	The increment to give each initial `Angle`.
in	*startPos*	The initial position to consider in `Tuple`.
in	*endPos*	The final position to consider in `Tuple`.

7.13.2.3 is_on_circarc()

```
template<class T >
bool is_on_circarc (
            Point2< T > p0,
            Point2< T > pi,
            Point2< T > pf,
            Point2< T > p )
```

Function that computes a circle given 3 points and check if a point is on the circle arc.

This function computes the 3 parameters a, b, c for a circle with equation $x^2 + y^2 + ax + by + c = 0$ using Cramer method. Then checks if the given point is on the circle: if it's not then returns `false`, otherwise checks the angle with respect to the initial point and the final point to see if the point is on the arc.

Template Parameters

T	The type of the point.

Parameters

p0	The initial point of the arc.
pi	An intermediate point of the arc.
pf	The final point of the arc.
p	The point to verify if it's on the arc or not.

Returns

`true` if the point is on the arc, `false` otherwise.

7.13.2.4 sinc()

```
double sinc (
            double t ) [inline]
```

Compute the sinc of the function defined as:

$$sinc(t) = \frac{sin(t)}{t} \quad t \neq 01 \quad t = 0$$

Parameters

in	*t*	The value of the angle to be used.

Returns

The result of the previous formula.

7.13.2.5 toBase()

```
Tuple<Angle> toBase (
            Tuple< Angle > z,
            int n,
            int base,
            const Angle & inc,
            int startPos,
            int endPos )
```

Convert a value in base 10 to base `base` in a `Tuple`. To each value an inc is muiltiplied and the initial `Angle` is added.

Parameters

in	*z*	A `Tuple` containing all the initial `Angle`s.
in	*n*	The value to be converted.
in	*base*	The base.
in	*inc*	The increment.
in	*startPos*	The starting position of the `Tuple` of `Angle`s.
in	*endPos*	The ending position of the `Tuple` of `Angle`s.

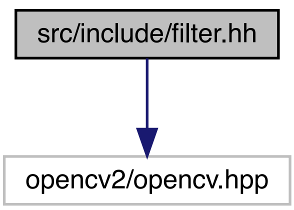
Returns

A vector containing the digits of the number converted to the specified base.

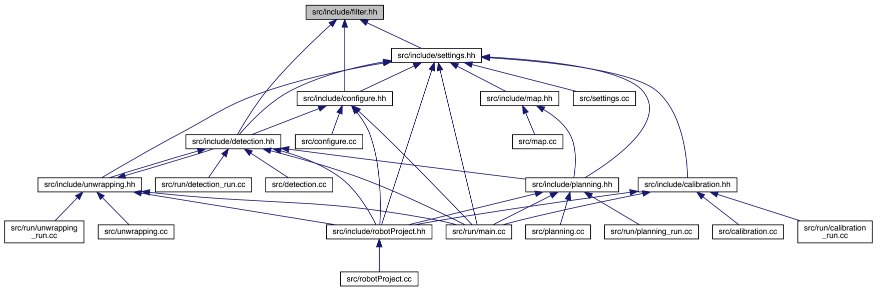
7.14 src/include/filter.hh File Reference

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

Include dependency graph for filter.hh:



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



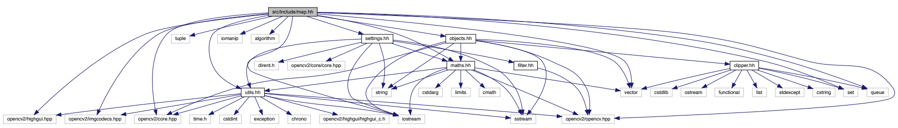
Classes

- class Filter

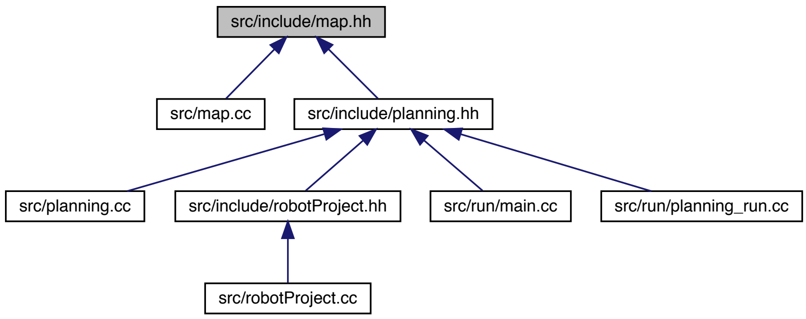
7.15 src/include/map.hh File Reference

```
#include <vector>
#include <set>
#include <queue>
#include <tuple>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <maths.hh>
#include <settings.hh>
#include <utils.hh>
#include <objects.hh>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
```

Include dependency graph for map.hh:



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



Classes

- class Mapp

Enumerations

- enum OBJ_TYPE {
  FREE, VICT, OBST, GATE,
  BODA, OUT_OF_MAP }

7.15.1 Enumeration Type Documentation

7.15.1.1 OBJ_TYPE

```
enum OBJ_TYPE
```

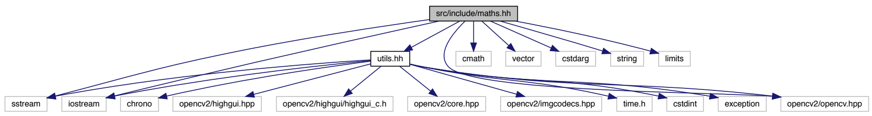
Enumerator

FREE	
VICT	
OBST	
GATE	
BODA	
OUT_OF_MAP	

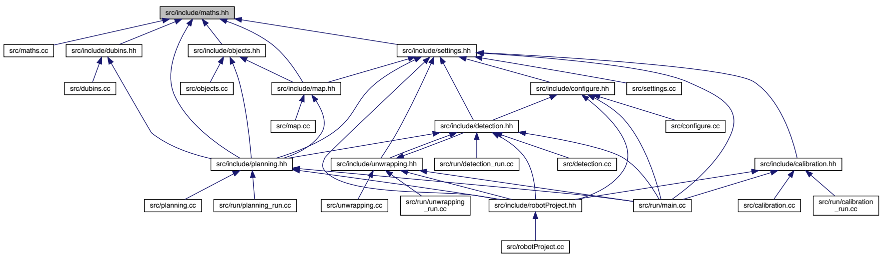
7.16 src/include/maths.hh File Reference

```
#include <utils.hh>
#include <iostream>
#include <cmath>
#include <vector>
#include <cstdarg>
#include <sstream>
#include <string>
#include <opencv2/opencv.hpp>
#include <limits>
```

Include dependency graph for maths.hh:



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 DInf numeric_limits<double>::infinity()
- #define Epsi numeric_limits<double>::epsilon()
- #define A_2PI Angle(6.2831853071-Epsi, Angle::RAD)

    Default Angle for 2pi rad.
- #define A_360 Angle(360.0-Epsi, Angle::DEG)

Default Angle for 360 degree.
- #define A_PI Angle(M_PI, Angle::RAD)

  Default Angle for pi rad.
- #define A_180 Angle(180, Angle::DEG)

  Defualt Angle for 180 degree.
- #define A_PI2 Angle(M_PI/2.0, Angle::RAD)

  Default Angle for pi/2 rad.
- #define A_90 Angle(90, Angle::DEG)

  Defualt Angle for 90 degree.
- #define A_DEG_NULL Angle(0, Angle::DEG)

  Default Angle for 0 rad.
- #define A_RAD_NULL Angle(0, Angle::RAD)

  Defualt Angle for 0 degree.
- #define tupleIter typename vector<T>::iterator
- #define tupleConstIter const typename vector<T>::iterator

Enumerations

- enum DISTANCE_TYPE { EUCLIDEAN, MANHATTAN }

Functions

- bool equal (const double &A, const double &B, const double E=Epsi)

  Function to compare two dubles as $|A - B| < \varepsilon$.
- template<class T >
  T pow2 (const T x)
- void invertAngle (Angle &a)

  Transform the angle given i the new reference system where x and y will be swapped.

Variables

- const double DEGTORAD =(M_PI/180.0)
- const double RADTODEG =(180.0/M_PI)

7.16.1 Macro Definition Documentation

7.16.1.1 A_180

```
#define A_180 Angle(180, Angle::DEG)
```

Defualt Angle for 180 degree.

7.16.1.2 A_2PI

```
#define A_2PI Angle(6.2831853071-Epsi, Angle::RAD)
```

Default Angle for 2pi rad.

7.16.1.3 A_360

```
#define A_360 Angle(360.0-Epsi, Angle::DEG)
```

Default Angle for 360 degree.

7.16.1.4 A_90

```
#define A_90 Angle(90, Angle::DEG)
```

Defualt Angle for 90 degree.

7.16.1.5 A_DEG_NULL

```
#define A_DEG_NULL Angle(0, Angle::DEG)
```

Default Angle for 0 rad.

7.16.1.6 A_PI

```
#define A_PI Angle(M_PI, Angle::RAD)
```

Default Angle for pi rad.

7.16.1.7 A_PI2

```
#define A_PI2 Angle(M_PI/2.0, Angle::RAD)
```

Default Angle for pi/2 rad.

7.16.1.8 A_RAD_NULL

```
#define A_RAD_NULL Angle(0, Angle::RAD)
```

Defualt Angle for 0 degree.

7.16.1.9 DInf

```
#define DInf numeric_limits<double>::infinity()
```

7.16.1.10 Epsi

```
#define Epsi numeric_limits<double>::epsilon()
```

7.16.1.11 tupleConstIter

```
#define tupleConstIter const typename vector<T>::iterator
```

7.16.1.12 tupleIter

```
#define tupleIter typename vector<T>::iterator
```

7.16.2 Enumeration Type Documentation

7.16.2.1 DISTANCE_TYPE

```
enum DISTANCE_TYPE
```

Enumerator

EUCLIDEAN	
MANHATTAN	

7.16.3 Function Documentation

7.16.3.1 equal()

```
bool equal (
            const double & A,
            const double & B,
            const double E = Epsi )  [inline]
```

Function to compare two dubles as $|A - B| < \varepsilon$.

Parameters

in	*A*	First number.
in	*B*	Second number.
in	*E*	ε, set at `std::numeric_limits<double>::epsilon()` as default.

Returns

`true` if $|A - B| < \varepsilon$, `false` otherwise.

7.16.3.2 invertAngle()

```
void invertAngle (
            Angle & a )
```

Transform the angle given i the new reference system where x and y will be swapped.

Parameters

[in/out]	a The angle that need to be inverted.

7.16.3.3 pow2()

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

7.16.4 Variable Documentation

7.16.4.1 DEGTORAD

```
const double DEGTORAD =(M_PI/180.0)
```

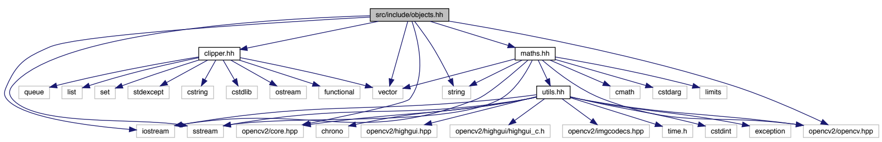
7.16.4.2 RADTODEG

```
const double RADTODEG =(180.0/M_PI)
```

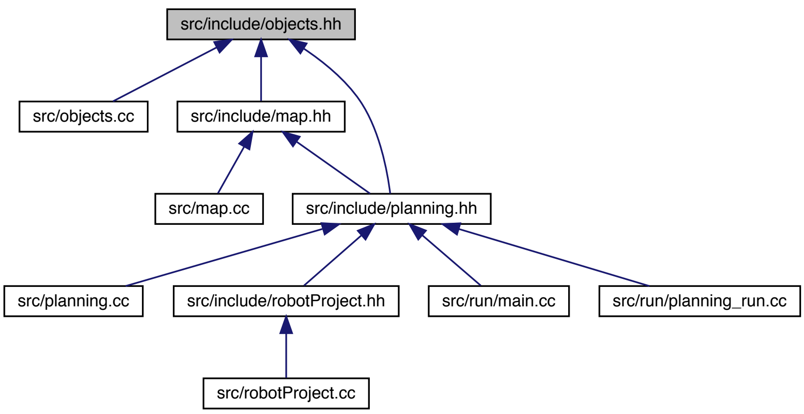
7.17 src/include/objects.hh File Reference

```
#include <iostream>
#include <vector>
#include <sstream>
#include <string>
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include "clipper.hh"
#include "maths.hh"
```

Include dependency graph for objects.hh:



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



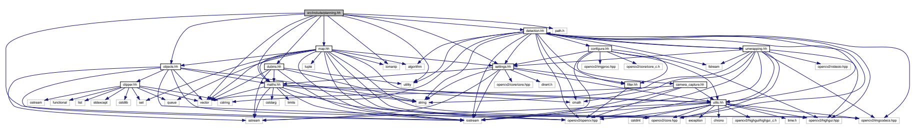
Classes

- class Object
- class Obstacle
- class Gate
- class Victim

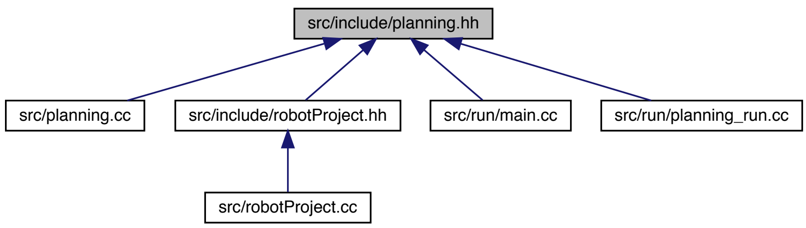
7.18 src/include/planning.hh File Reference

```
#include <iostream>
#include <vector>
#include <utility>
#include <iomanip>
#include <map.hh>
#include <maths.hh>
#include <settings.hh>
#include <objects.hh>
#include <detection.hh>
#include <dubins.hh>
#include "path.h"
```

Include dependency graph for planning.hh:



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



Namespaces

- Planning

Functions

- vector< Configuration2< double > > Planning::planning (const Mat &img)

    The function plan a route from the actual position of the robot up to the final gate through all the victims.
- void Planning::createMapp ()

    The goal is to load, all the neccessary data, from files and create a Mapp that store everything.
- vector< Point2< int > > Planning::convertToVP (const vector< vector< Point2< int > > > &arr)

    Convert a vector of vector of points into a vector of points (AKA collapse everything).
- vector< Point2< int > > Planning::convertToVP (const vector< vector< Configuration2< double > > > &arr)

    Convert a vector of vector of configurations into a vector of points (AKA collapse everything).
- vector< Configuration2< double > > Planning::convertToVC (const vector< vector< Configuration2< double > > > &arr)

    Convert a vector of vector of configurations into a vector of configurations (AKA collapse everything).
- vector< Configuration2< double > > Planning::convertToVC (const vector< vector< Point2< int > > > &arr)

    Convert a vector of vector of points into a vector of configurations (AKA collapse everything).
- void Planning::draw (const vector< vector< Point2< int > > > &vv, string name)

    Show in a window the representation of the map with the addition of the points and segment taken from the parameters.
- void Planning::draw (const vector< vector< Configuration2< double > > > &vv, string name)

    Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters.
- void Planning::draw (const vector< vector< Configuration2< double > > > &vv, const vector< Configuration2< double > > &left, const vector< Configuration2< double > > &right, string name)

    Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters. Plus a set of grey points (left vector) and black points (right vector).
- void Planning::loadVVP (vector< vector< Point2< int > > > &vvp, FileNode fn)

    The function load from the given fileNode a vector of vectors of Point2<int>.
- void Planning::loadVP (vector< Point2< int > > &vp, FileNode fn)

    The function load from the given fileNode a vector of Point2<int>.
- int ** Planning::allocateAAInt (const int a, const int b)

    Allocate a dynamic 2D array of int.
- int *** Planning::allocateAAAInt (const int a, const int b, const int c)

    Allocate a dynamic 3D array of int.
- int **** Planning::allocateAAAAInt (const int a, const int b, const int c, const int d)

    Allocate a dynamic 4D array of int.
- double ** Planning::allocateAADouble (const int a, const int b)

    Allocate a dynamic 2D array of int.
- double *** Planning::allocateAAADouble (const int a, const int b, const int c)

    Allocate a dynamic 3D array of double.
- double **** Planning::allocateAAAADouble (const int a, const int b, const int c, const int d)

    Allocate a dynamic 4D array of double.
- Point2< int > ** Planning::allocateAAPointInt (const int a, const int b)

    Allocate a dynamic 2D array of Points.
- template<class T >
  void Planning::deleteAA (T **arr, const int a)
- template<class T >
  void Planning::deleteAAA (T ***arr, const int a, const int b)
- template<class T >
  void Planning::deleteAAAA (T ****arr, const int a, const int b, const int c)
- vector< vector< Point2< int > > > Planning::minPathNPointsWithChoice (const vector< Point2< int > > &vp, const double bonus, const bool angle)

Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.

- vector< vector< Point2< int > > > Planning::minPathNPoints (const vector< Point2< int > > &vp, const bool angle)

  Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > Planning::minPathTwoPoints (const Point2< int > &p0, const Point2< int > &p1, const bool angle)

  Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > Planning::minPathTwoPointsInternal (const Point2< int > &startP, const Point2< int > &endP, double **distances, Point2< int > **parents)

  Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > Planning::minPathTwoPointsInternalAngles (const Point2< int > &startP, const Point2< int > &endP, double ***distances, int ****parents, const double initialDir)
- int Planning::angleSector (const double &d)

  Compute the sector of an angle.
- void Planning::intToVect (int c, vector< int > &v)

  Converts an integer into the vector of its digits. The result is inverse respect to the given integer.
- void Planning::resetDistanceMap (double **distances, const double value)

  It reset, to the given value, the matrix of distances given, to compute again the minPath search.
- void Planning::resetDistanceMap (double ***distances, const double value)

  It reset, to the given value, the matrix of distances given, to compute again the minPath search.
- vector< Point2< int > > Planning::sampleNPoints (const vector< vector< Point2< int > > > &vvp, const int n)

  It extracts from the given vector of vector of points, a subset of points that always contains the first one and the last one of each vector.
- vector< Point2< int > > Planning::sampleNPoints (const vector< Point2< int > > &points, const int n)

  It extracts from the given vector of points, a subset of points that always contains the first one and the last one.
- vector< Point2< int > > Planning::samplePointsEachNCells (const vector< Point2< int > > &points, const int step)

  It extracts from the given vector of points, a subset of points that always contains the first one and the last one.
- void Planning::fromVcToPath (vector< Configuration2< double > > &vc, Path &path)

  Convert a vector of point to a path, from Enrico's notation to Paolo's notation.
- int Planning::getNPoints ()

  Get the numper of points needed for the function sampleNpoints.
- void Planning::plan_dubins (const Configuration2< double > &_start, vector< vector< Configuration2< double > > > &vvConfs)

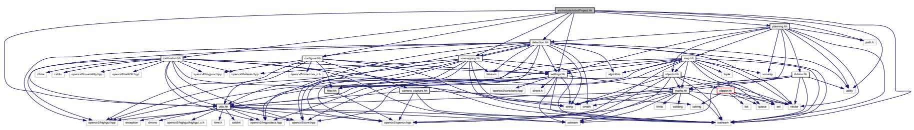
Variables

- static constexpr double Planning::baseDistance = -1.0
- static constexpr double Planning::baseDir = -1.0
- const int Planning::foundLimit = 20
- const int Planning::foundLimitAngles = 40
- static const int Planning::nPoints = 50
- constexpr double Planning::initialDistAllowed = 20.0

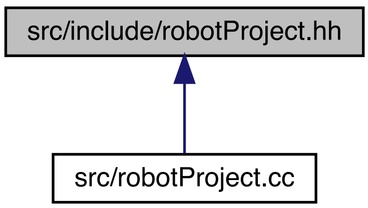
7.19 src/include/robotProject.hh File Reference

```
#include <utility>
#include <utils.hh>
#include <detection.hh>
#include <unwrapping.hh>
#include <calibration.hh>
#include <planning.hh>
#include <configure.hh>
#include <settings.hh>
#include <iostream>
#include "path.h"
```

Include dependency graph for robotProject.hh:



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



Classes

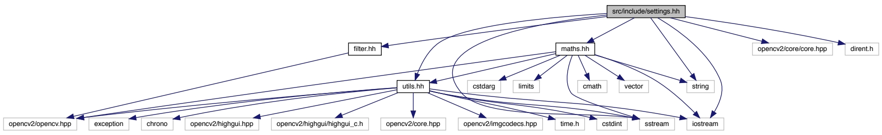
- class RobotProject

7.20 src/include/settings.hh File Reference

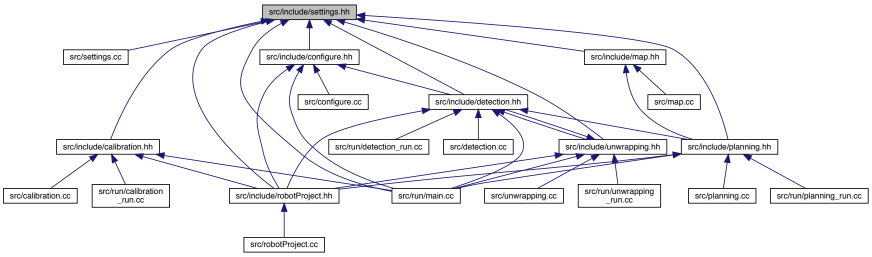
```
#include <filter.hh>
#include <maths.hh>
#include <utils.hh>
#include <opencv2/core/core.hpp>
#include <iostream>
#include <string>
#include <dirent.h>
```

`#include <sstream>`

Include dependency graph for settings.hh:



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



Classes

- class Settings

Variables

- Settings ∗ sett

    Global variable defined in main.cc.

7.20.1 Variable Documentation

7.20.1.1 sett

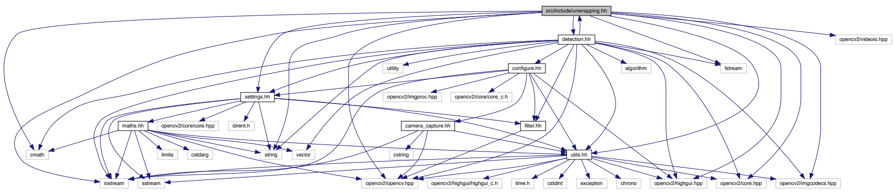
```
Settings* sett
```

Global variable defined in main.cc.

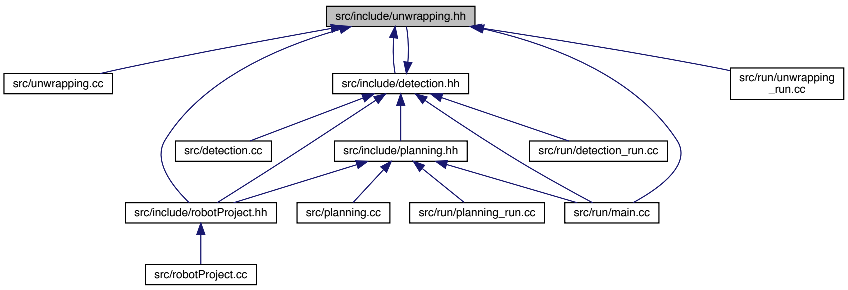
7.21 src/include/unwrapping.hh File Reference

```
#include <utils.hh>
#include <settings.hh>
#include <detection.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 (const bool _imgRead=true, Mat ∗img=nullptr)

  Take some images according to a xml and unwrap the black rectangle inside the image after appling undistortion trasformation.
- void createPointsHigh (const vector< Point > &rectLow, vector< Point > &rectHigh)

  Store in the given vector the white corners in the same order as the given black ones.
- void loadCoefficients (const string filename, Mat &camera_matrix, Mat &dist_coeffs)

  Load coefficients from a file.
- void find_rect (vector< Point > &_rect, const int &width, const int &height)

  Since the border of the arena might not always be clean but might have some imperfection, this functions computes the four vertixes taking all the points and computing the four that are the clostest to the corner of the image.

7.21.1 Function Documentation

7.21.1.1 createPointsHigh()

```
void createPointsHigh (
            const vector< Point > & rectLow,
            vector< Point > & rectHigh )
```

Store in the given vector the white corners in the same order as the given black ones.

Parameters

in	*rectLow*	A vector where the low corners of the rectangle (black markers position) are stored.
out	*rectHigh*	A vector where the high corners of the rectangle (white markers position) will be stored.

Store in the given vector the white corners in the same order as the given black ones.

Parameters

in	*rectLow*	The 4 points that define the black rectangle on the floor.
out	*rectHigh*	The vector that will contain the 4 points that will define the high corner that pass through the white points.

7.21.1.2 find_rect()

```
void find_rect (
            vector< Point > & _rect,
            const int & width,
            const int & height )
```

Since the border of the arena might not always be clean but might have some imperfection, this functions computes the four vertixes taking all the points and computing the four that are the clostest to the corner of the image.

Parameters

in	*_rect*	The vector of cv::Point to work on.
in	*width*	The width of the image.
in	*height*	The height of the image.

Since the border of the arena might not always be clean but might have some imperfection, this functions computes the four vertixes taking all the points and computing the four that are the clostest to the corner of the image.

Parameters

	[in/out]	_rect The vector containing the actual possible points that delimt the rectangle.

Parameters

in	*width*	The width of the original image from which the points of the rectangle come from.
in	*height*	The height of the original image from which the points of the rectangle come from.

7.21.1.3 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	*filename*	The string that identify the location of the xml file.
out	*camera_matrix*	Where the 'camera_matrix' matrix is saved.
out	*dist_coeffs*	Where the 'distortion_coefficients' matrix is saved.

7.21.1.4 unwrapping()

```
int unwrapping (
            const bool _imgRead,
            Mat * img )
```

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

Parameters

in	*_imgRead*	Boolean flag that says if load or not the image from file, or as a function parameter. In addition, also the return procedure change if true the image is saved on the disk otherwise is saved on the img function parameter. True=load and store on file.
	[in/out]	img The image that eventually is loaded from the function. And the one that will be modified for returning the elaborated frame.

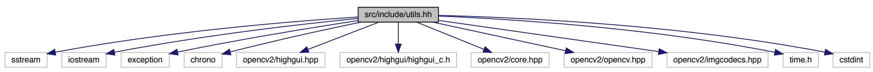
Returns

A 0 is return if the function reach the end.

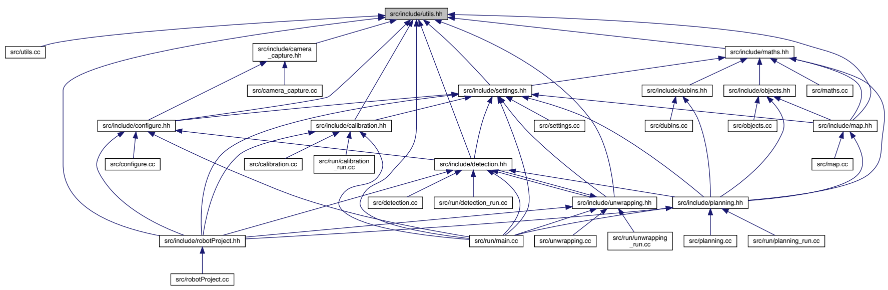
7.22 src/include/utils.hh File Reference

```
#include <sstream>
#include <iostream>
#include <exception>
#include <chrono>
#include <opencv2/highgui.hpp>
#include <opencv2/highgui/highgui_c.h>
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
#include <time.h>
#include <cstdint>
```

Include dependency graph for utils.hh:



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



Classes

- class MyException< T >

Namespaces

- CHRONO
- timeutils

Macros

- #define NAME(x) #x

    Returns the name of the variable.
- #define COUT(x)

    Print a messag to stderr.
- #define INFO(msg)

    Print the name of a variable and its content. Only if DEBUG is defined.

Typedefs

- typedef chrono::high_resolution_clock Clock

Enumerations

- enum CHRONO::TIME_TYPE { CHRONO::SEC, CHRONO::MSEC, CHRONO::MUSEC, CHRONO::NSEC }
- enum EXCEPTION_TYPE { GENERAL, EXISTS, SIZE }

Functions

- string CHRONO::getType (TIME_TYPE type, string ret="")
- double CHRONO::getElapsed (Clock::time_point start, Clock::time_point stop, TIME_TYPE type=MUSEC)
- string CHRONO::getElapsed (Clock::time_point start, Clock::time_point stop, string ret, TIME_TYPE type=MUSEC)
- void my_imshow (const char ∗win_name, Mat img, bool reset=false)

    Function to show images in an order grill.
- void mywaitkey (const char c='q')

    Function to use after my_imshow() for keeping the image opened until a key is pressed.
- void mywaitkey (string windowName)

    Function to use after my_imshow() for keeping the image opened until a key is pressed. When a key is pressed a specific window is closed.
- int64_t timeutils::timespecDiff (struct timespec ∗timeA_p, struct timespec ∗timeB_p)
- double timeutils::getTimeS ()

7.22.1 Macro Definition Documentation

7.22.1.1 COUT

```
#define COUT(
                x )
```

Print a messag to stderr.

7.22.1.2 INFO

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

Print the name of a variable and its content. Only if DEBUG is defined.

7.22.1.3 NAME

```
#define NAME(
            x ) #x
```

Returns the name of the variable.

7.22.2 Typedef Documentation

7.22.2.1 Clock

```
typedef chrono::high_resolution_clock Clock
```

7.22.3 Enumeration Type Documentation

7.22.3.1 EXCEPTION_TYPE

```
enum EXCEPTION_TYPE
```

Enumerator

GENERAL	
EXISTS	
SIZE	

7.22.4 Function Documentation

7.22.4.1 my_imshow()

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

Function to show images in an order grill.

Parameters

win_name	The name of the window to use.
img	The Mat containing the image.
reset	If true the image is going to be placed in 0,0 i.e. the top left corner of the screen.

7.22.4.2 mywaitkey() [1/2]

```
void mywaitkey (
            const char c )
```

Function to use after my_imshow() for keeping the image opened until a key is pressed.

7.22.4.3 mywaitkey() [2/2]

```
void mywaitkey (
            string windowName )
```

Function to use after my_imshow() for keeping the image opened until a key is pressed. When a key is pressed a specific window is closed.

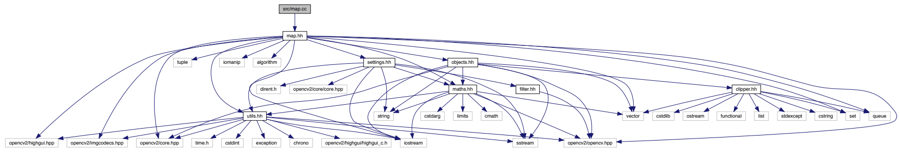
Parameters

windowName	The window to close after pressing a key.

7.23 src/map.cc File Reference

```
#include <map.hh>
```

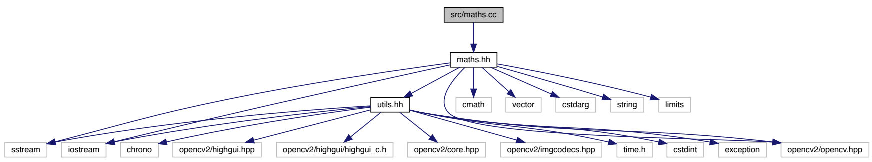
Include dependency graph for map.cc:



7.24 src/maths.cc File Reference

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

Include dependency graph for maths.cc:



Functions

- void invertAngle (Angle &a)

  Transform the angle given i the new reference system where x and y will be swapped.

7.24.1 Function Documentation

7.24.1.1 invertAngle()

```
void invertAngle (
            Angle & a )
```

Transform the angle given i the new reference system where x and y will be swapped.

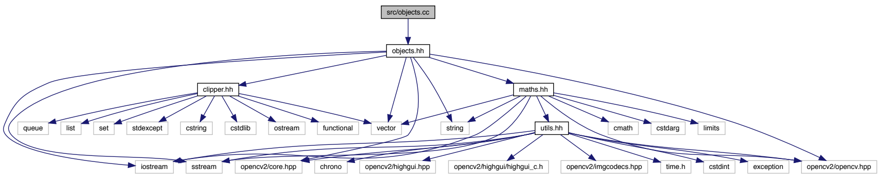
Parameters

[in/out]	a The angle that need to be inverted.

7.25 src/objects.cc File Reference

```
#include "objects.hh"
```

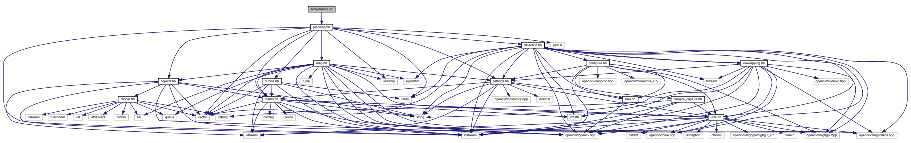
Include dependency graph for objects.cc:



7.26 src/planning.cc File Reference

`#include "planning.hh"`

Include dependency graph for planning.cc:



Namespaces

- Planning

Macros

- #define DELTA M_PI/180.0
- #define ROB_KMAX KMAX
- #define ROB_PIECE_LENGTH 20
- #define BONUS 5
- #define BEST
- #define SCALE 1000.0
- #define INCREASE 10
- #define SCRAP 3

Functions

- vector< Point2< int > > Planning::convertToVP (const vector< vector< Point2< int > > > &arr)

  Convert a vector of vector of points into a vector of points (AKA collapse everything).
- vector< Point2< int > > Planning::convertToVP (const vector< vector< Configuration2< double > > > &arr)

  Convert a vector of vector of configurations into a vector of points (AKA collapse everything).
- vector< Configuration2< double > > Planning::convertToVC (const vector< vector< Configuration2< double > > > &arr)

  Convert a vector of vector of configurations into a vector of configurations (AKA collapse everything).
- vector< Configuration2< double > > Planning::convertToVC (const vector< vector< Point2< int > > > &arr)

Convert a vector of vector of points into a vector of configurations (AKA collapse everything).
- void Planning::draw (const vector< vector< Point2< int > > > &vv, string name)

  Show in a window the representation of the map with the addition of the points and segment taken from the parameters.
- void Planning::draw (const vector< vector< Configuration2< double > > > &vv, string name)

  Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters.
- void Planning::draw (const vector< vector< Configuration2< double > > > &vv, const vector< Configuration2< double > > &left, const vector< Configuration2< double > > &right, string name)

  Show in a window the representation of the map with the addition of the configurations and segment taken from the parameters. Plus a set of grey points (left vector) and black points (right vector).
- vector< Configuration2< double > > Planning::planning (const Mat &img)

  The function plan a route from the actual position of the robot up to the final gate through all the victims.
- void Planning::createMapp ()

  The goal is to load, all the neccessary data, from files and create a Mapp that store everything.
- void Planning::loadVVP (vector< vector< Point2< int > > > &vvp, FileNode fn)

  The function load from the given fileNode a vector of vectors of Point2<int>.
- void Planning::loadVP (vector< Point2< int > > &vp, FileNode fn)

  The function load from the given fileNode a vector of Point2<int>.
- int Planning::getNPoints ()

  Get the numper of points needed for the function sampleNpoints.
- int ** Planning::allocateAAInt (const int a, const int b)

  Allocate a dynamic 2D array of int.
- int *** Planning::allocateAAAInt (const int a, const int b, const int c)

  Allocate a dynamic 3D array of int.
- int **** Planning::allocateAAAAInt (const int a, const int b, const int c, const int d)

  Allocate a dynamic 4D array of int.
- double ** Planning::allocateAADouble (const int a, const int b)

  Allocate a dynamic 2D array of int.
- double *** Planning::allocateAAADouble (const int a, const int b, const int c)

  Allocate a dynamic 3D array of double.
- double **** Planning::allocateAAAADouble (const int a, const int b, const int c, const int d)

  Allocate a dynamic 4D array of double.
- Point2< int > ** Planning::allocateAAPointInt (const int a, const int b)

  Allocate a dynamic 2D array of Points.
- vector< vector< Point2< int > > > Planning::minPathNPointsWithChoice (const vector< Point2< int > > &vp, const double bonus, const bool angle)

  Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< vector< Point2< int > > > Planning::minPathNPoints (const vector< Point2< int > > &vp, const bool angle)

  Given couples of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > Planning::minPathTwoPoints (const Point2< int > &p0, const Point2< int > &p1, const bool angle)

  Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- vector< Point2< int > > Planning::minPathTwoPointsInternal (const Point2< int > &startP, const Point2< int > &endP, double **distances, Point2< int > **parents)

  Given a couple of points the function compute the minimum path that connect them avoiding the intersection of OBST and BODA.
- int Planning::angleSector (const double &d)

  Compute the sector of an angle.

- vector< Point2< int > > Planning::minPathTwoPointsInternalAngles (const Point2< int > &startP, const Point2< int > &endP, double ***distances, int ****parents, const double initialDir)
- void Planning::intToVect (int c, vector< int > &v)

  Converts an integer into the vector of its digits. The result is inverse respect to the given integer.
- void Planning::resetDistanceMap (double **distances, const double value)

  It reset, to the given value, the matrix of distances given, to compute again the minPath search.
- void Planning::resetDistanceMap (double ***distances, const double value)

  It reset, to the given value, the matrix of distances given, to compute again the minPath search.
- vector< Point2< int > > Planning::sampleNPoints (const vector< vector< Point2< int > > > &vvp, const int n)

  It extracts from the given vector of vector of points, a subset of points that always contains the first one and the last one of each vector.
- vector< Point2< int > > Planning::sampleNPoints (const vector< Point2< int > > &points, const int n)

  It extracts from the given vector of points, a subset of points that always contains the first one and the last one.
- vector< Point2< int > > Planning::samplePointsEachNCells (const vector< Point2< int > > &points, const int step)

  It extracts from the given vector of points, a subset of points that always contains the first one and the last one.
- void Planning::fromVcToPath (vector< Configuration2< double > > &vc, Path &path)

  Convert a vector of point to a path, from Enrico's notation to Paolo's notation.
- template<class T >
  bool Planning::check_dubins_D (Dubins< T > &D)
- template<class T >
  bool Planning::check_dubins_DS (DubinsSet< T > &DS)
- bool Planning::compute_roundabout_dubins (DubinsSet< double > &new_DS, Configuration2< double > _start, const vector< Configuration2< double > > &vC, uint &vC_id, bool gate=false)
- template<class T >
  DubinsSet< double > Planning::victims_dubins (const vector< Configuration2< T > > &vC1, const vector< Configuration2< T > > &vC2, uint &vC1_pos, uint &vC2_pos)
- template<class T >
  DubinsSet< double > Planning::start_end_dubins (const Configuration2< double > &anchorPoint, const vector< Configuration2< T > > &vConfs, uint &pos, const bool start)
- vector< Configuration2< double > > Planning::vvCtovC (Tuple< Tuple< Configuration2< double > > > vv)
- vector< Configuration2< double > > Planning::vvvCtovC (Tuple< Tuple< Tuple< Configuration2< double > > > > vvv)
- Angle Planning::compute_final_angle (Configuration2< double > gate)
- void Planning::inter_victims (vector< vector< Configuration2< double > > > &vvConfs, vector< int > &vI, DubinsSet< double > &path, vector< DubinsSet< double > > &victimV)
- void Planning::plan_dubins (const Configuration2< double > &_start, vector< vector< Configuration2< double > > > &vvConfs)

Variables

- Mapp * Planning::map
- Configuration2< double > Planning::conf
- const double Planning::angleRange = 12*M_PI/180
- const int Planning::nAngles = 90
- const int Planning::range = 3
- const double Planning::DEGTORAD

7.26.1 Macro Definition Documentation

7.26.1.1 BEST

```
#define BEST
```

7.26.1.2 BONUS

```
#define BONUS 5
```

7.26.1.3 DELTA

```
#define DELTA M_PI/180.0
```

7.26.1.4 INCREASE

```
#define INCREASE 10
```

7.26.1.5 ROB_KMAX

```
#define ROB_KMAX KMAX
```

7.26.1.6 ROB_PIECE_LENGTH

```
#define ROB_PIECE_LENGTH 20
```

7.26.1.7 SCALE

```
#define SCALE 1000.0
```

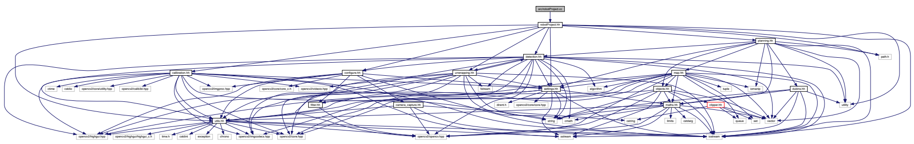
7.26.1.8 SCRAP

```
#define SCRAP 3
```

7.27 src/robotProject.cc File Reference

```
#include "robotProject.hh"
```

Include dependency graph for robotProject.cc:



Variables

- Settings ∗ sett =new Settings("./exam/data/")

  Global variable defined in main.cc.

7.27.1 Variable Documentation

7.27.1.1 sett

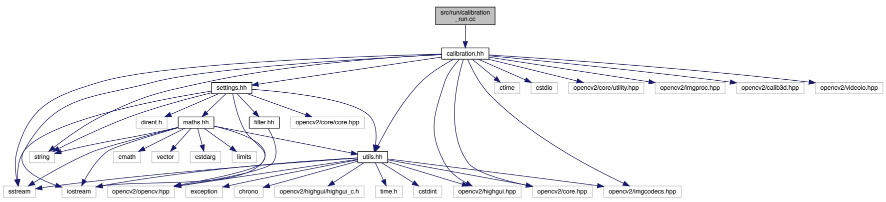
```
Settings* sett =new Settings("./exam/data/")
```

Global variable defined in main.cc.

7.28 src/run/calibration_run.cc File Reference

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

Include dependency graph for calibration_run.cc:



Functions

- int main ()

7.28.1 Function Documentation

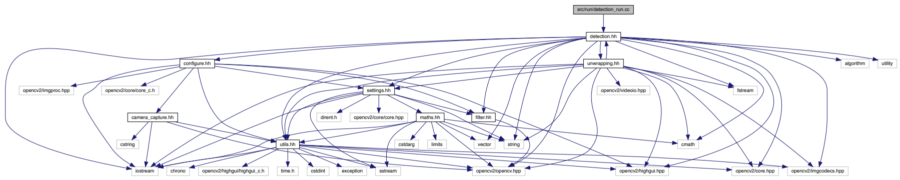
7.28.1.1 main()

```
int main ( )
```

7.29 src/run/detection_run.cc File Reference

```
#include <detection.hh>
```

Include dependency graph for detection_run.cc:



Functions

- int main ()

7.29.1 Function Documentation

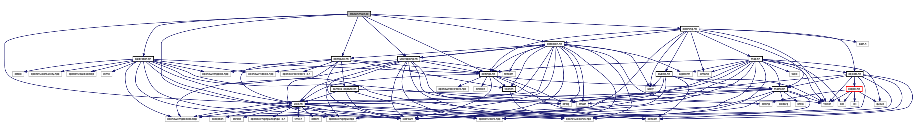
7.29.1.1 main()

```
int main ( )
```

7.30 src/run/main.cc File Reference

```
#include <utils.hh>
#include <detection.hh>
#include <unwrapping.hh>
#include <calibration.hh>
#include <planning.hh>
#include <configure.hh>
#include <settings.hh>
#include <iostream>
```

Include dependency graph for main.cc:



Functions

- int main ()

Variables

- Settings * sett =new Settings()

    Global variable defined in main.cc.

7.30.1 Function Documentation

7.30.1.1 main()

```
int main ( )
```

7.30.2 Variable Documentation

7.30.2.1 sett

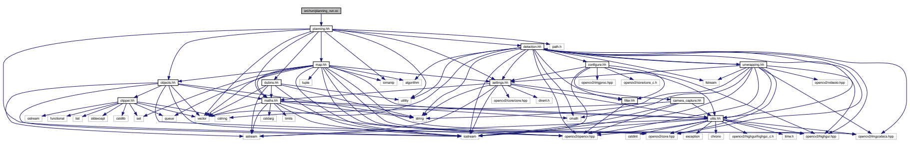
```
Settings* sett =new Settings()
```

Global variable defined in main.cc.

7.31 src/run/planning_run.cc File Reference

```
#include <planning.hh>
```

Include dependency graph for planning_run.cc:



Functions

- int main ()

7.31.1 Function Documentation

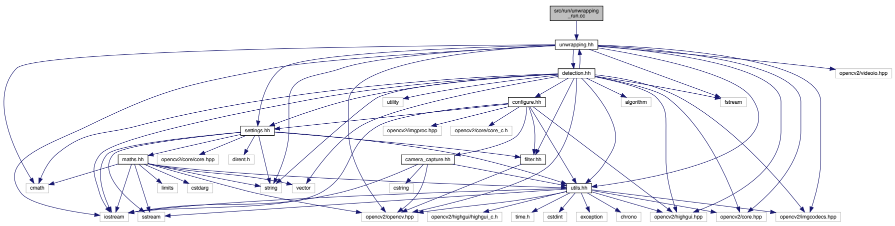
7.31.1.1 main()

```
int main ( )
```

7.32 src/run/unwrapping_run.cc File Reference

```
#include <unwrapping.hh>
```

Include dependency graph for unwrapping_run.cc:



Functions

- int main ()

7.32.1 Function Documentation

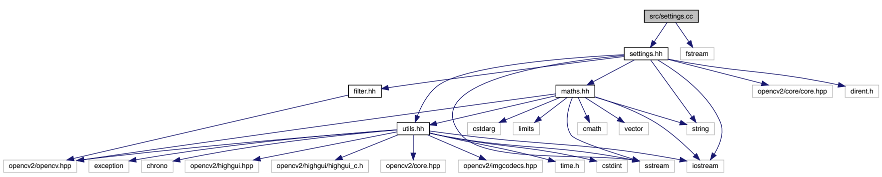
7.32.1.1 main()

```
int main ( )
```

7.33 src/settings.cc File Reference

```
#include "settings.hh"
#include <fstream>
```

Include dependency graph for settings.cc:



Macros

- #define NPOS string::npos

  Shortcut for string::npos.

Functions

- vector< string > getFiles (const string &path)

  Function to get all files in directory. From https://stackoverflow.com/questions/612097/how-can-i-get-the-li
- void vecToFile (FileStorage &fs, vector< int > x)

7.33.1 Macro Definition Documentation

7.33.1.1 NPOS

```
#define NPOS string::npos
```

Shortcut for string::npos.

7.33.2 Function Documentation

7.33.2.1 getFiles()

```
vector<string> getFiles (
            const string & path )
```

Function to get all files in directory. From https://stackoverflow.com/questions/612097/how-can-i-get-the

Parameters

Path	The path to check.

Returns

A vector containing the names of the files in the directory.

7.33.2.2 vecToFile()

```
void vecToFile (
            FileStorage & fs,
            vector< int > x ) [inline]
```

Writes a vector to a file.

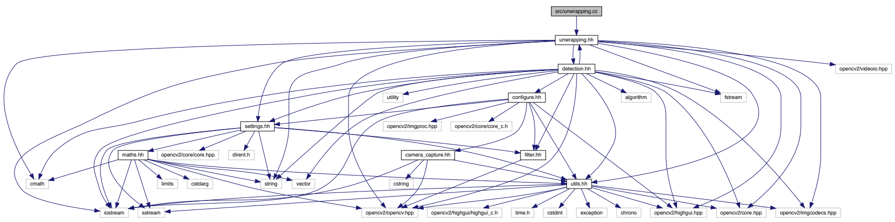
Parameters

fs	The FileStorage where to write the vector.
x	The vector to write.

7.34 src/unwrapping.cc File Reference

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

Include dependency graph for unwrapping.cc:



Macros

- #define AREA_RATIO 0.7
- #define AREA_MIN 500

Functions

- static double distance (Point c1, Point c2)

    Compute the euclidean distance.
- int unwrapping (const bool _imgRead, Mat ∗img)

    Take some images according to a xml and unwrap the black rectangle inside the image after appling undistortion trasformation.
- void createPointsHigh (const vector< Point > &rectLow, vector< Point > &rectHigh)

    Given the lower rectangle's points compute and retrieve the higher rectangle's points. \deteils The matching between the vectors is guarantee: the i elements of the vectors is refered to the same x,y point (with different z). At the moment the white points are constant and not taken from the image).
- void find_rect (vector< Point > &_rect, const int &width, const int &height)

    Compute the right vertexes from a vector of points that more or less define the rectangle.
- void loadCoefficients (const string filename, Mat &camera_matrix, Mat &dist_coeffs)

    Load coefficients from a file.

7.34.1 Macro Definition Documentation

7.34.1.1 AREA_MIN

```
#define AREA_MIN 500
```

7.34.1.2 AREA_RATIO

```
#define AREA_RATIO 0.7
```

7.34.2 Function Documentation

7.34.2.1 createPointsHigh()

```
void createPointsHigh (
            const vector< Point > & rectLow,
            vector< Point > & rectHigh )
```

Given the lower rectangle's points compute and retrieve the higher rectangle's points. \deteils The matching between the vectors is guarantee: the i elements of the vectors is refered to the same x,y point (with different z). At the moment the white points are constant and not taken from the image).

Store in the given vector the white corners in the same order as the given black ones.

Parameters

in	*rectLow*	The 4 points that define the black rectangle on the floor.
out	*rectHigh*	The vector that will contain the 4 points that will define the high corner that pass through the white points.

7.34.2.2 distance()

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

Compute the euclidean distance.

Parameters

in,out	*c1*	The first point.
in,out	*c2*	The second point.

Returns

The euclidean distance.

7.34.2.3 find_rect()

```
void find_rect (
            vector< Point > & _rect,
            const int & width,
            const int & height )
```

Compute the right vertexes from a vector of points that more or less define the rectangle.

Since the border of the arena might not always be clean but might have some imperfection, this functions computes the four vertixes taking all the points and computing the four that are the clostest to the corner of the image.

Parameters

	[in/out]	_rect The vector containing the actual possible points that delimt the rectangle.
in	*width*	The width of the original image from which the points of the rectangle come from.
in	*height*	The height of the original image from which the points of the rectangle come from.

7.34.2.4 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	*filename*	The string that identify the location of the xml file.
out	*camera_matrix*	Where the 'camera_matrix' matrix is saved.
out	*dist_coeffs*	Where the 'distortion_coefficients' matrix is saved.

7.34.2.5 unwrapping()

```
int unwrapping (
            const bool _imgRead,
            Mat * img )
```

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

Parameters

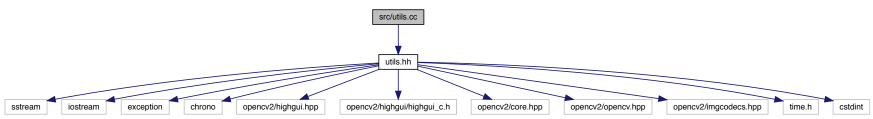
in	*_imgRead*	Boolean flag that says if load or not the image from file, or as a function parameter. In addition, also the return procedure change if true the image is saved on the disk otherwise is saved on the img function parameter. True=load and store on file.
	[in/out]	img The image that eventually is loaded from the function. And the one that will be modified for returning the elaborated frame.

Returns

A 0 is return if the function reach the end.

7.35 src/utils.cc File Reference

`#include "utils.hh"`
Include dependency graph for utils.cc:



Namespaces

- timeutils

Functions

- void my_imshow (const char ∗win_name, cv::Mat img, bool reset)

  Function to show images in an order grill.
- void mywaitkey (const char c)

  Function to use after my_imshow() for keeping the image opened until a key is pressed.
- void mywaitkey (string windowName)

  Function to use after my_imshow() for keeping the image opened until a key is pressed. When a key is pressed a specific window is closed.
- int64_t timeutils::timespecDiff (struct timespec ∗timeA_p, struct timespec ∗timeB_p)
- double timeutils::getTimeS ()

7.35.1 Function Documentation

7.35.1.1 my_imshow()

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

Function to show images in an order grill.

Parameters

win_name	The name of the window to use.
img	The Mat containing the image.
reset	If true the image is going to be placed in 0,0 i.e. the top left corner of the screen.

7.35.1.2 mywaitkey() [1/2]

```
void mywaitkey (
            const char c )
```

Function to use after my_imshow() for keeping the image opened until a key is pressed.

7.35.1.3 mywaitkey() [2/2]

```
void mywaitkey (
            string windowName )
```

Function to use after my_imshow() for keeping the image opened until a key is pressed. When a key is pressed a specific window is closed.

Parameters

windowName	The window to close after pressing a key.

Index