VLTIF

Generated by Doxygen 1.7.5.1

Tue Mar 6 2012 23:48:22

Contents

Chapter 1

Directory Hierarchy

1.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 2

Class 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

Directory Documentation

5.1 src/core/ Directory Reference

Files

- file Enumerations.cpp
- file Enumerations.h
- file GeometryUtilities.cpp
- file GeometryUtilities.h
- file MatUtilities.cpp
- file MatUtilities.h
- file Options.cpp
- file Options.h
- file Parser.cpp
- file Parser.h
- file PointTracker.cpp
- file PointTracker.h
- file VideoUtilities.cpp
- file VideoUtilities.h

5.2 src/feature/ Directory Reference

Files

- file FeaturePointUtilities.h
- file VehicleDetection.cpp
- file VehicleDetection.h

5.3 src/ Directory Reference

Directories

- directory core
- directory feature
- directory structures
- directory ui

Files

- file main.cpp

5.4 src/structures/ Directory Reference

Files

- file DetectorType.cpp
- file DetectorType.h
- file Lane.cpp
- file Lane.h
- file SiftType.cpp
- file SiftType.h
- file SurfType.cpp
- file SurfType.h
- file Vehicle.cpp
- file Vehicle.h

5.5 src/ui/ Directory Reference

Files

- file LaneDrawing.cpp
- file LaneDrawing.h
- file Mouse.cpp
- file Mouse.h

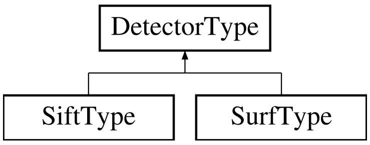
Chapter 6

Class Documentation

6.1 DetectorType Class Reference

```
#include <DetectorType.h>
```

Inheritance diagram for DetectorType:



Public Member Functions

- DetectorType ()
- DetectorType (const DetectorType &orig)
- virtual ∼DetectorType ()
- virtual string isType () const
- virtual void compute_frame (const Mat &img, vector< KeyPoint > &keypoints, vector< Lane >const &lanes) const =0
- virtual void reset_detector ()=0

6.1.1 Detailed Description

Definition at line 22 of file DetectorType.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 DetectorType::DetectorType ( )

Definition at line 10 of file DetectorType.cpp.

```
                              {
}
```

6.1.2.2 DetectorType::DetectorType (const DetectorType & *orig*)

Definition at line 13 of file DetectorType.cpp.

```
                                                         {
}
```

6.1.2.3 DetectorType::∼DetectorType ( ) [virtual]

Definition at line 16 of file DetectorType.cpp.

```
                               {
}
```

6.1.3 Member Function Documentation

6.1.3.1 virtual void DetectorType::compute_frame (const Mat & *img*, vector< KeyPoint > & *keypoints*, vector< Lane >const & *lanes*) const [pure virtual]

Implemented in SurfType, and SiftType.

Referenced by main().

6.1.3.2 string DetectorType::isType ( ) const [virtual]

Reimplemented in SiftType.

Definition at line 19 of file DetectorType.cpp.

```
                                     {
    return "DetectorType";
}
```

6.1.3.3 virtual void DetectorType::reset_detector ( ) [pure virtual]

Implemented in SurfType, and SiftType.

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

- src/structures/DetectorType.h
- src/structures/DetectorType.cpp

6.2 Enumerations Class Reference

```
#include <Enumerations.h>
```

Public Member Functions

- Enumerations ()
- Enumerations (const Enumerations &orig)
- virtual ∼Enumerations ()
- Vec3b Scalar2Vec (const Scalar &scal)

Static Public Member Functions

- static Scalar color_interp (Scalar const &cA, Scalar const &cB, const double val, const double addon=0)
- static string bool2string (const bool &val)

Public Attributes

- Scalar RED
- Scalar GREEN
- Scalar BLUE

6.2.1 Detailed Description

Definition at line 22 of file Enumerations.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Enumerations::Enumerations ( )

Definition at line 10 of file Enumerations.cpp.

References BLUE, GREEN, and RED.

```
                           {

    Scalar RED   = Scalar(0,0,255);
    Scalar GREEN = Scalar(0,255,0);
    Scalar BLUE  = Scalar(255,0,0);

}
```

6.2.2.2 Enumerations::Enumerations (const Enumerations & *orig*)

Definition at line 18 of file Enumerations.cpp.

```
                                                    {
}
```

6.2.2.3 Enumerations::∼Enumerations ( ) [virtual]

Definition at line 21 of file Enumerations.cpp.

```
                            {
}
```

6.2.3 Member Function Documentation

6.2.3.1 static string Enumerations::bool2string (const bool & *val*) [inline, static]

Convert a boolean value to string

Parameters

val	Boolean value

Returns

string equivalent

Definition at line 58 of file Enumerations.h.

Referenced by Parser::write_configuration().

```
                                                 {
        if( val == true)
            return "true";
        else
            return false;
    }
```

6.2.3.2 static Scalar Enumerations::color_interp (Scalar const & *cA,* Scalar const & *cB,* const double *val,* const double *addon* = 0) [inline, static]

Interpolate between two Scalar Colors

Parameters

cA	Color A
cB	Color B
val	Value to interpolate by
addon	- weight for translation (default = 0). This will add if $>$ 0.5, subtract if $<$ 0.5

Returns

The new color

Definition at line 42 of file Enumerations.h.

Referenced by compute_point_density(), and main().

```
                                 {

    double nval = val;
    if( nval > 0.5 )      nval = std::min( nval + addon, 1.0);
    else if( nval < 0.5 ) nval = std::max( nval - addon, 0.0);

    Scalar out( (1-nval)*cA[0] + nval*cB[0], (1-nval)*cA[1] + nval*cB[1], (
  1-nval)*cA[2] + nval*cB[2]);
    return out;
}
```

6.2.3.3 Vec3b Enumerations::Scalar2Vec (const Scalar & *scal*) [inline]

Definition at line 28 of file Enumerations.h.

Referenced by compute_point_density().

```
                                    {
    return Vec3b(scal[0], scal[1], scal[2]);
}
```

6.2.4 Member Data Documentation

6.2.4.1 Scalar Enumerations::BLUE

Definition at line 67 of file Enumerations.h.

Referenced by Enumerations().

6.2.4.2 Scalar Enumerations::GREEN

Definition at line 66 of file Enumerations.h.

Referenced by Enumerations().

6.2.4.3 Scalar Enumerations::RED

Definition at line 65 of file Enumerations.h.

Referenced by Enumerations().

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

- src/core/Enumerations.h
- src/core/Enumerations.cpp

6.3 Lane Class Reference

```
#include <Lane.h>
```

Public Member Functions

- Lane ()
- bool isInside (Point pt) const
- void draw (Mat &img) const
- void addVertex (Point pt)
- void changeLast (Point pt)
- size_t vertex_count () const
- const vector< Point > & getVertices () const
- bool finalize ()
- void clear ()
- size_t size ()
- void pop ()
- Rect bbox () const

Public Attributes

Drawing Parameters

- Scalar color
- int thickness
- int lineType
- int shift

Private Attributes

- vector< Point > vertices
- model::polygon < model::d2::point_xy< int > > poly
- Rect m_bbox

6.3.1 Detailed Description

Definition at line 32 of file Lane.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Lane::Lane ( )

Definition at line 10 of file Lane.cpp.

```
           :
   color(CV_RGB(50,255,50)),
   thickness(2),
   lineType(8),
   shift(0)
{}
```

6.3.3 Member Function Documentation

6.3.3.1 void Lane::addVertex (Point *pt*)

Definition at line 30 of file Lane.cpp.

References vertices.

Referenced by draw_lanes(), and load_lanes().

```
{
   vertices.push_back(pt);
}
```

6.3.3.2 Rect Lane::bbox ( ) const

Definition at line 113 of file Lane.cpp.

References m_bbox.

```
                    {
    return m_bbox;
}
```

6.3.3.3 void Lane::changeLast (Point *pt*)

Definition at line 35 of file Lane.cpp.

References vertices.

Referenced by draw_lanes().

```
{
   vertices.back() = pt;
}
```

6.3.3.4 void Lane::clear ( )

Definition at line 101 of file Lane.cpp.

References vertices.

Referenced by draw_lanes(), and load_lanes().

```
                {
    vertices.clear();
}
```

6.3.3.5 void Lane::draw (Mat & *img*) const

Definition at line 81 of file Lane.cpp.

References color, lineType, shift, size(), thickness, and vertices.

Referenced by draw_lanes().

```
{
   const int size = static_cast<int>(vertices.size());

   //create a pointer to point to
   const Point* pts_ptr = &(vertices[0]);

   polylines(
      img,
      &pts_ptr,
      &size,
      1,
      true,
      color,
      thickness,
      lineType,
      shift
   );
}
```

6.3.3.6 bool Lane::finalize ( )

Call when finished modifying

Definition at line 50 of file Lane.cpp.

References m_bbox, poly, and vertices.

```
{
    if ( vertices.size() < 3 ) return false;

    using boost::assign::tuple_list_of;
    using boost::make_tuple;
    using boost::geometry::append;

    Point minP = vertices[0], maxP = vertices[0];
    for( size_t i=1; i<vertices.size(); i++){
        if( vertices[i].x < minP.x ) minP.x = vertices[i].x;
        if( vertices[i].y < minP.y ) minP.y = vertices[i].y;
        if( vertices[i].x > maxP.x ) maxP.x = vertices[i].x;
        if( vertices[i].y > maxP.y ) maxP.y = vertices[i].y;
    }
    m_bbox = Rect(minP.x, minP.y, maxP.x-minP.x, maxP.y-minP.y);

    //int* points = &(vertices[0]);
    for ( vector<Point>::iterator i = vertices.begin(); i != vertices.end(); ++i
        )
       append(poly, tuple_list_of(i->x,i->y));
    append(poly, tuple_list_of(vertices[0].x,vertices[0].y));

    return true;
}
```

6.3.3.7 const vector< Point > & Lane::getVertices ( ) const

Definition at line 45 of file Lane.cpp.

References vertices.

```
{
    return vertices;
}
```

6.3.3.8 bool Lane::isInside (Point *pt*) const

Definition at line 75 of file Lane.cpp.

References poly.

```
{
    boost::tuple<int,int> p = boost::make_tuple(pt.x,pt.y);
    return within(p,poly);
}
```

6.3.3.9 void Lane::pop ( )

Definition at line 109 of file Lane.cpp.

References vertices.

```
              {
     vertices.pop_back();
}
```

6.3.3.10 size_t Lane::size ( )

Definition at line 105 of file Lane.cpp.

References vertices.

Referenced by draw(), and draw_lanes().

```
                 {
    return vertices.size();
}
```

6.3.3.11 size_t Lane::vertex_count ( ) const

Definition at line 40 of file Lane.cpp.

References vertices.

```
{
   return vertices.size();
}
```

6.3.4 Member Data Documentation

6.3.4.1 Scalar Lane::color

Use CV_RGB(r,g,b) macro where r,g,b are [0,255] for uchar images and [0.0, 1.0] for floating point images

Definition at line 58 of file Lane.h.

Referenced by draw().

6.3.4.2 int Lane::lineType

8 or 4 for 4-connected or 8-connected Bresenham algorithm or CV_AA for anti-aliased lines using Gaussian filtering

Definition at line 61 of file Lane.h.

Referenced by draw().

6.3.4.3 Rect Lane::m_bbox [private]

Definition at line 72 of file Lane.h.

Referenced by bbox(), and finalize().

6.3.4.4 model::polygon<model::d2::point_xy<int> > Lane::poly [private]

Definition at line 71 of file Lane.h.

Referenced by finalize(), and isInside().

6.3.4.5 int Lane::shift

Number of fractional bits to shift in the point coordinates (probably want to leave 0)

Definition at line 63 of file Lane.h.

Referenced by draw().

6.3.4.6 int Lane::thickness

In pixels

Definition at line 60 of file Lane.h.

Referenced by draw().

6.3.4.7 vector<Point> Lane::vertices [private]

Definition at line 70 of file Lane.h.

Referenced by addVertex(), changeLast(), clear(), draw(), finalize(), getVertices(), pop(), size(), and vertex_count().

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

- src/structures/Lane.h
- src/structures/Lane.cpp

6.4 Options Class Reference

```
#include <Options.h>
```

Public Member Functions

- Options ()
- Options (const Options &rhs)
- virtual ∼Options ()

Public Attributes

- size_t frame_rate

- size_t start_frame
- size_t frame_count
- size_t interest_point_max_life
- bool show
- string window_name
- string video_filename
- string video_output_filename
- string detector_type
- DetectorType $*$ detector
- VideoCapture cap
- VideoWriter vout
- VideoWriter bout
- VideoWriter dout
- Mat frame
- Mat black_frame
- Mat gray_frame
- Mat density_frame
- char key
- int stop_frame
- PointTracker pointHistory
- vector< Lane > lanes
- bool saveLanes
- bool loadLanes
- bool saveAvgFrame
- bool loadAvgFrame
- string laneFilename
- string avgFilename
- bool equalizeHistogram
- bool SIFT_lane_crop
- SURF_PARAMS surfParams
- SIFT::CommonParams siftCommonParams
- SIFT::DescriptorParams siftDescriptorParams
- SIFT::DetectorParams siftDetectorParams
- bool DEBUG
- int density_window

6.4.1 Detailed Description

Definition at line 26 of file Options.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Options::Options ( )

Definition at line 10 of file Options.cpp.

References show, and window_name.

```
                 {
    window_name = "Window";
    show = false;
}
```

6.4.2.2 Options::Options (const Options & *rhs*)

Definition at line 15 of file Options.cpp.

References window_name.

```
                                   {
    window_name = rhs.window_name;
}
```

6.4.2.3 Options::∼Options ( ) [virtual]

Definition at line 19 of file Options.cpp.

```
                  {
}
```

6.4.3 Member Data Documentation

6.4.3.1 string Options::avgFilename

Definition at line 66 of file Options.h.

Referenced by lane_manager(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.2 Mat Options::black_frame

Definition at line 51 of file Options.h.

Referenced by main().

6.4.3.3 VideoWriter Options::bout

Definition at line 47 of file Options.h.

Referenced by init(), and main().

6.4.3.4 VideoCapture Options::cap

Definition at line 45 of file Options.h.

Referenced by init(), and main().

6.4.3.5 bool Options::DEBUG

Definition at line 77 of file Options.h.

Referenced by init(), and main().

6.4.3.6 Mat Options::density_frame

Definition at line 53 of file Options.h.

Referenced by main().

6.4.3.7 int Options::density_window

Definition at line 79 of file Options.h.

Referenced by main(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.8 DetectorType∗ Options::detector

Definition at line 43 of file Options.h.

Referenced by main(), and Parser::parse_config_file().

6.4.3.9 string Options::detector_type

Definition at line 42 of file Options.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.10 VideoWriter Options::dout

Definition at line 48 of file Options.h.

Referenced by init(), and main().

6.4.3.11 bool Options::equalizeHistogram

Definition at line 68 of file Options.h.

Referenced by init(), and main().

6.4.3.12 Mat Options::frame

Definition at line 50 of file Options.h.

Referenced by main().

6.4.3.13 size_t Options::frame_count

Definition at line 34 of file Options.h.

Referenced by init(), and main().

6.4.3.14 size_t Options::frame_rate

Definition at line 32 of file Options.h.

Referenced by main(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.15 Mat Options::gray_frame

Definition at line 52 of file Options.h.

Referenced by main().

6.4.3.16 size_t Options::interest_point_max_life

Definition at line 36 of file Options.h.

Referenced by init(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.17 char Options::key

Definition at line 54 of file Options.h.

Referenced by main().

6.4.3.18 string Options::laneFilename

Definition at line 65 of file Options.h.

Referenced by lane_manager(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.19 vector< Lane > Options::lanes

Definition at line 60 of file Options.h.

Referenced by lane_manager(), and main().

6.4.3.20 bool Options::loadAvgFrame

Definition at line 64 of file Options.h.

Referenced by lane_manager(), Parser::parse_config_file(), and Parser::write_-configuration().

6.4.3.21 bool Options::loadLanes

Definition at line 62 of file Options.h.

Referenced by lane_manager(), Parser::parse_config_file(), and Parser::write_-configuration().

6.4.3.22 PointTracker Options::pointHistory

Definition at line 58 of file Options.h.

Referenced by init(), and main().

6.4.3.23 bool Options::saveAvgFrame

Definition at line 63 of file Options.h.

Referenced by lane_manager(), Parser::parse_config_file(), and Parser::write_-configuration().

6.4.3.24 bool Options::saveLanes

Definition at line 61 of file Options.h.

Referenced by lane_manager(), Parser::parse_config_file(), and Parser::write_-configuration().

6.4.3.25 bool Options::show

Definition at line 38 of file Options.h.

Referenced by init(), main(), Options(), Parser::parse_config_file(), and Parser::write_-configuration().

6.4.3.26 bool Options::SIFT_lane_crop

Definition at line 70 of file Options.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.27 SIFT::CommonParams Options::siftCommonParams

Definition at line 73 of file Options.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.28 SIFT::DescriptorParams Options::siftDescriptorParams

Definition at line 74 of file Options.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.29 SIFT::DetectorParams Options::siftDetectorParams

Definition at line 75 of file Options.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.30 size_t Options::start_frame

Definition at line 33 of file Options.h.

Referenced by init(), lane_manager(), main(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.31 int Options::stop_frame

Definition at line 56 of file Options.h.

Referenced by main(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.32 SURF_PARAMS Options::surfParams

Definition at line 72 of file Options.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.33 string Options::video_filename

Definition at line 40 of file Options.h.

Referenced by init(), lane_manager(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.34 string Options::video_output_filename

Definition at line 41 of file Options.h.

Referenced by init(), Parser::parse_config_file(), and Parser::write_configuration().

6.4.3.35 VideoWriter Options::vout

Definition at line 46 of file Options.h.

Referenced by init(), and main().

6.4.3.36 string Options::window_name

Definition at line 39 of file Options.h.

Referenced by init(), main(), and Options().

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

- src/core/Options.h
- src/core/Options.cpp

6.5 Parser Class Reference

```
#include <Parser.h>
```

Static Public Member Functions

- static void parse_config_file (int argc, char **argv, Options &options, const string &filename="data/options.cfg")
- static void write_configuration (Options &options)

6.5.1 Detailed Description

Definition at line 33 of file Parser.h.

6.5.2 Member Function Documentation

6.5.2.1 void Parser::parse_config_file (int *argc,* char ** *argv,* Options & *options,* const string & *filename* = "data/options.cfg") [static]

DISPLAY PARAMETERS

INTEREST POINT HISTORY PARAMETERS

SIFT PARAMETERS

SURF PARAMETERS

CORE INTEREST POINT PARAMETERS

Definition at line 12 of file Parser.cpp.

References Options::avgFilename, Options::density_window, Options::detector, Options::detector_type, SURF_PARAMS::extended, Options::frame_rate, SURF_PARAMS::hessianThreshold, Options::interest_point_max_life, Options::laneFilename, Options::loadAvgFrame, Options::loadLanes, SURF_PARAMS::numOctaveLayers, SURF_PARAMS::numOctaves, Options::saveAvgFrame, Options::saveLanes, Options::show, Options::SIFT_lane_crop, Options::siftCommonParams, Options::siftDescriptorParams, Options::siftDetectorParams, Options::start_frame, Options::stop_frame, Options::surfParams, SURF_PARAMS::upright, Options::video_filename, and Options::video_output_filename.

Referenced by main().

```
                    {


    //basic variables
    string default_config_filename = "";
    string show_video;
    string SIFT_lane_crop;
    int siftNumOctaves;
    int siftNumOctaveLayers;
    int siftFirstOctave;
    int siftAngleMode;
    bool SIFT_lc;
    string saveFrame, loadFrame, avgFilename, saveLanes, loadLanes,
      lanesFilename;
    int siftIsNormalize, siftRecalculateAngles;
    double siftMagnification, siftThreshold, siftEdgeThreshold;

    double surfHessianThreshold;
    int surfNumOctaves, surfNumOctaveLayers, surfExtended, surfUpright;


    /**********************/
    /* CREATE PARSERS */
    /**********************/
    //create parser for generic use, command-line only
    po::options_description generic("Allowed options");
    //create parser for config file and command line related
    po::options_description config_file("Configuration File");

    /****************************/
    /* ADD OPTIONS TO PARSERS     */
    /****************************/
    //construct generic options
    generic.add_options()
            ("help", "produce help message")
            ("config,c", po::value<string > (&default_config_filename)->
      default_value("data/options.cfg"),
            "name of the configuration file ( default is data/options.cfg )");

    /*********************************/
    /* construct config file options   */
```

```
/**********************************/
config_file.add_options()

        ("SHOW_VIDEO", po::value<string > (&show_video)->default_value("
  false"), " SHow the video")
        ("START_FRAME", po::value<size_t > (&options.start_frame)->
  default_value(0), "Starting Location of Video")
        ("STOP_FRAME", po::value<int>(&options.stop_frame)->default_value(-
  1), "Starting Location of Video")
        ("FRAME_RATE", po::value<size_t > (&options.frame_rate)->
  default_value(25), "Starting frame rate")
        ("VIDEO_FILENAME", po::value<string > (&options.video_filename)->
  default_value("NONE"), "Name of video file to play")
        ("VIDEO_OUTPUT_FILENAME", po::value<string > (&options.
  video_output_filename)->default_value("out.avi"), "Name of output results")

        //LANE DRAWING PARAMETERS
        ("SAVE_AVERAGE_FRAME", po::value<string > (&saveFrame)->
  default_value("false"), " Save Average Frame")
        ("LOAD_AVERAGE_FRAME", po::value<string > (&loadFrame)->
  default_value("false"), " Load Average Frame")
        ("AVERAGE_FRAME_FILENAME", po::value<string > (&options.avgFilename
  )->default_value("data/a.jpg"), "Name of average frame")
        ("SAVE_LANES", po::value<string > (&saveLanes)->default_value("
  false"), " Save Lane Data")
        ("LOAD_LANES", po::value<string > (&loadLanes)->default_value("
  false"), " Load Lane Data")
        ("LANE_DATA_FILE", po::value<string > (&options.laneFilename)->
  default_value("data/l.txt"), "Name of lane data file")


        ("INTEREST_POINT_MAX_FRAME_LIFE", po::value<size_t > (&options.
  interest_point_max_life)->default_value(50), "Max life of an interest point")

        ("SIFT_LANE_CROP", po::value<string > (&SIFT_lane_crop)->
  default_value("false"), "Crop lanes for SIFT Computation")
        ("SIFT_NUM_OCTAVES", po::value<int>(&siftNumOctaves)->default_value
  (3), "number of sift octaves")
        ("SIFT_NUM_OCTAVE_LAYERS", po::value<int>(&siftNumOctaveLayers)->
  default_value(4), "number of sift octave layers")
        ("SIFT_FIRST_OCTAVE", po::value<int>(&siftFirstOctave)->
  default_value(options.siftCommonParams.firstOctave), " First octave level, - means upsa
  ")
        ("SIFT_ANGLE_MODE", po::value<int>(&siftAngleMode)->default_value(
  options.siftCommonParams.FIRST_ANGLE), " angle mode")
        ("SIFT_ISNORMALIZE", po::value<int>(&siftIsNormalize)->
  default_value(options.siftDescriptorParams.DEFAULT_IS_NORMALIZE), "whether or not to
   normalize angles")
        ("SIFT_RECALCULATE_ANGLES", po::value<int>(&siftRecalculateAngles)
  ->default_value(options.siftDescriptorParams.GET_DEFAULT_MAGNIFICATION()), "
  Whether or not to recalculate angles")
        ("SIFT_MAGNIFICATION", po::value<double>(&siftMagnification)->
  default_value(options.siftDescriptorParams.GET_DEFAULT_MAGNIFICATION()), "default
   sift magnification")
        ("SIFT_EDGE_THRESHOLD", po::value<double>(&siftEdgeThreshold)->
  default_value(options.siftDetectorParams.GET_DEFAULT_EDGE_THRESHOLD()), "default
   edge threshold")
        ("SIFT_THRESHOLD", po::value<double>(&siftThreshold)->default_value
  (options.siftDetectorParams.GET_DEFAULT_THRESHOLD()), "default threshold")


        ("SURF_HESSIAN_THRESHOLD", po::value<double>(&surfHessianThreshold)
  ->default_value(400), "Default Hessian Threshold")
        ("SURF_NUM_OCTAVES", po::value<int>(&surfNumOctaves)->default_value
  (4), "Default Number of Octaves")
        ("SURF_NUM_OCTAVE_LAYERS", po::value<int>(&surfNumOctaveLayers)->
  default_value(2), "Default Number of Octave Layers")
        ("SURF_EXTENDED", po::value<int>(&surfExtended)->default_value(0),
  "Default Value of Extended")
        ("SURF_UPRIGHT", po::value<int>(&surfUpright)->default_value(0), "
  Default Upright Value")

        ("DENSITY_WINDOW_WIDTH", po::value<int>(&options.density_window)->
  default_value(51), "window size")
        ("INTEREST_POINT_METHOD", po::value<string > (&options.detector_type
  )->default_value("SIFT"), "Type of interest point detector to use");


/********************************************************************/
/*                                                  DO NOT
   CHANGE!!!!!!!!!!!!!                              */
/********************************************************************/

//this is a new description to allow us to combine the command line and
   config file
// inputs for use in the command-line only options. Should also contain
   hidded once
// they become relevant
po::options_description cmdline_options;
cmdline_options.add(generic).add(config_file);

//this is a new description which will add hidden descriptions once the
   hidden options
// are deemed necessary. Check multiple_sources.cpp in the boost program
   options example
// code to learn how to integrate this
po::options_description config_file_options;
config_file_options.add(config_file);

//This is a new description which will show visible options not hidded.
   This is important as
// will be what gets printed to the screen when the help gets called
po::options_description visible("Allowed options");
visible.add(generic).add(config_file);

//create variable map and map the command line arguements to it
po::variables_map vm;
po::store(po::parse_command_line(argc, argv, cmdline_options), vm);
po::notify(vm);

/****************************************/
/* CHECK FOR CONFIG FILE ARGUEMENTS */
/****************************************/
ifstream ifs(default_config_filename.c_str());
if (!ifs) {
    cout << "can not open config file: " << default_config_filename << "\n"
  ;
    exit(0);
} else {
    //if the filestream does exist, then load config arguements and parse
    po::store(po::parse_config_file(ifs, config_file_options), vm);
    po::notify(vm);
}
/**************************/
```

```
/* PRINT HELP OPTIONS */
/**************************/
if (vm.count("help")) {
    cout << visible << "\n";
    exit(0);
}


/******************************************************************/
/* CONVERT STRING INPUTS INTO APPROPRIATE CONFIG FILE OPTIONS */
/******************************************************************/
if (SIFT_lane_crop == "true")
    options.SIFT_lane_crop = true;
else
    options.SIFT_lane_crop = false;


//create sift common params
options.siftCommonParams.nOctaves = siftNumOctaves;
options.siftCommonParams.nOctaveLayers = siftNumOctaveLayers;
options.siftCommonParams.firstOctave = siftFirstOctave;
options.siftCommonParams.angleMode = siftAngleMode;

options.siftDescriptorParams.isNormalize = siftIsNormalize;
options.siftDescriptorParams.magnification = siftMagnification;
options.siftDescriptorParams.recalculateAngles = siftRecalculateAngles;

options.siftDetectorParams.edgeThreshold = siftEdgeThreshold;
options.siftDetectorParams.threshold = siftThreshold;

options.surfParams.hessianThreshold = surfHessianThreshold;
options.surfParams.numOctaves = surfNumOctaves;
options.surfParams.numOctaveLayers = surfNumOctaveLayers;
options.surfParams.extended = surfExtended;
options.surfParams.upright = surfUpright;


if (options.detector_type == "SIFT") { //using sift method
    options.detector = new SiftType(options.SIFT_lane_crop, options.
  siftCommonParams, options.siftDetectorParams, options.siftDescriptorParams);
} else if (options.detector_type == "SURF") {
    options.detector = new SurfType(options.surfParams);
} else {
    cout << "ERROR: Interest Point Detector Method unknown" << endl;
    exit(0);

}


//check to make sure that video file exists
if (fs::exists(fs::path(options.video_filename)) != true) {
    throw string("ERROR: video filename does not exist");
}

if (show_video == "true")
    options.show = true;
else
    options.show = false;


if (saveFrame == "true") options.saveAvgFrame = true;
else options.saveAvgFrame = false;
```

```
    if (loadFrame == "true") options.loadAvgFrame = true;
    else options.loadAvgFrame = false;
    if (saveLanes == "true") options.saveLanes = true;
    else options.saveLanes = false;
    if (loadLanes == "true") options.loadLanes = true;
    else options.loadLanes = false;

}
```

6.5.2.2 void Parser::write_configuration (Options & *options*) [static]

Definition at line 208 of file Parser.cpp.

References Options::avgFilename, Enumerations::bool2string(), Options::density_window, Options::detector_type, SURF_PARAMS::extended, Options::frame_rate, SURF_PARAMS::hessianThreshold, Options::interest_point_max_life, Options::laneFilename, Options::loadAvgFrame, Options::loadLanes, SURF_PARAMS::numOctaveLayers, SURF_PARAMS::numOctaves, Options::saveAvgFrame, Options::saveLanes, Options::show, Options::SIFT_lane_crop, Options::siftCommonParams, Options::siftDescriptorParams, Options::siftDetectorParams, Options::start_frame, Options::stop_frame, Options::surfParams, SURF_PARAMS::upright, Options::video_filename, and Options::video_output_filename.

Referenced by main().

```
                                                {

    ofstream fout;
    fout.open("data/_options.cfg");

    //video options
    fout << "SHOW_VIDEO     = " << Enumerations::bool2string(options.show) <<
      endl;
    fout << "START_FRAME    = " << options.start_frame << endl;
    fout << "STOP_FRAME     = " << options.stop_frame << endl;
    fout << "FRAME_RATE     = " << options.frame_rate << endl;
    fout << endl;


    fout << "VIDEO_FILENAME = " << options.video_filename << endl;
    fout << "VIDEO_OUTPUT_FILENAME = " << options.video_output_filename << endl
      ;
    fout << endl;

    //lane drawing parameters
    fout << "SAVE_AVERAGE_FRAME     = " << Enumerations::bool2string(options.
      saveAvgFrame) << endl;
    fout << "LOAD_AVERAGE_FRAME     = " << Enumerations::bool2string(options.
      loadAvgFrame) << endl;
    fout << "AVERAGE_FRAME_FILENAME = " << options.avgFilename << endl;
    fout << endl;
    fout << "SAVE_LANES     = " << Enumerations::bool2string(options.saveLanes)
       << endl;
    fout << "LOAD_LANES     = " << Enumerations::bool2string(options.loadLanes)
       << endl;
    fout << "LANE_DATA_FILE = " << options.laneFilename << endl;
    fout << endl;
    //Interest point parameters
    fout << "INTEREST_POINT_MAX_FRAME_LIFE = " << options.
      interest_point_max_life << endl;
    fout << "DENSITY_WINDOW_WIDTH          = " << options.density_window <<
      endl;
    fout << endl << endl;

    fout << "INTEREST_POINT_METHOD = " << options.detector_type << endl;

    fout << endl << endl;
    fout << "#SIFT Parameters" << endl;
    fout << "SIFT_LANE_CROP           = " << Enumerations::bool2string(options.
      SIFT_lane_crop) << endl;
    fout << "SIFT_NUM_OCTAVES         = " << options.siftCommonParams.nOctaves
      << endl;
    fout << "SIFT_NUM_OCTAVE_LAYERS   = " << options.siftCommonParams.
      nOctaveLayers << endl;
    fout << "SIFT_FIRST_OCTAVE        = " << options.siftCommonParams.
      firstOctave << endl;
    fout << "SIFT_ANGLE_MODE          = " << options.siftCommonParams.angleMode
       << endl;

    fout << "SIFT_ISNORMALIZE         = " << options.siftDescriptorParams.
      isNormalize << endl;
    fout << "SIFT_RECALCULATE_ANGLES  = " << options.siftDescriptorParams.
      recalculateAngles << endl;
    fout << "SIFT_MAGNIFICATION       = " << options.siftDescriptorParams.
      recalculateAngles << endl;
    fout << "SIFT_EDGE_THRESHOLD      = " << options.siftDetectorParams.
      edgeThreshold << endl;
    fout << "SIFT_THRESHOLD           = " << options.siftDetectorParams.
      threshold << endl;

    fout << endl << endl;
    fout << "#SURF PARAMETERS" << endl;
    fout << "SURF_HESSIAN_THRESHOLD    = " << options.surfParams.
      hessianThreshold << endl;
    fout << "SURF_NUM_OCTAVES          = " << options.surfParams.numOctaves <<
      endl;
    fout << "SURF_NUM_OCTAVE_LAYERS    = " << options.surfParams.numOctaveLayers
       << endl;
    fout << "SURF_EXTENDED             = " << options.surfParams.extended <<
      endl;
    fout << "SURF_UPRIGHT              = " << options.surfParams.upright <<
      endl;


    fout.close();

}
```

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

- src/core/Parser.h
- src/core/Parser.cpp

6.6 PointComp Struct Reference

```
#include <PointTracker.h>
```

Public Member Functions

- bool operator() (Point const &a, Point const &b)

6.6.1 Detailed Description

Functor which will compare two OpenCV Points. Used for the set stl container.

Parameters

a	First point
b	Second point

Returns

true if point a is closer to the origin than b.

Definition at line 29 of file PointTracker.h.

6.6.2 Member Function Documentation

6.6.2.1 bool PointComp::operator() (Point const & *a*, Point const & *b*)

Function operator for the Point Comparison Structure

Parameters

a	First point
b	Second point

Returns

The point which is nearest to the origin

Definition at line 16 of file PointTracker.cpp.

References PointDistanceL2().

```
                                                          {
    return ( PointDistanceL2(a, Point(0,0)) < PointDistanceL2(b, Point(0,0)));

}
```

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

- src/core/PointTracker.h
- src/core/PointTracker.cpp

6.7 PointTracker Class Reference

```
#include <PointTracker.h>
```

Public Member Functions

- PointTracker ()

  default constructor
- vector< Tuple > update_points (vector< KeyPoint >const &list)

  update the point positions
- int point_match (Point2f const &pt)

  Point Match.
- void set_point_max_life (const int max_life)

Private Attributes

- set< Point, PointComp > background_list
- vector< Tuple > point_list
- double MIN_DISTANCE
- size_t num_frames
- size_t MAX_NUM_FRAMES

6.7.1 Detailed Description

Class used to track and monitor the keypoints.

Definition at line 64 of file PointTracker.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 PointTracker::PointTracker ( )

default constructor

Default constructor for the point tracking structure

Definition at line 40 of file PointTracker.cpp.

References MAX_NUM_FRAMES, MIN_DISTANCE, and num_frames.

```
                                    {
    MIN_DISTANCE = 2.5;
    num_frames = 0;
    MAX_NUM_FRAMES = 50;
}
```

6.7.3 Member Function Documentation

6.7.3.1 int PointTracker::point_match (Point2f const & *pt*)

Point Match.

Check if a match exists between a point and the point list. Return the match index if found.

Parameters

pt	point to verify

Returns

index of match, -1 if no match found.

Definition at line 98 of file PointTracker.cpp.

References MIN_DISTANCE, point_list, and PointDistanceL2().

Referenced by update_points().

```
                                                        {

    double dist;
    for(size_t i=0; i<point_list.size(); i++){

        dist = PointDistanceL2( pt, point_list[i].centroid);

        if(dist < MIN_DISTANCE)
            return i;
    }
    return -1;

}
```

6.7.3.2 void PointTracker::set_point_max_life (const int *max_life*) [inline]

Definition at line 75 of file PointTracker.h.

Referenced by init().

```
                                                            {
        MAX_NUM_FRAMES = max_life;
    }
```

6.7.3.3 vector< Tuple > PointTracker::update_points (vector< KeyPoint >const & *list*)

update the point positions

Update the points, adding new points and removing redundant ones.

Parameters

list	of keypoints

Returns

revised list of point data structs

Definition at line 52 of file PointTracker.cpp.

References MAX_NUM_FRAMES, num_frames, point_list, and point_match().

Referenced by main().

```
                                                                 {

int idx;

//take the input list and check for collisions
for( size_t i=0; i<list.size(); i++){

    //check to see if a point matches
    idx = point_match(list[i].pt);

    if( idx >= 0 ){
        point_list[idx].found = true;

        point_list[idx].centroid.x = ((num_frames)/(double)(num_frames+1))*
  point_list[idx].centroid.x + (1/(double)(num_frames+1))*list[i].pt.x;
        point_list[idx].centroid.y = ((num_frames)/(double)(num_frames+1))*
  point_list[idx].centroid.y + (1/(double)(num_frames+1))*list[i].pt.y;
        continue;
    }
    //otherwise, add it to the list
    else{
        point_list.push_back(Tuple(list[i].pt));
    }
}

//iterate through point list, updating the strength of each point
for( int i=0; i<(int)point_list.size(); i++){
    point_list[i].strength = (point_list[i].span*point_list[i].strength +
  point_list[i].found)/(double)(point_list[i].span+1);
    point_list[i].found = 0;
    point_list[i].span = min( point_list[i].span+1, MAX_NUM_FRAMES);


    if( point_list[i].strength < 0.3 )
        point_list.erase( point_list.begin() + i--);
}

num_frames++;
if(num_frames > MAX_NUM_FRAMES)
```

```
        num_frames = MAX_NUM_FRAMES;

    return point_list;
}
```

6.7.4 Member Data Documentation

6.7.4.1 set< Point, PointComp > PointTracker::background_list [private]

Definition at line 81 of file PointTracker.h.

6.7.4.2 size_t PointTracker::MAX_NUM_FRAMES [private]

Definition at line 86 of file PointTracker.h.

Referenced by PointTracker(), and update_points().

6.7.4.3 double PointTracker::MIN_DISTANCE [private]

Definition at line 84 of file PointTracker.h.

Referenced by point_match(), and PointTracker().

6.7.4.4 size_t PointTracker::num_frames [private]

Definition at line 85 of file PointTracker.h.

Referenced by PointTracker(), and update_points().

6.7.4.5 vector< Tuple > PointTracker::point_list [private]

Definition at line 83 of file PointTracker.h.

Referenced by point_match(), and update_points().

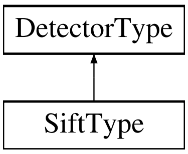
The documentation for this class was generated from the following files:

- src/core/PointTracker.h
- src/core/PointTracker.cpp

6.8 SiftType Class Reference

`#include <SiftType.h>`

Inheritance diagram for SiftType:



Public Member Functions

- SiftType ()
- SiftType (const bool &lc, SIFT::CommonParams const &cp, SIFT::DetectorParams const &detp, SIFT::DescriptorParams const &desp)
- SiftType (const SiftType &orig)
- virtual ∼SiftType ()
- virtual string isType () const
- virtual void compute_frame (const Mat &img, vector< KeyPoint > &keypoints, vector< Lane >const &lanes) const
- virtual void reset_detector ()

Private Attributes

- SIFT sift
- bool crop_lanes
- SIFT::CommonParams commonParams
- SIFT::DescriptorParams descriptorParams
- SIFT::DetectorParams detectorParams

6.8.1 Detailed Description

Definition at line 26 of file SiftType.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 SiftType::SiftType ( )

Definition at line 10 of file SiftType.cpp.

References commonParams, descriptorParams, detectorParams, and sift.

```
                   {

    commonParams = sift.getCommonParams();
    detectorParams = sift.getDetectorParams();
    descriptorParams = sift.getDescriptorParams();

}
```

6.8.2.2 SiftType::SiftType (const bool & *lc*, SIFT::CommonParams const & *cp*, SIFT::DetectorParams const & *detp*, SIFT::DescriptorParams const & *desp*)

Definition at line 18 of file SiftType.cpp.

```
                                            : crop_lanes(lc), detectorParams(detp),
        descriptorParams(desp){

}
```

6.8.2.3 SiftType::SiftType (const **SiftType** & *orig*)

Definition at line 25 of file SiftType.cpp.

```
                                            {
}
```

6.8.2.4 SiftType::∼SiftType ( ) [virtual]

Definition at line 28 of file SiftType.cpp.

```
                          {
}
```

6.8.3 Member Function Documentation

6.8.3.1 void SiftType::compute_frame (const Mat & *img*, vector< KeyPoint > & *keypoints*, vector< **Lane** >const & *lanes*) const [virtual]

Run sift on the current frame and output the keypoints and their descriptors

Parameters

in	*img*	Image to be evaluated
out	*keypoints*	Keypoints to be shown

Implements DetectorType.

Definition at line 35 of file SiftType.cpp.

References crop_lanes, cropSubImage(), expandRect(), insideLane(), pointsEqual(), and sift.

```
                                        {

    keypoints.clear();
    vector<KeyPoint> points, tpoints;
    //crop lane image
    if( crop_lanes == true ){

        Rect bbox;
        Mat subImg;

        //iterate over each crop, computing the sub-image, then sift
        for( size_t i=0; i<lanes.size(); i++){
            tpoints.clear();

            bbox = expandRect(lanes[i].bbox(), 50, Size(img.cols, img.rows));
            subImg = cropSubImage(img, bbox);


            sift(subImg, Mat(), tpoints);

            //append points to list
            for(size_t k=0; k<tpoints.size(); k++){
                Point2f tp = Point2f(tpoints[k].pt.x+bbox.x, tpoints[k].pt.y+
      bbox.y);
                tpoints[k].pt = tp;
                points.push_back(tpoints[k]);
            }
        }

        //remove unique points
        unique(points.begin(), points.end(), pointsEqual);

    }
    else{

        sift(img, Mat(), points);

    }

    for( size_t i=0; i<points.size(); i++)
        if( insideLane(points[i].pt, lanes) == true ){
            keypoints.push_back(points[i]);
        }
}
```

6.8.3.2 string SiftType::isType ( ) const [virtual]

Reimplemented from DetectorType.

Definition at line 31 of file SiftType.cpp.

```
                             {
    return "SiftType";
}
```

6.8.3.3 void SiftType::reset_detector ( ) [virtual]

Implements DetectorType.

Definition at line 80 of file SiftType.cpp.

References commonParams, descriptorParams, detectorParams, and sift.

```
                             {
    sift = SIFT(commonParams, detectorParams, descriptorParams );
}
```

6.8.4 Member Data Documentation

6.8.4.1 SIFT::CommonParams SiftType::commonParams [private]

Definition at line 52 of file SiftType.h.

Referenced by reset_detector(), and SiftType().

6.8.4.2 bool SiftType::crop_lanes [private]

Definition at line 51 of file SiftType.h.

Referenced by compute_frame().

6.8.4.3 SIFT::DescriptorParams SiftType::descriptorParams [private]

Definition at line 53 of file SiftType.h.

Referenced by reset_detector(), and SiftType().

6.8.4.4 SIFT::DetectorParams SiftType::detectorParams [private]

Definition at line 54 of file SiftType.h.

Referenced by reset_detector(), and SiftType().

6.8.4.5 SIFT SiftType::sift [private]

Definition at line 49 of file SiftType.h.

Referenced by compute_frame(), reset_detector(), and SiftType().

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

- src/structures/SiftType.h
- src/structures/SiftType.cpp

6.9 SURF_PARAMS Class Reference

`#include <SurfType.h>`

Public Attributes

- double hessianThreshold
- int numOctaves
- int numOctaveLayers
- int extended
- int upright

6.9.1 Detailed Description

Definition at line 23 of file SurfType.h.

6.9.2 Member Data Documentation

6.9.2.1 int SURF_PARAMS::extended

Definition at line 29 of file SurfType.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.9.2.2 double SURF_PARAMS::hessianThreshold

Definition at line 26 of file SurfType.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.9.2.3 int SURF_PARAMS::numOctaveLayers

Definition at line 28 of file SurfType.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.9.2.4 int SURF_PARAMS::numOctaves

Definition at line 27 of file SurfType.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

6.9.2.5 int SURF_PARAMS::upright

Definition at line 30 of file SurfType.h.

Referenced by Parser::parse_config_file(), and Parser::write_configuration().

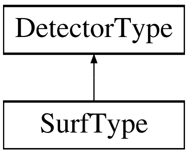
The documentation for this class was generated from the following file:

- src/structures/SurfType.h

6.10 SurfType Class Reference

```
#include <SurfType.h>
```

Inheritance diagram for SurfType:



Public Member Functions

- SurfType ()
- SurfType (const double ht, const int noct, const int noctlay, const int ext, const int up)
- SurfType (SURF_PARAMS const &params)
- SurfType (const SurfType &orig)
- virtual ∼SurfType ()
- virtual void compute_frame (const Mat &img, vector< KeyPoint > &keypoints, vector< Lane > const &lanes) const
- virtual void reset_detector ()
- virtual string isType () const

Private Attributes

- SURF surf
- double hessianThreshold
- int numOctaves
- int numOctaveLayers
- bool extended
- bool upright

6.10.1 Detailed Description

Definition at line 34 of file SurfType.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 SurfType::SurfType ( )

Definition at line 10 of file SurfType.cpp.

References hessianThreshold, and surf.

```
                   {
    hessianThreshold = surf.hessianThreshold;

    surf = SURF(hessianThreshold, 4, 2, false, true);
}
```

6.10.2.2 SurfType::SurfType (const double *ht,* const int *noct,* const int *noctlay,* const int *ext,* const int *up*)

Definition at line 23 of file SurfType.cpp.

References extended, hessianThreshold, numOctaveLayers, numOctaves, surf, and upright.

```
                          :
hessianThreshold(ht), numOctaves(noct), numOctaveLayers(noctlay), extended(ext)
      , upright(up)
{

    surf = SURF(hessianThreshold, numOctaves, numOctaveLayers, extended, upright
      );
}
```

6.10.2.3 SurfType::SurfType (SURF_PARAMS const & *params*)

Definition at line 16 of file SurfType.cpp.

References extended, hessianThreshold, numOctaveLayers, numOctaves, surf, and upright.

```
                                             :
hessianThreshold(params.hessianThreshold), numOctaves(params.numOctaves),
numOctaveLayers(params.numOctaveLayers), extended(params.extended), upright(
      params.upright)
{
    surf = SURF(hessianThreshold, numOctaves, numOctaveLayers, extended, upright
      );
}
```

6.10.2.4 SurfType::SurfType (const SurfType & *orig*)

Definition at line 30 of file SurfType.cpp.

```
                                       {
}
```

6.10.2.5 SurfType::∼SurfType ( ) [virtual]

Definition at line 33 of file SurfType.cpp.

```
                              {
}
```

6.10.3 Member Function Documentation

6.10.3.1 void SurfType::compute_frame (const Mat & *img*, vector< KeyPoint > & *keypoints*, vector< Lane > const & *lanes*) const [virtual]

Run surf on the current frame and output the keypoints and their descriptors

Parameters

in	*img*	Image to be evaluated
out	*keypoints*	Keypoints to be shown

Implements DetectorType.

Definition at line 36 of file SurfType.cpp.

References insideLane(), and surf.

```
                                               {

    keypoints.clear();
    vector<KeyPoint> points;
    surf(img, Mat(), points);

    for( size_t i=0; i<points.size(); i++)
        if( insideLane(points[i].pt, lanes) == true ){
            keypoints.push_back(points[i]);
        }

}
```

6.10.3.2 string DetectorType::isType ( ) const [virtual, inherited]

Reimplemented in SiftType.

Definition at line 19 of file DetectorType.cpp.

```
                                                  {
    return "DetectorType";
}
```

6.10.3.3 void SurfType::reset_detector ( ) [virtual]

Implements DetectorType.

Definition at line 49 of file SurfType.cpp.

```
                                {
    throw string("ERROR: not implemented yet");
}
```

6.10.4 Member Data Documentation

6.10.4.1 bool SurfType::extended [private]

Definition at line 58 of file SurfType.h.

Referenced by SurfType().

6.10.4.2 double SurfType::hessianThreshold [private]

Definition at line 55 of file SurfType.h.

Referenced by SurfType().

6.10.4.3 int SurfType::numOctaveLayers [private]

Definition at line 57 of file SurfType.h.

Referenced by SurfType().

6.10.4.4 int SurfType::numOctaves [private]

Definition at line 56 of file SurfType.h.

Referenced by SurfType().

6.10.4.5 SURF SurfType::surf [private]

Definition at line 54 of file SurfType.h.

Referenced by compute_frame(), and SurfType().

6.10.4.6 bool SurfType::upright [private]

Definition at line 59 of file SurfType.h.

Referenced by SurfType().

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

- src/structures/SurfType.h
- src/structures/SurfType.cpp

6.11 Tuple Class Reference

```
#include <PointTracker.h>
```

Public Member Functions

- Tuple ()

  Default Constructor.
- Tuple (const Point2f &pt)

  Parameterized Constructor.

Public Attributes

- Point2f centroid

  Center of point.
- size_t found

  Detection Status Code.
- double strength

  Strength/stability of point.
- size_t span

  Number of frames the point has existed.

6.11.1 Detailed Description

Class which contains relevant information for keypoints. The point, if it was detected in the frame, the strength / stability of the point, and the length of time is has existed in the video.

Definition at line 39 of file PointTracker.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Tuple::Tuple ( )

Default Constructor.

Sets the centroid to the origin with no strengh.

Default Constructor

Definition at line 25 of file PointTracker.cpp.

```
            : centroid(Point2f(0,0)), found(0), strength(0), span(0){
}
```

6.11.2.2 Tuple::Tuple (const Point2f & *pt*)

Parameterized Constructor.

Create a default point with no strength

Parameters

pt	Point

Definition at line 33 of file PointTracker.cpp.

```
                                        : centroid(pt), found(1), strength(0), span(0){
}
```

6.11.3 Member Data Documentation

6.11.3.1 Point2f Tuple::centroid

Center of point.

Definition at line 49 of file PointTracker.h.

6.11.3.2 size_t Tuple::found

Detection Status Code.

Definition at line 52 of file PointTracker.h.

6.11.3.3 size_t Tuple::span

Number of frames the point has existed.

Definition at line 58 of file PointTracker.h.

6.11.3.4 double Tuple::strength

Strength/stability of point.

Definition at line 55 of file PointTracker.h.

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

- src/core/PointTracker.h
- src/core/PointTracker.cpp

6.12 Vehicle Class Reference

```
#include <Vehicle.h>
```

Public Attributes

- vector< Point > points

6.12.1 Detailed Description

Definition at line 20 of file Vehicle.h.

6.12.2 Member Data Documentation

6.12.2.1 vector<Point> **Vehicle::points**

Definition at line 25 of file Vehicle.h.

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

- src/structures/Vehicle.h

6.13 VehicleDetection Class Reference

```
#include <VehicleDetection.h>
```

Static Public Member Functions

- static vector< Vehicle > computeCandidates (const Mat &img)
- static vector< Vehicle > classifyCandidates (const Mat &img, vector< Vehicle >const &candidates)

6.13.1 Detailed Description

Definition at line 23 of file VehicleDetection.h.

6.13.2 Member Function Documentation

6.13.2.1 vector< **Vehicle** > VehicleDetection::classifyCandidates (const Mat & *img*, vector< **Vehicle** >const & *candidates*) `[static]`

Definition at line 20 of file VehicleDetection.cpp.

```
                              {

   //output list of vehicle candidates
    vector<Vehicle> vehicles;



    return vehicles;
}
```

6.13.2.2 vector< Vehicle > VehicleDetection::computeCandidates (const Mat & *img*) [static]

Definition at line 10 of file VehicleDetection.cpp.

```
                                                                                  {

    //output list of vehicle candidates
    vector<Vehicle> candidates;



    return candidates;
}
```

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

- src/feature/VehicleDetection.h
- src/feature/VehicleDetection.cpp

Chapter 7

File Documentation

7.1 src/core/Enumerations.cpp File Reference

```
#include "Enumerations.h"
```

7.2 src/core/Enumerations.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h>
```

Classes

- class Enumerations

Defines

- #define COLOR_RED Scalar(0,0,255)
- #define COLOR_GREEN Scalar(0,255,0)
- #define COLOR_BLUE Scalar(255,0,0)

7.2.1 Define Documentation

7.2.1.1 #define COLOR_BLUE Scalar(255,0,0)

Definition at line 13 of file Enumerations.h.

Referenced by main().

7.2.1.2 #define COLOR_GREEN Scalar(0,255,0)

Definition at line 12 of file Enumerations.h.

Referenced by compute_point_density(), draw_lanes(), and main().

7.2.1.3 #define COLOR_RED Scalar(0,0,255)

Definition at line 11 of file Enumerations.h.

Referenced by compute_point_density(), and main().

7.3 src/core/GeometryUtilities.cpp File Reference

```
#include "GeometryUtilities.h"
```

Functions

- Rect expandRect (const Rect &bbox, size_t const &dist, Size const &imgSize)
- bool pointsEqual (KeyPoint const &a, KeyPoint const &b)
- double PointDistanceL1 (const Point &a, const Point &b)
- double PointDistanceL2 (const Point &a, const Point &b)

7.3.1 Function Documentation

7.3.1.1 Rect expandRect (const Rect & *bbox,* size_t const & *dist,* Size const & *imgSize*)

Definition at line 3 of file GeometryUtilities.cpp.

Referenced by SiftType::compute_frame().

```
{

    int minX = std::max( bbox.tl().x-dist, (size_t)0);
    int minY = std::max( bbox.tl().y-dist, (size_t)0);
    int maxX = bbox.br().x+(2*dist);
    int maxY = bbox.br().y+(2*dist);
    if( maxX >= imgSize.width ) maxX = imgSize.width-1;
    if( maxY >= imgSize.height) maxY = imgSize.height-1;


    return Rect(minX, minY, maxX-minX, maxY-minY);
}
```

7.3.1.2 double PointDistanceL1 (const Point & *a,* const Point & *b*)

Definition at line 23 of file GeometryUtilities.cpp.

```
                                                                        {
    return fabs((a.x-b.x) + (a.y-b.y));
}
```

7.3.1.3 double PointDistanceL2 (const Point & *a,* const Point & *b*)

Definition at line 27 of file GeometryUtilities.cpp.

Referenced by PointComp::operator()(), and PointTracker::point_match().

```
                                                                        {
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
```

7.3.1.4 bool pointsEqual (KeyPoint const & *a,* KeyPoint const & *b*)

Definition at line 16 of file GeometryUtilities.cpp.

Referenced by SiftType::compute_frame().

```
                                                                      {
    if( fabs(a.pt.x - b.pt.x) < 0.0001 )
        if( fabs(a.pt.y - b.pt.y) < 0.0001 )
            return true;
    return false;
}
```

7.4 src/core/GeometryUtilities.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h>
```

Functions

- Rect expandRect (const Rect &bbox, size_t const &dist, Size const &imgSize)
- bool pointsEqual (KeyPoint const &a, KeyPoint const &b)
- double PointDistanceL1 (const Point &a, const Point &b)
- double PointDistanceL2 (const Point &a, const Point &b)

7.4.1 Function Documentation

7.4.1.1 Rect expandRect (const Rect & *bbox,* size_t const & *dist,* Size const & *imgSize*)

Definition at line 3 of file GeometryUtilities.cpp.

Referenced by SiftType::compute_frame().

```
                                                                                       {
    int minX = std::max( bbox.tl().x-dist, (size_t)0);
    int minY = std::max( bbox.tl().y-dist, (size_t)0);
    int maxX = bbox.br().x+(2*dist);
    int maxY = bbox.br().y+(2*dist);
    if( maxX >= imgSize.width ) maxX = imgSize.width-1;
    if( maxY >= imgSize.height) maxY = imgSize.height-1;


    return Rect(minX, minY, maxX-minX, maxY-minY);
}
```

7.4.1.2 double PointDistanceL1 (const Point & *a*, const Point & *b*)

Definition at line 23 of file GeometryUtilities.cpp.

```
                                                                     {
    return fabs((a.x-b.x) + (a.y-b.y));
}
```

7.4.1.3 double PointDistanceL2 (const Point & *a*, const Point & *b*)

Definition at line 27 of file GeometryUtilities.cpp.

Referenced by PointComp::operator()(), and PointTracker::point_match().

```
                                                                     {
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
```

7.4.1.4 bool pointsEqual (KeyPoint const & *a*, KeyPoint const & *b*)

Definition at line 16 of file GeometryUtilities.cpp.

Referenced by SiftType::compute_frame().

```
                                                                    {
    if( fabs(a.pt.x - b.pt.x) < 0.0001 )
        if( fabs(a.pt.y - b.pt.y) < 0.0001 )
            return true;
    return false;
}
```

7.5 src/core/MatUtilities.cpp File Reference

```
#include "MatUtilities.h"
```

Functions

- Mat cropSubImage (Mat const &img, Rect const &bbox)

7.5.1 Function Documentation

7.5.1.1 Mat cropSubImage (Mat const & *img*, Rect const & *bbox*)

Definition at line 4 of file MatUtilities.cpp.

Referenced by SiftType::compute_frame().

```
                                                    {
    //create new image
    Mat newImg(Size(bbox.width, bbox.height), img.type());

    //move pixels over
    for(size_t i=0; i<bbox.width; i++)
        for(size_t j=0; j<bbox.height; j++){
            if(img.type() == CV_8UC1){
                newImg.at<uchar>(j,i) = img.at<uchar>(j+bbox.y, i+bbox.x);
            }
            else{
                throw string("ERROR: type not supported");
            }
        }

    return newImg.clone();
}
```

7.6 src/core/MatUtilities.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h> ×
#include <iostream>
```

Functions

- Mat cropSubImage (Mat const &img, Rect const &bbox)

7.6.1 Function Documentation

7.6.1.1 Mat cropSubImage (Mat const & *img*, Rect const & *bbox*)

Definition at line 4 of file MatUtilities.cpp.

Referenced by SiftType::compute_frame().

```
                                                    {
```

```
    //create new image
    Mat newImg(Size(bbox.width, bbox.height), img.type());

    //move pixels over
    for(size_t i=0; i<bbox.width; i++)
        for(size_t j=0; j<bbox.height; j++){
            if(img.type() == CV_8UC1){
                newImg.at<uchar>(j,i) = img.at<uchar>(j+bbox.y, i+bbox.x);
            }
            else{
                throw string("ERROR: type not supported");
            }
        }

    return newImg.clone();
}
```

7.7 src/core/Options.cpp File Reference

`#include "Options.h"`

7.8 src/core/Options.h File Reference

`#include <cv.h>` `#include <cvaux.h>` `#include <highgui.h>` `#include <string>` `#include "PointTracker.h"` `#include "../structures/DetectorType.h"` `#include "../structures/SiftType.h"` `#include "../structures/SurfType.h"` `#include "../structures/Lane.h"`

Classes

- class Options

7.9 src/core/Parser.cpp File Reference

`#include "Parser.h"` `#include "Enumerations.h"`

7.10 src/core/Parser.h File Reference

`#include <boost/filesystem.hpp>` `#include <boost/program_options.hpp>` `#include <cv.h>` `#include <cvaux.h>` `#include <highgui.h>` `#include <fstream>` `#include <string>` `#include "Options.h"` `#include "../structures/SiftType.h"` `#include "../structures/SurfType.h"`

Classes

- class Parser

7.11 src/core/PointTracker.cpp File Reference

```
#include "PointTracker.h"
```

7.12 src/core/PointTracker.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h> #include <set> #include <vector> #include "GeometryUtilities.h"
```

Classes

- struct PointComp
- class Tuple
- class PointTracker

7.13 src/core/VideoUtilities.cpp File Reference

```
#include "VideoUtilities.h"
```

Functions

- Mat median_frame (const string &filename, const size_t start_frame)

7.13.1 Function Documentation

7.13.1.1 Mat median_frame (const string & *filename*, const size_t *start_frame*)

Definition at line 3 of file VideoUtilities.cpp.

Referenced by lane_manager().

```
                                                                       {

    //
    size_t frame_skip  = 20;
    size_t frame_count = 100;
    Mat frameOut;
    size_t rows, cols;
    //open video and pull frames
    VideoCapture cap(filename.c_str());

    vector<Mat> frames;
    Mat tframe;

    //skip to position
    for( size_t i=0; i<start_frame; i++)
        cap >> tframe;

    rows = tframe.rows;
    cols = tframe.cols;

    //insert frames
    for( size_t i=0; i<frame_count; i++){

        cap >> tframe;
        //skip junk frames
        cvtColor( tframe, tframe, CV_BGR2GRAY);
        frames.push_back(tframe.clone());
        for(size_t j=0; j<frame_skip; j++)
            cap >> tframe;
    }

    //do median filtering on each pixel
    cvtColor( tframe, tframe, CV_BGR2GRAY);
    vector<uchar> pixels;
    frameOut = Mat(Size(cols, rows), CV_8UC1);
    for(size_t i=0; i<cols; i++){
        for(size_t j=0; j<rows; j++){

            //clear buffer
            pixels.clear();

            //add pixels to buffer
            for(size_t k=0; k<frames.size(); k++)
                pixels.push_back( frames[k].at<uchar>(j,i) );

            //sort buffer
            sort( pixels.begin(), pixels.end());

            frameOut.at<uchar>(j,i) = pixels[pixels.size()/2];

        }
    }

    return frameOut;
}
```

7.14 src/core/VideoUtilities.h File Reference

`#include <cv.h> #include <cvaux.h> #include <highgui.h> ×`
`#include <algorithm> #include <iostream>`

Functions

- Mat median_frame (const string &filename, const size_t start_frame)

7.14.1 Function Documentation

7.14.1.1 Mat median_frame (const string & *filename,* const size_t *start_frame*)

Definition at line 3 of file VideoUtilities.cpp.

Referenced by lane_manager().

```
                                                                    {

    //
    size_t frame_skip  = 20;
    size_t frame_count = 100;
    Mat frameOut;
    size_t rows, cols;

    //open video and pull frames
    VideoCapture cap(filename.c_str());

    vector<Mat> frames;
    Mat tframe;

    //skip to position
    for( size_t i=0; i<start_frame; i++)
        cap >> tframe;

    rows = tframe.rows;
    cols = tframe.cols;

    //insert frames
    for( size_t i=0; i<frame_count; i++){

        cap >> tframe;
        //skip junk frames
        cvtColor( tframe, tframe, CV_BGR2GRAY);
        frames.push_back(tframe.clone());
        for(size_t j=0; j<frame_skip; j++)
            cap >> tframe;
    }

    //do median filtering on each pixel
    cvtColor( tframe, tframe, CV_BGR2GRAY);
    vector<uchar> pixels;
    frameOut = Mat(Size(cols, rows), CV_8UC1);
    for(size_t i=0; i<cols; i++){
        for(size_t j=0; j<rows; j++){

            //clear buffer
            pixels.clear();

            //add pixels to buffer
            for(size_t k=0; k<frames.size(); k++)
                pixels.push_back( frames[k].at<uchar>(j,i) );

            //sort buffer
            sort( pixels.begin(), pixels.end());

            frameOut.at<uchar>(j,i) = pixels[pixels.size()/2];

        }
    }
    return frameOut;
}
```

7.15 src/feature/FeaturePointUtilities.h File Reference

#include <cv.h> #include <cvaux.h> #include <highgui.h> #include <vector> #include "../core/PointTracker.h" × #include "../core/Enumerations.h"

Functions

- size_t compute_area (size_t i, size_t j, Size bbox)
- void compute_point_density (Mat &imageOut, vector< Tuple > &keypoints, Size imgSize, Size bbox)

7.15.1 Function Documentation

7.15.1.1 size_t compute_area (size_t *i*, size_t *j*, Size *bbox*)

Definition at line 110 of file FeaturePointUtilities.h.

```
                                                                    {

    size_t area = 0;
    for( size_t i=0; i<bbox.width; i++){
        for( size_t j=0; j<bbox.height; j++){
            if( i >= 0 && j >= 0 )
                area += 1;
        }
    }

    return area;
}
```

7.15.1.2 void compute_point_density (Mat & *imageOut*, vector< Tuple > & *keypoints*, Size *imgSize*, Size *bbox*)

Definition at line 29 of file FeaturePointUtilities.h.

References COLOR_GREEN, Enumerations::color_interp(), COLOR_RED, and Enumerations::Scalar2Vec().

Referenced by main().

```
                    {

    Enumerations REF;
```

```
Mat integralImage = Mat(Size(imageOut.cols, imageOut.rows), CV_32FC1);
Mat integralImage2= Mat(Size(imageOut.cols, imageOut.rows), CV_32FC1);
Mat densityImage  = Mat(Size(imageOut.cols, imageOut.rows), CV_32FC1);
double area;
int xmin, xmax, ymin, ymax;
int count;
Scalar val;
int half = bbox.height/2;

imageOut = Scalar(0,0,0,0);
integralImage = Scalar(0,0,0,0);
integralImage2 = Scalar(0,0,0,0);
densityImage = Scalar(0,0,0,0);

//iterate through points, incrementing the count
for(size_t i=0; i<keypoints.size(); i++){
    if( keypoints[i].strength < 0.4 || keypoints[i].span < 5)
        integralImage.at<float>(keypoints[i].centroid) += .11;
    else if( keypoints[i].strength >= 0.4 )
        integralImage.at<float>(keypoints[i].centroid) -= .1;
}

//build the integral image
for(int i=0; i<integralImage.cols; i++)
    for(int j=0; j<integralImage.rows; j++){

        xmin = max( i-1, 0);
        ymin = max( j-1, 0);

        if( i == 0 && j == 0 )
            integralImage2.at<float>(j,i) = integralImage.at<float>(j,i);
        else if( i == 0 )
            integralImage2.at<float>(j,i) = integralImage2.at<float>(j-1, i
  )
                                          + integralImage.at<float>(j,i);
        else if( j == 0 )
            integralImage2.at<float>(j,i) = integralImage2.at<float>(j, i-1
  )
                                          + integralImage.at<float>(j,i);
        else
            integralImage2.at<float>(j,i) = integralImage.at<float>(j,i) +
                                            integralImage2.at<float>(ymin,
  i) +
                                            integralImage2.at<float>(j,
  xmin) -
                                            integralImage2.at<float>(ymin,
  xmin);
    }

//compute the density map
for(int i=1; i<integralImage.cols; i++)
    for(int j=0; j<integralImage.rows; j++){

        xmin = max(i-half-1, 0);
        ymin = max(j-half-1, 0);
        xmax = min(i+half  , integralImage.cols-1);
        ymax = min(j+half  , integralImage.rows-1);

        //iterate over map
        densityImage.at<float>(j,i) = integralImage2.at<float>( ymax, xmax)
```

```
                                        - integralImage2.at<float>( ymin, xmax)
                                        - integralImage2.at<float>( ymax, xmin)
                                        + integralImage2.at<float>( ymin, xmin)
      ;
            //densityImage.at<float>(j,i) /= (xmax-xmin-1)*(ymax-ymin-1);
        }

    for(int i=1; i<integralImage.cols; i++)
        for(int j=0; j<integralImage.rows; j++){
            if( densityImage.at<float>(j,i) < 0 )
                densityImage.at<float>(j,i) = 0;
            if( densityImage.at<float>(j,i) > 1)
                densityImage.at<float>(j,i) = 1;
        }

    normalize( densityImage, densityImage, 0, 1, CV_MINMAX);
    for(int i=1; i<integralImage.cols; i++)
        for(int j=0; j<integralImage.rows; j++){
            imageOut.at<Vec3b>(j,i) = REF.Scalar2Vec(REF.color_interp(COLOR_RED
      , COLOR_GREEN, densityImage.at<float>(j,i)));
        }
}
```

7.16 src/feature/VehicleDetection.cpp File Reference

#include "VehicleDetection.h"

7.17 src/feature/VehicleDetection.h File Reference

#include <cv.h> #include <cvaux.h> #include <highgui.h> #include <iostream> #include <vector> #include "../structures/Vehicle.h"

Classes

- class VehicleDetection

7.18 src/main.cpp File Reference

#include <iostream> #include <vector> #include <cv.h> × #include "feature/VehicleDetection.h" #include <cvaux.h> #include <highgui.h> #include "ui/LaneDrawing.h" #include "feature/FeaturePointUtilities.h" #include "core/Enumerations.h" #include "core/Options.h" #include "core/Parser.h" × #include "ui/Mouse.h" #include "structures/Vehicle.h"

Functions

- void init (Options &options)
- int main (int argc, char **argv)

Variables

- int pX
- int pY
- size_t mouse_flag

7.18.1 Function Documentation

7.18.1.1 void init (Options & *options*)

Definition at line 160 of file main.cpp.

References Options::bout, Options::cap, Options::DEBUG, Options::dout, Options::equalizeHistogram, Options::frame_count, Options::interest_point_max_life, mouse_flag, Options::pointHistory, PointTracker::set_point_max_life(), Options::show, Options::start_frame, Options::video_filename, Options::video_output_filename, Options::vout, and Options::window_name.

Referenced by main().

```
                            {

    options.DEBUG = true;
    options.equalizeHistogram = true;

    options.frame_count = options.start_frame;

    mouse_flag = 0;

    //create window
    if (options.show == true) {
        namedWindow(options.window_name.c_str());
        namedWindow("black frame");
        namedWindow("density frame");
    }

    //open video file
    options.cap.open(options.video_filename);

    int ex = static_cast<int> (options.cap.get(CV_CAP_PROP_FOURCC));
    Size S = Size(
            (int) options.cap.get(CV_CAP_PROP_FRAME_WIDTH),
            (int) options.cap.get(CV_CAP_PROP_FRAME_HEIGHT));
    options.vout.open(options.video_output_filename.c_str(), ex, options.cap.
      get(CV_CAP_PROP_FPS) + 5, S, true);
    options.bout.open("data/black_output.avi", ex, options.cap.get(
      CV_CAP_PROP_FPS) + 5, S, true);
    options.dout.open("data/density_output.avi", ex, options.cap.get(
      CV_CAP_PROP_FPS) + 5, S, true);
    options.pointHistory.set_point_max_life(options.interest_point_max_life);
}
```

7.18.1.2 int main (int *argc,* char ∗∗ *argv*)

start lane drawing section

END OF LANE DRAWING

build interest point background dictionary

compute list of candidates vector<Vehicle> candidates = VehicleDetection::computeCandidates(options.density_frame);

compute validated list of vehicles vector<Vehicle> classifiedVehicles = VehicleDetection::classifyCandidates(options.frame, candidates);

Definition at line 41 of file main.cpp.

References Options::black_frame, Options::bout, Options::cap, COLOR_BLUE, COLOR_GREEN, Enumerations::color_interp(), COLOR_RED, DetectorType::compute_frame(), compute_point_density(), Options::DEBUG, Options::density_frame, Options::density_window, Options::detector, Options::dout, Options::equalizeHistogram, Options::frame, Options::frame_count, Options::frame_rate, Options::gray_frame, init(), Options::key, lane_manager(), Options::lanes, mouse_flag, Parser::parse_config_file(), Options::pointHistory, Options::show, Options::start_frame, Options::stop_frame, PointTracker::update_points(), Options::vout, Options::window_name, and Parser::write_configuration().

```
                                {

    //create options
    Options options;
    options.DEBUG = true;

    if( options.DEBUG == true)
        cout << "Start of program" << endl;


    //parse command-line options
    if( options.DEBUG == true)
        cout << "Start of parser" << endl;
    Parser::parse_config_file(argc, argv, options, "data/options.cfg");
    if( options.DEBUG == true)
        cout << "End of parser" << endl;

    //initialize remaining options
    init(options);

    vector<KeyPoint> points;
    vector<Tuple> showPoints;

    lane_manager(options);
    mouse_flag = 2;
    //load first frame
    for (size_t i = 0; i < options.start_frame; i++)
        options.cap >> options.frame;
    options.black_frame = Mat(Size(options.frame.cols, options.frame.rows),
      CV_8UC3);
    options.density_frame = Mat(Size(options.frame.cols, options.frame.rows),
      CV_8UC3);

    while (options.frame.data && (options.stop_frame == -1 || (int) options.
      frame_count < options.stop_frame)) {

        //compute keypoints
        cvtColor(options.frame, options.gray_frame, CV_BGR2GRAY);

        //
        if( options.equalizeHistogram == true )
            equalizeHist( options.gray_frame, options.gray_frame);

        options.detector->compute_frame(options.gray_frame, points, options.
      lanes);

        //add keypoints to frame record
        showPoints = options.pointHistory.update_points(points);

        //compute density image
        compute_point_density(options.density_frame, showPoints, Size(options.
      frame.cols, options.frame.rows), Size(options.density_window, options.
      density_window));


        //track
        //some tracking functions


        //draw keypoints on frame
        for (size_t i = 0; i < showPoints.size(); i++)
            if (showPoints[i].span > 8) {
                circle(options.frame, Point(showPoints[i].centroid.x,
      showPoints[i].centroid.y), 1, Enumerations::color_interp(COLOR_RED, COLOR_GREEN,
      showPoints[i].strength), 1);
                circle(options.black_frame, Point(showPoints[i].centroid.x,
      showPoints[i].centroid.y), 1, Enumerations::color_interp(COLOR_RED, COLOR_GREEN,
      showPoints[i].strength), 1);

            } else {
                circle(options.frame, Point(showPoints[i].centroid.x,
      showPoints[i].centroid.y), 1, COLOR_BLUE, 1);
                circle(options.black_frame, Point(showPoints[i].centroid.x,
      showPoints[i].centroid.y), 1, COLOR_BLUE, 1);
            }

        //draw lanes onto images
        for (size_t z = 0; z < options.lanes.size(); z++) {
            options.lanes[z].draw(options.density_frame);
            options.lanes[z].draw(options.black_frame);
            options.lanes[z].draw(options.frame);
        }

        if (options.show) {

            imshow(options.window_name.c_str(), options.frame);
            imshow("black frame", options.black_frame);
            imshow("density frame", options.density_frame);
            options.key = waitKey(options.frame_rate);
        } else {
            if (options.frame_count % 10 == 0)
                cout << options.frame_count << endl;
            options.key = waitKey(1);
        }

        options.dout << options.density_frame;
        options.vout << options.frame;
        options.bout << options.black_frame;

        if (options.key == 27)
            break;


        options.cap >> options.frame;
        options.frame_count++;
        options.black_frame = Scalar(0, 0, 0, 0);
        options.density_frame = Scalar(0, 0, 0, 0);
    }

    Parser::write_configuration( options );

    return 0;
}
```

7.18.2 Variable Documentation

7.18.2.1 size_t mouse_flag

Definition at line 34 of file main.cpp.

Referenced by draw_lanes(), init(), main(), and mouseFunc().

7.18.2.2 int pX

Definition at line 33 of file main.cpp.

Referenced by draw_lanes(), and mouseFunc().

7.18.2.3 int pY

Definition at line 33 of file main.cpp.

Referenced by draw_lanes(), and mouseFunc().

7.19 src/structures/DetectorType.cpp File Reference

```
#include "DetectorType.h"
```

7.20 src/structures/DetectorType.h File Reference

`#include <cv.h> #include <cvaux.h> #include <highgui.h> ×`
`#include <string> #include "Lane.h"`

Classes

- class DetectorType

7.21 src/structures/Lane.cpp File Reference

`#include "Lane.h"`

Functions

- bool insideLane (Point2f const &pt, vector< Lane >const &lanes)

7.21.1 Function Documentation

7.21.1.1 bool insideLane (Point2f const & *pt,* vector< Lane >const & *lanes*)

Definition at line 19 of file Lane.cpp.

Referenced by SiftType::compute_frame(), and SurfType::compute_frame().

```
                                                                                {
    for( size_t i=0; i<lanes.size(); i++){
        if(lanes[i].isInside(pt) == true)
            return true;
    }
    return false;
}
```

7.22 src/structures/Lane.h File Reference

`#include <cv.h> #include <cvaux.h> #include <highgui.h>`
`#include <iostream> #include <boost/assign.hpp> #include`
`<boost/geometry.hpp> #include <boost/geometry/geometries/box.-`
`hpp> #include <boost/geometry/geometries/linestring.-`
`hpp> #include <boost/geometry/geometries/point_xy.hpp> ×`
`#include <boost/geometry/geometries/polygon.hpp> #include`
`<boost/geometry/geometries/adapted/boost_tuple.hpp>`

Classes

- class Lane

Functions

- BOOST_GEOMETRY_REGISTER_BOOST_TUPLE_CS (cs::cartesian)
- bool insideLane (Point2f const &pt, vector< Lane >const &lanes)

7.22.1 Function Documentation

7.22.1.1 BOOST_GEOMETRY_REGISTER_BOOST_TUPLE_CS (cs::cartesian)

7.22.1.2 bool insideLane (Point2f const & *pt,* vector< **Lane** >const & *lanes*)

Definition at line 19 of file Lane.cpp.

Referenced by SiftType::compute_frame(), and SurfType::compute_frame().

```
                                                                           {

    for( size_t i=0; i<lanes.size(); i++){
        if(lanes[i].isInside(pt) == true)
            return true;
    }
    return false;
}
```

7.23 src/structures/SiftType.cpp File Reference

```
#include "SiftType.h"
```

7.24 src/structures/SiftType.h File Reference

#include <cv.h> #include <cvaux.h> #include <highgui.h> #include <algorithm> #include <string> #include <vector> × #include "DetectorType.h" #include "../core/GeometryUtilities.h" #include "../core/MatUtilities.h"

Classes

- class SiftType

7.25 src/structures/SurfType.cpp File Reference

```
#include "SurfType.h"
```

7.26 src/structures/SurfType.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h>
#include <vector> #include "DetectorType.h" #include "Lane.h"
```

Classes

- class SURF_PARAMS
- class SurfType

7.27 src/structures/Vehicle.cpp File Reference

```
#include "Vehicle.h"
```

7.28 src/structures/Vehicle.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h> ×
#include <vector>
```

Classes

- class Vehicle

7.29 src/ui/LaneDrawing.cpp File Reference

```
#include "LaneDrawing.h"
```

Functions

- void lane_manager (Options &options)
- vector< Lane > draw_lanes (const Mat &avg_frame)
- vector< Lane > load_lanes (const string &filename, const string &vidname, bool &fail)
- void save_lanes (const string &filename, vector< Lane > &lanes, const string &vidname)

7.29.1 Function Documentation

7.29.1.1 vector<Lane> draw_lanes (const Mat & *avg_frame*)

Make the user draw the lanes onto the median image

Parameters

avg_-frame,median	frame

Returns

list of lanes

Definition at line 81 of file LaneDrawing.cpp.

References Lane::addVertex(), Lane::changeLast(), Lane::clear(), COLOR_GREEN, - Lane::draw(), mouse_flag, mouseFunc(), pX, pY, and Lane::size().

Referenced by lane_manager().

```
                                        {

    cout << "-> " << avg_frame.cols << " x " << avg_frame.rows << endl;
    //start drawing loop
    Mat tframe;
    vector<Lane> existing_lanes;
    Lane tempLane;

    size_t prev_flag = mouse_flag;
    namedWindow("Lane Drawing Phase");
    setMouseCallback("Lane Drawing Phase", mouseFunc, 0);
    char key;

    cout << "aaa" << endl;

    while (true) {

        //clone original
        tframe = avg_frame.clone();

        cvtColor(avg_frame, tframe, CV_GRAY2BGR);

        //copy all exising polygons
        for (size_t i = 0; i < existing_lanes.size(); i++){
            existing_lanes[i].draw(tframe);
            rectangle( tframe, existing_lanes[i].bbox().tl(), existing_lanes[i]
      .bbox().br(), Scalar(220, 220, 0), 1);
        }

        //copy current polygon under investigation
        if (mouse_flag == 1) {
            mouse_flag = 0;
            if (tempLane.size() > 0) {
                tempLane.changeLast(Point(pX, pY));
                tempLane.addVertex(Point(pX, pY));
            }
            if (tempLane.size() == 0) {
                tempLane.addVertex(Point(pX, pY));
                tempLane.addVertex(Point(pX, pY));
            }
        }

        tempLane.draw(tframe);
        if (tempLane.size() > 1)
            tempLane.changeLast(Point(pX, pY));

        imshow("Lane Drawing Phase", tframe);
        key = waitKey(30);

        if (key == ’q’) {
            break;
        }
        if (key == ’x’) {
            if (tempLane.size() >= 3) {
                existing_lanes.push_back(tempLane);
                existing_lanes.back().color = COLOR_GREEN;
                existing_lanes.back().finalize();
                tempLane.clear();
            }
        }
    }

    destroyWindow("Lane Drawing Phase");

    return existing_lanes;
}
```

7.29.1.2 void lane_manager (Options & *options*)

Compute the Median Frame

Definition at line 5 of file LaneDrawing.cpp.

References Options::avgFilename, draw_lanes(), Options::laneFilename, Options::lanes, load_lanes(), Options::loadAvgFrame, Options::loadLanes, median_frame(), save_lanes(), Options::saveAvgFrame, Options::saveLanes, Options::start_frame, and Options::video_filename.

Referenced by main().

```
                                {

    Mat avg_frame;

    //check to see if the lanes already exist, otherwise skip
    if( options.loadLanes == true ){
        if( bf::exists(bf::path(options.laneFilename))){
            bool fail;
            options.lanes = load_lanes(options.laneFilename, options.
  video_filename, fail);

            if( fail == false )
                return;
        }
    }

    /***********************/
    /*    AVERAGE LANE     */
    /***********************/
    //load the average frame
    bool frameExists = false;
    if( options.loadAvgFrame == true ){
        if(bf::exists(bf::path(options.avgFilename))){
            avg_frame = imread(options.avgFilename, 0);
            frameExists = true;
        }
        else{
            avg_frame = median_frame(options.video_filename, options.start_frame
      );
        }
    }
    else{
        avg_frame = median_frame(options.video_filename, options.start_frame);
    }

    //save the average frame
    if( options.saveAvgFrame == true && frameExists == false ){
        imwrite( options.avgFilename.c_str(), avg_frame);
    }

    /************************/
    /*     LANE DRAWING     */
    /************************/
    //load data
    bool laneExists = false;
    bool fail;
    if( options.loadLanes == true ){
        if( bf::exists(bf::path(options.laneFilename))){
            options.lanes = load_lanes(options.laneFilename, options.
      video_filename, fail);
            if( fail == true ){
                laneExists = false;
                options.lanes = draw_lanes( avg_frame );
            }
            else
                laneExists = true;
        }
        else{
            options.lanes = draw_lanes( avg_frame );
        }
    }
    else{
        options.lanes = draw_lanes( avg_frame );
    }

    //save data
    if( options.saveLanes == true && laneExists == false ){
        save_lanes( options.laneFilename, options.lanes, options.video_filename
      );
    }

}
```

7.29.1.3 vector< Lane > load_lanes (const string & *filename*, const string & *vidname*, bool & *fail*)

Definition at line 148 of file LaneDrawing.cpp.

References Lane::addVertex(), and Lane::clear().

Referenced by lane_manager().

```
        {

    vector<Lane> lane_list;
    Lane tLane;
    Point pt;

    ifstream fin;
    fin.open(filename.c_str());

    //load video filename
    string fname;
    fin >> fname;
    if( vidname != fname ){
        fail = true;
        return lane_list;
    }
    else
        fail = false;


    //load size of lanes
    size_t lSize, npnts;
    fin >> lSize;

    for(size_t i=0; i<lSize; i++){

        //clear lane
        tLane.clear();

        //start loading points
        fin >> npnts;
        for(size_t j=0; j<npnts; j++){
            fin >> pt.x >> pt.y;
            tLane.addVertex(pt);
        }
        lane_list.push_back(tLane);
        lane_list.back().finalize();

    }

    fin.close();

    return lane_list;
}
```

7.29.1.4 void save_lanes (const string & *filename*, vector< Lane > & *lanes*, const string & *vidname*)

Definition at line 193 of file LaneDrawing.cpp.

Referenced by lane_manager().

```
            {

    //open stream
    ofstream fout;
    fout.open(filename.c_str());
    fout << vidname << endl;

    fout << lanes.size() << endl;
    vector<Point> pnts;
    for( size_t i=0; i<lanes.size(); i++){
        pnts = lanes[i].getVertices();

        fout << pnts.size() << " ";
        for(size_t j=0; j<pnts.size(); j++)
            fout << pnts[j].x << " " << pnts[j].y << " ";
        fout << endl;
    }

    //close stream
    fout.close();

}
```

7.30 src/ui/LaneDrawing.h File Reference

`#include <boost/filesystem.hpp>` `#include <cv.h>` `#include <cvaux.h>` `#include <highgui.h>` `#include <fstream>` `#include <iostream>` `#include <vector>` `#include "../core/VideoUtilities.h"` `#include "../core/Options.h"` `#include "../structures/Lane.h"` `#include "../core/Enumerations.h"` `#include "Mouse.h"`

Functions

- void lane_manager (Options &options)
- vector< Lane > load_lanes (const string &filename, const string &vidname, bool &fail)
- void save_lanes (const string &filename, vector< Lane > &lanes, const string &vidname)
- vector< Lane > draw_lanes (const Mat &avg_frame)

Variables

- size_t mouse_flag
- int pX
- int pY

7.30.1 Function Documentation

7.30.1.1 vector<Lane> draw_lanes (const Mat & *avg_frame*)

Make the user draw the lanes onto the median image

Parameters

avg_-frame,median	frame

Returns

list of lanes

Definition at line 81 of file LaneDrawing.cpp.

References Lane::addVertex(), Lane::changeLast(), Lane::clear(), COLOR_GREEN, -Lane::draw(), mouse_flag, mouseFunc(), pX, pY, and Lane::size().

Referenced by lane_manager().

```
                                            {

    cout << "-> " << avg_frame.cols << " x " << avg_frame.rows << endl;
    //start drawing loop
    Mat tframe;
    vector<Lane> existing_lanes;
    Lane tempLane;

    size_t prev_flag = mouse_flag;
    namedWindow("Lane Drawing Phase");
    setMouseCallback("Lane Drawing Phase", mouseFunc, 0);
    char key;

    cout << "aaa" << endl;

    while (true) {

        //clone original
        tframe = avg_frame.clone();

        cvtColor(avg_frame, tframe, CV_GRAY2BGR);

        //copy all exising polygons
        for (size_t i = 0; i < existing_lanes.size(); i++){
            existing_lanes[i].draw(tframe);
            rectangle( tframe, existing_lanes[i].bbox().tl(), existing_lanes[i]
      .bbox().br(), Scalar(220, 220, 0), 1);
        }

        //copy current polygon under investigation
        if (mouse_flag == 1) {
            mouse_flag = 0;
            if (tempLane.size() > 0) {
                tempLane.changeLast(Point(pX, pY));
                tempLane.addVertex(Point(pX, pY));
            }
            if (tempLane.size() == 0) {
                tempLane.addVertex(Point(pX, pY));
                tempLane.addVertex(Point(pX, pY));
            }
        }

        tempLane.draw(tframe);
        if (tempLane.size() > 1)
            tempLane.changeLast(Point(pX, pY));

        imshow("Lane Drawing Phase", tframe);
        key = waitKey(30);

        if (key == 'q') {
            break;
        }
        if (key == 'x') {
            if (tempLane.size() >= 3) {
                existing_lanes.push_back(tempLane);
                existing_lanes.back().color = COLOR_GREEN;
                existing_lanes.back().finalize();
                tempLane.clear();
            }
        }
    }

    destroyWindow("Lane Drawing Phase");

    return existing_lanes;
}
```

7.30.1.2 void lane_manager (Options & *options*)

Compute the Median Frame

Definition at line 5 of file LaneDrawing.cpp.

References Options::avgFilename, draw_lanes(), Options::laneFilename, Options::lanes, load_lanes(), Options::loadAvgFrame, Options::loadLanes, median_frame(), save_lanes(), Options::saveAvgFrame, Options::saveLanes, Options::start_frame, and Options::video_filename.

Referenced by main().

```
                                    {

    Mat avg_frame;

    //check to see if the lanes already exist, otherwise skip
    if( options.loadLanes == true ){
        if( bf::exists(bf::path(options.laneFilename))){
            bool fail;
            options.lanes = load_lanes(options.laneFilename, options.
      video_filename, fail);

            if( fail == false )
                return;
        }
    }

    /**********************/
    /*    AVERAGE LANE    */
    /**********************/
    //load the average frame
    bool frameExists = false;
    if( options.loadAvgFrame == true ){
        if(bf::exists(bf::path(options.avgFilename))){
            avg_frame = imread(options.avgFilename, 0);
            frameExists = true;
        }
        else{
            avg_frame = median_frame(options.video_filename, options.start_frame
      );
        }
    }
    else{
        avg_frame = median_frame(options.video_filename, options.start_frame);
    }

    //save the average frame
    if( options.saveAvgFrame == true && frameExists == false ){
        imwrite( options.avgFilename.c_str(), avg_frame);
    }

    /***********************/
    /*     LANE DRAWING    */
    /***********************/
    //load data
    bool laneExists = false;
    bool fail;
    if( options.loadLanes == true ){
        if( bf::exists(bf::path(options.laneFilename))){
            options.lanes = load_lanes(options.laneFilename, options.
      video_filename, fail);
            if( fail == true ){
                laneExists = false;
                options.lanes = draw_lanes( avg_frame );
            }
            else
                laneExists = true;
        }
        else{
            options.lanes = draw_lanes( avg_frame );
        }
    }
    else{
        options.lanes = draw_lanes( avg_frame );
    }

    //save data
    if( options.saveLanes == true && laneExists == false ){
        save_lanes( options.laneFilename, options.lanes, options.video_filename
      );
    }

}
```

7.30.1.3 vector< Lane > load_lanes (const string & *filename,* const string & *vidname,* bool & *fail*)

Definition at line 148 of file LaneDrawing.cpp.

References Lane::addVertex(), and Lane::clear().

Referenced by lane_manager().

```
           {

    vector<Lane> lane_list;
    Lane tLane;
    Point pt;

    ifstream fin;
    fin.open(filename.c_str());

    //load video filename
    string fname;
    fin >> fname;
    if( vidname != fname ){
        fail = true;
        return lane_list;
    }
    else
        fail = false;


    //load size of lanes
    size_t lSize, npnts;
    fin >> lSize;

    for(size_t i=0; i<lSize; i++){

        //clear lane
        tLane.clear();

        //start loading points
        fin >> npnts;
        for(size_t j=0; j<npnts; j++){
            fin >> pt.x >> pt.y;
            tLane.addVertex(pt);
        }
        lane_list.push_back(tLane);
        lane_list.back().finalize();

    }

    fin.close();

    return lane_list;
}
```

7.30.1.4 void save_lanes (const string & *filename,* vector< Lane > & *lanes,* const string & *vidname*)

Definition at line 193 of file LaneDrawing.cpp.

Referenced by lane_manager().

```
            {

    //open stream
    ofstream fout;
    fout.open(filename.c_str());
    fout << vidname << endl;

    fout << lanes.size() << endl;
    vector<Point> pnts;
    for( size_t i=0; i<lanes.size(); i++){
        pnts = lanes[i].getVertices();

        fout << pnts.size() << " ";
        for(size_t j=0; j<pnts.size(); j++)
            fout << pnts[j].x << " " << pnts[j].y << " ";
        fout << endl;
    }

    //close stream
    fout.close();

}
```

7.30.2 Variable Documentation

7.30.2.1 size_t mouse_flag

Definition at line 34 of file main.cpp.

7.30.2.2 int pX

Definition at line 33 of file main.cpp.

7.30.2.3 int pY

Definition at line 33 of file main.cpp.

7.31 src/ui/Mouse.cpp File Reference

```
#include "Mouse.h"
```

Functions

- void mouseFunc (const int actualEvent, const int x, const int y, const int flag, void ∗)

7.31.1 Function Documentation

7.31.1.1 void mouseFunc (const int *actualEvent,* const int *x,* const int *y,* const int *flag,* void ∗)

Definition at line 3 of file Mouse.cpp.

References mouse_flag, pX, and pY.

Referenced by draw_lanes().

```
               {

    // shift+leftclick = rightclick (for MAC users)
    int event = actualEvent;

    //only use function if program is in drawing mode
    if( mouse_flag >= 2 )
        return;
    pX = x;
    pY = y;

    switch (event){

      case (CV_EVENT_LBUTTONDOWN):


          if( mouse_flag == 0 ) mouse_flag = 1;
          else mouse_flag = 0;

          break;
    }
}
```

7.32 src/ui/Mouse.h File Reference

```
#include <cv.h> #include <cvaux.h> #include <highgui.h> ×
#include <iostream>
```

Functions

- void mouseFunc (const int actualEvent, const int x, const int y, const int flag, void ∗)

Variables

- int pX
- int pY
- size_t mouse_flag

7.32.1 Function Documentation

7.32.1.1 void mouseFunc (const int *actualEvent,* const int *x,* const int *y,* const int *flag,* void ∗)

Definition at line 3 of file Mouse.cpp.

References mouse_flag, pX, and pY.

Referenced by draw_lanes().

```
               {

    // shift+leftclick = rightclick (for MAC users)
    int event = actualEvent;

    //only use function if program is in drawing mode
    if( mouse_flag >= 2 )
        return;
    pX = x;
    pY = y;

    switch (event){

      case (CV_EVENT_LBUTTONDOWN):


          if( mouse_flag == 0 ) mouse_flag = 1;
          else mouse_flag = 0;

          break;
    }
}
```

7.32.2 Variable Documentation

7.32.2.1 size_t mouse_flag

Definition at line 34 of file main.cpp.

Referenced by draw_lanes(), init(), main(), and mouseFunc().

7.32.2.2 int pX

Definition at line 33 of file main.cpp.

Referenced by draw_lanes(), and mouseFunc().

7.32.2.3 int pY

Definition at line 33 of file main.cpp.

Referenced by draw_lanes(), and mouseFunc().