Robot Planning and its application Project

Generated by Doxygen 1.9.2

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 student Namespace Reference

Classes

- struct configuration
- struct dubinsArc
- struct dubinsCurve
- struct triplet
- struct adjNode
- struct graphEdge
- class graph
- class myMotionValidator

Typedefs

- typedef bg::model::d2::point_xy< double > **point_type**
- typedef bg::model::polygon< point_type > **polygon_type**

Enumerations

- enum **dubins_primitives** {
  LSL , **RSR** , **LSR** , **RSL** ,
  RLR , **LRL** , **MAXIMUM_NUMBER_OF_CURVES** }

Functions

- bool coll_LineLine (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)
- bool coll_LineCircle (double x1, double y1, double x2, double y2, dubinsArc arc)
- double max (double a, double b)
- double **min** (double a, double b)
- double cross2D (double M_00, double M_01, double M_10, double M_11)
- double dot2D (double M_00, double M_01, double M_10, double M_11)
- void plotXLine (bool res, double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

- void plotXCircle (bool res, double x1, double y1, double x2, double y2, double xC, double yC, double radius, cv::Point a1, cv::Point a2, double verse)
- double sinc (double x)

  Sinc with Taylor series approximation, used to check correctness of solution.
- double mod2pi (double angle)

  Normalize an angle, [0, 2∗PI) (can be useful for findRobot as well)
- double rangeSymm (double angle)

  Normalize an angular difference (range (-pi, pi])
- bool check (double s1, double k0, double s2, double k1, double s3, double k2, double th0, double thf)

  Checks the correctness of a Dubin's solution by taking as input the scaled parameters.
- configuration getNextConfig (configuration curr, double k, double s)
- dubinsArc constructArc (configuration currConf, double k, double L)
- dubinsCurve constructDubinsCurve (configuration initialConf, double s1, double s2, double s3, double k0, double k1, double k2)
- void printConfiguration (configuration config)

  Prints info about the configuration of the robot (x,y, yaw).
- void printArc (dubinsArc arc)

  Prints info about arcs that compose the Dubin's Curve.
- std::chrono::high_resolution_clock::time_point startTime ()

  Returns this moment in time.
- void stopTime (std::chrono::high_resolution_clock::time_point start, bool unit)

  Gets the start moment and tells how much time passed between that and now.
- void scaleToStandard (configuration initial, configuration final, double kmax, double &scTh0, double &scThf, double &scKmax, double &lambda)

  Scale path finding problem into (-1,0) - (1,0) range.
- void scaleFromStandard (double lambda, double sc_s1, double sc_s2, double sc_s3, double &s1, double &s2, double &s3)

  Scale the solution to the standard problem back to the original problem.
- bool dubins_LSL (double sc_th0, double sc_thf, double sc_Kmax, double &sc_s1, double &sc_s2, double &sc_s3)
- bool dubins_RSR (double sc_th0, double sc_thf, double sc_Kmax, double &sc_s1, double &sc_s2, double &sc_s3)
- bool dubins_LSR (double sc_th0, double sc_thf, double sc_Kmax, double &sc_s1, double &sc_s2, double &sc_s3)
- bool dubins_RSL (double sc_th0, double sc_thf, double sc_Kmax, double &sc_s1, double &sc_s2, double &sc_s3)
- bool dubins_RLR (double sc_th0, double sc_thf, double sc_Kmax, double &sc_s1, double &sc_s2, double &sc_s3)
- bool dubins_LRL (double sc_th0, double sc_thf, double sc_Kmax, double &sc_s1, double &sc_s2, double &sc_s3)
- std::pair< int, dubinsCurve > dubins_shortest_path (configuration initial, configuration final, double Kmax)
- std::vector< dubinsCurve > multipoint (const configuration &robot, std::vector< Point > &points, double gate_th)
- void **plot** (cv::Mat image, double ∗∗pts_1, cv::Scalar c1, double ∗∗pts_2, cv::Scalar c2, double ∗∗pts_3, cv::Scalar c3)
- double ∗∗ **get_arc_points** (dubinsArc arc, int npts)
- void **plot_dubins** (std::vector< dubinsCurve > curves)
- void **printAdjList** (adjNode ∗ptr, int i)
- Point getCenter (const Polygon &poly)
- bool myStateValidityCheckerFunction (const ob::State ∗state)
- ob::ValidStateSamplerPtr **allocOBValidStateSampler** (const ob::SpaceInformation ∗si)
- void **drawSolutionTree** (std::vector< Point > RRT_list, cv::Mat &image)
- double **gate_angle** (const Polygon &gate, const Polygon &arena, double &gate_mid_w, double &gate_mid↩ _h)

- void missionTwo (std::shared_ptr< ompl::base::SpaceInformation > si, std::vector< Point > &point_list)
- void missionOne (std::vector< Point > &point_list, std::vector< Point > &RRT_list, std::shared_ptr< ompl::base::SpaceInformation > si, std::shared_ptr< ob::SE2StateSpace > space, cv::Mat &image)
- int find_closest (Point &self, std::vector< Point > &samples)

  Find the point closest to a given point.
- double distance (Point &a, Point &b)

  Returns an euclidean distance.
- bool student_planPath (const Polygon &borders, const std::vector< Polygon > &obstacle_list, const std::vector< std::pair< int, Polygon >> &victim_list, const Polygon &gate, const float x, const float y, const float theta, Path &path, const std::string &config_folder)
- void student_loadImage (cv::Mat &img_out, const std::string &config_folder)
- void student_genericImageListener (const cv::Mat &img_in, std::string topic, const std::string &config_folder)
- void student_imageUndistort (const cv::Mat &img_in, cv::Mat &img_out, const cv::Mat &cam_matrix, const cv::Mat &dist_coeffs, const std::string &config_folder)
- bool student_extrinsicCalib (const cv::Mat &img_in, std::vector< cv::Point3f > object_points, const cv::Mat &camera_matrix, cv::Mat &rvec, cv::Mat &tvec, const std::string &config_folder)
- void student_findPlaneTransform (const cv::Mat &cam_matrix, const cv::Mat &rvec, const cv::Mat &tvec, const std::vector< cv::Point3f > &object_points_plane, const std::vector< cv::Point2f > &dest_image_points_plane, cv::Mat &plane_transf, const std::string &config_folder)
- void student_unwarp (const cv::Mat &img_in, cv::Mat &img_out, const cv::Mat &transf, const std::string &config_folder)
- bool student_findRobot (const cv::Mat &img_in, const double scale, Polygon &triangle, double &x, double &y, double &theta, const std::string &config_folder, const bool DEBUG)
- bool student_processMap (const cv::Mat &img_in, const double scale, std::vector< Polygon > &obstacle_list, std::vector< std::pair< int, Polygon >> &victim_list, Polygon &gate, const std::string &config_folder, const bool DEBUG)
- void **get_arc_points** (double **pts, dubinsArc arc, int npts)
- void **loadImage** (cv::Mat &img_out, const std::string &config_folder)
- void **genericImageListener** (const cv::Mat &img_in, std::string topic, const std::string &config_folder)
- bool **extrinsicCalib** (const cv::Mat &img_in, std::vector< cv::Point3f > object_points, const cv::Mat &camera_matrix, cv::Mat &rvec, cv::Mat &tvec, const std::string &config_folder)
- void **imageUndistort** (const cv::Mat &img_in, cv::Mat &img_out, const cv::Mat &cam_matrix, const cv::Mat &dist_coeffs, const std::string &config_folder)
- void **findPlaneTransform** (const cv::Mat &cam_matrix, const cv::Mat &rvec, const cv::Mat &tvec, const std::vector< cv::Point3f > &object_points_plane, const std::vector< cv::Point2f > &dest_image_points_plane, cv::Mat &plane_transf, const std::string &config_folder)
- void **unwarp** (const cv::Mat &img_in, cv::Mat &img_out, const cv::Mat &transf, const std::string &config_folder)
- bool **processMap** (const cv::Mat &img_in, const double scale, std::vector< Polygon > &obstacle_list, std::vector< std::pair< int, Polygon >> &victim_list, Polygon &gate, const std::string &config_folder)
- bool **findRobot** (const cv::Mat &img_in, const double scale, Polygon &triangle, double &x, double &y, double &theta, const std::string &config_folder)
- bool **planPath** (const Polygon &borders, const std::vector< Polygon > &obstacle_list, const std::vector< std::pair< int, Polygon >> &victim_list, const Polygon &gate, const float x, const float y, const float theta, Path &path, const std::string &config_folder)
- void mouseCallback (int event, int x, int y, int, void *p)
- std::vector< cv::Point2f > pickNPoints (int n0, const cv::Mat &img)
- bool processGreen (const cv::Mat &img_hsv, cv::Mat &green_mask, bool DEBUG)

  Function to process the green color of the image that comes from the simulator.
- bool processObstacles (const cv::Mat &img_in, const cv::Mat &img_hsv, const double scale, std::vector< Polygon > &obstacle_list, bool DEBUG)

  Finds the red obstacles and saves them in the obstacle_list.
- bool processBorders (const cv::Mat &img_in, const cv::Mat &img_hsv, const double scale, std::vector< Polygon > &obstacle_list, bool DEBUG)

  Finds the black borders of the arena and treats them as trapezoidal obstacles.

- bool processGate (const cv::Mat &img_hsv, cv::Mat &green_mask, const double scale, Polygon &gate, std::vector< Polygon > &off_borders, bool DEBUG)

  Finds the green gate.
- bool processVictims (const cv::Mat &img_in, const cv::Mat &img_hsv, cv::Mat &green_mask, const double scale, std::vector< std::pair< int, Polygon >> &victim_list, bool DEBUG)

  Processes the green circles that correspond to the victims, to collect their position and their id.
- float findRobotRadius (const cv::Mat &img_in, const cv::Mat &img_hsv, const double scale, bool DEBUG)

  Processes the blue triangle of the robot to find the radius of the circle, in order to perform the offset.
- bool offsetObstacles (const float OFFSET, const cv::Mat &img_in, const double scale, const std::vector< Polygon > &found_obstacles, std::vector< Polygon > &offsetted_obstacles, const bool DEBUG)

  Uses Clipper to offset the found_obstacles all together, returing a list of processed_obstacles where they are merged.
- bool offsetEachObstacle (const float OFFSET, const cv::Mat &img_in, const double scale, std::vector< Polygon > &found_obstacles, std::vector< Polygon > &offsetted_obstacles, const bool DEBUG)

  Offsets each obstacle in found_obstacles at a time by using the Clipper library.
- int findTemplateId (cv::Mat &processROI, std::vector< cv::Mat > &templates, bool DEBUG)

  Processes a region of an image against the number templates, and returns the id of the best match.
- cv::Mat rotate (cv::Mat src, double angle)

  Function to rotate a part of an image by a certain angle, used when processing the victims and doing template matching.

Variables

- const bool **DEBUG_plan** = false
- double **SCALE**
- int dubins_primitives_ksigns [6][3]
- Polygon **this_borders**
- polygon_type **arena**
- polygon_type **valid_gate**
- std::vector< polygon_type > **boost_obstacle_list**
- std::vector< Polygon > **coll_obstacles**
- std::vector< Point > **full_tree**
- double **SOL_TIME** = 1.0
- cv::Mat **graph_image**
- bool **done** = false
- double ** **free_edges**
- const bool DEBUG_Map = false

  Debug variable for map processing.
- const bool DEBUG_Robot = false

  Debug variable for robot location.
- bool **DEBUG_VICT** = false
- int **additional_off** = 7

4.1.1 Detailed Description

Main file with the functions required by the simulator. The actual functions called are organized in different files for a cleaner code.

4.1.2 Function Documentation

4.1.2.1 coll_LineCircle()

```
bool student::coll_LineCircle (
            double x1,
            double y1,
            double x2,
            double y2,
            dubinsArc arc )
```

Detects collisions between a segment and an arc, identified as a circle

Parameters

x1	First x coordinate of the segment
y1	First y coordinate of the segment
x2	Second x coordinate of the segment
y2	Second y coordinate of the segment
arc	dubinsArc

Returns

The result of the collision check

Make positive if not

To avoid being considered as collisions

No intersection points found

2 intersection points with circle

1 collision point with circle, delta equal to zero

Find if t1 and t2 are a collision of the actual segment or just of the rect containing it

Real arc collision

Exit, collision found

Right turn

Real arc collision

Exit, collision found

Real arc collision

Exit, collision found

Make positive if not

To avoid being considered as collisions

No intersection points found

2 intersection points with circle

1 collision point with circle, delta equal to zero

Find if t1 and t2 are a collision of the actual segment or just of the rect containing it

Real arc collision

Exit, collision found

Right turn

Real arc collision

Exit, collision found

Real arc collision

Exit, collision found

4.1.2.2 coll_LineLine()

```
bool student::coll_LineLine (
             double x1,
             double y1,
             double x2,
             double y2,
             double x3,
             double y3,
             double x4,
             double y4 )
```

COLLISION_CORE

Detects collisions between a couple of segments, and returns the boolean outcome.

Parameters

x1	First x coordinate of the first segment
y1	First y coordinate of the first segment
x2	Second x coordinate of the first segment
y2	Second y coordinate of the first segment
x3	First x coordinate of the second segment
y3	First y coordinate of the second segment
x4	Second x coordinate of the second segment
y4	Second y coordinate of the second segment

Returns

The result of the collision check

4.1.2.3 constructArc()

```
dubinsArc student::constructArc (
            configuration currConf,
            double k,
            double L )
```

Returns a dubinsArc datastructure, which uniquely identifies the arc given its initial and final configurations, its length and curvature.

4.1.2.4 constructDubinsCurve()

```
dubinsCurve student::constructDubinsCurve (
            configuration initialConf,
            double s1,
            double s2,
            double s3,
            double k0,
            double k1,
            double k2 )
```

Constructs the whole Dubin's curve given the initial configuration, and the precomputed parameters of the three arcs. These are the length of the arcs - s1, s2, s3 - and their curvature - k0, k1, k2.

4.1.2.5 cross2D()

```
double student::cross2D (
            double M_00,
            double M_01,
            double M_10,
            double M_11 )
```

Computes the cross product of a 2-Dimensional matrix. Variables are named after their ficticious index in the matrix.

4.1.2.6 dot2D()

```
double student::dot2D (
            double M_00,
            double M_01,
            double M_10,
            double M_11 )
```

Computes the dot product of two arrays. Variables are named after their ficticious index in the matrix.

4.1.2.7 dubins_LRL()

```
bool student::dubins_LRL (
            double sc_th0,
            double sc_thf,
            double sc_Kmax,
            double & sc_s1,
            double & sc_s2,
            double & sc_s3 )
```

Function to implement the finding of the path made of a Left curve, followed by a
Right curve, followed by a Left curve. Takes as input the initial and final orientations, and the maximum curvature. It returns the length of the three segments.

4.1.2.8 dubins_LSL()

```
bool student::dubins_LSL (
            double sc_th0,
            double sc_thf,
            double sc_Kmax,
            double & sc_s1,
            double & sc_s2,
            double & sc_s3 )
```

Performs the computation of the Dubin's curve composed by an arc to the left, then a straight line, then an arc to the left. Takes as input the initial and final orientations, and the maximum curvature. It returns the length of the three segments.

4.1.2.9 dubins_LSR()

```
bool student::dubins_LSR (
            double sc_th0,
            double sc_thf,
            double sc_Kmax,
            double & sc_s1,
            double & sc_s2,
            double & sc_s3 )
```

Performs the computation to check for the Dubin's curve composed by an arc to the left, then a straight line, then an arc to the right. Takes as input the initial and final orientations, and the maximum curvature. It returns the length of the three segments.

4.1.2.10 dubins_RLR()

```
bool student::dubins_RLR (
            double sc_th0,
            double sc_thf,
            double sc_Kmax,
            double & sc_s1,
            double & sc_s2,
            double & sc_s3 )
```

Function to implement the finding of the path made of a Right curve, followed by a Left curve, followed by a Right curve. Takes as input the initial and final orientations, and the maximum curvature. It returns the length of the three segments.

4.1.2.11 dubins_RSL()

```
bool student::dubins_RSL (
            double sc_th0,
            double sc_thf,
            double sc_Kmax,
            double & sc_s1,
            double & sc_s2,
            double & sc_s3 )
```

Function to implement the finding of the path made of a Right curve, followed by a
Straight line, followed by a Left curve. Takes as input the initial and final orientations, and the maximum curvature. It returns the length of the three segments.

4.1.2.12 dubins_RSR()

```
bool student::dubins_RSR (
            double sc_th0,
            double sc_thf,
            double sc_Kmax,
            double & sc_s1,
            double & sc_s2,
            double & sc_s3 )
```

Performs the computation to check for the Dubin's curve composed by an arc to the right, then a straight line, then an arc to the right. Takes as input the initial and final orientations, and the maximum curvature. It returns the length of the three segments.

4.1.2.13 dubins_shortest_path()

```
std::pair< int, dubinsCurve > student::dubins_shortest_path (
            configuration initial,
            configuration final,
            double Kmax )
```

Solve the Dubins problem for the given input parameters. Return the type and the parameters of the optimal curve.

4.1.2.14 getCenter()

```
Point student::getCenter (
            const Polygon & poly )
```

Returns the Point struct with the coordinates of the passed Polygon's center

4.1.2.15 getNextConfig()

```
configuration student::getNextConfig (
            configuration curr,
            double k,
            double s )
```

Computes next configuration given the current one, the length of the arc and the curvature. Applied to get the nextConf in dubinsArc.

4.1.2.16 max()

```
double student::max (
            double a,
            double b )
```

COLLISION_UTILITY

4.1.2.17 missionOne()

```
void student::missionOne (
            std::vector< Point > & point_list,
            std::vector< Point > & RRT_list,
            std::shared_ptr< ompl::base::SpaceInformation > si,
            std::shared_ptr< ob::SE2StateSpace > space,
            cv::Mat & image )
```

Implementation of the mission one: collect all the victims in order and go to the gate, by avoiding the obstacles. Takes as input:

Parameters

point_list	The victims, in order, and the gate
RRT_list	the output list containing a series of states along the path
si	A variable of OMPL needed by the planner
space	Another variable of OMPL
image	An image for debugging purposes.

Close with the last segment

Free the memory from the previous goal TODO remove

Close with the last segment

Free the memory from the previous goal TODO remove

4.1.2.18 missionTwo()

```
void student::missionTwo (
            std::shared_ptr< ompl::base::SpaceInformation > si,
            std::vector< Point > & point_list )
```

Trial of implementing mission two, by connecting all the points found in one iteration of the RRTstar between each other (a sort of PRM).

4.1.2.19 mouseCallback()

```
void student::mouseCallback (
            int event,
            int x,
            int y,
            int ,
            void * p )
```

Saves the selected points into the result variable, for usage by the calibration function. (taken from professor_↩ interface.cpp)

4.1.2.20 multipoint()

```
std::vector< dubinsCurve > student::multipoint (
            const configuration & robot,
            std::vector< Point > & points,
            double gate_th )
```

Computes the Dubin's shortest path between a list of points, the first of which is the robot and the last of which is the gate. Both brute-froce approach and the optimized/iterative one are implemented.

4.1.2.21 myStateValidityCheckerFunction()

```
bool student::myStateValidityCheckerFunction (
            const ob::State * state )
```

State validity checker, used by the sampler in order to see if a point is valid.

Returns

true if the point is inside the arena and in a free area, or on the gate. False if it is outside the arena or on an obstacle.

4.1.2.22 pickNPoints()

```
std::vector<cv::Point2f> student::pickNPoints (
            int n0,
            const cv::Mat & img )
```

Function which asks the user to select the 4 points to be used for the calibration of the camera (the arena corners). (taken from professor_interface.cpp)

4.1.2.23 plotXCircle()

```
void student::plotXCircle (
            bool res,
            double x1,
            double y1,
            double x2,
            double y2,
            double xC,
            double yC,
            double radius,
            cv::Point a1,
            cv::Point a2,
            double verse )
```

Function to plot a possible collision between two lines If the image is scaled incorrectly, change the fact variable. Plot line, circle and the two arc points as circles, the smallest as the starting one.

Left turn

Left turn

Plot line, circle and the two arc points as circles, the smallest as the starting one.

Left turn

Left turn

4.1.2.24 plotXLine()

```
void student::plotXLine (
            bool res,
            double x1,
            double y1,
            double x2,
            double y2,
            double x3,
            double y3,
            double x4,
            double y4 )
```

COLLISION_PLOT

Function to plot a possible collision between two lines If the image is scaled incorrectly, change the fact variable.

4.1.2.25 student_extrinsicCalib()

```
bool student::student_extrinsicCalib (
            const cv::Mat & img_in,
            std::vector< cv::Point3f > object_points,
            const cv::Mat & camera_matrix,
            cv::Mat & rvec,
            cv::Mat & tvec,
            const std::string & config_folder )
```

Implementation of extrinsicCalib() functions. From 4 user-inserted points, it gets an estimate of the arena corners in 3D and computes the rotation and translation vectors to map them into 2D. These will be used to unwarm the image, transforming i t from 3d to 2d in a correct way. (Finds arena pose from 3D(object_points)-2D(image_in) point correspondences.) Returns true if the operation was successful.

4.1.2.26 student_findPlaneTransform()

```
void student::student_findPlaneTransform (
            const cv::Mat & cam_matrix,
            const cv::Mat & rvec,
            const cv::Mat & tvec,
            const std::vector< cv::Point3f > & object_points_plane,
            const std::vector< cv::Point2f > & dest_image_points_plane,
            cv::Mat & plane_transf,
            const std::string & config_folder )
```

Implementation of the findPlanTransform() function of the student_interface. Performs the 3D to 2D transformation of the arena.

4.1.2.27 student_findRobot()

```
bool student::student_findRobot (
            const cv::Mat & img_in,
            const double scale,
            Polygon & triangle,
            double & x,
            double & y,
            double & theta,
            const std::string & config_folder,
            const bool DEBUG )
```

Student implementation of the findRobot() function. It processes the blue area of the arena, identifies the triangle which represents the robot, processes it to find its center (x,y) and orientation (theta). If DEBUG is true, more information about the process is printed and images are shown.

4.1.2.28 student_genericImageListener()

```
void student::student_genericImageListener (
            const cv::Mat & img_in,
            std::string topic,
            const std::string & config_folder )
```

Implementation of the genericImageListener function from the student_interface. It saves the img_in into the config_folder/camera_image directory, upon request of the user. Press 's' when the image is shown in order to save it.

4.1.2.29 student_imageUndistort()

```
void student::student_imageUndistort (
            const cv::Mat & img_in,
            cv::Mat & img_out,
            const cv::Mat & cam_matrix,
            const cv::Mat & dist_coeffs,
            const std::string & config_folder )
```

Implementation of the imageUndistort() function of student_interface. It takes the img_in and performs the undistortion based on the camera parameters.

4.1.2.30 student_loadImage()

```
void student::student_loadImage (
            cv::Mat & img_out,
            const std::string & config_folder )
```

Implementation of the loadImage() function of the student_interface. It takes the imges from the config_↩ folder/camera_image folder and loads them into the simulator.

4.1.2.31 student_planPath()

```
bool student::student_planPath (
            const Polygon & borders,
            const std::vector< Polygon > & obstacle_list,
            const std::vector< std::pair< int, Polygon >> & victim_list,
            const Polygon & gate,
            const float x,
            const float y,
            const float theta,
            Path & path,
            const std::string & config_folder )
```

Performs the computation of a collision free path, that goes through all the victims's in order (mission 1) by using Dubin's manoeuvres. It uses the RRTstar library of OMPL. Close the arena polygon

Close the obstacle polygon

Close the arena polygon

Close the obstacle polygon

4.1.2.32 student_processMap()

```
bool student::student_processMap (
            const cv::Mat & img_in,
            const double scale,
            std::vector< Polygon > & obstacle_list,
            std::vector< std::pair< int, Polygon >> & victim_list,
            Polygon & gate,
            const std::string & config_folder,
            const bool DEBUG )
```

Implementation of the processMap() function of the student_interface. It takes the img_in coming from the simulator or the camera, processes it to find the obstacles that the robot has to avoid and the victims that it needs to save. It uses some helper functions that divide the processing in a more structured way.

4.1.2.33 student_unwarp()

```
void student::student_unwarp (
            const cv::Mat & img_in,
            cv::Mat & img_out,
            const cv::Mat & transf,
            const std::string & config_folder )
```

Implementation of the unwarp() function of the student_interface. It performs the last step in image processing which unwarps the image according to the transformation matrix previously computed.

4.1.3 Variable Documentation

4.1.3.1 dubins_primitives_ksigns

```
int student::dubins_primitives_ksigns[6][3]
```

Initial value:

```
= {
        {  1,  0,  1 },
        { -1,  0, -1 },
        {  1,  0, -1 },
        { -1,  0,  1 },
        { -1,  1, -1 },
        {  1, -1,  1 }
    }
```

Definition of the curvatue signs corresponding to the different dubins primitives functions.

Chapter 5

Class Documentation

5.1 student::adjNode Struct Reference

Public Attributes

- int **id**
- int **cost**
- adjNode * **next**

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

- include/path_functions.hpp

5.2 student::configuration Struct Reference

Public Attributes

- double **x**
- double **y**
- double **th**

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

- include/dubins_functions.hpp

5.3 student::dubinsArc Struct Reference

Public Attributes

- configuration **currentConf**
- double **len**
- double **k**
- configuration **nextConf**

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

- include/dubins_functions.hpp

5.4 student::dubinsCurve Struct Reference

Public Attributes

- dubinsArc **a1**
- dubinsArc **a2**
- dubinsArc **a3**
- double **L**

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

- include/dubins_functions.hpp

5.5 student::graph Class Reference

Public Member Functions

- **graph** (std::vector< graphEdge > &edges, int n, int N)

Public Attributes

- adjNode ** **head**

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

- include/path_functions.hpp

5.6 student::graphEdge Struct Reference

Public Attributes

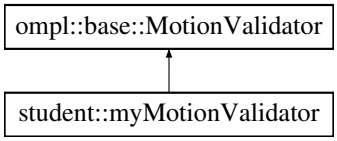
- int **startNode**
- int **endNode**
- double **weight**

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

- include/path_functions.hpp

5.7 student::myMotionValidator Class Reference

Inheritance diagram for student::myMotionValidator:



Public Member Functions

- **myMotionValidator** (ob::SpaceInformation ∗si)
- **myMotionValidator** (const ob::SpaceInformationPtr &si)
- bool checkMotion (const ob::State ∗s1, const ob::State ∗s2) const override
- bool **checkMotion** (const ob::State ∗s1, const ob::State ∗s2, std::pair< ob::State ∗, double > &lastValid) const override

5.7.1 Detailed Description

Class for motion validation, used by the planner to check if two states can be connected. It relies on our implementation of collision checking.

5.7.2 Member Function Documentation

5.7.2.1 checkMotion()

```
bool student::myMotionValidator::checkMotion (
            const ob::State * s1,
            const ob::State * s2 ) const  [inline], [override]
```

Close with the last segment

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

- path/student_planPath.cpp

5.8 student::triplet Struct Reference

Public Attributes

- int **id**
- int **next**
- double **cost**

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

- include/path_functions.hpp

Index