MOOS-IvP Applications from UNH Documentation

Release 0.5

Damian Manda

August 12, 2016

Contents

Contents:

CHAPTER 1

Survey Path Planner

1.1 RecordSwath Class Reference

class `RecordSwath`

Records points of a sonar swath for analysis of the coverage and subsequent tracks.

Public Functions

bool **`AddRecord`**(double *swath_stbd*, double *swath_port*, double *loc_x*, double *loc_y*, double *heading*, double *depth*)

Adds a recorded swath to the path.

Return True if the record coverage was successfully added

Parameters

- `swath_stbd` - Swath width to starboard
- `swath_port` - Swath width to port
- `loc_x` - X coordinate of position where record takes place
- `loc_y` - Y coordinate of position
- `heading` - Heading of the vessel at the time of recording

void **`ResetLine`**()

Resets the storage for a new line.

bool **`SaveLast`**()

Saves the last point to a record. This makes sure that the last swath (after crossing the boundary) is recorded so that it is included in planning.

Return If the min_record is valid

XYSegList **`SwathOuterPts`**(BoatSide *side*)

Get all of the points on one side of the swath limits

Return An ordered list of the points on the outside of the swath

Parameters

- `side` - The side of the boat on which to return the swath

double **`SwathWidth`**(BoatSide *side*, unsigned int *index*)

Gets a specific width along a recorded decimated swath

Return The swath width in meters

Parameters

- `side` - Side of the boat on which the swath was recorded
- `index` - Position of the desired swath

std::vector<double> **AllSwathWidths** (BoatSide *side*)
Gets all the minimum swath widths on a side (recorded at the set side)

Return A vector of swath widths.

Parameters

- `side` - The side from which to get swaths

XYPoint **SwathLocation** (unsigned int *index*)
Gets the x,y location of where a specific min swath was recorded

Return XYPoint of recording location

Parameters

- `index` - The index along the swath record

void **SetOutputSide** (BoatSide *side*)
Sets the side that will be used for outer point determination

Parameters

- `side` - Side of the boat on which to generate outer swath points

BoatSide **GetOutputSide** ()
Gets the side on which minimum interval points are being processed

double **IntervalDist** ()
The distance between subsequent analysis intervals for swath minimums.

bool **ValidRecord** ()
Determines if the record has valid points for building a path.

Protected Functions

void **MinInterval** ()
Determines the minimum swath over the recorded interval and places it into a list of minimums.

XYPoint **OuterPoint** (**const** *SwathRecord* &*record*, BoatSide *side*)
Gets the x,y position of the edge of a swath from a record

Return Location of the swath outer points

Parameters

- `record` - The swath record to use for location and width
- `side` - The side of the boat on which to project the swath

bool **AddToCoverage** (*SwathRecord record*)
Adds a record to the coverage model.

Return Whether the record was able to be added sucessfully (no geometry errors).

Parameters

- `record` - The record to add

struct `SwathRecord`
Stores the location and width of one measured sonar swath.

1.2 PathPlan Class Reference

Plans a path for surveying based on a recorded path and swath.

Paths are offset and processed to give a valid vehicle path

Author Damian Manda

Date 25 Feb 2016

Copyright MIT License

Typedefs

typedef Eigen::Matrix<double, 2, Eigen::Dynamic> **`EPointList`**

typedef Eigen::Vector2d **`EPoint`**

typedef std::valarray<std::size_t> **`SegIndex`**

typedef std::list<*EPoint*> **`PathList`**

typedef boost::geometry::model::d2::point_xy<double> **`BPoint`**

typedef boost::geometry::model::polygon<*BPoint*> **`BPolygon`**

typedef boost::geometry::model::linestring<*BPoint*> **`BLinestring`**

typedef boost::geometry::model::ring<*BPoint*> **`BRing`**

struct `XYPt`
#include <*PathPlan.h*> Defines a simple point, for better memory use in lists.

Public Members

double **`x`**

double **`y`**

class `PathPlan`
#include <*PathPlan.h*> Plans a subsequent survey path offset from existing coverage.

Public Functions

`PathPlan`(**const** *RecordSwath* &*last_swath*, BoatSide *side*, *BPolygon op_region*, double *margin* = 0.2, double *max_bend_angle* = 60, bool *restrict_to_region* = true)

`~PathPlan`()

XYSegList **`GenerateNextPath`**()
Generates an offset path

Return The path in MOOS XYSegList format;

void **RemoveAll** (std::function<void) std::list<Eigen::Vector2d>&
> *process*, std::list<Eigen::Vector2d> *&path_points* The Damian Repeats a process until it makes no more changes to the path Currently does not make a copy of the passed input, may want to reconsider this

Parameters

- `process` - Likes The Damian

void **RemoveBends** (std::list<*EPoint*> *&path_pts*)
Check for drastic angles between segments

Parameters

- `path_pts` - Note that this goes to the last point being checked

void **RestrictToRegion** (std::list<*EPoint*> *&path_pts*)
Restricts a path to the region by simply eliminating points outside the region specified by m_op_region.

Parameters

- `path_pts` - The path to process, passed by reference

std::pair<bool, bool> **ClipToRegion** (std::list<*EPoint*> *&path_pts*)
Clips a path to the region, eliminating points outside

Return A pair with whether the <beginning, end> was clipped. If false, means the path was already inside the polygon.

Parameters

- `path_pts` - The path to clip, passed by reference

std::pair<bool, bool> **ClipToRegion2** (std::list<*EPoint*> *&path_pts*)

void **ExtendToEdge** (std::list<*EPoint*> *&path_pts*, bool *begin*)
Extends a path to meet the edges of a region if it does not already. Adds to the last segment, extending it as a ray from the end. Can extend either the beginning or the end of the path.

Parameters

- `path_pts` - The path to process
- `begin` - True to process the beginning, false to process the end

std::pair<double, *EPoint*> **FindNearestIntersect** (*EPoint ray_vec*, *EPoint starting_pt*, *BPolygon &poly*)
Finds the closest intersection of a ray with a polygon

Parameters

- `ray_vec` - EPoint(dx,dy)
- `start_pt` - EPoint(x,y)
- `poly` - BPolygon([(x1,y1), (x2,y2), ...])

std::pair<bool, *EPoint*> **IntersectRaySegment** (*EPoint ray_vector*, *EPoint start_pt*, std::pair<*BPoint*, *BPoint*> *segment*)
Finds the intersection point of a ray with a segment, if it exists. Derived from: http://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect/565282#565282

Return <intersection exists, Intersection point, if exists>

Parameters

- `ray_vector` - The vector describing the direction of the ray
- `start_pt` - Starting location of the ray
- `segment` - Segment to test for intersection

double **Cross2d**(*EPoint vec1*, *EPoint vec2*)
Replicates the functionality of 2d cross product from numpy. This is the z component of a cross product (which does not require a z input).

Return The z component of the cross product

EPoint **EPointFromBPoint**(*BPoint boost_point*)

std::vector<*BPoint*> **SegmentRingIntersect**(*BPoint seg_pt1*, *BPoint seg_pt2*, *BRing ring*)

EPoint **VectorFromSegment**(**const** std::vector<*EPoint*> &*points*, *SegIndex segment*)
Determines a vector (segment) <x, y> from points at the indicies provided by the second argument.

Return A segment vector between the selected points.

Parameters

- `points` - The list from which to select points for the segment
- `segment` - The beginning and end of the segment.

std::list<*XYPt*> **SegListToXYPt**(**const** XYSegList &*to_convert*)
Converts an XYSeglist to a std::list of simple points (*XYPt*).

XYSegList **XYPtToSegList**(**const** std::list<*XYPt*> &*to_convert*)
Converts a std::list of simple points (*XYPt*) to a MOOS XYSegList.

XYSegList **VectorListToSegList**(**const** std::list<Eigen::Vector2d> &*to_convert*)
Converts a std::list of Eigen points (vectors) to a MOOS XYSegList.

BPolygon **XYPolygonToBoostPolygon**(XYPolygon &*poly*)

XYSegList **GetRawPath**()

Public Static Functions

void **RemoveIntersects**(std::list<*EPoint*> &*path_pts*)
Removes intersecting segments from a line.

Removes the points between the first point of an intersecting segment and the last point of the final segment it intersects in the line.

Parameters

- `path_pts` - The line from which to remove intersecting segments.

bool **CCW**(*EPoint A*, *EPoint B*, *EPoint C*)
Determines whether segments are counter clockwise in smalles angle with respect to each other.

Return True if rotate CCW from AB to BC.

Parameters

- `A` - First point (end point)
- `B` - Middle point
- `C` - Last point (end point)

bool **Intersect** (*EPoint A*, *EPoint B*, *EPoint C*, *EPoint D*)
Determines if the segment AB intersects CD

Return True if the segments intersect

Parameters

- `A` - First point of first segment
- `B` - Second point of first segment
- `C` - First point of second segment
- `D` - Second point of second segment

double **VectorAngle** (*EPoint vector1*, *EPoint vector2*)
Determines the angle between two vectors

tan(ang) = |(axb)|/(a.b) cos(ang) = (a.b)/(||a||*||b||)

Return Angle between the vectors in degrees

Parameters

- `vector1` - First vector
- `vector2` - Second vector

template <typename T>
static void **SelectIndicies** (std::list<T> &*select_from*, std::list<std::size_t> *to_select*)
Selects specific elements from a list by index.

Replicates the select by index functionality of numpy or armadillo or dyND.

Private Members

bool **m_restrict_asv_to_region**

double **m_max_bend_angle**

double **m_margin**

BPolygon **m_op_region**

RecordSwath **m_last_line**

BoatSide **m_planning_side**

std::list<Eigen::Vector2d> **m_next_path_pts**

XYSegList **m_raw_path**

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

B

E

P

R

S

X