libvstp — Very Simple Trajectory Planner

Anthony Mallet, Aurélien Kamel

November 2003

1 Purpose

libvstp (which stands for "very simple trajectory planner") is a *very* simple geometric path planning C library. It uses state of the art techniques which do not take into account robot kinematics nor dynamics. The robot is supposed to be a disk, of a configurable diameter. Thus, this software is for people who are looking to basic path planning algorithms and want to focus on other proplems than path planning.

This library is intented to be an efficient way to get the shortest path from one point to another, in a 2D environment cluttered with obstacles, modeled with 2D polylines. The path is computed by making a search in a visibility graph where the nodes are the end points of the obstacles and the edges link together the nodes which can be reach by a straight line. In order to optimize the visibility graph computation, a 2D grid is used to hash the segments of the obstacles.

The source code is available for download here :
`http://softs.laas.fr/openrobots/path/vstp.php`

2 User interface

Vstp comes with an optional user interface, written in Tk. This interface is mostly for debugging purpose, so the goal is to keep is as simple as possible. To use it, simply launch your preffered wish interpreter (an interpreter for the Tk language). The interface package is located in the `$libdir` you configured during the compilation of the packages, in the subdirectory `vstpinterface`. It can be loaded with the `load` TCL command:

```
tcl> load your_lib_dir/vstpinterface/libvstpinterface.so
```

In the same directory, a sample map `exmap.xml` is provided. You can load it from the graphical interface.

3 Files format

Vstp reads list of segments in files. These files must be written in XML, with the following structure:

```
<map>
   <segment>
      <point><x> vvv.vv </x><y> vvv.vv </y></point>
      <point><x> vvv.vv </x><y> vvv.vv </y></point>
   </segment>

   <segment>
      ...
   </segment>

   ...
</map>
```

Segments are considered as a frontier between free space and occupied (obstacle) space. Thus, they are oriented: the order in which the two points appear is important. The orientation is such the free space is located on your right hand when you go from the first point to the second. `vvv.vv` in the map file represents the actual coordinate of the point, in floating point format.

4 Data structures

typedef struct vstp_data vstp_data;

vstp_data is an opaque type which is used to maintain a context between the different API functions calls. You should not try to modify its value nor to access the structure members. You can use automatic storage to keep a vstp_data, although passing a value which was not returned by a call to vstp_DataInit() is not supported.

```
typedef struct vstp_segment {
   double x0, y0;
   double x1, y1;
} vstp_segment;
```

The vstp_segment structure is used to pass to or get from functions that manipulate such objects an array of 2D segments. x0 and y0 represent the coordinates of the first point; x1 and y1 those of the second.

```
typedef struct vstp_traj {
   double x, y;
   struct vstp_traj *next;
} vstp_traj;
```

This structure is a linked list of points which represent a planned trajectory. x and y are the coordinates of the current point and next is a reference to the next point, or NULL for the last one.

5 API reference

vstp_data *vstp_DataInit(double rRadius, double gridSize, double idealDist, double maxDist);

double rRadius	Robot radius, in meters.
double gridSize	Grid resolution, in meters.
double idealDist	Ideal distance to obstacles, in meters.
double maxDist	Maximum distance to obstacles, in meters.

Creates a vstp context and returns it. You must keep the returned value and pass it to the other functions. The grid resolution (gridSize) is the size of the hashing array: the ideal value depends on the density of the environment, but you should typically start with a value close to the robot radius. The idealDist and maxDist parameters are used for improving trajectories which will otherwise pass very close to the obstacles.

void vstp_DataFree(vstp_data *vstpdata);

vstp_data *vstpdata	VSTP context.

Destroys a vstp context and release any storage associated to it. You cannot use the vstpdata anymore once this function returns.

int vstp_SetRobotRadius(vstp_data *vstpdata, double rRadius);

vstp_data *vstpdata	VSTP context.
double rRadius	Robot radius, in meters.

Changes the current robot radius. Note that this invalidates the visibility graph and a call to vstp_RefreshGraph must be done after this function and before a call to vstp_PlanTraj.

int vstp_SetGridSize(vstp_data *vstpdata, double gridSize);

vstp_data *vstpdata	VSTP context.
double gridSize	Grid resolution, in meters.

Changes the current grid size (hashing array). Note that this invalidates the visibility graph and a call to vstp_RefreshGraph must be done after this function and before a call to vstp_PlanTraj.

vstp_segment *vstp_GetSegFromMapFile(char *map, int *size);

char *map	File name containing a segment file.
int *size	Number of segments read.

This reads a segment list in the file map and returns an array of size segments. You should not try to modify the actual content of the array (unless you want some unexpected results). Segments are kept internally, so you do not need to free the memory associated to the array.

int vstp_AddSegPkg(vstp_data *vstpdata, vstp_segment *seg, int n, int type);

vstp_data *vstpdata	VSTP context.
vstp_segment *seg	Segment list.
int n	Number of segments.
int type	Type of segments.

This adds the array seg of n segments to the current vstp map. Segments can be of two types: VSTP_SEGMENT for actual obstacles (walls, doors and so forth) or VSTP_OBSTACLE for virtual obstacles (forbidden areas, ...). The function returns an id (integer) which can be used to further delete the list of segments. A call to vstp_RefreshGraph must be done after this function and before a call to vstp_PlanTraj.

int vstp_DelSegPkg(vstp_data *vstpdata, int id);

vstp_data *vstpdata	VSTP context.
int id	Reference of segments to be removed.

Remove a list of segments from the current map. The id must be one returned by a previous call to vstp_AddSegPkg. Once the segments are deleted, vstp_RefreshGraph must be called before vstp_PlanTraj.

int vstp_RefreshGraph(vstp_data *vstpdata);

vstp_data *vstpdata	VSTP context.

Recompute the visibility graph. This can take a while, but it needs only to be called once. Some functions like vstp_AddSegPkg invalidate the current graph: they are explicitely mentionned as such in their synopsis.

vstp_traj *vstp_PlanTraj(vstp_data *vstpdata, double x0, double y0, double x1, double y1, int improve);

vstp_data *vstpdata	VSTP context.
double x0	X coordinate of initial position.
double y0	Y coordinate of initial position.
double x1	X coordinate of final position.
double y1	Y coordinate of final position.
int improve	Whether to try to improve the trajectory or not.

This is the main function, which plans a trajectory from the point (x0,y0) to (x1, y1). If improve is not zero, the resulting trajectory is moved away from the obstacles where possible. This option is still experimental and in some situations you might get weird trajectories. To be sure to obtain the shortest path, use improve = 0.