The Stable Pushing Planner

Kevin M. Lynch Constantinos Nikou Matthew T. Mason
The Robotics Institute
Carnegie Mellon University
August 20, 1995

1 Overview

This document describes the code implementing the stable pushing planner presented in "Stable Pushing: Mechanics, Controllability, and Planning," by Kevin M. Lynch and Matthew T. Mason. This paper is to appear in the International Journal of Robotics Research and is available on the World Wide Web at `http://www.cs.cmu.edu/afs/cs/usr/lynch/research/kml-stable.html`.

The code allows a user to automatically find pushing plans to move an object among obstacles. The strategy uses stable pushes, which keep the pushed object fixed to the pusher as it moves. The pusher is assumed to be a flat edge. The code consists of three major portions:

1. The object editor, which allows the user to enter a polygonal object using a graphical interface. The user must specify the center of mass of the object, the coefficient of friction at the pushing contacts, and the edges of the object that the robot can push. The user may specify that the pusher act on the convex hull of the object. The editor then automatically determines the set of stable pushing directions for each edge, for use in the pushing planner.

2. The pushing world editor, which allows the user to specify pushing problems for the objects created with the object editor. This consists of drawing the obstacles and specifying the start and goal positions. The editor also invokes the planner, and if a solution is found, it displays and animates the path.

3. The push planner. This is normally invoked by the world editor, but it can also be used from the command line. The planner also (if desired) writes an Adept V+ file to execute the pushing path on an Adept robot.

The object and world editors are written in Tcl/Tk, and the planner is written in C. For best results, the editor should be run on a fast machine with a color monitor and plenty of swap space.

2 The object editor

The object editor is a graphical interface that allows the user to analyze objects for pushing. The object editor is invoked from within the world editor, described in the next section.

The polygonal object is entered by mouse clicks. The center of mass (technically, its center of friction) is assumed to be at (0, 0), and the user must specify the coefficient of friction at

the pushing contacts. By clicking on an edge of the object, the user marks (or unmarks) it as a possible edge for pushing. The user may also choose to use the pushing edges defined by the convex hull of the object.

Once the user has entered the object, specified the friction coefficient, and chosen a set of possible pushing edges, the set of stable pushing directions for each edge can be calculated. The planner actually uses only the extreme velocity directions, as described in the paper. These velocity directions are displayed as (v_x, v_y, ω), where v_x is the x velocity of the center of mass, v_y is the y velocity, and ω is the angular velocity. When the user runs the analysis, lines will appear on the screen that bound the sets of stable rotation centers. These lines are described in Figure 11 of the paper. They are provided to aid the user's intuition when checking the stable pushing directions for a single edge.

Once a set of pushing edges has been chosen and the stable pushing directions have been calculated, the user can save the object to the file `objects.tcl` read by the world editor. The object is now available for specifying pushing problems. In practice, only a small number of pushing edges should be used, as the time and space required by the search increases with the number of available pushing motions. It is desirable, however, that the set of pushing edges yield *small-time local controllability*, so the object can be maneuvered in tight spaces. See the paper for details.

Some details to be aware of:

- There is no function to load a previously analyzed object into the object editor, or for deleting an object from `objects.tcl`. The user can do this manually by editing the file.
- The editor may be excessively slow if the grid spacing (under `Options`) is too small. This is because the editor must draw crosses at each grid point.

2.1 Global variables

Colors used in the object editor are defined in `editcolors.tcl`. The following global variables are defined in `obeditglobals.tcl`.

SAFETY As described in Figure 11(b) of the paper, the lines a distance r^2/p from the center of mass of the object should be slightly more distant. This is accounted for by multiplying r^2/p by the constant SAFETY. The default value is 1.0.

FUZZY This constant is used for fuzzy math, to account for loss of precision by floating point math. For example, to test if a point lies on a line, we actually only require that it lies very close to it.

FuzzyDist If two extremal stable rotation centers are within a distance FuzzyDist of each other, they are assumed equivalent, and one of them is discarded.

MaxRCDist If a stable rotation center is greater than MaxRCDist from the center of mass of the object, then it is essentially a translation, and it is discarded.

3 The world editor

The world editor allows the user to draw obstacle fields, move the start and goal position of the object, invoke the planner, display the results of the planner, and print them to a postscript file. The interface should be mostly self-explanatory.

The `Options` button allows the user to change the size of the pushing world and the resolution of the snap grid. The user can also specify the cost function for the planner. The cost of the plan is given by the cost of the action (pushing direction) changes, the pushing edge changes, and the number of pushes. The cost for each of these must be integral, as described in the paper. The user may also specify the size of the goal region, the pushing step length, and the size of the configuration space grid used to check for prior occupancy. (All angles are specified in degrees.) We recommend using the default goal size, step length, and grid size to ensure "reasonable" run times for the planner.

Some things to know:

- In most text entries (except the Load, Save, and PS file windows) pressing Return will not secure the entry, but will write a string similar to "\0xd". This string *will* be included in the entry, so it is possible to enter, for example, a grid spacing of "3.0\0xd". In fact, all entries will accept any string. So entering a grid spacing of "Bob" will not result in a editor error, but the planner will give an error (or strange result). Also, highlighting text in an entry has no use nor function.

- With Polygon entry, if MaxVertices is set to 10, sometimes the user can enter a polygon with up to 10 vertices, other times up to 9. This has to do with the way Tcl/Tk deals with lists of coordinates for a polygon.

- The animate function will use the last path (default: `push.path`) created. Also, running the planner will write a temporary file called `planner.tmp`.

- Obstacle rotation always begins with 0 degrees, meaning if you rotate an obstacle, close the rotation window, and decide to rotate the object again, the CURRENT position will be considered 0 degrees.

- When saving both a problem and its path, the extension of the problem name is ignored when writing to the path. For example, a problem called "test.problem" would have an associated path name of "test.path". A problem called "test.fred" would also have an associated path name of "test.path", so different problem names should not only differ by filename extension.

3.1 Global variables

Colors used in the object editor are defined in `editcolors.tcl`. The world editor uses the following global variables, specified in `globals.tcl`.

ColorFile The editor was designed to be used on a color monitor. If you must use a monochrome monitor, however, set this variable to `editbw.tcl` instead of `editcolors.tcl`. This will make the editor easier to use.

PMConversion The conversion factor from screen pixels to "virtual millimeters," where "millimeters" = pixels/PMConversion. (Note that the unit is millimeters for use with the Adept. Otherwise the scale does not matter.)

XMAX Default size of the world in the x direction, in millimeters.

YMAX Default size of the world in the y direction, in millimeters.

MaxObstacles Maximum number of obstacles that the editor will allow to be placed in the world.

MaxVertices Maximum number of vertices that an obstacle can have.

4 The push planner

The push planner `pplanner` is made from `pplanner.c`. It is normally invoked by the world editor, but can also be invoked from the command line by `pplanner [-adept]` <*infile*>, where *infile* is the problem specification file. If no file is specified, the planner defaults to the file `push.problem`. If the `-adept` flag is given, the planner will write a V+ pushing program for an Adept robot.

The planner is a simple best-first search as described in the paper. In this implementation, we only consider collisions between the object and the obstacles; we ignore collisions between the pushing robot and the obstacles. The robot is essentially assumed to be an infinitely thin pushing surface.

4.1 Global variables

The planner uses the following global variables, defined in `pglobals.h`.

INFILE Problem file used by planner if no other file is specified (only relevant when running planner from the command line).

CONTROLOUT Name of the control file written by the planner.

PATHOUT Name of the path file written by the planner (must be the same as editor global PathOut).

MAXOBSTACLES Maximum number of obstacles allowed in the world (must not be less than editor global MaxObstacles).

MAXVERTICES Maximum number of vertices allowed in an obstacle (must not be less than editor global MaxVertices).

MAXCONTROLS Maximum number of different velocities one pushing edge can be associated with.

MAXEDGES Maximum number of edges an object can have.

MAXCOST Upper limit on allowable cost of a pushing plan.

ROBOTPOLY Maximum number of polygons in the object representation.

MAXNODES Maximum number of search nodes. This value should be as large as possible (preferably 1 million or more) subject to memory limitations.

XCELLS The maximum number of cells along the x-axis of the configuration space grid. The size of each cell in the x direction, given in the world editor, should be sufficiently large that XCELLS cells covers XMAX, the x size of the world.

YCELLS The maximum number of cells along the y-axis of the configuration space grid. The size of each cell in the y direction, given in the world editor, should be sufficiently large that YCELLS cells covers YMAX, the y size of the world.

THETACELLS The maximum number of cells along the θ-axis of the configuration space grid. The size of each cell in the θ direction, given in the world editor, should be sufficiently large that THETACELLS cells covers 2π.

TOOLOFFSET For use with Adept files. Distance between the center of the gripper and the plane of the pusher.

BACKUP For use with Adept files. Distance to place the pusher behind the object when lowering/raising the pusher.

5 Running plans on an Adept robot

The planner produces Adept programs by copying `prog.header`, then writing the plan, and finally closing the program by copying `prog.trailer`. The final file is `push.pg`.

In our lab, we execute pushing plans with an Adept robot by doing the following:

1. Create a pushing problem, solve it, and write the Adept file.

2. Print out a postscript version of the world and place it at a fixed position in the Adept's workspace. Place the object at the start position on the paper world.

3. Transfer the file `push.pg` to an Adept directory with the file `zero.pg`. `zero.pg` handles the case where joint 4 (gripper orientation) of the Adept reaches a joint limit during execution.

4. Load the two files into memory and run the program by `execute pushing`.

To reproduce this setup in your lab, modify the file `prog.header` so that the BASE definition specifies the origin of your pushing world. In the file `pglobals.h`, set TOOLOFFSET to whatever is appropriate for your pusher. `prog.header` and `prog.trailer` assume that the pusher is sitting at the location `pusher`, and the Adept picks up the pusher at the beginning of the plan and places it back at the same position at the end. Modify these two files according to your setup.

The dimensions of the printed world will be XMAX by YMAX in millimeters.