Visual Waypoint Navigation Algorithm

Created: May 14, 2001; Last modified: May 14, 2001

This document covers the trainable visual waypoint navigation algorithm as implemented in WaypointMoveGenerator.

1 General description

This algorithm performs navigation using color visual cues in the environment. Given a series of color values, it searches the environment for each color in order. When it finds a color waypoint in its visual field, it moves towards it until it is “close enough” to be considered “at the waypoint”. After reaching a waypoint, the algorithm moves on to search for the next waypoint.

The algorithm supports learning waypoint colors before starting navigation. During the optional learning phase, the trainer presents a example of each color in the center of the visual field, and the colors are sampled and stored as the target waypoint colors. After learning all waypoints, the algorithm performs navigation.

2 Algorithm input

2.1 Configuration input

The algorithm reads configuration values from a text file at startup. The following values must be specified in the configuration file.

Value name	Legal values	Description
horizontal_view_angle	floating point, in $[0, 360]$	the number of degrees spanned horizontally by the camera's visual field
tolerance	floating point, in $[0, 1]$	the amount a pixel can vary in hue and still be considered as part of a waypoint
see_waypoint_fraction	floating point, in $[0, 1]$	fraction of pixels in the view that must match the waypoint color (with tolerance) before we assume the waypoint is present in our current view
at_waypoint_fraction	floating point, in $[0, 1]$	fraction of pixels in the view that must match the waypoint color (with tolerance) before we consider ourselves to be "at" the waypoint
loop	integer, in $[0, 1]$	controls whether we loop through the waypoints
learn_waypoint_colors	integer, in $[0, 1]$	controls whether we go through a learning phase before navigation
learning_start_delay	integer, in $[0, \mathrm{intMax}]$	the amount of time to delay in seconds before starting the learning phase
between_color_learning_delay	integer, in $[0, \mathrm{intMax}]$	the amount of time to delay in seconds between learning each waypoint color
number_of_waypoints	integer, in $[0, \mathrm{intMax}]$	the number of waypoints
waypoint_0_hue	floating point, in $[0, 1]$	the HSB hue value of waypoint 0
waypoint_1_hue	floating point, in $[0, 1]$	the HSB hue value of waypoint 1
$\vdots$		

2.2 Data input

For each move generated, the algorithm takes a color stereo image pair and a depth map as input. Only the left channel image is used by the algorithm.

3 Learning implementation

During the optional learning phase, the algorithm samples the input images and uses the samples to configure the waypoint color values.

Before sampling for a particular waypoint, the algorithm sends a *flashLight* move and specifies a number of flashes corresponding to the waypoint number being learned. Immediately following these flashes, an image is taken from the camera. The center hundredth of the image is analyzed (a tenth of the image width, and a tenth of the image height), and an average is taken over all pixels in this region to produce the waypoint color.

After learning all waypoints, the algorithm moves immediately to the navigation phase.

4 Navigation implementation

4.1 Waypoint search

For each waypoint, the algorithm performs a search of the visual environment. While not enough pixels match the waypoint color (as specified by *see_waypoint_fraction*), the algorithm repeatedly rotates in one direction by half its visual field angle.

4.2 Approaching waypoints

When more than *see_waypoint_fraction* pixels in the visual field match the waypoint color, the algorithm measures the center of the supposed waypoint using a weighted average of all matching pixel coordinates. Using the *horizontal_view_angle* value, the algorithm rotates so that this center point is in the center of the visual field.

When the weighted center point of the supposed waypoint is in the center of the visual field, and while the waypoint does not fill the visual field (as specified by *at_waypoint_fraction*), the algorithm performs forward translation towards the waypoint. Once the fraction of matching pixels exceed *at_waypoint_fraction*, the algorithm determines that it has reached the current waypoint. A *flashLight* move is sent, with the number of flashes representing the current waypoint, and then the entire procedure is repeated for the next waypoint.

If looping is enabled, the algorithm starts over with the first waypoint after reaching the last waypoint.