Kinect and A*

with voice control

About the project

This project is the result of Matthew Barulic's work at GTRI ELSYS during the summer of 2012. The project equips Mobile Robot's Seekur Jr robot with a kinect, which it uses to autonomously generate a map of its environment while navigating successfully to a user-defined destination.

The robot uses the Point Cloud Library (PCL) to process the 3D data from the kinect.

http://www.pointclouds.org

Mobile Robot's ARIA library was used for communication with Seekur Jr's microcontroller.

http://www.mobilerobots.com/Software/ARIA.aspx

A custom implementation of the A* path finding algorithm was written based partly on the following explanation:

http://www.policyalmanac.org/games/aStarTutorial.htm

Voice commands and feedback were added using Microsoft's Speech API (SAPI):

http://www.microsoft.com/en-us/download/details.aspx?id=10121

The only additional sensor added to the robot was Microsoft's Kinect sensor.

http://www.microsoft.com/en-us/kinectforwindows/

This project was developed using Visual C++ 2010 Express Edition:

http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express
NOTE: the express edition does not ship with the ATL library required by SAPI, this library is available for free however as part of the Windows Driver Kit (WDK):
http://msdn.microsoft.com/en-us/library/windows/hardware/gg487428.aspx

A video of the robot in action is available here:

https://www.youtube.com/watch?v=CHovZuMBuj4

Running the code

The code for this project was developed in Visual C++ Express on a laptop running Windows 7. Most of the code is cross-platform compatible with both Mac OSX and Linux distributions. To compile the project on a Mac or Linux machine, simply use the source code under KinectAndAStar. KinectAndAStar_WithVoiceControl contains the source code for running on a windows machine.

To run this code on another machine, the only requirements are the Point Cloud Library and Mobile Robot's ARIA library.

When the program starts, either the TTS voice or the command prompt (depending on which version you are running) will prompt you for two relative coordinates for the goal position.

When using the command prompt, the first coordinate is the number of meters forward/ backward, where forward is positive. The second coordinate is left/right where left is positive.

When using voice control, the TTS voice will ask you for the "first dimension of the goal position." Voice control allows you to give coordinates in any order using the following format. ([...] indicates optional pieces of the phrase. # signs indicate single digit numbers.)

[point #] meter[s] (forward / backward / left / right)

So, for example to specify a goal position that is 5.5 meters in front of the robot and 0.6 meters to the left, you could use one of the following methods:

Command prompt:

Please type the goal x(forward) coordinate and press enter: 5.5
Please type the goal y(horizontal) coordinate and press enter: 0.6

Voice control:

"Five point five meters forward."
"Zero point six meters left."
(**note:** You could also say these two phrases in the opposite order.)

Now, with the destination set, the robot will begin to navigate to that position while building a map of the area it sees. The robot's moves are divided into two types: "Changing direction" and "Moving forward." This allows the robot to simplify its path into longer, straight lines.

In order for the robot to consider a point in space as drivable, it must be the center of a 0.6m radius circle of clear space. This value can be changed by editing the third argument in the following line of code:

```
result = ASPathFinder::FindPath(flat_map, stepsize, /*Safety radius*/0.6, robotPose, end, path);
```

Please note that the search grid used for path finding is not perfect. It is, afterall, a grid whose origin lies at the center of the robot. It is possible for the algorithm to not find any drivable points through a narrow corridor where the only drivable locations run straight through the center. In practice however, the robot will usually try to double back and try some other path, which will move the search grid and it may eventually line up.

This program also loads a model of the robot from the SeekurJr_scaled.stl file.