Autonomous Navigation: The Quest for a Real Title
Comp 352

Micah Wylde
Jeffrey Ruberg

May 3, 2011

1 Introduction

1.1 Related work

2 Autonomous Driving

2.1 Simulation

3 Methods

To create an autonomous agent that navigates while simulating a car's behavior and traffic laws, we took three general approaches: deliberative planning through A* search, reactive navigation through a dynamical system, and a hybrid of deliberative planning and reactive motion. As for the agents' environment, we create graphical worlds out of real map data. The code architecture consists of a server and a separate client for each agent.

3.1 Code Architecture

The project is written in Ruby, under JRuby to utilize Java2D for the graphical display. Specifically, all server code runs solely under JRuby, but client code can run under any flavor of Ruby. Agents are represented both on the client and server end; server agents perform motion- and display-related calculations, and client agents contain all the navigation inference and decision-making and ultimately send decisions (restricted to behavior variables) back to the server againt.

A brief description of the various source files will follow.

app.rb This file is the point of entry for the program. This same entry point is used to, based on various command-line options, start a server, start a new agent, or run tests.

client_agent.rb This file contains the `ClientAgent` class, the super class which every client agent inherits. Client agents receive messages from remote server agents, process the message (based on their form of navigation), and then send back a response in their behavior variable space.

constants.rb This file contains various general constants that may be used in several locations or files, or that may be particularly useful to tweak with.

display.rb This file contains all of the display code used to generate our rendering of the world.

map.rb This file contains the classes, specifically `Map`, which encode information provided from real map data.

pqueue.rb A priority queue implementation useful for A* search.

remote_agent.rb This file contains the `RemoteServerAgent` class, which is a subclass of `ServerAgent`. Essentially, a remote server agent is a server agent which is tied to a specific client agent and communicates with that client agent.

server_agnet.rb This file contains the `ServerAgent` class, which contains all the base representation and calculations needed for an agent (for example, the server agent computes various points needed to display the agent graphically).

server.rb This file contains the socket server to which new agents connect.

socket.rb MICAH

util.rb This file contains a collection of geometry classes (`Point`, `Vector`, etc.) which are useful in the display and other various places (most notably in dynamical navigatoin calculations).

agents/astar.rb This file contains a client agent that deliberatively plans paths using A* search.

agents/dynamical.rb This file contains a client agent that navigates through a purely dynamical system-based approach.

agents/hybrid.rb This file contains a client agent that navigates through a combination of deliberative planning and dynamical systems.

agents/simple.rb This file contains a very primitive client agent (that can hardly be called an agent) which allowed us to easily test the motion calculations performed by server agents.

3.2 Simulation Environment

Our environment takes place in a graphical world we have constructed from scratch. We start with real map data specifying nodes (intersections) and connections (roads) between nodes, and create a graph representation; we also convert positions specified in terms of latitude and longitude into a scale of meters (where the minimum latitude position is given an x-coordinate of zero, and likewise for longitude and y-coordinates). We mostly store the road data as this graph of nodes and neighborhood relationships, but we also create road objects which own wall line segments; while the map is being processed, we also clip/extend these walls to create realistic-looking chunks of road between two nodes.

To run the display, we start a server by running the script `bin/driving`. In the display, we have implemented mouse dragging, zooming, pausing, agent following (on by default), and agent manipulation/placement.

3.3 Deliberative Planning

3.4 Reactive Navigation

3.5 Hybrid Approach

4 Results

5 Conclusion

5.1 Unreached goals

- cached map rendering