Plan.js: A Motion Planning Library for Javascript

Clinton Freeman

Abstract—**Plan.js is an open source javascript library that provides motion planning for web applications.**

I. Introduction

IN the past ten years, there has been a large movement in industry toward web browser based applications and away from native desktop programs. This is due in large part because of the inherent cross-platform nature of in-browser applications. As a part of this movement, numerous open source libraries have been developed that provide functionality that has been available in other languages such as C/C++ for quite some time. While several libraries have covered important areas such as 3D graphics (three.js) and collision detection (ammo.js), there is not a widely used motion planning library to the best of my knowledge.

Plan.js is a JavaScript library that addresses this gap in available software. It contains many of the commonly used motion planning algorithms such as RRT, RRT*, PRM, and exact geometric methods for 2D motion planning. The source code is available to others on GitHub, and users may view visualizations of the algorithms live in the browser.

II. Related Work

The Open Motion Planning Library (OMPL) is a standard resource for motion planning algorithms for desktop C++ applications [1]. It contains implementations of a wide variety of algorithms, including RRT*, PRM, and others. Many years of work have gone into the implementations available in this library.

Emscripten [2] is a compiler developed by Mozilla that compiles native C/C++ code into a subset of Javascript known as *asm.js*. asm.js javascript files are able to be compiled by the browser ahead of time into machine code in order to gain a significant boost in performance. Mozilla claims that using Emscripten and asm.js can result in performance that is only twice as slow as native applications. Major game engines including Unity3D and UnrealEngine are providing asm.js ports of their software that can run at interactive speeds.

As mentioned in the introduction, ammo.js is a javascript library for performing collision detection. This library is created by compiling the popular Bullet collision detection library with Emscripten. This is one route that one can take when attempting to port a library over to javascript. However, the performance for ammo.js is somewhat questionable.

III. Problem Definition

Note: *The problem definition section is dependent upon how I decide to go about creating plan.js. For example, if I decide that I am going to hand-write a particular set of motion planning algorithms, then this section will contain the inputs/output of those algorithms and any related terminology. However, I am currently investigating compiling an existing C++ library such as OMPL into Javascript by using Emscripten. In that case, the "input" will consist of the OMPL C++ source code, and my "output" will consist of corresponding JavaScript code. It still is not clear to me at this point whether compiling OMPL is actually feasible by the end of the semester; if it isn't then I will fill this section in with the subset of planning algorithms I am able to implement myself.*

Native code is composed of machine instructions specific to a particular computer architecture (e.g. x86). This is in contrast to *interpreted* code, which is composed of instructions that are translated into actual machine instructions by a virtual machine at run time. Native code has historically run significantly faster than interpreted code, although over time the gap has become much smaller. While native code (generated by languages such as C++) has an edge when it comes to runtime performance, interpreted code (generated by languages such as Java) is generally able to run across a larger number of platforms without modification.

IV. Methods

I have decided that I will either port OMPL in its entirety, or port a smaller subset that does not have trouble with dependencies. OMPL's main dependency on Boost should not be a problem, but it will be problematic to port multithreaded algorithm implementations. So far I have:

- Downloaded and compiled OMPL on Ubuntu.
- Downloaded and compiled examples programs using the Emscripten SDK.
- Read documents related to compiling large projects such as ammo.js.
- Attempted to compile OMPL directly. I'm currently sorting through a huge number of errors related to various incompatibilities.

V. Results

I plan to evaluate the performance of each algorithm empirically across different browsers to identify potential bottlenecks. I am particularly interested in how javascript implementations compare to native C++ implementations. Since I will be generating the library by porting C++ code to a corresponding JavaScript file, I should be able to make a direct performance comparison that would be of interest to the larger web community. I expect the performance to be similar to the performance of ammo.js (a port of the Bullet physics engine).

VI. Conclusion and Future Work

There is not much time left in the semester, but with enough effort I should be able to get a working demo of motion planning in the browser. Here is a rough set of milestones, to be completed "as fast as possible":

1) Get OMPL compiled and an example algorithm running in the browser.
2) Get the OMPL visualization application ported and visualizing an algorithm.
3) Generate test data and run a battery of tests between native and javascript versions.
4) Analyze data, produce graphs.
5) Give recommendations about whether porting OMPL is going to give reasonable performance for web applications.

References

[1] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[2] Wikipedia, "Emscripten — wikipedia, the free encyclopedia," 2014, [Online; accessed 18-April-2014]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Emscriptenoldid=601756691