manifold Documentation

Release 0.0.1

Open Source Robotics Foundation

May 03, 2020

Contents

Contents:

CHAPTER 1

What is Manifold?

Manifold is an open source library that allows to load RNDF road network files.

- What programming language can I use to interface Manifold?

C++ is our native implementation and so far the only way to use the library.

CHAPTER 2

Installation

Instructions to install Manifold on all the platforms supported: major Linux distributions, Mac OS X and Windows.

2.1 Ubuntu Linux

Setup your computer to accept software from *packages.osrfoundation.org*:

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable
`lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

Setup keys:

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

Install Manifold:

```
sudo apt-get update
sudo apt-get install libmanifold0-dev
```

2.2 Mac OS X

Manifold and several of its dependencies can be compiled on OS X with Homebrew using the osrf/simulation tap. Manifold is straightforward to install on Mac OS X 10.9 (Mavericks) or higher. Installation on older versions requires changing the default standard library and rebuilding dependencies due to the use of c++11. For purposes of this documentation, I will assume OS X 10.9 or greater is in use. Here are the instructions:

Install Homebrew, which should also prompt you to install the XCode command-line tools:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
↪install)"
```

Run the following commands:

```
brew tap osrf/simulation
brew install manifold0
```

2.3 Windows

At this moment, compilation has been tested on Windows 7 and 8.1 and is supported when using Visual Studio 2013. Patches for other versions are welcome.

This installation procedure uses pre-compiled binaries in a local workspace. To make things easier, use a MinGW shell for your editing work (such as the Git Bash Shell with Mercurial), and only use the Windows cmd for configuring and building. You might also need to disable the Windows firewall.

Make a directory to work in, e.g.:

```
mkdir manifold-ws
cd manifold-ws
```

Clone and prepare the Ignition Math dependency:

```
hg clone https://bitbucket.org/ignitionrobotics/ign-math -b ign-math2
cd ign-math
mkdir build
```

In a Windows Command Prompt, load your compiler setup, e.g.:

```
"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.bat" amd64
```

In the Windows Command Prompt, configure and build:

```
cd ign-math\build
..\configure
nmake install
```

Clone Manifold:

```
cd ..\..\
hg clone https://bitbucket.org/osrf/manifold
cd manifold
```

Configure and build:

```
mkdir build
cd build
..\configure
nmake
nmake install
```

You should now have an installation of ign-manifold in `manifold-ws/manifold/build/install`.

Now build the examples:

```
cd ..\example
mkdir build
cd build
```

(continues on next page)

(continued from previous page)

```
..\configure
nmake
```

Now try an example. In one Windows terminal run:

```
rndf_info <_your_rndf_file>
```

2.4 Install from sources (Ubuntu Linux)

For compiling the latest version of Manifold you will need an Ubuntu distribution equal to 14.04 (Trusty) or newer.

Make sure you have removed the Ubuntu pre-compiled binaries before installing from source:

```
sudo apt-get remove libmanifold0-dev
```

Setup your computer to accept software from *packages.osrfoundation.org*:

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable
`lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

Setup keys:

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

Install prerequisites. A clean Ubuntu system will need:

```
sudo apt-get update
sudo apt-get install cmake pkg-config python ruby-ronn
```

Clone and prepare the Ignition Math dependency:

```
hg clone https://bitbucket.org/ignitionrobotics/ign-math -b ign-math2 /tmp/ign-math
cd /tmp/ign-math
mkdir build
cd build
```

Configure and build:

```
cmake ..
make -j4
sudo make install
```

Clone the repository into a directory and go into it:

```
hg clone https://bitbucket.org/osrf/manifold /tmp/manifold
cd /tmp/manifold
```

Create a build directory and go there:

```
mkdir build
cd build
```

Configure Manifold (choose either method a or b below):

A. Release mode: This will generate optimized code, but will not have debug symbols. Use this mode if you don't need to use GDB.

```
cmake ../
```

Note: You can use a custom install path to make it easier to switch between source and debian installs:

```
cmake -DCMAKE_INSTALL_PREFIX=/home/$USER/local ../
```

B. Debug mode: This will generate code with debug symbols. Manifold will run slower, but you'll be able to use GDB.

```
cmake -DCMAKE_BUILD_TYPE=Debug ../
```

The output from `cmake ../` may generate a number of errors and warnings about missing packages. You must install the missing packages that have errors and re-run `cmake ../`. Make sure all the build errors are resolved before continuing (they should be there from the earlier step in which you installed prerequisites).

Make note of your install path, which is output from cmake and should look something like:

```
-- Install path: /home/$USER/local
```

Build Manifold:

```
make -j4
```

Install Manifold:

```
sudo make install
```

If you decide to install Manifold in a local directory you'll need to modify your `LD_LIBRARY_PATH`:

```
echo "export LD_LIBRARY_PATH=<install_path>/local/lib:$LD_LIBRARY_PATH" >> ~/.bashrc
```

Now build the examples:

```
cd ../
mkdir build
cd build
cmake ..
make
```

Now try an example. In a terminal run:

```
rndf_info <_your_rndf_file>
```

2.4.1 Uninstalling Source-based Install

If you need to uninstall Manifold or switch back to a debian-based install when you currently have installed the library from source, navigate to your source code directory's build folders and run `make uninstall`:

```
cd /tmp/manifold/build
sudo make uninstall
```

CHAPTER 3

How to contribute

Manifold is an open source project based on the Apache License Version 2.0, and is maintained by hardworking developers for everyone's benefit. If you would like to contribute software patches, read on to find out how.

3.1 Development process

We follow a development process designed to reduce errors, encourage collaboration, and make high quality code. The process may seem rigid and tedious, but every step is worth the effort (especially if you like applications that work).

3.1.1 Steps to follow

1. Are you sure? Has your idea already been done, or maybe someone is already working on it?

   Check the issue tracker.

2. Fork Manifold. This will create your own personal copy of the project. All of your development should take place in your fork.

3. Work out of a branch:

```
hg branch my_new_branch_name
```

Always work out of a new branch, never off of *default*. This is a good habit to get in, and will make your life easier. If you're solving an issue, make the branch name `issue_` followed by the issue number. E.g.: `issue_23`.

4. Write your code.

   This is the fun part.

5. Write tests.

   A pull request will only be accepted if it has tests. See the *Test coverage* section below for more information.

6. Compiler warnings.

   Code must have zero compile warnings. This currently only applies to Linux.

7. Style.

   A tool is provided to check for correct style. Your code must have no errors after running the following command from the root of the source tree:

```
sh tools/code_check.sh
```

The tool does not catch all style errors. See the *Style* section below for more information.

8. Tests pass.

   There must be no failing tests. You can check by running `make test` in your build directory.

9. Documentation.

   Document all your code. Every class, function, member variable must have doxygen comments. All code in source files must have documentation that describes the functionality. This will help reviewers, and future developers.

10. Review your code.

    Before submitting your code through a pull request, take some time to review everything line-by-line. The review process will go much faster if you make sure everything is perfect before other people look at your code. There is a bit of the human-condition involved here. Folks are less likely to spend time reviewing your code if it's bad.

11. Small pull requests.

    A large pull request is hard to review, and will take a long time. It is worth your time to split a large pull request into multiple smaller pull requests. For reference, here are a few examples:

    - Small, very nice
    - Medium, still okay
    - Too large

12. Pull request.

    Submit a pull request when you ready.

13. Review.

    At least two other people have to approve your pull request before it can be merged. Please be responsive to any questions and comments.

14. Done, phew.

    Once you have met all the requirements, you're code will be merged. Thanks for improving Manifold!

3.1.2 Style

In general, we follow Google's style guide. However, we add in some extras.

"this" pointer All class attributes and member functions must be accessed using the `this->` pointer. Here is an example.

Underscore function parameters All function parameters must start with an underscore. Here is an example.

Do not cuddle braces All braces must be on their own line. Here is an example.

Multi-line code blocks If a block of code spans multiple lines and is part of a flow control statement, such as an `if`, then it must be wrapped in braces. Here is an example

++ operator This occurs mostly in `for` loops. Prefix the `++` operator, which is slightly more efficient than postfix in some cases.

PIMPL/Opaque pointer If you are writing a new class, it must use a private data pointer. Here is an example, and you can read more here.

const functions Any class function that does not change a member variable should be marked as `const`. Here is an example.

const parameters All parameters that are not modified by a function should be marked as `const`. This applies to parameters that are passed by reference, pointer, and value. Here is an example.

Pointer and reference variables Place the `*` and `&` next to the variable name, not next to the type. For example: `int &variable` is good, but `int& variable` is not. Here is an example.

Camel case In general, everything should use camel case. Exceptions include protobuf variable names.

Class function names Class functions must start with a capital letter, and capitalize every word.

`void MyFunction();` : Good

`void myFunction();` : Bad

`void my_function();` : Bad

Variable names Variables must start with a lower case letter, and capitalize every word thereafter.

`int myVariable;` : Good

`int myvariable;` : Bad

`int my_variable;` : Bad

3.1.3 Reduce Code Duplication

Check to make sure someone else is not currently working on the same feature, before embarking on a project to add something to Manifold. Check the issue tracker looking for issues with similar ideas.

3.1.4 Write Tests

All code should have a corresponding unit test. Manifold uses GTest for unit testing.

Test coverage

The goal is to achieve 100% line and branch coverage. However, this is not always possible due to complexity issues, analysis tools misreporting coverage, and time constraints. Try to write as complete of a test suite as possible, and use the coverage analysis tools as guide. If you have trouble writing a test please ask for help in your pull request.

Manifold has a build target called `make coverage` that will produce a code coverage report. You'll need lcov installed.

1. In your `build` folder, compile Manifold with

   `-DCMAKE_BUILD_TYPE=Coverage`:

```
cmake -DCMAKE_BUILD_TYPE=Coverage ..\
make
```

2. Run a single test, or all the tests:

```
make test
```

3. Make the coverage report:

```
make coverage
```

4. View the coverage report:

```
firefox coverage/index.html
```

3.2 Debugging Manifold

3.2.1 Meaningful backtraces

In order to provide meaningful backtraces when using a debugger, such as GDB, Manifold should be compiled with debugging support enabled. When using the ubuntu packages, specially the `-dbg` package, this support is limited but could be enough in most of the situations. This are the three level of traces which can be obtained:

Maximum level of debugging support This only can be obtained compiling Manifold from source and setting the `CMAKE_BUILD_TYPE` to `DEBUG`. This will set up no optimizations and debugging symbols. It can be required by developers in situations specially difficult to reproduce.

Medium level of debugging support This can be obtained installing the `libmanifold0-dbg` package or compiling Manifold from source using the `RELWITHDEBINFO` `CMAKE_BUILD_TYPE` mode (which is the default if no mode is provided). This will set up `-O2` optimization level but provide debugging symbols. This should be the default when firing up gdb to explore errors and submit traces.

Minimum level of debugging support This one is present in package versions (no `-dbg` package present) or compiling Manifold from source using the `RELEASE` `CMAKE_BUILD_TYPE` option. This will set up the maximum level of optimizations and does not provide any debugging symbol information. This traces are particularly difficult to follow.

3.3 Code Check

Code pushed into the repository should pass a few simple tests. It is also helpful if patches submitted through bitbucket pass these tests. Passing these tests is defined as generating no error or warning messages for each of the following tests.

3.3.1 Static Code Check

Static code checking analyzes your code for bugs, such as potential memory leaks, and style. The Manifold static code checker uses cppcheck, and a modified cpplint. You'll need to install cppcheck on your system. Ubuntu users can install via:

```
sudo apt-get install cppcheck
```

To check your code, run the following script from the root of the Manifold sources:

```
sh tools/code_check.sh
```

It takes a few minutes to run. Fix all errors and warnings until the output looks like:

```
Total errors found: 0
```

CHAPTER 4

API

Please, visit this link for version 0.x.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search