AN ANALYSIS OF CURRENT ALGORITHMS FOR PATH PLANNING IN THE RETURN AND DELIVERY JOURNEY OF GROUND BASED COURIER ROBOTS

By

Joshua Daly

Supervisor(s): Mr. Arnold Hensman

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
B.SC. (HONOURS) IN COMPUTING
AT
INSTITUTE OF TECHNOLOGY BLANCHARDSTOWN
DUBLIN, IRELAND
March 11, 2015

Declaration

I herby certify that this material, which I now submit for assessment on the programme of study leading to the award of **B.Sc. (Honours) in Computing** in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfillment of the requirements of that stated above.

Dated: March 11, 2015

Author: _____
Joshua Daly

Abstract

Abstract should be clear, concise, and should cover the entire project in a fraction of the space, consider:

- Keep the word count low around 250 words.
- Avoid an jargon or ambiguous language.
- Do not use abbreviations.
- Do not reference anything.
- Briefly cover the motivation, problem statement, approach, results and conclusions.

Acknowledgements

Remember to thank the following people:

- My family and friends for putting up with me during the course of this project.
- Arnold Hensman for providing supervision and leading me into robotics.
- Tucker Balch for producing an excellent tutorial on grid based navigation.
- Sven Koenig for taking the time to respond to my queries.
- The Arduino, Raspberry Pi, and DFRobot communities for being awesome.

Table of Contents

List of Tables

List of Figures

Abbreviations

DARPA	Defence Advanced Research Project Agency
SLAM	Simultaneous Localisation and Mapping

Chapter 1

Core Implementation

1.1 Building the Project

1.1.1 Obtaining the Source

The latest version of the project's source code can be checked out via *git* using:

git clone https://github.com/swordmaster2k/botnav.git

Or downloaded as a ZIP file from `https://github.com/swordmaster2k/botnav`.

Alternatively the most up to date version at the time of printing is available on the CD at the front of this thesis.

1.1.2 Compiling the D* Lite Cython Module

The planning algorithm D* Lite must be compiled as a Cython module, the original source code was provided by Maxim Likhachev of CMU and Sven Koenig of USC in C. It has been modified to make it compatible with the core Python system using Cython, as Python is implemented in C [**?**] it is inherently compatible with the sample of D* Lite that is provided by its authors.

To build D* Lite you will need Python3.4, the Python3.4 headers, gcc, and make. It **must** be built for each platform on which it will execute as C compiles to machine code making it *target dependent*. From a terminal navigate to the source code directory *BotNav/algorithm/dstarlite_build/*.

The make file contains two build rules:

1. *make* - which builds the module *dstarlite_c.so*
2. *make clean* - cleans all previous output files from the build process

Once the module file *dstarlite_c.so* has been successfully built for the target platform it can simply be dropped into the parent directory *BotNav/algorithm/*. The Python source code contains a reference to the module and will automatically link it in at execution time.

1.1.3 Running it in Python3

By default the project is set-up to run a sample simulation with *sample.map* using the GridNav path planning algorithm. It will output the results of its run to the *BotNav/maps/output/* directory and is configured to compute a path from every traversable cell to the goal. To run it simply navigate to the path containing *tester.py* and type:

python3 tester.py config.botnav

The argument passed to *tester.py* is the path to the default configuration file, the contents of which will be discussed later in this chapter. After running the above command in a terminal the result shall look similar to Figure 1.2.

```
Planner: GridNav

Total Planning Steps: 1
Total Vertices: 100

Vertex Accesses: 47
Average: 47.0

Total Planning Time (seconds): 0.001
Average Planning Time (seconds): 0.001
```

Figure 1.1: An example of the type of output generated after running GridNav over *sample.map* in simulation mode.

1.2 Running Simulations

Simulations provide an easy means of testing each planning algorithm in controllable environments, it speeds up the testing process immeasurably, and provides reproduce-able results. The most powerful feature of simulations is the ability to plan a path from every free cell in the environment, the result of each traversal is placed into a separate timestamped folder. This data can then be easily mined and analysed in order to gauge how each algorithm performed for a particular scenario.

1.2.1 Configuration

When carrying out simulated runs three parameters must be set in the corresponding configuration file they are *map*, *mode*, and *planner*. Below is an example of the configuration required to run a simulated trial using D* Lite:

map=maps/simple.map

planner=d_star_lite

mode=simulated

The most important parameter setting here is *mode* which is set to simulated. At run time this informs the *Tester* class that we want to run an experimental simulation across all traversable

cells and that we do not need a communications channel via *Proxy*. In simulation mode all of the instructions are invoked on the "dummy" robot class *SimulatedRobot*, the method *go_to(self, x, y)* simply takes the x and y coordinates, introduces some random drift, and assigns them:

```
def go_to(self, x, y):
    # Introduce a little uncertainty.
    self.x = (x + random.uniform(-0.2, 0.2)) * self.cell_size
    self.y = (y + random.uniform(-0.2, 0.2)) * self.cell_size

    self.trail.append([self.get_cell_x(), self.get_cell_y()])

    self.state = "Travelled"
```

1.2.2 Sample Output

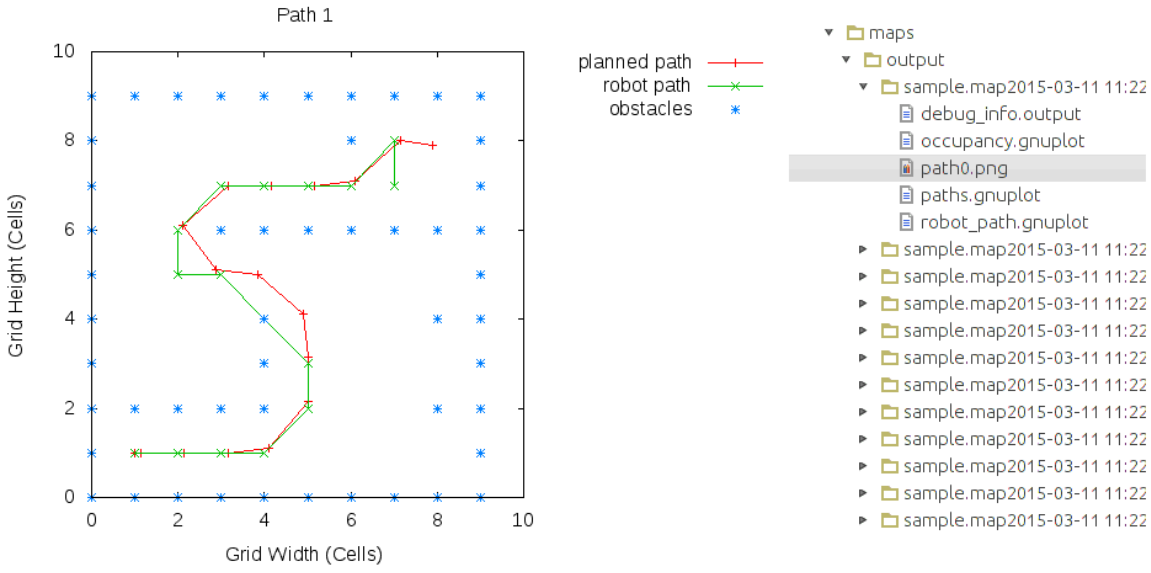

Figure 1.2: A plot file for the first path from the sample map generated by gnuplot (left), and the contents of the output folder after the run. Note that each folder is timestamped (right).

1.3 Using a Real Bot

The real test for any path planning algorithm is its practical effectiveness [6] and the only way to gauge this is using a physical robotic platform i.e."a Real Bot". The simulations that we have performed here are very limited in nature as they do not take into account any variability in the mechanics of the robot. While it is perfectly possible to model variations such as drift, odometry error, friction, and battery drain in a simulation it is far more practical to simply use a physical platform.

There are a number of factors that we must consider when using a physical robot, to begin with we will need some form of communications mechanism be it wired (USB, Ethernet, Serial) or wireless (ZigBee, WiFi, Bluetooth). It is through this medium that the *Proxy* class will process the data to and from the robot. Then there is drift, occurrences such as wheel slippage can lead to the robot veering off course, we will assume that this issue is dealt with at a lower level than the path planning system.

1.3.1 Configuration

The contents of the configuration file for a hardware robot depends on the communications medium being employed. At present the system supports three forms of communication Bluetooth, IP, and Serial. Before any of these can be used the run mode must set to *physical*. Below is an example configuration using Bluetooth which requires the MAC address of the device and a port number:

map=maps/sample.map
planner=grid_nav
mode=physical
connection=bluetooth
mac=00:00:12:06:56:83
port=0x1001

For IP based configurations an *ip*, *mac*, and *port* setting will be required. Serial based connections need only a *baud*, and *port*, examples of both are available in the comments of the default configuration file:

```
# if connection == bluetooth
#     mac=
#     port=
# elif connection == ip
#     ip=
#     mac=
#     port=
# elif connection == serial
#     baud=
#     port=
```

1.4 How the Planner Works

Explain how the planner was implemented in code, the control logic, looping structure, reacting to change, and the conditions for terminating the planner.

1.4.1 Five Simple Steps

Every planner in robotics splits the problem into five simple steps, include them under this section as numbered items. State any recursive operations that take place. Also include a flowchart or diagram in some form.

1.4.2 Abstracting Away from the Algorithm

Art the heart of the object oriented patterns used in this project is *abstraction*, a technique that enables a developer to generalise the functionality of a class through abstract inheritance. Through this process the complexity of an object can be reduced and its efficiency greatly increased [**?**]. It has been used extensively in the implementation of the path planning algorithms which all inherit from the base class *AbstractAlgorithm*. From this class each algorithm inherits a series of default behaviours some of which are listed here:

- plan(self) -
- pop_next_point(self) -
- update_occupancy_grid(self, cells) -
- print_debug(self, stream) -

The real advantage to using abstraction here is the *pluggable* nature it provides. Every algorithm that inherits from the *AbstractAlgorithm* base class implements a common set of behaviours and properties. Regardless of which algorithm is currently being used for planning we simply access the same behaviour or property. The implementation details of each algorithm is hidden from the *Planner* whom only cares about the results.

1.5 Open Field D*

Core of the project very important, state that every implementation of Field D* to date is closed source NASA's code is not available, nor is Carnegie Mellon's. Open Field D* is significant because it bucks this trend making it open to ITB students and others.

1.5.1 Modifying D* Lite

Point out the key differences between D* Lite and Field D* from a coding perspective, nodes to cell corners how this is represented, linear interpolation. Using Georgia Institute of Technologies D* Lite code state the modifications required to get Field D*.

1.5.2 Basic Implementation

Cover basic implementation of Field D*, most importantly state any problems encountered, or variations/optimisations made during the coding stage.

Bibliography

[1] W. Burgard, "Introduction to mobile robotics robot motion planning." Online Lecture Notes.

[2] E. Dijkstra, "A note on two problems in connexion with graphs," tech. rep., Numerische Mathematik, 1959.

[3] A. Hensman, "Data structures and algorithms bsc. in computing semester 5 lecture 8." Graph Applications.

[4] T. Balch, "Grid-based navigation for mobile robots," tech. rep., Georgia Tech, 1995.

[5] D. Ferguson and et al., "A guide to heuristic-based path planning," tech. rep., Carnegie Mellon University, 2005.

[6] D. Ferguson and A. Stentz, "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," tech. rep., Carnegie Mellon University, 2007.

[7] W. Knight, "Driveless cars are further away than you think," tech. rep., MIT, October 2013.

[8] S. Koenig and M. Likhachev, "D* lite," in *AAAI-02 Proceedings*, pp. 476–483, AAAI, 2002.

[9] G. Mies, "Military robots of the present and the future," in *AARMS*, vol. 9, pp. 125–137, AARMS, May 2010.

[10] C. Ray, F. Mondada, and R. Siegwart, "What do people expect from robots?," tech. rep., IEEE, 2008.

[11] D. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles," tech. rep., Eno, October 2013.

[12] C. Frey and M. Osborne, "The future of employment: How susceptible are jobs to computerisation?," tech. rep., University of Oxford, 2013.

[13] U. Nehmzow, *Mobile Robotics A Practical Introduction*, ch. 5, pp. 95–108. Springer, 2nd ed., 2003.

[14] A. Stentz, "A complete navigation system for goal acquisition in unknown environments.," tech. rep., Autonomous Robots 2, 1995.

[15] P. Hart, N. Nilsson, and B. Rafael, "A formal basis for the heuristic determination of mimimum cost paths.," in *IEEE trans. Sys. Sci. and Cyb.*, vol. 4, pp. 100–104, IEEE, 1968.

[16] D. Ferguson and A. Stentz, "Field d*: An interpolation-based path planner and replanner," tech. rep., Carnegie Mellon University, 2007.

[17] J. Letzing, "Amazon adds that robotic touch." Online, March 2012. http://online.wsj.com/news/articles/SB10001424052702304724404577291903244796214.

[18] T. Stentz, "Cmus path planning software lets curiosity find its way." Online, 2014.

[19] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," tech. rep., University of Southern California, 2009.

[20] J. Daly, P. Butterly, and L. Morrish, "Implementing odometry and slam algorithms on a raspberry pi to drive a rover," tech. rep., Institute of Technology Blanchardstown, 2014.