Robotics and Autonomous Systems 62 (2014) 737–751

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Concurrent dynamic programming for grid-based problems and its application for real-time path planning

Stephen Cossell *, José Guivant

School of Mechanical and Manufacturing Engineering, The University of New South Wales, Sydney, Australia

HIGHLIGHTS

- We present a highly concurrent grid-based robot motion planner.
- The planner is implemented to run on modern graphics hardware.
- The algorithm runs in $\mathcal{O}(N)$ time for N cells in the grid.
- Can exhaustively plan on a 50ha campus with 1 m $\times$ 1 m resolution in real-time.

ARTICLE INFO

Article history:
Received 11 October 2011
Received in revised form
21 February 2014
Accepted 3 March 2014
Available online 20 March 2014

Keywords:
Concurrent dynamic programming

ABSTRACT

This paper presents a concurrent approach for solving dynamic programming optimization problems such as the generation of optimal cost-to-go functions for robot motion planning in dense environments. Such optimization techniques are core to many robotics problems, but traditional approaches are inherently impractical due to their computational complexity. This limitation usually results in a configuration space being subsampled, lower configuration space coverage, less frequent planner updates, or the use of sub-optimal graph-based road map methods. The proposed approach provides mathematically identical results to traditional grid-based motion planning solvers in at least an order of magnitude less time by leveraging the concurrent architecture found in modern graphics hardware. Although results given here are presented in a robot navigation context, they are also applicable to other dynamic programming problems.

1. Introduction

A current robust and accurate method of robot motion planning involves representing an environment as a grid-based configuration space. For example, [1] use laser range sensor data to construct a two-dimensional occupancy grid before generating an exhaustive cost-to-go function for planning. An optimal path is then extracted from the cost-to-go function, similar to [2], and is approximated to a nonholonomic solution under the constraints of the Unmanned Ground Vehicle's (UGV's) dynamic model. While planning on grid-based configuration spaces is accurate, generating a cost-to-go function can be computationally expensive [3,4].

The remainder of this paper provides a review of existing robot motion planning techniques and highlights prior approaches that either suffer from or attempt to defer the computational complexity of existing grid-based motion planners. A brief background is given in Section 3 to provide the reader with context of the proposed approach. Each evolution of the algorithm and implementation is then presented in Section 4, with experimental results given to properly assess the performance of each flavor in Section 5. In particular, the differing flavors presented focus on reducing the number of repeated redundant calculations performed in the GPU together with minimizing CPU time managing these subregions, with the CPU management being performed post iteration. The flavors are summarized here as follows for the reader's convenience:

Subregions Grouping cells within an occupancy grid and choosing to evaluate cells within this group together on the graphics hardware to properly balance GPU execution time against CPU management time.

Caching subregion cell values Caching a subregion's cell values at the post iteration check of t_i so they can be used again at the t_{i+1} post iteration check without having to be re-read from the graphics hardware.

* Corresponding author. Tel.: ++61432934381.
E-mail addresses: scos506@gmail.com, macgyver@unsw.edu.au (S. Cossell), j.guivant@unsw.edu.au (J. Guivant).

http://dx.doi.org/10.1016/j.robot.2014.03.002
0921-8890/

Delayed post iteration check Performing the post iteration check every 2nd and 3rd iteration.

Statistically scheduled post iteration check Performing the post iteration check on a schedule that targets the statistically appropriate iterations to perform the check on.

A discussion is then raised in Section 5.6 on the generalized performance of the concurrent algorithm relative to a traditional sequential algorithm as a function of the characteristics of modern and possible future processor hardware. Future work is then discussed before conclusions are given.

2. Literature review

Grid-based motion planners have been successfully used in academic and industrial robotics applications for a number of decades [5–7]. In contrast to topological approaches, metric-based approaches provide precise measurements of the spatial layout of an environment to the resolution of the configuration space representation and sensor measurements. This property, therefore allows a more optimal solution to be calculated than approximated graph-based approaches such as open space road mapping methods. The main practical limitation concerns the computational complexity of calculating a solution [4]. Often a compromise such as reduced coverage area or lower resolution is applied to counter this, at the cost of accuracy [8,9]. The required update rate when new sensor data is made available is also a significant factor [10]. A common technique is to use a hybrid metric and topological approach to represent an environment [11]. Here small areas store a metric-based representation with each area arranged globally via a graph-based representation. This is employed as less detail is often required to plan on a global scale, while detail is paramount when planning through dense areas within an environment. This technique is akin to modern approaches for managing map deformity and assisting in loop closure [12]. Although *divide and conquer* motion planning approaches allow more efficient generation of paths they are still limited by the computational power available to a system.

Over the course of the last half century, one of the main methods of countering problems that were too computationally expensive at a point in time was to wait for the next generation of hardware to be released by chip manufacturers [13]. In recent years, however, central processing units (CPUs) have began to reach their physical limit in terms of processing power [14]. The industry has begun moving towards a multi-core architecture that side steps the physical limitations of a single processor. However, software engineers must face increasingly difficult challenges in developing applications for the concurrent paradigm [14,15].

In addition to multi-core CPUs, Graphics Processing Units (GPUs) currently have hundreds or thousands of processors designed specifically for Single Instruction, Multiple Data (SIMD) type calculations [16,17]. The continued increase in GPU performance is mainly driven by the computer game and interactive entertainment industries, but an increasing number of academic and industrial applications have seen GPUs used for non-graphical applications [18,19]. This technique is known as general purpose computation on graphical processing units (GPGPU) and has seen applications and advancements in fields such as database query execution [20] to DNA sequencing [21].

A number of GPU specific algorithmic techniques have arisen, particularly from [18]. An important technique that the proposed method takes advantage of is the *Ping-Ponging Method*, which involves storing the gradual evaluated values of an occupancy grid in two equally sized buffers of memory. Each iteration one buffer is set as read-only, with the other buffer set as a write buffer. Numerous kernels[1] are then executed in parallel, with each referring to the read-only buffer for preexisting cell values and outputting the calculated result for a cell at that iteration to the write buffer. Between each iteration the roles are reversed, and hence the data bounces back and forth between buffers. This double buffering method prevents race conditions occurring and the GPU automatically synchronizes each kernel between algorithm iterations.

Traditional motion panning algorithms based on Dijkstra's algorithm are sequential in nature and they abide by the principle of optimality [22,23]. Likewise, in a grid-based domain, each individual cell is evaluated sequentially, as lower cost neighboring cells are recursively calculated back to an initial zero-cost destination cell. A technique previously presented in [24,25] shows that cells can be evaluated concurrently in theory, producing a quantitatively identical result to traditional sequential algorithms.

3. Background

The theoretical basis for the research presented in this paper is summarized in this section and is based on the work presented in [24]. At the core of each evolutionary step of the algorithm, a *kernel* represented by Algorithm 1, is run on each cell in an occupancy grid in the GPU. The kernel queries the global cost values of the eight neighboring cells and then calculates a value for the assigned cell based on Eq. (3.0.1). The kernel is run on multiple cells in a given occupancy grid concurrently using the aforementioned Ping-Ponging method.

Algorithm 1 Kernel pseudocode run on each cell in an occupancy grid.

```
best cost := undefined
N := set of 8-way neighbors of current cell
for n ∈ N do
  if n_cost is defined then
    current cost := n_cost + travel_cost(n, current cell)
    if best cost is undefined or current cost < best cost then
      best cost := current cost
    end if
  end if
end for
current cell_cost := best cost
```

$$
\begin{aligned}
&C^*(x; x_{\text{goal}}) = \min_{u \in U(x)} \left(C^*(x'; x_{\text{goal}}) + L(x, u)\right)\\
&x' = f(x, u)\\
&x, u \in C_{\text{free}}\\
&u \in U(x)
\end{aligned}
\tag{3.0.1}
$$

$$
L(x, u) = \begin{cases} \text{avg}(C^*(x), C^*(u)) \times \sqrt{2} & \text{if diagonal} \\ \text{avg}(C^*(x), C^*(u)) & \text{otherwise} \end{cases}
$$

The initial *vanilla* implementation requested every cell in an occupancy grid to be evaluated by the kernel at every iteration. This approach involved a simple implementation, but lead to the majority of cells being evaluated redundantly each iteration. For example, cells that are a great distance from the initial zero cost destination cell were continually evaluated as undefined for the initial iterations of the algorithm, as the virtual wavefront of cells being usefully evaluated had not reached the same area of the occupancy grid yet. In addition, cells that are close to the initial zero cost cell were redundantly reevaluated as the same value repeatedly in the later stages of the algorithm's execution.

An initial counter measure to performing these redundant calculations was also presented in [24] and is referred to as the method of the *Expanding Texture*. The main aim of this method was to reduce the number of redundant cell calculations for cells that

[1] A kernel is the name given to the small program that runs on many cores in the GPU concurrently. It is identical code applied to a range of data.