Real-Time localisation based on range template matching accelerated by an GPU

Boris Smidt
University of Antwerp
Antwerp, Belgium
Email: boris.smidt@student.uantwerpen.be

Rafael Berkvens
University of Antwerp
Antwerp, Belgium
Email: rafael.berkvens@uantwerpen.be

Maarten Weyn
University of Antwerp
Antwerp, Belgium
Email: maarten.weyn@uantwerpen.be

***Abstract*—A ship requires a location and an orientation to navigate trough a channel or river. We propose the usage of the radar system and a map to get a correct location. Because this is a large problem we will focus on the localization part of the problem and use a laser range finder instead of radar. We compared an iterative closest point(ICP) algorithm and a grid filter to solve the localization. We used an Graphics Processing Unit to accelerate the grid filter algorithm and achieved a 1000 fold in performance compared to a naive CPU algorithm. This allowed us to track the robot. We chained up two grid filters to improve the sampling efficiency of the grid filter this is called a multi resolution grid filter. This enabled us to check the location at the map resolution. The result is a real time algorithm achieve a 99.9 % accuracy within 0.5m and a rotational accuracy of 90% at 1.5 degree. The ICP algorithm however is unable to do a global localization nor could it improve a generated GPS coordinate. We concluded that the grid filter is a viable solution to solve the localization problem when calculated using a GPU.**

I. Introduction

A ship requires a location and an orientation to navigate trough a channel or river. However a GPS returns a broad location without orientation and thus is not a viable solution. We propose the usage of a radar system for localization. The radar scan will be compared to the map of the channel to extract the location. This is a large and complex problem and has to be split up into multiple problems. These include radar sensor models, sensor fusing and localization. This paper limits itself to the localization process. We will use a laser range finder instead of a radar. Because the laser range finder provides less data and is therefor easier to use. To solve the localization problem we will use matching algorithms.

The first algorithm is the Iterative Closest Point (ICP) algorithm. This algorithm iteratively matches two pointclouds: the range scan and the map. A point cloud is a set of points, this research uses 2D data thus our pointclouds are a set of two 2D points.

The second algorithm is a template matching algorithm, the grid filter, this is a correlation based algorithm. The grid filter tries to match the range data to the map in certain possible positions and orientations and returns the weight of these tests.

II. Acceleration Frameworks

The purpose of this paper is to create real-time algorithms. This means that the calculation time of the algorithm may not exceed the update time of a scan. The LMS100 laser has a scan frequency between 25 and 50Hz, this equals 40 to 20ms of calculation time. The following frameworks are used to improve the algorithm performance.

OpenMP [1] is a compiler based library and works by using compiler hints to parallelize the code. The library supports multiple parallel programming models. But the main focus is the fork-join model. The programmer has to manually manage the number of threads used.

OpenCL [1] is a programming language and framework to support multiple kinds of computing devices including CPU, GPU, FPGA and DSP. The parallel programming model used in OpenCL is the data parallel model e.g. vector operations. OpenCL uses kernels to parallelize the code. This is a small C99 syntax function which gets executed on the selected computing device. The kernel is compiled at runtime which enables system specific optimization. These kernels are mostly used to replace the loop body. The main drawback of OpenCL is code portability. The code has to be written for a certain computing device e.g a CPU kernel will be different from a GPU kernel. Another problem of the OpenCL C/C++ SDK is verbosity, a 100 lines of code [2] has to be written to call a simple kernel. OpenCL 1.x only has C as a kernel language thus every kernel had to be rewritten for different data types. This is solved in version 2.1 by offering a new kernel language named OpenCL C++ which allows for template meta programming. To solve these issues there are high level wrappers OpenCL C [3]. An alternative to OpenCL is CUDA but this is limited to Nvidia hardware. This research uses OpenCL as a backend because this is the industry standard.

VexCL [4] is a vector expression template library for OpenCL/CUDA written in C++. It lets the developer write simple vector arithmetic functions in C++ which drastically reduces the boilerplate code. The library generates the kernel code for these functions at runtime. This has two advantages: it allows the program to generate optimized code for the CPU and GPU and allows for type independent code. VexCL offers some useful features like caching kernels and load managing between the computing devices. The library still offers low level control and can call custom OpenCL kernels. There are other libraries with the same goal as VexCL but these use different strategies. For example ViennaCL wraps CUDA, OMP and OpenCL BLAS libraries into a single unified library.