Design Documentation

for

RISC-V Architecture for Motion Planning Algorithms in Autonomous Drone Applications

github.com/AnthonyKenny98/Thesis

Anthony JW Kenny

Electrical Engineering
Advisor: Vijay Janapa Reddi

November 3rd, 2019
Version: 3.1

Abstract

This thesis aims to design RISC-V computer architecture that supports the fast execution of motion planning algorithms for drone applications. First, the computation of sampling-based motion planning algorithms commonly used in autonomous drones (such as Rapidly-exploring Random Tree (RRT), Rapidly-exploring Random Tree Star (RRT*), Probabalistic Road Map (PRM)) will be profiled on an unmodified RISC-V processor. From this profiling, common bottlenecks and hotspots in execution will be identified. Based on these results, this project will extend the RISC-V Instruction Set Architecture (ISA) and design a modified processor to support the extensions.

1 Project Summary

1.1 Problem Statement

Current processors cannot compute motion planning algorithms quickly enough for robots to operate in high complexity environments. Autonomous drones are a specific case of robots requiring real-time motion planning in complex environments. The state-of-the-art strategy of using a Graphics Processing Unit (GPU) to accelerate the execution of these algorithms requires too much power to be cost-effective or feasible for drones to sustain flight for useful periods of time.

1.2 End User

The end user of this project is a developer of autonomous drones. Such developers have a need for computing hardware that executes motion planning algorithms faster

and more power efficiently than existing methods. This thesis will provide a processor design that is synthesizable on an Field Programmable Gate Array (FPGA), giving developers a processer for which a Real-Time Operating Systems (RTOS), or bare metal code, can be written. Additionally, these developers have a requirement that using a new processor for a drone will not require a massive investment in re-development. As such, this thesis will provide the toolchain necessary to compile C code into executable instructions on the new processor.

1.3 Project Goals

This thesis aims to design a RISC-V processor, optimized for motion planning computation, that is synthesizeable on an FPGA and adheres to the requirements outlined in Section 1.4. It will also provide the tools necessary to complile programs for the processor.

The nature of research into accelerating computation through modified computer architecture is such that, when asked a question of how fast/efficient/small/etc a system must be, the answer is, with consideration to certain trade-offs, as fast/efficient/small/etc as it can be! Thus, when defining goals for certain metrics such as speed or power efficiency, this thesis will do so by comparing performance of the modified processor with benchmark performance of an unmodified, off-the-shelf RISC-V processor synthesized on the same FPGA.

1.4 Project Requirements

Table 1 outlines conceptually the requirements of the project.

Requirement	**Description**
RISC-V Compliance	This project will extend the RISC-V ISA to add new instructions. The contstraint of RISC-V compliance means that the new ISA must follow RISC-V conventions, and that the processor can implement any program compiled into the original RISC-V ISA.
Synthesizable	The processor design must be such that it is practically useable by drone developers. Since having such a processor design produced on a chip is beyond this project's scope, this project must deliver a processor defined in an Hardware Description Language (HDL) that is synthesizable on an FPGA.
Speed	One of the motivating factors of this project is the need for motion planning algorithms to execute faster for autonomous drones to become more useful in real world applications.
Power consumption	The second motivating factor is the need for computation aboard drones to be as power efficient as possible to enable them to remain in flight for long enough periods of time.

Table 1: Conceptual Outline of Project Requirements

2 Overall System Specifications

Let the overall system be defined as 2 part: the Processor Module and the Compiler/Assembler Module.
The following subsections detail the technical specifications of the overall system. System specifications based on comparison with benchmarks will be compared against the same programs running on an unmodified, off-the-shelf RISC-V processor synthesized on the same FPGA.

2.1 Speed

2.1.1 Quantitative Description

This project must deliver a system that, given a program that implements, in C, an algorithm often used in autonomous drone motion planning, executes said algorithm an order of magnitude faster than a generic RISC-V processor.

2.1.2 Justification

This project aims to achieve a speedup of at least one order of magnitude (10 times) when compared to benchmark performance, in the execution of pure motion planning algorithm programs. The justification for an order of magnitude speedup comes from similar projects that accelerated motion planning algorithms, acheiving speedups from between 1 and 3 orders of magnitude. These indclude approaches using external hardware accelerators[1], reprogrammable hardware and parallelization on FPGAs[2][3][4], and chip redesigns[5][6].

2.1.3 Measurement

There will be two broad stages of measurement for this metric. First, in simulation, the Vivado Design Suite[7] will allow the execution of a given compiled program to be timed on a simulated processor that is defined in an HDL. Secondly, in synthesis, a to-be-determined tool will allow for me to time the execution of the same program, now on a processor physically synthesized on an FPGA.

2.2 Power Consumption

2.2.1 Quantitative Description

This project must deliver a system that has comparable power consumption as an unmodified RISC-V processor operating on the same FPGA. Comparable will be defined as within a tolerance range of 10%.

2.2.2 Justification

Power is defined as energy dissipated over time.[8] As such, when considering the application of this system in autonomous drones, we want to minimize the amount electrical energy committed to the computation of paths. Since the primary goal of this

thesis is to reduce the execution time, it can aim to keep power use comparable between the benchmark system and the new system. If power remains roughly constant, but the time taken to execute a program is reduced 10 times, we should see a proportional improvement in energy efficiency.

2.2.3 Measurement

The Vivado Design Suite[7] will allow for simulated power consumption estimates, but the important measurement will be comparing the new processor to the unmodified processor on the FPGA, running the same program. I am still to determine how exactly to measure this, but the parameters for testing are known and shown above.

3 Recompiler & Assembler Specifications

The first subcomponent of the system is the Compiler & Assembler Module. It has two specifications, Correctness and Optimality.

3.1 Correctness

3.1.1 Quantitative Description

This project must deliver a Compiler & Assembler Module that, given a program defined in C, can compile and assemble this program into machine code, in a manner that follows the chosen ISA correctly.

3.1.2 Justification

When announcing the IBM System/360 in 1964, IBM said the following: "Instruction Set Architecture is the structure of a computer that a machine language programmer must understand to write a correct program for that machine."[9] That is, it is a contract between the compiler and the hardware, so that the compiler can write machine code that will work for a given computer processor. In this way, for a given ISA (whether RISC-V, or the extened RISC-V this project will design), the Compiler & Assembler Module must adhere to that ISA to compile instruction sets that will execute correctly on the processor.

3.1.3 Measurement

Testing for correct compiling under the original RISC-V ISA is relatively simple. Does the Generic Compiler (which shouldn't be altered by this project) produce correct RISC-V Assembly. Then, the Recompiler module will operate on those RISC-V assembly instructions to produce a recompiled assembly instruction set that follows the project's extended RISC-V ISA. The Assembler will then assemble this into machine code that can be directly loaded into the processor's Instruction Memory. Testing for this required extensive and complete unit tests to be written during this project.

3.2 Optimality

3.2.1 Quantitative Description

This project must deliver a Compiler & Assembler Module that, given certain extended instructions that this project defines, recompiles all regular RISC-V assembly code that is suitable for recompilation.

3.2.2 Justification

The RISC-V ISA is an ISA that supports user-level ISA extensions and specialized variants[10], which may allow the number of instructions per program to be reduced through clever redesign of a processor and new instructions. However, to reap the full performance rewards of these extensions, a compiler must use the new instructions whenever it is able.

3.2.3 Measurement

This will be challenging to measure. I plan to write extensive unit tests that will determine manually the optimal recomplication from RISC-V to extended RISC-V assembly, and then compare that to the output of the recompiler module. There will also be tests to check that these substitutions are occuring in larger, more complicated programs.

4 Processor Specifications

The second subcomponent of the system is the Processor Unit. It has two specifications, RISC-V Compliance and Syntheizability.

4.1 RISC-V Compliance

4.1.1 Quantitative Description

The project must deliver a processor that is RISC-V Compliant, meaning that it can support any correctly compiled RISC-V Assembly Code.

4.1.2 Justification

A processor must be such that it supports the execution of all instructions defined in the ISA for which it was designed. So too must this project's finished processor be able to correctly support any correctly compiled RISC-V assembly code. This may range from simple programs compiled into either the original or extended RISC-V ISA, to complete operating systems, whether for drone applications or a generic linux distribution, for example.

4.1.3 Measurement

The RISC-V organisation has provided a Github Repo for testing a processor for RISC-V compliance.[11] Once it passes this, this processor should be able to run any program or OS compiled into RISC-V assembly.

4.2 Synthesizable

4.2.1 Quantitative Description

The project must deliver a processor defined in an HDL that is synthesizable on an FPGA for the project to be useful for drone developers. While this project will use and test with the Diligent Zync-7000 System on Chip (SoC)[12], the design should be synthesizable on most Zync boards.

4.2.2 Justification

Many papers that have worked in the area of accelerating motion planning algorithms for robot applications have delivered a finished product of a processor/accelerator design implemented in an HDL for synthesis on an FPGA.[2][3][4] This FPGA can then be used to control the robot or share computational load with a co-processor.

4.2.3 Measurement

Measurement for this specification is relatively simple. Once the processor is designed in an HDL, it can be synthesized onto an FPGA using the Vivado Design Suite, given the design is synthesizable (although this is not always simple to achieve). If it is not synthesizable, the design suite will throw and error. Finally, while the design should be correct and RISC-V compliant by the tests performed in section 4.1, to be safe, these compliance tests along with any other unit tests designed during this thesis will then be run on the FPGA processor.

5 Analysis

The design process has, quite necessarily, begun with significant anaylsis of the RRT algorithm. This section will first describe RRT, and then explain my analysis of the algorithm.

5.1 RRT

RRT is an algorithm designed to efficiently search, and thus plan a path through, a high-complexity environment by randomly sampling points and building a tree. The algorithm randomly samples points, draws an edge from the nearest currently existing node in the tree, to grow the tree in the space. It is inherently biased to grow towards large unsearched areas of the problem. RRT was developed by S. LaVelle [13] and J. Kuffner [14]. It is used in autonomous robotic motion planning problems such as

autonomous drones, the focus of this thesis.

The RRT Algorithm with Collision Detection can be seen in Algorthim 1.

Algorithm 1: Rapidly-exploring Random Tree with Collision Detection

```
Inputs: Space S with obstacles
Output: Collision free graph G with K nodes & edges
G.init();
for k = 1 to K do
    while !pointCollision(node_new) do
        q_rand ← getRandomNode();
        q_near ← findNearestNode();
        q_new ← stepFromTo();
    end
    e_new ← newEdge(q_near, q_new)
    if !edgeCollision(e_new) then
        G.addNode(q_new);
        G.addEdge(e_new);
    else
        k = k − 1;
    end
end
```

5.2 RRT Profiling

To restate, the aim of this thesis is to design a computer processor with reduced execution time of motion planning algorithms, such as RRT. As such, it is important to understand the elements of the algorithm that have the highest percentage of CPU execution time. To determine this, it was necessary to implement my own, naive but typical, RRT in C. This program could then be compiled and analysed using a software performance profiling tool. With this, I could design experiments to determine the critical RRT functions (those occupying a majority of CPU time) and see how this varies given different paramaters.

5.2.1 Implementation of Algorithm

The first step was to source a simple, naive implementation of RRT in C that could be analysed. The two options were to either find an existing, public implementation or to develop my own. Many implementations I found online[15][16][17][18] were unsuitable for my purposes, as they had extraneous Graphical User Interface (GUI)s, reliance on external Application Programming Interface (API)s, and other features that distorted analysis of algorithmic hotspots. This made them largely useless for my performance analysis. I needed a clean, minimal implementation of RRT so that I could easily identify the percentage of CPU time each function took up.

As such, I implemented my own version. It can be found here. It follows the Algorithm defined above closely. I took care to be able to parametrize factors such as

Number of Nodes, Number of Obstacles, Obstacle Size, State Space Size, and Epsilon (acceptable distance between two nodes). For monitoring correctness, I build in an optional GUI that shows the tree, starting node, and obstacles. Figure 1 demonstrates the GUI and the RRT implementation.

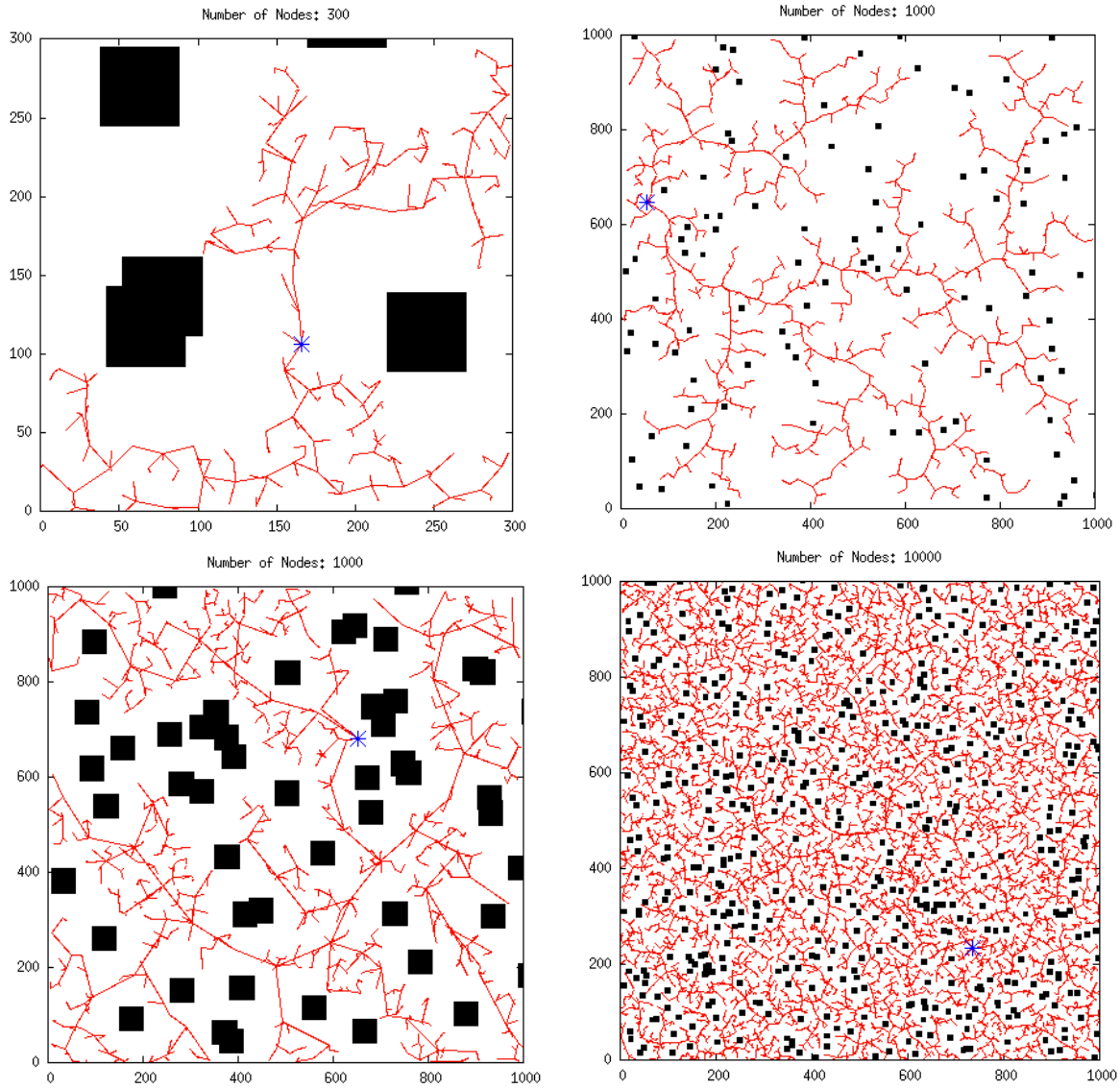


Figure 1: RRT Implementation shown by GUI

5.2.2 Top Down Analysis

The idea of top-down analysis is a practical method to quickly identify true bottlenecks in out-of-order processors, explained well in the paper by A. Yasin[19]. It was influenced by a basic approach that examines the functions that take the highest percentage of CPU time, and drills down to examine their sub-routines. This functionality is provided by software by Intel, called VTune Amplifier.

VTune Amplifier performance profiler is an application for software performance analysis. It provides functionality to examine hotspots for CPU execution time through a top down analysis, shown below in Figure 2. As can be seen from the figure, the top down analysis tool shows the percentage of CPU time taken up by each function. I used this tool to profile the algorithm's performance as I changed certain parameters.

Source Function Stack	CPU Time: Total ▼	CPU Time: Self	Function (Full)	Source File
Total	100.0%	0s		
_start	100.0%	0s	_start	
__libc_start_main	100.0%	0s	__libc_start_...	libc-start.c
main	100.0%	0s	main	rrt.c
rrt	100.0%	0s	rrt	rrt.c
point_collision	51.5%	0.184s	point_collision	rrt.c
findNearestNo	41.4%	1.879s	findNearestN...	rrt.c
edgeCollisions	7.0%	0.067s	edgeCollisions	rrt.c

Figure 2: Top Down Analysis Functionality provided by VTune Amplifier

5.2.3 Experimental Design

As stated above, I parametrized my implementation to be able to vary state space, number of nodes, number of obstacles, obstacle size, and Epsilon. When considering the application of autonomous drones, the two most important factors are number of nodes and number of obstacles. Number of nodes is important for any implementation of RRT, as this is a determining factor of execution speed. A higher number of nodes will also yield shorter paths/higher probability of finding a path to a goal. Number of obstacles is important for autonomous drones, as the obstacles they face in their operating environments (complex natural environments) are often irregularly shaped. As such, these obstacles must often be discretized into a number of smaller obstacles. This is unlike applications such as robotic arms, where the size of (often regularly shaped) obstacles is more important. Given this, I ran a series of tests, varying number of nodes and number of obstacles, to determine the relative use of CPU time for each sub-routine in my RRT implementation.

5.2.4 Results

The left side of Figure 3 shows % of CPU use for a given number of nodes, while the right side shows the same but for a given number of obstacles. the results below show that the three main sub-functions within the RRT implementation are `edgeCollision()`, `pointCollision()`, and `findNearestNode()`. The importance of this is discussed in the design section.

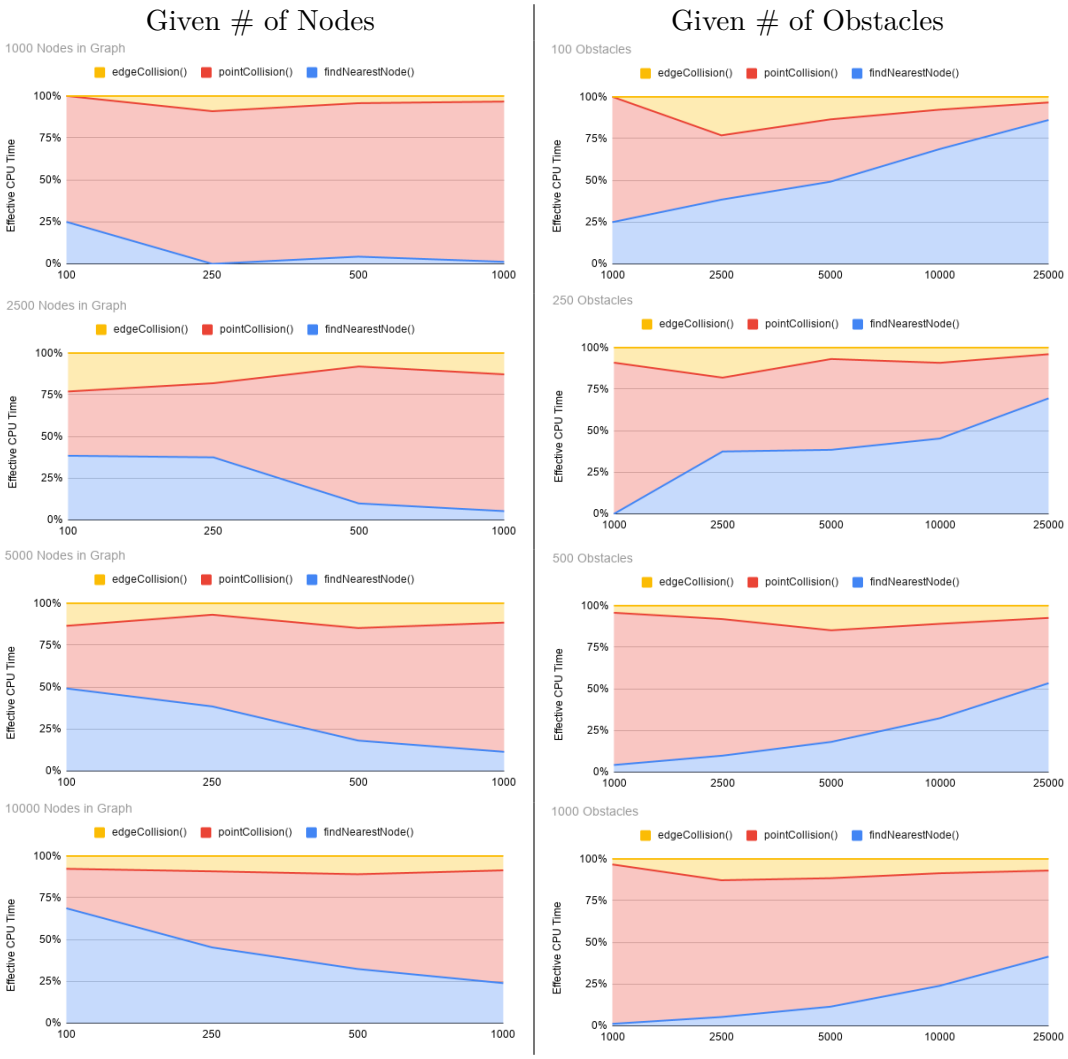
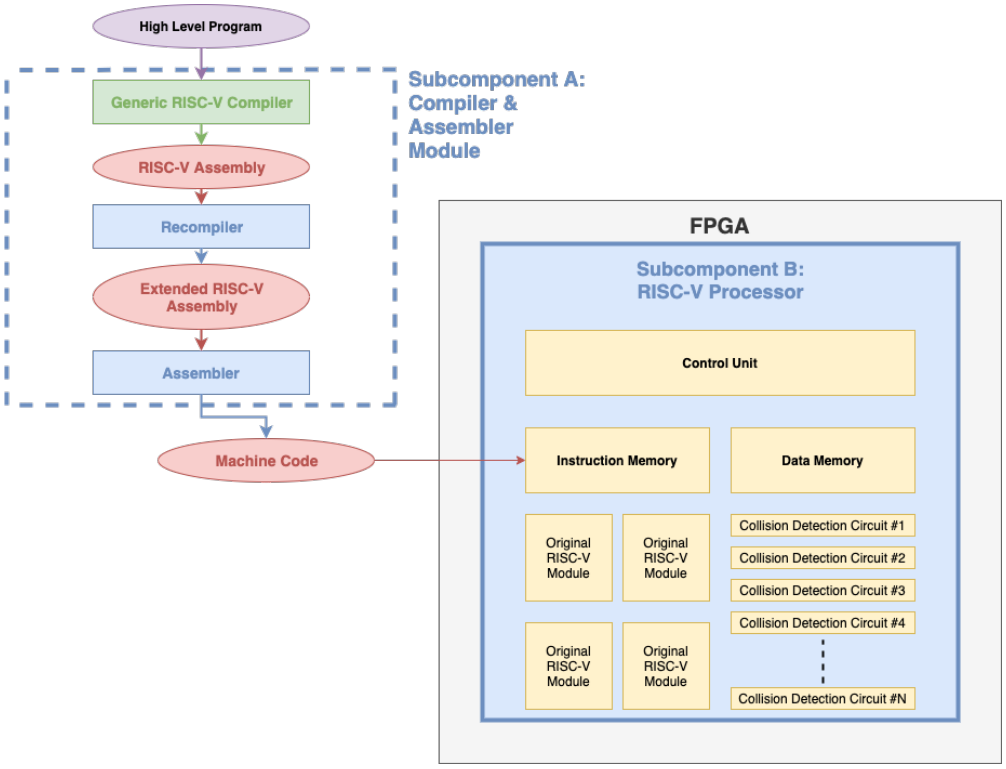


Figure 3: % of CPU Time per Function

6 Design

6.1 System Diagram

The overall system can be defined as two subcomponents: a Compiler & Assembler Module, and a modified RISC-V Processor.



6.2 Subcomponent B Design

As can be seen in the RRT profiling results shown in Figure 3, `pointCollision()` occupies the majority of CPU execution time. Looking at the graphs on the left, where for each graph the number of nodes remain constant, it can be seen that for a given number K of nodes in a graph, the percentage of CPU time occupied by `pointCollision()` increases. When considering the application of RRT to autonomous drones, it is important to note that improving performance as obstacles increase (and the space becomes more complex) is crucial. As such, it makes the most sense to target the execution of `pointCollision()` on the processor.

S. Murray et al., in their paper Robot Motion Planning on a Chipdescribes a method for reducing the execution time of collision detection within the context of the PRM algorithm.[5] They introduced the concept of a Collision Detection Circuit (CDC). They implement a series of N collision detection circuits on the processor, where N is

the maximum number of nodes in the graph, to simultaneously compute which nodes collide with obstacles in the state space. While their approach centered on discretizing the state space into a series of "depth pixels", I believe that the necessary calculations to perform the `pointCollision()` function can be converted into boolean logic and built into a set of N CDCs in my modified RISC-V processor.

I will be implementing the entire processor in a HDL, most likely Verilog, and then using Xilinx to simulate the processor and synthesize onto an FPGA. Due to setbacks in the approach I will be taking to actually synthesize the processor, the board has only just been ordered, and will arrive soon for me to start this process.

6.3 Subcomponent A Design

I've placed the Subcomponent A Design section beneath Subcomponent B in this document because it logically follows that the design of Subcomponent A will rely very much on the ultimate design of Subcomponent B. The design of the extended instructions for the RISC-V ISA will depend on how the CDCs interact with the processor, and how best to feed these calculations from the control module to the CDCs in the processor. As such, much of this is currently unknown and will depend on how the project develops. I do know that I will rely on the RISC-V GNU toolchain (available on GitHub) to compile C code. From there, I will need to convert assembled code from basic RISC-V to my extended RISC-V ISA.

7 Logistics

7.1 Project Timeline

Project Milestone	**Completion Date**
Checkpoint 2: Design Documentation	3 November 2019
Synthesize RISC-V Processor on FPGA	17 November 2019
Checkpoint 3: Mid Year Presentation	3 December 2019
Synthesize with successful CDCs	15 December 2020
Implement RISC-V Extensions and Compiler Module	29 December 2020
Complete First Round Analysis	12 January 2020
Checkpoint 4: Poster Session and Peer Reviews	9 February 2020
Second Design Iteration and Analysis	23 January 2020
Checkpoint 5: Report Draft	28 February 2020
More Analysis	8 March 2020
Finalize Presentation and Report	22 March 2020
Final Oral Presentation	24 March 2020
Final Written Report	3 April 2020
Final Poster	6 May 2020

7.2 Budget

Item	Link	Cost
Zynq Board	https://www.xilinx.com/products/boards-and-kits/1-elhabt.html	$495

8 List of Acronyms

API Application Programming Interface

CDC Collision Detection Circuit

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

GUI Graphical User Interface

HDL Hardware Description Language

ISA Instruction Set Architecture

PRM Probabalistic Road Map

RRT Rapidly-exploring Random Tree

RRT* Rapidly-exploring Random Tree Star

RTOS Real-Time Operating Systems

SoC System on Chip

References

[1] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, *The Microarchitecture of a Real-Time Robot Motion Planning Accelerator*.

[2] S. Murray, W. Floyd-Jones, G. Konidaris, and D. J. Sorin, "A Programmable Architecture for Robot Motion Planning Acceleration," tech. rep.

[3] N. Atay and B. Bayazit, "A motion planning processor on reconfigurable hardware," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2006, pp. 125–132, 2006.

[4] G. S. Malik, K. Gupta, K. M. Krishna, and S. R. Chowdhury, "FPGA based combinatorial architecture for parallelizing RRT," in *2015 European Conference on Mobile Robots, ECMR 2015 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., nov 2015.

[5] S. Murray, W. Floyd-Jones, Y. Qi, D. Sorin, G. Konidaris, and D. Robotics, "Robot Motion Planning on a Chip," tech. rep.

[6] P. Zhi, X. Li, Z. Zhang, S. Karaman, and V. Sze, "High-throughput Computation of Shannon Mutual Information on Chip," tech. rep.

[7] "Vivado Design Suite," 2018.

[8] "American electricians' handbook," *Journal of the Franklin Institute*, vol. 256, p. 99, jul 1953.

[9] IBM, "IBM 360," 1964.

[10] V. I. B. U.-l. Isa, A. Waterman, Y. Lee, D. Patterson, K. Asanovi, and B. U.-l. Isa, "The RISC-V Instruction Set Manual v2.1," *2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings*, vol. I, pp. 1–32, 2012.

[11] J. Bennett and L. Moore, "RISC-V Compliance Github Repository."

[12] "Diligent Zync-7000 SoC."

[13] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," *In*, vol. 129, pp. 98–11, 1998.

[14] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, pp. 378–400, may 2001.

[15] RoboJackets, "RRT," 2019. https://github.com/RoboJackets/rrt.

[16] M. Planning, "rrt-algorithms," 2019. https://github.com/motion-planning/rrt-algorithms.

[17] Sourishg, "rrt-simulator," 2017. https://github.com/sourishg/rrt-simulator.

[18] Vss2sn, "Path Planning," 2019. https://github.com/vss2sn/path{_}planning.

[19] A. Yasin, "A Top-Down method for performance analysis and counters architecture," tech. rep., 2014.