FPGA-AMP: FPGA Accelerated Motion Planning

Matt Ngaw, Yufei Shi, Chris Stange
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Hardware acceleration of key robotic computations is essential to make robots viable for tasks that require fast response times and reactivity to the environment. This report describes the motivation, design, and testing plans of a system capable of improving the performance and power efficiency of robotics motion planning through Field-Programmable Gate Array acceleration.**

***Index Terms*—Field-Programmable Gate Array, Hardware Acceleration, High-level Synthesis, Motion Planning, Rapidly-exploring Random Trees, Robotics**

1 INTRODUCTION

Recent advancements in computing technologies have facilitated the emergence of increasingly sophisticated robots, which play an ever-growing role in our society [1], [2]. Robots are becoming more capable of performing tasks that normally risk people's safety and health. These tasks typically require quick thinking and fast reaction times. Thus, the hardware driving the computation within robots must keep up with the increasing challenges of their use cases.

Key steps in the robotics computing pipeline include perception, motion planning and dynamics. Motion planning is particularly crucial because it is one of the more compute-intensive tasks. Motion planning is the task of calculating a series of valid configurations to get from a starting position to a destination position [3]–[5]. There are various algorithms for generating motion plans, among which Rapidly-exploring Random Trees (RRT) is often used to efficiently search a high-dimensional space for collision-free trajectories. [6]–[9].

RRT has conventionally been run on central processing units (CPUs) and graphics processing units (GPUs). While providing generality in computing and ease of programming, CPUs are not performant at computing RRT. GPUs, while better suited for an algorithm with parallelism like RRT, are not power efficient enough to be viable in a robotics system with power constraints. There are many use cases for robots where performance and power efficiency are paramount. Thus, it is necessary to find a different solution.

Field Programmable Gate Arrays (FPGAs) are capable of significantly accelerating algorithms like RRT while consuming less power compared to traditional CPU or GPU implementations. Hence our project aims to leverage FPGAs to enhance the speed and efficiency of motion planning. We plan to develop an end-to-end system that uses the FPGA-AMP accelerator to guide the motion of a robotic arm.

2 USE-CASE REQUIREMENTS

2.1 Accurate Motion Planning

Motion plans that are collision-free are essential for autonomous robotics. Robotic systems that fail to avoid obstacles will likely be unsafe and be repeatedly broken. These social and economic ramifications necessitate that the system we create generate accurate motion plans. Our motion planning module will consist of two steps, collision detection and path generation. Using data generated by a perception module, collision detection is needed to determine what viable, collision-free paths exist in the state space. Path generation then uses this result to find the shortest path between the start and goal position. Algorithms that are commonly used for motion planning are Probabilistic Roadmap (PRM) and Rapidly Exploring Random Tree (RRT). Both of these algorithms are non-deterministic which means they are not guaranteed to converge on a single solution. Algorithms for path generation consist of A* search, Dijkstra Algorithm, and heuristic path smoothing steps. We target designing a motion planning system that will **generate accurate, collision-free paths 95 percent of the time.**

2.2 Rapid Motion Planning

Using robotics in dangerous environments that are typically reserved for humans involves enabling the robot to adapt and react to its surroundings. In order to do so they must be able to quickly generate motion plans that account for dynamic, changing scenes. When run on modern CPUs, motion planning is generally too slow to handle such situations [1]. Academic research shows that using an FPGA for motion planning can achieve a 1000x speedup over a CPU [4], [5]. Due to the fact that motion planning makes up the majority of the processing time on a robot, this speedup should be sufficient to allow the robot to interact with dynamic scenes [1]. Due to improvements in modern CPUs since the research we cite, we target a **10x speedup of motion planning over a baseline CPU implementation**.

2.3 Efficient Motion Planning

Power and energy efficiency are important when considering the environmental impacts of electricity generation and the long-term economics of using robots. Efficiency is also critical for robotic systems that may be used in remote environments and/or are battery-powered. Modern CPU implementations of motion planning are not only slow, they are also extremely power inefficient. While GPUs achieve considerable speedup over CPUs, they suffer the same problem of power inefficiency. Therefore we plan on using an FPGA which will allow us to maintain the speedup advantages of the GPU at a fraction of the power cost. We target a **70 percent decrease in power and a 90 percent decrease in energy consumption when generating motion plans versus a baseline CPU implementation.**

3 ARCHITECTURE

The solution architecture is comprised of three major components: perception module, motion planning accelerator, and dynamics module.

3.1 Perception Module

The perception module is responsible for gathering and shaping environmental data, subsequently translating this information into a data structure that represents the real 3D space. This module is important in recognizing the forms and spatial positioning of objects within the scene. This module consists of both the depth-sensing camera and the on-board CPU in Figure 1.

Achieving accurate environmental perception is possible by employing computer vision and space mapping algorithms. While a multi-camera setup is one approach, offering extensive visual coverage, we have opted for using depth-sensing cameras. The rationale behind this choice is our aim to ensure the module's effectiveness in navigating complex and dynamically changing environments, where deploying numerous cameras may not be feasible.

The process of environment modeling and its conversion into a 3D data structure are integrated as one step through the execution of mapping algorithms on a CPU. In our design, we choose to adopt the voxel representation for its flexibility and compatibility with both software and hardware systems. This allows us to easily build an octo-tree of voxels and also encode them into a bit-stream, which can be stored compactly in a contiguous chunk of memory.

3.2 Motion Planning Accelerator

The motion planning accelerator stands as the cornerstone of our system, embodying the project's primary focus. Its responsibility is to guide the movement of the robot by finding a collision-free motion plan given a pair of `<start,end` positions. It leverages the 3D scene model generated by the perception module and runs motion planning algorithms on top of it.

To optimize performance and efficiency, we plan to divide the motion planning accelerator into two subcomponents: the RRT and collision detection accelerator, alongside the shortest path Accelerator. This division is illustrated in Figure 1. The RRT and collision detection accelerator is designed to rapidly generate a collision-free path through the environment, navigating the complexities of the 3D model with agility and precision. Complementing this, the shortest path accelerator takes the feasible paths identified by its counterpart and refines them, calculating the most efficient route from all possibilities. This component is instrumental in optimizing the trajectory for minimal distance or time, depending on the application's specific requirements.

Together, these accelerators form a system capable of delivering accurate, collision-free, high-performance motion planning solutions. The segregation of the two subcomponents enables us to perform optimizations such as pipelining, which can potentially bring significant performance improvements.

3.3 Dynamics Module

The dynamics module is the final module in the system that is responsible for converting the output from the motion planning accelerator into actionable control commands for the robot. This module's core functionality involves taking the computed path from the motion planning accelerator and ensuring that the motion can be physically feasible for the robot.

Once a path has been validated for physical feasibility, the dynamics module proceeds to translate this path into a series of control commands in a format recognizable by the robot controller. The integration of inverse-kinematics modules within this phase is crucial, as it enables the system to adapt the planned path to the unique capabilities and limitations of the robot's arm. This ensures that every movement is not only possible but also optimized for mechanical efficiency and safety, reducing the risk of errors or mechanical stress on the robot.

4 DESIGN REQUIREMENTS

4.1 Accurate Motion Planning

In order for our motion planning module to generate accurate and near optimal paths, we must maintain a high granularity representation of our state space. Voxels will be used to represent a partition of the state space and will be labeled by the perception module as either free or occupied. In order to achieve a high granularity representation, each voxel will only represent a minuscule volume. This will allow for precise paths and the ability to maneuver in tight spaces. **We will partition the state space into 3 dimensions, with each dimension spanning on the order of 1000 voxels**. The radius of reach for our robotic arm is approximately 0.5 meters. Using 1000 voxels in each dimension means each voxel represents 1 cubic centimeter.

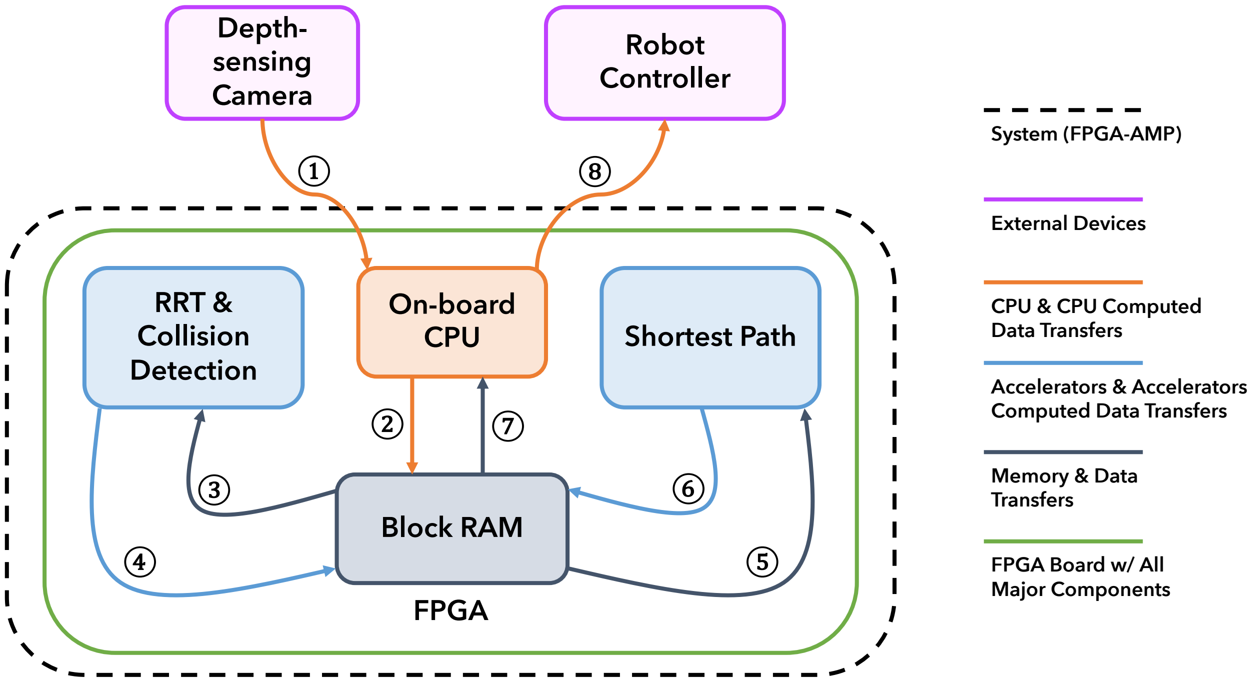
Depth-sensing Camera
Robot Controller
RRT & Collision Detection
On-board CPU
Shortest Path
Block RAM
FPGA
①
②
③
④
⑤
⑥
⑦
⑧
System (FPGA-AMP)
External Devices
CPU & CPU Computed Data Transfers
Accelerators & Accelerators Computed Data Transfers
Memory & Data Transfers
FPGA Board w/ All Major Components

Figure 1: Overview of the FPGA-AMP solution architecture. ①: The depth-sensing camera feeds the raw 3D data it captures to the on-board CPU. ②: The on-board CPU processes the raw 3D data, maps it into a grid of voxels, and serializes all voxels into a bit-stream, which is then stored on the block RAM on the FPGA board. ③, ④: The RRT and collision detection accelerator builds an RRT from a given pair of `<start,end>` positions while doing collision detection. It then stores the RRT it built as well as a collision-free path from `start` to `end` into the block RAM. ⑤, ⑥: The shortest path accelerator runs search algorithms to find the shortest collision-free path based on the output from the RRT and collision detection accelerator and stores the result in the block RAM. ⑦, ⑧: The CPU accesses the final result, runs inverse-kinematics algorithms to make sure that the robot arm can move along the path, and converts the path to a series of movement commands to the robot arm control Arduino board.

4.2 Rapid Motion Planning

In order to react to dynamic scenes, the robotics pipeline latency must be minimized. The average human reaction time is on the order of 200ms [10]. Because our system targets replacing humans in dangerous scenarios, it will have to react in a similar time frame. Our system has 3 main pipeline stages, perception, motion planning, and dynamics. We target a 300ms reaction time, which gives each stage 100ms to operate. Therefore, **the perception system must sample and generate collision voxels every 100ms in order to feed the other pipeline stages and meet the reaction time requirement.**

4.3 Efficient Motion Planning

In order for our system to achieve efficient motion planning, we must use a highly efficient and powerful FPGA. We will use an AMD Kria KR260 FPGA that consumes **30 Watts**, an optimal level of power consumption for battery-powered robots. The AMD Kria KR260 contains scalar cores, over 1000 DSPs, and significant block ram [11]. Our baseline CPU implementation will be run on an AMD 5950x. The 5950x has 16 cores, consumes 105 Watts, and is at a similar price point as the FPGA. We used these numbers to generate our targeted power reduction. Our targeted energy reduction was calculated using the power figure and our targeted speedup.

5 DESIGN TRADE STUDIES

For this project, we had many design choices to make and thus performed trade studies for each design decision. The decisions include: motion planning algorithms, shortest path algorithms, acceleration devices, hardware development tooling, and robot type.

5.1 Motion Planning Algorithm

The two motion planning algorithms used most often in robotics are Probabilistic Roadmap (PRM) and Rapidly Exploring Random Tree (RRT) [1]. Both algorithms are non-deterministic and involve taking random samples of the state space. These random samples are then connected via edges to form a graph. Eventually, the graph represents a subset of the viable, collision-free trajectories. Our decision to use RRT was primarily based on the host of high-quality literature on its acceleration [4], [5]. RRT contains intrinsically parallel properties which make it ideal for hardware acceleration.

5.2 Shortest Path Algorithm

The two shortest path algorithms we considered are Dijkstra's Algorithm and A* search. Dijkstra's is ideal when using PRM [12]. However, research has been done to combine RRT and A* into a more efficient algorithm called RRT* [8]. For this reason, we plan on using A*. We will integrate our RRT and A* implementations into RRT* if there is sufficient time and perceived performance gains.

5.3 Acceleration Device

5.3.1 CPU

Running motion planning on a CPU is relatively simple as open-source implementations of it are widely available in robotics libraries. As discussed before, there is parallelism to be reaped from RRT but it is hard to take advantage of on CPUs. Collision detection entails many independent, parallel calculations, while conventional CPUs only have around 8-16 threads available. The only benefit of keeping the motion planning computation on a CPU is that the CPU is responsible for orchestrating the rest of the robotics system, there is no overhead associated with transferring data to some external processor.

5.3.2 GPU

Graphics Processing Units (GPUs) are able to exploit the parallelism in algorithms like RRT and achieve significant speedup over CPUs. This being said, they suffer from extreme power inefficiency. Modern consumer GPUs can consume up to 450 Watts. The long-term economics and environmental impact of such a solution do not make sense.

5.3.3 ASIC

Application-specific integrated circuits (ASICs) take hardware acceleration to the extreme, providing even more performance gains and power efficiency than the solution option that we settled on. However, ASICs also have drawbacks, namely the high upfront development costs and long design cycles. In a scenario where hardware acceleration of robotics is operating at scale, fabricating ASICs may make sense, but for the purpose of a semester-long design challenge, we need a platform that is not as expensive and better suited for prototyping.

5.3.4 FPGA

We settled on FPGA acceleration because of the hardware acceleration and efficiency it provides while also being easily reconfigurable, striking a middle ground between CPUs, GPUs, and ASICs. As stated before, hardware acceleration is great for computation that has lots of parallelism, thus making motion planning a great use case for an FPGA. FPGA accelerated motion planning and robotics in general is also an active area of research, and we have obtained much guidance from the literature we have read so far on the subject.

5.4 Hardware Development

5.4.1 HDL

The traditional method for designing an accelerator is to use a hardware description language (HDL) like SystemVerilog. Using such a language allows for much lower and finer control over the accelerator that we build. Having full control is double-edged since the ability to control all the details in our system means we have to worry about all the details in our system. While all three of us in our group have experience with SystemVerilog, for a semester-long project we decided that we needed something that would enable us to develop our accelerator faster.

5.4.2 High-level Synthesis

High-level synthesis (HLS) came to mind as a better alternative to HDLs. For some background, HLS takes a software description of some computation and extracts the dataflow within it to get a hardware description. The translation from software to hardware preserves the correctness of the design. When making a change to

HLS is a better choice for this project for a few reasons. First, it is a more accessible development method at the cost of control over the finer details of the design. For example, it is hard to force the HLS compiler to put a register in between two signals, while it is trivial in an HDL. However, this level of control is not necessary for us to get speedup in our accelerator. The second reason is the fast design cycles we get from using HLS.

5.5 Robot

There are multiple reasons why we decided to use a robotic arm as the testbed for FPGA-AMP. Accelerated motion planning is most useful in high-dimensional spaces and for systems that have multiple degrees of freedom. In these scenarios, motion planning is inherently more complex and computationally intensive. Using a robotic arm means that we must do motion planning in 3D space. The robotic arm we have selected has 6 servo motors which means many degrees of freedom. Working with a robotic arm provides a sufficiently complex problem and real-world applications.

6 SYSTEM IMPLEMENTATION

Our hardware platform of choice is the AMD Kria KR260 Robotics Kit. This platform is new to us, and so since we are uncertain if we can get our tools (namely Vitis and Vitis HLS) to run on it, we also have the Avnet Ultra96v2 FPGA board as a backup. This is the same board that we used in 18-643.

6.1 Datapath Optimizations

6.1.1 Loop Unrolling

In many algorithms, there are sections of code that are repeatedly executed while being independent of each other. Such loops can be unrolled so that their execution is laid out in parallel on hardware. This allows the computation to be done more quickly while making use of more hardware resources.

In RRT, the most parallelizable portion is collision detection, since all points on a motion path can be paired with all potential collision points and checked for collisions in parallel. In high-level synthesis, loop unrolling is done explicitly by annotating loops in the software description of the computation with pragmas describing how the loop should be unrolled.

6.1.2 Pipelining

For long sequential computations, pipelining is another optimization technique that we will use in our accelerator to improve performance. Pipelining takes a long computation and cuts it up into smaller chunks, inserting registers between each chunk. By cutting up the computation and drawing it out over multiple cycles, the execution of multiple loop iterations can be overlapped. Cutting up long computations in hardware also improves the critical path of the circuit, increasing the clock frequency.

For RRT, there is a sequential step in generating the next random point and trying to extend the tree towards that point. Similar to loop unrolling, there is an HLS annotation that we must make to pipeline a certain computation.

6.2 Memory I/O Optimizations

6.2.1 DRAM

The DRAM on the FPGA serves as the intermediary location for both incoming data and the results generated during the execution of RRT. Facilitating this process, the host program running on the Arm core is responsible for transferring the perception data to the DRAM. The FPGA will continuously retrieve information from the perception data from the DRAM. Subsequently, the kernel performs calculations on the perception data and sends a motion plan back to DRAM for the CPU to read. The DRAM embedded within the Ultra96-V2 FPGA has a storage capacity of 2GB, accompanied by a measured read latency of approximately 500ns, which equates to several dozen cycles, contingent upon the computation's target frequency. This long latency emphasizes the need to do computations on locally buffered data, thereby minimizing the frequency of access to DRAM. The access pattern of DRAM is configurable using high-level synthesis.

6.2.2 LUTRAM & BRAM

For faster, more local storage of data to process, both LUTRAM and BRAM are memories available on-chip. LUTRAM is memory implemented using look-up-tables, the basic logic element for FPGAs. LUTRAM is the most local form of memory, but it comes with a trade-off that using LUTs for memory means fewer LUTs for logic/computation.

On the other hand, BRAM is a specialized memory, meant specifically for storing data on-chip. They allow for dual-ported accesses, meaning we are able to read and write to the memory simultaneously. Using HLS, we are able to configure the memory to different shapes. This means we can take a long array of bits and shape it such that we access it on a wider front, grabbing multiple bits at a time, allowing us to feed our datapath with more data. Most of our on-chip memory use will be done using BRAM.

6.3 Perception

The raw environmental data is collected by an Xbox Kinect One camera using its depth-sensing module. The camera is connected to the CPU via an adapter and USB link. The raw data is transferred by establishing a software bridge between the camera and the CPU through an open-source Kinect vision library, `iai_kinect2` [13]. The CPU processes the raw data and maps the raw environmental data into voxels in a 3D scene by running `octomap`, an open-source library [14]. The `octomap` library provides functionalities such as adding nodes, traversing the map, as well as its visualization component, `octovis`.

6.4 Inverse-Kinematics and Arm Control

Inverse-kinematics is the process of converting the path generated by the Motion Planning Accelerator into control signals for the robotic arm. In order to do this, we plan on using an open-source library called Visual-Kinematics [15]. The robotic arm is controlled by an Arduino which uses a simple API to set the angles of the servo motors [16]. The commands will be communicated from the scalar cores on the FPGA to the Arduino over UART [17].

7 TEST & VALIDATION

For all test iterations, we will use the same setup of the Xbox Kinect One camera and the robotic arm. The only difference is the motion planner, which can be either FPGA-AMP or the reference CPU implementation. The scene will be either static (pick and place task for the arm) or dynamic (pick from a shelf and place on a moving belt) and each scene will be used in 20 test iterations. A `Python` script will be used to generate random `<start,end>` position pairs for each test iteration.

7.1 Tests for Accurate Motion Planning

This section addresses our testing for the accuracy of FPGA-AMP's motion planning output (a design requirement mentioned in subsection 4.1).

For each test iteration, we will observe if the robot arm collides with any obstacle in the scene. Each collision indicates an incorrectly generated motion plan and counts as a failure.

Due to RRT being a heuristic algorithm, there is no guarantee that it will always generate the exact motion plan. So for all correctly generated motion plans, we compare the swept volume of the robotic arm's motion to the swept volume of the robotic arm's motion based on a reference implementation run on a CPU. Each comparison with less than 90% overlapped swept volume counts as a failure.

To meet the design requirement, the number of failures should be equal to or less than 5% of the total number of test iterations. Meeting the design requirement of 95% accuracy means that FPGA-AMP is able to meet the corresponding use-case requirement.

7.2 Tests for Rapid Motion Planning

This section addresses our testing for the performance of FPGA-AMP's motion planning algorithm execution (a design requirement mentioned in subsection 4.2).

For each test iteration, we will measure the wall-clock time of FPGA-AMP and the CPU reference implementation running motion planning algorithms for the same pair of `<start,end>` parameters. The time measurement starts when the motion planner receives the pair of parameters and stops when it produces a motion plan.

To meet the design requirement, FPGA-AMP's measured wall-clock time should be equal to or less than 10% of the CPU reference implementation's measured wall-clock time. Meeting the design requirement of 10$\times$ speedup means that FPGA-AMP is able to meet the corresponding use-case requirement.

7.3 Tests for Efficient Motion Planning

This section addresses our testing for the energy efficiency of FPGA-AMP's motion planning algorithm output (a design requirement mentioned in subsection 4.3).

For each test iteration, we will record the average power consumption and total energy consumption of FPGA-AMP and the CPU reference implementation running motion planning algorithms for the same pair of `<start,end>` parameters. The power and energy measurements for both implementations include all components necessary for booting the system and keeping it running.

To meet the design requirement, FPGA-AMP's measured energy consumption should be equal to or less than 2% of the CPU reference implementation's energy consumption. Meeting the design requirement of 98% less energy means that FPGA-AMP is able to meet the corresponding use-case requirement.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in Figure 2.

8.2 Team Member Responsibilities

- Baseline RRT: Yufei, Chris
- Simulation Environment: Matt, Yufei, Chris
- HLS-FPGA Environment: Matt
- μarch Design: Matt, Yufei, Chris
- Porting RRT to HLS: Matt
- Optimization: Matt, Yufei, Chris
- Perception: Yufei
- Robotic Arm Dynamics: Chris
- Robotic Arm & FPGA Integration: Matt
- Full System Integration: Matt, Yufei, Chris

8.3 Bill of Materials and Budget

We have a total budget of $600.00 and we are using $348.98 so far. For the bill of materials, see Table 1.

8.4 Risk Mitigation Plans

The biggest risk is that none of us have extensive experience with robotic perception and dynamics before. To account for this, we have started learning and reading about relevant topics and existing solutions that we could adopt early into the project. We have sourced some open-source libraries and plan to adopt them instead of writing our implementation of the algorithms for perception and dynamics tasks. In addition, we have also budgeted approximately two weeks of dynamic slack time in case any surprises come up.

9 RELATED WORK

Robotic motion planning using RRT is a well-researched topic [6]–[9], yet there isn't much work specifically accelerating motion planning using FPGAs. The work by Murray *et al.* [4], [5] is the closest we could find. Our work differentiates by not using pre-computed collision data and targeting a robotic arm, which has a higher degree of freedom in motion.

There are some work [14], [18] that accelerated motion planning using RRT and targeted robotic arms. But our work differentiates by using FPGA acceleration whereas theirs used neural networks.

10 SUMMARY

In summary, our design of FPGA-AMP uses FPGA to accelerate motion planning, a critical step in robotic computing. Our design aims to improve the performance of running motion planning by 10× compared to traditional CPU-based approaches with approximately the same initial cost and much less operating cost (due to less power consumption).

The upcoming challenges we are facing are robotic perception and dynamics systems implementation and system integration (described in more detail in subsection 8.4). One other challenge is in optimizing the FPGA-AMP implementation to make sure it meets the performance requirements. Since we have not started optimization yet, it is unclear to us how much optimization techniques and efforts are needed to meet the 10× performance goal.

Glossary of Acronyms

- CPU - Central Processing Unit
- FPGA - Field-Programmable Gate Array
- GPU - Graphics Processing Unit
- RRT – Rapidly-exploring Random Trees

References

[1] S. Liu, Z. Wan, B. Yu, and Y. Wang, *Robotic computing on fpgas*. Springer, 2021.

[2] C. R. Garrett, R. Chitnis, R. Holladay, *et al.*, “Integrated task and motion planning,” *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[3] Z. Wan, A. Lele, B. Yu, *et al.*, “Robotic computing on fpgas: Current progress, research challenges, and opportunities,” in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2022, pp. 291–295.

[4] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, “The microarchitecture of a real-time robot motion planning accelerator,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[5] S. Murray, W. Floyd-Jones, G. Konidaris, and D. J. Sorin, “A programmable architecture for robot motion planning acceleration,” in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, IEEE, vol. 2160, 2019, pp. 185–188.

[6] S. M. LaValle, J. J. Kuffner, B. Donald, *et al.*, “Rapidly-exploring random trees: Progress and prospects,” *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[7] M. Mohanan and A. Salgoankar, “A survey of robotic motion planning in dynamic environments,” *Robotics and Autonomous Systems*, vol. 100, pp. 171–185, 2018.

[8] I. Noreen, A. Khan, and Z. Habib, “Optimal path planning using rrt* based approaches: A survey and future directions,” *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[9] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *Ieee access*, vol. 2, pp. 56–77, 2014.

[10] Wikipedia, *Mental chronometry — Wikipedia, the free encyclopedia*, `http://en.wikipedia.org/w/index.php?title=Mental%20chronometry&oldid=1206420368`, [Online; accessed 01-March-2024], 2024.

[11] [Online]. Available: `https://www.xilinx.com/products/som/kria/kr260-robotics-starter-kit.html#specifications`.

[12] Wikipedia, *Probabilistic roadmap — Wikipedia, the free encyclopedia*, `http://en.wikipedia.org/w/index.php?title=Probabilistic%20roadmap&oldid=1209859792`, [Online; accessed 01-March-2024], 2024.

[13] T. Wiedemeyer, *Iai kinect2*, University Bremen: Institute for Artificial Intelligence, 2014 – 2015. [Online]. Available: `https://github.com/code-iai/iai_kinect2`.

[14] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots*, 2013, Software available at `https://octomap.github.io`. DOI: `10.1007/s10514-012-9321-0`. [Online]. Available: `https://octomap.github.io`.

[15] Dbddqy, *Dbddqy/visual_kinematics: Simple python-based kinematics solver for robot arm.* [Online]. Available: `https://github.com/dbddqy/visual_kinematics?tab=readme-ov-file`.

[16] Arduino-Libraries, *Arduino-libraries/braccio: Arduino braccio library.* [Online]. Available: `https://github.com/arduino-libraries/Braccio`.

[17] [Online]. Available: `https://www.arduino.cc/reference/en/language/functions/communication/serial/`.

[18] Q. Gao, Q. Yuan, Y. Sun, and L. Xu, “Path planning algorithm of robot arm based on improved rrt* and bp neural network algorithm,” *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 8, p. 101 650, 2023.

Description	Model #	Manufacturer	Quantity	Cost @	Total
Xbox Kinect One Camera	v1	Microsoft	1	$49.99	$49.99
Kinect Adapter for Xbox One	B-SPQxbox0001	KABCON	1	$24.99	$24.99
Ultra96 Development Board	v1	AVNET	1	from Capstone inventory	$0.00
Ultra96 Development Board	v2	AVNET	1	from 18-643 inventory	$0.00
Kria KR260 Robotics Starter Kit	SK-KR260-G	AMD	1	from AMD	$0.00
Arduino Braccio Robotic Arm	RB-Ard-81	RoboShop	1	$274.00	$274.00
					$348.98

Table 1: Bill of materials

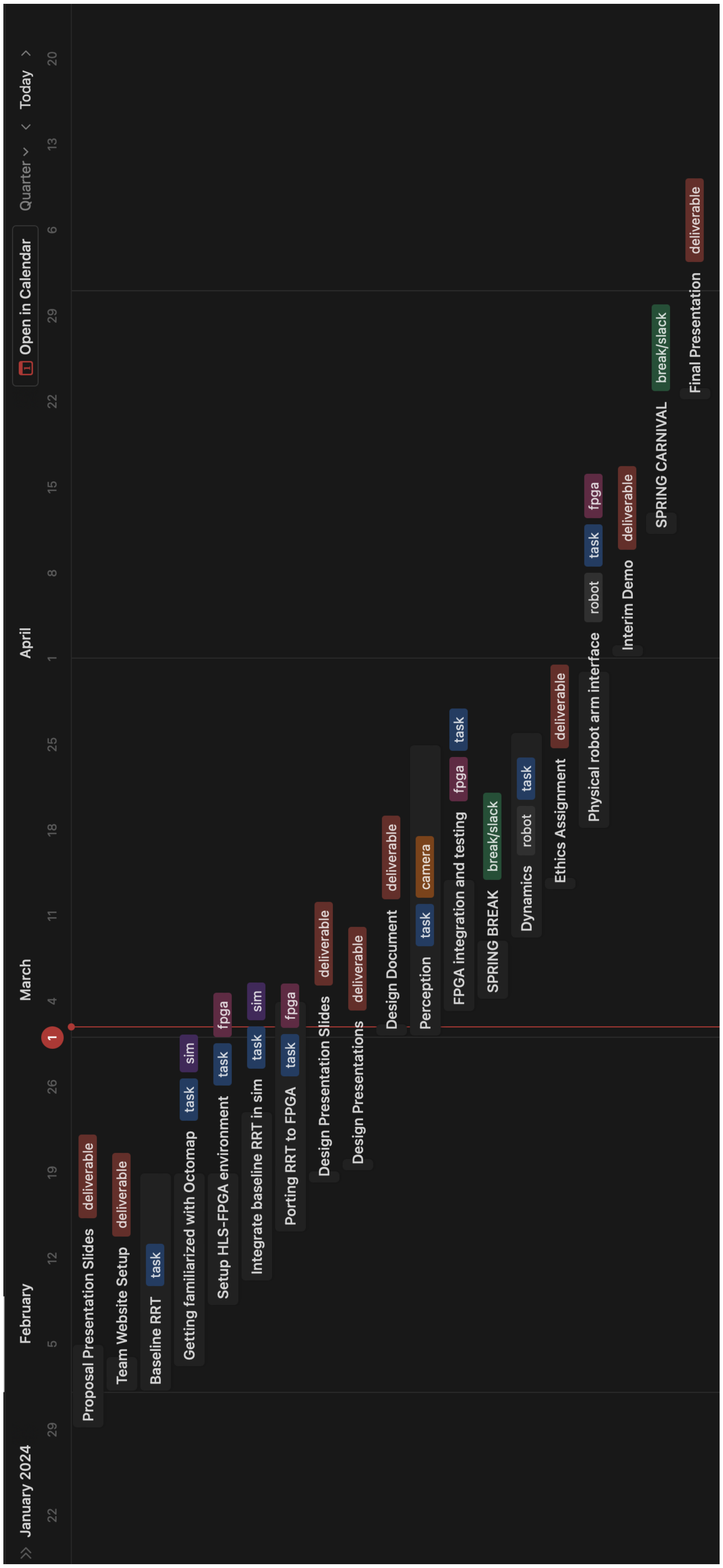

Figure 2: Gantt Chart. Tasks are labeled with types. Refer to subsection 8.2 for division of labor.