

FPGCave: Hardware Accelerated Point Cloud Registration

George Jurgiel
Department of E.E.C.S.
Massachusetts Institute of Technology
Cambridge, MA, USA
gjurgiel@mit.edu

Michelle Touma
Department of E.E.C.S.
Massachusetts Institute of Technology
Cambridge, MA, USA
mtouma@mit.edu

Abstract—We propose a design for hardware accelerated point cloud registration using an FPGA. By feeding the FPGA data from a 3D time of flight LIDAR a real time 3D map of an environment such as cave passage can be constructed. The FPGA compares consecutive frames of points using a simplified version of an iterative closest point algorithm. This finds an optimal ridged transform between the two frames to align them. By accumulating these transforms, a set of many frames are put into a common map coordinate space.

I. BACKGROUND

A. Cave Mapping

Mapping the internal passageways of caves has several challenges. There is no way to get a birds-eye view of the whole cave at once, and GPS is unable to determine location underground. The passages are also irregular in shape and move in three dimensions. All this information is vital for cavers to have so they can determine where they are in the cave. The process of mapping a cave is typically done by choosing survey points along the wall or ceiling of the passage and measuring the distances and angles between them to map out a path. The shapes and useful features of those passages are simply recorded in sketches taken by the mappers. This process is often done through the assisted use of LIDAR range finders, but is still sometimes done with analog tape measures. We aim to improve this process with a full 3D LIDAR to capture the passage and it's shape all at once.

B. Hardware

The LIDAR we are suing for this project is the O3D303, a 3D time of flight LIDAR. The camera sends out a pulse of infrared light which gets reflected off the environment to the cameras 176x132 optical sensor. By measuring the time it takes for the light to travel the camera can detect depth and thus generate a point cloud.

Many mapping algorithms will also use an inertial measurement unit (IMU) to determine how a camera moves, and thus reconstruct a map. This method however is very prone to accelerometer drift resulting in an inaccurate map. For this reason our design forgoes an IMU and instead determines the motion fo the camera and final map from only the point cloud data using a modified Iterative Closest Point (ICP) method.

To perform the ICP algorithm efficiently our design centers around a the Spartan FPGA on AMDs Urbana Development Board. This board also has 2 Gigabits or D-RAM that the design heavily uses. As an intermediate between the FPGA and LIDAR we are using a Raspberry pi to handle some communications and data storage. This setup is shown in Figure 1.

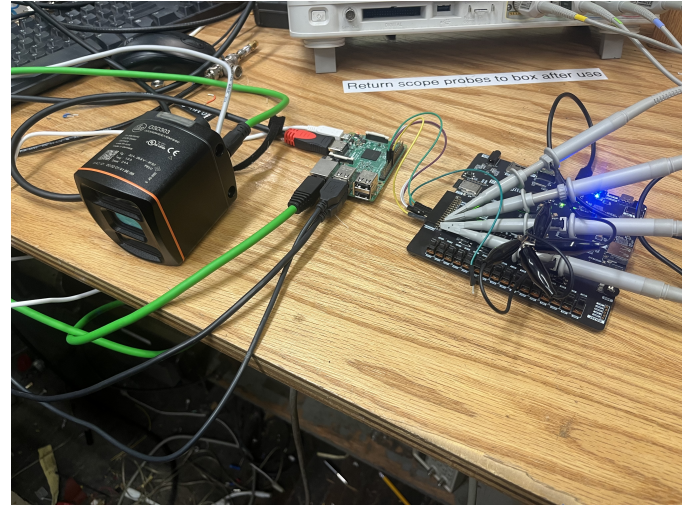


Fig. 1. Hardware Setup

C. Iterative Closest Point

Iterative Closest point (ICP) is a common method for point cloud registration. This is the task of aligning two overlapping point clouds of the same object which reside in different coordinate spaces. For ICP, each frame from the 3D LIDAR has it's origin at the location of the camera. As the camera moves the origin moves, so as the camera moves consecutive frames must be compared and aligned to put them in the same map coordinate space.

ICP does this by first pairing points from two consecutive frames that are predicted to represent the same actual point in space. This is typically done with a simple nearest neighbor algorithm. Then a ridged transformation of the points is determined to best align the all corresponding points. This process is repeated on the same two consecutive frames until the error

is satisfactorily minimized. Since consecutive frame from the LIDAR will have little moment, our design can dramatically reduce the number of required iterations. We additionally constrain the problem to only rotations in one dimension to avoid having to compute a single value decomposition which would be prohibitively resource expensive.

II. TOP LEVEL

At a high level the design acts as a simple data processing pipeline. However since multiple iterations through an entire frame of data are required for each pipeline stage data must be temporally stored in memory between stages, and each stage must read from and write to their own buffers without overlap. Those buffers can then be shifted over by one allowing data to be sent to the next pipeline stage as shown in Figure 2. This allows multiple frames to be in different stages of processing at the same time increasing throughput.

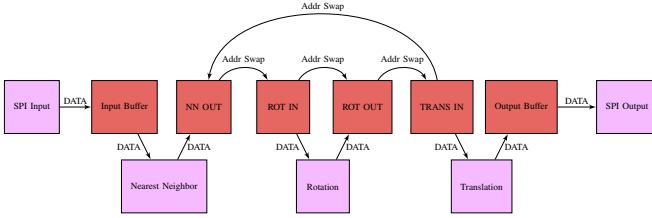


Fig. 2. Top Level Pipeline

However, since each frame is quite large this temporary storage must be loaded off the FPGA into DRAM. This puts a centralized memory manager at the center of the hardware design in figure 3 obfuscating the actual pipelined nature of the data processing. Data is piped between the memory manager and the processing modules using an AXI Stream protocol. Additional data buffers sit at the input and output of the pipeline for external data to be written to or read from over a custom SPI protocol.

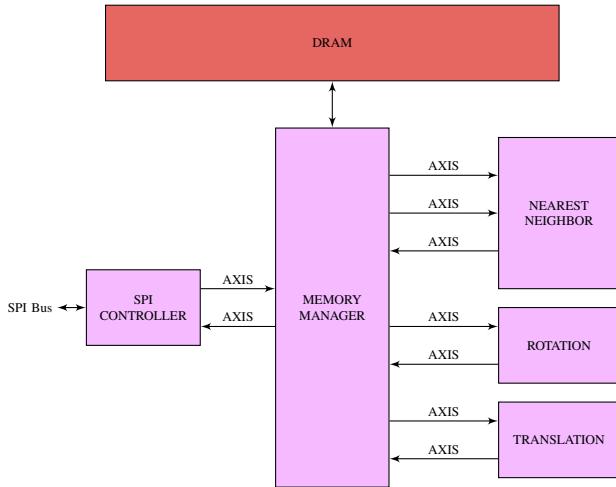


Fig. 3. Top Level Block Diagram

III. LIDAR INTERFACE

The O3D303 is a 3D LIDAR is made for industrial applications, so it communicates over TCP/IP allowing it to be easily send data over an Ethernet network. While this works well for it's intended use case, it is overly bulky and complex for simply transferring data to a local device, particularly an FPGA. TCP/IP is a protocol better suited to be handled on software and implementing it in hardware is complex and would provide negligible benefit. For this reason we choose to handle TCP/IP communication with the LIDAR using an external processor, a Raspberry Pi 3B, and transfer data to and from the Raspberry Pi over a custom SPI protocol shown in Figure 4, and a real measurement of these signals is shown in Figure 5.

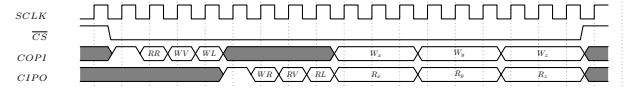


Fig. 4. SPI Timing Diagram

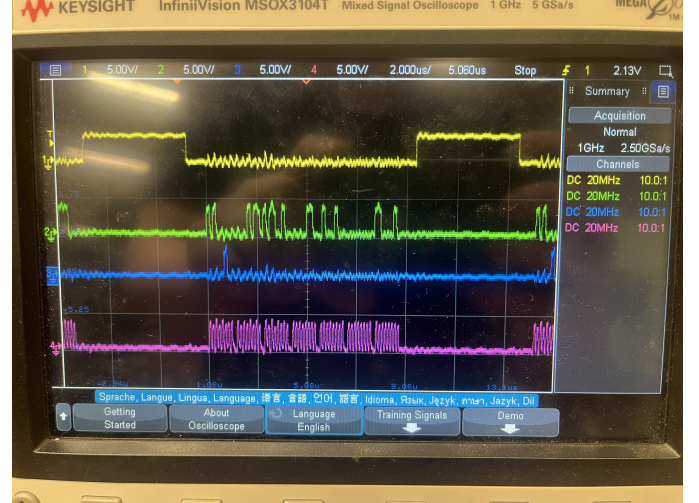


Fig. 5. SPI Scopeshot

A single SPI message is 7 bytes long, starting with one control byte establishing connection and allowing devices to express what data they wish to receive and what data they have available. The Raspberry Pi controller initiates communication with a high bit then proceeds to send a Read Ready bit (*RR*), a Write Valid bit (*WV*), and Write Last bit (*WL*). The Read Ready bit expresses that the controller wished to read back data from the FPGA, while the Write Valid bit expresses that the controller has new data to give to the FPGA. The Write last bit is used to express if this is the last point in a frame of points all taken at the same time.

This is followed by a similarly formatted string of 4 bits from the peripheral expressing if the peripheral is ready for a write (*WR*), is it has valid data for the controller to read (*RV*) and if that valid point is the last one in the peripherals output point buffer.

After this control bit, each side sends their point if it was requested. The point data is ordered, X, Y, Z in MSB order with each dimension having a bit depth of 16.

On the controller side there is a C++ program that handles capturing data from the LIDAR and sending and receiving data from the FPGA. It also generates a point cloud viewer that automatically updates the map point cloud to verify it's operation.

IV. MEMORY MANAGEMENT

As alluded to in the top level diagrams memory is divided into three main sections: Pipeline Memory, and Input Buffer, and an Output Buffer as show in in figure 6. The design requires four pipeline buffers whee the buffers rotate between modules. Each pipeline buffer needs to be able to store the maximum possible number of points, 23232, bout may hold fewer as points get filtered out during the process. The input and output buffers can be sized arbitrarily, but it generally makes sense to make the input be able to hold around 32 frame in case there is a lag in sending the data, while making the output as large as possible so the coprocessor does not need to pull data off the FPGA as often.

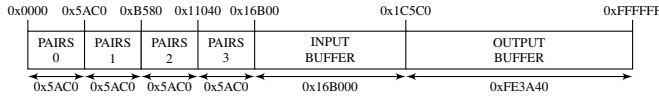


Fig. 6. D-RAM Layout

Shown in figure 7, the memory manager presents two (or three) AXI Stream ports to each module to read or write from. Modules can set a flag which will cause the memory manager to start streaming data to it, or the module can start streaming data to the memory manager to be written. The memory manager rotates through these ports in a form of time division multiplexing to handle the numerous sources of reads a writes all at once. It also keeps track of the state of the top level pipline to properly route this data to and from the correct sections of memory.

To properly handle the bursty nature of DRAM the inputs and outputs are all buffered with clock domain crossing FIFOs. there are additional stacker and unstackers to decompose the 128 bit chunks used by D-RAM into he 16 bit coordinate components. While the system rotating through memory ports and keeping track of address is written by us, the actual memory controller we use to communicate to DRAM is the UberDDR3 memory controller, a nice open source memory controller.

To keep track of where data should be written in the input and output buffer the traffic generator uses an address FIFO in which the start and end addresses of each new chunk of data get added as new data is written, and then get pulled out as that same data is read out. To generate addresses for reading and writing the traffic generator used counters design to loop across a limited range and values defined by the address FIFO. These requests are also stores in the command FIFO to keep

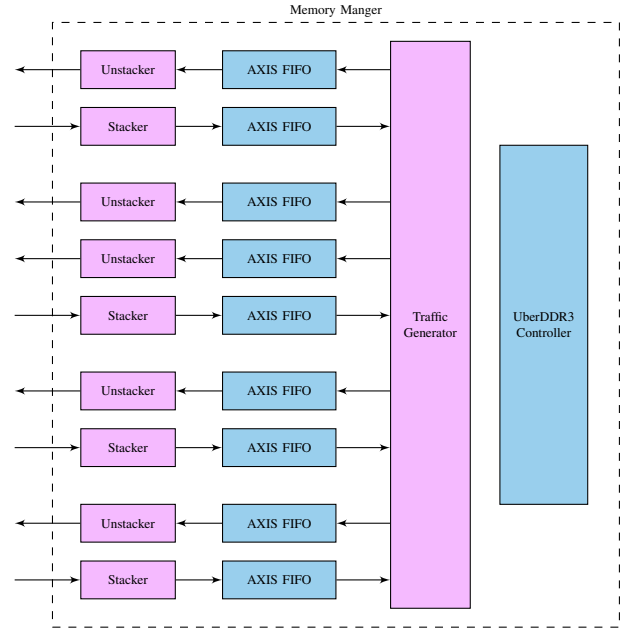


Fig. 7. Memory Manager Block Diagram

track of what transactions have occurred which is necessary due to the bursty nature of DRAM.

V. NEAREST NEIGHBOR

Nearest neighbor correspondence between the points of consecutive frames is done iteratively by checking every possible pairing and only taking the closest one for each point in the current frame. A threshold is also set for pairings that are too far apart so only sections of overlap are considered meaning the pairings may be variable in size. Points from the current and previous frame are streamed to nearest neighbor module but and very different rates. For each point from the current frame, all the points from the previous must streamed through the module to find the nearest. The module is the most simple in construction but takes by far the most resources.

VI. ROTATION PREDICTION

The rotation portion of the transform between frames is predicted iteratively, by going through a set of possible rotations, trying each rotation and finding the total error. Whichever transform had the least error is taken as correct. The sine and cosine values that make up this transform can be calculated using cordic algorithm. Early testing showed that by restricting the speed of the camera, we could use a python script to find to nearly guarantee that we didn't exceed a certain angle of rotation theta. The rotation module stores an accumulator that keeps track of the error associated with the current angle being tested. Additionally, the module uses fixed point tricks to represent the sine and cosine values of the angles being tested.

VII. TRANSLATION PREDICTION

The Translation portion of the transform can be accurately predicted by comparing the centroids of the two previous frame and the rotated current frame. This gives a translation matrix that gets incorporated into the rotation matrix, giving us a final transform matrix. This transform matrix is multiplied by the previous transform matrices and applied to the current frame, transforming it into the original frame's coordinate space.

VIII. RESULTS

The above sections outlines what was intended to be created for this project and how it would be achieved. Unfortunately, for reasons outside the scope of this report our group suffered significant logistical issues meaning this design was not successfully implemented in full. We had much success implementing communication from the Liard to the FPGA and back top the Raspberry Pi, with an additional visualizer added to the project while testing to view the depth information from the 3D LIDAR shwon in Figure 8. Many of the components of the main memory manager for DRAM were proven to work successfully in simulation, but we failed to show they worked with DRAM when put together. Finally the actual ICP algorithm was shown to work in proof of concept python scripts but was never implemented in Verilog. All this work is available in our projects git repository at <https://github.mit.edu/6205F25/fa25-6205-team33>.



Fig. 8. FPGA HDMI Output

IX. CONCLUSIONS

This was, to put it bluntly, a disaster of project management from it's inception. Organizationally, the project members failed to stay in regular communication regarding the state of the project. This lead to regularly failing to achieve deadlines and large amounts of stress for the group members who felt unsure what they had to complete and when. When changes to the plan had to be made, they occurred too late to properly change course exacerbating the problems.

The project idea itself was also poorly conceived. Research was done before hand to check the viability of this idea, and some examples were found of papers using hardware acceleration for SLAM algorithms. However, the existence of examples was incorrectly conflated with the viability of this as a project for the class. Something that warrants an entire research paper is unfortunately not replicable in half a semester a fact that in hindsight should have been obvious. Additional issues were found as the project continued as details these SLAM papers would gloss over turned out to have be quite complex in their own right.

This led to the related issue that the concepts behind the algorithm that we attempted to implement were challenging to implement in software that members were much more familiar with, and thus translating this to hardware was delayed too long. It was also found that even in the example papers we had found, many operations were not actually computed in hardware like we had predicted.

Overall I think the project could have been much more successful if it had been better organized, and the constraints that ended up being added in had been present from the start. It has emphasized to this group the importance of actively spending effort on organization, and determining how a project should work upfront.

From a technical standpoint, I think we found that this problem just isn't well optimized for hardware. Individual operations can be sped up quite significantly with hardware, such as the ridged transforms and nearest neighbor, but the algorithm as a whole is not well suited to the pipelining approaches that work well in hardware. Even with more time and better organization I don't think the project would be able to work particularly well.

More specific to the technical work done for this project, I don't think managing memory with a large set of axis streams getting triggered by clock domain crossing signals was effective. I think this put too much memory organization challenges on the traffic generator. Instead a system where each module generates requests to read a write to individual addresses on its own would work better. Structuring this such that these are still "virtual" addresses that the traffic generator routes to specific blocks of memory is likely how I would do this, but it would allow each module to have more control over the data it is receiving and split up the complexity to make for a more readable project.

X. DIVISION OF WORK

trying to best leverage the skills and knowledge of the group members, George primarily worked on communication with the LIDAR, memory management, and coordinate the top level state of the pipeline. Michelle was in turn tasked with devising and implementing the modified ICP algorithm and the corresponding Nearest Neighbor, Rotation, and Translation modules. Michelle unfortunately left the group before these ICP modules were successfully implemented in Verilog, and George was not able to complete that task before then end of the project. Before having to leave the group work on the

abstract and presentations was divided evenly, but the large majority of the preliminary and the final report was written by George.

ACKNOWLEDGMENT

Thanks to Joe Steinmeyer (course instructor and our project advisor), Monica Chan (TA), Yoland Gao (TA), Kiran Vuk-sanaj (TA), and Hasan Zeki Yildiz (TA), and the entire 6.205 Teaching staff.

Thanks to Oliver Trevor for project advice, the MIT Radio society for a space to store hardware and work on the project, and MIT Swapfest for finding the LIDAR for this project was purchased.

And thanks to Sunny and Kalashnikov, the cats of Tetazoo, for emotional support.