6.375 Final Report: Hardware Acceleration for Power Intelligent Wearable Navigation Systems

Andre Aboulian, Ishwarya Ananthabhotla

May 11, 2016

1 Project Objective

In today's world, wearable electronics and devices are becoming of interest in multiple contexts– from the perspective of aesthetics, health and medical monitoring, assistive devices, and connectivity on-the-go. While technology has thus far been focused on computational productivity, feature expansion, and small form-factor, absorption of such technology by the consumer market is strongly limited by excessive power consumption. As such, it is in the best interest of a hardware designer to consider the inclusion of dedicated hardware whenever possible, but also simple algorithms for intelligent power delivery that can be concurrently managed by this hardware.

This project seeks to explore this concept as a part of a system designed to be a wearable navigation device for visually-impaired individuals. The MTL Energy Efficient Circuits Group has developed an initial prototype of such a device that employs a Time of Flight (ToF) camera to capture depth data about the environment and process it to identify the presence of obstacles within a user's field-of-view, finally providing haptic feedback about this information. However, a power expenditure upwards of 15 Watts, resulting from large frame dimensionality, active illumination panels, and statically high frame rates, presents a sufficient barrier to usability and "wearability" throughout the day. In order to work towards a next generation prototype that is more power-friendly for such an application, the overarching goals of this final project are twofold– to work with the next generation ToF camera (that is smaller in dimensions) to (1) reproduce critical and computationally features from the old system (namely, generating a point cloud) and improve them by implementing them in hardware, and (2) implement a set of dynamic power scaling algorithms for the aforementioned power hungry features, also in hardware. Succinctly stated, our in-scope objectives for this project are as follows:

- Demonstrate that the depth data to point cloud data conversion can be implemented in Bluespec for hardware
- Demonstrate that three power scaling algorithms can be implemented in hardware, without visibly (qualitatively) altering the point cloud
- Demonstrate a full system with a simulated feedback loop using pre-captured camera data, implemented on the Zynq Mini-ITX 100 Board
- Understand the benefits of implementing the point cloud algorithm and the power scaling algorithms in hardware, in terms of space, speed or power

A detailed explanation of the camera specifications and of the power scaling algorithms can be found in the section below.

2 Background

2.1 Time of Flight Camera

The ToF camera used in this system is Texas Instrument's OPT8320 chipset, with a frame dimension of 80x60 (QQQVGA) and a range of approximately 1 meter given the default, short-range illumination configuration for this proof-of-concept. The ToF camera operates by actively modulating infrared illumination at a known frequency and capturing the light that is returned to a sensor array after it has been reflected from various obstacles in the environment. The sensor array calculates the phase difference between the reflected light and the pulsed light, and uses this quantity in conjunction with the known unambiguous range (which is a function of the modulation frequency) to extrapolate the distance to the object. This process is undertaken for each pixel in the sensor array, so each frame returned by the ToF camera is a 80 x 60 depth map of the scene in its field of view. The maximum internal frame rate of this chipset is 4000fps, and the default configuration has a peak power expenditure of 2 Watts. However, typical applications of this device require custom illumination panels tailored for larger distances, which are likely to consume much more power.

In order to provide some context for the way the power consumption of the chipset changes as a function of frame rate, illumination, and integration, sample measurements for the default configuration can be found in the plots in Figure 1 and Figure 2.

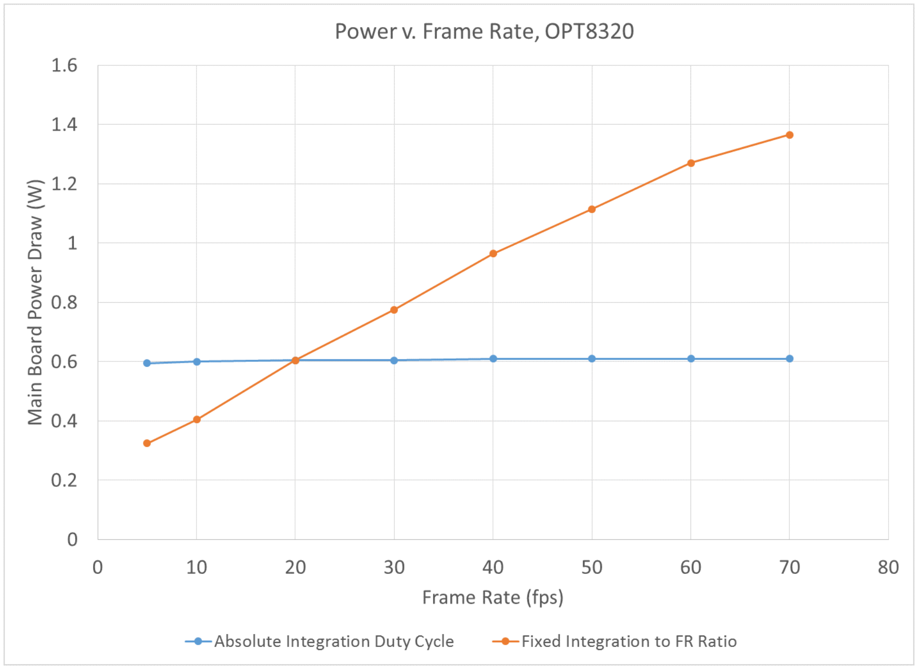


Figure 1: OPT8320 main board power consumption as a function of frame rate, at illumination panel current of 30mA. The blue plot shows a fixed integration duty cycle of 20 percent, where the orange plot displays a fixed ratio equal to the integration duty cycle over frame rate, set here at 0.01. For most applications, integration is required to increase with the frame rate for best results.

2.2 Algorithms

Four sets of algorithms – Point Cloud Generation, Scene Statistics, Scene Differencing, and Step Rate Detection – each as a separate Bluespec module, have been implemented as a part of this project.

2.2.1 Point Cloud Generation

The first is the transformation of the two-dimensional depth map from each camera frame into a 3-D point cloud. The points correspond to coordinates in the real world. The generation of a point cloud is critical to any guidance and mapping application, and requires efficient computation because it needs to be performed pixel-by-pixel and frame-by-frame. The depth map to point cloud conversation can be thought of as a geometric transformation derived from the pinhole camera model, shown in Figure **??** below:

While the step-by-step derivation will be bypassed here, the final transform is concluded to be:

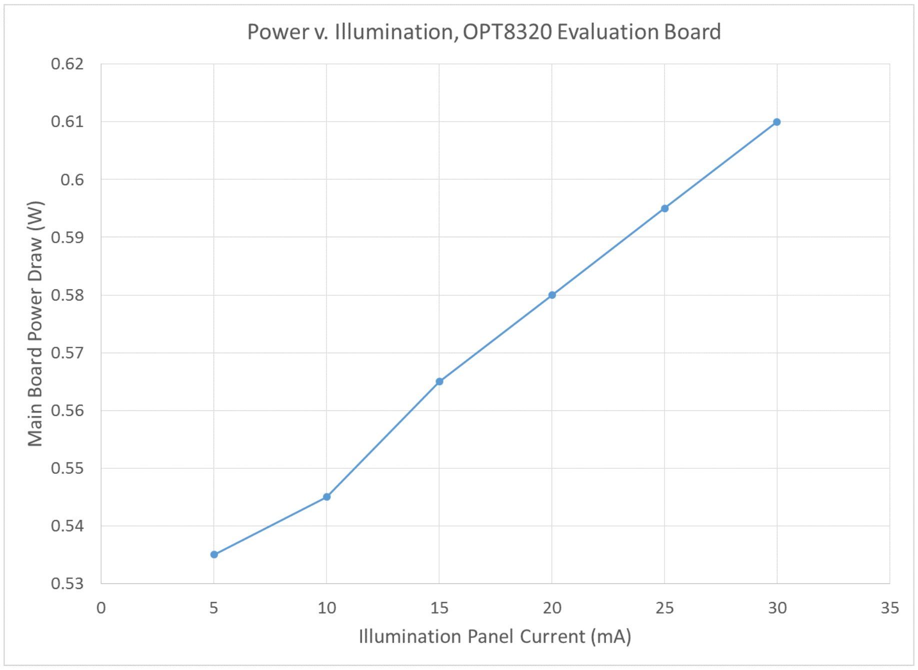


Figure 2: OPT8320 main board power consumption as a function of illumination, at frame rate of 30 fps and 20 percent integration.

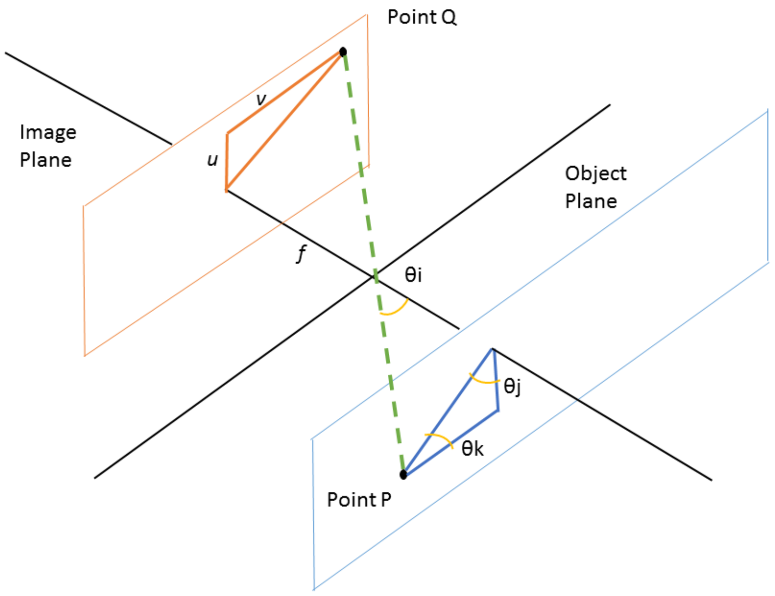


Figure 3: This figure shows the pinhole camera model that was used in the pointcloud transformation for this study. We find point P, the point in the real world, point Q, the point on the image plane, and depth ray D that is read by the imager.

$$
\begin{aligned}
X_P &= \left[\frac{c}{2f_{mod}} \cdot \frac{\phi(u,v)}{2\pi}\right] \cdot \sqrt{\frac{f}{1+u^2+v^2}} \\
Y_P &= \left[\frac{c}{2f_{mod}} \cdot \frac{\phi(u,v)}{2\pi}\right] \cdot \sqrt{1-\frac{f}{f+u^2+v^2}} \cdot \frac{u}{\sqrt{u^2+v^2}} \\
Z_P &= \left[\frac{c}{2f_{mod}} \cdot \frac{\phi(u,v)}{2\pi}\right] \cdot \sqrt{1-\frac{f}{f+u^2+v^2}} \cdot \frac{v}{\sqrt{u^2+v^2}} \\
u &= (x - \frac{Width}{2}) \cdot \tan(\frac{FOV_x/2}{Width/2}) \\
v &= (y - \frac{Height}{2}) \cdot \tan(\frac{FOV_y/2}{Height/2})
\end{aligned}
\tag{1}
$$

where FOV_x and FOV_y are the horizontal and vertical fields of view respectively, $Width$ is the width of the frame in pixels, $Height$ is the height of the frame in pixels, and x and y are the indices of the Point Q as read from the ToF sensor array.

2.2.2 Scene Differencing

The "Scene Differencing" algorithm, the first of the three power scaling algorithms, is essential toward determining how quickly the environment is changing around the user. Each successive frame is compared to the previous one to ascertain whether they are roughly the same. The similar frame is said to be "skipped", meaning it provides little new data as compared to the previous frame.

This algorithm performs a gaussian convolution across each depth frame as a low pass filter, to filter out noise from the comparison. The effect of the filter is visible in Figure 4. A pixel-by-pixel difference is taken between the current and previous frames, the sum of which is the total difference in the frame. If this value is below a particular threshold g_{skip}, then the similar frame is marked "skipped". The entire comparison is expressed computationally in Equation 2.

$$SkipFrame := \sum |gaussian(Phase_{x,y,i}) - gaussian(Phase_{x,y,i-1})| < g_{skip} \tag{2}$$

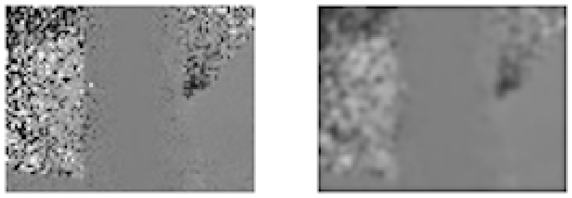


Figure 4: A frame of Phase data (left) transformed with a Gaussian kernel (right).

2.2.3 Step Rate

An additional algorithm, this time to characterize the motion of the user relative to the environment, is implemented with the use of data from an Inertial Measurement Unit (IMU) that would be integrated into a new system (and existed on the previous system). Periodic peaks in vertical acceleration are detected by passing temporal data through an FFT block and peak finding in order to determine the "Step Rate" at which a user was moving. This step rate algorithm would be used in conjunction with the scene differencing algorithm to scale the ToF Camera's frame rate dynamically.

2.2.4 Scene Statistics

The final algorithm implemented in this system is a "Scene Statistics" algorithm to be able to dynamically scale the base illumination voltage required by the LED panel. For the illumination configuration present in this system, three different values of illumination can be chosen for the LED panel. An intentional metric that favors the largest, nearest objects in determining the minimally sufficient illumination for a given frame is imposed. Given the confidence and phase that is assigned to each pixel in each frame, the following procedure is followed to compute a change in illumination value (increase by one step, decrease by one step, or remain the same):

1. Compute the histogram H_p for all 4800 pixels, assigning bins with a resolution of 0.15m per bin
2. Impose confidence threshold function upon the histogram to obtain $t_c(d)$, where d is distance
3. Select the N largest bins that comprise of M percent of the total pixels in the scene
4. Compute the direction of change in illumination as:
   $v_p(n) = \frac{t_c(n)}{H_p(n)} \leq 0.25$

   $V_p = \sum_N v_p(n)$

   $v_d(n) = \frac{t_c(n)}{H_p(n)} \geq 0.75$

   $V_d = \sum_N v_d(n)$

   $\Delta I = sign(w_1 V_p + w_2 V_d)$

   where v_p and v_d are positive and negative votes respectively assigned to a single bin, V_d and V_p are the frame's collective positive and negative votes, and w_1 and w_2 are tunable weights that can be set if, for example, a low-power mode values power reduction assignments more than power increase modes, or vice versa.
5. Return ΔI

Each of these algorithms were first implemented in a Python software simulation framework as a reference for the hardware implementation. The microarchitecture and Bluespec implementation details for each of the algorithms can be found in the sections below.

3 High Level Design and Test Plan

The diagram in Figure 5 presents a high-level abstraction of our system. As shown, each of the four algorithms detailed above are translated into an independent Bluespec module. Each module was developed separately with a corresponding Bluespec test bench that was used to verify the output against our software simulation before synthesis and full system integration. The system is completed by a software piece, which is responsible for delivering phase and confidence data from the camera to the four modules and aggregating their outputs into a controller that updates the frame rate and illumination parameters for the next set of frames to be delivered. This software-to-hardware interface was written using Connectal.

The implementation of this system was done on a Zynq Mini-ITX100 board, a platform containing an FPGA for the hardware end and an ARM core to serve as the host for the software end. Test infrastructure was implemented to allow the ARM core to load pre-captured camera frames into a frame server, "deliver" them to the Bluespec architecture to emulate the behavior of a live camera based on the feedback from the modules, and record its behavior for offline analysis. This allowed us to verify the performance of the entire system.

Details regarding the architectures of individual modules and of the software camera controller interface, as well as a presentation of test results, are described in the sections below.

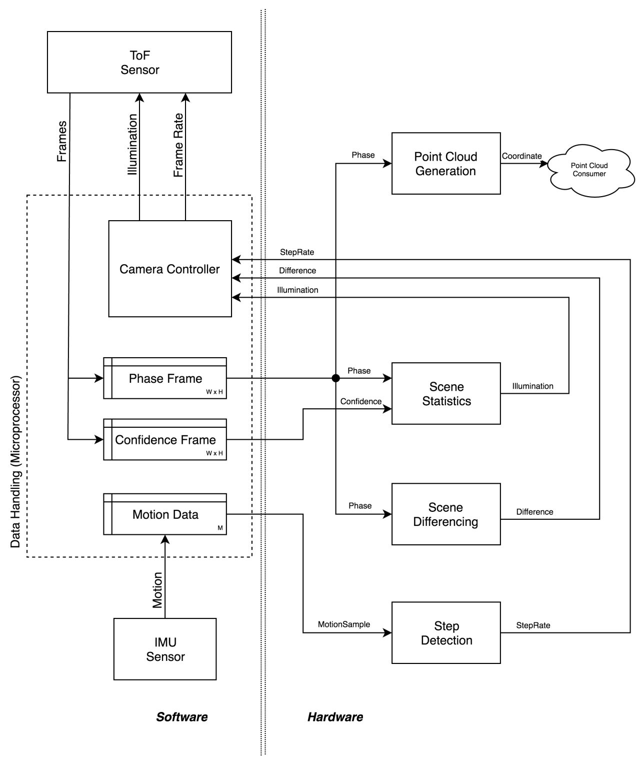


Figure 5: High level diagram of the system.

4 Microarchitecture

Each of our four algorithms required rearchitecting to translate them from their software counterparts into the hardware domain. Storage, recursion, indexing, and arithmetic influenced the design of our hardware modules. The overall hardware system in Figure 4 shows the interactions between modules and input/output queues between components. Each of the four blocks is elaborated in this section.

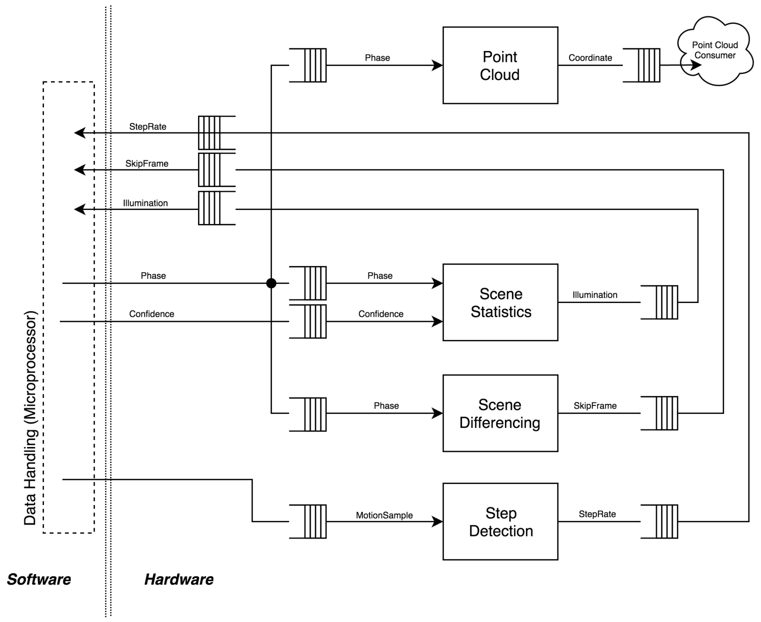
4.1 Pointcloud

A point cloud coordinate can be computed from each phase pixel without knowledge of the overall frame. For this reason, the interfaces to the module consumes one phase pixel for each coordinate that is produced. A complete frame is computed after 4800 cycles.

The bulk of the point cloud algorithm computes a normalized spatial coordinate corresponding to each pixel position. Since this depends only on the location of the input pixel and not its value, the computation is redundant across frames. The location of each pixel is taken relative to the center of the pixel plane, so the unsigned result is identical in each quadrant of the plane due to radial symmetry. To leverage these properties, these normalized coordinates are pre-computed for only a single quadrant of the pixel plane. These values are loaded into the BRAM, and a sign-change is applied after retrieval relative to the quadrant accessed.

Phase pixels are input in column-major order throughout the system. The *Fetch* stage relies on this specification to iterate over all X, Y locations using its internal *Pixel Counter*. This pixel location is mapped to a quadrant-insensitive index "PCIndex" used to query the corresponding coordinate from BRAM. The pre-computed coordinates are normalized, for simplicity, for a Phase magnitude of 1.0.

The *Scale* block dequeues an input Phase and a normalized Coordinate at once. Note that the output of the BRAM has a built-in FIFO. The Phase and Coordinate pair maintain synchrony purely by the order in which they are expected. The *Scale* block simply scales the normalized Coordinate by the input Phase to



Type	**Purpose**	**Bit Equivalent**
Phase	single phase pixel	UInt#(12)
Confidence	single confidence pixel	UInt#(12)
MotionSample	single IMU point	FixedPoint#(16,16)
StepRate	computer step rate	FixedPoint#(16,16)
SkipFrame	current frame similar to previous	Boolean
Illumination	illumination power level	UInt#(2)
Coordinate	3D point cloud coordinate	Vector#(3, FixedPoint#(8,16))

Figure 6: Microarchitecture overview of the system. Data types are indicated in the table.

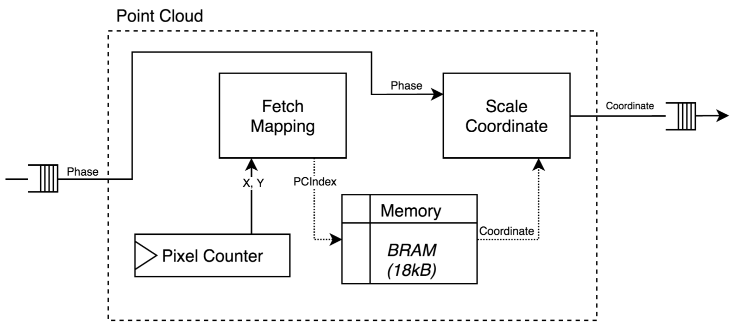
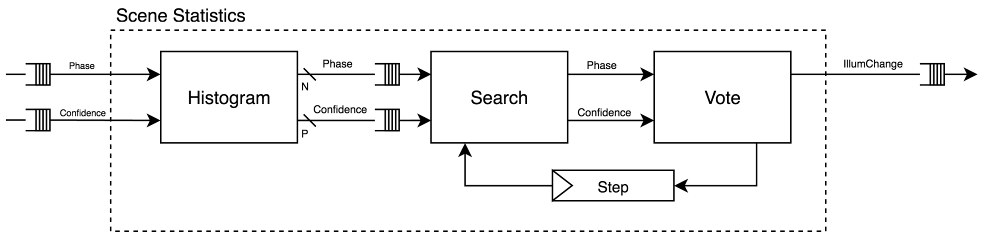


Figure 7: Microarchitecture blocks within the "Point Cloud" module.

obtain the point cloud Coordinate, queuing this output for later consumption.

4.2 Scene Statistics



Type	Purpose	Bit Equivalent
N	number of Phase bins	CONST 30
P	number of Confidence bins	CONST 30

Figure 8: Microarchitecture blocks within the "Scene Statistics" module.

The microarchitecture of the Scene Statistics module is shown in Fig. 8 above. As shown, phase data is input pixel-by-pixel into the histogram block, and assigned to a bin after a conversion from phase to metric distance. The histogram is contained in a vector of registers. Additionally, the confidence value assigned to each pixel is compared to the confidence threshold function, and stored in a threshold vector of registers if the result is positive. After all 4800 pixels in a frame have been received and processed in this manner, the histogram and threshold vectors are fed to the *Search* block so that the largest bin could be identified. The value in the largest bin is then passed to the *Vote* block in order to assign an upvote or a downvote to that bin, and checked to verify the percentage of total pixels that have been assigned votes so far for this frame. If the minimum number of pixels needed to be processed, set by the check threshold, had already been in bins that were assigned votes, the sum of the votes would be written to the output FIFO. Otherwise, the control flow of the module would return to the Search block to identify the next largest histogram bin.

In terms of design, note that FIFO's are placed at the input and output of the entire Scene Statistics block, and between the histogram and the remainder of the processing blocks. However, the *Search* and *Vote* blocks remain rigid without consequence. All post-histogram operations take no more than a few hundred cycles regardless of the pixel distribution, whereas the histogram operation alone requires 4800 cycles.

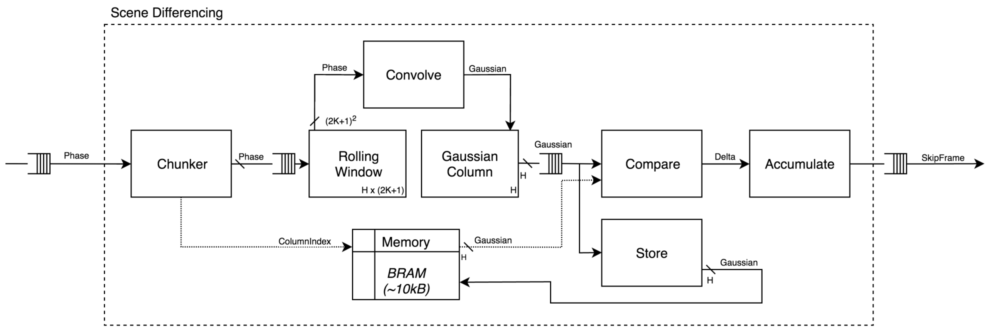
4.3 Scene Differencing

The scene differencing module consumes Phase pixel values and produces a SkipFrame boolean after an entire frame has been processed. The SkipFrame flag is set if the current frame was roughly similar to the previous one. Note that no output is produced after the first frame, since the concept of a previous frame is not yet valid.

The Phase pixels arrive in column-major order, so data travels through the pipeline in Figure 9 as columns of height H. The *Chunker* assembles individual Phase pixels into such columns. The *Rolling Window* maintains enough columns to perform the convolution with a K-width kernel. The window consumes a new Phase column once the existing one has been processed, discarding the oldest column.

The *Convolve* block computes a single Gaussian each cycle, using the phase values within the window. The Gaussian is always centered on the middle K^{th} column and moves down the column each cycle, as depicted in Figure 10. When surrounding pixels are out of bounds, their value is assumed to be zero. The view of the *Rolling Window* as a simply a grid of registers is oversimplified, in that the nested vectors have validity flags associated with them. This is necessary for the beginning and end of each frame, so that these invalid columns are considered "out of bounds" rather than affected by the beginning of the next frame. The results of the Gaussians are stored in the *Gaussian Column*, which is queued for the next stage once it has been completely populated.

Notice that in the microarchitecture diagram (Figure 9), the *Chunker* requests the previous frame's corresponding Gaussian column from the BRAM. So once the current frame's Gaussian column is later



Type	**Purpose**	**Bit Equivalent**
H	number of pixels in column	CONST 80
K	convolution width	CONST 3
Gaussian	gaussian pixel value	UInt#(16)
Delta	absolute difference in pixel values	UInt#(16)

Figure 9: Microarchitecture blocks within the ”Scene Differencing” module.

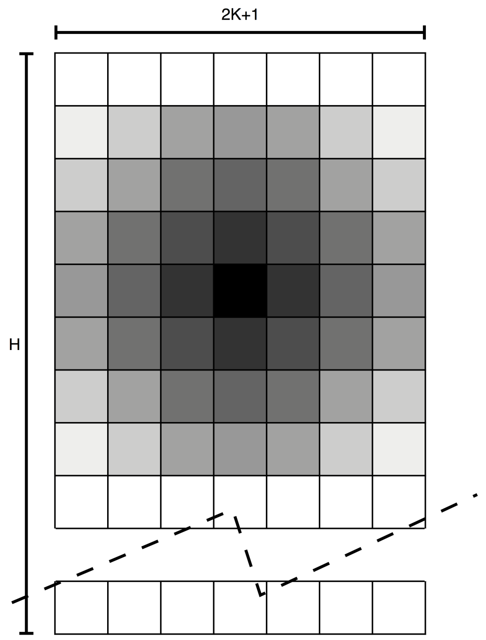
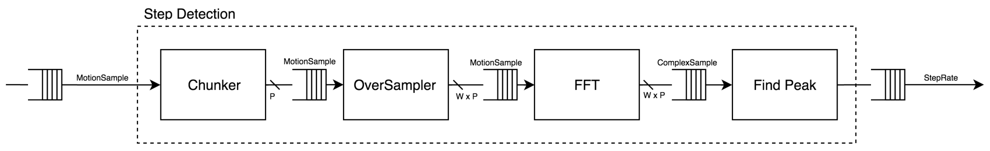


Figure 10: The *Rolling Window* for a K-width kernel is comprised of a series of Phase registers, represented as individual cells. The Gaussian kernel is depicted as densities (Black = 1.0) and superimposed on the convolved Phase values. The kernel is always centered around the middle column; here, the gaussian of $y = 4$ is being computed.

enqueued, the BRAM's built-in FIFO is ready to dequeue the corresponding column from the previous frame. The *Compare* block then dequeues the same Gaussian column from the current and previous frames together, and computes the sum of the differences between pixel pairs as "Delta". Simultaneously, the *Store* block writes back the new Gaussian column to memory. The *Accumulate* block adds the deltas for each column in the frame and outputs a positive SkipFrame flag if the sum is less than a calibrated threshold.

4.4 Step Rate



Type	Purpose	Bit Equivalent
MotionSample	single point of accelerometer data in time domain	FixedPoint#(16,16)
ComplexSample	single point in frequency domain	Complex#(FixedPoint#(16,16))
StepRate	computer rate of user stepping	FixedPoint#(8,16)
P	number of new motion samples taken per second	CONST 64
W	number of seconds spanned by the FFT window	CONST 2

Figure 11: Microarchitecture blocks within the "Step Rate" module.

The microarchitecture of the Step Rate module is shown in Fig. 11 above. The module operates by receiving as input FixedPoint accelerometer measurements from external MPU9250 data taken at a sample rate of 20 Hz. The data stream is chunked into frames of 128 data points, which allows each frame to contain approximately 6 seconds worth of step information. These frames are then passed through a *Chunker* block which delivers subsequent frames with 50 percent overlap to the previously delivered frame. These frames are then passed through a custom super-folded, pipelined FFT implementation and then through a Cordic block in order to obtain a magnitude representation of the data. Lastly, the magnitude spectrum of each frame is passed to the *Find Peak* block, which computes the step rate associated with the frame and returns the data to ARM processor's camera control module.

A high level architecture of the Superfolded FFT can be found in Fig. 12. A radix-2, Pease transform based implementation is used. Note that the implementation uses only single Butterfly node to minimize logic and area utilization. The original circular implementation that we developed as part of the class required more DSPs than our original evaluation platform, the Zynq Zedboard, had to offer, which motivated the move to a superfolded implementation.

4.5 Camera Controller

The software camera controller is responsible for delivering the camera frames to the four bluespec blocks and computing a new set of control parameters for the camera based on the outputs of the modules. Since this project used Connectal to bridge the software and hardware components, the camera controller was written in C++ for the ARM core located on the Mini-ITX100. For the illumination delta outputs, a simple moving average across three seconds (the number of frames would vary based on the frame rate) was maintained and assigned as the new illumination value. For the skip frame flags that resulted from the scene differencing module and the step rate value that resulted from the step rate module, the following simple proportional controller was used.

$$FrameRate_n = FrameRate_{n-1} + (k + q \cdot StepRate)(Desired - \frac{1}{N}\sum SkipRate_{n-1}) \tag{3}$$

where k is the proportional gain term, and q is a tuning factor regarding the degree to which the gain can be perturbed by the step rate. Intuitively, we seek to accelerate the change in step rate based on the presence or lack of rapid user motion.

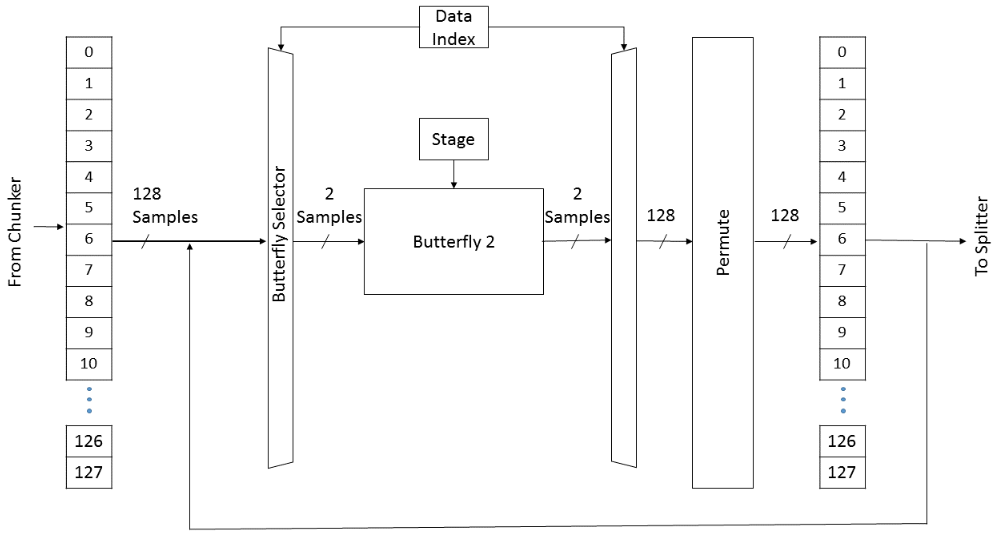


Figure 12: High level diagram of the superfolded circular FFT module used to compute the step rate in the IMU module.

While ultimately we would have liked to have been able to compile the ToF Camera SDK for the ARM core for real time delivery of frames and feedback for parameter updates, there were several hurdles to successful compilation of the software. After discussion with our mentor, it was decided that the camera data would instead be simulated by a "frame buffer"that would load phase and confidence data from a file pre-captured at a high frame rate and at all stages of illumination, and "deliver" the next appropriate frame based on the feedback received from the Bluespec modules.

5 Implementation Evaluation

5.1 Synthesis Results

The system described above was synthesized for the Mini-ITX100 board, and select results regarding critical timing and memory allocation are presented in this section.

The critical path length of the entire system is 17ns, which is a result of the FFT block within the Step Rate module. With this timing constraint, the system can be clocked at 60Mhz. A summary of the throughout of each module operating at this clock frequency compared with the constraint imposed by the frequency of input frame delivery to that module is shown in the table below.

Module	Throughput (Samples per Second)	Frame Delivery Constraint
Scene Statistics	12,000	4Khz
Step Rate (IMU)	100,000	20Hz
Scene Differencing	12,500	4Khz
Point Cloud	12,500	4Khz

As can be seen, even at the maximum internal ToF camera frame rate of 4000fps, the throughput of each of the four modules in not constrained and there is no latency. However, for the modules that operate on a pixel-by-pixel basis such as the point cloud module, this presents a limit to scalability without requiring a redesign for larger frame dimensions.

Additionally, statistics regarding the utility of resources, such as memory, BRAM, LUTs, and DSPs, are given in the table below:

Resource	Amount/ Number Used	Percentage
Slice LUTs	72301	26.06%
Slice Registers	60028	10.82%
Block RAM Tiles	24	3.2%
DSPs	29	1.44%

The utility statistics show that the Mini-ITX100 platform could easily accommodate expanded versions of each of these modules or additional features to the system, given that approximately a third of its maximum capacity has been reached with our current synthesis. For example, if the throughput considerations can be addressed, this FPGA platform would be capable of handling input data from a QVGA ToF camera (320 x 240 pixels instead of 80 x 60) as well.

5.2 System Response

In addition to testing each of the modules individually against known input and output pairs, the system's behavior was tested hollistically. A few strategic inputs were recorded, each of which targeted one subsystem at a time.

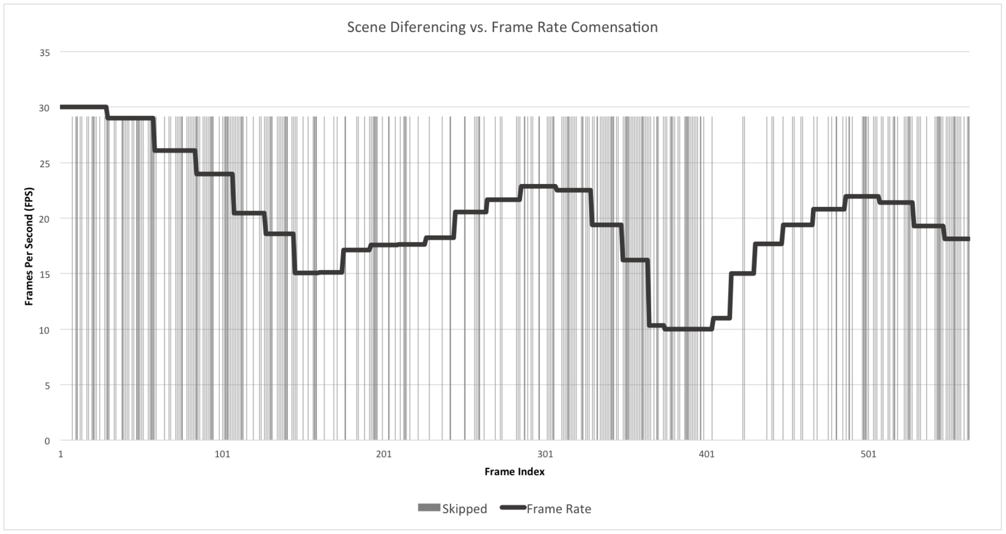


Figure 13: System response to intervals of camera motion and stillness.

In the first test case, the camera alternates between being held still and being shaken rapidly at 5 second intervals for a total of approximately 25 seconds. The frame rate response is plotted against the detected skipped frames in Figure 13, confirming the expected results. Dense periods of vertical lines indicates that many frames were skipped. The scene is changing slower than the camera is sampling, and so the frame rate decreases to compensate for this condition. The camera controller works to maintain a constant frame rate. So when the density of the lines decreases – which indicates the scene is moving faster than the camera is sampling – the frame rate increases to compensate for the added motion.

The same camera input data used in the previous test case is coupled with a periodic input from the IMU, emulating a constant step rate. Recall that the camera controller compensates the frame rate proportional to the number of skipped frames, and the step rate influences the gain of this feedback loop. We can clearly observe in Figure 14 that when a non-zero step rate is imposed, the results are similar to the previous result but the response is much faster.

In the third test case, the camera begins close to a blank wall and moves away in constant-distance increments at 5 seconds intervals. The camera distance input is plotted against the illumination response in Figure 15. The illumination generally varies as expected, increasing brightness as the distance to the subject increases. Notice the glitch at time step $t = 0$ when the camera begins with full illumination but

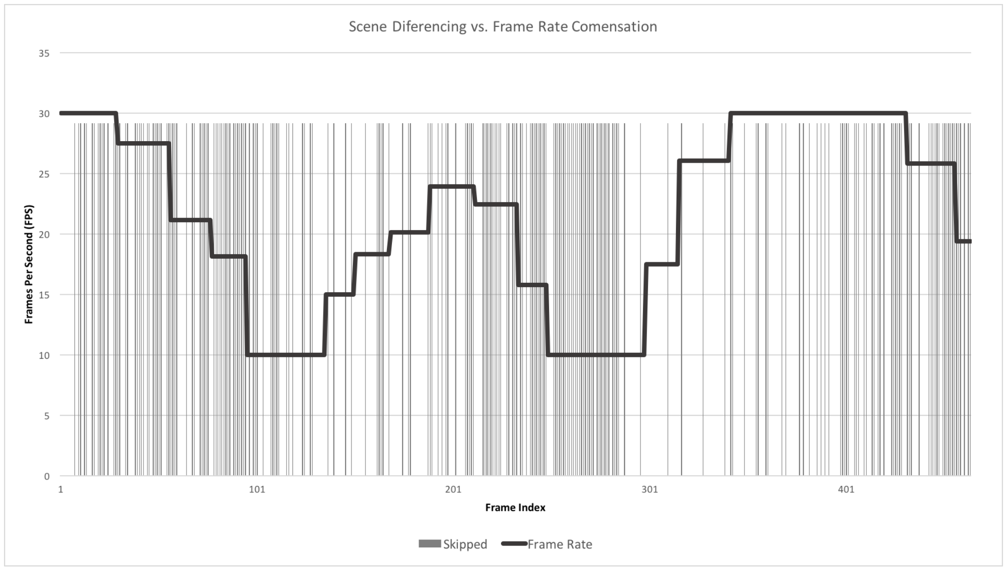


Figure 14: System response to the same intervals of camera motion and stillness. A constant step rate is emulated on the IMU, which increases the gain of frame rate responses.

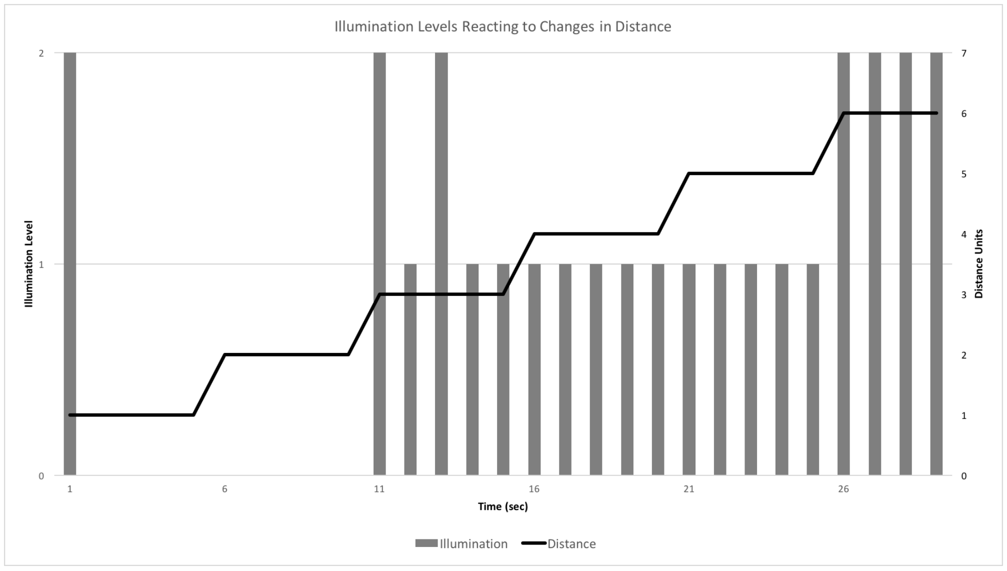


Figure 15: The illumination responds to the changing distance to the subject.

compensates downward by the next time step due to the close proximity. Glitches at $t = 11$ and $t = 13$ are suspected to be due to noise in the distance transition.

5.3 Challenges

After the algorithms and architectures had been designed, there were several implementation challenges that we encountered that forced us to consider major redesigns or caused unexpected delays in our timeline. The earliest discovery was that the class's circular pipeline FFT implementation exceeded the number of DSPs provided by the Zedboard (which was the platform we originally intended to work on) for a 128 point operation. Given that we required an FFT that large for the application, a superfolded FFT that used only a single Butterfly node was implemented as a replacement.

Additionally, performing arithmetic in hardware proved to be a challenge in a few different instances. The geometric transform required to convert a phase pixel to X, Y, and Z coordinates involved trigonometric operations, square roots, etc, and we had expected to have been able to perform these operations in Bluespec using libraries or Xilinx cores. However, the conversion factors had to instead be manually generated in stored in lookup tables, which forced us to incorporate a BRAM in the point cloud design.

Similarly, the scene statistics module relied on many division operations to histogram quickly (instead of a searching and sorting mechanism). When the synthesis for this module was first performed, it was discovered that the division operator caused an extremely long critical path length as it could not be treated as a multicycle operation. It took several days for us to trace this issue, restructure our implementation to allow for multicycle divides, and integrate the Xilinx division core with the rest of our system for another synthesis run.

There were many other challenges that resulted from attempting to assemble the system end-to-end. For example, it was quite difficult to work with Connectal to setup the framework and to implement even a simple control algorithm in C++, with which we did not have much experience. Capturing and using data from the camera for the test benches was not straightforward and required intermediary conversions to and from fixed point binary representations. Simulating the camera data on board the ARM core when the SDK compile failed, matching data types between C++ and Bluespec, and reading and writing files from the ARM core were all seemingly trivial pieces of the puzzle that ultimately took more time than intended as we worked towards completion of the project.

6 Design Exploration

There are several possible extensions or design explorations that could be considered, now that the system has been completed. One very important step would be to consider expanding this implementation to work with phase and confidence frames of any dimension. For example, all of the Bluespec modules would need to parametrized to be independent of dimensionality, which would require a significant refactoring of the code. Additionally, because the number of cycles needed to generate an output is proportional to the number of points in a frame for several of the modules, a decrease in throughput might be a constraining factor for this system to work with arbitrarily large frames sizes. Lastly, for the modules that require BRAM for the storage of frames or lookup tables for computation, larger frame sizes will cause an increase in required space.

Another smaller extension might entail refactoring the implementation such that it does not require any processing to be done on entire frames of data. This would primarily be the case for the scene statistics architecture, that requires a complete histogram to have been formed before votes can be assigned to the largest bins for the scaling of illumination. Ultimately, this would likely require a redesign in the scene statistics algorithm that would trade-off search completeness for efficiency and speed.

Finally, from a system completeness perspective, the current implementation could be advanced to incorporate the camera control loop into the hardware architecture, once camera frames could directly be clocked off of the chipset to the FPGA instead of through a USB interface, as it is being done currently for simplicity.

7 Acknowledgements

We would like to warmly thank our project mentors, Jamey Hicks and John Ankcorn, for offerring us their patience, wisdom, and expertise throughout this project. We would like to express our gratitude toward Professor Arvind, who has taught us a skill set that will be useful well beyond the end date of this course. And, of course, a special thanks to our TA Ming Liu for his kindness and diligence.