Image Processing Vision System for Manned and Unmanned Aircraft

CS 463 - Senior Capstone Final Report

June 10th, 2016

Group 4 - HawkEye Crew

Hailey Palmiter

Scott Griffy

Ryan Kitchen

Abstract

This document is a final report constructed using the IEEEtran style guidelines. The report provides an introduction to our senior project, a project description, and a personal overview from each member of the team about what they learned over the course of the project. The report includes our original requirements document, design document, and technological review with an added discussion about each document. This discussion describes how the project had changed over the course of the year in comparison to how it was perceived during the planning process. Also included is our weekly blog posts that had been generated over the course of the year to keep our TA and our sponsors up to date with our progress, and our final poster that was used at the Engineering Expo. Appendix 1 covers essential code listings and Appendix 2 displays the hardware we were able to interact with and a few output images from our vision system.

Contents

I. Introduction

Rockwell Collins, our project sponsor, designs video vision systems for pilots to use during flight. Pilots often use these enhanced image systems to help them see better in rough weather conditions, and to generally assist during different flight operations, such as landing. An example of this functionality could be overlaying a pilot's view with graphics to help the pilot locate a landing strip in a storm, or automatically turning a night-vision camera off when it is not needed to save power. The hardware used for this system must limit the power consumption it pulls from the aircraft, and also must be lightweight in order to have the least affect on the airplane. Even an extra five pounds added to the plane during a year can add up to thousands of dollars in fuel costs. In order to provide pilots with this specific low power, low weight, and enhanced image processing, Rockwell Collins develops software on Field Programmable Gate Arrays (FPGAs). This piece of hardware makes the code very complex and very costly to develop. New vision enhancements can take weeks or months to develop on the FPGAs. The FPGAs are currently Rockwell Collins only option that meets the requirements needed to create practical systems that pilots can use effectively.

Our goal is to provide a proof of concept for an alternative to Rockwell Collins' FPGAs. It must meet the performance metrics, provide faster implementation time, and reduce the cost of production. Specifically, we are designing a proof of concept using single board computers (SBCs). Single board computers differ from FPGAs because they have a standardized execution environment, which allows simplified code to be executed on it, reducing the development time. Single board computers also use low cost hardware and don't consume much power. If we can prove that the video quality produced by the SBCs is adequate for pilots to use, they will meet all the requirements needed to develop a practical vision system on. In the air, FPGAs often take feeds from multiple cameras and run a lot of processing algorithms on those images. In order to be effective, the SBCs should also be able to handle this operation. Our project aims to test and measure the capabilities of a single board computer by delivering a multiple-stream video display that has been processed to a high degree. We believe the best candidate for this is to use NVIDIA's single board computer, the Jetson TX1. Our goal is to fully test the Jetson TX1's ability to provide enhanced imaging. This proof of concept will result in measurements that will help Rockwell Collins determine the practicality of using single board computers for their vision systems.

Our clients are Carlo Tiana and Weston Lahr through Rockwell Collins. The members of te HawkEyed Crew are Scott Griffy, Hailey Palmiter, and Ryan Kitchen. Ryan Kitchen worked on the technical side of the project. Scott helped with some of the technical side and wrote documentation. Hailey edited and wrote documentation and managed the project. We communicated with the clients to define requirements and secure hardware necessary for the project. We also got a lot of the hardware from our instructor, Kevin McGrath.

II. Original Requirements Document

A. Introduction

1) Purpose: Rockwell Collins currently develops their vision systems on Field Programmable Gate Arrays, making the time from design to product very long. Rockwell Collins is considering using Single Board Computers as a way to speed up this cycle. Our goal is to provide a proof of concept for a vision system on a Single Board Computer for Rockwell Collins. This proof of concept should help Rockwell Collins measure the practicality of Single Board Computers for vision systems.

2) Scope: Development will be limited to the Jetson TK1 or Jetson TX1. Evaluation of hardware platforms other than the TK1 and TX1 will be a stretch goal. In which case, these boards will serve as the base unit for comparison. Software shall interface to a minimum of 2 cameras, receive video, and display it on a monitor while logging metrics. Metrics will be frames per second and operations per second.

3) Definitions:

SBC	Single board computer
RC	Rockwell Collins
FPS	Frames per second
FPGA	Field-programmable gate array
USB	Universal serial bus
PoC	Proof of concept
SVS	Simple vision system

4) References: Carlo Tiana, Airborne Vision Systems Expert at Rockwell Collins Weston Lahr, Senior Software Engineer at Rockwell Collins, Jetson TK1 Embedded Hardware, NVIDIA

5) Assumptions and Constraints: We're assuming that either Jetson model can handle some sort of camera vision system. From initial investigations it appears that the Jetson TX1 is a likely candidate to support our project goals.

B. Overall Description

This section will give a bird's eye view of the project, outlining the major features and their purpose.

1) Product Perspective:

a) System Interfaces: Linux, Jetson Distro

b) User Interfaces: While using our deliverable, the user should be able to view output image from the Jetson.

c) Software Interfaces: Our deliverable should be a demonstrate of the Jetson's video output, charts that display capability performance, information about the complexity and amount of video processing operations. A stretch goal would include to have the same output of the software running on a second screen.

d) Hardware Interfaces: We must choose an interface between our program and the chosen camera attached to the Jetson SBC. We must also interface with the relative camera processing systems on the Jetson.

e) Communication Interfaces: We will communicate with at least two different camera streams, from two separate cameras attached to our SBC.

2) Product Functions: The delivered demonstration should accurately demonstrate the performance limitations of the Jetson TK1 or TX1. Should the timeframe allow, we are to continue investigating other SBCs and providing a similar overview of other researched SBCs.

3) User Characteristics: Our users will be RC developers, looking to implement a given filter for vision processing platforms.

4) Constraints: Our deliverable must run on either the Jetson TK1 or Jetson TX1, and should simply test the Jetson?s capabilities. We will use 1080p cameras.

5) Assumptions and Dependencies: We are assuming that the Jetson can at least handle some level of video processing.We are also going to rely on libraries and the operating system provided by Nvidia for the Jetson.

6) Stretch Goal Timeline: Stretch goals defined throughout this document may be delayed until all other requirements are finished and are not required to be in the deliverable at any point.

C. Specific Requirements

This section will outline some of the more specific parts of the project along with exact requirements that the deliverable should meet.

1) External Interfaces:

Two 1080p Cameras These will be used to transfer video data to the Jetson to be processed and displayed on a monitor.

Monitor Any monitor able to adequately display the output of the chosen SBC for the purposes of evaluation.

Jetson Should be able to read and process video data and stream it out to a monitor.

2) Functions: The system shall take input from the cameras and perform as much video processing on it as possible before dipping under the 30 fps minimum. If the system can't handle 30fps without any extra video processing, it should simply be displayed at the fastest fps possible.

3) Performance Requirements: The system should support at least 2 cameras with a stretch goal of 3 cameras. The system should support video output on one monitor, and if possible the video should be displayed at 30fps.

4) Design Constraints: Since this is a proof of concept, the only real design constraint is that it should be able to correctly assess and demonstrate the hardware limitations of the Jetson SBC.

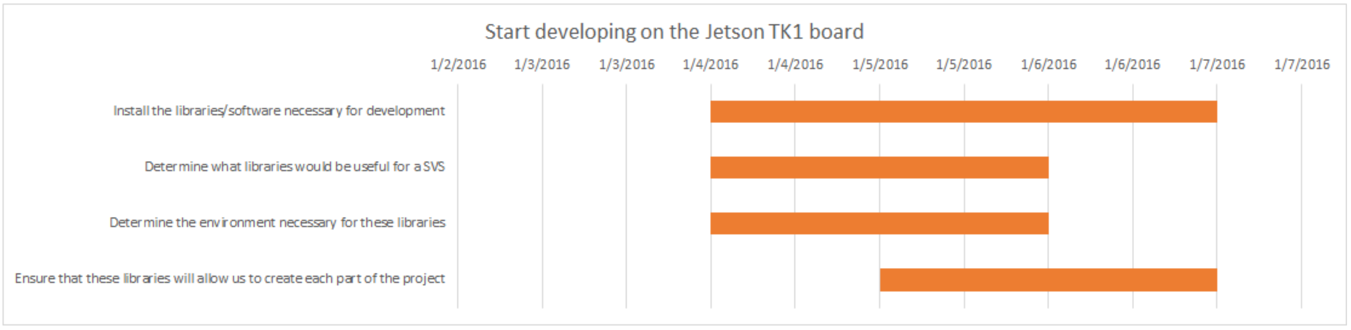
5) Maintainability: Maintainability, outside of our timeframe, is a low-priority requirement as we are only delivering a proof of concept for the capabilities of our SBC.

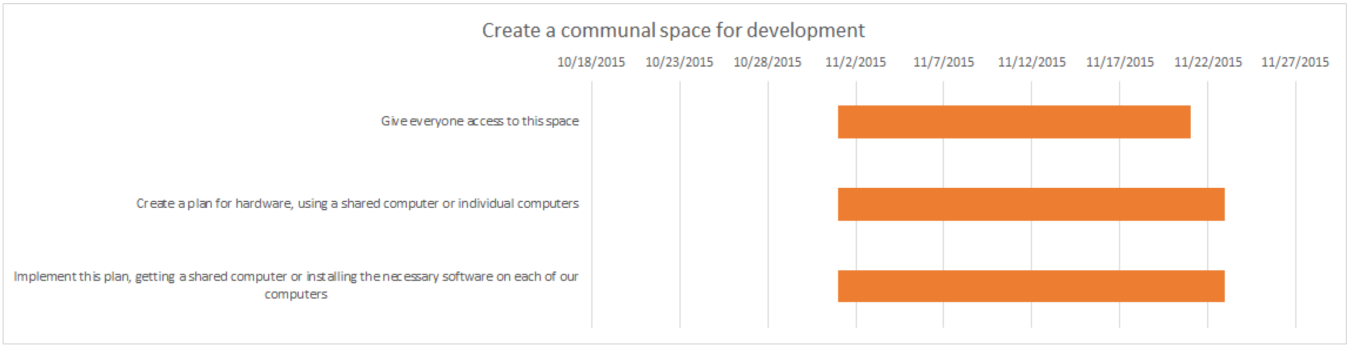
6) Portability: Our software should give some idea of how similar code would perform on other platforms, even though the PoC itself doesn?t have to be portable. The same goes for the camera and monitor.In order to help our stretch goal of evaluating a second SBC, it would help if the code, cameras, and monitor were all collectively portable to the second platform.

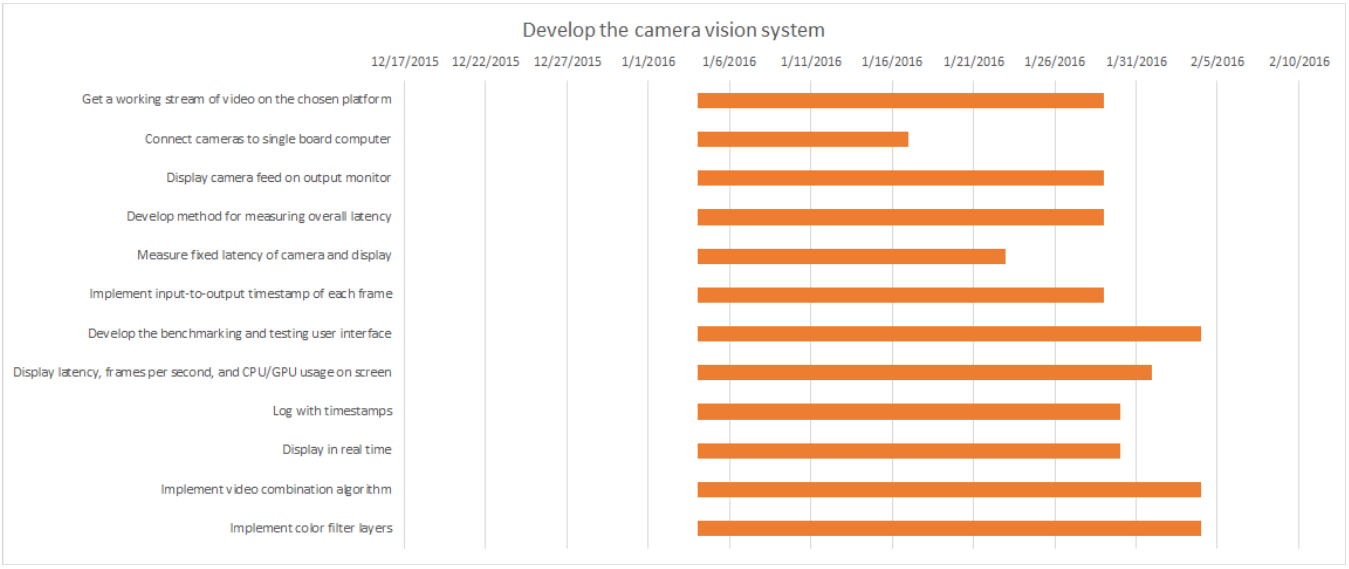
D. Stretch Goals

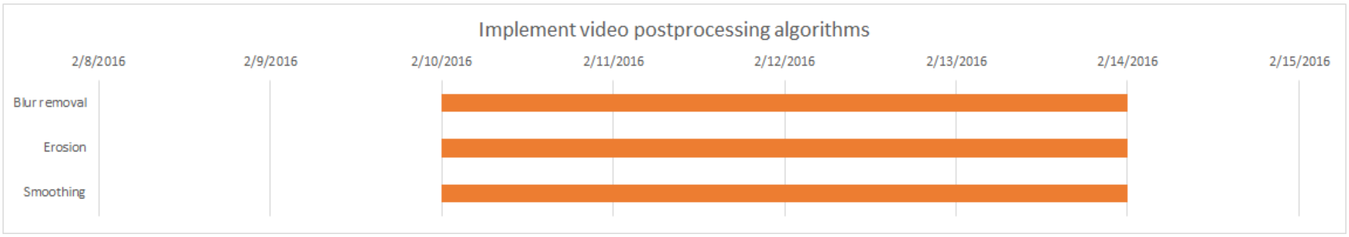
Some stretch goals were listed throughout the requirements. One purpose of the stretch goals is to further increase the effectiveness of the proof of concept. These added goals further stress the limitations of the Jetson TK1 or TX1, to further expand the capabilities of these SBC. A few other stretch goals not mention above would be implementing the same system on a similar SBC, and comparing that with the Jetson TK1 or TX1, or implementing object tracking as a video processing operation to test performance of the SBCs.

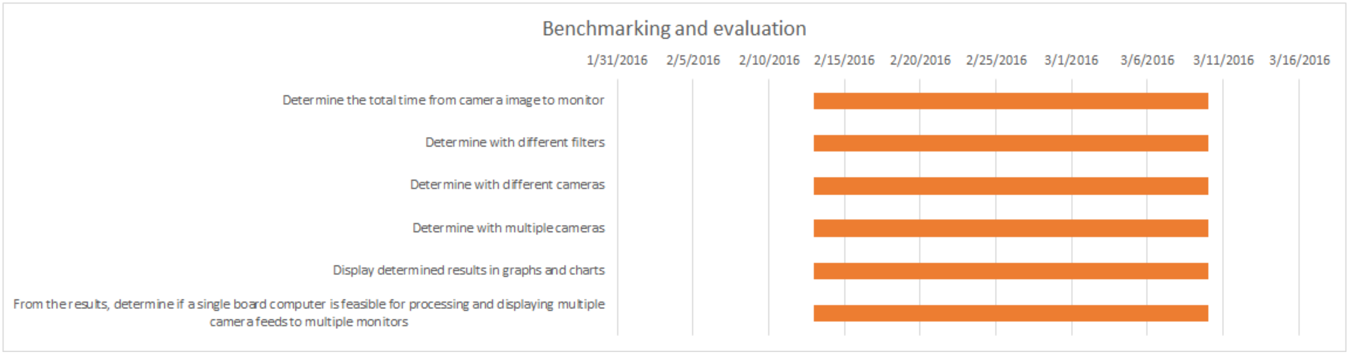
E. Gantt Chart

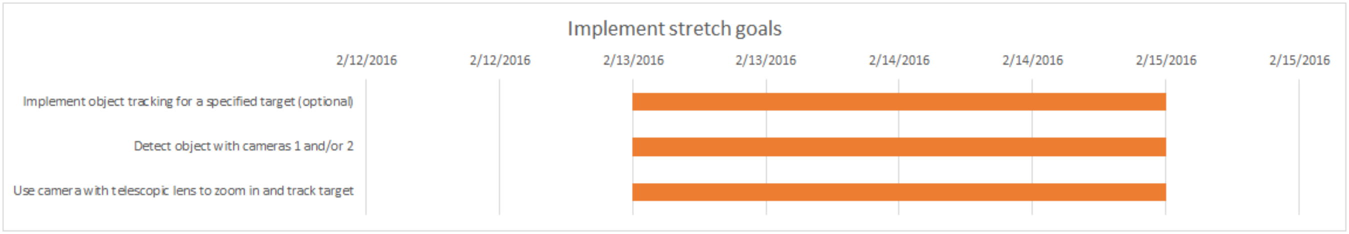












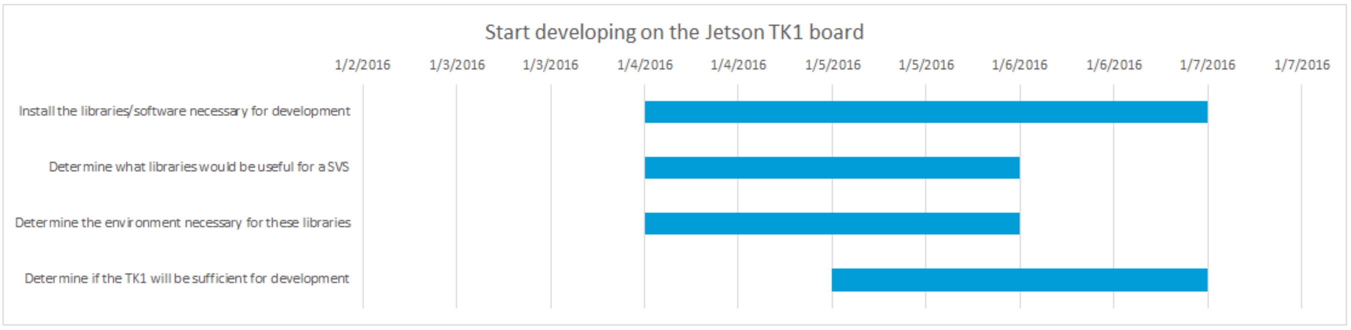
III. Requirements Document Changes

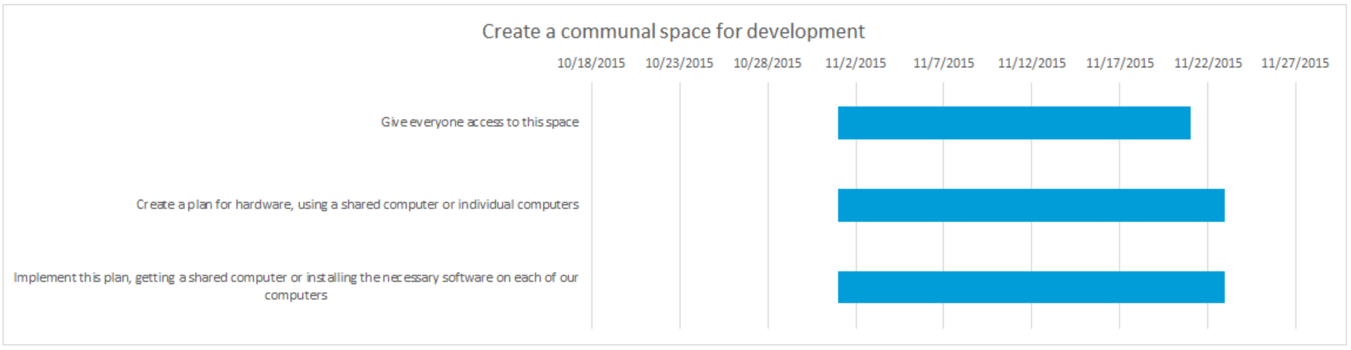
Before real development began, we had reread our requirements document and noticed that some changes and additions needed to be made. The revision was needed in order to refine our project with the discovery of problems/improvements we had found. It was a needed process as we wanted to have full clarity on what needed to be accomplished with this project, and be able to display those requirements with our sponsor. The changes were reviewed with our sponsor and agreed upon before any final changes were made.

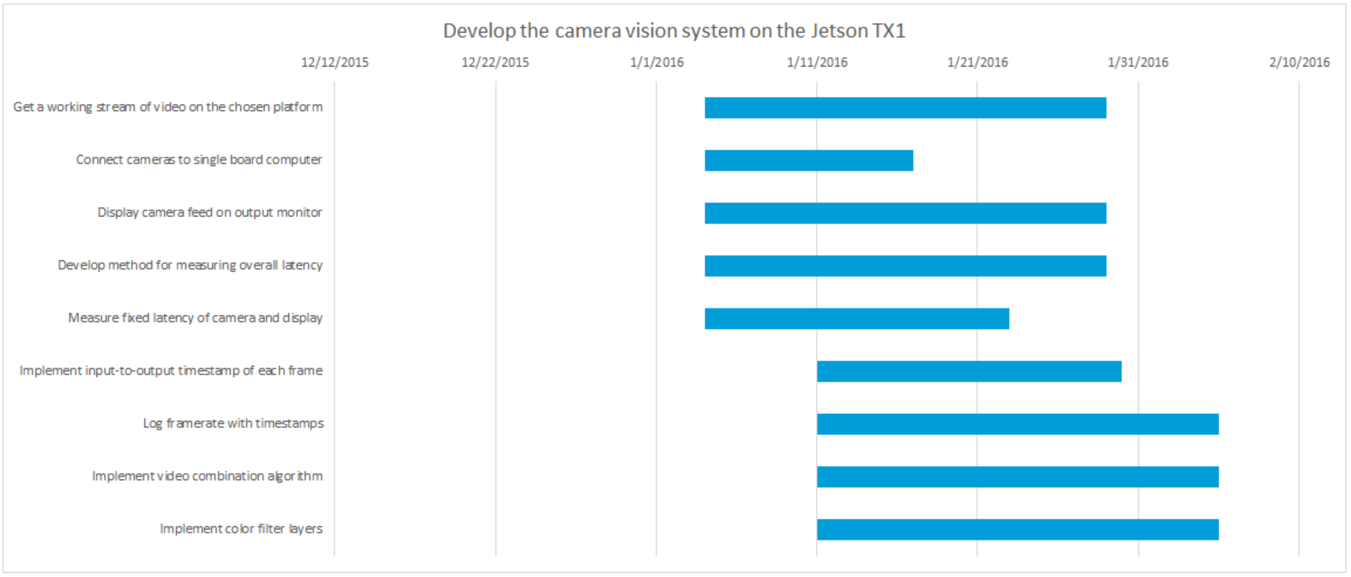
Only a few minor things were changed, but they had a big impact on how we define our deliverable according to their standard. Our client was very happy to see the revisions we had made and all final revisions were made in conjunction with our document. The table below shows the minor changes we had made.

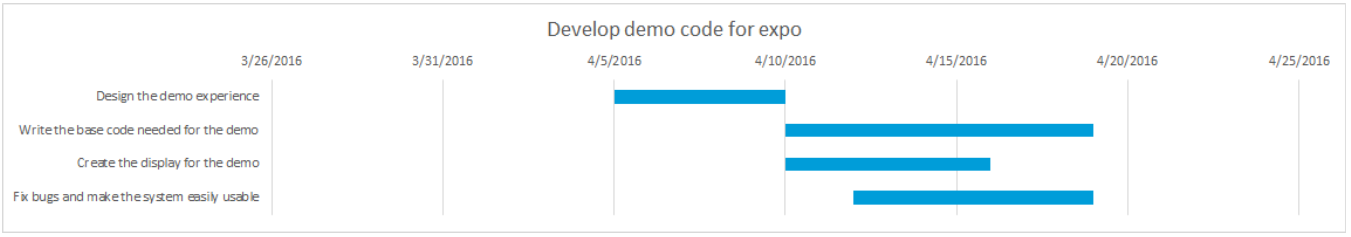
Requirements Document Changes			
1	Two Working Cameras	Added Requirement	In our original document we had not specified more than one camera, but our sponsor specifically wanted to make that a requirement as the system must be able to handle more than one camera. Therefore it was added into our requirements document.
2	Use of Either Jetson TK1 or TX1	Redefined Requirement	Before the revision, the requirements document was specifying the use of only the Jetson TK1. After much research on our end we had decided that either Jetson's TK1 or TX1 would be adequate for development, and actually plan to rely more heavily on the Jetson TX1. This change allows us to have more flexibility between the two Jetson SBC's, which provides protection if we cannot get one to work properly or to the meet the requirements needed.

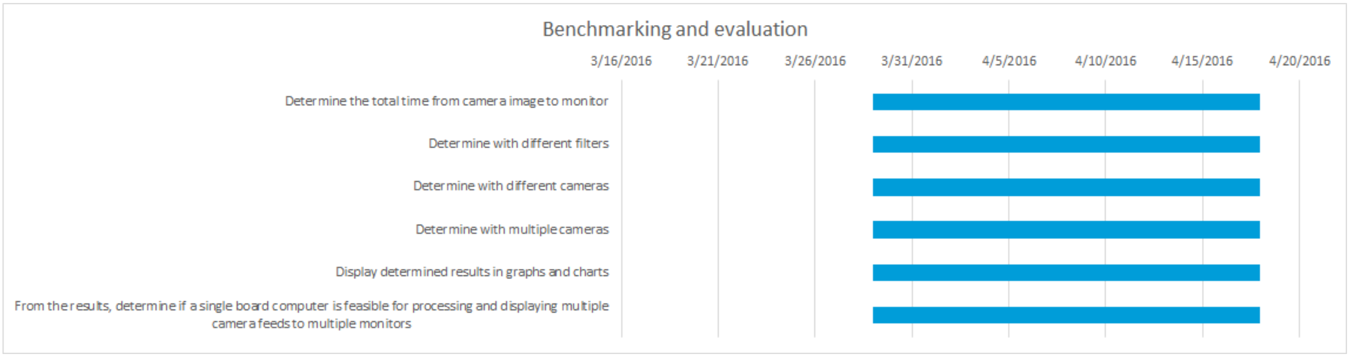
A. Final Gantt Chart

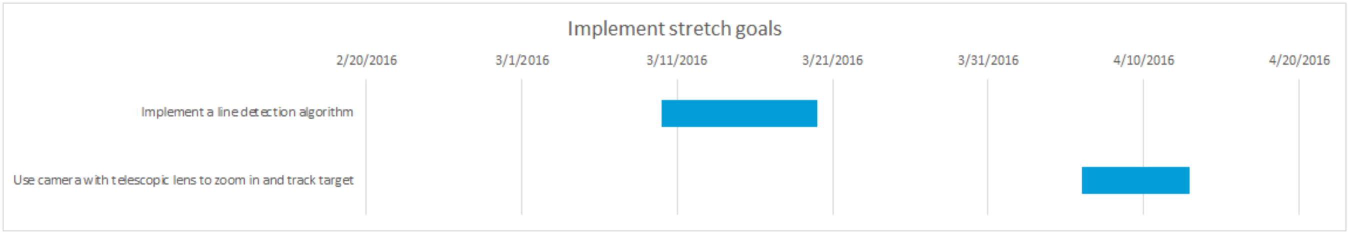












IV. Design Document

A. Introduction

1) Scope: This software will implement a framework to develop, test, and benchmark video processing algorithms. The users will be able to test different combinations of algorithms using one or multiple camera inputs and produce output to one or multiple display windows. This software will be used by HawkEye Crew to determine if the Nvidia Jetson TK1 or other off the shelf single board computers have high enough performance to replace FPGA-based video processing systems currently used by Rockwell Collins.

2) Purpose: This software description document will provide HawkEye Crew with a road map to complete development of the software and fulfill the requirements in our Software Requirements Specification. This document explains how the system is going to work, who is going to use it, and how it is meant to be used.

3) Intended Audience: The intended audience of this design document is the developers who will design the system *us* and the sponsors of this project at Rockwell Collins.

4) References: IEEE. IEEE Std 1016-2009 IEEE Standard for Information Technology System Design Software Design Descriptions. IEEE Computer Society, 2009

B. Definitions

SBC	Single board computer
RC	Rockwell Collins
FPS	Frames per second
FPGA	Field-programmable gate array

USB Universal serial bus

PoC Proof of concept

UML Unified Modeling Language

DFD Data Flow Diagram: Shows how data moves between different components in the system

ER Diagram Entity Relationship Diagram: Shows how different data structures within the software are connected to each other

SDD Software Design Description

SRS Software Requirements Specification

SVS Simple Vision System

YUV A color space typically used as part of a color image pipeline

JSON JavaScript Object Notation

Modular Video Processing System The HawkEye Video Processing System's internal video processing algorithm management and execution system.

C. Conceptual Model for Software Description

1) Software Design in Context: For the HawkEye Video Processing System, we will be using a functional design method. Our system is designed to be modular and have standardized interfaces, so we will be able to develop new features and drop them in without modifying existing code. The reasoning for using a functional design rather than object oriented is because our software is focused on processing individual video frames in real time. Each algorithm instance will run once and has no state, only input and output data. For that reason functional programming is the best choice for this design.

2) Software Design Descriptions within the Life Cycle: This document details how we are going to the implement software to meet our requirements. During the course of development, our requirements and our SRS may be updated. This document includes support for both our current requirements and provides room to add additional requirements. It also adds additional operational requirements and functionality beyond those specified in the requirements document. This document will also provide a reference for the creation of our testing plan.

D. Design Architecture

1) Stakeholder Concerns and Requirements: Our client Rockwell Collins would like to know if single board computers are a viable option to implement simple vision systems on. They've enlisted us, the HawkEyeCrew, to implement a proof of concept to measure the effectiveness of single board computers in this context.

We, the HawkEye Crew, will be using this software to test various implementations of video processing algorithms. In order for our system to be a valid proof of concept, our system will need to meet certain performance benchmarks and functional requirements described by our SRS. Our design concerns will be meeting these requirements.

2) Description of Architectural Design: Our system is designed to be a framework and testbed for the implementation of video processing algorithms. This design handles the hardware input interface, video output to the monitor, data flow between different algorithms, parallel operation of multiple operations, and benchmarking of individual components and overall aggregate timing.

In order to provide maximum flexibility and testing capabilities, the video processing algorithms and input/output devices are built as separate, interchangeable modules. These different modules are organized by the user in a configuration file, which determines both how the modules connect to each other and how the log file is formatted. This enables us to track the performance of individual algorithms and maximize our CPU and GPU usage while maintaining the performance requirements for this project.

3) Validation of Design: The core requirements of our system that this design either implements or implements the capability of testing are as follows:

1) System must be capable of processing multiple video streams
2) System must enable thorough testing of the Jeston's video processing capabilities
3) Frame rate must be at least 30FPS
4) End to end latency must be less than 100ms

These requirements are met by the following architectural features:

1) The modular video processing system enables multiple input and output devices.
2) The modular video processing system will enable us to maximize the usage of CPU and GPU processing power, which will show the maximum performance of the Jetson.
3) This software design includes built in speed benchmarking, so we will be able to track FPS.
4) This benchmarking system also produces latency details, so we will be able to determine if this system is capable of meeting the latency requirements Rockwell Collins is looking for.

4) Overview of Viewpoints and Design Languages: These viewpoints have been chosen to provide a complete description of the design and show how the design is compliant with the SRS. Each one provides details crucial to understanding how the design works and is a reference for implementation.

1) **Context viewpoint:** This viewpoint shows the different potential users of the software and how they would interact with our system.
   Design Languages: Use Case Diagram
2) **Structure viewpoint:** This viewpoint shows how the streaming video flows through the system, and identifies the internal and external data connection points in the system. This viewpoint also shows the components of the system which enable connection of cameras via USB as per the SRS.
   Design Languages: Data Flow Diagram

3) **Interaction viewpoint:** This viewpoint shows the order of operations on processing a video frame, as well as how the timing is integrated to meet the performance benchmarking requirement.
   Design Languages: UML Sequence Diagram
4) **Information viewpoint:** This viewpoint details how the modular video processing system determines the order of operations for the various elements and algorithms it can create. It shows how the system meets the requirement for modularity and the capability to use multiple input and output devices.
   Design Languages: Entity Relationship Diagram
5) **State Dynamics Viewpoint:** While the algorithm implementations in our design are stateless, the system itself is not. This viewpoint shows the transitions between different states and provides a road map for different states which will need to be individually tested.
   Design Languages: UML State Transition Diagram

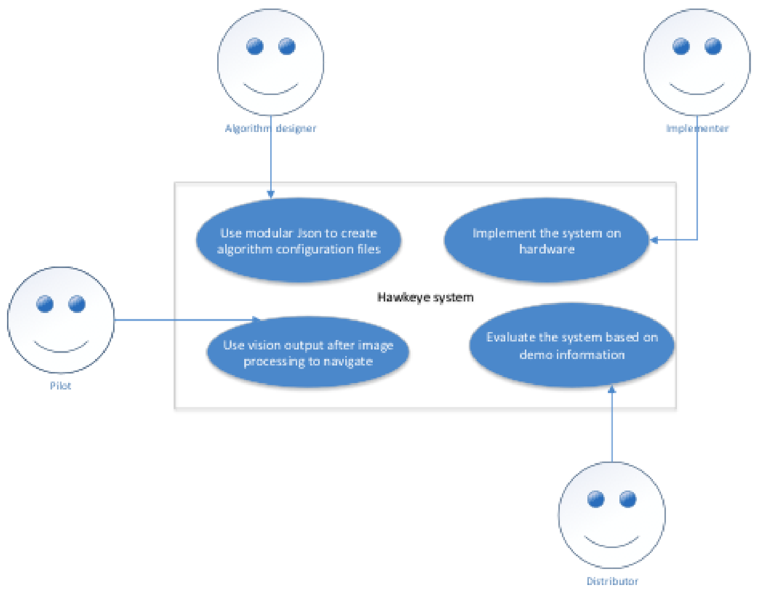
E. Design Viewpoints

1) Context Viewpoint:

a) Users and Design Concerns: The design features in this document are chosen to create value for several different kinds of users and stakeholders. These users interact with the software in different ways.

1) **Pilot:** Our intended user will be a pilot of a manned or unmanned aerial vehicle. This software is designed to provide increased and extensible functionality for onboard camera systems. Because our product is a proof of concept, there won't be a real pilot, but the pilot has been included as a user because the application of this design is to create better equipment for pilot use. This pilot will be 'using' our product and needs the video stream to meet the requirements outlined in the SRS.
2) **Distributor:** Rockwell Collins is the sponsor and primary stakeholder in this project. They will evaluate the product and decide where to use it. They will also decide the feasibility of using this design in production and whether continuing to use single board computers for real-time video processing is a good idea.
3) **Implementer:** The implementer has to set up the simple vision system on their given hardware. Our proof of concept should give some notion as to how to do this and also the feasibility and easiness of it. The concepts demonstrated in this design document can be used on different hardware, not just the NVIDIA Jetson.
4) **Algorithm Designer:** The algorithm designer interacts with our software in different ways than the end user and implementer. This user will create custom modules and link them together with our easy to use configuration system. This configuration should serve as a design for other systems.

b) Use Case Diagram:


Algorithm designer
Implementer
Use modular Json to create algorithm configuration files
Implement the system on hardware
Hawkeye system
Pilot
Use vision output after image processing to navigate
Evaluate the system based on demo information
Distributor

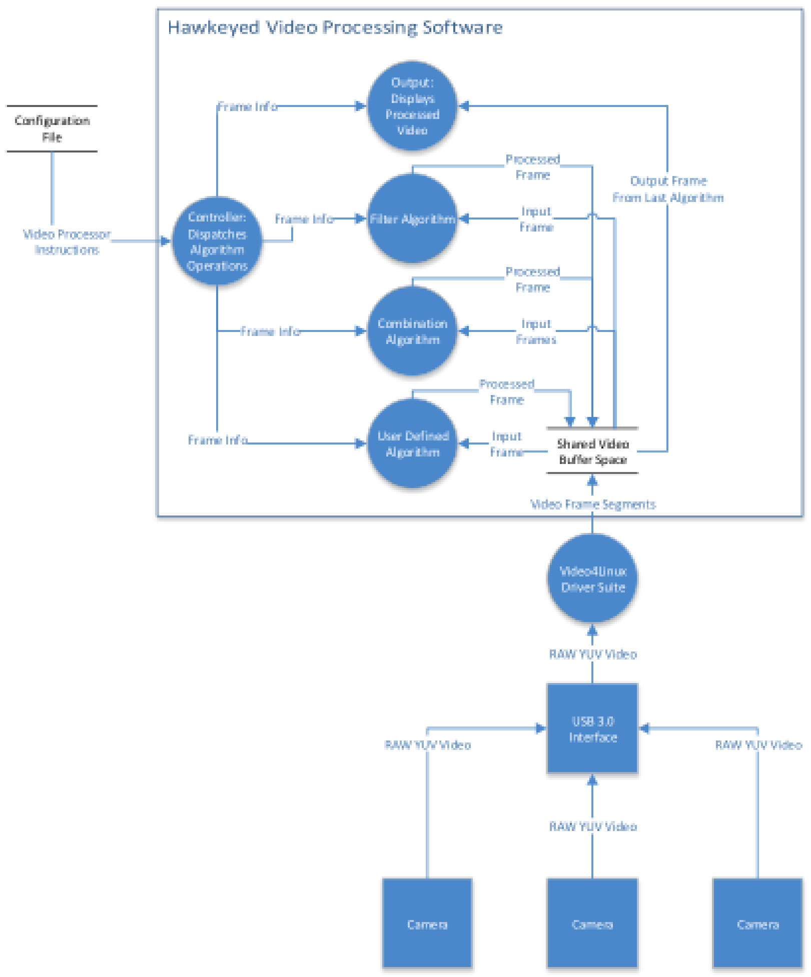
2) Structure Viewpoint: This section shows the high level organization of our software design.

a) Design Concerns: The primary design concern of this section is to show the interaction between critical components of our software design. This is valuable during implementation because it provides a reference for the required parts of the system, and also shows how the different pieces fit together.

b) Design Elements: There are three key design elements in our software design. These elements are important because they are the basis for the entire modular video processing system.

1) **Video Buffer:** Shared data space provided by video4linux or created internally. Video data is in YUV format and access is provided through a void pointer to a contiguous block of memory.
2) **Algorithm Module:** Algorithm modules are single purpose elements that operate on one or more input buffers and provide output to one or more output buffers. They consist of a C function which performs the operations, and a module definition structure which provides the name and expected input and outputs of a module.
3) **Algorithm Controller:** This element is responsible for ordering and executing Algorithm Modules. It is the primary routine in this software design. It reads from a JSON configuration file, creates a module execution tree, and manages the execution of concurrently running modules.

c) Structure Description: The HawkEye video processing software receives frames from the Video4Linux driver as a pointer to a buffer. The Algorithm Controller invokes each video processing algorithm in the order described by the configuration file, by passing them pointers to the buffers assigned to them. The video processing algorithms perform actions directly on the video buffer in order to avoid the performance hit from accessing additional memory. When the last video processing algorithm is complete, the buffer is flushed to the output device.



d) Data Flow Diagram:

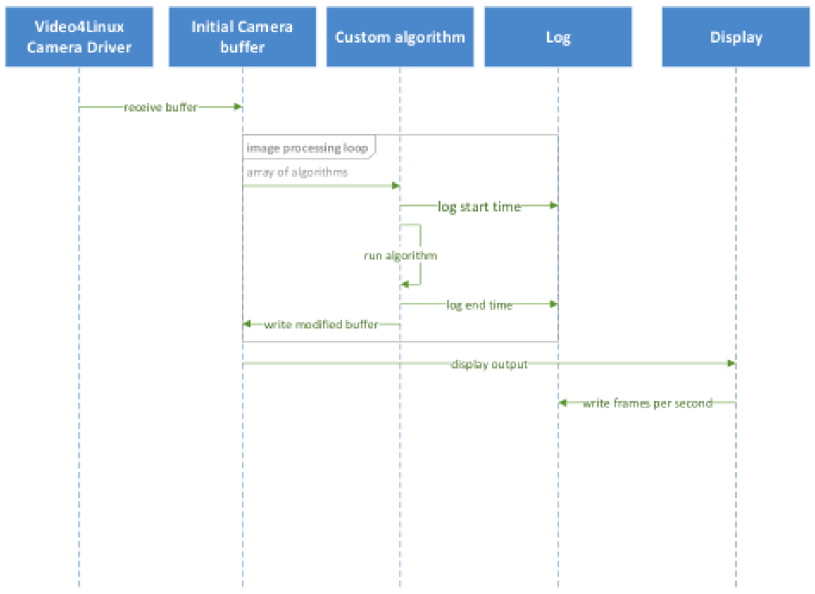
3) Interaction Viewpoint:

a) Design Concerns: The main goal of this project is to determine how much video processing can be done on a SBC while maintaining a frame rate of 30 FPS and latency below 100ms. To this end, we as the user have two design concerns that need to be addressed.

1) **Latency measurement:** In order to keep overall latency under 100ms while maximizing the amount of processing being done, we need to know how long each algorithm takes to process a single frame, and also how long it takes from input to output.
2) **FPS Monitoring:** Our software must process video in real time. For the purposes of this project, that has been specified as 30 FPS by our SRS. This is separate from the latency requirement because our system may actually be operating on multiple frames simultaneously, so the frames per second may not necessarily dictate the total latency.

b) Design Elements: This viewpoint contains several design elements that depict major components of our software design.

1) **Camera Buffer:** Initially, camera buffer will store the data immediately loaded by the camera and will then be repurposed to hold the modified image later. This modified image will come from the custom algorithm code.
2) **Custom Algorithm:** The custom algorithm is loaded into the system when it starts up through the JSON configuration files and dynamically linked libraries. This portion of the system will perform operations on the camera buffer and log the time at start and time at completion. It will then write back to the camera buffer, the modified image.
3) **Log:** The log is a feed that will keep track of the frame rate and information about what processes are running. This should be viewable separately from the display.
4) **Display:** The last thing that should happen when we are rendering frames is the stream to the display. This will show the final, completely processed image. Our client has contacted us and told us that the display should only show the video stream, not other metrics like frames per second or operations per frame.
5) **Video4Linux Camera Driver:** This is our direct access point to the video stream from the USB camera. It enables us to receive video frames from any cameras supported by the Linux operating system.



c) UML Sequence Diagram:

4) Information Viewpoint:

a) Design Concerns:

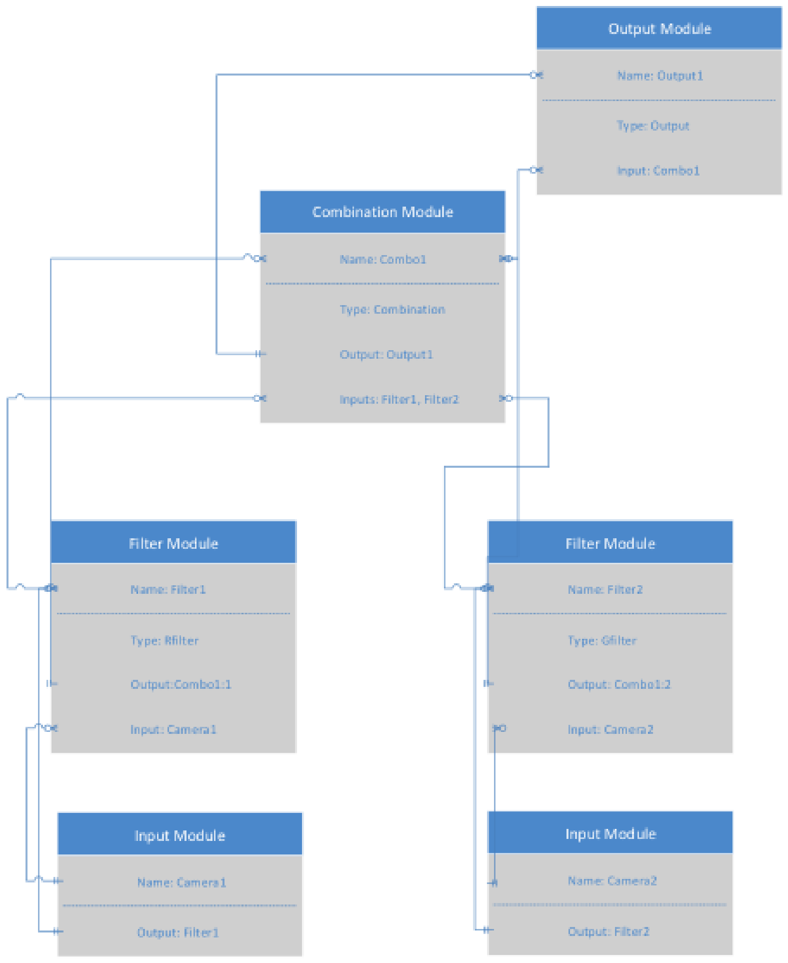
1) **Configuration:** This viewpoint addresses how the user will configure our software. It shows how the modules are connected to each other and the information that needs to be provided for the software to create its internal structure.

2) **Internal Data Structure:** In order to implement the modular image processing system, our design needs an internal data structure in order to determine the execution path through the various modules.

b) Design Elements: There are two design elements in this viewpoint:

1) **Module Instances:** Module instances are C structures that describe either input device modules, output device modules, or algorithm execution modules. They consist of a name, type, and the names of the connected input and output modules.
2) **JSON Configuration File:** In order to create the internal data structure for our application, the user must write a configuration file in JSON format. The elements in this configuration file are detailed in an example below (Example JSON Configuration File).

c) Description and Rationale: The HawkEye video processing software needs to be able to handle various configurations of multiple camera inputs and produce multiple outputs, as well as performing multiple video processing algorithms. In order to do this, it needs to know what operations to perform in what order, and where to put the data. To do this, our software uses a tree-like data structure that connects the camera inputs, algorithm modules, and outputs. An input module definition specifies what modules a camera sends data to. Each of those modules then has its own output specification, which can either go to another algorithm module or an output module.



d) Sample Entity Relationship Diagram:

e) Example JSON Configuration File: The corresponding JSON configuration file for the above ER diagram.

```
{
   "Output1":{
            "type":"output",
            "input":"Combo1"
     },
     "Combo1":{
            "type":"Combination",
            "inputs":"Filter1,Filter2",
            "output":"Output1"
     },
    "Filter1":{
            "type":"Rfilter",
            "input":"Camera1",
            "output":"Combo1:1"
     },
     "Filter2":{
            "type":"Gfilter",
                "input":"Camera2",
                "output":"Combo1:2"
        },
        "Camera1":{
            "type":"Input",
                "device":"/dev/video0",
                "output":"Filter1"
        },
        "Camera2":{
                "type":"Input",
                "device":"/dev/video1",
                "output":"Filter2"
       },
        (not used in this example but also an option)
        "Tracker":{
                "type":"Module",
                "filename":"tracker.so"
       }
  }
```

The one rule for the JSON files is that each entity must be declared before its inputs; this top down approach enables the JSON parsing algorithm to generate the module tree in a single pass. The one exception to this rule is cyclical algorithms that operate on their own output, either directly or indirectly. Support for such algorithms may be implemented, but this is not part of the current project scope.

5) State Dynamics Viewpoint:

a) Design Concerns: Our program will be in multiple states while it is streaming and processing data from the camera to the display. In order to keep track of the states, we will have two state diagrams, one for the camera and a second for the software for processing.

b) Design Elements:

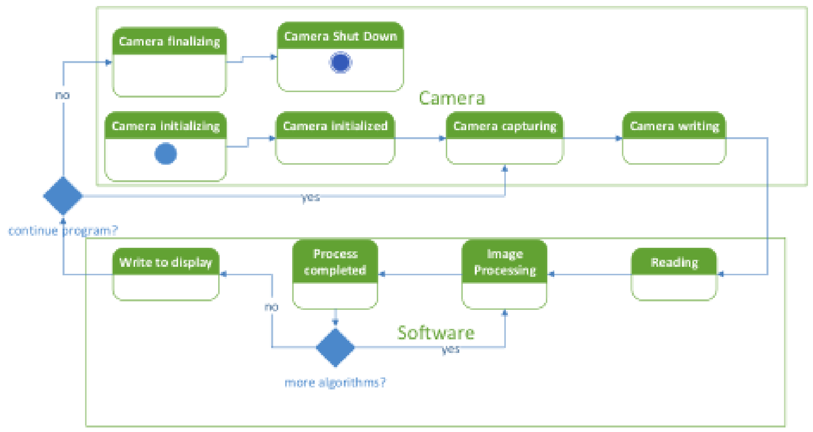
1) **Camera:** For the purposes of this viewpoint, Camera shall refer to the combination of the physical camera hardware and the system level driver that exists outside of the HawkEye video processing application.
2) **Software:** Refers to the HawkEye video processing application.

c) Overview: The camera must be initialized before we can capture frames from it. After initialization the camera is ready for image capture. During image capture, the camera will write to a buffer on the host system. Once a frame has been written to the buffer, the software can begin image processing using the first algorithm in its configuration. When the algorithm is completed, the software will check if there is another algorithm to be performed. These two states of image processing and checking for more algorithms will loop until all algorithms have been completed. We will then write the finished image buffer to the screen. After that, we will check if the program needs to continue running or if it has received a shutdown command. To shut down we will finalize the camera, deactivate it, and stay in that state.

d) States:

1) **Camera Initializing:** This is the starting state. Before we can capture frames from the camera, we must first initialize it. In this state we will also set up and synchronize the memory shared by the camera and our software.
2) **Camera Initialized:** After we have initialized the camera it will be ready for capture. The software will have to start the camera's frame capture.
3) **Camera Capturing:** The camera will take a moment to capture the data.
4) **Camera Writing:** The camera will have to write the data to memory.
5) **Image Reading:** Importing data from the camera's output into the software. This step should be simultaneous with camera writing as we are using the same memory.
6) **Image Processing:** The image data in the buffer will be processed in place by the modular video processing system.
7) **Process completed:** Once the algorithm is done operating, it will notify the controller that it has finished.
8) **Write to Display:** The software will write the finalized image buffer to the display, displaying the video.
9) **Camera Finalizing:** During this state, the camera is sending any remaining data and shutting down.
10) **Camera Shut Down:** The camera is turned off.

e) State Transition Diagram: Because the two state machines interact often, we have combined them in the same graphic. The following is our UML state machine:



V. Design Document Changes

A. *Overall Description Changes*

We ended up having a physical copy of the performance of the system at expo instead of working it into our software. This was because we were advised that using the software to measure it's own performance would lead to problems, leading us to measure the performance of the system using a high speed camera which can't be done automatically.

B. *Specific Requirements*

One of the goals of the original design document was to create requirements that were reasonably achievable. Because of this mindset, the requirements in the document did not change very much.

C. *Information Viewpoint*

We had originally intended to create a modular json system, but scrapped it later on because it wasn't very valuable and created unecessary complexity.

VI. Technology Review

A. *Camera Interface (Hardware Level)*

There are three different camera interfaces available on the Jetson development board. Each one offers a different set of benefits and challenges.

1) USB 3.0: USB 3.0 is a high-speed plug and play interface that offers a theoretical 384 MB/s data throughput. One USB 3.0 port is available on the Jetson, but additional ports can be added using the mini PCI Express expansion slot.

Benefits:

- Availability of USB 3.0 cameras
- Plug and play, with no camera configuration

Drawbacks:

- Jetson onboard USB 3.0 does not meet the USB 3.0 specification for bandwidth
- Some cameras do not have USB 3.0 drivers for ARM
- USB 3.0 only supports one camera per port due to bandwidth limits
- Increased cost due to board expansion

2) Gigabit Ethernet: The Jetson has one-gigabit Ethernet port available. Gigabit ports can handle 125 MB/s, however like USB 3.0 there are expansion cards available that add two additional ports.

Benefits:

- All gigabit cameras use a standardized interface and require no driver
- More high quality cameras available than the USB 3.0 offers
- Simple configuration with video4linux

Drawbacks:

- Most cameras require external power supplies
- Gigabit cameras typically cost more than the USB 3.0 alternatives

3) MIPI Camera Interface: The Tegra K1 processor on the Jetson has a direct camera interface, which is designed for onboard, low power camera modules. There are two MIPI interfaces, one 4 channel and one 1 channel interface, but any additional interfaces are unavailable.

Benefits:

- High speed, built in interface
- Low power consumption
- Lower latency because the video processing can be done before buffering

Drawbacks:

- Very limited number of cameras supported
- Specific to the Tegra K1, making it not portable
- Increased hardware complexity
- Increased software complexity due to lack of standardized drivers

4) ***Decision:*** We decided to use the USB 3.0 cameras because of the availability of low-cost high-quality cameras. Also, Rockwell Collins has several USB cameras they would like us to use. For our customer, it will be best to base the system around the USB 3.0 option. This may require adding a USB expansion card, but the other advantages of using USB 3.0 make it the first choice for camera connection.

B. Video Subsystem (Software Level)

We will have a framework for streaming the video from different components (camera, image processing unit, display).

1) OpenCV: We could use OpenCV to stream the video and setup processing of the video stream. This seems like the easiest approach, but would constrict us to the functions given by OpenCV, which might be enough for a proof-of-concept.

Benefits:

- Able to write scripts using python
- Can use OpenCL to take advantage of the GPU

Drawbacks:

- Hides functionality
- Questionable performance

2) Direct Memory Handling: We could handle the video stream ourselves on top of v4l provided by the Jetson, but would require some low-level knowledge of the Jetson?s hardware. It would allow for more control over the video stream and give better performance.

Benefits:

- Fast performance
- Full control of operations

Drawbacks:

- Requires low-level knowledge of the Jetson architecture
- Complex, which make it much easier to develop bugs
- Not portable to other boards

3) Gstreamer: We could use Gstreamer, a free library with video functionality. Gstreamer is widely used as an industry standard, and works on different architectures. If we choose to use Gstreamer we are limited to the API it provides, which is pretty limited because it?s meant for video conversion.

Benefits:

- Lightweight
- Portable
- Scriptable
- Easy pipeline-design

Drawbacks:

- Questionable performance
- May not be able to make use of hardware components

4) ***Decision:*** We will start developing with OpenCV, but if the performance is to slow we may switch to a different option. Video4Linux may be enough to easily move the stream to different components of the Jetson.

C. Image Processing

In order to demonstrate the processing power of the Jetson Tk1, we are going to implement some video processing algorithms.

1) On-board GPU: We could make use of the GPU on the Jetson to do image processing to test the performance. GPUs are commonly used for this type of processing and there are many ways to access the functionality of it.

Benefits:

- Easy to use
- Easy to maintain
- Easy to update

Drawbacks:

- Best performance (compared to the ISP)

2) Integrate Stream Processor: We could use the two ISPs on the Jetson to do image processing. ISPs are meant for video processing and would provide the best performance. It would allow us to bypass the memory if we connect the camera directly to the camera port. This would require tuning the video processing operations to use the ISPs interface.

Benefits:

- Fast
- Can bypass memory buffers

Drawbacks:

- Not-entirely portable
- Only one camera port

3) PCIe GPU: The Tegra has a PCIe port that could be used to install a more powerful GPU for video processing. This could be a good option if the Jetson doesn?t have the required video processing power out of the box.

Benefits:

- Could potentially be very powerful

Drawbacks:

- Uses more power

4) ***Decision:*** We are going to use the GPU for our image processing. Use of the GPU is widely supported by applications and is easy to code and maintain. Also OpenCV on the Tegra Tk1 directly supports it, and many other choices have support for GPU processing. It will also be able to handle our requirement of 30 frame rates per second processing.

D. Demo Interface

We will need a framework in order to display the video and user interface.

1) QT Framework: The QT framework is a stable library that provides user interface. It is cross-platform and has a simple API, but it is not a lightweight framework. Its ability to display video properly is undetermined.

Benefits:

- Easy to use
- Many features

Drawbacks:

- Bulky library
- Not specifically for video streaming

2) OpenCV GUI: OpenCV has various UI elements built in allowing for video display and user interaction. It is built into OpenCV and since we have decided to use OpenCV this makes it very simple.

Benefits:

- Part of the OpenCV library
- Easy to code

Drawbacks:

- Slow
- Limited API
- Limited functionality to control the look and feel

3) OpenGL: We could use OpenGL to display the video and interface. This would give us full control over the look and feel and it would be very responsive.

Benefits:

- Fast
- Portable
- Powerful (full control of display)

Drawbacks:

- Long development time
- Hard to maintain

4) ***Decision:*** We will use the OpenCV built-in GUI for the interface, as it is simple to use if we are already using it for the memory manipulation and video processing. It could even be used in combination with other choices, and would still be a good choice as the API is easy to use and is large enough to have all the features we want.

VII. Technology Review Document Changes

A. *Technology Changes*

1) Camera Interface: For our camera interface we ended up using the USB 3.0 camera setup. This made our code simple, but did require us to work with a USB 3.0 expansion card which had trouble supplying the necessary power to the cameras, requiring the camera's power supplies to be connected while in use.

2) Video Subsystem: For our video software subsystem, we ended up using a different system entirely: CUDA. We talked about OpenCV's performance issues in the Technology Review and they ended up constraining us too much. OpenCV was great for getting a simple program working, but it only ran at around 14 frames per second which was not viable. Writing lower level code in CUDA kernels and using the CUDA API allowed us to take more control over the memory in the TX1 resulting in higher frame rate.

3) Image Processing: We ended up using the on-board GPU for our image processing which is what we thought we were going to use. It was the best option because it was the simplest to code and investigate and it supposedly would meet the requirements, so we had no reason to believe it wouldn't be capable for the project and if it failed we wouldn't have wasted much time.

4) Demo Interface: In our technology review, we discused a lot of software that we thought we might use to display the output of our camera system and decided to use OpenCV to create the project. We ended up using OpenGL instead. Our discussion of the benefits and drawbacks of OpenCV vs OpenGL in the technology review show why we eventually made the switch. The OpenCV GUI ended up being too slow compared to OpenGL. This was because of the benefit we list in OpenGL, saying that it was powerful. We had full control over the buffers that were send to the window for rendering, which allowed us to employ memory mapping magic to reduce the number of memory copies and thus achieve the latency and framerate required for the project. While this was more expensive in terms of coding hours, it ended up being necessary for the project. Though attempting to use OpenCV was a worthwhile effort to learn more about our development environment quickly.

VIII. Weekly Blog Posts

During the duration of our project, our team posted weekly blog posts at the end of every week on Friday to recap progress, problems, and ideas we had come across during that week. We used these blog posts as a way to keep our TA and clients updated with the most current information about our project. This section contains all of our blog posts broken down by term, then by date of each Friday they were posted.

A. *Fall Term 2015*

1) October 16, 2105:

a) First Meeting and Problem Statement Developed: This week consisted of our initial meeting with our sponsors (we found out that we more or less have two of them), and we were also able to construct a complete problem statement of the project. The meeting occurred Monday in the the Valley Library. Our sponsors were able to drive down from Rockwell Collins in Wilsonville to have a face-to-face meeting. It was very nice to be able to have this meeting and to clarify many of our questions/concerns about the project. After the meeting was completed, the team split up different bullet points of the problem statement to work on. A Google document was created, where we were all able to collaborate and work on the document. Once the content was created we went through and edited it all to have the same voice and proper language. The final document was then created as a PDF and sent our sponsor to be signed. They were very happy with the outcome and feel we were able to capture an accurate and complete vision of the project. During this week we also created the group project page and blog site, and were able to do a little research involving the capabilities of the NVIDIA Jetson board. This concluded the week for our project.?

2) October 23, 2105:

a) Requirements Document: This week we drafted a requirements document and sent it off to out client for review. We are planning to meet with him soon over a WebEx VOIP, which I've never used. Hopefully that works out well. We're going to get a signature for our requirements document for next week. We also fixed some issues with SharePoint this week. We accidentally created two sites and in a very tense moment, deleted one. Also getting everyone the correct permissions on SharePoint has been difficult.

3) October 30, 2105:

a) Requirements Document Continued...: This week overall has been a slow one, waiting for review and input. It took until later this week to get some feedback from our client about the rough draft of the requirements document, which was needed for further advancement on the document. The feedback was very helpful in letting us know that we need to narrow our requirements down, and focus on additives once we can finish the main requirements. We have begun to make a more defined document with added descriptions and better defined requirements. Our document will be completed and signed by Wednesday, November 4th.

4) November 6, 2105:

a) Final Revision of Requirements Document: This week, due to an extension, we were supposed to turn in our final draft of the requirements document signed by our sponsor. On Tuesday, the class was informed that many of our requirements documents were not up to standard. We now have until Tuesday of this coming week to rewrite our documents and submit them again. We were given an IEEE standards document of how to correctly structure a requirements document. We took this document as a group and restructured our requirements. We added a cover page, a table of contents, and much more description of the project as a whole and each individual task as well. On Wednesday, we had our first meeting with our TA, Xinze. He was very helpful in giving us feedback for our original document. Once we had updated the old document into the newer format, we sent him a copy of it and got even more feedback. Once the document was completed, and the Gantt chart created according to our requirements was added, we sent it to our customer for his input. We are still currently waiting on a response. I believe that we have a pretty solid requirements document now compared to our original attempt. We also have several other assignments coming up in the few weeks, like a technology review, an elevator pitch, and a rough draft of our poster.

5) November 13, 2105:

a) Final Final Revision of the Requirements Document: We revised the requirements document yet again and submitted an almost finished version to our class. We then were able to contact our client for edits and are working on finally finalizing it. We also wrote up the tech document. It looks like we're going to start out using OpenCV and see if we can setup up the system using that and then optimize it from there. Also we might switch to using the TX1, a newer version of the TK1.

We're having a meeting with our client on Monday and should prepare for that.

6) November 20, 2105:

a) Rockwell Collins Meeting and Poster Development: On Monday this week, we met with Carlo and Weston from Rockwell Collins for our monthly check in. There were several items on the agenda:

- We went over the revisions we made to the Requirements document after receiving comments from Carlo via email last week
- We discussed alternative single board computers, including the Sapphire Tech Step Eagle board currently being used by Rockwell Collins for a similar project as well as the new Jetson TX1
- We formed a plan for evaluating camera options. Rockwell Collins is providing us with one USB 3.0 camera to test and see if we can get it to work with our Jetson. If it is not supported by the Jetson, we will begin looking at other options
- We discussed our Gantt Chart and received recommendations from Carlo about additional milestones we could put in to add more specificity to our project plan.
- Carlo and Weston are still unable to connect to our sharepoint site, so Hailey is working with them to see if she can get them set up.
- We discussed potential dates for our next meeting, which will be somewhere between December 14-16. Depending on if we get the hardware to work together, we may meet in person either in Portland or Corvallis.

After our meeting, we also created a rough draft of our project poster for the Engineering Expo.

7) November 27, 2105:

a) Thanksgiving: We had the second half of thanksgiving off and so we weren't able to accomplish much more than planning out a few meetings for the next week.

8) December 4, 2105:

a) Design Document and Progress Report: This week we created a design document and progress report. The design document followed the IEEE Std 1016-2009 format, which we went through and met on Monday to start brain storming. Our main concern with this document was identifying our viewpoints and creating corresponding diagrams for each. Below gives an overview of our viewpoints and the corresponding design languages:

1) **Context viewpoint:** This viewpoint shows the different potential users of the software and how they would interact with our system.
   Design Languages: Use Case Diagram
2) **Structure viewpoint:** This viewpoint shows how the streaming video flows through the system, and identifies the internal and external data connection points in the system. This viewpoint also shows the

components of the system which enable connection of cameras via USB as per the SRS.
Design Languages: Data Flow Diagram

3) **Interaction viewpoint:** This viewpoint shows the order of operations on processing a video frame, as well as how the timing is integrated to meet the performance benchmarking requirement.
Design Languages: UML Sequence Diagram

4) **Information viewpoint:** This viewpoint details how the modular video processing system determines the order of operations for the various elements and algorithms it can create. It shows how the system meets the requirement for modularity and the capability to use multiple input and output devices.
Design Languages: Entity Relationship Diagram

5) **State Dynamics Viewpoint:** While the algorithm implementations in our design are stateless, the system itself is not. This viewpoint shows the transitions between different states and provides a road map for different states which will need to be individually tested.
Design Languages: UML State Transition Diagram

The design document will be our roadmap to implementing our system to meet our requirements. We took special care in ensuring this document was done correctly as it is going to be a point of reference for the remainder of the year. We also created our progress report which covers a summary of the activities, problems, and solutions we came across during this Fall term. This winter break we will begin to familiarize ourselves with the Jetson development environment, and possibly connect our camera system to the board to find out how to connect the two environments.

B. Winter Term 2016

1) January 8, 2016:

a) HawkEye Crew is back at it: After a long, and much needed break, we are back into the grind of Winter term. This first week was busy for us. We began this week by finding that our Tx1 had a manufacturing problem, preventing it from connecting to wifi. We notified our instructor who gave us another board that turned out to be defective in the same way. Soon, we will get a Tx1 that definitely works correctly and start developing on it, but in the meantime, we've been working with the Tk1.

We installed graphics drivers on the Tk1 as well as drivers for the camera we currently have. We found that we need to recompile the Jetson Tk1's operating system in order to grab full 2048x2048 images from the camera, but we're able to grab images at a lower resolution using the FlyCapture API currently.

We got some extra hardware: a USB hub and two USB wifi dongles. Kevin supplied the hub and one of the USB wifi dongles.

We're trying to setup a meeting with Carlo this week to go over milestones and a timeline, which we built this week.

We also met with our TA today and learned what we need to do for the alpha stage. We must write up a document describing the process so far. The criteria for this document hasn't been defined clearly yet. We also have to make a video demonstrating our product. We were thinking of showing the camera in action and going over some of the code/interface for the project. This video has to explain each point of the requirements document which we should look over.

We still have to get a second camera working and setup a framework for image processing before the alpha stage.

2) January 15, 2016:

a) Beginning the Development Process: Week 2 of Winter term of is beginning to smooth itself out from last weeks discovery of the Jetson Tx1 manufacture bug and the difficulties of connecting to Wifi with the Tk1. We had to recompile the Tk1 with a special version of linux4tegra, called "Grinch", to get on output from the USB 3 camera. It proved to be not a simple task, as we still have some connection and display issues with the video. We began to realize that development on the Tk1 may not even be worth the struggle, and decided we need to get a working Tx1.

With Kevin to the rescue, we were able to pick up a new Tx1 from him that is fully functional, with all the bells whistles (including working on board Wifi). We were able to pick that up on Thursday, so we haven't had much time yet to try to get a working product between the Tx1 and our USB 3.0 camera, but plan to try and get that up and running before our meeting with Carlo on Monday via Webex.

Carlo requested a monthly milestone marker as to show what we should have to present to him and Weston at each of our monthly meetings. We created milestones Winter term, which will cover both our alpha and beta releases. Our rough idea of milestones is listed below.

January Meeting: January 11th

- Download graphics driver onto Jetson TK1
- Download device driver for camera onto TK1
- Display image onto monitor

February Meeting: Alpha Release February 11th

- Download all needed software onto Jetson TX1
- Display image from built-in camera on monitor
- Display image from RC camera and built-in camera
  - Stitch the two images together
- Have option for applying filters, but not actual apply the filters

March Meeting: Beta Release March 14th

- Fully functional filters
- Stretch Goals:
  - Modularity with JSON
  - Multiple monitor display
  - Object tracking
  - Display video operation metrics
  - Compare with different single board computers

To finish off the week we all three sat down at meeting to revise our requirements document, as it is what we need to be moving forward with for the release of our alpha and beta product releases. Only a few things were changed, like making sure we have two working cameras for our system and to not specify only using the Tk1 or the Tx1. As we move forward we plan to rely more heavily on the Tx1, but want to allow ourselves the room to be able to fall back on the Tk1 if need be to complete our project. We will be uploading that document to the Sharepoint tonight and presenting the updated to our client on Monday at our monthly meeting.

3) January 22, 2016:

a) Meet-up and Decisions: We met with Carlo this week to show Carlo our current progress on the project. Our progress included a working video output from on board camera on the Jetson Tx1. We were able to display the image at 1080p with 30 or 60 fps. This video stream used embedded streamer code to transform the image buffer generated from the camera into a Video4Linux compatible format which then was read with OpenCV and displayed.

After the presentation of our project to our client, we went through the milestones we had sent Carlo last week. We are ahead of schedule with most of our February milestones complete. We just need to add a second camera and generate a few filters.

We have asked Carlo to send us the model number of the 4 other PointGrey cameras he has on hand. We also have asked for a list of typically used types of filters for aircraft vision systems.

We are to send Carlo a block diagram of the program flow from camera to display. This will be used to judge if it will be possible to incorporate RC's virtual camera system, easily, into our software.

We are also going to send some sample output of the video we have produced.

Lastly we will be researching a few options of cameras to add to our project and sending them to Carlo for his input on which ones would suite his image of the project.

We are focusing on getting two cameras' output displayed before moving on to fixing compatibility with other cameras or filters, and decided that using two USB 3.0 cameras would be the fastest way to yield this result.

4) January 29, 2016:

a) PointGrey Camera on Point: Last week we were able to get the ?camera that is onboard the Jetson Tx1 to display an image at 1080p and 30fps. We were having much more difficulty with the USB 3.0, PointGrey, camera that Rockwell Collins had supplied for us, but this week we managed to get it to work. ?

We were missing two big elements as to why we had so much trouble with getting a video output from it last week. We recompiled the Linux kernel to increase the USB buffer size to accommodate higher resolution image capture from the PointGrey camera. We also acquired code from PointGrey to interface between the proprietary camera API and openCV.

After we were able to get the video output working we implemented two basic edge detection functions using several of openCV's built-in functions to gain familiarity with the openCV API. Now that we have the USB 3.0 PointGrey camera working and running some filter capabilities, we have discussed the need for additional cameras that Carlo has available for us.

In class this week we went over the requirements needed for our Alpha release, which will contain a written document and 20-30 minute presentation of our progress with the project so far. The Alpha release must have all capabilities present, but do not need to have full functionality. For our project, this means having a video stream from multiple cameras displayed at decent frame rate and the ability to apply simple filters to the video stream. The next few weeks will comprise of writing the document needed for the release and moving forward with operating two cameras into one output.

5) February 5, 2016:

a) Modularity and Multiple Cameras: A major goal for this week was to get a second camera. This would allow us to be able to begin furthering our software to handle processing of multiple video streams into one output. Multiple cameras are a requirement that is needed for us to able to have a complete project at Beta release, but something we wanted to be able to at least support with our Alpha release. We were able to get in contact with Carlo, and set-up a meeting allow us to (very generously) borrow two more PointGrey cameras with additional lenses as well. On Thursday Hailey drove up to Rockwell Collins in Willsonville to have an informal meeting/lunch with both Carlo and Weston to pick up the cameras.

Last weekend Ryan got a full modular system built within our software. ?Our software reads a JSON configuration file to set-up the modules??. This allows for more flexibility with the use of different cameras and their software. It also gives us the ability to different filter modules and only apply the ones we want a specific time. It could handle more flexibility with use of different types of cameras only being used at specific times or for specific reasons. For example, a long range camera may only be activated when needed to find the runway. It could then be turned on and processing when needed, but deactivated to not decrease the performance metrics when not needed.

At the beginning of the week, the modular system was able to output a static image which allowed us to demonstrate that the system was working. We then created a module to handle the PointGrey camera software. This allowed us to get two of the cameras up and running with one video output. We haven't gotten the USB PCIe card yet, so we can't actually implement dual cameras over USB3 yet, but were able to run one on USB 3.0 and the other on USB 2.0.? Even though one of the cameras was being ran with USB 2.0, we were still able to capture decent frame rate and are below our required latency.? This weekend we will be fine tuning some of the video capabilities for our Alpha release presentation.

This week we also began the outline of our written midterm progress report and will hopefully have a solid rough draft for our meeting with Xinze on Monday.

6) February 12, 2016:

a) Midterm Progress Report: This week no real progress on the software was made because we were busy creating our Alpha release. As we had all of the required features implemented to have an alpha release we were able to focus on getting that out instead of rushing to finish any last features needed. The release consisted of a written progress report and a 20 min video presentation of the project to include a powerpoint and demonstration. The video presentation basically covered the topics within the written report and a demonstration of the output of our project. We used Prezi to create our power point presentation.

The report that covered a brief introduction of our senior project, our current progress, problems that have impeded our progress, and any preliminary results that have been gathered. It also described what is left to do before all of the requirements are met for the project along with the potential for some stretch goals if time allows. The report includes a few interesting pieces of code and a description of what this code does. It was constructed using the IEEEtran style guidelines.

On Thursday, we were also able to get our PCIe card that will allow us to hook up multiple USB 3.0 cameras. In the next weeks, this will be the direction we are moving with development. For our Beta release we are going to need to get two cameras running on USB 3.0, increase our performance metrics, and build our own filters.

7) February 19, 2016:

a) Slow Week: This week, we didn't accomplish much. We all had a lot of work outside the class this week. We met on Monday with our TA and explained to him about the progress we had made. He asked us about how our current alpha-level software works, all of the problems we had encountered, and our plan for completing the beta in the next few weeks. He also informed us about the coming class meeting next week. We are planning to meet this weekend to do more work on the beta project.

We also showed our client our video presentation and he approved of our progress.

8) February 26, 2016:

a) Technical Stuff: This week, we've been trying to make the visual frame rate more smooth on the Jetson. ?While we're capturing and processing the images, we have the required frame rate, but, display to the screen is very slow. Our metrics claim that we can get 30fps on the screen, but the resulting visual display is obviously not 30fps.

One solution we have to this problem is to use a low level API to display the images. We are planning to leave the capture and processing system in place, but simply replace the way we display the images, giving us more control over the display process. Our plan for replacement is to use GLUT which is a window system based on OpenGL. We have setup this replacement but haven't implemented the benefits of using GLUT which is zero copy. This means writing the camera images directly to a buffer that is read by the GPU. because the GPU and the CPU share memory, this is possible.

Also we have very inconsistent results with these frame rates. Some of this was solved by writing more configuration details into our code, but the cameras sometimes remain in strange states and we are working on solving this.

We also found today that the Jetson TX1 isn't displaying on a monitor and wasn't responding to ethernet and had to reflash. This is mostly due to bad configuration on the last power down. Thankfully we were prepared for a situation like this and were able to reflash the TX1.

9) March 4, 2016:

a) FPS Up To 30 Using CUDA: This week we met our requirement for one camera by using CUDA libraries to process and handle the video buffers better. We currently have one camera outputting frames at 30 fps. We are trying to get this same performance with two cameras to fully meet our requirements.

We've also been getting the frame rate above 30 fps using more tricks in CUDA.

Another feature we added this week was a line detection filter, which can run without dropping our frame rate any significant amount. Eventually we will test the Jetson with these filters to see how many operations it will take to notice a drop in fps.

We also worked on the poster and end of term report a bit. We are having some trouble compiling tex files, but that should be sorted out soon.

We've also been having strange inconsistencies in the state of the cameras between operations of the software. This is mostly solved by unplugging and replugging the cameras, but we're trying to come up with a software solution to this problem.

10) March 11, 2016:

a) Everything's Working: This week we completed the requirements listed in our design document. ?We currently have 2 cameras streaming at 33 frames per second and 3 cameras running at 31 frames per second over USB3. We also got one camera running at around 100 frames per second but this program is unstable and crashes more than our program that runs at 30 frames per second.

We are planning to move forward using this program that grabs and displays frames at 100 fps, working out the bugs as we go along. This program can run at 100 fps because we spawn different threads for converting the color space from raw to RGB. Before, this was being done in the same thread as the capture thread which delayed the frame rate. Spawning new threads for processing had sped up this process.

We are planning to speed up frame capture even more by doing the color format conversion on the GPU using CUDA. This should get us to the maximum frame rate possible on the TX1.

C. Spring Term 2016

1) March 25, 2016:

a) Spring Break Progress: It's spring break and we have got a lot done this week.

1) **Color space conversion algorithm (Bayer Demosaicing):** To increase the flexibility of our software and provide a platform to implement additional computer vision algorithms, we implemented a very fast CUDA-based color space conversion algorithm that converts the raw Bayer Color Filter Array image data to RGBA format. This enables us to create algorithms based on not only intensity but color as well. This algorithm replaces the CPU-intensive conversion method provided by Point Grey, which is only capable of converting about 30 frames per second maximum on our hardware. With the new CUDA-based algorithm, our software can process 100 frames per second with both full color and grayscale outputs. This allows us to use both algorithms requiring color and grayscale images.
2) **Our own line segment detection algorithm:** Additionally, we have begun development of a custom, entirely new line CUDA-based line segment detection algorithm. Our system is based on (but entirely different from) a CPU-based algorithm called LSD, which stands for Line Segment Detector (http://www.ipol.im/pub/art/2012/gjmr-lsd/article.pdf). The CPU based algorithm is far too slow to ever be used in real time, but we hope to get our GPU based algorithm to detect all lines in an image in under 1ms. In it's current phase, our line segment detector can detect sharp edges in about 3ms, but cannot differentiate between straight and curved lines. However, the newest version that is currently

in development will perform all of its operations within the GPU's register space or L1 cache memory, which will eliminate 80% of the high-latency global memory operations required by the algorithm and result in a much faster processing time.

3) **Next Steps:** Our line segment detection algorithm not only detects individual lines, but also rapidly finds intersecting line segments, which lays the groundwork for us to detect runways even without the runway lighting system. Our runway detector will look for long, narrow, roughly rectangular shapes (with compensation for obstructions and intersecting runways), and then identify markings within the runway, if possible (depending on the range and focus). It will then combine this information with any detected lighting patterns to determine the probability that it is in fact a runway. If a runway is detected, it will overlay the detected lines onto the screen, as well as an appropriately scaled directional indicator that shows the direction of the landing strip.
4) **Parallell Processing:** This week, we also created an entirely new data interchange format designed specifically for parallel processing multiple video streams. This class, called Frame, holds both the RGB and Mono images for an individual frame, keeps track of the time it was created and the time it is destroyed, and also synchronizes execution of GPU and CPU modules on the same data. CUDA has what are known as streams, which are independent execution queues for GPU kernels. They allow multiple kernels to run on the same GPU without waiting for each other or interfering with each other in any way. Each frame object has its own stream, which avoids congestion on the GPU. An example would be, if Frame1 is ready to be processed by the line segment detection algorithm, but Frame2 is currently being demosaiced, and they both utilized the default stream, Frame1 would have to wait until Frame2 is done with demosaicing before it could start the line segment detector. Furthermore, Frame2 might even start its line segment detector before Frame1's thread could wake up and begin execution, so Frame2 might even be processed before Frame1. With streams, this situation is avoided entirely because Frame1 can be processed while Frame2 is demosaicing, and is not blocked or delayed. This method resulted in much greater stability and an overall higher frame rate.?

2) April 1, 2016:

a) Meeting with Carlo: This week we met with Carlo and Weston to show them our progress. ?The meeting went well. We were able to demo the project so far, which they were impressed by. They had questions about how exactly we measured our frame rate. We were using software timers to test our system, which apparently can lie about the frame rate. They suggested that we use a high-speed camera and an LED to measure exactly how long it takes for the image to show up on the screen. We are currently working on using a GoPro for this as we do not have access to any higher speed cameras.

We also talked about our display at expo. Carlo and Weston said that creating a scaled air strip would be a good idea. They suggested putting NIR (Near Infra-Red) LEDs on the air strip and detecting them with our NIR camera to show off an asymmetrical camera system, which is an important part of arial vision systems.

We also explained our system to them in detail, such as how our program is designed and how the program flows. Carlo and Weston wanted more information about our program and suggested that we create diagrams that explained all parts of our system.

All in all they were very impressed. We are planning to meet again soon.

3) April 8, 2016:

a) Building the Runway: This week, we've been working on creating a runway for the model for expo. We're trying to make the scale correct relative to a real airport so we've been going over FAA documents trying to get all the markings right.

We also met with Xinze and described our progress so far, along with our meeting last week.

We also decided that we're going to use a GoPro, if we can,? to measure the true latency of our system.

4) April 15, 2016:

a) Gearing Up: This week we sent out a lot of emails. A bit of the background processes in the last couple weeks was figuring out exactly what hardware we needed to create a good presentation at expo. We've moved to get a few pieces of hardware:

- A lens spacer to make our 8mm lense less blurry at the required distance.
- An NIR filter so that our NIR camera doesn't pick up visible light.
- A GoPro with 120 fps to test the latency of our cameras correctly.
- NIR LEDs to line the runway, which will be picked up by our NIR camera.
- A zoom lens to see the letters on our model runway.

We've almost finished creating the model runway and it looks great. We still need to add the center lines to it. Also we've acquired the GoPro from Kevin and got it working.? We might still need an SD card expander to read from the GoPro, but I've got one.

5) April 22, 2016:

a) Expo Poster Final Touches: At the start of this week we acquired a GoPro from Kevin to use for testing the latency of our program. Our initial latency testing shows about 84ms for a color image with edge detection and 24ms for the 3 cameras combined with the sobel filter in black and white. We able to see that the color conversion was causing a great increase in our overall system and have been working this week to try to improve that.

We also created an overlay filter to counter the parallax effect for our live demonstration at the engineering expo. This visually aligns the inputs from the two cameras so that the runway appears in the same location on both images. Shifting the pixels of one image to match the other image is how this is . Our demonstration will combine the image of a standard camera to identify the runway during daylight with our NIR camera to detect the approach lights when it is dark. The overlay filter will combine the two camera's images and properly align them to get a clear output image.

We also have ordered the NIR LEDs, and received the camera spacers and the NIR filter we ordered last. We are still trying to get in contact with Carlo to receive the zoom lens. This lens will allow use to identify the runway numbers better by zooming in on them whenever it is needed.

?

6) April 29, 2016:

a) Minor Tweaks and Improvements: The logistics for this week consisted of a meeting with Xinze to discuss our current progress and/or any bugs we are having with our project (we currently have no issues), and also about the submission of the poster to the library for printing. ?

We also acquired some LED strip lights from Kevin to use as our approach lights on our runway. We are planning to operate them using a Raspberry Pi. We have been in contact with Carlo about the zoom lens and he is mailing that to Hailey's home address, that way, next week, we can begin further development of the zoom functionality for our demonstration.

Carlo also mentioned he had a few comments on the poster as feedback. We are currently waiting for his email description to make any changes to our poster, which we then will submit to the library for printing.

On the development side for this week, we developed a new, cleaner version of our code base using a new ”Pipeline” class. This class provides parallelization without any effort from the developer, who simply has to implement the image processing functions and use them within an inherited virtual class member function. Additionally, both the display output and the pipeline input only grab the most recent frames, which reduces latency. By using this new pipelined approach, we were able to drop the end-to-end latency for our most intensive operations from 85 ms down to less than 60 ms, as measured with our goPro high speed camera. This decrease in latency appears to be roughly an inverse linear relationship. This is all with respect to the number of concurrent pipeline threads, up to around 6 concurrent threads, after which the increase in efficiency seems to be minimal. The decrease in latency came with only a very minimal drop in frame rate, which came down to around 90fps, which is still more than three times the frame rate we require. This approach also works much better with multiple cameras, as the pipeline class takes care of the setup and teardown of the cameras, as opposed to doing all of that in the main/host program.

7) May 6, 2016:

a) Spring Midterm Progress Report: This week mainly consisted of working on midterm progress/release report and presentation. We have now finished both with only a few minor touches to the video to make and turning it all in. Last week we had divided up all the sections that needed to be added and/or changed to the document. That organization made this report very easy to construct and were able to work on it without having to meet a bunch. We did get a room in the library in order to record using Hailey's headset so that the audio for our presentation was all the same and very clear.

?We also received the zoom lens from Carlo this week and will begin implementing our full demonstration for the Engineering Expo next week.

??

8) May 13,2016:

a) Preparing for Expo: This week we were each working independently to ensure that expo goes smoothly?.

Some of the things we did this week were:

- Finishing the runway board, adding the runway number and the middle lines
- Creating a simplified version of the program

- Fixing the bugs in the most version of the program and making it more stable

9) May 20, 2016:

a) EXPO!: This week we did a lot of preparation for expo. This included finishing up the code as well as assembling the display and making sure it would all operate correctly during expo.

Expo was really fun and a great opportunity to meet people in industry. I was really great to see our efforts pay off and people really enjoyed learning about our project, even kids. It seemed that the industry professionals were pretty impressed with it too as we got a lot of interest.

We've got some follow ups with our client, but besides that we simply have the final report to do and then our term is complete.

10) May 27, 2016:

a) Engineering Expo Results: This week during class, Kevin recapped how the engineering expo had gone all for the computer science department. We found out that NVIDIA was happy about the use of the Jetson TX1 in our project and the capabilities we were able to achieve from it. We also found out that we were voted first place out of all the computer science projects and received a gift bag as our reward.

?Also this week the requirements for the final report and presentation were released. Our group has looked over the requirements and are planning to begin working on it next week.

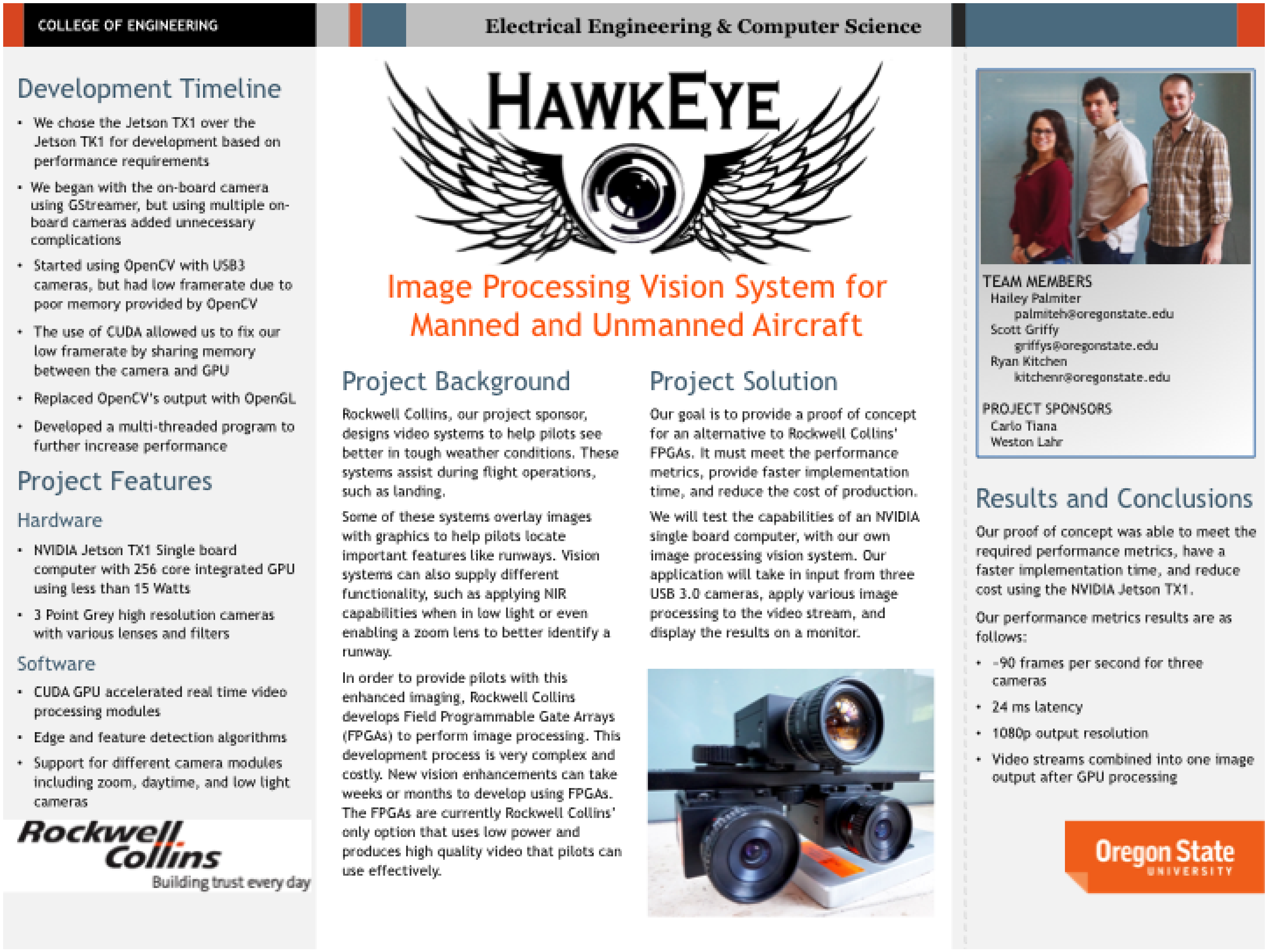
?

11) June 3, 2016:

a) Final Report: This week we picked up our flash drive for all of the documentation and code we have generated over the duration of our project. It will be attached to the back of our written report, which we have started this week. On Wednesday we had a meeting to break up all of the different part of our final report as it is a very large document we are generating. We talked minimally about the presentation, but have decided to put that off till we are finished with the written report. Once that is finished we will make an outline to follow for our presentation. We are hoping to have the written report done early next week.

We have also contacted Carlo to try to figure out when we will be able to bring the project up to Rockwell Collins. We are hoping that the end of next week will work for him as we should be complete with our presentation and live demonstration by then.

IX. Final Poster



COLLEGE OF ENGINEERING

Electrical Engineering & Computer Science

Development Timeline

- We chose the Jetson TX1 over the Jetson TK1 for development based on performance requirements
- We began with the on-board camera using GStreamer, but using multiple on-board cameras added unnecessary complications
- Started using OpenCV with USB3 cameras, but had low framerate due to poor memory provided by OpenCV
- The use of CUDA allowed us to fix our low framerate by sharing memory between the camera and GPU
- Replaced OpenCV's output with OpenGL
- Developed a multi-threaded program to further increase performance

Project Features

Hardware

- NVIDIA Jetson TX1 Single board computer with 256 core integrated GPU using less than 15 Watts
- 3 Point Grey high resolution cameras with various lenses and filters

Software

- CUDA GPU accelerated real time video processing modules
- Edge and feature detection algorithms
- Support for different camera modules including zoom, daytime, and low light cameras

Rockwell Collins

Building trust every day

HAWKEYE

Image Processing Vision System for Manned and Unmanned Aircraft

Project Background

Rockwell Collins, our project sponsor, designs video systems to help pilots see better in tough weather conditions. These systems assist during flight operations, such as landing.

Some of these systems overlay images with graphics to help pilots locate important features like runways. Vision systems can also supply different functionality, such as applying NIR capabilities when in low light or even enabling a zoom lens to better identify a runway.

In order to provide pilots with this enhanced imaging, Rockwell Collins develops Field Programmable Gate Arrays (FPGAs) to perform image processing. This development process is very complex and costly. New vision enhancements can take weeks or months to develop using FPGAs. The FPGAs are currently Rockwell Collins' only option that uses low power and produces high quality video that pilots can use effectively.

Project Solution

Our goal is to provide a proof of concept for an alternative to Rockwell Collins' FPGAs. It must meet the performance metrics, provide faster implementation time, and reduce the cost of production.

We will test the capabilities of an NVIDIA single board computer, with our own image processing vision system. Our application will take in input from three USB 3.0 cameras, apply various image processing to the video stream, and display the results on a monitor.

TEAM MEMBERS
Hailey Palmiter
palmiteh@oregonstate.edu
Scott Griffy
griffys@oregonstate.edu
Ryan Kitchen
kitchenr@oregonstate.edu

PROJECT SPONSORS
Carlo Tiana
Weston Lahr

Results and Conclusions

Our proof of concept was able to meet the required performance metrics, have a faster implementation time, and reduce cost using the NVIDIA Jetson TX1.

Our performance metrics results are as follows:

- ~90 frames per second for three cameras
- 24 ms latency
- 1080p output resolution
- Video streams combined into one image output after GPU processing

Oregon State UNIVERSITY

X. Project Documentation

A. Project Overview

Our project is a proof of concept to determine the usability of the NVIDIA Jetson TX1 for aerospace computer vision applications. It demonstrates this capability by processing multiple video streams with multiple image filters at high frame rates and low latency. This program is designed to run without needing any interaction from the user, however we have included the ability to switch between different video processing modes in order to demonstrate the various filters and camera views. The HawkEye Vision System captures video from cameras, processes it using various filter algorithms, and displays it on the screen. It utilizes both the CPU and GPU simultaneously to create a very high throughput, low latency video feed. The CPU portion of the program is responsible for capturing video from the camera using Point Grey?s Flycapture API, and managing the parameters of the GPU programs (which are called kernels). The CPU program also manages user input and sets up the OpenGL output system.

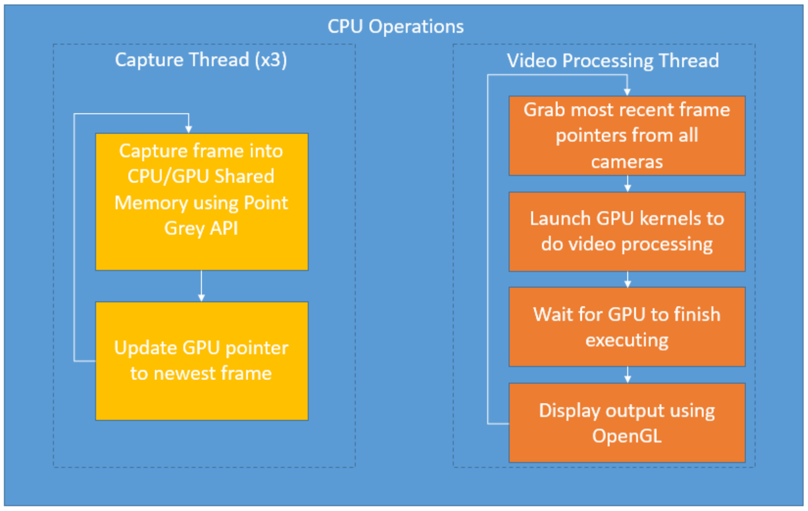

CPU Operations
Capture Thread (x3)
Capture frame into CPU/GPU Shared Memory using Point Grey API
Update GPU pointer to newest frame
Video Processing Thread
Grab most recent frame pointers from all cameras
Launch GPU kernels to do video processing
Wait for GPU to finish executing
Display output using OpenGL

Fig. 1: CPU Operation - showing the functions of the different CPU threads

The GPU does all the heavy lifting in this program. It performs multiple video processing operations simultaneously. The GPU program is divided into several streams, each of which operates independently of the others, much like the multiple threads on a CPU. The Jetson allows multiple GPU kernels to run at the same time, as long as they are in different streams. In the following diagram, each box represents a different GPU Kernel (with the exception of the bottom right box which is actually the same as the three bottom left kernels, just abbreviated to save space). These kernels are executed in order by their corresponding CUDA streams, which is essential.

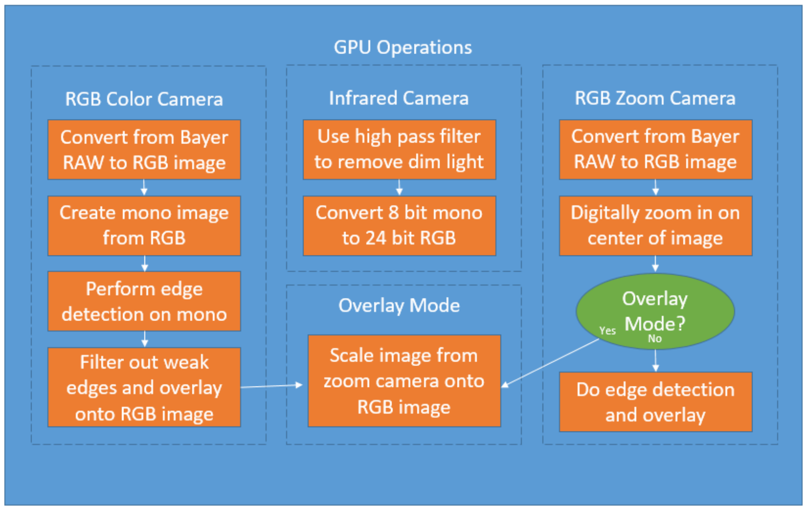

GPU Operations
RGB Color Camera
Convert from Bayer RAW to RGB image
Create mono image from RGB
Perform edge detection on mono
Filter out weak edges and overlay onto RGB image
Infrared Camera
Use high pass filter to remove dim light
Convert 8 bit mono to 24 bit RGB
Overlay Mode
Scale image from zoom camera onto RGB image
RGB Zoom Camera
Convert from Bayer RAW to RGB image
Digitally zoom in on center of image
Overlay Mode?
Yes
No
Do edge detection and overlay

Fig. 2: GPU Operation - showing the functions of the different CUDA streams

B. Installation and Execution

This software is designed to be a proof of concept that demonstrates the computing power of the NVIDIA Jetson TX1. It is not meant to be installed on other hardware, as it requires the use of the GPU present in the Jetson, and is specifically optimized to use the Jetson?s CPU/GPU shared memory that is not present on other devices. There are a number of example programs created using various filters and elements of the HawkEye system. They are run simply by executing them on the console.

This software is designed specifically for the NVIDIA Jetson TX1 and for use with Point Grey USB 3.0 Cameras. It can be recompiled for other systems with CUDA-enabled GPUs however this has not been tested.

C. Example Controls

The example program we used at Expo has several controls.

Camera Views:

[1] RGB Camera

[2] NIR Camera

[3] Zoom Camera

Filters:

- RGB and Zoom Camera Views:

  [G] Enable edge detection

  [F] Disable edge detection

  [H] Decrease edge detection threshold (more edges)

  [J] Increase edge detection threshold

  [{] Zoom out

[**}**] Zoom in

- RGB Only:

  [**Z**] Enable zoom overlay

  [**X**] Disable zoon overlay

- IR Only:

  [**A**] Decrease IR Threshold (brighter image)

  [**S**] Increase IR Threshold (see only bright lights)

Quit Programs:

[**Q**] Quit

XI. Discovery of New Technologies

The primary resource used for this project was the CUDA 7.0 User manual, found on NVIDIA?s website. This manual contains a best practices guide which explains how to use various optimizations and produce code that best takes advantage of the GPU hardware.

The edge detection algorithm we used is an adaptation of the LSD Line Segment Detector, which was originally developed as a graduate research project. The original research article can be found at http://www.ipol.im/pub/art/2012/gjmr-lsd/.

Lastly we made use of various documentation from Point Grey, which described how to set up the cameras, alter various settings, and determine the appropriate Bayer conversion mode (there are multiple different input types). Their website is www.ptgrey.com.

XII. Personal Experiences

A. Hailey Palmiter

1) What technical information did you learn?: I began this project with no real background in hardware and no background in vision systems to include graphics libraries. Therefore the entire technical side of our project was a learning experience for me. It started with the use of the Jetson TK1 and TX1 development environment that used Ubuntu. I have interacted with the Linux operating system in the Operating Systems classes, but never like I did with this project. The PointGray camera had their own drivers we had to install and work with. PointGrey also included its own graphical user interface called FlyCapture that we had to manipulate. Thus we needed to understand what was happening within those packages and interface in order to have it do what we needed it to.

The language I got the most exposure with was OpenGL we had to interact with several of its libraries and its toolkit called GLUT. I also learned about CUDA and using CUDA kernels to handle filter operations on our captured image buffers. I learned how an image is captured from the camera and stored into a buffer, which we created into a shared buffer between the GPU and CPU. I also learned how to use parallel processing with image buffers to increase latency and frame rate speeds.

During the creation of the runway I was also able to work with a micro controller in order to program our LED strip lights to produce a runway like approach lighting system.

2) What non-technical information did you learn?: A few non-technical things that I learned involved creating latex documents, FAA runway standards, and creating formal documentation on our project. Using latex and the IEEEtran formatting style we were able to create very professional looking documents. There was a steep learning curve for latex, but once I had figured it out it made all of our documentation very simple to create. Even though I had learned about requirements documents, design documents, and technical reviews in the Software Engineering classes, it was a whole new learning experience creating them for a real industry like project.

Another non-technical experience was learning how interact and deal with a client. During the planning process of our project we had to have a lot more involvement with our client to make sure we were establishing all of the requirements they had for us and were also creating the project they had in mind. Later in development I had to keep them updated with our progress, set up meeting, create an outline for the meetings along with producing meeting minutes after it had concluded. Throughout the entire project we needed to assure that they were informed and happy with what we were doing for them, while also following the requirements document we had established.

3) What have you learned about project work?: I have learned that there are a lot of moving parts to a project and that it is much more rewarding than working on homework project that only lasts a few weeks. I learned that being flexible and using more of an agile development process was very beneficial especially for our type of project being more of research and testing project as we were trying to develop the best proof of concept we could. The agile process allowed for us to reevaluate the software and hardwares we were using to find the best possible fit.

4) What have you learned about project management?: I have learned that project management runs much smoother once you are able to find out how other people work. Knowing your team, their strengths and weaknesses, availability, and how they work best is a piece of how to lead people well. I learned how to set each of my team members up for success by assigning tasks appropriately based on what they enjoyed, what they were good at, and the time they had to work on a specific task.

I also learned that staying organized and setting deadlines/goals for things helped us all to stay focused and to complete the things we needed to on time.

5) What have you learned about working in teams?: I have learned that working in a team doesn't matter if you are all the best or worst of friends, but what matters is being able to leave it all at the door to create a professional work environment where all entities are comfortable and feel as being an important piece of the project. I was lucky that our team all got a long well and worked really well together, but I believe it also taught me what a good team dynamic should look and feel like. Each of us had a strength to bring to the project, which created a very strong team dynamic. I also learned about compromising with others on ideas as well as being able to help support team members with tasks when needed. I also learned that working in a team allows each member to bring their own knowledge and expertise to the group allowing everyone to learn from them.

6) If you could do it all over, what would you do differently?: If I could do it all over again there would only be a few things I would do differently as I felt our project went very well. One thing I would do is to allow more flexibility from the beginning planning stages. As this was a proof of concept we should have realized that we were going to need to be very flexible with our developmental process. We began our documents very structured not planning to allow much flexibility. This also would have also promoted a more in-depth technology review, which may have given us a better platform to begin development on. The only other thing I would have done differently was to be more involved in the implementation of new code. I spent much of my time creating documents, setting up meetings, joining in team codings sessions and reviews, but not much on doing my own development using new technologies.

B. Scott Griffy

1) What technical information did you learn?: I learned a lot about the hardware of single board computers. I often work with graphics on desktops, but you don't have to dive into the hardware when you're working on a desktop, all the functionality is given to you through library calls and APIs. On the Jetson we literally had to recompile the kernel at times to get more functionality out of the board. I've never had to recompile a kernel to complete a project and knowing how to do so improved the technical tools I have at my disposal for computer projects.

I also learned a lot on the software side of graphics processing. I learned about CUDA and the cuda libraries, especially how to maximize the performance when I'm working with graphics libraries. When I'm working on higher-level projects, most of the performance enhancements I make are in the algorithm, but when you're working with low level hardware, the difference is in the implementation. Memory copies are the enemy in latency for graphics systems and after we minimized those with custom buffers for the camera capture and GPU textures, we got the best latency. I also learned more about how camera capture APIs work and how to access the physical registers on a camera.

I feel that I learned more technical stuff that will help me in future projects if needed, but I can't specifically recall it here.

2) What non-technical information did you learn?: I think the biggest takeaway that I got out of this project was working with a team in a long term project like this. I've worked in groups on smaller projects before but never for a whole year. In some ways working with a group for a longer amount of time is easier and in some ways it's harder. In a short project, the difficulty is learning how to work with the group in the short amount of time. When you open it up to a larger project, you realize that group projects

I also learned about how to explain technical things to a non-techincal audience. This was important for some of the documents we wrote as well as at expo. Often, when I'm working on a project, I get caught up in the code and never boil it down so that I can explain it to people. This might be the most important part of any technical work, explaining it to other people so it can be used and improved upon.

3) What have you learned about project work?: When you're working on a project, you need to make sure that you're on task and that the task you're doing is important. This is the most important part of project work and a lot of the effort in a project goes towards defining these tasks. If your project is well thought-out and you're on task, the project will get completed and be successful and if you've met and defined good requirements, then all you need to do when you're working on your own is stay on task. This might seem simple, but it's really easy to get off task and try to feature creep on your own, or end a meeting early deciding that you'll figure out the rest of the tasks on your own. These are both examples of bad project work and should be avoided.

4) What have you learned about project management?: Setting up meetings and assigning work isn't always easy. Sometimes the correct path isn't clear, but you still have to make a decision. Sometimes you don't have time to weigh all the options and simply need to choose a route that might not be optimal. Management is about a lot of small things and corner cases. There's never a correct answer and you never make the best decisions. The important part is simply making decisions and not being entirely wrong. Because making an okay decision is a lot better than making no decision at all.

5) What have you learned about working in teams?: When you're working in teams with people you have to worry about workloads. You don't want one person to do all the work while the others have too much free time. That's why the work needs to be split up.

6) If you could do it all over, what would you do differently?: If I could do it all over again, I wouldn't change much. We really did the best we could at each stage of the project. Maybe I would've been more involved with the image processing algorithms, because I fell behind a bit and was playing catch up learning the vision processing algorithms at the end of the term.

C. Ryan Kitchen

1) What technical information did you learn?: Through this project, I learned how to program NVIDIA GPUs to do anything I want them to. I learned a lot about programming optimizations, especially avoiding memory reads and writes, and maximizing the utilization of memory caches. I also learned many new things about video processing, including various methods of edge and object detection, as well as how physical optics such as focus and brightness affect programming computer vision algorithms.

Another thing I learned a lot about was interfacing with proprietary hardware. We used Point Grey cameras for our project, which required us to work around a proprietary hardware interface API with little to no documentation. In order to get some of our stuff to work, we actually had to dig through the header files for the camera API and basically guess and check, since most of the functions did not have any documentation whatsoever. For example, the ability to capture video directly into user buffers was a breakthrough which allowed us to avoid copying data from the CPU to the GPU.

One important technical skill I learned from this project is the ability to combine different libraries that do the same thing. In pursuit of performance, we integrated OpenCV with CUDA, OpenGL with GLUT, and SDL, with all of them being options.

The biggest takeaway for me was learning CUDA, it is an extremely versatile programming interface for GPUs and can be used for numerous other things besides graphics. When I started this project, I barely knew what CUDA was. By the end of the project, I became very experienced with it and am actually teaching others to use it in other projects.

2) What non-technical information did you learn?: As a result of our choice to make our project aerospace themed, I learned a lot about aviation markings, the purpose of different kinds of runway lighting, and the challenges that pilots face landing in inclement weather. We used the official FAA documentation for runway markings and lighting to design our runway model and make it as realistic as possible.

3) What have you learned about project work?: This project taught me the value of persistence and flexibility. Our final product was completely different from how we originally designed it, and we had a number of changes along the way. At several points, we had to completely rewrite major parts of the program to increase performance and decrease complexity. Being able to detach from specific design concepts and explore alternatives has been the most valuable skill I have learned.

4) What have you learned about project management?: In order to produce the best possible product, it was necessary to separate tasks according to individual areas of expertise. By allowing each person to focus on their strengths, we were able to get more done in the same amount of time. This enabled us to exceed our requirements and develop different and diverse skills.

5) What have you learned about working in teams?: One thing I learned from this is the importance of setting objectives and creating minutes before meetings. A lot of our meetings lacked direction and were only marginally productive, as we spent a lot of time getting on tangents and not planning out our work. As a result of this, a lot of our work was very disconnected and disorganized. Towards the end of the year we began planning our meetings out better and this caused us to stay on topic and get a lot more done.

6) If you could do it all over, what would you do differently?: I would have liked to have better divided up the code and created predefined interfaces/function prototypes so that we could have divided up the programming work better. One difficulty we had was continually changing aspects of the program architecture and it caused issues as we were all working separately. If we had explicit interfaces between different parts and well defined goals, we would have been able to be more productive as a team.

Appendix

Essential Code Listings

Unless otherwise specified, all code is in /HawkEyed_Video/HawkEye_2.0.

The following code is the buffered camera system that is used in our final product. The ”Frame” type is a class we designed that maintains the data until it is done being used and then releases it to be captured into again.

```
#include <cuda_runtime.h>
#include "base_module_kernels.h"
#include "/home/ubuntu/HawkEyedVideo/include/FlyCapture2.h"
#include <iostream>
```

```
#include <unistd.h>
#include <pthread.h>
#include "frame.h"
#include "camera_2.0.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

using namespace FlyCapture2;
using namespace std;

#ifndef BUFFER_SIZE
#define BUFFER_SIZE 10
#endif


#define FPS_MAX 60

/// keeps Flycapture from releasing the buffer until the image has been processed


void *capture(void*);

class PGCamera{
private:
  int current_frame;
  int current_output;
  Camera camera;
  Frame frames[BUFFER_SIZE];
  unsigned char *input_buffer;
  unsigned char *input_buffer_d;
  Image rawImages[BUFFER_SIZE];
  PGRGuid guid;
  bool firstrun;
  bool monoOutput,rgbOutput;
  CameraType camType;
  char *name;

  pthread_mutex_t mutex;
  pthread_t thread;

  void fail(){
     cout << "Camera " << name << " failed!" << endl;
     delete this;
```

```
        raise(SIGINT);
    }


    bool running;

public:
    static int camera_no;
    static BusManager busMgr;
    volatile int framesready;

    void setup(){
        framesready=0;
        current_frame=0;
        current_output=0;
        mutex=PTHREAD_MUTEX_INITIALIZER;
      // set up user buffer to recieve data
      cudaHostAlloc((void**)&input_buffer, 1920*1080*sizeof(char)*BUFFER_SIZE,
          cudaHostAllocMapped);
      cudaHostGetDevicePointer((void**)&input_buffer_d,(void*)input_buffer,0);

      // tell frames not to destroy memory when done
      // create CUDA stream for each frame
      for(int i=0;i<BUFFER_SIZE;i++) {
         frames[i].disableFree();
      }
      // activate camera


      Error error = camera.Connect(&guid);
      if(error != PGRERROR_OK){
        cout << "Failed to connect to camera " << name << "!" << endl;
        fail();
      }

      //reset camera
    //  camera.WriteRegister(0x610,0);
    //  camera.WriteRegister(0x610,1);


      //set up user buffers
      if((camera.SetUserBuffers(input_buffer,1920*1080*sizeof(char),BUFFER_SIZE))!=
          PGRERROR_OK){
          fprintf(stderr,"Failed to setup buffers\n");
    fail();
      }


      // get camera info and print it out
      CameraInfo camInfo;
      camera.GetCameraInfo(&camInfo);
      cout << camInfo.vendorName << " " << camInfo.modelName << " "
                << camInfo.serialNumber << endl;

      // set up camera output
      Mode k_fmt7Mode=MODE_0;
      //if(camType==MONO) k_fmt7Mode= MODE_0;
      //else k_fmt7Mode=MODE_7;
      const PixelFormat k_fmt7PixFmt = PIXEL_FORMAT_RAW8;
      Format7Info fmt7Info;
      bool supported;
      fmt7Info.mode = k_fmt7Mode;
      error = camera.GetFormat7Info(&fmt7Info, &supported);
      Format7ImageSettings fmt7ImageSettings;
      fmt7ImageSettings.mode = k_fmt7Mode;
      fmt7ImageSettings.offsetX = (fmt7Info.maxWidth - 1920) / 2;
      fmt7ImageSettings.offsetY = (fmt7Info.maxHeight - 1080) / 2;
      fmt7ImageSettings.width = 1920;
      fmt7ImageSettings.height = 1080;
      fmt7ImageSettings.pixelFormat = k_fmt7PixFmt;
      bool valid;
      Format7PacketInfo fmt7PacketInfo;

      // Validate the settings to make sure that they are valid
      error = camera.ValidateFormat7Settings(&fmt7ImageSettings, &valid,
                                              &fmt7PacketInfo);
      if (error != PGRERROR_OK) {
        cout << "Invalid camera settings" << endl;
        fail();
      }

      error = camera.SetFormat7Configuration(
          &fmt7ImageSettings, fmt7PacketInfo.recommendedBytesPerPacket);
      if(error != PGRERROR_OK){
        cout << "Failed to set format!" << endl;
        fail();
      }

      //camera.WriteRegister(0x968,60,1);
      /*Property myFrameRate;
      myFrameRate.type = FRAME_RATE;
      myFrameRate.absValue = 60.0;
      camera.SetProperty(&myFrameRate);
      */
      error = camera.StartCapture();
      if(error!=PGRERROR_OK){
        cout << "Failed to start image capture!" << endl;
        fail();
      }

      Property frmRate;
      frmRate.type = FRAME_RATE;
      error = camera.GetProperty(&frmRate);
      if (error != PGRERROR_OK) {
        // PrintError( error );
        exit(-1);
      }
      cout << "Maximum frame rate is " << fixed << frmRate.absValue << " fps" << endl;
      running=true;
      usleep(1000);
      pthread_create(&thread,0,capture,(void*)this);
    }


    bool isRunning(){
      bool var;
      pthread_mutex_lock(&mutex);
      var=running;
      pthread_mutex_unlock(&mutex);
      return var;
    }

    ~PGCamera(){
      for(int i=0;i<BUFFER_SIZE;i++){
        frames[i].enableFree();
      }
      pthread_mutex_lock(&mutex);
      running=false;
      pthread_mutex_unlock(&mutex);
      int x,*xx=&x;
      pthread_join(thread,(void**)&xx);
      cudaFreeHost(input_buffer);
      camera.StopCapture();

      camera.Disconnect();
    }
  public:

    PGCamera(const char *newname){
      camType=RGB;
      unsigned int cameras_connected;
      busMgr.GetNumOfCameras(&cameras_connected);
      if(camera_no + 1 > cameras_connected) {
        if(camera_no==100) printf("Must use GUIDs for ALL connected cameras or NONE\n");
        else printf("No camera %d\n", camera_no);
        raise(SIGINT);
      }
      busMgr.GetCameraFromIndex(camera_no,&guid);
      camera_no++;
      name = (char*)malloc(sizeof(char)*strlen(newname)+1);
      strcpy(name,newname);
      setup();
    }

    PGCamera(const char *newname,const char *filename){
      camType=RGB;
      int fd = open(filename,O_RDONLY);
      if(fd==-1) {
        printf("No file %s\n",filename);
        raise(SIGINT);
      }
      read(fd,(void*)&guid,16);
      close(fd);

      name = (char*)malloc(sizeof(char)*strlen(newname)+1);
      strcpy(name,newname);
      camera_no = 100;
      setup();
    }

    PGCamera(const char *newname,const char *filename,CameraType ct){
      int fd = open(filename,O_RDONLY);
      read(fd,(void*)&guid,16);
      close(fd);

      name = (char*)malloc(sizeof(char)*strlen(newname)+1);
      strcpy(name,newname);
      camera_no = 100;
      camType=ct;
      setup();
```

```

}


bool readyToOutput(){
  bool ret;
  pthread_mutex_lock(&mutex);
  if(framesready>0) ret=true;
  else ret=false;
  pthread_mutex_unlock(&mutex);
  return ret;
}

Frame *grabFrame(){
  while(!readyToOutput())usleep(1000);
  int output_frame = current_output;
  current_output = ++current_output % BUFFER_SIZE;
  pthread_mutex_lock(&mutex);
  framesready--;
  pthread_mutex_unlock(&mutex);
  return &(frames[output_frame]);
}

bool readyToCapture(){
  bool ret;
  pthread_mutex_lock(&mutex);
  if(framesready+1<BUFFER_SIZE) ret=true;
  else ret=false;
  pthread_mutex_unlock(&mutex);
  return ret;
}


void capture_loop(){
   clock_t clocks_per_frame = CLOCKS_PER_SEC/FPS_MAX;
   Error error;

   clock_t start_time,stop_time;
   while(isRunning()){

     while(framesready+1>=BUFFER_SIZE) usleep(1000);


    start_time=clock();
    frames[current_frame].synchronize();
```

```
      frames[current_frame].start();
      Image rawImage;
      error = camera.RetrieveBuffer(&rawImage);
      if(error!=PGRERROR_OK) cout << "error";
      // get offset and apply to device buffer
      void *offset_d =(void*)(input_buffer_d+(rawImage.GetData()-input_buffer));
      frames[current_frame].start();
      frames[current_frame].getFromRAW(offset_d,1920,1080,camType);

      current_frame = ++(current_frame) % BUFFER_SIZE;
      pthread_mutex_lock(&mutex);
      framesready++;
      pthread_mutex_unlock(&mutex);

      stop_time = clock()-start_time;
      //cout<<stop_time << "us" << endl;
      if(stop_time<clocks_per_frame) {
  //cout << "need to sleep " << clocks_per_frame-stop_time << " cycles" << endl;
  //usleep(clocks_per_frame-stop_time);


      }
    }
  }

};

void *capture(void *th){
    PGCamera *thiss=(PGCamera*)th;
    thiss->capture_loop();
};

int PGCamera::camera_no=0;
BusManager PGCamera::busMgr;

PGCamera *newCamera(const char *name){
   return new PGCamera(name);
}

PGCamera *newCamera(const char *name, const char *filename){
  return new PGCamera(name,filename);
}

PGCamera *newMonoCamera(const char *name, const char *filename){
  return new PGCamera(name,filename,MONO);
```

```
}

void deleteCamera(PGCamera *cam){
  delete cam;
}

Frame *grabFrame(PGCamera *cam){
  return cam->grabFrame();
}
```

This code is the edge detection algorithm used in the demo. It uses a pixel and 3 of its neighbors to determine the direction and magnitude of a change in brightness. This example shows how to develop a CUDA kernel and some of the optimizations we used in order to reduce latency.

```
#include "edge_overlay.h"
#include <cuda_runtime.h>
#define BPP 4


texture<unsigned char, 2> tex;


static __device__ float getXY(int x, int y){
  //return (float)input[y*1920+x];
  return (float)tex2D(tex,x,y);
}

__global__ void _edge_overlay(Pixel *output, Pixel threshold){
  int x = blockIdx.x*blockDim.x+threadIdx.x;
  int y_start = threadIdx.y*540;
  float a,b,c,d;
  for(int i=0;i<540;i++){
    int y=y_start+i;
    int idx = ((y)*1920+x)*BPP;
    if(!(x<2 || x>=1918 || y>=1078 || y<2)){

  float a=getXY(x,y);
  float b=getXY(x+1,y);

      c=getXY(x,y+1);
      d=getXY(x+1,y+1);

      float gx = (b+d)-(a+c);
      float gy = (c+d)-(a+b);
      Pixel mag = Pixel(sqrt(gx*gx+gy*gy)/2.0);
```

```
        if(mag>threshold){
    output[idx] = 0;
    output[idx+1] = 255;
    output[idx+2] = 0;
        }
      }
    }
}

void edgeOverlay(Pixel *output, Pixel *input, Pixel threshold, cudaStream_t stream){
  size_t offset;
  cudaChannelFormatDesc channeldesc=cudaCreateChannelDesc<unsigned char>();
  cudaBindTexture2D(&offset,tex,input,channeldesc,1920,1080,1920);

  dim3 threads(128,2);
  dim3 blocks(1920/128);
  _edge_overlay<<<blocks,threads,0,stream>>>(output,threshold);
  cudaUnbindTexture(tex);
}
```

This code is the main program for our demonstration application, which was used for the benchmarking and at expo. It shows how different kernels can be strung together via streams, and the operations changed during program execution.

```

#include <GL/glew.h>
#include <GL/freeglut.h>

#include <cuda_runtime.h>
#include <cuda_gl_interop.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

//#include "base_module_kernels.h"

#include <helper_functions.h> // includes for SDK helper functions
#include <helper_cuda.h>      // includes for cuda initialization and error checking
#include "camera_2.0.h"
#include "demosaic.h"
#include "threshold.h"
#include "newfilters/edge_overlay.h"
#include "newfilters/scale_overlay.h"
#include "newfilters/digitalZoom.h"
```

```
#include <signal.h>

#include "frame.h"

//
// Cuda example code that implements the Sobel edge detection
// filter. This code works for 8-bit monochrome images.
//

int camno=1;
typedef unsigned char Pixel;
Pixel   edge_thresh=15;
PGCamera *cam1,*cam2,*cam3;
Pixel *rgbImage,*monoImage;
Pixel *overlayImage,*zoomImage;


void cleanup(void);
void initializeData() ;



/*
int lineShowMode=0;
line lines[4];
bool lines_set[4];
int current_line=0;
*/
#define REFRESH_DELAY     1000/60 //1000/60 //ms


// Code to handle Auto verification
int fpsCount = 0;       // FPS count for averaging
int fpsLimit = 8;       // FPS limit for sampling
unsigned int frameCount = 0;
StopWatchInterface *timer = NULL;
unsigned int g_Bpp;

// Display Data
static GLuint pbo_buffer = 0;  // Front and back CA buffers
struct cudaGraphicsResource *cuda_pbo_resource; // CUDA Graphics Resource (to transfer PBO)

static GLuint texid = 0;       // Texture for display
unsigned char *pixels = NULL;  // Image pixel data on the host

```

```

#define OFFSET(i) ((char *)NULL + (i))

void computeFPS()
{
    frameCount++;
    fpsCount++;

    if (fpsCount == fpsLimit)
    {
        char fps[256];
        float ifps = 1.f / (sdkGetAverageTimerValue(&timer) / 1000.f);

        //glutSetWindowTitle(fps);
        std::cout << ifps <<'\n';
        fflush(stdout);
        fpsCount = 0;

        sdkResetTimer(&timer);
    }
}


bool filterOn=false;
bool zoomPreview=false;

int zoomPix=1920/2;
Pixel threshold=40;
Frame *f=0;
Frame *irf=0;
// This is the normal display path
void display(void)
{
    sdkStartTimer(&timer);
    Frame *f2;
    Pixel *output;

    Pixel *data = NULL;

    // map PBO to get CUDA device pointer
    checkCudaErrors(cudaGraphicsMapResources(1, &cuda_pbo_resource, 0));
    size_t num_bytes;
    checkCudaErrors(cudaGraphicsResourceGetMappedPointer((void **)&data, &num_bytes,
        cuda_pbo_resource));

    rgbImage = data;

    switch(camno){
      case 1:
  f=grabFrame(cam1);
  if(filterOn) demosaic_GBRG(rgbImage,monoImage,(Pixel*)f->get_mono_d(),0,*(f->getStream())
      );
  else demosaic_GBRG(rgbImage,0,(Pixel*)f->get_mono_d(),0,*(f->getStream()));

  if(zoomPreview){
    f2=grabFrame(cam2);
    demosaic_GBRG(zoomImage,0,(Pixel*)f2->get_mono_d(),1,*(f2->getStream()));
    int x = (1920-zoomPix)/2;
    int y = (1080-int((1080.f*(float)zoomPix/1920.f)))/2;
    digitalZoom(overlayImage,zoomImage,x,y,zoomPix,*(f2->getStream()));

  }
  if(filterOn) edgeOverlay(rgbImage,monoImage,edge_thresh,*(f->getStream()));

  if(zoomPreview){
    f2->synchronize();
    scaleOverlay(rgbImage,overlayImage,1200,500,712,*(f->getStream()));
  }
      break;
      case 2:
  f=grabFrame(cam3);
  thresholdFilter(rgbImage,(Pixel*)f->get_mono_d(),threshold,*(f->getStream()));
      break;
      case 3:
  f=grabFrame(cam2);
  demosaic_GBRG(overlayImage,monoImage,(Pixel*)f->get_mono_d(),1,*(f->getStream()));
  if(filterOn) edgeOverlay(overlayImage,monoImage,edge_thresh,*(f->getStream()));

  int x = (1920-zoomPix)/2;
  int y = (1080-int((1080.f*(float)zoomPix/1920.f)))/2;

  digitalZoom(rgbImage,overlayImage,x,y,zoomPix,*(f->getStream()));
      break;
    }


    //cudaMemcpy((void*)data,(void*)rgbImage,1920*1080*4,cudaMemcpyDeviceToDevice);

    f->synchronize();


    //if(sobelize) g_SobelDisplayMode = SOBELDISPLAY_SOBELTEX;
    checkCudaErrors(cudaGraphicsUnmapResources(1, &cuda_pbo_resource, 0));
    //comboFilter(frame1,frame2);


    glClear(GL_COLOR_BUFFER_BIT);

    glBindTexture(GL_TEXTURE_2D, texid);
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pbo_buffer);
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 1920, 1080,
                    GL_RGBA, GL_UNSIGNED_BYTE, OFFSET(0));
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);

    glDisable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    glBegin(GL_QUADS);
    glVertex2f(0, 0);
    glTexCoord2f(0, 0);
    glVertex2f(0, 1);
    glTexCoord2f(1, 0);
    glVertex2f(1, 1);
    glTexCoord2f(1, 1);
    glVertex2f(1, 0);
    glTexCoord2f(0, 1);
    glEnd();
    glBindTexture(GL_TEXTURE_2D, 0);
    glutSwapBuffers();

    sdkStopTimer(&timer);

    computeFPS();
}

void timerEvent(int value)
{
    if(glutGetWindow())
    {
        glutPostRedisplay();
        glutTimerFunc(REFRESH_DELAY, timerEvent, 0);
    }
}


void reshape(int x, int y)
{
    glViewport(0, 0, x, y);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 1, 0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void interrupt(int sig){
    cleanup();
}
void cleanup(void)
{
    cudaGraphicsUnregisterResource(cuda_pbo_resource);

    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
    glDeleteBuffers(1, &pbo_buffer);
    glDeleteTextures(1, &texid);
    deleteCamera(cam1);
    deleteCamera(cam2);
    deleteCamera(cam3);

    sdkDeleteTimer(&timer);

    cudaDeviceReset();
}

void initializeData()
{
    GLint bsize;
    unsigned int w, h;
    int width=1920;
    int height=1080;
    int channels=4;
    pixels= (unsigned char *) malloc(sizeof(unsigned char) * width * height *channels);
    w = width;
    h = height;
    g_Bpp = 4;

    //deleteTexture();
    memset(pixels, 0x0, g_Bpp * sizeof(Pixel) * 1920 * 1080);

    glGenBuffers(1, &pbo_buffer);
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pbo_buffer);
    glBufferData(GL_PIXEL_UNPACK_BUFFER,
                 g_Bpp * sizeof(Pixel) * 1920 * 1080,
                 pixels, GL_STREAM_DRAW);

    glGetBufferParameteriv(GL_PIXEL_UNPACK_BUFFER, GL_BUFFER_SIZE, &bsize);

    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);

    // register this buffer object with CUDA
    checkCudaErrors(cudaGraphicsGLRegisterBuffer(&cuda_pbo_resource, pbo_buffer,
        cudaGraphicsMapFlagsWriteDiscard));

    glGenTextures(1, &texid);
    glBindTexture(GL_TEXTURE_2D, texid);
    glTexImage2D(GL_TEXTURE_2D, 0, ((g_Bpp==1) ? GL_LUMINANCE : GL_RGBA),
                 1920, 1080,  0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
    glBindTexture(GL_TEXTURE_2D, 0);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glPixelStorei(GL_PACK_ALIGNMENT, 1);

}


void initGL(int *argc, char **argv)
{
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowSize(1920,1080);
    glutCreateWindow("CUDA Edge Detection");
    glutFullScreen();

    glewInit();
}


void keyboard(unsigned char key, int /*x*/, int /*y*/){

  switch(key){
     case 'q': raise(SIGINT);
```

```
      case '.':
      case '>': threshold++;
       break;
      case ',':
      case '<': threshold--;
       break;

      case '1': camno=1;
      break;
      case '2': camno=2;
      break;
      case '3': camno=3;
      break;

      case 'f': filterOn=true;
      break;
      case 'g': filterOn=false;
      break;
      case 'h': edge_thresh--;
      break;
      case 'j': edge_thresh++;
      break;


      case 'a': threshold--;
      break;
      case 's': threshold++;
      break;

      case '[': zoomPix+=5;
      break;
      case ']': zoomPix-=5;
      break;


      case 'z': zoomPreview=true;
      break;
      case 'x': zoomPreview=false;
      break;
   }
   printf("%d\n",threshold);
}


/*
```

```
void mouse_move(int x, int y){
  lines[current_line].x2=x;
  lines[current_line].y2=y;
  lines[current_line].setup();
  lineShowMode=1;
}
  */

int main(int argc, char **argv)
{


    //setup();

    initGL(&argc, argv);
    cudaGLSetGLDevice(gpuGetMaxGflopsDeviceId());
    //cudaMalloc((void**)&gradout,1920*1080*sizeof(uchar2));
    cudaMalloc((void**)&rgbImage,1920*1080*sizeof(char)*4);
    cudaMalloc((void**)&monoImage,1920*1080*sizeof(char));
    cudaMalloc((void**)&overlayImage,1920*1080*sizeof(char)*4);
    cudaMalloc((void**)&zoomImage,1920*1080*sizeof(char)*4);
    //ircam = newMonoCamera("bob","41C6NIR.pgrguid");
    cam1 = newMonoCamera("bob","41C6C.pgrguid");
    cam2 = newMonoCamera("larry","23S6C.pgrguid");
    cam3 = newMonoCamera("sally","41C6NIR.pgrguid");
    // cam2 = newMonoCamera("larry","41C6NIR.pgrguid");
   // cam3 = newCamera("sally","23S6C.pgrguid");
    //cam = newCamera("larry","23S6C.pgrguid");
    //if(argc>1) cam=newMonoCamera("bob",argv[1]);
    //else cam = newCamera("bob");
    sdkCreateTimer(&timer);
    sdkResetTimer(&timer);

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    //glutMouseFunc(mouse);
    //glutMotionFunc(mouse_move);
    glutReshapeFunc(reshape);

    initializeData();

    glutCloseFunc(cleanup);

    glutTimerFunc(REFRESH_DELAY, timerEvent,0);
    glutMainLoop();
    signal(SIGINT,interrupt);
}
```

This code is our pipeline system, which was developed and used to achieve even lower latency for our software, but was not included in the final version because of stability issues, but would be the preferred method for future development. It demonstrates a more effective way to process multiple frames at the same time, and optimizes to only grab the most recent frames instead of processing all frames in order. It allows the programmer to avoid directly having to implement multithreaded, multi-stream processing, and instead focus on just the algorithm development.

```
#ifndef HAWK_PIPELINE
#define HAWK_PIPELINE

typedef unsigned char Pixel;
#include <pthread.h>
#include <cuda_runtime.h>
#include <unistd.h>

#define MAX_PIPELINES 5
#define BUFFER_SIZE 10
#define MAX_CAMERAS 3
#define START_DELAY 1000

void *start_pipeline(void *);

class HawkEye_Pipeline{
protected:
  Pixel *buffers[BUFFER_SIZE];
  volatile bool running;
  volatile int nextBuffer;
  volatile int latestOutput;
  int lastOutput;
  pthread_mutex_t buffer_mutex;
  pthread_mutex_t output_mutex;
  pthread_mutex_t input_mutex;
  pthread_mutex_t ready_mutex;
  int num_pipelines;
  volatile int current_buffers[MAX_PIPELINES];
  pthread_t threads[MAX_PIPELINES];
  cudaStream_t streams[MAX_PIPELINES];
public:
  int add_pipeline(){
    if(num_pipelines==MAX_PIPELINES) return -1;
    pthread_create(&(threads[num_pipelines]),0,start_pipeline,(void*)this);
    usleep(START_DELAY);
    num_pipelines++;
  }

  void setup(){

    add_pipeline();
  };

  HawkEye_Pipeline(){
    num_pipelines=0;
    running=true;
    nextBuffer=0;
    latestOutput=-1;
    lastOutput=-1;
    buffer_mutex=PTHREAD_MUTEX_INITIALIZER;
    output_mutex=PTHREAD_MUTEX_INITIALIZER;
    input_mutex=PTHREAD_MUTEX_INITIALIZER;
    ready_mutex=PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_lock(&ready_mutex);
    for(int i=0;i<BUFFER_SIZE;i++){
      cudaMalloc((void**)&(buffers[i]),1920*1080*sizeof(char)*3);
    }
    for(int i=0;i<MAX_PIPELINES;i++){
      current_buffers[i]=-1;
      cudaStreamCreate(&streams[MAX_PIPELINES]);
    }
  }


  int getBuffer(){
    pthread_mutex_lock(&buffer_mutex);
    int op = nextBuffer;



    nextBuffer = (++nextBuffer)%BUFFER_SIZE;
    bool ok=true;
    do{
      if(nextBuffer==lastOutput) ok=false;
      for(int i=0;i<num_pipelines;i++)
  if(nextBuffer==current_buffers[i]) ok=false;
      if(!ok){
  nextBuffer = (++nextBuffer)%BUFFER_SIZE;
      }
    }while(!ok);
    pthread_mutex_unlock(&buffer_mutex);
    return op;
  }

  void setLatestOutput(int n){
    pthread_mutex_lock(&output_mutex);
    latestOutput=n;
    pthread_mutex_unlock(&output_mutex);
    pthread_mutex_unlock(&ready_mutex);
  }

  virtual void pipeline(Pixel *output,int pipeline_no){
    printf("FUCK");;
  }

  void _pipeline(){
    int pipeline_no = num_pipelines;
    while(running){
      current_buffers[pipeline_no]=-1;
      int buf=getBuffer();
      current_buffers[pipeline_no]=buf;
      /// do stuff
      this->pipeline(buffers[buf],pipeline_no);
      cudaStreamSynchronize(streams[pipeline_no]);
      // output to buffers
      setLatestOutput(buf);

    }
    pthread_exit(0);
  }

  Pixel *getLatestOutput(){
    pthread_mutex_lock(&ready_mutex);
    int buf = latestOutput;
    lastOutput=buf;
    return buffers[buf];
  }
};


void *start_pipeline(void *pip){
  ((HawkEye_Pipeline*)pip)->_pipeline();
}



```

```
#endif
```

Appendix

Hardware and Output Image Examples