The Report Committee for Karl Robert Arndt
Certifies that this is the approved version of the following report:

Evaluation of a Mobile Computing Platform for Image Processing

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Adnan Aziz

Andreas Gerstlauer

Evaluation of a Mobile Computing Platform for Image Processing

by

Karl Robert Arndt, B.S. EE

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2010

Dedication

This report is dedicated to my wife Haley, without whose unwavering support and sacrifice this would not have been possible.

Acknowledgements

I would like to acknowledge Dr. Adnan Aziz for supervising this report, and truly going above and beyond in doing so. His guidance was invaluable at every turn. I would also like to thank Dr. Andreas Gerstlauer for his guidance early in the project and for reading and approving the final version.

December 2, 2010

Abstract

Evaluation of a Mobile Computing Platform for Image Processing

Karl Robert Arndt, M.S.E.

The University of Texas at Austin, 2010

Supervisor: Adnan Aziz

Many modern mobile applications, such as Unmanned Aerial Vehicles (UAVs), require sophisticated processing capability with low power consumption in a small form factor. UAVs, for example, may require a platform capable of controlling a camera, performing digital signal processing techniques on the pictures to detect faces or motion, and guiding the vehicle based on decisions made from the processed data. Additionally, since the vehicle is mobile and aerial, its effectiveness is heavily dependent on the size and power consumption of the platform. In this report, we explore this set of requirements and how well they are met with a Texas Instruments OMAP SoC on a BeagleBoard. Specifically, we report on the computational performance and power drawn by the OMAP General Purpose Processor (GPP) when performing a facial detection algorithm with OpenCV. We also analyze the performance enhancement possible by offloading the facial detection algorithm to the OMAP DSP coprocessor. In

summary we find that the Beagleboard would be an appropriate platform for a simpler UAV capable of pre-processing still images taken every few seconds, but not for processing video data real-time. We conclude by describing other applications that are suitable for the Beagleboard.

Table of Contents

List of Tables

List of Figures

List of Illustrations

Chapter 1: Context

The use of Unmanned Aerial Vehicles (UAVs) by the U.S. Military has seen significant growth in the past decade, and this growth is expected to continue well into the future [14]. One of the primary objectives of UAV missions is to gather still image and video data, every second of which is processed by human analysts [15]. The time and personnel investment required to accomplish this task is already significant, and with the anticipated growth of UAVs, it could become overwhelming. A UAV platform capable of performing on-board complex image and video analysis, such as facial detection, could relieve the human analysts of a portion of the time-consuming processing by performing some pre-processing of the image and video data. Additionally, missions could be made more effective if the UAV was able to make flight decisions based on its own real-time analysis of image and video data, possibly eliminating the need for human virtual "pilots" to guide the UAV.

GOAL OF REPORT

In this report, we attempt to analyze one particular hardware platform, the Beagleboard with Texas Instruments OMAP3530 System-on-Chip (SoC), for its suitability as an onboard image and video processing platform for a UAV. We use the OpenCV Computer Vision library to perform facial detection image processing on the OMAP General Purpose Processor (GPP), and benchmark that routine for both speed and power consumption. We then consider the speed improvements potentially made possible by .porting the facial detection algorithm to the OMAP Digital Signal Processor (DSP), and weigh them against the communication overhead and power consumption tradeoffs. Finally, we report on the overall suitability of the Beagleboard as a UAV platform in light of the results.

BEAGLEBOARD

The Beagleboard is ideally suited as a test platform for the development and analysis of an onboard image processing platform. The core of the Beagleboard is a Texas Instruments OMAP3530 SoC, which integrates an ARM Cortex A8 General Purpose Processor (GPP) running at 500MHz and a TMS320C64x+ Digital Signal Processor (DSP) running at 360MHz on a single chip [2]. This combination of GPP and DSP on a single chip makes the OMAP chip uniquely suited to the UAV application, as it can run a traditional operating system such as Linux for control processes (flight control, camera control, etc.) as well as complex image processing routines in a single compact footprint. The TI OMAP SoC is intended for use as a mobile applications processor [4], and can be found in many mobile phones, including the popular Motorola Droid. Accordingly, it has been optimized for low power consumption, which meets an important requirement for a UAV application, since they are required to run on battery power for long distances. Measuring 3" x 3.1" x 0.75" and weighing approximately 37 grams, the Beagleboard is also relatively small and light.

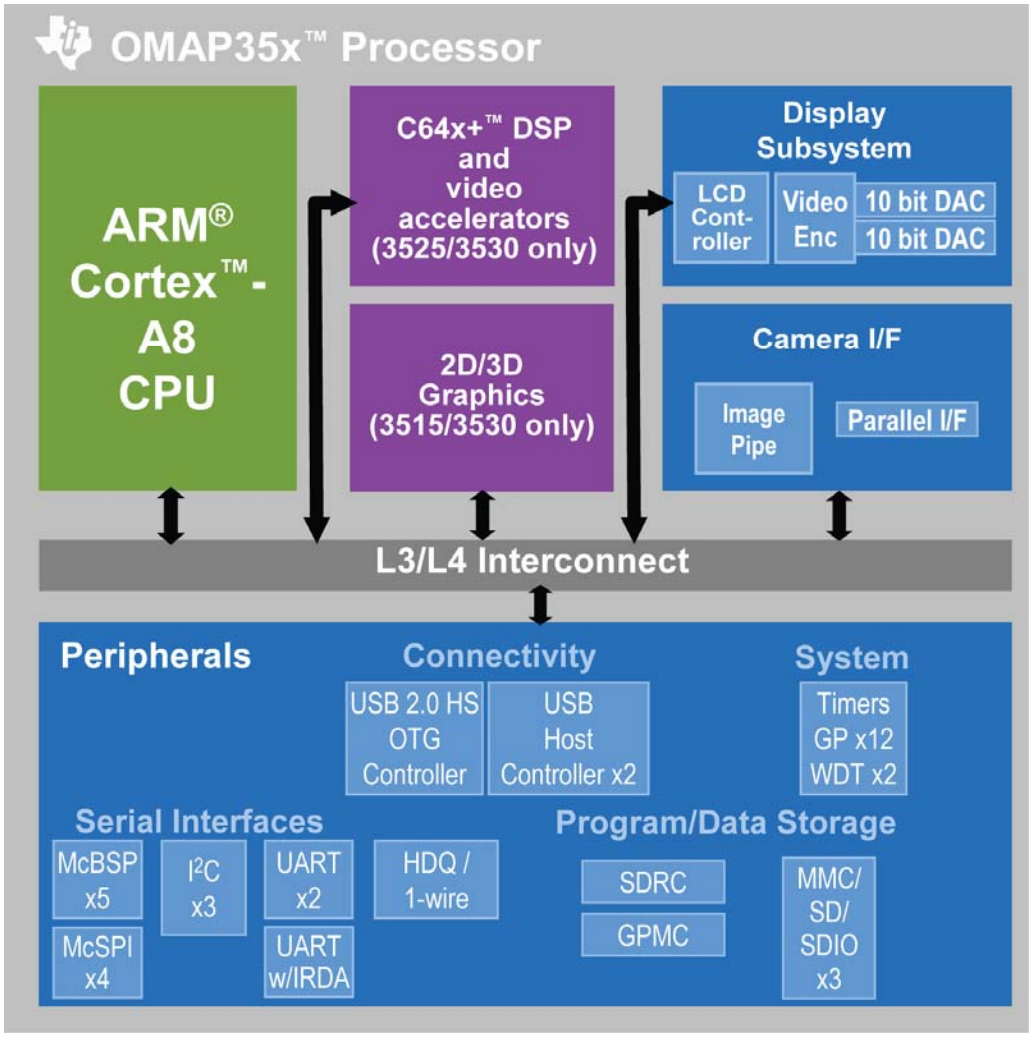

Figure 1: OMAP35x Block Diagram

The OMAP3530 package allows memory to be physically stacked on top of the chip in a Package-on-Package (PoP) configuration, further reducing board footprint. The Beagleboard OMAP utilizes PoP to include 256MB of DDR SRAM and 256MB of NAND Flash ROM for storing a bootloader. The Beagleboard also includes an SD memory socket interfaced to the OMAP General Purpose Memory Controller (GPMC) for additional Flash memory, which can used to store an operating system. For this application , I chose Angstrom Linux as the operating system, as it is widely used on the

Beagleboard, has a strong user community, and can be easily customized with an online distribution builder.

Several peripheral communication protocols are supported on-chip for communication both on- and off-board, including:

- 3 UART Channels
- 3 High-Speed I2C Channels
- 3 Multi-Channel BSP Channels
- 3 Multi-Channel SPI Channels
- 1 USB 2.0 Host
- 1 USB 2.0 On-the-Go (OTB)

One of the UART channels is brought out onto the board, level-shifted to RS-232 levels, and connected to a header connector for terminal interface to the GPP Operating System. The remaining UART channels are also brought onto the board, without a level-shifting, and are available for user expansion on the Beagleboard expansion header. All of the I2C channels are brought out onto the board as well, one used to communicate with the TI TPS65950 Power Management chip, and the others routed to the expansion header for user expansion. The USB 2.0 Host port is brought out onto the board and connected to a standard USB Host connector, used in this application to connect a Logitech Pro 9000 digital camera, which will gather images for facial detection processing.

Several additional peripheral devices, not used in our particular application, are available on the Beagleboard, shown in Illustration 1, including S-Video, DVI-D, and audio in and out.

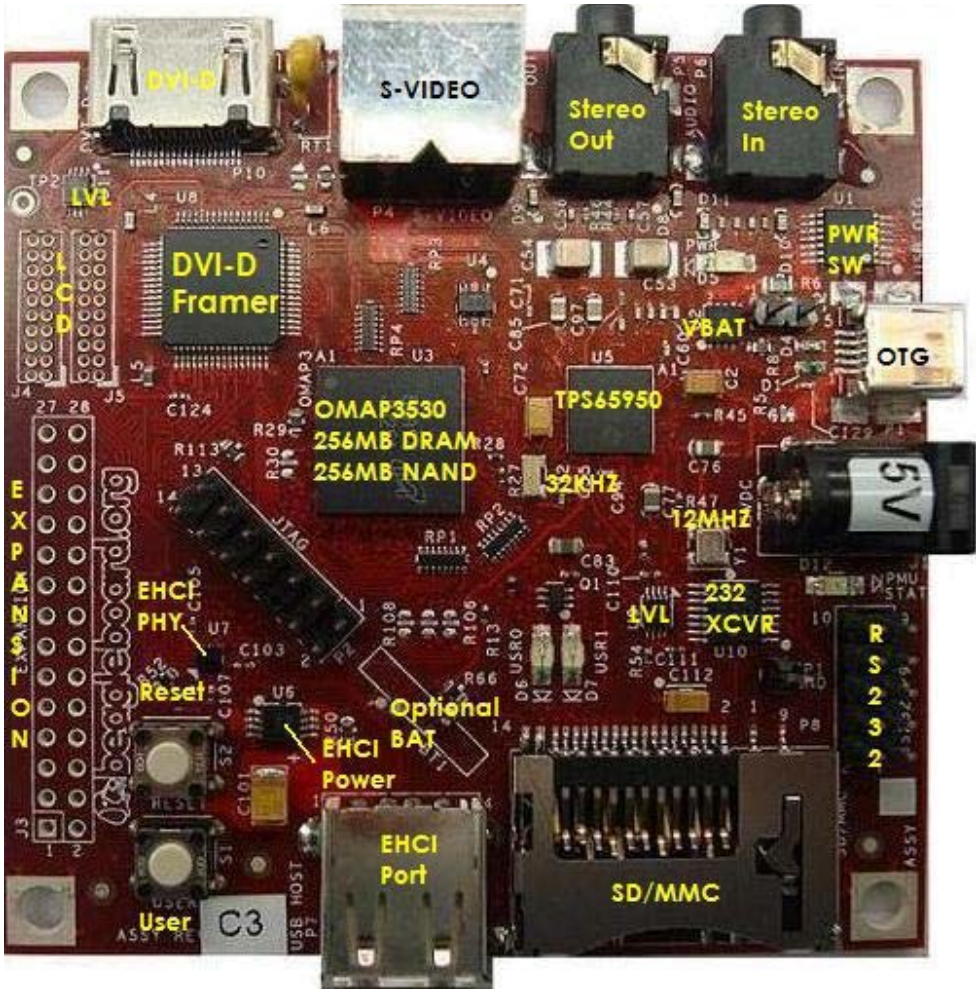

Illustration 1: Beagleboard

Angstrom Linux

I selected the Angstrom distribution of the Linux operating system to run on the ARM GPP. The primary advantages of Angstrom are that is has been ported to the Beagleboard, it has been extensively tested by the Beagleboard community, and that custom filesystems can be generated online with the OpenEmbedded-based Narcissus tool [8]. For this application, I was able to create a custom Angstrom build that contained a kernel configured for Beagleboard and a custom filesystem loaded with pre-compiled

OpenCV and DSPLink libraries. While the Beagleboard is equipped with a DVI-D output, and so can support a GUI Linux interface, I opted for a terminal interface through the serial port due to the embedded nature of the UAV application. The kernel image and filesystem are stored on an SD card, which is plugged into the Beagleboard's SD socket for loading in the boot process.

By default, the Beagleboard boots from the OMAP PoP NAND Flash, which contains the U-Boot bootloader. U-Boot then loads the kernel image from the SD card into the PoP RAM, passes the boot arguments that specify the filesystem location and format and the terminal port and settings. U-Boot then issues the boot command to boot from the memory address at which the kernel image was placed, and the kernel boots to the Angstrom login prompt.

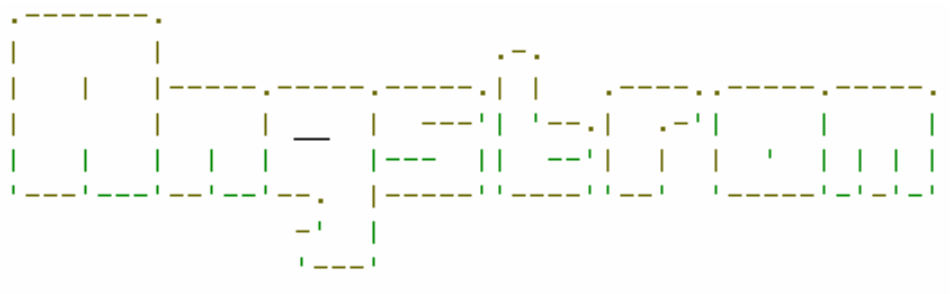

Illustration 2: Angstrom Linux Login Prompt

Texas Instruments DSPLink

The OMAP GPP and DSP communicate using the TI DSP/BIOS Link (DSPLink) Inter-Processor Communications (IPC) software package. The DSPLink libraries allow a GPP application to bootload the DSP with a DSP binary stored in the Linux filesystem,

and then communicate with that DSP process. This DSP binary consists of the DSP/BIOS Operating System and the individual processes that will be run. DSPLink uses a shared memory space and inter-processor interrupts to facilitate the communication between the two processors.

The GPP side of the DSPLink stack consists of an OS Adaptation Layer, a Link Driver, and a Process Manager, controlled through a thin DSPLink API. The DSPLink content on the DSP side consists solely of a Link Driver resident in the DSP/BIOS Operating System. The OS Adaptation Layer separates the DSPLink functionality from the specific GPP Operating System for OS portability. The Link Drivers control GPP-DSP transfers across the physical OMAP interconnect bus. The Process Manager keeps track of the other components of the stack and processes calls from the API through to the Link Driver. Finally, the DSPLink API abstracts the functions of the Process Manager and Link Driver to the user [3].

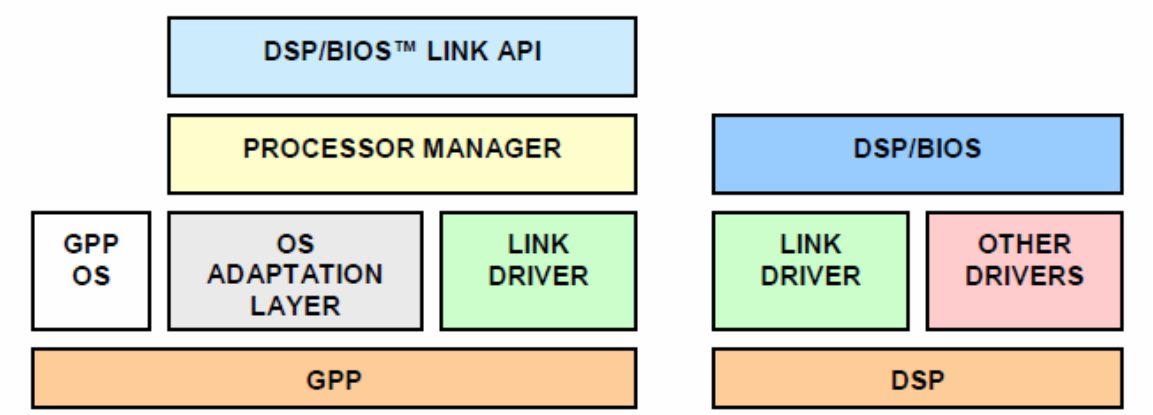

Figure 2: DSPLink Stack

Chapter 2: OpenCV Application

The OpenCV Computer Vision library was used to perform facial detection on still images. We used facial detection as the primary benchmark for speed and power consumption measurements on the GPP, but it is only one example of the power of OpenCV and only one of many potentially useful complex algorithms for UAVs. We applied and benchmarked this algorithm on several images of different resolutions containing different numbers of faces.

OpenCV Overview

Originally developed by Intel, OpenCV is a C library of algorithms geared toward real-time image processing, particularly for Computer Vision applications. The basic unit that is operated on by the algorithms is an image frame stored in a C struct, which may be sourced from a still image or from a single frame of a video stream, so OpenCV handles still images and video streams in the same frame-by-frame manner [1]. We use the facial detection algorithm as both a representative benchmark of OpenCV's capabilities and a potentially valuable feature of a UAV processing platform.

Facial Detection

The facial detection algorithm used by OpenCV was originally proposed by Paul Viola and Rainer Lienhart, and uses a decision tree composed of a cascade of Haar-like feature classifiers to scan through the image and decide if faces are present [10]. A Haar-like feature is generated by calculating the differences in pixel intensities in a certain region of interest. Certain pixel intensity differences can be more or less common to typical facial structures, and thereby be used to decide if the region of interest contain a face or not. For example, a typical face viewed from the front will have two darker

regions for the eyes, separated by some lighter space [12]. A single classifier decides if one of these common characteristics is met. Since many of these Haar-like features would need to be present to decide with a high degree of confidence that a face is present, the individual classifiers are cascaded into a large decision tree that is applied to every region within an image. The classifer cascade is created by training with both positive images (those with faces) and negative images (those without faces) [13]. In this training operation, the individual Haar-like features are determined and a cascaded decision tree formed, based on the similar pixel intensity difference aspects of known faces and in the absence of faces. In this way, the cascade can be customized to certain kinds of faces, faces at a certain angle, etc. This training process is very time consuming, so fortunately a cascade trained to general frontal face detection is provided with OpenCV. The cascade is stored as an .xml file and passed into the facial detection program.

SUITABILITY

OpenCV is well-suited to a mobile UAV platform in general and for the Beagleboard platform in particular because it is designed specifically for real-time image processing, runs in Linux, controls Linux camera drivers within the library calls, and is written in C, allowing it to be relatively easily ported to, among other things, a Digital Signal Processor.

Chapter 3: General Purpose Processor Evaluation

The ARM Cortex A8 GPP was evaluated for speed and power using the facial detection algorithm in OpenCV and a series of general benchmark tests.

BEAGLEBOARD BASELINE

The Beagleboard was baselined for power consumption at several points during the boot-up process and while performing representative general filesystem tasks in order to create comparison data points for the OpenCV and benchmark routine measurements. Current measurements were taken with a multimeter in-line with the Beagleboard 5 VDC power supply. The results reported are, unless otherwise stated, for the maximum current during the reported operation, which are reasonably representative of the average current as well. Aside from very short ramp-up and ramp-down periods, the Beagleboard input current peaked and remained at the reported values for the duration of the operation. Current measurements were taken during the initial boot-up to U-Boot, idle at the U-Boot prompt, idle at the Linux prompt, with Linux performing a filesystem search and with Linux loading a kernel module.

Boot Stage	Current (mA)	Power @ 5VDC (W)
Initial Bootup	450 max, 350 avg	2.25 max, 1.75 avg
Idle at U-Boot Prompt	360	1.8
Idle at Linux Prompt	280	1.4
Linux Filesystem Search	370	1.9
Linux Kernel Module Load	370	1.9

Table 1: Beagleboard Baseline Measurements.

OPENCV FACIAL DETECTION

The OpenCV facial detection algorithm was run on several different images with different resolutions and numbers of faces, with power consumption and run time measurements collected for each. The run time was measured as the time to complete the detect_and_draw routine, which performs the actual facial detection work, since the overhead of reading an image into a IplImage struct and writing the detected image back out to a file could be implemented in different ways in different applications.

Since the UAV application we are considering would receive images and video streams from an onboard camera, we needed to prove that OpenCV could capture frames from a camera connected to the Beagleboard. We chose the Logitech Pro 9000 webcam because it connects to the host with USB and is supported by the Linux uvcvideo driver, which is included in the Angstrom Linux build. I wrote a camera test program that uses the OpenCV libraries to read an image from the Logitech Pro 9000 using uvcvideo, store it in a IplImage struct, and save it back to a .jpg file. This test program also serves to illustrate the basic image frame-handling constructs in OpenCV.

```
#include <opencv/highgui.h>
#include <stdio.h>

int main() {

  int key;
  char saved_filename[] = "webcam_shot.jpg";


  CvCapture* capture = cvCaptureFromCAM( -1 );
  if( !capture ) {
      fprintf( stderr, "ERROR: capture is NULL\n");
      return -1;
  }

  printf("cvCaptureFromCAM was successful!\n");

  IplImage* frame;
  frame = cvQueryFrame( capture );

  if(!cvSaveImage(saved_filename,frame,0))
      printf("Could not save: %s\n",saved_filename);

  cvReleaseCapture( &capture );
  return 0;
}
```

Figure 3: Exerpt from camera_test.c

The cvCaptureFromCAM command accesses the camera with the uvcvideo driver, takes a picture, and stores the image in a CvCapture struct, which is only used for temporarily storing frames received from a Capture function. The image is then loaded into a IplImage struct with cvQueryFrame. Once in the IplImage struct, the frame is ready to be processed by OpenCV algorithms, such as facial detection. The purpose of this example is only to prove that OpenCV can capture images from a USB camera connected to the Beagleboard's USB host connection, so the image is simply saved to a .jpg file and released. The output of this camera test program, webcam_shot.jpg, shows the author at work.

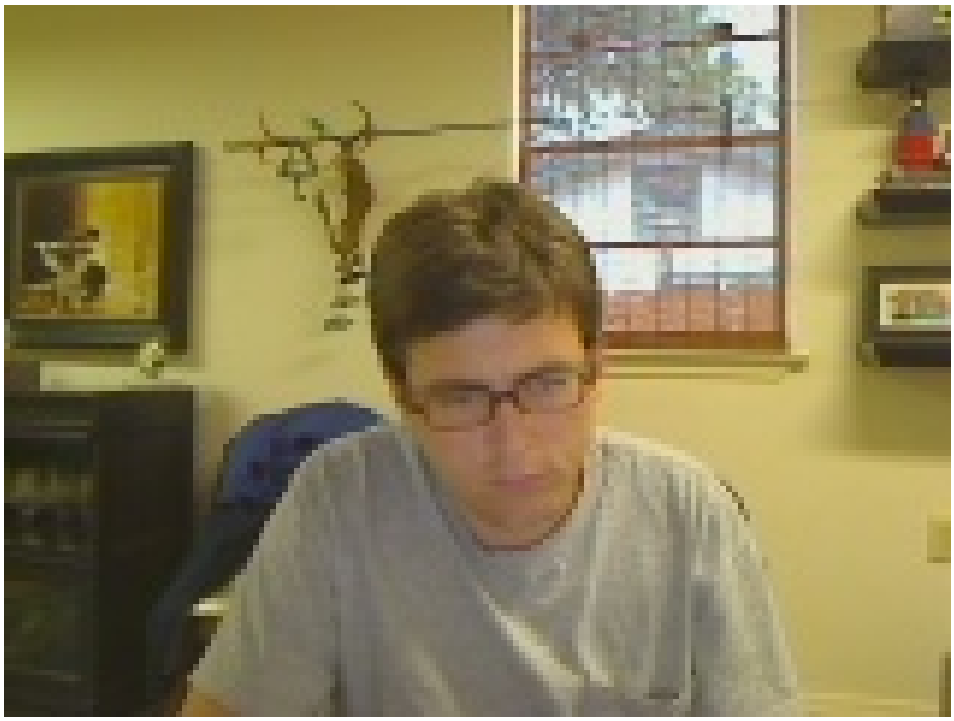

Illustration 3: webcam_shot.jpg – OpenCV Camera Capture Test

Unfortunately, the color format of the image produced by the Logitech camera is not compatible with the OpenCV facial detection algorithm, so it could not be run with images sourced from the camera. Posts from the OpenCV user community suggest that this can be remedied with a patch, but this process is not fully defined and out of the scope of this report. As a workaround to this issue, I utilized various images from Google Images on which to run the facial detection algorithm instead of images from the camera. This approach still demonstrates that OpenCV can gather images from a camera and successfully run a facial detection algorithm on them, as the images are operated on from the IplImage struct either way. The only difference from the camera_test.c code is that the image is loaded from a file with cvLoadImage instead of from the camera with cvCaptureFromCAM. A uvcvideo-compatible USB camera that produces images in a standard RGB color format would be needed to eliminate the need for the workaround.

The face_detect.c program is a slightly modified version of the program provided in the OpenCV Wiki [10]. After verifying the information passed into the program, the image, contained in the IplImage struct frame_copy, is passed to the detect_and_draw function for facial detection. This function is timed and the duration reported.

```
// Call the function to detect and draw the face
start = clock();
detect_and_draw( frame_copy );
end = clock();
elapsed = ( (double) (end - start) ) / CLOCKS_PER_SEC;
printf("detect_and_draw took %3.3f seconds\n",elapsed);
```

Figure 4: Excerpt from face_detect.c

The detect_and_draw function takes the IplImage struct containing the image and passes it and the classifier cascade to cvHaarDetectObjects, which detects and counts the faces. The quantity and locations of the faces detected by cvHaarDetectObjects is then passed to a routine that draws a rectangle around each on the original image. The original code from the OpenCV Wiki then displays this modified image, but since our Beagleboard platform does not use a GUI interface, I modified the code to simply save the modified image to a file.

```
// Function to detect and draw any faces that is present in an image
void detect_and_draw( IplImage* img )
{
    int scale = 1;

    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width/scale,img->height/scale), 8, 3 );

    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;

    // Clear the memory storage which was used before
    cvClearMemStorage( storage );

    // Find whether the cascade is loaded, to find the faces. If yes, then:
    if( cascade )
    {

        // There can be more than one face in an image. So create a growable sequence of faces.
        // Detect the objects and store them in the sequence
        CvSeq* faces = cvHaarDetectObjects( img, cascade, storage,
                                            1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
                                            cvSize(40, 40) );

        // Loop the number of faces found.
        for( i = 0; i < (faces ? faces->total : 0); i++ )
        {
           // Create a new rectangle for drawing the face
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );

            // Find the dimensions of the face,and scale it if necessary
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;

            // Draw the rectangle in the input image
            cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
    }

    // Show the image in the window named "result"
//    cvShowImage( "result", img );
    if(!cvSaveImage(saved_filename, img, 0))
      printf("Could not save: %s\n", saved_filename);

    // Release the temp image created.
    cvReleaseImage( &temp );
}
```

Figure 5: Exerpt from face_detect.c - detect_and_draw

This program was run with several images of differing resolutions and numbers of faces in order to give a broad sampling of performance results. The first image from Google Images, Faces_3.jpg, has a resolution of 501x300 and five faces, all of which were detected and identified by OpenCV. It is interesting to note that a sixth face, apparently that of monkey on one of the individuals' shirts, was identified as well.

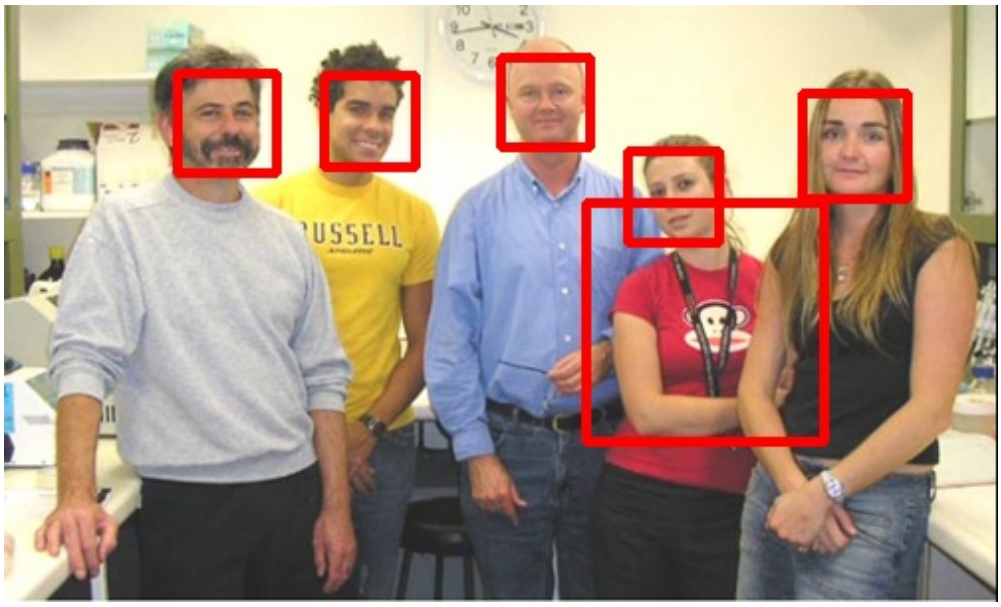

Illustration 4: Faces_3.jpg with Detected Faces

The second image, Faces_2.jpg, has the highest resolution at 600x377, and contains four faces, only one of which was detected by OpenCV. The detection failures in this image are likely due to the relative size of the faces or, more accurately, the number of pixels in each face. It is clear that, while the resolution of this image is higher, the faces are much smaller than in Faces_3. Since the algorithm is based on pixel intensity differences, it is reasonable to assume that, in light of the failures in this image, that the number of pixels per face appears to make a considerable difference in the accuracy.

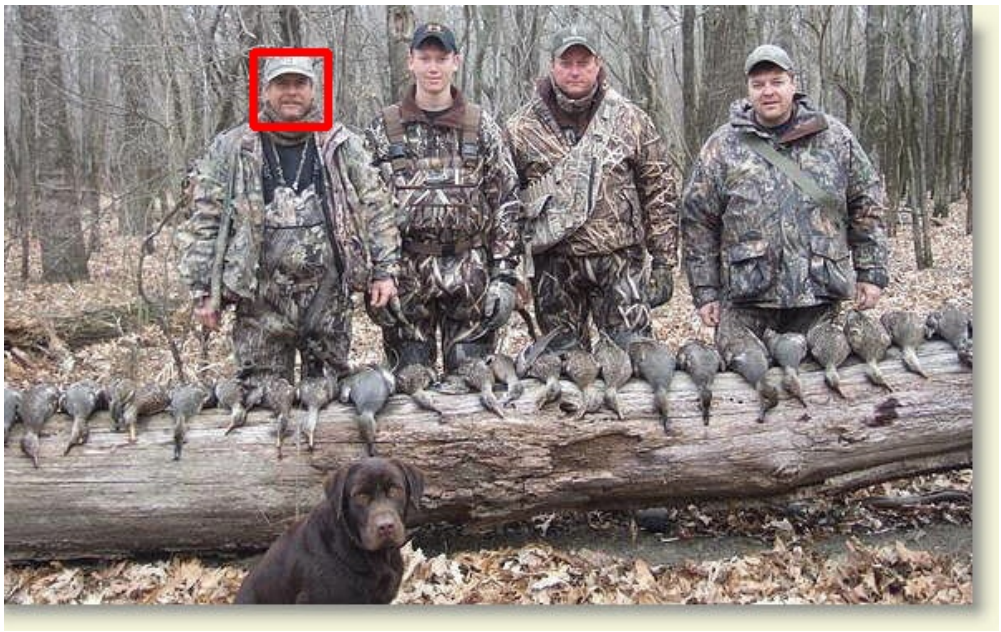

Illustration 5: Faces_2.jpg with Detected Faces

The final image, Faces_1.png, has the smallest resolution, at 450x303, and two faces, none of which were detected by OpenCV. While the resolution is less than that of Faces_2, the individual faces are relatively much bigger, and likely contain more pixels. This may illustrate the fact that the detection algorithm is dependent on the trained classifier cascade used, and will not necessarily detect faces that seem just as obvious as the detected faces to the human eye.

Illustration 6: Faces_1.png

Both run-time and power consumption measurements were taken for each of these images.

OpenCV detect_and_draw routine	Resolution	# of Faces Detected	Power (W)	Run Time (s)
Faces_3.jpg	501x300	6	1.9	2.28
Faces_2.jpg	600x377	1	1.9	3.46
Faces_1.png	450x303	0	1.9	2.06

Table 2: OpenCV Facial Detection on OMAP GPP.

The measurement data shows that the run time of the algorithm is dominated by the resolution of the input image, and not by the number of successful face detections.

Since the results seem to show that the number of pixels per face, whether by overall image resolution or relative sizes of the faces in the image, affect the accuracy of the algorithm, I propose that Faces_3, at 2.28 seconds, represents a relatively good coarse run-time benchmark for accurate facial detection on OpenCV.

BENCHMARKS

Since facial detection is not the only application that could be useful for a UAV platform, we chose a few general benchmarks to give additional insight into the performance of the GPP. These benchmarks include a Fibonacci Sequence, an FIR filter, and a Sort Algorithm, which are meant to serve as a sampling of useful operations that represent a broad spectrum of potential applications. The benchmarks running on the OMAP GPP produced the following results:

Benchmark	Power (W)	Run Time (s)
Fibonacci Sequence – 10,000 Iterations	1.9	9.09
FIR Filter – 10,000,000 Iterations	1.9	38.62
Sort Algorithm – 10,000,000 Iterations	1.9	121.7

Table 3: General Benchmarks on OMAP GPP.

The most interesting trend shown in these measurements is that the maximum power consumed by the Beagleboard is approximately 1.9 Watts, regardless of the type of operation. As shown in the results in earlier sections, this is also the maximum power consumed by the OpenCV facial detection program for all input images, Linux filesystem search, and kernel module load. This is significant data as it reveals that the Beagleboard

draws a maximum of approximately 1.9 Watts when executing any set of instructions on the GPP, assuming no additional peripherals are attached.

Chapter 4: Digital Signal Processor Evaluation

We evaluated the performance of the Beagleboard with the OMAP DSP enabled in order to determine if a worthwhile performance improvement could be gained by porting OpenCV to the DSP. The performance results reported thus far were measured with programs running exclusively on the GPP and with the DSP powered off. Moving some or all of the facial recognition algorithm to the DSP would likely improve the execution speed of those portions, but would also incur performance hits for the additional power consumption incurred by the DSP itself and for the overhead in execution time required for passing information back and forth between the GPP and DSP.

DSPLINK EXAMPLES

The DSPLink package comes with example applications that implement various types of data transfers between the GPP and DSP to verify that DSPLink is working correctly. Using the Loop and Message example programs as representative data transfers that would be required in any application utilizing the DSP, we measured the power consumed by the DSP in general and the transfer delays incurred by offloading some portion of an application program from the GPP to the DSP.

The first step in communicating with the DSP is to turn it on from the GPP using the lpmON.xv5t utility, included with DSPLink. The complement utility, lpmOFF.xv5t, turns the DSP off, and issuing the two sequentially (lmpOFF.xv5t followed by lpmON.xv5t) performs a reset of the DSP, which is required whenever a new DSP executable is to be loaded on the DSP [5]. Once the DSP is turned on, the idle power of the Beagleboard increases from 1.4W to 1.7W, showing that the DSP in an idle state consumes 300mW, a roughly 21% increase in power consumption over the Beagleboard

running only with the GPP. Once the DSP is on, it is ready to receive its executable from the GPP and begin execution.

The Loop example implements a streaming data interface between the GPP and DSP, which would be needed in any application for which the DSP acts as a co-processor to the GPP, and simply performs one signal processing component of a larger program running on the GPP. In the facial detection example, this type of interface could be used to transfer the image to the DSP for the detect_and_draw function and to send the resulting image with the faces identified sent back to the GPP. Loop sends buffers of a user-specified size for a user-specified number of iterations from the GPP to the DSP and back. This example was run and timed in Linux (using time –p) at 1, 100, and 1000 iterations with a buffer size of 1kB.

DSP Loop Runs	Run Time (s)	Power (W)
1 Iteration	0.05	2.2
100 Iterations	0.14	2.2
1000 Iterations	0.9	2.2

Table 4: DSP Loop Example Results

The Message example is similar to Loop, but sends messages back and forth between the GPP and DSP instead of streaming data. This type of interface is used to allow the GPP to control the flow of the DSP executable, and would be used in almost any application for control and notification between the GPP and DSP. In Message, the message size is fixed, but the user may specify the number of iterations, and the individual message transfer times are reported by the program, excluding all overhead

such as the initial loading of the DSP executable, etc, which are included in the Linux-timed execution of Loop. The average round-trip time for a single message is approximately 120 microseconds. One example output of the Message is shown below.

```
========== Sample Application : MESSAGE ==========
Entered MESSAGE_Create ()
Leaving MESSAGE_Create ()
Entered MESSAGE_Execute ()
Transferring 10 iterations took 0 seconds 1221 microseconds.
RoundTrip Time for 1 message is 122 microseconds.
Leaving MESSAGE_Execute ()
Entered MESSAGE_Delete ()
Leaving MESSAGE_Delete ()
================================================
```

Illustration 7: DSP Message Example Output

ANALYSIS

A typical UAV application utilizing both the OMAP's GPP and DSP, such as facial detection, would likely utilize both communication formats analyzed in this section. The streaming data interface, emulated by Loop, would be used to transfer the image to be processed from the GPP to the DSP and back, while the Message interface would be used to allow the GPP to control the flow of the routine and for the processors to notify each other. Assuming a 100kB file size, which is a reasonable average of the file sizes used in the OpenCV measurements, the results suggest that the time to load the DSP executable and transfer the image to it, including all associated overhead, would take approximately 0.14 seconds. Conservatively accounting for a much larger DSP executable and assuming that this time doubled, the operation would still only take just under 0.3 seconds. The only other communication needed between the GPP and DSP would be control messages, and the data suggests that, at approximately 120 microseconds, the duration of these are negligible compared to the executable and data transfer times. So, the overhead delay incurred by utilizing the DSP to perform the

detect_and_draw function of the facial detection program is estimated at 0.3 seconds, or approximately 13% of the run time for detect_and_draw on the GPP (2.28 s). Therefore, if DSP can execute the detect_and_draw function 13% more efficiently than the GPP, the total execution time would break even.

The additional performance hit, however, comes from increased power consumption from running the DSP. The Beagleboard with the GPP running the facial detection consumed approximately 1.9W, while with the DSP running the Loop and Message examples consumed 2.2W, an approximately 16% increase. It is reasonable to assume that the Loop and Message examples did not fully consume the complex datapath of the DSP, which consists of six ALUs and two multipliers, and so did not exhibit maximum power consumption. So, the additional power consumption of the DSP running a complex algorithm could be significantly more than 16%.

Chapter 5: Discussion

SUITABILITY FOR UAV APPLICATIONS

We have considered the suitability of the Beagleboard and OMAP3530 for an Unmanned Aerial Vehicle application providing complex, real-time image and video processing. From the perspective of sheer capability, we have shown the Beagleboard can perform very complex image processing algorithms, using facial recognition as an example. And, at 370mA, the maximum current drawn by the Beagleboard while running any significant program, like facial detection or flight control, could run for just under six hours with a 2.2Ah battery, assuming the UAV motors were driven by a separate power plant. This type of battery is available for UAV applications in Lithium Sulfide technology and would weigh about 30 grams [16].

Unfortunately, the facial detection program takes a considerable amount of time, approximately 2.28 seconds in what was considered the most applicable benchmark in our tests. For processing real-time video data, this translates to an acceptable frame rate for the incoming video stream of about 0.4 frames per second, which is not an acceptable for video. The minimum frame rate necessary to create the illusion of moving video is generally considered to be about 15 frames per second [17], so a more than 30x improvement in the execution speed would be necessary to detect faces in a minimally-acceptable video stream. Additionally, this does not account for the overhead in Linux for simultaneously providing flight controls and other ancillary functions, which was the original motivation for utilizing a multi-threaded operating system. When considering this overhead, the performance improvement would need to be even greater.

Utilizing the DSP on the OMAP to process more complex functions of the facial detection algorithm could improve the performance overall, but due to the overhead

involved in communicating from the GPP to the DSP and back, the DSP would need to provide an approximately 13% performance improvement just to break even with the algorithm being executed on the GPP alone. This level of performance improvement may be possible, depending on the efficiency of the DSP compiler and its utilization of the DSPs robust data path, but it seems very unlikely that an additional 30x improvement could be realized.

This does not mean, however, that the Beagleboard is completely unsuitable for UAV applications. Not all missions may require full-motion video streams to be analyzed; a smaller, lightweight UAV that could provide periodic still images, once every three seconds or so, pre-processed for faces, buildings, etc. could also be valuable in certain situations, and the Beagleboard could provide a platform for such an application. Additionally, in this application, it seems that the use of the DSP would unnecessarily increase the power consumption. Since its performance improvement cannot provide full-motion video anyway, the power consumption improvement for not using the DSP would seem to be more valuable than being able to take still images a slightly higher rate, and the UAV would be able to use a lighter battery or fly longer distances.

OTHER APPLICATIONS

In light of this performance analysis, we can propose other applications for which the Beagleboard running OpenCV would be well suited. OpenCV can be run much faster on a powerful desktop machine; the applications for which the Beagleboard is more well suited require lower power consumption and/or a small, lightweight, and inexpensive form factor.

Automatic License Plate Readers

City and County governments currently use Automatic License Plate Reader (ALPR) technology to scan the license plate numbers of vehicles automatically while driving their patrol cars. The plate numbers gathered are checked against a database of offenders, and an notification generated if a match is found. The cameras used for these systems are sophisticated, bulky, and expensive, since they must correct for different angles and driving at speed. If the Optical Character Recognition (OCR) and optical correction could be done with OpenCV instead of expensive cameras, these ALPR systems could be constructed with relatively simple cameras and made smaller and less expensive.

Abandoned Baggage Detector

Airport security personnel need to be informed when suspicious baggage is left alone in a terminal, but it is difficult and personnel-intensive for humans to keep track of every piece of baggage in every location in a large airport. Using the processing capability of OpenCV on the Beagleboard, a compact enclosed system with an inexpensive camera could be used to periodically take a picture of certain area and scan to see if a piece of luggage is still in the same place that it was during the last image. If a certain threshold is met, the Beagleboard could alert a central command station that it has detected suspicious baggage. Since the systems would be relatively inexpensive, they could be placed all over the airport to cover as many locations as possible.

Appendices

SOURCE CODE

The source code for the Face Detect, Camera Test, and GPP Benchmark programs can be found online at:

face_detect.c:
https://docs.google.com/leaf?id=0B_8AwVfI6p_hNzYxN2MwMzQtZWUxOS00YjJjLWI4NmUtZjQ1ZDZmZWEzMWJj&hl=en

camera_test.c:
https://docs.google.com/leaf?id=0B_8AwVfI6p_hMTJhNWM1MDgtY2QwMC00MmZjLWIwNTYtMjQ0MGU0YWZkNjIw&hl=en

gpp_benchmark.c:
https://docs.google.com/leaf?id=0B_8AwVfI6p_hMDVjYTU0NzItYmFmYy00NGQ5LTliN2UtNDA4MGU0MjUyNzEy&hl=en

References

[1] Bradski, G. and Kaehler, A. *Learning OpenCV*. Cambridge, MA: O'Reilly Media, Inc., 2008.

[2] Coley, G. *Beagleboard System Reference Manual Rev C4*. Revision 0.0. <http://beagleboard.org>., 2009.

[3] Texas Instruments. *DSP/BIOS Link User Guide*. Version 1.64. November 2009.

[4] Texas Instruments. *OMAP35x Applications Processor Technical Reference Manual*. May 2010.

[5] Texas Instruments Embedded Processors Wiki. Accessed August 2010 – November 2010 <http://processors.wiki.ti.com/index.php/Main_Page>

[6] ETH PIXHAWK: MAV Computer Vision. Accessed August 2010 – November 2010 <http://pixhawk.ethz.ch>

[7] Google Code Beagleboard Wiki. Accessed August 2010 – November 2010 <http://code.google.com/p/beagleboard>

[8] The Angstrom Distribution | Embedded Power. Accessed August 2010 – November 2010 <http://www.angstrom-distribution.org>

[9] OpenCV Wiki. Accessed August 2010 – November 2010 <http://opencv.willowgarage.com>

[10] "Face Detection using OpenCV." OpenCV Wiki. Accessed August 2010 – November 2010 <http://opencv.willowgarage.com/wiki/FaceDetection>

[11] "HighGUI Refence Manual." Cognotics. Accessed August 2010 – November 2010 <http://www.cognotics.com/opencv/docs/1.0/ref/opencvref_highgui.htm>

[12] "Haar-like features." Wikipedia. 2 October 2010. Accessed 20 November 2010 <http://en.wikipedia.org/wiki/Haar-like_features>

[13] Seo, Naotoshi. "Tutorial: OpenCV haartraining." *Naotoshi Seo note site*. November 2010 <http://note.sonots.com/SciSoftware/haartraining.html>

[14] Chavanne, B.H. "U.S. Army Continues Heavy Focus on UAS." *Aviation Week*. 7 January 2010. October 2010 <http://www.aviationweek.com>.

[15] Drew, C. "Military is Awash in Data From Drones." *New York Times*. 10 January 2010. October 2010 <http://www.nytimes.com>.

[16] "High energy, lightweight, batteries." Barnard Microsystems Limited. 21 November 2010 <http://barnardmicrosystems.com/L4E_batteries.htm>

[17] "Video." Wikipedia. 21 November 2010 <http://en.wikipedia.org/wiki/Video>