UNIVERSITY OF SOUTHAMPTON
Faculty of Physical and Applied Sciences

A group design project report submitted for the award of
Master of Electronic Engineering

Supervisor: Dr. Rob Maunder
Examiner: Dr. Jeff Reeve

Unmanned Aircraft Camera Module (GDP Group 18)

by Andrew Busse,
John Charlesworth,
Michael Hodgson,
Piyabhum Sornpaisarn,
Paramithi Svastisinha

December 15, 2011

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES

A group design project report submitted for the award of Master of Electronic Engineering

by Andrew Busse,
John Charlesworth,
Michael Hodgson,
Piyabhum Sornpaisarn,
Paramithi Svastisinha

The SkyCircuits autopilot module is an advanced autopilot module aimed at unmanned aircraft vehicles (UAVs) for the recreational, academic and commercial markets. Previous camera systems used on the autopilot saved images to on-board storage, meaning it was not possible to determine how good the taken images were without landing the aircraft. By utilising the built-in wireless transmission capabilities of the autopilot we were able to implement a remote camera module for this autopilot system, allowing the remote triggering and download of images designed for use while the UAV is in flight.

Acknowledgements

The Authors of this document would particularly like to thank Dr. Rob Maunder, for providing good feedback throughout the semester, and for some excellent feedback on this report; and Dr. Matt Bennett of SkyCircuits, for facilitating what has been a thoroughly interesting and challenging project for all parties, and for being available to debug our Autopilot module when we were facing serious issues with it. It was a privilege to have been members of this group.

Thanks also go to numerous other members of ECS, staff and students alike, who have provided help, support and feedback during this project.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction - (ab)

1.1 Description of Problem

This project seeks to produce a system that is able to interface a camera with an autopilot on board an Unmanned Aerial Vehicle (UAV). The desired module must capture a digital still image from the camera, then transmit this data via a low-bandwidth serial link to the autopilot. The autopilot then re-transmits this data over its wireless (RF) link to a ground station running our customers ground software. Then our customer's software provides a TCP/IP interface, which allows us to capture this data in our own program, and then display the captured image data to the user. Both the Autopilot and our customer's ground station software currently exist.

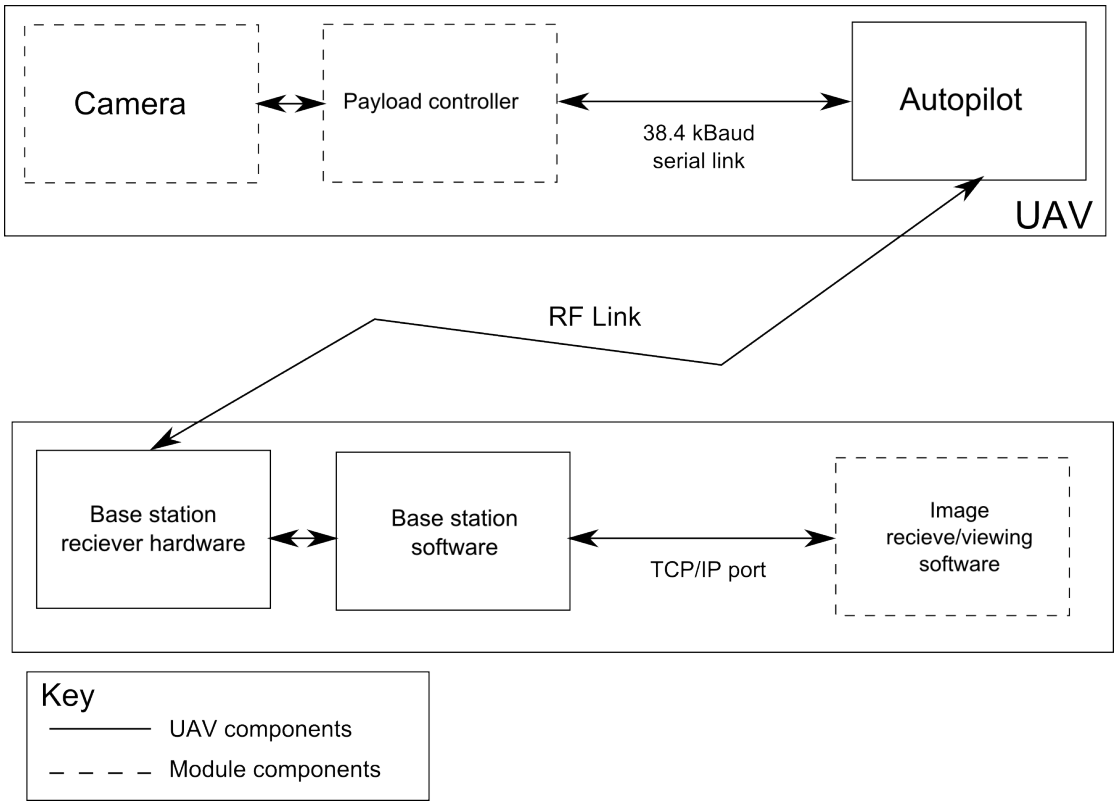


FIGURE 1.1: Block diagram displaying how the separate parts of this project interact, and which parts of this project we need to implement.

Deliverables for this project would be a suitable camera for the project, an electronic module presented (preferably) on PCB, and an additional piece of ground station software to interact with the payload module. It needs to be in a state such that it would require minimal additional work before being presented to the clients of our customer as a potential "Wireless Camera Payload" solution that they would be able to implement. Therefore, delivery of User Documentation, and a repository containing all hardware and software necessary to easily reproduce the project is required.

1.1.1 SkyCircuits

SkyCircuits Ltd. [1] is a company that designs and sells autopilot modules and ground station software for unmanned aircraft. These modules are able to take control of a UAV from take-off to landing, along a user-defined path, with no need for human intervention. SkyCircuits market two different autopilots: A more expensive model aimed at the recreational, academic and commercial markets, and a cheaper, less powerful module, aimed at the Recreational market only.

It was this company's autopilot module that recently successfully flew the world's first

3D-printed UAV, the Southampton University Laser Sintered Aircraft (SULSA), [2] an event which was also covered by the media [3].

Our contact from SkyCircuits for this project is its director, Dr. Matt Bennett.

1.2 Existing Solutions

Currently, SkyCircuits' customers have implemented a system where they have retrofitted a high-resolution camera to a UAV, but this acts entirely independently of the autopilot module, and images are stored locally, therefore there is no way of knowing whether an image taken during flight is any good. The camera used for this system is very expensive (approximately £2000). The system we present in this project is significantly cheaper than this existing setup.

1.3 Project Motivation

The cost and inflexibility of the existing solution 1.2 means that it is unlikely that recreational users will want to replicate this system. Creating a camera payload for the autopilot that is cheaper, more functional (in that it would be able to exploit the wireless link furnished by the autopilot), and open source, is in our customer's interest as it may be useful in furthering interest around their products.

Also, this project serves to test a key piece of functionality of the autopilot modules to its limit - the payload module port. This project serves to test the robustness of this feature, and help iron out any potential bugs, and encourage development of further payload modules. The ability to further develop this project to include other types of payload and further potential applications of the autopilot would potentially increase interest around the products.

Both of the autopilot modules marketed by SkyCircuits Ltd. offer the "payload" feature.

1.4 Solution

Our solution, presented in Chapter 6, comprises a complete system implemented on PCB, connected to a UAV-mountable board camera, which, when prompted by our software, is able to capture an image, store it locally on an SD card, then send it over the wireless link provided by the autopilot module to a "ground station" (a laptop configured with the autopilot's wireless receiver and SkyCircuits' software). Our software interacts with the existing software's TCP/IP port, and can send images via the autopilot to the payload, and accept data from the payload and display it on screen.

1.5 Report Structure

Below is a short summary of what each section of this report contains:

1.5.1 Background Research

This section introduces various topics that have been relevant to this project, including:

- Image Compression Theory
- JPEG Image Compression

1.5.2 Specification

The specification in its submitted form, along with justification for the inclusion of each point.

1.5.3 Planning

Our project plan, including our initial Gantt Chart, Risk Analysis, Skills Audit, Work Allocation, Planned Communication, Resources, Milestones and Technical Plan.

1.5.4 Design Choices

This section explains the various design choices which we have made during this project, including a discussion of the other options that were available to us at the time, and a justification for our choice.

1.6 Authorship

The main author of each section of this report is indicated by initials in the title of either the chapter, section, subsection, etc. that the author was responsible for writing. Editing the report was the entire group's responsibility.

- ab - Andrew Busse
- jc - John Charlesworth
- mh - Micheal Hodgson

- ms - Paramithi “Mitch” Svastisinha
- ps - Piyabhum Sornpaisarn

Chapter 2

Background Research

2.1 Custom Image Compression (jc)

Before starting research into image compression standards such as JPEG [6], a range of general image compression techniques were researched and a selection of these were also implemented in MATLAB, see appendix C.

2.1.1 Overview of Available Methods (jc)

The idea of image compression is to reduce the amount of data used to describe the image, whilst retaining as much as possible of the information contained in the image. there are two main types of image compression: lossless and lossy.

Lossless compression techniques retain all the information in the image but reduce the size of the data required to describe it, although they can't achieve the same level of compression as lossy techniques. This is usually done by looking for patterns in the data and using these patterns to encode the data in a more efficient way. Some examples of lossless techniques are: run-length encoding or entropy encoding (such as huffman encoding, see section 2.1.3).

Lossy compression techniques discard some of the information in the image in order to be able to compress it further than is possible with lossless compression. The two main types of lossy image compression are those based on reducing the information in the colour space (often utilising the way the human eyes works in order to maintain perceived image quality) and those based on transforming the image, usually into a version of the frequency space.

If the image is being compressed so that it can be sent over a medium then this can be done either as a single action: the image is compressed and sent once and any information

that is lost is lost forever, or progressively: the image is compressed first with a high loss of information and then the additional information is sent so that the loss of the image being sent reduces as more data is received.

Reducing the resolution of an image could also be considered as lossy compression as it is a very simple method of reducing the amount of data required to describe an image by reducing the information in the image.

2.1.2 Downsampling (jc)

The most obvious method for sending an image progressively is to simply send pixels in such an order that at any time they form a down sampled (lower resolution) version of the image. This way rather than having the image fill in from the top left hand corner the image can become less pixelated as more data is downloaded.

2.1.3 Huffman Encoding (jc)

Huffman coding is a type of entropy encoding. Huffman code is a variable length code, what is special about it is that it uses "prefix" (also known as "prefix-free") codes whereby the code for one symbol is never the prefix for the code of any other symbol meaning that an input can be immediately recognised assuming you know the start point of the sequence [39].

Huffman codes are assigned to symbols based on the frequency of occurrence of the symbol, with the most frequent symbols being assigned the shortest code and the least frequent the longest, this way the data is compressed without any information being lost. [39]

For example the sentence "This is an example sentence" contains 1: h, x, m, p and l; 2: t, i and a; 3: s and n; and 4: e and would be encoded as follows:

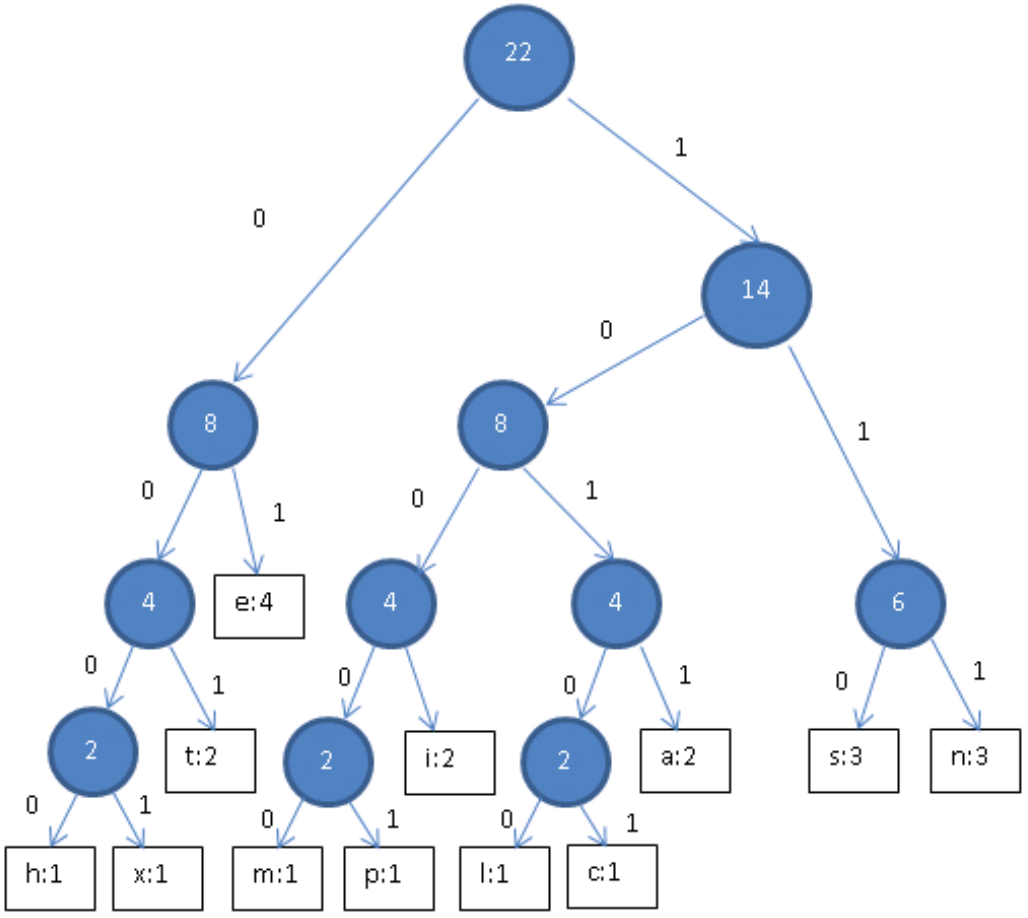


FIGURE 2.1: Example Huffman tree used to encode the sentence "This is an example sentence"

This will then encode the letters as:

Letter	**Prefix code**
e	01
s	110
n	111
t	001
i	1001
a	1011
h	0000
x	0001
m	10000
p	10001
l	10100
l	10101

TABLE 2.1: Example Huffman code table

It can be seen from table 2.1 that the Huffman code has encoded the most common symbol with the shortest code and none of the codes are prefixes of any of the others.

2.1.4 Colour Space Conversion (ms)

Raw image information is defined using the red, green, blue (RGB) colour model. In this additive model, the pixel's colour is determined by a sum of red, green, and blue light. If each value for R, G, and B is represented with a single byte, this means that each pixel in an image has 3 bytes associated to it, so an image with **n** pixels would contain $\mathbf{3} * \mathbf{n}$ bytes worth of information. For a 640x480 image, this means we have 921,600 bytes of information to send through.

Instead of storing all the image data using the RGB colour model, chroma sub-sampling allows the file to be compressed without losing any information that the human eye can discern. The image is put into a luma-chrominance color space (YCbCr). In this form, the color of a pixel is described by two values, one (*luma*) (Y) which describing its *luminance* (brightness), and one (*chrominance*) which describes its colour (represented using two components Cb and Cr). These values are nonlinear representations of the actual luminance and chrominance values [11].

Although this also separates a single pixel into three values like the RGB colour model, it allows for a more compact file due to the fact that the luma and chroma samples are not equally discerned by the human eye. In the RGB colour model, all three values must be taken at the same resolution in order to display the colour of one pixel. Lowering the resolution in this colour space quickly reduces the quality of an image.

Chroma subsampling, or the YCbCr colour model, takes advantage of the fact that the human eye more easily distinguishes subtle differences in luminance than colour [11]. Therefore, by supporting different levels of resolution for the chroma and luma values, it is possible to lower the amount of information stored within an image which appears identical to the uncompressed image to human eye.

As an example, consider a 4x4 pixel image. Using the RGB colour model, the image would be represented with 48 bytes, 3 bytes for each pixel. With chroma subsampling, we can group 2 or 4 pixels together to form a *chrominance pixel*. All the original image pixels in this new "pixel" would still be associated with a luma value (Y), but they would all share the same chrominance information (Cb and Cr). This means an image with **n** pixels would now contain $\mathbf{n + 2n/c}$ bytes worth of information, where c is the size (in image pixels) of the chrominance pixel. In this example, using a resolution of 2 adjacent pixels gives us 32 byte and using a resolution of 2x2 pixel blocks gives us 24 bytes.

Commonly used chrominance subsampling patterns are designated as a string of 3 integers separated by colons. The relationship among the integers denotes the degree of vertical and horizontal subsampling. The first integer is usually 4 and represents the now standard luma horizontal sampling rate. The second digit specifies the horizontal subsampling of both Cb and Cr with respect to luma [10].

If the third digit is the same as the second, this denotes that there is no vertical subsampling. If the third digit is zero, 2:1 vertical subsampling for both Cb and Cr is indicated. This strange notation is due to the invention of this notation predating the concept of vertical subsampling [10].

The figure below compares 5 subsampling patterns used against a 2x2 pixel image.

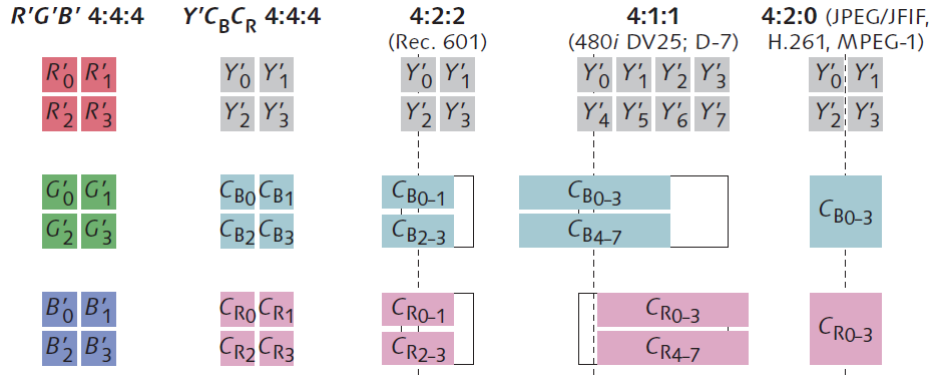


FIGURE 2.2: Different chroma subsampling schemes for a 2x2 pixel colour image [10].

2.1.5 Transform Based Compression (jc)

Some simple progressive encoding was done as an investigation into using the fast fourier transform (FFT) and the discrete cosine transform (DCT) for image compression and how these could be sent progressively.

The fourier transform works by approximating a signal with the sum of sinusoids of different frequencies. The transform function (given an array as an input) outputs an array of complex numbers whose magnitude is the magnitude of the sinusoid at that frequency and whose argument is the phase.

The fast fourier transform (FFT) is an optimised version of the discrete fourier transform (DFT) which implements the fourier transform in discrete rather than continuous space, this enables its implementation by computers.

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)} \tag{2.1}$$

where $\omega_N = e^{(-2\pi i)/N}$ is an Nth root of unity.

Equation 2.1 is the equation for the 1 dimensional FFT, to implement the FFT in 2 dimensions (as you would need to for an image) you simply take the 1 dimensional FFT of each column and then of each row of the result.

The DCT approximates a signal with the sum of cosine functions of different frequencies. The DCT is similar to the FFT in that it transforms into a frequency space, but the DCT produces strictly real values.

$$F_{u,v} = \alpha(x)\alpha(y) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f_{x,y} \cos\left[\frac{\pi}{M}(u+\frac{1}{2})x\right] \cos\left[\frac{\pi}{N}(v+\frac{1}{2})y\right] \tag{2.2}$$

for $0 \leq u \leq M-1$ and $0 \leq v \leq N-1$

x is the pixel row

y is the pixel column

u is the horizontal DCT frequency

v is the vertical DCT frequency

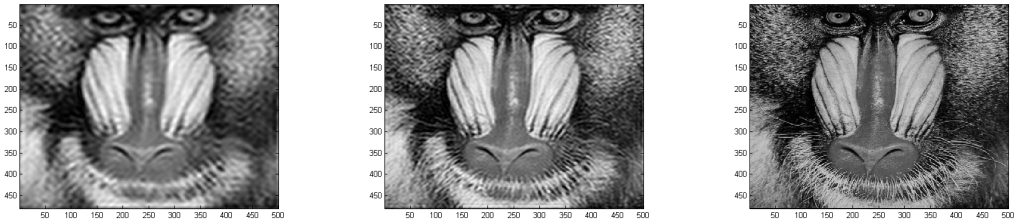
$$\alpha(x) = \begin{cases} 1/\sqrt{M} & , u = 0 \\ 2/\sqrt{M} & , 1 \leq u \leq M-1 \end{cases}$$

$$\alpha(y) = \begin{cases} 1/\sqrt{N} & ,v = 0 \\ 2/\sqrt{N} & , 1 \leq v \leq N-1 \end{cases}$$

The 2 dimensional DCT equation is given in equation 2.2, this is the equation exactly as implemented in the MATLAB function "dct2" used in C.

2.1.5.1 Whole Image (jc)

The first algorithm in appendix C was a progressive scheme based on taking an FFT of the whole image and then reconstructing the image using more and more of the frequency content of the image starting with the lowest frequencies.



(a) Image with only low frequency content

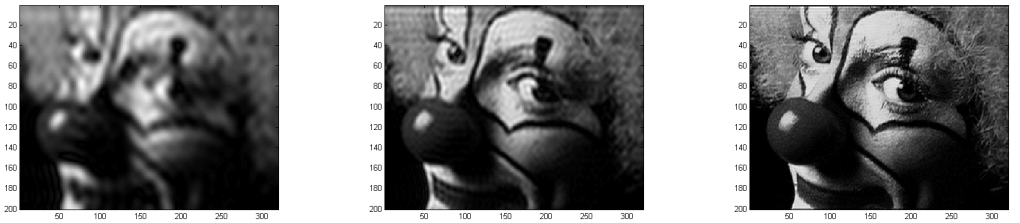
(b) Image with medium and low frequency content

(c) Image with all its frequency content

FIGURE 2.3: Images reconstructed from increasing amounts of an FFT of the whole image, see appendix C for parameters

Figure 2.3 how the level of detail in the picture increases as more of the frequency content is used to display it. Potentially the image data could be transmitted so that the low frequency data is received first and additional frequency content is then sent progressively so that the image gains detail smoothly.

The second algorithm in appendix C used the DCT rather than the FFT. The advantage of the DCT over the FFT for this purpose is that the FFT generates complex values thereby effectively doubling the amount of data whereas the the DCT only produces real values.



(a) Image with only low frequency content

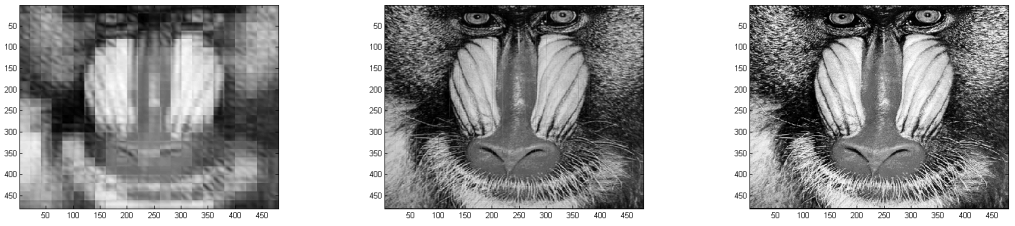
(b) Image with medium and low frequency content

(c) Image with all its frequency content

FIGURE 2.4: Images reconstructed from increasing amounts of a DCT of the whole image, see appendix C for parameters

2.1.5.2 Sectioned Image (jc)

As FFTs or DCTs can be quite computationally intensive when done with large arrays of values it is sometimes sensible to first break the image down into smaller blocks and then do the transformation on these. This would reduce the complexity of the computations while retaining the same functionality.



(a) Sectioned image with only low frequency content

(b) Sectioned image with medium and low frequency content

(c) Sectioned image with all its frequency content

FIGURE 2.5: Images reconstructed from increasing amounts of a fft of sections of the image, see appendix C for parameters

Figure 2.5 shows much the same as figure 2.3 but with the image first broken down into small (definable in the code (see appendix C), 20x20 blocks were used for these images) sections and then the frequency components of these transmitted progressively so that the detail within each section increases as more data is sent and thus the detail of the whole image also increases.

2.2 JPEG Image Compression (ms)

2.2.1 Introduction

The images obtained from a camera will generally be in the JPEG image format according to the EXIF (EXchangeable Image File Format for digital still cameras) (EXIF Version 2.2)[6]. JPEG is a form of image compression standard named after its developers, the Joint Photographic Experts Group[9]. This section will provide a quick overview of the JPEG image compression algorithm and attempt to explain the information available by analysing the JPEG file structure.

2.2.2 JPEG Compression Process

The JPEG compression algorithm uses 8 stages to compress an image: [12]

2.2.2.1 Stage 1: Colour Space Conversion

The image undergoes chroma subsampling from the RGB colour model to a 4:2:2 or a 4:2:0 YCbCr colour model. Both subsampling patterns are approved by the EXIF and the pattern used is dependant on the camera's specifications. (see section 2.1.4)

2.2.2.2 Stage 2: Block Segmentation

The image is then separated into 8x8 or 16x16 pixel blocks, called MCUs (Minimum Coded Unit) depending on whether the image has been chroma subsampled to a 4:2:2 or 4:2:0 model respectively[6].

2.2.2.3 Stage 3: Discrete Cosine Transform

The image is transformed from a spatial domain representation to a frequency domain representation using the Discrete Cosine Transform[12]. (see section 2.1.5)

2.2.2.4 Stage 4: Quantization

Using the wave equations from the DCT step, the algorithm sorts them from low-frequency components (gradual colour changes) to high-frequency components (sudden colour changes), from the top left to the bottom right corner of the image. The algorithm discards the high-frequency details due to the human eye not recognizing them as well as low-frequency details. This is done through division of all DCT wave equation coefficients using a quantization table and then rounding the result to the nearest integer. This quantization table differs between the majority of digital cameras and software packages[12].

2.2.2.5 Stage 5: Zigzag Scan

The matrix obtained through quantization is then re-ordered from the top-left corner into a 64-element vector in a zig-zag pattern. This is done to ensure that the high-frequency components which are most likely to round to zero after quantization, make up the lower right hand part of the matrix.

2.2.2.6 Stage 6: DPCM on DC components

DPCM is known as Differential Pulse Code Modulation. It is the process of calculating the block-to-block average difference of the DC components across the entire MCU block and encoding the average as a change from the previous block's value[12].

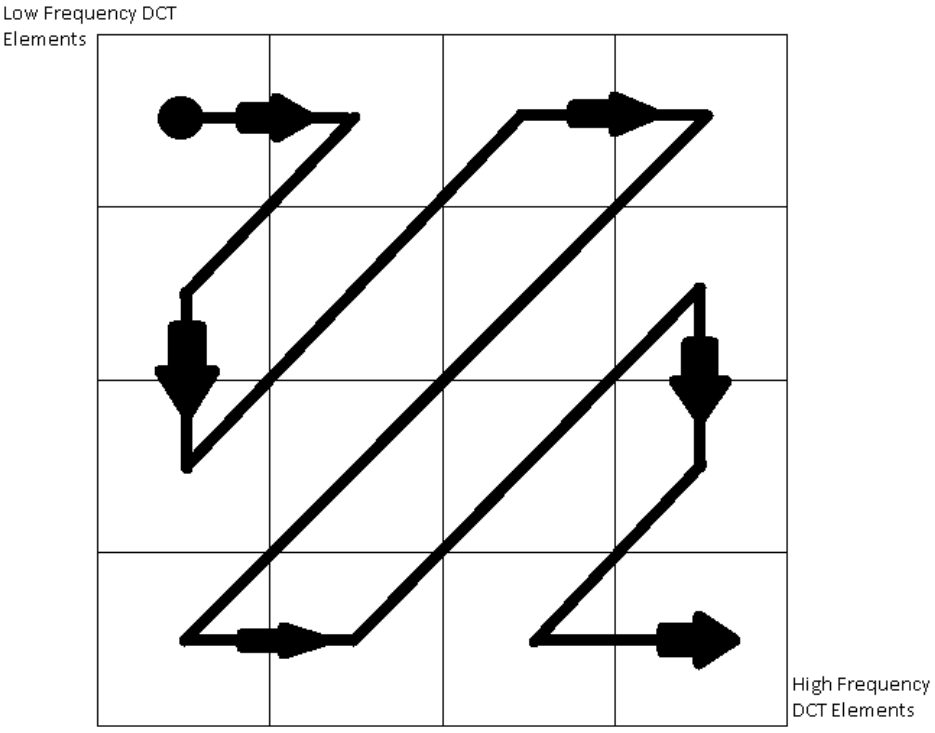


FIGURE 2.6: Example of a zigzag scan [12]

2.2.2.7 Stage 7: RLE on AC components

RLE is known as Run Length Encoding. It is the process of storing each value of the AC components of the 64-element vector with the number of zeros preceding it for the purposes of the final stage[12].

2.2.2.8 Stage 8: Entropy Coding/ Huffman Coding

Finally, a dictionary representing commonly used value strings with shorter code is created. Common string and patterns are represented by short codes while less frequently used strings are represented with longer codes. (see section 2.1.3)

2.2.3 JPEG Structure

A JPEG file can be separated into two main parts. The first part of the JPEG file is composed of segments containing information concerning various properties of the image which must be read in order to display the image from its compressed form. The

number of segments within one JPEG file varies from picture to picture. The second part contains the entropy-encoded image data, which can be decoded using the information provided from the headers of the file.

2.2.3.1 JPEG Segments

The JPEG headers are capable of storing most of the metadata related to an image, not all of which is necessary for the decompression of the image. The following headers are those which contain all the information necessary for a successful decompression of the JPEG image, as well as those which can be found in all JPEG images.

The table below details some important segments which are necessary for the image to be properly displayed: [6]

TABLE 2.2: JPEG File Layout

Segment Name	**Marker Name**	**Marker Code**	**Description**
SOI	Start Of Image	0xD8	Start of compressed image data
APPn	Application Segment n	0xEn	Exif application information segment
DQT	Define Quantization Table(s)	0xDB	Quantization table definition.
DHT	Define Huffman Table(s)	0xC4	Huffman table definition.
SOF	Start Of Frame	0xC0	Frame parameter data
SOS	Start Of Scan	0xDA	Scan parameter data

The entropy-encoded image data appears after the SOS segment, right before the EOI header marker.

For information concerning the actual content of each of the JPEG segments, please refer to the appendix A.

2.2.3.2 Entropy-encoded image data

The image data is received chrominance pixel by chrominance pixel. The first bytes received are the luma values of the individual image pixels. In the case of JPEG images, they are received from left to right, top to bottom, moving horizontally before vertically. The luma values are then followed by the Cb and Cr values of the top-left pixel, in that

order. After all the data for a single chrominance pixel has been sent (6 bytes) those of the next chrominance pixel are sent in the same format, read in the same order as the image pixels within one chrominance pixel. This continues until the EOI marker is found.

2.3 Network and Port Connection - (ps)

TCP/IP is suitable for device communication and it can be implemented in many software languages. The connection can refer to an interconnect between host and router. Host is a computer that runs a program such as a web browser, or any software on the computer [28]. A Router is a device that can forward communication from one machine to the another. There are many ways to link machines using TCP. A port connection is one way of connecting the TCP from the router to the host. Port can refer to hardware such as a USB connection or it can refer to a processes on the machine [29]. A protocol is an agreement about the packet structre and what they mean between communicating programs. Information that will sent through the port will be a sequence of data bytes. These byte sequences generated are called packets [28]. It includes user data and control information.

TCP/IP is designed to detect and recover its losses, and other errors that might occur so the application does not have to deal with this [28] i.e. the application data does not have to break up the data on its own. It has functions to connect to the computer, send and receive commands, display flags, and set up the port values.

2.4 SkyCircuits Autopilot (ab)

Our customer kindly lent us one of their more advanced, SC2 autopilot modules for use in this project, as well as use of the latest version of their Ground Control Station software.

This module [4] is a robust, professional unmanned avionics system, able to take complete control of an unmanned aircraft from take-off to landing. The device is low power (0.8W at 5V), lightweight (290g including aluminium enclosure), with a comprehensive sensor range (able to measure height, pitch, roll and yaw to an impressive level of accurary), capable of controlling up to 6 servomotors, and able to operate at a number of wireless frequencies.

Wireless communication with the ground station is done with XBee PRO modules, with an RF data rate of 250kbps and a line of sight (LOS) range of up to 1.6km. Error checking is performed, but data cannot be guaranteed to arrive.

2.4.1 Autopilot Payload Module Interface

The SkyCircuits autopilot module allows extension modules named 'payload modules' to be connected to the autopilot. These payload modules are connected via a RS485 serial connection at 38.4 kBaud, allowing several payload modules to be connected at once in a daisy-chain configuration. All payload modules are connected to common Transmit (TX) and Recieve (RX) lines, where the RX line is used by the autopilot to send commands and data to the payload modules, and the TX line is used by all daisy-chained modules to communicate with the autopilot.

Since the TX line is shared between all modules only one payload module can be transmitting at once over the link, with all other payload modules required to leave the line tristated. This means that each payload module must know when it is allowed to use the transmit line so as not to clobber any other payload module. In this system this is achieved by the use of 'transmit tokens' handed out by the autopilot over the RX line. A 'transmit token' is sent to each payload module in turn, informing the module that it is clear to transmit data. With only one payload module connected to the autopilot, these tokens are sent out every 20 ms.

This two way communication link is used to implement a command interface to the autopilot. A payload module can execute commands on the autopilot, allowing a variety of useful and interesting possibilities for payload module design. Of interest to us however, is the ability of the payload module to set shared memory. Since this shared memory can be accessed through the ground station software this allows us to send data through the autopilot link to the ground station. Shared memory is allocated to each payload module, accessible on the ground station using the command:

```
payload[payload_num].mem_bytes[mem_bytes_num]
```

LISTING 2.1: Accessing shared memory from ground station, typed directly into the software.

Where *payload_num* is the ID number identifying the payload and *mem_bytes_num* is the index of the set of shared memory to be accessed.

Each shared memory set is of variable length and can be set from the payload module using the following function (code provided by customer):

```
send_set_class_indexed_item_indexed(CLASS_PAYLOAD, module_id,
CLASS_PAYLOAD_MEM_BYTES, mem_bytes_num, message_to_send,
message_to_send_length)
```

LISTING 2.2: Setting shared memory from payload module, typed directly into the software.

Where *CLASS_PAYLOAD* and *CLASS_PAYLOAD_MEM_BYTES* are constants informing the autopilot that the *mem_bytes* item of the *payload* object should be set, *module_id* is the ID of the payload module, *mem_bytes_num* is the same as used in listing 2.1 and *message_to_send* is the message to be sent (consisting of an array of length *message_to_send_length*, the first element of which should be the number of bytes to set in the shared memory).

The ground station software can also send data directly to a payload module through the autopilot, where it will be sent on to the RX line of the RS485 bus as a 'packet' of data. The *send_bytes* command is used to do this, as can be seen in listing 2.3.

```
payload[0].send_bytes BYTE1 BYTE2 ... BYTEN
```

LISTING 2.3: Sending bytes 1 to N from ground station to payload module, typed directly into the software. BYTE1...N can be in decimal (e.g. 5) or hexidecimal format (e.g. 0x5).

Chapter 3

Specification

3.1 Introduction (jc)

This chapter will describe the brief we were given and an overview of the system and how the elements to be created will sit with the existing components.

The functionality intended for the system is also broken down and the priority given to each task is explained. The things that we should have to give to our customer at the end of the project are also described.

3.2 Brief (jc)

The original agreed project specification and plan, handed in on the first week of the project, was to design, build, and test an electronic module capable of capturing still images from an unmanned aerial vehicle (UAV) and transmitting the images to a ground station. The module must use the UAV autopilots low-bandwidth RS485 serial link (38.4 kBaud). A program must be written to interface with the ground station software over a TCP/IP link, allowing commands to be sent to the electronic module in the UAV and image data to be received and then displayed to the user on the ground. The electronic module should be constructed suitably ruggedly for use in the environment of the UAV.

3.3 Block Diagram (jc)

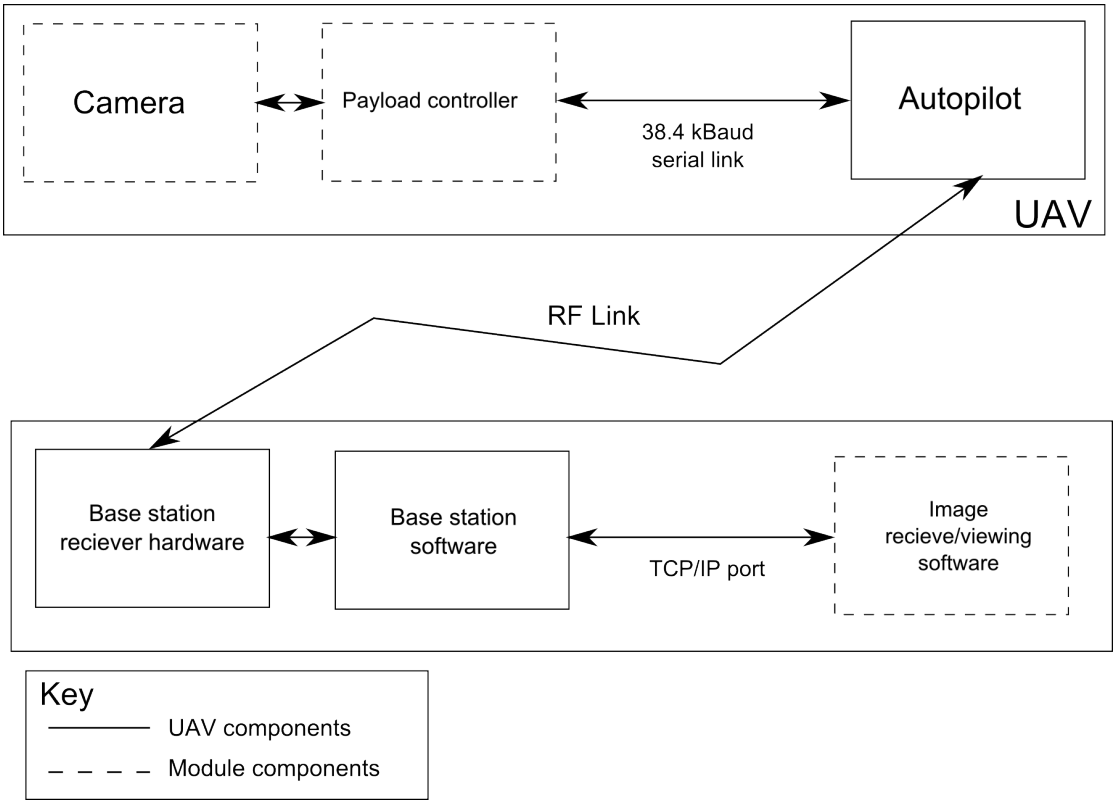


FIGURE 3.1: Block diagram of the specified system

The above diagram (figure 3.1) gives an overview of how the system described in the specification all comes together. The parts with a solid outline are parts that we were given by our customer to fit our system around. The parts with the dashed outline are the elements of the design that we need to implement ourselves.

Inside the UAV itself will go the camera, the payload controller and the autopilot. The autopilot is provided by our customer and also provides our wireless link to the ground. The camera is simply a camera so that photographs can be taken. The payload controller is the part that needs the most work, it needs to be able to control the camera; take and get images as well as being able to communicate with the autopilot and interpret any commands sent up from the base station (also referred to as the ground station).

On the ground station side of things there is a radio frequency (RF) link to the autopilot (simulated by a USB cable for development purposes) which attaches to a computer running the ground station software provided by our customer. The ground station software provides two TCP/IP ports as interfaces, one for sending commands to the ground station software and to the autopilot and one for streaming data from the autopilot. It is these TCP/IP ports that our software on the ground has to interface with so that it

can send commands up to the payload controller and then receive and display an image.

3.4 Objectives (jc)

The aim of the project was to achieve the following criteria, which were also prioritized in order to ensure organized and efficient work - high priority points are essential or very important to basic system functionality, medium priority points are not essential for basic functionality but help expand the system into a more complete system and low priority points are optional extras that should be added if time permits or if they are very low effort:

3.4.1 Specification A

The image should be encoded in such a way that a low quality image will be available quickly, the quality of which would improve as more information is downloaded. This was given a **high priority** as raw images of an acceptable resolution (see below) contain a large amount of data and we have a slow communications link therefore it would be extremely beneficial to be able to view a low quality image quickly so that you don't have to wait for the full image download to judge its appropriateness.

3.4.2 Specification B

Minimise the time needed to download the images from the UAV to the base station. The time from the users prompt until the image has been fully downloaded will be measured against the theoretical 3 minutes necessary to transmit a full image without using any compression. The goal will be to obtain a full image in less than 3 minutes. This 3 minute figure is the time it was calculated it would take to download a raw 640×480 24bit RGB image (approx. 900kB) at 38.4kBaud. This was given a **high priority** as 3 minutes is a long time to wait for an image to download and downloading images is the point of our system.

3.4.3 Specification C

The module weight will be less than 250g. This was given a **medium priority** since although a weight of under 250g would be required for flight, it would be preferable to have a system that is too heavy than one that does not work at all.

3.4.4 Specification D

Image resolution of at least 640×480**.** This was given a **medium priority**, this value was chosen as a trade off between the amount of data in the image and the quality of the image. Higher resolutions would be acceptable if the download time can be suitably reduced and lower resolutions may also be useful if they are much faster to download.

3.4.5 UAV Camera Commands

Allow the user to perform the following actions on the UAVs camera from the ground station:

3.4.5.1 Specification E

Prompt the UAV to capture and download an image. This was given a **high priority** as the main aim of the system is to be able to capture images.

3.4.5.2 Specification F

Cancel the downloading of any image while the image is being downloaded. This was given a **medium priority** because, given the potentially high download times, a lot of time could be saved by being able to cancel the download of an image you do not want before it has fully downloaded but it is not essential to the basic functionality of the system.

3.4.5.3 Specification G

Resend an image in case the current preview is corrupted. This was given a **low priority** as it does not affect basic functionality.

3.4.5.4 Specification H

Interrupt the download of an incomplete image and allow the user to save the incomplete image. Please note that this is distinct from the *Cancel* (3.4.5.2) requirement mentioned above since it requires the saving of incomplete images. This was given a **low priority** as it does not affect basic functionality, but could potentially save the user some time.

3.4.5.5 Specification I

Select the resolution settings of the image. This was given a **low priority** as it does not affect basic functionality but could be a useful feature for some users.

3.4.5.6 Specification J

Display a progress indicator which will show the percentage of the image data received, as well as a time estimate for the rest of the image to be downloaded. This was given a **low priority** as it does not affect basic functionality but allows the user to feel more confident that the system is working.

3.4.5.7 Specification K

The image capture will be triggered automatically by the UAV using triggers built into the autopilot. This was given a **low priority** as it does not affect basic functionality but could be something a user might like to be able to do.

3.4.5.8 Specification L

Allow the user to command the image capture to trigger periodically over a user specified time interval will be added if time permits. This was given a **low priority** as it does not affect basic functionality but may be a useful feature for some users.

3.4.5.9 Specification M

Images will be transmitted in colour as opposed to black and white. This was given a **low priority** as black and white images are still useful but colour images would be preferred.

3.4.5.10 Specification N

The user should be able to select between a colour image and a black and white image. This was given a **low priority** as it does not affect basic functionality.

3.5 Deliverables - (ms)

The deliverables which were planned to be produced by the end of the project and given to the customer include:

3.5.1 Hardware:

Camera module, constructed on PCB (if time permits, otherwise on strip-board), including layout designs.

3.5.2 Software:

All firmware for the electronic module, and software on the base station for viewing images. The full source code and all executable files will be included.

3.5.3 Documentation:

Technical and User Documentation. This includes all schematics related to hardware as well as all other documents concerning both the software and hardware delivered.

3.5.4 Public repository:

The full source code, all schematics, and all documents concerning both the software and hardware will be included on a public repository so that the customer may share this information with his clients.

Chapter 4

Planning

4.1 Introduction (ps)

This section will describe the partitioning of tasks into milestones as well as provide the steps to be taken to achieve the final outcome. The risk management has been planned carefully to ensure a back up plan for any possible risk. This section also includes the work allocated to each group member. It also describes the group resources such as the budget, laboratory equipment, and customer's supply.

4.2 Risk Management (ms)

A risk assessment was performed at the beginning of the project to make sure that the appropriate actions could be taken when confronted with a problem. The following table shows the risks the group has prepared for, the likelihood and impact of the risk which determines the priority of the risk, and the appropriate action in response to the risk.

Risk	**Likelihood (1: Low, 5: High)**	**Impact (1: Low, 5: High)**	**Action**
Faulty Components	4	4	Order spares where feasible. Source new/replace faulty components as soon as possible otherwise.
Team member becoming unavailable	2	4	At least two people per task. Reallocating people to different tasks as required.
Team member facing difficulties	5	2	At least two people per task. Good team communication. Prioritise necessary tasks first.
Tasks overrunning	4	3	Plan redundancies into Gantt Charts. Reallocate resources when necessary. Communicate between team members often.
Loss of files/source code	1	5	Proper use of source control. Back up all source code.
Loss of access to facilities	1	4	Work from home laboratory. Buy missing components if necessary

4.3 Work Allocation (ps)

In order to ensure that the group functions effectively, the work must be allocated to the group members and the group ressources must be managed. The group members analysed the specification of the project well to best suit the tasks to the member's skills. The work will be allocated on a regular basis. The following factors have been considered thoroughly before assigning tasks to any team members:

- Availability of each member.
- The number of members that can work on a single task.
- The deadline of the task.
- The skills required for the task

Planned Work Allocation

It is very essential that the other obligations of the group members are taken consideration during work allocation. The group members will work effectively if they are assigned work they are interested in with well clarified roles. This can be determined using a skills audit. The skills audit highlights the skills and knowledge the group members possess prior to the task. Each task has been assigned a task leader who will work with their co-workers when problems are encountered during their task. In order to avoid tasks overrunning, the tasks have been set earlier deadlines so that there is spare time to handle unexpected problems.

The results of the skill audit are:

- Andy: Power, PCB, Control, MATLAB
- Mitch: Image Processing, MATLAB, ASM, Report Writing
- John: MATLAB, Image Processing, Control, Radio Transmission
- Peak: Digital Control, Display, MATLAB, Information Theory
- Michael: Programming, μProcessors, Interfaces

The dependencies of the tasks are also important to consider. This may cause problems when tasks with many dependencies get stuck and cause many tasks to delay. This can be solved by well partitioning the tasks so that there is always a task available when others can't be completed.

The required number of people per task limits the task allocation. The customer has only provided a single UAV and only one camera should been purchased, so only one person can work on each device at a time. This is mitigated by having a laboratory space where the UAV will be placed at all times so that any team member can access it. To manage the limited resources, the only time that the individual can access the UAV is when he wants to test the system.

It is important to clarify the tasks and ensure that all tasks are properly assigned. Figure4.1 shows how the tasks are allocated between each member of the group. The blue box between members are shared tasks. If the blue box stand on its own it means only one person is assigned to that task.

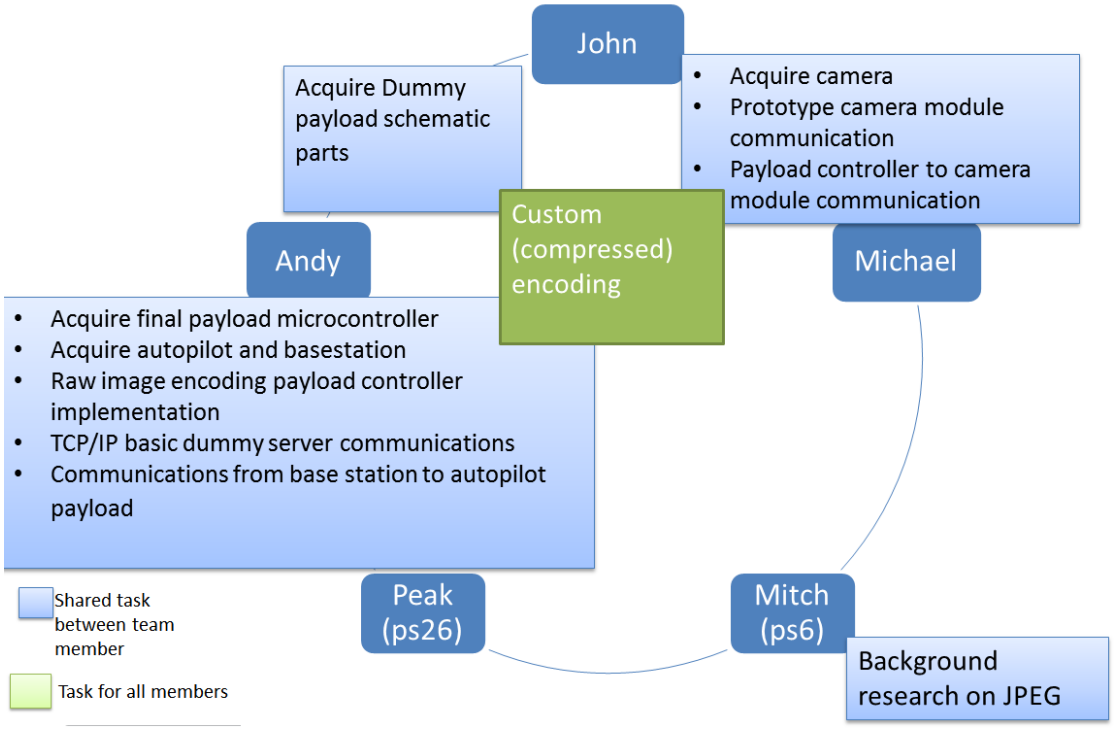


FIGURE 4.1: Initial task allocation of the project

4.3.0.1 Schedule - (jc)

Considerable thought was also given to the scheduling of the tasks, an initial Gantt chart was drawn up at the beginning of the project (see appendix H) with a breakdown of the tasks as we imagined them at the start of the project.

During our regular meetings this Gantt chart was used as a gauge for our progress and was amended when needed (i.e. our initial timing turned out to be slightly optimistic).

4.4 Team resources (jc)

This section of the management chapter covers how we controlled our use of physical resources rather than time or personnel resources. This consists of managing our budget and what we had to buy as well as what resources we had already or were provided with.

4.4.1 Budget (jc)

Our overall budget was £200 and we stayed within that while at the same time not letting it constrain us when we needed to purchase components.

Component description	**Quantity**	**Combined cost**
RS485 bus transceiver (SOIC14)	3	9.684
RS485 bus transceiver (DIP14)	3	9.756
RS485 bus transceiver	1	6.96
Logic level converter (SOT235)	3	9.684
RJ45 Socket	4	3.072
4M flash chip (8DIP)	4	2.592
Arduino mega protoshield	1	5.46
Arduino uno protoshield	1	4.38
microSD breakout board	1	5.99
Camera module - serial JPEG TTL	3	132
	Total cost:	189.58

TABLE 4.1: Table of expenditures

The above table (table 4.1) details the way that the project budget was spent. The first column is a brief description of the component, the second column is the number that have been ordered and the third column is the total cost of those components.

The biggest drain on the budget was the camera modules as at £44 each these were the most expensive components. The first camera breaking was a set-back and some alternative camera options were investigated quickly but it was decided that the uCam was the best option and a new one was ordered, thankfully there was still plenty left of the budget to accommodate this.

4.4.2 Electronic Material (jc)

4.4.2.1 Laboratory Equipment

We were given access to the "Advanced Electronics Lab" on level 3 of the Zepler building. This gave us some bench-space and access to a wide range of lab equipment.

Of the lab equipment available to us the only pieces we actually used were the digital oscilloscope and the bench-top power supply. The oscilloscope was used at various stages through development to verify signals were doing what they should be, where they should be. The power supply was used to power the camera when it was discovered that the combination of SD-card and camera was drawing more current than could be supplied by the arduino.

4.4.2.2 From the customer

We were provided with a skycircuits autopilot [1] as described in 2.4 and some software to interface with it from the ground.

We were also provided with some example microcontroller code and a schematic for a dummy payload module i.e. a payload module that can recognise transmit tokens but that doesn't actually do anything.

4.4.2.3 From ourselves

We also had access to a range of equipment that we owned personally this included a selection of development boards:

- Arduino uno, see section 5.4.3 [16]
- Olimexino-STM32 [20]
- mbed LPC1768 Header Board [19]

This gave us a range of possibilities for investigation into different implementations and initial development.

Another bit of personally owned equipment that proved useful during development and testing was a USB to serial cable which was used to initially test the second camera with the 4D systems software [34], see section 8.1.1. This cable was also then used to attach the debug line for the rest of the development on the arduino platform.

The last bit of personally owned equipment that was used in this project was a micro SD card which was used for storing the images that come off the camera (see section 6.3). The mounting for this used during development was loaned from ECS stores, this was replaced during later stages of development.

4.5 Group Communication (ab)

4.5.1 Formal Meetings

For the duration of the project, we held weekly meetings on most Tuesday Afternoons from 4pm onwards in the Hartley Library. This would allow us to review progress made against progress expected, and to modify our project plan and allocation of work for the following week.

Minutes of these meetings would be stored in our repository. The timing of these meetings (the afternoon before our weekly meetings with our supervisor) will allow us to present a clear outline of new developments and our week-by-week plans to our supervisor. The meetings with our supervisor will allow the group members to assess their situation and re-organize themselves if necessary.

4.5.2 Methods of Communication (ms)

The group was prepared to use two primary methods of communication during the project in-between group meetings. This would allow the group to share any urgent information with each other easily. These methods of communication are discussed below:

4.5.2.1 E-mail

Email would be the primary form of communication between group members in between the weekly group meetings. It will be the responsibility of each group member to regularly check their e-mail and keep up to date with the latest developments of the project. Information which is useful for all members of the project or not directed at a specific individual would be sent by e-mail. E-mail also has the advantage of communicating software directly between members and leaving an audit trail to keep track of the project's progress.

4.5.2.2 Telephone

Each group member's contact phone numbers were shared at the beginning of the project to complement e-mail communications. A contact by phone number would be useful if very urgent information needs to be communicated immediately between project members. Although e-mail communication should suffice, telephone communication would be used to instantly contact a group member who may not be keeping up with e-mail exchanges.

4.5.3 Source Control

We decided to use version control software throughout the project, to manage everything from minutes to source code and payload schematics. As well as being a very useful tool to facilitate group work, it also helps us to meet our customer's request to deliver this as an open source project - at the end of the project, a repository could easily be gardened and presented, with the addition of appropriate open-source licenses at the end of the project.

Initially, we used a central Subversion (SVN) repository hosted on ECS' UGForge service. It was decided to use this as most of the group had experience using SVN. This worked well for most of the project, but unfortunately, something was committed to this repository that could not be open sourced (the customer's Ground Control Station software). This presented us with a problem, as although it would be trivial to revert this commit with the "$ svn merge" command, this creates a new commit, and the file in question would remain in the repository's history and could still be accessed.

In theory, it would be possible to remove this commit from SVN history by using the "$ svnadmin dump" command, filtering the offending commit out of the dumpfile, and regenerate the repository with the "$ svnadmin load" command. However, in practice, this was not possible as our repository contains binary information above 64kB, therefore the ASCII editor used to modify the dumpfile would cut off data in the binary file after the 64kB limit, rendering any following data useless.

It would be possible to start a new SVN repository and check in our repository up to but not including the offending commit, but we would lose all metadata (i.e. committer, commit time, etc.). Therefore, it was decided to switch to git, a distributed version control system (which would allow us to modify project history). Converting the repository using git-svn was a trivial process. Although not all of the group was confident using this tool, git lets a committer commit on behalf of another user. Moving to git also allowed us to use github, a hosting service, which is free to open-source projects (such as ours), which also provides us with some additional project statistics [21].

Chapter 5

Design Decisions

Throughout the design and implementation of the project there were many points at which decisions were made about the design of the system. This chapter attempts to give an insight into some of the design decisions taken throughout the project with a discussion of the alternative choices that were open to us at the time and the reasons why a particular choice was made.

This chapter also details thel milestones of the project. These milestones can be seen in section 5.9.

5.1 Camera Module (jc)

A range of different camera modules were researched and considered for use in the project.

5.1.1 Approach: Compact Digital Camera

A huge range of compact digital cameras are available for purchase and this was considered as a possible approach.

Advantages:

- High resolution of typically 5+ Megapixels.
- Fully encapsulated.
- High quality images possible thanks to automatic focus and zoom.

Disadvantages:

- Difficult/impossible to communicate with: most modern digital cameras have a USB socket for downloading images off the camera but it is much rarer to be able to send the camera commands (e.g. take photo) and for those cameras where it is possible the command structure is not usually simple.
- Requires USB host device in order to properly communicate as only USB host devices provide power via USB and can have USB peripherals connected to them.
- Relatively large and heavy compared to the other approaches considered.
- Expensive, prices starting around £60.

In order to communicate with a camera via USB a USB host is required. The process is usually fairly complex so driver is normally used. This is extremely difficult to implement on a device compact enough for use in this project.

5.1.2 Approach: USB Camera

Many USB webcams are available for purchase and this was considered as a possible approach.

Advantages:

- Low cost of typically around £15.
- High resolution of typically 1 to 3 Megapixels.
- Small size of usually less than 50mm by 50mm.
- Cabling included so can be easily positioned in the UAV.

Disadvantages:

- Often poorly documented: no datasheets or datasheets aimed solely at use with a computer.
- Difficult to communicate with: These devices are usually designed to be used with a computer so communication is usually done using a proprietary driver (a piece of software that defines how to interact with a peripheral). In order to maintain the proprietary nature of these documentation and/or source code is often not available for them.
- Requires USB host device in order to properly communicate as only USB host devices provide power via USB and can have USB peripherals connected to them.

- High resolution means longer total transfer times, including time to transfer from the camera and time spent on any compression/ manipulation.

In order to communicate with a USB camera a USB host is required, and as the process is usually fairly complex a driver is normally used. These drivers are fairly complex pieces of code and they are designed for use on computers, they are also often poorly documented so it would be extremely difficult to implement them on a device compact enough for use in this project.

5.1.3 Approach: Analogue Camera

Camera modules with analogue composite outputs are available, usually for cctv or surveilence purposes, and were also considered as an approach.

Advantages:

- Range of resolutions available.
- Small size: usually less than 50mm x 50mm.
- This is a mature market so there are plenty of options available.

Disadvantages:

- Awkward to communicate with: in order to decode the output of these devices we either need to convert from analogue to digital and then decode the signal (this type of camera usually outputs in composite or component format both of which are complex encodings, particularly when converted to digital) or put the signal through a decoder IC which are better documented but still complex, and the correct chip must be used.
- Require case and cabling for placement in UAV.
- Can be expensive at up to around £80.
- Often video rather than still cameras.
- Often black and white rather than colour.

There are a very wide range of cameras of this type available with different features however there is a significant amount of additional complexity introduced by the requirement to decode the analogue output into something that can be understood by a microcontroller.

5.1.4 Approach: Serial Camera Module

Some camera modules are available with UART rather than USB serial communications and these to were considered as an approach.

Advantages:

- Well documented: these cameras usually come with datasheets and are often designed for use with electronic projects.
- Easy to communicate with as they do not require a USB host or and analogue to digital conversion, additionally most microcontrollers have a built in UART port.
- Range of resolutions available.
- Small size: usually less than 50mm x 50mm.

Disadvantages:

- Require case and cabling for placement in UAV.
- Moderately expensive at around £40.
- The lack of case means that the camera is susceptible to shorts and static.

The ability to communicate via UART combined with the good documentation should allow for control to be established fastest with this camera type.

5.2 Approach Chosen: Serial Camera module (jc)

The camera type chosen was a serial camera as described in 5.1.4. We chose this camera type because it was the easiest to communicate with as it did not require a USB host or any kind of analogue to digital conversion and decoding. The serial camera type also has most of the advantages of the analogue camera type but is easier to communicate with.

The particular model chosen was uCam (microCam) or the "Camera Module - Serial JPEG TTL" from the "coolcomponents" website.

FIGURE 5.1: Photo of the 4D systems uCam, sourced from [13]

The feature set of the uCam is as follows:

- It can output images in both RAW and jpeg formats
- It can output RAW images at a range of resolutions:
  - 80 x 60
  - 160 x 120
  - 320 x 240
  - 640 x 480
  - 128 x 128
  - 128 x 96
- It can output JPEG images at a range of resolutions:
  - 80 x 64
  - 160 x 128
  - 320 x 240

  - 640 x 480

- It can output RAW images with a range of colour settings:
  - 2bit Gray Scale
  - 4bit Gray Scale
  - 8bit Gray Scale
  - 8bit Colour
  - 12bit Colour
  - 16bit Colour
- It will auto-detect baud rates from 14400 to 115200
- It has selectable baud rates up to 1228800
- Small physical size at 32mm x 32mm
- Well documented

The camera was chosen because this feature set meets the specification and also allows for additional functionality, such as setting the resolution, if the full feature set is exploited.

5.3 Local Image Storage (ab)

With a maximum image size of 640 × 480 pixels, an uncompressed JPEG image's file size would be approximately 900kB. This is greater than the internal memory on the ATmega168, ATmega328 and ATmega644 (1KiB, 2KiB, and 4KiB respectively) microprocessors that were used during development of this project, therefore it cannot be guaranteed that a complete single image can be stored locally, without adding some extra memory.
The advantages of being able to store a full image locally are:

- Would allow all original images taken during flight to be viewed once the UAV has been landed.
- Would make the implementation of automatic repeat requests (resending of any corrupted payload -> ground station packets) much easier.
- Customer mentioned that currently, customers take images with their retrofitted cameras, but store the images locally instead of wirelessly transmitting them. Keeping the functionality to store all images taken on-board and then plug something into the ground station to view these once the UAV has been landed would be desirable, but not essential (hence why this is not mentioned in the specification).

The disadvantages:

- The material cost of the payload (in both component cost and PCB area) would be increased.
- Addition of an extra component increases the project's complexity.

5.3.1 Approach: Flash Chip

One option considered was the implementation of an external flash memory chip (for evaluation purposes, the 4Mbit AMIC A25L040-F [41] was purchased)

Advantages:

- Non-volatile memory. Allows data to be retained even if power to the payload module is cut off.
- Excellent data integrity: 100000 Write/Erase cycles, and >10 years data retention.
- Relatively good operating current of 20mA makes it suitable for battery-powered application.

Disadvantages:

- Chips that contain a significant amount of memory storage (e.g. 1Gbit and above), come in packages that are awkward to prototype (BGA, TSOP)
- Interfacing this type of device with a PC (for debugging) would require a similar amount of work to the ground station software.
- Non-standard interface. The command set for the 4Mbit device is not the same as for a 1Gbit device.

5.3.2 Approach: SD Card

This is essentially the same as the flash chip above, but with an additional controller chip, and packaged in a standard case.

Advantages:

- Non-volatile memory. Allows data to be retained even if power to the payload module is cut off.
- Same data integrity as for the above item.

- Getting image data from the card would be as trivial as removing the card from the payload and inserting it into the ground station laptop. SD card readers are cheap, and ubiquitous.
- Good value for money, when compared to price-per-Mbit of the other two options under consideration. (For example, a 2GB microSD card (including microSD -> SD adapter) from Amazon costs £4.

Disadvantages:

- Open source libraries can only make use of an SD card's SPI Bus [15] interface, not the quicker (but patent-covered) four-bit SD bus.
- Only standard SD cards are supported, SDHC are partially supported, but not the more recent SDXC cards, which are increasingly popular.
- File system must be standard (FAT16 or FAT32)

5.3.3 Approach: SRAM

SRAM was considered as a possible option as we had all used this before.

Advantages:

- We have previous experience of interfacing these with AVRs from a second-year lab,
- This type of memory performs read/write operations much quicker than the other two options.
- Very low operating current of 10mA makes it suitable for this battery-powered application.

Disadvantages:

- Standard SRAM is volatile, so data is prone to being lost if the payload module power is lost, or the payload module is reset. (While non-volatile SRAM does exist, it is much more expensive per Mbit (£18/Mbit) than the other options under consideration.)
- Comparatively expensive
- Communication with the device requires connection to all data, address and control pins (or implementation of shift registers with the same desired effect), this would require more board space. (The other two devices can communicate via an SPI bus, a four wire connection)

5.3.4 Approach Chosen: SD card

This approach was chosen as these are very easy to interface with (standard libraries exist), decreasing our development time, and they essentially have the same advantages as Flash Chips, plus the ability to easily plug it into a PC.

5.4 Payload Controller Hardware (mh)

The hardware used to implement the payload controller is an important consideration.

The exact requirements for this module depend on other design decisions made, such as the method used to interface with the camera module. However since the module must communicate with the autopilot over a RS-485 serial connection at 38.4 kBaud this requires the module have access to a UART (Universal asynchronous receiver/transmitter).

5.4.1 Approach: Digital Signal Processor (DSP) Development Board

An appropriate Digital Signal Processor (DSP) development board would provide fast digital signal processing capabilities to the controller. This would be especially useful for the situations mentioned in section 6.5.

Advantages:

- DSPs often have very good analogue to digital conversion built in as standard. If an analogue camera is used a DSP may provide the necessary fast ADC without any extra components.
- Libraries for tasks such Fast Fourier Transforms or other signal and image processing related applications are often readily available for DSP chips. For some methods of performing progressive JPEG manipulation (see section 6.5) this could be useful.
- Large number of miscellaneous signal processing related hardware features built in which could be useful in as-yet unforeseen circumstances.

Disadvantages:

- DSP chips are often complex and may require significant work to produce a system running without a development board.
- DSP development boards often cannot be used effectively in a final version of a project due to size and power constraints.

- There is a steep learning curve to using these devices and the group has very little experience with them.

DSP boards are a specialist tool, which could be especially useful - if not required - if implementing certain image processing algorithms such as some of those detailed in section 6.5. The downside of this is a more complex overall system.

5.4.2 Approach: Atmel 8-bit AVR

The Atmel 8-bit AVR family is a well-known set of low cost, single chip microcontrollers. Specific parameters taken from [40].

Advantages:

- All members of our group have had experience working with AVR chips.
- Programming and debugging devices are readily available to us.
- Creating stand-alone systems easy: no need for large amounts of additional components.
- Commonly used devices mean large community built around the device family meaning support and additional libraries are more likely to be available.
- Some initial payload code available to us already implemented on a AVR, see section 4.4.
- Large range of different devices which work with the same programming tools and minimal code changes available.
- Many through-hole versions of the devices are available, making prototyping significantly easier (and cheaper, due to lack of need for breakout boards) than using surface mount components.

Disadvantages:

- Most 8-bit AVR devices are reasonably low speed (20MHz maximum), meaning they could be unsuitable for heavy calculations such as image processing.
- 8-bit AVRs do not have floating point processing units, making some image processing tasks much slower to complete on an AVR.
- Less communication options than some other, more complex, devices. This limits the number of peripheral devices that can be connected to the controller at once.

- Limited RAM available on the commonly used devices which are available in through-hole format. A common amount of RAM for a larger through-hole AVR is 4 kbytes.
- The smaller, cheaper, devices have reasonably limited program memory, with 64 kbytes being a typical amount for a larger through-hole device. This could be a problem if implementing large amounts of code.

AVRs are robust, widely used, general purpose microcontrollers. The large community surrounding them is a particular advantage, as it means many libraries and development boards are available for the devices. Our previous experience with the devices is also a key advantage, although it is important to be aware of the devices limits.

5.4.3 Approach: Atmel 8-bit AVR with Arduino Prototyping Platform

The Arduino platform is a low cost, open-source development platform for the Atmel AVR microcontroller family. The latest versions of the development platform consists of an ATMega328 microcontroller along with voltage regulators, inbuilt USB programmer and serial interface and 16MHz crystal. A software library for the device provides a wrapper around many of the complexities of the AVR platform, allowing more rapid development to be possible than with a pure AVR. A simple IDE is used to program the device and a wide range of extra libraries are available which target the platform. The physical platform is designed to be easily extended using expansion boards called 'shields'.

The advantages and disadvantages of the Atmel 8-bit AVRs mentioned in section 5.4.2 also apply.

Advantages:

- Just an ATMega328 'under the hood' meaning that the additional power of the chip can be leveraged if needed.
- Many common tasks such as setting the baud rate of the serial link are made very simple by the Arduino libraries, meaning development time can be spent working on more challenging features.
- Very widely used by hobbyists, meaning a large number of open-source libraries are available for the platform.
- Platform is easily available meaning others should more easily be able to replicate project. Widely used platform means plenty of support is available for others wishing to re-implement this project.

- Since only a 'shield' expansion board would need to be manufactured this would cut time otherwise spent designing and making board for AVR.

Disadvantages:

- Using the Arduino libraries means a certain amount of unused overhead is likely.
- The Arduino libraries hide much of the complexity of the device. This can mean advanced features are hidden or not exposed, negating some of the advantages of the library when advanced features are needed.
- Design would consist of two parts: Arduino board and expansion 'shield'. This could reduce durability and increase size.

Using an Arduino has the advantage of speeding up development at the beginning of the project. The large number of libraries built for the Arduino is another key advantage, as is the ease of re-implementation.

5.4.4 Approach Chosen

We chose to use an Atmel 8-bit AVR as the platform for our implementation of the payload controller with the use of the Arduino prototyping platform to help speed development. Our main reason for choosing this platform is because we already have experience in using this microcontroller, meaning less time to develop and test the system over a DSP based solution. Another key reason was the ease of implementing AVR based systems as stand-alone boards.

5.5 Communication between Payload Controller and Ground Station (mh)

One of the key tasks of the payload controller is to communicate with the Ground Station via the Autopilots 38.4 kBaud serial link. In order to fulfil the specification the payload controller must be capable of receiving messages/commands sent from the Ground Station image viewer software (see section 7) through the autopilot link, as well as be capable of sending data to the Ground Station Image Viewer.

The autopilot provides a number of ways to communicate between a payload module and ground station software, see section 2.4.

5.5.1 Approach: Shared Memory for Both Directions

One possible way of producing two-way communications would involve two sets of shared memory: one for ground station to payload messages and another for payload to ground station messages. Both the payload and the autopilot would need to poll the shared memory to check for new messages.

Disadvantages:

- The payload must access shared memory by sending a command to request a copy of its contents to be sent via the payload-autopilot serial link. This extra overhead would slow down communications (such as downloading an image).

5.5.2 Approach: Send Bytes Command and Shared Memory

The send bytes command could be used to send messages from the Ground Station to the payload, with shared memory being set by the payload to send messages back from the payload to the Ground Station.

Advantages:

- No overhead needed to get messages on the payload, they are sent directly from the autopilot to the payload, no polling needed.

5.5.3 Approach Chosen: Send Bytes Command and Shared Memory

Using shared memory for one direction and send bytes for the other seemed like a sensible choice due to the lower overhead needed.

5.6 Image Manipulation (ms)

The design of the progressive image display system to be implemented on the payload and the ground station required some important decisions to be made before development could begin. This included the type of image file which would be manipulated, how the image sent from the camera on the UAV would be accessed, and the method of storing the data and sending it to the ground station.

5.6.1 Image File to Manipulate

The first design decision was to choose the type of image file which would be the most practical to work with given the properties of the camera and the capabilities of the payload software.

5.6.1.1 Approach: Custom Raw Image Manipulation

This approach involves using the custom image manipulation algorithm as specified in section 2.1 on bitmap-type RGB images.

Advantages:

- Would be able to handle the images given by the camera regardless of the format the camera saves the image.
- Avoids quantization to allow for lossless compression: No information is lost when the image is compressed and decompressed.
- Uses a simple, reliable, and easy to understand image compression technique (Discrete Cosine Transform)
- Can more easily create a progressive display of the image due to having less information to decode.

Disadvantages:

- Restricted to decoding raw images; cannot deal with JPEG images.
- Does not compress the image as efficiently as more advanced image compression techniques.

5.6.1.2 Approach: JPEG File Manipulation

This approach involves manipulating JPEG standard image files instead of raw image files. This requires the microcontroller to work around the compression offered by the JPEG file standard (see section 2.2).

Advantages:

- Compatible with most standard camera outputs.
- Takes advantage of the reliable and efficient JPEG compression algorithm.

- Uses quantization to allow for a greater degree of image compression.

Disadvantages:

- Compression is lossy: image loses more and more information each time it is compressed (saved and opened).
- Requires more sophisticated and complex decompression algorithms.
- Requires more background research to understand and manipulate.

5.6.1.3 Chosen Approach: JPEG Manipulation

Before the camera was agreed upon, work began on the custom raw image compression algorithm. This allowed us to get a good idea of how the progressive display of the image would work in the final product. When the camera was chosen, it's output included JPEG image files. Since JPEG provides superior compression to simple custom methods this approach was chosen to help produce a fast initial system.

5.6.2 JPEG File Manipulation

This section weighs the decision of implementing a form of custom JPEG image manipulation to allow the image to be displayed progressively, as requested by the customer.

5.6.2.1 Approach: Custom JPEG Manipulation

This design approach details the choice to code an algorithm which would modify the image received from the camera in order to display it progressively. Ideally, this would send an extremely low detail image to the ground station which would progressively improve while more information is sent from the camera by implementing the higher frequency components of the image.

Advantages:

- Allows an image to be sent and be manually evaluated sooner.
- Allows the user to cancel images which begin to show unwanted information, saving processing time.
- Can possibly allow the image information to be optimized, sending only the information necessary for display.

Disadvantages:

- Requires effort and time, including research into the manipulation of the JPEG format.
- Benefit is minimal if the sending a simple image is already very quick.

5.6.2.2 Approach: Standard JPEG Transfer

This design approach simply displays the JPEG image from the camera as it is received.

Advantages:

- Requires little to no effort (already done by default).
- All the metadata of the image is kept unchanged.

Disadvantages:

- Takes time to send all the image data to the ground station.
- During the transfer period, state of image is unknown and can be wasted if image is undesirable.
- Image sends all the metadata, including metadata which can be ignored.

5.6.2.3 Chosen Approach: Custom JPEG Manipulation

Prior to knowing the exact speed it would take to send an image from the camera to the ground station, it was decided that custom JPEG manipulation would optimize the image further and help reach the target time specified (3 minutes, see 3.4.2). Therefore, tasks concerning the custom manipulation of the JPEG image were planned and worked on at the beginning of the project. However, when a working module was constructed, the time needed to send the image was already well within the time specified. Because this was the primary advantage of the custom JPEG manipulation process, the task's priority was dropped accordingly.

5.6.3 Entropy-Encoded Data Storage

Another major decision to make was the method of storing the entropy-encoded image data of the JPEG file, which is divided into chrominance pixels made up of 3 different components.(See section 2.1.4 for more information on chrominance pixels.)

5.6.3.1 Approach: Store As Chrominance Pixels

This method stores the chrominance pixels in the image one by one. The data is stored in a linked list of chrominance pixels, containing a certain number of Y values and one pair of Cb and Cr values.

Advantages:

- Makes the YCbCr colour modelling method more obvious. This method intuitively partitions the encoded image data using the YCbCr convention.
- Requires little data manipulation. Structures are made as input stream is read in, one at a time.
- Comprehensive memory allocation. Allocates the appropriate memory for one chrominance pixel as needed.

Disadvantages:

- Encoded image data becomes more difficult to apply progressive scans to.
- No spatial information is stored. Data is stored as a linear list of chrominance pixels instead of a table representing the image.
- Memory must be allocated dynamically. More vulnerable to insufficient memory crashes during the extraction process.

5.6.3.2 Approach: Store Components Seperately

This method of accessing the image data involves storing the three components Y, Cb, and Cr into tables.

Advantages:

- Makes progressive scanning easier. The three components can be selected individually.
- Information can be stored in a table to represent the image spatially.
- Memory can be allocated either dynamically or prior to the read process.

Disadvantages:

- Requires more manipulation of the entropy-encoded image data as it is read in. Components must be monitored to not lose chrominance pixel grouping.

- Chrominance information must be stored either unintuitive (a smaller table than the image) or inefficiently (an image sized table with empty values).
- Generally more difficult to monitor the information stored.

5.6.3.3 Chosen Approach: Store As Chrominance Pixels

The entropy-encoded data will be stored as chrominance pixels for the sake of clarity. It is also a method which could be coded more easily than separate component storage. This allowed the AVR to work with the progressive JPEG encoder sooner to avoid the task from overrunning.

5.7 Ground Station Image Viewer (ps)

5.7.1 Programming Language

The decision depend on the team member who has been assigned as a task leader. Because the program will be using to implement a GUI application, therefore it need an object oriented program for implementing this. The programming languages that will be look into are Python, Javam and C# programming language. There are many factors to consider before the final decision is decide.

Python Language

Python Language programming has variety of useful factors for network programming. The code on this language has been proved to be short and effective. The class `socket` has all the communication method which can be use. The TCP can be use by the program to send a text command on one machine to the another [29]. There is a makefile function which the user can use to avoid the missing line of data or an extra line from next cycle of data[29, 30]. It has support of GUI interface which the programmer can debug, and use a profiling tools [31].

Advantages:

- has a `socket()` class which can be use to connect to TCP/IP port
- support GUI: it can use to create and debugging window application
- can run with c code

Disadvantages:

- new to the developer
- might be time consuming to learn if the developer has a same advantages in a familiar software

Java Language

Java language is an object oriented program which can be use in any kind of operating system. It derives some syntax from c/c++ which the developer is familiar to. This language has been using widely for applications and gaming and it also support the TCP class. Java has two classes for TCP/IP, which are `Socket`, and `ServerSocket`. At both ends of the communication can be identified by IP address and port number [32].

Advantages:

- it object oriented program
- `Socket` and `ServerSocket` class support TCP/IP
- support GUI application
- has `graphic` class which can use to display images

Disadvantages:

- it can not inherit other language code to use in the program
- the developer is not that familiar with the program
- limit the effectiveness of customs protocols [33]

C# Language

C# language is an object oriented program which can be use in any operating system. It has ideally similar Socket class as the Java programming has. It also support the window application program. The program has been use by the developer, therefore it can save time during the implementation of the program.

Advantages:

- familiar to the developer
- `Socket` and `SerialPort` class support the TCP/IP connection
- has graphic class which can use to display image

- can make and debug window application

The program that will be using to implement the program is C# language. This is because the program is familiar with the user so it will save time of studying a new language. It also support TCP/IP and window application which are main classes needed to implement the program. Although Java and Python has been use by many papers and book on network programming, but it need to take time to learn a new language so the developer decided to use C# [29, 30, 32, 33].

5.7.2 Approach: Different C# .NET Classes

The C# language on Microsoft Visual Basic Studio is a development environment for creating our Ground Station Image Viewer application. Choosing the right class to connect to the port helps the developer save time to program the application.

5.7.2.1 SerialPort Class

The `System.IO.Ports` namespace contains classes for controlling serial ports. The `SerialPort` class is the most important one. It has ability to synchronous and event-driven I/O, access to pin and break states, and access to serial driver properties[23]. By using this class, the COM port setting has to be memorized which will be simple to load and saves from/to disk.

```
SerialPort( portName, baudRate, parity bit, dataBits, StopBits )
SerialPort.Read(Char(),Int32,Int32);
SerialPort.Write(Char(), Int32, Int32)  ;
```

Listing 5.1: Serial Port class connection read and write method

Advantage

- can change baud rate, handshaking, parity bit, and stop bit
- can change the internal of a hardware
- can send and receive data from a defined connection
- can program hardware

Disadvantage

- need to set up all the component (baud rate, handshaking, parity bit, and stop bit)
- the set up make the code long and might cause error
- the setting of the signal out must be the same as the set up in the UAV

5.7.2.2 Socket Class

The Socket is more programmer friendly, robust and a high level connection class. The program can easily connect to the port by using the method in Listing5.2 . This class sends and array of bytes to the console port and to the data stream port and receive data from the port by code in Listing5.2.

```
Connect (String host name,Int32 port number)
Send(byte[] command, Int32 length, SocketFlags)
Receive(byte[] data, Int32 length, SocketFlags)
```

LISTING 5.2: Socket class connect receive and send method

Advantage

- can set up the connection by simply stated the host name and port number
- can connect to TCP/IP port
- it uses the set up of the port, therefore the parity, stop bit, and the baud rate are the same
- can send and receive data from the port
- can use delegates which give more flexibility on synchronous control of sockets [35]

Disadvantage

- can not program hardware
- can not set up the baud rate, stop bit or any other properties of the hardware

The socket class was the class chosen for the UAV port connector. It support TCP connection between the client and the server to do low level communication work[27]. The changes of class improve efficiency of connection between port and the program. The baud rate and extra bits are automatically adjusted to be the same as the port, so the setting is identical between the input and output.

5.8 Physical Implementation (ab)

Our specification states that the final module delivered to the customer should be on PCB, if time permits. Using existing breakout board type devices (such as the Arduino) and a daughter board would make the project far cheaper for one-off applications (one-off PCBs can be rather expensive) while still fitting this specification point, but would

increase construction time to reproduce it, and increase the weight and volume of the module, two critical components of the project.

However, producing a bespoke PCB has the advantage of being smaller and lighter, and assembly could be handled by an external company.

Producing a stripboard solution would be implemented only if time and budgetary constraints are against us: whilst being the cheapest option, it would be larger than a PCB solution and not as robust.

5.8.1 Power Source

We also need to consider how such a module will be powered, and plan for both the following considerations:

5.8.1.1 Approach: Battery Powered

Providing a header on our final payload to attach an external power source should the Autopilot not be able to provide enough power would seem a sensible option. As this battery is likely to be Lithium-Ion or Lithium-Polymer, implementing a charging circuit may be considered, but would be dangerous to test. (Lithium Polymer batteries are prone to damage if incorrectly charged)

5.8.1.2 Approach: Autopilot Powered

It should be possible to source enough power from the Autopilot, which can comfortably provide up to 200mA at 3.3V. Designing our module to be as low-powered as possible would be beneficial (this extends the time the autopilot will be able to fly)

5.9 Milestones - (mh)

In order to assess our progress throughout this report the tasks required to implement our system have been broken up into milestones as shown below. Each milestone refers to a real and measurable step towards a complete system.

To assist comprehension the milestones have been arranged into a number of categories, relating to the way in which the project work has been broken up.

5.9.1 Camera Module Communication Milestones

5.9.1.1 Milestone: Basic Camera Connection

- A basic connection to the camera should be established where the camera responds correctly to connection commands.

5.9.1.2 Milestone: Image from Camera

- Milestone 5.9.1.1 above should be completed.
- An image should be shown to have been downloaded from the camera - this should be repeatable without need to reset the camera.
- This image may be of any resolution and colour type.

5.9.1.3 Milestone: Changing Camera Resolution

- Milestone 5.9.1.2 should be completed.
- An image should be shown to have been downloaded from the camera as in the *Image from Camera* milestone.
- This image should be of a specified resolution which should be changeable.

5.9.1.4 Milestone: Changing Camera Colour Type

- 5.9.1.2 milestone should be completed.
- An image should be shown to have been downloaded from the camera as in the *Image from Camera* milestone.
- This image should be of a specified colour type which should be changeable.

5.9.1.5 Milestone: Camera Payload Controller Implementation

- Milestones 5.9.1.2 and 5.9.3.3 should be completed.
- The implementaion of milestone 5.9.1.2 should be implemented on the same hardware as the final payload controller will use.

5.9.1.6 Milestone: JPEG Header Extractor Integration

- A functional JPEG header information extractor must be implemented on the AVR microcontroller.
- This feature must be tested with the ground station progressive image viewer. (requires milestone 5.9.5.5 to be fully tested)

5.9.2 Payload to Ground Station Communications Milestones

5.9.2.1 Milestone: Payload Responding to Transmit Tokens

- The payload module should respond to transmit tokens sent by the autopilot.
- The `payload[0].timeout` command when input into the customers ground station software should return `FALSE`.

5.9.2.2 Milestone: Payload Setting Shared Memory on Autopilot

- Milestone 5.9.2.1 should be completed.
- The payload module should set the `MEM_BYTES` shared memory assigned to it to a recognizable value.

5.9.2.3 Milestone: Payload Receiving Messages from Ground Station

- The payload should receive and respond to data sent by the `payload[0].send_bytes` command when executed on the ground station.

5.9.2.4 Milestone: Image Sending to Ground Station via Autopilot Link

- Milestones 5.9.1.5 and 5.9.2.2 should be completed.
- An image taken with the camera module should be sent to the ground station by the payload controller via the autopilot link.

5.9.2.5 Milestone: Triggering Image Capture from Ground Station

- Milestone 5.9.2.3 should be completed.
- A specified set of data sent using `payload[0].send_bytes` from the ground station should trigger the payload to take a picture.

5.9.2.6 Milestone: Changing Camera Resolution from Ground Station

- Milestones 5.9.1.5, 5.9.2.3 and 5.9.1.3 milestones should be completed.
- A specified set of data sent using `payload[0].send_bytes` from the ground station should change the camera resolution with the same outcomes as the *Changing Camera Resolution* milestone.

5.9.2.7 Milestone: Changing Colour Type from Ground Station

- Milestones 5.9.1.5, 5.9.2.3 and 5.9.1.2 milestones should be completed.
- A specified set of data sent using `payload[0].send_bytes` from the ground station should change the camera resolution with the same outcomes as the *Changing Camera Colour Type* milestone.

5.9.2.8 Milestone: Progressive Image Download

- Milestone 5.9.2.4 should be completed.
- An image should be downloaded progressively from the payload to the ground station.

5.9.3 SD Card Milestones (ab)

5.9.3.1 Milestone: Basic SD IO

- A file must be written to, then read from, an SD card using a microcontroller.

5.9.3.2 Milestone: File Numbering System

- Milestone 5.9.3.1 needs to be completed
- A method which allows you to store a file under a new, unused filename must be implemented.

5.9.3.3 Milestone: Full Image to SD card

- Milestones 5.9.3.3 and 5.9.1.2 need to be completed
- A full image needs to be captured from the camera, and stored on an SD card, using some prototype hardware solution.

5.9.4 Physical Implementation Milestones

Many of the milestones listed above rely upon a working physical implementation of the system at each stage so these will not be duplicated. However, there are a few milestones which are specific to the physical implementation of the system.

5.9.4.1 Milestone: Payload Breadboard Prototype

- A finalized schematic.

5.9.4.2 Milestone: Payload PCB Design

- A finalized PCB design ready to be sent for manufacture.

5.9.4.3 Milestone: Complete System on PCB

- A working system implementation on PCB.

5.9.5 Ground Station Image Viewer Milestones

5.9.5.1 Milestone: Basic Dummy Server Communications

- Some prototype programs should be produced to help understand TCP/IP communications. One server program serving some test data over a TCP/IP port and one client program connecting to this port and sending/receiving test data.

5.9.5.2 Milestone: Communications with Customers Ground Station Software

- Milestone 5.9.5.1 should be completed.
- Communication with the customers ground station software via TCP/IP console and data port. Data should be streamed from some known source and verified correct and commands should be sent via the port and verified that they execute in the program.

5.9.5.3 Milestone: Receive Image from Payload

- Milestone 5.9.5.2 should be completed.
- Data sent from the implementation of 5.9.2.4 should be decoded and saved/displayed on the ground station in some way.

5.9.5.4 Milestone: Functional User Interface

- A functional user interface that allows users to take images and view them using the implementations for the milestones described above.

5.9.5.5 Milestone: Progressive Image Viewer

- Image must be displayed progressively, incorporating higher frequency elements as time goes on.
- Image viewer must be tested with the extractor implemented on the AVR (requires milestone 5.9.1.6 to be fully tested)

Chapter 6

Implementation - Payload

6.1 Overview (mh)

The payload controller module is an important part of the system, responsible for interfacing with the camera module and communicating with the ground station image viewer software via the autopilot. Since this single module encapsulates a significant amount of the complexity of the project it was deemed sensible to split it up into sub-modules, which could be worked on in parallel by different members of the team.

With this in mind the payload controller module was split into four main submodules:

- Camera Module Communication
- Communication with Ground Station via Autopilot
- SD Card Image Buffering
- Progressive JPEG Manipulation

6.2 Camera Module (jc)

The first step in the implementation of the camera module was to verify the communication with the camera by talking to it via a pc (milestone 5.9.1.2). Once this step was complete the next step was to implement communications between the camera and a microprocessor, with the pc for debugging. With the microcontroller able to communicate with the camera this module was ready for integration with the payload (milestone 5.9.1.5).

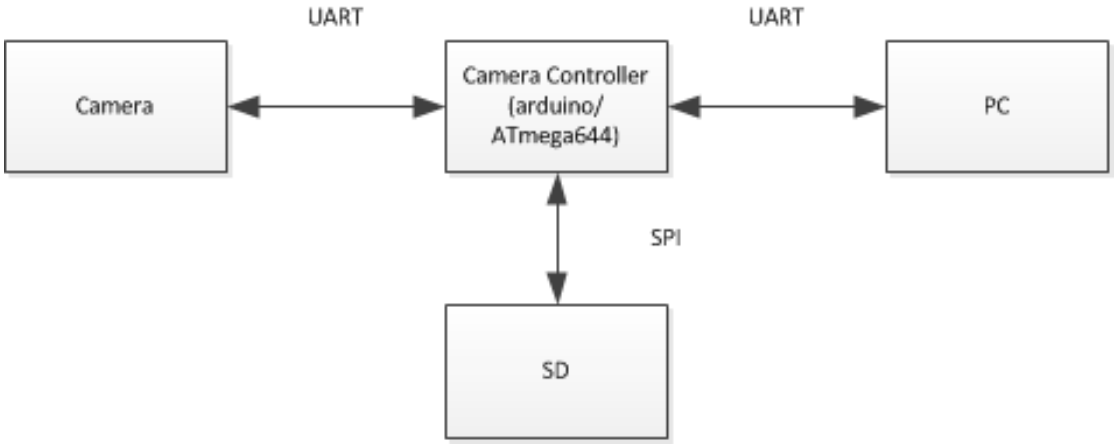


FIGURE 6.1: Block diagram of the camera module showing where communication takes place

Figure 6.1 shows how the camera module is broken down separate to the rest of the system. The camera and the camera controller communicate to set up the camera and get images, the camera controller then stores the image to the SD card and debug messages are sent to the computer during these processes so that the camera is less of a black-box.

During the development of the system 2 cameras (uCam [13]) died on us. The first very early on in development and the second after the system had been effectively finished apart from a few minor tweaks and implementation of some low priority functionality. These component malfunctions were most likely due to the uCam being sensitive to static so after the first one failed we were careful to only handle the second one on an anti-static mat whilst wearing a wrist-strap, however someone must have gotten careless towards the end of the project.

6.2.1 First camera

The first attempt at communicating with the original camera was done using the arduino uno microcontroller board. The arduino was chosen as the platform for this first communication because it is simple to program and its programming environment provides various useful libraries, for example for serial communication [16]. The hardware also provides an easily accessible serial port.

The first implementation of this code managed to occasionally sync (see figure 6.2) with the camera (milestone 5.9.1.1) but would generate errors when trying to send any further commands. After examining the code and the camera's datasheet it was discovered that the camera was only syncing occasionally because the arduino was using a slower baud

rate than the camera could auto-detect, after increasing the baud rate on the arduino the camera would sync every time but the code still generated errors after this point.

It was at this point that the first camera broke. It was no longer syncing (or sending back anything at all, verified by oscilloscope readings), where before it had been doing so reliably and so it was checked with the 4D systems software and shown not to be working 8.1.1.

6.2.2 Second camera

Once it was established that the first camera was dead a couple of cheaper surface mount camera modules were purchased as possible replacements however making a successful physical (and therefore also communication) connection to these components proved excessively difficult so a new camera of the same type as the previous one [13] was ordered. The successfull connection test is seen in section 8.1.2, with debug info observed using a serial monitor.

6.2.2.1 Serial cable to computer

The operation of the new camera module was verified using a usb to serial cable connected to a pc which was running the sample program which was provided by the camera's manufacturer [34]. Using this set up it was shown that the camera was working and that it was possible to get images from it, see section 8.1.1.

6.2.3 Arduino implementation

With the operation of the camera verified it was reconnected to the arduino board and the code run again with the same result as before: it would correctly sync (see figure 6.2 and section 8.1.2) but would generate errors when trying to send any further commands. On inspection of the code it became clear that this was because the same serial line was being used for both communication with the camera and debug messages and that these debug messages were interfering with further communications. Debug messages were therefore moved to a software serial line and sent to the computer via the usb to serial cable and observed using a serial monitor.

The code is broken down so that there are functions for each type of command that the camera can receive, functions that verify the responses from the camera and functions that combine these together in the correct sequence in order to perform a useful task with the camera.

6.2.3.1 Camera synchronisation

The first task in communicating with the camera is to synchronise the serial channel, the uCam datasheet [13] gives the correct protocol to do this.

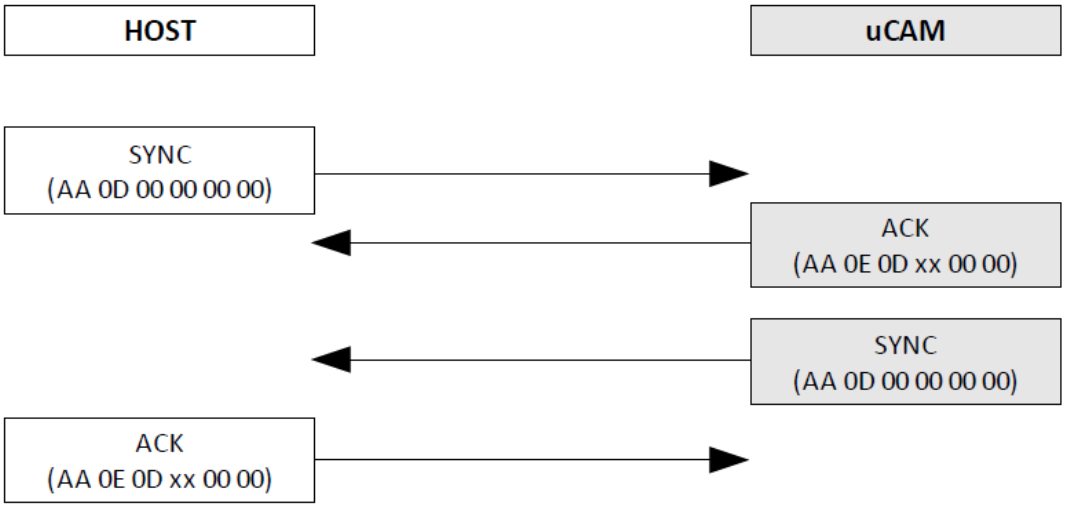


FIGURE 6.2: Command protocol for synchronisation, sourced from [13]

It should be noted that in 6.2 the first sync sent from the host will be repeated until an acknowledgement is received, with everything working correctly this process usually takes 3 or 4 syncs before an ACK is received.

The inclusion of a separate debugging line allows for messages at each stage of this process to verify that it is connecting correctly and to indicate where any errors may have occurred. With all debug messages enabled this function will output "Sending syncs" followed by a "." for each sync command sent and then "ACK received" and "SYNC received", the main code will then output a message to the effect that contact has been successfully established.

6.2.3.2 Taking a snapshot

With the camera successfully synchronised the controller needs to be able to trigger the camera to take a photograph and then retrieve said photograph, again the uCam datasheet [13] gives an example of how to implement this.

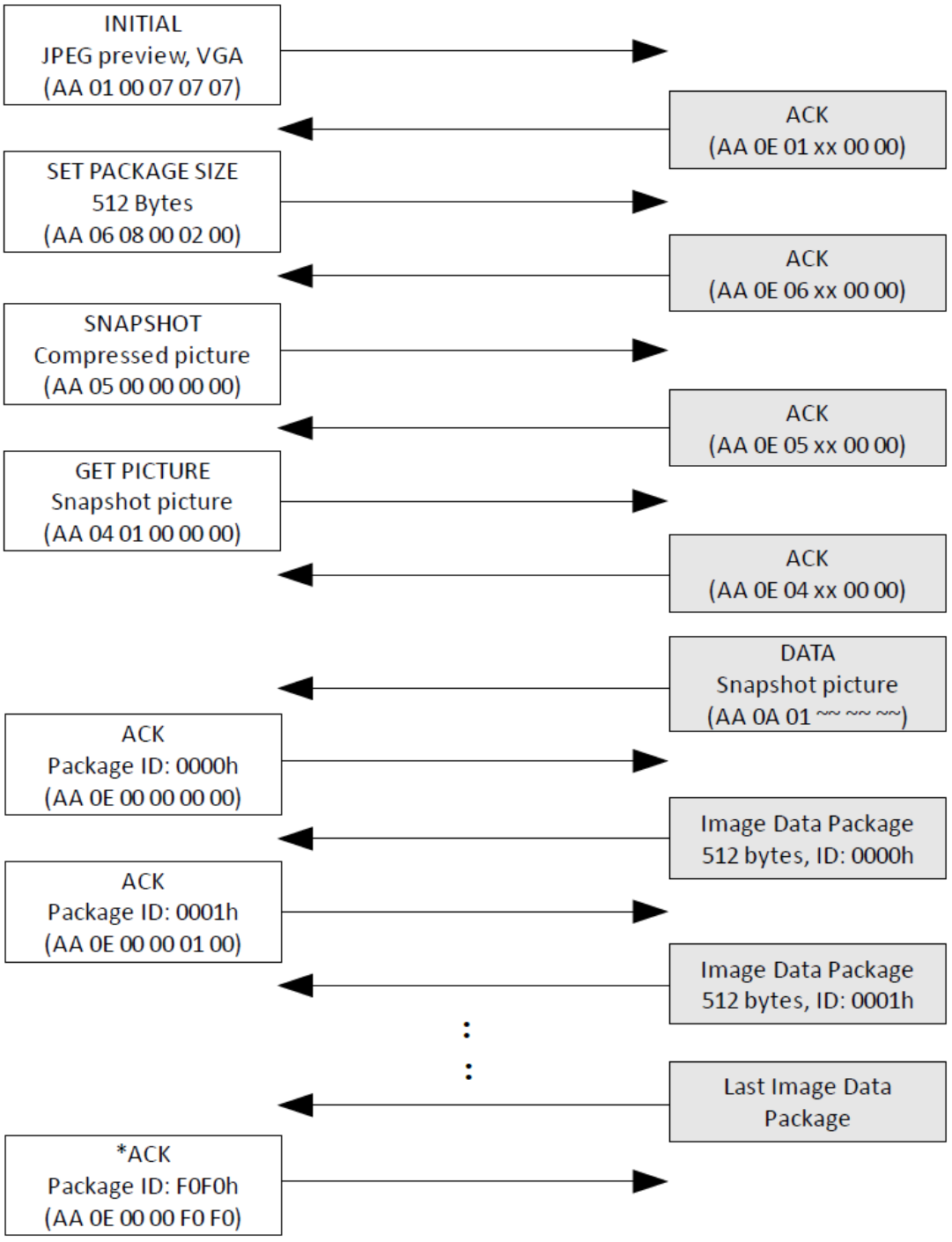


FIGURE 6.3: Command protocol for taking and retrieving a jpeg snapshot at 640x480 resolution, sourced from [13]

It should be noted in 6.3 that the values in the commands are for the example and are not necessarily the values used in the code.

The "INITIAL" command has parameters that allow the image type and resolution to be set, in 6.3 the image type is set to JPEG and the resolution is set to 640x480, these

values are kept as the standard in the code.

The "SET PACKAGE SIZE" command is only relevant for jpeg images and set the size of the data packages that are sent when one requests an image. In 6.3 this value is set to 512 bytes however in the code this is set to 64 bytes as it was originally thought that the system might have to transmit these packages as soon as receiving them so they needed to be a reasonable size to fit in with the timing between transmit packets sent by the autopilot.

The "SNAPSHOT" command tells the camera to take a single picture, its parameter sets whether the image is jpeg compressed or raw, in 6.3 this is set to be a jpeg image and this is left as the default in the code.

The last command sent by the controller is the "GET PICTURE" command which tells the camera to send image data, it has one parameter which sets which image is sent. In 6.3 this is set to send the snapshot image and this is also how it is set in the code. As well as sending back an ACK this function will also send back a "DATA" message which tells the controller what type of image is being returned and how large it is.

Once the controller has acknowledged the "DATA" message the camera starts to send data packages, each of which needs to be acknowledged, until the entire image is transferred. Whilst this process is going on the data packages are saved to an SD-card as described in section 6.3.

The use of the debug communication channel again allowed for detailed analysis of this process with the maximum amount of debug messages being one before each command is sent saying what that command is and another message once and ACK is received, there is also then a message with the size of the data in and then messages with the package number and the byte number within that package, the data can also be output to the debug channel, see section 8.1.3.

6.2.4 Miscellaneous Problems

Before the SD-card was implemented the sending of the data directly via the debug channel was attempted and so was its reconstruction into a jpeg image on the computer, this could not be made to work however, see section 8.1.3. On looking at the data as it came off the arduino via the software serial debug line it became clear (particularly when looking at the counting values, whose values were already known) that the data was not being transmitted reliably. This could have been because of noise on the line, the fact that the software serial line is communicating via pins on the arduino that aren't optimised for the function or simply failings in the arduino software serial library [14]. For this reason this method of getting an image was abandoned, this also had to be taken into consideration whilst thinking about the integration of this part of the system:

the software serial line is not a reliable method of transmitting the data so some other method would have to be used.

Another problem that was noticed after the camera control had been integrated into the payload controller, as described in section 6.6, was that every other time the system was powered up the camera would require a reset/power cycle. This was discovered to be because the Rx wire to the camera was left floating, this was fixed by the addition of a 10kΩ pull-up resistor, see appendix E for schematics.

It was also noticed quite early on that the camera required 3.3V logic [13] levels but the arduino outputs 5V logic levels, the arduino could receive logic at 3.3V easily enough but the 5V output was causing occasional errors. In order to fix this problem the 5V output from the arduino was taken through a potential divider of a 10kΩ resistor and a 3V zener diode.

6.3 Interfacing the Arduino with an SD card (ab)

Interfacing the arduino with an SD card is made possible using the arduino SD card library [15]. Using an Arduino Uno, a multiple-size SD card socket from Zepler stores, and the "ReadWrite" test program provided in the arduino-022 IDE, we connected the following Arduino pins to the following SD card pins:

- Arduino GND - SD GND (pins 3 and 6)
- Arduino 3V3 - SD Vdd (pin 4)
- Arduino Digital pin 11 - SD MOSI (pin 2)
- Arduino Digital pin 12 - SD MISO (pin 7)
- Arduino Digital pin 13 - SD SCLK (pin 5)
- Arduino Digital pin 4 - SD CS (pin 1)

Communication with the SD card only works in SPI mode, unfortunately the built-in SD card library does not support 1-wire or 4-wire SD mode.

6.3.1 File Naming System

This example program is useful, but only allows us to write to one file name at a time. Therefore, a method was written, see appendix C.3, lines 62-68 that detects all of files present on the SD card, increments the filename, and writes the new data to that filename.

6.4 Communication with Ground Station via Autopilot (mh)

Considering the overall aim of this project: to produce a system by which images can be downloaded over-the-air from a payload module to a ground station, some method of communicating between the payload module and ground station is an essential component in the system.

The specification requires the payload module to communicate with the ground station using the autopilot's payload module interface (discussed in section 2.4.1).

6.4.1 Existing Code

Our customer had provided us with some payload module communication AVR code - written for a ATMega168 - for communicating with the autopilot. This code was the basis on which the payload controller's communication link was built.

The code provided a number of useful utilities:

- Basic connection to the autopilot, including responding to transmit tokens.
- Ability to set shared memory on the autopilot.
- Ability to receive messages sent from the Ground Station to the autopilot.

This base code was modified slightly after a bug was found in its handling of the transmit enable signal. The RS485 communication protocol used for the autopilot-payload link (as described in section 2.4) requires a 'transmit enable' signal to be asserted when the payload is transmitting. This signal should be asserted just before data is to be sent and cleared just after. However, the original payload base code cleared this signal in an interrupt service routine (ISR) which fired after the transmit buffer of the UART was ready to accept new data. Since this transmit buffer would be ready to accept new data before the data was actually sent over the physical connection this lead to the transmit enable signal being cleared before all data had been sent, causing strange behavior on the RS485 link where the signal would decay over a significant period of time on transitions from high to low. This bug did not seem to cause any problems, and the behavior was only noticed when testing the system with an oscilloscope. It was considered sensible to fix the bug in case it did cause problems later.

The fix for this problem was reasonably simple: a new ISR was set up which fired only when the current transmission had actually completed, and the command to clear the transmit enable signal was moved into this ISR.

6.4.2 Establishing Contact with the Autopilot (mh)

The first step in implementing the communications link was to establish contact with the autopilot using the existing module code, the relevant milestone being 5.9.2.1.

A problem was encountered during this step where the payload would not respond to a transmit token in any way. Debugging this problem with a oscilloscope showed that the autopilot was not sending transmit tokens.

After spending some time ruling out problems with our own design that could be causing this, including looking at the RS485 chip in case it was conflicting with the autopilots serial bus or malfunctioning, we contacted our customer and queried whether it could be a problem with the autopilot itself.

Our customer responded by acknowledging that it was a problem with the autopilot and providing us with an updated firmware for the autopilot which would send the transmit tokens correctly.

After this bug was fixed the payload responded as expected, section 8.2.1 describes the successful testing of this part of the implementation.

6.4.3 Setting Shared Memory on the Autopilot (mh)

Once basic contact had been established the next task was to set shared memory on the autopilot - as per milestone 5.9.2.2. The existing code provided allowed this to be completed quickly using the built in functions. Section 8.2.2 details the tests carried out to validate this was working.

6.4.4 Recieving Messages from the Ground Station (mh)

It was important to ensure that the payload recieved messages correctly from the ground station before continuting with the implementation of this section, the customers provided code allowed this to be completed quickly also. Section 8.2.3 describes the steps taken to test this.

6.4.5 UAV Camera Communication Protocol (mh)

As discussed in section 5.5 it was decided that our communications protocol would use shared memory and *send_bytes* commands, allowing two way communications between the payload controller and ground station software to be established. With the ability to receive and send data with these messages tested as described above, implementation

could now begin on implementing a communications protocol used to talk between the our ground station image viewer software and the payload.

The method through which this shared memory is accessed via the ground station image viewer is discussed in chapter 7.

This two way communications link is the interface between the payload and the ground station software, so some standard protocol was required. It was decided that a message based system would be used, with the messages from the ground station to the payload module being sent using *send_bytes* and the messages sent from the payload to the ground station being put into shared memory. Each message is composed of two elements, one byte for the message ID - unique to each type of message - and a variable number of data bytes (depending on the message type.) The different message types are detailed below:

Messages sent from Ground Station To Payload

- **Take Picture**
  - *Data:* None
  - Prompts payload module to capture an image and save it to the SD card.
- **Image Download Request**
  - *Data:* Image ID
  - Requests the payload send the image with ID *Image ID* to the ground station. This message allows any image stored by the payload module to be downloaded over the connection, increasing flexibility.
- **Configure Camera**
  - *Data:* Colour Type, Raw Image Resolution, JPEG Image Resolution
  - Sets the image resolution and colour mode of the camera. Only the JPEG mode has been tested so far.
- **Cancel Download**
  - *Data:* None Resolution
  - Cancels the current download taking place.

Messages Sent from Payload to Ground Station

- **Picture Taken**
  - *Data:* Image ID

  - Informs the ground station software that an image has been taken and saved to the SD card. *Image ID* is the ID of the image that has been saved to the SD card.

- **Image Download Info**

  - *Data:* Number of Image Packets
  - Sent by the payload after a successful *Image Download Request* message from the ground station. Informs the ground station how many *Image Data* packets to expect.

- **Image Data**

  - *Data:* Packet Number, Image Data
  - This message contains an amount of actual image data. Sent after a *Image Download Info* message which is in turn in response to an *Image Download Request* message. The whole image is sent over *Number of Image Packets* packets (as defined by the *Image Download Info* message.) *Packet Number* informs the ground station which of these packets the message is carrying. *Image Data* contains the actual image data for this packet and is variable size, with a maximum size of 50 bytes.

Implementing this communications protocol was a significant challenge and was a major component of the payload module implementation.

Our implementation for the payload follows an event loop style, where an initial series of steps are preformed initializing the camera and SD card, after which the code then enters a continuously running loop which checks to see if any messages have been received from the ground station.

When a message is received from the ground station (as sent by `send_bytes` command) a switch-case conditional structure then decodes which message had been sent and initiates the appropriate function on the payload (for example calling the camera code to take a picture when sent a *Take Picture* command).

The basic sequence for sending data from the payload controller to the autopilot is to use the utilities provided by the customers existing code to place messages into the autopilots shared memory to be accessed by the ground station, as described earlier. Only one set of shared memory is used, and is overwritten every time a new message is to be sent from the payload controller to the ground station. Other features such as configuration of camera resolution are implemented using this basic mechanism, figure 6.4 describes many of the communication sequences that can occur.

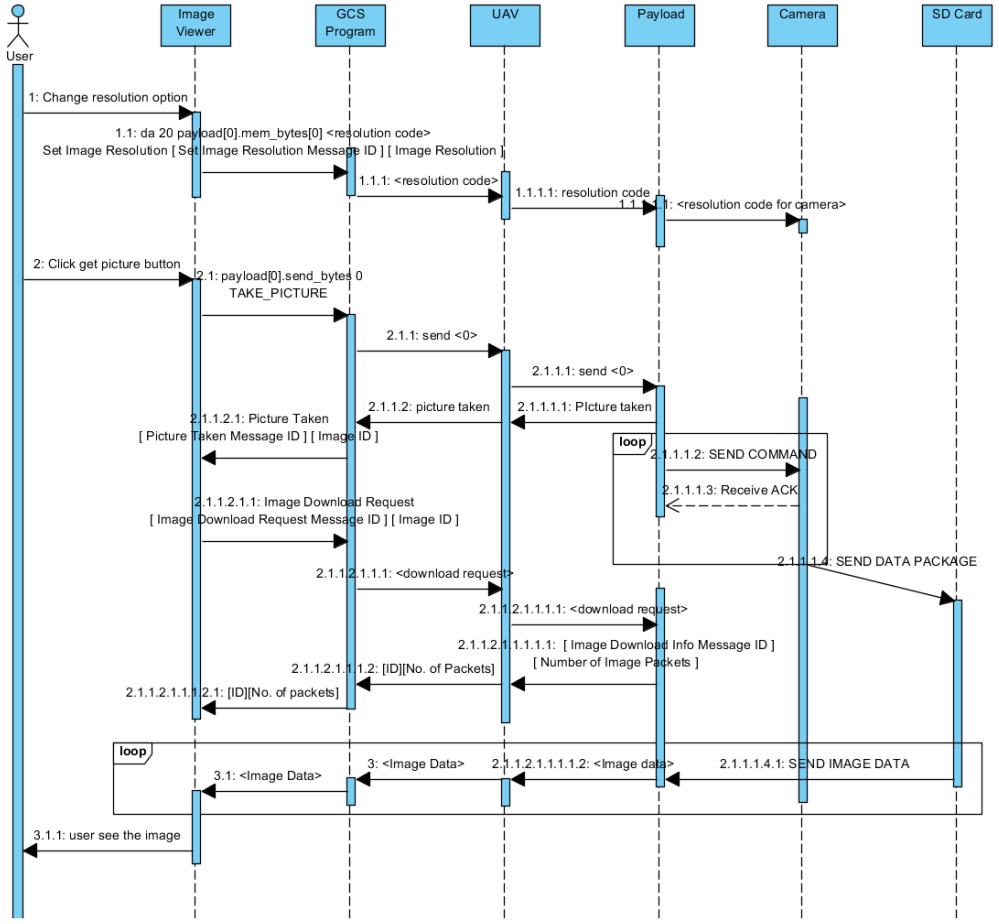


FIGURE 6.4: Sequence Diagram of the Data flow

Image data is also sent in the same basic manner, however the image is broken up into 'packets' of data which can be placed into the shared memory of the autopilot one at a time, as described by the *Image Data* message described above. One packet is sent for every transmit token sent by the autopilot, ensuring that the payload module does not saturate the autopilot link with data. While packets are being sent, the main event loop still checks for messages being sent from the ground station, allowing functionality such as the cancel download message to be implemented. In this way the payload module can transmit a whole image over the autopilot connection.

The testing of the image sending system is described in section 8.2.4, verifying the milestones as described.

Verification of milestone 5.9.2.6 (changing image resolution) can be seen in test 8.1.4. Unfortunately changing colour type (milestone 5.9.2.7) was not fully implemented due to time constraints.

6.4.6 Problems Encountered (mh)

A number of problems were encountered and overcome during the implementation of the payload-ground station communication code, a few of the most important are reproduced here.

Autopilot Bug (ab)

The Autopilot sends "Transmit" tokens to the payload module every 20ms, to which a payload module sent an "ACK" token, even when it does not transmit any data. We encountered a problem whereby if our we tried to send any data to the payload during an "ACK", the autopilot would stop sending any "Transmit" tokens at all, effectively cutting off all communication to the payload module.

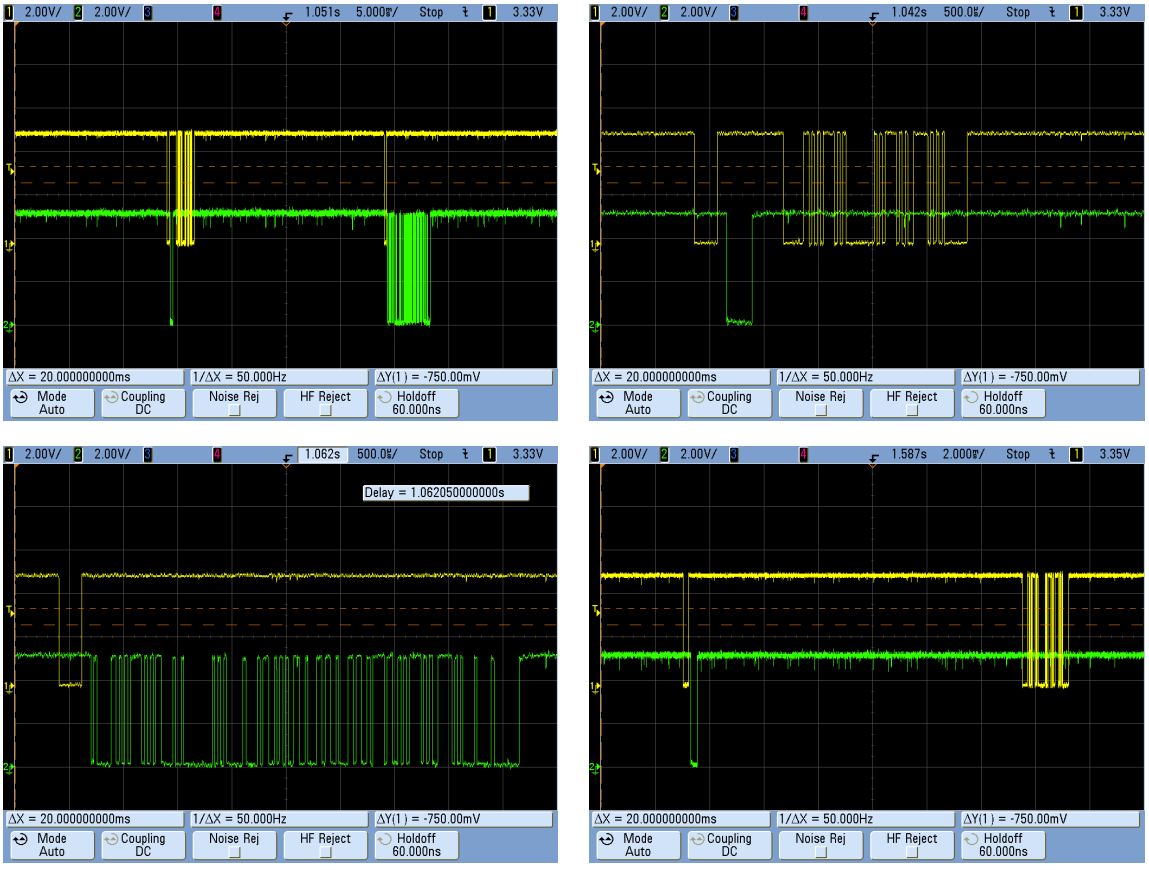


FIGURE 6.5: Oscilloscope traces for the situation where the autopilot does not break: In yellow is the Autopilot TX, in green the Payload TX. In this situation, after an ACK token is received by the autopilot, a SEND_BYTES instruction is sent. On the next transmit token, a series of bytes is sent to the autopilot, and the autopilot continues to send Transmit tokens.

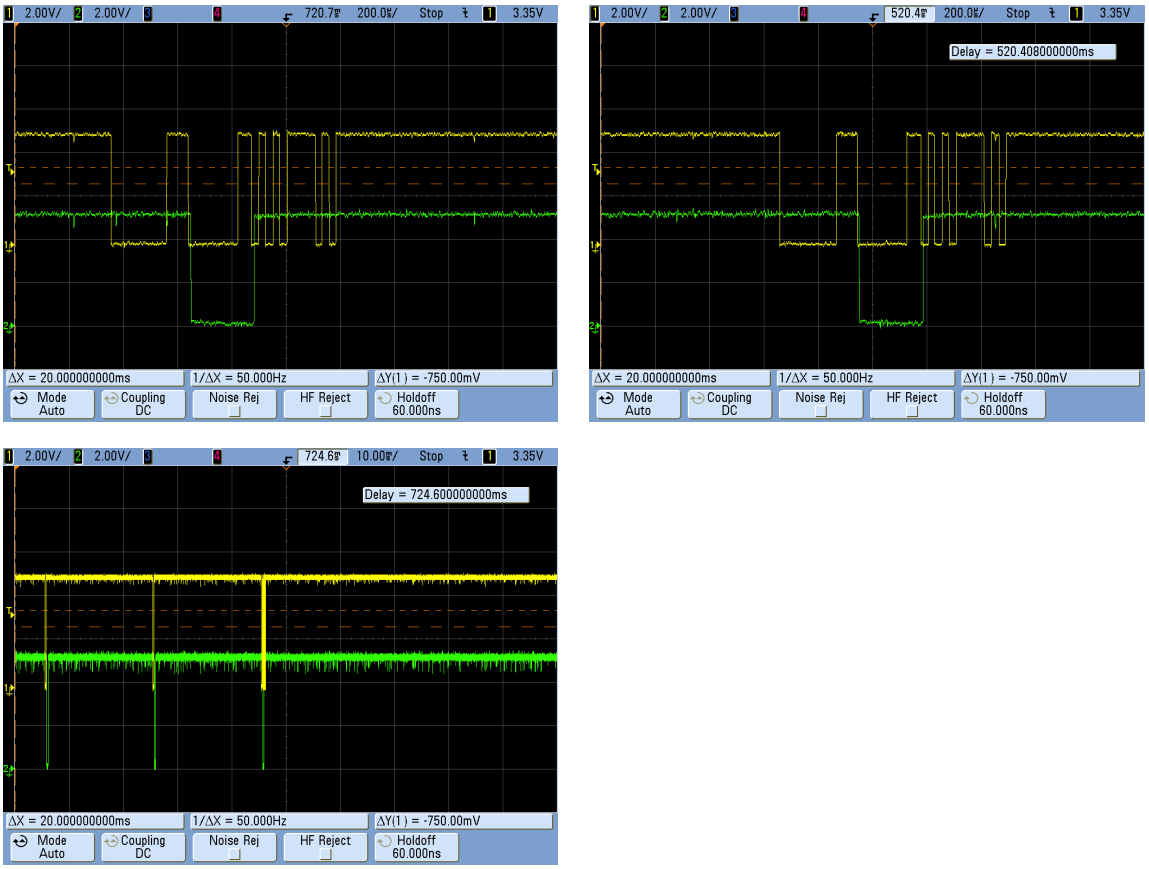


FIGURE 6.6: Oscilloscope traces for the situation where the autopilot breaks: In yellow is the Autopilot TX, in green the Payload TX. In this situation, whilst an ACK token is received by the autopilot, a SEND_BYTES instruction is sent. No more transmit tokens are sent, therefore communication with the payload stops.

Not sure whether this was a bug with the Autopilot, we got in contact with our customer, sending all scope traces to him and a description of how to reproduce the error. Not being able to reproduce it with his dummy payload (strange, as our dummy payloads only differed in that his used a surface-mount version of an ATmega168 and a MAX3070 transceiver instead of MAX489), he then came to ECS and debugged the Autopilot with us.

After a significant amount of time debugging the Autopilot and our payload, it was discovered this was indeed a problem with the Autopilot, which our customer was able to fix and subsequently update the firmware of our Autopilot.

Volatile Variables and ISRs (mh)

Another problem encountered regarded a flag variable being checked in a function called in the main event loop. The flag was the continuation condition of a while loop, and although it was being set in another section of code this new set value was not being

propagated to the while loop code, meaning that the while loop would continue forever. This problem was discovered (helped greatly by the debug interface described in section 6.7) . After some consideration, research and experimentation it was discovered that the problem lay with the C compilers optimization. The flag was being set in a interrupt service routine (ISR), which the compiler assumed could not be called during the while loop call, therefore it was optimized out, causing the loop to continue forever. The solution to this was to set this flag variable as *volatile*, preventing it from being optimized out in this manner. The same was applied to all global variables modified in ISRs.

6.5 Progressive JPEG Manipulation (ms)

6.5.1 Introduction

For information on the reasoning behind implementing progressive JPEG manipulation, please see the JPEG manipulation section in the Approaches considered section. (see section 5.6.2).

The progressive JPEG manipulation algorithm was planned to be done in this way:

1. Extract the JPEG information from the JPEG headers using the AVR microcontroller.
2. Send the necessary image data for a low detail version of the picture (includes both header information and entropy-encoded image data) to the ground station.
3. Decode the image data received at the ground station.
4. Progressively display the image on the ground station.
5. Repeat stages 2 through 4, increasing the detail resolution each time until all the image data has been sent for a perfect representation of the photograph.

The JPEG headers contain all the information needed to recreate the compression of the concerned image. This includes the Huffman table codes and quantization table coefficients, which give precise information on how the image has been encoded. Because these values are dependent on the properties of the image (colour differences of adjacent cells, frequency of colours, etc.), it is not possible to apply an algorithm to display JPEG images without taking into account these properties of the image.

In order to extract the necessary JPEG information from the headers, a JPEG information extractor would be programmed in C and implemented on the AVR microcontroller. Software on the ground station would then receive the incomplete image data

and progressively display an image to the user using the Huffman and quantization tables obtained from the AVR.

The AVR based JPEG information extractor has undergone three major revisions. A functional prototype was first implemented in C# due to familiarity with the coding language. When the C# prototype started reacting properly to jpeg testing images, the code was translated to C to be implemented with the AVR. In order to be able to test the extractor before implementing it onto the AVR, an executable C version of the extractor would be designed to be able to read JPEG images and print the extracted information on a console.

At the end of the project development period, only the extractor was successfully implemented. The C executable version of the extractor has been fully tested against a range of test images. A form of image displaying software has also been constructed which uses the Huffman table information obtained from the extractor, but a progressively higher resolution display of the image was not able to be fully implemented.

6.5.2 AVR Header Search (commandCheck)

The JPEG extractor will be able to obtain the number of packets the camera is sending directly from the camera. After receiving the size of the JPEG file (in bytes), the JPEG Information Extractor would read the bytes of the JPEG stored in the SD card. Because the information will not all be available instantly, the extractor uses a class to read the next byte in the input stream, one at a time. This C class **read_byte()** is how the JPEG extractor gains access to the JPEG input stream.

As the extractor reads in the bytes from the input stream, it checks to see if the byte indicates the start of a segment (0xff). If this is the case, the next byte identifying the segment is read and the **JPEGMethod()** class is called which uses a switch command to determine the actions to take. If not, the data is discarded and the extractor continues to read the next byte from the input stream.

6.5.3 AVR Header Data Extraction (JPEGMethod)

The ground station image displayer will not need all the possible information that can be extracted from a JPEG file. The JPEG extractor will only recognize JPEG headers of segments containing useful information. If the header is not recognized by this class, it is assumed to not have any important information and is ignored; the code continues to scan through the input stream for the next header byte. Below is a list of segments the extractor obtains important information from and an explanation of the intended purpose of the values it extracts. Refer to appendix A for information on the full contents of each of the segments.

6.5.3.1 SOF0: Start Of Frame (0xc0)

The extractor mostly obtains general information from the SOF0 segment. It saves the height and width of the image for the ground station software to prepare the progressively scanned image's dimensions and keeps the data precision as a basic means of error checking. If the data precision is not 8, indicating a byte, then the image will not be able to be read by the ground station.

The number of components helps indicate the colour space which will be used. Ideally, the image will use the YCbCr colour space. Currently, there is no functionality for Y, I, and Q components which is not used by UK-based cameras. Grey-scale images have yet to be tested with the extractor.

The sampling factor information from this section will indicate the chroma subsampling pattern used (either 4:2:2 or 4:2:0) and influence the shape of the chrominance pixel structures created during the treatment of the SOS segment. (see section 2.1.4 for information on chroma subsampling) The sampling factor information will also define the dimensions of the MCU (8x8 or 16x16).

Below is a screen shot taken from the executable version of the JPEG information extractor from a standard JPEG image. All screenshots were taken from the console of the Eclipse development environment.

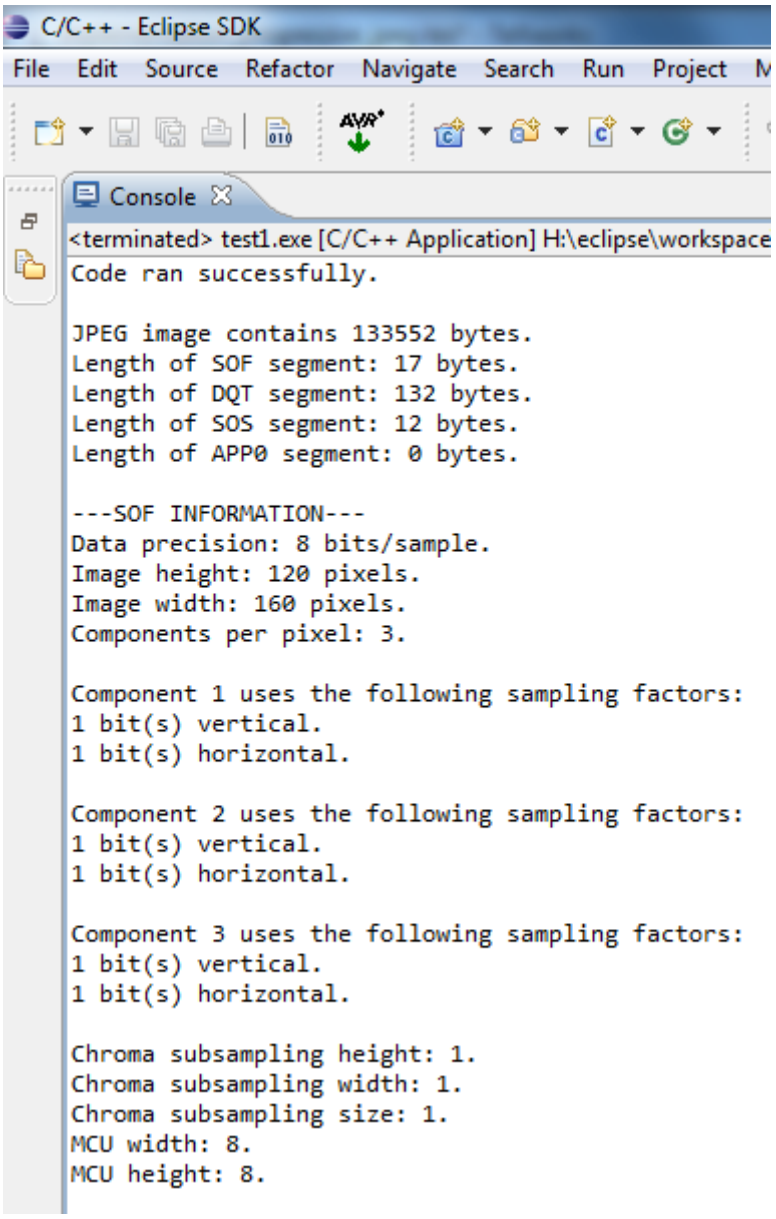


FIGURE 6.7: SOF information obtained from the executable

6.5.3.2 DHT: Define Huffman Table(s) (0xc4)

Obtaining the Huffman table information is vital for the progressive scanning of the image to function properly. Due to the fact that JPEG images are Huffman-encoded, it is necessary to obtain the Huffman table information used to encode them in order to decode the image data properly into a visible format.

Each DHT segment may contain multiple Huffman tables. Usually, one DHT segment would contain a pair of Huffman tables, one for DC and one for AC, to be used for image decoding. Additionally, testing has revealed that multiple DHT segments may exist in a single JPEG file.

The extractor creates a unique Huffman table structure to store all the information from the DHT segment. This information is stored within a linked list of Huffman table structures.

In order to choose the correct Huffman table among the many tables which are used, the JPEG file issues an identification number to each Huffman table which is shared only by its DC/AC counterpart. The Huffman table structure keeps this information as well as the DC or AC nature of each Huffman table so that the ground station software can easily find the appropriate Huffman table to use to decode a pixel component.

After storing the classification details of the Huffman table, the extractor analyses the number of symbols for each of the code lengths 1...16. It then creates arrays of appropriate sizes to store all symbols contained in the Huffman table. The organization of this data allows a full Huffman table to be reconstructed using one Huffman table struct, as shown below:

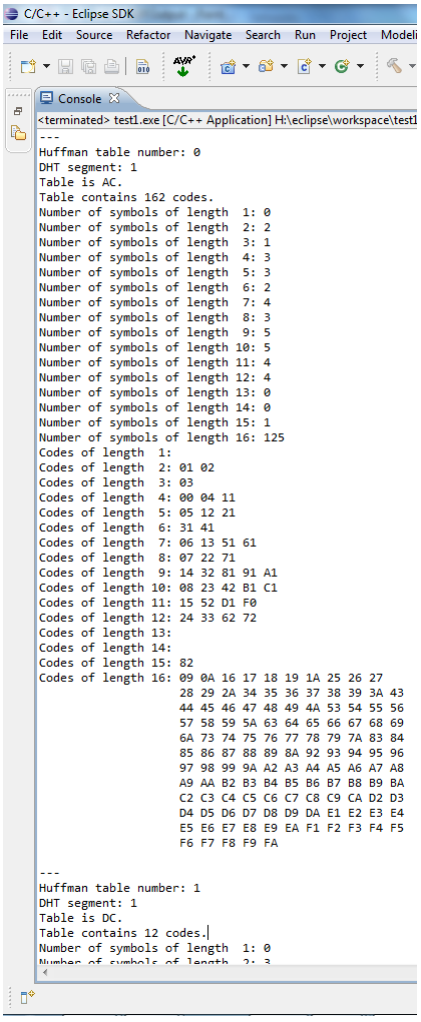


FIGURE 6.8: DHT information obtained from the executable

6.5.3.3 SOS: Start Of Scan (0xda)

The SOS segment is the final header containing JPEG compression information before the entropy-encoded image data is read in. It specifies the AC and DC Huffman tables which must be associated with each component in the scan. The JPEG information extractor uses this information to be able to assign each entropy-encoded image byte with the Huffman tables used to encode it. This way, the ground station image decompression algorithm can easily associate each image byte with an operation to reverse the Huffman table modification in order to display the original image.

6.5.4 Image Data Treatment

As explained in section (2.2.3.2), the entropy-encoded image data would be stored in chrominance pixel structures. This structure stores information for all the luma values within the chrominance pixel, as well as both chrominance values associated with it. Specifically, it contains:

- An ID number to keep track of the image bytes.
- The AC Huffman table ID number of each of the components.
- The DC Huffman table ID number of each of the components.
- The value of the component.

If the ground station software needs to determine what component the byte value corresponds to, it will need to identify either the order of appearance of the components (ID number) or the Huffman table information of the components. If we consider n to be the number of image pixels in a chrominance pixel, then we associate the first $n-2$ bytes with the Y (luma) Huffman tables, the $n-1$ byte with the Cb Huffman tables, and the n^{th} byte with the Cr Huffman Table. This information continues to be read by the extractor until the EOI header is found, or until there are no more bytes coming in from the input stream.

6.6 Integration of Payload Sub-Modules (mh)

6.6.1 Integration of Camera Module with SD Card

Both the camera module and SD card module were initially prototyped and partially tested separately, but were integrated at a fairly early stage. This integration was the easiest way of testing the camera module, since the images from the camera could be written to the SD card, allowing for easy inspection.

6.6.2 Integration of Camera Module and SD Card with Payload-Ground Station Communication

Integrating the camera module with the payload to ground station communications was a major milestone for the project, since completing it would mean we would have a basic working system. It was also one of the most challenging parts of the project. This step was performed iteratively over time, with the most basic features being implemented, integrated and tested before more advanced features were attempted

Integration of the Payload-Ground Station communications with the Camera module was completed on the ATMega644P hardware, please see section 6.6.5 for more info about this aspect and an idea of the challenges and problems faced.

6.6.3 Integration of Progressive JPEG Manipulation

Unfortunately due to time constraints we were not able to integrate the progressive JPEG manipulation code into the payload module, please see section 6.5 for more information.

6.6.4 Implementation on Arduino

Initially we planned to implement the system on the Arduino prototyping platform with an expansion 'shield' board providing any extra hardware needed - the reasons for this are discussed in section 5.4.3. The camera module connection code had also already been developed and tested on the Arduino.

This plan relied on the implementation of a multiplexed serial solution, where the single serial port of the Arduino would be used to connect to two different devices (the camera and the autopilot) at different baud rates and message formats. It was planned that this would be achieved by the use of tristate buffers to form a bus with the two devices TX and RX lines. The payload software could then take care of altering the message format and speed when needed.

Unfortunately, due to time constraints alongside setbacks such as those discussed in sections 6.4.6 and 6.2.1 it was decided that implementing the system on the Arduino platform may not the best way to proceed. The main reason for this decision was the number of unknowns in the planned multiplexed implementation - since it was not really how the hardware is designed to be used, it was not certain it would provide satisfactory results in the limited time that we had.

6.6.5 Implementation on ATMega644P

Several options were considered alongside the multiplexing option discussed above. One alternative workaround discussed was to use a separate AVR as an interface to the autopilot and connect it to the Arduino using bit-banged SPI or the inbuilt two-wire interface (TWI, sometimes known as I2C). This approach was attractive as it meant we could use the existing Arduino code for the camera and SD card and use the existing ATMega168 code for the autopilot communication. However, this solution would add another complex component to the system along with another communications link, increasing complexity significantly.

Another option considered was using an AVR device with multiple UARTs so one could be used for the camera and another for the autopilot link. However, the code we had written for interacting with the SD card buffer relied upon some Arduino specific libraries. Finding new libraries and rewriting this code would be time consuming and inefficient, especially since we had already invested time in creating a working SD card solution.

Further investigation into the Arduino SD card library code (for more info regarding the library see reference [15]) suggested that it was coded to work with the ATMega644P chip as well as the ATMega168 and ATMega328P. Both the ATMega168 and ATMega328P feature only one UART, but the ATMega644P features two, making it attractive for use in this project (see [17] for more info about the ATMega644P).

Since the ATMega644P provides two UARTs and the SD card library we were using supported this chip it was decided that using the 644P would be a sensible way forward.

It is important to note that frequent progress reviews meant that the problem of how to implement this integration and the need for 2 serial lines was caught early and controlled before it could become more damaging.

As mentioned above in section 6.4.1 the code provided to us for communication with the autopilot was written for an ATMega168 chip. Porting this code and the existing Arduino camera code to the 644P was not a trivial task, although it was simplified by the use of the Sanguino library - a port of the Arduino core libraries to the 644P (see [18]). Some changes were made the the Sanguino libraries for use in our project, for example the Sanguino handling code for one of the UARTs was disabled to allow it to be configured manually for autopilot communications. Some small pieces of code that were causing compilation issues were also modified.

6.7 Debug Interface (mh)

While the payload module was being developed debugging was a major concern. Having access to an oscilloscope very useful when debugging some particularly difficult problems. However, while building the payload module it was clear more complex details about the internal state of the program as it was running would be useful.

While prototyping the camera module on the Arduino platform this problem was solved by using the *SoftwareSerial* library (for information regarding the library see [14]) to send textual debug information via a serial to USB cable to PC. This proved very useful when prototyping camera communications.

A slightly different approach was taken when using the ATMega644P. Initial testing of the software serial library on this platform suggested that it would not work without a significant investment in time fixing it. It was therefore decided that it would be faster to send debug messages over 'bit-banged' SPI to a spare Arduino board which then forwarded this data to the connected PC over serial. Again, having the use of this debug interface proved very useful while implementing the system, and was well worth the small amount of extra time taken to develop it.

Figure 6.9 shows an example of this debug interface seen on a serial terminal on a PC.

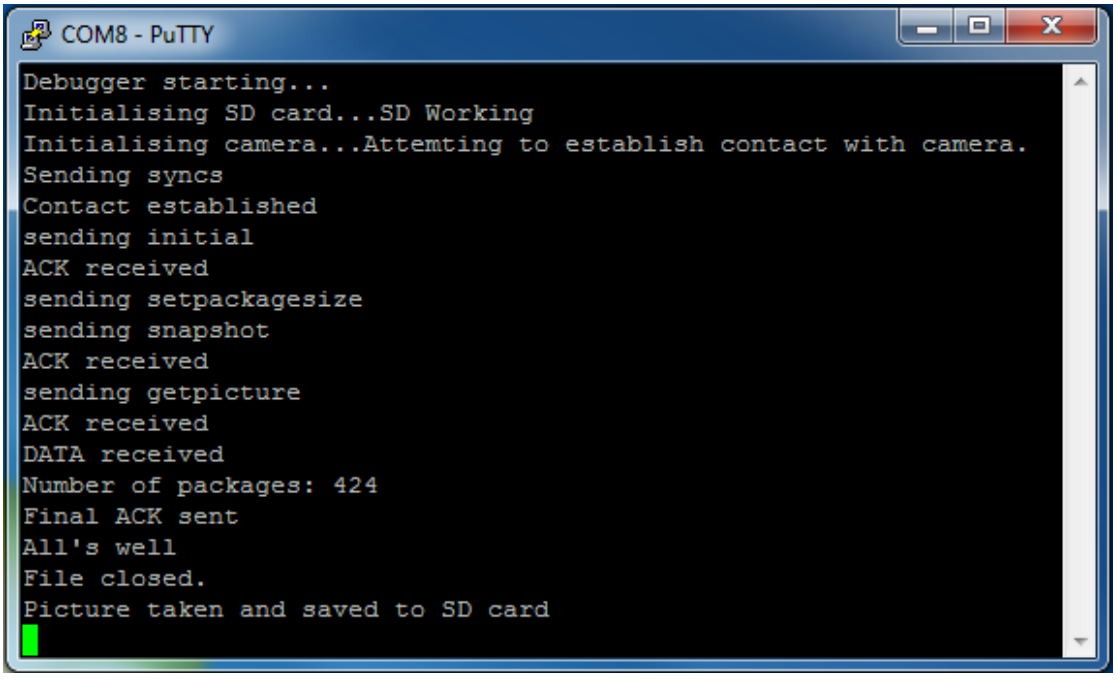


FIGURE 6.9: Debug text sent from the payload over the debug interface.

6.8 Physical Implementation (ab)

The final, delivered module is a $88mm \times 62mm$ PCB. The PCB manufactured for the project was done so for free, using Spirit Circuits' "Go Naked" service [22]. The PCB itself is a "tracks and holes" only service - no soldermask or silkscreen is applied. The schematic of the circuit delivered is available in Appendix E.2, and of the PCB layout in Appendix F. A waterproof lacquer will be applied to the PCB to prevent condensation from shorting tracks together, and the module itself presented in a waterproof container before flight testing.

A quote of £56.39 has been obtained from PCB Train [1] for a single PCB, with electrical testing, on a 15 day lead time. Please note that Spirit Circuits have only manufactured our prototype free of charge on a one-off basis. Being impressed with this service, we may recommend that clients contact them for a competitive quote.

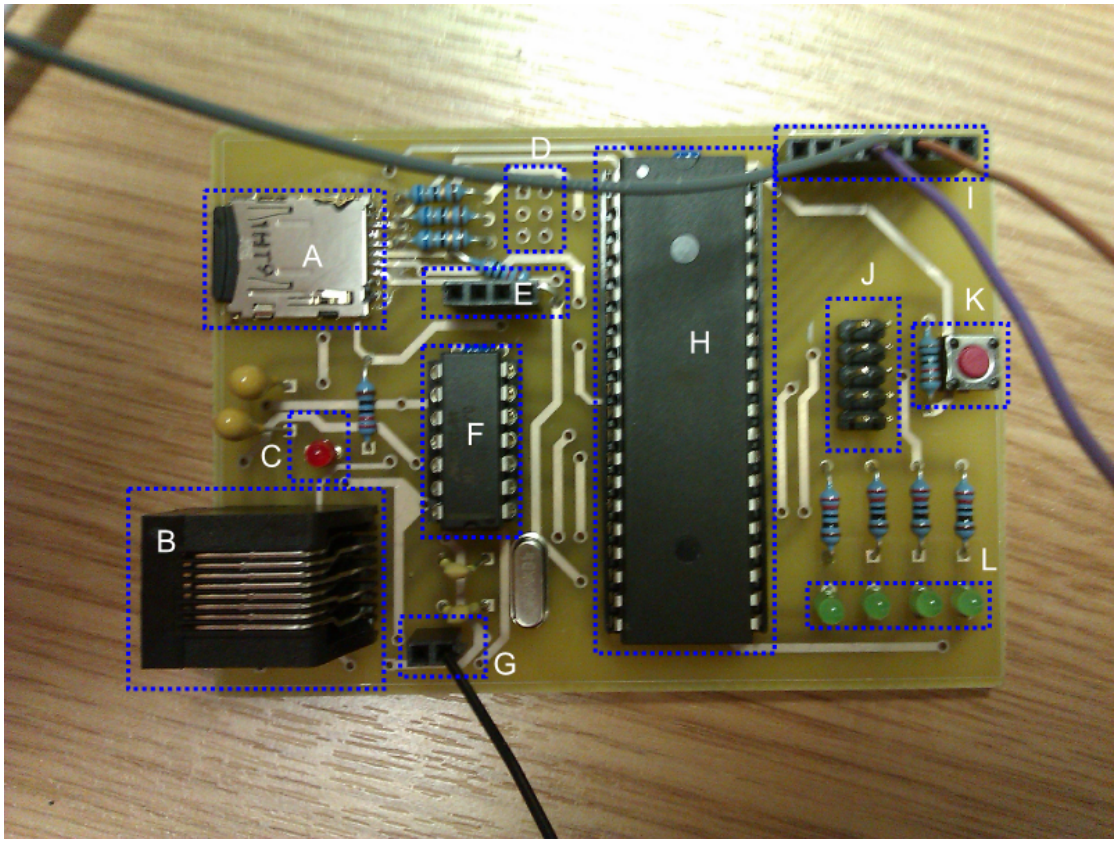


FIGURE 6.10: Image of the final payload, under test, before lacquer is applied. R11 can be seen between the Camera header and a via near the SD card Vcc.

.

In figure 6.10, the letters refer to the following features of the payload module:

[1] http://www.pcbtrain.co.uk/quote-and-order-pcb/

- **A:** microSD Card slot
- **B:** RJ45 socket, more popularly known as an ethernet socket. (Actually implements the RS485 protocol)
- **C:** Power LED. Shows whether 3V3 from the ethernet socket is connected.
- **D:** ISP programming header socket. As seen, this is not soldered on, as we have been using the JTAG header for testing, but will be soldered before delivery to the customer.
- **E:** Camera header. The camera can be connected to our PCB simply by attaching the hook-up wire in the correct order. (L-R: 3V3, GND, Camera TX, Camera RX)
- **F:** MAX489 (RS485 Transceiver) [2]. Cheaper version of the MAX3070 used in the sample peripheral.
- **G:** Power header. L: 3V3, R: GND.
- **H:** ATmega644P [17]
- **I:** ATmega644P Port A expansion header. (In the image, this is being used as a connection to an Arduino Uno so that we may view the debug information)
- **J:** JTAG programming header
- **K:** Reset button
- **L:** Debug LEDs

[2] http://uk.farnell.com/9725148

Chapter 7

Implementation - Ground Station (ps)

On the ground station the user only has access to the image viewer program. The programmer makes use of both the data stream port and the console port to send commands to and receive JPEG photos from the UAV.

7.1 The development process

Because the GUI includes many functions on the application and links to the TCP/IP ports, the GUI part consider as a very big program. Therefore, in order to complete the GUI step by step, a plan for the development is required. The connection to the datastream can be done by using a console application described in section8.3.1. When the connection is established, the appliction can listen to the data stream port as in section8.3.2. These console applications will be incorporated into a larger program in later section 7.13.

These are the development stages that the GUI has went through:

1. Understanding what the customer wants.
2. Determining the hardware and software specifications.
3. Designing the GUI.
4. Developing a smaller program which simulates the UAV camera's signals in order to make the testing easier and less time consuming.
5. Learning the .NET class that can support the connection to TCP/IP, and communicate between the host and device.

6. Using GUI to link the ground station software to access the UAV.

7. Distributing the GUI.

7.2 Initial Use Case Diagram

The use case diagram shows what functions the user can use in the program. It includes all the specification that the customer wants for a complete system. Figure 7.1 shows all possible actions the user can perform on the program, such as saving, opening and deleting any JPEG image from the computer. The user can also connect to the UAV if it is not already connected. He can use the image viewer program to prompt the UAV payload to return an image which will be displayed on the picture box.

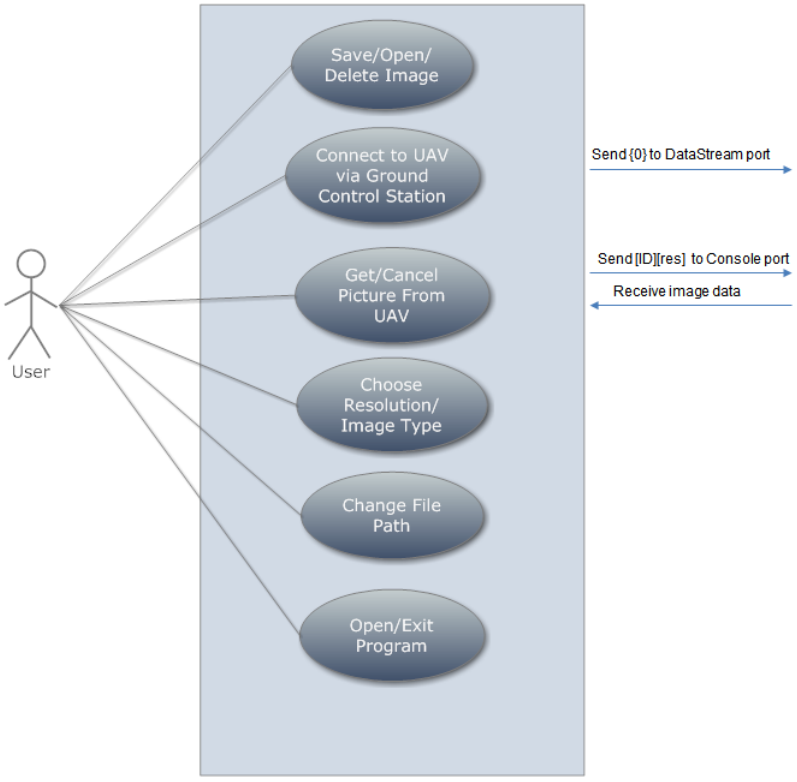


FIGURE 7.1: Use case diagram of the GUI

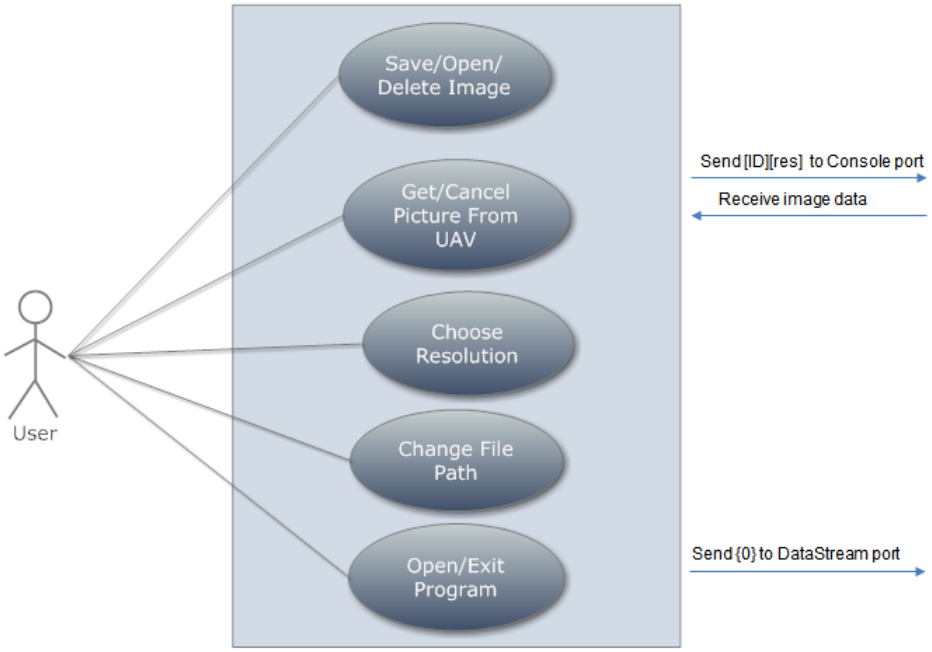


FIGURE 7.2: Final use case diagram of the GUI

7.3 The Design

The user of our application is assumed to have limited programming experience, so the program will need to be simple to understand. Figure 7.3 show the first prototype view of the GUI. During the downloading process, the application should stay active so that the cancel button can be used. The `Gallery` button will link to another page which will show a gallery of images taken. The `Left` and `Right` buttons can navigate the picture box to view an earlier picture or a later picture. The `Cancel` button will cancel the download of the receiving image, so that corrupted pictures can be cancel. The user mode of the application can access only the main features, such as taking picture, changing directory, and cancelling the download of a picture. It allows the user to choose the resolution and picture type (raw or JPEG) to be transmitted from the UAV to the ground station. However, the user doesn't have access to changing the command sent, changing the receiving data, and any interaction with the UAV because to avoid of any errors.

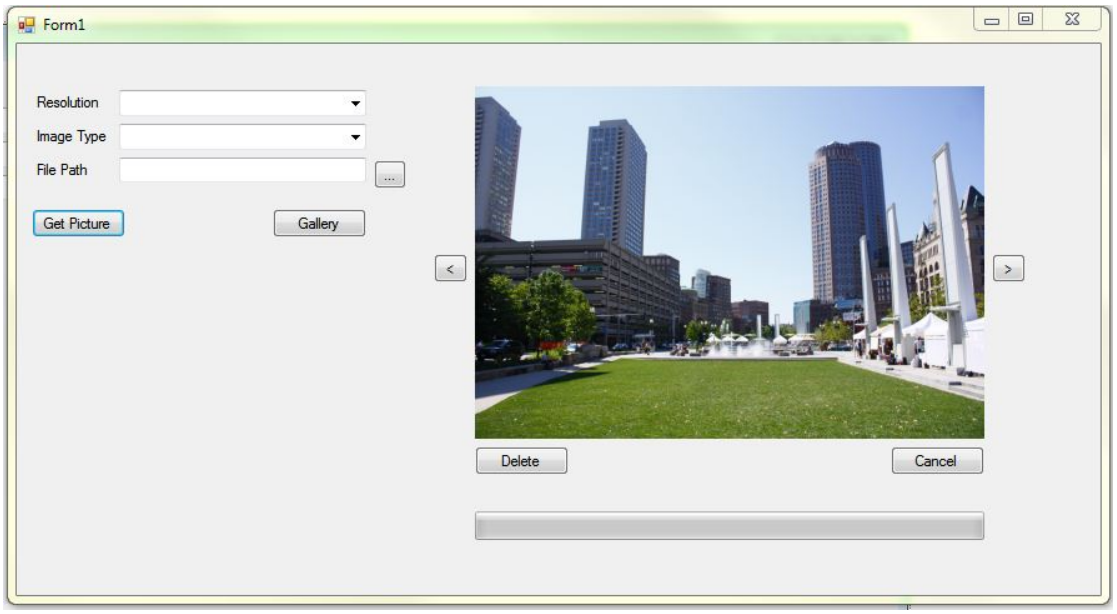


FIGURE 7.3: The initial design of GUI

The GUI has been planned to have functions such as auto triggering, selecting the image type, the resolution type, and file path chosen, a progress bar, a help button, and stop and delete buttons. Figure 7.4 is the screen shot of the final GUI.

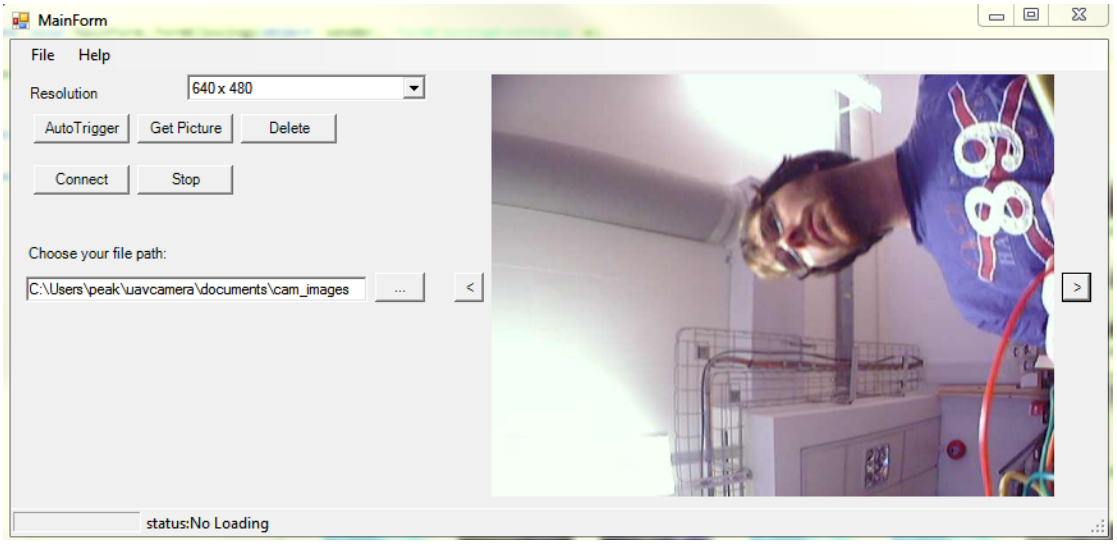


FIGURE 7.4: final GUI

7.4 Final Use Case Diagram

Figure 7.2 shows the final use case diagram. The connect button was not implemented because the ground station will connect automatically to the UAV when the program

is opened. The raw image is too large and takes to long to send down the connection so it was not implemented. Therefore, the Milestone5.9.2.7 will not be implemented.

7.5 Class Diagram

The Initial Class Diagram

Figure 7.5 shows initial classes and methods of the image viewer program. The `JPEGFileReader` Class has functions for decoding and encoding the JPEG file. There will be a decoding/encoding algorithm because the image will take a long time to download to the ground station.

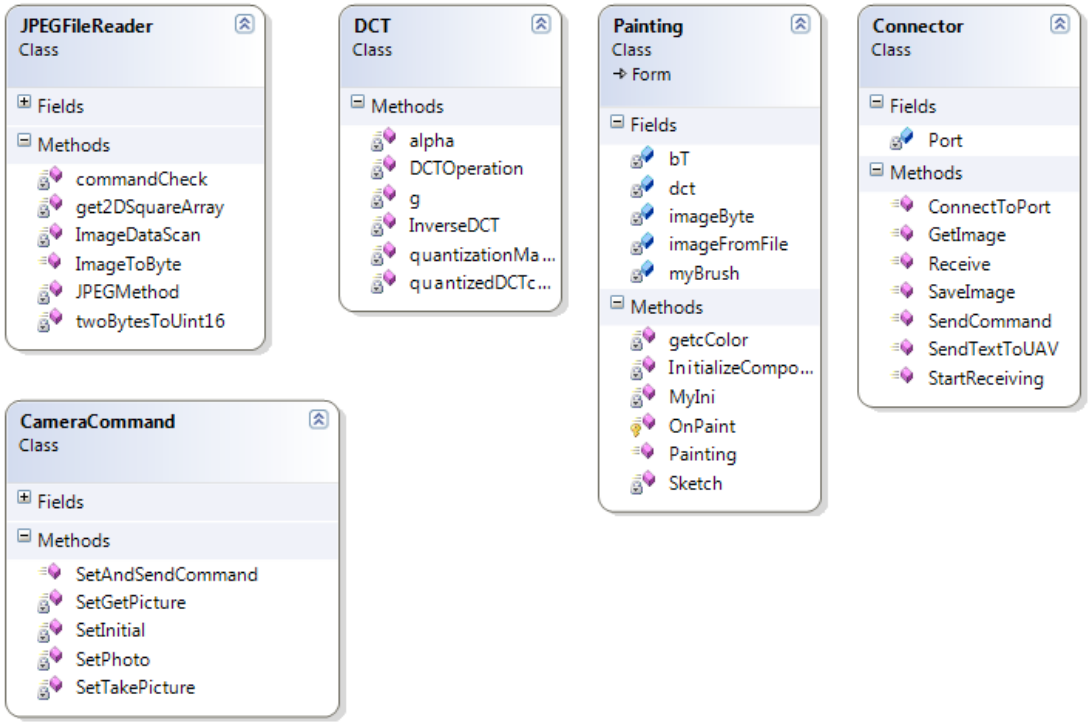


FIGURE 7.5: The initial design of GUI classes

The `DCT` class has many arithmetic operations and equations which have to be implemented on the image viewer program.

The `Painting` class is supported by the `DCT` class. The intention of this class is to display an encoded image point by point on the `pictureBox`. Using this method, the `pictureBox` can display an image from the first pixel transmitted.

The `CameraCommand` class is designed to send the data from the ground station to the camera. The idea is to make the camera sync with the payload by using ground station commands. The `SetAndSendCommand` class is used to set the byte command and then send it to the payload via the console port. The `SetGetPicture()`,`SetInitial()`, `SetPhoto()`, and `SetTakePicture()` methods are used for setting the correct byte to send through the `SetAndSendCommand` class.

The Class Diagram

The class diagram has been implemented very differently from the planned one. This is because the new plan is to decode the image on board the UAV and then transmit that image to the ground station in a compressed JPEG file. The `DCT` class is not needed anymore because all the DCT calculation will be done onboard the UAV. The `CameraCommand` has been taken away because the payload will receive an image viewer command from the ground station and then the payload will send another different signal to the camera. Therefore, the command sent to the camera from the payload does not have to be the same as the command sent from the ground station to the payload. The `Painting` class is used to draw each pixel onto the `pictureBox`, but it has not been implemented in the final program because the JPEG encoding will be done onboard the UAV. Therefore, the milestone5.9.2.8 will not be implemented. This is also due to the time limitations. More detail about the progressive image can be found in the section6.5.

7.6 Before Connection to the Image Viewer Program

Figure 7.7 shows a diagram of how the connection of the hardware should be. The UAV ground receiver is a USB-compatible device that uses Zigbee to communicate. The USB device driver has been developed by the customer so the hardware can be accessed by the ground station software, and other applications. The USB is active when the host asks for a data. A host is the computer network which the UAV connects to. The data is in queue until the host asks for the data.

7.7 GUI data flow diagram

Table 7.1 shows how the command is sent and received to and from the ground station. Figure 7.8 gives a brief detail of how the data communicates between the payload and the image viewer program. A more detailed diagram on the data communication can be found in the figure 6.4. This has been tested in section8.3.3.

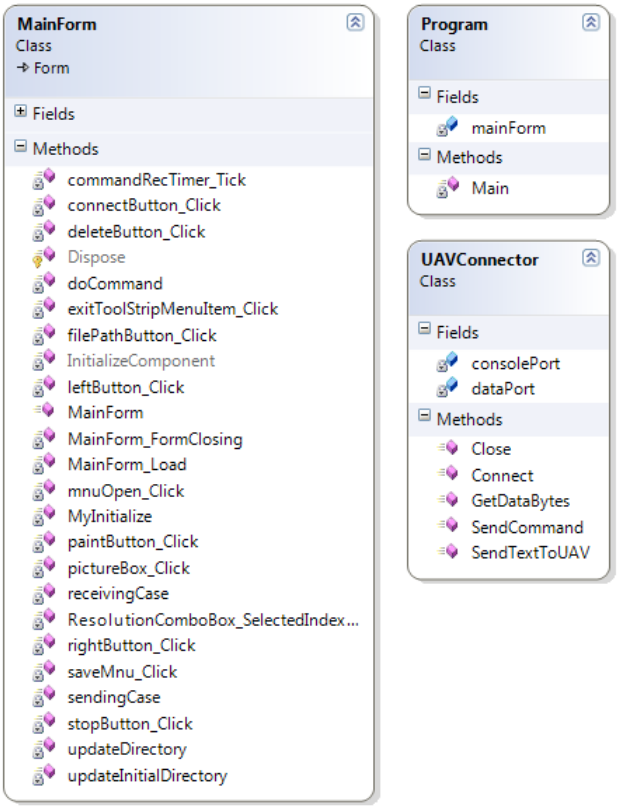


FIGURE 7.6: Final class diagram of the GUI

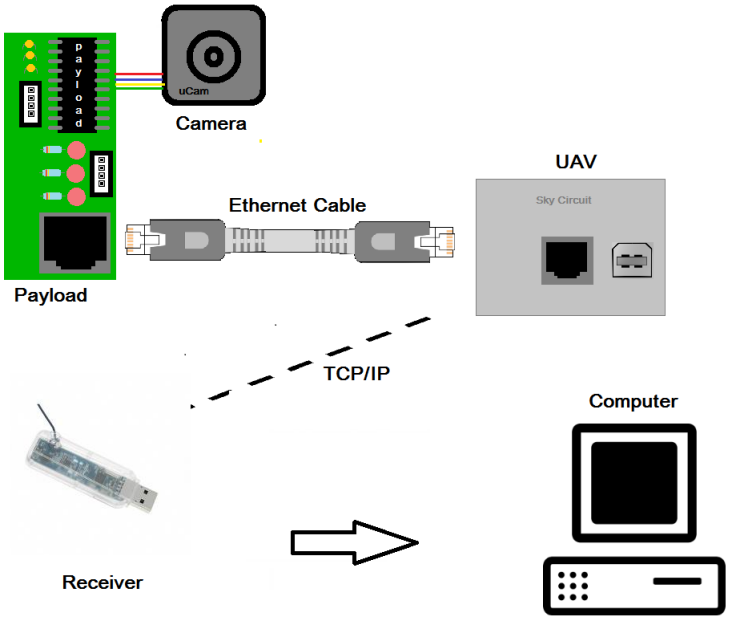


FIGURE 7.7: The connection of the hardware

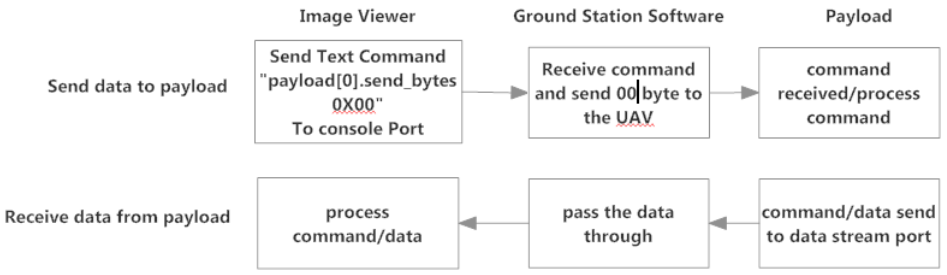


FIGURE 7.8: The connection of data stream port

Command	Address Byte
Command from Ground	
SEND_ZERO_TOKEN	0
TAKE_PICTURE	0
SEND_DOWNLOAD_REQUEST	2 [MSB] [LSB]
Command received at Ground	
PICTURE_TAKEN	1 [MSB] [LSB]
DOWNLOAD_INFO	3 [MSB] [LSB]
IMAGE_DATA	4 $\overbrace{[packetnumber]}^{2bytes}\overbrace{[imagedata]}^{datalength}$.

TABLE 7.1: Command table

7.8 Get Image Algorithm

When the user clicks on the `Get Picture` button, the program should send a **string command** to the ground station software. Then, the ground station software generates a **byte command** to be transmitted by TCP to the payload. Afterwards, the payload sends a "Picture Taken" command back through the data stream port to the Image Viewer Program.

When the program starts running, it initializes the port and commands the customer's application to tell the UAV to stream data to the data stream port. Figure 7.9 shows the connection between the UAV data stream port and the ground station.

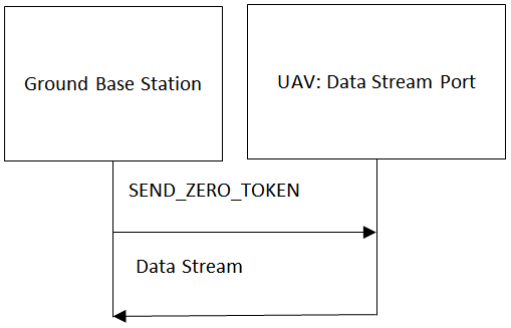


FIGURE 7.9: The connection of data stream port

7.9 Important Code

This section describes the code that is important for the program. The entire code will not be described but it will be in the appendices. The program must be able to perform these actions to the port: connect to, receive data from and send data to.

The following .NET C# classes will be used in the ground station program:

- `FileStream` creates a file.
- `BinaryWriter` writes the byte data into a specific file made by the `FileStream` class.

UAV Connection

This section references appendix D.2 lines 18-38.

The `Socket` class has functions to send and receive byte and string data. The connection protocol is using the `PortConnect()` method.

The design of the .NET `Socket` class simply connects to the port by a `PortConnect()` command regardless of changes to the baud rate, stop bits, and parity bits. This advantage makes the `Socket` class a more compatible class than the `SerialPort` class to work with the UAV. This class has tested using a console application before implementing it in a final GUI. This will complete milestone5.9.2.1.

7.9.1 Start of the program

This section references appendix D.1 line 78-87

The `FileStream` class is initialized by using the method `FileMode.Create()` in order to generate files. The `BinaryWriter` writes binary bytes into a file in the directory of

the `FileStream`. The `BinaryWriter` class creates a binary file using specific data layout for its bytes.

7.9.2 Text Command

This section references appendix D.2 line 68-85.

The TCP/IP protocol transfers data without modifying them. It allows the application to freely encode the data [28]. The ground station software allows the Image Viewer program to send a stream composed of a string in bytes. It will then read the command bytes and send it to the payload on the UAV. The codes have shown a correct way to implement the string and send a byte array to the payload.

`The Socket.Send()` method sends bytes to the ground station software. The "@ " sign indicates that the command is correct. For example, consider the bit of code: uavConn.SendTextToUAV("da 20 payload[0].mem_ bytes[0]"); The text `''da 20 payload[0].mem_ bytes[0]''` will be converted into a char array and then into a byte array. The byte array will then send the command to the console port and then to the payload through the method `consolePort.Send(toUAVByte, toUAVChar.Length)`. This will complete Milestone5.9.2.3.

In order to test that the payload receives the same data, we use the oscilloscope to see the signal. The byte displayed on the payload is the same as the byte sent from the ground station. Therefore, the Milestone5.9.2.5 is completed. The different resolutions send different byte commands to the payload. The byte commands change if the comboBox options change. Therefore, we can say that Milestone5.9.2.6 has completed.

```
for (int i = 3; i < packetSize; i++)
{
  opFile.Write(packet[i]);
  numBytes++;
}
```

Listing 7.1: writing binary file

At every cycle of the data being received, the `opFile.Write()` method will write the packet data into the file in the directory. After the cycle finished, the file will be saved and the image will be displayed in the picture box.

7.9.3 Get Picture Button

This `Get Picture` button will use both the send and receive functions of the program. The work flow of the `Get Picture` signal is shown in figure 7.10. If the sequence of signal is sent and received correctly, the photo received from the UAV will be displayed on the photo box in the program. This will complete Milestone5.9.2.4.

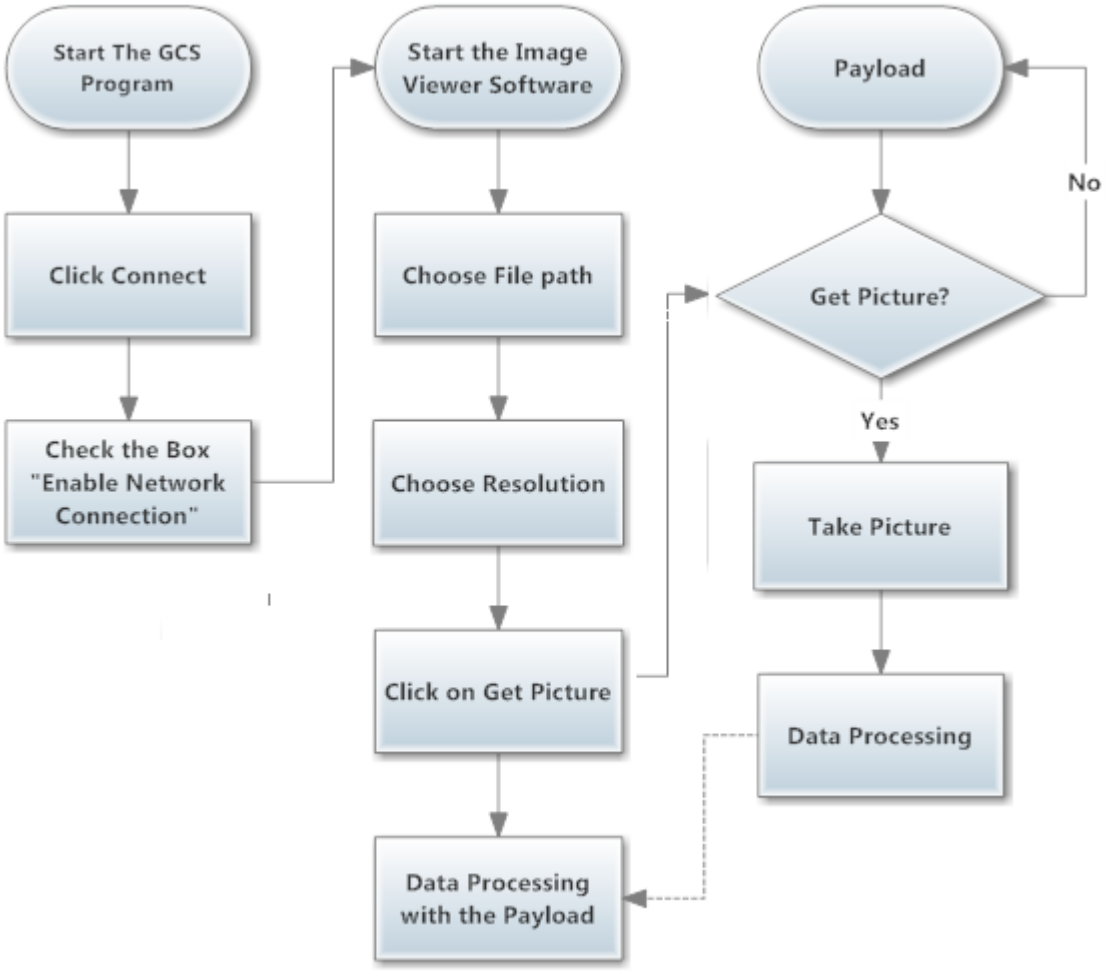


FIGURE 7.10: Final work flow diagram of the GUI

7.9.4 Implementation - Way-point Triggering (ms)

7.9.4.1 Description

The ground station is capable of assigning way-points to the payload which will allow the camera to take pictures at a given location.

This is achieved by sending a simple command script from the ground station to be uploaded by the payload controller. The way-point is designated by the user through the ground station software. The script tells the UAV controller to continuously check the distance separating itself from the way-point. When the UAV reaches is within 200 metres of the way-point, a 0 byte is sent to the camera to prompt the camera to take a picture.

After taking a picture at the way-point, the camera is delayed for 10 seconds to avoid

taking another picture at the same way-point. When the 10 second delay is over, the camera repeats the operation and continuously checks its distance from the next waypoint.

7.9.4.2 Pseudo-code description

Below is a brief pseudo-code description of the script sent by the ground station to the payload to take an image at designated way-points:

- while !(UAV distance from next way-point *le* 200 metres)
  - Do nothing.
- end while
- Prompt camera to take an image.
- wait 10 seconds
- Re-enter while loop

7.9.5 Other functions

The `Delete` button works like a normal file deleting button. However, when there is only one picture in the file, the program is not allowed to delete that file. This is because even if the pictureBox is set to null, the last image displayed is still pointing to the deleting file. However, the disadvantage of the `Delete` button is that if the wanted photo was deleted accidentally, it may take a long time to launch the UAV again and take the same photo.

FIGURE 7.11: The resolution in combo box

The camera has resolution options as shown in Figure7.11. This can be useful when the downloading speed must be fast. The lower the resolution, the faster the data is transmitted to the ground. The GUI's combo box allows the user to choose any wanted resolution in the options. The resolution option gives the user more control over the camera.

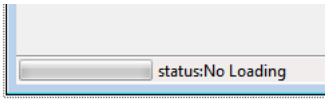


FIGURE 7.12: The resolution in combo box

7.10 Complete System

After implementation has been completed, the different functions will be tested together. When all the parts come together, the ground station program fulfills the requirements of the specification. Figure 7.13 shows a final working GUI of our program.

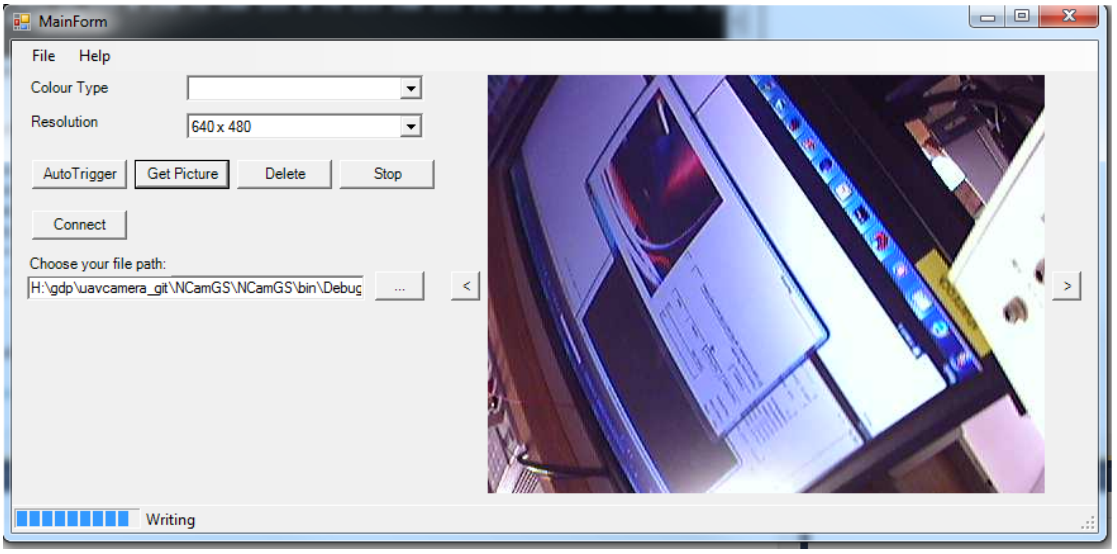


FIGURE 7.13: The complete system

Chapter 8

Testing

Any successful engineering project requires thorough testing. Testing throughout the implementation of the system is essential, allowing the various stages of development to be verified before the next feature or stage is implemented.

Similarly testing is an important documentation tool, which allows us to determine the exact specifications of the implemented system and helps us understand how it is expected to perform in real-life scenarios.

During the implementation of this system, thorough testing was an absolute requirement. This chapter describes the various tests undertaken throughout implementation in the terms of the milestones described in section 5.9.

Throughout this chapter, we will refer to milestones by their section numbers. For example, the *Basic Camera Connection* milestone will be referred to as milestone 5.9.1.1.

8.1 Camera Module Testing - (mh)

The camera module and controller were tested throughout their development and thus problems were quickly discovered and swiftly rectified.

8.1.1 Test: uCam Existing Software (jc)

The uCam has a freely downloadable piece of software (downloadable from [34]) that allows you to test the uCam and verify its functions. If you can connect the uCam to a COM port on the computer then you can sync with it and then take and view pictures in the different available formats and resolutions.

This allowed for not just the functionality of the camera to be verified but said functionality was then compared to the specification and it was shown that the uCam should be able to provide what is required from the camera module.

The uCam was connected to a host PC using a USB to Serial cable and tested using the software. The software connected to the camera successfully and downloaded an image as expected.

This task did not verify any milestones, but it did show clearly that the camera was working. In the event of the camera becoming unresponsive, this software helped us diagnose the problem, since if the camera worked with this software it was clearly a problem with our implementation, while if the camera did not work with this software then it would suggest the camera itself was at fault. Before declaring any of the cameras used in the project faulty they were checked using this software.

8.1.2 Test: Basic Connection Test on Arduino (mh)

The first step when communicating with the camera module is to establish a connection to it. The implementation code follows the procedure shown in figure 6.2 to connect to the camera, sending SYNC commands until the appropriate response is detected from the camera, indicating a successful connection and so the success of this test.

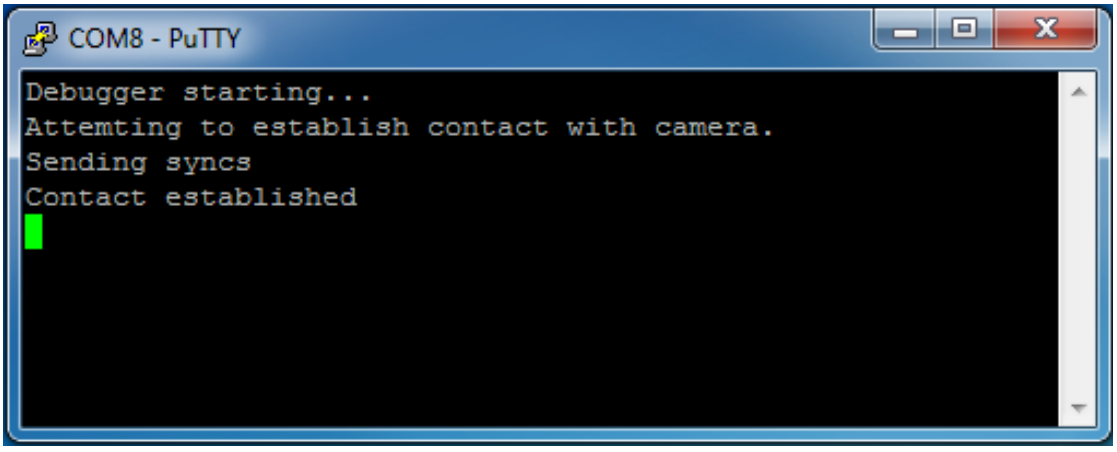


FIGURE 8.1: Testing our implementation of the basic connection to the camera module.

Figure 8.1 shows the result of a test for basic connection to the camera module. In figure 8.1 the line "Contact established" means the Arduino has detected these correct responses. With this successful connection, milestone 5.9.1.1 is validated, meaning the implementation was free to move on.

8.1.3 Test: Image from Camera - (mh)

The first attempt at a simple test for this was to use the debug connection to the Arduino camera controller implementing the camera connection code to relay image data to a PC, unfortunately as described in section 6.2.4 initial implementations of this were not successful, since the debug connection was not completely error free, causing the image to be malformed and unreadable.

However, at this point in the implementation the basic SD card communication had already been implemented as described in section 6.3. Because of this, it was decided that a much more sensible test would be to write the image data directly to the SD card. The test comprises of running the camera connection code on the microcontroller with the camera and SD card attached. Figure 8.2 shows the debug information sent from the controller during this test and 8.3 shows the image saved to the SD card. The debug information suggests that the connection and download of image data was successful and the image on the SD card verifies this.

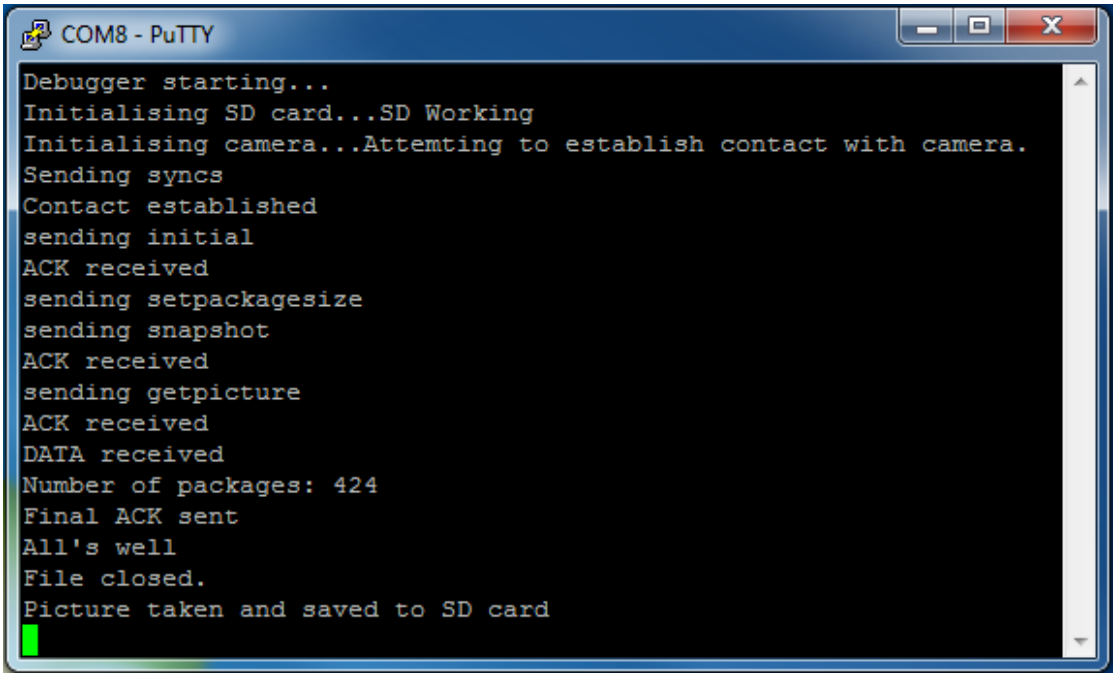


FIGURE 8.2: Debug information sent from camera controller to PC while taking a picture and saving it to an SD card.

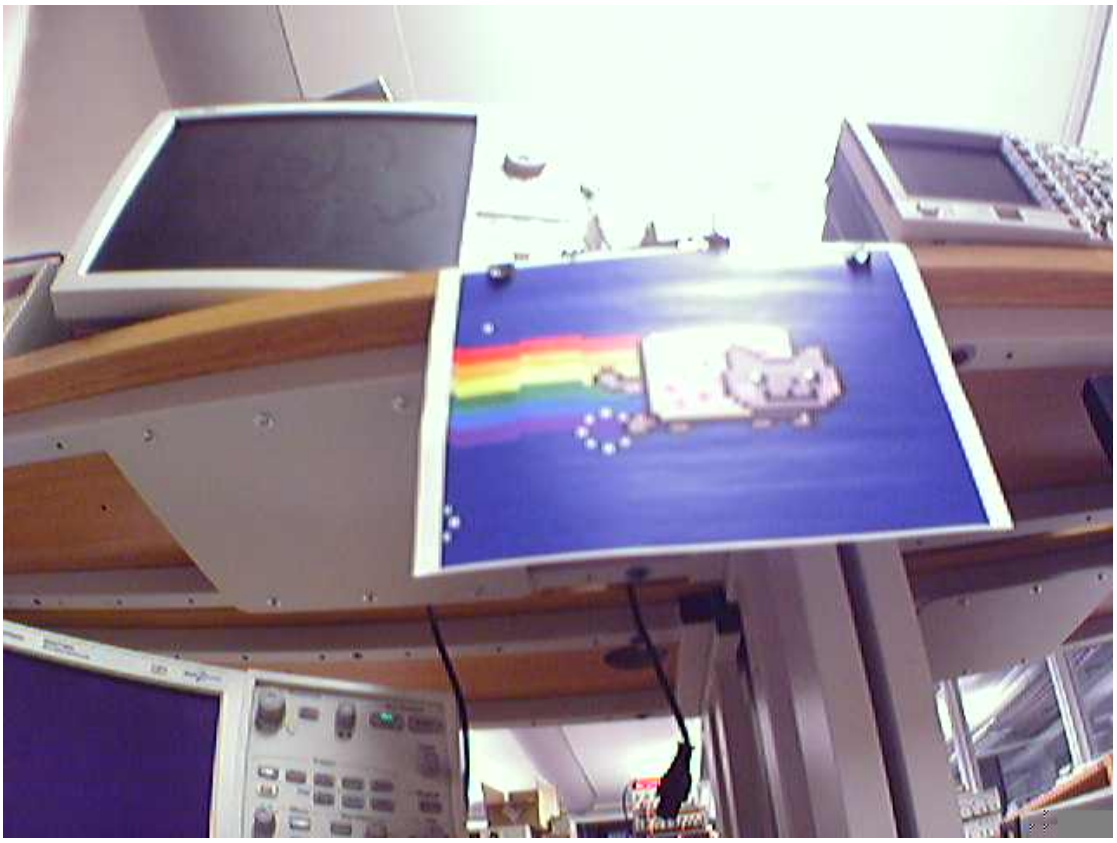


FIGURE 8.3: Test image captured using the Arduino implementation of the camera controller

The testing during development means that functionality of the controller has been verified already, the test that needs to be performed on this element of the system is to time how long it takes to get an image from the camera and save it to the SD-card and check that this is well within the 3 minutes allowed by the specification. Timing of this process showed that it took around 5 seconds to complete, so well within the 3 minute limit in the specification.

This test verifies milestone 5.9.1.2. The resolution of the image is 640x480.

8.1.4 Test: Change Resolution - (mh)

Since the option to change resolution was a low priority task it was implemented and tested after the basic main system - including image transmission over the autopilot link - was operational.

This test involved sending a Change Camera Resolution command to the payload module over the autopilot link from the ground station, checking the debug line for the correct response and viewing the images downloaded from the camera to verify

resolutions.

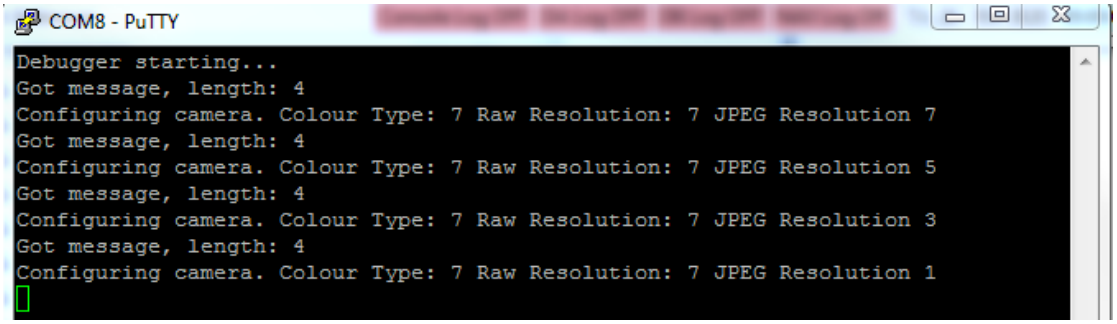


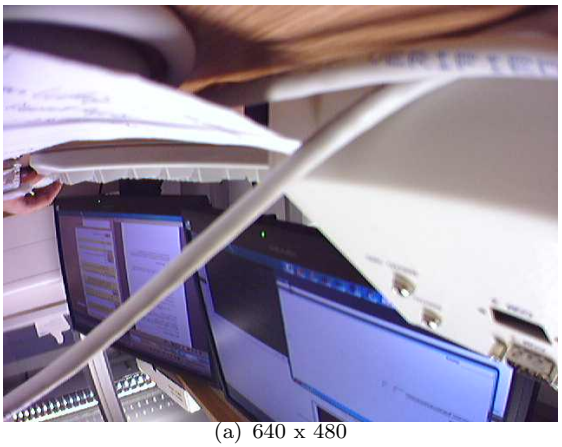
FIGURE 8.4: Test images captured using the integrated implementation of the camera controller

Figure 8.4 shows the debug information sent from the payload module, the four "Configuring Camera" messages in the figure correspond to resolution change messages sent from the ground station. The resolution was changed to the four JPEG resolution values available one after another:

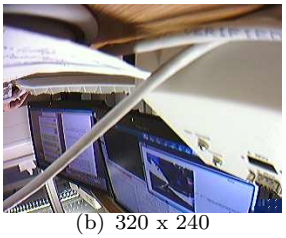
- **640 x 480** - Corresponds to "JPEG Resolution 7"
- **320 x 240** - Corresponds to "JPEG Resolution 5"
- **160 x 128** - Corresponds to "JPEG Resolution 3"
- **80 x 64** - Corresponds to "JPEG Resolution 1"

The presence of the debug information recieved from the payload does not validate anything alone, therefore an image was taken at each resolution, shown in figure 8.5.

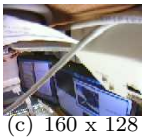
This test validates milestones 5.9.1.3 and 5.9.2.6.



(a) 640 x 480



(b) 320 x 240



(c) 160 x 128

(d) 80 x 64

FIGURE 8.5: Images taken at various resolutions - these were saved by the ground station software after they were downloaded over the autopilot link.

8.2 Payload to Ground Station Communication Testing - (mh)

8.2.1 Test: Establishing Contact with Autopilot

This test simply requires the controller to establish contact with the autopilot, recieving and responding to transmit tokens sent by the autopilot. Figure 8.6 shows that payload is responding to the transmit tokens sent by the autopilot and not timing out before sending them. Figure 8.7 shows a scope trace of the TX and RX lines of the payload - autopilot link

This test verifies milesone 5.9.2.1.

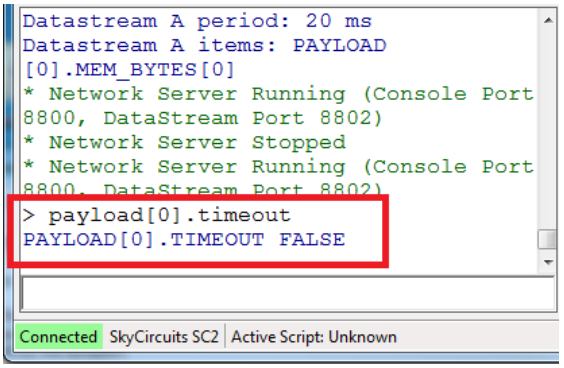


FIGURE 8.6: Timeout false on the customers ground station software shows that the payload is responding to transmit tokens correctly.

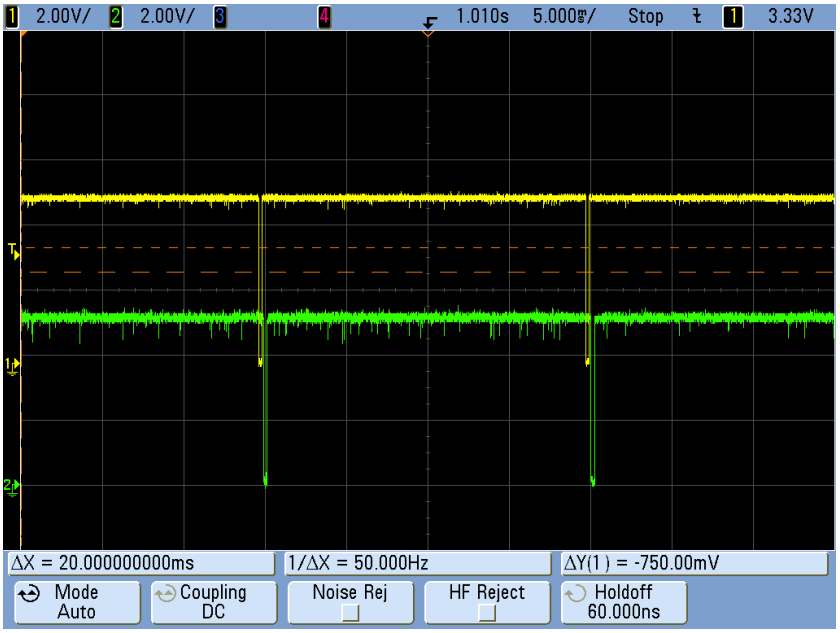


FIGURE 8.7: Scope trace showing the autopilot sending the transmit token as the top trace (autopilot TX) and the payload responding as the bottom trace (autopilot RX.)

8.2.2 Test: Payload Setting Shared Memory on Autopilot

This test simply attempts to validate the ability of the payload module to set shared memory on the autopilot. By setting the shared memory to one value on the ground station and then changing it by setting it with the payload module we can verify the payload module has set the value correctly.

Figure 8.8 shows the shared memory being set by the user on the ground station software and then queried after the payload has set it - the memory has changed to the value requested by the payload.

Milestone 5.9.2.2 is validated.

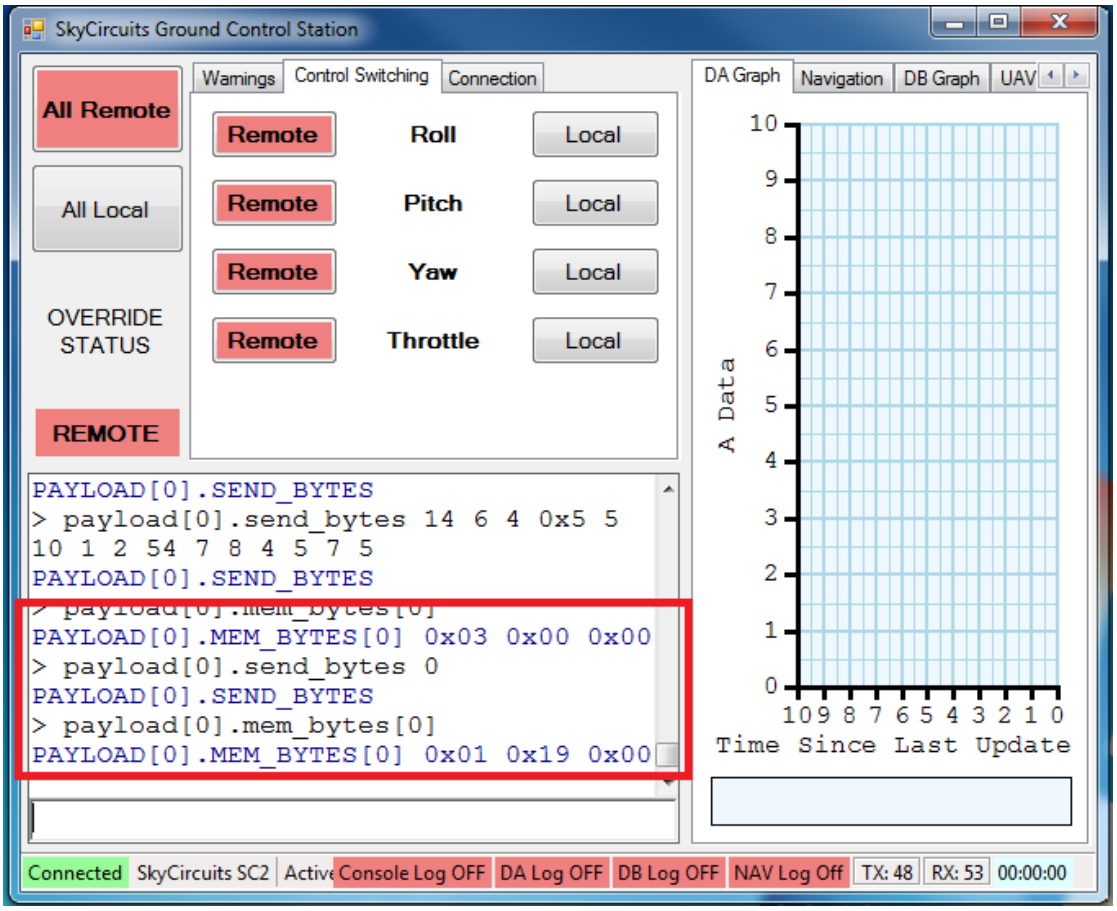


FIGURE 8.8: mem_bytes changes to the expected value after first query.

8.2.3 Test: Payload Receiving Messages from Ground Station

This test attempts to validate the ability of the payload to receive and decode messages from the autopilot. A message of length 1, content: 0x00 is sent using `payload[0].send_bytes 0` to the payload through the ground station software. The debug connection is used to validate that the correct message is received, shown in figure 8.9. The test is successful since the data is received by the payload as expected.

Milestone 5.9.2.3 is validated.

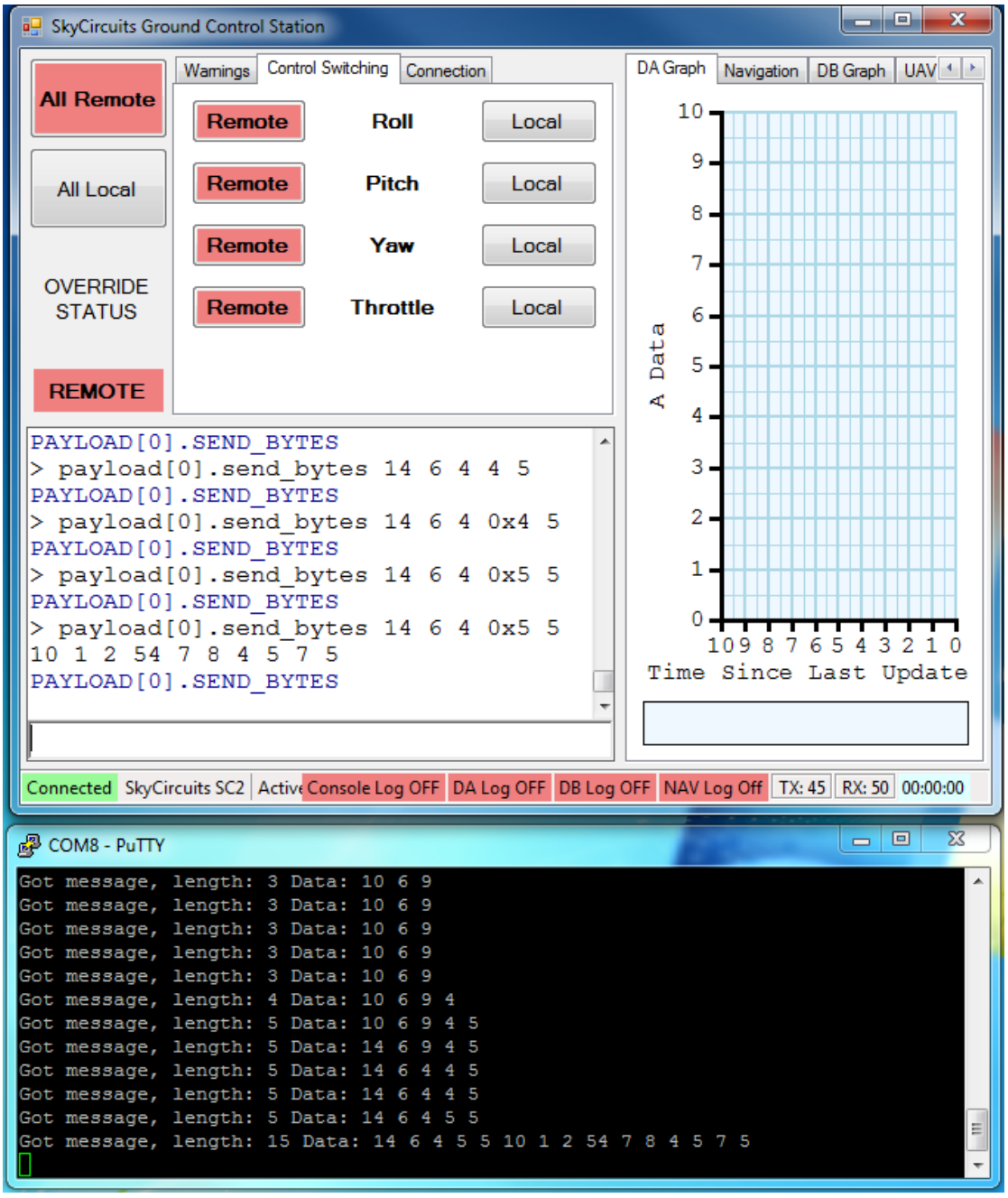


FIGURE 8.9: Payload (bottom) responding to send_bytes command.

8.2.4 Test: Image Sending to Ground Station via Autopilot Link

Initially in order to test payload controller to ground station communications implementation a PC console program was written to receive the incoming image data payload controller. By inputting a `payload[0].send_bytes 0` command into the ground station software the payload module triggers an image capture and the console application requests a download.

Figure 8.10 shows the download of an image from the payload controller to ground station via the autopilot link while figure 8.11 shows an example of an image downloaded. This test validates milestones 5.9.2.4 and 5.9.2.5.

This test was completed on the breadboard prototype, validating milestone 5.9.4.1.

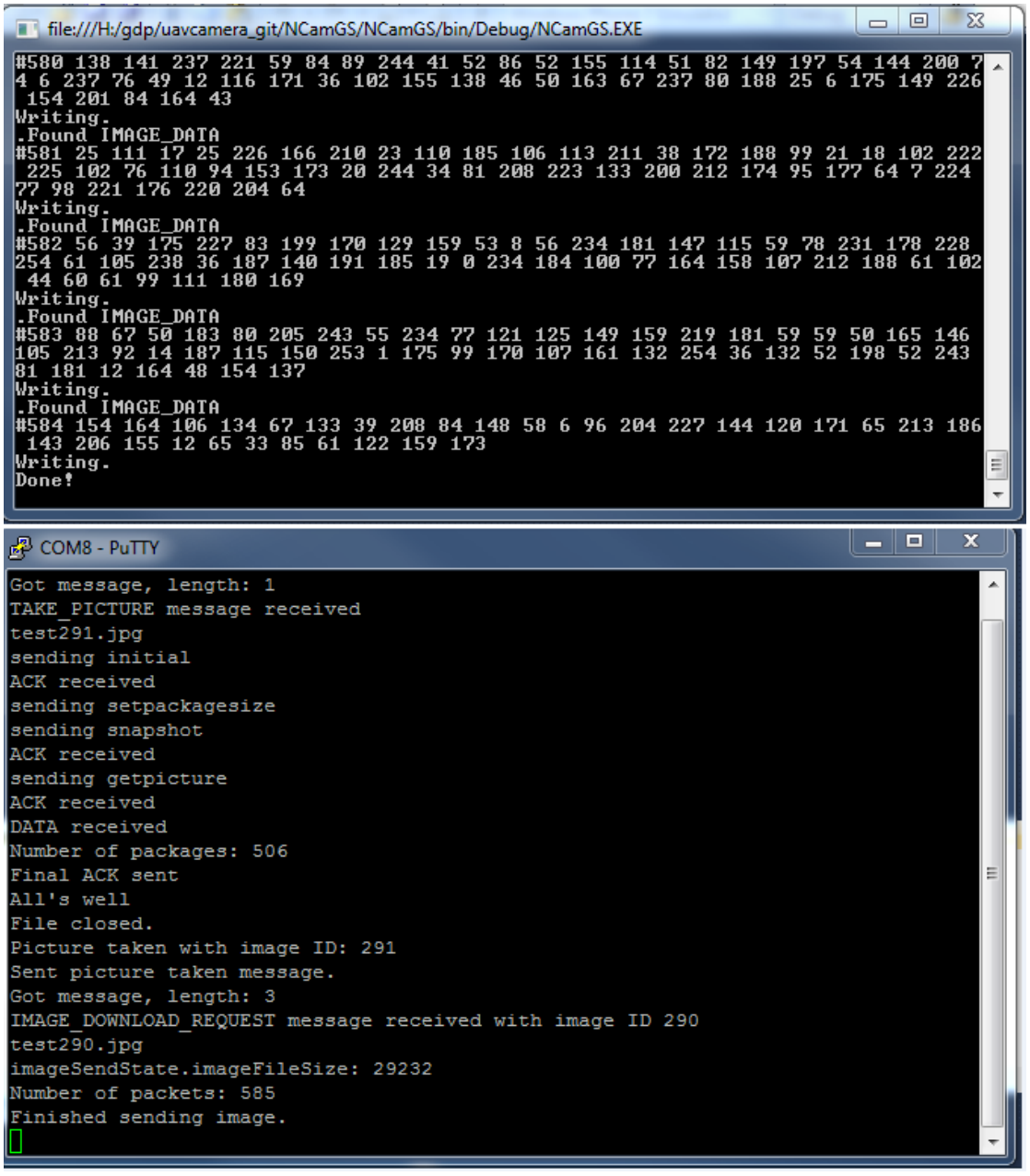


FIGURE 8.10: Screenshot showing test of image capture and send via autopilot link. Top window is ground station console software recieving image and bottom console window is debug infomation from the payload controller.

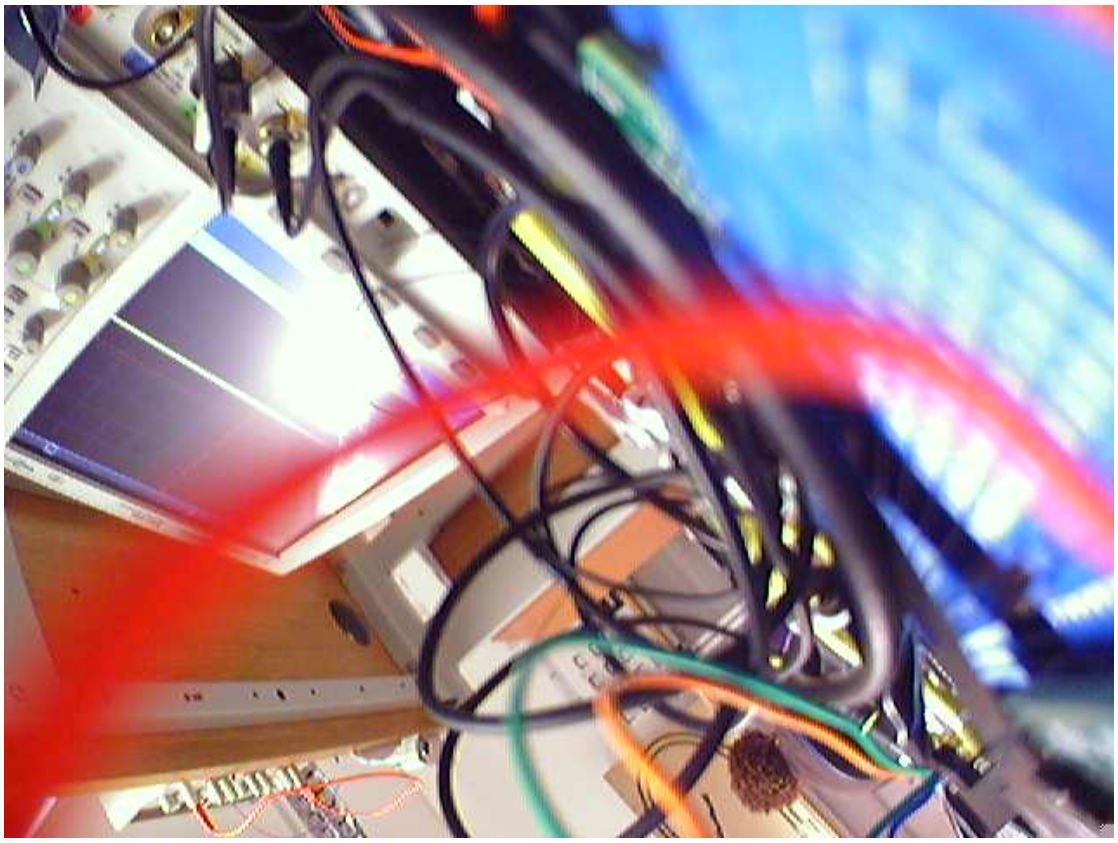


FIGURE 8.11: Example of image downloaded while testing payload to ground station image sending

8.3 Ground Station Image Viewer (ps)

The GUI needs extensive testing. This is because each individual feature has to be individually tested. Because of this, some functions of the UAV that are not dependent on the GUI are tested with a console application in order to save time.

8.3.1 Using Console Application: Testing Connection to UAV, Sending a Token to the Streaming Port

The UAV connector uses a .NET class called System.Net.Sockets. The methods that we use in this class are Sockets.Connect(), Sockets.Send(), Sockets.Receive(). To test the function of this class a separate console application has been developed to debug only a specific part of the program. Sockets.Connect() can be tested by using the Console to display the error code of the running application. Users are not allowed to change the port and host names, as these are set values. Another problem may be that one has forgotten to connect the UAV / enable its Network Server. Appendix D.3

shows a Connector class that is used for testing the Sockets.Connect(), Read() and Write() methods. Figure 8.12 shows a console application testing the Connect() function. The method is to send a zero token to the data stream and the GCS program will display the text,`'' New datastream client connected from 127.0.0.1:49586''` which is highlighted in green. This text shows that the program works properly and we have accessed to the datastream port of the UAV. However, the command that sends the datastream is not tested in this part. This will test the Milestone 5.9.5.1.

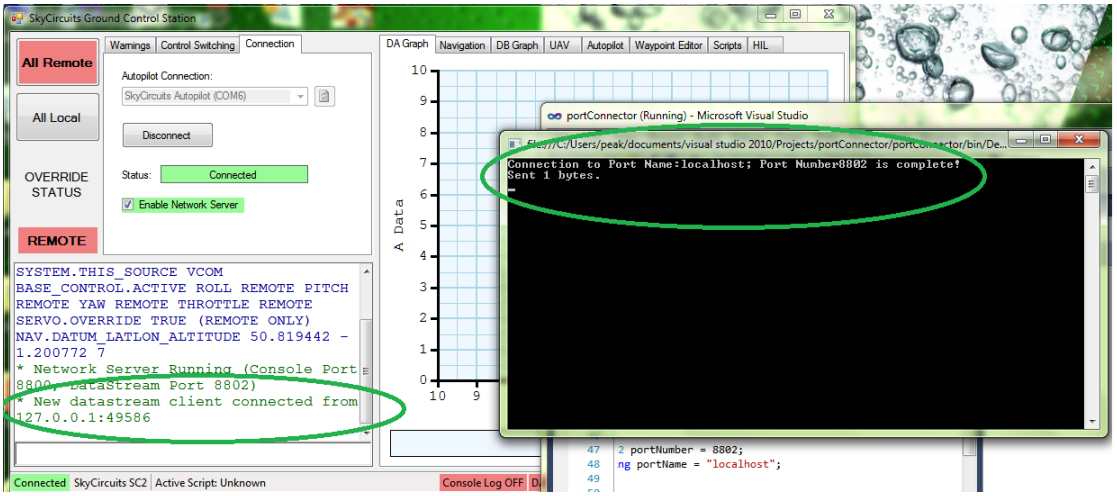


FIGURE 8.12: A screen shot showing the connection with the UAV was successful

8.3.2 Using Console Application: Testing Connection to UAV, Receive Data from Stream Port

Figure 8.13 shows a data stream displaying in our test console application. Firstly, in the GCS program, we write "da 30 ht". This code makes the height data stream to the ground station, this data then displays on both the graph and in our console application, and we can see the number clearly changes when the height changes. This is testing that the data stream is working correctly and we have a correct way of getting the data. Therefore, the Milestone 5.9.5.2 is validated.

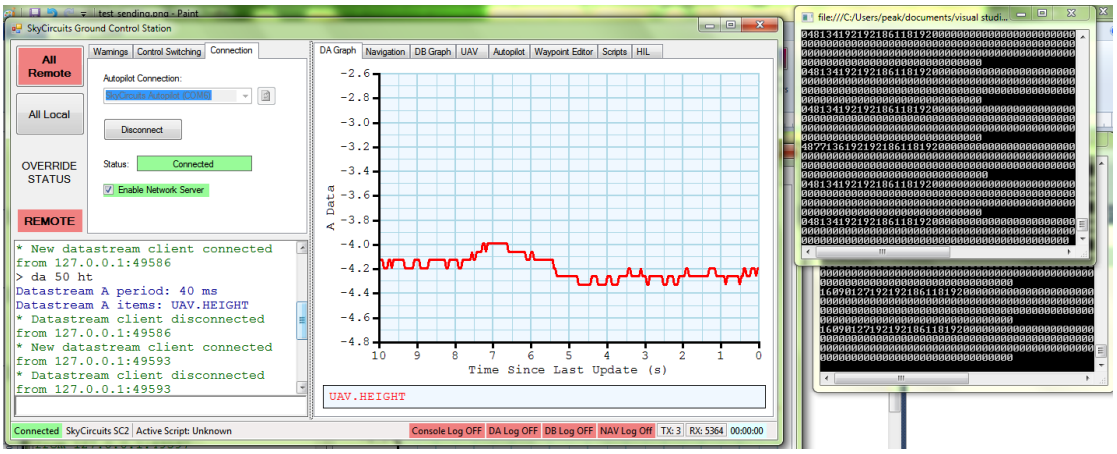


FIGURE 8.13: A screen shot shows that the data streams successfully

8.3.3 Using Console Application: Testing Send Command to Console Port

Figure 8.14 shows communication with the console port. The ground station program allows the user to use the console port to send a string command to it. This string command can be tested by the ground station program, which will produce a line with the "@" sign on the front of the command sent from the outside of the program. If the data is correct, the ground station program will detect it and send the data to UAV. In the console program, when the user types anything in the console line, it will send byte data to the GCS program. If the text is a command, the program will process the command and send a byte value as set in the command to the UAV. The first test is try and type 'hello' into the console line. The GCS will then see the word 'hello' but there is no command available for it. Therefore, in the GCS console line, it display 'unknown class HELLO'. Because we send the word via the console line, there is a sign in front of the 'hello' word in GCS program. The next line we type in is "da 50 b". GCS software recognises this command, therefore it changes the display in the graph to give information about banking of the UAV. This test validates Milestone 5.9.5.1.

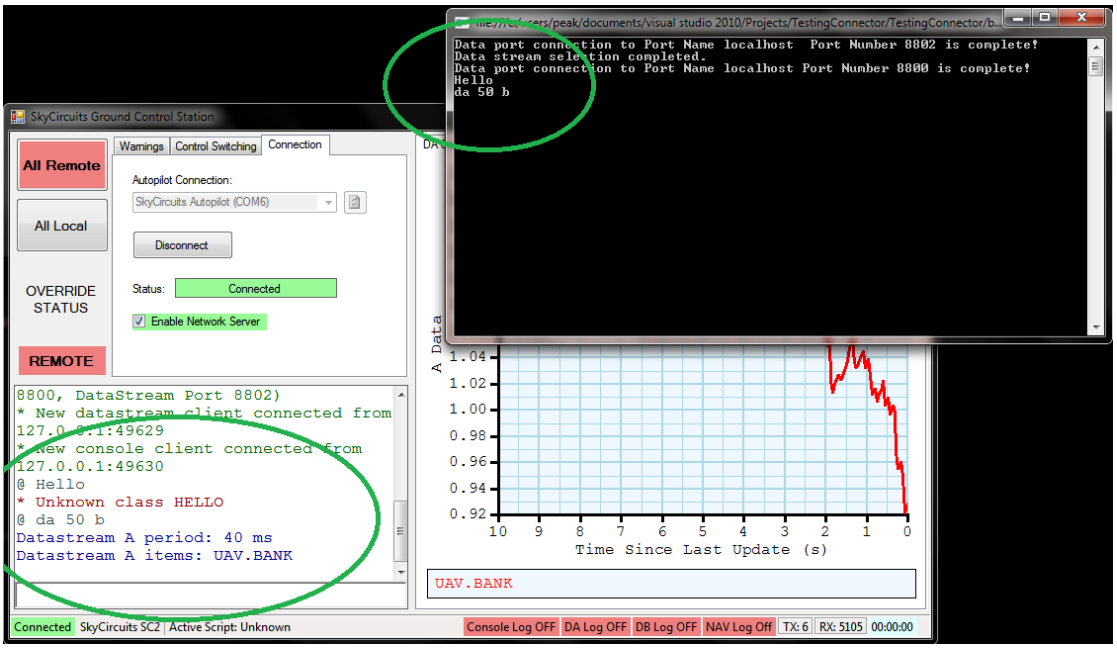


FIGURE 8.14: A screen shot showing the text sending change the GCS (Ground Control Station) graph value

8.3.4 Using Window Application: Testing Get Image Button

The connection between the aircraft and the ground station is using a TCP port, which allows data to be sent from the aircraft in any format. In order to test this relationship, a camera will take a picture. If the pictureBox displays an image the similar to that expected, the image data is correct. Figure 8.15 shows completed, combined classes together and receiving image data. This testing combines all the existing methods together in the Window application. At this point the connector class will be in the main window application. When the user clicks the 'Get Picture' button on the window application, the program will communicate bidirectionally, sending and receiving that which already stated in section 7.8. From the figure, it clear that the image is downloading and the image has displayed correctly. This test validates Milestone 5.9.5.3.

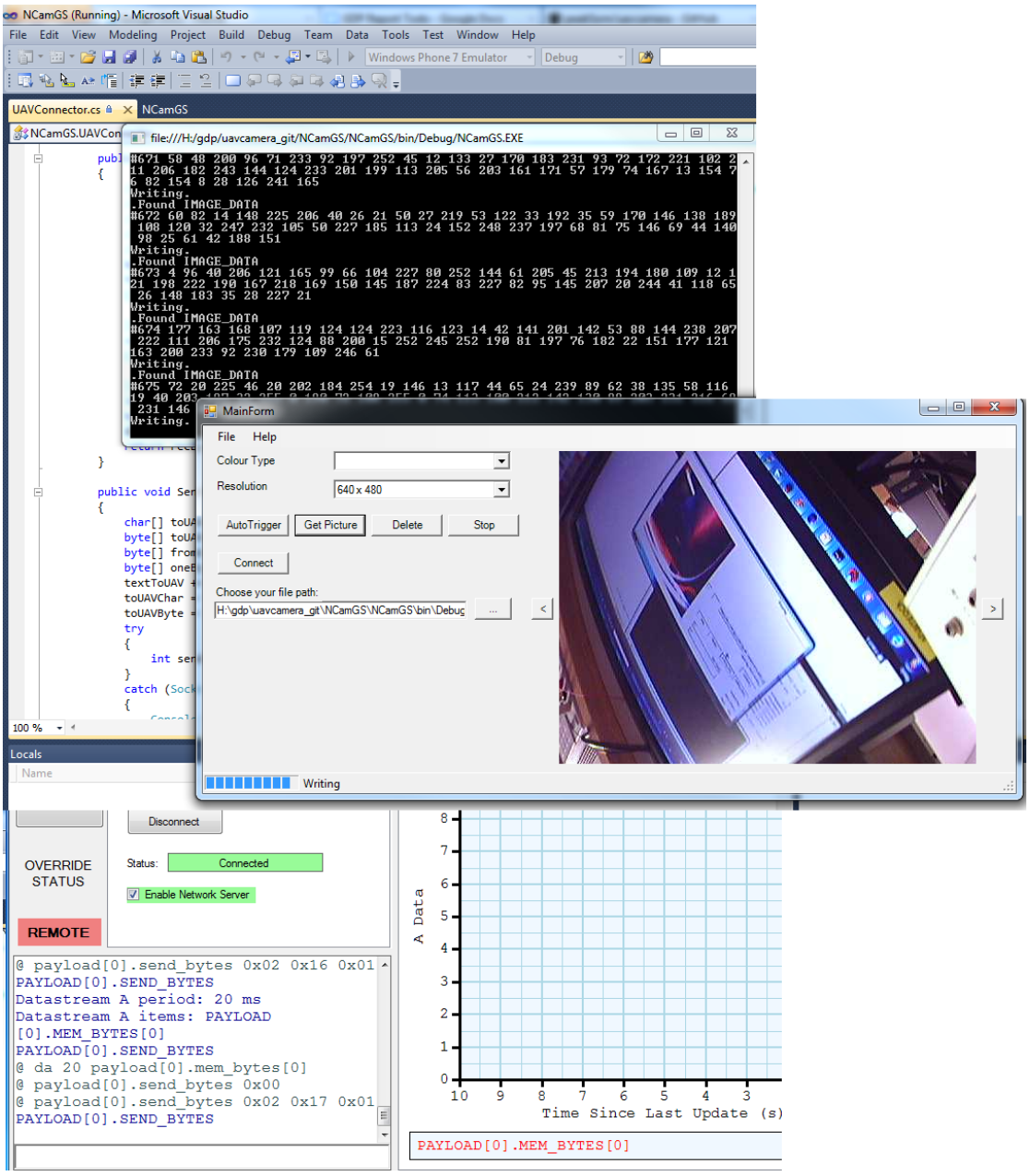


FIGURE 8.15: A screen shot showing the picture has been taken correctly

8.3.5 Using Window Application: Testing Cancel Image

When the "Cancel" Button is pressed the image stops downloading and is saved in its current state. This test was successful fulfilles specification items 3.4.5.2 and 3.4.5.4.

8.3.6 Using Window Application: Testing Resolution Option

When the resolution option changes, the Image viewer program will change the command byte that gets sent to the Console Port of the UAV. This can be tested by changing the combo box in the image viewer program and clicking on the Get Picture button. If it is working correctly, the ground station will see the correct resolution chosen of the picture taken. Figure 8.16 shows a working resolution changing module. The PuTTy terminal shows the resolution command that was sent back from the payload. This proves that the resolution option is working. This test completes Milestone 5.9.2.6.

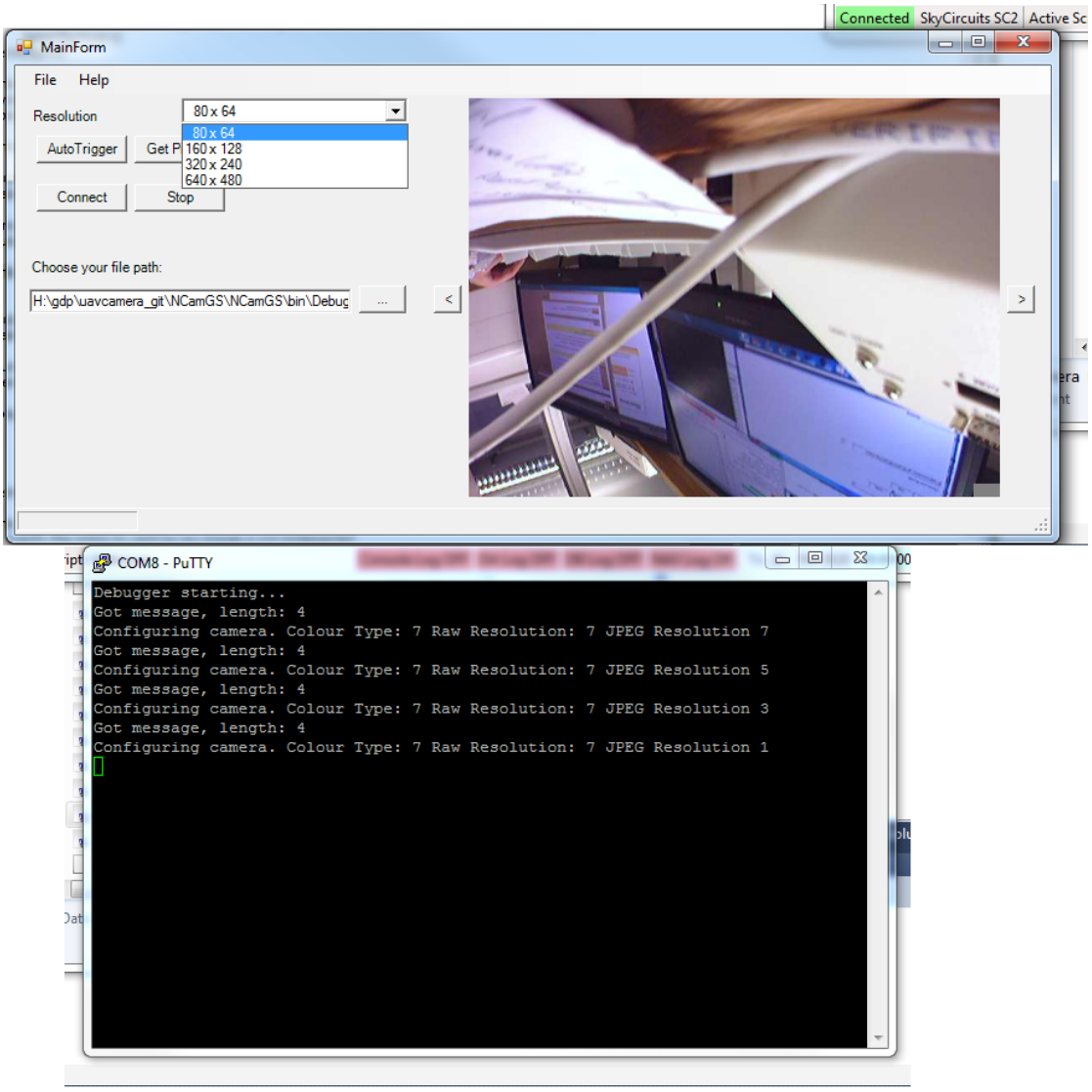


FIGURE 8.16: A screen shot showing that a chosen resolution sends different signals to the payload

8.3.7 Functional User Interface

The tests listed above verifying Milestone 5.9.5.4 are completed. Figure 8.17 shows a complete user interface that is functional and ready for the customer to use.

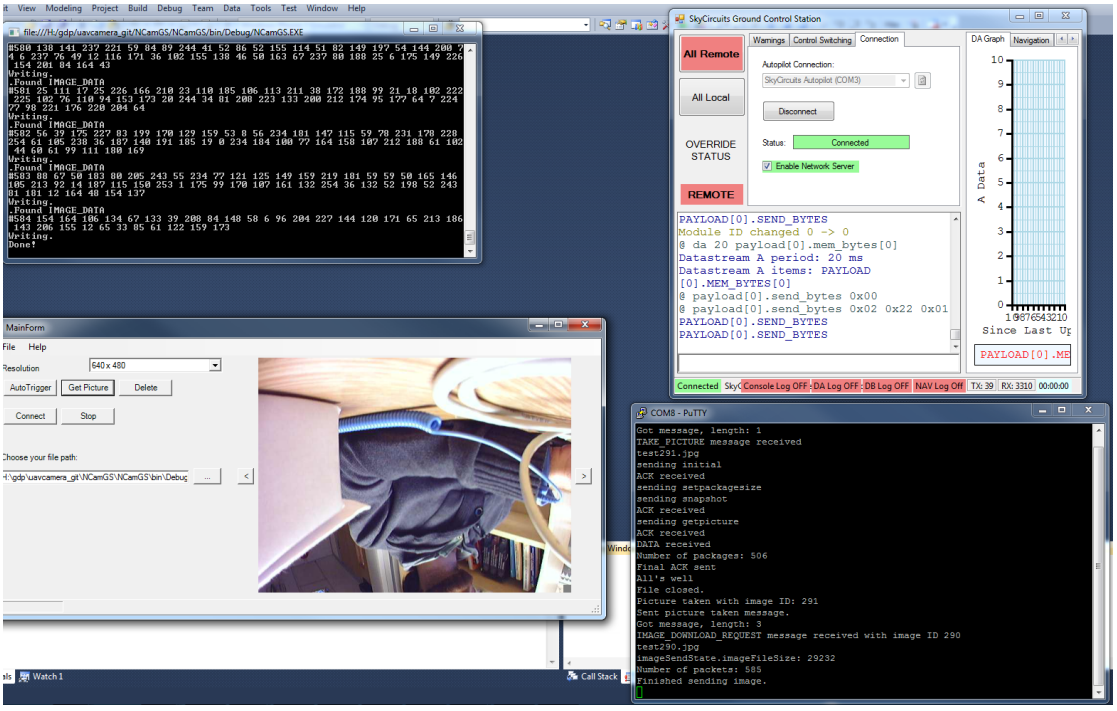


FIGURE 8.17: A screen shot showing a complete UI

8.4 JPEG Extractor Testing - (ms)

8.4.1 Introduction

Before the JPEG extractor can be integrated into the AVR microcontroller, it is necessary to make sure that it is successful in extracting the header information. This is achieved by coding the JPEG extractor as a C executable file which takes in a JPEG image file and prints out the information that it will send to the ground station software.

In order to ensure the validity of the information that is extracted from the JPEG file, the extractor's output is compared against the output of the JPEGsnoop file extractor. JPEGsnoop is a free Windows application which takes a JPEG image and outputs the information content of all of the JPEG file's segments [12] . This software provides a much more complete JPEG segment information extraction and includes all the information extracted from the created JPEG extractor C executable. This is a sample output of the JPEGSnoop application applied to the Leaves2.jpg test image.

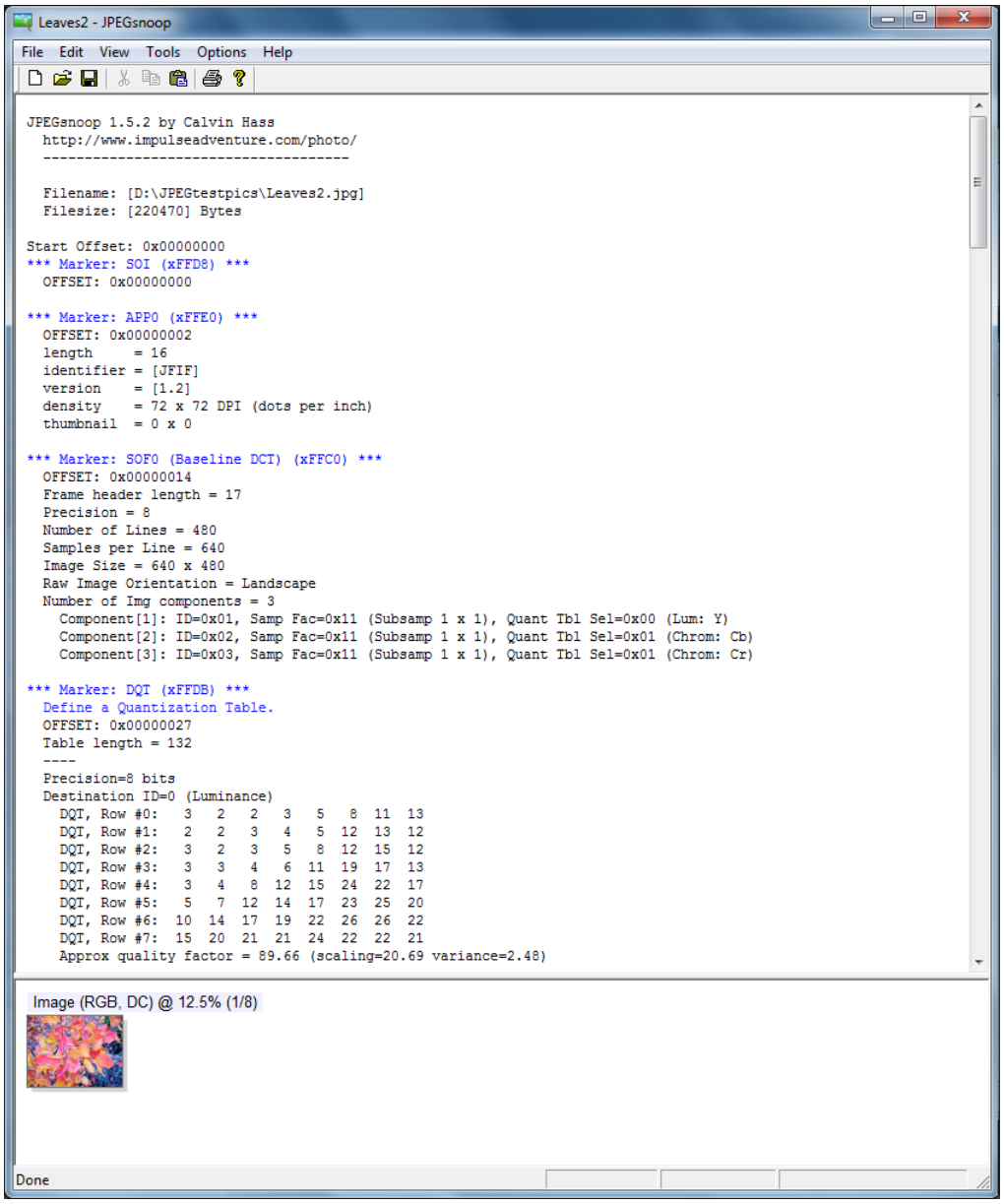


FIGURE 8.18: JPEG segment information obtained from JPEGSnoop.exe

The images used for the basis of the following tests were 8 JPEG images of size 640x480 or less. The pictures are deliberately chosen to be complex in order to ensure that the JPEG extractor can handle any image given to it by the camera. The comparison figures below show some of the different images used for testing. During all 8 tests, the output of the C executable was compared with the output of JPEGSnoop.exe.

8.4.2 Test Results

This section will detail how the C executable's output was compared against the output of JPEGSnoop for each important JPEG segment. The C executable was

compiled and run using the Eclipse development platform and the output was printed out on the Eclipse console display.

The following segments were tested:

8.4.2.1 Test: SOF0

The SOF0 information is displayed almost identically between both outputs. The C executable extractor does not derive the image size nor the image orientation, because that information will not be needed for a progressive display of the image.

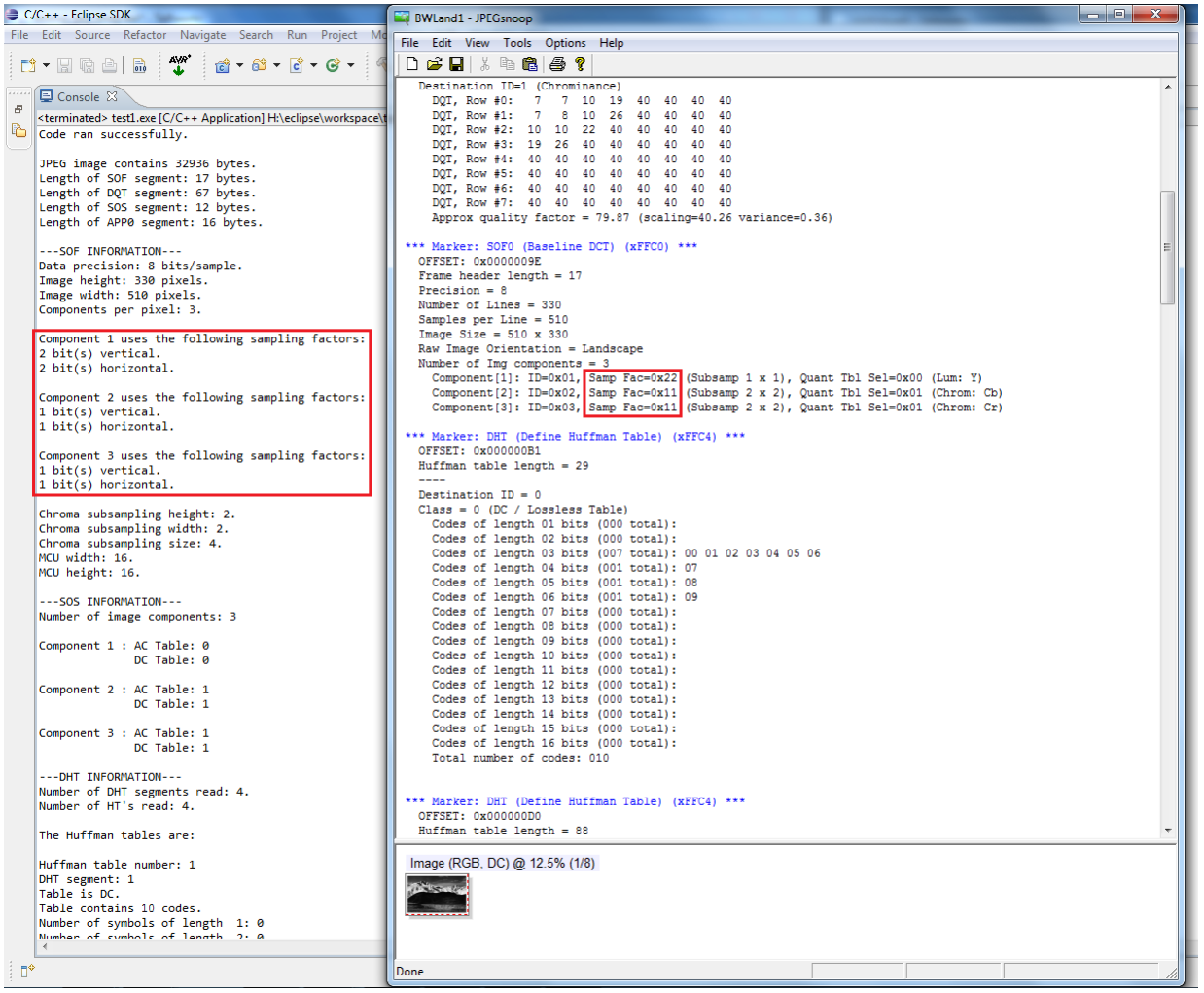


FIGURE 8.19: SOF0 segment information obtained from JPEGSnoop.exe

Note that the sampling factors on each display have been highlighted with red squares. This is to avoid confusion between the sampling factor and the subsampling factor.

8.4.2.2 Test: DHT

The extractor keeps track of the DHT segment where the Huffman table was extracted to better understand the layout of the JPEG file's DHT segments. The JPEGSnoop extractor simply places the information in a more compact format compared to the display method used by the executable, which has no bearing on the actual information stored.

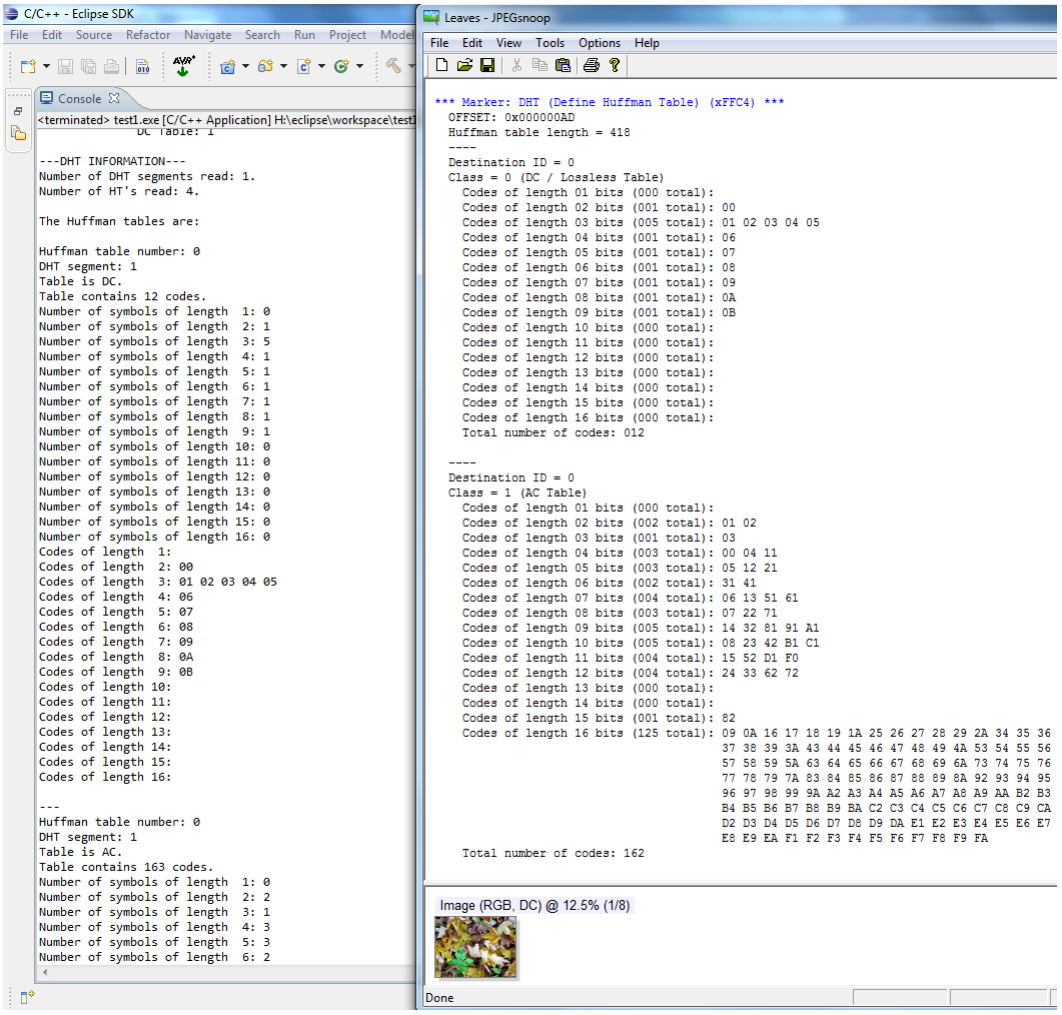


FIGURE 8.20: DHT segment information obtained from JPEGSnoop.exe

8.4.2.3 Test: SOS

The SOS segment information is read in correctly. The red squares indicate the DHT tables which are the same in both outputs.

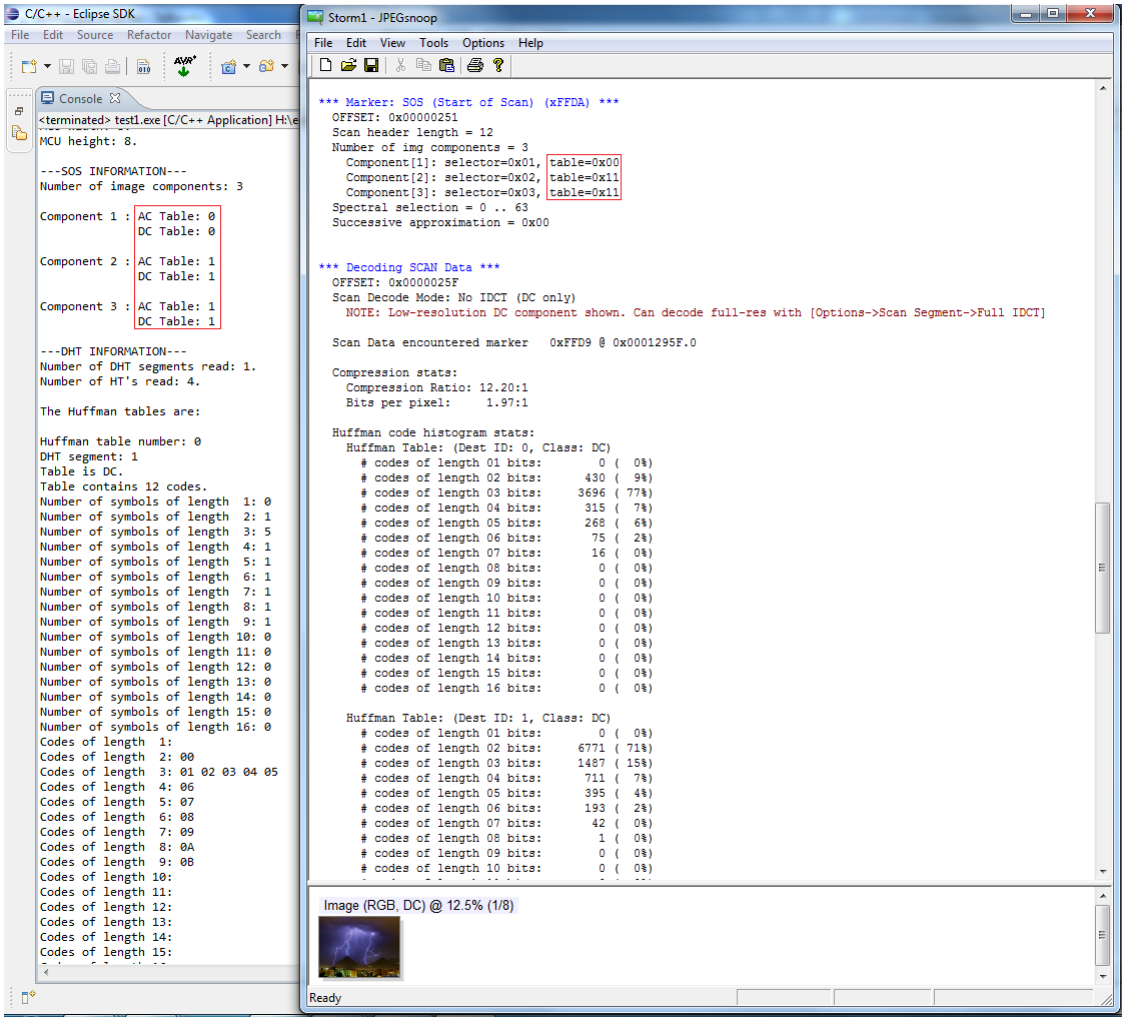


FIGURE 8.21: SOS segment information obtained from JPEGSnoop.exe

8.5 Test: Waypoint Triggering

Figure 8.22 shows the autopilot triggering camera capture and 8.23 is an example image taken in this way.

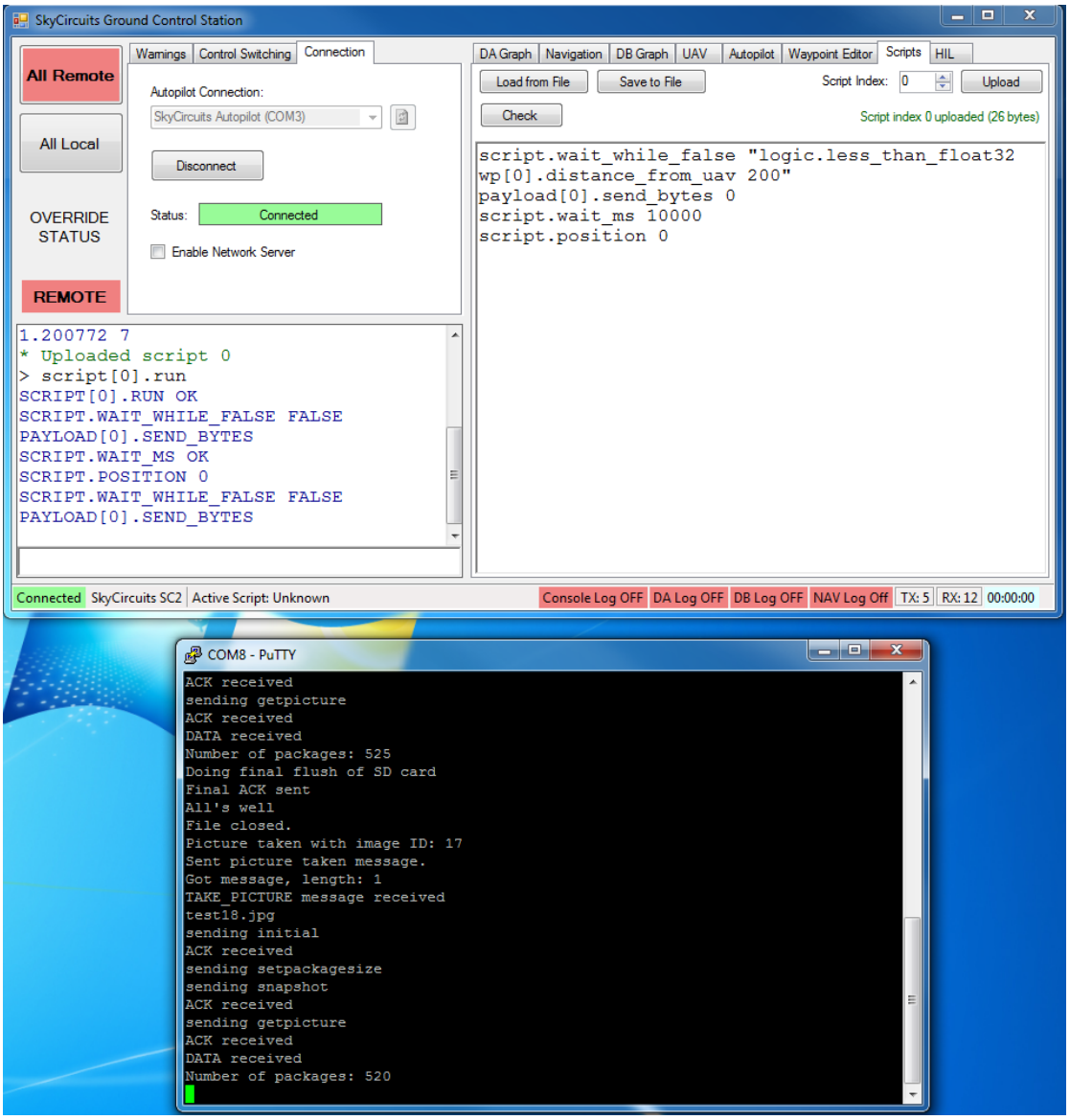


FIGURE 8.22: Autopilot triggering image capture from waypoint script. The payload debug terminal shows the capture being triggered by the script.

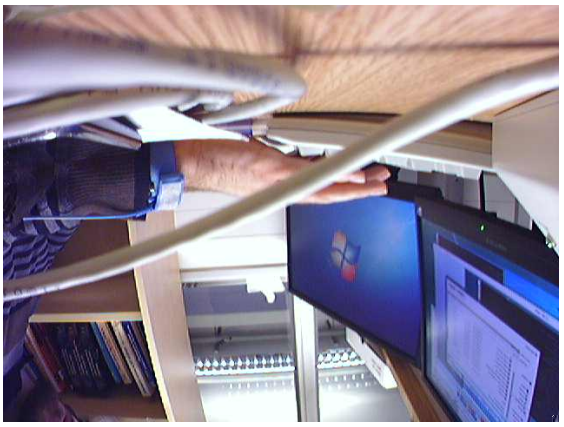


FIGURE 8.23: An example of an image taken from waypoint triggering.

8.6 System Test: 640 x 480 Image Capture and Download Time - (mh)

We now measure the average time taken to capture and download a single 640 x 480 pixel image from the camera through the entire capture progress, from pressing the "Get Picture" button on the UI to the being downloaded over the serial link to and displaying on the UI. Please note that this section assumes no packets are lost during the download process.

A total of 10 measurements were taken, as shown in table 8.1. The time taken for the image capture to the SD card and subsequent download are both shown. A mean time of **19 seconds** can be seen. Variations in time are down to the variations in size of the JPEG images. This is well within the 3 minutes required by the specification (see 3.4.2).

This test was completed on the final PCB, validating milestone 5.9.4.3.

8.7 System Test: Final Module Weight - (ab)

The final populated PCB and camera weighed 57g alone and 70g with the plastic container for camera - well below the 250g specified in 3.4.3.

8.8 Flight Testing (ab)

In theory, not much should change between our laboratory tests and a flight test: the only difference is that the payload is operated in a harsher environment, and the

Capture to SD (s)	**Download to UI (s)**	**Total Time from Get Picture button to Display on UI (s)**
8	13	21
9	15	24
7	11	18
6	11	17
7	12	19
7	11	18
7	11	18
7	12	19
8	11	19
7	11	18
	Mean:	19.1

TABLE 8.1: Download times for 640 x 480 image, recorded by stopwatch so subject to some human error. In all cases download was started by pressing the "Get Picture" button on the ground station UI.

Autopilot to Ground Station is via a XBee wireless link, not a USB Serial cable. The robustness of the wireless link is not as good as the serial link, and we expect packets to either be corrupted or dropped entirely.

It was initially planned to perform a flight test with our customer and his supporting team, our supervisor, and our entire group at Cheesefoot Head in Winchester on the 9th December 2011.

Unfortunately, on the 7th December, our board camera stopped functioning. This meant the planned flight test has to be cancelled.

Another flight test is set to be scheduled in early January, weather-permitting. A third board camera has been ordered, and our payload module appears to be functioning correctly, this bodes well for our upcoming flight test. We plan to develop the packet verification and automatic repeat request parts of our software in the intervening time.

Chapter 9

Conclusions (ab)

This section will discuss to what extent we have fulfilled the requirements of this project, mainly in terms of our initial specification, including a justification for why certain aspects were not met, and an analysis of whether the project, as it currently stands, could be reasonably presented to our customer as a working solution.

9.1 Commented Specification

The initial specification is available as Appendix G.

This is then analysed and justified in Section 3

9.1.1 Fully Completed Parts

- 3.4.2 : The mean time to download an image, with no packet loss, is 19 seconds ,reference 8.6 . This figure is obtained from testing with the serial link only.
- 3.4.3 : The combined weight of the electronic module and board camera is 57g. Combined with the camera enclosure and ethernet cable (i.e. the entire additional weight to the UAV to add our module), this is 70g. 8.7
- 3.4.4 : We are able to get an image resolution of up to 640×480 pixels. 8.6 .
- 3.4.5.1 : We provide software that prompts the UAV to capture an image, download it and then displays it on screen. 8.3
- 3.4.5.2 3.4.5.4 : Our software is able to cancel the download of an image mid-transmission. When the "Cancel" button is clicked during image transmission, the image is saved in its partial form in the local directory, and in its full form on the SD card. If it is cancelled before transmission begins, then the full image is still written to the SD card.

- 3.4.5.5 : Our software allows you to select any resolution that is featured on the camera, and download an image at that resolution. This feature has been recently implemented and tested.

- 3.4.5.9 : We are able to transmit colour images as opposed to black and white due to our choice of camera - our camera is only able to provide colour images. Therefore it is implemented, but only through our choice of camera, not through design.

9.1.2 Partially Completed Parts

- 3.4.1 : This task was lowered in priority when we discovered that our camera compressed JPEG images a lot more severely than we were expecting (image files are approximately 30kB in size, we were expecting these files to be 900kB in size). To reflect the time invested in this, we have produced some code that partially implements this part of the specification, available in section 8.4

- 3.4.5.6 : A progress indictor does exist (Appendix I, p7) , but the display only indicates the number of packets received - it does not indicate the time remaining to download as this includes the time taken to capture an image from the camera and to store it on the SD card.

9.1.3 Incomplete Parts

- 3.4.5.3 : Resending of the same image is not implemented. This is, however, a low priority task, which could have been implemented if we had more time. In theory, this would be rather trivial to implement.

- 3.4.5.4

- 3.4.5.8 : Automatic image capture at user-specified time intervals was another low priority task which we were unable to implement, but which should be rather trivial to implement due to the SkyCircuits software's scripting interface, but slightly more difficult than the above two.

- 3.4.5.10 : Selecting between colour and black-and-white images was included in the specification as we considered that this would reduce the image transmission time. However, this was not a feature supported by our board camera, implementing it on our AVR would be a non-trivial process, and the priority to reduce transmission time was reduced as the colour image files themselves were much smaller than expected.

9.2 Deliverables

This will discuss to what extent we have delivered everything we specified in our specification G

9.2.1 Hardware

3.5.1 : We have delivered a suitable board camera, and an electronic module on PCB which interfaces the camera to the autopilot's payload port. All schematics, layouts, gerber files etc. are provided in our central Github repository [21], and as Appendices E and F

9.2.2 Software

3.5.2 : The electronic module delivered to the customer is presented with firmware loaded on to it, and the user interface software presented as an executable file (requiring only that the .NET framework is installed on the PC, however this is the same dependancy as for the SkyCircuits software). All firmware for the electronic module, as well as source code for our executable Ground Station Software, is presented in

9.2.3 Documentation

3.5.3 : Documentation for our Ground Station Software is delivered with this project see appendix I. It is assumed that our customers' clients are technically competent, therefore documentation on how to order a PCB with our gerber files, how to construct the PCB, and how to flash an ATmega644P with our source code was deemed unnecessary.

9.2.4 Public Repository

3.5.4 : All files related to our project are available in our central GitHub repository [21]. Most things in there are open-source: source code is licensed under GPLv3 (GNU Public License version 3), [36] images (including PCB layouts and schematic designs) are licensed under CC BY-SA 2.0 UK (Creative Commons Attribution ShareAlike) [37], and all documentation under the FDL (Free Documentation License), [38]. This is as-requested, and should encourage sharing this information amongst our customers' clients.

9.3 General Conclusions

In conclusion, we have presented the customer with exactly what was requested: an electronic module that interfaces a board camera with the provided Autopilot module, which exploits the Payload feature of the module and copes well with the low-bandwidth Serial link.

Not only this, but we have delivered a fully Open-Sourced project that could be replicated by anyone with access to a SkyCircuits Autopilot module, and accompanying software.

We have mostly fulfilled all of the objectives we set out in our initial Specification 3, the reason for us neglecting to completely fulfil it being either it becoming obsolete (for the high- and medium- priority tasks), or a simple lack of time towards the end of the project.

Our customer was satisfied enough with the Il Matto-based solution to be prepared to flight-test that, and present it to his customers. We have enhanced both the payload module and the Ground Station Software since. It is unfortunate that we have not been able to do this test before the report deadline, but it is planned to perform this test in early January, before the final presentations. All indications point to this test flight being successful, should weather conditions be favourable.

Chapter 10

Evaluation

10.1 Evaluation on Project Management (ps)

In order to successfully meet the specifications of the project, the task needs to be prioritized so that high priority tasks are implemented before the others. The tasks with high risk, many dependencies, and requiring the most time to complete were the first set of tasks to be completed. Because we prioritized the tasks well, we had enough time to implement the high priority objectives of the customer's specification. Although some low priority tasks were completed some tasks such as the progressive image display have been omitted due to time constraints.

The project management took advantage of group meetings. Each members of the group had to produce a formal documentation in order to keep track of and assign the new tasks. Some meetings did not have enough progress done beforehand and were not as effective as hoped. If the group agreed to and managed to implement something on every meeting, it would have made the implementation of the project much faster.

One thing to be learned from this project is the delivery of components which can be very fast or very slow. Therefore, before ordering parts, the group would need to check the delivery time and make sure that the delay had a minimal effect on the progress of work. The hardware arrived on a slightly delayed and so some members of the team who were assigned tasks related to the hardware were forced to reallocate and perform other tasks while waiting for them to be delivered.

10.2 Group Meeting Evaluation (ms)

10.2.1 Group Meetings

Having meetings on a weekly basis proved to be very efficient for the group. Because individual group members met with each other on a regular basis throughout the project outside the weekly meetings, the meetings allowed enough time between them to keep them from being redundant.

When certain important deadlines were nearing, the group would naturally meet up on a more frequent basis and attempt to tackle important challenges together, even when no meeting was planned. The meetings allowed the group to assess their progress and were vital to avoiding tasks overrunning.

Minutes of these meetings are available both in our repository [21] (documents/minutes), and as an appendix B.

10.2.2 Minutes Evaluation

Minutes were taken during each weekly meeting. Some minutes were taken during other important meetings as well, including meetings with the customer and the supervisor. Although, the group member who took minutes was not monitored as well as it could have been, this did not have any significant effect on the quality and/or consistency of the minutes.

The minutes taken were useful for making sure that all group members were able to check what tasks they were assigned and catch up on meetings that they could not assist. The only major setback with the minutes was the lack of urgency with which they were distributed to the other group members. Some group members would update their minutes immediately after the meeting, whereas others would not until after the next meeting. Despite not seriously setting back the project, this undermined the usefulness of a nonetheless helpful tool.

10.3 Evaluation on Group Communication (ps)

10.3.1 Source Control

As describe in section4.5.3, there were problems with subversion where some of the files that could not be put into the public folder were accidentally placed there. This cause some of the group members tries to fixed this problem. Although the subversion was effective, we have decided to change it to git. Git is easy to use and all the

members can learn it fast and there are no furthur problem. This save time of combining report because git warn the user when they are submitting something that other member is working on it. Therefore, all the work flowed well and we have minimized the submission of work problems.

10.3.2 E-mail

The way of communication by email was effective for our group. Because all the members know how to use it, there is no time wasting on learning a new way of communication. However, it is likely that some of the group members misread some of the email or did not react to them immediately. To make this a more powerful way of communication and to ensure that all the members have read the message, the group should have had a rule that on every email, there must be a reply from all of the group member.

10.3.3 Supervisor Meeting

The formal meeting with supervisor that we have described in section4.5 has been done successfully. In order to make sure these weekly meetings with the project supervisor were fruitful questions were prepared beforehand, to ensure that the project was on track and the most immediate tasks were made very clear. The customer also came to the meeting once in every two weeks and would evaluate the progress of the project. This made the project flow smoothly due to having instant feedback from the creator of the UAV module. The customer's considerate feedback and presence ensured a successful project. Nonetheless, the group could have made more out of the meetings by showing some of the progress on the report throughout the project development period. This would have saved the time needed to finish the report.

10.3.4 Telephone (ms)

Telephone was used rarely. Generally, it was used in emergencies, for example, when a group member was unable to attend a meeting and could not inform the others by e-mail. It was also used when members were unexpectedly absent from certain important meetings. Overall, e-mail and telephone communication complemented each other well.

10.3.5 Internet Relay Chat (ms)

As the project advanced during the development phase, more and more tasks required group members working together in order to be successfully completed. This is due to

separate modules needing to communicate with each other and having different people assigned into developing the different modules in parallel. As this became a more apparent issue, the group adopted two IRC channels to keep in constant contact with each other.

The first IRC channel was used for development and keeping the different module developers in contact with each other as much as possible. The second IRC channel was mainly used for the purpose of writing the project report, due to the large group work involved in creating one cohesive report from the efforts of different individuals.

Both IRC channels proved invaluable in the later phases of the project, but due to the individual nature of the early development phases of the project, were not necessary to be implemented early on. However, it would have been useful to prepare an IRC at the beginning of the project, if only to have another communication tool to help the group members work together.

10.4 Encountered Risks (ms)

10.4.1 Encountered Risk: Faulty Components

Two major components were found to be faulty very early on. The first ordered camera module arrived in an unresponsive state. Due to the price of the device, the group preferred attempting to debug the device before ordering a new camera module. Eventually however, another device needed to be ordered. The second camera module lasted for the majority of the development process, but also become unresponsive during the final weeks of the project, after the PCB had been received. A third camera was ordered immediately for testing purposes. Choosing a less fragile camera, or taking special precautions when dealing with the camera would have mitigated this risk better.

The other faulty component was the autopilot module which was given to the group by the customer. Due to the nature of the autopilot, being a custom component from the customer, it was not possible to simply order a new autopilot module. The group decided to contact the customer for help, and worked with the customer to debug the autopilot software as discussed in section 6.4.6.

10.4.2 Encountered Risk: Team Members Unavailable

Some members of the team were too occupied to attend all the weekly group meetings and the weekly meetings with the supervisor. However, this was not a severe problem and individual members were always willing to allocate meeting times when asked

directly. The group members were willing to warn the others when they would be unavailable for a large period of time. This issue did not cause large set-backs in the project.

10.4.3 Encountered Risk: Difficulties

Whenever difficulties were faced, the group member facing the difficulty would inform the other group members about the problem. If the difficulty concerns a high priority task, the individual group members would reassign tasks to make sure that all the high priority tasks were completed as soon as possible. Thanks to the skills audit (see section 4.3), this made sure that any difficulties faced in the high priority tasks were taken care of.

10.5 Actual Task Allocation (ps)

In the real work environment, the unexpected situation happens at all time. Each members of the group faced different problems. Some problems were easy to solved but some were very difficult and time consuming. The back up plan and unplanned problem solving skills are needed. Although many problems happened, but all the team members have been flexible to support each other and problems have been solved quite well. Figure10.1 shows an actual work that each member has done in the project. There are more individual work than the initial allocation because some team members are confident of working alone.

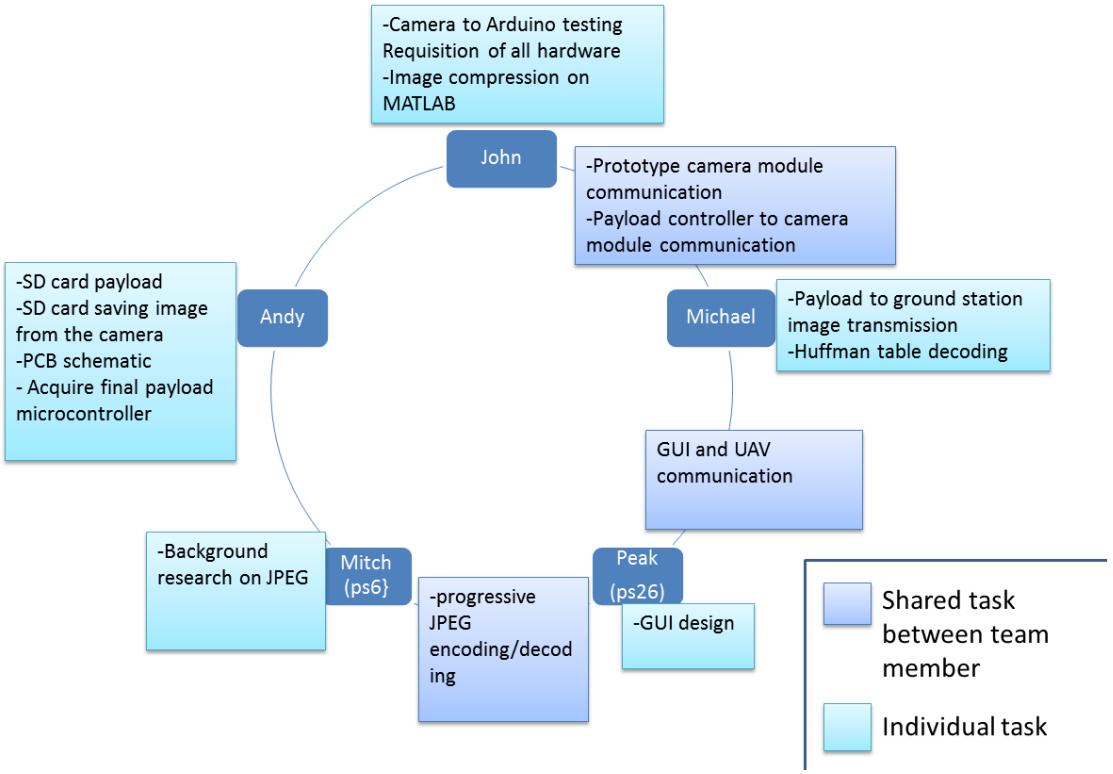


FIGURE 10.1: A diagram show a final work allocation of the team

Figure 3.1 shows a block diagram of the whole system that the developers has slitted in to blocks of tasks.

The hardwares have uncontrollable acquiring time. Some of the group members has problem with the delay of the hardware deliveries. Therefore, the time has to extended longer than the gantt chart. So those members were doing other tasks that is hardware independant. And at this stage, the background reading is very necessary so the developers has used this waiting time to plan and develop the software, so when the components arrive, all the task can be implemented.

The task that depend on acquiring the camera are the communication between the payload and the camera,prototype camera module, and image encoding. The problem that the developers have is the camera delivered is a faulty, the reordered of camera has been taken. These tasks has been suspended until the camera have arrived. Therefore, the new camera has to be purchased and the allocation of work is necessary. There are part that is not depended on the hardware such as encoding/decoding images, and ground station software. So the members who were assigning to work on the camera has been allocate to do the software part first.

The payload controller has the problem with the hardware part of the UAV. It did not respond to the UAV signal at first. Therefore, we assign another group member to

support this problems. The task leader of this has been assigned to do another software module which have to be implemented. This problem also need a support from the customer to update the UAV firmware. After problem have been noticed, it has been solved successfully.

The SD card memory task was considered as a small task in the planned work, but in the real implementation it is very important. Therefore, we assign on of the member responsible to this task. The SD started to implemented after the camera have been arrive and implemented correctly.

Because there is only one camera purchased, only one member of the team assign to keep the camera. There are problems with the cameras including the delay of deliveries, and camera faulty. The task leader of this has been assign to research on the progressive image on MATLAB in order to make a prototype presentation of a compression image taken. The task has been delayed from the time set to an individual. But after the second working camera arrived, the task has been done beautifully and there is enough time to combine with other tasks.

10.5.0.1 Schedule - (ms)

Compared to the planned Gantt chart, some tasks overran significantly (e.g. camera related tasks). The original Gantt chart was updated as we went along, and two of these Gantt charts are shown in appendix H. Keeping a Gantt chart was a very useful tool, because it allowed the group to compare their progress against their plan and therefore identify tasks which are overrunning and take appropriate action.

Figure 10.2 attempts to describe the basic order and integration of tasks with respect to time.

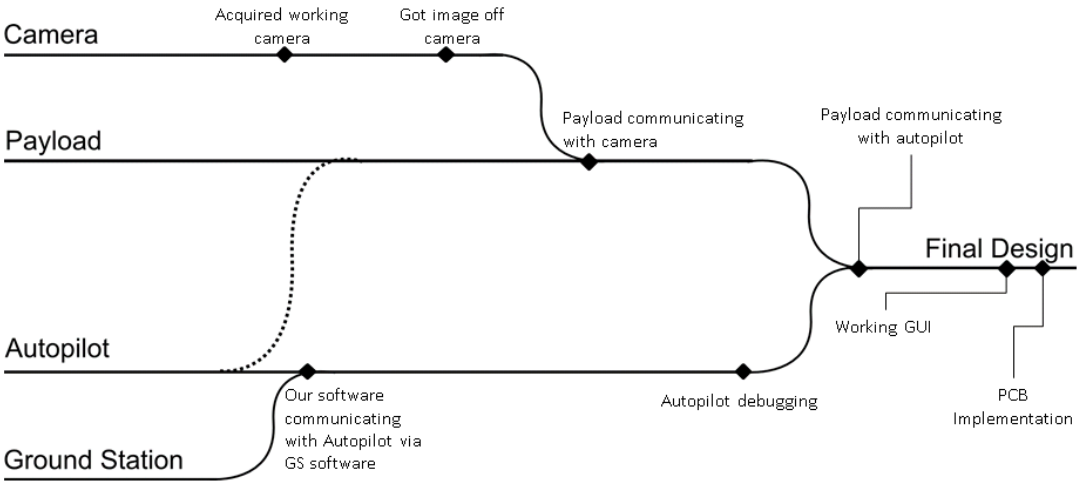


FIGURE 10.2: Diagrammatic explanation of the various streams of work and how they combined to produce a successful project.

10.6 Evaluation on Project Efficiency (ps)

The overall design of our project has been done very effectively. It fully satisfy the high and medium priorities that we has set it. The project has been done successfully and the outcomes was very pleased by the customer. However, the project has been changing throughout the time, especially in the hardware part. The changes on the structure of the payload communication has changed from the planned one. The SD card has been added to the payload in order to save the picture on board. It has been described in detail in section 6 The drawback of this is that we have to assign a task leader for this and so the other task might go slower. In a group, we have already decided that implementing the SD card will be the best use of our time and human resources. The microprocessor has changed as well as we describe on section6.4. Therefore, these changes make the whole project much better and more effective. Every changes in the design means there will be more time spending to recover the task that have been completed. Therefore, we could have done the project better if the task was planned at first so there will be no changes in the middle of the project. Progressive image downloading was the task that was about to implement and we have put many people to look at it. It has been proven that it is possible to implement by the MATLAB simulation (section 5.6) and also from a lot of research which the team member has been researching(section2.2. Although it seems like a very promising architecture of send the image, the time limitation make this progressive image

impossible to implement on the real device. In the middle of the project, the group meeting has conclude that the progressive image is taking too much of our human resource time and energy. Therefore, the task has been omitted and there are big reallocation of people to do other tasks. This might be a big mistake of the group because we have spent a lot of time doing this task.

10.7 Future work (jc and ab)

This section of the evaluation looks at potential additions to this project, which we would have done if time permitted. This starts with parts of the specification that we were not able to implement, and continues with some more advanced additions that we have considered.

10.7.1 Ground Station Software (ab)

The Ground Station Software which we have submitted remains rather buggy. There is a memory leakage issue which causes both our and SkyCircuits' software to crash after viewing a certain number of images, we have not had the time to resolve. This is not altogether critical, as the software is functional, and once re-started will re-connect to the autopilot. As the autopilot is autonomous, can take control of an aircraft from takeoff to landing, and is subservient to a manual controller, this is not a critical issue, but one that

It is also not feature-complete (see 9.1.3), something which could be resolved with some fairly trivial modifications to our code.

10.7.2 Progressive Image Manipulation (ms)

The progressive image display was not fully implemented by the end of the project. Below is a list of possible work which can be undertaken to complete the progressive image display system.

10.7.2.1 Progressive JPEG Overview (jc)

A progressive JPEG codec could be implemented on our system. The way we intended to implement a progressive JPEG codec is to first partially decode the JPEG image from the camera in order to gain access to the DCT frequency data of the image, from this point we could then send this in a non standard order, ideally so that the lowest frequency component of each MCU is send first followed by the higher frequency components so that the image on the ground station image viewer software would improve in quality in a similar manner to that seen in the code of appendix C.

10.7.2.2 JPEG Header Extractor (ms)

The information obtained by the JPEG header extractor has not been completely tested. The DQT information has yet to take into account multiple quantization tables

existing within a single DQT segment: only one quantization table is read. This problem can be fixed by preparing a linked list of quantization tables in a similar fashion to the DHT segment. 6.5.3.2

The AVR implementation will require testing with the ground station image displayer before it can make it into a final product.

10.7.2.3 Ground Station Software (ms)

Little work was done for a ground station progressive image displayer. An application would have to be coded which will read in the information provided by the extractor and display the image progressively. This means that the application would need to reverse the steps used to compress a raw image into JPEG given the coefficients of the Huffman tables and the quantization tables.

Attempting to reverse the custom compression of raw images MATLAB code using DCT can be a good starting point. If successful, reversing the quantization and Huffman encoding of the JPEG image would be required to obtain the information necessary to display the image.

10.7.3 Video Streaming (ab)

A rather interesting advancement of this project would be to try and implement real-time video streaming from the UAV. This would almost certainly have to use a dedicated wireless link. The 38.4 kBaud available in the autopilot modules is too restrictive for this sort of application, but the autopilot would still be able to control such an application (i.e. able to start/stop transmission, toggle power to the module, etc.) but things like minimising delay between video capture and video display, finding a suitably fast and robust wireless link (which implements automatic repeat requests) would make this task a very difficult one. It may be preferable to try and hack an existing wireless video module to one's own purposes (if such modules exist and are well documented)

10.7.4 Using Multiple Cameras (ab)

Another interesting application would be to mount multiple cameras (of the same model purchased for this project) on the UAV, and either take pictures from all the cameras at the same time, from some of the cameras at the same time, etc. The main issue here will be multiplexing the serial connection to all of the cameras, something considered for this project but ultimately not chosen.

10.7.5 Using Multiple Types of Camera (ab)

It would be useful to provide a range of camera modules to the customer, and an electronic module that would be able to communicate with all of them. This would include all the types of camera we considered (USB, serial and analogue), with varying resolutions. This would be rather difficult to implement, as the standards used across two types of the same camera interface can be rather different, so a driver would need to be written for each type of camera used. Also, the budget for such a project would be quite high.

10.7.6 SD card filesystem (jc)

It would be a nice additional bit of functionality if the sd card on the UAV could be accessed as if it were another file system on the computer running the ground station software. This would allow any of the photos stored on the UAV to be viewed easily and would also mean that the UAV could be used as an unusual form of data transfer, carrying files from one place to another.

This would also be useful if the payload was extended to contain other sensors that outputted files of a different type to the jpeg image from the camera.

10.7.7 HDR Photography (jc)

Image processing could be implemented in the image viewer software to enable high dynamic range (HDR) imaging. This could be useful for finding objects within a certain colour/intensity range by expanding the dynamic range over that region.

10.7.8 3D photography (jc)

If multiple cameras are implemented then these could be separated and the images from both combined to form 3 dimensional images, the vertical resolution of the 3d image would be proportional to the separation of the cameras.

An alternate method would be to use a similar method to synthetic aperture radar (SAR) and take one photograph, fly on a bit further and then take another. The speed and/or gps data from the UAV could then be used to work out the distance between the two photographs which could then be fed into the image processing algorithm in order to again generate a 3d image (presuming sufficient overlap within the images).

Appendix A

JPEG Segment Contents

A.1 SOF0: Start Of Frame (0xc0):

Table A.1: SOF0 Marker Content

Field	**Size**	**Comments**
Length	2 bytes	Equal to 8 + the number of components used in the colour scheme * 3
Data precision	1 byte	Value shown in bits/sample. Usually 8 signifying bytes.
Image height	2 bytes	—-
Image width	2 bytes	—-
Number of components	1 byte	indicates the number of components used in the colour scheme
Component information	3 bytes / component	See component information table below.

Table A.2: SOF Component Information

Byte Number	**Information**
1	Component ID (1 = Y, 2 = Cb, 3 = Cr, 4 = I, 5 = Q)
2	Sampling factors (bits 0...3 : vertical, bits 4...7 horizontal)
3	Quantization table number

A.2 DHT: Define Huffman Table(s) (0xc4):

Table A.3: DHT Marker Content

Field	**Size**	**Comments**
Length	2 bytes	—-
HT Information	1 byte	See HT Information table below
Number of Symbols	16 bytes	Number of symbols with codes of length 1...16. The sum of these values (n)must be ≤ 256
Symbols	n bytes	Symbols of the HT in order of increasing code length

Table A.4: HT Information Byte Content

Bit	**Information**
0...3	ID number of HT
4	type of HT, 0 = DC table, 1 = AC table
5...7	Unused

A.3 DQT: Define Quantization Table(s) (0xdb):

Table A.5: DQT Marker Content

Field	**Size**	**Comments**
Length	2 bytes	—-
QT Information	1 byte	See QT Information table below
Bytes	n bytes	QT values. n = 64 * (QT precision + 1)

Table A.6: QT Information Byte Content

Bit	**Information**
0...3	ID number of QT
4...7	precision of QT (0 = 8 bit, else 16 bit)

A.4 SOS: Start Of Scan (0xda):

Table A.7: SOS Marker Content

Field	**Size**	**Comments**
Length	2 bytes	Equal to 6 + 2 * (number of components in scan)
Number of components in scan	1 byte	Within range 1 to 4
Component information	2 bytes / component	See component information table below.
Ignorable Bytes	3 bytes	Separates header information from entropy-encoded image data

Table A.8: SOS Component Information Content

Byte Number	**Information**
1	Component ID (1 = Y, 2 = Cb, 3 = Cr, 4 = I, 5 = Q)
2	HT to use (bits 0...3 : AC Table ID, bits 4...7 DC Table ID)

Appendix B

Meeting Minutes

Below is a listing of minutes taken at all of our meetings in the Hartley Library.

B.1 05-10-11

```
Minutes- 05 Oct 2011


Topic of discussion
-Spec. of the GDP
-Gant Chart
-Assigning Task
-Project meeting
-Client's meeting

Spec of the GDP

        -Power- 5V & 3.3V ~100mA
        -Weight ~100g
        -design, test, capture and transmit an image from UAV to base station
        -camera module
        -transmit with low bandwidth
        -use autopilot datalink
        -camera<--> microcontroller<-->auto pilot <--> base station <--> software
        -(<-->=communicate 2 direction)
        -design a camera module to obtain image
        -construct final module on PCB
        -write a software to interface with the ground station over TCP/IP link
        -allow the software to receive and display image
        -camera module on PCB
        -make all schemetic and code accessible by the customers
        -test hardware on UAV
        -Readme file for the software

Task to do
        -acquire equipment
                -order (camera, parts)
                -wait
```

```
        -get picture from camera
        -encode(compression of image)/decode (decompression of image)
        -transmit image
        -get picture to the software
        -UI
        -background reading
        -testing
        -component selection
        -camera command (triggering, etc.)
        -TCP/IP
        -build & test (Bread board and PCB)
        -Integration
        -prepare presentation
        -make these communications work:camera<--> microcontroller<-->auto pilot
    <--> base station <--> software
        -report writing
```

B.2 06-10-11

```
List of questions:

* Which OS will be used?
* What software are we expected to use/are allowed to use/are not allowed to use/
    is recommended?
* Will the camera require an auto-focus feature?
* Is there an expected/existing camera set-up?
* Which cameras are suitable (will required prepared list)?
* Is there any documentation/specification available for the auto-pilot?
* Will the auto-pilot need to be modified in any significant way during the
    project?
* Will a form of blur reduction be required for the camera?
* What does the 38.4 kbs transfer rate refer to?
* Will it be necessary to build software into any pre-existing hardware, or will
    the code be stand-alone?
* What does the "ground station" refer to?
* How will the camera be powered?
* How robust and reliable would the communications link be?
* Should the camera be in black and white or colour?
* Will the camera need a prompt, or will it take images at certain intervals?
* Are there weight limitations that must be respected?
* When do we receive the module for testing?
* Which image format is acceptable? .jpg?
* Will we get to play with a UAV?
* Will the camera need to be waterproof?
* Any requirements for lens? (Pinhole etc?)
* Does it need to have any specific output format (e.g. digital)?
* Do we have to use the autopilot or can we build a standalone system?

Skills audit:

* Andrew: Power systems, PCB construction, control, MATLAB
* Mitch: Image Processing, MATLAB, ASM, reports.
* John: MATLAB, Image processing, control, radio-transmissions
* Peak: Digital control, display, MATLAB, bitrate tranfer
* Micheal: C++, Python, C#, java, microprocessors, systems interfacing, getting
    images from a camera
```

```
Camera list recommendations:
* UK based retailer
* variety of resolutions and features to choose from
* Must have proof of proper functioning (reviews)
* up to  100
```

B.3 11-10-11

```
Location: Library, Study Room 1A
Date/Time: 11/10/2011, 1600-1730
Agenda: * Review progress made, and assign tasks for the week ahead
In Attendance: Mitch, Andy, Peak, John, Michael

-----
Work Done:
* Obtained a Camera
* Basic MATLAB compression code has been achieved
* Work on getting some useful data out of the camera has started (using Arduino,
    in progress)

-----
Work for the coming week (12/10 - 19/10)
* Get image off the camera (need a memory buffer, multiple resolutions) [John,
    Michael]
* Parse image data from the camera [John, Michael]
* Order remaining parts for prototyping autopilot peripheral from Farnell on
    12/10 am [John, Michael]
* Construct a prototype autopilot peripheral, based on client's schematics [Andy,
     Peak]
* Play around with DSPs given to us by stores/supervisor, run an FFT on them (-
    Progressive JPEG?) [Andy, Peak]
* MATLAB Compression - Image Specific, and general [Mitch, Michael, John]
* Payload Controller - Ground Station Protocol (i.e. sending commands to a
    payload, receiving data back from it) (SCPI?) [Andy, Peak]

-----
Targets for 19/10
* AVR (or any other uProcessor) code to get an image from the webcam onto a
    computer
* Code to display image from the webcam (raw data, and jpeg_.
* Obtain all parts for prototyping
* Prototype for the payload on breadboard, with schematic
* DSP selected > FFT function achieved on DSP
* 50% image specific compressing MATLAB code
* 50% lossless compression MATLAB code
```

B.4 18-10-11

```
- mitch has looked up lots on image compression, has pseudocode, will try to look
     at doing it in C, will also write up different compression algorithms
```

```
- peak has some working code for communicating with the autopilot

- we now have components

What we have done:
-----------------
- UAV responding: data extracted
- can recieve data from UAV

- compression + decompression:
    - algorithm pseudo code
- components recieved
-Matlab starter code

Next tasks:
---------------
- Write up theory.
- Implement compression algos in C (encoding) and decoding in any language -
    mitch, peak
    - ensure encoding works on micro-controller

- Basic construction of payload
- Interactionn with payload
- Get any camera working
```

B.5 25-10-11

```
24/25-10-11, Advanced Electronics Lab, Zepler Building Level 3

Agenda: Presentation finalisation

Agreed that this presentation needs to focus on what we plan to do, and project
    management should feature heavily. As well as humour. Tomorrow afternoon is
    going to be rather dull.

Presentation Structure:
* Introduction and problem description (Michael)
* Specification, decreasing order of priority (John)
* Management: Skill Audit, Risk Management, Contingencies (Mitch)
* Breakdown of tasks (Gantt Chart), Progress made (Andy, Peak)
* MATLAB video of what's currently working (John)
* Summary (Michael)

Now we all need to go away, revise and rehearse.

Spoke with Nic Green on 23-10, Mahesan Niranjan on 24-10 for presentation
    feedback.

Nic:
* "Boy Band" theory - make speaker changes as seamless as possible
* Insert slides on risk assessment and management
* "Speaker Notes": What we see on the screen are our notes, not our presentation.
* Inject humour where possible - even the risk management
* Rehearse

Mahesan:
```

```
* Record ourselves - there are some mannerisms we need to iron out.
* We overran slightly, need to stop waffling, umming and ahing.
* Don't rely on notes - looks like we're reading.
* Different techniques work for different people. Go away and play.

=====
26/10/11, After Presentation:

Seemed to go well. Matt couldn't come as he was learning to fly in France. Can't
    really blame him. Speaking to Rob afterwards, we aren't currently doing
    anything he wouldn't, feedback seemed to be positive but we could've
    mentioned our JPEG image compression research in a bit more detail (DCT,
    wavelets, etc.) Seminar no. 2 needs to be very much more technical.
```

B.6 01-11-11

```
01-11-11, 4pm-5pm, Hartley Library Room 1A

Agenda: Start setting some deadlines, thrash out some ideas for the final product
    .

Deadline 1: Feature Freeze ("bells + whistles" features) - Next Wednesday (09-11)
Deadline 2: PCB Production Deadline - Next Friday (11-11)
^ This allows two weeks for delivery/testing of the boards before the
    presentations

Possibilities for deliverables:
- Arduino + daughter board. Daughter board contains SD card, ethernet socket,
    transceiver module, level converter, etc. (Swish, easy to get a hold of,
    ATmega168 and 328P compatible, but not much memory)
- If arduino not powerful enough, port to Olimexino (built in microSD! Cortex M3,
     not AVR). Language is basically the same, with the Maple IDE libraries.
    Potential supply issues with the Olimex - coolcomponents is continually
    running out.
- AVR on Stripboard?

=======
02-11-11, Rob Maunder's Office.

- We now have "Il Matto", a playing-card sized, arduino-inspired dev board made
    for last year's D4 lab. Contact Steve Gunn for more info.
- Based on an ATMega644
- No charging circuit has actually been implemented on it (there is some
    breadboard space and a schematic/PCB layout saying what needs to go where
- Project plan, reaction to circumstances seems to be a good 'un
- We ought to start putting effort into the report.
- LaTeX, Google Docs. (Former is nicer for version control)
```

B.7 14-11-11

```
14 Nov 2011 2:30:
```

```
- andy not here

- possible that transmission fast enough not to require custom compression
- drop down custom compression priority
        - have one person working on the custom compression stuff
        - mitch will be working on this
        - mitch will assume we get 64 bytes at a time from the camera to the
    custom compression code
        - 'c read binary file'
        - goals by next monday: 21/11/2011
                - extract relevant jpeg header info (by tuesday evening)
                - use this in the huffman coder (talk to michael)
                - rest we will worry about later
- john will try to add bus/tristate buffer for the tx rx on arduino to allow
    camera and
payload to communicate
- peak will work on getting payload to work
- michael will also work on getting payload to work
- when finished payload peak will work on ground station UI
- andy working on payload code for the ardunio
- andy assigned to pcb production (should talk more about this tomorrow)

essential tasks remaining:
        - payload to autopilot communication (all on one chip)
        - ground station software and UI
```

B.8 15-11-11

```
15-11-2011, 4pm-5.30pm, Hartley Library Room 1B

Agenda: Beat together a report structure, review any further progress made.

Taking some inspiration from the example LaTeX GDP report on Steve Gunn's webpage
     (NB: This is highly subject to change / re-ordering):

=====
* Introduction (mentioning that it's also for proof "payload" of concept)
* Background Research
    - Image Compression: JPEG Structure / Huffman Coding
    - Choice of Camera / Payload Documentation / GCS.exe
    - Hardware Selection: Payload Controller (AVR/Arduino/Olimex/etc.) /
    Programming Language (C/C#)
* Specification
    - Proposal (Handed in first week)
    - Consideration of options
* Development [1]
    - Research, possible solutions, design approaches
* Hardware [1]
* Software [1]
* Overview
    - Final Deliverables
* Project Management
    - Gantt Charts (more than just first one)
    - Risk Management (from presentation 1, also risks encountered)
    - Work Allocation (planned, actual)
    - Meetings (minutes in annexe?)
```

```
* Future Work [2]
    - (Real-time) video, (higher-res) other cameras, better GUI, progressive scan
    , radar, additional setting choices
* Evaluation [2]
* Conclusions
* Documentation
    - User Manual
    - Technical Docs
    - Source Code (github/sourceforge/somewhere public) (GPL'd)
    - BOM (Bill of Materials)
* APPENDICES
    - BOM
    - Gantt
    - Source Code
    - Presentations (links to, not all the slides)
    - Minutes
    - SVN / git log (github charts).
=====

[1] Think it may be nicer to combine this into "Technical Content" section. (or
    something slightly better worded):
    - Camera <-> Payload Controller (SD Card)
    - Payload Controller (SD Card) <-> Autopilot
    - (Autopilot <->) Ground Station Software
[2] These two sections could probably be combined. (Evaluation containing what
    future work could be done)

PROGRESS MADE: Image from camera to SD card. Complete. Including a file numbering
     situation. Some quirks that need to be ironed out, though (power cycling of
    camera, removal of camera RX/TX to program arduino) . (svn r97, git 04e4d70b,
     10/11/11) JOY!

TO DO: Docs, Payload link (Matt Bennett has been emailed), Progressive JPG, User
    Interface. See yesterday's minutes for who's been re-assigned to what.
```

Appendix C

Code Listings

C.1 MATLAB code for custom image compression

Compression1.m

```
clear all
close all

% load clown
load mandrill

X = (X-min(min(X)))./max(max(X-min(min(X))));

%% fft

freqX = fft2(X);

testFreqX = zeros(size(freqX));

shiftedFreqX = fftshift(freqX);

centre = size(freqX)/2;
figure

for z = 10:-0.5:1.5
    squarewidth = round(centre(1)/z);
    squareheight = round(centre(2)/z);
    % squarewidth = 10;
    % squareheight = 16;
    testFreqX = zeros(size(freqX));

    for x = -squarewidth:squarewidth;
        for y = -squareheight:squareheight;
            testFreqX(centre(1)+x,centre(2)+y) = shiftedFreqX(centre(1)+x,centre(2)+y);
        end
    end

    testFreqX = fftshift(testFreqX);

    testX = abs(ifft2(testFreqX));
```

```
37      imagesc(testX)
        colormap gray
39      drawnow
        pause(0.1)
41  end

43  %% dct

45  cfreqX = dct2(X);
    bigness = size(cfreqX);
47
    % squarewidth = 10;
49  % squareheight = 16;
    for z = 10:-0.5:1.5
51
        ctestFreqX = zeros(size(cfreqX));
53
        squarewidth = round(bigness(1)/z);
55      squareheight = round(bigness(2)/z);

57      for x = 1:squarewidth;
            for y = 1:squareheight;
59              ctestFreqX(x,y) = cfreqX(x,y);
            end
61      end

63      testX = abs(idct2(ctestFreqX));

65      imagesc(testX)
        colormap gray
67      drawnow
        pause(0.1)
69  end
```

Listing C.1: Whole image, transform based compression code

chunkCompression.m

```
1   clear all
    close all
3
    % load clown
5   load mandrill

7   X = (X-min(min(X)))./max(max(X-min(min(X))));

9
    %% fft
11
    freqX = fft(fft(X)')';
13
    testFreqX = zeros(size(freqX));
15
    shiftedFreqX = fftshift(freqX);
17
    centre = size(freqX)/2;
19  figure
```

```
for z = 10:-0.5:1.5
    squarewidth = round(centre(1)/z);
    squareheight = round(centre(2)/z);
    % squarewidth = 10;
    % squareheight = 16;
    testFreqX = zeros(size(freqX));

    for x = -squarewidth:squarewidth;
        for y = -squareheight:squareheight;
            testFreqX(centre(1)+x,centre(2)+y) = shiftedFreqX(centre(1)+x,centre(2)+y);
        end
    end

    testFreqX = fftshift(testFreqX);

    %     testX = abs(ifft2(testFreqX));
    testX = abs(ifft(ifft(testFreqX)')');

    imagesc(testX)
    colormap gray
    drawnow
    pause(0.1)
end

%% enchunk

chunksize = 20; % must be even
imagesize = size(X);

% X2 = X;
% X = X(1:480,1:480);

for x = 1:round(imagesize(1)/chunksize)
    for y = 1:round(imagesize(2)/chunksize)
        chunk(x,y).data = X((((x-1)*chunksize)+1):(x*chunksize), (((y-1)*chunksize)+1):(y*chunks
    end
end

%% chunks fft

for z = 0.5:(chunksize/2)-0.5

    for chunkcount = 1:round(imagesize(1)/chunksize)*round(imagesize(2)/chunksize)

        chunk(chunkcount).freqX = fft(fft(chunk(chunkcount).data)')';

        chunk(chunkcount).testFreqX = zeros(size(chunk(chunkcount).freqX));

        chunk(chunkcount).shiftedFreqX = fftshift(chunk(chunkcount).freqX);

        centre = (chunksize/2)+0.5;

        for x = centre-z:centre+z;
            for y = centre-z:centre+z;
                chunk(chunkcount).testFreqX(x,y) = chunk(chunkcount).shiftedFreqX(x,y);
            end
        end

        chunk(chunkcount).testFreqX = fftshift(chunk(chunkcount).testFreqX);
```

```
81          %     testX = abs(ifft2(testFreqX));
            chunk(chunkcount).testX = abs(ifft(ifft(chunk(chunkcount).testFreqX)')');
83
        end
85
        for x = 1:round(imagesize(1)/chunksize)
87          for y = 1:round(imagesize(1)/chunksize)
                ReconTestX((((x-1)*chunksize)+1):(x*chunksize), (((y-1)*chunksize)+1):(y*chunksize))
89          end
        end
91
        figure
93
        imagesc(ReconTestX)
95      colormap gray
        drawnow
97      pause(0.5)
    end
```

Listing C.2: Secioned image, transform based compression code

C.2 Arduino code

```
    // Camera -> SD Card Code
 2
    // PIN LAYOUT:
 4  // Digital 13  <->  SD CLK
    // Digital 12  <->  SD MISO / Data 0 / SO
 6  // Digital 11  <->  SD MOSI / CMD / DI
    // Digital 4   <->  SD D3
 8  // Arduino RX  <->  Camera TX
    // Arduino TX  <->  Camera RX
10
    #include <SoftwareSerial.h>
12  #include <SD.h>
    #include <stdio.h>
14
    #define DLOG(...)  sSerial.print(__VA_ARGS__)
16  //#define DLOG(...)
    byte SYNC[] = {0xAA,0x0D,0x00,0x00,0x00,0x00};
18  byte RXtest[6];

20  // Serial PC cable to view error messages
    #define sRxPin 2
22  #define sTxPin 3

24  SoftwareSerial sSerial = SoftwareSerial(sRxPin, sTxPin);
    char filePrefix[] = "test"; //Prefix of file name written to the SD card
26  char fileExt[] = ".jpg"; //File extension of the file being written to
    char fileName[13]; //Contains full file name of file being written to
28  char numBuf[5]; //Number buffer, for the "find a file that doesn't exist" procedure. Could be a
    File jpgFile;
30
    void setup()
32  {
```

```
    // start serial port at 115200 bps
34  Serial.begin(115200);
    sSerial.begin(9600);
36
    // start software serial library for debugging
38  pinMode(sRxPin, INPUT);
    pinMode(sTxPin, OUTPUT);
40  // Pin 10 = Chip Select
    pinMode(10, OUTPUT);
42
    // If SD not connected, stop execution
44  if (!SD.begin(4)) {
      DLOG("SD Failed");
46    return;
    }
48  DLOG("SD Working");

50  //DLOG("Attemting to establish contact.\n\r");
    establishContact();  // send a byte to establish contact until receiver responds
52  //DLOG("Contact established, captain\n\r");
    delay(2000); // 2 second pause
54

56 }

58 void loop()
   {
60   sSerial.read(); //Wait for something to be sent over the PC serial line

62   boolean fileExists = true;
     int fileNum = 0;
64   while(fileExists){
       fileNum++;
66     snprintf(fileName, sizeof(fileName), "%s%i%s", filePrefix, fileNum, fileExt); // prints "tes
       fileExists = SD.exists(fileName); // checks whether that file is on the SD card.
68   }

70   sSerial.println(fileName);
     setupSD(fileName); // creates the file on the sd for writing
72
     boolean wellness = takeSnapshot(); //Take a snapshot
74   if(wellness){ //If no error is detected
      DLOG("All's well\n\r");
76    delay(2000);
     }else{ //Error has been detected
78     DLOG("Trouble at mill...\n\r");
     }
80   jpgFile.close(); // Close JPG file
   }
82
   // setup the file on the sd card
84
   void setupSD(char fileName[]){
86   // For Optimisation of the SD card writing process.
     // See http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1293975555
88   jpgFile = SD.open(fileName, O_CREAT | O_WRITE);
   }
90
   // synchronise with camera
```

```
92 void establishContact() {
     DLOG("Sending syncs\n\r\r"); // Send SYNC to the camera.
94   //This could be up to 60 times
     while(1){
96     while (Serial.available() <= 0) {
         DLOG(".");
98       sendSYNC();
         delay(50);
100    }

102    receiveComd();
    // Verify that the camera has sent back an ACK followed by SYNC
104    if(!isACK(RXtest,0x0D,0x00,0x00))
         continue;
106    DLOG("ACK received\n\r");
       receiveComd();
108
      if(!isSYNC(RXtest))
110     continue;

112    DLOG("SYNC received\n\r");
       // Send back an ACK
114    sendACK(0x0D,0x00,0x00,0x00);

116    break;
     }
118 }

120 // takes a snapshot
    boolean takeSnapshot() {
122
     // setup image parameters
124  DLOG("sending initial\n\r");
     delay(50);
126  sendINITIAL(0x07,0x07,0x07);
     receiveComd();
128  if(!isACK(RXtest,0x01,0x00,0x00))
         return false;
130  DLOG("ACK received\n\r");

132  // Package size = maximum of 512 bytes
     // Arduino serial.read() buffer size = 128 bytes
134  unsigned int packageSize = 64;

136  // sets the size of the packets to be sent from the camera
     DLOG("sending setpackagesize\n\r");
138  delay(50);
     sendSETPACKAGESIZE((byte)packageSize,(byte)(packageSize>>8));
140  receiveComd();
     if(!isACK(RXtest,0x06,0x00,0x00))
142      return false;
     DLOG("ACK received\r");
144
     // camera stores a single frame in its buffer
146  DLOG("sending snapshot\n\r");
     delay(50);
148  sendSNAPSHOT(0x00,0x00,0x00);
     receiveComd();
150  if(!isACK(RXtest,0x05,0x00,0x00))
```

```
        return false;
152   DLOG("ACK received\n\r");

154   // requests the image from the camera
      DLOG("sending getpicture\n\r");
156   delay(50);
      sendGETPICTURE(0x01);
158   receiveComd();
      if(!isACK(RXtest,0x04,0x00,0x00))
160       return false;
      DLOG("ACK received\n\r");
162   receiveComd();
      if(!isDATA(RXtest))
164       return false;
      DLOG("DATA received\n\r");
166
      // Strips out the "number of packages" from the DATA command
168   // then displays this over the serial link
      unsigned long bufferSize = 0;
170   bufferSize = RXtest[5] | bufferSize;
      bufferSize = (bufferSize<<8) | RXtest[4];
172   bufferSize = (bufferSize<<8) | RXtest[3];

174   unsigned int numPackages = bufferSize/(packageSize-6);

176   DLOG("Number of packages: ");
      DLOG(numPackages,DEC);
178   sSerial.println();

180   byte dataIn[packageSize];
      int flushCount = 0;
182
      for(unsigned int package = 0;package<numPackages;package++){
184
        //DLOG("Sending ACK for package ");
186     //DLOG(package,DEC);
        //DLOG("\n\r");
188     sendACK(0x00,0x00,(byte)package,(byte)(package>>8));

190       // receive package
          for(int dataPoint = 0; dataPoint<packageSize;dataPoint++){
192         while (Serial.available() <= 0) {} // wait for data to be available n.b. will wait forev
            dataIn[dataPoint] = Serial.read();
194           if(dataPoint > 3 && dataPoint < (packageSize - 2)){ //strips out header data
              //sSerial.print(dataIn[dataPoint],BYTE);
196           jpgFile.print(dataIn[dataPoint]);
                if(flushCount == 511){
198               jpgFile.flush();
                  flushCount = -1; // because it adds one straight away after
200             }
              flushCount++;
202           }
            //DLOG(dataPoint);
204         //DLOG("\n\r");
          }
206     //DLOG("Package ");
        //DLOG(package);
208     //DLOG(" read successfully\n\r");
```

```
210   }

212   sendACK(0x0A,0x00,0xF0,0xF0);
      DLOG("Final ACK sent\n\r");
214   return true;
    }
216
    // The following are all self-explanatory.
218 // Follows command structure described in the camera's datasheet

220 void sendSYNC() {
      sendCommand(0xAA,0x0D,0x00,0x00,0x00,0x00);
222 }

224 void sendACK(byte comID, byte ACKcounter, byte pakID1, byte pakID2) {
     sendCommand(0xAA,0x0E,comID,ACKcounter,pakID1,pakID2);
226 }

228 void sendINITIAL(byte colourType, byte rawRes, byte jpegRes) {
     sendCommand(0xAA,0x01,0x00,colourType,rawRes,jpegRes);
230 }

232 void sendSETPACKAGESIZE(byte paklb, byte pakhb) {
     sendCommand(0xAA,0x06,0x08,paklb,pakhb,0x00);
234 }

236 void sendSNAPSHOT(byte snapType, byte skip1, byte skip2) {
     sendCommand(0xAA,0x05,snapType,skip1,skip2,0x00);
238 }

240 void sendGETPICTURE(byte picType) {
     sendCommand(0xAA,0x04,picType,0x00,0x00,0x00);
242 }

244 // sends commands to the camera
    void sendCommand(byte ID1, byte ID2, byte par1, byte par2, byte par3, byte par4) {
246   byte Command[] = {ID1,ID2,par1,par2,par3,par4};
      Serial.write(Command, 6);
248 }

250 // receives messages from the camera
    void receiveComd() {
252   for(int z = 0;z<6;z++){
        while (Serial.available() <= 0) {}
254     RXtest[z] = Serial.read();
      }
256 }

258 // verifies ACK
    boolean isACK(byte byteundertest[],byte comID, byte pakID1, byte pakID2){
260   byte testACK[] = {0xAA,0x0E,comID,0x00,pakID1,pakID2};
      for(int z = 0;z<6;z++){
262     if((z != 3) && (byteundertest[z] != testACK[z]))
          return false;
264   }
      return true;
266 }

268 // verifies SYNC
```

```
boolean isSYNC(byte byteundertest[]){
  for(int z = 0;z<6;z++){
    if(byteundertest[z] != SYNC[z])
      return false;
  }
  return true;
}

// verifies DATA
boolean isDATA(byte byteundertest[]){
    if((byteundertest[0] != 0xAA) || (byteundertest[1] != 0x0A)){
      return false; }
  return true;
}
```

Listing C.3: Arduino code, used up until the point we started using the Il Matto. This was written in the arduino-022 IDE

C.3 Waypoint Triggering Autopilot Script

```
script.wait_while_false "logic.less_than_float32 wp[0].distance_from_uav 200"
payload[0].send_bytes 0
script.wait_ms 10000
script.position 0
```

Listing C.4: Waypoint triggering code written for the SkyCircuits autopilot. The send_bytes command is sent when the payload is within 200 meters of waypoint 0. The code will then wait 10 seconds before running again.

Appendix D

Image Viewer Code

D.1 Main Form code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.Threading;
using System.Net.Sockets;

namespace NCamGS
{
    public partial class MainForm : Form
    {

        private static int uavDataLength = 512;
        //Connector streamPort = new Connector();


        UAVConnector uavConn = new UAVConnector();



        byte[] uavDataPrevious = new byte[uavDataLength];
        private const byte
            TAKE_PICTURE = 0x00,// send to console port payload[0].sendbytes 0
            PICTURE_TAKEN = 0x01,//wait for picture taken (from data port) so do payload[0].memb
            // it will be [ID][MSB][LSB]
            SEND_DOWNLOAD_REQUEST = 0x02, // send to console port send download request
            //[ID][MSB][LSB]
            DOWNLOAD_INFO_COMMAND = 0x03, // from cam to ground
            //[3][MSB][LSB]
            TAKEN_IMAGE_DATA = 0x04;// image data
```

```
38

40          int filePathCount = 0;
            string[] jpegList;
42          int jpegOnlyCount = 0;
            bool stopCommand = false,takeNewPicture=false;
44

46          string fileDirectory = null;

48
            public MainForm()
50          {
                InitializeComponent();
52              MyInitialize();
            }
54          private void MyInitialize()
            {
56
                uavConn.Connect(0);
58              progressBar.Minimum = 1;
                progressBar.Step = 1;
60              progressBar.Value = 1;
                fileDirectory= System.IO.Directory.GetCurrentDirectory();
62              filePathTextBox.Text = System.IO.Directory.GetCurrentDirectory();
                jpegList = Directory.GetFiles(fileDirectory, "*.jpg");
64              uavConn.SendTextToUAV("payload.broadcast_bytes 255 0");
                ResolutionComboBox.SelectedIndex = 3;
66
            }
68

70          private void paintButton_Click(object sender, EventArgs e)
            {
72              takeNewPicture = true;
                doCommand();
74
            }
76          private void doCommand()
            {
78              statusLabel.Text = "Starting";
                progressBar.Value = 1;
80              string fileName = string.Format
                ("uavPictureAt{0:yyyy-MM-dd_hh-mm-ss-tt}.jpg", DateTime.Now);
82              FileStream fileStream;

84                  fileStream = new FileStream(filePathTextBox.Text+
                    "\\"+fileName, FileMode.Create);
86              BinaryWriter opFile = new BinaryWriter(fileStream);

88

                uavConn.SendTextToUAV("da 20 payload[0].mem_bytes[0]");
90
                stopCommand = false;
92              byte[] zeroToken = { 0 }; // send 0 to receive data
                uavConn.SendCommand(zeroToken);
94
                uint imageID = 0;
96
```

```
            while (stopCommand == false)
98          {
                Application.DoEvents();
100             Console.Write(".");
                byte[] packet = uavConn.GetDataBytes();
102             int packetSize = packet.Length;
                if (packetSize > 0)
104             {
                    if (packet[0] == 1 && packetSize == 3)
106                 {
                        // got PICTURE_TAKEN
108                     statusLabel.Text = "Found PICTURE_TAKEN";
                        Console.WriteLine("Found PICTURE_TAKEN");
110                     imageID = (uint)packet[1] + (uint)(packet[2] << 8);
                        break;
112                 }
                }
114         }

116         byte[] imageDownloadRequest = { 2, (byte)imageID, (byte)(imageID >> 8) };
            uavConn.SendCommand(imageDownloadRequest);
118
            uint lastPacketNum = 500;
120
            uint totalPackets = 500;
122
            bool startImage = false;
124         long numBytes = 0;
            while (stopCommand==false)
126         {
                Application.DoEvents();
128             // add cancel code check

130             Console.Write(".");
                byte[] packet = uavConn.GetDataBytes();
132             int packetSize = packet.Length;
                //progressbar maximum
134
                if (packet[0] == 3) // is a IMAGE_DOWNLOAD_INFO packet
136             {
                    statusLabel.Text = "Found IMAGE_DOWNLOAD_INFO";
138                 Console.WriteLine("Found IMAGE_DOWNLOAD_INFO");
                    totalPackets = (uint)packet[1] + (uint)(packet[2] << 8);
140                 progressBar.Maximum = (int)totalPackets;
                }
142             else if (packet[0] == 4) // is a IMAGE_DATA packet
                {
144                 statusLabel.Text = "Found IMAGE_DATA";
                    Console.WriteLine("Found IMAGE_DATA");
146                 uint packetNum = (uint)packet[1] + (uint)(packet[2] << 8);

148                 Console.Write("#" + packetNum + " ");

150                 for (int i = 3; i < packetSize; i++)
                    {
152                     Console.Write(packet[i] + " ");
                    }
154                 Console.WriteLine();
```

```
156
158                 if (packetNum == 0)
                        startImage = true;
160
                    if (startImage == false)
162                     continue;

164                 if (packetNum == lastPacketNum)
                    {
166                     //Console.WriteLine("End of image.");
                        continue;
168                 }
                    statusLabel.Text = "Writing";
170                 Console.WriteLine("Writing.");
                    for (int i = 3; i < packetSize; i++)
172                 {
                        opFile.Write(packet[i]);
174                     numBytes++;
                    }
176                 progressBar.PerformStep();
                    if (packetNum == totalPackets - 1)
178                     break;
                    lastPacketNum = packetNum;
180
                }
182         }
            if (stopCommand == false)
184         {
                fileStream.Close();
186             opFile.Close();
                progressBar.Value = 1;
188
                statusLabel.Text = "Done!";
190             Console.WriteLine("Done!");

192

194

196             updateInitialDirectory(fileName);

198             Array.Resize(ref jpegList, jpegList.Length + 1);
                jpegList[jpegList.Length - 1] = fileDirectory + "\\" + fileName;
200             jpegList = Directory.GetFiles(fileDirectory, "*.jpg");

202
                Image img = Image.FromFile(fileDirectory + "\\" + fileName);
204             pictureBox.Image = img;
                statusLabel.Text = null;
206         }
            if (stopCommand == true)
208         {

210             fileStream.Close();
                opFile.Close();
212             File.Delete(filePathTextBox.Text + "\\" + fileName);
                statusLabel.Text = "Stop";
214         }
```

```
216         }
            private void receivingCase(byte[] receivedData)
218         {
                byte command = 0;
220             switch(command)
                {
222                 case PICTURE_TAKEN:
                    break;
224                 case DOWNLOAD_INFO_COMMAND:
                    break;
226                 case TAKEN_IMAGE_DATA:
                    break;
228             }
            }
230         private void sendingCase()
            {
232             byte command=0;
            switch(command)
234         {
                case TAKE_PICTURE:
236
                    break;
238             case SEND_DOWNLOAD_REQUEST:

240                 break;

242         }
            }
244
            private void MainForm_Load(object sender, EventArgs e)
246         {

248         }

250         private void updateInitialDirectory(string fileName)
            {
252             fileDirectory =filePathTextBox.Text;
                fileName = fileDirectory +"\\"+ fileName;
254             jpegList = Directory.GetFiles(fileDirectory, "*.jpg");
                if (takeNewPicture == true)
256             {
                    Array.Resize(ref jpegList, jpegList.Length + 1);
258                 jpegList[jpegList.Length - 1] = fileName;
                    takeNewPicture = false;
260                 filePathCount = jpegList.Length;
                }
262             jpegOnlyCount = jpegList.Length;
                if(jpegOnlyCount!=0)
264             {
                    for (filePathCount = 0;
266                 jpegList[filePathCount] != fileName;
                     filePathCount++)
268                 {

270                 }
                }
272             if (jpegOnlyCount == 0)
                {
```

```
274                 rightButton.Hide();
                    leftButton.Hide();
276                 pictureBox.Image = null;
                }
278             else
                {
280                 rightButton.Show();
                    leftButton.Show();
282             }
            }
284         private void updateDirectory(string fileName)
            {
286             bool fileNameIsDirectory = false;
                if (fileName.Substring(fileName.Length - 4, 4).Contains("."))
288             {
                    fileDirectory = fileName.Substring(0, fileName.LastIndexOf("\\"));
290                 fileNameIsDirectory = false;

292             }
                else
294             {
                    fileDirectory = fileName;
296                 fileNameIsDirectory = true;
                }
298             filePathTextBox.Text = fileDirectory;
                try
300             {
                    jpegList = Directory.GetFiles(fileDirectory, "*.jpg");
302             }
                catch
304             {
                }
306             jpegOnlyCount = jpegList.Length;
                if (jpegOnlyCount != 0)
308             {
                    if (fileNameIsDirectory == false &&
310                 ( fileName.Substring(fileName.Length - 4, 4).Contains(".jpg") ||
                    fileName.Substring(fileName.Length - 4, 4).Contains(".JPG")))
312                 {
                        try
314                     {
                            for (filePathCount = 0; jpegList[filePathCount] != fileName; filePathCou
316                         {

318                         }
                        }
320                     catch
                        {
322                         filePathCount--;
                        }
324                 }
                    else
326                 {
                        filePathCount = 0;
328                 }
                }
330             if (jpegOnlyCount == 0)
                {
332                 rightButton.Hide();
```

```
                leftButton.Hide();
334             pictureBox.Image = null;
            }
336         else
            {
338             try
                {
340                 pictureBox.Image = Image.FromFile(jpegList[filePathCount]);

342                 pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
                }
344             catch
                {
346             //do something
                }
348             rightButton.Show();
                leftButton.Show();
350         }
        }
352     private void mnuOpen_Click(object sender, EventArgs e)
        {
354         OpenFileDialog fileOpen = new OpenFileDialog();

356

            fileOpen.Title = "Select file to open:";
358         fileOpen.Filter ="(*.JPG)|*.JPG;|All files (*.*)|*.*";

360

            if (fileOpen.ShowDialog() == DialogResult.OK)
362         {
                updateDirectory(fileOpen.FileName);
364             if (jpegList.Length != 0)
                {
366

                    pictureBox.Image = Image.FromFile(jpegList[filePathCount]);
368             }
                else
370             {
                    pictureBox.Image = Image.FromFile(jpegList[0]);
372             }
            }
374         pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
            fileOpen.Dispose();
376     }

378     private void commandRecTimer_Tick(object sender, EventArgs e)
        {
380

        }
382
        private void leftButton_Click(object sender, EventArgs e)
384     {
            if (filePathCount == 0)
386         {
                leftButton.Hide();
388             try
                {
390                 pictureBox.Image = Image.FromFile(jpegList[filePathCount]);
                }
```

```
392                 catch
                    {
394                     leftButton.Hide();
                    }
396             }
                else
398             {
                    rightButton.Show();
400                 try
                    {
402                     pictureBox.Image = Image.FromFile(jpegList[--filePathCount]);
                    }
404                 catch (OutOfMemoryException)
                    {
406                     DialogResult result1 = MessageBox.Show("Incomplete JpegFile founded.
                        Do you want to delete it?"
408                                                 , "Incomplete JpegFile founded!",
                                                     MessageBoxButtons.YesNo);
410                     if (result1 == DialogResult.Yes)
                        {
412                         File.Delete(jpegList[filePathCount]);
                            updateDirectory(jpegList[filePathCount]);
414                         pictureBox.Image = Image.FromFile(jpegList[filePathCount--]);
                        }
416
                    }
418             }
            }
420
            private void rightButton_Click(object sender, EventArgs e)
422         {
                if (filePathCount < jpegOnlyCount - 1)
424             {
                    leftButton.Show();
426                 try
                    {
428                     pictureBox.Image = Image.FromFile(jpegList[++filePathCount]);
                    }
430                 catch(OutOfMemoryException)
                    {
432                     DialogResult result1 = MessageBox.Show
                        ("Incomplete JpegFile founded. Do you want to delete it?"
434                                                             ,"Incomplete JpegFile founded!",
                                                              MessageBoxButtons.YesNo);
436                     if (result1 == DialogResult.Yes)
                        {
438                         File.Delete(jpegList[filePathCount]);
                            updateDirectory(jpegList[filePathCount]);
440                         pictureBox.Image = Image.FromFile(jpegList[filePathCount--]);
                        }
442


444
                    }
446             }
                else rightButton.Hide();
448         }

450
```

```
452         private void filePathButton_Click(object sender, EventArgs e)
            {
454
                DialogResult result = folderBrowserDialog1.ShowDialog();
456             if (result == DialogResult.OK)
                {
458                 filePathTextBox.Text = folderBrowserDialog1.SelectedPath;
                    updateDirectory(folderBrowserDialog1.SelectedPath);
460             }
               folderBrowserDialog1.Dispose();
462         }

464         private void saveMnu_Click(object sender, EventArgs e)
            {
466             if (filePathTextBox.Text == null)
                {
468                 saveFileDialog1.InitialDirectory = Convert.ToString
                                        (Environment.SpecialFolder.MyDocuments);
470
                }
472             else
                {
474                 saveFileDialog1.InitialDirectory = filePathTextBox.Text;
                }
476             saveFileDialog1.Filter = "(*.jpg)|*.jpg|(*.bmp)|*.bmp|All Files (*.*)|*.*";
                saveFileDialog1.FilterIndex = 1;
478
                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
480             {
                    try
482                 {
                        //updateInitialDirectory(saveFileDialog1.FileName);
484                     pictureBox.Image.Save(saveFileDialog1.FileName);
                        Array.Resize(ref jpegList, jpegList.Length + 1);
486                     jpegList[jpegList.Length-1] = saveFileDialog1.FileName;
                        jpegOnlyCount++;
488                 }
                    catch (Exception ex)
490                 {
                        MessageBox.Show(ex.Message);
492                 }

494             }
            }
496
            private void exitToolStripMenuItem_Click(object sender, EventArgs e)
498         {
                DialogResult result;
500             result = MessageBox.Show("Are you sure you want to exit?", "Exit", MessageBoxButtons
                if (result == DialogResult.No)
502             {

504             }
                if (result == DialogResult.Yes)
506             {
                    Application.Exit();
508             }
            }
```

```
510
        private void deleteButton_Click(object sender, EventArgs e)
512     {
            DialogResult result = new DialogResult();
514
            if (pictureBox.Image != null)
516         {
                result = MessageBox.Show("Are you sure you want to delete this file?", "Delete p
518         }
            if (result == DialogResult.No)
520         {

522         }
            if (result == DialogResult.Yes)
524         {
                pictureBox.Image.Dispose();
526             if (filePathCount < jpegOnlyCount - 1)
                {
528                 pictureBox.Image = Image.FromFile(jpegList[filePathCount + 1]);
                }
530             else if (jpegOnlyCount == 1)
                {
532                 pictureBox.Image = Image.FromFile(jpegList[0]);
                    //add default image and We are done!!!
534             }
                else
536             {
                    pictureBox.Image = Image.FromFile(jpegList[filePathCount - 1]);
538             }

540             try
                {
542
                    File.Delete(jpegList[filePathCount]);
544                 if (jpegList.Length != 0)
                    {
546                     pictureBox.Image = Image.FromFile(jpegList[jpegOnlyCount - 1]);
                    }
548                 else
                    {
550                     pictureBox.Image = null;
                    }
552
                    updateDirectory(fileDirectory);
554
                }
556             catch (Exception ex)
                {
558                 MessageBox.Show(ex.Message);
                }
560         }
        }
562

564     private void pictureBox_Click(object sender, EventArgs e)
        {
566         if (filePathCount < jpegOnlyCount - 1)
            {
568             leftButton.Show();
```

```
                try
570             {
                    pictureBox.Image = Image.FromFile(jpegList[++filePathCount]);
572             }
                catch (OutOfMemoryException)
574             {
                    DialogResult result1 = MessageBox.Show("Incomplete JpegFile founded.
576                  Do you want to delete it?"
                                                , "Incomplete JpegFile founded!",
578                                                 MessageBoxButtons.YesNo);
                    if (result1 == DialogResult.Yes)
580                 {
                        pictureBox.Dispose();
582                     File.Delete(jpegList[filePathCount]);
                        updateDirectory(jpegList[filePathCount]);
584                     pictureBox.Image = Image.FromFile(jpegList[filePathCount--]);
                    }
586
                }
588         }
            else rightButton.Hide();
590     }

592     private void stopButton_Click(object sender, EventArgs e)
        {
594         stopCommand = true;

596         if(filePathCount!=0)
            updateDirectory(jpegList[filePathCount]);
598     }

600     private void connectButton_Click(object sender, EventArgs e)
        {
602         uavConn.Connect(0);

604         uavConn.SendTextToUAV("da 20 payload[0].mem_bytes[0]");
            uavConn.SendTextToUAV("payload.broadcast_bytes 255 0");
606     }

608     private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
610         uavConn.Close();
        }
612
        private void ResolutionComboBox_SelectedIndexChanged(object sender, EventArgs e)
614     {
            switch (ResolutionComboBox.SelectedIndex)
616         {
                case 0:
618                 uavConn.SendTextToUAV("payload[0].send_bytes 5 0x07 0x07 0x01");
                    break;
620             case 1:
                    uavConn.SendTextToUAV("payload[0].send_bytes 5 0x07 0x07 0x03");
622                 break;
                case 2:
624                 uavConn.SendTextToUAV("payload[0].send_bytes 5 0x07 0x07 0x05");
                    break;
626             case 3:
                    uavConn.SendTextToUAV("payload[0].send_bytes 5 0x07 0x07 0x07");
```

```
628                     break;
                    default:
630                     break;
                }
632         }
        }
634 }
```

Listing D.1: Main Form of GUI

D.2 `UAVConnector` Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.Net.Configuration;
using System.IO;

namespace NCamGS
{
    class UAVConnector
    {
        Socket dataPort = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType
        Socket consolePort = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolT

        public void Connect(byte dataStream)
        {
            try
            {
                dataPort.Connect("localhost", 8802);
                //MessageBox.Show("Connection to Port Name" + portName + "Port Number" + portNum
                Console.WriteLine("Data port connection to Port Name {0}  Port Number {1} is com

                // send selection byte to select datastream 0
                byte[] dataStreamSelection = { dataStream };
                dataPort.Send(dataStreamSelection);
                Console.WriteLine("Data stream selection completed.");

                consolePort.Connect("localhost", 8800);
                Console.WriteLine("Data port connection to Port Name {0} Port Number {1} is comp
            }
            catch
            {
                Console.WriteLine("Error code\n Unable to connect to UAV:\n Please check that:\n
                //MessageBox.Show("Error code\n Unable to connect to UAV:\n Please check that:\n
            }
        }

        public byte[] GetDataBytes()
        {
            //byte[] getByteSendBytes = new byte[numBytes]; // you need to send 0's to get bytes
            //for(int i = 0; i < numBytes; i++) {
            //    getByteSendBytes[i] = 0;
            //}
            //dataPort.Send(getByteSendBytes);
            byte[] sizeByte = { 0 };
            try
            {
                dataPort.Receive(sizeByte, 1, SocketFlags.None);
            }
            catch
            {

            }
            int numBytes = sizeByte[0];
            byte[] recBytes = new byte[numBytes];
            try
```

```
            {
59              dataPort.Receive(recBytes, numBytes, SocketFlags.None);
            }
61          catch
            {
63
            }
65          return recBytes;
        }
67
        public void SendTextToUAV(string textToUAV)
69      {
            char[] toUAVChar = new char[512];
71          byte[] toUAVByte = new byte[512];
            byte[] fromUAVByte = new byte[1000];
73          byte[] oneByteArray = new byte[1];
            textToUAV += "\n";
75          toUAVChar = textToUAV.ToCharArray();
            toUAVByte = System.Text.Encoding.ASCII.GetBytes(toUAVChar);
77          try
            {
79              int sendByte = consolePort.Send(toUAVByte, toUAVChar.Length, SocketFlags.None);
            }
81          catch (SocketException ex)
            {
83              Console.WriteLine("ERROR: " + ex.Message);
            }
85      }
        public void SendCommand(byte[] command)
87      {
            // use to send commmand to the uav
89          string toUAV;
            //string fromUAV;
91          char[] toUAVChar = new char[512];
            byte[] toUAVByte = new byte[512];
93          byte[] fromUAVByte = new byte[1000];
            byte[] oneByteArray = new byte[1];
95
            toUAV = "payload[0].send_bytes";
97
            for (int commandByteCount = 0; commandByteCount < command.Length; commandByteCount++
99          {
                oneByteArray[0] = command[commandByteCount];
101             toUAV += " 0x" + BitConverter.ToString(oneByteArray);
                //toUAV += " 0x" +System.Text.Encoding.ASCII.GetString(oneByteArray);
103         }

105         toUAV += "\n";
            toUAVChar = toUAV.ToCharArray();
107         toUAVByte = System.Text.Encoding.ASCII.GetBytes(toUAVChar);
            try
109         {
                int sendByte = consolePort.Send(toUAVByte, toUAVChar.Length, SocketFlags.None);
111         }
            catch (SocketException ex)
113         {
                Console.WriteLine("ERROR: " + ex.Message);
115         }
            //Port.Receive(fromUAVByte);
```

```
117             //fromUAV = System.Text.Encoding.ASCII.GetString(fromUAVByte);
        }
119
        public void Close()
121     {
            dataPort.Close();
123     }

125 }
}
```

Listing D.2: `UAVConnector` Class

D.3 Connection class for testing

```
class Connector
{
    static int requestCounter;
    static ArrayList hostData = new ArrayList();
    static StringCollection hostNames = new StringCollection();
    Socket UAV = new Socket(AddressFamily.InterNetwork,
     SocketType.Stream, ProtocolType.Tcp);

    public byte[] dataFromUAV()
    {
        byte[] uavData = new byte[1];
        byte[] testMessage = {0}; //send zero token to
        start the datastream port to receive something
        try
        {
            int sendByte = UAV.Send(testMessage, testMessage.Length,
            SocketFlags.None); //send zero token to the UAV
            Console.WriteLine("Sent {0} bytes.", sendByte); //print on the console
             so we know what we know
            int size = 8;

            while (true)
            {
                int byteCount = UAV.Receive(uavData, size, SocketFlags.None);
                 //receive data from the uav of fixed length size
                for(int countIn=0; countIn<uavData.Length;countIn++)
                Console.WriteLine("{0}", uavData[countIn]); //
            }
        }
        catch(SocketException uavMessage)
        {
            Console.WriteLine("{0} Error code:{1}.",
             uavMessage.Message, uavMessage.ErrorCode);
        }
        return uavData;
    }

    public void ConnectToPort()
    {
        const Int32 portNumber = 8802;
        const string portName = "localhost";
        try
        {
            UAV.Connect("localhost", portNumber);
            Console.WriteLine("Connection to Port Name:{0}; Port Number{1} is complete!"
            ,portName,portNumber);
        }
        catch (SocketException uavMessage)
        {
            Console.WriteLine("{0} Error code:{1}.", uavMessage.Message
            , uavMessage.ErrorCode);
        }
    }
}
```

Listing D.3: `Connector` for testing with console application

Appendix E

Schematics

All schematics in this Appendix were generated using gschem from the gEDA toolsuite. [1]

These are all available in our GitHub repository [21], under the schematics/ folder. PDF and PS files are also available in the schematics/images folder.

Note that you need to run

```
$ gschem SCHEMATIC_NAME
```

from within this folder to use the non-standard symbols available in that folder.

[1] http://geda.seul.org/wiki/geda:gaf

E.1 Dummy Payload

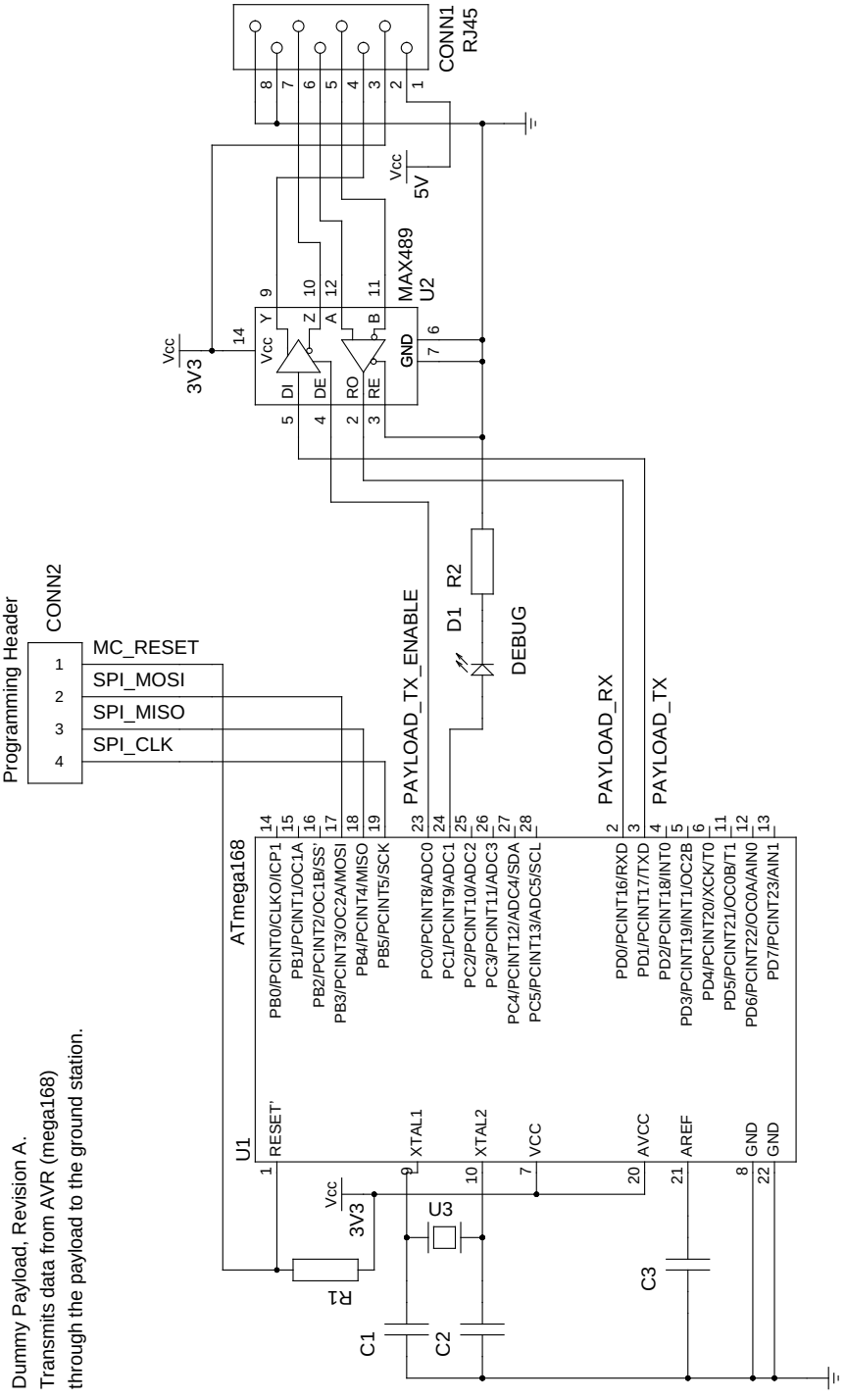


FIGURE E.1: Schematic of the circuit used to test communication from a "dummy" payload to the autopilot module. Largely based on a schematic provided by our customer, and runs a slightly modified version of source code provided again, by our customer.

E.2 Final Payload Module

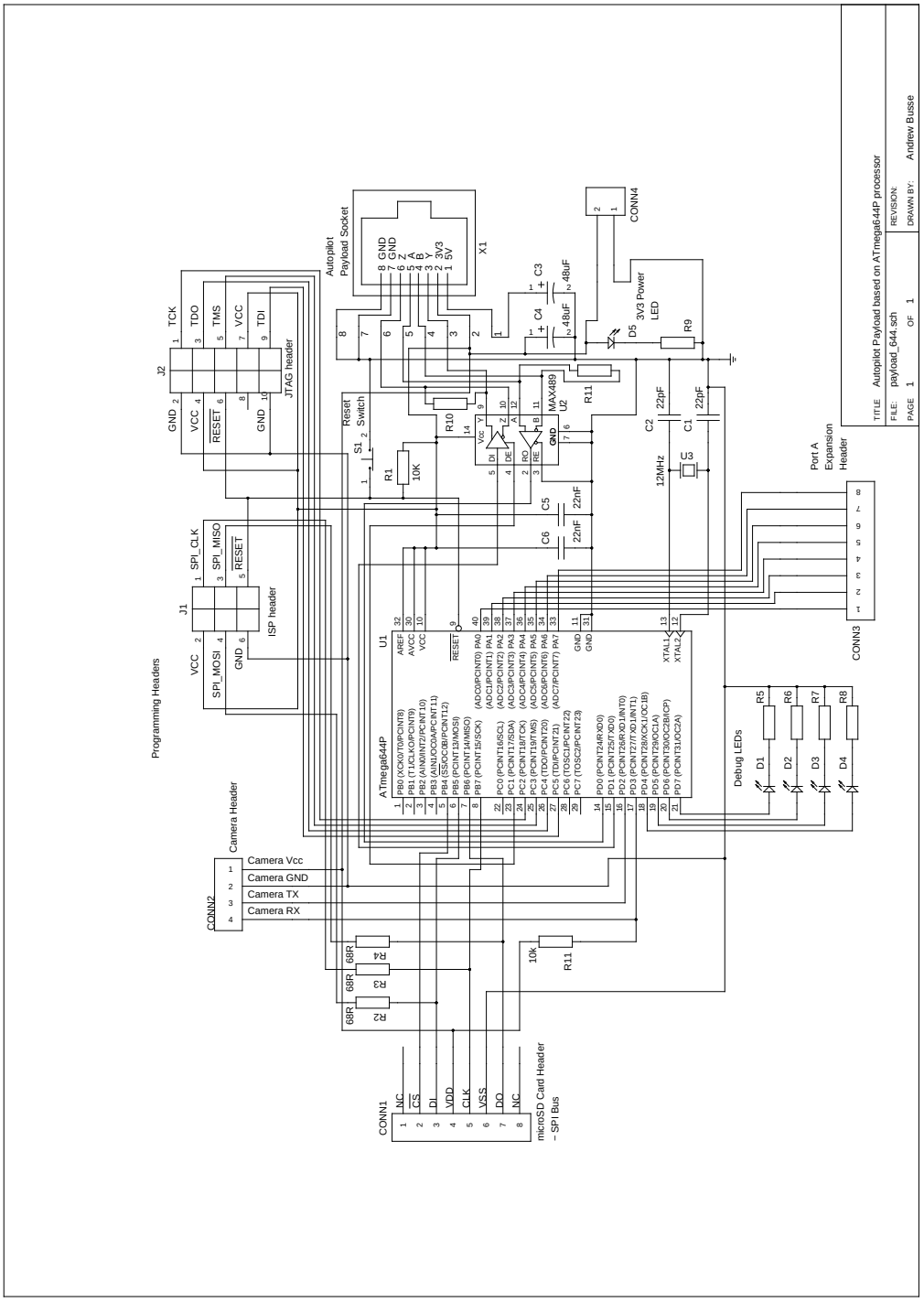


FIGURE E.2: Final Schematic of the implemented payload module. See 6.10 for a detailed explanation of which components are for what purpose

E.3 Payload: Arduino Uno with additional ATmega168

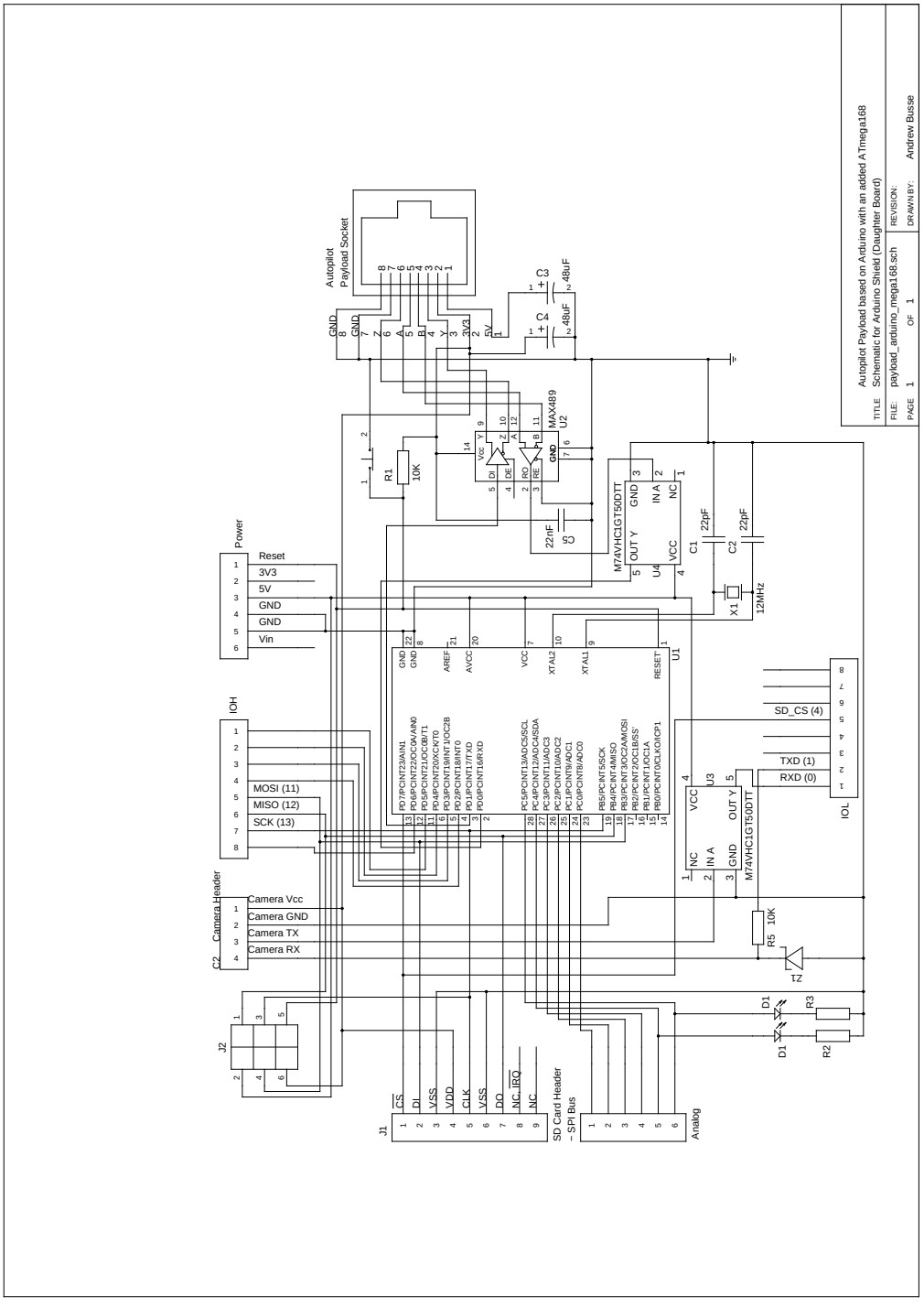


FIGURE E.3: Schematic for a considered payload module: an Arduino Uno with a daughter board containing an ATmega168 for Autopilot to Payload module communication.

E.4 Payload: Arduino Uno with Multiplexed UART line

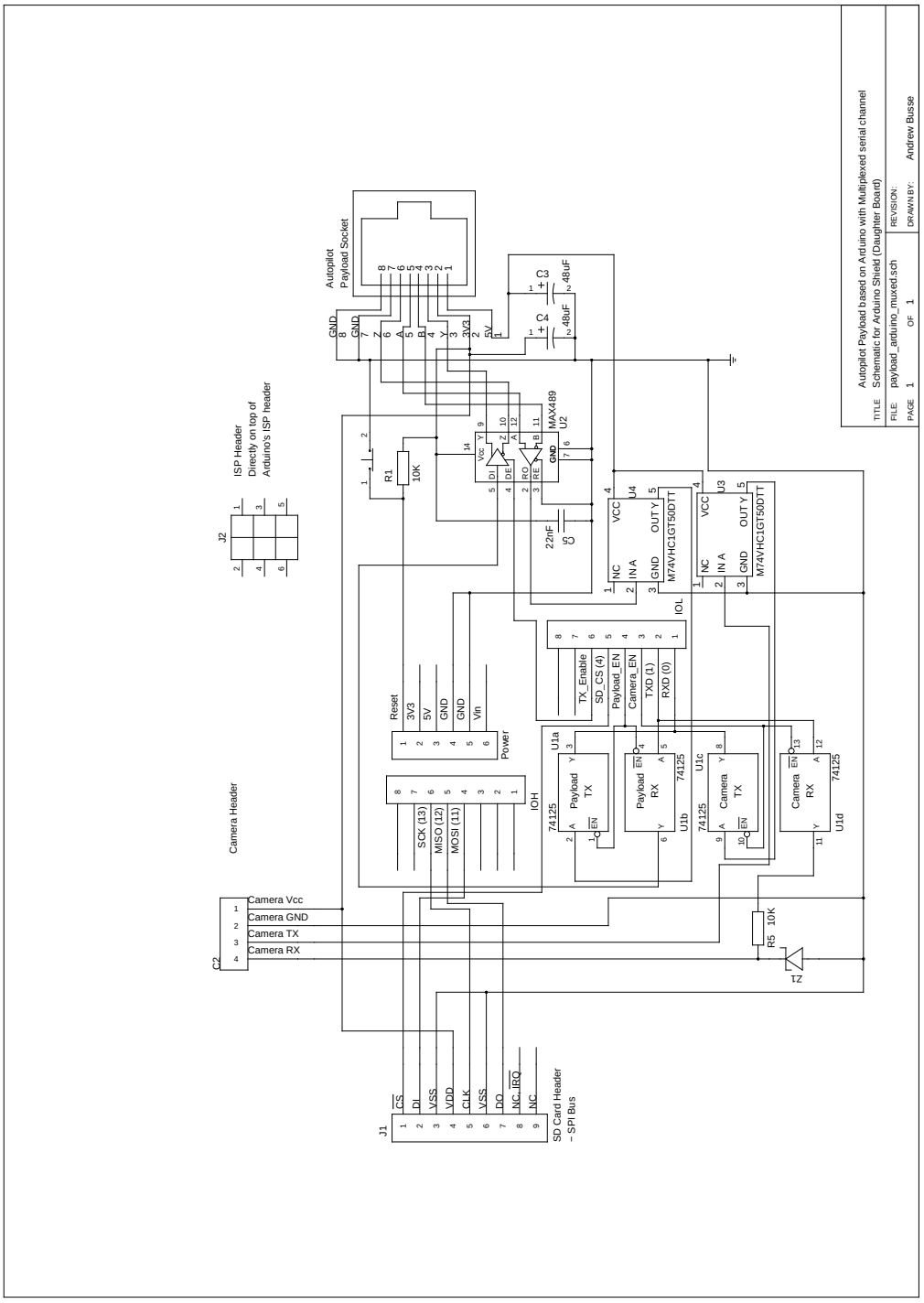


FIGURE E.4: Schematic for a considered payload module: an Arduino Uno with its UART line multiplexed between camera and autopilot communication

Appendix F

PCB Layout

The layout of the final PCB module. As seen in figure 6.10

This was made using PCB [1] from the gEDA toolsuite.

Gerber files, and full source files, are available in our central GitHub [21] repository, under schematics/payload_644.pcb and schematics/gerbers

Running

```
$ gsch2pcb projectrc
```

from within schematics/ should update any changes you make to payload_644.sch.

[1] http://geda.seul.org/wiki/geda:gaf

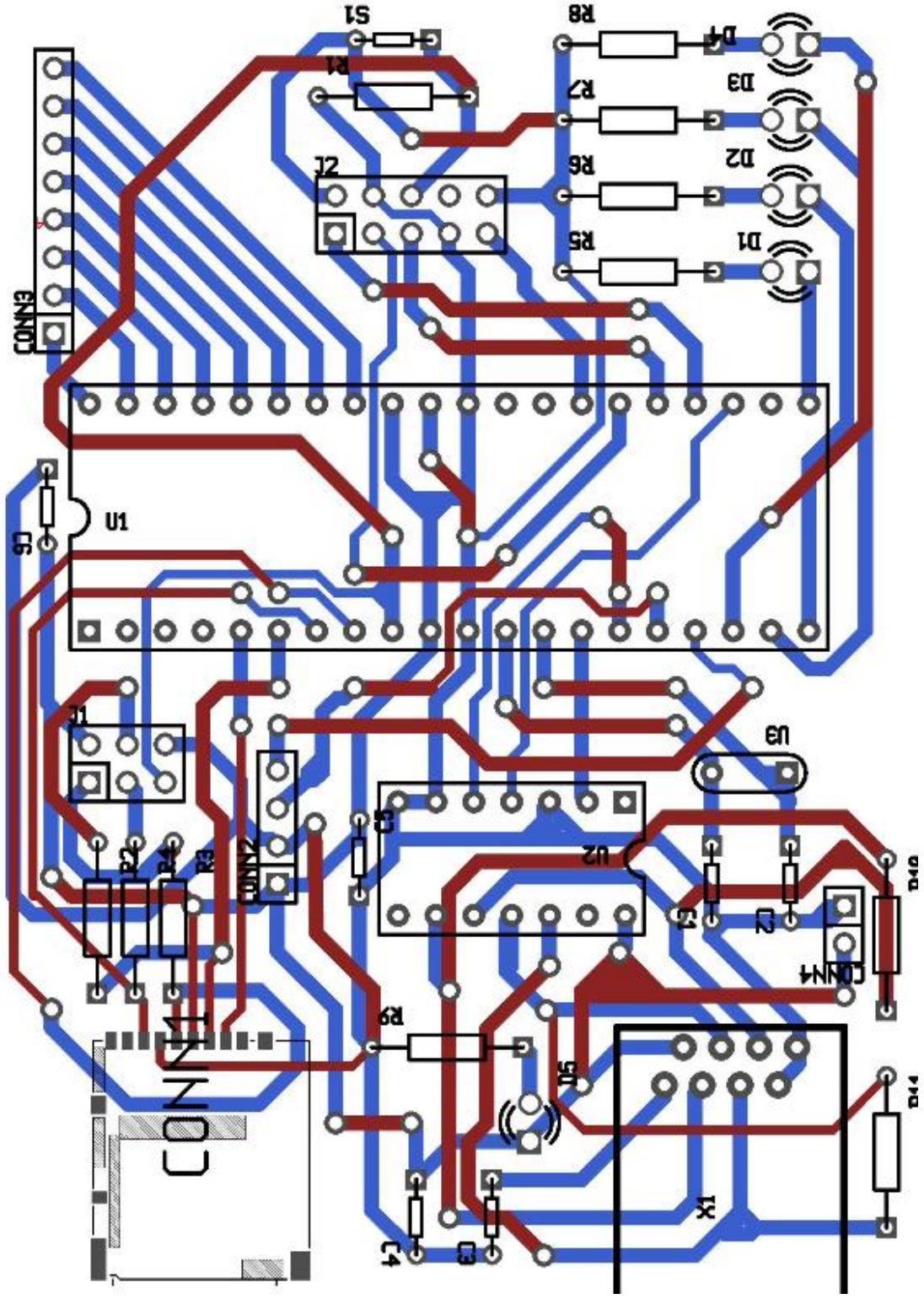


FIGURE F.1: Layout of our delivered PCB module. Dark Red: component layer. Light Blue: solder layer

Appendix G

Agreed Specification

This Specification is our specification as it was submitted on to C-BASS on Monday 10th October 2011

School of Electronics and Computer Science
ELEC6050 MEng Group Design Project

Project Specification and Plan

Title: Unmanned Aircraft Camera Module (GDP Group 18)

Supervisor: Rob Maunder (rm@ecs.soton.ac.uk)

Team Members: John Charlesworth (jgac1g08@ecs.soton.ac.uk)
Paramithi Svastisinha (ps6g08@ecs.soton.ac.uk)
Piyabhum Sornpaisarn (ps26g08@ecs.soton.ac.uk)
Andrew Busse (ajb2g08@ecs.soton.ac.uk)
Michael Hodgson (mh23g08@ecs.soton.ac.uk)

Customer: Dr. Matt Bennett, SkyCircuits (m.bennett@skycircuits.com)

Project Specification:

To design, build and test an electronic module capable of capturing still images from an unmanned aerial vehicle (UAV) and transmitting the images to a base station. The module must use the UAV autopilot's low-bandwidth RS485 serial link (38.4 kBaud). A program must be written to interface with the base station software over a TCP/IP link, allowing image data to be received and displayed to the user. The electronic module will be constructed using strip-boards and will later be implemented on PCB if time is available.

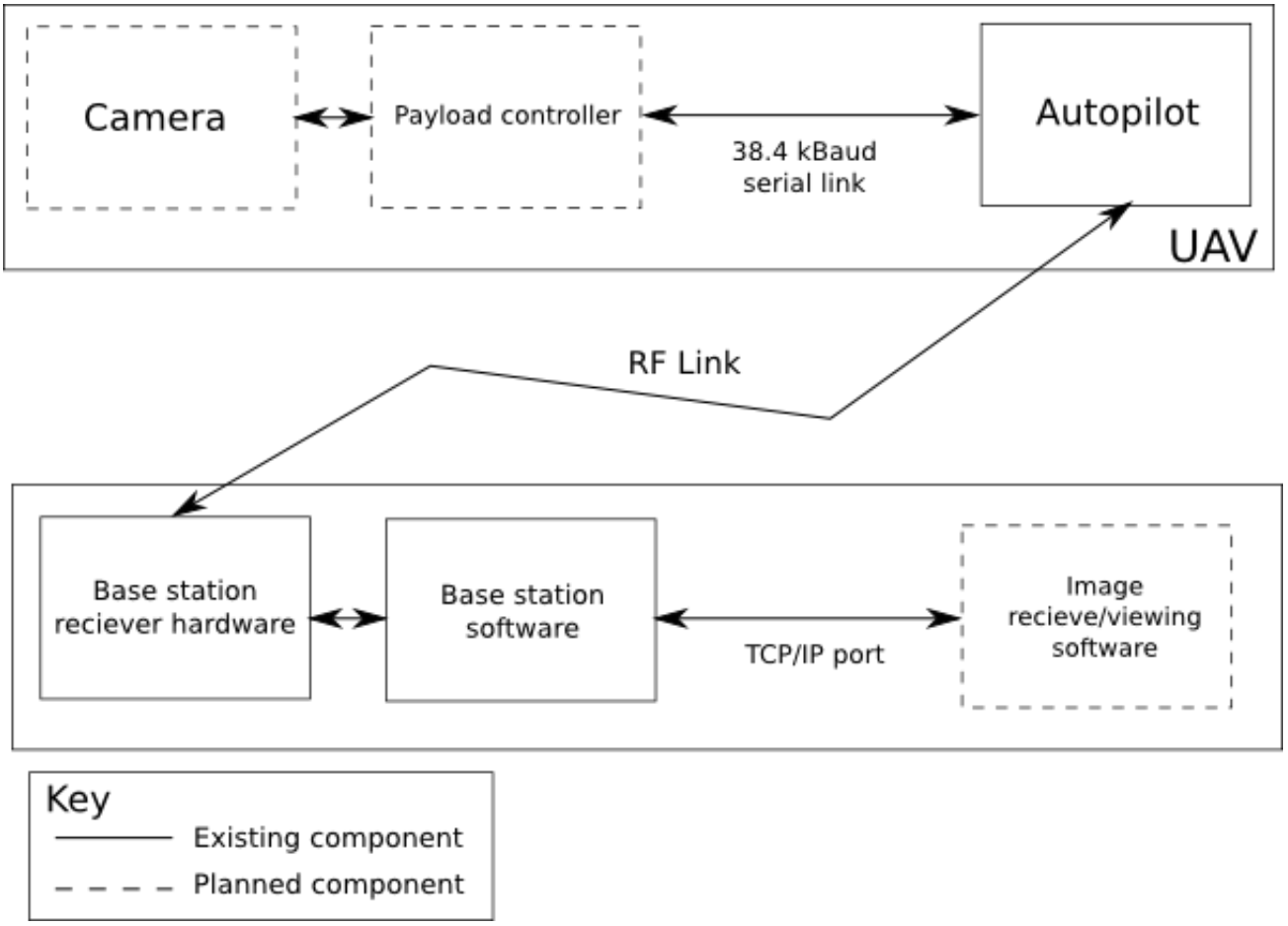
The aim of the project will be to achieve the following criteria:

- The image will be encoded in such a way that a low quality image will be available quickly, the quality of which will improve as more information is downloaded. [high priority]
- Minimise the time needed to download the images from the UAV to the base station. The time from the user's prompt until the image has been fully downloaded will be measured against the theoretical 3 minutes necessary to transmit a full image without using any compression. The goal will be to obtain a full image in **less than 3 minutes**. [high priority]
- The module weight will be **less than 250g**. [medium priority]
- Image resolution of **640x480**. [medium priority]
- Allow the user to perform the following actions on the UAV's camera from the base station:
    - Prompt the UAV to **capture and download an image.** [high priority]
    - **Cancel** the downloading of any image while the image is being downloaded. [medium priority]
    - **Resend** an image in case the current preview is corrupted. [low priority]
    - **Interrupt** the download of an incomplete image and allow the user to **save** the incomplete image. [low priority]
    - Select the **resolution settings** of the image. [low priority]
    - Display a **progress indicator** which will show the percentage of the image data received, as well as a time estimate for the rest of the image to be downloaded. [low priority]
    - The image capture will be triggered automatically by the UAV using triggers built into the autopilot. [low priority]
    - Allow the user to command the image capture to **trigger periodically** over a **user-specified time interval** will be added if time permits. [low priority]

- Images will be transmitted in **colour** as opposed to black and white. [low priority]
- The user can select between a colour image and a black and white. [low priority]

Deliverables to the customer include:

- Hardware: Camera module, constructed on PCB (if time permits, otherwise on strip-board), including layout designs.
- Software: all firmware for the electronic module, and software on the base station for viewing images. The full source code and all executable files will be included.
- Documentation: Technical and User Documentation. This includes all schematics related to hardware as well as all other documents concerning both the software and hardware delivered.
- Public repository: The full source code, all schematics, and all documents concerning both the software and hardware will be included on a public repository so that the client may share this information with his clients.



Complete Block Diagram of the System

Appendix H

Gantt Charts

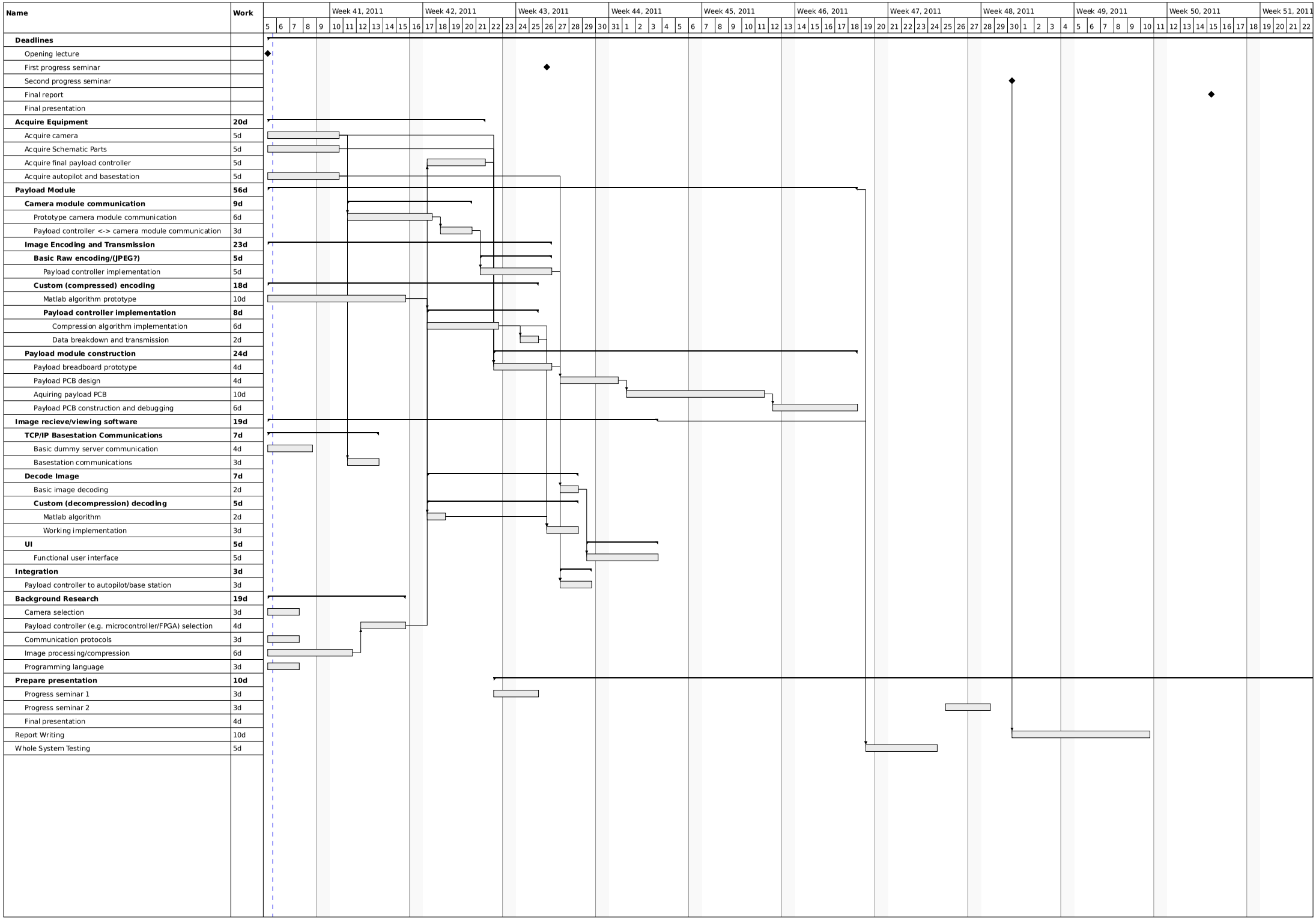
This section shows the Gantt Charts produced for this project. The first one, displayed over the first three pages, shows our initial Gantt chart, as produced for handin with the specification (10th October 2011).

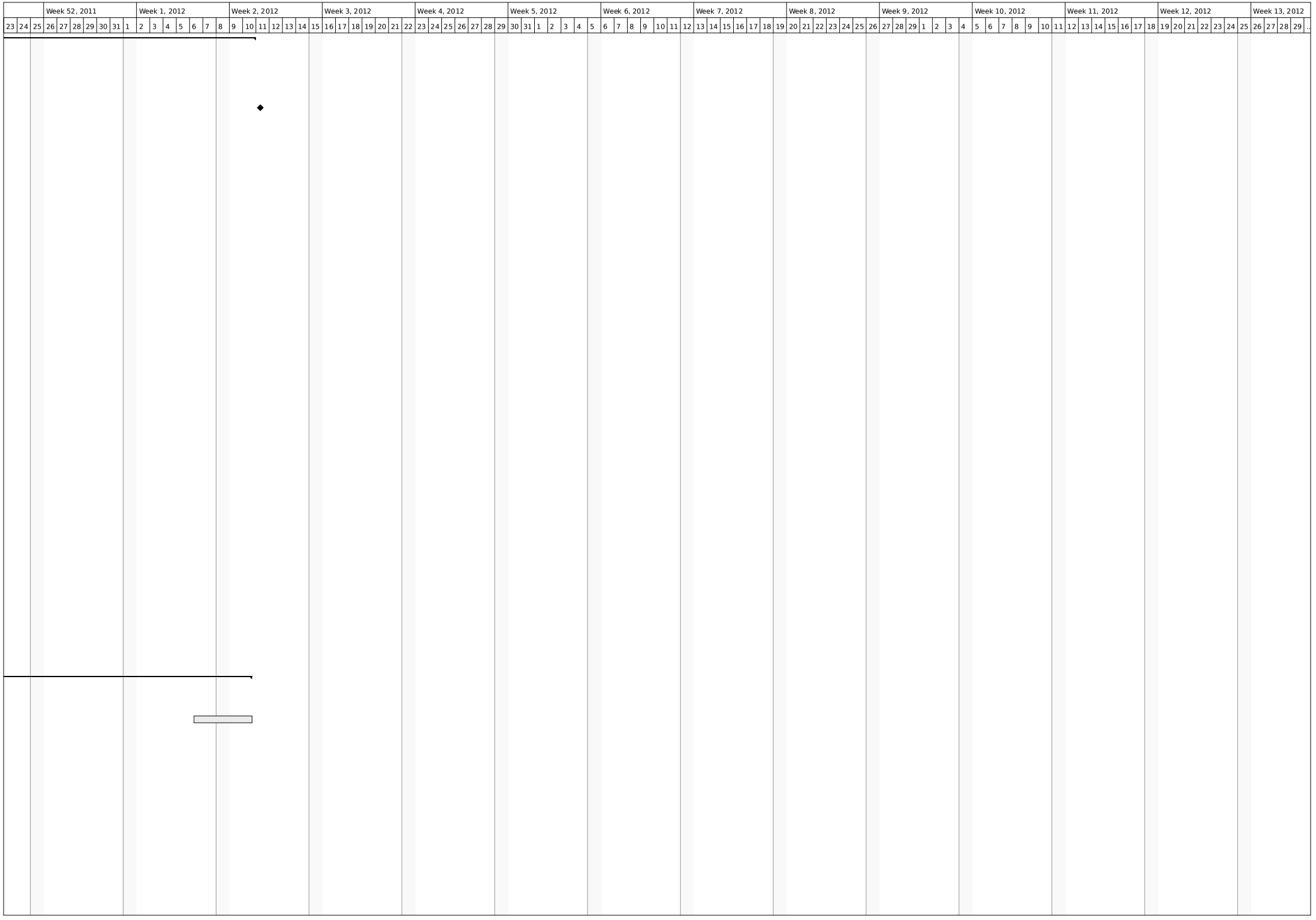
The second, shown over pages 4 to 6, shows a revised version of our Gantt Chart, produced on 8th November 2011.

The third, shown over pages 7 to 9 shows the final version of our Gantt chart as of 15th December 2011.

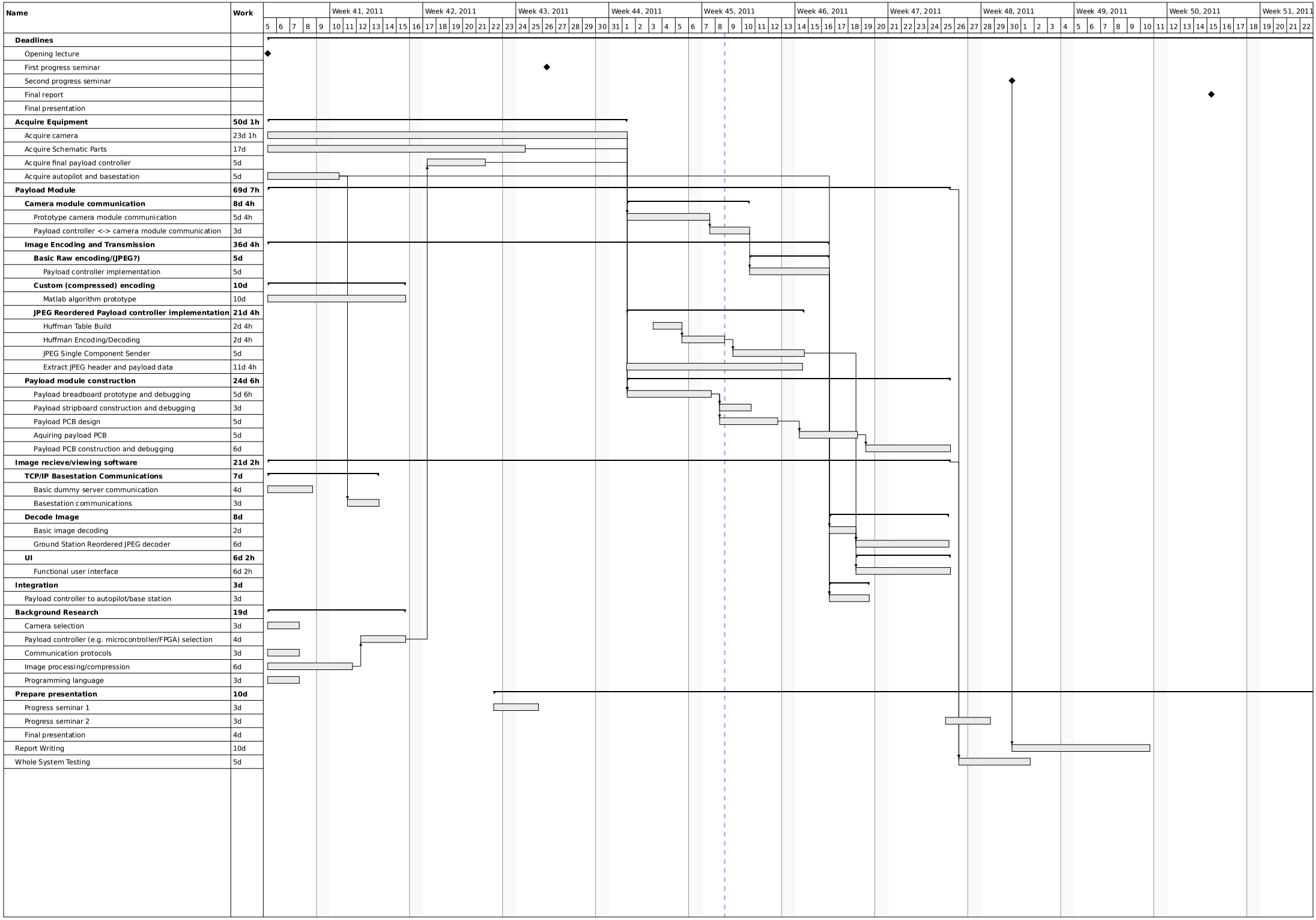
These Gantt Charts have been produced using the Planner tool from GNOME office. [1]

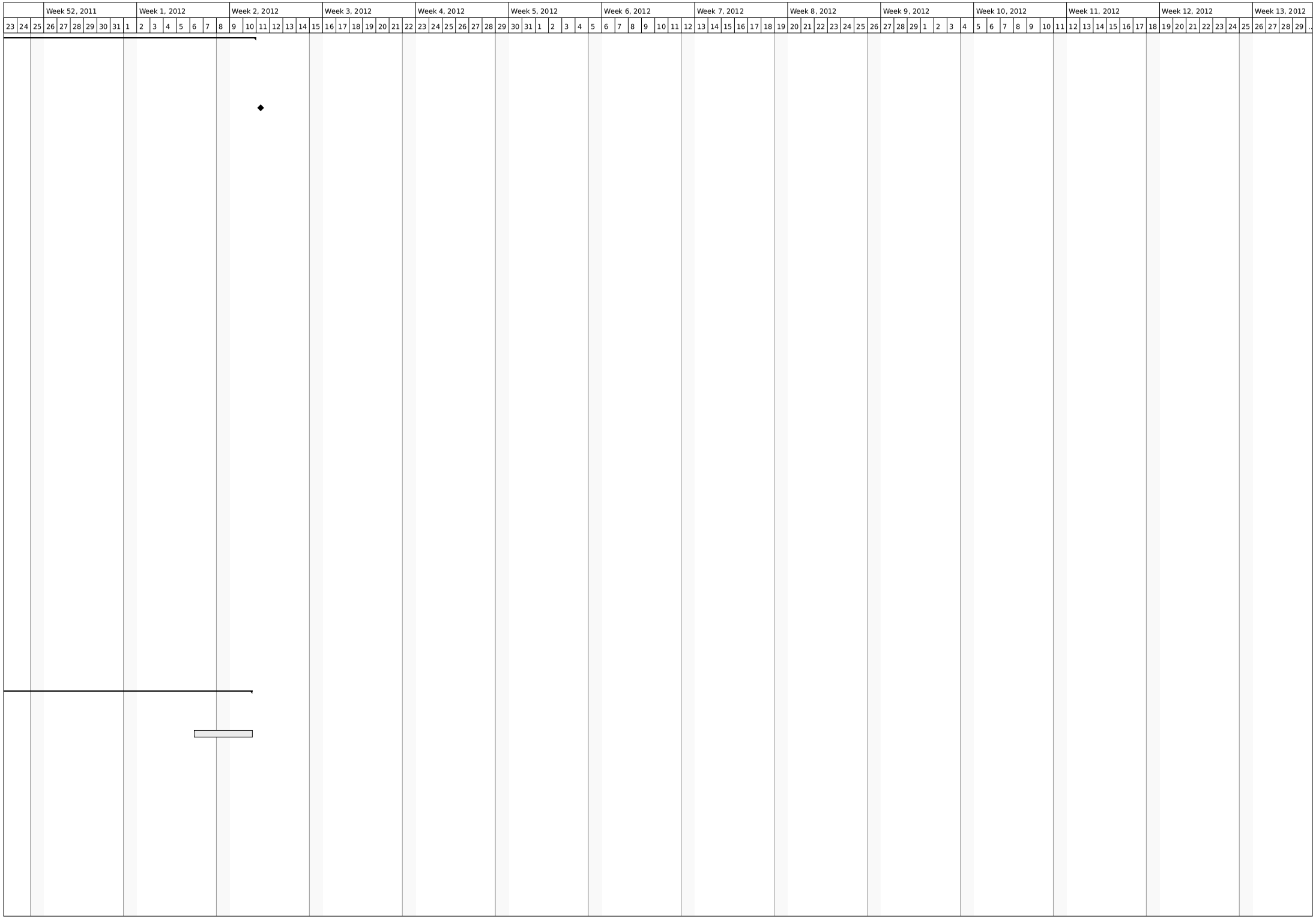
[1] http://live.gnome.org/Planner


Name
Work
Week 41, 2011
Week 42, 2011
Week 43, 2011
Week 44, 2011
Week 45, 2011
Week 46, 2011
Week 47, 2011
Week 48, 2011
Week 49, 2011
Week 50, 2011
Week 51, 2011
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Deadlines
Opening lecture
First progress seminar
Second progress seminar
Final report
Final presentation
Acquire Equipment 20d
Acquire camera 5d
Acquire Schematic Parts 5d
Acquire final payload controller 5d
Acquire autopilot and basestation 5d
Payload Module 56d
Camera module communication 9d
Prototype camera module communication 6d
Payload controller <-> camera module communication 3d
Image Encoding and Transmission 23d
Basic Raw encoding/(JPEG?) 5d
Payload controller implementation 5d
Custom (compressed) encoding 18d
Matlab algorithm prototype 10d
Payload controller implementation 8d
Compression algorithm implementation 6d
Data breakdown and transmission 2d
Payload module construction 24d
Payload breadboard prototype 4d
Payload PCB design 4d
Aquiring payload PCB 10d
Payload PCB construction and debugging 6d
Image recieve/viewing software 19d
TCP/IP Basestation Communications 7d
Basic dummy server communication 4d
Basestation communications 3d
Decode Image 7d
Basic image decoding 2d
Custom (decompression) decoding 5d
Matlab algorithm 2d
Working implementation 3d
UI 5d
Functional user interface 5d
Integration 3d
Payload controller to autopilot/base station 3d
Background Research 19d
Camera selection 3d
Payload controller (e.g. microcontroller/FPGA) selection 4d
Communication protocols 3d
Image processing/compression 6d
Programming language 3d
Prepare presentation 10d
Progress seminar 1 3d
Progress seminar 2 3d
Final presentation 4d
Report Writing 10d
Whole System Testing 5d

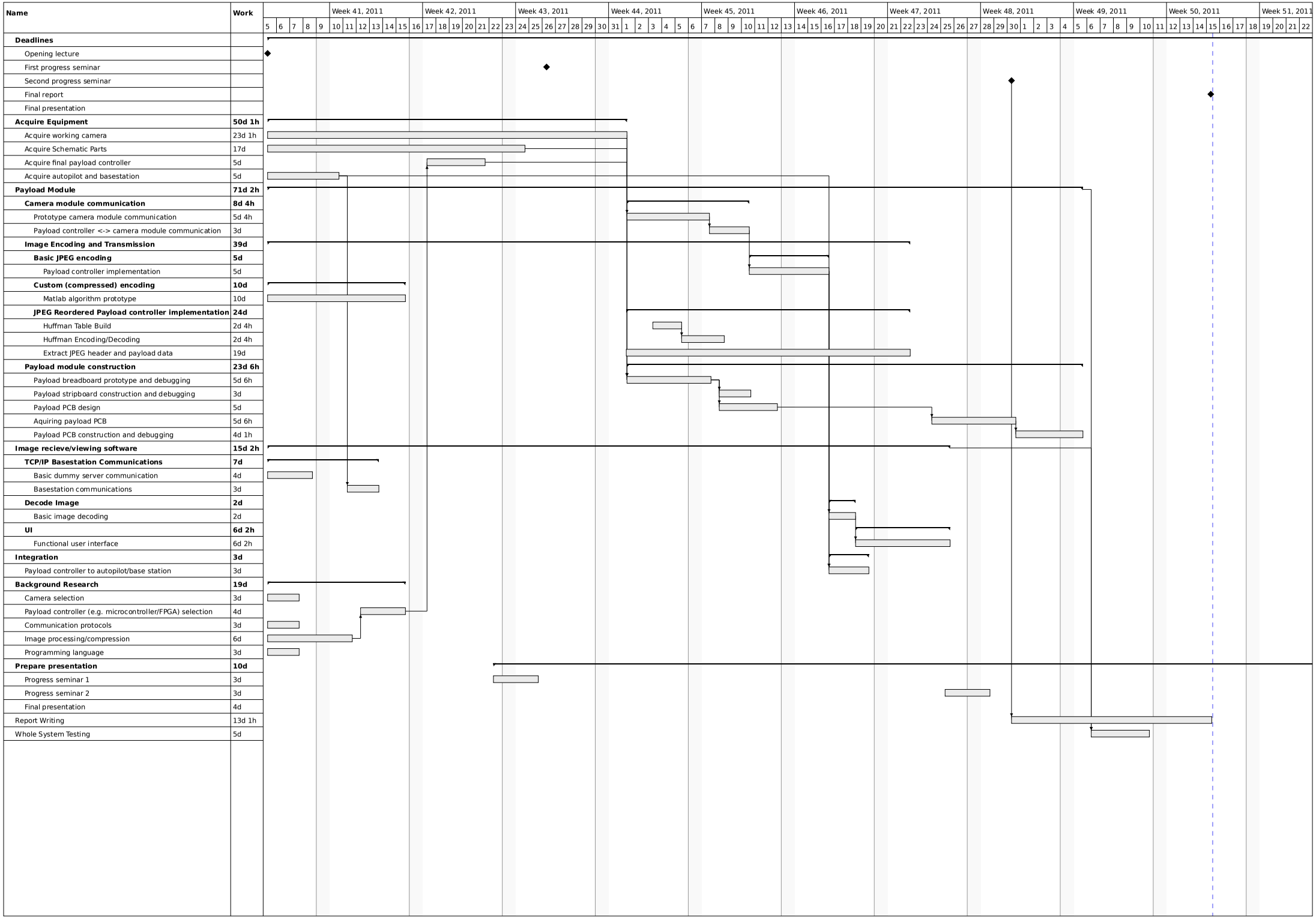

Week 52, 2011
Week 1, 2012
Week 2, 2012
Week 3, 2012
Week 4, 2012
Week 5, 2012
Week 6, 2012
Week 7, 2012
Week 8, 2012
Week 9, 2012
Week 10, 2012
Week 11, 2012
Week 12, 2012
Week 13, 2012
23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

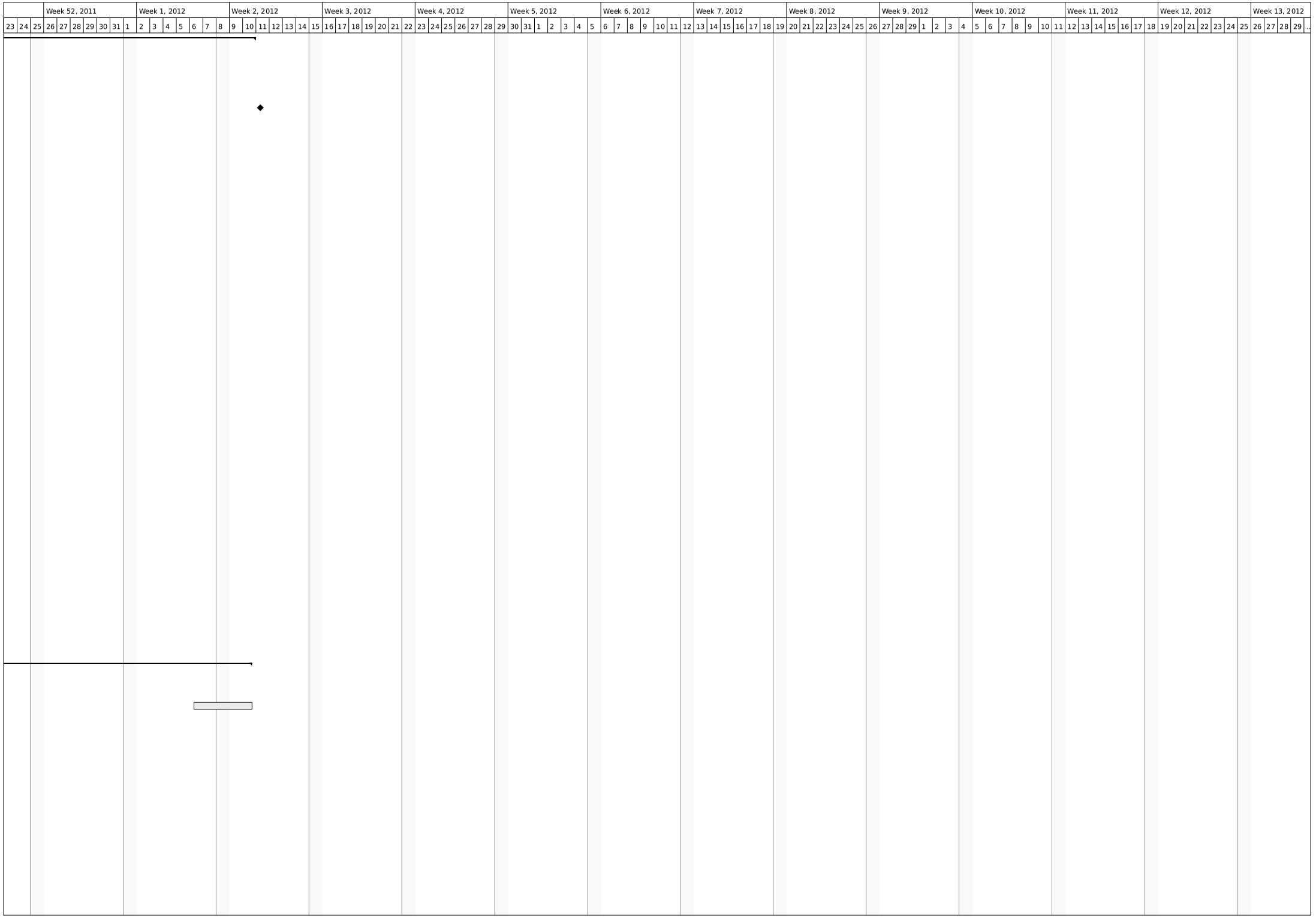
WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to	% Complete
1	**Deadlines**	**5 Oct**	**11 Jan**		**84d**		**0**		0
1.1	Opening lecture	5 Oct	5 Oct	N/A	N/A	84d	0		0
1.2	First progress seminar	26 Oct	26 Oct	N/A	N/A	66d	0		0
1.3	Second progress seminar	30 Nov	30 Nov	N/A	N/A	27d	0		0
1.4	Final report	15 Dec	15 Dec	N/A	N/A	23d	0		0
1.5	Final presentation	11 Jan	11 Jan	N/A	N/A		0		0
2	**Acquire Equipment**	**5 Oct**	**21 Oct**	**20d**	**15d**	**69d**	**0**		0
2.1	Acquire camera	5 Oct	10 Oct	5d	5d	50d	0		0
2.2	Acquire Schematic Parts	5 Oct	10 Oct	5d	5d	50d	0		0
2.3	Acquire final payload controller	17 Oct	21 Oct	5d	5d	40d	0		0
2.4	Acquire autopilot and basestation	5 Oct	10 Oct	5d	5d	71d	0		0
3	**Payload Module**	**5 Oct**	**18 Nov**	**56d**	**39d**	**40d**	**0**		0
3.1	**Camera module communication**	**11 Oct**	**20 Oct**	**9d**	**9d**	**65d**	**0**		0
3.1.1	Prototype camera module communication	11 Oct	17 Oct	6d	6d	53d	0		0
3.1.2	Payload controller <-> camera module communication	18 Oct	20 Oct	3d	3d	53d	0		0
3.2	**Image Encoding and Transmission**	**5 Oct**	**26 Oct**	**23d**	**19d**	**60d**	**0**		0
3.2.1	**Basic Raw encoding/(JPEG?)**	**21 Oct**	**26 Oct**	**5d**	**5d**	**60d**	**0**		0
3.2.1.1	Payload controller implementation	21 Oct	26 Oct	5d	5d	53d	0		0
3.2.2	**Custom (compressed) encoding**	**5 Oct**	**25 Oct**	**18d**	**18d**	**61d**	**0**		0
3.2.2.1	Matlab algorithm prototype	5 Oct	15 Oct	10d	10d	58d	0		0
3.2.2.2	**Payload controller implementation**	**17 Oct**	**25 Oct**	**8d**	**8d**	**61d**	**0**		0
3.2.2.2.1	Compression algorithm implementation	17 Oct	22 Oct	6d	6d	58d	0		0
3.2.2.2.2	Data breakdown and transmission	24 Oct	25 Oct	2d	2d	58d	0		0
3.3	**Payload module construction**	**22 Oct**	**18 Nov**	**24d**	**24d**	**40d**	**0**		0
3.3.1	Payload breadboard prototype	22 Oct	26 Oct	4d	4d	40d	0		0
3.3.2	Payload PCB design	27 Oct	31 Oct	4d	4d	41d	0		0
3.3.3	Aquiring payload PCB	1 Nov	11 Nov	10d	10d	40d	0		0
3.3.4	Payload PCB construction and debugging	12 Nov	18 Nov	6d	6d	40d	0		0
4	**Image recieve/viewing software**	**5 Oct**	**3 Nov**	**19d**	**26d**	**53d**	**0**		0
4.1	**TCP/IP Basestation Communications**	**5 Oct**	**13 Oct**	**7d**	**8d**	**71d**	**0**		0
4.1.1	Basic dummy server communication	5 Oct	8 Oct	4d	4d	75d	0		0
4.1.2	Basestation communications	11 Oct	13 Oct	3d	3d	71d	0		0
4.2	**Decode Image**	**17 Oct**	**28 Oct**	**7d**	**11d**	**58d**	**0**		0
4.2.1	Basic image decoding	27 Oct	28 Oct	2d	2d	53d	0		0
4.2.2	**Custom (decompression) decoding**	**17 Oct**	**28 Oct**	**5d**	**11d**	**58d**	**0**		0
4.2.2.1	Matlab algorithm	17 Oct	18 Oct	2d	2d	64d	0		0
4.2.2.2	Working implementation	26 Oct	28 Oct	3d	3d	58d	0		0
4.3	**UI**	**29 Oct**	**3 Nov**	**5d**	**5d**	**53d**	**0**		0
4.3.1	Functional user interface	29 Oct	3 Nov	5d	5d	53d	0		0
5	**Integration**	**27 Oct**	**29 Oct**	**3d**	**3d**	**62d**	**0**		0
5.1	Payload controller to autopilot/base station	27 Oct	29 Oct	3d	3d	62d	0		0
6	**Background Research**	**5 Oct**	**15 Oct**	**19d**	**10d**	**74d**	**0**		0
6.1	Camera selection	5 Oct	7 Oct	3d	3d	81d	0		0
6.2	Payload controller (e.g. microcontroller/FPGA) selection	12 Oct	15 Oct	4d	4d	40d	0		0
6.3	Communication protocols	5 Oct	7 Oct	3d	3d	81d	0		0
6.4	Image processing/compression	5 Oct	11 Oct	6d	6d	40d	0		0
6.5	Programming language	5 Oct	7 Oct	3d	3d	81d	0		0
7	**Prepare presentation**	**22 Oct**	**10 Jan**	**10d**	**69d**		**0**		0
7.1	Progress seminar 1	22 Oct	25 Oct	3d	3d	66d	0		0
7.2	Progress seminar 2	25 Nov	28 Nov	3d	3d	37d	0		0
7.3	Final presentation	6 Jan	10 Jan	4d	4d		0		0
8	Report Writing	30 Nov	10 Dec	10d	10d	26d	0		0
9	Whole System Testing	19 Nov	24 Nov	5d	5d	40d	0		0


Name
Work
Week 41, 2011
Week 42, 2011
Week 43, 2011
Week 44, 2011
Week 45, 2011
Week 46, 2011
Week 47, 2011
Week 48, 2011
Week 49, 2011
Week 50, 2011
Week 51, 2011
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Deadlines
Opening lecture
First progress seminar
Second progress seminar
Final report
Final presentation
Acquire Equipment 50d 1h
Acquire camera 23d 1h
Acquire Schematic Parts 17d
Acquire final payload controller 5d
Acquire autopilot and basestation 5d
Payload Module 69d 7h
Camera module communication 8d 4h
Prototype camera module communication 5d 4h
Payload controller <-> camera module communication 3d
Image Encoding and Transmission 36d 4h
Basic Raw encoding/(JPEG?) 5d
Payload controller implementation 5d
Custom (compressed) encoding 10d
Matlab algorithm prototype 10d
JPEG Reordered Payload controller implementation 21d 4h
Huffman Table Build 2d 4h
Huffman Encoding/Decoding 2d 4h
JPEG Single Component Sender 5d
Extract JPEG header and payload data 11d 4h
Payload module construction 24d 6h
Payload breadboard prototype and debugging 5d 6h
Payload stripboard construction and debugging 3d
Payload PCB design 5d
Aquiring payload PCB 5d
Payload PCB construction and debugging 6d
Image recieve/viewing software 21d 2h
TCP/IP Basestation Communications 7d
Basic dummy server communication 4d
Basestation communications 3d
Decode Image 8d
Basic image decoding 2d
Ground Station Reordered JPEG decoder 6d
UI 6d 2h
Functional user interface 6d 2h
Integration 3d
Payload controller to autopilot/base station 3d
Background Research 19d
Camera selection 3d
Payload controller (e.g. microcontroller/FPGA) selection 4d
Communication protocols 3d
Image processing/compression 6d
Programming language 3d
Prepare presentation 10d
Progress seminar 1 3d
Progress seminar 2 3d
Final presentation 4d
Report Writing 10d
Whole System Testing 5d


Week 52, 2011
Week 1, 2012
Week 2, 2012
Week 3, 2012
Week 4, 2012
Week 5, 2012
Week 6, 2012
Week 7, 2012
Week 8, 2012
Week 9, 2012
Week 10, 2012
Week 11, 2012
Week 12, 2012
Week 13, 2012
23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to	% Complete
1	**Deadlines**	**5 Oct**	**11 Jan**		**84d**		**0**		0
1.1	Opening lecture	5 Oct	5 Oct	N/A	N/A	84d	0		0
1.2	First progress seminar	26 Oct	26 Oct	N/A	N/A	66d	0		0
1.3	Second progress seminar	30 Nov	30 Nov	N/A	N/A	27d	0		0
1.4	Final report	15 Dec	15 Dec	N/A	N/A	23d	0		0
1.5	Final presentation	11 Jan	11 Jan	N/A	N/A		0		0
2	**Acquire Equipment**	**5 Oct**	**1 Nov**	**50d 1h**	**23d 1h**	**60d 6h**	**0**		0
2.1	Acquire camera	5 Oct	1 Nov	23d 1h	23d 1h	34d 6h	0		0
2.2	Acquire Schematic Parts	5 Oct	24 Oct	17d	17d	41d	0		0
2.3	Acquire final payload controller	17 Oct	21 Oct	5d	5d	43d	0		0
2.4	Acquire autopilot and basestation	5 Oct	10 Oct	5d	5d	71d	0		0
3	**Payload Module**	**5 Oct**	**25 Nov**	**69d 7h**	**45d**	**34d**	**0**		0
3.1	**Camera module communication**	**1 Nov**	**10 Nov**	**8d 4h**	**8d 4h**	**47d 2h**	**0**		0
3.1.1	Prototype camera module communication	1 Nov	7 Nov	5d 4h	5d 4h	35d 2h	0		0
3.1.2	Payload controller <-> camera module communication	7 Nov	10 Nov	3d	3d	34d 2h	0		0
3.2	**Image Encoding and Transmission**	**5 Oct**	**16 Nov**	**36d 4h**	**36d 5h**	**42d 2h**	**0**		0
3.2.1	**Basic Raw encoding/(JPEG?)**	**10 Nov**	**16 Nov**	**5d**	**5d**	**42d 2h**	**0**		0
3.2.1.1	Payload controller implementation	10 Nov	16 Nov	5d	5d	34d 2h	0		0
3.2.2	**Custom (compressed) encoding**	**5 Oct**	**15 Oct**	**10d**	**10d**	**69d**	**0**		0
3.2.2.1	Matlab algorithm prototype	5 Oct	15 Oct	10d	10d	69d	0		0
3.2.3	**JPEG Reordered Payload controller implementation**	**1 Nov**	**14 Nov**	**21d 4h**	**12d**	**44d**	**0**		0
3.2.3.1	Huffman Table Build	3 Nov	5 Nov	2d 4h	2d 4h	37d 4h	0		0
3.2.3.2	Huffman Encoding/Decoding	5 Nov	8 Nov	2d 4h	2d 4h	38d	0		0
3.2.3.3	JPEG Single Component Sender	9 Nov	14 Nov	5d	5d	38d	0		0
3.2.3.4	Extract JPEG header and payload data	1 Nov	14 Nov	11d 4h	11d 4h	44d 3h	0		0
3.3	**Payload module construction**	**1 Nov**	**25 Nov**	**24d 6h**	**21d 6h**	**34d**	**0**		0
3.3.1	Payload breadboard prototype and debugging	1 Nov	7 Nov	5d 6h	5d 6h	35d	0		0
3.3.2	Payload stripboard construction and debugging	8 Nov	10 Nov	3d	3d	47d	0		0
3.3.3	Payload PCB design	8 Nov	12 Nov	5d	5d	34d	0		0
3.3.4	Aquiring payload PCB	14 Nov	18 Nov	5d	5d	34d	0		0
3.3.5	Payload PCB construction and debugging	19 Nov	25 Nov	6d	6d	34d	0		0
4	**Image recieve/viewing software**	**5 Oct**	**25 Nov**	**21d 2h**	**45d**	**34d**	**0**		0
4.1	**TCP/IP Basestation Communications**	**5 Oct**	**13 Oct**	**7d**	**8d**	**71d**	**0**		0
4.1.1	Basic dummy server communication	5 Oct	8 Oct	4d	4d	75d	0		0
4.1.2	Basestation communications	11 Oct	13 Oct	3d	3d	71d	0		0
4.2	**Decode Image**	**16 Nov**	**25 Nov**	**8d**	**8d**	**34d 2h**	**0**		0
4.2.1	Basic image decoding	16 Nov	18 Nov	2d	2d	34d 2h	0		0
4.2.2	Ground Station Reordered JPEG decoder	18 Nov	25 Nov	6d	6d	34d 2h	0		0
4.3	**UI**	**18 Nov**	**25 Nov**	**6d 2h**	**6d 2h**	**34d**	**0**		0
4.3.1	Functional user interface	18 Nov	25 Nov	6d 2h	6d 2h	34d	0		0
5	**Integration**	**16 Nov**	**19 Nov**	**3d**	**3d**	**44d 2h**	**0**		0
5.1	Payload controller to autopilot/base station	16 Nov	19 Nov	3d	3d	44d 2h	0		0
6	**Background Research**	**5 Oct**	**15 Oct**	**19d**	**10d**	**74d**	**0**		0
6.1	Camera selection	5 Oct	7 Oct	3d	3d	81d	0		0
6.2	Payload controller (e.g. microcontroller/FPGA) selection	12 Oct	15 Oct	4d	4d	42d 7h	0		0
6.3	Communication protocols	5 Oct	7 Oct	3d	3d	81d	0		0
6.4	Image processing/compression	5 Oct	11 Oct	6d	6d	43d 7h	0		0
6.5	Programming language	5 Oct	7 Oct	3d	3d	81d	0		0
7	**Prepare presentation**	**22 Oct**	**10 Jan**	**10d**	**69d**		**0**		0
7.1	Progress seminar 1	22 Oct	25 Oct	3d	3d	66d	0		0
7.2	Progress seminar 2	25 Nov	28 Nov	3d	3d	37d	0		0
7.3	Final presentation	6 Jan	10 Jan	4d	4d		0		0
8	Report Writing	30 Nov	10 Dec	10d	10d	26d	0		0
9	Whole System Testing	26 Nov	1 Dec	5d	5d	34d	0		0


Name
Work
Week 41, 2011
Week 42, 2011
Week 43, 2011
Week 44, 2011
Week 45, 2011
Week 46, 2011
Week 47, 2011
Week 48, 2011
Week 49, 2011
Week 50, 2011
Week 51, 2011
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Deadlines
Opening lecture
First progress seminar
Second progress seminar
Final report
Final presentation
Acquire Equipment 50d 1h
Acquire working camera 23d 1h
Acquire Schematic Parts 17d
Acquire final payload controller 5d
Acquire autopilot and basestation 5d
Payload Module 71d 2h
Camera module communication 8d 4h
Prototype camera module communication 5d 4h
Payload controller <-> camera module communication 3d
Image Encoding and Transmission 39d
Basic JPEG encoding 5d
Payload controller implementation 5d
Custom (compressed) encoding 10d
Matlab algorithm prototype 10d
JPEG Reordered Payload controller implementation 24d
Huffman Table Build 2d 4h
Huffman Encoding/Decoding 2d 4h
Extract JPEG header and payload data 19d
Payload module construction 23d 6h
Payload breadboard prototype and debugging 5d 6h
Payload stripboard construction and debugging 3d
Payload PCB design 5d
Aquiring payload PCB 5d 6h
Payload PCB construction and debugging 4d 1h
Image recieve/viewing software 15d 2h
TCP/IP Basestation Communications 7d
Basic dummy server communication 4d
Basestation communications 3d
Decode Image 2d
Basic image decoding 2d
UI 6d 2h
Functional user interface 6d 2h
Integration 3d
Payload controller to autopilot/base station 3d
Background Research 19d
Camera selection 3d
Payload controller (e.g. microcontroller/FPGA) selection 4d
Communication protocols 3d
Image processing/compression 6d
Programming language 3d
Prepare presentation 10d
Progress seminar 1 3d
Progress seminar 2 3d
Final presentation 4d
Report Writing 13d 1h
Whole System Testing 5d


Week 52, 2011
Week 1, 2012
Week 2, 2012
Week 3, 2012
Week 4, 2012
Week 5, 2012
Week 6, 2012
Week 7, 2012
Week 8, 2012
Week 9, 2012
Week 10, 2012
Week 11, 2012
Week 12, 2012
Week 13, 2012
23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to	% Complete
1	**Deadlines**	**5 Oct**	**11 Jan**		**84d**		**0**		0
1.1	Opening lecture	5 Oct	5 Oct	N/A	N/A	84d	0		0
1.2	First progress seminar	26 Oct	26 Oct	N/A	N/A	66d	0		0
1.3	Second progress seminar	30 Nov	30 Nov	N/A	N/A	22d 5h	0		0
1.4	Final report	15 Dec	15 Dec	N/A	N/A	23d	0		0
1.5	Final presentation	11 Jan	11 Jan	N/A	N/A		0		0
2	**Acquire Equipment**	**5 Oct**	**1 Nov**	**50d 1h**	**23d 1h**	**60d 6h**	**0**		0
2.1	Acquire working camera	5 Oct	1 Nov	23d 1h	23d 1h	35d 6h	0		0
2.2	Acquire Schematic Parts	5 Oct	24 Oct	17d	17d	43d	0		0
2.3	Acquire final payload controller	17 Oct	21 Oct	5d	5d	45d	0		0
2.4	Acquire autopilot and basestation	5 Oct	10 Oct	5d	5d	72d	0		0
3	**Payload Module**	**5 Oct**	**5 Dec**	**71d 2h**	**53d**	**27d**	**0**		0
3.1	**Camera module communication**	**1 Nov**	**10 Nov**	**8d 4h**	**8d 4h**	**48d 2h**	**0**		0
3.1.1	Prototype camera module communication	1 Nov	7 Nov	5d 4h	5d 4h	35d 2h	0		0
3.1.2	Payload controller <-> camera module communication	7 Nov	10 Nov	3d	3d	35d 2h	0		0
3.2	**Image Encoding and Transmission**	**5 Oct**	**22 Nov**	**39d**	**42d**	**38d**	**0**		0
3.2.1	**Basic JPEG encoding**	**10 Nov**	**16 Nov**	**5d**	**5d**	**43d 2h**	**0**		0
3.2.1.1	Payload controller implementation	10 Nov	16 Nov	5d	5d	35d 2h	0		0
3.2.2	**Custom (compressed) encoding**	**5 Oct**	**15 Oct**	**10d**	**10d**	**70d**	**0**		0
3.2.2.1	Matlab algorithm prototype	5 Oct	15 Oct	10d	10d	70d	0		0
3.2.3	**JPEG Reordered Payload controller implementation**	**1 Nov**	**22 Nov**	**24d**	**19d**	**38d**	**0**		0
3.2.3.1	Huffman Table Build	3 Nov	5 Nov	2d 4h	2d 4h	49d 4h	0		0
3.2.3.2	Huffman Encoding/Decoding	5 Nov	8 Nov	2d 4h	2d 4h	50d	0		0
3.2.3.3	Extract JPEG header and payload data	1 Nov	22 Nov	19d	19d	38d	0		0
3.3	**Payload module construction**	**1 Nov**	**5 Dec**	**23d 6h**	**29d 6h**	**27d**	**0**		0
3.3.1	Payload breadboard prototype and debugging	1 Nov	7 Nov	5d 6h	5d 6h	36d	0		0
3.3.2	Payload stripboard construction and debugging	8 Nov	10 Nov	3d	3d	48d	0		0
3.3.3	Payload PCB design	8 Nov	12 Nov	5d	5d	36d	0		0
3.3.4	Aquiring payload PCB	24 Nov	30 Nov	5d 6h	5d 6h	27d 1h	0		0
3.3.5	Payload PCB construction and debugging	30 Nov	5 Dec	4d 1h	4d 1h	27d	0		0
4	**Image recieve/viewing software**	**5 Oct**	**25 Nov**	**15d 2h**	**45d**	**35d**	**0**		0
4.1	**TCP/IP Basestation Communications**	**5 Oct**	**13 Oct**	**7d**	**8d**	**72d**	**0**		0
4.1.1	Basic dummy server communication	5 Oct	8 Oct	4d	4d	76d	0		0
4.1.2	Basestation communications	11 Oct	13 Oct	3d	3d	72d	0		0
4.2	**Decode Image**	**16 Nov**	**18 Nov**	**2d**	**2d**	**41d 2h**	**0**		0
4.2.1	Basic image decoding	16 Nov	18 Nov	2d	2d	35d 2h	0		0
4.3	**UI**	**18 Nov**	**25 Nov**	**6d 2h**	**6d 2h**	**35d**	**0**		0
4.3.1	Functional user interface	18 Nov	25 Nov	6d 2h	6d 2h	35d	0		0
5	**Integration**	**16 Nov**	**19 Nov**	**3d**	**3d**	**44d 2h**	**0**		0
5.1	Payload controller to autopilot/base station	16 Nov	19 Nov	3d	3d	44d 2h	0		0
6	**Background Research**	**5 Oct**	**15 Oct**	**19d**	**10d**	**74d**	**0**		0
6.1	Camera selection	5 Oct	7 Oct	3d	3d	81d	0		0
6.2	Payload controller (e.g. microcontroller/FPGA) selection	12 Oct	15 Oct	4d	4d	45d	0		0
6.3	Communication protocols	5 Oct	7 Oct	3d	3d	81d	0		0
6.4	Image processing/compression	5 Oct	11 Oct	6d	6d	46d	0		0
6.5	Programming language	5 Oct	7 Oct	3d	3d	81d	0		0
7	**Prepare presentation**	**22 Oct**	**10 Jan**	**10d**	**69d**		**0**		0
7.1	Progress seminar 1	22 Oct	25 Oct	3d	3d	66d	0		0
7.2	Progress seminar 2	25 Nov	28 Nov	3d	3d	37d	0		0
7.3	Final presentation	6 Jan	10 Jan	4d	4d		0		0
8	Report Writing	30 Nov	15 Dec	13d 1h	13d 1h	22d 6h	0		0
9	Whole System Testing	6 Dec	10 Dec	5d	5d	26d	0		0

Appendix I

User Documentation

UAV Image Viewer User Guide

GDP Group 18

December 15, 2011

1 Before Starting the UAV Image Viewer Program

In order to connect the payload and camera to the UAV, the user needs to connect an ethernet cable between the Autopilot and Payload modules. The Autopilot powers the payload module, as well as providing a medium for the two modules to communicate. The user should not have to connect the camera manually as it should already be connected.

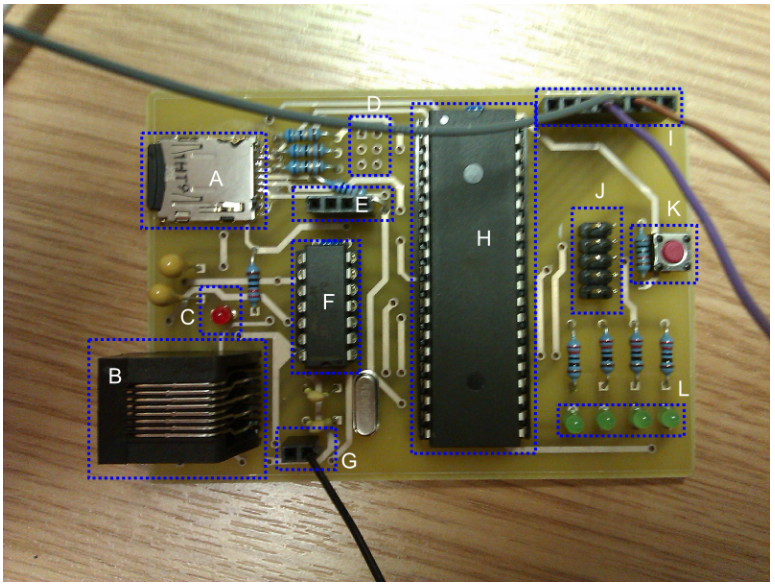


Figure 1: The Payload Module

Please check the following features on the payload once it has been connected to the Autopilot and the Autopilot is powered:

- The Power LED (C) should be lit.

- A microSD card should be present in socket (A). It is preferred if this card is SDSC and not SDHC or SDXC standard. Please make sure this card has enough memory to take the images.
- Images stored locally are up to approximately 30kB in size, and are stored in the filename test$NUMBER.jpg (replace $NUMBER with an integer between 0 and 32768)
- If you are using more payload modules than this, and they are sourcing more than 150mA from the Autopilot, please power this payload module externally. On Header G, connect 3.3V to the left-hand, and GND to the right-hand socket.
- (K) is a reset button.
- In case the camera is disconnected, from L-R on (E), are 3.3V, GND, Camera TX and Camera RX. Please refer to the Camera Schematics. [1]

Full schematics, PCB layout and Gerber Files are available in our official GitHub repository [2], in the schematics/ file.

On the ground side, the Autopilot's receiver module needs to be connected to a USB socket. Figure 2 shows the diagram of how to connect the payload to the UAV and the UAV data receiver to the computer.

For the UAV Image Viewer to work correctly, the receiver must first be connected to the ground station. In order to connect to the receiver, all the hardware has to be connected as described above. The GCS station then connects to the Autopilot receiver by clicking on the Connect button. If the program does not connect to the Autopilot automatically, the user can select it from the combo box. Then check the "Enable Network Server" box. Figure 3 shows a screen shot of the GCS program. If the autopilot does not appear, the user should check the connection of the UAV and make sure that the driver of the USB has already installed.

To install the USB driver:

Run "Device Manager" → Right click on the UAV → Click on the update driver→ Find the driver (.inf file) → install it → UAV is ready to use.

2 UAV Image Viewer Ground Station

The file tab in the top left corner allows the user to open a file, then display it in the picture box, and save this image to a specified directory. It also gives the option to exit the program. Figure 5 shows the diagram of the file tab.

[1] `http://www.4dsystems.com.au/downloads/micro-CAM/Docs/uCAM-DS-rev4.pdf`

[2] `https://github.com/uavcamera/uavcamera`

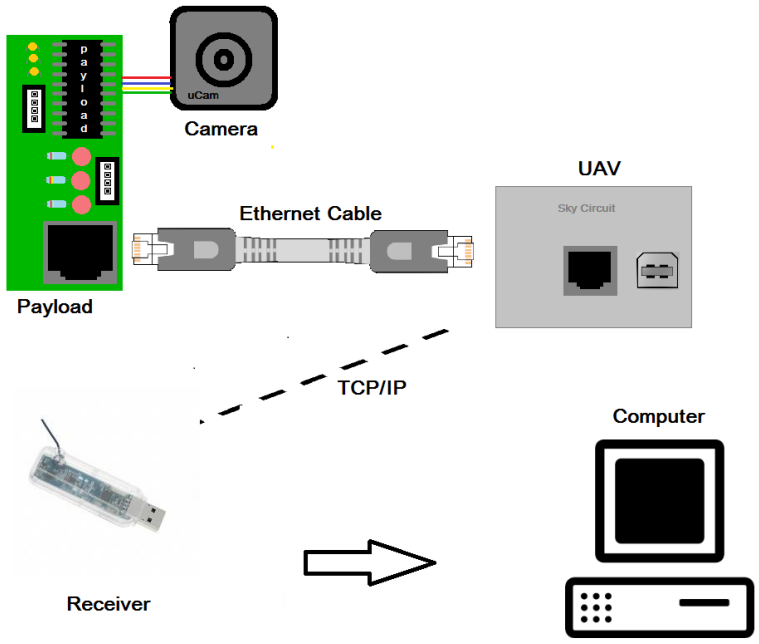


Figure 2: The Clip Art of the Hardware Connection Diagram

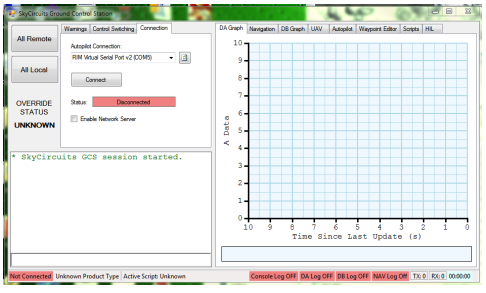


Figure 3: The Ground Control Station Software

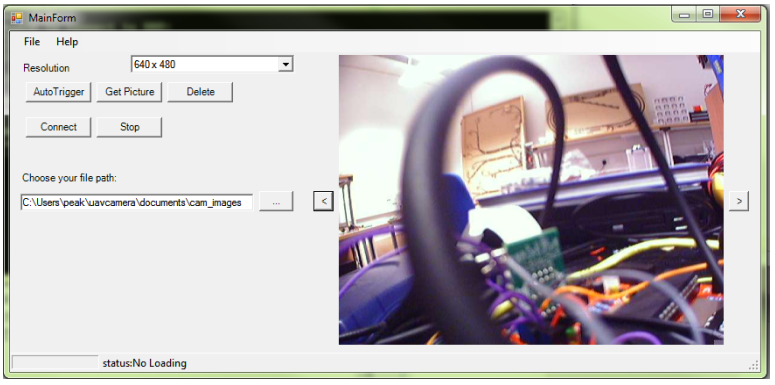


Figure 4: The UAV Image Viewer Software

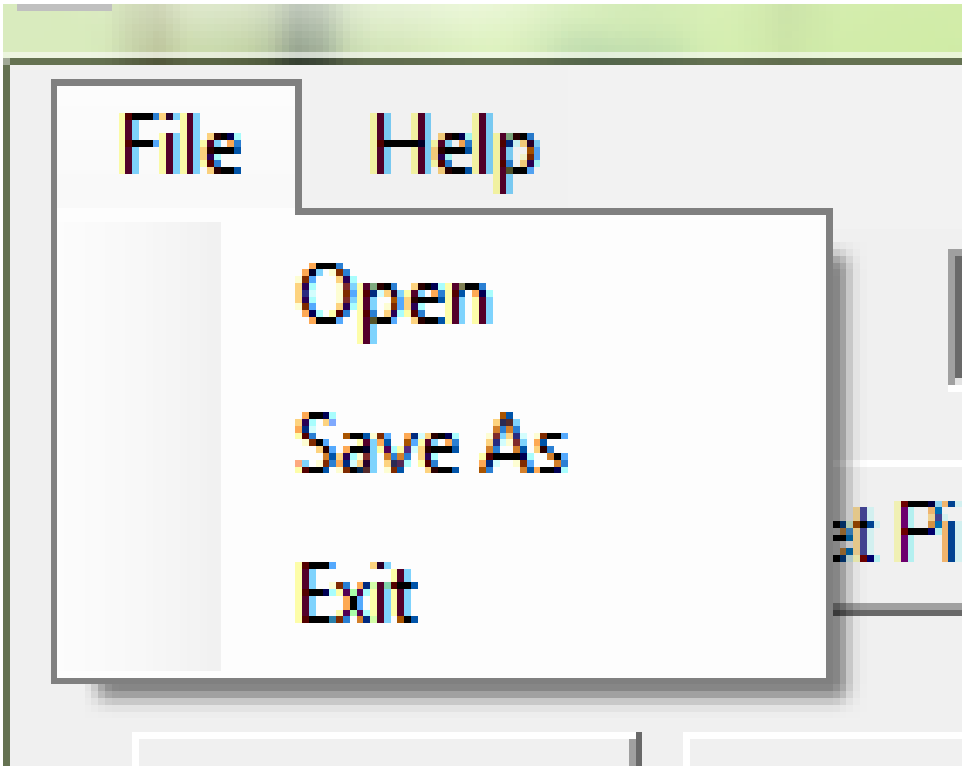


Figure 5: Open Tab

The user can choose the resolution from the drop-down menu in the top left. There are four options - all of the resolutions that the camera can take. The lower the resolution, the quicker the image downloads.

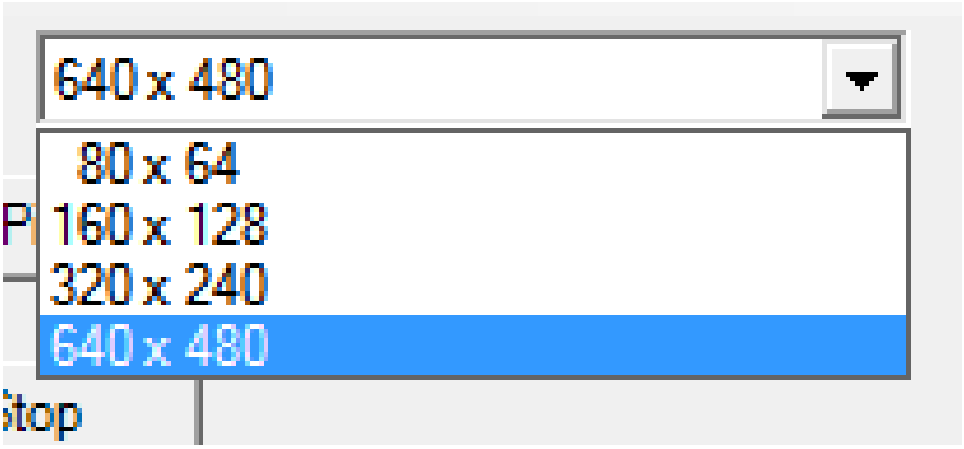


Figure 6: The Resolution Option

Figure7 shows some buttons present in our application. The **AutoTrigger** button allows the user to take multiple photos and save them on the on-board microSD card. The **Get Picture** button will allow the user to take a picture and send it directly to the ground station. The default file name of the picture will be uavPictureAtyyyy-MM-dd_hh-mm-ss-tt.jpg. The photo will be saved in the file path specified in the text box as shown in figure 8. The name depends on the system date and time at the time of taking picture. It will take about 10-20 seconds from clicking "Get Picture" to displaying the image.

Figure 7: The Buttons Options of the Image Picture Viewer

The left and right buttons near the picture allow the user to change the photo displayed to any locally stored jpeg image in the selected file path. The button is shown in figure 9.

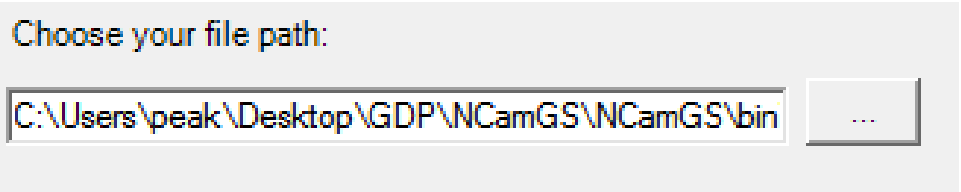


Figure 8: The file path text box

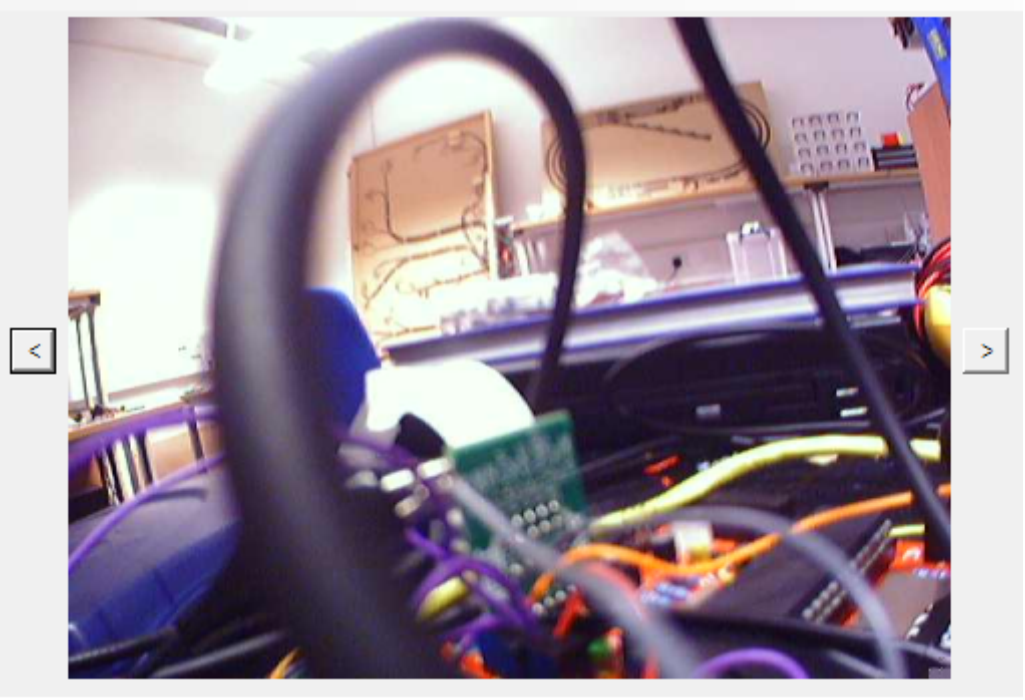


Figure 9: The Picture Box with left and right buttons

The progress bar shows the proportion of the image that has arrived at the ground station. The status label will tell what signal is send and receiving so the user can keep track of what is going on in the payload.

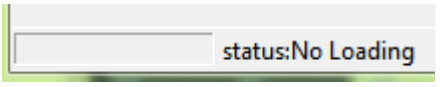


Figure 10: The progress bar and the status text

2.1 Take Picture

The use can simply click on the Get Picture button in the program. Figure 7 shows where the button is.

Bibliography

[1] SkyCircuits Ltd. Web. 13th Dec. 2011 http://www.skycircuits.com/

[2] SULSA *The "Southampton University Laser Sintered Aircraft* Web. 13th Dec. 2011 http://www.soton.ac.uk/~decode/index_files/Page804.htm

[3] New Scientist *3D printing: The world's first printed plane* Web. 13th Dec. 2011 http://www.newscientist.com/article/dn20737-3d-printing-the-worlds-first-printed-plane.html

[4] SkyCircuits *SC2 autopilot module* Web. 14th Dec. 2011 http://www.skycircuits.com/autopilot/sc2

[5] Tortorella, Richard. *Image Doctoring: JPEG Encoding and Analysis*. Rep. NARCAP, May 2009. Web. 11 Oct. 2011. http://www.narcap.org/reports/narcap_IR-01_DigHoaxing.pdf.

[6] Japan Electronics and Information Technology Industries Association (JEITA). *Digital Still Camera Image File Format Standard (Exchangeable Image File Format for Digital Still Cameras: Exif)* Version 2.2. Tech. Japan Electronic Industry Development Association (JEIDA), 12 June 1998. Web. 14 Oct. 2011 http://www.exif.org/Exif2-2.PDF.

[7] Netravali, Arun N., and Barry G. Haskell. *Digital Pictures: Representation and Compression*. New York: Plenum, 1988. Print.

[8] *JPEG File Layout and Format*. Original URL: http://www.funducode.com/freec/Fileformats/format3/format3b.htm (defunct). DCube Software Technologies, 5 July 2002. Web. 28 Oct. 2011. http://class.ee.iastate.edu/ee528/Reading%20material/JPEG_File_Format.pdf

[9] WinZip International LLC. *JPEG Compression*. Tech. Version 1.0. 11 Sept. 2008. Web. 28 Oct. 2011. http://www.winzip.com/wz_jpg_comp.pdf.

[10] Poynton, Charles. *Chroma Subsampling Notation*. Charles Poynton. 24 Jan. 2008. Web. 31 Oct. 2011. http://poynton.com/PDFs/Chroma_subsampling_notation.pdf.

[11] Kerr, Douglas A. *Chrominance Subsampling in Digital Images.* Rep. Issue 2. 3 Dec. 2009. Web. 31 Oct. 2011. http://dougkerr.net/pumpkin/articles/Subsampling.pdf.

[12] Hass, Calvin. *ImpulseAdventure - Digital Photography Articles.* 2008. Web. 22 Nov. 2011. http://www.impulseadventure.com/photo/.

[13] 4D Systems. *uCam Serial JPEG Camera Module Data Sheet* 8 Jul. 2010. http://www.4dsystems.com.au/downloads/micro-CAM/Docs/uCAM-DS-rev4.pdf.

[14] Arduino. *SoftwareSerial Library.* Web. 9th Dec. 2011 http://www.arduino.cc/en/Reference/SoftwareSerial.

[15] Arduino. *SD Library.* Web. 10th Dec. 2011 http://www.arduino.cc/en/Reference/SD.

[16] Arduino. *Serial Library.* Web. 1st Dec. 2011 http://www.arduino.cc/en/Reference/Serial.

[17] Atmel. *ATMega644P* Web. 10th Dec. 2011 http://www.atmel.com/dyn/products/product_card.asp?part_id=3896

[18] Sanguino. *What is Sanguino?* Web. 10th Dec. 2011 http://sanguino.cc/.

[19] mbed. *Welcome to mbed!* Web. 10th Dec. 2011 http://mbed.org/.

[20] Olimex. *Olimexino-stm32 development board Users Manual* Web. Oct. 2011 http://www.olimex.com/dev/DUINO/OLIMEXINO-STM32/OLIMEXINO-STM32.pdf.

[21] Github. *Our "Official", Central Repository* Web. 11th Dec. 2011 https://github.com/uavcamera/uavcamera

[22] *Spirit Circuits Go Naked* Web. 11th Dec. 2011 http://www.spiritcircuits.com/services/go-naked

[23] C.D. CLEANU, V. TIPONU, I. BOGDANOV, S. IONEL, I. LIE *C# and .NET Framework for uC communication protocol Proceedings of the 11th WSEAS International Conference on COMPUTERS, Agios Nikolaos, Crete Island, Greece, July 26-28, 2007 implementation* 2008. Web. 22 Nov. 2011. www.wseas.us/e-library/conferences/2007cscc/papers/561-338.pdf.

[24] Kakit Tsui *Ad Hoc Network:Generic USB Device Driver Development* (2003) http://crisp.ece.cornell.edu/mengproj/alan_report.doc.

[25] Keith Clark,Peter J. Robinson,Richard Hagen.(2001) Multi-threading and message communication in Qu-Prolog.*Theory and Practice of Logic Programming* . 1 (3), p283-301

[26] Keith Clark,Peter J. Robinson,Richard Hagen.(2004) *Beginning .NET game programming in C#.* United States of America: Apress.

[27] Xiaoyun Xie.(2004) *Distributed Objects System in C#.* Rochester, United States of America.

[28] David B. Makofske, Michael J. Donahoo, Kenneth L. Calvert.(2004) *TCP/IP sockets in C# practical guide for programmers.* Amsterdam: Elsevier.

[29] Norman Matloff. (2009) *Tutorial on Network Programming with Python.* California

[30] Guido van Rossum. (1994) *Extending and Embedding the Python Interpreter.* The Netherlands

[31] M. F. SANNER. (1999) *PYTHON: A PROGRAMMING LANGUAGE FOR SOFTWARE INTEGRATION AND DEVELOPMENT.* California

[32] Kenneth L. Calvert, Michael J. Donahoo. (2008) *TCP/IP Sockets in Java Practical Guide for Programmers.* Kentucky:Elsevier

[33] Elliot R. Harold, Michael J. Donahoo. (1997) *Java Network Programming.* USA:O'Reilly, p35

[34] 4D Systems. Web. 13th Dec. 2011. http://www.4dsystems.com.au/prod.php?id=82.

[35] Robert Harkness and Malcolm Crook and David Povey.(2007) *Programming Review of Visual Basic.NET for the Laboratory Automation Industry.* Journal of the Association for Laboratory Automation vol.12, p.25-32. http://www.sciencedirect.com/science/article/pii/S1535553506005302.

[36] GNU *The GNU Public License version 3* Web. 14th Dec. 2011 http://www.gnu.org/licenses/gpl-3.0.html

[37] Creative Commons *Attribution-ShareAlike 2.0 UK: England & Wales (CC BY-SA 2.0)* Web. 14th Dec. 2011. http://creativecommons.org/licenses/by-sa/2.0/uk/

[38] GNU *The GNU Free Documentation License version 1.3* Web. 14th Dec. 2011. http://www.gnu.org/copyleft/fdl.html

[39] D.A. Huffman *A Method for the Construction of Minimum-Redundancy Codes.* Proceedings of the I.R.E. Sept. 1952. p1098-p1102

[40] Atmel.*megaAVR Parameters* 15th Dec. 2011. http://www.atmel.com/dyn/products/param_table.asp?category_id=163&family_id=607&subfamily_id=760

[41] *A25L040-U - 16Mbit Low Voltage, Serial Flash Memory With 100MHz Uniform 4KB Sectors - AMIC Technology* 8 Jul. 2010. http://pdf1.alldatasheet.com/datasheet-pdf/view/244476/AMICC/A25L040-U.html.

Glossary

Terms

YCbCr Colour space used to represent JPEG images. Composed of three values:

- **Y** luma component.
- **Cb** blue-difference chroma component.
- **Cr** red-difference chroma component.

4:x:y Chroma subsampling notation.

- **4** Luma horizontal (and vertical) sampling reference.
- **x** Cb and Cr horizontal sampling factor.
- **y** Cb and Cr horizontal sampling factor. If 0, indicates 2:1 vertical subsampling for both Cb and Cr.

Abbreviations

IC Integrated Circuit

RF Radio Frequency

TCP/IP Transmission Control Protocol/Internet Protocol

PCB Printed Circuit Board

RGB Red,Green and Blue

AC Alternating Current

DC Direct Current

JPEG Joint Photographic Experts Group, creators of the jpeg compression method. Used interchangeably with the jpeg image type.

EXIF EXchangeable Image File format for digital still cameras.

MCU Minimum Coded Unit

DPCM Differential Pulse Code Modulation

RLE Run-Length Encoding

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

DCT Discrete Cosine Transform

WHT WalshHadamard Transform

KLT Karhunen-Loève Transform

SOI Start Of Image

SOF Start Of Frame

DHT Define Huffman Table(s)

HT Huffman Table

DQT Define Quantization Table(s)

QT Quantization Table

SOS Start Of Scan

EOI End Of Image

UART Universal Asynchronous Serial Receiver/Transmitter

LOS Line Of Sight

USART Universal Synchronous/Asynchronous Serial Receiver/Transmitter

SD(SC) card Secure Digital (Standard Capacity) card (Capacity <= 2GiB

SDHC Secure Digital High Capacity (2GiB <= Capacity <= 32GiB)

SDXC Secure Digital eXtended Capacity (32GiB <= Capacity <= 2TiB)

UAV Unmanned Aerial Vehicle

GPLv3 GNU Public License version 3

CC BY-SA Creative Commons Attribution ShareAlike License

FDL GNU Free Documentation License

BGA Ball Grid Array

TSOP Thin Small-Outline Package

IDE Integrated Development Environment

SPI Serial Peripheral Interface Bus

JTAG Joint Test Action Group