Oregon Institute of Technology

Automated Smart Blinds

Lessons Learned

*Authored by*:

Andrew Deraita

Michael Roberts

Chad Revel

# Table of contents

[Table of contents 1](#_Toc42445602)

[Introduction 2](#_Toc42445603)

[Technical issues 2](#_Toc42445604)

[- OV2640 FIFO limit 2](#_Toc42445605)

[- JPEG Compression 3](#_Toc42445606)

[- Image Format 3](#_Toc42445607)

[- WiFi Image Transfer 4](#_Toc42445608)

[Plan changes 5](#_Toc42445609)

[schedule changes 5](#_Toc42445610)

[conclusion 6](#_Toc42445611)

## Introduction

Many issues were discovered and solved during development of the Automated Smart Blinds. While most problems were easy to solve, one module produced a series of problems: the camera. Issues with this module cascaded, starting with image size issues, which lead to communication issues, and eventually leaving us just short of our initial goals of sending images to the web service. The issues we had with the camera are detailed in the technical section below but are summarized as follows.

1. Raw images too large. 8Mhz SPI interface means camera FIFO cannot be read fast enough to take pictures large enough for person recognition.
2. Changing to JPEG for compressed images required interfacing with the sensor itself, which meant an I2C interface driver had to be created and memory maps for the sensor configuration were constructed.
3. Image data was not polling correctly from the SPI interface, losing some of the header metadata.
4. Failure to upload to the server due to the image size being too large for single packets and could not form packets correctly to get image data to the server.

## Technical issues

### OV2640 FIFO limit

The OV2640 camera module has a built-in 8-bit microcontroller with 512 bytes of memory. This means, the largest images that the camera can capture have to be stored in this memory space. Since we needed images of resolution at least 1024 x 768 for human face recognition, raw images were too large to fit.  
While this buffer space can be emptied as it is filled by the camera sensor, the 8MHz SPI interface could not remove image data from the buffer faster than the image is created. Due to this, we had to enable JPEG compression in the sensor itself, to supply smaller sized images.

### JPEG Compression

To enable JPEG compression, the OV2640 unit must be communicated directly, instead of going through the SPI interface with the microcontroller. This was not originally part of our schedule, so time to research and implement I2C drivers took additional time. Once we had an I2C interface, we were able to select compression options and set image sizes so they could be communicated successfully back to the PIC32MZ.

### Image Format

Now that the camera was configured and images could be transferred to the PIC32MZ, it was time to determine if the data was actually JPEG image data. All files have metadata attached to the front and back of the file, so it was necessary to ensure the files being received are valid JPEG files. We discovered that the image data was mostly correct, but a minor issue in the I2C driver caused the first byte of data to be discarded, corrupting the data.

Once solved, we can clearly see that the file has all appropriate header information to indicate it is a JPEG file. See [Figure 1](https://docs.google.com/document/d/1b3Q9muHGuk2ZljEyvhwpfLNopaOsRYFGu83mDbrUaZk/edit#bookmark=id.x3n0wn262vuj) for an example of correct header information, confirmed by [fileformat.com](https://wiki.fileformat.com/image/jpeg/).

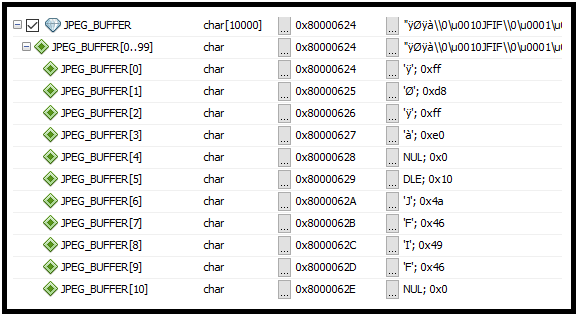


Figure 1: Debug view of JPEG\_BUFFER, showing JPEG header information.

### WiFi Image Transfer

The final part of the camera is transferring the image from the PIC32MZ through the WINC1500 to the Web Service. The initial plan was to embed the image file into a SOAP package but were unable to form the packets correctly. We switched to using a HTTP GET request, but then ran into an issue with fragmenting the packets.

The largest packet size for transmissions to the server were 1400 bytes, which could not fit some of our image files. Due to this, we had to fragment packets and send multiple transmissions for a single image. We were never able to get this to work correctly. We could upload an empty image, but the data stream being sent for the JPEG file was not formatted correctly.

A major complication with us determining the issues with our WiFi communication was our inability to capture packets sent over the WiFi. Due to inability to determine the actual bytes on the wire, we had to abandon this aspect of our project to meet deadlines. The images are currently stored on the PIC32MZ but will not be sent over to the website when a proximity alarm is triggered.

## Plan changes

Due to the requirement to implement unexpected communication protocols, Drew was required to go over his allotted time to successfully capture images. During week 20 of development, Drew was behind on his other testing and implementation phases which cascaded to a problem for the other members. For instance, the website overhaul was originally planned for the second semester but was shelved as more effort was needed to complete the camera modules. Due to the team's strong communication and teamwork, we were able to overcome all but the image transfer barrier. Images are taken in the correct format locally to the device but are not viewable on the website.

## schedule changes

As we were using a Gantt Chart to track our progress through this project, it became clear that Drew would not meet his target goals for Camera implementation. After extending the development time an additional two weeks, the group decided to furlough the image transfer and focus on completing the remainder of the project. Extra meetings were scheduled to address this issue, especially with transferring images to the web server. The entire team essentially lost two weeks of development due to lack of research into the camera.

## conclusion

Many lessons we learned throughout this project, not just related to the camera module. Each member had to overcome unexpected obstacles involving their implementations. Due to these challenges, we have decided there are a few important points we would pass down to future junior project teams.

1. Do more research than you think is necessary. Order and test all parts as best as possible before deciding on a component.
2. Plan additional deadtime during final stages of development. Regardless of how you plan, issues will arise. We recommend scheduling at least a week or two of review before the second semester’s final deadline.
3. Ensuring modules do not need any additional hardware to implement. This means checking target voltages and communication speeds. Triple check size constraints of assembly, giving extra space as a buffer for any potential alterations.
4. Manage the expectations of your project. This means having a clear goal for each module and knowing when to admit defeat and re-organize that aspect of the project.