Contents

Chapter 1

Firmware (DGMROB001)

1.1 Subsystem introduction

This subsystem deals with the development of firmware for the microcontrollers in both the camera/transmitter and receiver modules. The fundamental goals of this subsystem were to enable the camera/transmitter module to take photographs of the red-winged starlings, and to transmit this photographs to the receiver module without disturbing the birds' nests. The source code for this subsystem is available in the project repository on GitHub.

Section 1.2 presents the subsystem's user requirements, interpretted functional requirements, and the corresponding system specifications. Section 1.3 discusses high-level design decisions that informed the subsystem development process. Section 1.4 details the design process and the functionality of the final implementation. Finally, section 1.5 shows the acceptance test procedures, proving that the user requirements have been adequately met.

The system provided from the hardware subsystem includes an ESP32-Cam for the camera/transmitter module, and a Raspberry Pi Pico W for the receiver. The ESP32-Cam module includes an onboard SD card reader and camera, and integrated WiFi. Externally, it is connected to PIR movement sensors, and an RF receiver (for remote waking from the receiver device). The PIR sensors and RF receiver utilise the same pins as the SD card; the external peripherals must thus be disabled (via a switch connected to a GPIO pin) before the SD card can be initialised. The receiver module includes onboard WiFi. Externally, it is connected to a 16x2 I2C LCD, an SPI SD card reader, and and four pushbuttons for user input.

1.2 Requirements and specifications

User requirements (UR) where determined from discussions with the red-winged starling researchers [1]. The user requirements were interpretted to form functional requirements (FR) for the subsystem, and subsystem specifications (SS). These are reflected in Table 1.1

UR/FR/SS	Description
UR1	Triggered with movement.
FR1	Image capture triggered quickly after PIR sensor reading to see the object that moved.
FR2	Image capture fast enough to see object that moved.
SS1	Image capture triggered on PIR sensor high.
SS2	Image captures within 1 second of PIR trigger.
UR2	Data access without disturbing nests.
FR3	A wireless communication protocol must be set up to transfer camera/transmitter data to the receiver.
SS3	Appropriate two-way communication protocol using WiFi implemented.
SS4	System transmits metadata from the camera/transmitter to the receiver.
SS5	System transmits a sample image from the camera/transmitter to the receiver.
SS6	System transmits all captured footage from the camera/transmitter to the receiver.
UR3	Repeated images or video footage.
FR4	Camera must capture multiple time per trigger or take video footage.
SS7	Module captures images a configurable number n images each time it is triggered.
UR4	Access to images in real time.
FR5	Tranmission from the camera/transmitter module must be available on-demand from the receiver module.
SS8	Transmission from the camera/transmitter to the receiver is triggered by a high reading from the RF linker.
UR5	Data gathering over 7-week breeding season without frequent camera/transmitter battery changes.
FR6	Camera/transmitter module must utilise power-saving modes where possible.
SS9	Camera/transmitter module enters deep sleep mode after finishing image capture or data transmission.
UR6	Configurable image quality and trigger frequency.
FR7	Camera/transmitter module must include remotely updateable configuration options.
SS10	Camera/transmitter module reads a configuration file and loads the appropriate configuration values.
SS11	New configuration file can be sent from receiver to camera/transmitter which is loaded on next boot.
FR8	Device takes good quality photographs.
SS12	Default device settings allow for photographs of adequate quality on a qualitative assessment.
UR7	Deployment of multiple camera/transmitter modules for different nests.
FR9	Receiver must hanlde separate image storage and configuration options for many camera/transmitter modules.
SS13	Each camera/transmitter module stores and transmits a unique device ID.
SS14	Receiver module stores a separate folder for data relating to separate deployed modules.
FR10	Receiver device has appropriate functionality to allow user interaction.
SS15	Device features a menu which can be controlled by the user.
SS16	Menu shows appropriate feedback to the user at all times.
SS17	User interface is easy to understand.

Table 1.1: Requirement and specification analysis for the firmware subsystem

1.3 High-level design decisions

This section details high-level design decisions that were made prior to any implementation and testing. These decisions determined key aspects of the development process.

1.3.1 Development environment

Both the ESP32 used for the camera/transmitter and the Raspberry Pi Pico W used for the receiver support a variety of code development environments, including their own native C/C++ frameworks, Arduino, MicroPython and CircuitPython. A summary of the support and features of the various frameworks is presented in Table 1.2.

Device	Environment	Library support	Online support-/userbase	Abstraction level
ESP32	*C/C++ with ESP-IDF*	Extensive	Extensive	Low
	Arduino	Extensive	Extensive	Moderate
	MicroPython	Limited	Minimal	High
	CircuitPython	Limited	Minimal	High
Raspberry Pi Pico W	*Native C/C++ SDK*	Moderate	Moderate	Low
	Official Arduino port	Moderate	Moderate	Moderate
	Community Arduino port	Extensive	Extensive	Moderate
	MicroPython	Extensive	Extensive	High
	CircuitPython	Extensive	Moderate	High

Table 1.2: Summary of common coding environments for the available microcontroller

With the exception of MicroPython and CircuitPython for the ESP32, the library and online support is likely sufficient for this application. Abstraction level is thus a key consideration. Frameworks with low abstraction levels are typically used in professional applications that require fine control over the device hardware. This often makes using these frameworks much more time consuming, without extensive prior experience with the framework. On the other hand, high abstraction levels sometimes abstract away desired hardware control. A moderate abstraction level is preferred, to strike a balance between these considerations.

Arduino thus as a clear choice for the ESP32. Although MicroPython and CircuitPython may provide all the functionality needed for the the Raspberry Pi Pico W, Arduino was chosen such that the entire project was coded in the same framework, thus requiring knowledge of only the one framework in the current and future iterations of the project.

The Raspbery Pi Pico W supports two Arduino ports - the 'official' one, and a community maintained one. Contrary to what might be expected, the community port offers better library and online/user support, as it has been in development longer; thus, the community port was used.

1.3.2 File transfer method

Meeting SS3, SS4, SS5, and SS6 requires data transmission over WiFi. For this, numerous protocols exist, including encrypted and unencrypted protocols. An encrypted protocol was not necessary for this application, as the devices did not transmit sensitive data. The encryption involved inherently makes these protocols more complex and often less efficient; as such, the choices were narrowed to unencrypted protocols.

Standard unencrypted protocols include User Datagram Protocol (UDP), Tranmission Control Protocol (TCP), File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP). UDP and TCP are low-level protocols; using a standard higher-level prtocol such as FTP or HTTP is more appropriate unless very fine control of data transmission is needed. FTP and HTTP both operate on top of TCP. Of FTP and HTTP, HTTP is much more widely used owing to its application in the World Wide

Web, and thus has better library support and documentation. HTTP was thus chosen as the preferred option.

1.3.3 Server and client location

Communication over HTTP is based on a client-server model, where a client sends a request to a known host. In the current iteration of the project, either device could viably be the server or the client, given HTTP supports two-way communication.

However, in future iterations of the project, there are particular merits to hosting the server on the receiver device. The server requires a known IP address such that the client knows where to direct it's requests. Once a client has connected to the server, the server sends it's response directly back to whatever client responded to it. The network settings of the UCT campus WiFi do not allow devices to set static IP addresses, meaning it is challenging to host the server on the campus WiFi. If the camera/transmitter acts as the client, it does not need a known IP address when connected to campus WiFi, and could connect to a remotely hosted server.

Aditionally, hosting the server on the receiver device allows for traditional web browsers to remotely connect as a client. This could allow for applications such as updating configuration or viewing stored images on the receiver device from a smartphone or laptop in future iterations of the projects.

1.3.4 Configuration file

In line with SS10, the camera/transmitter module must read a configuration file. There were three options for storing this configuration file: flash memory, EEPROM, or a file on the SD card. Storing configuration on the SD card was chosen, since it enables easy deployment of a new module with a custom configuration, simply by loading the appropriate configuration file onto the SD card before the module is deployed.

There are numerous ways to store configuration files, including the XML, JSON, YAML, and INI file formats. All of these formats are plaintext files, and each could be adequately adapted to be used in the project with the correct text parsing. However, of the available options, JSON stands out, since it is a human-readable format, making it easier to create a new configuration file without any external tool. It is also well supported on the Arduino framework via the ArduinoJson library [2].

1.3.5 Multi-photo vs video capture

To determine SS7, the requirement was either to take multiple photographs per trigger or to capture video footage. Existing implementations of the latter on the ESP-32 cam module either use resolutions much lower than the module is capable of or can only achieve very low framerates [3]. Additionally, examination of the source code used for video capture reveals that the module is only natively capabale of capturing a single frame at a time, and utilising video footage requires complex considerations of the timing of frame captures and the encoding of the video files. Within the time limits of this project, it was deemed that still images would be more appropriate, especially since the requirement could still be met by capturing multiple still images per movement trigger.

1.4 Submodule design

1.4.1 Camera/transmitter module firmware flow

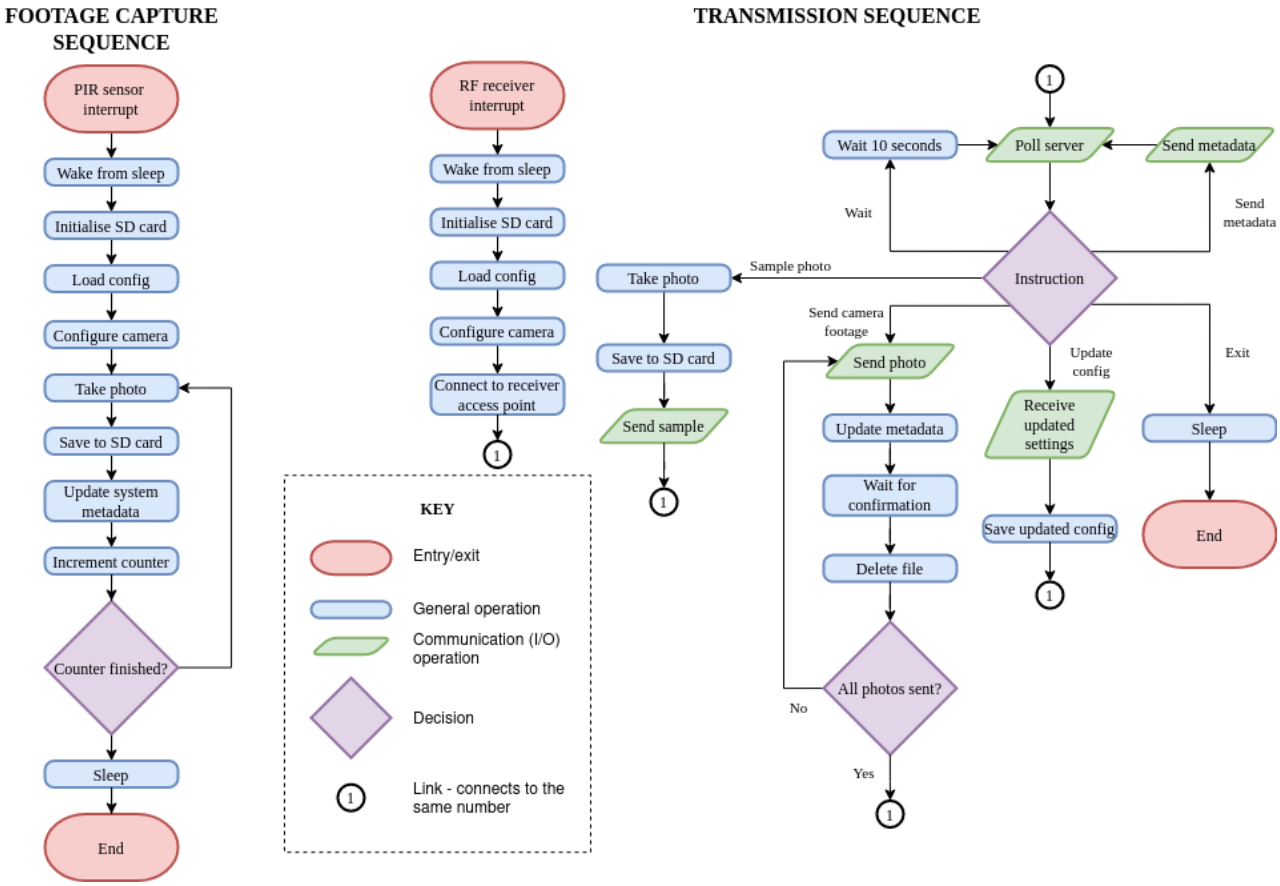

Figure 1.1: Flowchart representing the program flow of the ESP32-cam camera/transmitter module

Figure 1.1 demonstrates a high-level program flow of the Camera/Transmitter module. The two sequences operate independently. The module has two primary purposes - capturing footage, and transmitting it. Footage capture is triggered the PIR sensors. Transmission can be triggered manually from the reveiver module, with an input that comes via the RF receiver. The system automatically goes back to sleep if it fails to initialise key peripherals, fails to connect to WiFi, or if the web requests fail more than 3 times in a row.

1.4.2 Implementation of configuration file

As decided in subsection 1.3.4, the camera/transmitter module reads a customiseable JSON configuration file to determine it's camera and networking configuration. If the configuration file is unavailable or unreadable, or any configuration parameters are invalid, any missing parameters are loaded as defaults. The configuration can be updated remotely, overwriting the previous configuration.

1.4.3 HTTP server implementation and API

The HTTP server was implemented using an asynchronous web server library for the Raspberry Pi Pico W citations[4]. This library was based on a similar library for the ESP32 [5]. An asynchronous library was preferred to a synchronous one, since it allowed the main user interface and control of the

receiver module to operate independently of the web server, meaning that web server operations would not cause unexpected delays or lack of responsiveness in the UI. This choice was also made in the interests of future developement; a future application with multiple clients (e.g., broswers on mobile phones) connecting to the server simultaneously could experience significant issues with responsiveness on the UI if the server operated synchronously. ESP32 library is well used and includes extensive code examples, which were a valuable tool in the development process [4]. Although the library for the Raspbery Pi Pico W is not as well documented, it is built on the same principles as the ESP32 library and the examples and documentation from the ESP32 were thus still relevant.

To meet this functionality of, the server Application Programming Interface (API) detailed in Table 1.3 was developed. Additionally, all requests required a 'Device-ID' header, which specified the ID of the incoming device; this would be used by the server to differentiate between different camera/transmitter modules if multiple were deployed.

Communication from receiver to transmitter was performed via GET requests, since these indicate that the client is fetching a resource from the server. Communication from transmitter to receiver was sent via PUT requests. PUT requests indicate an idempotent transfer from client to server, meaning that a request with the same form as a previous request is sent, the previous resource is overwritten. This was deemed more appropriate than a non-idempotent POST request, since idempotency was in line with the way the images were stored on the receiver. An image received from the same device with the same name was assumed to be a repeat send of a previous image, and would thus overwrite the image stored on the receiver. This meant that the camera/transmitter module could safely resend images in the instance of an unexpected error from the response from the server, without risk of duplicating said images.

Endpoint	**Request type**	**Description**
/next_instruction	GET	Client requests a new instruction on what operation to perform from the server. Instruction is delivered in plaintext in the response body. Instructions are as shown in Figure 1.1.
/metadata	PUT	Client uploads a file of type 'application/json' to the server. File contains the number of images currently stored on the camera/transmitter, and the percentage of SD card storage used. Server saves the file as 'metadata.json' in the directory corresponding to the device ID.
/upload_image	PUT	Client uploads a file of type 'image/jpeg' to the server. File name is a specified request header 'Photo-Name'. Server saves the photo with the given name to the directory corresponding to the device ID. Used multiple times in succession to send all images from client to server.
/sample_image	PUT	Client uploads a file of type 'image/jpeg' to the server. Server saves the photo as 'sample.jpeg' to the directory corresponding to the device ID.
/update_config	GET	Client requests an updated configuration file. Server responds with file of type 'application/json'. File sent is 'config.json' from the directory corresponding to the device ID on the server.

Table 1.3: Summary of common coding environments for the available microcontroller

1.4.4 Receiver control flow

The core requirement was to give the user control over the instructions sent to the camera/transmitter module, as discussed in subsection 1.4.1. An overview of the control flow for the receiver module is shown in Figure 1.2.

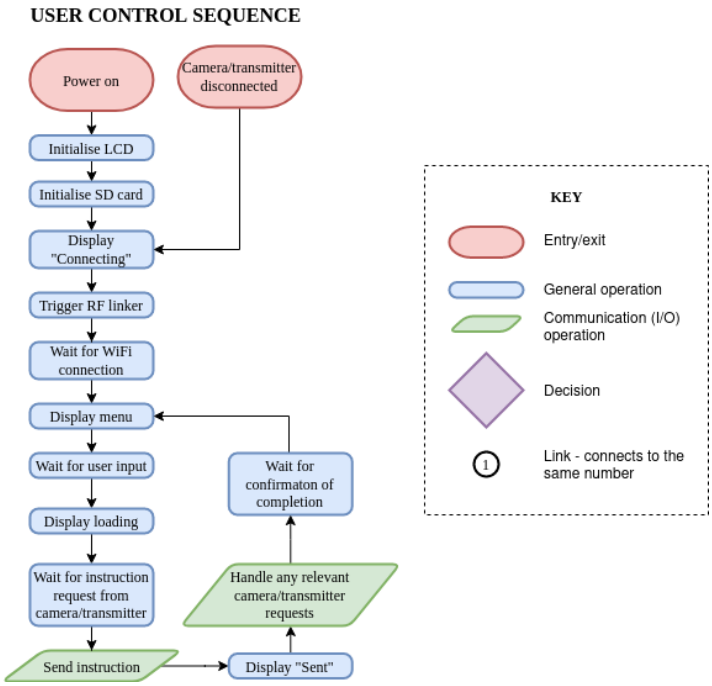

Figure 1.2: Flowchart representing the program flow of the Raspberry Pi Pico W reciever module

Buttons can either be read inside the program control loop, or using hardware interrupts. The lack of hardware debouncing informed this decision. Software debouncing can easily be implemented by checking when a button was last triggered, and not reading the button input again until a specified timeframe has elapsed. However, with interrupts, there is the added consideration that the intterupt handler triggers every time the button is triggered. This prevents the main loop from executing, and in this instance, could slow down the asychronous web server. As such, the buttons were read in the main loop.

1.4.5 User Interface (UI) design

Figure 1.3: Sample of the menu display for the receiver module

The hardware available for the user interface was a 16x2 character Liquid Crystal Display. To meet SS17, the user interface must be easy to understand. On a small LCD screen, it is not possible to display a lot of information. Showing such information would often require shortening or abstracting it to levels that can easily become incomprehensible. The user interface was developed with a simple menu, giving only the options availble from subsetion 1.4.1. If more functions were necessary on the receiver device, it may have been necessary to investigate a user interface with multiple sub-menus (each containing groups of similar functions). However, the small size of the display limits the ability

to show the user feedback about their current position in the program; keeping the menu simple is thus important for user comperehensibility. It was thus deemed more appropriate to have a single menu screen. A sample of the dispalyed menu is shown in Figure 1.3. The menu options were 'Get Metadata', 'Update Config', 'Send Photos', 'Sample Photo', and 'Exit'.

Menu navigation is performed via the 4 buttons, each of which is assigned a dedicated task - Up, Down, Confirm, and Cancel. The user can learn precisely what to expect from these buttons because their functions do not change. The Up and Down buttons scroll through the menu. The Confirm button confirms the currently selected option. The Cancel buttons cancels the current operation if it has not yet been sent to the camera/transmitter.

To meet SS16, the user interface must show constant feedback, to indicate the current state of the device and to show that the device is still operational. To achieve this, the following were integrated into the user design:

- As the user scrolls through the menu, an arrow (>) in the top-right corner indicates that the function displayed on the top line of the menu can be selected.
- While the device is waiting for connection to the camera/transmitter, it displays 'Connecting' and a constantly moving character. The menu options are not visible, making it clear to the user that the menu functions will not operate until the camera/transmitter is connected.
- While the device is waiting to send an instruction or waiting for confirmation from the camera/-transmiter that the instruction has been completed, a constantly moving character is displayed to indicate to the user that the reciever device is still operational. The selected option remains visible on the screen so that the user can be confident about what the device is currently doing.
- Once the instruction to the camera/transmitter is sent and can not longer be cancelled, ''>SENT>' is displayed on the screen, to make the current status of the function clear to the user.

1.4.6 Camera sensor settings

The OV2640 camera has a wide variety of configuration options which can be adjusted to suit the particular needs of the deployment. These options were all included in the configuration file, as the settings for each location would best be determined by the researchers involved upon deployment. However, a select number of configuration options were considered for the purposes of choosing default values, which could provide a workable positon from which the configuration could be adjusted.

The number of captures per trigger (3) and interval between triggers ()60s) were implemented based on the suggestions from the project stakeholders [1]. Other settings considered were the automatic exposure control, exposure level, brightness, contrast, and saturation. The camera was tested with various settings in fairly well-lit conditions, which most of the nests are likely to have. Dark conditions could not be tested owing the the limitations faced with the infrared filter on the camera discussed in chapter **??**. Qualitative assessments of the image quality were performed to determine the optimum settings.

It quickly became apparent that the automatic exposure control did not function effectively. Even in moderate light, the images taken with automatic expsure control were too dark to make out any

detail. Turning off the automatic exposure control and manually setting the exposure to it's maximum value was the only way to yield photos with enough light to make out any detail. The brightness, contrast, and saturation settings made minimal difference to the image quality, though the qualitative assessment was that the clearest images were produced with higher values. A sample image with the final chosen settings is shown in Figure 1.4.

Figure 1.4: Sample image taken chosen camera settings

1.4.7 Simulation and testing practices

Testing was performed on the system hardware. A number of tools and practices were used to aid in the development process:

- Debug mode: A debug c/c++ macro was available in the script. When defined, this macro enabled a number of serial outputs to aid in the process of debugging. This was defined using a macro so that the code used for debugging was not unnecessarily present in the final implementation.
- Unit testing procedures: A number of c/c++ macros were used to define unit tests, which could test portions of the firmware individually. These were used to individually validate functions.
- Curl commands: HTTP requests were made in testing using Curl [6]. The tool could easily be used to create well-formatted HTTP requests with the desired information. This allowed testing of the web server on the receiver device in isolation from the transmitter device.
- Python webserver: A simple python web server was developed for testing the HTTP requests from the camera/transmitter device.

1.5 Acceptance Tests

The submodule acceptance tests are listed and detailed in Tables 1.4 and 1.5. In all cases, the firmware from the project repository must be flashed to the microcontrollers. In some cases, ATPs use the serial port to display information. The serial port is not used in normal operation as it is not necessary, and conflicts with the switch enable pin for the PIR sensors and RF linker. In these cases, the ATP specifies a specific macro to define in the code. This is performed by uncommenting the line in the code reading '#define ATPx' (where x is the ATP number) prior to flashing the code to the relevant microcontroller. For ATPs which use the serial port, the trigger disable switch (normally connected to GPIO1) must be disconnnected and manually pulled high within 5 seconds of triggering the device.

The SD card for the camera/transmitter module should be formatted with the FAT32 filesystem; the SD card for the receiver module should be formatted with the exFAT filesystem. Unless otherwise

specified, the storage of both SD cards should be empty prior to starting the test.

UR/FR/SS	**Description**
ATP & related SS	ATP1 - SS1, SS8, SS12
Description	Photo is captured upon trigger from PIR sensor, and transmission mode is triggered from the RF linker.
Procedure	Standard: Power on the camera/transmitter module. Trigger either PIR sensor once by moving in front of it, or emulate a trigger by manually pulling pins 2 or 4 HIGH. Open the SD card on a computer and observe its contents. Enable the ATP4 macro on the camera/transmitter module. Power on the module and monitor the serial port. Power on the receiver module (which will automatically awaken the camera/transmitter module using the RF-linker).
Acceptance criteria	3 photographs are present in the 'photos' directory on the SD card. On qualitative assesment, the photographs are of adequate quality. Serial monitor confirms awoken in transmission mode after using RF-linker.
Pass/fail	PASS
ATP & related SS	ATP2 - SS2
Description	Footage capture is triggered within 1 second.
Procedure	Power on the camera/transmitter module. Prepare a stopwatch and orient the camera to view the stopwatch. Simultaneously start the stopwatch and trigger a photo capture as described in ATP1. Open the SD card on a computer and view the first image. Repeat 5 times.
Acceptance criteria	The average time shown on the stopwatch is less than 1 second.
Pass/fail	PASS
Notes	The human error from the requirement to simultaneously start the stopwatch and trigger the camera may introduce some error in the reading; however, this error should be sufficiently small compared to the pass criteria to still gain a valid sense of the camera trigger time.
ATP & related SS	ATP3 - SS15, SS16, SS17
Description	Menu navigation is present and intuitive.
Procedure	Power on the receiver module. Navigate the menu using the up, down, confirm, and cancel buttons.
Acceptance criteria	Up button scrolls the menu upwards. Down button scrolls the menu downwards. Confirm buttton triggers the currently selected action and, after a brief waiting period, indicates that the request has been sent. Up and down buttons do not work after selecting confirm. Cancel button returns display to the menu if the request has not yet been sent. On a qualitative assesment, menu navigation is intuitive and provider adequate feedback about the device state.
Pass/fail	PASS
ATP & related SS	ATP4 - SS3, SS4
Description	Metadata is sent over network.
Procedure	Trigger the footage capture as described in ATP1 at least 5 times. Open the SD card on a computer, and note the number of images present and the percentage of storage currently used. Power on on the receiver module and wait for it to connect to the camera/transmitter module. Navigate the menu and select the 'SEND METADATA' option. Open the receiver SD card on a computer and observer the contents of 'cam1/metadata.json'
Acceptance criteria	Number of photographs and percentage of storage used indicated on the reciever are the same as the numbers manually observed on the camera/transmitter SD card.
Pass/fail	PASS
ATP & related SS	ATP5 - SS3, SS5
Description	Sample photo is sent over network.
Procedure	Power on both modules. Wait for the devices to connect to each other. Select the 'SEND SAMPLE' option. Wait for confirmation of completion. Open the receiver SD card on a computer and observe its contents.
Acceptance criteria	A sample image is present at the 'cam1/sample.jpeg' on the SD card.
Pass/fail	PASS
ATP & related SS	ATP6 - SS3, SS6
Description	All captured photos are sent over network.
Procedure	Power on camera/transmitter module. Trigger the camera/transmitter module as described in ATP1 at least 10 times. Open the camera/transmitter SD card on a computer and observe its contents. Replace the SD card in the camera/transmitter module. Power on the receiver module and wait for it to connect to the camera/transmitter module. Select the 'SEND PHOTOS' option. Wait for completion. Open the receiver SD card on a computer and observe its contents.
Acceptance criteria	All images initially present in the 'photos' directory of the camera/transmitter module SD card are present in the 'cam1' directory of the receiver SD card.
Pass/fail	PASS

Table 1.4: ATPs 1-6 for the firmware subsystem

UR/FR/SS	Description
ATP & related SS	ATP7 - SS10
Description	Configuration options are loaded from SD card.
Procedure	Enable the ATP7 macro on the camera/transmitter module. Copy the contents of 'config-ATP7-1.json' into 'config/config.json' on the SD card. Power on the module trigger and it in capture mode following the procedure from ATP1. Observe the outputted exposure level, number of captures per trigger, and time between triggers on the serial monitor. Repeat with 'config-ATP7-2.json'.
Acceptance criteria	First run: outputted values are as specified in the config file. Second run: outputted values are the defaults (1200, 3, 60), since the values from the config file are invalid.
Pass/fail	PASS
ATP & related SS	ATP8 - SS11
Description	Update configuration options remotely.
Procedure	Power on both modules. Load the contents of 'config-ATP8.json' into 'cam1/config.json' on the SD card. Wait for the modules to connect, then navigate the menu and select the 'UPDATE CONFIG' option. Wait for confirmation. Open the camera/transmitter SD card on a computer and observe its contents.
Acceptance criteria	The content of 'config/config.json' from the SD card is the same as 'config-ATP9.json'.
Pass/fail	PASS
ATP & related SS	ATP9 - SS9
Description	Receiver/transmitter utilises deep sleep when not in use.
Procedure	Verify by observing source code that the ESP32 module utilises deep sleep functions after capture/transmission modes are complete.
Acceptance criteria	Functionality is present.
Pass/fail	PASS
ATP & related SS	ATP10 - SS13, SS14
Description	Receiver module can integrate with multiple devices.
Procedure	Copy the conents of 'config-ATP10.json' to 'config/config.json' on the receiver/transmitter SD card. Repeat the procedure described for ATP6.
Acceptance criteria	Images are saved to the 'cam2' directory rather than the 'cam1' directory on the receiver SD card.
Pass/fail	PASS

Table 1.5: ATPs 7-10 for the firmware subsystem

1.6 Conclusions

This section discussed firmware development for the Red-Winged Starling Nest Cameras. The user requirements were detailed and derived into functional requirements and specifications. The ultimate goal of this subsystem was to integrate the various hardware peripherals and microcontrollers into a system that was capable of taking images, transmitting the images wirelessly to a receiver device, using and remotely updateable configuration, and interfacing with users in an intuitive way.

The Arduino framework was chosen for firmware development. An HTTP server was implemented for wireless communication between the camera/transmitter and receiver modules. The configuration was stored and loaded on a human-readable JSON file. The web server was hosted on the receiver device to enable future applications that might include access to the server from mobile broswers.

A system firmware flow was developed for the camera/transmitter device that triggered footage capture on detection of movement, and triggerd transmission mode upon connection from the receiver device. An appropriate web server was developed to server for the receiver device was developed to serve the endpoints required by the camera/transmitter device. A user control scheme and user interface was developed to enable user interaction with the receiver device. A qualitative assessment of the soft-configurable camera settings was performed to determine optimum default settings.

A set of acceptance test procedures were developed from the subsystem specifications. All acceptance tests were passed.

Bibliography

[1] S. Hofmeyer and S. Cunningham, personal communication, Mar 2024.

[2] B. Blanchon, "ArduinoJson," https://github.com/bblanchon/ArduinoJson, 2024, [Accessed 12-05-2024].

[3] J. Zahary, "ESP32-CAM-Video-Recorder," https://github.com/jameszah/ESP32-CAM-Video-Recorder, 2022, [Online; accessed 12-May-2024].

[4] K. Hoang, "AsyncWebServer_RP2040W," https://github.com/khoih-prog/AsyncWebServer_RP2040W, 2023, [Accessed 12-05-2024].

[5] H. Gochkov, "ESPAsyncWebServer," https://github.com/me-no-dev/ESPAsyncWebServer, 2022, [Accessed 12-05-2024].

[6] M. Hostetter, D. A. Kranz, C. Seed, C. Terman, and S. Ward, "Curl: a gentle slope language for the web." *World wide web journal*, vol. 2, no. 2, pp. 121–134, 1997.