AUTOMATED HYPERSPECTRAL FIELD SCANNER FOR VEGETATION MONITORING

Bachelor Thesis

By: Moritz Prüm
AUTOMATED HYPERSPECTRAL FIELD SCANNER FOR VEGETATION MONITORING

A Thesis Submitted in Partial Fulfilment of the Requirements of the Degree of

Bachelor of Science

in

Environment and Energy

by

Moritz Prüm

Oststraße 20

47475 Kamp-Lintfort

Matriculation Number:

13192

Submission Date:

29th August 2017
Abstract

Vegetation monitoring is an important tool in a variety of fields, e.g. precision agriculture and plants’ phenotyping. The system design of existing non-invasive ground based systems is usually not open-source. Therefore, these systems cannot be reliably modified and thus are not suitable to advanced applications, such as prototyping of new sensors. This thesis describes the development of a ground based hyperspectral monitoring system for use in plant science and sensor testing. A well-documented and automated working prototype has been developed and initial tests show promising results. Further work in system modularity and redundancy is needed.

Keywords: vegetation monitoring, hyperspectral imaging, embedded systems, low-power design, system integration.
Contents

1 Introduction ............................................................................................................ 1

2 Theoretical Background ......................................................................................... 2
   2.1 The Electromagnetic Spectrum .......................................................................... 2
   2.2 Remote Sensing ................................................................................................. 2
   2.3 Hyperspectral Imaging ....................................................................................... 3
   2.4 Push-broom Imaging Spectrometer ..................................................................... 5

3 System Design ........................................................................................................ 6
   3.1 Overview ........................................................................................................... 6
   3.2 Hardware .......................................................................................................... 7
      3.2.1 Development Board (Teensy) ..................................................................... 8
      3.2.2 Real Time Clock .......................................................................................... 9
      3.2.3 GSM Module ............................................................................................... 9
      3.2.4 Stepper Motor ............................................................................................ 10
      3.2.5 Limit Switches ............................................................................................ 12
      3.2.6 Raspberry Pi ............................................................................................... 13
      3.2.7 Further Hardware Components .................................................................... 13
      3.2.8 Enclosure ................................................................................................... 14
      3.2.9 Power Supply and Management .................................................................. 17
   3.3 Software ............................................................................................................ 21
      3.3.1 Microcontroller ........................................................................................... 21
      3.3.2 Raspberry Pi ............................................................................................... 22
      3.3.3 Server ........................................................................................................ 27
      3.3.4 Web Application ......................................................................................... 28
      3.3.5 Sequence Diagrams .................................................................................... 31

4 Results and Discussion ......................................................................................... 36

5 Summary and Outlook ......................................................................................... 41

6 References ............................................................................................................. 44

A. Appendix .............................................................................................................. 47
List of Figures

Figure 1: The electromagnetic spectrum .......................................................... 2

Figure 2: A comparison the reflectance of photosynthesizing green leaf, senescing yellow and red leaves and a shed brown leaf of a Sweetgum tree ......................... 4

Figure 3: A hyperspectral cube where many images are stacked to form a cube with two spatial (x,y) and one spectral dimension (z) ..................................................... 5

Figure 4: The optical chain of a hyperspectral push-broom spectrometer ........... 5

Figure 5: Overview of the main system ................................................................ 6

Figure 6: Hardware components of the system .................................................... 8

Figure 7: DS1337 wiring circuit ......................................................................... 9

Figure 8: SIM800C wiring circuit ..................................................................... 10

Figure 9: Limit switch wiring circuit .................................................................. 12

Figure 10: 3D model of the platform and camera enclosure .............................. 15

Figure 11: The calculated viewing angles added to the 3D model of the platform to check for interferences by the camera enclosure lid .............................................. 17

Figure 12: Modified Lumix battery with an adjustable DC to DC buck converter .... 18

Figure 13: Power distribution diagram for the system ....................................... 19

Figure 14: The main page of the web application for viewing the system status, logs and on-board webcam. For the starting and aborting measurements ................... 30

Figure 15: Web application pages for modifying the system (left image) and Photon Focus (right image) settings ........................................................................... 31

Figure 16: System sequence diagram for a RTC alarm wake up ......................... 33

Figure 17: System sequence diagram for a user wake up .................................... 34

Figure 18: Test setup of the field scanner system ............................................. 36
List of Tables

Table 1: Motor settings ........................................................................................................ 11
Table 2: Focal lengths of the camera lenses and sensor/image dimensions of the spectrograph and the Lumix camera ................................................................. 16
Table 3 Power supply requirements of the components .................................................. 17
Table 4: Idle and leakage current consumption of the components ............................. 18
Table 5: List of commands that the Raspberry Pi and server exchange and their definitions. ......................................................................................................................... 29
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>AT</td>
<td>attention</td>
</tr>
<tr>
<td>BIOS</td>
<td>basic input output system</td>
</tr>
<tr>
<td>CAD</td>
<td>computer-aided design</td>
</tr>
<tr>
<td>CD</td>
<td>compact disc</td>
</tr>
<tr>
<td>CMOS</td>
<td>complementary metal-oxide-semiconductor</td>
</tr>
<tr>
<td>CSS</td>
<td>cascading style sheets</td>
</tr>
<tr>
<td>DC</td>
<td>direct current</td>
</tr>
<tr>
<td>DIE</td>
<td>integrated development environment</td>
</tr>
<tr>
<td>DSR</td>
<td>digital single lens reflex</td>
</tr>
<tr>
<td>DTR</td>
<td>data terminal ready</td>
</tr>
<tr>
<td>FOV</td>
<td>field of view</td>
</tr>
<tr>
<td>FTP</td>
<td>file transfer protocol</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>GigE</td>
<td>gigabit Ethernet</td>
</tr>
<tr>
<td>GIL</td>
<td>global interpreter lock</td>
</tr>
<tr>
<td>GPIO</td>
<td>general purpose input output</td>
</tr>
<tr>
<td>GSM</td>
<td>global system for mobile</td>
</tr>
<tr>
<td>HTML</td>
<td>hypertext markup language</td>
</tr>
<tr>
<td>HTTP</td>
<td>hypertext transfer protocol</td>
</tr>
<tr>
<td>I2C</td>
<td>inter-integrated circuit</td>
</tr>
<tr>
<td>IC</td>
<td>integrated circuit</td>
</tr>
<tr>
<td>IPvx</td>
<td>internet protocol (version x)</td>
</tr>
<tr>
<td>IPxy</td>
<td>international protection marking</td>
</tr>
<tr>
<td>JSON</td>
<td>javascript object notation</td>
</tr>
<tr>
<td>LED</td>
<td>light emitting diode</td>
</tr>
<tr>
<td>LTE</td>
<td>long-term evolution</td>
</tr>
<tr>
<td>Mbps</td>
<td>megabit per second</td>
</tr>
<tr>
<td>MJPEG</td>
<td>motion joint photographic experts group</td>
</tr>
<tr>
<td>NAT</td>
<td>network address translation</td>
</tr>
<tr>
<td>NDVI</td>
<td>normalized difference vegetation index</td>
</tr>
<tr>
<td>NTP</td>
<td>network time protocol</td>
</tr>
<tr>
<td>OOP</td>
<td>object oriented programming</td>
</tr>
<tr>
<td>RAM</td>
<td>random access memory</td>
</tr>
<tr>
<td>rps</td>
<td>rounds per second</td>
</tr>
<tr>
<td>RTC</td>
<td>real time clock</td>
</tr>
<tr>
<td>SCL</td>
<td>serial clock</td>
</tr>
<tr>
<td>SDA</td>
<td>serial data</td>
</tr>
<tr>
<td>SSH</td>
<td>secure shell</td>
</tr>
<tr>
<td>TCP</td>
<td>transmission control protocol</td>
</tr>
<tr>
<td>UART</td>
<td>universal asynchronous receiver-transmitter</td>
</tr>
<tr>
<td>UML</td>
<td>unified modelling language</td>
</tr>
<tr>
<td>USB</td>
<td>universal serial bus</td>
</tr>
<tr>
<td>VPN</td>
<td>virtual private network</td>
</tr>
<tr>
<td>XML</td>
<td>extensible markup language</td>
</tr>
</tbody>
</table>
1 Introduction

Remote sensing has gained increasing attention in many fields, particularly in the area of agriculture. The world’s increasing population and therefore its growing demand for food, feed and fuel supply, are fundamental reasons to broaden the understanding of crops and to be able to increase their yields, especially in the light of climate change (Rahman, 2015).

Global warming has been effecting the yield of crops severely. Beyond a certain temperature range, the major concerns are the decreasing crop yield, because the plants speed up in development, therefore grow too rapidly to produce useful biomass. The biomass production is further tempered due to generally dryer soil conditions (Global Warming and Agriculture, 2008, pp. 1-2).

Remote sensing plant diseases is of importance in agriculture. Measurements to determine the incidence, severity and negative effects of the diseases are of great significance when dealing with field crop, horticulture, plant breeding and plant research in general. These observations are also means for precise plant protection measures and to predict the propagation of the diseases (Mahlein, 2016, pp. 241-242).

Acquisition of remote sensing data can be achieved with airborne vehicles or stationary ground based systems. The area covered by the stationary systems is smaller than airborne systems, though the resolutions of their data is higher. This allows observations that are more detailed and have higher precision in a small area. Specifically, stationary systems that can cover a small test field with a few experimental plots are regarded as beneficial in plant research.

A few commercial and non-commercial stationary ground based systems exist. They acquire data about plants with non-invasive methods. For example, the Phenospex FieldScan (Phenospex, 2017) or the Field Scanalyzer by Lemnatech (LemnaTec, 2017). Nevertheless, in-depth system design of these products is not available to the public. Therefore, changes in the systems cannot be achieved as easily and testing other sensors that are not part of the company’s offering can be challenging. As a result, these systems are useful for direct plant research but are not platforms suitable for developing and testing sensors.

This thesis describes the development of a stationary ground based system for vegetation monitoring. In contrast to existing solutions the proposed system design is open source, meaning that its functioning can be well understood and modified whenever needed.
The procedure of this paper is divided into various sections. In the first section, the theoretical background concerning hyperspectral imaging and remote sensing is covered. Afterwards (Section 3), an overview of the system design is followed by the description of the hardware components and software. The functioning of the system and the results of a first test run are provided in Section 4. Finally, the conclusions and recommendations for further development are provided in Section 5.

2 Theoretical Background

2.1 The Electromagnetic Spectrum

The electromagnetic spectrum is characterized by a broad range of frequencies and wavelengths, each associated with a specific quantity and intensity of energy. The spectrum ranges from pico-meter (gamma rays) to the kilo-meter range (radio waves), whereby the energy level is highest for gamma rays and lowest for radio waves, as depicted in Figure 1. Whereas the human eye can only see the visible region of the spectrum (400-700 nm), hyperspectral imaging can collect and process information from beyond the visible part of the electromagnetic spectrum as described in the following section.

![The electromagnetic spectrum](https://commons.wikimedia.org/wiki/File:EM_spectrum.svg)

**Figure 1: The electromagnetic spectrum.**

Source: https://commons.wikimedia.org/wiki/File:EM_spectrum.svg

2.2 Remote Sensing

The use of non-invasive methods allows insights into plant biochemistry and dynamic plant features without damaging the plant that is investigated over time. Whenever these
traits have to be observed over time, invasive or destructive methods cannot be used (Burkart, 2015).

In the past decades, remote sensing using hyperspectral imaging has been gaining increased attention in the fields of agriculture to monitor against droughts, diseases and nutrient stress or to monitor shoreline changes, which can be used to prevent coastal erosion or to track natural disasters and to develop mitigation strategies.

The most commonly known remote sensing systems include satellites, airplanes, drones and small-scale applications, whereby every application has certain advantages and disadvantages. For instances, satellites have the biggest range and can therefore cover the large scales. However, due to long distances between the sensor and the earth’s surface, the atmosphere is interfering by scattering and absorption of long-wave radiation reflected by the earth surface. Therefore, the quality of the images from satellites could be reduced and compared with images taken from a smaller distance (e.g. drone). Most of the sensing devices used for remote sensing record information about the spectral reflectance including hyperspectral imaging applications as described in the following (Jensen, 2014, pp. 2-29)

### 2.3 Hyperspectral Imaging

Hyperspectral imaging is the process of taking images and assigning numerical values (spectral radiance) to each pixel, utilizing a range of wavelengths across the electromagnetic spectrum. Through the use of statistical analysis and specialized software, the pixels can be then sorted and characterized in order to distinguish between groups of pixels which describe for instance plant characteristics or environmental conditions. For instances, a well-known index derived from hyperspectral imaging is the Normalized Difference Vegetation Index (NDVI) which can be used as a measure for plant health in agriculture. For the process of photosynthesis, plants are absorbing red and blue light and reflect Near Infrared (NIR) as depicted in Figure 2.

Healthy plants generally show a higher reflection in the NIR range compared to unhealthy plants. These differences in plant health can be quantified with the NDVI which is calculated by dividing the difference between the reflectance of near infrared and the red with their sum \((\text{NIR} – \text{VIS}) / (\text{NIR} + \text{VIS})\).
In general, the NDVI requires to measure two spectral bands, which can be derived using hyper spectral cameras. The sensors of these imaging systems usually have ten or more bands of data with a bandwidth of 1 to 15 nanometers. The resolution of a camera depends on two components, spatial and spectral. The spatial resolution of a sensor is defined as the smallest recognizable spatial frequency. This frequency can be measured in the laboratory using a pattern of point or bars with uniform spacing measured in millimeter. Digital images are created using an electro optical sensor. These sensors consist of a silt, a lens and a chip. The chip contains pixels that are sensitive to photons of different wavelengths and accept photons as a collection of electrons or charge. This collection represents the intensity and wavelength of the light received. An analog to digital converter changes the charge into an intensity value known as digital number (DN). The lens is providing focus and magnification of the photons and the silt is limiting the amount of photons that pass through the chip. The outcome of a hyper spectral imaging system is the assembly of each wave-length band into a three-dimensional hyperspectral data cube as shown in Figure 3.
2.4 Push-broom Imaging Spectrometer

Up-to date, there are two major branches of spectral imaging, push-broom imaging and whiskbroom imaging. In comparison to the whiskbroom imaging system, the push-broom spectrometer, with a higher signal-to-noise ratio (SNR), has been widely used in airborne and space borne remote sensing. In general, push-broom spectrometers provide data in the form of hyperspectral cubes as depicted in Figure 4. The hyperspectral cube has generally two dimensions, one spectral dimension representing the spectral signature of every spatial pixel, and one across-track and along track dimension which is provided by the motion of the platform. The plane which is formed by the across-track dimension and spectral dimension is called frame which has P spectral (P rows) pixels and M across-track (M-columns) pixels.

Figure 3: A hyperspectral cube where many images are stacked to form a cube with two spatial \((x,y)\) and one spectral dimension \((z)\).
Source: modified from https://commons.wikimedia.org/wiki/File:Mono_multi_and_hyperspectral.svg

Figure 4: The optical chain of a hyperspectral push-broom spectrometer.
Source: Dell’Endice, 2009, p. 634
3 System Design

The main focus of this thesis is to assemble and program a remotely deployed hyperspectral imaging system, including hardware and software design, that is remotely accessible, capable of autonomous data acquisition and data transfer.

This section covers a general overview of the system. Furthermore, the detailed assembly of the hardware components and the written software will be presented.

3.1 Overview

The purpose of the designed system is to automate and control a remotely deployed hyperspectral imaging setup and to transmit the measurement data wirelessly to a remote server. The need for administrating the system also requires wireless accessibility. Furthermore, the system has to be powered off of a Photovoltaic system due to its remote location and unavailability of a mains connection.

![Figure 5: Overview of the main system](image-url)
The system is deployed over a field; whose spectral data of vegetation is the subject of interest. As depicted in Figure 5, the system consists of a truss, which is the supporting structure on which all other components are fitted. Two linear guides are mounted below the top part of the truss and stretch from left to right. The motor platform has four rollers attached, which are hung up on these guides and therefore, allows the motor platform to move freely along the horizontal axis. Additionally, a timing belt is attached horizontally. The motor platform, being the core of the system, is fitted with all the hardware necessary to initiate and control measurements as well as to store and transmit acquired data.

A stepper motor is mounted to the platform, that has a pulley attached to its shaft, which in turn is guided by the timing belt mentioned previously. Thus, running the motor in a specific direction “pulls” the platform along the truss towards a desired location. The motor is fitted with an internal motor controller that allows control and also provides positional feedback.

Hyperspectral images are acquired via a hyperspectral push-broom sensor and spectrograph, that are mounted below the platform. A mini-computer is responsible for initiating the hyperspectral camera and storing the acquired data locally on a hard-drive. The positional feedback of the stepper motor is essential for this process, since the push-broom sensor/spectrograph combination “only” acquires the spectral signature of a single line of the target (the target being the vegetation below the truss), the positional information is used to determine how the lines will be stitched/fused together.

A digital single lens reflex (DSLR) camera attached next to hyperspectral camera is utilized as reference for the hyperspectral images. In addition to the positional feedback, the DSLR camera images are gathered to aid the stitching of the hyperspectral lines.

### 3.2 Hardware

Several hardware components were required for the wireless accessibility and the autonomous functioning of the system. The following sections describe the assembly of the individual components and the reasons for their necessity.
3.2.1 Development Board (Teensy)

The development board used in the system is a Teensy 3.6 (PJRC, 2017), based on a 32-bit 180 MHz ARM® Cortex®-M4 processor with 1024 Kbytes flash storage, 256 Kbytes RAM\(^1\), 62 input/output pins, I\(^2\)C\(^2\), 6 serial ports and interrupts on every I/O pin being the noteworthy specifications.

A function of the Teensy is that of a “power manager” for the system (explained further in Section 3.2.9) by controlling relays that turn power on and off to the rest of the system. Additionally, it interfaces with the real time clock to set the current time and alarms and the interrupts were used to wake it up from low power mode.

The Teensy was chosen at the beginning of this thesis, due to the large amount of I/O pins and their interrupt capabilities, the 6 serial ports and speed.

---

\(^1\) Random Access Memory
\(^2\) Inter-Integrated Circuit
3.2.2 Real Time Clock

The real time clock (RTC) used is a Maxim Integrated™ DS1337 IC\(^3\) with two time-of-day alarms. Data is serially transmitted through an I\(^2\)C bus and the clock/calendar provides seconds, minutes, hours, day, date, month, and year information (Maxim Integrated, 2015).

![Figure 7: DS1337 wiring circuit](image)

Following Maxim’s guidelines, the IC required an external 32.768kHz crystal and two 10k Ohm pull-up resistors for the I\(^2\)C serial data (SDA) and serial clock (SCL) lines. An additional 10k Ohm pull-up resistor was attached to one of the alarm outputs (INTA) of the RTC. INTA is an open drain pin which internally is pulled to ground through a transistor once an alarm is triggered and left floating while idle. The I\(^2\)C connection and the INTA interrupt pins of the IC were connected to the I\(^2\)C interface and an interrupt pin on the Teensy board respectively. An overview of the connections can be seen in Figure 7.

The RTC was chosen due to the alarm feature, in order to signal the Teensy to wake up from low power mode, through programmable times.

3.2.3 GSM Module

The GSM\(^4\) module used is a low cost, non-branded breakout board for the SIMCOM SIM800c (SIMCom, 2017) which is a complete quad-band GSM solution capable of transmitting voice, SMS\(^5\) and data.

---

\(^3\) Integrated Circuit  
\(^4\) Global System for Mobile  
\(^5\) Short Message Service
Figure 8: SIM800C wiring circuit

Figure 8 illustrates the wiring of the module. The receiver and transmitter pins (RX_GSM and TX_GSM) of the UART\(^6\) interface were connected to the Teensy which made it possible to communicate to it via AT\(^7\) commands. The DTR\(^8\) pin controls the low power mode of the module (SIMCom, 2015). When the pin is at high level it enters the low power mode. Whereas, when pulled to ground with an NPN transistor for 10 ms, the device enters its normal operational mode. When in low power mode the module is not responsive over serial communication, but still is able to receive incoming calls. The RI pin activates whenever there is an incoming call which is detected by the Teensy through an interrupt to wake itself up from its low power mode, which in turn powers up the other system components.

The GSM module provides a method of remotely powering up the system and thus enabling the system to be interacted with e.g. data transfer, setting changes, manual measurements and general system administration.

### 3.2.4 Stepper Motor

The previously available stepper motor utilized to drive the platform along the truss is a Nanotec PD6-N8918L9504-S, IP67\(^9\) graded stepper motor (Nanotec, 2016). At full step, the motor has 200 steps per revolution and a stepping angle of 1.8°. The settings used in this thesis are depicted in Table 1. The integrated controller of the motor offers a

---

\(^6\) Universal Asynchronous Receiver-Transmitter
\(^7\) Attention
\(^8\) Data Terminal Ready
\(^9\) International Protection Marking
RS485 interface which enables to set and readout its operation modes, parameters and current position.

<table>
<thead>
<tr>
<th>Table 1: Motor settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step resolution</strong></td>
</tr>
<tr>
<td><strong>Step angle</strong></td>
</tr>
<tr>
<td><strong>Steps per revolution</strong></td>
</tr>
<tr>
<td><strong>Operating modes</strong></td>
</tr>
<tr>
<td><strong>Ramp type</strong></td>
</tr>
<tr>
<td><strong>Ramp (Acceleration)</strong></td>
</tr>
<tr>
<td><strong>Start frequency</strong></td>
</tr>
<tr>
<td><strong>Maximum frequency</strong></td>
</tr>
<tr>
<td><strong>Rounds per minute (RPM)</strong></td>
</tr>
</tbody>
</table>

Due to the nature of stepper motors, their rotation is not entirely smooth, as the motor is moving “step by step” (Acarnley, 2002). The goal of the used step resolution is to make the motor run as smooth as possible. With a 32\textsuperscript{nd} micro-stepping resolution the motor moves 32 steps for every one step that would have been moved at full step resolution. The resulting step angle can be calculated as following

$$Step \ angle = \frac{Step \ angle \ at \ full \ step}{Step \ resolution} \quad (3.1)$$

The result of Equation 4.1 is then used to calculate the steps per revolution

$$Steps \ per \ revolution = \frac{360°}{Step \ angle} \quad (3.2)$$

The rounds per minute (RPM) of the motor is calculated by the maximum frequency (target speed) and the steps per revolution (Equation 4.2) as following

$$RPM = \frac{Maximum \ frequency}{Steps \ per \ revolution} \quad (3.3)$$

Additionally, the time taken to reach the maximum frequency from the initial start frequency is calculated by

$$t = \frac{Maximum \ frequency - Start \ frequency}{Acceleration \ ramp} \quad (3.4)$$
A timing belt pulley is fitted on to the motors shaft, to enable positioning along the truss. The pulley is driven on a timing belt which is attached to the frame of truss. Measuring the pulleys diameter and determining its circumference enables to calculate the position of the motor platform relative to its starting position distance travelled per step (Equation 4.5).

\[
Distance \ per \ step = \frac{Circumference}{Steps \ per \ revolution}
\]  

(3.5)

With a pulley circumference of 245.05 mm, the resulting distance per step is approximately 0.03829 mm, which in turn is used as a reference for each hyperspectral image taken.

### 3.2.5 Limit Switches

Two IP67 graded, Panasonic AZD1004 (Panasonic, 2017) limit switches mounted on the far left and right of the motor platform, are utilized as end switches. Making it possible to detect whether the platform has reached either end of the truss. This is necessary to indicate the system to stop a measurement run that is taking place and to avoid the self-destruction of the system.

![Figure 9: Limit switch wiring circuit](image)

Figure 9 shows the wiring of the limit switches to two general purpose input output (GPIO) pins on the Raspberry Pi. The limit switches are connected to ground and the internal pull-up resistors for those pins were turned on to detect a change from a high signal (limit switch off) to a low signal (limit switch activated).
3.2.6 Raspberry Pi

The Raspberry Pi V3 (Raspberry Pi, 2017) is a single-board Linux computer and serves several purposes in this setup.

It has internet connectivity through a Huawei E3372 LTE\textsuperscript{10} USB\textsuperscript{11} modem (Huawei, 2017) and serves as the main point that is accessible remotely. The connectivity through the LTE modem provides the link to the server for data transfer and remote administration.

The Raspberry Pi controls the stepper motor and communicates to the Teensy and NUC. Additionally, it is connected to the NUC over Ethernet allowing the Raspberry Pi to access the measurement data on the NUC and transfer them to the server. The serial connection to the NUC is there to communicate with the hyperspectral image acquisition software.

A Wi-Fi dongle (Edimax, 2015) was added to the Raspberry Pi for a Wi-Fi access point. The reason for this will be further elaborated in the Software section.

The Raspberry Pi camera module (Waveshare, 2017) that was connected is for the user of the field scanner to be able to view the vegetation below via a web application. The web application will be presented in Section 3.3.4.

3.2.7 Further Hardware Components

A miniature linear actuator (Actuonix, 2016) with a stroke length of 100 mm was chosen suitable for the camera enclosure that will be covered in Section 3.2.8. Its input voltage can be chosen to be between 0 and 15 V DC and its body is IP54 rated.

Latching relays (Panasonic, 2016) were chosen to turn on and off several voltage sources to the system. The reason for using latching relays is that they can be turned on and off with individual coils, thus requiring no software implementation to save their state.

Their purpose will be covered in Section 3.2.9 and the circuitry can be seen in the schematic in the Appendix.

The following components were already available prior to the commencing of this thesis and were adopted as they are, with only minor changes.

The imaging sensor used for the hyperspectral imaging assembly is a Photon Focus BL1-D1312IE-80-G2 (Photonfocus, 2014). It futures a CMOS\textsuperscript{12} sensor with a resolution of 1312 x 1082 pixels and Global shutter. Its spectral resolution ranges from 320 to 1000 nm and thus covers a small part of the ultraviolet, the complete visible and close to half

\textsuperscript{10} Long-term Evolution
\textsuperscript{11} Universal Serial Bus
\textsuperscript{12} Complementary metal-oxide-semiconductor
of the near-infrared (IR-A) spectrum. Data transfer is accomplished through a GigE\textsuperscript{13} connection, whereas its power input and several input and output pins are accessible through a connector. Two of these pins are used as trigger outputs, were one of them is the same signal that also triggers the Photon Focus. This signal is then used by the Raspberry Pi to capture the current position of the field scanner platform (further explained in the following sections). The second trigger is used to release the shutter of the DSLR camera.

A spectrograph and lens complete the hyperspectral imaging assembly. The spectrograph used is a Specim ImSpector V10 with an image size corresponding to a standard 2/3-inch image sensor and has a spectral range of 400 to 1000 nm (Specim, 2016). The lens that is mounted to the spectrograph is a Schneider-Kreuznach Cinegon 1.4 with a spectral transmission range from 400 to 1000 nm (Schneider Kreuznach, 2008), thus matching the spectrograph and image sensor.

The hyperspectral imaging assembly, consisting of the imaging sensor, spectrograph and the lens, will be referred to as Photon Focus from here on.

The miniature desktop computer Intel NUC DC53427HYE\textsuperscript{14} (Intel, 2017) is used for the data acquisition from the Photon Focus imaging sensor through the GigE connection previously explained. An additional USB Ethernet adapter was added to the previously existing set up, to allow data transfer between the NUC and Raspberry Pi through an Ethernet cable. The reason for this connection will be clarified in the Software section.

The DSLR reference camera that was initially mentioned in the Overview section is a Panasonic Lumix GX1\textsuperscript{15} (Panasonic, n.d.). The Lumix gets triggered by the Photon Focus through an audio jack connector.

The complete triggering circuitry can be observed in the schematic in the Appendix.

3.2.8 Enclosure

The complete system is outdoors and one of the major necessities is to have its electronic components protected from bad weather conditions. The stepper motor and limit switches are both IP67 rated and therefore already well protected. Except for those two, the previously mentioned components have no or minimal protection and therefore susceptible to bad weathering conditions which can lead to device failure and damage.

The hyperspectral and DSLR cameras need a custom enclosure since their lenses need to be exposed during data acquisition and sealed when no measurement is taking place.

\textsuperscript{13} Gigabit Ethernet
\textsuperscript{14} Referred to as NUC from here on
\textsuperscript{15} Referred to as Lumix from here on
The CAD Design Software Autodesk® Fusion 360™ was used to model the complete frame (including: brackets, rollers and the gear). The frame dimensions were accurately measured and implemented in the model. The CAD models for the stepper motor and linear actuator were retrieved from their product pages.

Once the modelling of the base was complete, different enclosure options were simulated in the model. Fusion 360's option to align and add moveable joints to components made it possible to simulate moving parts. The solution that was chosen most fitting can be seen in Figure 10.

The chosen enclosure consists of a rectangular box with a lid that was connected by hinges. For the lid to be able to automatically open and close, the linear actuator that was described in Section 3.2.7 was fitted on one side of the box. It was clear from the 3D model that the lid had to be opened at an angle of more than 90° for it not to be interfering with the viewing angles of both the Hyperspectral and Lumix cameras.
Therefore, the linear actuator had to be mounted to specific positions on the side of the box and the lid, to uphold the functionalities of the cameras.

To find out the required opening angle of the camera enclosure lid, the fields of view (FOV) for both cameras were calculated. The FOV of an optical system is often expressed as the maximum angular size of the object as seen from the entrance pupil (Greivenkamp, 2004, p. 27).

The focal length of a lens or optical system is the distance from the center to the point on the optical axis where a bundle of incoming rays parallel to the optical axis get focused to. The focal length usually represents the distance between the center of projection and the image plane (Ikeuchi, 2014).

Additional to the focal length of the camera lenses, the sensor width and height are needed to calculate the FOV. The sensor dimensions for the Lumix camera are available from its datasheet (Panasonic, n.d.). Whereas the Cinegon lens of the hyperspectral camera is not directly attached to camera module itself, but to the end of the spectrograph right after its entrance slit. Therefore, the image dimensions of the spectrograph were used as alternative. The dimensions used are listed below.

<table>
<thead>
<tr>
<th>Lens</th>
<th>Focal Length (f)</th>
<th>Sensor Dimensions (HxW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinegon 1.4/8</td>
<td>8.2 mm</td>
<td>6.6 mm x 8.8 mm</td>
</tr>
<tr>
<td>Panasonic LUMIX G 14mm F2.5</td>
<td>14 mm</td>
<td>13 mm x 17.3 mm</td>
</tr>
</tbody>
</table>

Finally, the FOV was calculated using Equation 3.6 (Greivenkamp, 2004, p. 27). Where, $S$ is the Sensor width, height or diagonal length and $f$ is the focal length.

$$\text{FOV} \left[^\circ\right] = 2 \times \arctan \frac{S}{2f}$$ \tag{3.6}$$

The diagonal length of the Lumix sensor and the spectrograph image were calculated to be approximately 21.64 mm and 11 mm respectively, resulting a FOV of ca. 75°. The FOV of the hyperspectral camera equated to ca. 67.7°.

These calculated FOV’s where added into the 3D model to check for an interference by the lid of the camera box. Looking at Figure 11 it can be seen that there is enough clearance between the lid and the camera FOV’s to allow proper image acquisition.

\[17\] Resulting image dimensions of the spectrograph
3.2.9 Power Supply and Management

The main power source for the system are two 12 V 60 Ah Lead-Gel batteries. They are connected in series and supply a total voltage of 24 V DC. The stepper motor is powered directly off of the 24 V of the batteries. Whereas, the remaining components have different requirements. The different power supply requirements of the individual devices are summarized in Table 3 (NOTE: the LTE modem is powered by the USB of the Raspberry Pi).

<table>
<thead>
<tr>
<th>Component</th>
<th>Voltage (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stepper Motor</td>
<td>24 – 48</td>
</tr>
<tr>
<td>Photon Focus</td>
<td>12</td>
</tr>
<tr>
<td>NUC</td>
<td>12 – 19</td>
</tr>
<tr>
<td>Lumix camera</td>
<td>7.2</td>
</tr>
<tr>
<td>Teensy</td>
<td>3.6 - 6</td>
</tr>
<tr>
<td>Raspberry Pi</td>
<td>5</td>
</tr>
<tr>
<td>GSM Module</td>
<td>5</td>
</tr>
<tr>
<td>RTC</td>
<td>1.8 – 5.5</td>
</tr>
<tr>
<td>Linear Actuator</td>
<td>5</td>
</tr>
</tbody>
</table>

Two DC/DC Step-down converters were used to accommodate the different power supply needs of the components. A 12 V DC/DC converter (Recom RPA60-2412SFW)
(Recom, 2017) with a maximum output current of 5 A, was solely used to power the Photon Focus and NUC. Whereas, a 5 V DC/DC converter (Recom RP30-2405SF) with a maximum output of 6 A (Recom, 2015) was used for the remaining devices, except for the DSLR camera.

The DSLR, being a hand-held camera, runs off of 7.2 V Li-ion batteries. Since the battery has to be manually removed for charging, and its USB connection is only for data transfer purposes, an old camera battery was modified by removing its cells and adding an adjustable DC/DC buck converter. With the modified battery in place, the DSLR is powered externally. Figure 12 depicts the modified battery housing.

![Figure 12: Modified Lumix battery with an adjustable DC to DC buck converter](image)

The stand-by power consumption of the components, when no data transfer or measurement run is taking place, is undesirable in this battery powered system. Idle and leakage current consumption of the components was measured to calculate the amount of power the system uses when it doesn’t perform any task (Table 4).

<table>
<thead>
<tr>
<th>Component</th>
<th>Average Current (A)</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stepper Motor</td>
<td>0.224</td>
<td>Idle (No Load)</td>
</tr>
<tr>
<td>Photon Focus</td>
<td>0.077</td>
<td>Idle</td>
</tr>
<tr>
<td>NUC</td>
<td>0.052</td>
<td>Leakage current</td>
</tr>
<tr>
<td>Lumix</td>
<td>0.485</td>
<td>Idle</td>
</tr>
<tr>
<td>Teensy</td>
<td>0.089</td>
<td>Idle</td>
</tr>
<tr>
<td>Raspberry Pi</td>
<td>0.090</td>
<td>Leakage current</td>
</tr>
<tr>
<td>GSM Module</td>
<td>0.016</td>
<td>Idle</td>
</tr>
<tr>
<td>RTC</td>
<td>425 (nA)</td>
<td>Idle (Negligible)</td>
</tr>
<tr>
<td>Linear Actuator</td>
<td>0.079</td>
<td>Idle</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>1.112</strong></td>
<td></td>
</tr>
</tbody>
</table>
As can be seen from the table above, the system consumes around 1 A on average. The Photon Focus, stepper motor and linear actuator don’t possess power off functionalities and the Lumix can only be turned off physically. The RTC and GSM module are kept powered as they are the only way to remotely turn on the system. With its 425 nA, the power consumption of RTC is negligible. The GSM modules consumption can be decreased by utilizing its DTR pin, described in previous sections. This way its current consumption is lowered to around 2.3 mA. The Teensy turns into hibernation through software and lowers its current down to 26 µA. In this mode it can only be woken up with an interrupt, from either the RTC or GSM module.

To minimize power consumption during the systems idleness, the Teensy, with its minimal power consumption, acts as “power manager” by controlling the power supply to the unused devices. As can be seen in Figure 11, the Teensy controls relays to three voltage blocks.

![Figure 13: Power distribution diagram for the system](image)

For example, when a measurement alarm is triggered by the RTC, the Teensy wakes up from its low power mode and activates the relays which in turn power up the rest of the system. Once the system is on, the Raspberry Pi takes control and initiates a measurement run. When the measurement is over it signals the Teensy that its shutting down and the relays are powered off after a small delay.
Additionally, the USB power of the Lumix camera is controlled through a relay. This is due to the Lumix camera going into “Connected to a PC” mode when the USB connection is active. In this state it is unable to be triggered. Therefore, when a measurement is being initiated, the USB power is turned off and turned on to allow data transfer.

The Raspberry Pi boots by default when power is applied, which is ideal considering that power is applied through a relay controlled by the Teensy. The NUC on the other hand needs to be turned on physically by default. To circumvent this, the BIOS\textsuperscript{18} settings were changed to “boot on power”. This gave the NUC the same power on functionality as the Raspberry Pi.

\textsuperscript{18} Basic Input Output System
3.3 Software
In this section the software that was written for the system will be presented. This, includes software for the Teensy, Raspberry Pi, server and the web application. The programming languages used were Python, Arduino (C++), HTML\textsuperscript{19}, CSS\textsuperscript{20} and jQuery. All the software can be found on the CD\textsuperscript{21} that is attached to the back of this thesis and its contents are listed in the Appendix.

3.3.1 Microcontroller
The code for the Teensy was programmed through the Arduino IDE\textsuperscript{22} and includes a state machine (see Appendix) to process incoming serial commands from the Raspberry Pi. It enables the Pi to set the measurement alarms, synchronizing its time (to update the RTC time), send commands to the GSM Module, control the relays attached to the Teensy and to open and close the lid of the camera enclosure. The relays to the individual voltage blocks (Figure 11) can also be turned on and off.

The Teensy is kept in hibernation mode (low power mode) while the system is not performing a task as previously described in Section 3.2.9. The open source library Snooze (GitHub, 2017) was utilized to accomplish this. Two pins were programmatically set as interrupts. The pin number that is responsible for the interrupt is returned by this library. In this way the source of the wakeup (RTC alarm or GSM call) can be determined. The wake up source is kept in memory and sent to the Raspberry Pi when its booted, to allow a specific sequence to take place depending on this source.

Once the desired functionalities and responsibilities of the Teensy were established, a state machine for incoming serial commands was thought through. The state machine eased the process of changing and adding new serial commands to the code, especially in receiving time and alarms from the Raspberry Pi. Each character of the incoming serial command is processed separately, added to a buffer and a state is set according to the character received. Upon receiving a new line character, the state machine comes to an end and the string is processed according to the state that it is in. The available commands the Teensy can receive can be summarized as follows:

- List of alarm times to be set on the RTC
- Time and date to keep the RTC synchronized to the time on the Raspberry Pi
- Reading out the time and date set on the RTC
- Reading out the stored alarm times and deleting individual ones
- AT command to be passed on to the GSM module

\textsuperscript{19} Hypertext Markup Language
\textsuperscript{20} Cascading Style Sheets
\textsuperscript{21} Compact Disc
\textsuperscript{22} Integrated Development Environment
• Opening and closing of the camera enclosures lid (linear actuator)
• Turning the individual relays on and off

To process the individual commands, several functions were implemented in the code. The list of alarms received and stored in memory are iterated to sort in an ascending order and each alarm time is compared to the current time. The alarm closest to current time is set on the RTC right before the Teensy goes into low power mode and the system is shut down. This ensures the system will get powered on at predefined times set by the user. The way the user changes and/or adds alarm times will be presented in the following sections.

The linear actuator responds to a single pulse with a duration between 1000 to 2000 ms (Actuonix, 2016). The stroke of the actuator is completely retracted with a pulse duration of 1000ms and fully extended with a duration of 2000 ms. This was programmatically solved using Arduino’s servo library.

To set (on) or reset (off) the latching relays, each corresponding coil was turned on for 10ms as per their datasheets (Panasonic, 2016).

3.3.2 Raspberry Pi

Raspberry Pi’s functionality, as in the communication with the Teensy, NUC, Motor and Server, was programmed using Python 3 due to its large open source community, ease of use and rapid prototyping functionalities. For readability and further maintenance purposes the code is mainly object oriented and consists of the following parts:

• Teensy communication
• Motor communication
• NUC communication
• Socket client
• FTP\(^{23}\) client
• Main

Each part of the code is an individual class and each class inherits from the Multi-processing package (except for the main and motor communication parts) from python’s standard library, which allows true parallelism between processes by using sub processes instead of threads (Python, 2017) Python’s threading module is subject to the Global Interpreter Lock. The GIL prevents multiple native threads from running in parallel. Since the classes of the code each inherit from the multiprocessing module, they run in parallel after being created and initiated.

\(^{23}\) File Transfer Protocol
Communication between the different processes is achieved through so called pipes (Python, 2017). Basically, pipes are a shared memory space that tie two processes together. Bi-directional pipes with two ends were used. These two ends are two separate objects were an object each is passed to two separate processes. This allows these to pass messages, hence the processes are able to communicate with each other. Sending and receiving messages through these pipes works in a similar manner as reading and writing from and to a file, with a slightly different naming convention of the methods in Python.

Each pipe object can check if there is any data available in the shared space. This can be done through so called polling. Polling was used in most of the processes with a simple “if” statement for continuous execution without blocking the remaining code, enabling the processes to check for new incoming commands or to send messages/data to the Teensy, NUC and Server.

The Teensy communication process, is responsible for transmitting and receiving serial messages to and from the Teensy. The idea behind this process is to set new measurement alarms on the RTC as well as retrieving the wake up source (alarm or call). The retrieved wake source is then passed to the main process. As previously described, power to the individual hardware components (that are controlled by relays) are thus controllable through the Python code on the Raspberry Pi by sending the appropriate command string to the Teensy. For example, to turn on the 12 V powered block (NUC and Photon Focus) a “P12Y” is sent to the Teensy and to turn it off a “P12N” is passed on.

The Stepper motor, for positioning the platform along the truss thus the Photon Focus, is connected to the Raspberry Pi through a RS485 to USB converter. The motor can be set up and controlled solely through serial commands sent to its firmware. A separate python package was written to deal with the sending and receiving of these commands and used in combination with the main Python script. The required commands were taken from the programming manual for the specific model of the motor (Nanotec, 2013). Additionally, the current position (in steps) is requested from the motor this way. The conversion from steps to metric distance is then accomplished by utilizing the formulas described in the hardware section. This conversion is done in the main part of the Python script.

Communication with the NUC is accomplished serially. The C++ code on the NUC, responsible for initialization and acquisition of images from the Photon Focus, has been taken over from a previous project. Once the NUC is powered and booted, the C++ software starts automatically and waits for a command over serial to commence a measurement with the Photon Focus. The settings of the Photon Focus are stored in a
XML\textsuperscript{24} file. This file is read and used to establish a connection to the Photon Focus and is stored locally on the NUC.

Some sort of communication between the Server back end and the Python code on the Raspberry Pi was decided to be necessary to pass the current status of the system (measuring, idle or off) and to initiate and abort measurements. A TCP\textsuperscript{25} Socket connection was used.

The socket server resides on the server itself and the socket client that initializes connection to the server is part of the Python code on the Raspberry Pi. The functioning of the socket client process is similar to the Teensy and NUC processes, were it continuously checks for new incoming messages. This socket connection allows the user to start/stop measurements and to shut down the system from the web application that is hosted on the server, as well as giving the user visual and textual feedback on the current status of the system. This interaction will be further explained in Section 3.3.3.

The FTP client class is responsible for synchronizing the hyperspectral data stored on the NUC with the FTP server it is connected to. Once this process is initiated it continuously compares the data of the NUC with the data stored on the server side. Additionally, there is a continuous comparison between the file sizes locally and on the server. When a file on the server has a lower file size than the corresponding file locally (same file name), the FTP client reinitiates the file upload to the server at the specific byte offset. Overall this allows the servers data to be synchronized with that of the hyperspectral system. Uploads can be interrupted and resumed at the same offset. This process is optional since data plans with mobile carriers vary in costs, thus transferring several GB\textsuperscript{26} of data from one measurement run could be undesirable.

To ease the process of changing several system settings, a JSON\textsuperscript{27} file was added where the settings of the stepper motor, system wake up times that get set on the RTC and whether the measurement data should be synchronized with the server can be set and modified.

JSON is a lightweight and text based data-interchange format based on the JavaScript programming language. It is completely language independent. Parsing and generating JSON is well implemented in a wide range of programming languages (JSON, 2017).

The JSON file can be edited on the server which is then requested by the python code on the Raspberry Pi, to make sure up to date settings are used that are set by the user.

\textsuperscript{24} Extensible Markup Language
\textsuperscript{25} Transmission Control Protocol
\textsuperscript{26} Gigabyte
\textsuperscript{27} JavaScript Object Notation
This JSON file, in the case when the system is woken up by an RTC alarm, is requested from the server only once right before a measurement is started. Whereas, when a user wakes the system it is requested once at the very beginning and additionally before every measurement the user starts. This allows the user to potentially try out different motor settings for a measurement. The contents of this JSON file can be seen below.

```json
{
  "motor_settings": {
    "ramptype": "0",
    "ramp": "200",
    "stepmode": "32",
    "startfreq": "100",
    "maxfreq": "4800"
  },
  "other_settings": {
    "ftp_client": "False"
  },
  "alarms": {
    "measuring_alarms": {
      "a1": "13:00",
      "a2": "13:10",
      "a3": "13:20"
    }
  }
}
```

The main part of the python script on the Raspberry Pi, the part that gets executed upon its boot, runs as follows:

- All the pipe objects that are used to communicate between the processes are set up, that are then passed with the serial port names and baud rates (Teensy and NUC) and the network address and ports (Socket and ftp clients) to the individual processes that get initialized.
- The Teensy and socket processes are started, whereas the other processes are started after the wake up source (RTC alarm or user) is retrieved from the Teensy.
- The GPIO pins for the Photon Focus trigger, limit switches and the coils for the relay that turns the USB connection of the Lumix on and off are defined.
- The code is halted until a connection to the Teensy and socket server is established
- The system settings in JSON format are requested from the server. The settings are compared with the ones stored locally on the Raspberry Pi and are overwritten when there are any changes. The newest settings are then extracted from the JSON file.
- Metric distance per step of the stepper motor is calculated with the settings provided by the JSON file.
- The wake up source of the system is requested from Teensy. Additionally, the alarm times to be set on the RTC are extracted from the JSON file and are sent to the Teensy.
- Depending on the wakeup source two different scenarios get executed. Though both scenarios utilize the same measurement function.

The measurement function that is executed once a wakeup source is determined is listed below. Both wakeup scenarios utilize this function with minor differences.

- The USB data connection between the Lumix and the NUC is turned off via its relay and therefore the Lumix is able to get triggered by the Photon Focus.
- The code checks whether it is the first measurement run that is taking place since the systems boot. If it is, the camera enclosures lid is opened and the 12V relay that powers the NUC and Photon Focus is turned on. In case it isn’t the first measurement run, a restart command is sent to the NUC that reinitializes the data acquisition procedure.
- The code blocks until the NUC is booted and sends a ready command.
- The up to date XML settings for the Photon Focus are requested from the server.
- The stepper motor is driven to its home position which is the home limit switch.
- A text file is created on the Raspberry Pi to store the positional information of the stepper motor.
- When the NUC sends a “measuring” message the measurement function enters a while loop where the current position from the motor is requested and appended to the text file previously mentioned. This is done with each trigger event that is detected. Additionally, a check on whether an abort command has been received from the socket and if the end limit switch has been reached is done within this loop. Thus, the measurement run will be ended either through the user or by reaching the end of the truss. A stop command is then send to the NUC once one of these two events occur.
- When the measurement is ended the text file with the positional information is copied to the NUC, into the same folder that the measurement data had been saved to.
- The measurement function is ended and returns “True” if a complete measurement has taken place or “False” when it was aborted by the user.

Depending on the wakeup source the system is then either shutdown or a new measurement can be initiated. For a better overview this will be further explained in the Sequence Diagram section with the aid of diagrams.
On the NUC, the folder where all the measurement data and XML settings of the Photon Focus are being saved, was shared over the LAN of the Raspberry Pi and NUC. Thus, the Photon Focus settings file and the measurement data can be accessed from the Raspberry Pi. The measurement data can then be uploaded to the server if required or changes in the Photon Focus settings can be updated on the NUC.

Additionally, a Wi-Fi access point was set up on the Raspberry Pi. Which allows a user that is connected to it to access the Raspberry Pi through SSH\(^28\) for system maintenance, as well as to download the measurement data via FTP, without a wired connection to the system itself.

The Raspberry Pi camera, that was described in the Hardware section, was set up to capture the surroundings of the truss platform in an MJPEG\(^29\) format that is then streamed from the Raspberry Pi. The user can see this stream on the web application which is explained in Section 3.3.4.

### 3.3.3 Server

With the near exhaustion of address space with IPv4 and the transition to IPv6, several technologies are being utilized to include more devices and to keep IPv4 alive (Cannon, 2010, pp. 1-2)

One of these technologies known as Network Address Translation (NAT) enables a large number of devices to share a common public IPv4 address with the aid of the private IPv4 address space. These devices that are within a network that is behind a NAT gateway are not directly addressable from the internet.

The requirement for an intermediate system between the field scanner and the user resulted from the LTE connection being subject to NAT that was previously described. Since the LTE modem is behind a NAT Gateway, a single public IP address is shared among other devices that use the LTE service. The server, with its static public IP address, acts as an intermediate connection point. Allowing the field scanner and a client to initiate a connection through it. For this to be able to function an open source VPN (Virtual Private Network), namely OpenVPN, was installed on the server and the Raspberry Pi.

A VPN is a network that is created over the internet. Virtual implies that there is no real direct network connection between communicating hardware, but only a virtual connection that is set up by the VPN software. Whereas private means that the

---

\(^{28}\) Secure Shell  
\(^{29}\) Motion Joint Photographic Experts Group
connection is encrypted and only members with a valid certificate can join the network (Feilner, 2006).

The server is continuously running an OpenVPN server, that can be connected to with the use certificate files. When the field scanner wakes up from low power mode it initiates a connection to the VPN server and is therefore accessible to a client through the VPN.

The server also includes a Python script that is continuously running in the background. It is made of the following parts:

- FTP server
- Socket server
- Web framework

Same as the classes on the Raspberry Pi, the FTP and socket servers are both individual classes that inherit from the Multiprocessing package. Thus, allowing both to run parallel in the background. If the file transfer flag in the JSON settings file is enabled the Raspberry Pi initiates a connection to the FTP server upon boot and uploads the measurement data that is not yet uploaded to the server.

The socket server is part of the backend of the web application. It functions in the same manner as the socket client that is part of the Raspberry Pi’s code – it continuously checks for new incoming messages and messages can be sent from the main part (web framework) by utilizing pipe objects. This allows certain commands to be passed to the socket client on the Raspberry Pi and vice versa.

The web framework will be described in the following section.

### 3.3.4 Web Application

The web application created for the field scanner consists of a back and front end. The back end is based on the Flask (Flask, 2017) web framework. Flask is a web framework written in Python that allows to host and control web applications on a server. It was used since it is well documented and due to its simplicity.

The web application is hosted on the server’s own Open VPN IP address on the standard HTML port (80) and can only be accessed through the VPN itself.

So called route decorators were used to tell Flask what URL will trigger a certain function in the code. An example can be seen below. The index function will execute, thus render the HTML file index, when a user navigates to the IP address of the server’s Open VPN.

```python
@app.route('/')
def index():
    return render_template("index.html")
```
The functions corresponding to a certain route don’t necessarily have to render an HTML page, but can also be used to execute functions only on the server side without changing the contents displayed in the browser.

To avoid that a user of the field scanner has to wake up the system by dialing the GSM number from their own phone, an account for an online communication service, namely Twilio (Twilio, 2017a), was created. Twilio is an online service that allows its users to programatically initiate phone calls and other communication means via an API, which is also available for Python (Twilio, 2017b). The Twilio API was added to the backend of the web application and whenever a user navigates to http://10.8.0.1/_wake (10.8.0.1 is the server’s Open VPN address), the Flask backend will trigger the corresponding function for the route “/_wake”. This function then initiates a call to the field scanners GSM module through the Twilio API, therefore waking up the system.

The commands that the backend receives through the socket server determines what is displayed in the front end. This gives the user real time feedback on what the field scanner is currently doing. Commands that can be sent to the Raspberry Pi can start and stop measurement runs as well as turn off (low power mode) the field scanner system. The following table lists the commands that each side sends including their meanings.

<table>
<thead>
<tr>
<th>Command Raspberry Pi</th>
<th>Definition</th>
<th>Command Server</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>#WA</td>
<td>Wake up source: alarm</td>
<td>#D</td>
<td>System shutdown</td>
</tr>
<tr>
<td>#WC</td>
<td>Wake up source: call</td>
<td>#SM</td>
<td>Start measurement</td>
</tr>
<tr>
<td>#OFF</td>
<td>System is turning off</td>
<td>#SSM</td>
<td>Stop measurement</td>
</tr>
<tr>
<td>#M</td>
<td>Measurement started</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#NM</td>
<td>Measurement stopped</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By default, the individual routes in Flask are set to respond to a HTTP GET request. Most of the route functions in the web application use this method. Submitting contents of an HTML form (an HTML box with editable contents) from the browser to the server requires a POST method. The POST method was used for the pages where the user can edit the system and Photon Focus settings and submit these to the server. Upon submitting these settings, the Flask route triggers the function that deals with the POST request and saves the settings locally.

---

30 Application Programming Interface  
31 Hypertext Transfer Protocol
Additionally, the Flask backend is responsible for returning the system and Photon Focus settings that are requested from the Raspberry Pi. These JSON and XML files are saved locally on the server.

The front-end of the web application was programmed using HTML, CSS and JavaScript. HTML was used to give structure to the web application and to define its contents. This includes textual contents, buttons and images.

Styling of the web application was achieved through CSS and describes how the HTML content is displayed in the browser. Sizes, colors, heights and widths as well as margins of the HTML elements were defined through CSS.

A screenshot of the web applications main page can be seen in Figure 14 below.

![Screenshot of the web application](image)

**Figure 14:** The main page of the web application for viewing the system status, logs and on-board webcam. For the starting and aborting measurements.

JavaScript, specifically its library jQuery, was used to make the front-end of the website dynamic. Dynamic in this context means changing the HTML content without reloading the entire page. One aspect it was used for was to enable and disable certain buttons depending on the status of the field scanner. The previously mentioned socket commands are sent from the Raspberry Pi to the server. In turn, the server than sends status messages through a specific route (“/_updates”) with the use of JSON objects. The jQuery code extracts these status messages from the JSON file and enables or disables certain buttons in the web application accordingly. Thus restricting the user to certain options.

To give an example for the dynamic change, Figure 14 depicts the situation where the user woke up the system by pressing “wake up” in the web application. The user now
has the option to shut down the system, to start a measurement, to edit the system and Photon Focus settings and to start the webcam. The other buttons are disabled and not press-able.

The jQuery code also checks for idleness on the web application when the system is woken up by the user. Every second the code checks for the movement of the mouse pointer on the web applications front-end. When no movement is detected a counter is incremented and if it reaches 5 minutes it will send a shutdown command to the back-end. This ensures the system won’t be powered up for too long without any use.

The front-end pages where the user can edit the system and Photon Focus settings can be seen in Figure 15. Both pages consist of an editable HTML box that displays the current setting files that are stored on the server. The user has the option between saving the settings, reversing changes that where made or to return back to the main page.

Figure 15: Web application pages for modifying the system (left image) and Photon Focus (right image) settings

3.3.5 Sequence Diagrams

Now that the Python code of the Raspberry Pi, the server’s functionality and the web application have been covered, the interaction between these as a whole will be explained with the aid of diagrams.

The Unified Modelling Language (UML) is a general purpose modelling language standard, that is widely used to visualize the design of a software system (Rumpe, 2016, p. 2). Even though it is mainly used to visualize OOP, it can still be used as a tool to document a system in a non-OOP manner.

A sequence diagram describes the interactions between objects and focuses on the chronological order of the messages that are passed to accomplish a specific task (Seidl,
In the context of this thesis it will be used to visualize the interactions between the software of the different hardware components of the field scanner and not the interaction between objects.

Two sequence diagrams were created to provide a better overview of the order of communication and procedures of the system. The two scenarios consist of the predefined wake up by the RTC (Figure 16) and the wake up by a user (Figure 17).

As can be seen in these figures, the sequence moves from left to right and from top to bottom. The lines spanning from top to bottom are so called lifelines, were each line represents a part of the field scanners software.

The horizontal lines are messages passed from one lifeline to another, in this case messages passed from the web application to the Teensy, from the Teensy to the Raspberry Pi and so on. A solid line with a solid arrowhead represents synchronous communication were a return message is expected were the dotted lines represent the return messages. In order to make the diagrams more compact not all return messages have been implemented. A stick arrowhead represents an asynchronous message, were no return message is expected. The arrows pointing at solid circles are lost messages that get sent to a recipient outside the scope of the diagram that it belongs to. In contrast, arrows coming from a solid circle are found messages, where the source of the message is not in the scope of the diagram.

The green rectangles in the diagrams are fragment combination elements that enclose a certain sequence. The types of fragment combination elements used in the diagrams are “opt” (optional) and “alt” (alternative). The sequence that is enclosed by an optional fragment gets executed when the argument below the label is true. The alternative fragment on the other hand is separated into two by a dashed line. The top and bottom part of the fragment are two distinct options that, like the optional fragment, get executed depending on the arguments. It should be noted that only one argument will execute within an alternative fragment.
Figure 16: System sequence diagram for a RTC alarm wake up
Figure 17: System sequence diagram for a user wake up
Comparing the diagrams for both wakeup scenarios it can be seen that the initial booting of the system up until the online message that is displayed on the user's browser is the same, except that a different wake up source is passed to the Raspberry Pi.

The sequences that follow after the online message are different for both scenarios, as explained in previous sections.

When a user wakes the system through the web application, he or she has the option to commence a measurement several times as long as the system is online. The system and Photon Focus settings can be changed and are applied prior to each measurement that is started manually by the user. For example, allowing measurements with different motor speeds, integration times, etc.

Both scenarios utilize the same measurement function that was described in previous sections. A measurement will be stopped by the limit switch when the wake up source is the RTC, whereas additionally the measurement can be stopped by the user through the web application when the wakeup source is through the GSM module.

The webcam stream can be started during both sequences at any time independent of the wakeup source.

The system will be shut down by either pressing the shutdown button or being idle on the web application for 5 minutes when the user woke the system up. On the other hand, the system shuts down automatically after a measurement if it was woken up by the RTC.
4 Results and Discussion

The field scanner system, consisting of all parts from the hardware and software sections, was assembled for testing purposes. The setup can be seen in Figure 18 below.

![Figure 18: Test setup of the field scanner system](image)

Testing of the system was done gradually to check for proper functionality of individual parts. Due to time limitations and unexpected complexity of setting up the system up to this point, it was not possible to assemble the platform and mount it on the truss. Therefore, the results presented here are limited to the state of the system as achieved so far.

First the Teensy, RTC and GSM Module were connected and tested, to check their proper functionalities. The state machine to process incoming serial commands that was described in Section 3.3.1, worked as anticipated. The RTC was able to wake up the Teensy from low power mode. Furthermore, the Teensy also woke when the phone number of the temporary SIM card that the GSM module contained was called.

The linear actuator that opens and closes the camera box lid was tested and implemented in the Teensy’s software. The lid opened correctly when an RTC alarm was triggered or an incoming call was received by the GSM module. The opening angle of the lid had a $2^\circ$ deviation from the modelled angle due to a measurement mistake during mounting. Nevertheless, this small deviation does not compromise the intended functions of the cameras, thus the system.
Next, the Raspberry Pi and its communication with the Teensy was tested. The testing of the different parts of the Raspberry Pi’s Python code was performed individually which simplified the process of debugging.

The communication with the stepper motor proved to function as intended. For instance, setting the speed, acceleration as well as starting and stopping the motor worked. To ensure consistent starting positions of the platform a routine was implemented that is executed before each measurement run. This sequence consists of moving the motor a certain distance forward whilst checking for the end limit switch, then moving all the way to the home limit switch. This would be necessary, for example, when the wind moves the platform during idleness. After implementing the limit switches into the code, this sequence was tested by manually triggering the switches and resulted in the expected outcome.

The hyperspectral imaging assembly, consisting of the image sensor, the spectrograph and the lens were successfully set up and connected to the NUC. Since there was minimal documentation on the possible NUC commands and the image acquisition source code was not available at the time, the communication had to be reverse engineered. An available code from another project that communicated with the NUC and Photon Focus was analysed and the valid NUC commands were derived from it. The commands were then separately tested to ensure that they were interpreted correctly, resulting in proper functioning of the data acquisition.

Each trigger pulse that the Photon Focus outputs with an acquired image was counted and the resulting count was compared to the number of images that were acquired on the NUC. This was done to make sure that the Raspberry Pi does not miss a trigger pulse, since a corresponding position for each hyperspectral image acquired is necessary. Initially, the number of positions that were written to a text file were lower than the number of images. This was due to a serial read timeout for the stepper motor communication that was set too high which led to large delays in the Raspberry Pi’s code. Once this timeout was set to zero and the rest of the code was adapted accordingly, the count of positions and images matched in each test.

The socket communication on the server and Raspberry Pi side were tested using Putty, which is a free and open-source terminal program that supports raw socket connections. The web applications back and front –ends responded correctly to the different commands.

In all the tests described up until now, the hardware components were either powered over USB or from their own dedicated power supplies. The DC/DC converters and the relays that were described in Section 3.2.9, were soldered onto a Perfboard and initially
tested with LEDs\textsuperscript{33} to ensure correct functioning. The LEDs acted as indicators on whether the relays were on or off. All hardware components were then connected to their designated relays and voltages.

After all communications between different parts of the system had been tested separately, the system was tested as a whole. This includes several tests on the intended functioning when an RTC alarm is triggered and when a user wakes the system through the web application.

The pre-set alarms woke the system seamlessly, the camera enclosure lid opened, the relays turned on and data acquisition commenced while the motor was driven. After manually triggering the end limit switch, image acquisition stopped, the system was shut down and the Teensy turned off the relays and went into low power mode after setting the next alarm on the RTC. During this sequence the web application responded correctly and the individual steps of the measurement were observed in the system log dialog box.

Then, the user wake up scenario was examined several times. The web application was opened and the wakeup button was pressed, followed by the GSM module being called through the Twilio API as intended. The wakeup source was then passed correctly from the Teensy to the Raspberry Pi then to the web applications back-end through the socket connection. The result being that the front-end changed its mode to “online” and starting manual measurements was possible. Aborting a manual measurement and shutting down the system through the web-application also worked correctly.

At the end of each scenario the number of positions acquired by the Raspberry Pi were once again compared to the number of images acquired by the NUC and were matching throughout.

Correct results from changing the Photon Focus and system settings on the web application were confirmed. Meaning that the settings were correctly transferred from the server to the Raspberry Pi and through it to the NUC over the LAN connection. When incorrect system settings (JSON file), for example typos or wrong naming of specific settings, were entered, the Raspberry Pi fell back to its default settings. Though, no defaults were available for the Photon Focus XML settings file, therefore some incorrect settings resulted in an error of the data acquisition that could not be traced back in software by the Raspberry Pi’s Python code. Additionally, missing restrictions on values, for example the minimum and maximum speed of the motor, ended in jerky movements with too low speeds and errors for speeds that were too high.

\textsuperscript{33} Light Emitting Diode
The Raspberry Pi does not have an internal RTC and depends on an internet connection to set the correct time and date. When the Raspberry Pi is powered after a shutdown its time gets reset to the default UNIX time (1st January 1970) and relies on an NTP server to set the correct time once an internet connection is available. The issue that was encountered in this context was the occasional delay in the internet connection. The connection would be established after the Python code was started and the time synchronized with the Teensy would be incorrect. The outcome was that the RTC alarm would not wake the system at the intended time. During the tests, this was temporarily solved by writing a short Shell script that would check for an available internet connection before executing the Python script upon boot.

The initial idea behind the FTP server and client was to upload the measurement data to the server every time the field scanner wakes up. This would allow a user to download the data from the server at any time without waking up the field scanner. Later on it was realized that the costs of transferring measurement data over the LTE connection, where each measurement will have several GB, would be very expensive.

Considering the motor settings used (Section 3.2.4), where the motor moves at 0.75 rps and a full revolution equates to approximately 245.05 mm, the platform would travel 183.8 mm per second. Assuming a moveable span of 10 m along the truss and a 30 fps image acquisition rate of the Photon Focus, where each frame is circa 2.7 MB, the total size of a single measurement would result in roughly 4.4 GB. If daily monitoring of the vegetation is required, this would end up in data transfers of 100 to 200 GB per month which would exceed any commercially available data plans.

Additionally, the time it takes to transfer this amount of data would take too long which would defeat the purpose of a low powered system. In ideal conditions, the Huawei LTE modem with its maximum upload speed of 50 Mbps (Huawei, 2017), would take approximately 11 minutes to transfer one measurement to the server. Of course, this upload speed is in ideal conditions. The average upload speed of current LTE providers is around 15 Mbps (CHIP Online, 2017). With this speed a measurement would be uploaded in around 40 minutes.

The camera enclosure is designed to be mounted to the platform with four bolts and fixed in place with spring washers and wing nuts. If needed, this allows the removal of the enclosure without taking down the complete platform. This eases the procedure of

---

34 Network Time Protocol
35 Rounds Per Second
36 Megabit Per Second
maintenance of the cameras enclosed and additionally has the potential to make the system modular.

During the tests it was possible to connect to the Wi-Fi access point. Connections to the Raspberry Pi through SSH and FTP were made. This was deemed to be useful to edit and upload settings and code to the Raspberry Pi, as well as downloading the measurement data through the shared folder, which all functioned correctly. The same is true through the VPN connection with the LTE modem.

The Twilio trial account created for this thesis does not have an expiration date and comes with a certain amount of credit for testing purposes. As long as the calls that were made through it were not answered on the other end, in other words the Python API would initiate and hang up the outbound call, no credits would be used up.

As described in Section 3.2.9, low power consumption was partially achieved. Disconnecting the Raspberry Pi, NUC, Photon Focus and stepper motor from their power supplies, with the relays, proved to be good in saving power. Although, the efficiency of the DC/DC converter had not been considered during testing. When the teensy is in low power mode the efficiency of the converter is below 10% (Recom, 2015). This should be considered when further work on the field scanner is commenced.

The idea behind capturing the position of each hyperspectral image taken is to ease the interpolation between these. When each image is referenced to a position along the truss, the distance between each image can be calculated and data interpolated depending on this distance. To verify the results of these interpolations, an object with a known size, for example a square panel, could be used to compare the spatial dimensions of the Photon Focus with that of the spatial dimension of the interpolated data.

By default, the OpenVPN connection that is established on the Raspberry Pi, binds to all network interfaces and connects to the VPN server through an interface that has internet connectivity. In case the field scanner gets relocated, this feature could be used to exchange the LTE connection with a Wi-Fi connection, that would still connect to the VPN server, therefore retaining the web-applications functionality and the remote access of the system. Nevertheless, the LTE connection could still be used as a fall back option when the Wi-Fi connection fails.
5 Summary and Outlook

The present thesis described the hardware and software design process of an automated hyperspectral field scanner. Even though a photovoltaic system which is a fundamental part of the field scanner to ensure complete autonomous data acquisition has not been covered in this thesis, a working prototype has been achieved. Mandatory parts of the system have been brought to realization and act as a solid foundation for further work and completion of the field scanner.

Several steps to move the field scanner system from a prototype to a functioning product are necessary. Furthermore, several ideas and additional functions would improve the system as whole and ease its interaction with a user.

Reflectance values are generally used to calculate vegetation indices, for example for an NDVI image. In the context of the field scanner, reflectance is the fraction of incident solar power that is reflected by the vegetation below it. In order to derive reflectance, the incident solar power has to be measured. Most commonly this is achieved with reference surfaces that reflect the incoming power that in turn can be acquired by a hyperspectral imaging system. These reference surfaces reflect up to 99% (Labsphere, 2017) of the incoming radiation and usually come as panels. A reference panel should be added to the field scanner by means of mounting it in a weather proof enclosure that opens prior to a measurement and is in the field of view of the Photon Focus. If this is done, an additional rain sensor should be added to the system to prevent the reference panel from getting damaged by water.

Alternatively, a point spectrometer, which is a spectrometer that measures the intensity of radiation at a single point that it is directed to, could be used to measure solar radiation with a so called cosine receptor (Ocean Optics, 2017). If the spectrometers can be inter-calibrated, the point spectrometer would continuously be measuring the solar radiation. Measuring solar irradiance besides those of plants is beneficial for example during variations in cloud coverage. Nevertheless, further research has to be conducted on whether this is applicable and inter-calibration is possible.

So far, the logging of events and errors of the system had not been implemented. The FTP connection of the Raspberry Pi to the server could be used to synchronize the log files. Additionally, samples of the hyperspectral data could be pre-processed on the field scanner and uploaded to the server this way. The user could then access and view these samples without the need of waking up the system. Additional alarms that wake up the system can be easily implemented so that data would be processed at certain times.

As briefly mentioned in Section 4, system modularity can be achieved with the camera enclosure that was built. If weather proof connectors are used for the interfaces of the
cameras, the enclosure would be easily removable from the field scanner and replaceable with another one. The software for the new cameras could then be enabled either through a hardware switch on the field scanner and/or through the JSON settings file that was mentioned in Section 3.3.2.

To ensure easier editing and to prevent typing mistakes of the system and Photon Focus settings through the web-application, the editable text boxes could be replaced with sliders and checkboxes, rather than changing the settings in the displayed files. This would be useful to limit certain parameters, a feature which is missing in the current prototype.

Additionally, logging and displaying the charge level of the batteries and the power consumption of the system would add useful information about the field scanner. The battery voltage could be monitored with a simple voltage divider. Whereas the current consumption can be measured with a hall effect sensor, for example with the Allegro ACS717 (Allegro, 2016).

To achieve an autonomous system with photovoltaic panels, measuring the power consumption of the field scanner during measurement is needed, to be able to correctly size the solar panels and to calculate how many measurements a day can be made. Also, once the amount of power for a measurement is known, it can be used to start the measurement conditionally, meaning only when there is enough charge in the batteries.

So far, the system was planned with an overly optimistic mind set, i.e. every component is expected to fulfil its function and there is not much room for failure. Since this is usually not the case and because the system has to operate in field conditions, special attention should be payed to making individual parts of the system redundant and more fail safe then they currently are. For instance, the Raspberry Pi should be able to recognize when the stepper motor or other components are not functioning properly and avoid further execution of measurements but instead inform the user about the problem. In addition, an extra set of limit switches should be implemented some small distance from the first set and be able to shut down the motor in case the main switches don’t trigger for whatever reason. Next, perhaps most importantly, there should be a backup microcontroller with a separate power supply, that could shut down the main system if it detects some malfunction, e.g. critical instabilities of the main battery pack. These, and similar measures, besides more extensive error handling, are highly desirable for achieving redundancy and safety of the system.

Overall, a working prototype of the field scanner has been designed and presented in this Thesis. Even though the system needs further testing, improvements and calibration, once the transition from prototype to a thoroughly tested system is completed, it will
acquire hyperspectral data that is a vital asset to monitor vegetation. The system can thus be used in a variety of fields including phenotyping, precision agriculture and in general plant research. The modularity of the system and the eventual possibility of swapping and combining sensors can further expand its range of possible applications.
6 References


A. Appendix

This section gives additional information including the CD contents, circuit schematics and the Teensy state machine.

A.1 CD Contents

- **Software**
  - Raspberry Pi
    - Main code
    - Motor code
  - Teensy
  - Web application
    - Flask app
    - HTML Code
    - JavaScript Code
    - CSS code
    - JSON settings
    - XML settings

- **Diagrams**
  - Sequence diagrams
  - Circuit schematic
  - State machine
  - Images
A.2 Circuit schematic
<table>
<thead>
<tr>
<th>Corresponding FUNCTION</th>
<th>State</th>
<th>Clear</th>
<th>NONE</th>
<th>ALARM</th>
<th>TIME</th>
<th>SET</th>
<th>READONLY</th>
<th>DELETE</th>
<th>ATCNO</th>
<th>CAMBOX</th>
<th>POWER</th>
<th>CLEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>gBA(j)</td>
<td>A</td>
<td>state = ALARM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = DELETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBC(j)</td>
<td>C</td>
<td>state = CAMBOX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBN(j)</td>
<td>N</td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CAMBOX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBF(j)</td>
<td>P</td>
<td>state = POWER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = POWER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBS(j)</td>
<td>S</td>
<td>state = ATCNO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBT(j)</td>
<td>T</td>
<td>state = TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBX(k)</td>
<td>X</td>
<td>state = DELETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBY(j)</td>
<td>Y</td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBNcm(j)</td>
<td>?</td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CHECK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBDig(j)</td>
<td>0…9</td>
<td>state = READ</td>
<td>flag rc = A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = DELETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBDrv(j)</td>
<td></td>
<td>state = SET</td>
<td>flag rc = A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBCtm(j)</td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gB Cycl(j)</td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBSh(sh)</td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gBNfl(j)</td>
<td></td>
<td>state = NONE</td>
<td>reset src</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>state = NONE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A.3 State machine
Declaration of Authenticity

I, Moritz Prüm, hereby declare that the work presented herein is my own work completed without the use of any aids other than those listed. Any material from other sources or works done by others has been given due acknowledgement and listed in the reference section. Sentences or parts of sentences quoted literally are marked as quotations; identification of other references with regard to the statement and scope of the work is quoted. The work presented herein has not been published or submitted elsewhere for assessment in the same or a similar form. I will retain a copy of this assignment until after the Board of Examiners has published the results, which I will make available on request.