"GoPro" Style Multispectral Camera for Mobile NDVI Analysis

Final Report

A Report on the Creation and Applications of a Portable Handheld Multispectral Camera

Written By

Ryan Libiano (libiano2)

Amudhini Pandian (ramyaap2)

Omar Camarema (ocamar2)

Muthu Ganesh Arunachalam (muthuga2)

University of Illinois Urbana-Champaign

Department of Electrical and Computer Engineering

November 29th 2020

FA2020 ECE110 Honors Lab Project

Table of Contents

1. Introduction

1.1 Statement of purpose

Our project was a portable multispectral camera to be used for NDVI (Normalized Difference Vegetation Index) which is a simple graphical indicator to assess whether the target being observed contains live green vegetation [1]. Many places lack access to the services that offer NDVI imaging to analyze their crop fields and LANDSAT NASA/USGS imagery can often be too large or too lengthy to download and analyze on regular home computers (even more in rural environments where internet speeds reach 39.01 Mbps average) [2]. This project was inspired mostly by LANDSAT NDVI images and partly by the Instructables article "DIY Plant Inspection Gardening Drone" [3]. Ultimately, we approached this problem by building a portable, handheld unit that can be deployed by anyone of varying technological backgrounds (I.E. farmers in places that suffer from the information gap in 3rd world countries or rural America) We also set out to make the project as open source (MIT Licensing), cheap (~$200), and ergonomic as possible.

Our project is based around the Jetson Nano Developer Kit, the PiNoIR camera, the Adafruit GPS Breakout Board and OpenCV. The PiNoIR camera has a blue/green polarizing filter to capture only the red light coming into the camera and has its IR cut

filter removed to read only NIR (Near InfraRed) light reflected off objects. The system then uses the Jetson Nano to handle the processing and algorithms of the image analysis with OpenCV and the Adafruit GPS Breakout Board geotags the image for later reference and analysis.

1.2 Features

The system features live viewing through a 640x480p (scalable up to 3280x2464) touchscreen monitor with GPS telemetry and the ability to see the NDVI images in real time. The system also 1920x1080p24FPS video recording and 3280x2464 still frame images with live geotagging. The user also has the option to add different modules such as an IR flash (feature implemented into software but not physically native to the design), an extra GPS antenna, power over 5V 4A barrel jack, and conventional shutter camera harness for use with high altitude drone platforms or even aircraft (possible, yet not rated or tested).

2. Design

2.1 Systems Diagrams

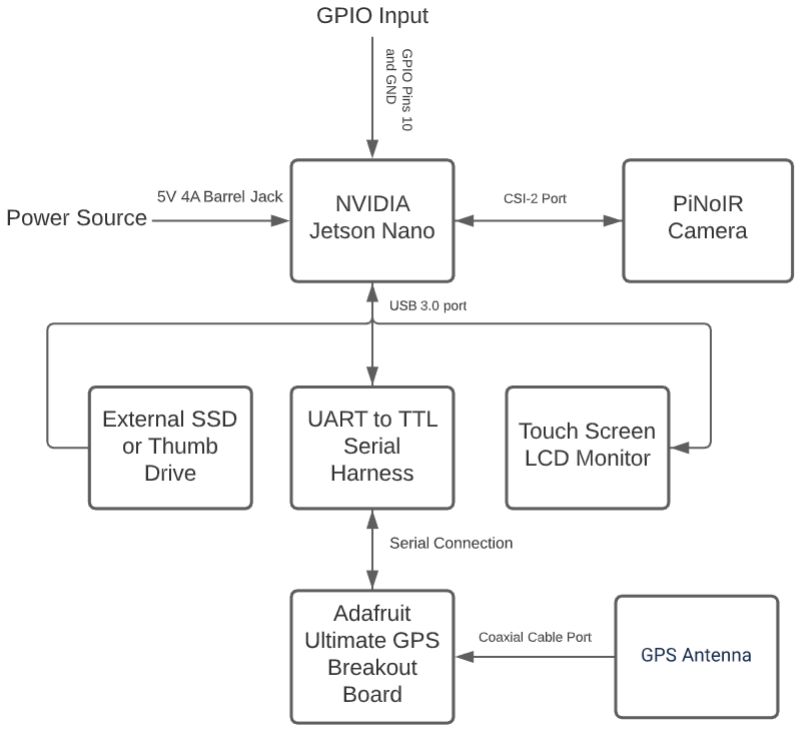

Figure 1. Systems Block Diagram

2.2 Systems Overview

2.2.1 PiNoIR Camera

This project uses a PiNoIR camera that is modified to capture the specific EM spectrum of IR light. The CCD board, which is based off
the IMX219 sensor, has the IR cut filter removed and a blue/green filter attached to capture the RGB/BGR light and to cut the red values out. The IR filter is removed in order to see the NIR bands of light. The camera is connected to the Jetson over the CSI MIPI port on the Jetson Nano.

2.2.2 NVIDIA Jetson Nano

The Jetson Nano is the "brain" of the operation and handles everything all the input/output of all the sensors and systems, as well as using computer vision software to capture, analyze and geotag frames that the user tells it to take. The board features 128 Maxwell cores and Tegra architecture which makes it the perfect fit for this computer vision-based project. Ultimately the basis of this is using OpenCV 4.5.0 to process the images and a nifty python script to save, geotag and mark images for later viewing and further analysis. The Nano also holds all the IO for the shutter harness for drone use, the button to signal camera frame capture, the HDMI and USB ports for the GPS module and serial connections, and the MIPI camera cable for interfacing with the camera.

2.2.3 Adafruit GPS Breakout Board

The Adafruit GPS Breakout Board has the MTK3339 module which is a GPS module that features a built-in ceramic antenna and 10 Hz updates on the GPS network. The module is specifically built for makers and enthusiasts and is perfect for low-cost GPS applications. We use this module to capture the GPS location of the camera for geotagging captured images and to let the user have full situational awareness of where they are analyzing and deploying the camera.

2.2.4 LCD Touchscreen Monitor

The LCD Touchscreen Monitor is a generic 1280x960p touchscreen monitor for use with projects like this one. The monitor is a generic rebranded monitor that features on board controls for brightness, volume and touch interfaces and a low-profile package for use in prototyping and maker/hobbyist use. This touch screen monitor will be the main human interface device and will be the bridge between what the sensors analyze and what the operator makes sense of this analysis. It displays a live NDVI feed from the camera, as well as having regularly updating GPS telemetry for the operator to know where they are using the camera and where to look in future agricultural operations.

3. NDVI process

3.1 Background of NDVI

The NDVI process is based off Near-infrared spectroscopy which is a spectroscopic method that uses the near-infrared region of the electromagnetic spectrum [4] to delineate the health of plants by measuring the amount of Near InfraRed (NIR) light reflected off the leaf cells as they re-emit the solar radiation in the near-infrared spectral region [1]. This is caused by the chlorophyll pigments which absorb red (visible light) and cell structures of plants to reflect off NIR light. Unhealthier plants and most non-plant material tend to return more red light as compared to most healthier plants which return more NIR light [1]. Ultimately the more leaves a plant has, the more chlorophyll it has, which in turn means there is more NIR light being reflected. The basis of creating an index for measuring the plants of health is an algorithim as follows,

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)}$$

Figure 2. NDVI Calculation [1]

where NIR denotes the spectral measurements of NIR and red (visible) light [1]. By assigning each pixel with a set of values of NIR and red light, a false color map of the plants health in a region can be created and viewed just as if one was viewing a thermal or infrared heat map.

3.2 Application to project

The way we are applying this algorithm to our project is quite simple and requires little to no processing power. The camera is specially modified to only capture the NIR and red (visible light) spectrum. Doing this we create a videocapture instance in OpenCV which grabs the feed of the camera and by using the .read() function we create a 3x3280x2464 NumPy array. Using the .split() function, we capture the blue, green and red values from the camera (which in this case is only red and NIR) and setup the algorithm to calculate the false color map. Each color channel is then assigned a part in the NumPy array and the “depth” of the 3d array serves as each color channel of the image. The algorithm pulls its respected values from the NumPy array and then each returned value is assigned a false color for the false color map. Then the frame is shown to the user on the screen with the .imshow() function and the process is repeated again for every frame captured by the camera.

4. Results

4.1 From the preliminary tests of using an NDVI algorithm

Below is an image created using the NDVI algorithm (Figure 3.) while following the guide "Calculate NDVI Using NAIP Remote Sensing Data in the Python Programming Language" (courtesy of EarthLab)[5]. We are still trying to grab the resulting image files from the camera, as we can not just pull out the SD card and copy the image due to formatting and partition difference between Ubuntu 18.04 LTS and Windows so we are unable to show our image results from the Jetson, but below is an example of using downloaded LANDSAT images and data to apply the same algorithm we used on the Jetson in a more controlled environment (based off of EarthLab).

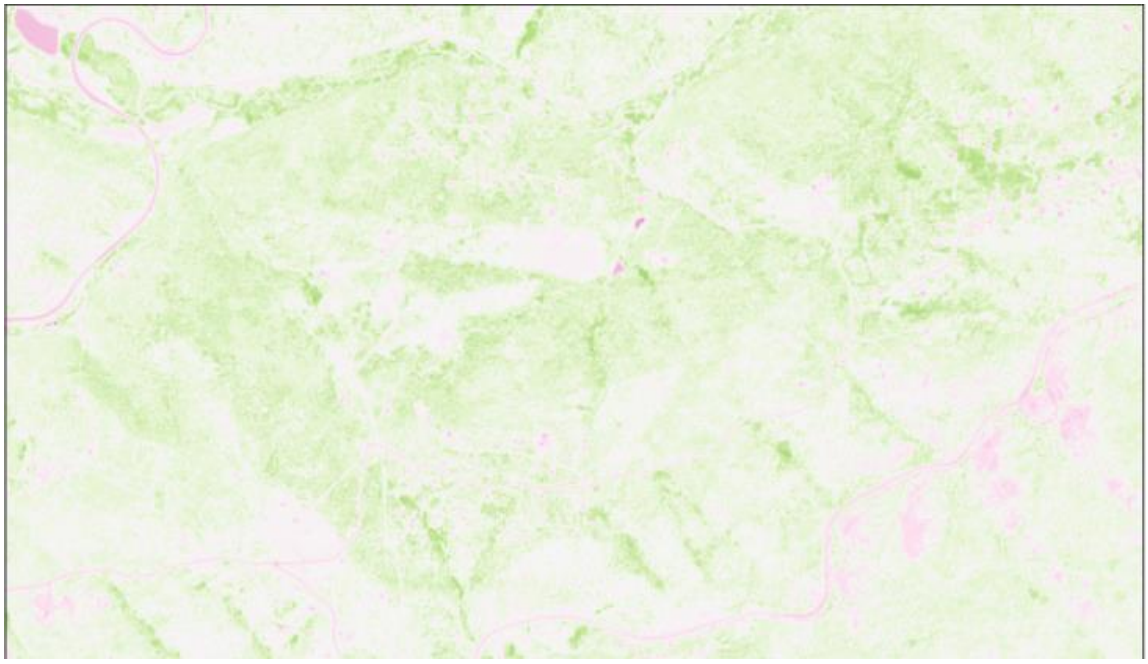

Figure 3. NDVI image using algorithm given to us from Earth Lab [5]

4.2 NDVI Results

Results incomplete.

5. Future Considerations

5.1 Datalink

The original focus of this project when brought up was to mount it on a drone and perform autonomous NDVI analysis on a cheap drone platform. However, due to budget and time constraints, we instead opted to design a handheld platform while adding the ability to mount to any drone platform. This cost mainly came from trying to devise a way for the operator to view what the sensor is viewing from a

long range. Some considerations made were using satellite communication over the Iridium satellite network, sending telemetry over 4G cellular communications, and building a completely new peer to peer duplex datalink network based off a 2.4Ghz long range Wi-Fi ground station (like what DJI drones have).

5.2 New Carrier Board

The Jetson Nano module communicates over a SODIMM 288 pin connector to the carrier board which is like what a laptop ram module uses. Another consideration for the future of this project is to try to integrate all the sensors and human interface devices into one carrier board. This would require some knowledge in SMD soldering, IO controllers and motherboard design, but doing this will greatly reduce the cost of manufacturing and would allow us to have a smaller form factor camera. We would integrate the GPS module (based off the MTK3339 module), the camera module (hopefully this time with a proprietary sensor package based off the IMX219 with autofocus and zoom) and the screen into one board. This is a challenging prospect and requires knowledge of electronics and computer design way past our focus, yet it is still a consideration we would make.

5.3 AI

One final design consideration we would've have implemented given the time is taking advantage of TensorFlow and TensorRT packages that the Nano SDK comes with to be able to read and analyze the data the same way a human would. We would train the AI with different pictures of NDVI correlated with humidity, temperature, water and irrigation, and UV index to inference these values from an NDVI image captured by our sensor package and output them to an "interactive" map uploaded to a web server where the user can click on different parts of the heatmap and see these values that were inferenced. However, this requires more sensor modules such as a temperature sensor and humidity sensor to inference the water and irrigation values. Another challenge is building an actual front end to the web server itself, not to mention that building the backend requires a vast amount of information which is not readily available.

6. Challenges

6.1 Problems

The challenges that we came across were as follows.

- Communications between GPS Module and carrier board
- GStreamer video position error
- Overheating issues

The solutions for these problems are outlined in 6.2.

6.2 Solutions

We fixed the problems labeled above respectively.

- Instead of trying to communicate between the Jetson and the GPS module through UART or built in GPIO serial, we instead used a USB-TTL adaptor and communicate over the /dev/ttyUSB0 serial to send and receive messages from the GPS module itself. This however has a caveat as we lost a USB port for diagnostics and were left with 3 USB 3.0 ports instead of 4. This left us with less possibility of installing an external drive to grab the photos.
- The GStreamer video position error was caused by a fault in the default GStreamer and OpenCV that was bundled with the NVIDIA Jetson Nano. This was fixed by rebuilding OpenCV and GStreamer to its latest versions. However, once the code is running once, the GStreamer error shows up again and a full rebuild of OpenCV is required to run it again. This makes it difficult to grab any results and we may have to wait for OpenCV to push a fix since the problem is still open on GitHub. [6]
- The overheating issue was solved simply by adding a PWM fan onto the heatsink. This however leaves higher power draw and more danger to the operator.

7. Appendix

7.1 Citations

[1] “Normalized difference vegetation index,” Wikipedia, 04-Sep-2020. [Online]. Available: https://en.wikipedia.org/wiki/Normalized_difference_vegetation_index. [Accessed: 09-Dec-2020].

[2] N. F. Mendoza, “Many US rural areas still suffer slow internet speeds,” TechRepublic, 19-Sep-2019. [Online]. Available: https://www.techrepublic.com/article/many-us-rural-areas-still-suffer-slow-internet-speeds/. [Accessed: 09-Dec-2020].

[3] Imetomi and Instructables, “DIY Plant Inspection Gardening Drone (Folding Tricopter on a Budget),” Instructables, 09-Sep-2019. [Online]. Available: https://www.instructables.com/DIY-Plant-Inspection-Gardening-Drone-Folding-Trico/. [Accessed: 09-Dec-2020].

[4] “Near-infrared spectroscopy,” Wikipedia, 04-Dec-2020. [Online]. Available: https://en.wikipedia.org/wiki/Near-infrared_spectroscopy. [Accessed: 09-Dec-2020].

[5] “Calculate Vegetation Indices in Python,” Earth Data Science - Earth Lab, 17-Feb-2020. [Online]. Available: https://www.earthdatascience.org/courses/use-data-open-source-python/multispectral-remote-sensing/vegetation-indices-in-python/. [Accessed: 09-Dec-2020].

[7] Opencv, “cv::VideoCapture::set does not work when using GStreamer · Issue #10324 · opencv/opencv,” GitHub. [Online]. Available: https://github.com/opencv/opencv/issues/10324. [Accessed: 10-Dec-2020].

7.2 Code

```
1. import board
2. import serial
3. import adafruit_gps
4. import cv2 as cv
5. import numpy as np
6. import time
7. import busio
8.
9. #set a gps instance for initialization and live video
10.  uart = serial.Serial("/dev/ttyUSB0", baudrate = 9600, timeout = 3000)
11.  gps = adafruit_gps.GPS(uart)
12.  GSTREAMER_PIPELINE = 'nvarguscamerasrc sensor_mode=0 ! video/x-
   raw(memory:NVMM), width=3280, height=2464, format=(string)NV12,
   framerate=20/1 ! nvvidconv flip-method=0 ! video/x-raw, width=600,
   height=480, format=(string)BGRx ! videoconvert ! video/x-raw,
   format=(string)BGR ! appsink'
13.  but_pin = 18
14.  GPIO.setmode(GPIO.BOARD)
15.  GPIO.setup(but_pin, GPIO.IN)
16.
17.  #define a gps intitialize ffunction
18.  def initialize():
19.      gps.send_command(b"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
20.      gps.send_command(b"PMTK220,1000")
21.      gps.update()
22.          if not gps.has_fix()
23.          print("gps finding fix...")
24.          continue
25.      print("+" * 40)
26.      print('Fix timestamp: {}/{}/{}
   {:02}:{:02}:{:02}'.format(gps.timestamp_utc.tm_mon,
   gps.timestamp_utc.tm_mday, gps.timestamp_utc.tm_year,
   gps.timestamp_utc.tm_hour,  gps.timestamp_utc.tm_min,
   gps.timestamp_utc.tm_sec))
27.      print('Latitude: {} degrees'.format(gps.latitude))
28.      print('Longitude: {} degrees'.format(gps.longitude))
29.      print('Fix quality: {}'.format(gps.fix_quality))
30.      print('# satellites: {}'.format(gps.satellites))
31.      if gps.satellites is not None:
32.          print("# satellites: {}".format(gps.satellites))
33.      if gps.altitude_m is not None:
34.          print("Altitude: {} meters".format(gps.altitude_m))
35.      if gps.speed_knots is not None:
36.          print("Speed: {} knots".format(gps.speed_knots))
37.      if gps.track_angle_deg is not None:
38.          print("Track angle: {} degrees".format(gps.track_angle_deg))
39.      if gps.horizontal_dilution is not None:
40.          print("Horizontal dilution:
   {}".format(gps.horizontal_dilution))
41.      if gps.height_geoid is not None:
42.          print("Height geo ID: {} meters".format(gps.height_geoid))
```

```
43.      print("+" * 40)
44.  #define a gps display function
45.  def display():
46.      last_print = time.monotonic()
47.      while True:
48.          gps.update()
49.          current = time.monotonic()
50.          if current - last_print >= 1.0:
51.              last_print
52.              if not gps.has_fix:
53.                  print("lost fix, re-acquiring")
54.                  continue
55.              print("+" *40)
56.              print('Fix timestamp: {}/{}/{}
    {:02}:{:02}:{:02}'.format(gps.timestamp_utc.tm_mon,
    gps.timestamp_utc.tm_mday, gps.timestamp_utc.tm_year,
    gps.timestamp_utc.tm_hour,  gps.timestamp_utc.tm_min,
    gps.timestamp_utc.tm_sec))
57.              print('Latitude: {} degrees'.format(gps.latitude))
58.              print('Longitude: {} degrees'.format(gps.longitude))
59.              print('Fix quality: {}'.format(gps.fix_quality))
60.              print('# satellites: {}'.format(gps.satellites))
61.              print('Altitude: {} meters'.format(gps.altitude_m))
62.  # define camera run function
63.  def run():
64.      cap = cv2.VideoCapture(GSTREAMER_PIPELINE, cv2.CAP_GSTREAMER)
65.      while True:
66.          #read the videocapture instance as a NumPy array
67.          image = cap.read()
68.
69.          #split the b,g,r color channels
70.          b, g, r = cv2.split(image)
71.
72.          #calculate NDVI
73.
74.          #bottom of ndvi algorithim
75.          bottom = (r.astype(float) + b.astype(float))
76.          bottom[bottom == 0] = 0.01
77.
78.          #perform ndvi algorithim
79.          ndvi = (r.astype(float) - b) / bottom
80.          ndvi = ndvi.astype(np.uint8)
81.          # Display
82.          cv2.imshow('image', ndvi)
83.          #take photo if button is pressed
84.          if if GPIO.wait_for_edge(but_pin, GPIO.FALLING):
85.              #save and label the image with the GPS coordinates
86.
    cv2.imwrite(("/home/uiucagdrone/Desktop/output/ndvi.POS:{},{},{}/Date:{
    },{},{}/Time:{},{},{}".format(gps.latitude, gps.longitude,
    gps.altitude_m, gps.timestamp_utc.tm_mon, gps.timestamp_utc.tm_mday,
    gps.timestamp_utc.tm_year,  gps.timestamp_utc.tm_hour,
    gps.timestamp_utc.tm_min, gps.timestamp_utc.tm_sec) + ".png", ndvi))
87.          else:
```

```
88.             continue
89.         # If we press ESC then break out of the loop
90.         c = cv2.waitKey(7) % 0x100
91.         if c == 27:
92.             break
93.
94.  #actual main code
95.
96.  #initialize gps
97.  gps.initialize()
98.  print('ready to go')
99.  #give time for the system to set itself up
100. time.sleep(5)
101. # start the multiproccessing
102. if True:
103.     p1 = Process(target = display)
104.     p1.start() #to display constantly changing gps data for the used
105.     p2 = Process(target = run)
106.     p2.start() #to display the actual NDVI through the viewfinder
107.     p1.join()
108.     p2.join()
109.
110.
```