Transformation of Raspberry Pis to Detect Fires for the Fire Urgency Estimator in Geosynchronous Orbit (FUEGO)

Woodrow Wang,[1*] Eugene Tian,[1] Nicolas Captier,[2] Robin Lafever,[3] Carl Pennypacker[3]

[1]College Station High School,
4002 Victoria Ave, College Station, TX 77845, USA
[2]University Pierre et Marie Curie, 4 Place Jussieu, Paris 75005, France
[3]Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94705, USA

[*]To whom correspondence should be addressed: E-mail: woodywang153@gmail.com

11 August 2015

Abstract

With a recent surge in forest fires throughout California, the demand for a reliable fire urgency estimator has dramatically risen. This study illustrates the transformation of cheap Raspberry PiCamera modules into efficient fire detectors that can be mounted onto airplanes and drones. By scanning pixel properties of infrared and visible light images, the FUEGO PiCameras' images can rapidly alert local fire departments of potential fires. In order to be placed in production, the system must meet three critical requirements: (i) the system is motile and not restricted by any wires, (ii) the system easily mounts to an airplane or UAV, and (iii) the system identifies fires with minimal false alarms so as not to waste resources. At the moment, the FUEGO PiCameras examine heavily processed images with perhaps slightly distorted spectral properties; ideally, extracting raw data images would more accurately display the true spectral properties of the image. Regardless, at an affordable cost, the current Raspberry Pi system accurately identifies fires with minor deficiencies.

1 Introduction

As forest fires become a daunting concern for life in California, researchers at the Lawrence Berkeley National Laboratory (LBNL) have sought to develop an efficient fire detection system. The recent surge in conflagrations across California exhorts researchers to take action.

The Fire Urgency Estimator in Geosynchronous Orbit (FUEGO) seeks to place a relatively ordinary infrared satellite into orbit to scan a fixed geographical latitude for fires (Pennypacker 2014). Without notable funding and an urge to move on from the brainstorming phase, the FUEGO team has chosen to begin implementation of an altitude-layered detection system beginning with Raspberry Pis (RPis). Raspberry Pis are low cost, credit-card sized computers with Python compatibility. They can easily be manipulated to take high resolution pictures with different kinds of cameras and filters. These light and portable computers work well while mounted on planes or drones due to their size and facile connectivity. Hence, this project chronicles the design of a fire detection assembly using cost-friendly Raspberry PiCameras.

As the PiCamera's role is to take pictures while mounted on an unmanned drone or plane, we seek a system capable of being remotely controlled via a laptop. Thus, we experiment with designs for feasible mounting mechanisms for an octocopter provided by LBNL's nuclear science research division and a Cessna 172. With funding severely restricting purchases of legitimate infrared detectors, which could cost many thousands of dollars, inference of spectral properties in individual pixels is necessary. In order to identify a forest fire, we search for a unique spectral fingerprint to discern fire from reflected light. The PiCameras mount onto airborne systems and snap pictures seconds apart from each other. Then, software runs through the photos pixel by pixel to examine properties that might identify as a fire.

In this project, we are using two Raspberry Pis. The Pi referred to as the VisiPi takes pictures in the standard visible light spectrum; the other Pi referred to as the InfraPi has a camera board that permits infrared light as well as visible light. By adding a Wratten 87 filter to only permit light greater than 770 nm and essentially block visible wavelengths, we create an affordable, makeshift infrared camera.

The paper is organized as follows. In Section 2, we describe the configuration of the RPis to take reliable pictures through remote control and the analysis of the Pis' physical properties. In Section 3, we briefly discuss the implications of mounting the RPis to a drone or plane and the parameters surrounding the design. In Section 4, we present the thoughts behind the still developing Python software to discern fires in the RPis' photos. Finally, in Section 5 we discuss and summarize the results of our work while presenting future issues that require resolution.

2 Configuration of the Raspberry Pis

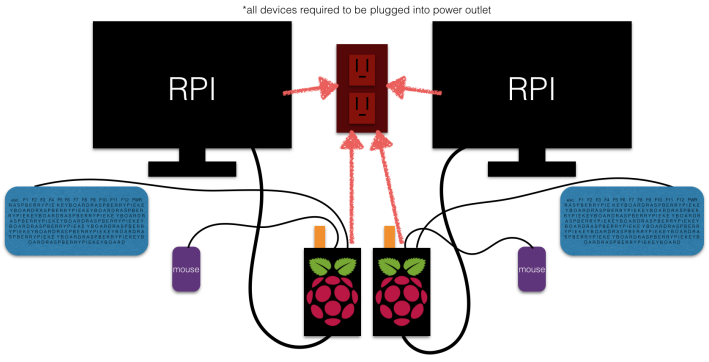

Figure 1: Simplified schematic view of the original Raspberry Pi setup. Credit: Sormeh Yazdi, Wellesley College.

The original setup of the Raspberry Pis was bulky and not feasible for flight. It required a desktop, keyboard, and mouse for each Pi, as well as multiple HDMI cables. The initial setup also required AC adapters for the RPis, which would not necessarily be available on a drone or plane.

With reduction of wires in the setup required, we sought to establish a remote desktop connection to the RPis. After thoroughly scavenging around Raspberry Pi tutorials, we initially thought the best method of remotely accessing the Pis would be through multiple Ethernet connections. Since WiFi is not necessarily readily available universally, accessibility may be hindered if our cameras fly above a remote mountain. Thus, by setting up static IP addresses for both the Pis and a laptop, we could access both Pis through two separate remote desktop connections as long as Ethernet cables were available.

At the time of implementation, this solution seemed acceptable; with the implementation of an external battery pack, we could now execute the code for taking pictures anywhere without the requirements of a power outlet and heavy monitors. This setup was actually tested on a flight through primitive taping of Ethernet cables along the belly of the plane, but the arduous work of adhering and removing the cameras and cables from the plane probed new development.

With small WiFi dongles on the Raspberry Pis, a device to device connection or ad-hoc network with the Pis can be established. Essentially, the ad-hoc network works similarly to a Bluetooth connection between one's phone and a laptop. After several days of failure, we successfully forced the RPis to emit an ad-hoc network. At this point, we can access the Pis without the need of WiFi routers or any potentially tangling wires. With a decentralized wireless network that does not require any existing infrastructure, the RPis are not limited by the burden of carrying routers and Ethernet cables. With the ad-hoc network in place, the system can truly be remotely controlled from any location on Earth.

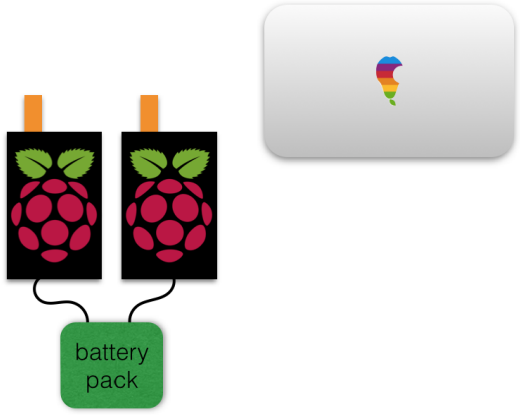

Figure 2: Schematic view of the condensed Raspberry Pi system capable of remote access through an ad-hoc network. The orange tabs protruding from the Pi represent the WiFi dongles required for emission of the ad-hoc network. Credit: Sormeh Yazdi, Wellesley College.

2.1 Physical Properties of the Pi

Before testing the Raspberry Pis in the field with legitimate fires, we had to calculate their physical properties. By placing a meter stick on the ground and taking pictures from varying heights above the target, we calculated the horizontal, vertical, and diagonal field of view to a high degree of accuracy. Furthermore, using simple trigonometry and the respective RPi's field of views, we found the size of individual pixels at varying altitudes of flight. The results are summarized as follows.

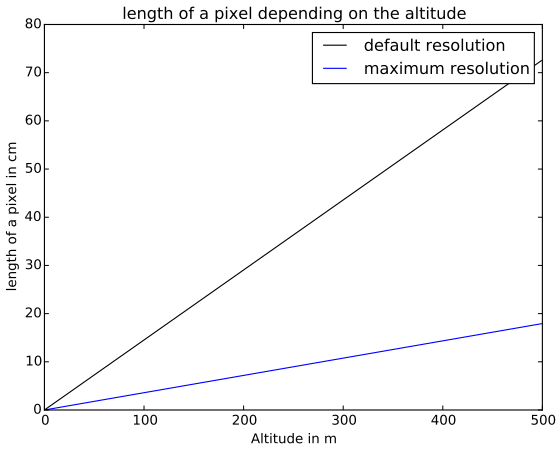

Figure 3: Graph of individual pixel size versus altitude, illustrating the fine resolution the PiCameras can reach while being remarkably cost-effective. The default resolution line shows pixel size at varying heights when the images taken are 640 x 480 pixels; likewise, the maximum resolution line depicts pixel size when images are 2592 x 1944 pixels.

The specifications of a variety of properties are summarized in Table 1.These results were confirmed by measurements taken from FUEGO's first trial flight (Wang, et al.).

2.2 Python Code for PiCameras

The Raspberry Pis are Linux-based single board computers with Python programming compatibility. Using Python 2.7, we are able to create software that takes pictures with nearly one second intervals between photos.

The code has several parameters that currently require user input. Upon execution, the Python program asks for the length of time for taking pictures, the interval of time between pictures, and the name of the file. Potentially, if the system is mounted on a drone or unmanned system without the possibility of user input, these required parameters can be reduced to default settings; however, default settings preclude facilitated manipulation of useful variables during data collection.

The Python code also automatically creates a text file with timestamps of when each picture was taken. This is crucial for the image analysis software to align photos from the respective InfraPi and VisiPi. We attempted to configure a GPS module on top of the Raspberry Pi circuit board but ran into some issues while soldering the unit. Apparently, adding the Adafruit Ultimate Pi Hat GPS would prevent the transmission of our ad-hoc network, which is unacceptable in terms of operating the system. Thus, a novel GPS system is urgently required.

Table 1: PiCamera Properties and Specifications

Properties of the PiCamera	**Specifications**
Dimensions	25 x 20 x 9 mm
Weight	3 g
Sensor	OmniVision OV5647
Default sensor resolution	640 x 480 pixels
Maximum sensor resolution	2592 x 1944 pixels
Horizontal field of view	54 degrees
Vertical field of view	42 degrees
F-Stop/Aperture	2.9

3 Mounts for the PiCamera System

On the initial flight test, we affixed the two Raspberry Pis and the external battery pack onto a rectangular, flexible board. We then adhered the entire board to the belly of a Cessna 172 with military-grade tape. While the shoddy setup theoretically works, the longevity of such a system is suspect (Wang, et al. 2015). Applying and removing the tape proves to be an arduous hassle. Thus, we have begun designing mounts for a system that could be relatively easily built in a machine shop at any functional workshop or even the machine shop at the Lawrence Berkeley National Laboratory.

With the RPi's horizontal field of view greater than 50 degrees, the most suitable part of the airplane for mounting seem to be the footstep, strut of the landing mechanism, or the belly of the plane. If the mount is too bulky, it could induce life-threatening aberrational motions on the plane, severely compromising safety and the effective capturing of photographs. Safety, of course, is of utmost importance when designing a mount. Below are a few preliminary designs that seek to minimize any unbalancing of the plane.

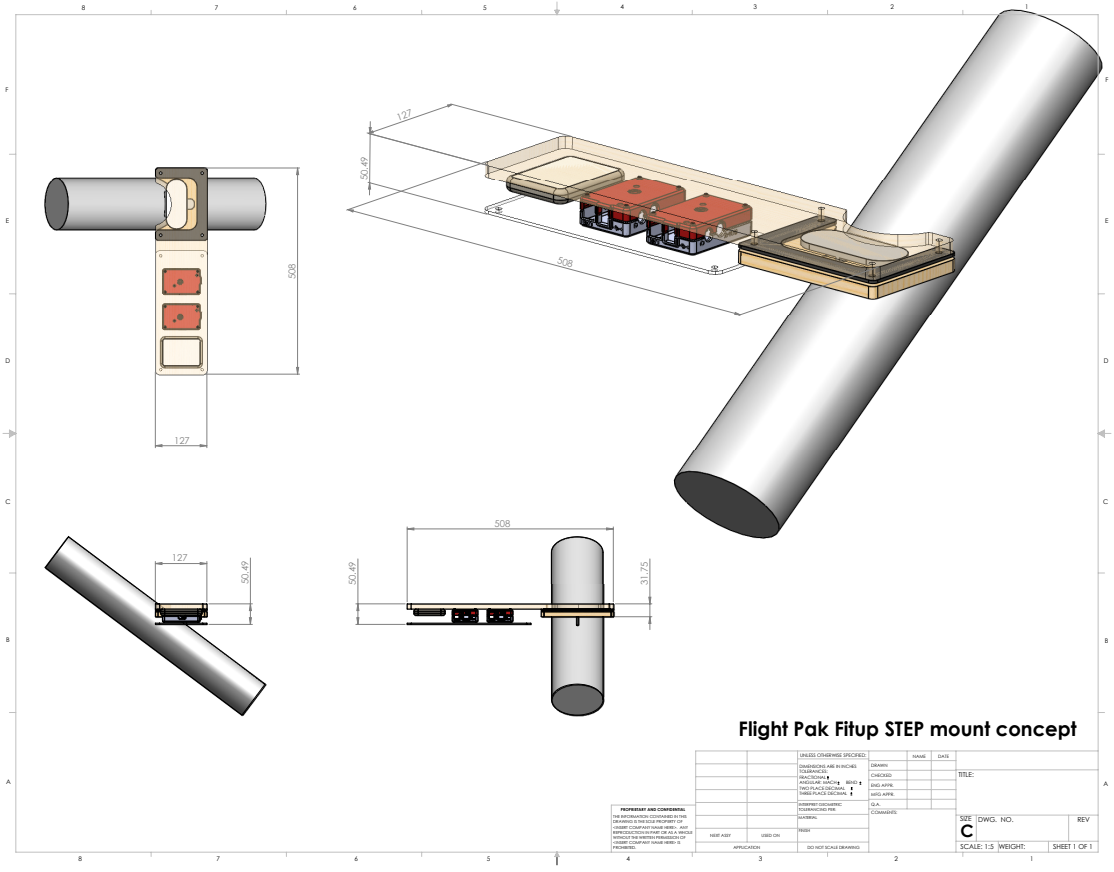

Figure 4: Schematic design of the mount for the footstep of a Cessna 172, with an upper view and multiple side views of the step mount. Credit: Robin Lafever, LBNL.

The step mount appears appealing because the footstep supplies a sufficient area for a sturdy clamp. The plank for the RPis would be placed in the air stream to reduce drag in the wind. With a streamline design, the mount in Figure 4 aims to reduce vibration in flight. Although, one apparent flaw is the inability to assure that the plank would not shutter minutely in the wind; even small vibrations could drastically compromise the quality of our images.

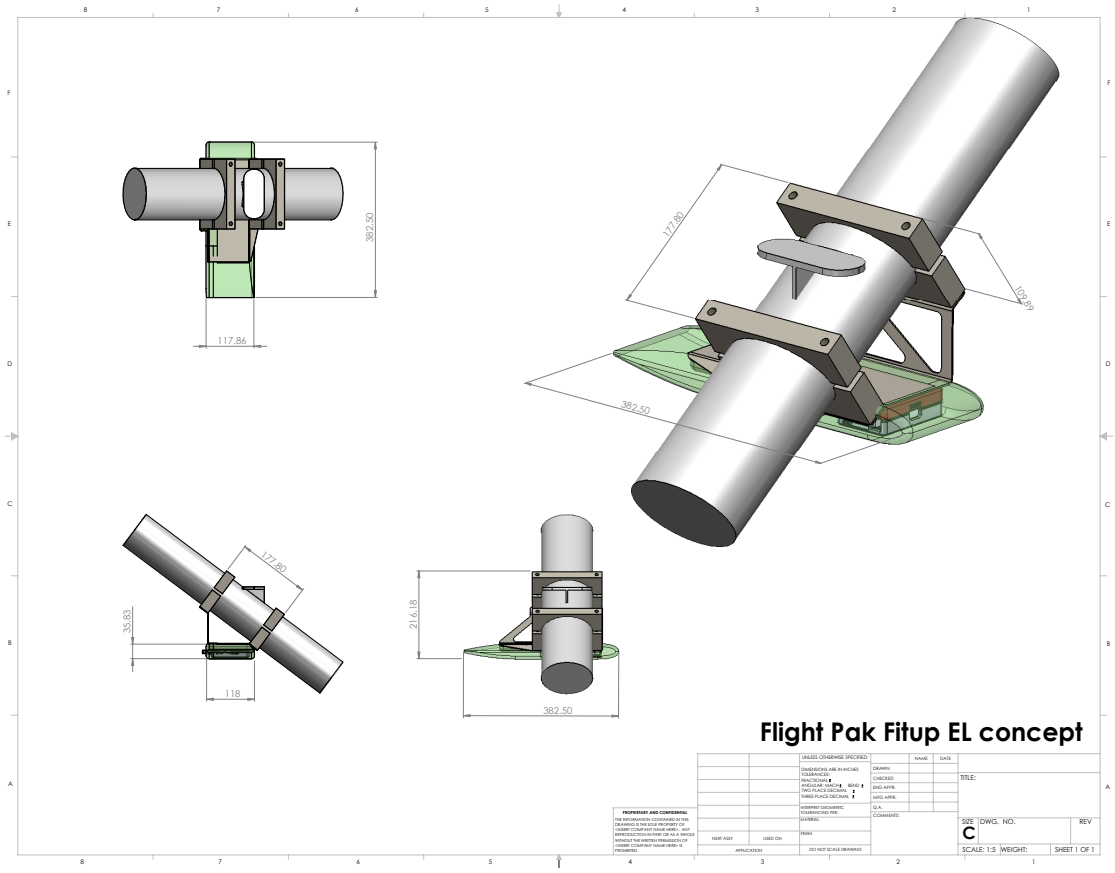

Figure 5: Schematic design of the mount for the strut of the Cessna 172's landing mechanism, with an upper, side, and front view. Credit: Robin Lafever, LBNL.

A mount on the strut of the landing mechanism appears favorable because it allows two clamps to clutch onto the struts. This design requires slightly more machining, for the angles must be measured perfectly to ensure tilted pictures are not taken. The RPis would be fitted into a tight, aerodynamic holder.

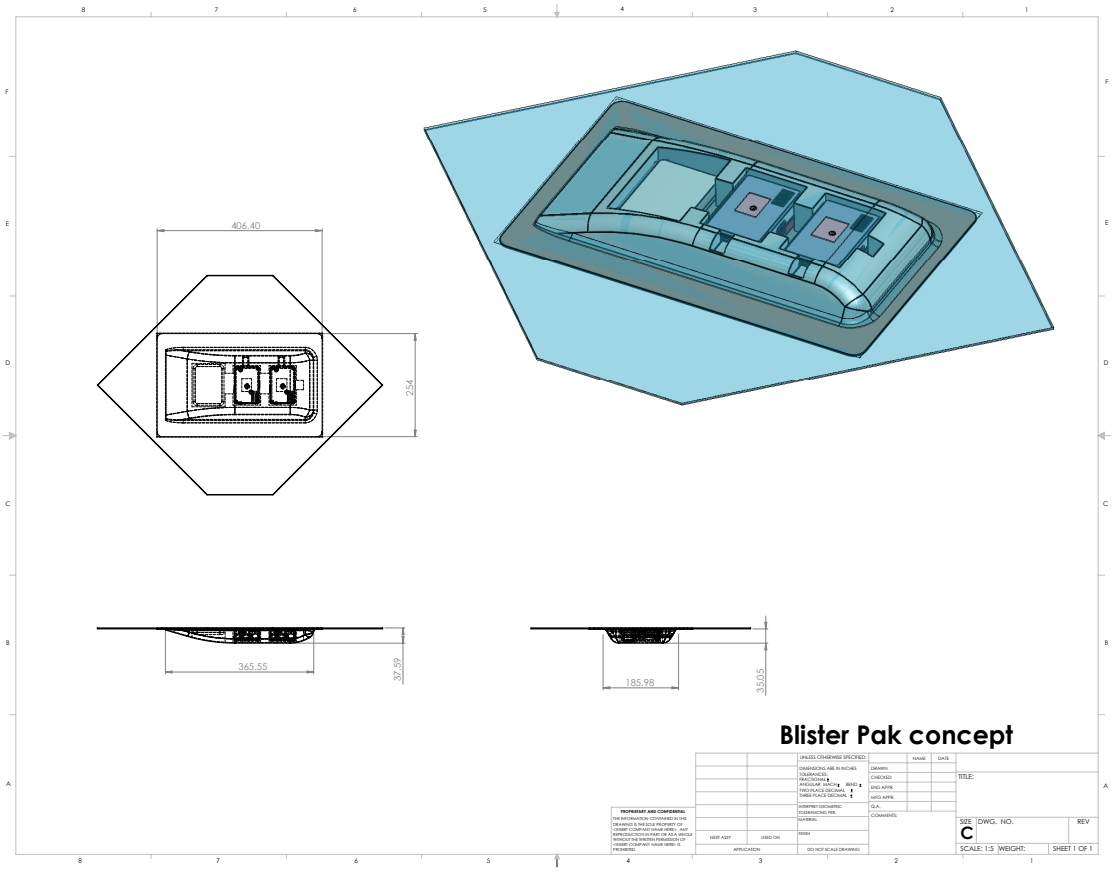

Figure 6: Schematic design of the mount for the underside of a plane. Credit: Robin Lafever, LBNL.

The final mock-up shows a mount designed for the belly of any aircraft. Using sturdy plates, the RPis would be set into a aerodynamic case and sealed upon the bottom of an airplane. If secured tightly to the plane, the RPis should take quality images lacking damage from flight vibrations. With a simple nut and bolt design, this mirrors an upgraded version of the assembly from our first flight. Following the initial flight, a protective case would be necessary due to unpredictable wind conditions. Loose wires and unexpected vibrations could easily prevent data capture. Thus, the "Blister Pak" design promotes protection for the RPis.

While these designs are in motion to be built, the lack of consistent funding precludes their physical prototyping construction.

4 Image Processing with Python Software

4.1 Development of the Algorithm

The processing software must meet two critical requirements:

1. The fire is correctly identified in each photo
2. False alarms are minimized by sufficient verification tests

Two image processing algorithms have been developed, courtesy of Nicolas Captier, University of Paris. Knowing that a fire should generally emit more infrared light than weakly reflected sunlight, we deduced that perusing the infrared photo for initial identification of suspect fires would be semi-reliable.

The original processing software, written in Python, works relatively effectively for identifying fires. First, the Python program separates the InfraPi's image into boxes of 10x10 pixels. Then, in the infrared image, the computer searches for clusters of pixels more intense than a specified threshold using this relatively simple Python line of script (Sahami, 2008):

```
Intensity=0.299*pix[x,y][0]+0.587*pix[x,y][1]+0.114*pix[x,y][2]
```

The following image shows the result of Python's scan for pixel intensity throughout the entire infrared picture. The program replaces any suspect fire pixels with bright red pixels. The lucid fire is clearly illuminated by the massive amount of red pixels in the vicinity of the fire; however, the reflective aluminum surface to the upper left of the fire appears glaringly bright to the software.

Figure 7: The original processing software's test for intensity in the infrared picture.The red pixels denote possible fire candidates.

After thorough examination of the standard wildfire's emission spectra, we have concluded that minuscule trace amounts of blue photons should be emitted from the flame. In order to remove the false alarms caused by reflective surfaces, the program performs a test on the homologous image taken by the VisiPi. The program narrows its search to only previously identified fire candidates; with a clearly visible red pixel, the program recolors any suspect pixels that have high counts in the red channel and low counts in the blue channel.

Figure 8: The finished result of the original processing software. The few red pixels on the fire denote a successful identification of the fire.

Although yielding a promising result, the beta software's algorithms are imperfect. The threshold counts of red and blue pixels in each respective channel must be manually set, and as of yet, we have no efficient way of automatically doing so. Furthermore, only about ten percent of the fire is correctly identified, whereas the rest is left as unsuspected candidates.

In an effort to perfect the system, we have established a new parameter for the second test placed on the images taken by the VisiPi. In our new processing software, we retain the first examination for high intensity pixels in the infrared photos.

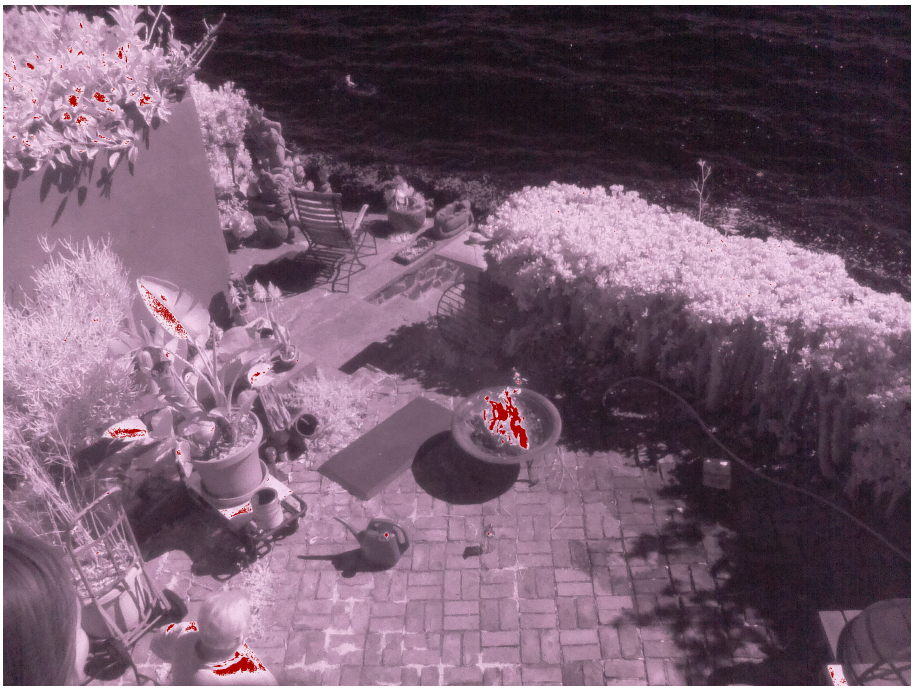

Figure 9: The result of the initial detection for intense pixels. The red pixels represent the program's identification of fire candidates.

With the highly reflective leaves misidentified as a fire, a new test is used to reduce false alarms. The new program sifts through the suspicious pixels of the infrared photo and makes sure the pixels fit this new parameter:

R >G >B

In general, after analyzing individual pixels from hundreds of photos of fire, we have deduced that red pixel counts must be higher than green pixel counts, which must be higher than blue pixel counts. Thus, after running the infrared photo through this trivial test, the following result yields in Figure 10.

Figure 10: The finished result of the new detection software. The red pixels clearly encompass most of the fire and no false alarms are alerted.

4.2 Flight Confirmation of Algorithm

On August 7th, 2015, we flew on a Cessna 172 to verify the processing software's effectiveness. With a flame about a half a meter wide in a target's backyard, we flew over several times to test the detection software from about 300 meters in the air. Unfortunately, the VisiPi became disconnected due to a loose power cable during takeoff. Regardless, we followed through with the flight test and searched for the staged fire.

Without pictures from the VisiPi to reduce false alarms, we had to rely on the InfraPi's images to detect fires. Thus, lowering exposure time to an optimal setting would be the only efficient way to reduce background sunlight. In Figure 11, various pictures at different exposures are displayed.

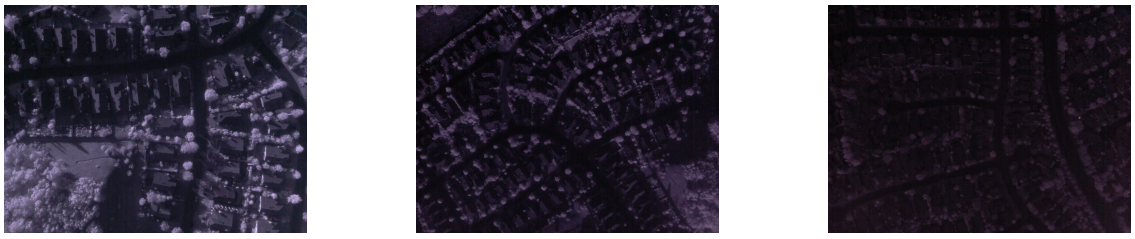

Figure 11: Flight images from an altitude of 300 meters, while the photo on the fire right seems completely black, its exposure time proves optimal. From left to right, the images were taken with exposure times of 3333, 333, and 33 μs, respectively.

As shown in the first image of Figure 11, background light and saturation of images is an extreme issue. At high altitudes, tree leaves can be highly reflective and appear bright in our processing infrared images. Hence, by reducing the exposure time to around 100 μs, we could winnow out the flame's infrared signal from the background reflected sunlight. Since the fire's light directly emits towards the camera's detectors while sunlight must be reflected, reducing exposure time increases the contrast between the two respective signals. The longer the camera lens is open, the more dimly reflected light would be detected; therefore, the extreme reduction of exposure time effectively isolates the fire's intense heat signal. Even without the VisiPi's images to reduce false alarms, the Python software found multiple images with the fire at low exposure times. Pictured in Figure 12 is one of many photos of target fires from an airborne test, and Figure 13 shows the Python software's detection of the fire.

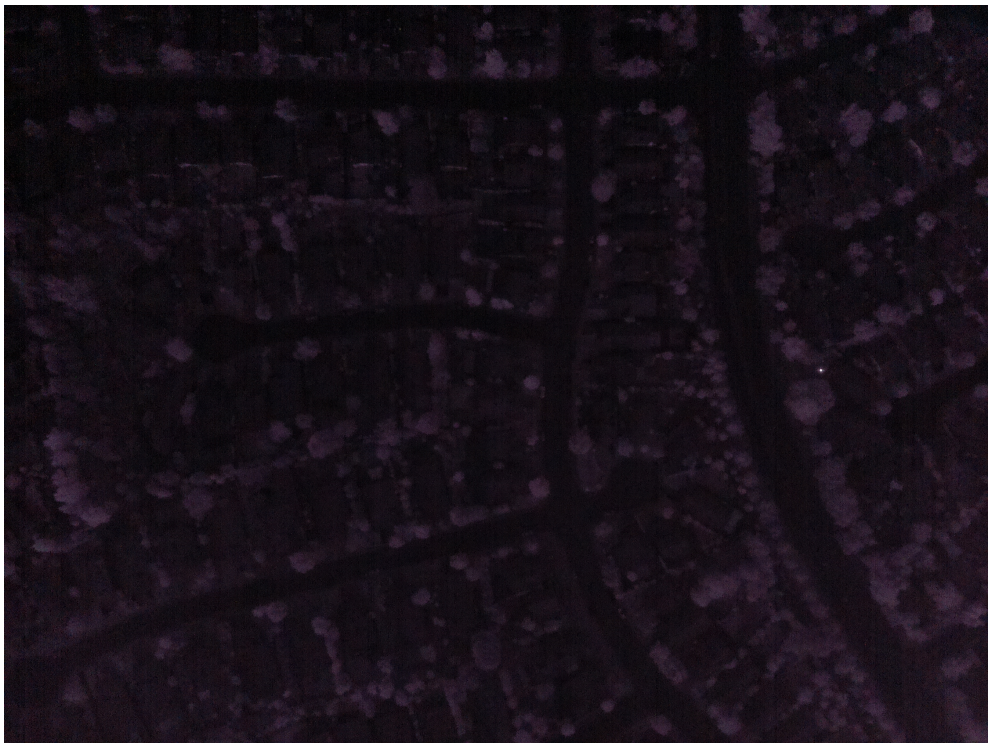

Figure 12: Initial picture before execution of the Python software to detect the fire. The fire can be seen as a distinct white dot in the middle of the right-hand side of the photo. This photo was taken with an exposure time near 50 μs.

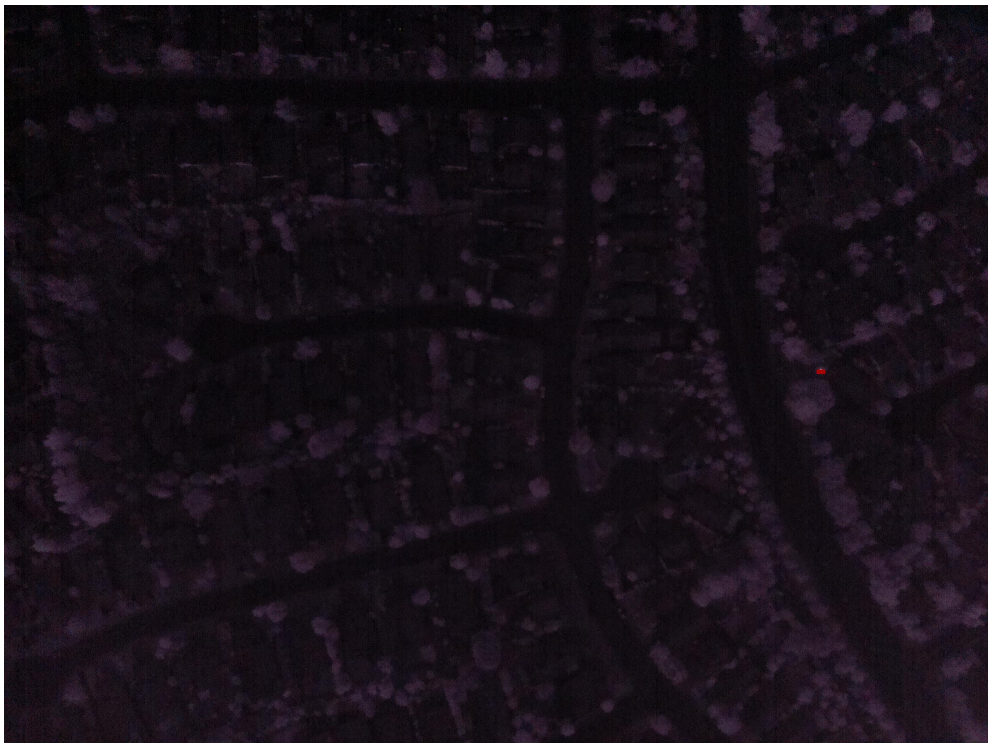

Figure 13: Product of our Python algorithm for detecting fires. The candidate fire is replaced by a lucid red pixel by the Python software. The red pixel denotes the target fire's location.

At a remarkably low exposure time, or quick shutter speed, most of the reflected light from the sun is reduced from the picture. Using Google Maps, we confirmed the target fire's location and verified the successful detection. Without any false alarms, the VisiPi's pictures were not required to identify the candidate fire. With proper reduction of background reflected sunlight, our Python procedure can easily distinguish fires.

5 Discussion and Conclusions

The experimental results appear to affirm the reliability of the new processing software. The new processing software identifies wildfires at a near perfect accuracy on the condition that red flames can be seen in the VisiPi's images. The PiCamera system can take pictures, align them, and identify fires if present. Having been verified by multiple flight tests, the accuracy of the system sparks new hope for early recognition of fires. A framework process for detecting fires can be represented as follows:

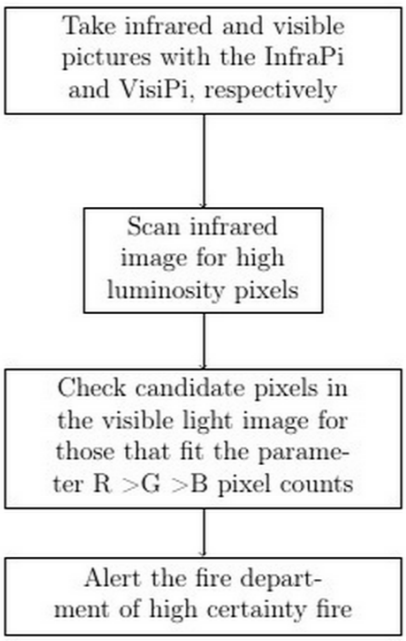

Figure 14: Flowchart of the FUEGO PiCamera fire detection system.

Notably, the prototype PiCamera fire detection system inspects heavily processed images from cheap camera boards. Ideally, the system should be redesigned to scan raw images with less issues of white balance and unnecessary processing. The project is hindered by these issues:

1. GPS System: When analyzing the initial pictures from FUEGO's first flight, we painstakingly had to manually match each image's location with Google Maps to find potential fires. Since we were simply looking for fires in target locations with no image analysis software in place, manual analysis of the pictures was arduous, yet necessary. With a real georeferencing GPS in place, each image would ideally have a stamp in the metadata with the exact coordinates of the image. Courtesy of Dr. Pennypacker, an Adafruit Ultimate Pi Hat GPS has already been purchased, but the installation of the hat seems to disable network configuration abilities through tampering with necessary GPIO pins. Hence, the Raspberry Pis are still left barren of any GPS system. Hopefully this issue can easily be resolved by perhaps reconfiguring the current Ultimate GPS Hat to work concurrently with the Pi, or installing a new system. In foresight, the location of an image is critical to the detection and extinguishing of a potentially fatal fire.

2. Image Saturation: When first analyzing images, we found that the pixels around fires and extremely reflective surfaces were completely saturated, meaning the pixel count for R,G, and B were all 255. Saturated pixels are hard to analyze because we essentially cannot differentiate the identity of saturated pixels from each other. Hence, we attempted to lower exposure times, creating less saturated pixels and images. If the lens is open for less time, less light will be allowed in. Lower exposure times resulted in better pictures, but we still had the issue of white saturated images in supposedly "infrared" pictures. If a flame appears white,

the computer erroneously assigns it high values of green and blue pixels, as well as the correct high number of red pixels, to make a white pixel. If we want to examine the spectral and physical properties of the fire's photons, high counts of green and blue pixels would deceivingly imply high counts of green and blue photons on the detectors. At first we blamed poor processing on saturation, but white balance became the dormant culprit.

3. Automatic White Balance: As a camera takes a picture, it automatically performs white balance to create an appealing picture for the human eye. Without white balancing the images on the InfraPi, pictures would only have pixels in the red channel, creating an unappealing, yet more scientifically accurate red image. What was originally blamed on saturation is now known to be an issue of white balance. The "infrared" photos are put through an automatic processing software in the cameras that balances the bright, intense pixels of the fire to white. Thus, the white pixels of the fire have large counts near 255 of pixels in the green and blue channels. As we know that few, if any, green and blue photons can get through the visible light filter, these pseudo pixels are clearly a result of processing, or camera tricks to appeal to the eye. We need to turn off white balance and get raw data from before the camera's preset processing; however, this proves slightly easier said than done. Turning white balance off requires one to set the gains of the camera, which are values the FUEGO team is unsure of. Thus, we look to utilizing raw Bayer data.

4. Raw Bayer Data: Since the camera photosensors are actually made of a Bayer array of little R,G,B sensors, analyzing the spectral properties of the light hitting the detectors is convoluted. Developed by Kodak, the Bayer filter is an array made 25 percent R, 25 percent B, and 50 percent G. The larger amount of green detectors mimics the ratio of green cones to blue and red cones in human eyes. Essentially, the raw data from a Bayer filter of photo-sensors retrieves a third of the light a normal camera would receive using three separate detectors and estimates the quantities of other pixel values through a process called interpolation. Simply put, the raw data that is demosaiced into RGB but not processed through white balance is the best data we can potentially obtain from the cameras. We hope to analyze the true spectral properties of the fire. As of now, we look for high intensity pixels in the infrared pictures; with raw data, we may be able to look for the pixels with the highest number of pixel counts in the red channel, instead of examining overall brightness. For scientific usage, a pre-processed image is far superior to an appealing image to the eye. Through much research, there seems to be an attribute in the "Picamera.array" module that allows raw Bayer data capture, which we have used; however, we have had trouble interpreting the messy raw data files.

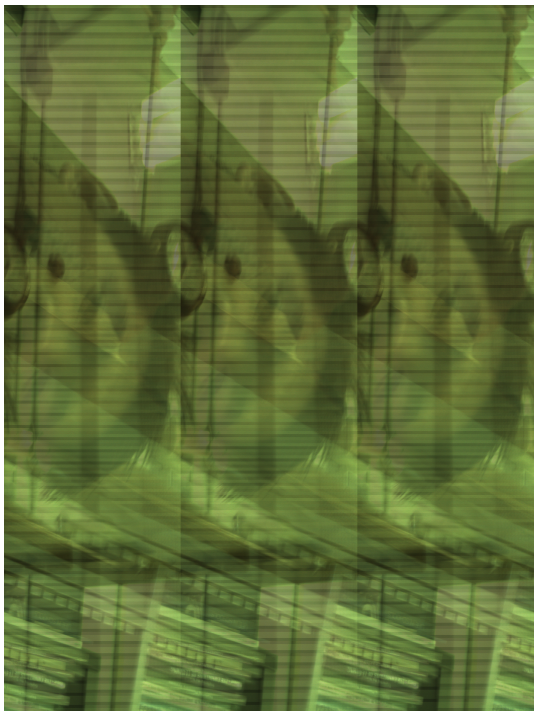

Figure 15: Unexpected result of raw data capture from Raspberry Pis. The image is heavily tinted green and separated into three panels for unknown reasons.

For an unresolved reason, the raw, unprocessed images appear unusually green; however, this makes sense as the Bayer filter consists of a mosaic pattern that is 50 percent green. Ideally, a raw image would provide a better indication of the true wavelengths of light that hit the PiCamera's detectors, but the supposedly pre-processed images prove difficult to be used analytically. Our results should not discourage future pursuit of raw data images. An image void of post processing such as white balance should supply spectral details in the pixel counts, which can easily be analyzed by a Python script. The next step to perfecting the FUEGO system is obtaining and analyzing clean raw data images.

Despite the multitude of unresolved issues, the system is stable and can effectively take pictures in a timely manner while accurately identifying fires. With more funding and greater public interest, even the elegant mounts presented in Section 3 can be built. Remarkably, simply by checking for intensity in an infrared image and confirming the parameter R >G >B, the cost-effective PiCameras can locate fires with high certainty. With a few tweaks to the system, the FUEGO RPi system may be able to detect a real wildfire in the near future.

Moreover, as affirmed by the FUEGO flight test, proper reduction of exposure time can accentuate the fire while dimming reflected sunlight. Even without the VisiPi's images in flight, the fire detection system worked, enhancing the reliability of FUEGO. The FUEGO PiCamera system can operate on land and in flight at an altitude of up to 300 meters. As the PiCamera's fire detection system hopes to bolster support for the much larger FUEGO unit, our results enlighten bright hope for the future of the project. If successfully implemented, the FUEGO system promises to stop forest fires around the world and better humanity's standard of living.

Acknowledgments

Special thanks to Carl Pennypacker for guiding us throughout the project and always being delightfully supportive. Courtesy of Dr. Pennypacker, we have been able to expand our scientific knowledge and help the environment. We are grateful for Robin Lafever's unwavering dedication to push the team forward and supply us with necessary materials. Again, we would like to acknowledge both Carl's and the Lawrence Berkeley National Laboratory's hospitality.

References

Pennypacker C., *FUEGO: a satellite system for rapid location of wildfires* (SPIE, 2014).

Sahami, Mehran., CS106A Stanford University lecture, 2008

Wang W., Captier N., Tian E. *FUEGO's First Flight*, 2015