UNIVERSITY OF SOUTHERN CALIFORNIA
DEPARTMENT OF AEROSPACE AND MECHANICAL ENGINEERING

AME-441a: SENIOR PROJECTS LABORATORY
Fall 2019

Fire Prevention System
Design Report

Andrew Rooney, Clayton Marceau, Davey Kim, Sophie Fast
acrooney@usc.edu

13 December 2019

Abstract

California has faced a substantial loss of housing due to rampant wildfires that have hit the area. Many home fires are not started due to direct contact from the wildfire but rather from traveling embers which can spread a fire over one mile from the main source. These spot fires spark from small embers, for which this paper introduces a novel system to extinguish them before posing a substantial threat. The fully autonomous and stand alone system uses thermal imaging detection paired with image recognition to precisely identify hot spots, and targets them with a two axis nozzle to douces the embers and eliminate any threat thereafter. After 15 weeks of product development, refinement, and testing a successful prototype was created, making California residents one step closer to being wildfire safe.

Introduction

Each year the relatively high temperatures and low rainfall during the summer months create conditions which allow for the ignition and spread of wildfires in much of the United States. Over the years, wildfire season has become increasingly devastating to fire prone areas as seen by the extensive burning of California where in 2018, 1.9 million acres were engulfed in flames resulting in 22,751 burned buildings and over $3.5 billion in damage.

One of the common misconceptions associated with building loss during wildfires is that they occur from direct flame contact by the main body of the wildfire. Research shows that buildings are lost due to the growth of initially small fires in or around buildings. Volatile winds that are a result of wildfires can transport burning embers up to one mile in all directions, which puts many properties in danger, even those not in the vicinity of the fire. As a result, some people are given little notice to evacuate and in some cases they even stay at their homes to fight the embers themselves with water hoses. This poses a serious threat to individuals and is not an effective way to keep the house safe and extinguish the problematic embers.

The Fire Prevention System was created in order to better protect properties from wildfire embers and reduce the risk of human casualties. The system consists of an autonomous fire detection and prevention unit that can be installed on the roof of any building. It is comprised of image sensor, servo control, and water pump subsystems that will be further detailed below.

The crux of this design is a point-and-shoot water hose that receives instructions from a heat detection system which locates the object(s) of interest (i.e. burning ember(s) on a roof). A two-axis water pump nozzle, whose angles can be manipulated according to the distance required, douces water to extinguish the object of interest. In order to determine the angles of the water pump nozzle, a projectile motion calculation was determined following the diagram shown in Figure 1.

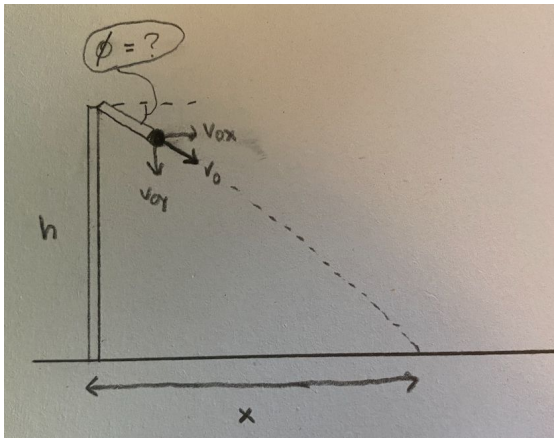

Figure 1: *Projectile motion diagram with angle ϕ unknown. Note that height, h, distance, x, and initial velocity, v0, is known. Angle ϕ represents the pitch angle denoted in the following sections.*

The projectile in the y direction can be represented by Equation 1,

$$y = y_0 + v_{0y} * t + 1\frac{1}{2}gt^2 \quad (1)$$

and the projectile in the x direction can be represented by Equation 2.

$$x = v_{ox} * t \quad (2)$$

Knowing that $y_0 = h$, $v_{0x} = v_0\, cos(\phi)$, and $v_{0y} = v_0\, sin(\phi)$, Equation 2 can be solved for t and substituted into Equation 1 to yield

$$0 = h(1 - sin^2(\phi)) + x \quad (3)$$

where,

$$a = \frac{g*x^2}{2*v_0^2} \quad (4)$$

Equation 3 can be solved using trigonometric identities to yield its final form in Equation 5.

$$\phi = \frac{1}{2}(arccos(\frac{a-h}{\sqrt{h^2+x^2}}) \cdot \quad (5)$$

The distance, x, is determined by the hottest object detected by the camera, which will be articulated in the following sections. Each test trial will produce a unique distance x, which will be inserted into Equation 5 to determine the desired pitch angle, ϕ, of the water pump nozzle.

Position-controlled servo motors were employed to manipulate the water pump nozzle to its desired pitch angle, ϕ, and planar angle, θ. Given that servo motors are controlled via pulse width modulation (PWM), the input to this subsystem (in degrees) must be converted to the appropriate PWM signal for the servos to operate. An algorithm was developed to translate the desired angle (in degrees) to a corresponding pulse width signal that the servo can interpret. Given the servo motors specifications according to its datasheet [see Appendix], Figure 2 displays the safe pulse width operating range of the servos, accompanied by its corresponding angles and direction.

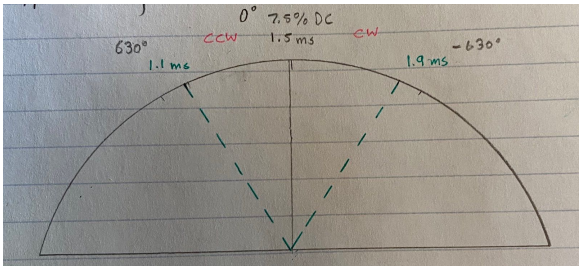

Figure 2: *The HiTec 785-HB Sail Winch Servo motor safe range of operation. Note that the minimum pulse width of 1.1 ms rotates the servo motor 1.75 turns counterclockwise, and the opposite behavior is yielded from the maximum pulse width of 1.9 ms. The servo motor can be set to its neutral position given a pulse width of 1.5 ms.*

Given the safe operating range, a mathematical formula can be derived to translate an angle input (in degrees) to the necessary pulse width signal sent to the servo motors. The expected frequency for the control signal of these servo motors is $50\ Hz$, as specified in its datasheet [see Appendix]. By relating period to frequency in Equation 6,

$$T = \frac{1}{f} \tag{6}$$

the given period for the servo motor can be determined to be $T = 20\ ms$. From there, the duty cycle, the percentage at which one period of a signal is activated, can be calculated using Equation 7.

$$DC = \frac{PW}{T} * 100\% \tag{7}$$

Knowing T, the duty cycle, DC, can be calculated for all pulse widths between the servo's safe operating range of $PW_{min} = 1.1\ ms$ and $PW_{max} = 1.9\ ms$. Table 1 demonstrates the process of translated an angle input (in degrees) to the appropriate duty cycle given the servo motor specifications.

Table 1: *Corresponding duty cycle output given angle input in degrees. All intermediate values can be linearly scaled to determine exactly the duty cycle for the given angle. Note that this servo operates at a control signal frequency of 50 Hz, and gear ratios were used upon integration to the system.*

Angle [$\circ$]	Pulse Width [ms]	Duty Cycle [%]
630	1.1	5.5
0	1.5	7.5
-630	1.9	9.5

Upon setting the angles of the two servo motors to the desired position, water was expelled from the nozzle to extinguish the object of interest. The water pump subsystem utilized a BACOENG 12V DC Submersible Pump that delivered $20\,ft.$ of pressure head and $12\ Gallons\ Per\ Minute\ (GPM)$. In order to experimentally calculate whether the pump would provide enough pressure to lift the fluid 53 in. and cover a 120 in. x 151in. grid, the Bernoulli Equation was utilized as shown in Equation 8

$$\frac{P_1}{\rho g} + \frac{V_1^2}{2g} + z_1 = \frac{P_2}{\rho g} + \frac{V_2^2}{2g} + \qquad (8)$$

given certain assumptions and controlled variables.

Experimental Technique

The experimental setup consisted of three main subsystems: image sensor, servo control, and fluid pump. These subsystems together captured, processed, and sent information to other subsystems in order to achieve one cohesive design meeting the functional requirements. Information flowed cyclically and sequentially from image sensor to servo control to fluid pump. This functionality follows the idea of detecting object(s) of interest, positioning pump according to said object of interest, and expelling fluid to extinguish it.

The primary goal of the image sensor subsystem was to capture image frames of radiant heat across the largest area possible while maintaining a modest cost. Long wave infrared sensors were the obvious choice given this design constraint, as they detect radiation across the infrared spectrum. Thermal radiation emitted by humans and its surrounding objects are accounted for within this infrared spectrum. The FLiR Lepton module served as a great choice

for its resolution, high field of view, low latency, and precision respective to its low cost. Figure 3 displays the FLiR lepton with its appropriate pinouts.

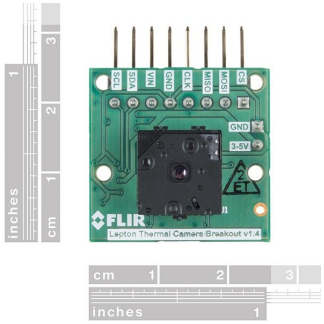

Figure 3: *The FLiR Lepton connected to its breakout board. The pinouts allow for SPI and I2C communication protocols to be configured as a peripheral device to computers such as a Raspberry Pi.*

A Raspberry Pi 3 B+ was the ideal candidate for a computing platform for this subsystem as well as the others, given its size, power output, and cost. The FliR's connection to the Pi's pinouts are displayed in Figure 4.

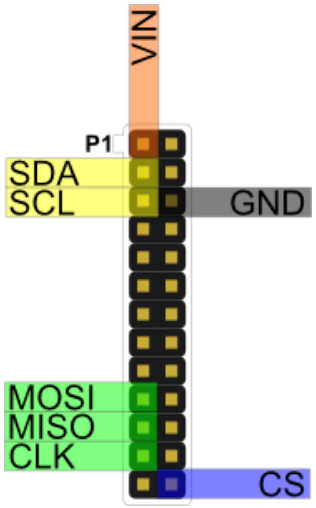

Figure 4: *Image sensor subsystem: FLiR breakout board connection to the Raspberry Pi GPIO pins. Connections made via F/F jumper cables. Note that that common ground is shared across all subsystems.*

The image sensor subsystem was strategically positioned at a height of 10 ft. in order to capture the largest area possible while emulating a similar height to that atop a roof, as visualized in Figure 5. Figure 5 shows its vertical field of view to cover $191\ in. - 46\ in. = 151\ inches$.

Provided that the camera's horizontal field of view (HFOV) covered 120 *inches* , the matrix for testing was established to be $120\ in. \times 151\ in.$.

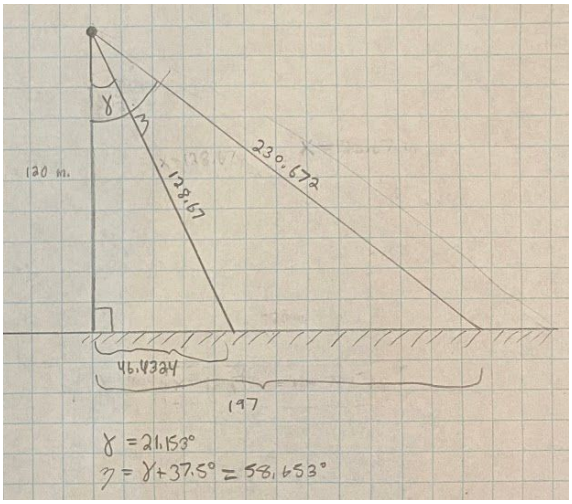

Figure 5: *Position of camera [black circle] at a height of 10 ft. above the surface. Camera suspended in this position using a wooden 2x4. Angle* γ *represents the angle of the camera with respect to the 2x4, and the angle between* γ *and* ζ *represent the vertical field of view (VFOV) of the camera.*

Given the known resolution of $80 \times 60\ pixels$ for the camera to cover a surface area of $120\ in. \times 151\ in.$, conversion from pixel coordinates to x and y distances (in inches) could be easily computed from a raw captured image. To account for distortion, however, a four point image transform was required prior to image processing to transform the "perspective" view into a "top-down" or "birds eye" view. Therefore, the transformed image was seemingly taken at an angle of $90\ degrees$ with respect to the horizontal at the center of the matrix. After determining the brightest object in the image, the pixel coordinates corresponding to its location could be easily converted into an x and y distance (in inches). These x and y distances [in inches] were inputted into Pythagorean's theorem to determine the hypotenuse and planar angle, θ , of the triangle it formed, as seen in Figure 6.

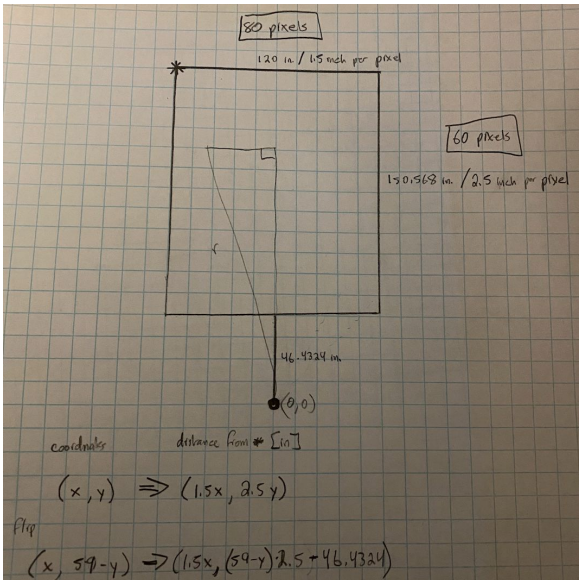

Figure 6: *Cartesian coordinates x and y, outputted from the image sensor, converted to polar coordinates to yield a distance, r, and planar angle, θ. Note that the coordinates had to flipped and offset due to the location/orientation of the camera module.*

This process yielded two figures: distance, r, and planar angle, θ. Planar angle θ represents the angle for which the nozzle needs to be positioned; pitch angle ϕ was then calculated given the distance, $r = x$, using the projectile motion equation described in Equation 5.

Upon completion of image capturing and processing, the servo control subsystem was ready for instructions. The range of motion for the nozzle attached to its housing was roughly $100 \circ$ in both θ and ϕ directions. Provided gear ratios of $GR_{theta} = \frac{5}{1}$ and $GR_{phi} = \frac{2}{0.75}$ for θ and ϕ respectively, position controlled servo motors capable of rotating $500 \circ$ were required. Sail winch servo motors, specifically the HiTec HS-785HB, were the clear choice as they are capable of multi turn rotation, but are not continuous servo motors which are velocity controlled. The schematic for the servo motors attached to the same Raspberry Pi as the image sensor subsystem can be seen in Figure 7.

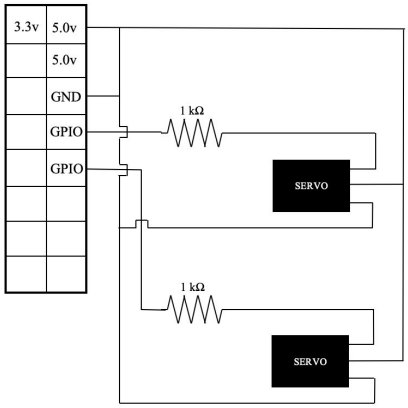

Figure 7: *Servo subsystem: Raspberry Pi GPIO pins connected to two servo motors. GPIO pin 8 controlled the "ϕ" servo motor, and GPIO pin 10 controlled the "θ" servo motor. Note that that common ground is shared across all subsystems.*

The image sensor subsystem yielded two angels, ϕ and θ, in degrees; servo motors connected to a Raspberry Pi are pulse width controlled, meaning these angles must be converted to pulse width signals according to Equation 7. An algorithm was programmed according to calculations above into the Raspberry Pi to perform these conversions from angle to duty cycle quickly and automatically. Upon conversion, those values would be sent to their respective servo motors.

Once the housing system properly positioned the nozzle to hit the object of interest, the fluid pump must be actuated for a discrete period of time to douce the area. Figure 8 represents the schematic allowing for an on/off switch for the pump to be developed via an algorithm and implemented into the Pi.

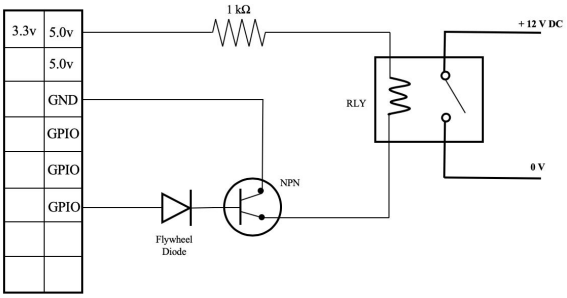

Figure 8: *Pump subsystem: NPN relay switch circuit to activate the water pump. When GPIO pin 14 is powered "OFF" current through the relay coil decreases and magnetic field collapses. Note that that common ground is shared across all subsystems.*

This subsystem closes a circuit to give power to the water pump via an algorithm that waits for all previous computation to be completed. When powered, the water pump expels water through the nozzle at a known velocity to hit the object of interest, according to theoretical calculations.

These three subsystems were first implemented and tested independently to verify calculations, to ensure all components were working, and to demonstrate the system met the design's expectations. Then, the three subsystems were integrated together to comprise the full fire prevention system. Figures 9, 10, and 11 provide visuals on the full setup, which was tested in a simplified, controlled environment.

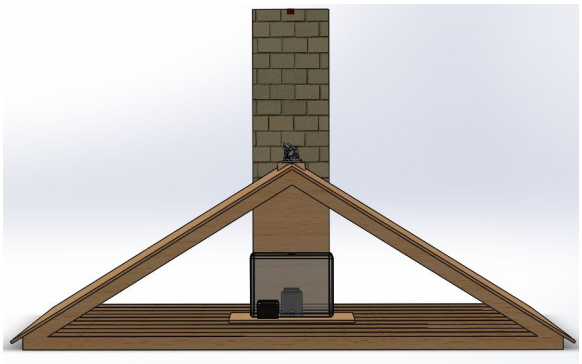

Figure 9: *Full scale mockup of setup. Note that the test setting was to full scale, but did not include the pitch in the roof. Testing was completed using a flat surface. The red box at the top of the chimney represents the image sensor, the middle housing represents the servo subsystem, and the bottom container represents the pump subsystem.*

Figure 10: *Full scale setup from the perspective of the camera. Note that this perspective provides a distorted view on the roof itself. Therefore, the true distance given by the camera must be measured by a distorted view of the raw image captured by the camera.*

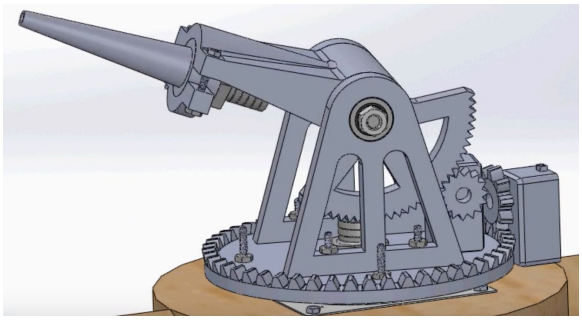

Figure 11: *Detailed view of the housing for the water pump nozzle, including the two servo motors in place to control pitch angle and planar angle, ϕ and θ, respectively.*

The accompanying software algorithm employed to control the system can be seen in Appendix A.

Results & Discussion

Preliminary results of the fully functioning system is displayed in Figure 12.

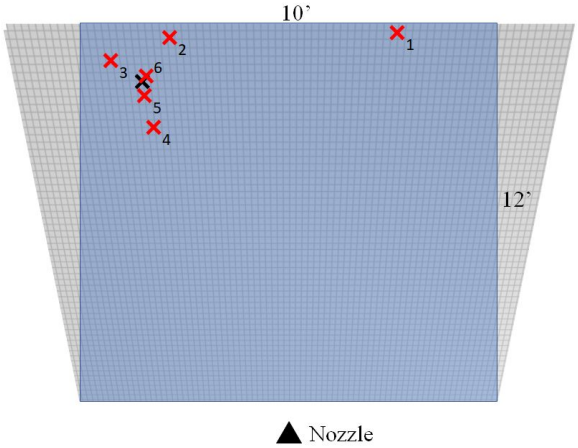

Figure 12: *Several test results overlaid on the matrix. Note the blue box represents the birds eye view of roof section, and the gray box represents the view from the perspective of the camera. The distortion algorithm employed to achieve the blue box view from the gray box view can be seen in Appendix.*

The results of this experiment were governed by the overall success of the system to extinguish fires in locations within the specified grid size. In order to achieve a successful result: an accurate extinguishing of the fire, three main subsystems had to perform correctly. These subsystems consist of the image sensor, servo control, and water pump subsystems. These subsystems are all intricately intertwined with each other through software within the Raspberry Pi, which can be seen in the Appendix. The results for the imaging subsystem were achieved first. Given the experimental setup of the camera described in the previous section, it was

confirmed that the resolution was adequate to pick up heat signatures of common barbeque coals at any location within the grid. Next, the brightest pixel algorithm was tested to verify that the code could accurately pinpoint where the hottest point in the coal was. This code worked, however upon testing it was noticed that for objects on fire with a substantial sized flame, the hottest point will not be at the source of the fire (coal) rather it would be the flame above the coal which led to accuracy discrepancies. Furthermore, the distortion algorithm was then tested to translate the view from the camera into a birds eye view which allowed for the creation of coordinate system of pixels. The origin of this coordinate system was established directly underneath the nozzle. The imaging subsystem worked and was the first finished system and was not changed. The primary imaging process is shown below in Figures 13-15.

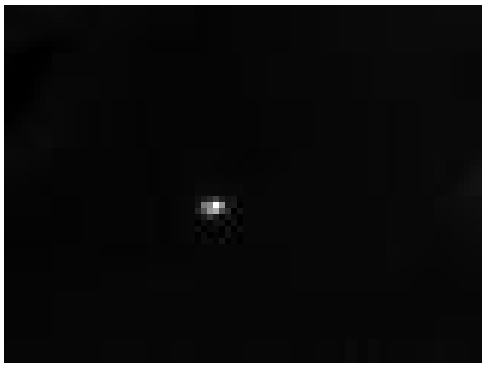

Figure 13: *Raw image capture taken from FLiR Camera. The image output dimensions are 60x8x1, where the third channel is a grayscale value. The white spot represents the hottest pixel in the camera's field of view.*

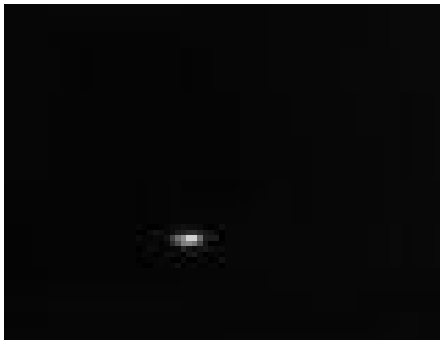

Figure 14: *Warped rendition of Figure 1. This distorted view represents the matrix as if the image was captured from a top-down view in the center of the matrix. The algorithm employed to accomplish this distortion can be found in Appendix.*

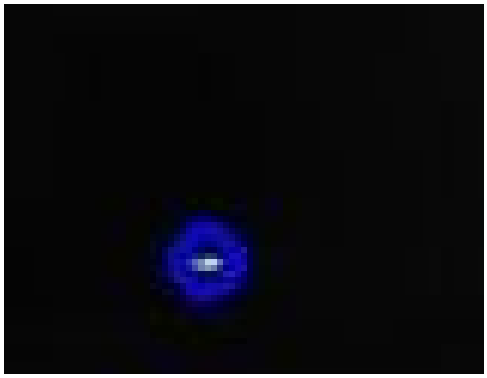

Figure 15: *Location of "Brightest Pixel" as noted by the blue circle of. To prevent "one hot" detection, the algorithm inspects 5 pixels radially to confirm objects location. The pixel outputs (x,y) pixel coordinate to be translated into Cartesian coordinates in further processing.*

The servo control subsystem was the most involved of the subsystems, as it went through many revisions throughout the testing process. There were two servo motors that targeted the two axis nozzle. These servo motors are position controlled and have the capability to rotate 1.75 revolutions away from their home state in either direction. The servo motors run off of associated duty cycles, with 7.5 ms being neutral, 1.9 ms being 1.75 turns CW, and 1.1 ms being 1.75 turns CCW. In order to accurately actuate the motors, the coordinates given by the pixel grid created in the imaging subsystem had to be converted to corresponding duty cycles as per Equation 7. This was relatively simple for the "theta" motor that actuated the nozzle on the horizontal axis. The horizontal axis was 79 pixels wide, which corresponds to a pixel size of 1.5 inches given the 10 ft shown in the experimental setup. Given that the origin was directly below the nozzle, the y-axis divided the grid directly in half, so the horizontal components were in the range between [-39, 39]. After receiving the warped output from the imaging subsection the code "counted" how many pixels the brightest spot was from both the horizontal and the origin. From here, the Pythagorean theorem was used to calculate the angle value away from the horizontal the pixel was. This angle was then converted into a duty cycle, which was read by the servo to actuate it. This was successfully achieved, however accuracy was not ideal unless the system was calibrated perfectly. This was shown in "Demo Day", as in the beginning our system was very accurate and towards the end, after the camera stand had been slightly moved due to human intervention, θ accuracy decreased. The pitch gear, angle ϕ, was more difficult to actuate, as it was initially controlled by projectile motion calculations yielding Equation 5. Once θ was determined, the hypotenuse, r, of the triangle was determined which was the overall distance the coal was from the nozzle. Given that distance, a calculated initial velocity at the end of the nozzle, and the height that the water was shooting from, an angle ϕ was calculated according to Equation 5. Upon testing, it was found that the ϕ angle was very inaccurate, and would only work when the coals were placed at a vertical distance correlating to a 0 ° angle of ϕ (when the

water shoots directly straight). Therefore, a new way to determine the ideal ϕ angle was conceived. The new idea was to linearly interpolate the duty cycle values that correspond to the maximum and minimum vertical distances of the matrix in order to derive an equation for a line. The vertical distance from the origin was received in pixel location from the camera and then converted into inches and then put in the linear equation to output a corresponding duty cycle which was then read by the servo motor. This drastically increased accuracy for all distances in the matrix. Finally, in order to make the targeting subsystem more robust, a "duty cycle wiggle" was implemented to paint a box around the area surrounding the hottest pixel to guarantee that the whole coal would be extinguished. This system successfully worked, as shown on "Demo Day".

The water pump subsystem was the final subsystem created, however it was the simplest. The hardware was set up on a breadboard as described in Figure 8. The relay had a designated 4 second interval so that water could be supplied to the hot target for 4 seconds. This software was relatively simple and can be seen in the Appendix. After basic testing, it was found to work smoothly.

After each subsystem was finalized, overall tests were conducted to determine the accuracy of the water stream. If the stream was accurate, then the coal was extinguished and the results were successful. Figure 15 outlines a test performed with four coals spaced randomly within the matrix.

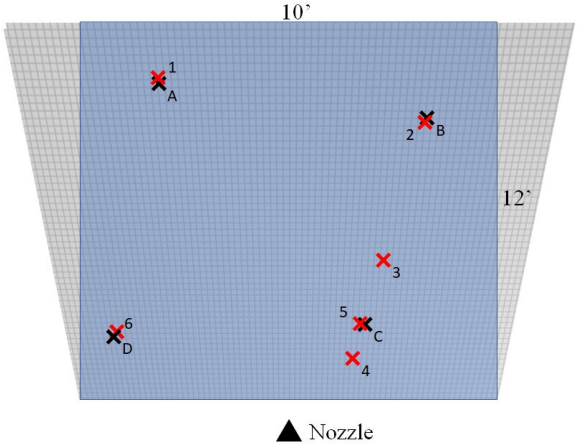

Figure 16: *"Demo Day" test results overlaid on the matrix. Note the blue box represents the birds eye view of roof section, and the gray box represents the perspective view of the camera. The distortion algorithm employed to achieve the blue box view from the gray box view can be seen in Appendix.*

The black Xs denote where hot coals are in the matrix while the red Xs denote the area where the water stream hit. Coals A, B, and D were extinguished on the first shot. The attempts for coal C

were not accurate, this is because coal C was a very large flame and the camera detected the ambient heat rising off of the large fire. At first, it seemed as though the calibration was incorrect; so, we changed the ϕ angle, and upon running the test again the camera correctly identified the ember we had originally undershot. We then removed this change in calibration and re ran the test, where the camera correctly identified the ember and hit the target.

The test apparatus used to mount the camera was a 10 Ft long 2x4. This set up lacked stability and there was slight movement between tests at the top of this pole. this slight movement would slightly affect our calibration each time we tested causing us to be off the target at times and in random directions which could not be accounted for. due to this unpredictable inaccuracy we decided to introduce a slight wiggle in our nozzle to cover a wider area around each ember. this wiggle would cancel out the unpredictable movement of the camera on the pole and ensure that we would hit our target. if the camera were mounted on a more rigid surface the wiggle factor could be removed and we could directly hit each target with much less margin for error.

Conclusions

Experiment has shown that the design met the functional requirements and expectations matched theoretical predictions. This major success of this design project however did come with minor discrepancies, and justifications for such errors pose investigations for future design iterations. These discrepancies include calibration procedures, environmental conditions (wind, air resistance), and image processing.

The current design has the area of interest “hard coded” to a discrete 120 in. by 151 in. matrix, and the camera’s position with respect to this area must be calibrated during each testing iteration. Current calibration procedures rely on putting hot spots in the four corners of the matrix, representing the total field of view of the camera in both the horizontal and vertical directions. The camera’s position is then adjusted to fit these four corners within one image, signifying that the setup was aligned with the project’s design. Future improvements to the design would be making this matrix dynamic, where the “hard coded” trigonometric calculations would be implemented in an algorithm, and the matrix would be determined by the placement of the four “hot corners” irrespective of the camera’s position.

Environmental conditions and the fluid’s interaction with the environment upon its expulsion from the nozzle also pose future investigations in creating a system with more precision in more versatile environments. Establishing variables representing these environmental effects, as well implementing a water pump capable of changing the exit velocity of the fluid, would allow for environmental effects to be accounted for. As a result, this system could be implemented in various conditions and yield results of high precision.

Current image processing identifies the “hottest” pixel seen by the camera irrespective of a temperature threshold. Therefore, in the event of no burning objects in the camera’s view, the camera will still detect the “hottest” object in its view. This yields erroneous behavior of the subsequent subsystems, as the camera has no ability to process the captured image and set a threshold temperature for which any object below it is deemed “not hot”. Implementing a temperature threshold for the camera to determine “hot” versus “not hot” would make the system much more realistic and reliable for a truly autonomous product.

Appendix A

```
# import the necessary packages
import cv2
import numpy as np
import matplotlib.pyplot as plt
from pylepton import Lepton
from imutils import paths
import imutils
import math as m
import RPi.GPIO as GPIO
import time
import time
from scipy.spatial import distance as dist
import RPi.GPIO as GPIO     # Import GPIO Library
import time                 # Import time library for a delay

#
# Capture image data
# capture() is a 12-bit non-normalized raw sensor data
# we constrast extend it since bandwidth tends to be narrow
#
def capture_image(file_name):
        print("Capturing image named %s.jpg..." % file_name)
        with Lepton() as l:
                a,_ = l.capture()
                cv2.normalize(a, a, 0, 65535, cv2.NORM_MINMAX) #extend constrast
                np.right_shift(a, 8, a) #fit data into 8 bits
                print("Image capture complete!")
                cv2.imwrite(file_name + ".jpg", np.uint8(a)) #write it
                return file_name + ".jpg"

#
# Find brightest spot in image
#
def bright(img, radius):
        # load the image and convert it to grayscale
        image = cv2.imread(img) # load the image
        orig = image.copy() # clone original image
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #grayscale image

        # apply a Gaussian blur to the image then find the brightest region
        # first apply a Gaussian blue to the image to remove high frequency noise
        gray = cv2.GaussianBlur(gray, (radius, radius), 0)
        # apply Gaussian blur with the supplied radius
```

```
        # call cv2.minMaxLoc to find brightest pixel in image
        (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(gray)
        image = orig.copy()
        cv2.circle(image, maxLoc, radius, (255, 0, 0), 2)

        # display the results of our newly improved method
        #cv2.imshow("Brightest pixel value", image)
        #cv2.imwrite("brightest_pixel.jpg", image)
        #cv2.waitKey(0)

        print("Pixel location for maxVal is: %s" % (maxLoc,))
        return maxLoc

#
# Order points
#
def order_points(pts):
        # sort the points based on their x-coordinates
        xSorted = pts[np.argsort(pts[:, 0]), :]

        # grab the left-most and right-most points from the sorted
        # x-coodinate points
        leftMost = xSorted[:2, :]
        rightMost = xSorted[2:, :]

        # now, sort the left-most coordinates according to their
        # y-coordinates so we can grab the top-left and bottom-left
        # points, respectively
        leftMost = leftMost[np.argsort(leftMost[:, 1]), :]
        (tl, bl) = leftMost

        # now that we have the top-left coordinate, use it as an
        # anchor to calculate the Euclidean distance between the
        # top-left and right-most points; by the Pythagorean
        # theorem, the point with the largest distance will be
        # our bottom-right point
        D = dist.cdist(tl[np.newaxis], rightMost, "euclidean")[0]
        (br, tr) = rightMost[np.argsort(D)[::-1], :]

        # return the coordinates in top-left, top-right,
        # bottom-right, and bottom-left order
        return np.array([tl, tr, br, bl], dtype="float32")

#
# @arg image: to apply perspective transform
```

```
# @arg pts: to be ordered
# @return warped image
#
def four_point_transform(image,pts):
	# obtain a consistent order of the points
	# unpack them individually
	print("Applying image distortion...")
	rect = order_points(pts)
	(tl, tr, br, bl) = rect

	# compute width of the new image
	# max distance between br/bl or tr/tl x-coordinates
	widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
	widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
	maxWidth = max(int(widthA), int(widthB))

	# compute height of new image
	# max distance between tr/br or tl/bl y-coordinates
	heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
	heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
	maxHeight = max(int(heightA), int(heightB))

	# now we have dimensions of new image

	# construct set of destination points to obtain top-down view
	# --specify points in tl,tr,br,bl order
	dst = np.array(
		[
		[0,0],
		[maxWidth - 1, 0],
		[maxWidth - 1, maxHeight - 1],
		[0, maxHeight - 1]
		],
		dtype = "float32")

	# compute the perspective transform matrix and then apply it
	M = cv2.getPerspectiveTransform(rect,dst)
	warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

	print("Distortion complete!")

	# return the warped image
	return warped

#
```

```
# Take coordinate output and determine distance & theta from hose
#
def maxLoc_to_distance(maxLoc):
	x_coord, y_coord = maxLoc
	x = 1.5*(39-x_coord) #instead of (79-x_coord)
	print("X is:")
	print(x)
	# (1) x was calibrated and is accurate
	y = (59-y_coord)*2.516666 + 24# 17 is the distance from nozzle to  origin   #46.4324 is
distance from camera to origin
	print("Y is: ")
	print(y)
	# (2) y was calibrated and is accurate
	c = np.sqrt(np.power(x,2) + np.power(y,2))
	print("C is: ")
	print(c)
	#(3) c calibrated and accurate
	theta = np.arccos(y/c)
	if (x < 0):
		theta = -theta
		print("Theta is: ")
		print(theta)
	else:
		print("Theta is: ")
		print(theta)
	#(4) theta calibrated and accurate
	return c, theta

#
# Projectile motion equations for phi (corresponds to the phi servo motor)
# @param initial_velocity [ft/s] initial exit velocity from nozzle
#
def projectile(distance):
	slope = -0.00641687
	if(distance < 51):
		dutyCyclePhi = 7.5
		print("Distance too short. Out of range...")
	else:
		dutyCyclePhi =-0.01061*distance+8.8415
	return dutyCyclePhi # in inches


def SetPhiAngle(angle):
	phiAngle = angle
	#print(phiAngle)
```

```
	GR_PHI = 2/0.75
	#print(GR_PHI)
	phiDegree = phiAngle * GR_PHI + 670 # too far, tune it up
	#print(phiDegree)
	dutyCyclePhi = 5.5 + phiDegree * 0.003174603175
	print("Calculated DC for phi: ")
	print(dutyCyclePhi)
	#phi.ChangeDutyCycle(8)
	#time.sleep(5)
	lowerBound = 6.25
	upperBound = 8.30
	if ((dutyCyclePhi < lowerBound) or (dutyCyclePhi > upperBound)):
		print("Out of range...")
	else:
		print("Phi is in correct range!")
		phi.ChangeDutyCycle(dutyCyclePhi)
		time.sleep(10)
	phi.ChangeDutyCycle(0)
	time.sleep(1)
	phi.stop()


def SetThetaAngle(angle):
	thetaAngle = angle
	#print(thetaAngle)
	GR_THETA = 5/1
	#print(GR_THETA)
	thetaDegree = thetaAngle * GR_THETA + 630
	#print(thetaDegree)
	dutyCycleTheta = 5.5 + thetaDegree * 0.003174603175
	print("Calculated DC for theta: ")
	print(dutyCycleTheta)
	#theta.ChangeDutyCycle(8)
	#time.sleep(5)
	for x in range(0, 5):
		theta.ChangeDutyCycle(dutyCycleTheta)
		time.sleep(0.5)
		theta.ChangeDutyCycle(dutyCycleTheta+0.1)
		time.sleep(0.5)
		theta.ChangeDutyCycle(dutyCycleTheta-0.1)
		time.sleep(0.5)
	theta.ChangeDutyCycle(0)
	time.sleep(1)
	theta.stop()
```

```
def SetAngles(phi_ang, theta_ang):
        # Servo setup
        phiServoPin = 10
        thetaServoPin = 8
        GPIO.setmode(GPIO.BOARD)

        GPIO.setup(phiServoPin, GPIO.OUT)
        GPIO.setup(thetaServoPin, GPIO.OUT)

        phi = GPIO.PWM(phiServoPin, 50)
        theta = GPIO.PWM(thetaServoPin, 50)
        phi.start(0)
        theta.start(0)

        # Pump setup
        switchPin = 7
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(switchPin, GPIO.OUT)

        dutyCyclePhi = phi_ang - 0.1
        print("Calculated DC for phi: ")
        print(dutyCyclePhi)
        #phi.ChangeDutyCycle(8)
        #time.sleep(5)

        thetaAngle = theta_ang
        #print(thetaAngle)
        GR_THETA = 4.8
        #print(GR_THETA)
        thetaDegree = thetaAngle * GR_THETA + 630
        #print(thetaDegree)
        #dutyCycleTheta = 7.5
        dutyCycleTheta = 5.5 + thetaDegree * 0.003174603175
        print("Calculated DC for theta: ")
        print(dutyCycleTheta)
        #theta.ChangeDutyCycle(8)
        #time.sleep(5)

        lowerBound = 6.25
        upperBound = 8.30
        CHANGE_THETA = 0.1
        CHANGE_PHI = 0.1
        if ((dutyCyclePhi+CHANGE_PHI > upperBound) or (dutyCyclePhi < lowerBound)):
                print("Cannot flutter")
                print("Duty cycle was...")
```

```
        print(dutyCyclePhi)
        for x in range(0, 2): #spray at desired location
                SLEEP  = 0.5
                theta.ChangeDutyCycle(dutyCycleTheta)
                time.sleep(SLEEP)
                phi.ChangeDutyCycle(dutyCyclePhi)
                GPIO.output(switchPin, GPIO.HIGH) #activate pump
                time.sleep(SLEEP)
                theta.ChangeDutyCycle(dutyCycleTheta+CHANGE_THETA)
                time.sleep(SLEEP)
                theta.ChangeDutyCycle(dutyCycleTheta-CHANGE_THETA)
                time.sleep(SLEEP)
                phi.ChangeDutyCycle(dutyCycleTheta-CHANGE_PHI)
                time.sleep(SLEEP)

else:
        print("Phi is in correct range!")
        for x in range(0, 2): #spray at desired location
                #CHANGE_THETA = 0.1
                #CHANGE_PHI = 0.1
                SLEEP = 0.5
                theta.ChangeDutyCycle(dutyCycleTheta+CHANGE_THETA)
                time.sleep(SLEEP)
                phi.ChangeDutyCycle(dutyCyclePhi+CHANGE_PHI)
                GPIO.output(switchPin, GPIO.HIGH) #activate pump
                time.sleep(SLEEP)
                theta.ChangeDutyCycle(dutyCycleTheta)
                time.sleep(SLEEP)
                phi.ChangeDutyCycle(dutyCyclePhi)
                time.sleep(SLEEP)
                theta.ChangeDutyCycle(dutyCycleTheta-CHANGE_THETA)
                time.sleep(SLEEP)
                phi.ChangeDutyCycle(dutyCyclePhi-CHANGE_PHI)
                time.sleep(SLEEP)
                theta.ChangeDutyCycle(dutyCycleTheta)
                time.sleep(SLEEP)
                phi.ChangeDutyCycle(dutyCyclePhi)
                time.sleep(SLEEP)

GPIO.output(switchPin, GPIO.LOW) #stop pump

phi.ChangeDutyCycle(0)
time.sleep(1)
phi.stop()
```

```
        theta.ChangeDutyCycle(0)
        time.sleep(1)
        theta.stop()


def main():
        # capture image
        orig_image = capture_image("12_2_test")

        # load the image
        image = cv2.imread("12_2_test.jpg")

        # grab the source coordinates (i.e. the list of (e,y) coords)
        PTS = "[(18,0), (61,0), (79,59), (0,59)]"

        # apply four point transform to obtain top-down view of the image
        pts = np.array(eval(PTS), dtype = "float32")
        warped_image = four_point_transform(image, pts)
        cv2.imwrite("warped_image.jpg", warped_image)

        # show the original and warped images
        #cv2.imshow("Original", image)
        #cv2.imshow("Warped", warped_image)
        #cv2.waitKey(0)


        coordinates = bright("warped_image.jpg",5)
        print("Pixel location of interest is:")
        print(coordinates)

        distance, theta_deg = maxLoc_to_distance(coordinates)
        theta_deg = -(m.degrees(theta_deg)) #[CHECK THIS] this flips the sign for duty cycle
        #print("Theta in degrees is: ")
        #print(theta_deg)

        print("Calculating projectile...")
        phi_deg = projectile(distance)

        print("Desired theta angle is: %s" % theta_deg)
        print("Desired phi angle is: %s" % phi_deg)
        """
        # Servo setup
        phiServoPin = 10
        thetaServoPin = 8
        GPIO.setmode(GPIO.BOARD)
```

```
        GPIO.setup(phiServoPin, GPIO.OUT)
        GPIO.setup(thetaServoPin, GPIO.OUT)

        phi = GPIO.PWM(phiServoPin, 50)
        theta = GPIO.PWM(thetaServoPin, 50)
        phi.start(0)
        theta.start(0)

        # Pump setup
        switchPin = 7
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(switchPin, GPIO.OUT)
        """
        #SetPhiAngle(-phi_deg)
        #SetThetaAngle(-theta_deg)
        SetAngles(phi_deg, -theta_deg)
        GPIO.cleanup()

try:
   while True:
      main()
except KeyboardInterrupt:
        GPIO.cleanup()
        pass
```