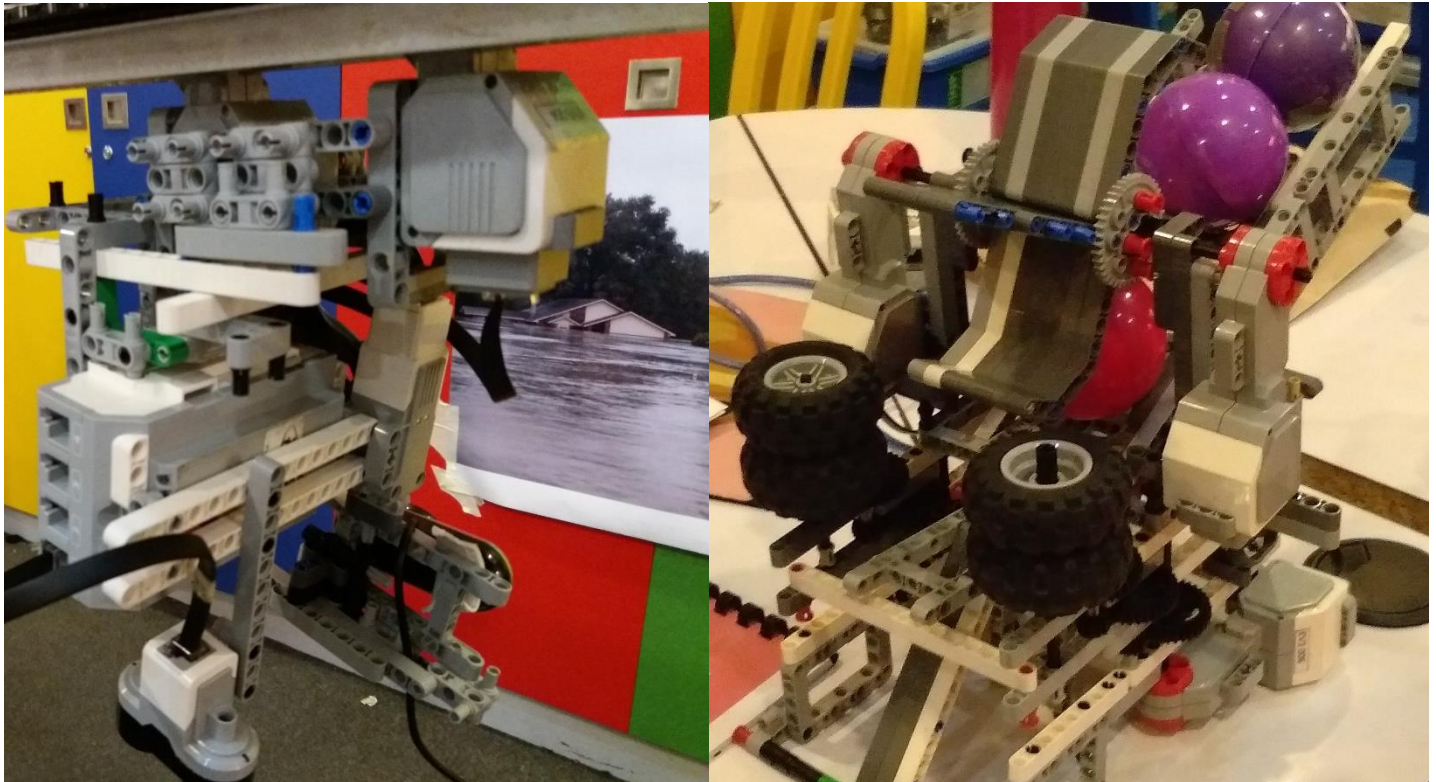




4F, Fisher Mall, Quezon City
Metro Manila



Project S.O.A.R - Smart Operations And Rescue

Drone

Prepared by

Almond Rose Obedoza

Haydn Alexander Tan

Grant Travis Wong

Sire Garcia

Owen Justin Goheco

Matias Nacario

CHAPTER ONE

INTRODUCTION

It was several years ago when drones started helping out in emergency response situations, conducting surveillance and locating structural damage. Today, almost every agency involved in disaster responses may somehow deploy a drone as a primary tool, to assess damage, search for victims, make 3D maps to track changes to certain areas over time, and more.

After search and rescue operations, which can take up to several weeks, rescuers have to focus on treating and taking care of the survivors they have already found.

If the search and rescue period were shorter, the survivors could be treated sooner. If it were more effective, more victims could be found and treated before it is too late.

To sum it up, several lives could be saved if we could somehow make the search and rescue phase quicker and more efficient. But how? Perhaps it would help if the rescuers had an eye in the sky...

STATEMENT OF THE PROBLEM

In 2018, one of the biggest disasters was a 7.5 earthquake that struck Palu, Indonesia, causing a tsunami that killed 2,256 people. In 2017, disasters cost \$306,000,000,000 in losses.

According to data from the United Nations Office for Disaster Risk Reduction, natural disasters from 2005 to 2014 cost 1.4 trillion in damage, impacted 1.7 billion people, and killed 0.7 million.

And those 0.7 million people also include rescuers, who go to remote and inaccessible areas to save lives, often putting themselves directly in harm's way in the process. The rescue crews that assist in search and rescue operations and damage assessments often find themselves around hazards like unstable buildings, broken electrical lines, dangerous water and flooding conditions, and more.

The search and rescue period can last for hours or even days after the disaster. Only after this can rescuers focus on treating the survivors they have already found, which can be a problem in situations like Hurricane Katrina (2005), wherein the process lasted for several weeks.

As disasters all over the world claim the lives of disaster victims and rescuers alike, the question is – what can be done to make the search and rescue period shorter and more effective, ultimately saving the lives of millions?

PURPOSE

The Smart Operations and Rescue (SOAR) drones will be equipped with Edge Artificial Intelligence (Edge AI) and sensors, thereby giving them the ability to scan an area from above and detect ongoing calamities, in order to help inform rescuers about these. When a fire is detected, SOAR drones with fire retardants can drop these into the flames to extinguish the fire. When a flood

is detected, other SOAR drones with facial recognition can locate people trapped in the flood and drop inflatables and life jackets to help them float to safety.

After an earthquake, rescuers and inspectors normally have to go inside buildings to search for structural damage and check if it is safe for people to go back inside. However, this is dangerous because the building could collapse while they are still inside. It also takes a long time to thoroughly search an entire building. SOAR drones can be used to search for damage like cracks, broken windows and doors, gas leaks, and water leaks, with sensors and artificial intelligence. Then, they can send the information to engineers, who can study the data and repair the damage.

SIGNIFICANCE OF THE STUDY

The SOAR Drone saves lives through 7 key actions:

- (1) By the time rescuers get informed of calamities happening in distant locations, it would likely be too late to save most of the victims. SOAR uses artificial intelligence to detect ongoing disasters within the area. Rescuers can view pictures of these disasters to help inform them about potential crises, near or far, and decide which calamity to attend to first. Our robot is also a drone, which enables it to fly over debris and quickly locate disasters.
- (2) SOAR helps rescuers save resources. Typically, rescuers would deploy rescue trucks and helicopters to search for victims and locate places that need attention such as burning buildings and flooded villages after a natural disaster. However, these vehicles are expensive. Our robot is a drone, which would typically cost only 75 cents per hour. SOAR can also fit through small spaces and fly over debris, thereby enabling it to quickly reach disasters and victims. It can legally soar up to one

hundred meters high to scan more terrain at once and detect disasters that rescuers could not see at eye level. Thus, our robot can make search and rescue operations more efficient and inexpensive.

(3) Rescuers normally treat the survivors they have already found *after* they conduct search and rescue operations. Since our robot makes the search and rescue phase quicker, rescuers and doctors can take care of the victims sooner, thereby increasing the chance that they will live.

(4) The SOAR drones can help out in finding and rescuing people during floods and maritime disasters, by locating them and throwing life jackets and inflatables to prevent them from drowning.

(5) Usually, after an earthquake, rescuers and inspectors search buildings for structural damage to make sure that it is safe for people to go back inside. However, this is risky because it is possible for the building to collapse while the inspectors are still inside. The SOAR drone can use AI and sensors to find damage in a building and send information about it to engineers. This can help save the lives of the inspectors and rescuers, and the people who live or work inside the building.

(6) When inspectors search for structural damage in a building, this process can take several days. SOAR can speed up this process because drones can fly and go through areas that are normally inaccessible to humans.

(7) The SOAR drone can also drop fire retardants into a burning structure, which can help save the lives of the people inside and around it. This can also help the rescuers and firefighters by preventing them from having to go too close to the structure and get exposed to high temperatures.

CHAPTER 2

RELATED LITERATURE

Drones for Assessing Structural Damage

Rescuers sometimes encounter gas leaks when conducting rescue operations in areas that are critically damaged. However, there are drones nowadays that are equipped with “sniffers,” or sensors that can detect high levels of methane, to help rescuers locate broken gas lines. After that, workers would shut down these lines and fix any breaches, thereby preventing explosions from occurring.

There are also existing drones with high-zoom sensors to assess damage and locate places where it may be dangerous for people to venture.

SOAR uses Artificial Intelligence to identify disasters and structural damage. However, in the future, our robot can be equipped with both AI and sensors to help it recognize calamities with more accuracy.

3D Mapping

Some drones are used to create 3D maps. These drones have high quality cameras and are used to take several pictures of a certain building. Researchers then use these pictures to make 3D maps of the building. They use these maps to read geometric information about it, and track

changes to certain areas over time. Some maps are also created using LiDAR, an instrument that uses laser beams to provide a precise 3D model of an object.

Drones with Thermal Sensors

Some drones use thermal sensors to detect human body heat. This helps rescuers locate and identify survivors in the aftermath of a disaster. It also helps them find people trapped under piles of debris. We can combine this technology with artificial intelligence so that SOAR can find people more accurately and efficiently.

Google AIY Vision Kit

The Google AIY Vision Kit is a kit that lets people build a smart camera with a Vision Bonnet and Raspberry Pi. This smart camera uses Edge AI to classify images.

Inference, or running an image or text through an AI model to output a result, is usually done on the Cloud. However, with Edge AI, inference is run on a device like the Vision Bonnet included in the AIY Vision Kit.

The Google AIY Vision Kit has pre-trained AI models that can classify faces as happy or sad, detect different types of food, and identify different objects in an image. We once tried experimenting to see if it could detect disasters with its image classification model, and it identified fires as fireboats, torches, stoves, candles, and fire screens. It also identified floods as bathtubs, boathouses, lakes, and amphibious vehicles.

CHAPTER 3

METHODOLOGY

PROCEDURE AND TIME FRAME

In 2018, we interviewed Dino Juan, a former military man who works on rescue missions, to get his thoughts on a prototype robot that we were proposing to help during floods. He mentioned that it would be helpful for his team if drones could be used to assess an area before his men were deployed.

In 2019, we learned about Edge AI through the Google AIY Vision Kit, which demonstrates how inference could be done locally, without the need to connect to a cloud service via the Internet. We realized the potential of applying this new technology to a drone to help in disaster rescue and risk management.

Earlier this year, we applied what we learned from the AIY Vision Kit to make an actual AI model on Google Cloud, which could detect fires and floods. Soon after that, we learned that it was possible to run an actual Edge AI model on our laptop, so we tried it out using Google Cloud and Python's TensorFlow library. After we got it to work, we made another Edge AI model that could detect different types of structural damage, like cracks, broken windows, and broken doors.

We continuously sought to improve our proposed solution based on inputs from experts on drones, artificial intelligence, and inspecting structures after an earthquake. Here are some of the important things that we have learned from the interviews:

INTERVIEW INFORMATION

Helicopters Versus Drones

Normally, rescuers would send out big rescue trucks and helicopters to find victims *and* rescue them. However, these are really expensive. A helicopter could cost around two thousand dollars per hour. It is better to use drones to search for victims and ongoing disasters, because they only cost around seventy-five cents per hour and they can fit through tight spaces, thereby making the search and rescue phase quick, efficient, and inexpensive. After SOAR locates some disasters and the rescuers have decided which one to attend to first, they can use rescue trucks or helicopters to do the actual rescue operations.

Sending Back Coordinates

Once a calamity is detected, the drone can send the disaster coordinates back to the rescue office. SOAR can also have lights that can flash a distress signal as well as its location in Morse code, in case connectivity is down after a disaster. Since our robot is a drone, the rescue team back at the police or fire station should be able to see it in the air.

Drones

Drones are ideal for disaster recovery because they can pass through areas that are difficult to access. They are small, which helps them get through tight spaces, and they can legally fly up to a hundred meters high, which lets them soar over debris and locate disasters happening far away. This helps them find disasters and victims in a shorter amount of time.

Edge AI

Our robot will use Edge AI to identify disaster zones. It can be given examples of calamities from different angles at different times of day, so SOAR can use artificial intelligence to accurately detect situations that need attention.

Inference is running an image or text through an AI model to get a result. This is normally done on the Cloud. However, with Edge AI, inference is run on a small device. This helps increase the privacy and speed of the AI model, and lessen its price.

Using Edge AI also means that rescuers do not have to pay to use online artificial intelligence services. In addition, they do not have to be dependent on the Internet, which they may not be accessible in the aftermath of a disaster.

Finding Structural Damage in a Building

After an earthquake, SOAR can be sent inside a building and use AI and sensors to find structural damage.

Usually, after an earthquake, rescuers and inspectors search buildings for damage to check if it is safe for people to go back inside. However, this puts them at risk because the structure could collapse while they are still inspecting it. It also takes a long time to thoroughly search an entire building.

Drones like SOAR can help make inspections like these faster, safer, more accurate, and more efficient. Our robot will use AI and sensors (like gas and high-zoom sensors) to pinpoint damaged parts of the structure that need closer inspection.

We also learned that it is important to measure the depth of cracks. Deeper cracks can cause the structure to collapse, regardless of how long or short they are. Our robot will have an ultrasonic sensor to check how deep the cracks in a building are, and inform rescuers if it finds a dangerous crack.

Existing Structural Damage Inspection Drones

We interviewed Nicholas Osborn, the founding director of AES digital hub. He and his team use drones and Google Vision AI technology to check for structural damage – like coating damage, lightning damage, edge wear, and cracks – on their windmills. This proves that it is possible to use artificial intelligence to search for different types of damage in buildings and structures. In the future, we can combine this technology with sensors (like gas, infrared, ultrasonic, and high-zoom sensors) so that our drone can do damage inspections more accurately.

We also learned that before they implemented artificial intelligence to help them with inspections, it took a while for the AES team to examine all the pictures of the windmills, determine whether or not there was structural damage, and figure out what type of damage was present in the picture. This is why artificial intelligence can make inspections easier, faster, and more efficient.

PROTOTYPE

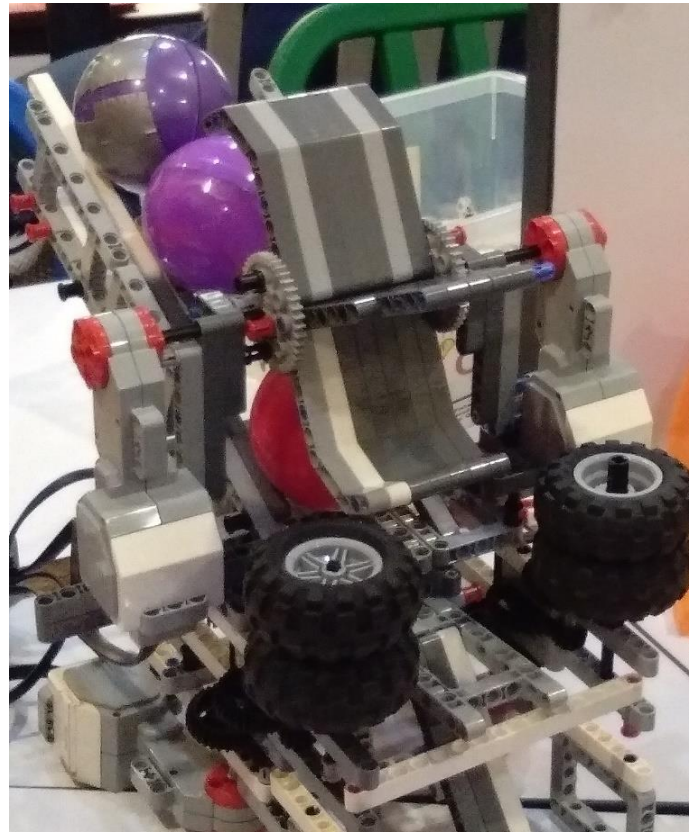
We built a working prototype to simulate SOAR: a robot with a webcam, to take pictures of images of disasters and structural damage for our AI model to analyze. This robot is attached to a

monorail mechanism to simulate a flying drone. We also built a launcher to throw paper life jackets, to simulate inflatables that inflate once they hit the water, which people can use to ride out of floods. We trained an AI model to detect fires and floods. After that, we made another model to detect structural damage, specifically cracks, broken doors, and broken windows. We also attached an ultrasonic sensor to an EV3 brick to simulate measuring the depth of cracks.

This is the robot with the webcam:



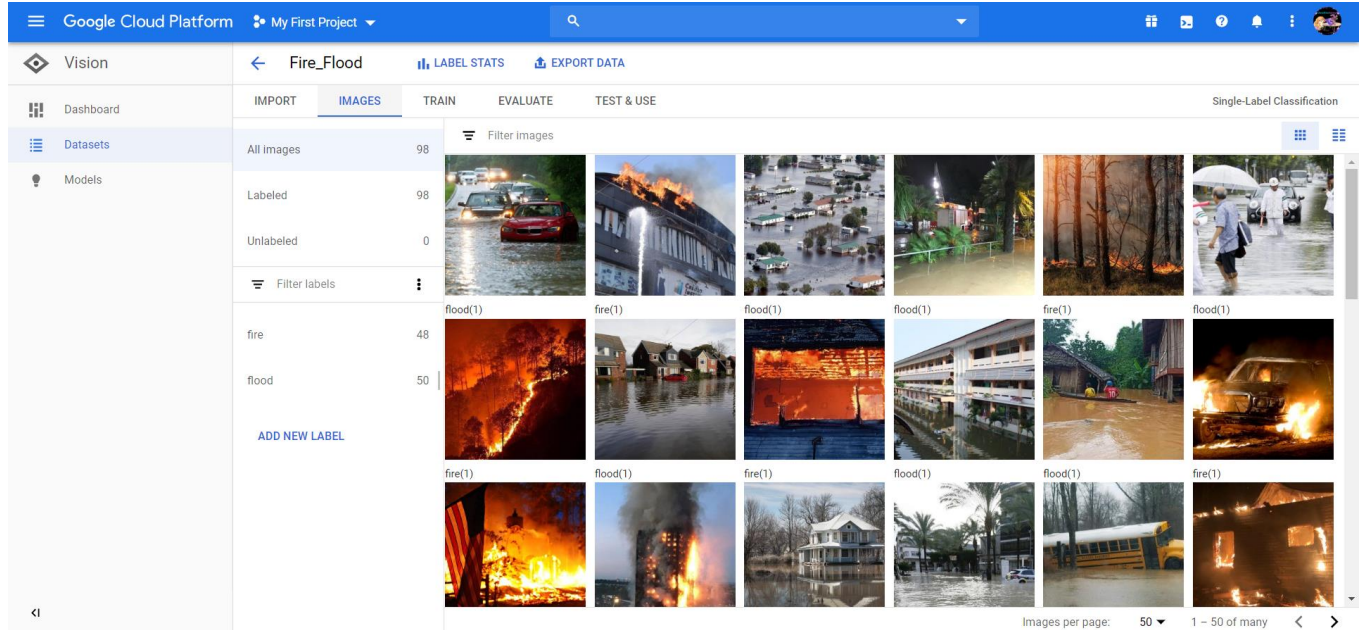
This is the launcher:



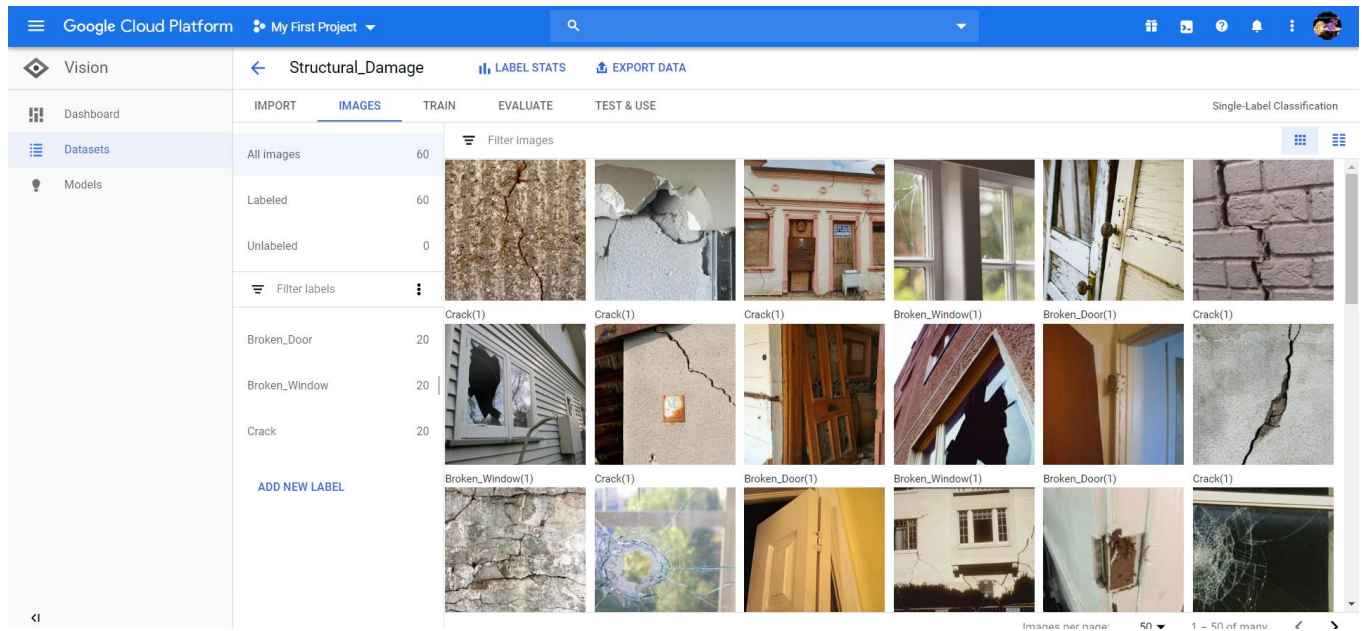
ANALYSIS PLAN

These are the pictures of our robot's code on EV3 MicroPython and Google Cloud:

We used Google Cloud to make two AI models – one to detect fires and floods, and another to search for structural damage. Below is a screenshot of the model that can detect different disasters:



And below is a picture of the model that can detect damage, specifically cracks and broken doors and windows:



We have three different robots.

Our first robot is the drone. It has a webcam that can take pictures of disasters and structural damage. These pictures will then be analyzed by an artificial intelligence model.

An EV3 brick lets us control an upside-down monorail mechanism. The drone is suspended on the monorail, which lets it move back and forth, to make the drone seem like it is flying in the air.

And finally, our last robot, the launcher, can throw paper life jackets to simulate actual life vests. In reality, it will be a drone that can launch inflatables for flood victims to ride and fire retardants to extinguish fires.

Aside from these, we also have a robot with an ultrasonic sensor to demonstrate measuring the depths of cracks. The LED light on the EV3 brick flashes red when it detects a deep crack.

This is the code to take a picture and send it to an AI model, which can then determine whether it is a picture of a fire or a flood. It writes to a file, with the first line showing the type of disaster (fire or flood) and the second showing the number of faces detected if the disaster is a flood. We originally used the Machine Learning for Kids site, which uses Scratch and IBM Watson, to make our AI model, which ran on the Cloud. This year, however, we started using Google Cloud Vision AI and Python's TensorFlow module to make an Edge AI model:

```

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import cv2
import numpy as np
import requests
import argparse
import tensorflow as tf
from PIL import Image
from PIL import ImageFont
from PIL import ImageDraw

key = cv2.waitKey(1)
webcam = cv2.VideoCapture(0)
while True:
    try:
        check, frame = webcam.read()
        ...
        print(check) #prints true as long as the webcam is running
        print(frame) #prints matrix values of each framecd
        ...
        cv2.imshow("Capturing", frame)
        key = cv2.waitKey(1)
        if key == ord('s'):
            cv2.imwrite(filename='saved_img.jpg', img=frame)
            webcam.release()
            print("Image saved!")
            break
        elif key == ord('q'):
            print("Turning off camera.")
            webcam.release()
            print("Camera off.")
            print("Program ended.")
            cv2.destroyAllWindows()
            break
    except KeyboardInterrupt:
        print("Turning off camera.")
        webcam.release()
        print("Camera off.")
        print("Program ended.")
        cv2.destroyAllWindows()
        break

def load_labels(filename):
    with open(filename, 'r') as f:
        return [line.strip() for line in f.readlines()]

picture="saved_img.jpg"
modelpath="model-export_icn_tflite-Fire_Flood_20200130033852-2020-01-30T08_36_48.493E_model.tflite"
inputmean=127.5
inputstd=127.5
labelfile="model-export_icn_tflite-Fire_Flood_20200130033852-2020-01-30T08_36_48.493E_dict.txt"

```



```

interpreter = tf.lite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# check the type of the input tensor
floating_model = input_details[0]['dtype'] == np.float32

# NxNxNx, Hx1, Wx2
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]
img = Image.open(picture).resize((width, height))

# add N dim
input_data = np.expand_dims(img, axis=0)

if floating_model:
    input_data = (np.float32(input_data) - inputmean) / inputstd

interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

output_data = interpreter.get_tensor(output_details[0]['index'])
results = np.squeeze(output_data)

top_k = results.argsort()[::-1][-5:][::-1]
labels = load_labels(labelfile)

for i in top_k:
    if floating_model:
        print('{:08.6f}: {}'.format(float(results[i]), labels[i]))
    else:
        print('{:08.6f}: {}'.format(float(results[i] / 255.0), labels[i]))

confidence=float(results[top_k[0]] / 255.0) #making this a float removes the brackets around the number
print(confidence)

if confidence < 0.7:
    print("safe")
    disasterDetected="safe"
else:
    disasterDetected=labels[top_k[0]]

print(disasterDetected)
message=disasterDetected
f = open("disastertype.txt", "w")
f.write(message)
f.close()

if disasterDetected=="flood":
    ip_request = requests.get('https://get.geojs.io/v1/ip/ip.json')
    my_ip = ip_request.json()['ip']
    geo_request = requests.get('https://get.geojs.io/v1/ip/geo/' + my_ip + '.json')
    geo_data = geo_request.json()
    myLocation={'latitude': geo_data['latitude'], 'longitude': geo_data['longitude']}
    print(myLocation)

    img = Image.open("Flood.png")
    draw = ImageDraw.Draw(img)
    # font = ImageFont.truetype(<font-file>, <font-size>)
    font = ImageFont.truetype("arial.ttf", 72)
    # draw.text((x, y), "Sample Text", (r,g,b))
    draw.text((100, 0), "# " + "latitude: " + geo_data['latitude'] + " longitude: " + geo_data['longitude'], (255,255,255), font=font)
    img.save('FloodLocation.png')
    img = cv2.imread('FloodLocation.png')

    cv2.imshow('Disaster and Coordinates', img)

    print("Flood detected! Detecting number of faces...")
    print("Taking one more picture to search for faces...")
    #pending whether or not to add this.
    #part 1: take pic
    while True:
        #set variables
        key = cv2.waitKey(1)
        webcam = cv2.VideoCapture(0)
        #actual pic taking
        while True:
            #open separate camera window that shows the image while you're taking it
            check, frame = webcam.read()
            ...
            print(check) #prints true as long as the webcam is running
            print(frame) #prints matrix values of each framecd
            ...
            cv2.imshow("Capturing", frame)
            key = cv2.waitKey(1)
            #save image when s key is pressed
            if key == ord('s'):
                cv2.imwrite(filename='saved_img.jpg', img=frame)
                webcam.release()
                print("Image saved!")
            break
        #turn off camera after saving image
    except KeyboardInterrupt:
        print("Turning off camera.")
        webcam.release()
        print("Camera off.")
        print("Program ended.")
        cv2.destroyAllWindows()
        break

    #part 2: detect faces
    xml file
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    #face_cascade = cv2.CascadeClassifier('haarcascade_upperbody.xml')

    #detect faces and assign results (among other things) to variables
    image = cv2.imread("saved_img.jpg")
    grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(grayImage)

    #print type(faces)

    #print "No faces found." if no faces are detected
    if len(faces) == 0:
        print("No faces found.")
    #print number of faces if faces are detected; show image
    else:
        print(faces)
        print(faces.shape)
        print("Number of faces detected: " + str(faces.shape[0]))

        for (x,y,w,h) in faces:
            cv2.rectangle(image, (x,y), (x+w,y+h), (0,255,0), 1)

    cv2.rectangle(image, ((0,image.shape[0] - 25)), (270, image.shape[0]), (255,255,255), -1)
    cv2.putText(image, "Number of faces detected: " + str(faces.shape[0]), (0, image.shape[0] - 10), cv2.FONT_HERSHEY_TRIPLEX, 0.5, (0,0,0), 1)

    cv2.imshow('Image with faces', image)
    #faces=str(faces)
    message="\n"+str(faces.shape[0])
    f = open("disastertype.txt", "a")
    f.write(message)
    f.close()
    break

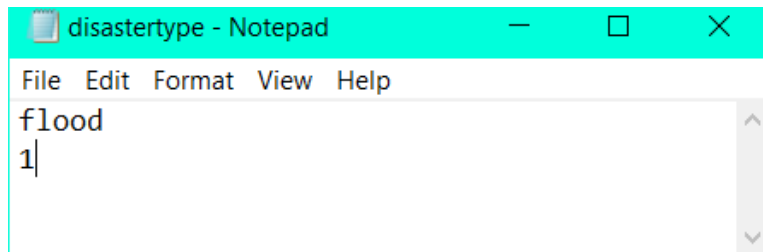
else:
    print("Fire detected!")
    ip_request = requests.get('https://get.geojs.io/v1/ip/ip.json')
    my_ip = ip_request.json()['ip']
    geo_request = requests.get('https://get.geojs.io/v1/ip/geo/' + my_ip + '.json')
    geo_data = geo_request.json()
    myLocation={'latitude': geo_data['latitude'], 'longitude': geo_data['longitude']}
    print(myLocation)

    img = Image.open("Fire.png")
    draw = ImageDraw.Draw(img)
    # font = ImageFont.truetype(<font-file>, <font-size>)
    font = ImageFont.truetype("arial.ttf", 72)
    # draw.text((x, y), "Sample Text", (r,g,b))
    draw.text((100, 0), "# " + "latitude: " + geo_data['latitude'] + " longitude: " + geo_data['longitude'], (255,255,255), font=font)
    img.save('FireLocation.png')
    img = cv2.imread('FireLocation.png')

    cv2.imshow('Disaster and Coordinates', img)

```

This is the file, which now shows that there is a flood and one person trapped in the water:

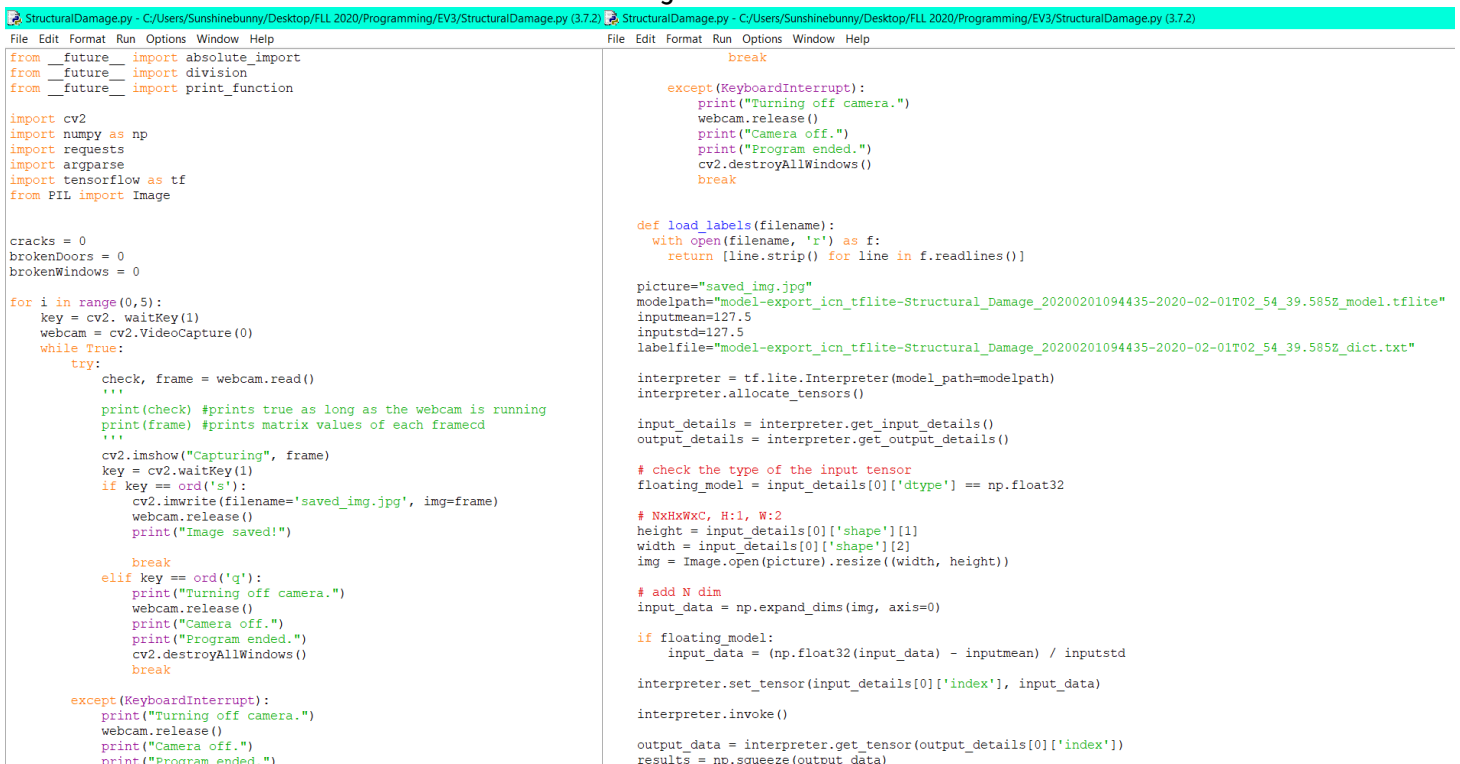


```

disastertype - Notepad
File Edit Format View Help
flood
1

```

This is the code to take a picture of an image of structural damage and send it to an AI model, which can then determine what kind of damage it is.



```

StructuralDamage.py - C:/Users/Sunshinebunny/Desktop/FLL 2020/Programming/EV3/StructuralDamage.py (3.7.2)
File Edit Format Run Options Window Help
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import cv2
import numpy as np
import requests
import argparse
import tensorflow as tf
from PIL import Image

cracks = 0
brokenDoors = 0
brokenWindows = 0

for i in range(0,5):
    key = cv2.waitKey(1)
    webcam = cv2.VideoCapture(0)
    while True:
        try:
            check, frame = webcam.read()
            print(check) #prints true as long as the webcam is running
            print(frame) #prints matrix values of each framecd
            cv2.imshow("Capturing", frame)
            key = cv2.waitKey(1)
            if key == ord('s'):
                cv2.imwrite(filename='saved_img.jpg', img=frame)
                webcam.release()
                print("Image saved!")

            break
        elif key == ord('q'):
            print("Turning off camera.")
            webcam.release()
            print("Camera off.")
            print("Program ended.")
            cv2.destroyAllWindows()
            break
    except (KeyboardInterrupt):
        print("Turning off camera.")
        webcam.release()
        print("Camera off.")
        print("Program ended.")

        break

    except (KeyboardInterrupt):
        print("Turning off camera.")
        webcam.release()
        print("Camera off.")
        print("Program ended.")
        break

def load_labels(filename):
    with open(filename, 'r') as f:
        return [line.strip() for line in f.readlines()]

picture="saved_img.jpg"
modelpath="model-export_icn_tflite-Structural_Damage_20200201094435-2020-02-01T02_54_39.585Z_model.tflite"
inputmean=127.5
inputstd=127.5
labelfile="model-export_icn_tflite-Structural_Damage_20200201094435-2020-02-01T02_54_39.585Z_dict.txt"

interpreter = tf.lite.Interpreter(model_path=modelpath)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# check the type of the input tensor
floating_model = input_details[0]['dtype'] == np.float32

# NxHxWxC, H:1, W:2
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]
img = Image.open(picture).resize((width, height))

# add N dim
input_data = np.expand_dims(img, axis=0)

if floating_model:
    input_data = (np.float32(input_data) - inputmean) / inputstd

interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

output_data = interpreter.get_tensor(output_details[0]['index'])
results = np.squeeze(output_data)

if floating_model:
    input_data = (np.float32(input_data) - inputmean) / inputstd

interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

output_data = interpreter.get_tensor(output_details[0]['index'])
results = np.squeeze(output_data)

top_k = results.argsort()[-5:][::-1]
labels = load_labels(labelfile)

for i in top_k:
    if floating_model:
        print('{:08.6f}: {}'.format(float(results[i]), labels[i]))
    else:
        print('{:08.6f}: {}'.format(float(results[i]) / 255.0, labels[i]))

confidence=float(results[[top_k[0]]] / 255.0)

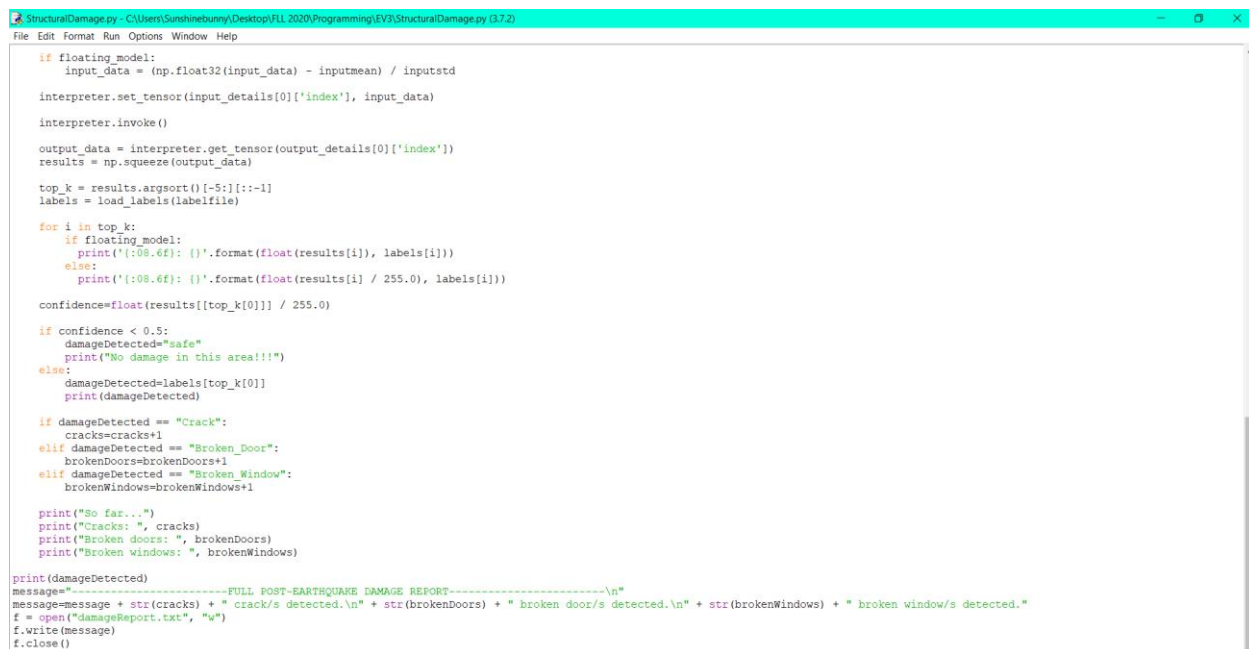
if confidence < 0.5:
    damageDetected="safe"
    print("No damage in this area!!!")
else:
    damageDetected=labels[top_k[0]]
    print(damageDetected)

if damageDetected == "Crack":
    cracks=cracks+1
elif damageDetected == "Broken Door":
    brokenDoors=brokenDoors+1
elif damageDetected == "Broken Window":
    brokenWindows=brokenWindows+1

print("So far...")
print("Cracks: ", cracks)
print("Broken doors: ", brokenDoors)
print("Broken windows: ", brokenWindows)

print(damageDetected)
message="-----FULL POST-EARTHQUAKE DAMAGE REPORT-----\n"
message=message + str(cracks) + " crack/s detected.\n" + str(brokenDoors) + " broken door/s detected.\n" + str(brokenWindows) + " broken window/s detected."
f = open("damageReport.txt", "w")
f.write(message)
f.close()

```



```

StructuralDamage.py - C:/Users/Sunshinebunny/Desktop/FLL 2020/Programming/EV3/StructuralDamage.py (3.7.2)
File Edit Format Run Options Window Help

if floating_model:
    input_data = (np.float32(input_data) - inputmean) / inputstd

interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

output_data = interpreter.get_tensor(output_details[0]['index'])
results = np.squeeze(output_data)

top_k = results.argsort()[-5:][::-1]
labels = load_labels(labelfile)

for i in top_k:
    if floating_model:
        print('{:08.6f}: {}'.format(float(results[i]), labels[i]))
    else:
        print('{:08.6f}: {}'.format(float(results[i]) / 255.0, labels[i]))

confidence=float(results[[top_k[0]]] / 255.0)

if confidence < 0.5:
    damageDetected="safe"
    print("No damage in this area!!!")
else:
    damageDetected=labels[top_k[0]]
    print(damageDetected)

if damageDetected == "Crack":
    cracks=cracks+1
elif damageDetected == "Broken Door":
    brokenDoors=brokenDoors+1
elif damageDetected == "Broken Window":
    brokenWindows=brokenWindows+1

print("So far...")
print("Cracks: ", cracks)
print("Broken doors: ", brokenDoors)
print("Broken windows: ", brokenWindows)

print(damageDetected)
message="-----FULL POST-EARTHQUAKE DAMAGE REPORT-----\n"
message=message + str(cracks) + " crack/s detected.\n" + str(brokenDoors) + " broken door/s detected.\n" + str(brokenWindows) + " broken window/s detected."
f = open("damageReport.txt", "w")
f.write(message)
f.close()

```


This is the code that controls the launcher:

```

main.py
1  #!/usr/bin/env python3
2  # so that script can be run from Brickman
3
4  from ev3dev.ev3 import *
5  from time import sleep
6
7  Sound.beep(0)
8
9  m1 = LargeMotor('outA') # either LargeMotor for the big motor or MediumMotor for the small motor
10 m2 = LargeMotor('outB')
11 m3 = LargeMotor('outC')
12 m4 = LargeMotor('outD')
13
14 fhand=open("disastertype.txt")
15 for line in fhand:
16     if line == "fire" or line == "safe":
17         contents=1
18         # if it's a fire there wouldn't be a next line so the loop will finish right here
19     else:
20         contents=line
21         # you might be wondering, what if the line is flood?
22         # but then if it is, it will go to the next line and replace the contents of the variable from "flood" to the number of faces in
23
24 int(contents)
25 print(contents)
26
27 for i in range (0, int(contents)+1):
28     #while True:
29     if i != 0:
30         m4.run_to_rel_pos(position_sp=135, speed_sp=1000, stop_action="hold") #OutD
31         m3.run_to_rel_pos(position_sp=135, speed_sp=1000, stop_action="hold") #OutC
32     for i in range (0, 300):
33         m2.run_to_rel_pos(position_sp=360, speed_sp=1000, stop_action="hold") #OutB
34         m1.run_to_rel_pos(position_sp=360, speed_sp=1000, stop_action="hold") #OutA
35     # position_sp is number of rotations: 360 is 1 rotation 180 is 1/2 rotation
26

```

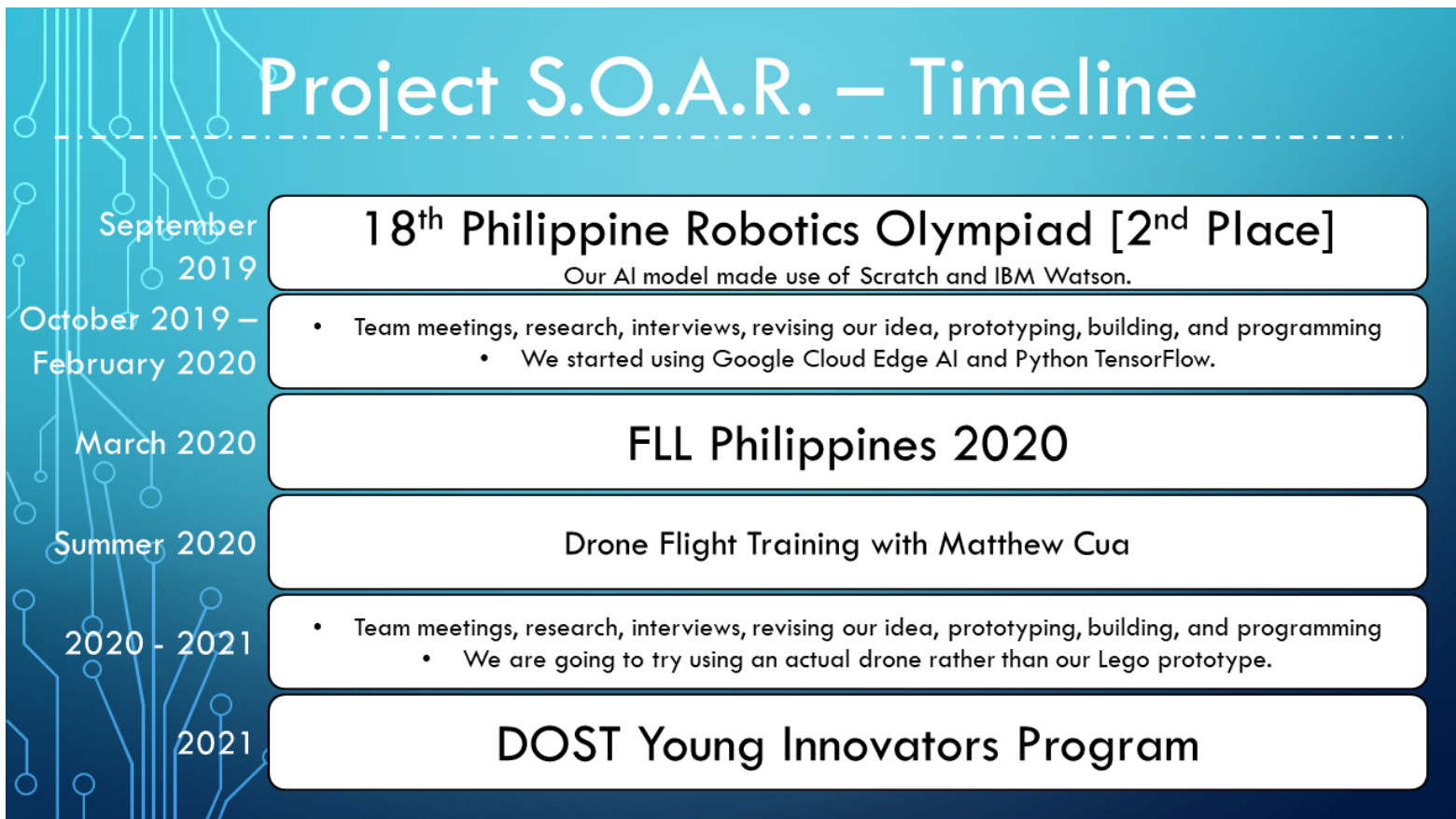
SCOPE AND LIMITATIONS

Some limitations were observed as part of the rules stated that we could only use what was available in our Lego sets and EV3 kits for our prototype:

1. We could not make an actual flying drone because we did not have the technical skills yet to create a flying drone out of Lego that could support the weight of the EV3 brick. Instead, we

attached SOAR to a monorail mechanism made of bricks in an EV3 kit. This helped us make our robot go back and forth without touching the ground, to simulate a flying drone.

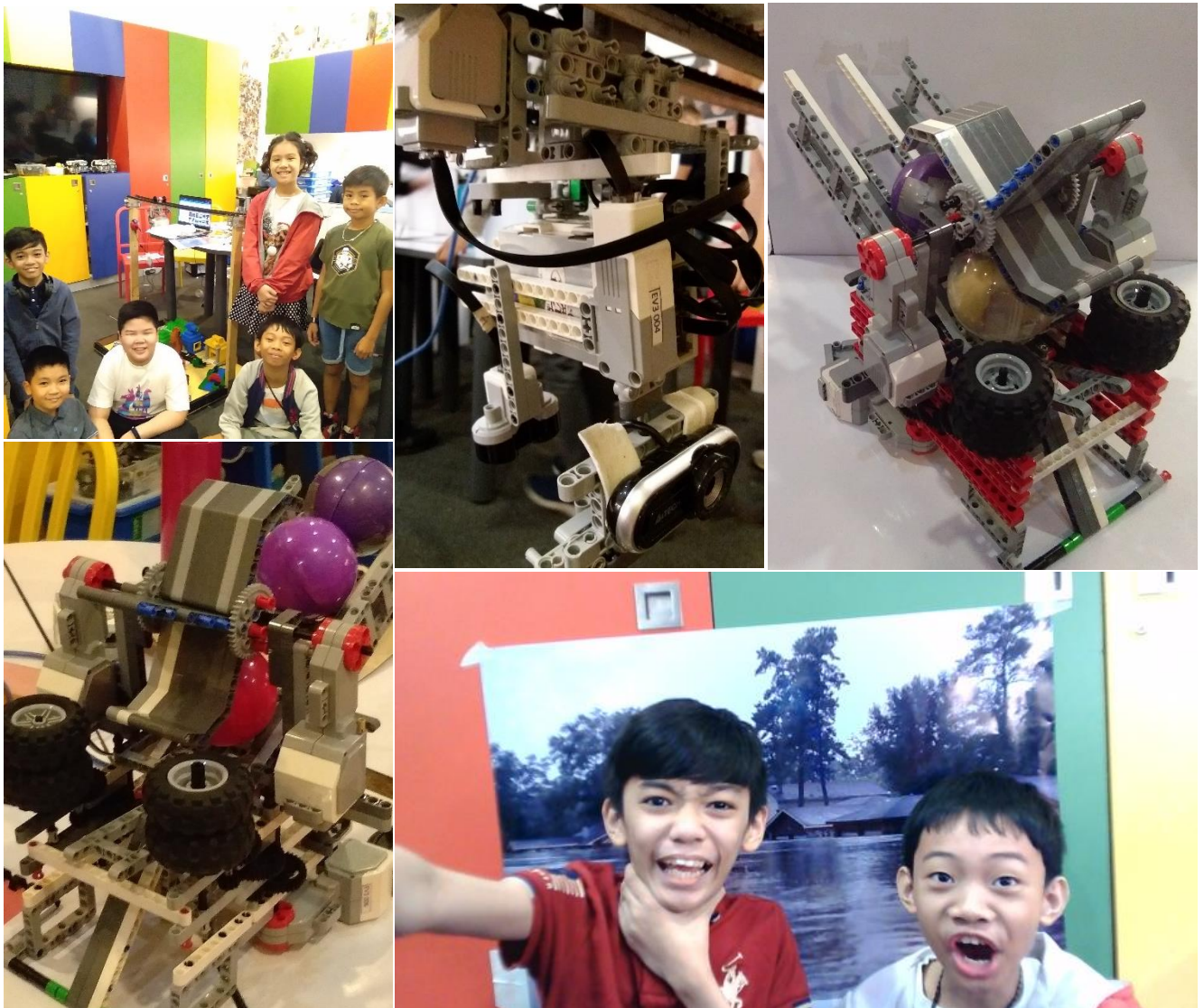
2. Our team is composed of 10 to 13-year-olds. We still do not have the knowledge to fully explain how the prototype would be constructed in the real world and its actual cost. We could only do research to prove that it is possible and affordable. We aim to follow the timeline below in order to implement a more realistic prototype in the future:



CHAPTER 4

RESULTS

The prototype ran successfully, demonstrating how the robot could go around the area and identify pictures of fires and floods using Google Cloud's visual recognition service. It could also detect the number of flood victims and write this to a file. The launcher was able to read the file and throw that number of paper life jackets to the victims. The structural damage inspection simulation also worked, and the robot was able to detect cracks, broken windows, and broken doors, and create a file showing the data.





```

pi@raspberrypi:~/AIY-projects-python/src/examples/vision $ ./image_classification
n.py --input Flood1.jpg
Result 0: amphibian/amphibious vehicle (prob=0.591797)
pi@raspberrypi:~/AIY-projects-python/src/examples/vision $ ./image_classification
n.py --input Flood2.jpg
Result 0: lakeside/lakeshore (prob=0.166138)
Result 1: gondola (prob=0.154907)
Result 2: boathouse (prob=0.117859)
pi@raspberrypi:~/AIY-projects-python/src/examples/vision $ ./image_classification
n.py --input Fire1.jpg
Result 0: torch (prob=0.323730)
Result 1: stage (prob=0.159058)
pi@raspberrypi:~/AIY-projects-python/src/examples/vision $ ./image_classification
n.py --input Fire2.jpg
Result 0: church/church building (prob=0.511719)
Result 1: fireboat (prob=0.117798)
pi@raspberrypi:~/AIY-projects-python/src/examples/vision $
  
```

My First Project

Fire_Flood LABEL STATS EXPORT DATA

IMPORT IMAGES TRAIN EVALUATE TEST & USE

Models TRAIN NEW MODEL

Fire_Flood_20200126094756

Average precision ?

1

Precision* ? 100%

Recall* ? 100%

* Using a score threshold of 0.5

Model ID ?	ICN8444473601691746304
Created	Jan 26, 2020, 9:48:28 AM
Base model	None
Data	98 images
Model type	Cloud
Train cost	16 node hours
Deployment state	Not deployed

SEE FULL EVALUATION

Train new model

1 Define your model

Model name *

Fire_Flood_20200126121849

 Cloud hosted

Host your model on Google Cloud for online predictions

 Edge

Download your model for offline/mobile use

CONTINUE

2 Set a node hour budget

START TRAINING

CANCEL

We tried testing Edge AI through the Google AIY Vision kit by showing pictures of burning buildings, which were classified as torches, fireboats, stoves, candles, and fire screens. When we tried using the Vision camera to identify images of floods, it categorized these images as lakes, boathouses and amphibious vehicles. These results show that there is potential to detect disasters, but the model would have to be retrained for this particular purpose.

With the help of our friends from the Google community, we were able to train and program two actual Edge AI models: one to detect fires and floods, and one to detect structural damage. We were also able to use Python's OpenCV module to make a program that could detect the number of people in a picture. The facial recognition was not always accurate due to the lighting in the room, or because the people in the picture were not facing forward. Because of this, SOAR will also use thermal sensors in the future to help it detect people more accurately, so that the right amount of life jackets and inflatables can be dropped and more people can be saved during a flood. However, facial recognition will still be needed, because thermal sensors cannot distinguish between objects that are near each other and have similar temperatures.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

With this prototype, we have proven that it is possible to train an AI model to detect disasters with image processing. In the future, we can apply this model to a drone camera to be used in disaster recovery and rescue operations. But what if we combine this with existing technology, such as thermal and gas sensors? Our robot would be able to help rescuers determine which disasters have higher risks. Calamities detected by both the AI and the sensors would be more accurate than the ones identified by only one of them. Even if SOAR will use artificial intelligence, sensors are also necessary because facial recognition is not always accurate due to lighting or the position of the people in the picture, and gas is invisible, making it difficult to detect using image recognition. However, artificial intelligence is still needed for accurate face detection because thermal sensors cannot distinguish between objects that are near each other if they have similar temperatures.

SOAR would also be able to assist in the three phases of disaster recovery: rescue, or locating and rescuing people; after a disaster, such as when people re-enter buildings after an earthquake; and mitigation, or preventing disasters. For the rescue phase, our robot uses facial recognition and thermal sensors to find people trapped in the waters after a tsunami or maritime disaster. To help out with the after a disaster phase, before people are allowed to go back inside a structure, our robot searches for damage inside, so that when the evacuees return to the building they will be safe and sound. Our future vision for the drone is for it to be able to go around an area daily to search for ongoing or potential disasters (such as damaged roads and structures), so that

the local government can immediately attend to these. This can assist rescuers ,in the mitigation phase.

This is more than a competition to us. This summer, we will learn how to fly drones and get certified with the help of Mr. Matthew Cua, an expert on drones. Next year, we will also apply for the DOST Young Innovators Program so that, if our project is approved, we will be given the money to actually make SOAR and implement it in disaster prone areas to help save lives during calamities. We are dedicated to making SOAR a reality, to help save the lives of disaster victims and rescuers alike.

Because together, we can change the world, with unity, teamwork, and of course, the power of technology. Together, we can SOAR.

REFERENCES

- Everything You Need to Know about Drones in Disaster Recovery
(<https://www.gleassociates.com/everything-you-need-to-know-about-drones-in-disaster-recovery/>)
- How Drones are being used in Disaster Management
(<https://geoawesomeness.com/drones-fly-rescue/>)
- 5 Ways Drones are Being Used in Disaster Relief
(<https://safetymanagement.eku.edu/blog/5-ways-drones-are-being-used-for-disaster-relief/>)
- Drones used to assess damage after disasters
(<https://phys.org/news/2014-04-drones-disasters.html>)
- New drone-based approach to detecting structural damage during extreme events such as earthquakes
(<https://phys.org/news/2017-08-drone-based-approach-extreme-events-earthquakes.html>)
- Florence flooding: Drone footage shows hurricane damage in Wilmington area
(<https://abc11.com/weather/drone-view-hurricane-floods-wilmington-with-2-feet-of-rain/4265379/>)
- Drones Increasingly Get Ahead of Disaster Damage
(<https://statetechmagazine.com/article/2018/10/drones-increasingly-get-ahead-disaster-damage>)
- Phases of Disaster Recovery: Emergency Response for the Long Term
(<https://reliefweb.int/report/world/phases-disaster-recovery-emergency-response-long-term>)

-
- Disaster rescuers ready to help, but need support
(<https://www.rappler.com/move-ph/issues/disasters/89059-rescue-march-2015>)
 - Facts + Statistics: Global catastrophes
(<https://www.iii.org/fact-statistic/facts-statistics-global-catastrophes>)
 - MicroPython Documentation
(https://sites.google.com/site/ev3python/learn_ev3_python/using-motors)
 - Thermal Sensors
(<https://sciencing.com/advantages-disadvantages-infrared-detectors-6151444.html>)
 - Interview with Matthew Cua (Cua, M. (2019, August 10). Personal interview.)
 - Interview with Josef Monje (Monje, J. (2019, August 17). Personal Interview.)
 - Interview with Sony Valdez (Valdez, S. (2019, August 18). Personal Interview.)
 - Interview with Eric Tingatinga (Tingatinga, E. (2019, September 29). Personal Interview.)
 - Interview with Nicholas Osborn (Osborn, N. (2020, January 9). Email Interview.)