Professor Liang

Computer Science 43300 - Internet of Things

December 5, 2018

IoT Theft Detection System

By Sundeep Kakar, Navdeep Singh

Sundeep Kakar: [skakar@iu.edu](mailto:skakar@iu.edu)  
 2000129394

Navdeep Singh: [navdsing@iu.edu](mailto:navdsing@iu.edu)

0003416142

**I. Introduction**

This project explores developing a Theft Detection System as an IoT application using a Raspberry Pi 3 B+. This system is implemented over a CoAP server allowing for remote access and reporting. The system comprises of a buzzer, a camera module, and an LED. The system has an active mode and a standby mode. Depending on these modes, the buzzer is activated to indicate “theft”.

**II. Theory**

Our project utilizes SimpleCV, a computer vision library for python to analyze the and compare images. SimpleCV is a framework that offers easy implementations of the OpenCV library for python. It is functional with the images or video stream from a webcam, camera module, or IP cameras. OpenCV is a library of programming functions mainly aimed at real-time computer vision. This makes the library very CPU intensive. SimpleCV framework allows easy extraction, sorting, and filtering of image features. Image manipulations are fast making in order for the SimpleCV framework to have an optimal and clean link to the OpenCV library.

The python implemented involves image arithmetics to actually sense motion. SimpleCV looks at images as large matrices of pixel values ranging from 0 to 255. This allows for easy linear algebraic operations on the matrices. The matrices cannot contain any negative value as the RGB values of a pixel range from 0-255 as well. We use this to our advantage to compare images. For a stationary camera, we implement an algorithm where we assume that for two images where no motion has occurred, the respective pixel values will remain unchanged. So we subtract the two images i.e. subtract the respective pixel values. For unchanged pixels, this results in RGB(0,0,0) which is black. On previewing the subtracted image, it can be observed that all non-black pixels are the ones where motion has occurred. Finding the mean of this matrix gives us a percentage motion value of the matrix. This is then compared to a threshold value to conclude motion or not

CoAP or Constrained Application Protocol is a web protocol based on the REST API model. CoAP servers make resources available under a URL, and access is given to resources using methods such as GET, PUT, POST, and DELETE and OBSERVE. It’s very similar in terms of implementation when compared to HTTP. It can carry different types of data payload. That includes data formats such as XML and JSON. This protocol was designed for applications integrating machine-to-machine applications. CoAP can be implemented in smart devices. The request/response layer responsibility consists of getting the request and response that hold the method codes and response codes without having issues that consist of losing data or having it duplicated. For CoAP to be able to handle congestion and being able to enable light weigh applications, it must operate over UDP. Operating over the UDP, it implements two types of messages known as confirmable and non-confirmable in order to see the arrival of the packets and get acknowledgements. CoAP meets requirements such as extremely low overhead and multicast support.

For our project, we use Copper for CHrome (Cu4Cr) as a CoAP user-agent. Copper functions as a client and has to analyze the data we send using CoAP, we used Wireshark. Wireshark is a network-protocol analyzer. It lets you see and analyze network packets. It inspects protocols such as UDP, TCP, CoAP, and HTTP. It allows users to browse network data via a Graphical User Interface. In our project, we use it analyze CoAP packets being sent from the server to the client. This lets us examine our payload being sent through the CoAP server.

**III. Implementation & Analysis**

The purpose of our project is to solve a real world issue while also paying close attention to energy consumption and efficiency. A basic theft detection system would comprise of a IRw sensor aimed at an object where the sensor triggers an alarm when it can’t sense the object anymore. To optimize the same, we implement a camera over a sensor and through image manipulation detect the object. This allows to save the images taken, thereby giving us more information about the change of position of the object compared to a Yes or No (boolean value) returned by a simple sensor. Furthermore, we archive all images taken by the camera module in separate folders allowing us to go back and check on the item. This Theft Detection System has been implemented over a CoAP server using multiple resources.

Hello-World Resource:

The HelloWorld resource is part of the project to demonstrate basic property of CoAP and to verify that the CoAP server is running and functional.

Set-Initial-Image Resource:

This resource uses the PiCamera library to take an initial image with the object set in frame. This is an important resource as it gives the system a reference to what the current images pulled should look like with the object in frame. It invokes start\_preview() method to show what the camera is looking at that point. This gives user an opportunity to adjust the object in frame. After a 5 second preset delay, the camera takes an image and stores it in the folder “Initial Item Image” in the pi system files. This folder is referenced anytime a function needs to pull the image of the initial object.

Set-Mode Resource:

The theft detection system developed by us has two modes: Active Mode and Standby Mode. When the system is in active mode, it means that it’s “under lockdown”. Therefore, if any motion were to be detected, the buzzer alarm would start beeping. When the system is in Standby Mode, it proceeds as normal but does not beep when motion is detected. The Set-Mode resource returns the current mode of the system when a GET request is initiated using Copper. To change the same, we send in a PUT request with the mode configuration that we want to switch to (0 or 1). The LED also indicates the system mode to the user. When the system is in active mode, the LED is ON and when the system is in standby mode, the LED is OFF.

Theft-Detection-System Resource:

This is the most important resource of the circuit. On running a GET request using Copper on this resource, we receive a message ("ENGAGE / DEMO"). This resource has two ways of functioning. The difference between DEMO mode and ENGAGE mode is that DEMO only runs once. It collects an image, compares it to the initial image, and returns the result regardless of motion found or not. The ENGAGE mode however is the full fledged functioning of our theft detection system. It pulls an image from the camera module, compares it, and only returns if motion is detected. If no motion is detected, it loops by waiting for a few seconds, taking a new image and comparing that to the initial image. Before the end of the loop it sets the previous image i.e. the initial image to the one taken for comparison right before. The logic of this is that once the system has ran an iteration that showed no motion, the image taken during the first iteration can be set as the initial image we’re comparing to. This allows us to decentralize the system from the initial image taken during the setup phase. After the first iteration the system becomes independent of the initial image taken during setup phase, while also not overwriting it thereby allowing us to refer back to it if needed. Once the current image has been pulled from the camera module. IMAGE.show() method part of the SimpleCV library is called allowing the user to see the initial image and then the current image taken. Now that we have both these images stored as variables we find the subtracted image. SimpleCV defines images as an array of pixel values where each pixel has an RGB value. When we get [difference = intitialImage - currentImage], the difference is an image where the corresponding pixel values for the two images have been subtracted. The logic states that if no motion took place, then pixel values should be equal. If pixel values are equal subtracting them would result in the lowest RGB value which is rgb(0,0,0) or BLACK. Any motion that did take place, pixels for that part of the frame will be non-black. Using getNumpy().mean() method we fetch the mean pixel value of the matrix and compare it to a threshold. The threshold value is determined by factors such as lighting and object distance. If the mean value turns out to be higher than the threshold, it returns a GET packet with payload “Motion Detected”. In ENGAGE mode, the code runs indefinitely till motion is detected.

**IV. Conclusion**

In this project, we learned how to implement a Theft Detection System over CoAP using a Raspberry Pi 3 B+. The CoAP server could be accessed by a client on the same network. We learned how to interface GPIO processes and a CoAP server. We learned a lot about Computer Vision libraries in python and implementation of image math. The images are manipulated as an array of pixel values. We learned about REST API methods and how to implement them.

**V. Reference**

"Image Arithmetic — Tutorial". *Tutorial.Simplecv.Org*, 2018, http://tutorial.simplecv.org/en/latest/examples/image-math.html#color-space. Accessed 6 Dec 2018.

"Coapthon.Defines.Content\_Types Example". *Programtalk.Com*, 2018, https://programtalk.com/python-examples-amp/coapthon.defines.Content\_types/. Accessed 6 Dec 2018.

G.Tanganelli, C. Vallati, E.Mingozzi, "CoAPthon: Easy Development of CoAP-based IoT Applications with Python", IEEE World Forum on Internet of Things (WF-IoT 2015)

"Detecting A Car In A Parking Lot — Tutorial". Tutorial.Simplecv.Org, 2018, http://tutorial.simplecv.org/en/latest/examples/parking.html. Accessed 6 Dec 2018.

**Appendix I:**

**Code (Final\_Project.py)**

'''

Class : CSCI 43300

Name : Sundeep Kakar, Navdeep Singh

Project : Theft Detection System

'''

# Import Libraries #

from coapthon.server.coap import CoAP

from coapthon.resources.resource import Resource

from picamera import PiCamera

from SimpleCV import \*

from time import sleep

from datetime import datetime

from coapthon import defines

import os

import time

# RPi.GPIO library used with Buzzer and LED implementation

import RPi.GPIO as GPIO

from time import sleep

# Global Variables #

BuzzerPin = 11 # Buzzer GPIO Pin

LEDPin = 8 # LED GPIO Pin

MODE\_VAL = '0' # Theft Detection System Mode Variable

# IMAGE FUNCTIONS #

# Function: Takes Initial Image of object

def takeInitialImage():

print("\nTaking Initial Image...")

camera = PiCamera()

camera.start\_preview()

sleep(7) # Delay added to let the camera adjust to the lighting

camera.capture('/home/pi/Desktop/Initial\_Item\_Image/init\_image.png')

camera.stop\_preview()

camera.close() # \*Closeing important since Camera is envokes in multiple function

# Function: Takes Images

def takeImage():

camera = PiCamera() # Starts the camera

camera.start\_preview() # Begin preview

delay = 5 # Delay between images

print('Beginning Capture... ')

camera.capture('/home/pi/Desktop/ImagesCaptured/'+ str(datetime.now()) + '.png') #To Archive Folder

camera.capture('/home/pi/Desktop/Pulled\_Item\_Image/pulled\_image.png') #To Comparison Folder

sleep(delay) # Delay in between pictures

camera.stop\_preview()

camera.close()

# Function: Runs Theft Detection System in Demo Mode

def ImageDemo():

init\_Item = Image("/home/pi/Desktop/Initial\_Item\_Image/init\_image.png")

sleep(5)

curr\_Item = Image('/home/pi/Desktop/Pulled\_Item\_Image/pulled\_image.png')

print('\nImages ->')

init\_Item.show()

sleep(5)

curr\_Item.show()

sleep(5)

threshold = 2.5 # Threshold value to be compared with the image matrix mean

prev\_Item = init\_Item

diff = curr\_Item - prev\_Item

matrix = diff.getNumpy() # Matricizing values into array

mean = matrix.mean()

print("Matrixval = ",mean) # Prints mean for analysis

heading = "Image Difference = Motion"

diff.drawText(heading)

diff.show()

sleep(5)

flag = 0

if mean >= threshold:

print("Motion Detected!")

flag = 1

if(checkMode()=='1'):

print("Buzz")

Buzzloop(flag)

else:

print("No Change In Motion")

flag = 0

return MotionStatus(flag) # returns string value indicating if Motion = T or F

# Function: Runs Theft Detection System in Engage Mode. Engage mode runs indefinitely till motion is detected

def ImageAnalysis():

init\_Item = Image("/home/pi/Desktop/Initial\_Item\_Image/init\_image.png")

prev\_Item = init\_Item

flag = 0

while True:

camera = PiCamera() #starts the camera

camera.capture('/home/pi/Desktop/ImagesCaptured/'+ str(datetime.now()) + '.png') #To Archive

sleep(5)

camera.capture('/home/pi/Desktop/Pulled\_Item\_Image/pulled\_image.png') #To Comparison Folder

camera.close()

curr\_Item = Image('/home/pi/Desktop/Pulled\_Item\_Image/pulled\_image.png')

print('\nImages ->')

prev\_Item.show()

sleep(2)

curr\_Item.show()

sleep(2)

threshold = 2.5 # Threshold value

difference\_Item = curr\_Item - prev\_Item

matrix = difference\_Item.getNumpy()

mean = matrix.mean()

print("Matrixval = ",mean)

difference\_Item.show()

sleep(5)

if mean >= threshold:

print("Motion Detected!")

if(checkMode()=='1'):

flag = 1

Buzzloop(flag)

else:

print("No Change In Motion")

prev\_Item = curr\_Item # Decentralizes the initial image and loops with current and previous images

if(flag == 1):

break

return MotionStatus(flag)

# Function: Returns value Security mode (Active or Standby)

def checkMode():

global MODE\_VAL

return MODE\_VAL

# Function: Returns string Security mode (Active or Standby)

def displayMode():

global MODE\_VAL

if(MODE\_VAL=='0'):

return "Standby Mode..."

elif(MODE\_VAL=='1'):

return "Active Mode..."

def MotionStatus(status):

if(status == 0):

return "No Motion Detected..."

elif(status == 1):

return "Motion Detected!"

# Function: Sets Security mode (Active or Standby)

def setMode(mv):

global MODE\_VAL

MODE\_VAL = mv

print(MODE\_VAL)

if(MODE\_VAL=='0'):

print("MODE: STANDBY")

GPIO.output(LEDPin, GPIO.LOW) # LED indicates the mode

elif(MODE\_VAL == '1'):

print("MODE: LOCK DOWN!")

GPIO.output(LEDPin, GPIO.HIGH)

return MODE\_VAL

# Function: Runs the buzzer module

def Buzzloop(num):

if(num==1):

max\_time = time.time() + 10

while time.time()<max\_time: # Timer implemented to end loop after 10 seconds

GPIO.output(BuzzerPin, GPIO.HIGH)

time.sleep(0.01)

GPIO.output(BuzzerPin, GPIO.LOW)

time.sleep(0.01)

# CoAP RESOURCES #

# Resource class: Basic Hello World Resource

class HelloWorld(Resource):

def \_\_init\_\_(self, name="HelloWorld", coap\_server=None):

super(HelloWorld, self).\_\_init\_\_(name, coap\_server, visible=True, observable=True, allow\_children=True)

self.payload = str("Hello!")

self.resource\_type = "rt1"

self.content\_type = "text/plain"

self.interface\_type = "if1"

def render\_GET(self, request):

#self.payload = "Hello"

self.content\_type = "text/plain"

return self

def render\_PUT(self, request):

self.payload = request.payload

return self

def render\_POST(self, request):

res = HelloWorld()

res.location\_query = request.uri\_query

res.payload = request.payload

return res

def render\_DELETE(self, request):

return True

# Resource class: GET runs takeinitialImage()

class InitialImageSet(Resource):

def \_\_init\_\_(self, name="InitialImageSet", coap\_server=None):

super(InitialImageSet, self).\_\_init\_\_(name, coap\_server, visible=True, observable=True, allow\_children=True)

self.payload = 'Initial Image Saved!'

def render\_GET(self, request):

takeInitialImage()

return self

def render\_PUT(self, request):

self.payload = request.payload

print(self.payload)

return self

def render\_POST(self, request):

res = InitialImageSet()

res.location\_query = request.uri\_query

res.payload = request.payload

return res

def render\_DELETE(self, request):

return True

# Resource class: GET and PUT Security mode

class SetMode(Resource):

def \_\_init\_\_(self, name="SetMode", coap\_server=None):

super(SetMode, self).\_\_init\_\_(name, coap\_server, visible=True, observable=True, allow\_children=True)

self.payload = "Enter Buzzer Mode? [0-OFF | 1-ON]: "

def render\_GET(self, request):

self.payload = displayMode()

return self

def render\_PUT(self, request):

self.payload = request.payload

setMode(self.payload)

print(self.payload)

return self

def render\_POST(self, request):

res = SetMode()

res.location\_query = request.uri\_query

res.payload = request.payload

return res

def render\_DELETE(self, request):

return True

#Theft Detection Value

TDSval = 0

# Resource class: GET returns mode chosen, PUT allows to you to choose between Engage or Demo

class BeginTDS(Resource):

def \_\_init\_\_(self, name="BeginTDS", coap\_server=None):

super(BeginTDS, self).\_\_init\_\_(name, coap\_server, visible=True, observable=True, allow\_children=True)

self.payload = "ENGAGE / DEMO"

def render\_GET(self, request):

global TDSval

if(TDSval == 1):

print("Engage!")

takeImage()

self.payload = ImageAnalysis()

return self

elif(TDSval == 2):

print("Demo!")

takeInitialImage()

takeImage()

self.payload = ImageDemo()

return self

elif(TDSval == 0):

return self

print("Here")

return self

def render\_PUT(self, request):

global TDSval

if(request.payload in {"Engage","ENGAGE","E"}):

TDSval = 1

elif(request.payload in {"D","DEMO","Demo"}):

TDSval = 2

return self

def render\_POST(self, request):

res = BeginTDS()

res.location\_query = request.uri\_query

res.payload = request.payload

return res

def render\_DELETE(self, request):

return True

# COAP SERVER #

class CoAPServer(CoAP):

def \_\_init\_\_(self, host, port):

CoAP.\_\_init\_\_(self, (host, port))

self.add\_resource('HelloWorld/', HelloWorld())

self.add\_resource('InitialImageSet/', InitialImageSet())

self.add\_resource('SetMode/', SetMode())

self.add\_resource('TheftDetectionSystem/',BeginTDS())

# MAIN FUNCTION #

def main():

server = CoAPServer("0.0.0.0", 5683) # Server IP

print("CoAP Server Running...")

try:

server.listen(10)

except KeyboardInterrupt:

print ("Server Shutdown")

server.close()

GPIO.output(BuzzerPin, GPIO.LOW)

GPIO.cleanup() # Frees the GPIO so pins are not in use anymore

print ("Exiting...")

if \_\_name\_\_ == '\_\_main\_\_':

GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location

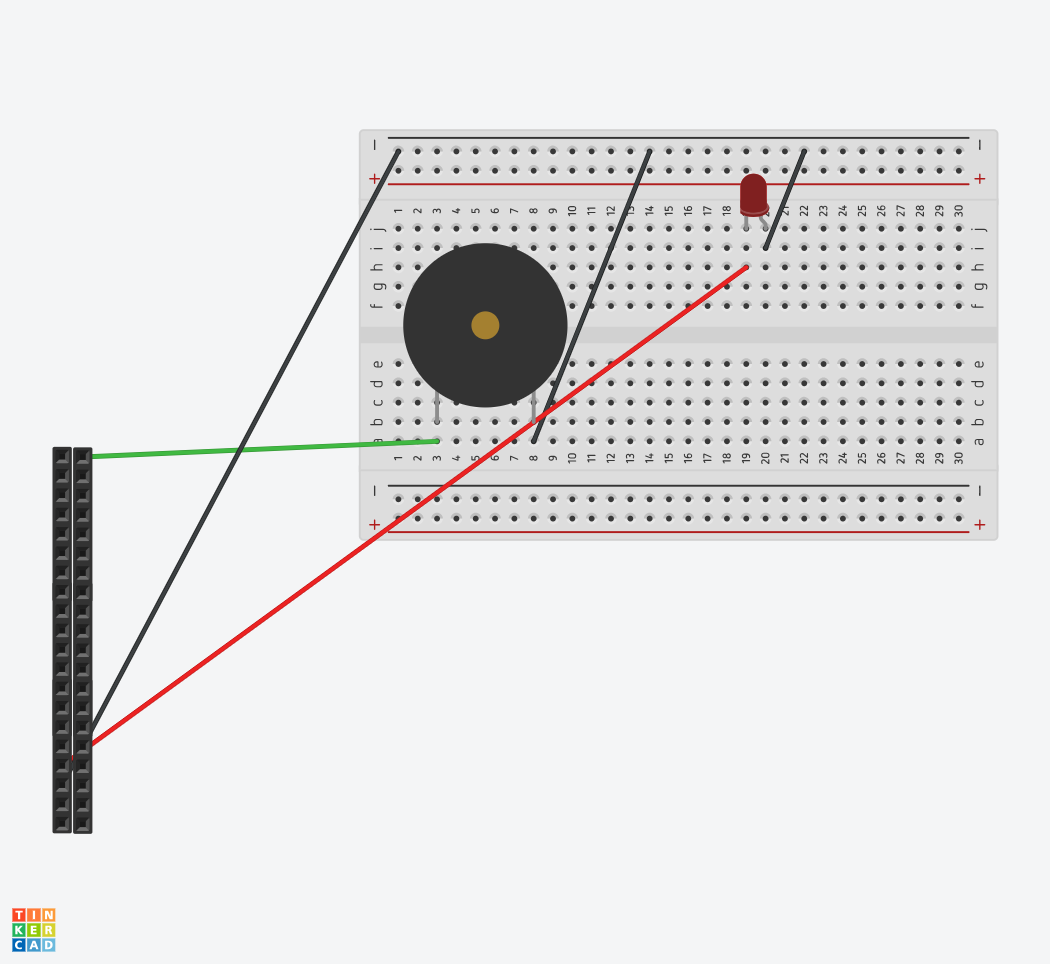
GPIO.setup(BuzzerPin, GPIO.OUT)

GPIO.output(BuzzerPin, GPIO.LOW)

GPIO.setup(LEDPin, GPIO.OUT, initial=GPIO.LOW)

main()

**Appendix II: Schematic Diagram**

****

Buzzer GPIO Pin = 11

LED GPIO Pin = 8

**Appendix III: Demonstration**

Link: <https://www.youtube.com/watch?v=uYGOCEKzvzU&feature=youtu.be>