Horus Event Tracker, named after the all-seeing-eye of Horus, is an application that allows administrators and space owners to monitor the crowd engagement of ad-hoc events and exhibitions. The application consists of three modules: The sensor-enabled Arduino connected to a Raspberry Pi, a Raspberry Pi and a Pi Camera, as well as social media analytics.

Using the sensors connected to the Arduino, a specific exhibit or small area can be monitored. The sound level of the area, the number of people passing by, as well as the number of wireless and bluetooth connections in the area is used to measure it's ‘Buzz’, which denotes how active the surrounding area is. The light sensor as well as the button module is used for users to rate the exhibition, or the specific area of the event, providing a preliminary glance at the user response. The average dwell time of visitors is also measured using the light and infrared sensor, which reflects how interesting the public finds the exhibit. The Raspberry Pi collects the data from the Arduino and updates the server every half an hour. In practice, this can be a highly scalable solution. Instead of prototyping boards, a custom sensor can be manufactured at low cost and placed in relevant areas and relocated as needed. The data from different sensor modules can be compared in order to determine which parts of the event are more popular, and which are less favored by the public. A limitation of this module is that it only scans for light and movement in one direction, thus is more suitable for exhibitions with an orientation, like a wall mounted installation, as opposed to a sculpture.

The Raspberry Pi camera uses the openFrameworks library and Footfall motion tracking application to count the number of people in a certain area. It monitors the traffic in the level, using these metrics to calculate the number of people in the area over time. By using a combination of the flat number of visitors, as well as the rate of change of visitors, event organizers can analyse the peak times, as well as extrapolate the ‘turnover rate’ of the event. Two limitations of this system is that very large exhibitions, e.g. in a Expo or Suntec Convention Hall, cannot be accurately measured. The camera also functions better when placed above entryways as it is able to provide more accurate data on the number of people entering or exiting. However, this can be circumvented by using an infrared camera, which would likely provide more precise measurements.

The last module scoops data from social media, and uses the TheySay sentiment analysis API to determine the user response to the exhibit. Using the keywords of the Exhibition name and Location, this module gauges the overall positive/negative sentiment of the event/exhibition, and displays the top ten most positive and most negative comments found in the sample, providing the organizer with constructive feedback on which parts of the event was done well, and which parts need improvement. Only Twitter is used at the current time, but it can eventually be expanded to include comments from Facebook and Instagram, in order to provide a more comprehensive look at the social media feedback.

(Shortened)

Horus Event Tracker is an application that allows administrators and space owners to

monitor the crowd engagement of ad-hoc events and exhibitions. The application consists of

three modules: The sensor-enabled Arduino connected to a Raspberry Pi, a Raspberry Pi

and a Pi Camera, as well as social media analytics.

Using the sensors connected to the Arduino, a specific exhibit or small area can be

monitored. The sound level of the area, the amount of motion detected, as well as the

number of bluetooth devices in the area is used to measure it’s ‘Buzz’ (how

much activity is in the area). The button module is used for users to rate the exhibition or part of the event, and the average dwell time of visitors is measured using the light sensor, which reflects the public’s interest level.

The Raspberry Pi camera uses the openFrameworks library and Footfall motion tracking

application to count the number of people in a certain area and monitors human traffic.

The sentiment analysis module scoops data from social media, and uses the TheySay sentiment analysis API to determine the user response to the exhibit. Using the keywords of the Exhibition name and Location, this module gauges the overall positive/negative sentiment of the

event/exhibition, and displays the top ten most positive and most negative comments found

in the sample, providing the organizer with constructive feedback.

Technical Documentation:

**Arduino Code**

const int thresholdvalue=15;

float Rsensor;

int pirsensor = 2;

int lightsensor = 1;

const int soundsensor = 0;

int adc\_key\_val[5] ={20, 50, 80, 200, 600};

int NUM\_KEYS = 5;

int adc\_key\_in;

int key=-1;

int oldkey=-1;

int counter = 0;

int cooldown = 0;

void setup()

{

pinMode(pirsensor,INPUT);

Serial.begin(9600);

}

void loop()

{

if (counter == 10000){

int pir = digitalRead(pirsensor);

if(pir == 1)Serial.println("M1");

else if(pir == 0)Serial.println("M0");

int light = analogRead(lightsensor);

Rsensor=(float)(1023-light)\*10/light;

if(Rsensor>thresholdvalue){

Serial.println("L0");

}

else {

Serial.println("L1");

cooldown = 0;

}

counter = 0;

int sound = analogRead(soundsensor);

double db = 20 \* log10 (sound);

Serial.println("S"+String(db));

}

adc\_key\_in = analogRead(A3);

key = get\_key(adc\_key\_in);

if (key != oldkey)

{

delay(50);

adc\_key\_in = analogRead(A3);

key = get\_key(adc\_key\_in);

if (key != oldkey)

{

oldkey = key;

if (key >=0 && cooldown==0){

cooldown = 1;

switch(key)

{

case 0:Serial.println("B1");

break;

case 1:Serial.println("B2");

break;

case 2:Serial.println("B3");

break;

case 3:Serial.println("B4");

break;

case 4:Serial.println("B5");

break;

}

}

}

}

counter++;

}

int get\_key(unsigned int input)

{

int k;

for (k = 0; k < NUM\_KEYS; k++)

{

if (input < adc\_key\_val[k])

{

return k;

}

}

if (k >= NUM\_KEYS)k = -1;

return k;

}

**Raspberry Pi code:**

import serial

import time

import requests

import bluetooth

from math import log10

scheduled = time.time() + 60

covered = False

motion = []

dwell = []

sound = []

light = []

ser = serial.Serial('/dev/ttyACM0',9600)

while True:

value = ser.readline()[0:-2]

print(value)

if value[0] == "B":

r = requests.post('http://horus-et.appspot.com/rating', data = {'score': value[1], 'piid' : "1"})

print(r)

elif value[0] == "L":

light.append(value[1])

if (value[1] == "0" and not covered):

covered = True

start\_time = time.time()

if (value[1] == "1" and covered):

covered = False

dwell.append(time.time() - start\_time)

print("dwelltime", time.time() - start\_time)

elif value[0] == "M":

motion.append(value[1])

elif value[0] == "S":

sound.append(value[1:])

if len(dwell) > 10:

average = sum(dwell)/len(dwell)

r = requests.post('http://horus-et.appspot.com/dwelltime', data = {'time': str(average), 'piid' : "1"})

dwell = []

print(r)

if (len(motion) > 100 or len(sound)>100):

nearby\_devices = bluetooth.discover\_devices(lookup\_names=True)

log\_devices = log10(len(nearby\_devices) + 1)

print(log\_devices)

avg\_motion = (motion.count("1")/len(motion)) \* 40

print(motion)

avg\_sound = (sum([float(x) for x in sound])/len(sound)) - 20

if log\_devices > 10:

log\_devices = 10

if avg\_sound > 40:

avg\_sound = 40

avg\_light = (light.count("1")/len(light)) \* 10

buzz = avg\_sound + avg\_light + avg\_motion + log\_devices

print(buzz)

r = requests.post('http://horus-et.appspot.com/buzz', data = {'buzz': str(buzz), 'piid' : "1"})

print(r)

light = []

motion = []

sound = []