Coursework 1: Sensing

1. Introduction

People living in basement flats often fall victim to theft, because street-facing windows offer an easy way of entry for thieves. The first part of this report is concerned with presenting the data collection setup for a smart window device, which provides theft protection and weather-informed data about activities on the street. The objective of this setup was collecting quantitative data about people's behaviour in front of the street-facing window, and to store it in cloud storage along with weather data from an API. Further efforts prior to advanced data analysis were to preprocess data in a way to determine when somebody is in front of the window, and to validate the reliability of the data sources.

2. Data sources and sensing set-up

Two distinct data sources, a thermal camera sensor and a weather API, yielded data for this project.

AMG8833 Thermal Camera

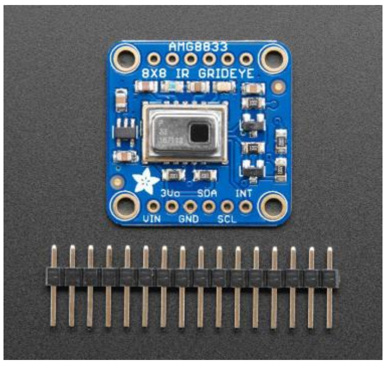


Figure 1: AMG8833 Sensor

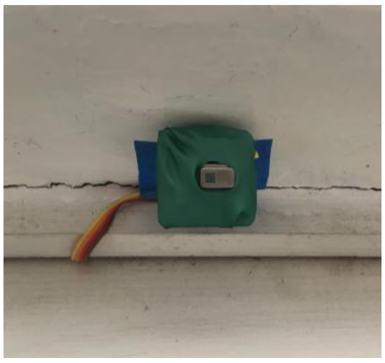


Figure 2: Sensor cover and mounting position

The AMG8833, shown in *Figure 1,* is an 8 by 8 pixel thermal camera that can detect people from up to seven metres away and reads temperatures in the range of 0 to 80 degrees Centigrade [1]. For the purposes of theft protection, this sensor provided the bulk of the data. There are no privacy issues, since the camera is too low-resolution to uniquely identify any person. Moreover, heat information is collected rather than light, which cannot be assigned to anyone in particular.

Weather API

In addition to ambient temperature data, which can also be inferred from the AMG8833's readings, the weather API provided weather classifiers [2]. The service, OpenWeatherMap, provided weather states such as 'Clouds' or 'Clear' for example. This data would be used to provide weather-informed stats about the user's vicinity, for the bulk of the time when no threat is present.

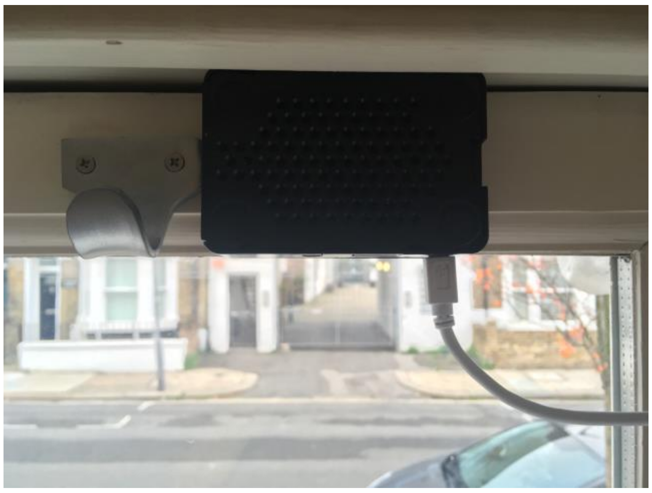


Figure 3: Raspberry Pi mounting position

Data was collected using a Raspberry Pi mounted inside the room (*Figure 3)*, connected to the street-facing AMG8833 mounted on the outside of the window (*Figure 2)*. Unlike a normal camera, thermal cameras cannot 'see' through windows, meaning the sensor needed to be placed outside. It was wrapped in rubber to protect it from the elements, resulting in an approximate field of vision depicted in *Figure 4.* The whole setup was independent of any other home devices and could therefore uninterruptedly collect data.

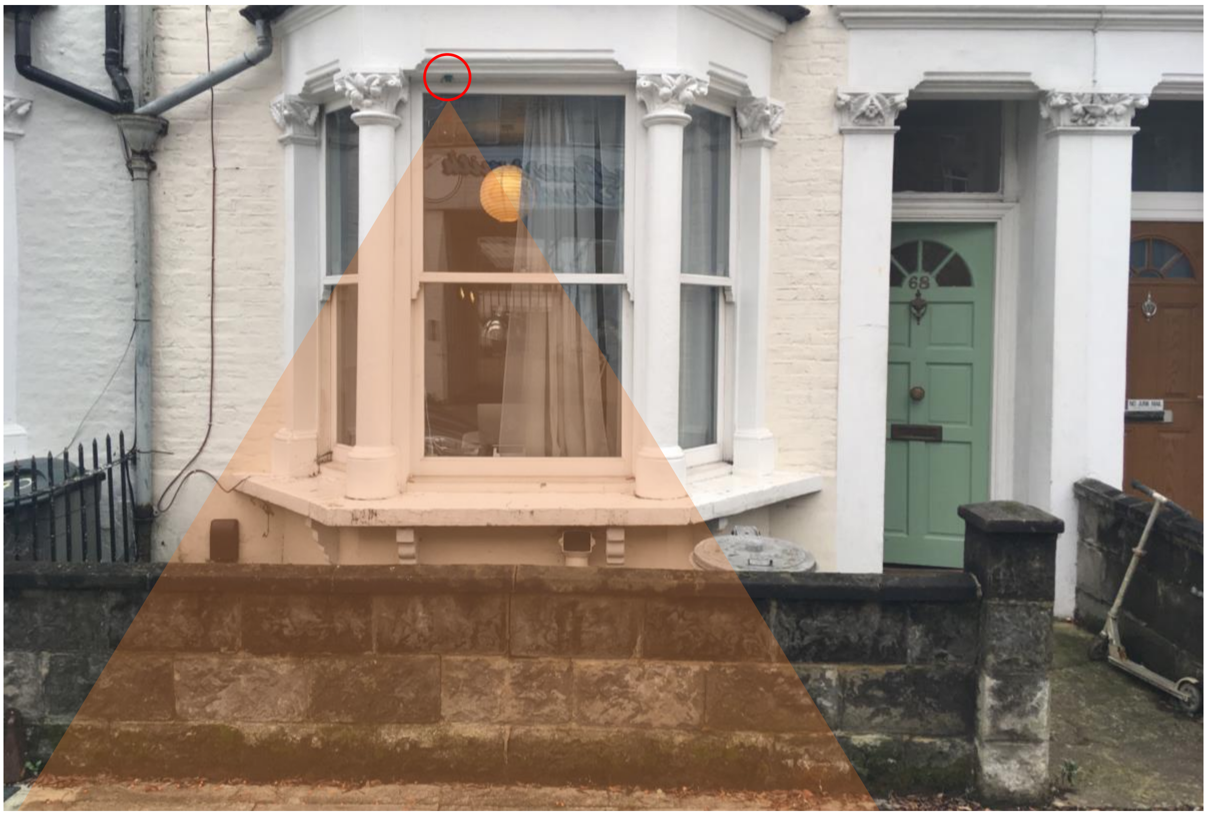


Figure 4: Outside view with approximate sensor field of vision shaded in orange. The sensor is circled in red.

3. Data collection and storage process

An important consideration with regards to data collection was to set the right sampling rate. The weather API updates the London weather twice per hour. It was clear that this data should be sampled at the highest frequency possible, since data sizes are not high and down sampling always remains an option. For the sensor data there was a trade-off between frame rate and file size. It was decided to use a sample rate of 1 Hz, as this ensured that whenever somebody walked by, he or she would be captured on one frame at least once. This decision was based on the observation that it takes around 2 seconds for someone to walk through the field of vision, and the sampling frequency should be chosen as twice of that (refer to *Table 1).*

Table 1: Sampling considerations for API and sensor data

Sensing Quantity	*Approximate Nyquist Frequency*	*Sampling Period*	*Sampling Frequency*
Weather	?	30 min	0.00056 Hz
Sensor Data	0.5 Hz	1 s	1 Hz

Data storage was another design consideration. For the purposes of this project, the Raspberry Pi was a great tool to collect data, but not to analyse it or prototype interfaces. Equally, data loss needed to be prevented in case the system broke down. That is why a cloud-based data storage solution was used. Both weather and sensor data were uploaded to Dropbox hourly. This was achieved using DropboxUploader [3], a bash script for RPi that handles uploading a specified folder to the cloud storage.

In terms of data security, Dropbox fulfilled the design requirements as all communication is encrypted during backing up. Similarly, the RPi was password-protected and all connection to it was performed using an encrypted VNC (Virtual Network Computing) or ssh communication.

4. Basic characteristics of the end-to-end systems set up and data

Data was collected over the course of a month at a fixed sampling rate of 1 Hz and 0.00056 Hz for sensor and weather data, respectively. The data was saved in JSON format for every hour using the following nomenclature.

data_YYMMDDHH.txt [sensor] and wdata_YYMMDDHH.txt [API]

Every JSON string contained a datetime timestamp and then either the sensor raw data (8x8 temperature array) or the API data (temperature + weather classifier). Two example JSON strings are presented below, for the sensor firstly and for the API data secondly.

["2018-12-09 19:00:13", [[8.0, 7.75, 6.25, 6.75, 6.0, 8.0, 8.0, 9.0], [7.75, 8.0, 7.0, 6.5, 6.25, 7.5, 7.25, 8.75], [8.5, 8.25, 7.0, 7.25, 6.75, 7.0, 8.75, 8.75], [8.5, 8.25, 7.0, 6.75, 7.25, 7.5, 7.75, 9.0], [8.0, 8.0, 7.25, 6.5, 7.75, 7.0, 8.0, 8.75], [8.0, 7.75, 8.25, 7.75, 7.75, 7.25, 8.0, 9.25], [7.75, 7.75, 7.75, 7.0, 7.5, 8.0, 7.75, 8.5], [8.5, 7.25, 7.5, 7.5, 7.0, 7.5, 8.5, 9.5]]]

["2018-12-05 03:20:00", {"temp_max": 7.0, "temp_kf": null, "temp_min": 5.0, "temp": 5.89}, "Rain", "light rain"]

Data Preprocessing: Concatenation

Since the data was saved as a file every hour to minimise data loss in case of a system breakdown, it needed to be concatenated prior to analysis. For the purposes of this report, the date range was set to 30.11.18 00:00 to 30.12.18 00:00, amounting to a month's worth of data overall. The DataConcat() class was written to handle this task for both sensor and weather data. The resulting files 'Sdata.txt' and 'Wdata.txt' form the basis for subsequent data analysis. Their uncompressed file sizes were 966 Mb and 193 Kb, respectively, revealing the bulk of data collected by the IR camera.

Data Preprocessing: When has a person passed?

The first data analysis challenge was to determine when a person had passed from the sensor data set. To achieve this, the distribution of peaks within the time-variant standard deviations of the thermal camera temperature array data were investigated. When no heat source was in the field of vision of the IR camera, all objects had a similar temperature, as they adjusted to ambient temperature. As soon as a heat source entered the field of vision, the standard deviation of heat values jumped, because there were now both 'cold' and 'warm' pixels. The custom peak finder was implemented based on the following rule.

"When the standard deviation of temperature values in sample n is more than twice the standard deviation of temperature values in sample n-1 and sample n+1, and the maximum temperature is higher than 14 degrees, a person must have passed."

The algorithm marked all samples that satisfied the rule, from which the following DataFrame objects could be created. Attention has been taken to ensure that people that stay in the field of vision for several samples are also considered.

Dataframe Construction

The data was imported into three Pandas DataFrame objects. Two data frames contained sensor data, one for when somebody has passed (df_pp) and for when not (df_temp), A final pandas data frame contained API data (df_api). Important metrics are summarised in *Table 2*.

Table 2: Summary of pandas DataFrame objects used for subsequent data analysis. 'PP' refers to Person Passed. df_pp contains all the instances where somebody passed ('PP' == 1.0) and df_temp contains all the instances where nobody passed ('PP' == 0.0).

DataFrame object	*df_pp*	*df_temp*	*df_api*
Entries (Raw)	670	2,303,860	1,030
Sampling Period	-	1 second	30 minutes
Datetime Start	2018-11-30 00:39:38	2018-11-30 00:00:00	2018-11-30 00:20:00
Datetime End	2018-12-29 23:34:18	2018-12-30 00:00:00	2018-12-30 00:50:00
Resample Frequency	None	Hourly	Hourly
Entries (Resampled)	670	721	721
Columns	'Sensor Data' 'Std. Dev. Temp.' 'PP' == 1.0	'Mean Temp.' 'PP' == 0.0	'Mean Temp.' 'Weather Classifier'

Data Validation

The first analytic step before gaining insights from the data was to validate the sensor and API data. *Figure 5* depicts sensor (df_temp) and API (df_api) mean temperature signals over the whole time range. The sensor mean temperature dataset is interpolated at some points, namely when somebody had passed (which happened 670 times), since these samples do not lend themselves to determine ambient temperature. The correlation between the two datasets was very strong at r = 0.949. This validates both data sources and makes following analysis much more reliable.

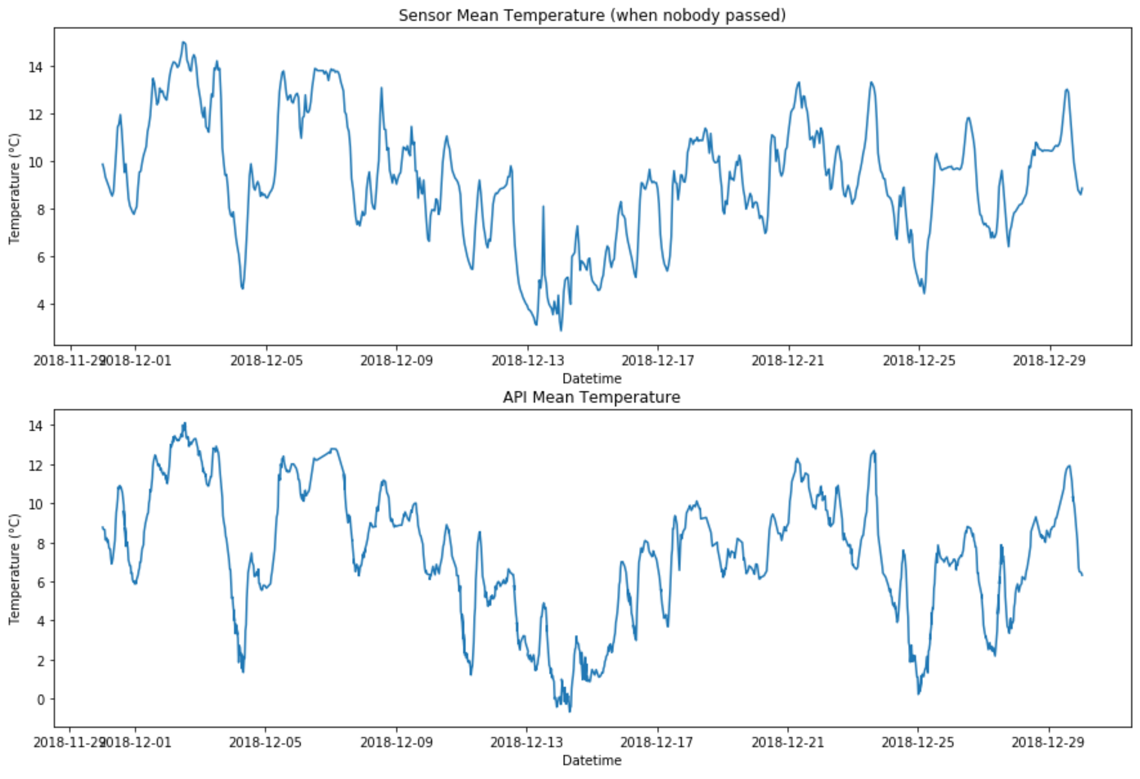


Figure 5: Data cross-correlation between sensor (top) and API (bottom) data sources over the whole date range. The correlation coefficient was r = 0.949, indicating a strong positive correlation between the two.

Coursework 2: Internet of Things

1. Data visualisation and interaction platform

The second half of this report discusses a smart window guard platform for data collected by the sensing setup discussed in the previous sections. It consists of a data dashboard with various plots created from data collected from the end of November to the end of December 2018. It would be locally hosted in the resident's WiFi and could be accessed from the outside through a web service. For prototyping purposes, this dashboard has been made publicly accessible on https://pa17.github.io/mac_siot/. In addition to the dashboard, an advanced adaptive tracker is demonstrated. It can send autonomous and triggered notifications to the user with images of a current threat via the instant messaging platform *Telegram*.

This second coursework is roughly split into two developments. Firstly, the analysis of regularly sampled data over the time range of a month, the data collection for which was presented in the former coursework. Data analysis, inferences and insights are all developed from the data set outlined in the previous sections. The adaptive tracker, secondly, was a more recent development and is presented as a prototype for advanced interactions. It does not yet integrate with the rest of the data analytics presented here, but acts as a proof-of-concept. As with the other developments, it is demonstrated in the video (https://youtu.be/ORkOIJEHw0g). The code is available as a jupyter notebook available in the mac_siot repository (/data_analysis).

2. Data analytics, inferences and insights

Although it was already established that 670 people had passed the window over the time range of a month, the data lent itself to performing a much more in-depth analysis of activity outside the window. This analysis is presented as follows.

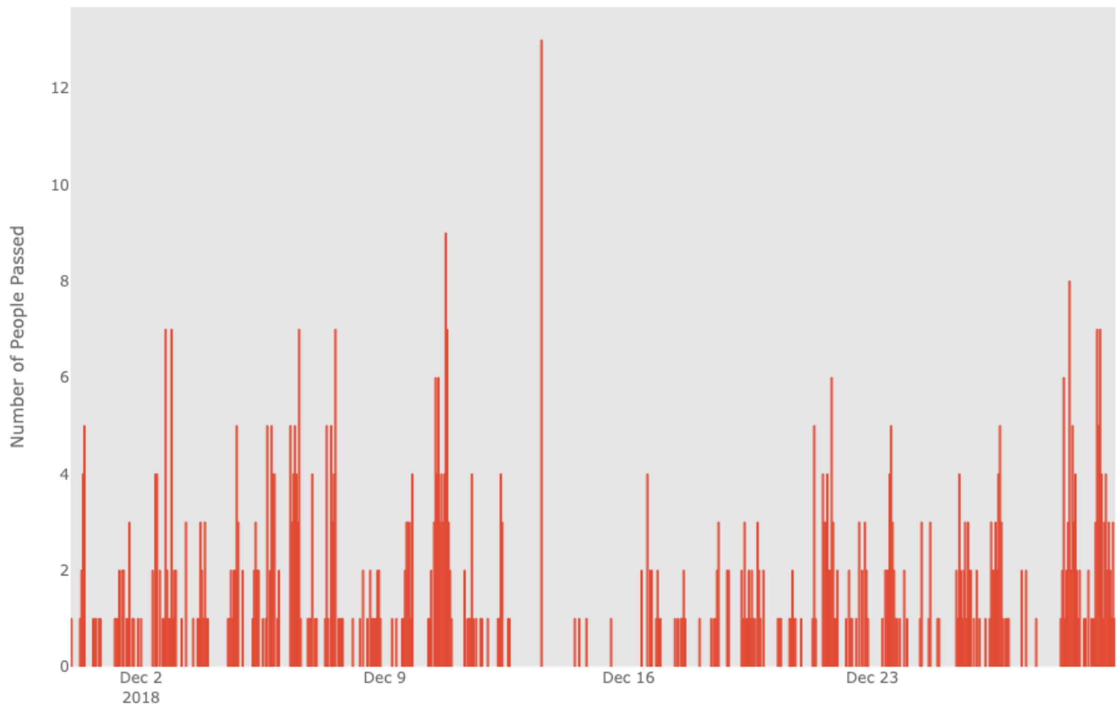


Figure 6: Frequency distribution of people passing the street from 30.11.2018 to 30.12.2018. An outlier range can be observed around the 13th of December.

Person Passed: Histograms

To illustrate the people that have passed the window over the month of interest, the occurrences were grouped into time bins of an hour's length each. *Figure 6* depicts the resulting frequency distribution, showing the histogram of people passing the window over the date range. An interactive version is available in the dashboard, allowing much closer inspection of specific time ranges through zooming.

Interestingly, around the 13th of December no people were recorded to have passed and occurrences picked up only on the 16th again. Of course this cannot be true, since people always pass by the window to some extent. Another look at *Figure 5* might answer why no people were detected during these days. Temperatures dropped below zero during that period and the sensor has a minimal measurement temperature of 0 degrees [1]. Therefore, it is highly likely that sensor malperformance (or mishandling of negative temperature values) has led to this erroneous behaviour during the period.

Autocorrelation

The periodicity of people's street activity behaviour was looked at based on the segmented data illustrated in *Figure 6.* Autocorrelation of that data is shown in *Figure 7*. It becomes apparent that the signal has a significant autocorrelation at 24 hours or 72 hours. This indicates that the periodicity of people passing by on the street follows a daily cycle, which matches intuition and personal experience.

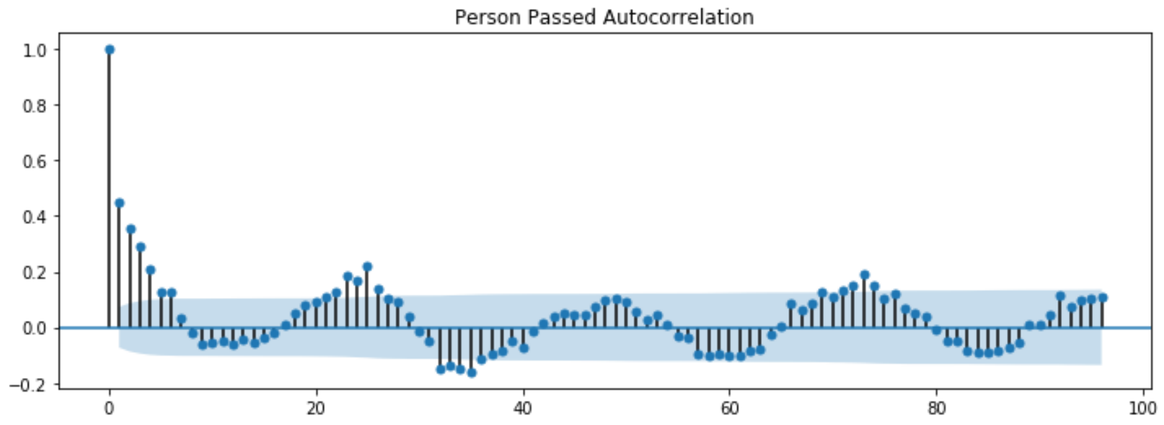


Figure 7: Autocorrelation signal of the frequency histogram data from Figure 6. Significant correlations occur at lags of n = 24 and n = 72.

Thermal Images

In order to move closer to implementing theft protection, the raw data of when somebody was in front of the window was visualised. The DataFrame containing the sensor raw data (df_pp) was sorted so that the samples with the highest standard deviations and temperatures were at the top, thereby ordering the dataset in terms of the most obvious threats.

The 8x8 raw temperature array was then turned into a 32x32 thermal image using cubic interpolation. This approach is credited to Adafruit and achieved using the scipy.interpolate library [1] [4]. *Figure 8* shows the 25 observations that showed the highest threat potential plotted as an array of heatmaps.

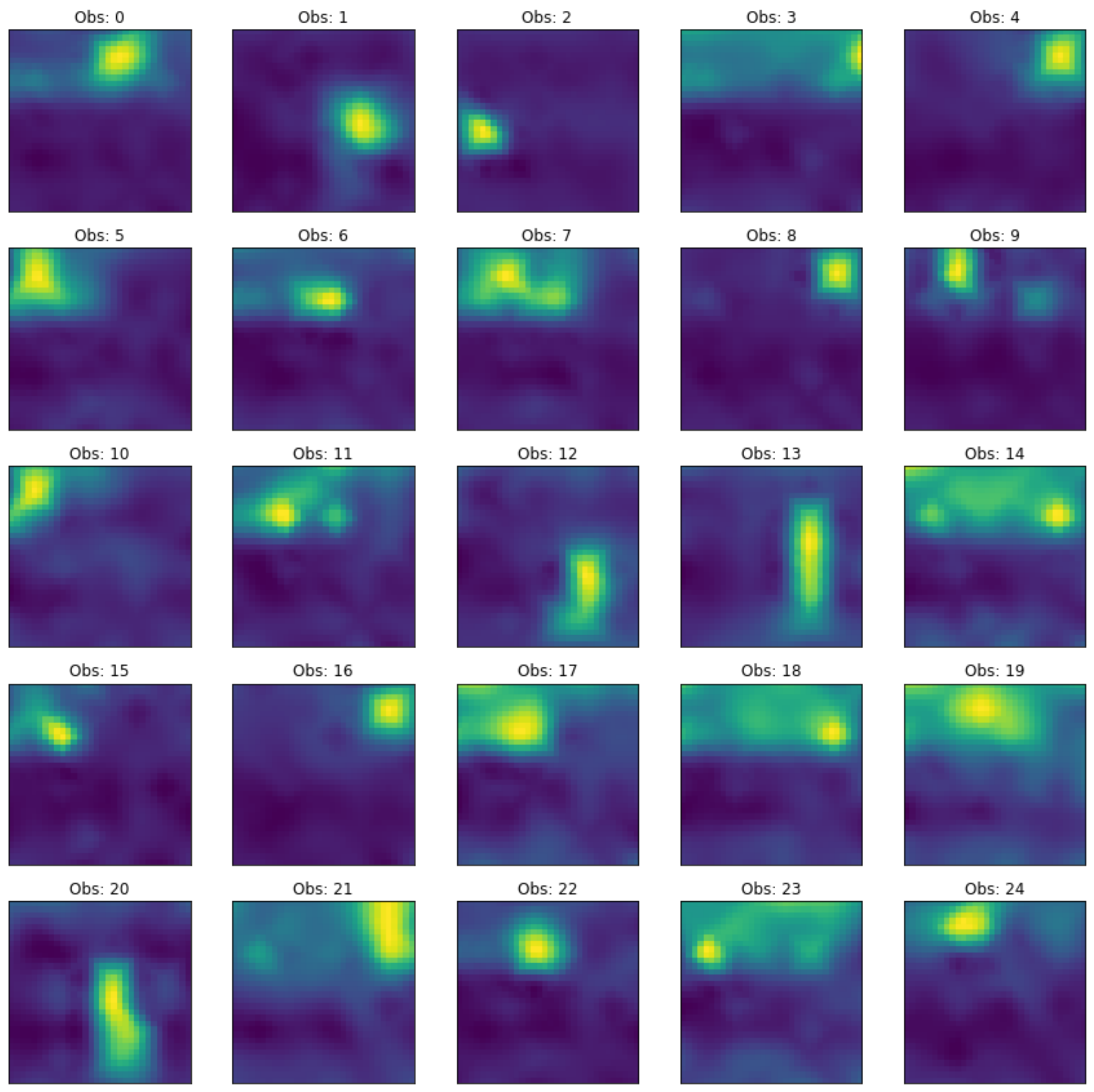


Figure 8: The 25 most threatening observations based temperature and standard deviation depicted as heatmaps

It is interesting to observe that the temperature clusters either happen in the top half of the image or longitudinally in the bottom half (e.g. observation 12, 13 and 20). This could be attributed to the fact that the top half of the thermal image is likely to be the street, whereas the bottom half is the porch outside the window. Observations 12, 13 and 20 could potentially be the garbage men collecting the trash from the bins or myself looking at the sensor. The potential street-type of observation (e.g. 4 or 9) are more local and are likely due to people being more distant. Larger temperature clusters as in observation 14 or 18 could be cars that are cooling down when parked. Some of these plots are available as interactive plotly graphs on the data dashboard, allowing for zooming and pixel-wise temperature information.

Weather Data

Finally, I investigated some of the weather data and created histograms for the different types of weather passer-bys walked through. *Figure 9* confirms the gloomy weather conditions London's residents experience during the colder months. This data is also presented on the dashboard for the routine case where nobody is trying to break into the house.

Dashboard

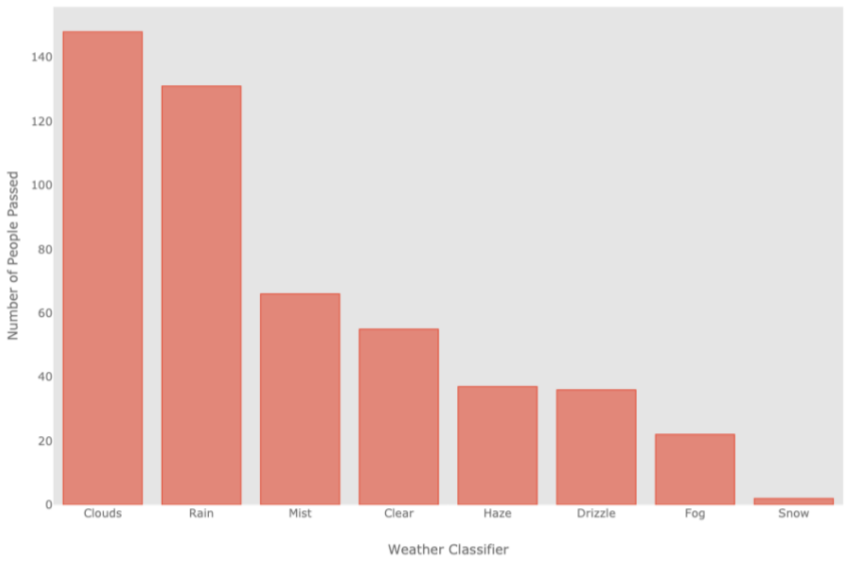
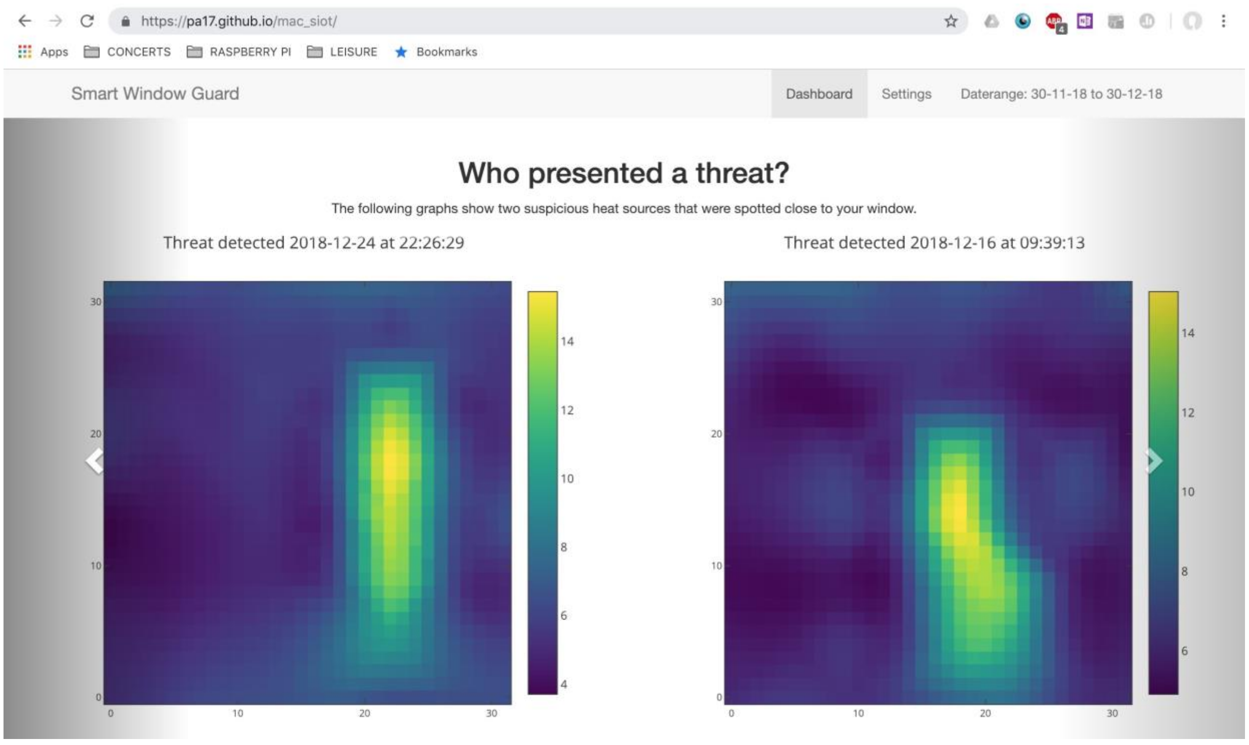


Figure 9: Frequency of people passing through different weather conditions.

Finally, the results from data analysis were all collated and presented on a web-based data dashboard. It displays graphs for the chosen date range and allows for data interaction using plotly's JavaScript-based iplots. The data can be updated by periodically rerunning a script that updates the plotly graphs. These are then automatically updated in the web interface, since they are streamed as <iframe> html objects. *Figure 10* shows a screenshot of the Bootstrap-based webpage featuring a carousel to swipe through the different plots. Please note that the website is not currently optimised for mobile devices. It is hosted as a GitHub page.



Who presented a threat?

The following graphs show two suspicious heat sources that were spotted close to your window.

Threat detected 2018-12-24 at 22:26:29

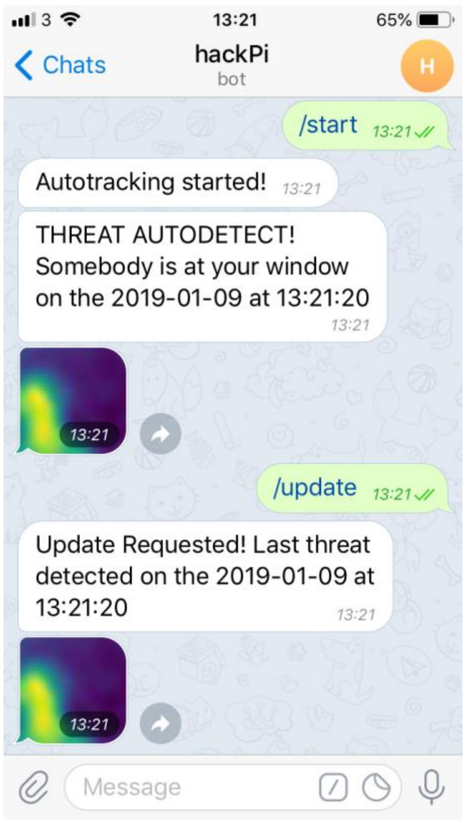
Threat detected 2018-12-16 at 09:39:13

Figure 10: Online data dashboard available on pa17.github.io/mac_siot. It features a carousel where a user can navigate through a number of data visualisations describing the street activity and threats.

Use Case Prototyping: Telegram

Further to data analysis and visualisation on the dashboard, use-case oriented interaction has also been prototyped. An adaptive tracker and instant messaging bot have been implemented. It is able to perform rudimentary data analysis directly on the Raspberry Pi and alert the user about threats via Telegram (a messaging platform similar to WhatsApp).

Figure 11 shows a screenshot demonstrating the features of the chatbot implementation. The user communicates with the bot via commands such as '/start' or '/update' to trigger different responses. Enabling autotracking configures the system to send autonomous alerts when a threat is detected. The presentation video shows a real-life use case using this feature. The bot does not only send text messages, but also 32x32 thermal images of the type shown in *Figure 8.* This allows the user to determine the gravity of the situation and potentially take other steps. The Telegram bot was implemented using the telepot library [5].



/start 13:21

Autotracking started! 13:21

THREAT AUTODETECT!
Somebody is at your window on the 2019-01-09 at 13:21:20 13:21

13:21

/update 13:21

Update Requested! Last threat detected on the 2019-01-09 at 13:21:20 13:21

13:21

Figure 11: Telegram Bot in autonomous (top) and trigger (bottom) mode.

Discussions on the important aspects of the project

This project proposed an end-to-end implementation of a threat detection system including sensing set-up, data collection and storage, data analysis, visualisation and advanced interactions. The prime achievement was the prototyping of the user-centred system with this breadth of associated functions. The software infrastructure is a scalable implementation that allows for further features to be added at any stage without having to redesign the whole system. That being said, the system focus meant compromising detail in the subsystems, as for instance in the data collection. The thermal camera proved to be very suitable and performed extremely well throughout operation. The following section discusses how this could be expanded.

Avenues for future work and potential impact

The threat detection system proposed in this report is a proof-of-concept and lays the basis for a subsequent detailed design of a smart window guard product. Future efforts should focus on implementing adaptive tracking strategies that are able to adjust sampling rate based on the gravity of the situation. This means during potential attacks, the frame rate should be increased to produce animations, and be decreased when nothing is happening. Considering the thermal camera is fixed, it would also be interesting to overlay the thermal image with a static image of the street so that the heat sources could be visualised as thermal 'ghosts'. This would exploit the sensor's advantage of protecting privacy, whilst giving the user more context of the image.

While a Raspberry Pi is very suitable for prototyping, the price point and size are too high for mass production. In order to advance this proof-of-concept to a viable product, electronics would need to be miniaturised, and be controlled with a WiFi-enabled embedded system. The sensing system would collect data and perform simple decision-making on adjusting sampling frequencies. The bulk of data processing would be performed on a server, and would then be presented to the user through a web interface and chatbot. Finally, adding audio warnings or integrating with home devices could discourage burglars from breaking in should be explored.

In conclusion, this report has proofed the suitability of thermal cameras for threat detection. This concept could potentially lead to a low-cost product that effectively alerts and protects residents in case of break-ins. Future work should focus on miniaturisation, audio warnings, connecting with other smart devices and adding advanced tracking features.

References

[1] Miller, D. (2017). Adafruit AMG8833 8x8 Thermal Camera Sensor. Retrieved from https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor/overview

[2] (2012 – 2019). OpenWeatherMap. Retrieved from https://openweathermap.org/

[3] Fabrizi, A. (2016). andreafabrizi/Dropbox-Uploader. Retrieved from https://github.com/andreafabrizi/Dropbox-Uploader

[4] Jones E, Oliphant E, Peterson P, *et al. (2001-).* SciPy: Open Source Scientific Tools for Python, http://www.scipy.org/

[5] Oala, N. (2018). nickoala/telepot. Retrieved from https://github.com/nickoala/telepot