Smart Anti-Theft Homes

An IoT-based approach to curb home intrusions

Arnav Prasad, Sharad Shukla, Bernd Steinkopf

GitHub: https://github.com/Sharadd15/Anti_Theft_System

Home intrusions pose a severe financial and physical risk for its victims. In our project, we seek to leverage recent advances in the smart home industry and develop an Internet of Things based approach for intrusion detection. By combining an array of different microcontrollers and sensors with our custom detection logic, we are able to accurately detect break-ins. We also provide a convenient interface to notify and interact with the affected residents.

Introduction

Home intrusion is a major concern for the residents of any place, regardless of the safety standards of the locality because they can happen anytime, with anyone. A stunning 88% of all robberies happen in residential areas, out of which ~77% are property crimes in which owners incur an average loss of approximately €2500. It is estimated with a good certainty that at this alarming rate of intrusions, 75% of total homes will be broken into by 2040.

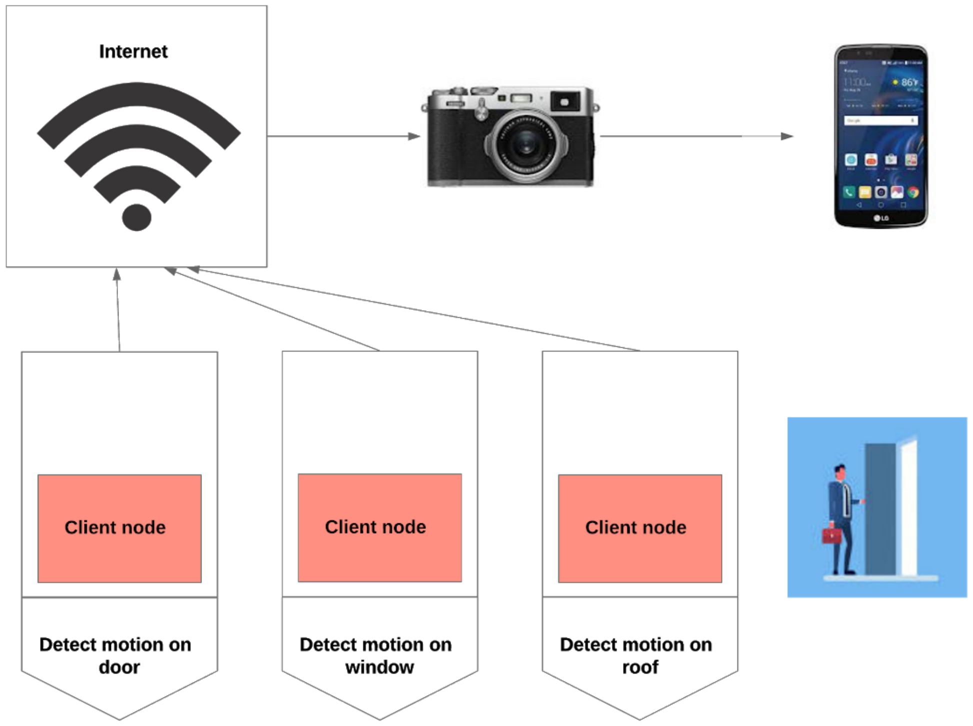
Our project aims at curbing such kinds of intrusions with the capabilities of Internet of Things (IoT) technology in our arsenal. We intend to reduce intrusions, and hence reduce the number of burglaries that happen on a daily basis. In this report, we shall analyse the root causes of intrusion, engineer methods to solve the issues and ultimately develop a product-standard implementation of the same. We shall introduce a few features which will add to the functionality of the project in virtue of the user (i.e., the owner of the home). Our Smart Anti-Theft system will alert the user of this IoT-based Android application whenever there is any intrusion that occurs while the owner is away from home (or even when the owner is inside the home).

Solution Design Phase I - Statistical Analysis

We initially aimed at understanding the common points of breaking in. It turned out that a majority 34% of the times intrusions happened from the front door, 23% from the windows and 22% from the back door. We figured that we must optimize the common cases (i.e. front doors, back doors and windows) first before escalating the project to the next level.

Solution Design Phase II - High Level Architecture

Our goal is essentially to reduce chances of thefts. In order to do that, we must analyse the controllable objects in our system. Our system comprises of a thief, the entrance (e.g. a door) and the passage inside the entrance. We know we can't control the thief, so that leaves the entry point and the passage. We identified three ways to reduce the theft chance- 1. Don't let the person break in, 2. Warn the person before he steals something, and 3. Inform the owner about the intrusion and allow him to take action. Informing the owner about the intrusion is the most rational thing to do since the owner may choose to take whatever action he/she wants against the intruder. We figured out that there's no way more convenient than informing the owner via notifications to the owner's phone, so we developed an android app to receive push notifications from the server. Furthermore, it's a guarantee that a person will have to cross the entrance (regardless of how they do it) - so sensing motion at the entrance is a valid option. Also, we wish to represent as much information as we can about the intruder- hence images of the entrance would be helpful. So we plan to check for intrusions, then click a picture with a camera (preferably night vision) and then send it piggybacked to a push notification to the owner.



Solution Design Phase III - Literature Analysis

Participant roles

1. Data sources - These participants are involved in supplying the relevant data in raw format to the system. Data sources in our product are PIR motion sensors, contact sensors (to detect change in state of door), and night vision camera.
2. Data sinks - These participants accept the raw data and then manipulate it as they were programmed to do. It may be merely just for storage purposes or possible for performing complex calculations based on it. Data sinks in our product are Microcontrollers (Raspberry Pi(s) in our case), Android server and Google Firebase. We use Firebase as storage for images.
3. Actuators - These participants act based on the history sequence of the entire (and sometimes a part of) system. Actuators are the operating system (which gives us push notifications), and hands (taking the right action on the phone).

Transfer protocol

We decided on MQTT over HTTP and CoAP because of the use case we are handling. MQTT offers 3 types of Quality of Services (namely, exactly once, at least once and at most once delivery guarantees) and we don't want to miss any intrusion reports due to some transmission faults. Hence we set the MQTT to exactly-once delivery promise mode. Also, MQTT offers an optimized method for subscribing and publishing which is an intensive task in our product. Finally, it is memory efficient and also has a higher throughput.

Software and Operating System

We fuse reactive programming with scheduled programming in Python by using publishing and subscribing events (which are parallel tasks) along with conditional statements & loops (which are serial tasks).

Data representation

We use JSON data representation since it is intuitive and also easy to use and operate upon, especially when we involve Google Firebase into play.

Algorithm

1. *Detection* - The contact sensor keeps a track of whether the door is open or closed and the PIR motion sensor detects if someone moved through the door (in or out). If someone opens the door and moves in or out, a signal will be sent to the camera.

2. *Activation of camera* - Once someone has been detected, the camera will click a picture of the entrance and we may identify the intruder. The picture is sent to Google Firebase.

3. *Data transfer via Google Firebase* - The camera feeds the information (images) to the Firebase which in turn forwards the data to the Android server.

4. *Push notification on Android* - The server actively sends push notification to the owner's phone.

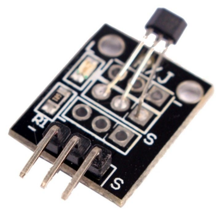
5. *Switch feature* - We don't want the house owner to be detected as an intruder so the owner keeps a switch with him which he could flip whenever he wants to exit or enter the home. The system will be disabled for 10 seconds, giving enough time to pass through the door.

Architecture & Implementation

Hardware

1. *Raspberry Pi* - Raspberry Pi is essentially a microcontroller that manages all the sensors and actuators as per our wish. Here, our wish is translated into a Python program. We use 4 Pi(s), 1 as the server and the other 3 as the clients.

2. *PIR sensor* - A passive infrared sensor (or PIR) measures infrared light radiating from objects in its field of view. The field of view is determined by the solid angle. This is the most commonly used sensor for the purpose of detecting motion of objects since each object has different heat signatures and hence every object is essentially unique in terms of the infrared radiated by it.



3. *Contact sensor* - This sensor detects change in magnetic field intensity in its vicinity by the means of electromagnetic induction. Hence we use this addition to the PIR motion

sensor in order to detect if the door state is changing (being opened or closed).

4. *Night vision camera* - This camera is used in order to observe the intruders in the dark.

MQTT

MQTT is a fast and lightweight message broker specifically designed for the Internet of Things. Unlike its competitors such as RabbitMQ, MQTT places special emphasis on low energy consumption and bandwidth usage. New messages are sent to a central server called broker from where they are forwarded to any connected clients. Isolated topic channels can be used to differentiate between messages intended for different audiences and to combat the flood of information often found in larger networks.

Our project uses Eclipse Mosquitto as an open source broker implementation that runs on the central Raspberry Pi node. The client nodes connect to it using Paho MQTT, a state-of-the-art Python library for MQTT client applications. We use one channel per client and all clients are set to deliver messages exactly once. MQTT does not support any security features in its original form. Our implementation bypasses this issue by having all edge nodes operate within an encrypted private network with no connection to the outside world. The network is created and maintained by the central node.

Firebase

Google's Firebase is a collection of different cloud services aimed at mobile developers. The lowest tier is provided free-of-charge and was deemed sufficient for a successful implementation of the proof-of-concept prototype developed by our team. The project itself makes use of the following modules: 1) Cloud Firestore, a NoSQL database; 2) Firebase Storage, a cloud store for large binary files; 3) Firebase Cloud Messaging, a push notification server for mobile apps.

Cloud Firestore is a high-speed NoSQL database and a modernized version of Google's Realtime Database. Entries are stored as heterogeneous documents identified by a single unique key. In our case, the database is used to store the timestamps of intruder alerts sent by the edge nodes along with additional metadata, such as the exact point of entry. Once the alerts have been entered, Firestore supports rapid ordering, filtering and full text search.

Firebase Storage offers a set of storage units called buckets. Each bucket can be used to store a large number of binary files that can later be accessed by users through the Firebase APIs. The project uses a bucket per user to store all relevant security footage captured by our camera setup. These files would normally be too large to be stored in a regular database like Firestore. Firebase Storage's specialized infrastructure, however, makes it possible to deliver the high resolution images to end user applications at acceptable speeds.

Firebase Cloud Messaging provides a convenient way for developers to send notifications to always-on devices with no static address, i.e. cell phones and tablets. By registering an app at an FCM endpoint, a persistent connection from the device to the server is established. For later identification, a unique token is generated for the device and stored on the server. Using the token, a device's outgoing connection can be reused to push an arbitrary message back to the device, making it appear such that the device can be reached under a fixed address.

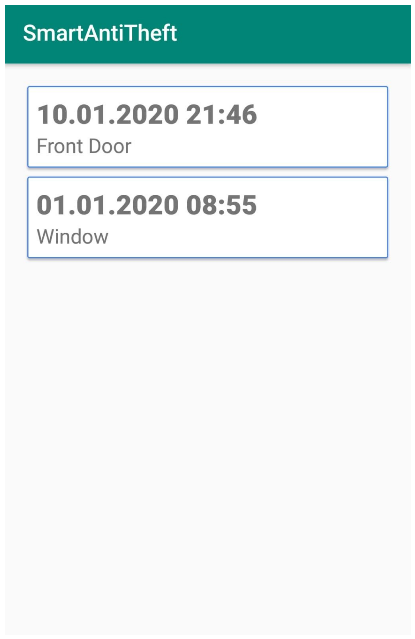
Access to the firebase modules is restricted by means of two different mechanisms. A static cryptographic key is generated through the web interface and stored as a .json file on the Raspberry Pi server. The key is then used in conjunction with the Firebase Admin SDK to establish a connection to the Firebase endpoints from within the server's code. End users operating the companion Android app described in the following section authenticate through a simple username/password scheme. For prototyping purposes, a single test user was created for the project.

Android App

As part of our anti-theft system, users have access to a companion app. Currently only Android devices are supported natively, but the app could potentially be ported to run on other operating systems as well. It was developed using Android Studio and the Kotlin programming language. All application data is pulled from the Firebase modules discussed previously. To connect to the respective APIs, we used the Google Services Plugin for Android and any applicable module support libraries.

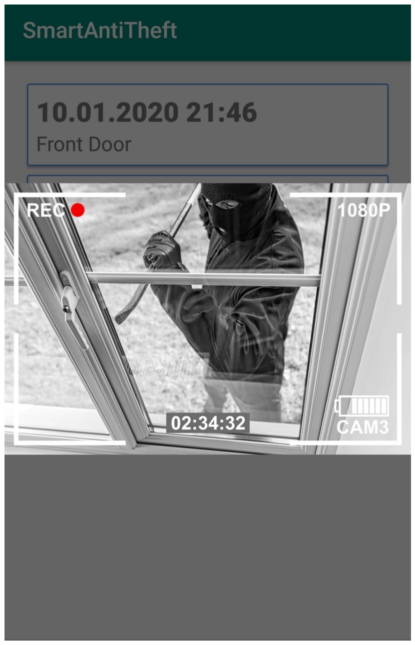
Upon opening the app, users are greeted by an overview of their alert history. All alerts are shown in chronological order, starting with the most recent. The tiles contain information about the exact time, date and location that the intrusion was detected. This data is pulled directly from the Cloud Firestore database. If the users presses on one of the tiles, the respective security footage is downloaded from the Firebase Storage server and displayed on the screen. Based on the image, the user can assess the severity of the situation and decide whether further action needs to be taken. This manual inspection step adds an important human-in-the-loop component to our anti-theft system, that drastically reduces the number of false positives observed in systems that solely rely on automated detection.

To provide users with real-time updates of alerts, the app is connected to the Firebase Cloud Messaging service. When triggered by the R-Pi server, this service instantly sends a new alert to the app, where it is displayed as a push notification, even if the app is running in the background. With a tap on the notification, the app is brought to the foreground and the new alert added at the top of a list, which is updated dynamically on the screen if the app is already in the foreground when receiving the notification.


SmartAntiTheft

10.01.2020 21:46
Front Door

01.01.2020 08:55
Window


SmartAntiTheft

10.01.2020 21:46
Front Door

A single Firebase login token can be shared by multiple app instances. This allows multiple users to see the same list of alerts and receive the same notifications. Such a behavior makes sense, e.g. if multiple people live in the same household. In such a case, new security alerts can be reviewed by any resident running the app, without the need for a master switch. This circumvents many problems found in literature, where anti-theft systems are often designed to only support single-person households.

Evaluation of solution

Assuming that the sensors, camera and the Raspberry Pis are functioning normally, the product works well and effectively for the doors and windows. If the intruder chooses to break the door instead of opening it (which is clever since the contact sensor won't be triggered and hence the Pi installed at the door would consider it as no intrusion), we installed a Pi on the roof in order to detect the person, as we mentioned before - it is guaranteed that the intruder will cross the entrance point regardless of how he/she does it.

As per our current implementation, the roof Pi will be activated only when the owner is not present in the house since otherwise the owner will be detected every single time. So if someone tries to break the door and if the owner is still in the house, then it won't notify the owner. (however, the sound of breaking the door might be sufficient to make sure the owner knows something is up, but we mustn't rely on probability since it's a basic question of security) This can be fixed by giving the owner an option to activate the roof Pi even if he/she is at home, which comes at the cost of inconvenience for the user since he would be detected every time and would decrease his alertness towards an alarm. (analogous to crying "wolf" all the time) Also, the current scenario that deals with is when there is only one person living in the house. This can be stepped a level up by adding biometrics like facial recognition or fingerprint which would make it simple to manage every person entering or exiting the house.

After the complete development of the product, we experimented at least 40 times (formally) with a common room located in some accommodation, where people kept coming and going. We checked if the notification is being sent regardless of factors like the speed of opening and closing the door, moving near the door from inside the room (but not opening the door - which should ideally not trigger the alarm) and it worked correctly 100% of the time. We couldn't experiment with breaking the door but our proof-of-concept is well thought and robust.

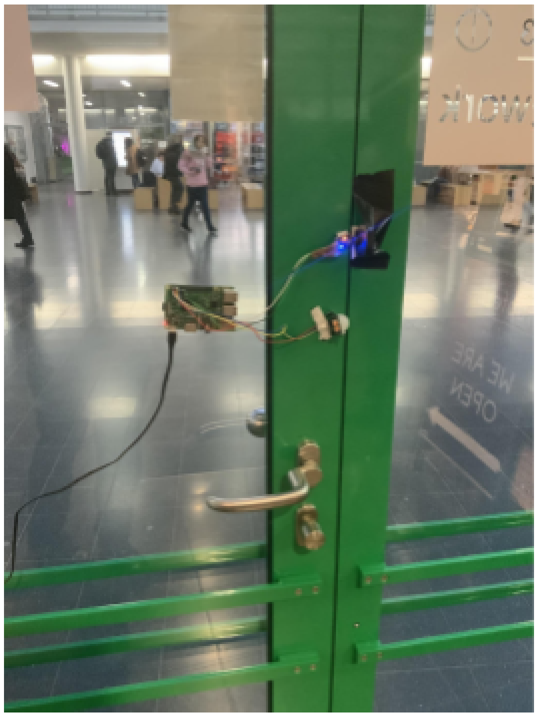
Failure mode and effect analysis

SNo	Problem	Reasoning	Solution	Solved
1	Night Vision cam keeps rebooting	Low power from USB	Connect to main power	Yes
2	Proximity sensor takes time to detect	Algorithmic errors	Make changes in the code	Partly
3	Proximity sensor has low range (and no. 2 persists)	Hardware specs incompatible w/ use case	Replace with contact sensor	Yes
4	Clients misbehaves upon re-subscription	Implementation errors	Make changes in the code	Yes
5	Owner of house detected as the thief	Missing feature	Implement a switch	Yes
6	Intrusion by breaking instead of opening	Missing feature	Implement another node on the “roof”	Yes
7	Android development errors	Library version conflicts	Upgrade to version compatible states	Yes

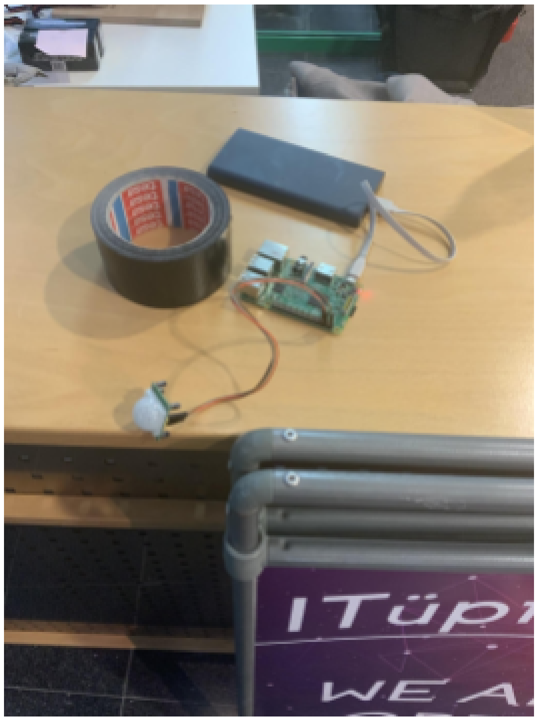
Results

After designing the whole project in virtue of the main aspects of IoT and learning from few of the resourceful lectures of the course and engineering all the problems incurred during the implementation, we finally came up with a robust Smart Anti-Theft system.

The running program of the system (Night vision camera on top left corner of the monitor)



The client detecting door intrusions



"Roof" sensor to detect intrusions if door-breaking is attempted

Conclusion and Outlook

In virtue of the future timeline of the project, the system could be extended in several non-IoT aspects. Using deep learning-based facial recognition, the system might be able to learn about the residents living in the building. This could help improve the alarm system, since biometric data is a much more reliable trigger than infrared sensors when it comes to detection of humans. The downside here is the heightened need for expensive computing power to run the necessary algorithms.

Through advanced machine learning, it should also be possible for the system to learn about the daily routines of its residents. Any deviations could serve as further data points in the automated detection of unwanted intruders. The problem with such a system could be coping with an increased rate of false positives.

With respect to user interaction, the companion app leaves room for multiple extensions. In large systems with a number of entry points, users might be able to set up and manage custom security policies through the app. For instance, a policy that switches on all alarms on certain days of the week comes to mind.

Overall, we have shown that our project fulfills the requirements set by us in the initial team meeting. The system is able to detect intruders, generate security footage, store the data at a safe location and notify the residents. In the spirit of IoT, this was achieved using mostly free or inexpensive hardware and open-source software.