Robust Occupancy Computation Based on WiFi Connectivity Events

1st Rithwik Kerur
University of California, Irvine
rkerur@uci.edu

2nd Yiming Lin
University of California, Irvine
yiminl18@uci.edu

***Abstract*—Occupancy is essential for energy saving and space planning in buildings. Current approaches rely on various sensor hardware, which is intrusive and probably expensive. This work proposed a robust, almost zero-cost occupancy computation approach using WiFi connectivity logs. The ubiquity of WiFi connectivity data enables our occupancy solution to be used passively and in all types of buildings with WiFi infrastructure. Estimating an accurate occupancy count from the WiFi connectivity data is challenging, and the streaming nature of connectivity logs requires the algorithms to be efficient enough to make (near) real-time computations. We proposed a series of approaches to efficiently and effectively compute occupancy at a given location inside a building in the given time range. The proposed occupancy strategy has been implemented, deployed, and is running in more than 30 buildings in 2 different universities, for nearly 3 years.**

***Index Terms*—Database, Data Cleaning**

I. Introduction

Occupancy, i.e., the number of people in a certain location at a given time, is essential for many applications, such as building energy control [3] and space planning [8]. For example, Agarwal, et al. [2] observed that the heating, ventilating, and air-conditioning (HVAC) energy consumption was reduced between 10% to 15% based on the occupancy detection in offices. Leephakpreeda, et al. [3] proposed an occupancy-based lighting control, and showed that the energy consumption of the system can be reduced between 35% to 75%. Additionally occupancy is also a key mitigation strategy for COVID-19 [9] as users could use occupancy-based applications to be aware of the people density inside buildings to reduce the chance of exposure. The strategies used to compute occupancy were explored in the literature, and they rely on various types of hardware to compute occupancy, such as Bluetooth, motion sensors, RFID tags [6], infrared, and video cameras [7]. All of these technologies require new hardware to be installed in the building which is expensive and intrusive. Furthermore, they have algorithmic limitations to deal with dynamic situations such as occlusions and signal attenuation interference.

This paper explores WiFi connectivity events data for a zero-cost, efficient and robust occupancy solution. WiFi connectivity data consists of sporadic connections between devices and nearby WiFi access points (APs), each of which may cover a relatively large area within a building, as shown in Figure 1. Figure 2.a is an example of the WiFi connectivity events table where each event records the information of the *mac address* of the device, the connection *timestamp*, and the

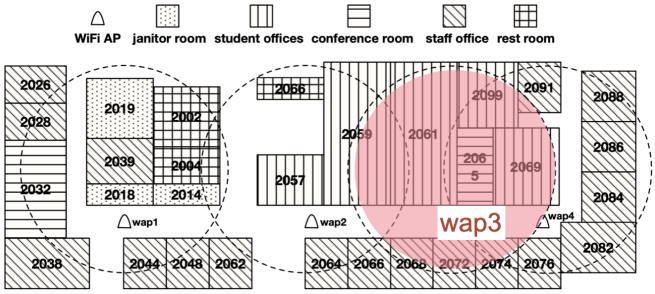

Fig. 1. Access Point Coverage

identification of the connected *WiFi AP*. WiFi connectivity data is attractive and crucial for indoor sensing for several reasons. First, WiFi infrastructures are ubiquitous in many modern buildings, and using connectivity logs does not incur any additional hardware costs. Second, occupancy based on WiFi connectivity events can be performed passively without the need to require users to actively participate in the computation process, such as installing software on their phones. Finally, the ubiquity of WiFi connectivity events allows us to use the occupancy system in all types of the buildings with WiFi equipped, such as supermarkets, offices and libraries.

Our recent work with LOCATER [1] studies the problem of semantic localization based on WiFi connectivity events using data cleaning technologies by assigning the location of a person to a semantic location inside the building, such as floor/region/room. Although LOCATER bridges the gap between WiFi connectivity data and the locations of people inside the building, knowing the location of each individual device is not enough to generate an accurate occupancy as there are several challenges to estimating the occupancy based on WiFi connectivity data. First of all, one person often has multiple devices, such as a phone, laptop, and iPad, so simply counting the number of appearances of devices in the WiFi connectivity data would lead to over-counting for actual occupancy. Moreover, the WiFi connectivity data is generated streamingly, that is, it is generated at a fast speed and large volume, which requires the designed algorithm to remove multiple devices belonging to the same person in near real-time. Second, not all connectivity logs are generated by *people*, and could instead be generated by static devices, such as a printer, or computers in a lab/office. Finally, many

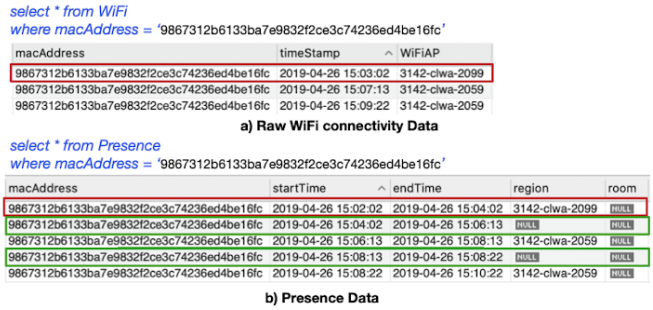

Fig. 2. WiFi Connectivity Table and Presence Table

passer-by devices that connected to some WiFi AP but left the corresponding region immediately should not be counted in the occupancy of the region of interest. Detecting passer-by devices from WiFi connectivity data is also an important factor to ensure an accurate occupancy estimation.

In the remainder of the paper, we start with the architecture of the occupancy system in Section II, followed by the detailed algorithms to resolve the above three challenges in Section III. We discuss the deployment of the occupancy system in Section IV and Section V concludes the paper.

II. Architecture

A. Occupancy Architecture

In this section, we describe the architecture of occupancy computation as well as the data flow, as shown in Figure 3. The system first takes raw WiFi connectivity events as the input. Each event log (i.e., tuple in the table) records the WiFi connectivity event when a device with mac address mac_i connected to a WiFi Access Point (WiFi AP) wap_i at time t_i. As an instance in Figure 2.a, the first tuple in the table represents that a device with a mac address starting with 9867 connected to the WiFi AP 3142-clwa-2099 at time 2019-04-26 15:03:02. To ask for the occupancy in a location in the given time range inside a building, one can issue a point query $Q = (st, et, loc)$, where st, et represent the start and end time stamps, and loc corresponds to the location of interest. [1] The output of the occupancy system will be an occupancy count, i.e., number of people, in location loc in the given time interval (st, et).

Before the occupancy can be accurately computed, we first use the LOCATER [1] to compute the location of each individual device from the raw WiFi connectivity events. The goal of LOCATER is to locate a device in a building to different levels of granularity of semantic location, such as room/region/floor, by cleaning the raw WiFi connectivity events. Given the raw Wifi connectivity events, LOCATER formulates the localization problem into several data cleaning problems. To predict the location of a device at a given time instance, LOCATER will first predict its coarse location (coarse localization), i.e., the region (the area covered by its

[1]Note that a window query of occupancy such as computing occupancy every k minutes in the given location could be easily returned by calling multiple point query.

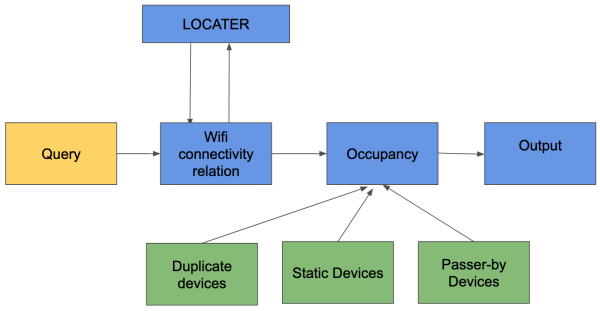

Fig. 3. Occupancy Architecture.

connected WiFi AP), and then disambiguate the rooms inside this region to give a prediction of the room location of this user (fine localization). Our evaluation indicates that LOCATER can answer this query effectively, taking around half a second on average, while achieving nearly 87% accuracy. In general, LOCATER is designed to be flexible enough to select the adequate level of localization needed for the application at hand - paying the additional overhead of fine-grained localization only if needed.

With LOCATER, let us assume that the occupancy system has a Presence relation as shown in Figure 2.b. The table stores information about the identifier of a device (i.e., *mac address*), its location (*region* and *room*), and the interval of time it was in the room (*startTime*, *endTime*). Note that materializing the Presence table fully would be prohibitively expensive (20 milliseconds per missing region location, and 400 milliseconds per missing room location), and we only materialize the missing locations related to the given occupancy query at query time.

With the Presence table on hand, an occupancy query $Q = (st, et, loc)$ in our system is interested in the fine-granularity, i.e., room location, and course-granularity, i.e., region location. ($loc \in \{room_i, region_i\}$) Such a query asking for the occupancy in the time interval (st, et) at location loc on the Presence table can then be easily expressed as a SQL query, such as *select distinct count(macAddress) where startTime = st and endTime = et and region = loc*. [2] However, the answer to such a query is observed to be inaccurate by comparing with the ground-truth occupancy number, since two devices could belong to the same user (*duplicate devices*), devices could be static devices such as printers and thus not a real person (*static devices*) and the connectivity logs could be from a passer-by instead of a person in the queried location (*passer-by device*). All of the above challenges could affect the accuracy of occupancy results, and we detail the solution for resolving each of them in the next section.

III. Occupancy Algorithm

A. Duplicate Device Detection

One person often carries multiple devices, and thus counting each connection for such devices would over-count the occupancy. The task of *duplicate device detection* is to determine

[2]If there does not exist a start time stamp (or end timestamp) in the Presence table that exactly match (st, et), we use the closest timestamps.

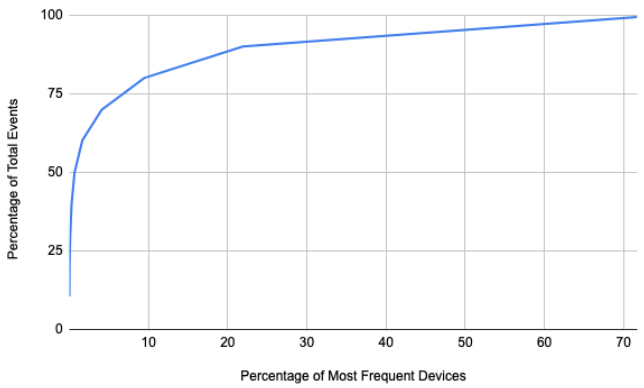

Fig. 4. The Percentage of Connectivity Events from the Most Connected Devices.

a set of devices that belong to the same person and remove the duplicates when counting occupancy. The intuition behind detecting duplicates is that two devices belonging to the same person should share a higher similarity in terms of their *trajectories* when compared with other device pairs that belong to different users. We start with considering the *duplicate device detection* problem in the static presence table, and then we extend it to the streaming presence data as is often the case in practice.

Consider a static WiFi presence table in Figure 2.b and the duplicate detection problem. Let $Sim(i, j, st, et)$ be the similarity of trajectories between devices d_i and d_j during the time interval (st, et), where the trajectory of device d_i is a set of consecutive tuples of device d_i in the time interval (st, et) in the Presence table. $Sim(i, j, st, et) = \frac{CD(i,j,st,et)}{et-st}$, where $CD(i, j, st, et)$ is the total length of the duration time when devices d_i and d_j are in the same region during time range (st, et). [3] We then define a similarity graph based on the devices and similarity metrics as defined above. In particular, let $G(V, E, st, et)$ be the similarity graph, where the set of nodes V are the set of devices appearing in the presence table in time range (st, et), and the edge set E represents the similarities between each pair of devices, i.e., $V = \{d_i | i = 1, 2, ..., n\}, E = \{e_{ij} | d_i \in V, d_j \in V, i \neq j\}$. We assign a weight w_{ij} for each edge e_{ij} as their similarity value $Sim(i, j, st, et)$ representing the similarity between devices d_i and d_j during time range (st, et). Any clustering algorithm can be applied to this graph to do clustering with a constraint that the number of devices in any cluster can be no larger than k. In the resulting clusters, the devices in one cluster naturally represent the devices belonging to the same user since they share the highest similarities with each other. Such a constraint is developed from context knowledge since a single person would not have more than k devices. In our implementation, we set k as 5, which turns out to be effective enough to enable accurate duplicate device detection.

[3]We choose region since the region-level localization by LOCATER is much cheaper than the room-level localization, and thus enables efficient computation of similarity.

Learning the similarities between devices in a relatively long time range (st, et) in the static data is often accurate enough to resolve the duplicate device challenge for devices appearing frequently in such a time interval. However, since the WiFi connectivity events and presence data flow into the system in a streaming way at a fast speed and with a large volume, there is a need to dynamically maintain a more accurate similarity graph for removing devices belonging to the same user effectively. Processing the streaming presence table for graph updating is challenging since the number of devices could be huge, (over a period of one month, there are roughly 20,000 unique devices) and we are not allowed to store all historical data in the disk (over a period of one month, and 30 buildings, 5.04 GB of data is generated, and the volume of data is continuously growing).

To resolve the duplicate devices over the streaming presence table, we first introduce one optimization to reduce the number of nodes in the similarity graph, followed by a strategy to incrementally update the graph periodically to support fast-search of duplicate devices in occupancy computation.

First, we classified the devices into *frequent* devices and *infrequent* devices. Intuitively, frequent devices generate a significantly higher number of connectivity events compared to infrequent devices. Figure 4 describes the percentage of the connectivity events generated by the most frequent devices. As we can see, roughly 20% devices contribute more than 80% connectivity events. In our implementation, we regard the top 20% frequently connected devices as the *frequent* devices.

To incorporate the above observation in the graph, instead of storing the information of all devices, we reduce the set of nodes to only store the similarities for frequent devices. To bootstrap the duplicate detection algorithms, we start with building a static similarity graph $G(V, E, st_0, et_0)$ using the Presence table in the time range (st_0, et_0), then we update such a graph G periodically using the new presence data as follows. In general, consider a graph $G^{'}$ we have already computed and a new graph ΔG built on the new presence data. We update the graph $G^{'}$ in the Algorithm 1. Let $G = (V, E)$ be the updated graph by combining the information from $G^{'}$ and ΔG, where the set of nodes in graph G is the union set of the nodes in $G^{'}$ and ΔG (Ln.1). This step enables the insertion of the new frequent devices discovered in graph ΔG to the current graph $G^{'}$. For each edge $e_{ij} \in E$, we update its weight w_{ij} (i.e., the similarity between devices d_i and d_j) as the summation of w_{ij} and Δw_{ij} (Ln.2-3). The clustering algorithm would also periodically be applied to the latest similarity graph to maintain the latest clusters. To use the maintained similarity graph and clusters to remove duplicate devices, given two devices d_i and d_j, we check if the corresponding nodes are in the same cluster or not. Note that in the graph we maintained, we discard the information of infrequent users to save more space and enable efficient graph updating. When a connectivity event is logged from an infrequent device, i.e., such a device would not be found in the graph, we just count it once. Although the occupancy computation for infrequent devices might be over-counting since

Algorithm 1: Similarity Graph Update Algorithm

Input: $G^{'} = (V^{'}, E^{'}), \Delta G = (\Delta V, \Delta E)$
Initialize graph $G = (V, E)$, $V = V^{'} \cup \Delta V$, $E = E^{'}$
for $\Delta e_{ij} \in \Delta E$ **do**
  $w_{ij} = w_{ij} + \Delta w_{ij}$
return G

we do not remove any duplicates, the impact of occupancy number for the overall occupancy would be small in practice, since the frequent devices generate most of the connectivity events.

Note that when the number of WiFi connectivity events that are used to compute pair-wise similarities between devices is small, e.g, at the beginning stage of the algorithm, the approach could have a chance to throw out false positives, since two devices belonging to different users might have similar trajectories during a relatively short interval in certain locations, e.g. people who use the university shuttle service, or have the same office times. However, when the algorithm uses more historical data to update the similarities, such false positives should quickly go down since it is unlikely that two devices belonging to different users have highly similar trajectories over a relatively long time and in all the locations they visited.

B. Passer-by Devices

When a person carrying WiFi-equipped devices passes by a region r covered by some WiFi AP without staying in the region r, such devices are called *passer-by* devices, and they should not be counted into occupancy of the region r. As an example, students walking by a building without entering it are false positives for occupancy estimation of the building and sub-regions inside that building. Given an occupancy query $Q = (st, et, loc)$ that asks for the occupancy of the location loc during time range (st, et), such passer-by devices are observed to only generate the connectivity events in a fairly short time interval and in limited numbers. Intuitively, to detect passer-by devices for a given occupancy query, we not only need to look back, (i.e., examine the historical connectivity data before the queried time, say $WiFi_{pre}$), but also look forward. (i.e., check the connectivity data after the queried time, say $WiFi_{post}$) We also incorporated the knowledge of *frequent* devices as in Section III-A learned from historical data. If a device is a frequent device and observed to generate connectivity logs in (st, et), we add it to occupancy counting. If such a device is not in the frequent device list, and its connectivity data is only observed to be generated around the queried time, but not in $WiFi_{pre}$ and $WiFi_{post}$, then such a device is probably a passer-by device and should be excluded from the occupancy counting. In our implementation, for the consideration of efficiency, we set the $WiFi_{pre}$ and $WiFi_{post}$ to be half an hour before and after the queried time, and it turns out to be able to effectively capture passer-by devices. If the occupancy query Q is posted now, such as *what is the occupancy in last k minutes*, then $WiFi_{post}$ will not be available at the query time. In this case, we adopt a *post-correction* strategy. If such a device is observed to have no connections in $WiFi_{pre}$, then we count it as a passer-by device at the query time. When $WiFi_{post}$ becomes available as time goes by, and if the connectivity logs of such a device are found in $WiFi_{post}$, we add its count back to the occupancy answer of Q.

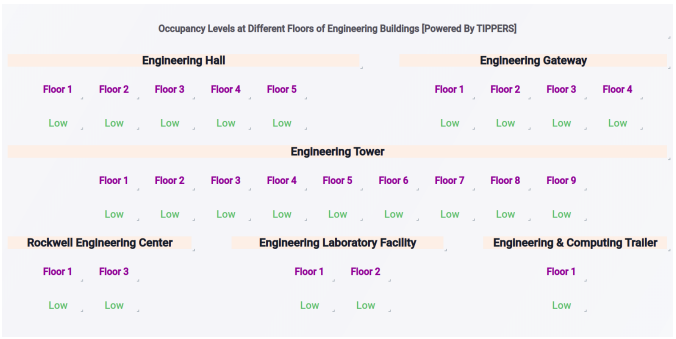

Fig. 5. Occupancy Dashboard.

C. Static devices

The connectivity logs generated by the static devices, such as printers and computers in a computer lab/personal offices, should not be counted as occupancy. Thus we need to detect static devices based on their connectivity patterns. Static devices are often observed to keep connecting to one *fixed* WiFi AP sporadically or periodically for a relatively *long* time, even during periods of low foot traffic, such as from 3:00 am to 6:00 am. We developed two simple heuristic strategies by capturing the above observations to detect static devices as follows. First, we identify a set of devices, each of which connects to only one *unique* WiFi AP during a relatively long time, since connecting to more than one WiFi AP implies the *moving* of such a device. Second, we collect a set of devices that frequently generate connectivity logs at night, such as from 3:00 am to 6:00 am as we set in our implementation since such connectivity events would not possibly be continuously generated by a person for a long time. We store the mac address of static devices and exclude them from the occupancy computation.

IV. Deployment and Evaluation

We have had the occupancy system deployed and running in more than 30 buildings in UCI and Ball State University campuses for nearly 3 years. Figure 5 is one application that displays the occupancy for each floor in engineering buildings in low/medium/high levels in the UCI campus. The system could also display the exact occupancy number for other granularities of locations, such as building/floor/region/room, at any customized time range. It is worth mentioning that the occupancy information displayed in the dashboard could be updated automatically based on the window query, such as giving the occupancy of a given location in the last 10 minutes, by ingesting the streaming data and updating the occupancy numbers in near real-time.

To evaluate the accuracy of our occupancy system, denoted as L-Occupancy (LOCATER-based Occupancy system), we

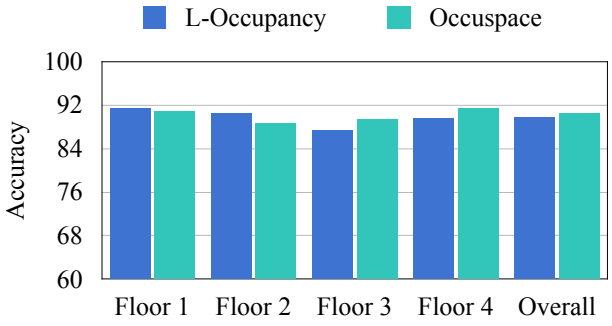

Fig. 6. Accuracy of L-Occupancy VS Occuspace.

manually collected 200 data points as the ground truth data in the Langson Library in University of California, Irvine. One such data point refers to $(st, et, location, occupancy)$, which represents the *occupancy* in the observed *location* in time range (st, et). In particular, we manually count the occupancy number on each floor, i.e., location is floor level, and the time interval is 5 minutes, i.e. $et - st = 5$ minutes. We compare L-Occupancy against a leading commercial specialized occupancy system, Occuspace [10], that is deployed in the Langson Library. We use accuracy to measure the quality of results from two occupancy systems. In particular, we count accuracy as the average difference between the predicted occupancy and true occupancy number divided by the truth over all data points in some given location. In Figure 6, we show the occupancy numbers from L-Occupancy and Occuspace on different floors in the library as well as the overall accuracy. As we can see, the performance of L-Occupancy is similar to Occuspace on all floors, i.e., they both achieve around 90% accuracy. Given that our solution is zero-cost while the hardware, as well as the annual maintenance expenses related to Occuspace, are expensive, such a result is very usable and should be impressive.

V. Conclusion

Occupancy is an important piece of information to enable building energy savings or space planning strategies. This paper proposed an occupancy solution based on the WiFi connectivity events data and provides an (almost) zero-cost solution, that estimates occupancy passively at (near) real-time. The occupancy system we built has been deployed and used in universities to have a real-world impact. Important future work involves how to build a large-scale occupancy benchmark to help evaluate the effectiveness of various occupancy strategies in different scenarios.

Acknowledgement

This material is based on research sponsored by HPI and DARPA under Agreement No. FA8750-16-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is partially supported by NSF Grants No. 1527536, 1545071, 2032525, 1952247, 1528995 and 2008993.

References

[1] Yiming Lin, Daokun Jiang, Roberto Yus, Georgios Bouloukakis, Andrew Chio, Sharad Mehrotra, Nalini Venkatasubramanian. LOCATER: Cleaning WiFi Connectivity Datasets for Semantic Localization. PVLDB, 14(3): 329- 341, 2021.

[2] Agarwal, Yuvraj, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng. "Occupancy-driven energy management for smart building automation." In Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building, pp. 1-6. 2010.

[3] Oldewurtel, Frauke, David Sturzenegger, and Manfred Morari. "Importance of occupancy information for building climate control." Applied energy 101 (2013): 521-532.

[4] Yang, Junjing, Mattheos Santamouris, and Siew Eang Lee. "Review of occupancy sensing systems and occupancy modeling methodologies for the application in institutional buildings." Energy and Buildings 121 (2016): 344-349.

[5] Leephakpreeda, Thananchai. "Adaptive occupancy-based lighting control via grey prediction." Building and environment 40, no. 7 (2005): 881-886.

[6] Tesoriero, Ricardo, R. Tebar, José A. Gallud, María Dolores Lozano, and Victor M. Ruiz Penichet. "Improving location awareness in indoor spaces using RFID technology." Expert Systems with Applications 37, no. 1 (2010): 894-898.

[7] Gu, Yanying, Anthony Lo, and Ignas Niemegeers. "A survey of indoor positioning systems for wireless personal networks." IEEE Communicatio

[8] Salimi, Shide, and Amin Hammad. "Optimizing energy consumption and occupants comfort in open-plan offices using local control based on occupancy dynamic data." Building and Environment 176 (2020): 106818.

[9] Lin, Yiming, Pramod Khargonekar, Sharad Mehrotra, and Nalini Venkatasubramanian. "T-cove: an exposure tracing system based on cleaning wi-fi events on organizational premises." Proceedings of the VLDB Endowment 14, no. 12 (2021): 2783-2786.

[10] How many people are at your library? gym? office? restaurant? Occuspace. (n.d.). Retrieved February 21, 2023, from https://occuspace.io/