Monitoring Workers on Construction Sites using Data Fusion of Real-Time Worker's Location, Body Orientation, and Productivity State

Mohammadali Khazen

A Thesis in

The Department of

Building, Civil, and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science in Building Engineering at

Concordia University

Montreal, Québec, Canada

March 2022

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Mohammadali Khazen

Entitled: **Monitoring Workers on Construction Sites through a Fusion of Real-Time Worker's Location, Body Orientation, and Productivity State Data**

and submitted in partial fulfillment of the requirements for the Degree of

Master of Applied Science (Building Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair

Dr. Po-Han Chen

_____ Examiner

Dr. Amin Hammad

_____ Examiner

Dr. Po-Han Chen

_____ Supervisor

Dr. Mazdak Nik-Bakht

_____ Supervisor

Dr. Osama Moselhi

Approved by _____

Dr. Mazdak Nik-Bakht, Graduate Program Director

March 2022 _____

Dr. Mourad Debbabi, Dean of Faculty

Abstract

Monitoring Workers on Construction Sites using Data Fusion of Real-Time Worker's Location, Body Orientation, and Productivity State

Mohammadali Khazen

Traditionally, on-site construction production monitoring depends primarily on manual processes that are time-consuming and error-prone. State-of-the-art technologies have been utilized lately to improve these processes to support timely decisions pertinent to the productivity and safety of onsite operations. This research introduces a novel construction site monitoring system to track workers' location, body orientation, and productivity state. The developed system uses Bluetooth Low Energy (BLE) based reference transmitting beacons fixed on job sites and a set of receiving beacons mounted on workers' hardhats, chests, and wrists. The system works via three modules, i.e. (i) RTLS (Real-Time Location System) module; (ii) body orientation detection module; and (iii) productivity state detection module.

The RTLS module is developed to continuously track the location of the workers and subsequently extract the actual labor workspaces. The RTLS is explicitly designed for construction by satisfying requirements for widespread on-site adoption, including cost efficiency, deployability, scalability, adjustability to the construction site dynamism, and the expected accuracy. The main features of the developed RTLS are (i) substituting commonly used BLE receivers with BLE receiving beacons; (ii) proposing a modular infrastructure placement strategy; (iii) deploying Trilateration and Min-Max as localization techniques; (iv) post-processing the worker's estimated locations.

As per the body orientation detection module, it identifies workers' body orientation on the job sites, using the impacts of signal blockage by a human body to identify an approximate worker's body orientation. It works based on geometrical relationships and Received Signal Strength Indicator (RSSI) values between the chest-mounted receiving beacon and the reference transmitting beacons. Last but not least, the productivity state detection module determines workers' productivity state (i.e., direct work, support work, delay) and travel state, using the accelerometer sensor embedded in the body-mounted receiving beacons. Consequently, the collected data of the system modules are fused to augment real-time knowledge of workers' status on job sites.

Acknowledgments

I would like to sincerely express my gratitude to my advisors, Dr. Mazdak Nik-Bakht, and Dr. Osama Moselhi, for their support throughout my research. Specifically, I am thankful for their patient guidance and enthusiastic encouragement during the development of my studies. It was a terrific learning experience for me working under their supervision, and my deepest respect goes to them.

I am grateful to my examining committee members — Dr. Amin Hammad and Dr. Po-Han Chen– – for their valuable comments and feedback. I would also like to thank Mr. Jeffrey Dungen, CEO of reelyActive company, for providing the equipment and validating the results, which undoubtedly added value to the research outcome. I extend my thanks to Mr. Hagire Emrani from SNC-Lavalin for providing site knowledge and expertise and validating the outcomes.

I would also appreciate the helpful advice of Abdelhady Ossama Hosny, Araham Martinez, and Hassan Bardareh, good colleagues and friends who are always around whenever I need any help and encouragement. I appreciate all support from my colleagues at COMPLECCiTY lab, especially Arash Hosseini. I also thoroughly enjoyed the time I spent working with students at Construction Automation Lab, especially Angat Bhatia

.

This thesis is dedicated to my parents, Zahra and Hamid.

Table of Contents

Lists of Figures

List of Tables

List of Abbreviations

Internet of Things	IoT
Real-Time Locating System	RTLS
Building Information Modeling	BIM
four-dimensional Building Information Modeling	(4D) BIM
Single-Board Computer	SBC
Bluetooth Low Energy	BLE
Ultra-Wide Band	UWB
Radio Frequency Identification	RFID
Global Positioning System	GPS
Wireless Fidelity	WiFi
Line-of-Sight	LoS
Non-Line-of-Sight	NLoS
Standard Deviation	SD
Average	Avg.
Direct Current	DC
Inertial Measurement Unit	IMU
Received Signal Strength Indicator	RSSI
Universally Unique Identifier	UUID
Neural Network	NN
Deep Neural Network	DNN
Conventional Neural Network	CNN
Machine Learning	ML

Random Forest	RF
Generalized Linear Regression	GLR
k-Nearest Neighbours	kNN
Gradient Boosting Decision Tree	GBDT
Mean Absolute Error	MAE
Root Mean Square Error	RMSE
Mean Percentage Error	MPE
Identity	ID

Chapter 1 – Introduction

1.1) Motivation and background

Construction job sites are dynamic environments with various workers and equipment operating simultaneously. The systematic control of construction operations can potentially change the business processes on the job sites by providing automated data acquisition and analysis for productivity and safety, among other applications [1],[2]. Traditionally, on-site monitoring techniques primarily depend on manual processes that are time-consuming and error-prone [3]. However, a range of state-of-the-art technologies has been applied lately to effectively assist construction managers and safety inspectors in making rational decisions supporting the management of daily construction activities and site monitoring [4]. Recent studies have highlighted that indoor localization applications can manage the worksite more effectively [5],[2]. One of the applications of real-time location estimation is to improve safety management on construction sites [6]. Unlike the labor-intensive and error-prone traditional methods of manual observation, automated safety monitoring allows continuous and accurate observation of construction site conditions [4]. For instance, workers' location data can help decision-makers detect common site accidents, such as collisions between workers and equipment and worker proximity to danger zones [7].

Furthermore, indoor localization technologies can improve on-site productivity by closely monitoring the workers. Worker's production rate is the primary indicator of productivity. Mainly, distinguishing between direct work time and travel idle/wait times for a specific construction task can give insight into more productivity models [8]. The recent use of Real-Time Locating Systems (RTLS) focusing on the geographical mapping of worker locations results in trajectories to quantify the time spent in specific workspaces [9]. Hence, construction worker tracking on construction sites allows identification and tracking of the workforce to support effective progress monitoring, activity sequence analysis, and productivity measurements and enhance safety management [10]. RTLS location data is often integrated with Building Information Modeling (BIM) to map location data on geometrical contextual information of job sites. This integration enables decision-makers to make informed decisions supporting the management of construction activities.

For example, geographical mapping of worker locations and trajectories quantifies the time spent in workspaces marked in the BIM model [11]. Identifying incidents of proximity/trespassing by the workers to the defined danger zone is another example of the integration of RTLS and BIM. Last

but not least, in the era of global pandemic diseases, helping determine the workspaces where the crews violate social distancing rule is another use case of such integration [12]. Moreover, many studies have adopted the head orientation of the workers to capture their visual attention on the job site [13]. Based on workers' field-of-view information, hazard proximity systems are developed to generate safety alarms once the workers are not aware of the hazards. On the other hand, many previous research efforts have monitored worker state to assess their productivity on the job sites. The analysis of workers' productivity information is used for various purposes, such as cost estimating and claim evaluation [11].

1.2) Problem Statement

Although potentials for the worker monitoring systems have been explored on construction sites, there are critical factors in developing such systems that have not been carefully considered in the previous research. Firstly, the existing BLE-based RTLS relies on mobile phones and Direct Current (DC) electronics needing electrical wiring to operate. It causes interference with the construction workflow of job sites and can adversely affect the construction workers' hazard recognition [2],[14]. Also, the previous research studies lack an infrastructure placement strategy that may be troublesome in the construction environment where the infrastructure must be relocated frequently. Secondly, BLE technology has not provided an impressive level of accuracy based on the numbers reported in the literature, so it may not be suitable for safety-related applications [9].

Thirdly, most previous studies only focused on RTLS location data to monitor workers on job sites. In most cases, location data cannot provide deep insight into the worker's status. Although some studies incorporated the worker's productivity state by analyzing the workers' displacement per time unit, that raises the concern that the productivity state data might be unreliable due to the inaccuracy associated with the RTLS location data. Besides, recent studies used Inertial Measurement Unit (IMU) sensors or computed the direction of the worker's displacement through location data to detect workers' field of view on a job site for safety management applications. The drawbacks of these techniques include increasing the system's implementing cost and unreliability associated with the RTLS location data.

1.3) Research Objectives

The main goal of this research is to study the possible use of RTLS data integrated with body orientation and productivity state data to monitor construction worker(s) with a focus on practical deployment, affordability, and sufficient accuracy. Accordingly, hardware/software infrastructure

and analysis models are proposed to monitor workers on construction sites. To address the problems stated in the previous section, the research objectives of this study are categorized as:

1. Developing a modular BLE-based infrastructure placement strategy with a minimal dependency on wiring and electricity outlets to strategically place BLE-based devices on job sites.

2. Developing a BLE-based RTLS to continuously locate workers on job sites with high accuracy in static and dynamic scenarios and low computational time.

3. Developing an independent body orientation detection module from RTLS location displacement data to identify workers' focus orientation.

4. Developing an independent productivity state detection module from RTLS data to identify workers' productivity states.

It is noted that the RTLS and body orientation detection modules are developed for the indoor environment. Besides, the productivity state detection module is developed for repetitive constriction activities, including painting, plastering, and masonry.

1.4) Thesis organization

This research is presented in four chapters. After a literature review on the comparison of different RTLS technologies, BLE-based RTLS configurations, and worker monitoring systems in chapter 2 (literature review), a worker monitoring system is introduced in chapter 3 (developed methods for worker monitoring system). This chapter presents an overview of the proposed methodologies by introducing the development of the RTLS module, body orientation detection module, and productivity state detection module. The RTLS module comprises three parts as follows: (i) RSSI-distance prediction model; (ii) localization estimation model; and (iii) estimated location post-processing model. Each body orientation detection and productivity state detection module comprises two parts: (i) data collection and (ii) model architecture and training. At the end of this chapter, the modules are validated and verified by a set of in-lab experiments, details of which are discussed. Finally, the concluding remarks, the research contributions and impacts, and limitations and future remarks are highlighted in chapter 4 (conclusion).

Chapter 2 – Literature Review

This chapter provides a literature study on the comparison of RTLS technologies, BLE-based RTLS, and worker monitoring systems sequentially. The first section provides a survey of the most used RTLS technologies in construction and compares them based on the project-related assessment factors. In the next section, the localization techniques used in BLE-based RTLS are discussed, and a comparison between varied RTLS settings and infrastructure reported in the literature is provided. Last but not least, different types of worker monitoring systems on job sites are explained.

2.1) Introduction of Different RTLS

Although the feasibility of deploying RTLS to manage the construction job site is highlighted in previous research, selecting the appropriate technology might differ from one project to another based on different factors. This section proposed practical criteria based on highlights in the selected experimental studies during different phases of their RTLS experiments to evaluate the localization systems. The studies include scientific journal papers in the construction engineering and management domain published after 2010. Using a construction site as a case study and providing a detailed explanation of the systems' setup and configurations are the metrics for selecting the papers. Then, an evaluation framework is created to assess the feasibility of implementing localization systems on job sites. Finally, the capabilities and weaknesses of the systems are investigated by using our proposed framework.

The criteria included in the framework to evaluate the localization technologies can be categorized into the following parts. *(i) Required System Infrastructure to be Installed on Construction Site* – One of the significant barriers to widely adopting localization systems on construction sites is their dependency on the number of system devices required to be installed on job sites. The time and effort required for installing the devices can pose significant challenges to the construction workflow. The system infrastructure required to be installed on-site must have a compact size to increase the adoption rate of localization technologies [12]. In some cases, the safety implications resulting from infrastructure such as cables between devices can outweigh the benefits the localization system may bring to the site. Specifically, the wired connections between the system devices for supplying Direct Current (DC) power or transmitting data can influence the safety of the workers [2].

(ii) System deployability in construction – Another decisive factor for deploying the localization systems on-site is their usability, referring to the level to which a worker expects the system to be free of effort to be used. Specifically, wearable devices should not inconvenience workers as they perform regular tasks [15]. The size of wearable devices that the workers must carry can affect the acceptance of the workers' localization technologies [16]. The hardware embedded in the worker should be adequate to wear without interferences to the workflow [21]. Furthermore, the system infrastructure of some of the technologies is sensitive to the orientations of the installed sensors on-sites. The orientation shift of the installed sensors of different technologies can negatively affect the localization performance [17]. In some technologies, the localization coverage is only extended to the defined portion of the construction site, where sensors are placed in a predetermined geometry and fixed orientation [18].

(iii) System vulnerability to indoor environments – The main challenge of using localization systems is the harsh environment of the job sites. The signals generated between devices can be adversely affected by Non-Line of Site (NLoS) propagation or other interference [19]. Specifically, performance degradation with an increase in distance between transmitting and receiving nodes in communication-based localization systems is observed. The radio signal is subject to reflection, diffraction, and scattering due to the construction objects in that environment [3]. The sensitivity of the technologies to a specific environment is a decisive factor in selecting the appropriate technology for the job sites [6]. *(iv) System localization accuracy* – The location data's reliability directly affects the decision-making activities in construction management [20]. The accuracy of localization technologies can be measured by error calculated as the average Euclidean distance between the ground truth and estimated location coordinates [9]. The level of positioning accuracy required for each application might differ from one another due to their various location information requirements such as symbolic location, absolute location, and relative location [21]. The deployment of safety alert systems requires a higher level of accuracy of the localization system, whereas, for the productivity-related operations, a mediocre level of accuracy might be acceptable [22].

Many IoT (Internet of Things) technologies are commonly used by indoor tracking solutions, including Bluetooth Low Energy (BLE), Radio Frequency Identification (RFID), and Ultra-Wideband (UWB) technologies [23]. Also, there are less commonly used alternatives for indoor localization, including embedded sensors, Lidar and laser scanning, high-resolution video camera, digital photogrammetry, and WiFi [2],[24]. The BLE technology-based system uses transmitters/receivers attached to the walls or ceilings of indoor environments to estimate the

location of the target node. A BLE-based system comprises a receiver and transmitter which can wirelessly communicate with one another. The BLE receiver is either fixed in a known location or worn by the workers, and it can capture the Received Signal Strength Indicator (RSSI) from the beacons to estimate the worker's location [19][25]. The two traditional BLE-based locating system architectures include (i) mobile beacons that send BLE signals to fixed infrastructure that act as receiver and gateway; (ii) mobile receiver (e.g., mobile phone) that act as a gateway to receive BLE signals from fixed beacons; (iii) mobile beacon to receive BLE signals from fixed beacons and send the data to a gateway (DirAct approach). BLE is considered the most cost-effective among other IoT-based technologies and appears reasonably accurate for many indoor localization applications in the construction domain [26].

Similar to the BLE technology, the RFID system typically consists of two components: readers and active or passive tags operating at a specific radiofrequency. RFID readers are placed around the sensing area, and the tag is worn by workers that are tracked and localized [27],[28]. It operates based on capturing the signal from a tag by the readers. The tag's distance from the reader is estimated, and then by having at least three readers, the tag's location is determined through triangulation [29]. There is a problem associated with the simultaneous identification of multiple tags, which can reduce the accuracy of the RFID systems [30]. Another drawback to this technology is that liquids and metals substantially affect the RFID system's readability range and data transfer rate, resulting in a poor system localization accuracy [31]. The Ultra-wideband (UWB) technology is another real-time location system with better performance, in terms of accuracy, compared to the RFID technology. Since it uses very narrow pulses of radiofrequency energy, it is appropriate for environments where the multi-path effect can happen [30]. Although UWB technology can theoretically achieve a high level of accuracy compared to other localization technologies, many papers showed that the performance of a UWB system is highly affected by Non-Line-of-Sight (NLOS) from readers to the target object [25]. Besides, the main disadvantage of implementing UWB technology is its expensive hardware investment (of up to $140 per square meter, compared to roughly $20 for RFID and $5 for BLE [27].

2.2) Comparison of Different RTLS

After introducing the technologies, a comparison is made based on the proposed criteria to identify the strength and weaknesses of the technologies.

(i) Required System Infrastructure to be Installed on Construction Site – The UWB technology infrastructure consists of receiver sensors required to be installed on the job site. A wired

connection between the sensors is necessary to utilize RTLS using the TDOA method. Additionally, a data transmission cable is needed to be run from a master sensor to the computer running the software platform [7]. DC power or power over ethernet (POE) are the alternatives to supply electrical power to the sensors. Regarding RFID technology, its hardware components typically include RFID mobile readers, RFID encapsulated tags, RFID label tags, and an RFID label tag printer [32]. By contrast to the UWB, the read-only (passive) RFID tags do not require a battery or electrical wiring since they are activated by the electromagnetic energy that the reader emits [31]. However, electrical power is needed for the hand-held RFID readers. The communication architecture of BLE technology is relatively similar to the RFID, and it includes three components transmitting node, receiving node, and a gateway [33]. A smartphone or a gateway can play the role of receiving node, and BLE beacons act as the transmitting nodes. In the case of using a smartphone as a receiver, BLE beacons are the only infrastructure needed to be installed on-site for the localization system [34]. As a result, a BLE-based system using a smartphone could be a technology that requires the minimum equipment to be installed on the job sites compared to others, and it requires the least amount of wiring work for its devices.

(ii) System deployability in construction – The requirement of establishing a high-quality data transmission cable between UWB receivers in dynamic environments such as job sites can affect the workers' safety and workflow. The process of calibrating the UWB sensors and the requirement of repeated calibrations can be time-consuming, especially when the power supply is down [17]. The total man-minutes for the deployment of four UWB sensors can range from 300 to 200 man-minutes, and the number increases further as the job site progresses [2]. The UWB tags an adequately small size of 24 mm x 13 mm to be worn by the workers without interference to the workflow [15]. Likewise, the BLE beacon has a small 13.5 mm x 13.5 mm size to be placed on the workers. However, using a smartphone as a receiver causes distraction for the workers, adversely affecting hazard recognition, safety risk perception, and safety performance of the construction workers [35]. As per RFID, its reader has a larger size than its equivalent in other technologies, and it has dimensions of 160 mm x 77 mm x 169 mm. This can cause inconvenience for the workers by holding it for a long time. On the other hand, the RFID has compact-size tags which can be easily placed on job sites without interfering with the construction workflow.

(iii) System vulnerability to indoor environments – In UWB, the system can only localize the target node when it falls within the defined project area. It means positions outside the boundaries of the defined coverage area would not be shown in the log file. Although the positioning accuracy of the UWB system can be in the range of a few centimeters, the system cannot perform well in

complex and noisy environments [17]. Besides, increasing the tag's distance from the receiver can significantly drop the localization accuracy. It is observed that by increasing the number of target tags, the localization accuracy can drop down to 33%. It can adversely affect the scalability of the UWB system when it comes to tracking many workers simultaneously [7]. Besides, the obstacles existing on job sites can cause UWB communication packet loss which can be the source of the localization error [15].

Concerning RFID technology, the major disadvantage to RFID is the interference among its components by some materials. The proximity of liquids and metals substantially affects the RFID system's readability range and data transfer rate [31]. Besides, the performance of RFID is susceptible to an environment with multipath effects and non-line-of-sight (NLOS) signal propagation between the tag and the receiver [29]. Similarly, the BLE technology is influenced by the multi-path effects of the obstacles in construction sites. Increasing the transmission power of the BLE tags can mitigate this effect to some extent; however, it can increase the energy consumption of the tags, resulting in higher maintenance costs by replacing the batteries. Moreover, it is found that when the transmitting node is placed at distances more than 2.00 m from the receiving node, the estimated distance between the beacons may not be reliable [36].

(iv) System localization accuracy – Since the localization accuracy of different systems cannot be used for one-to-one comparison due to the variations in different construction sites, specifications related to the testbed of the studies are also considered in addition to the localization error. The considered parameters are the testbed dimensions/area, testbed environment, the quantity of the used equipment, and the state of the target object during the experiment. Table 1 shows the specifications and the localization accuracy of the localization systems.

Table 1: Claimed Positioning Accuracy of the Localization Systems

Technology	Testbed Dimensions/ Area	Testbed Environment	Installed Devices On-Site	Target Object State	Accuracy	Ref.
UWB	40.0 m x 55.0 m	Open outdoor site without obstacles	4 UWB receivers	Stationary	Avg. Error = 0.18 (m)	[2]
UWB	13.4 m x 9.6 m	Laboratory containing many metallic surfaces	4 UWB receivers	Mobile	Avg. Error = 0.30 (m)	[16]

UWB	≈ 2400.00 m2	Construction pit (temporarily placing steel materials)	8 UWB receivers	Mobile	Avg. Error = 0.48 (m)	[15]
RFID	30.0 m x 30.0 m	Cast-in-place concrete building	21 RFID tags Unmanned aerial vehicle (UAV) equipped with GPS	Mobile	Avg. Error = 2.32 (m)	[37]
RFID	7.5 m x 5.5 m	Basement of an ongoing apartment without obstacle	4 RFID readers	Mobile	Avg. Error = 1.27 (m)	[29]
RFID	75.2 m2	Cast-in-place concrete building	24 RFID tags RFID tag printer	Mobile	Avg. Error = 1.00 (m)	[32]
BLE	10 m x 5.0 m	Rwoom cluttered with obstacles (walls)	12 BLE beacon	Stationary	MAE = 1.28	[36]
BLE	6.0 m x 3.0 m	Office with obstacles	4 BLE beacon	Stationary	Avg. Error = 0.70 (m)	[34]
BLE	9.0 m x 3.0 m	Construction site	8 BLE beacon	Mobile	Avg. Error = 0.98 (m)	[34]

As seen from the table, UWB technology can provide a high positioning accuracy, with an error of as low as 18 (cm). The BLE and RFID technologies can also deliver a reasonable level of localization accuracy by assigning more devices per unit area on the job sites. Due to the high positioning accuracy of UWB, it is generally used in construction activities that require a higher level of positioning accuracy, including critical crane lifts, erection of important steel structures, and off-site fabrication [2]. Another application of using UWB is construction resource (worker and equipment) tracking leading to safety monitoring practices by introducing safety boundaries and danger zones [6],[7]. Regarding safety management, RFID technology can also be used to provide decision-makers with a warning if a worker is in proximity to hazardous or tagged areas [27]. However, the most used application of RFID is the localization of assets to derive knowledge about construction project status [31].

The real-time characteristic of BLE localization technology makes it capable of exchanging a large amount of data. Thus, they are mainly integrated with building information modeling (BIM) for safety management and productivity monitoring applications on job sites [4]. BLE technology is also considered the most cost-effective among other IoT-based technologies and has low power consumption, which allows the beacons to run on batteries for many months. In addition, it provides a reasonable amount of accuracy for many indoor localization applications in the construction domain. Hence, BLE technology is used in this study to develop RTLS and, accordingly, the worker monitoring system.

2.3) BLE-based Real-Time Locating System (RTLS)

This section explains the common localization techniques used for BLE-based RTLS. Further, a comparison is made between various RTLS settings and infrastructure reported in the literature.

The most common location estimation techniques for the BLE technology are proximity detection, fingerprinting, and trilateration [38]. Each of these techniques has unique advantages and limitations, according to the domain of the localization application. The proximity technique provides approximate location information of a target node. In other words, it checks the presence of the target node to be located in radio coverage of the reference beacon. It only estimates the distance between the target node and the beacon instead of providing the exact location of the target node. [18]. Zhuang et al. (2016) proposed an RSSI real-time correction method using a separate Bluetooth gateway to detect and adjust the RSSI fluctuations of surrounding Bluetooth beacons in real-time. Also, they used Particle Swarm Optimization for optimizing BP Neural Network (PSO-BPNN) to train the RSSI distance model to reduce the localization error [33]. Mackey et al. (2020) applied various Bayesian filtering techniques, including Kalman, particle, and Non-parametric Information. They concluded that these filtering techniques could improve proximity estimation accuracy up to 30% [39].

Fingerprinting is a localization technique comprising two parts. Firstly, the RSSI values received by a measuring device in the known locations are recorded, and the location coordinates are in the fingerprint database. Then, the worker wearing the BLE device to be located measures the RSSI values and compares them with the data in the fingerprint database to predict location [40]. Huang et al. (2019) analyzed the relationship between RSSI and the distance between the beacons. They designed an algorithm to combine fingerprinting and geometric techniques to increase accuracy. The paper also evaluates the effects of an increase in the number of BLE fixed receivers and the localization performance [26]. Subedi et al. (2017) improved the traditional fingerprinting localization by combining it with the weighted centroid localization technique. They could reduce the required reference points by more than 40% without any adverse effect on the localization error [41]. Ke et al. (2017) combined the localization algorithm and hardware configuration to reduce the error rate. They applied Weighted K-Nearest Neighbors (WKNN) algorithm and a mean filter for smoothing the estimated locations [42].

Last but not least, the trilateration technique uses the RSSI value from the fixed BLE devices; the signal is used as a proxy of distance, acting as the radius of a circle with its center at the BLE devices. The intersection of three or more circles determines the worker’s location [43]. Paterna

et al. (2017) investigated the behavior of BLE channels and the multi-path effect on the accuracy of localization. They improved the system's performance by modifying the trilateration technique and applying the Kalman filtering on location data [30]. Huang et al. (2019) proposed hybrid trilateration and techniques to solve the problems resulting from the dense Bluetooth environment. The Kalman filter is used to merge the trilateration and results [22]. Shi et al. (2020) proposed a tri-partition RSSI classification to reduce RSSI fluctuation, and they used it as a tracing algorithm as an RSSI filter [44].

Several studies are reported for indoor localization using each of the three techniques mentioned above. Since a direct comparison of those systems only based on their accuracy will not be precise due to the contextual differences among the experiments. Hence, specifications related to the studies' test-beds and hardware requirements are also considered, along with their localization error. Besides, the proximity technique is excluded from the comparison since it can only provide approximate location information of the target. The comparison table consists of evaluation metrics including (i) test-bed dimensions, representing the dimensions of the RTLS coverage area; (ii) type of reference devices, introducing the type of fixed BLE devices used as reference nodes in the test-bed; (iii) number of the reference devices, showing the number of fixed BLE devices used as reference nodes in the test-bed; (iv) type of device worn by the target, representing the BLE tracking device worn by the worker; (v) accuracy, identified through the localization error of the RTLS; and (vi) localization techniques in the RTLS: representing the used localization techniques in the RTLS. Table 2 compares various BLE-based RTLS reported in the literature within these dimensions.

Table 2: Summary of relevant related research work

Testbed dimensions (m x m)	Type of reference devices	No. of reference devices	Type of devices worn by the target	Accuracy (m)	Technique	Year	Ref.
14.0 x 12.0	AC Gateway	13	BLE Beacon	MAE = 0.97	Fingerprinting	2021	[45]
8.0 x 8.0	AC Gateway	4	Smartphone	RMSE = 1.00		2019	[46]
7.1 x 4.2	BLE Beacon	9	Smartphone	MAE = 1.12		2019	[47]
25.0 x 15.0	Single Board Computer (SBC): Raspberry Pi	13	Smartphone	MAE = 1.18		2020	[48]

	/ BLE Beacon						
80 m²	BLE Beacon	26	Smartphone	MAE = 1.23		2021	[49]
4.0 x 3.0	BLE Beacon	5	Smartphone	MAE = 1.93		2017	[50]
14.0 x 11.0	AC Gateway	3	BLE Beacon	MAE = 2.58		2021	[51]
8.8 x 5.6	BLE Beacon	8	Smartphone	RMSE = 0.76		2020	[32]
10.0 x 5.0	BLE Beacon	12	Smartphone	MAE = 1.28		2019	[52]
8.0 x 3.5	AC Gateway	6	BLE Beacon	MAE = 1.78		2019	[53]
8.7 x 6.2	SBC (Raspberry Pi)	4	BLE Beacon	90% below 1.82	Trilateration	2017	[54]
6.0 x 5.5	SBC (Raspberry Pi)	3	BLE Beacon	MSE = 2.98		2019	[55]

2.4) Worker Monitoring System

This chapter provides a literature study on the integration of RTLS and BIM for safety management and productivity monitoring applications sequentially. Safety management is primarily about identifying incidents of danger zone proximity/trespassing by the workers. The productivity monitoring mainly includes (i) determining the heatmap of workspaces to compare the ‘as-planned’ VS ‘as-happened’; and (ii) analyzing workers’ time spent in workspaces.

Huang et al. (2021) used RTLS to detect the proximity of the workers to danger zones on construction sites. They processed the worker’s raw location data and extracted insightful information, including the target node’s position, orientation, and velocity. Then, their model detected the unsafe proximity of the workers to danger zones and generated safety alerts [56]. Park et al. (2017) integrated RTLS with BIM-based hazard identification for safety monitoring purposes. Potential unsafe areas are firstly defined in a BIM model, and workers’ location data were used to detect events where workers are exposed to pre-defined danger zones. In addition, they analyzed the trajectories patterns of the workers with respect to the danger zones [57]. Chan et al. (2020) integrated the field-of-view of workers as a proxy for hazard awareness to develop an improved proximity warning system to danger zones. They developed a rule-based model for the warning system, followed by a virtual experiment to evaluate the Integration of worker body orientation in safety alarm systems [58]. Teizer et al. (2013) the authors proposed a safety management system based on data segmentation of the real-time location data of the workers. A rule-based system determined the worker'' state with respect to the danger zones based on their velocity. For instance, a velocity higher than 0.3 m/s was considered a traveling activity, and lower than 0.3 m/s indicated as stopped traveling. Then, the worker'' proximity to danger zones was

automatically determined based on pre-defined criteria, including the target's speed and the cluster radius encompassing (x,y) coordinates [8].

As per productivity monitoring, Mohanty et al. (2020) used BIM to evaluate the worker'' productivity using the percentage of time spent in the assigned workspaces. The real-time location data of the workers was matched with their assigned workspace and was shown as a heatmap on the site layout to monitor the crowd areas [59]. Zhao et al. (2017) analyzed the worker'' daily movements to reveal work interruptions and non-direct moves. They identified workers' productivity by comparing their time spent distribution on each location to the planned workspace. The time spent was calculated based on the time difference between the start and finish time in which the worker entered his/her workspace (the first and the last location sample generated within the workspace) [60]. Zhao et al. (2020) developed an automated process of workspace generation based on the location data of workers. They initially created heatmaps based on the density of workers' presence, which could outline an estimate of the workspace with fuzzy borders. Accordingly, they proposed various applications, including detecting takt areas in takt time planning, estimating the threshold value of the uninterrupted presence of crews, and predicting potential congested zones [61]. Table 3 summarizes the choice of sensors and the type of data reported in the literature.

Table 3: Summary of Localization Integration with BIM in the construction domain

Application	RTLS Technology	Worker's Body orientation Data	Worker's Activity Data	Year	Ref.
Safety Management	BLE	Location Data	Location Data	2021	[56]
	UWB	Gyroscopes/ Accelerometer	--	2020	[58]
	BLE	--	--	2017	[57]
	RFID	--	--	2017	[62]
	UWB	--	Location Data	2013	[17]
	RFID	--	--	2010	[63]
	RFID	--	--	2010	[64]
Productivity Monitoring	BLE	--	Location Data	2020	[59]
	BLE	--	Location Data	2020	[61]
	BLE/Wi-Fi	--	Location Data	2018	[60]
	RFID	--	--	2016	[65]
	UWB	--	Location Data	2013	[20]

2.5) Gaps in the Literature

Although potentials for the worker monitoring systems have been explored on construction sites, there are critical factors in developing such systems that have not been carefully considered in

the previous research. The gaps can be categorized into three parts as follows. (i) On-Site Deployability of the Locating System – Mobile phones and Direct Current (DC) electronics needing electrical wiring to operate have been extensively used as receivers in the BLE-based RTLS. The requirement for establishing a cable from the reivers to DC power can interfere with the construction workflow the dynamic environment of construction sites. As per the mobile phone, the distraction caused by using it on-site can adversely affect the construction worker'' hazard recognition and safety risk perception. (ii) Lack of Placement Strategy of the System Infrastructure – Since the signal interference caused by the existing obstacles and BLE device'' sensing range can affect the localization performance, the system infrastructure needs to be relocated and be distributed according to the new localization needs and site layout as construction progresses. The previous research studies did not accommodate the placement of system infrastructure in their test-bed; instead, they placed the infrastructure to provide a coverage area for their specific test environment. It may be troublesome in the construction environment where the infrastructure must be relocated frequently due to the job site's dynamism. (iii) High Localization Accuracy and Precision – Generally, BLE technology has not provided an impressive level of accuracy based on the numbers reported in the literature. Although it may be suitable for applications that are less sensitive to accuracy, such as productivity monitoring and site attendance, safety-related applications require a high level of accuracy in real-time. (iv) Monitoring Workers Based On Location Data – The majority of the previous studies only focused on the location data to check the presence of a worker within workspaces/danger zones boundaries. The drawback of that method is that the worker'' location data cannot provide deep insight into the worker's status. For instance, a worker's presence in a workspace cannot necessarily reveal the worker's productivity state. As per safety management, detecting a worker's unsafe proximity to a danger zone might not be insightful since the worker might already be aware of it and must work or travel in its vicinity. (v) Extracting Worker'' Productivity State and Field Of View Information From RTLS Data – Some studies proposed incorporating the worker's productivity state by analyzing the worker'' displacement per time unit to address some of the discussed problems. That would raise the concern that the productivity state data might be unreliable due to the inaccuracy associated with the RTLS predicted location data, specifically in the scenarios where the worker travels a short path or is static for a long time. In more recent studies, the worker's field of view on a job site is also included in the worker'' information data for the safety management-related applications. The drawbacks of this technique include increasing the system's implementing cost and unreliability associated with the RTLS location data.

Chapter 3 – Developed Methods for Worker Monitoring System

This chapter introduces an automated system that enables real-time data-driven insights at the construction workforce, leveraging BLE beacons and the Building Information System (BIM). The system uses data fusion of real-time workers' location, body orientation, and productivity state to monitor workers on construction sites continuously. The system has been designed to have three accelerometer-embedded receiving beacons mounted on a worker's hardhat, chest, and wrist. The hardhat-mounted and chest-mounted receiving beacons capture BLE signals from the reference transmitting beacons strategically placed on a job site and transfer them to the cloud database through a gateway. The signals captured by the hardhat-mounted receiving beacon are used as a proxy of distance, and the worker's location is identified using localization techniques. Then, the location data is mapped on geometrical contextual information of job sites to check the presence of workers in workspaces. The detected transmitting beacons by the chest-mounted receiving beacon are used to identify the worker's body orientation. Last but not least, the acceleration data collected by the accelerometer embedded in the receiving beacons is used to identify the worker'' productivity state. This chapter provides an overview of the hardware and software, and algorithms deployed in the proposed worker monitoring system

3.1) General Architecture of the System

The system's communication architecture comprises four main components, i.e., Data Advertisement; Data Reception; Data Transfer, and Cloud Computing. The Data Advertisement component comprises transmission beacons fixed in the space, the primary function of which is to broadcast radio signals (BLE packets) that cover a particular area. The Data Reception component consists of receiving beacons worn or carried by the workers, which capture the packets from transmitting beacons, with an RSSI proportional to their distance, and add that information in the packet. The Data Transfer module is a gateway. The information (packet) collected by the receiving beacons is transmitted back to a central cloud computing system through the gateways. They receive packets in range and stream them to the cloud via WiFi.

Last but not least, Cloud Computing stores data packets in a database through which the localization models process the data [36]. In the proposed RTLS, the fixed reference transmitting beacons periodically broadcast signal data to the wearable receiving beacon through the data BLE packet. Simultaneously, upon receiving the beacon BLE advertising packet from the transmitting beacons, the wearable receiving beacon read the RSSI value using its radio circuitry. Then, the wearable receiving beacon forwards the measured RSSI data encapsulated in a data

BLE advertising packet (collected from the transmitting beacons) to the gateway. Finally, the gateway sends the data back to the cloud computing service (Pareto Anywhere middleware) via a WI-FI network. The overview of the communication architecture of the proposed system is shown in Figure 1.

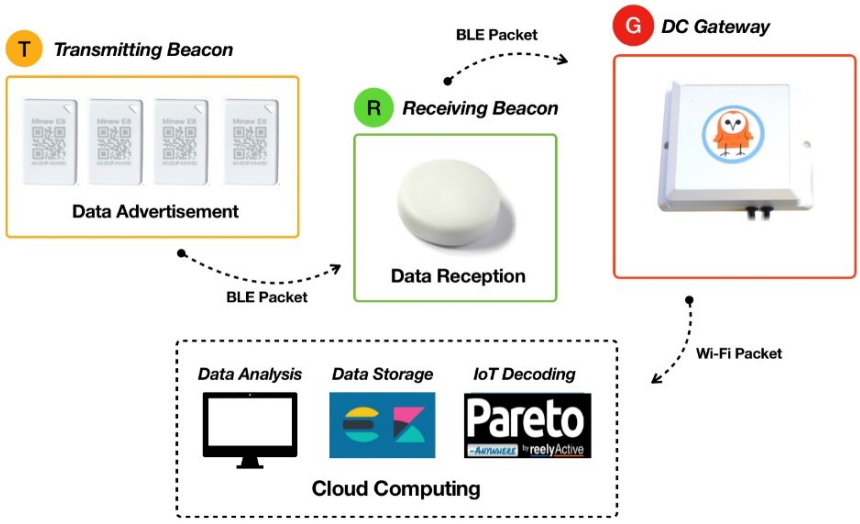


Figure 1: Overview of Communication Architecture of the Proposed System

A standard BLE advertising packet supports a payload size of 31 bytes, meaning that its carrying capacity is constrained by its payload size. Thus, the receiving beacon is programmed to broadcast the identity (ID) of a maximum of three transmitting beacons captured with the highest RSSI value at a time. The collected RSSI measurements with respect to the three transmitting beacons are recorded with their respective timestamp that indicates the date and time it was created. Since the infrastructure placement requires the minimum number of BLE devices per unit coverage area and their placement in the effective BLE devices sensing range, a modular infrastructure placement strategy on the job site is proposed. The proposed infrastructure placement strategy consists of repetitive modules similar in size, shape, and device placement that can be linked up to each other to cover the construction site. The modules can perform independently and are placed as required for localization on sites. Each module is composed of one gateway placed at its center and a certain number of square sub-modules, consisting of four fixed transmitters placed at each corner of the square. Moreover, the BLE beacons' configurations, such as Transmission Power and Scan Window, are tuned to trigger the sub-module change process in order for the receiving beacon to encounter sub-module change as it moves from one sub-module to the next one.

The software components of the monitoring system comprises three modules: (i) RTLS and workspace module, (ii) body orientation detection, and (iii) productivity state detection (appendices 1-3). The RTLS and workspace detection module pinpoints workers' real-time location and integrates it with a BIM model. To localize worker location, the RSSI values of the detected transmitting beacons by the hardhat-mounted receiving beacon are converted to a distance, acting as the circle's radius with its center at the transmitting beacon's location. The intersection centroid of the three circles is calculated using triangulation techniques. Then, the location data is mapped on geometrical contextual information of job sites to check the presence of workers in workspaces defined in the BIM model.

The developed RTLS algorithms generate three models: (i) RSSI-distance estimation model; (ii) Localization estimation model; and (iii) Localization post-processing model. After the records generated by the receiving beacon are stored in the dataset, the RSSI-distance estimation model converts the RSSI values in each record to a length representing the estimated distance between the transmitting beacon and the receiving beacon. The transmitting beacon’s ID in each record is then transformed by its (x,y) coordinates on the job site in order for the record to be in a readable format by the localization estimation model. The coordinate frame of the transmitting beacons is configured around a single transmitting beacon to be set as origin, and other transmitting beacons are normalized according to that origin. Once the locations of the transmitting beacons and the corresponding distances to the receiving beacon are determined, the localization model can process the data to estimate the receiving beacon’s position (x,y) coordinates. The estimated distance acts as the circle's radius with its center at the transmitting beacon's location. Depending on the radii and centers of the circles, the intersection centroid of the three circles helps predict the receiving beacon's location coordinates. Since the localization estimation model encounters various scenarios, a unique localization algorithm is developed for each arrangement. Finally, the estimated locations are post-processed to minimize distortions in the receiving beacon's location and improve the localization accuracy. Shifting the worker’s location to the strongest transmitting beacon and applying filtering techniques, including Exponential Smoothing, Simple Moving Average, and Kalman filters, are the processes followed in the localization post-processing model. The processing time for leading for estimating a worker’s location for 2160 records is about 6.5 seconds. The post-processing time for those records (2160) ranges from one to three seconds, depending on the type of filtering technique.

The body orientation detection module predicts the worker's body orientation in eight ordinal orientations on the job site. The module uses the RSSI values and geometrical relationships

between the detected transmitting beacons by the chest-mounted receiving beacon and the worker's predicted location. To this end, in-lab experiments were carried out to collect data between the chest-mounted receiving beacon and reference transmitting beacons for eight ordinal orientations. Then, a Deep Neural Network (DNN) model was trained to predict a worker's body orientation and focus orientation over a specific period of time.

Last but not least, the productivity state detection module detects workers' productivity states, using motion signals generated by the accelerometer embedded in the three mounted receiving beacons. The productivity states include direct work, non-direct work, walking, and idling. The productivity state is considered direct work once a worker performs the main task(s), whereas it is considered non-direct work once the worker performs the secondary task(s). The productivity state detection module comprises six models, and each is individually trained for a specific worker. For each model, the tri-axial acceleration data is segmented to generate activity frequency images that serve as the input to a trained Convolutional Neural Network (CNN) model to detect the worker's productivity state.

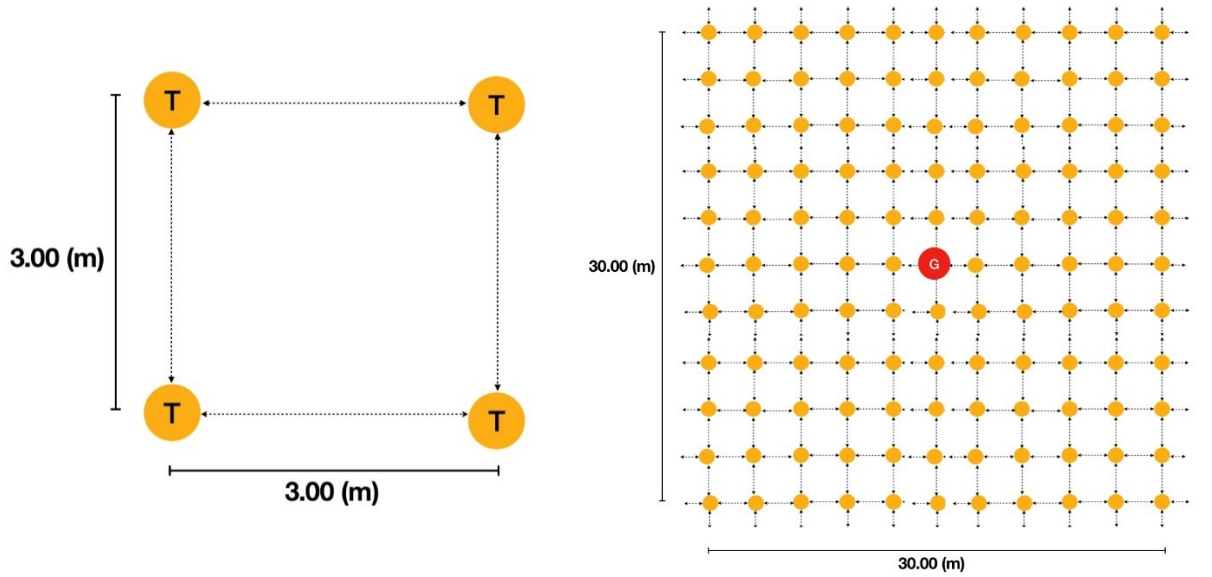
On average, the receiving beacon generates a record every 1.7 seconds with a standard deviation of 0.7 seconds. Since the sampling rate of the three receiving beacons is different, their generated records require time synchronization. The generated data of the chest-mounted and wrist-mounted receiving beacons are fuzed to the data of the hardhat-mounted receiving beacon. The time synchronization is based on the absolute minimum difference in the timestamp of the generated records. The workspace detection module uses the RSSI readings of the hardhat-mounted receiving beacon, and the body orientation detection system uses the readings of the chest-mounted receiving beacon. However, the productivity state detection module depends on the readings from the three receiving beacons. The modules can perform independently and be deployed as required for different applications except for the body orientation and workspace detection modules that rely on RTLS.

3.2) Infrastructure Placement Strategy

This section describes the proposed modular infrastructure to manage the placement of the fixed transmitting beacons and gateways on the job site. Then, the impacts of the BLE beacons' configuration and the developed algorithm on the efficiency of RTLS are explained.

This study proposes a modular placement system consisting of repetitive modules similar in size, shape, and device placement to distribute the RTLS infrastructure according to the site layout.

These modules can be linked to each other to cover the entire construction site or zones of interest. The modules can perform independently and are placed as required for localization on sites. Each module has a square shape composed of one gateway placed at its center, supporting a certain number of sub-modules. Each sub-module also has a square shape consisting of four fixed transmitting beacons placed at its corners. The gateway sensing range constrains the module's size, and the sub-module dimensions are determined by the maximum distance in which the transmitting beacon can reach the receiving beacon and send BLE packets for the RSSI-distance prediction. Based on the experiments' results, the gateway's effective range was estimated at 21 m. That is the maximum distance at which the receiving beacon can have a Line of Sight (LoS) with the gateway and broadcast the detected transmitting beacons. Thus, one gateway covers a circular area with a 21 m diameter, and accordingly, that recommends the size of 42 m for the module. Besides, the maximum allowable distance of the transmitting beacon to receiving beacon was determined as 4.25 m, resulting in a sub-module with 4.25 m diagonals. Therefore, the module with the side of 30 m and the sub-model with the size of 3.00 m is proposed. Figure 2 illustrates the top view of the sub-module and the module.



a) A sub-module consisting of four transmitting beacons (orange circles).

b) A module consisting of a gateway (red circle) and 100 sub-modules.

Figure 2: Top view of the sub-module and the module

Since a BLE packet has a limitation of payload size, only three transmitting beacons with the highest RSSI values among all the detected transmitting beacons by the receiving beacon are reported back to the gateway. Since the records sometimes contain transmitting beacons that do not belong to a sub-module, they can be categorized based on the location of their reference

transmitting beacons in a module. This can help classify the records based on the concentration level of its transmitting beacons in the module. The more concentrated the transmitting beacons of a record are, the more trustable the records will be for the localization model to estimate the location of the target node. Thus, based on the level of concentration of the transmitting beacons in records, they are categorized as (i) Logical, i.e., a record whose three broadcasted transmitting beacons belong to the same sub-module, and the maximum allowable distances between the transmitting beacons are equal to or less than the diagonal length of the sub-module; (ii) Semi-logical, i.e., a record that fulfills the 'Logical' requirement for the pair of the two strongest transmitting beacons, and the scenarios but not the third transmitting beacon; and finally, (iii) Non-logical, i.e., a record that belongs to neither Logical nor Semi-logical record. Figure 3 depicts the four possible scenarios for Semi-logical records.

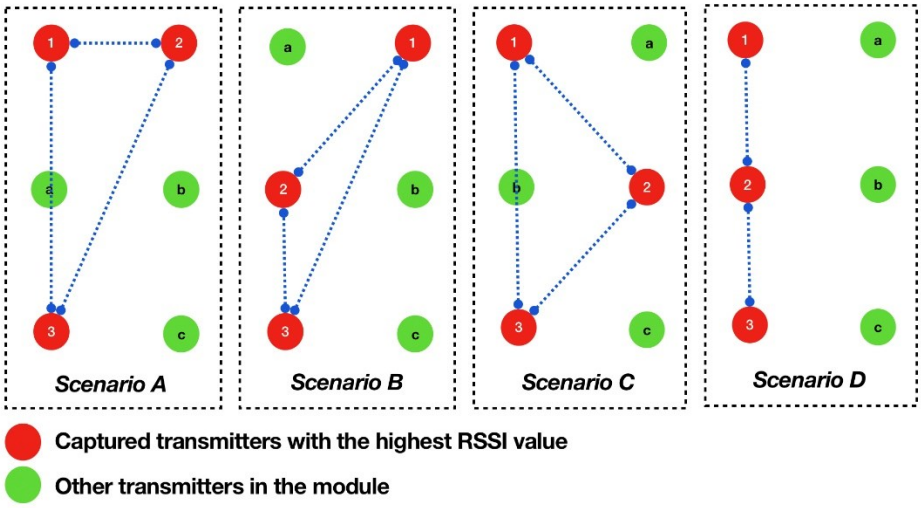


Figure 3 Possible scenarios for the Semi-logical records

When a receiving beacon is located in a sub-module, the transmitting beacons of that sub-module are expected to be the closest. In practice, however, sometimes receiving beacons capture stronger signals from transmitting beacons farther away than the closest ones, generating records with scattered transmitting beacons not necessarily belonging to the same sub-module. The result will be Semi- or Non-logical records, which must be reduced as much as possible, to avoid confusion for localization algorithms. A hardware-based solution to this issue is setting parameters for both transmitting and receiving beacons, dictating how far and how often the beacons can transmit and receive signals from one another. The related parameters include (i) transmission power determining how powerful the BLE signal is broadcasted, and it has a direct effect on the maximum range of the transmitting beacon's signal [66]; (ii) advertising interval, which determines the time period between the start of two consecutive advertisements from the

transmitting beacon [67]; (iii) scan interval controlling the time period between scans of the receiving beacon; and (iv) scan window, setting the duration of each scan for the receiving beacon to capture the advertisement packet from the transmitting beacons. In a dense network of sub-modules where the transmitting beacons are placed close to one another, interference of the transmitting beacons' signals can be reduced by decreasing the transmission power.

Moreover, improving the chance for the receiving beacon to detect its nearest transmitting beacons during its receiving period can reduce the number of Semi-logical records. This can be set by configuring how often the transmitting beacon broadcasts its advertising packet and how long the receiving beacon listens to the transmitting beacon's signals. The transmitting beacons must send signals in shorter periods than the scanning interval to ensure that at least one signal advertisement is captured during a scan interval [68]. Dividing the scan window by the advertising interval determines the number of chances for the receiving beacon to detect the closest transmitting beacons.

A set of experiments was conducted for different beacon configurations to understand the effect of configuration parameters on the system performance. The ratio of the Logical records (to all records) was considered the basis of the comparison among the settings. Five Transmission power levels (-16, -12, -8, -4, and 0 dBm) and 3 scan window durations (200, 400, and 600 ms) were tested for the experiment. To evaluate the effects of transmission power on the system's performance, the scan window, advertising, and scan intervals of the beacons were set to 600, 200, and 900 ms, respectively. Regarding the scan window effect, the transmitting beacon's transmission power was set to -8 dBm, and the advertising intervals and scan intervals were set to 200 and 900 ms, respectively. Table 4 compares the total and Logical records over various transmission powers and scan windows, respectively.

Table 4: Comparison of total, perfect and logical samples for various Transmission powers

No.	Configurations of the Beacons				Total Records	Logical Records
	(a) "Tweak" Configurations of the Beacons					
	Transmitting Beacon		Receiving Beacon			
	Transmission Power	Advertising Intervals	Scan Window	Scan Intervals		
1	0	200	600	900	2240	1497 (%67)
2	-4	200	600	900	2185	1477 (%68)
3	-8	200	600	900	2214	1633 (%74)
4	-12	200	600	900	2232	1732

						(%78)
5	-16	200	600	900	2274	1653 (%73)

No.	(b) "Tweak" Configurations of the Beacons				Total Records	Logical Records
	Transmitting Beacon		Receiving Beacon			
	Transmission Power	Advertising Intervals	Scan Window	Scan Intervals		
1	-8	200	600	900	2214	1633 (%74)
2	-8	200	400	900	2333	1766 (%76)
3	-8	200	200	900	1790	1106 (%62)

Table 4 (a) shows that configuring the transmitting beacon's transmission power can improve the system's performance by increasing the ratio of Logical records (to total records) from 67% to 78%. Also, except for -16 dBm, the ratio of Logical records went up as the transmission power of the transmitting beacons decreased. The transmission power of -12 and -8 dBm had the highest ratio of %78 and %76 accordingly. Since there is a high potential for signal interference on the construction site, stronger transmission powers are required to cut through the interference. Therefore, the transmission power of -8 dBm is more reasonable for the transmitting beacon deployed on the job sites. Moreover, it is clear from Table 4(b) that increasing the length of the scan window has a positive effect on the ratio of Logical records. It is found that by configuring the scan window size as twice as the advertising interval, the Logical records ratio can reach a high plateau of 77%.

Although the ratio of Logical records can be significantly increased by configuring the beacons (hardware solution), the remaining number of Semi-logical records generated by the system affects the system performance by decreasing the location sampling frequency. Hence, a novel algorithm (software solution) is developed to convert Semi-logical records to Logical ones. Figure 5 shows the processes involved in the record correction algorithm. In order for a Semi-logical record to be converted to a Logical one, the third broadcasted transmitting beacon should be substituted with the one which belongs to the same module of the other two transmitting beacons. The substitution of the third transmitting beacon is made by applying a set of processes on the Semi-logical records. Firstly, the algorithm calculates midpoints (X_{h1},Y_{h1}) and (X_{h2},Y_{h2}) of the lines connecting the third transmitting beacon (X_3,Y_3) to the other two (X_1,Y_1) and (X_2,Y_2) (see Figure 5). Then, the middle point between the previously found midpoints is calculated (X_m,Y_m). This point acts as a guide to specify the approximate area where the third transmitting beacon should

reasonably be located. The next step calculates distances between the guide point and transmitting beacons in the neighborhood (except those already included in the record). The transmitting beacon whose distance from the guide point is minimum is considered the correct third transmitting beacon in the record. In the final step, the correct transmitting beacon replaces the old one by keeping its RSSI value. The preserved RSSI value is associated with the old transmitting beacon, which is not necessarily the same as the RSSI of the replaced (correct) transmitting beacon. However, this can be used as the best approximation for the RSSI value of the replaced (correct) transmitting beacon. The algorithm's processes are depicted in Figure 4.

Input	:	Records = {user_id, $Distance_1, X_1, Y_1, Distance_2, X_2, Y_2, Distance_3, X_3, Y_3$, timestamp }//the collected records
Output	:	Records = {user_id, $Distance_1, X_1, Y_1, Distance_2, X_2, Y_2, Distance_3, X_3, Y_3$, timestamp }
Process 1	:	Define: //initial the mid-points of the lines connecting the strongest and second strongest transmitting beacons to the third transmitting beacon h_1 (x)= $\frac{(X_1+X_3)}{2}$, h_1 (y)= $\frac{(Y_1+Y_3)}{2}$ h_2 (x)= $\frac{(X_2+X_3)}{2}$, h_2 (y)= $\frac{(Y_2+Y_3)}{2}$
process 2	:	Calculate: //initial the mid-point between the points of h1 and h2 $m_x = \frac{(X_{h1}+X_{h2})}{2}$, $m_y = \frac{(Y_{h1}+Y_{h2})}{2}$
Process 3	:	Define: // initial the eligible transmitting beacons Tr_i I = (Tr_1 & Tr_2 & Tr_3), (I ∈ R) Calculate: // calculate the distance between m and all the transmitting beacons $d_i = \sqrt{(m_x - Tr_{i(x)})^2 + (m_y - Tr_{i(y)})^2}$ Judge: //determine the transmitting beacon whose distance is shorter If $d_i > d_1 > d_2 > \ldots > d_n$, (n ∈ N) Select Tr_i
Process 4	:	Replace: // substitute the third transmitting beacon with the newly selected transmitting beacon (Tr_i <=> Tr_3)

Figure 4: Semi-logical to Logical Record Converter

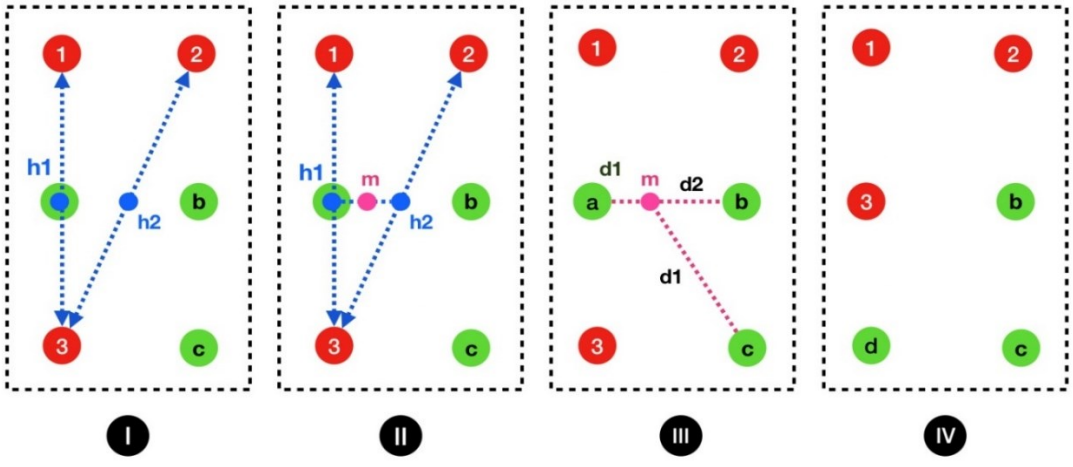


Figure 5: The processes in the records correction algorithm: (i) Calculating middle points between the third transmitting beacon and the other two transmitting beacons, (ii) Determining the guide point, (iii) calculating the distance between the guide point and the transmitting beacons, (iv) Predicted transmitting beacon replacement.

3.3) Real-Time Locating System (RTLS) Module

This section introduces the three models included in the developed RTLS: (i) RSSI-distance estimation model; (ii) Localization estimation model; and (iii) Localization post-processing model. Firstly, the RSSI-distance estimation model converts the RSSI values in each record to a length representing the estimated distance between the transmitting and receiving beacons. Once the locations of the transmitting beacons and the corresponding distances to the receiving beacon are determined, the localization model processes the data to estimate the receiving beacon's position (x,y) coordinates. The estimated distance acts as the circle's radius with its center at the transmitting beacon's location. The intersection centroid of the three circles helps predict the receiving beacon's location coordinates. Since the localization estimation model encounters various scenarios, a unique localization algorithm is developed for each arrangement. Finally, the estimated locations are post-processed to minimize distortions in the receiving beacon's location and improve the localization accuracy. The localization post-processing model includes two steps (i) shifting the worker's location to the strongest transmitting beacon and (ii) applying filtering techniques, such as Exponential Smoothing, Simple Moving Average, and Kalman filters.

3.3.1) RSSI-distance Prediction Model

The RSSI values captured from the three transmitting beacons must be reliably translated into physical distances for the Locating System to correctly identify the receiving beacon's location. The following parts explain the lab experiment completed to collect data and create RSSI-distance models for the developed system.

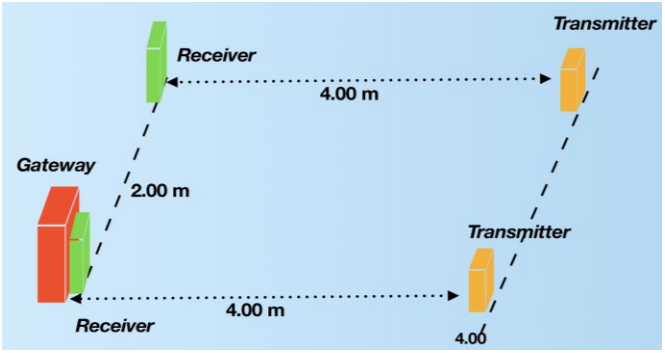
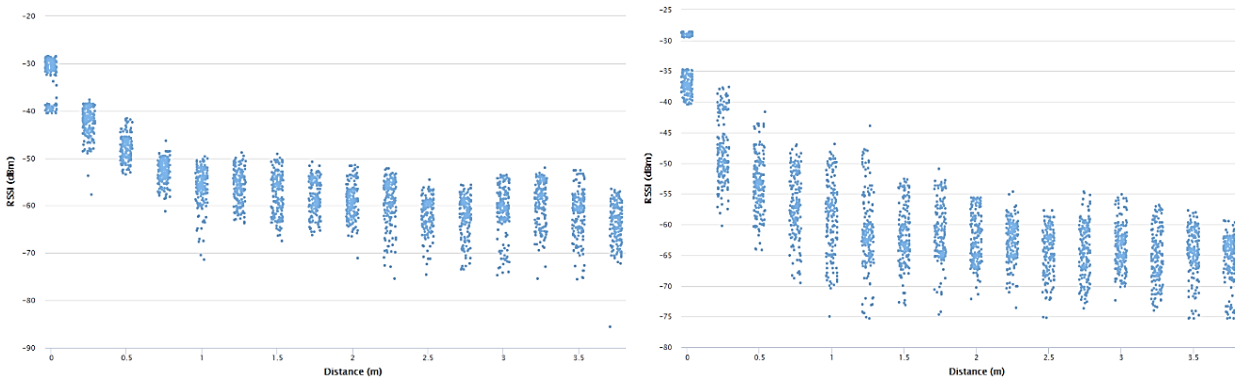


Figure 6: Placement of the devices for the experiments

In-lab experiments for developing the RSSI-distance relationship and evaluation of the localization system were conducted in the space of 9.00 m × 9.00 m X 3.20 m, as shown in Figure 1. This space provided an open space for testing and movable objects for creating different layouts and examining the effect of obstacles. The effects of signal reflections and noises on BLE signals caused by the furniture, equipment, and magnetic fields in the vicinity of the testbed were inevitable.

The receiver beacon was placed on seventeen reference points (stations) marked at 25 cm intervals on a straight line with a total length of four meters to ensure the exact position of the beacon while experimenting. The experiments were performed for four orthogonal orientations of the transmitting beacon with respect to the receiving beacon. Moreover, the experiments were repeated three times at each station to obtain a consistent dataset of RSSI values per each reference point. The receiving beacon was moved from the first station (distance from transmitting beacon = 0) to the seventeenth reference point (distance from transmitting beacon = 4.0 m). The staying time of the transmitting beacon at each reference point was one minute. In the interest of time, the same settings were implemented on two parallel straight lines, two meters away from one another.

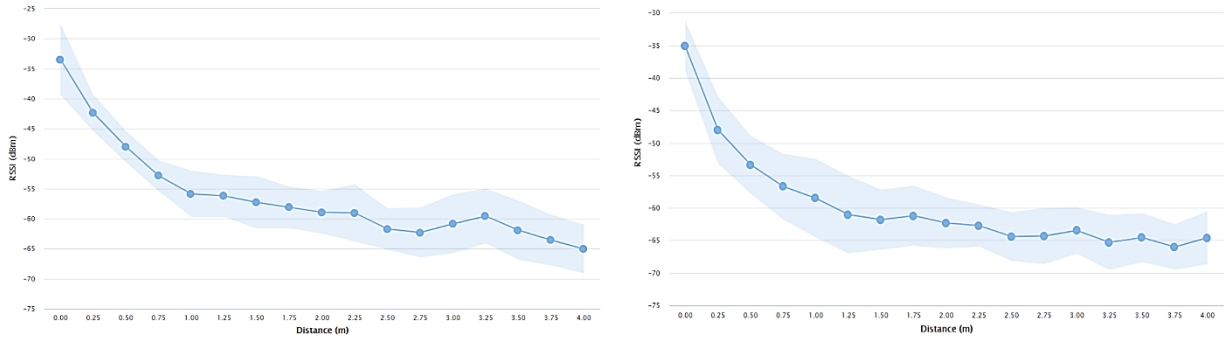
Two datasets were collected for the two parallel lines of the RSSI-distance experiment (which we refer to, as tests ‘a’ and ‘b’). Each dataset has three subsets, and each of them contains RSSI records for the four orientations of the transmitting beacon with respect to the receiving beacon at the seventeen reference points. The number of RSSI records per staying time period at each station (i.e. one minute) and the total number of records at each station considering all the orientations were 15 and 180 respectively. The total number of RSSI records for tests ‘a’ and ‘b’ were 2,955 and 2,959 data points respectively. The scatter plots of the RSSI records versus distance (between the transmitter and receiver) are provided for the two sets in Figure 2.



(a) Dataset ‘a’ (b) Dataset ‘b’

Figure 7: RSSI data collected in the experiment

The empirical attenuation relationship is depicted for tests ‘a’ and ‘b’ in Figure 3. As the distance between the transmitting and receiving beacons increases, the RSSI values decrease in all experiments up to a point then it reaches a low plateau. That point in tests ‘a’ and ‘b’ was at the 2.75 m and 3.25 m stations, respectively.

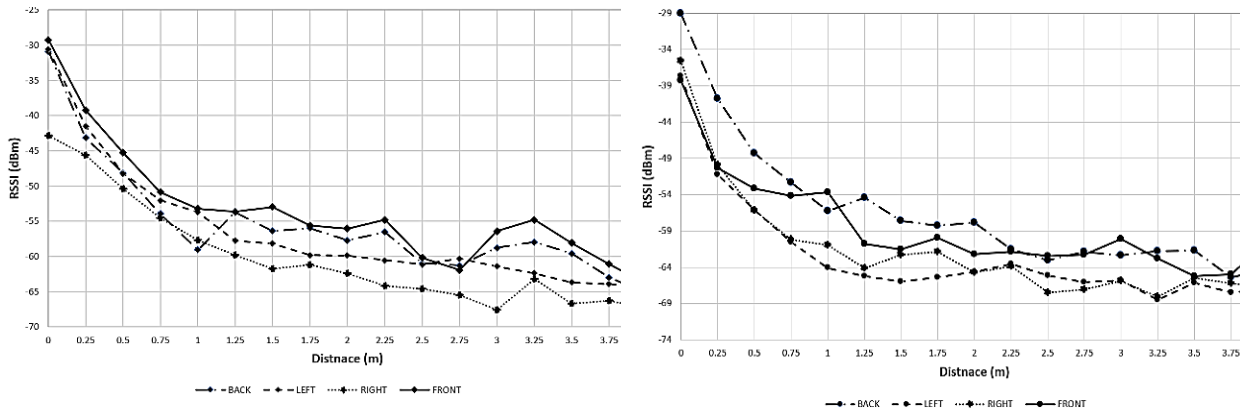


(a) Dataset ‘a’ (b) Dataset ‘b’

Figure 8: Mean and Standard deviation band plot for the experiment

The effect of transmitting the beacon’s orientation with respect to the receiving beacon and its impact on the RSSI was investigated in the experimental work. In Figure 4, the average RSSI records per distance over the four orientations are illustrated in different patterns of lines. As seen, at each distance, the receiving beacon seems to capture stronger RSSI values for the front and back orientations. Since the BLE chip inside the beacons (regardless of the beacon’s shape) has two major axes to transmit signals, the front and back orientations can be associated with one axis and the right and left ones to the other axis. This seems to cause the similarity of RSSI records between the pair of orientations per each axis and the difference between the orientation pairs of the opposite axes.

The receiving beacon was not able to receive the acceleration of the transmitting beacon, so the orientation shift of the transmitting beacon could not be determined. Besides, the beacons which are worn by workers can frequently change their orientation with respect to receivers on construction sites. Therefore, it is necessary to get enough RSSI records at different orientations of the transmitting beacon with respect to the receiving beacon to reach a more reliable RSSI-distance relationship on site.



(a) Dataset 'a' *(b)* Dataset 'b'

Figure 9: The average value of RSSI records per distance, separated based on the four orientations

In the model development, we examined the Path-loss propagation and machine learning models, looking for the most accurate one for the distance estimation model due to its direct impact on the reliability of the localization system.

The measured RSSI values in the same station are fluctuant with time due to the environmental noises. In order to remove the outlier RSSI values, a Gaussian filter which had been proposed by [69], was used in our study. The filter was set in the range of [μ − σ, μ + σ] of the RSSI records for each distance, and the measured values outside this range were ignored. Since studies have shown that the channel fading characteristics follow a lognormal distribution [33], RSSI-distance measurement generally uses the logarithmic distance path-loss model, which is formulated as:

$$RSSI = -10n * \lg\left(\frac{d}{d_0}\right) + A + X_\sigma \quad (1)$$

Where $RSSI$ is the Received Signal Strength Indication when the distance between receiving and transmitting nodes is d . Also, A is the RSSI value of the reference point with the known distance of d_0 from the transmitting node, which is captured by the receiving node. n is a path-loss coefficient related to the specific wireless transmission environment; the more obstacles in the test area, the larger the value of n will be. X_σ is a Gaussian-distribution random variable with a mean of 0 and variance $\sigma 2$. For the convenience of calculations, let d_0 = 1m and X_σ have a mean of zero, so the distance-loss model can be obtained as:

$$\overline{RSSI} = -10n * \log(d) + \overline{A} \quad (2)$$

where $\overline{A}$ is the average measured RSSI when the reference node is 1 m away from the transmitting node. To calculate the environmental parameter n, the following equation is used [22][70]:

$$n = \frac{A - \overline{RSSI}}{10*\log(d)} \tag{3}$$

Thus, the RSSI-distance relationship model can be obtained to predict the distance of the blind beacon through its RSSI value. In our study, to estimate the value of A, 312 RSSI records were collected when the distance was 1 m. As a result, A was estimated as -56.229 dBm. Regarding the n value, the average RSSI value of 16 different distances ranging from 0.25 m to 4.00 m with interval steps of 0.25 m were used. The value of n for the distances are shown in table 1. Hence, the average value of n is calculated as follows:

$$n = \sum_{i=1}^{16} n_{i*0.25} = 1.586 \tag{4}$$

Table 5: The value of n at different distances

Distance	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
$\boldsymbol{n}$	1.985	1.904	1.430	-	2.528	2.223	1.369	1.309
Distance	2.25	2.50	2.75	3.00	3.25	3.50	3.75	4.00
$\boldsymbol{n}$	1.444	1.618	1.408	1.195	1.243	1.224	1.407	1.501

Finally, the proposed RSSI-distance relationship is represented by Eq. (5)

$$RSSI = -15.86 \log(d) - 56.229 \tag{5}$$

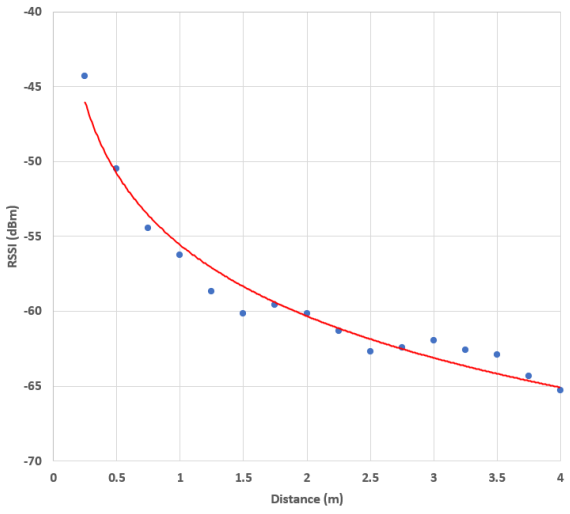


Figure 10: The average value of RSSI records per distance and RSSI-distance estimation model: red line represents the estimated RSSI-distance model, and the blue dots are the average RSSI values at each distance

In addition to the conventional method of estimating the values of A and n, the method of fitting a curve to the average RSSI value at different distances ($Avg.RSSI_i, d_i$), proposed by[69]. Among different curve fitting methods, Logarithmic used by [71] was deployed in our study (Zhu and Alsharari 2015). This model was developed based on the 5614 RSSI samples at the 16 test stations. The logarithmic regression was carried out by using the average of the RSSI values for each distance. Figure 5 demonstrates the average value of records per distance and the RSSI-distance estimation model. Using logarithmic curve fitting, the path loss model can be expressed by Eq. (6).

$$RSSI=-18.644 \log(d) - 55.573 \quad (6)$$

We trained machine learning models as alternatives to the path loss method for predicting the distance through the RSSI values. For distance prediction, this study tested several machine learning techniques, including Random Forest, Gradient Boosted Trees, Generalized Linear Regression Model, and KNN. After splitting the data into a training set (75%) and test set (25%), the hyperparameters for each model were finetuned, and finally, the best models from each technique were selected and compared.

Before training the model, two preprocessing steps were taken. Firstly, the outliers caused by environmental noise were identified based on their distance to their nearest RSSI records and were eliminated. RSSI records of each distance are ranked based on its distance to its 70 nearest neighbors. Based on trial and error, on the reference of the Random Forest (RF) model performance, the number of strongest outliers that were removed per each distance and from the training set is 12 and 204, respectively. Figure 6 shows the removed outliers as green dots. Secondly, the RSSI records were normalized by the z-transformation method. The average and standard deviation of the values were calculated, and the scaled value was calculated by Z = (x-Avg.) / SD. The average value and the standard deviation are 0 and 1. This normalization is necessary for some machine learning methods [72].

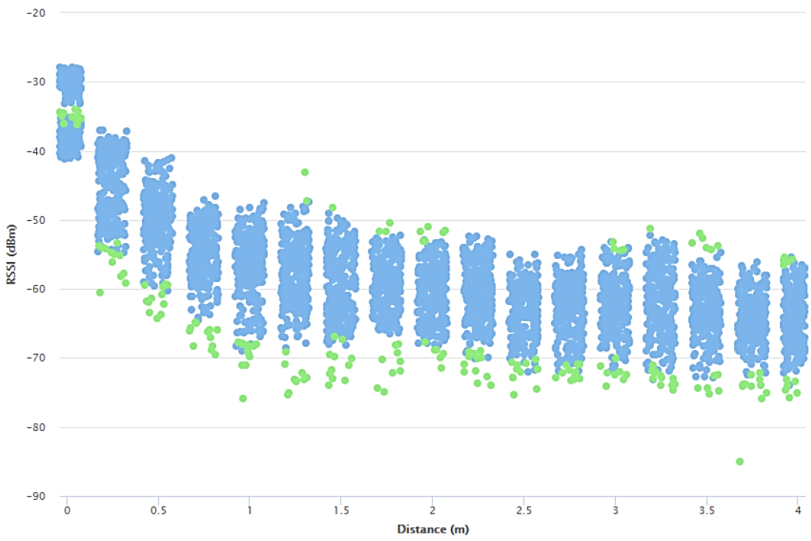


Figure 11: Scatter plot showing the removed outliers: the green dots are the removed outliers

Four machine learning models were trained in this study, and their performance was compared through their accuracy and error. They include Random Forest (RF), Gradient Boosting Decision Tree, Generalized Linear Regression (GLR), and k-Nearest Neighbours (kNN). RF is an ensemble learning method used for classification and regression. A random forest is an ensemble of multiple decision tree sets with the following modification: each tree node represents the best split for one specific attribute. Only a subset of specified attributes is considered for the splitting rule selection. After that, the erection of new nodes is repeated until the stopping criteria are met. The second used algorithm, gradient boosting decision tree, produces a prediction model in the form of an ensemble of decision trees models [73], built incrementally where each successive estimator gradually reduces the previous model's error. Building the model takes a longer time than the random forests since each tree has to be built based on the results of a prebuilt tree [74].

The third tested algorithm is GLM, which is an extension of the traditional linear regression models. The GLM is fitted by solving the maximum likelihood optimization problem. This model can accommodate many various types of target variables and covariate relationships. Lastly, the kNN is one of the most influential classification algorithms, trained on a set of labeled instances and making predictions based on the k labeled datapoints that are most similar to each unlabeled data point by using metrics such as Euclidean distance. We fine-tuned the model parameters for each model through trial and error and optimization to minimize the prediction model's errors.

The experiment was designed to compare varied techniques for the distance prediction model, which is a fundamental element in the proximity detection system. The prediction results are

discussed here for both Path-loss and machine learning models. Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Percentage Error (MPE) were considered as criteria to make a comparison between the performance of the models. It is noted that the records for 0 m distance between transmitter and receiver were removed from the calculation of MPE. Since the proposed model predicts the distance, MAE can better understand the actual error value. Error was defined as the distance between the estimate and actual coordinates in the experiment.

The logarithmic curve fitting model was associated with less distance prediction error than the conventional path-loss model. Table 3 shows the evaluation metrics for the two models. As seen, for the conventional path-loss model, the MAE, RMSE, and MPE were 1.176 m, 2.171 m, and -12.0 percent, respectively. Applying the logarithmic curve fitting model improved the accuracy of distance prediction. The MAE was reduced to 0.961 m, the RMSE was only 1.555 m, and MPE was improved to -9.9 percent. The results show that the error in the distance prediction model can be reduced by about 18 percent in terms of MAE by using a logarithmic curve fitting model.

Table 6: Statistics of the evaluation metrics (MAE, RMSE, and MPE) of the models

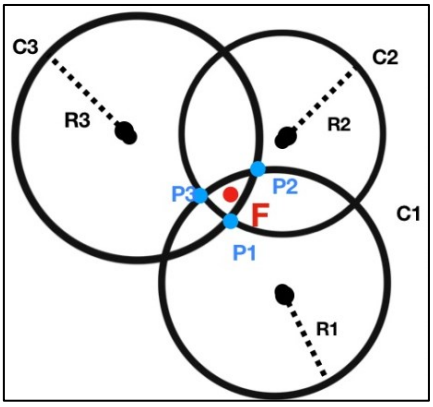
Model	Conventional Path-loss	Logarithmic curve fitting	Random Forest	Gradient Boosted Trees	Generalized Linear Model	KNN
MAE (m)	1.176	0.961	**0.641**	0.699	0.704	0.648
RMSE (m)	2.171	1.555	**0.822**	0.844	0.858	0.832
MPE (%)	-12.0	-9.9	**-25.1**	-34.7	-37.2	-18.7

As to machine learning models' performance, a 5-fold cross-validation was adapted. As seen in table 3, the Random Forest model outperformed all the models in terms of MAE and RMSE, as low as 0.641 and 0.822 m, respectively. By contrast, the Generalized Linear Model had the highest MAE, RMSE, and MPE of 0.704 m, 0.858 m, and -37.2, respectively, among the machine learning models. It can be seen that the Path-loss models had the lowest MPE in comparison with the machine learning models.

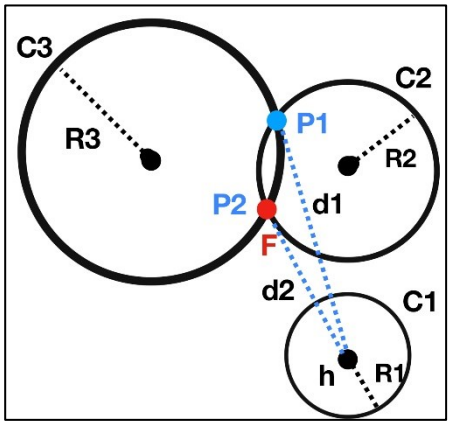
3.3.2) Localization Estimation Model

After estimating the distance between the receiving beacon and each transmitting beacon, the receiving beacon's location must be pinpointed. In this section, firstly, different scenarios created by the arrangement of the transmitting beacons and their estimated distance from the receiving beacon are explained. Then, the localization estimation model consisting of localization algorithms developed for each scenario will be presented. According to the prior literature, 'trilateration' and 'min-max' were selected as the localization techniques due to their acceptable

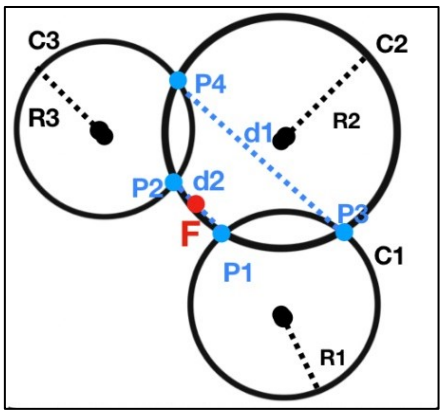
level of accuracy [54][75] however, they were advanced and upgraded. The trilateration technique determines the target's location by finding the intersection point of three circles representing distances between the target and reference nodes. However, mathematical trilateration is rarely feasible because there is, in practice, an area of possible locations rather than a single location point [32][54]. The localization estimation model can encounter various scenarios depending on the distances estimated between transmitting beacons and the receiving beacon. Hence, the arrangement of circles representing the estimated distance of the receiving beacon from transmitting beacons and coordinates of the transmitting beacons should be determined first. For each possible arrangement, the receiving beacon's location is calculated through a separate algorithm in the localization model. Given three estimated distances, i.e., r1, r2, and r3, between the transmitting beacons t1 through t3, and the target node (receiving beacon), three circles, i.e., C1, C2, and C3, can be drawn. Center points of these circles are the known positions of the reference transmitting beacons, and their radii are equal to the estimated distances between the target node and the transmitting beacons. Based on the estimated radii of the circles and the distance of their centers, the arrangement of each three pairs of transmitting beacons in a record should be investigated to determine whether two circles intersect. The combination of three pairs of transmitting beacons creates different scenarios that are explained in the following. In order to clarify the localization model, the scenarios with their corresponding algorithm to estimate the target node's location are discussed separately.



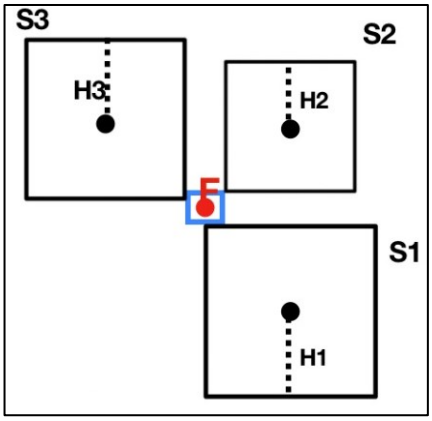
(a) Scenario (a)



(b) Scenario (b)



(c) Scenario (c)



(d) Scenario (d)

Figure 12: Scenarios of different arrangements for triangulation

Scenario (i) Three overlapping circles: As shown in Figure 12 (a), there is an area of overlap among the three circles. In this scenario, the target location is estimated as the centroid of the intersection area, created by the three points P1, P2, and P3 using the following formula:

$$F_x = \frac{P1_x + P2_x + P3_x}{3}$$
$$F_y = \frac{P1_y + P2_y + P3_y}{3} \quad (7)$$

Scenario (ii) Two overlapping circles and one isolated circle: When only two circles overlap in an area, and the third circle is isolated (Figure 12 (b)); after determining the intersection points P1 and P2, the distances (d1 and d2) from the center of the isolated transmitting beacon, i.e., h, to the two intersection points are calculated. Then, the final estimated location of the target is the point whose distance (to the isolated circle' center) is shorter.

Scenario (iii) Two circles overlapping the third, but not one another: Figure 12 (c) demonstrates the scenario that one circle (C2) intersects with the other two circles (C1, C3), but C1 and C2 themselves do not intersect. The intersection points P1 through P4 are calculated between the circles in this scenario. Then, the distances d1 and d2 between the intersection points of the separate circles (C1, C3) are calculated by equations (8) and (9). Finally, using equation (10), the midpoint F on the shorter distance (d1) is chosen as the final estimated location of the target beacon.

$$d1 = \sqrt{(P4_X - P3_X)^2 + (P4_Y - P3_Y)^2} \quad (8)$$

$$d2 = \sqrt{(P2_X - P1_X)^2 + (P2_Y - P1_Y)^2} \quad (9)$$

$$F_x = \frac{P1_x + P2_x}{2}$$
$$F_y = \frac{P1_y + P2_y}{2} \quad (10)$$

Scenario (iv) Three isolated circles: In rare cases, there are three short coverage areas whose corresponding circles do not intersect. Figure 12 (d) demonstrates the arrangement of the circles in this scenario. For this scenario, the min-max or the bounding-box method is deployed for the localization. This has a low computational complexity [76]. In this technique, the target beacon constructs a bounding box around each transmitting beacon, where the transmitting beacon is placed at the center, and the edge length of the bounding box is twice its estimated distance. The target beacon determines the intersection of the boxes, with boundary locations given by $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$ which are calculated from equations (11) through (14). Finally, the center point of this intersection box is considered as the estimated target location (x_{est} and y_{est}) which are calculated by equations (15) and (16) [75]. Since the circles do not intersect in this scenario, the min-max technique creates a hypothetical box whose edges are circumscribed by the edges of the transmitting beacons' bounding boxes.

$$x_{min} = max(x_1 - d_1, x_2 - d_2, x_3 - d_3) \quad (11)$$

$$x_{max} = min(x_1 + d_1, x_2 + d_2, x_3 + d_3) \quad (12)$$

$$y_{min} = max(y_1 - d_1, y_2 - d_2, y_3 - d_3) \quad (13)$$

$$y_{max} = min(y_1 + d_1, y_2 + d_2, y_3 + d_3) \quad (14)$$

$$F_x = \frac{(x_{min} + x_{max})}{2} \quad (15)$$

$$F_y = \frac{(y_{min} + y_{max})}{2} \quad (16)$$

3.3.3) Estimated Locations Post-Processing Model

Reflection and diffraction attributed to the presence of walls and floor (objects) within the indoor environment can produce multipath and fading effects, respectively. The multipath effects strongly affect the propagation of BLE signals and contribute to RSSI fluctuations [54]. This can cause distortions in the estimated distances between the receiving beacon and transmitting beacons, resulting in noise for the location data of a target node (receiving beacon) [77]. Therefore, the estimated locations are post-processed to minimize the effect of distortions on the location of the target node. Two post-processing steps are performed in this study.

(i) Shifting the Estimated Location to the Strongest Transmitting beacon - The localization model considers the transmitting node whose signal is received with the highest RSSI value as the closest and most reliable transmitting beacon for the localization. Accordingly, the estimated location of the target node will be shifted toward the location of that transmitting beacon. Firstly, the associated pair-wise weights between the estimated distances of the target node from the first and second transmitting beacons received with the highest RSSI value are computed. It is noted that the RSSI-distance model occasionally predicts the distance between the receiving beacon and the transmitting beacon with the highest RSSI value longer than the one with the second-highest RSSI value. Given the two distances d1 and d2 as the estimated distances between the worker and the two transmitting beacons, sorted ascendingly, the weights are calculated as follows:

$$w = \frac{d_2}{d_1} \tag{17}$$

Where d_1 is the estimated distance between the target node and the closest transmitting beacon and d_2 is the estimated distance between the target node and the second closest transmitting beacon. Then, the following equations are used to adjust and estimate the final location of the target node for all the scenarios.

$$x_{fin} = \frac{location\_X + X1 * w}{1 + w} \tag{18}$$

$$y_{fin} = \frac{location\_Y + Y1 * w}{1 + w} \tag{19}$$

Where $location\_X$ and $location\_Y$ are the coordinates of the estimated location by the localization model, and $X1$ and $Y1$ are the coordinates of the location of the strongest transmitting beacon.

(ii) Filtering Techniques - After shifting the estimated location to the strongest transmitting beacon, three filtering techniques are applied to smooth the locations' calculations. They include Simple Moving Average (SMA), Exponential Smoothing (ES), and Kalman Filtering (KF). SMA is the most common filtering algorithm implemented in localization tasks. Despite its simplicity, it reduces random noise while retaining a sharp step response [78]. A filtered record is calculated as the average of values within a symmetric window of size N around that record, where N is the pre-defined size of the window of the MA filtering [39]. It is given as:

$$record_{MA} = \frac{\sum_{i=1}^{N} record_i}{N} \tag{20}$$

The second technique used in this study, ES, is one of the most popular and easy-to-use filtering methods [79]. The basic formula of exponential smoothing is [80]:

$$S_t = \alpha x_t + (1 - \alpha) S_{t-1} \tag{21}$$

where S_t is the smoothed location at time t, x_t is the actual observation location at time t, S_{t-1} is the smooth location at time $t - 1$, and α is the smoothing constant with a domain between 0 and 1. The accuracy of the exponential smoothing model mainly depends on the selection of α.

The third tested technique was KF, which uses noisy observed data and data with other inconsistencies to estimate unknown states using a mathematical model. KF filter is a standard optimal estimation algorithm based on Bayesian filter theory [81]. It has two stages, prediction and update (correction). Firstly, the filter predicts the next state at time t based on the current state at a time (t-1) before the next state is made. The second stage computes a gain value G(t) based on the prior noise estimate. It then updates the posterior state and system noise estimations using the latest state observation and current gain value [39]. In our study, the target node (the location of a worker) is described by four parameters (state variables), which can be written in a state vector as follows:

$$\boldsymbol{x} = [\mathrm{x}, \mathrm{x\_vel}, \mathrm{y}, \mathrm{y\_vel}] \tag{22}$$

where x and y are the cartesian coordinates of the target node and x_vel and x_vel are the velocities in the x and y directions. The (x,y) coordinates are set as the starting location of the target, and the velocity is set to 0 as the initial value for our experiment [54]. The dynamics for each of our states in the current record "t" as a function of states in the previous record " t-1 " are given as the following equations:

$$\begin{aligned} &x(t) = x(t-1) + dt * x\_vel(t-1) \\ &x\_vel(t) = x\_vel(t-1) \\ &y(t) = y(t-1) + dt * y\_vel(t-1) \\ &y\_vel(t) = y\_vel(t-1) \end{aligned} \tag{23}$$

where dt represents the change in time (time-step), and it is assumed that (x, y) coordinates are updated based on current location and velocity. The formulas can be rewritten in matrix format as:

$$x(t) = F \times x(t-1) \tag{24}$$

where

$$\boldsymbol{F} = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

Besides, the state covariance P indicates how much the state variables influence each other's values, determining the dependency of the system on the initial state values. The values of the initial matrix P, i.e. (n), indicate the level of uncertainty that is considered for the estimated state (in this case, the location estimated by RTLS):

$$\boldsymbol{P} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 0 & n \end{bmatrix} \tag{26}$$

The measurement matrix H relates the measurements to the states' variables. z is the measurement vector, and X is the states' variables vector. The H function is used to obtain from the state variables vector x the values (in this case, the location) that are being measured [54]:

$$\boldsymbol{z} = [x, y] \tag{27}$$

$$\mathrm{z} = \mathrm{Hx} \tag{28}$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{29}$$

In this study, two variants of post-processing were examined for smoothing the estimated locations, namely 'mild' and 'intense.' Table 7 shows the parameter values used for the post-processing techniques at each level.

Table 7: The parameters changed for the post-processing levels

Post-Processing Technique	Filtering Type	Mild Post-Processing	Intense Post-Processing
Shifting the estimated location	–	w (weight) = 0.2 * W	w (weight) = 1 * w
Filtering technique	Exponential smoothing	α (alpha) = 0.3	α (alpha) = 0.8
	Simple moving average	neighbors = 20 records	neighbors = 6 records
	Kalman	observation covariance = (5 x previous estimated value)	observation covariance = (2000 x previous estimated value)

After localizing the worker's location, the location data is mapped on geometrical contextual information of job sites to check the presence of workers in workspaces defined in a BIM model. With that goal, workspaces and zones were created as 2-dimensional spaces by a user in Revit software [82]. Then, Dynamo scripting language was used to retrieve the identification of the workspaces, including their element ID and (x,y) coordinates of vertices composing the bounding edges. Lastly, Wooff's algorithm determined whether a worker's estimated location lies inside the boundaries of workspaces. It uses the property that the summation of all angles created between lines connecting the location of the worker point P and the ith and (i + 1)th vertices of a given workspace equals zero if point P lies outside the workspace and equals to ± 2π if point P lies

inside the workspace [83]. The angle Θ_i between the ith and (i + 1)th vertices is calculated as follows:

$$\Theta_i = \tan^{-1} \frac{(V_y - P_y)_{i+1}}{(V_x - P_x)_{i+1}} + \tan^{-1} \frac{(V_y - P_y)_i}{(V_x - P_x)_i} \quad (30)$$

Where (P_x, P_y) are the location coordinates of the worker and (V_x, V_y) are the location coordinates of the vertex of the workspace. Figure 13 shows the placement of a worker inside a typical workspace.

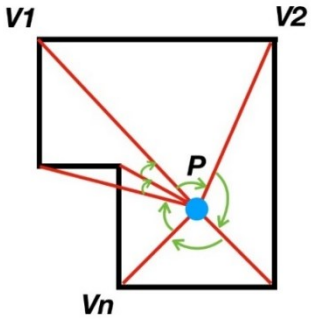


Figure 13: Wooff's Algorithm sums the angles created between lines connecting the location of the worker point P and the ith and (i + 1)th vertices of a workspace

3.4) Body Orientation Detection Module

The BLE technology faces challenges related to signal attenuation caused by the human body, which can shadow or even entirely obscure the BLE packets [84]. In other words, the human body prevents the receiving beacon from capturing signals from the transmitting beacons located behind. The body orientation detection module is developed to utilize such signal blockage to detect approximate workers' body orientation, using RTLS location data and RSSI values between chest-mounted receiving beacons and reference transmitting beacons. This section introduces the body orientation detection module that uses signal blockage by a human body to identify an approximate worker's body orientation on job sites. In fact, a receiving beacon mounted on the worker’s chest cannot capture signals from the reference transmitting beacons located behind the worker [84][85]. To this end, in-lab experiments were carried out to collect data between the chest-mounted receiving beacon and reference transmitting beacons for eight ordinal orientations. Then, an ANN model was trained to predict the body orientation, using signal data and geometrical relationships between the beacons. The details of the data collection and model development processes are provided in this section.

3.4.1) Data Collection for Body Orientation Detection Module

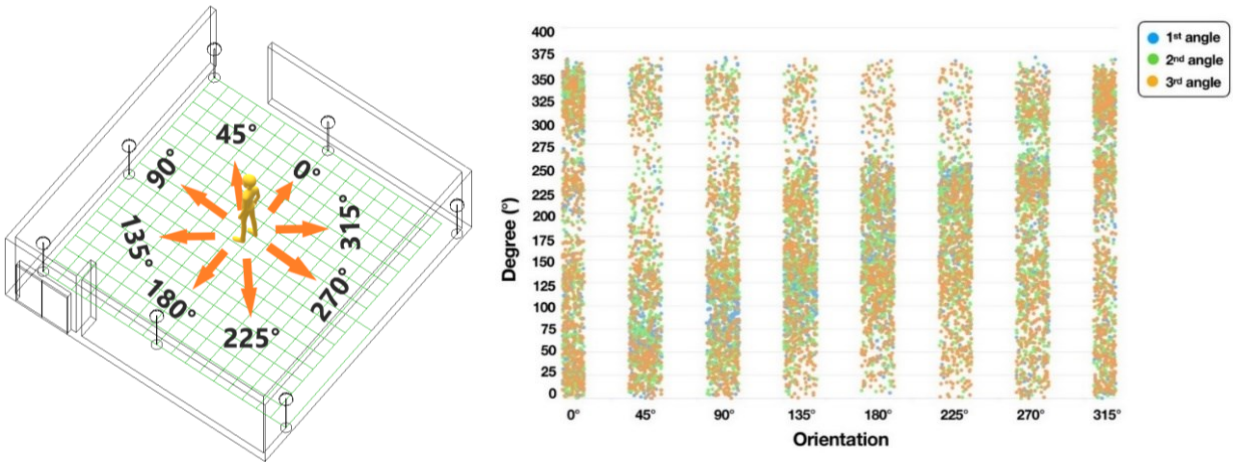
The angles between the worker's estimated location by the RTLS with respect to the location of the detected reference transmitting beacons are calculated. The angle is measured counter-clockwise and ranges from 0° to 360°. Figure 14 (a) shows the eight ordinal orientations defined in the laboratory. Given two points of $p1$ and $p2$ in 2-dimensional space, the angle from $p1$ to $p2$ is calculated by the following formula:

$$\vec{p}_1 \cdot \vec{p}_2 = |\vec{p}_1| \cdot |\vec{p}_2| \cdot \cos\ \theta \tag{31}$$

$$\theta = \cos^{-1}\left(\frac{x_1 x_2 + y_1 y_2}{\sqrt{(x_1^2 + y_1^2) \cdot (x_2^2 + y_2^2)}}\right) \tag{32}$$

Where (x_1, y_1) and (x_2, y_2) are the position coordinates of the worker and the reference transmitting beacon, respectively, and θ is the calculated angle from the worker to the reference transmitting beacon. Furthermore, the feature engineering process was performed on the data to create new input from the existing features. The feature engineered attributes include (i) distance of the worker from the reference transmitting beacons detected by the chest-mounted receiving beacon; (ii) distance of the reference transmitting beacons detected by the chest-mounted receiving beacon from one another; (iii) presence of the worker inside a hypothetical triangle whose vertices are the location of the reference transmitting beacons detected by the chest-mounted receiving beacon. It is noted that the reference transmitting beacons detected by the chest-mounted receiving beacon are not expected to exactly match the ones detected by the hardhat-mounted receiving beacon that are used for RTLS.

The data collection process was carried out in indoor environments with a testbed size of 8.00 m × 8.00 m, where nine reference transmitting beacons were installed hanging from the ceiling. A participant stood and walked on stations located 0.50 (m) apart from one another in the testbed area. The chest-mounted receiving beacon collected RSSI records from the reference transmitting beacons while the participant stood and turned 360° (in 45° intervals) in the counter-clockwise direction around the vertical axis. For each orientation, the participant stood and walked on the stations for a time period of roughly one (hour), and the number of collected records for each orientation was 1,416. Figure 14 (b) shows the scatter plot of the angle between the detected reference transmitting beacons from the worker’s location for the eight orientations during the experiment.



a) Eight ordinal orientations defined in the laboratory

b) Scatterplot of the angle between the detected reference transmitting beacons from the worker's location

Figure 14: Experimental setup for the data collection

3.4.2) Model Architecture and Training for Body Orientation Detection Module

ANNs are machine-learning algorithms based on a multi-layer perceptron and have been deployed for engineering problems. They are composed of input, hidden, and output layers that modify the weight between each layer to reduce the error between the actual and predicted values [86][87]. Generally, an ANN model that contains at least two hidden layers qualifies as a DNN model [87]. In this study, the optimal number of hidden nodes and layers is selected based on the best prediction performance on the test set. The input layer of the model has 13 features (including (i) two attributes for the angle of the chest-mounted receiving beacon from the two strongest reference transmitting beacons detected by the chest-mounted receiving beacon; (ii) two attributes for the RSSI value of the two strongest reference transmitting beacons detected by the chest-mounted receiving beacon; (iii) three attributes for the distance of the worker from the reference transmitting beacons detected by the chest-mounted receiving beacon; (ii) three attributes for a distance of the reference transmitting beacons detected by the chest-mounted receiving beacon from one another; (iii) one attribute for the presence of the worker inside a hypothetical triangle whose vertices are the location of the reference transmitting beacons detected by the chest-mounted receiving beacon. The features are normalized to increase the efficiency of the DNN model training. Since the eight body orientations have the same probability of being selected, the angle features follow a uniform distribution, and accordingly, they were normalized using min-max feature scaling. The following formula is used for the normalization so that the maximum and minimum values would be one and zero, respectively [72]:

$$E_{norm} = \frac{E - V_{min}}{V_{max} - V_{min}} \quad (33)$$

Where E_{norm} is the normalized element, E is the value of the corresponding element, V_{min} and V_{max} are the minimum and maximum values in the signal vector, respectively. The z-transformation method was used to normalize the rest of the features by removing the mean and scaling to unit variance[88]:

$$z = (x - u)/s \quad (34)$$

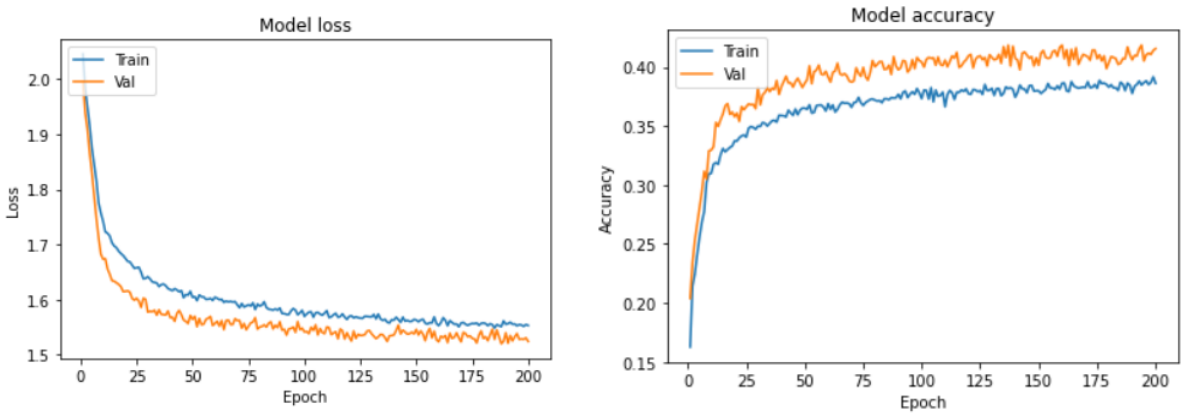
Where u is the mean of the records, and s is the standard deviation of the records. As per the Neural Network's hidden layers, the first and second hidden layers include 32 and 64 neural nodes, respectively, activated using a rectified linear unit (ReLU) activation function [89]. ReLU is the commonly used activation function in DNN and is faster than other functions [29]. For an input x, ReLU function $f(x)$ is as follows:

$$f(x) = \begin{cases} 0, \text{ if } x < 0 \\ x, \text{ if } x \geq 0 \end{cases} \quad (35)$$

Finally, the softmax function is deployed as an output layer to detect the body orientation of workers. For an input x, softmax function computes the probability that x belongs to an orientation c_k by the following formula [49]:

$$p(y = c_k \mid x; P) = \frac{e^{P_{c_k}^T x}}{\sum_{c_i=1}^{n} e^{P_{c_i}^T x}} \quad (36)$$

Where n is the number of classes (i.e., eight angular classes), $e^{P_{c_k}^T x}$ is the standard exponential function that is applied to each element of the input vector, and $\sum_{c_i=1}^{n} e^{P_{c_i}^T x}$ is the normalization term that ensures the sum of the function's output values equals one and ranges from zero to one. The number of hidden layers and neural nodes is determined by fine-tuning the model on the test set. The learning process is repeated 200 times (epochs) to prevent an occurrence that the optimal weight cannot be found. The calculated values of coefficient variation of the root mean square error (CVRMSE) were saved during the iteration. The model weights with the lowest CVRMSE were determined after each iteration. Figure 15 shows the training and validation losses during each epoch.



(a) Loss curve *(b) Accuracy curve*

Figure 15: Loss and accuracy curves for the trained convolutional neural network model

It is clear From Figure 15 that the training and validation losses follow the same decreasing curve, and over-fitting is not an issue for the trained model. The CNN model achieved an accuracy of 40% after 100 epochs, and the loss curve stabilized after the 200 epochs to a value close to 1.55. It is clear from Figure 15 that the model achieved low accuracy on the test and validation sets and cannot effectively capture the complexity of the data. This can be mainly due to the high variance of the detected transmitting beacons by the chest-mounted receiving beacons, as depicted in Figure 14 (b). The DNN model was trained using 80 percent of the collected data with a batch size of 4 and was tested using 20 percent of the data. Since the softmax layer assigns decimal probabilities to each label [90], the orientation with the highest probability is selected as the predicted orientation.

3.5) Productivity State Detection Module

This section introduces the productivity state detection module that is developed to identify the productivity states of workers. The productivity state of workers facilitates the calculation of workers' productivity rate by detecting the worker's actual work input. In this study, the productivity states and terminologies introduced by Moselhi et al. (2013) are used, which are as follows: direct work, support work, and delay [91]. It is noted that in some previous studies, the term "value-adding work" is used as "direct work", and the term "non-value-adding work" is used as "non-value-adding work" [12]. Since the productivity state cannot be determined once the worker is moving, "travel" is added as the fourth state to separate records that are detected as walking. In addition, the "travel" state can also be used for path planning, visualization of congested paths, and safety management [11][12]. The productivity states are defined as follows: (i) direct work, i.e., any state that involves movements directly leading to completion of the activity (e.g., painting

a wall with a paint roller); (ii) support work, e.g., any state that involves movements leading to getting or moving materials and tools, receiving instructions, and getting and fixing equipment (e.g., mixing material paint in a container); (iii) delay, e.g., any state that involves movements leading to unplanned breaks, staying on the workspace without performing any work, moving around without doing any activities related to the job, and waiting which is the time when workers are available, but no work can be performed.; and travel, e.g., any state that a participant is moving between workspaces. In this study, the productivity state is considered direct work once a worker performs the main construction task(s) whereas, it is considered support work once the worker performs the secondary construction task(s). The workers' body movements for the direct work and support work states vary based on each crew's main and secondary task(s), whereas all workers have the same body movements for the waking and delay states. The productivity state detection module comprises six models, and each is individually trained for a specific worker. Every model consists of (i) a frequency image generator which segments the tri-axial accelerometer data from the receiving beacons and converts them into frequency images; and (ii) a productivity state detection model, which is a two-dimensional Convolutional Neural Network (CNN) trained to detect the workers' productivity state by taking the generated frequency images as input. This section provides the data collection process, pre-processing, and training procedures for classifier models.

3.5.1) Data Collection for Productivity State Detection Module

One volunteer participated in the data collection process and was instructed to perform the main tasks involved in three repetitive construction activities, such as painting, plastering, and masonry. The performed tasks include laying bricks, placing mortar, mixing material, plastering, painting, traveling, and idling. The collected data for its main construction task(s) is labeled as a direct work state to train each worker model. The collected data for the construction tasks that are physically distinct from the main tasks is labeled a support work state for each model. Table 8 describes the construction tasks, and Table 9 provides a statistic of the data used to train the workers' models.

Table 8: Description of the collected productivity states and construction activities

Construction Task	Tools	Explanation
Travel	-	The worker walks at a regular pace.
Delay	-	It represents non-productive work, such as chatting, resting, etc.
Laying Brick	Free-hand performing	The participant handles bricks and tries to place them in a straight line.
Placing Mortar	Brick trowel	The participant puts mortar on the bricks and tries to level it with a brick trowel.

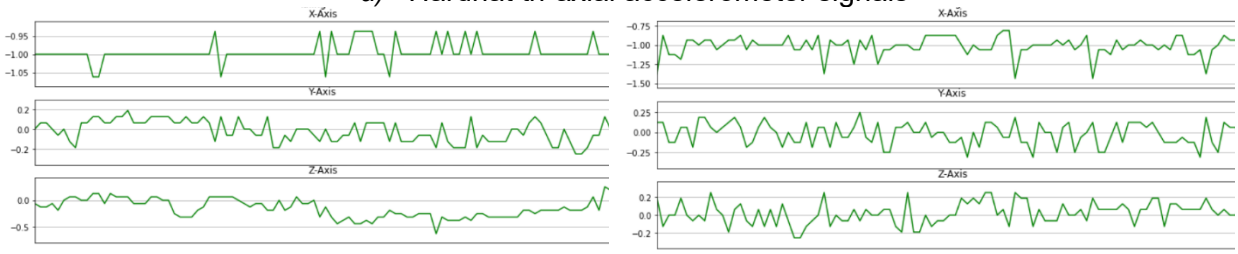
Mixing Material	Shovel	The participant mixes material, such as mortar and plaster on the ground
Painting	Paint roller	The participant applies paint on a wall surface by a roller and tries to apply it only vertically.
Plastering	Plastering trowel	The participant levels a plaster on a surface with a trowel.

Table 9: Statistics of the number of collected data for training the models

Worker	Direct work	Support work	Travel	Delay
Painter	2255 *(2255 applying paint)*	4510 *(1504 mixing material+ 1504 placing mortar + 1504 laying bricks)*	1342	1342
Plasterer	2255 *(2255 applying paint)*	2255 *(752 mixing material+ 752 placing mortar + 751 laying bricks)*	1342	1342
Plastering Helper	1342 *(1342 mixing material)*	671 *(224 painting+ 224 plastering+ 223 laying bricks)*	1342	1342
Mason	1342 *(671 placing mortar + 671 laying bricks)*	1342 *(671 painting + 671 plastering)*	1342	1342
Masonry Helper	1342 *(671 mixing material + 671 laying bricks)*	1342 *(671 painting + 671 plastering)*	1342	1342
Runner	2215 *(2215 laying bricks)*	671 *(671 painting)*	2215	671

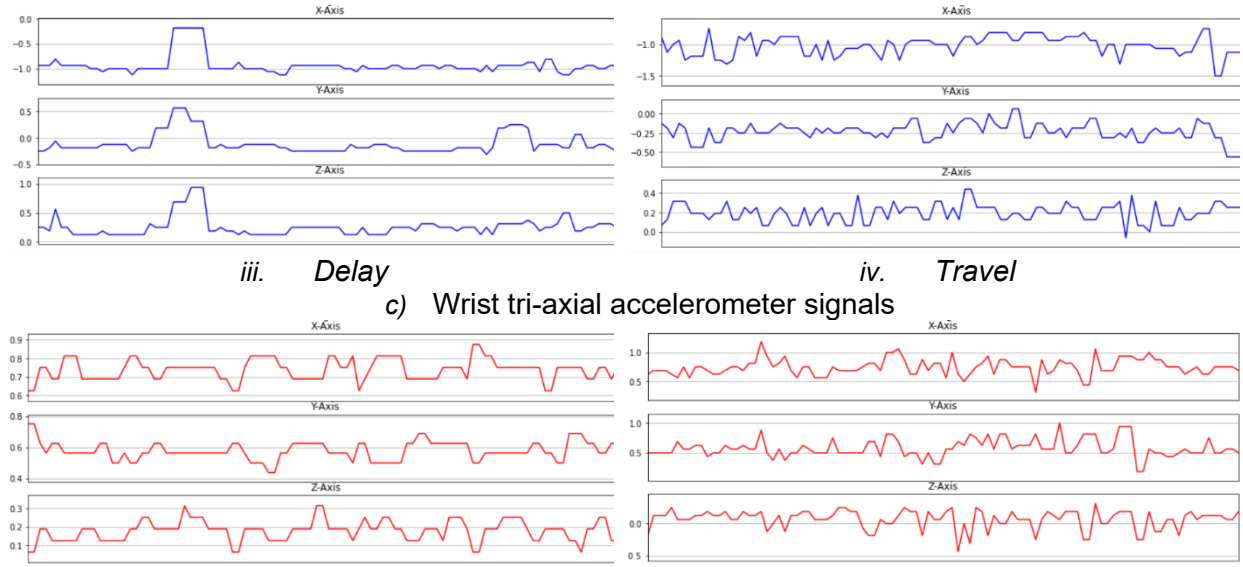
The collected data consists of acceleration values in the x, y, and z-axis, captured from the accelerometer embedded in the hardhat- and chest- and writs-mounted receiving beacons. The mean sampling rate for the collected acceleration data is one record per 1.6 seconds with a standard deviation of 0.7 seconds. Figure 16 shows the 100 consecutive tri-axial acceleration records for the two productivity states of travel and delay.

a) Hardhat tri-axial accelerometer signals



i. Delay *ii. Travel*

b) Chest tri-axial accelerometer signals



iii. Delay *iv. Travel*

c) Wrist tri-axial accelerometer signals

v. Delay *vi. Travel*

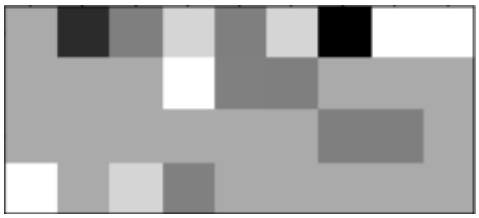
Figure 16: Illustration of the frequency-based activity images. The blue, red, and green lines represent the acceleration values of the beacons mounted on the hardhat, chest, and wrist, respectively.

After collecting data, the activity image generator creates frequency activity images from the raw tri-axial accelerometer signals of the three receiving beacons. Firstly, the acceleration change data is normalized using the z-transformation technique. Secondly, the segmentation technique divides the data into fixed window segments of four records with no inter-window gaps. The defined window size is found to be sufficient since it characterizes the proportion of time in which workers perform the construction tasks. The segments were stored in a two-dimensional image (matrix) with the size of N x M, where N is the time step (window size), and M is the number of multivariate time series (the three tri-axial acceleration data). Each image's grey shade color pixel is determined by its corresponding value in the matrix and ranges from -4.24 to 4.74. Figure 17 (a) illustrates a sample of the generated frequency images.

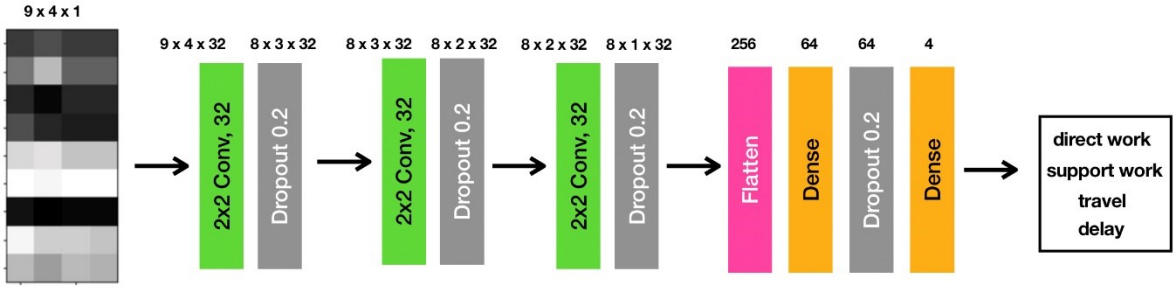
3.5.2) Model Architecture and Training for Productivity State Detection Module

In total, six separate CNN models are trained to detect the productivity state of the six workers. The trained models have the same architecture, but their training data are different from one another (as explained in section 4.3.1.). CNN is a deep neural network widely used for analyzing imaging data, and it is found to be more efficient than hand-crafted feature classification techniques for activity detection problems [92][93]. Since frequency images are low-resolution with less natural texture information, they require relatively fewer convolutional layers for activity classification problems compared to typical images [94]. Although a separate model is trained for

each worker, the architecture of the trained models for all the workers is identical. The input layer is the 9 x 4 pixel grayscale image (one channel) saved from the activity image generator. The first hidden layer of the model has 16 filters (kernels) with a size of (2 x 2), and it is followed by another two layers, each with 32 filters of the same kind. The filter matrix is slid over the image by one pixel (stride = 1), and for every position, element-wise multiplication (between the image and filter matrices) is computed. Then, the multiplication outputs are added to calculate the final value that produces a feature map. The feature map is then flattened in the next layer to create a single vector and passed through one fully connected layer with 32 neural nodes, followed by a dropout layer. Finally, a dense layer (softmax activation) with eight neural nodes is used for construction activity detection. This layer generates probabilities for each activity, and the maximum probability in the vector is selected as the image class. The number of hidden layers and neural nodes is determined by fine-tuning the models on the test set. ADAM optimizer with a learning rate of 0.001 is deployed to train the six models since it has good performance in deep neural network learning [94][95]. The proposed CNN network architecture for the model is shown in Figure 17 (b).



a) Sample of the frequency images obtained from the tri-axial accelerometer signals



b) Structure of the trained activity detection CNN model

Figure 17: Structure of the trained productivity state detection model

The models are trained using 90 percent of the generated images that are randomly selected, and the models are tested us the remaining 10 percent of the images. The training data is sent through the models, and an associated error is calculated based on the models' classification results. Accordingly, the models repeatedly refine the weights until all training data are used. Dropout [96] is set to 0.2 for the convolutional layers and the fully connected layer to prevent over-fitting in the models. The loss function is set to minimize the categorical cross-entropy, using the

Adam optimizer [95] with a learning rate of 0.001 [97], and the training is performed with a batch size of 32 and 50 epochs.

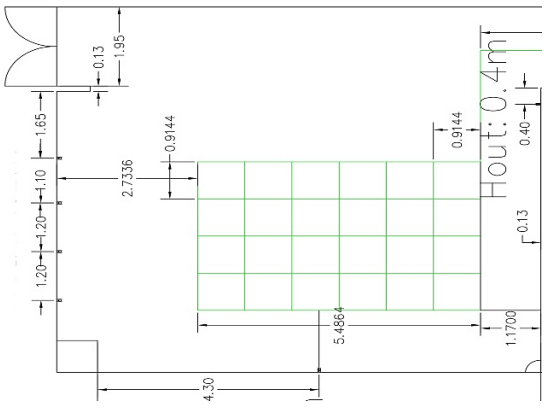
3.6) Test-bed Environment and System Setup

This section provides the experimental procedures and results of the system modules, including RTLS, body orientation detection module, and productivity state detection module. The modules' performance was tested separately in a controlled condition without distractions. Then, the worker monitoring system was evaluated in a simulated construction site, where participants performed construction operations designed to mimic the actual construction scenarios.

3.6.1) Real-Time Locating System (RTLS) Module

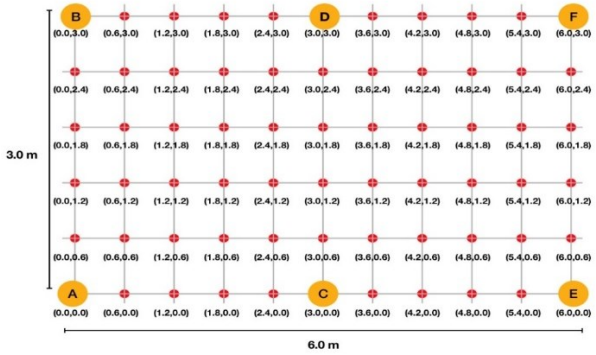
This section provides details of the experimental study and analysis of the proposed RTLS and describes the test-bed environment, system setup, and trained models and algorithms' performance. The experimental analysis focused on the potential factors that can affect the RTLS's performance, including (i) the post-processing (smoothing) intensity level, (ii) the post-processing techniques, and (iii) the location of the receiving beacon on the workers.

An in-lab experiment for analyzing the performance of the developed Locating System was conducted in a 9.00 (m) × 8.60 (m) area, referred to later as test-bed, where computers and electronic devices, and magnetic fields were present. The lab environment provided an open space for testing and creating layouts to help simulate the effect of obstacles on the job site. According to the study by Kalla et al. (2016), WiFi signals can cause failure in the connectivity between BLE devices and negatively affect the data transmission [98]. Since the gateway detected more than 50 wireless stations in the area used for the experiments, the effects of signal reflections and noise on BLE signals caused by the WiFi and objects in the vicinity of the test-bed have been inevitable. Figure 18 provides an overview of the test-bed.

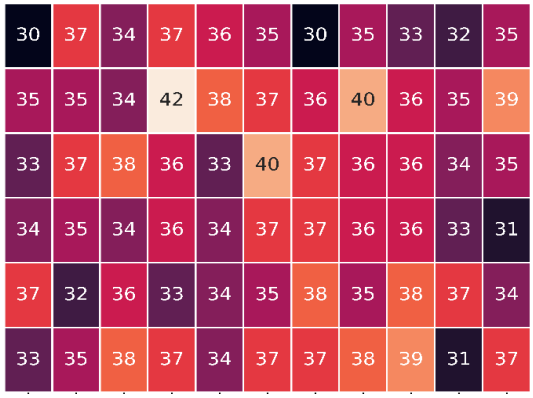


(a) Construction Automation Lab layout

(b) The layout of the test-bed and the grid



(c) Placement plan of the transmitting beacons and the reference points: The orange circles represent the transmitting beacons, and red circles represent the stations.

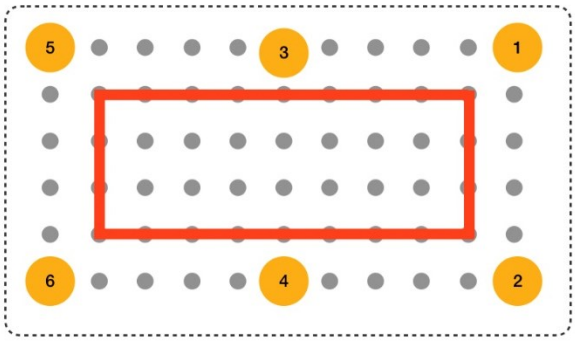


(d) Heat map of the number of records received by the target node at each station

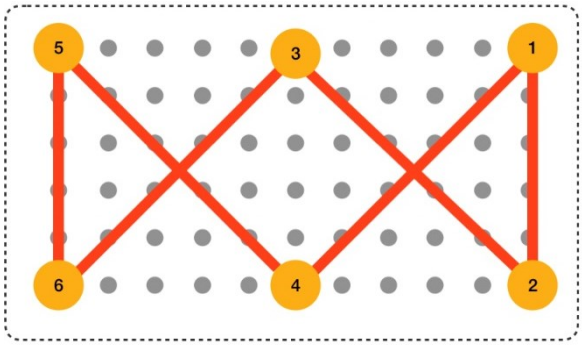
Figure 18: View of the in-lab test-bed and Placement plan of the devices and the reference points

Two sub-modules, consisting of six transmitting beacons, were deployed in the experiment. The receiving beacon, placed on top of a construction helmet, acted as the target node. The experimental methodology comprises two scenarios where the target node is static and dynamic to obtain a reliable system performance evaluation. For the static scenario, 66 reference gridpoints (stations) located 0.60 m apart from one another were marked in the test-bed (Figure 18) to ensure the target node's exact location while analyzing the system's localization accuracy. The number of records that the receiving beacon captured from the transmitting beacons at each station was around 35, and in total, 2,344 records were collected. Figure 18 shows the plan of the transmitting beacons and the reference grid points, and a heat map of the number of records per station is provided in Figure 18 (d).

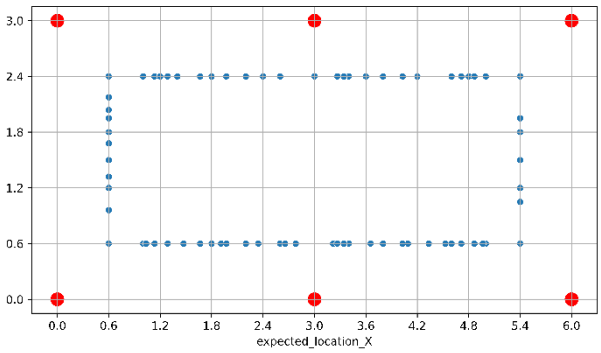
As for the dynamic scenario, the receiving beacon captured signals while closed-loop trajectory paths were walked continuously by the target node, as illustrated in Figure 19. Two trajectory patterns were considered, and for each of them, the experiment was repeated three times with various speeds to obtain a reliable estimation of the system performance when the target node is dynamic.



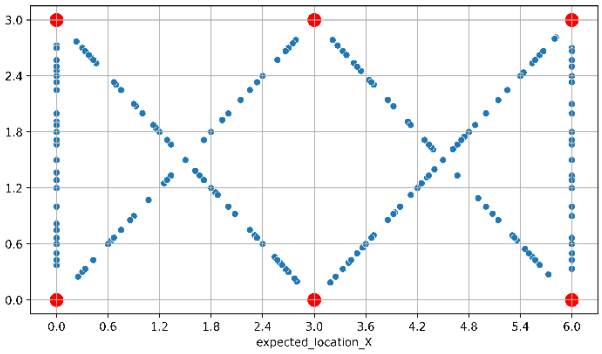
(a) Trajectory pattern (I)



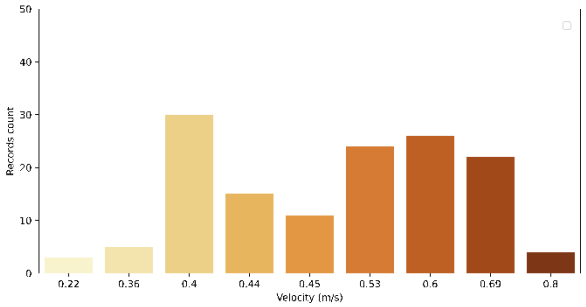
(b) Trajectory pattern (II)



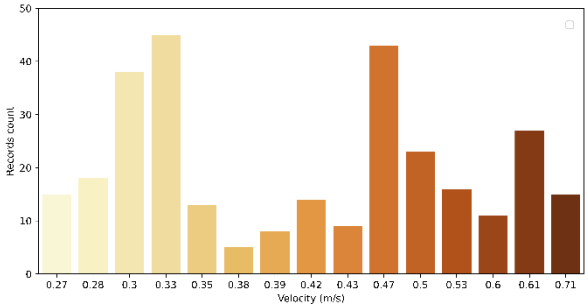
(c) Actual locations of the records for the trajectory pattern (I)



(d) Actual locations of the records for the trajectory pattern (II)



(e) Speed distribution for the trajectory pattern (I)



(f) Speed distribution for the trajectory pattern (II)

Figure 19: Trajectory patterns of the target and its actual locations on the trajectorys' paths and the speed distribution that it traveled

Since the number of turnovers included in the trajectory pattern (I) is less than that of the pattern (II), the target could travel pattern (I) with a higher speed. The number of records generated was 140 and 301 during the trajectory patterns (I) and (II), respectively. Figure 19 (c-d) illustrates the actual records generated during the trajectory patterns, and the speed distribution of the target during the trajectory patterns are shown in Figure 19 (e-f).

3.6.2) Productivity State and Body Orientation Detection Modules

The body orientation and productivity state detection modules were tested in a simulated construction site. Participants performed construction operations that were designed to mimic the actual construction scenarios. This section presents the experimental procedures and results. Three construction activities were considered to be performed by three crews, including painting, plastering, and masonry. The painting crew consisted of one worker (the painter), whereas the plastering and masonry crews were composed of two (the plasterer and the plastering helper) and three workers (the mason, the masonry helper, and the runner). A specific number of construction tasks were defined and assigned to previously instructed participants to perform for each worker. Figure 20 shows the main construction tasks performed by each worker.

Mixing material the paint with a brusher

Staining a paint roller with paint

Painting with a paint roller

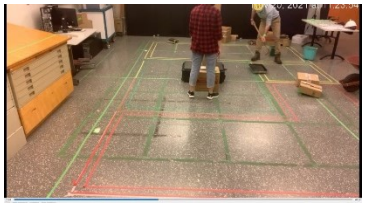
Sticking/removing types from surface

(a) Painter

Sticking/removing types from the surface

Plastering the surface

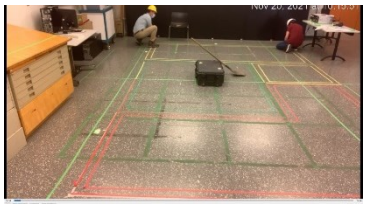
(b) Plasterer



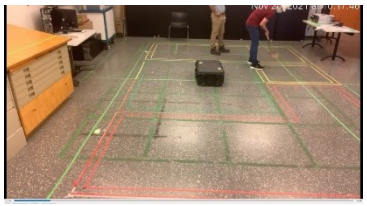
Making plaster

Waiting for plasterer

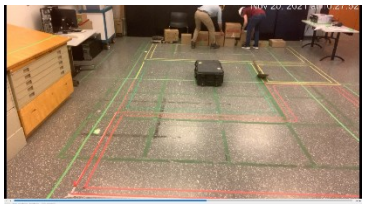
(c) Plastering helper



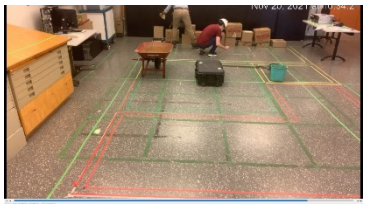
Marking the wall layout with chalks



Waiting for the helper

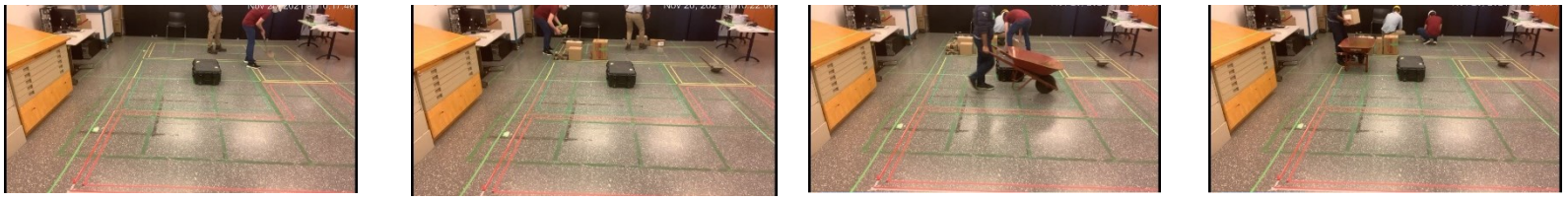


Laying bricks



Placing mortar on bricks

(d) Mason



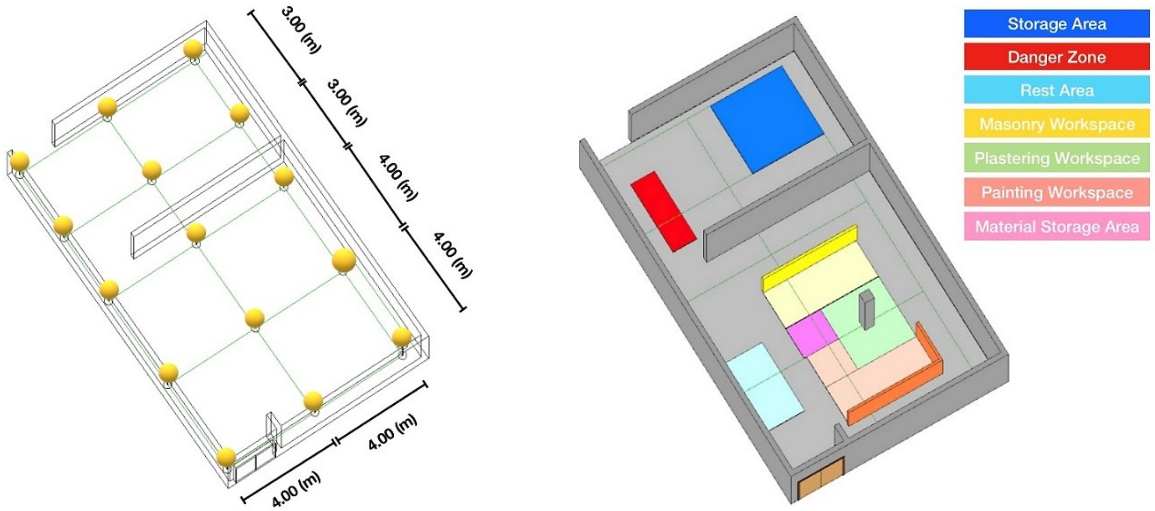
Making mortar *Laying bricks* *Carrying wheelbarrow* *Handling bricks*

(e) Masonry helper ***(f) Runner***

Figure 20: Simulated construction tasks performed by each crew

Two rounds of experiments were performed to analyze the system's performance. In the first round, called "single-crew" experiments, each crew was recorded performing its constriction activities separately from the other crews. While in the second round of experiments named "full-operation" experiments, the three crews (totally six workers) worked simultaneously to assess the potential effects of multiple sensors and the subsequent overlaps and interruptions on the system's performance. Each round of the experiments was repeated at least twice to obtain a consistent system evaluation. The experiments were also repeated by different participants playing the roles of various workers to investigate the effects of different working styles on the system's classification performance.

The experiments were conducted in a laboratory environment with a testbed of 14.0m × 8.0m, where computers, electronic devices, and magnetic fields were present. The lab environment provided an open space for testing, the objects referred to above created a layout that simulates the effect of obstacles on the job site. The experimental layout for the simulated construction site consisted of a painting workspace, plastering workspace, masonry workspace, danger zone, material storage area, and storage area. Furthermore, four video cameras were installed on the experimental testbed to record the workers' activities' ground truth. Figure 21 shows a schematic illustration of the site layout and the reference transmitting beacons placement.



a) System Infrastructure Placement *b) Simulated Construction Site Layout*

Figure 21: Simulated Construction Site Layout and Infrastructure Placement

3.7) System Performance

This section provides the system performance of the developed system's modules, including the real-time locating system (RTLS), body orientation detection, and productivity state detection.

3.7.1) Real-Time Locating System (RTLS)

The system localization accuracy is examined against the effects of the post-processing techniques/intensity, filtering techniques, localization, and placement of the receiving beacon on different human body parts. The distance error between the ground truth location $p = (x_{\text{real}}, y_{\text{real}})$ and the estimated location $\hat{p} = (x_{\text{calc}}, y_{\text{calc}})$ is computed using the Mean Absolute Error (MAE):

$$\text{Error} = \sqrt{(x_{\text{calc}} - x_{\text{real}})^2 + (y_{\text{calc}} - y_{\text{real}})^2} \quad (37)$$

where x_{calc} and y_{calc} are the coordinates of the target's estimated location, and x_{Real} and y_{Real} are the actual coordinates of the target.

Firstly, different intensity levels of post-processing on the localization accuracy for static and dynamic targets are compared. In order to show the effectiveness of post-processing on the estimated locations, the system was also tested on the raw location data, i.e., without shifting the estimated locations and applying the filtering techniques. Table 10 shows the impacts of the different strength levels of post-processing on the localization accuracy for both experiment scenarios.

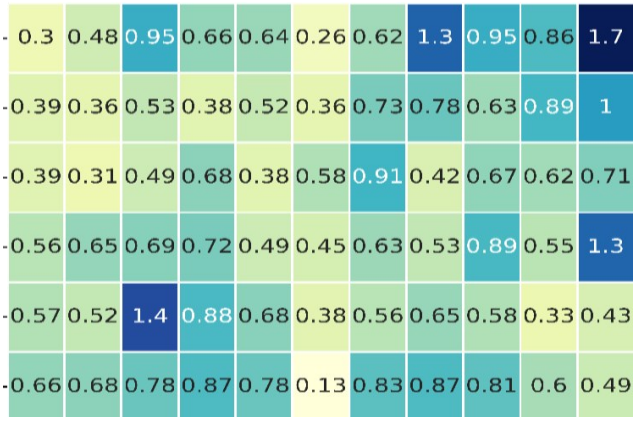
Table 10: Localization accuracy for the experiment scenarios with various post-processing methods

Filtering Technique	***Applying "Intense" smoothing in post-processing***						***Applying "Mild" smoothing in post-processing***					
	Static		Dynamic				Static		Dynamic			
			Trajectory pattern (I)		Trajectory pattern (II)				Trajectory pattern (I)		Trajectory pattern (II)	
	MAE	*SD*	*MAE*	*SD*	*MAE*	*SD*	*MAE*	*SD*	*MAE*	*SD*	*MAE*	*SD*
Kalman	0.64 (m)	0.43 (m)	1.85 (m)	0.84 (m)	1.50 (m)	0.99 (m)	0.78 (m)	0.56 (m)	0.66 (m)	0.55 (m)	0.51 (m)	0.43 (m)
Moving Average	0.65 (m)	0.44 (m)	2.21 (m)	1.05 (m)	1.81 (m)	1.05 (m)	0.73 (m)	0.46 (m)	0.72 (m)	0.48 (m)	0.56 (m)	0.49 (m)
Exponential Smoothing	0.75 (m)	0.51 (m)	1.05 (m)	0.80 (m)	1.57 (m)	1.07 (m)	0.90 (m)	0.69 (m)	0.99 (m)	1.02 (m)	0.89 (m)	1.03 (m)
Raw Locations	1.04 (m)	0.82 (m)	1.18 (m)	1.12 (m)	1.03 (m)	0.93 (m)	1.04 (m)	0.82 (m)	1.18 (m)	1.12 (m)	1.03 (m)	0.93 (m)

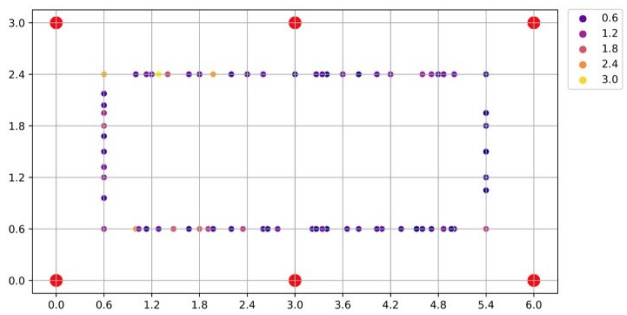
As shown in Table 10, the least effective filtering technique could improve the accuracy of the raw estimated locations by around 28 percent through applying an "intense" smoothing, and the mean error can be reduced to as low as 0.64 m using the Kalman filter for the static test scenario. In sharp contrast, applying the Kalman filter's intense post-processing increased the mean error by 178 percent. Since the target does not move (for a short time) in a static scenario, the filtering techniques could leverage the previous records generated with a similar location, resulting in the minimized effect of noisy estimated locations by applying an intense smoothing. However, the target movement in the dynamic test scenario had a significant negative impact on the effectiveness of applying the intense filtering. It is evident from Table 10 that applying mild post-processing on the raw estimated locations for the dynamic test scenario could reduce the mean error from 1.18 m to 0.66 m and from 1.03 m to 0.51 m for the trajectory patterns (I) and (II), respectively. In the best scenario, the "Mild" strength level of post-processing achieved a mean error and SD of 0.51 m and 0.43 m, respectively, for the dynamic test scenario. Besides, the mean localization error increased by about 15 percent by applying the mild smoothing post-processing in the static test scenario. Therefore, applying intense post-processing might result in better system performance for applications where the target maintains a static state such as asset tracking. However, considering a mild level is beneficial in the application where the target is dynamic, including worker localization.

Furthermore, the consistency of a target's estimated locations in the coverage area is analyzed in dynamic and static test scenarios to comprehend the precision of the system's performance. In this analysis, the mild and intense levels of post-processing using the Kalman filter were applied

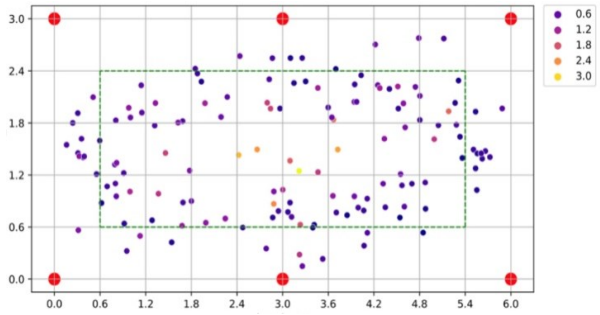
for the dynamic and static test scenarios, respectively. As per the static test scenario, a heatmap was produced to show the localization error of the static test experiment (under intense Kalman). The 66 centers gridpoints in the heatmap denote the test stations in the test bed. Regarding the dynamic test scenario, scatter plots of the distance error of the target's estimated locations in both trajectory patterns were created. Figure 22 (a) shows the heatmap of MAE of the estimated locations at each station and the. Scatter plots showing the system's precision in the coverage area are also depicted in Figure 22 (b).



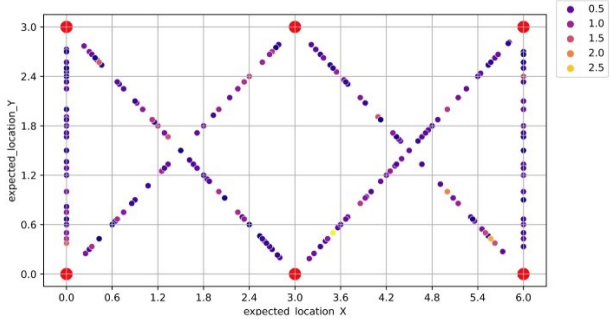
(a) Static test scenario – Heatmap of MAE of the estimated locations at each station



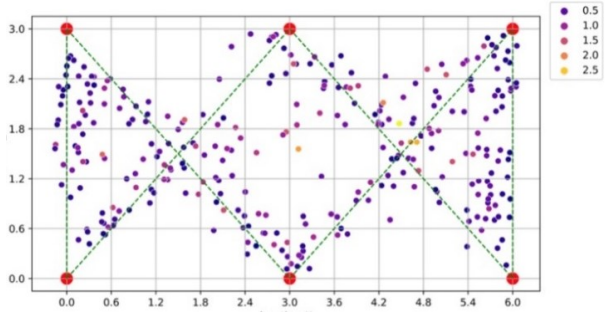
(i) *Actual locations of the target in the trajectory pattern (I)*



(ii) *Actual locations of the target in the trajectory pattern (I)*



(iii) *Actual locations of the target in the trajectory pattern (II)*



(iv) *Error of the estimated locations in the trajectory pattern (II)*

(b) Dynamic test scenarios – Scatter plots showing the system's precision in the coverage area

Figure 22: Performance of RTLS in the static and dynamic scenarios

As seen in Figure 22 (a), except for four test stations, the MAE of the rest of the stations is equal to or less than 1.00 (m). The results are shown in Figure 22 (b-ii & iv) conclude that the errors are uniformly distributed in the test-bed except for the areas around the turnovers.

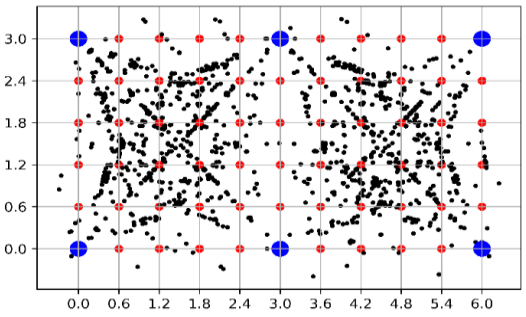
Secondly, to assess the sensitivity of the localization accuracy to the post-processing techniques, the raw estimated locations were compared against post-processed estimated results; i.e., estimated locations shifted towards the closest transmitting beacon; filtered estimated locations, and filtered shifted locations. According to the earlier results, intense and mild post-processing were applied for the static and dynamic test scenarios, respectively, to achieve the best results for each test scenario. Table 11 shows the statistics of the localization error for both test scenarios.

Table 11: Statistics of localization error for the raw and post-processed data

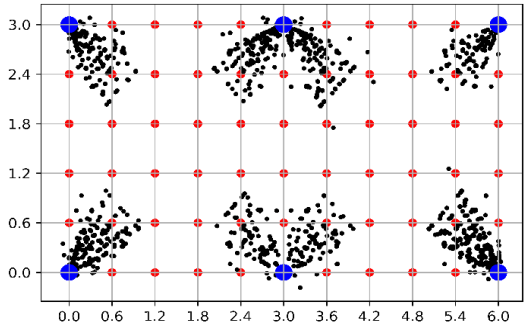
Target State	**Metrics**	**Raw Locations**	**Post Processing**		
			Shifting to the Strongest Transmitting beacon	*Filtering on the Raw Locations*	*Filtering on the Shifted Locations*
Static	MAE	1.03 (m)	1.04 (m)	0.84 (m)	0.65 (m)
	SD Error	0.76 (m)	0.81 (m)	0.46 (m)	0.44 (m)
Dynamic	MAE	1.08 (m)	1.00 (m)	0.66 (m)	0.56 (m)
	SD Error	1.00 (m)	1.04 (m)	0.50 (m)	0.48 (m)

As seen in Table 11, the “shifting to the strongest transmitting beacon” technique does not increase the system localization accuracy in the first place in the two test scenarios; however, its combination with the filtering techniques could reduce the mean localization error to 0.65 m and 0.56 m for the static and dynamic test scenarios, respectively. These numbers were 0.84 m and 0.66 m when the filtering techniques were applied to the raw locations in the static and dynamic test scenarios. Figure 23 shows the distribution of the estimated locations of the target in the static and dynamic test scenarios.

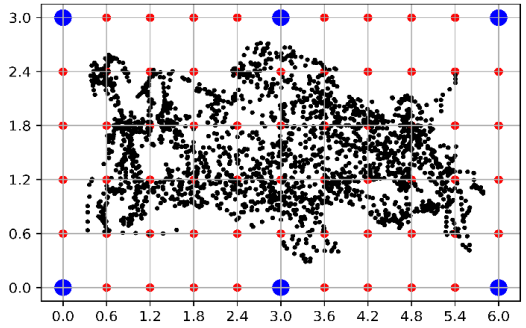
(a) Static Test Scenario



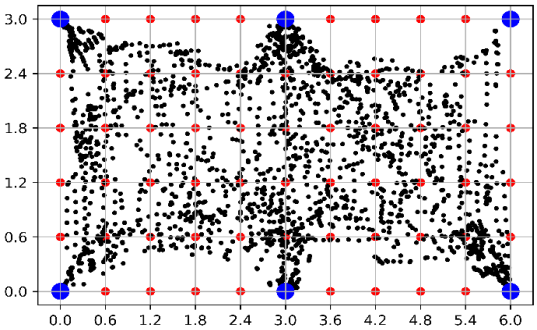
(i) Raw Locations



(ii) Shifting to the Strongest Transmitting beacon

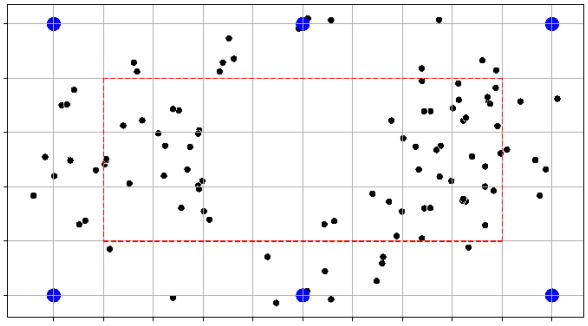


(iii) Filtering on the Raw Locations

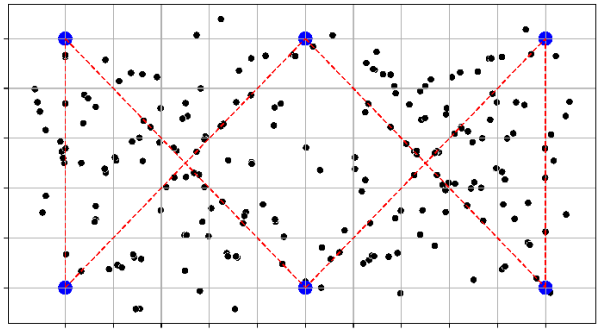


(iv) Filtering on the Shifted Locations

(b) Dynamic Test Scenario

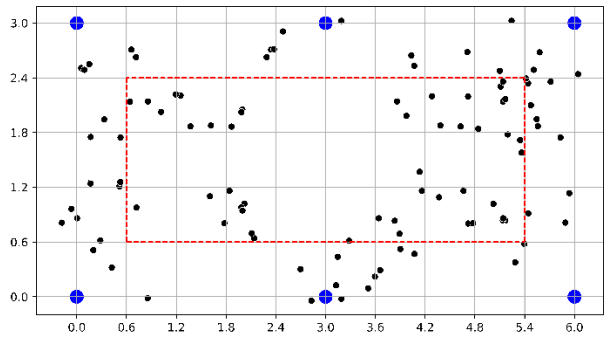


Trajectory pattern (I)

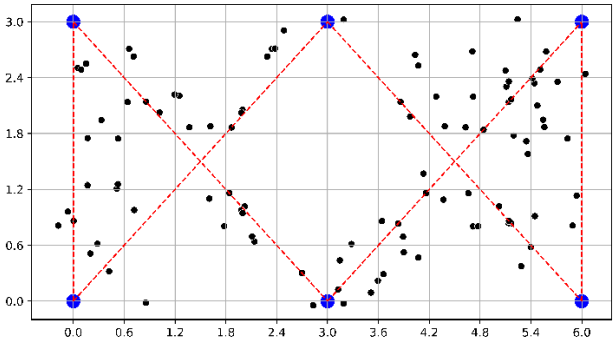


Trajectory pattern (II)

***(c)** Raw Estimated Locations*

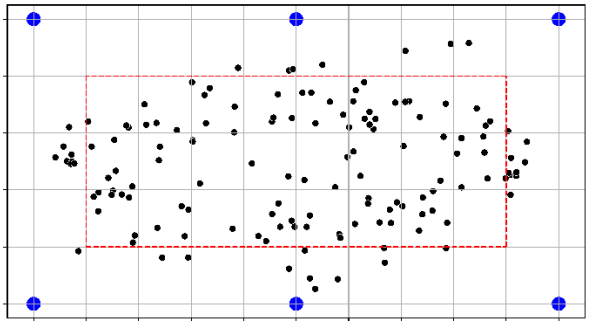


Trajectory pattern (I)

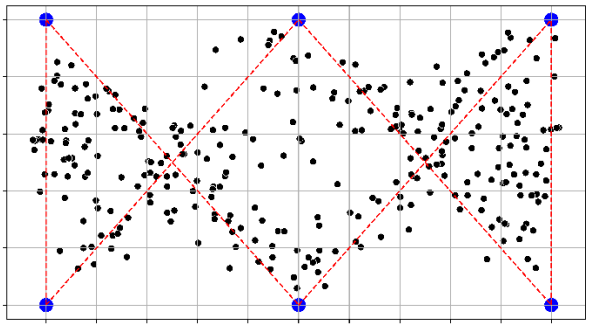


Trajectory pattern (II)

***(d)** Shifting to the Strongest Transmitting beacon*

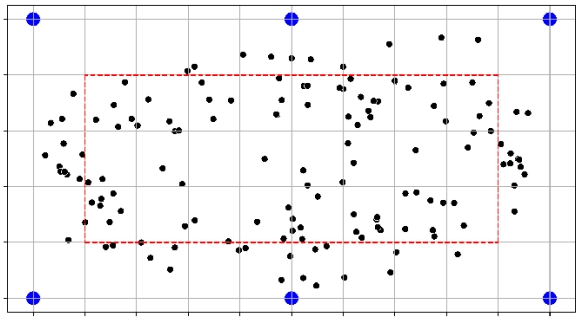


Trajectory pattern (I)

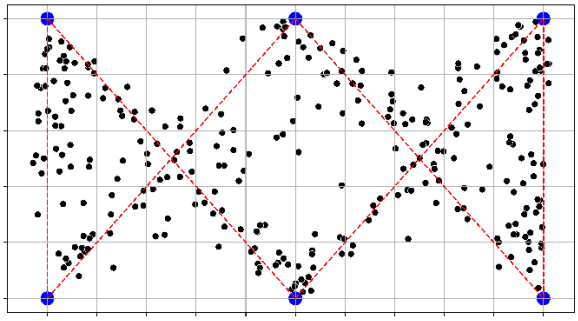


Trajectory pattern (II)

***(e)** Filtering on the Raw Locations*



Trajectory pattern (I)



Trajectory pattern (II)

***(f)** Filtering on the Shifted Locations*

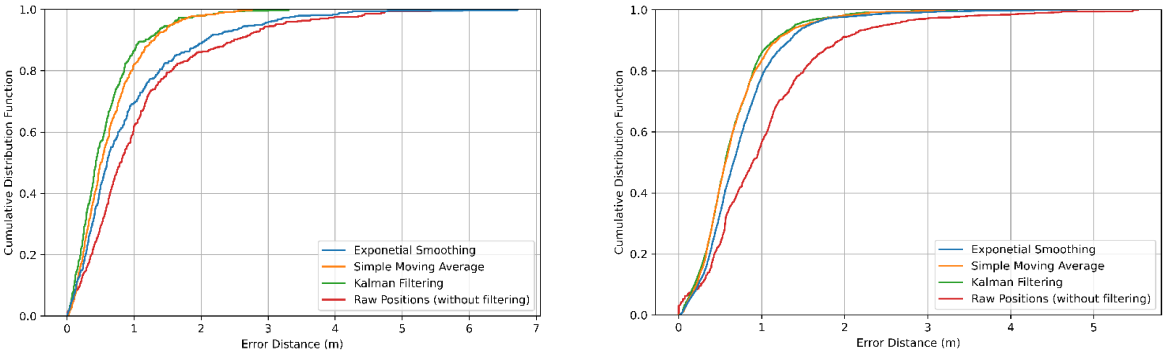
Figure 23: The distribution of estimated locations for the raw and post-processed data

Figure 23 (a) shows that the “shifting to the strongest transmitting beacon” technique moves the estimated locations toward the transmitting beacon's location, whereas applying the filtering on the raw locations tends to concentrate the estimated locations to the center of the sub-modules. It is observed that applying the filtering techniques on the “shifted locations” makes the estimated locations distributed over the sub-module area in the static test scenario. That was also observed for the dynamic test scenario, as shown in Figure 23 (b). The positive impacts of the combination of the two post-processing techniques on the raw estimated locations to make them follow the actual trajectories of the target are apparent for both test scenarios. Since the localization precision is critical in the deployability of the system for safety-related applications, the Cumulative Distribution Function (CDF) and localization errors in 90 % and 95 % of time metrics are used to investigate the system's performance in extreme cases. Table 12 shows the statistics of the error metrics for the static and dynamic test scenarios, and Figure 24 depicts the CDFs of the filtering techniques for both test scenarios

Table 12: Statistics of the error metrics between estimated and actual locations

for the static test scenario				
Metric	Raw Locations	Exponential Smoothing	Simple Moving Average	Kalman Filtering
MAE	1.04 (m)	0.75 (m)	0.65 (m)	0.64 (m)
SD Error	0.82 (m)	0.51 (m)	0.44 (m)	0.43 (m)
90% of Time Error	1.94 (m)	1.35 (m)	1.18 (m)	1.15 (m)
95% of Time Error	2.47 (m)	1.58 (m)	1.52 (m)	1.40 (m)
for the dynamic test scenario				
Metric	Raw Locations	Exponential Smoothing	Simple Moving Average	Kalman Filtering
MAE	1.08 (m)	0.90 (m)	0.63 (m)	0.56 (m)
SD Error	1.00 (m)	0.91 (m)	0.48 (m)	0.48 (m)
90% of Time Error	2.49 (m)	2.06 (m)	1.28 (m)	1.17 (m)
95% of Time Error	3.11 (m)	2.80 (m)	1.54 (m)	1.53 (m)

As seen from Table 12, applying the Kalman filter improved the localization accuracy by 39% and 48% in terms of the mean error for the static and dynamic test scenarios, respectively. The Exponential Smoothing was associated with more error compared to the others, with mean errors of 0.75 m and 0.90 m for the static and dynamic test scenarios, respectively. In contrast, the Kalman filtering techniques had the best performance in both scenarios, reducing the mean error to 0.56 (m) in the dynamic test scenario. Besides, Kalman filtering can substantially reduce errors in 90% of the time from 1.94 m to 1.15 m and from 2.49 m to 1.17 m for the static and dynamic test scenarios, respectively. As for the errors in 95% of the time, 38% and 43% improvement can be archived using the Moving Average and Kalman filtering techniques, respectively. To sum up, the Kalman filter outperformed other filtering techniques in terms of all the metrics in both test scenarios.

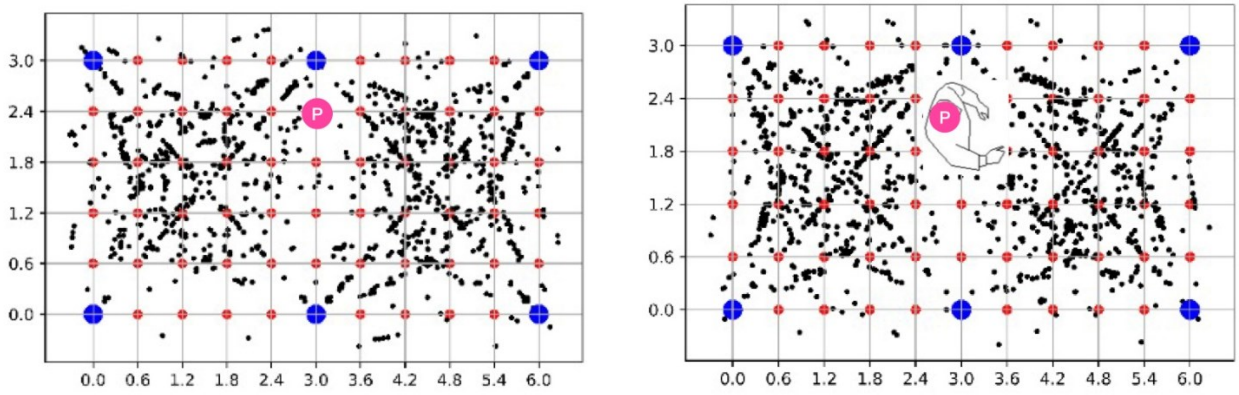


(a) Target State: Dynamic *(b) Target State: Static*

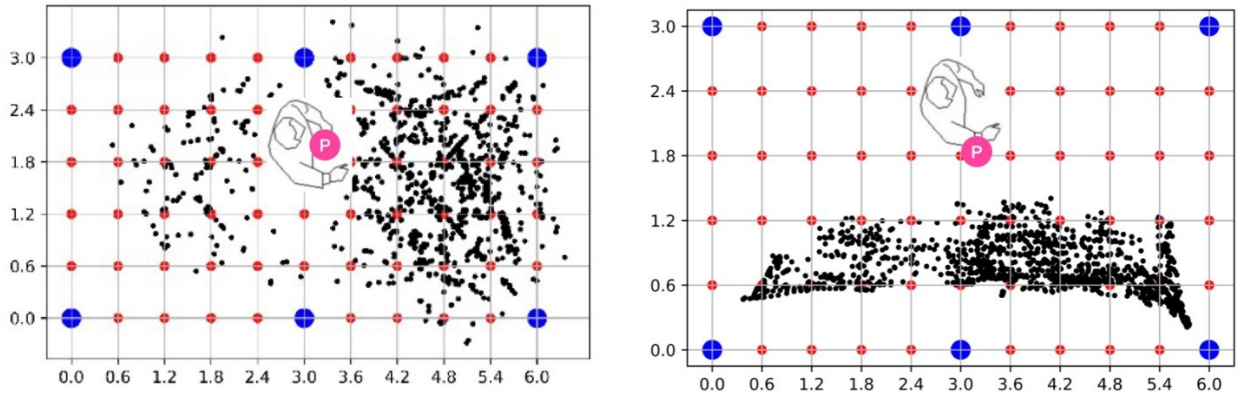
Figure 24: Cumulative Distribution Function (CDF) of the localization error for the experiment scenarios.

By comparing the CDF (Cumulative Distribution Function) plots between the filtering techniques, as shown in Figure 24, it is observed that the performance difference between the filtering techniques is more evident in the dynamic test scenario than the static one. It is observed that up to around 90% of the time, the orange line (Moving Average) is slightly lower than the green line (Kalman filter), indicating better performance for the Kalman filter. Both filtering techniques had a better performance than ES, specifically in the dynamic test scenario. Lastly, the curves for the filtering techniques seem to be steeper than raw locations, confirming the effectiveness of applying filtering techniques on the estimated locations to enhance the system's localization accuracy.

Thirdly, the effect of the Receiving beacon placement on a human body is examined. The signal attenuation caused by the human body shadows or obscures the BLE packets. This can directly affect the RSSI-distance estimation model, resulting in unreliable localization of the receiving beacon. The target locations for different receiving beacon placements on the human body are compared to demonstrate these effects. The placements considered for this experiment include (i) on the top of a hardhat, (ii) on the chest of workers, and (iii) on the worker's wrist. Additionally, the receiving beacon placement on a tripod at the same level as the transmitting beacons was used as a benchmark to compare the estimated locations for the different placements. The estimated locations of the different placements of the receiving beacon in the test-bed are depicted in Figure 25.



(a) Tripod *(b) Hardhat*



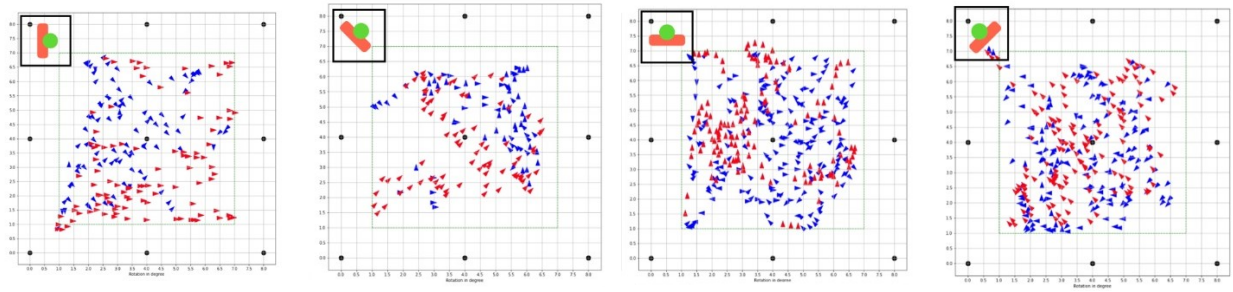
(c) Chest *(d) Right Wrist*

Figure 25: The distribution of estimated locations for the placements of the receiving beacon

As seen, the human body signal absorption affects the estimated locations for the “chest” and “right wrist” scenarios. This is because the human body prevents the receiving beacon from capturing BLE packets from the transmitting beacons located behind. Such blockage can, at the same time, create additional opportunities for monitoring the site processes, as discussed in the next section. Since the receiving beacon has Line-of-Sight with the transmitting beacons in the “on top of a hardhat” scenario, its estimated locations are similar to those of the “tripod” scenario. Hence, placing the receiving beacon on the top of the workers’ hardhat can be the best position to track the workers.

3.7.2) Body Orientation Detection Module

Firstly, the performance of the model is evaluated on the test set. The head-orientation detection model was evaluated using a validation set containing 2,279 records collected for the eight ordinal orientations. The absolute error of the orientation detection model, defined as the difference between the actual and detected body orientations, was used as the performance measure. Figure 26 shows the estimated body orientation of the target projected on its estimated locations, and Table 13 shows the statistics of the error metrics for the body orientation detection model.



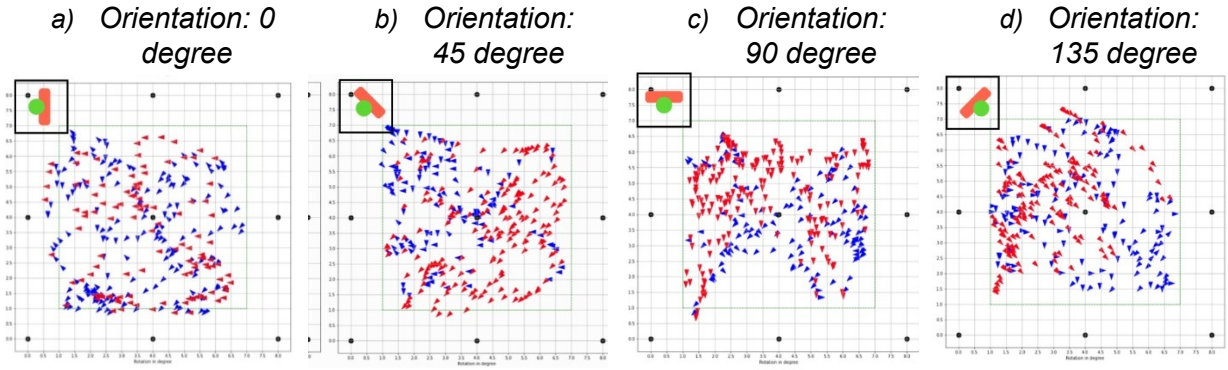


Figure 26: Estimated orientations of the target – Red triangles show the correct predicted orientation, and blue triangles show the incorrectly predicted orientation. The ground truth body orientation of the target is illustrated on the top-left of each figure.

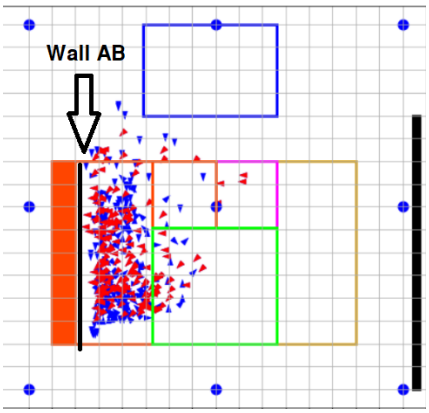
Table 13: Statistics of the error metrics between estimated and actual body orientations

Orientation	0°	45°	90°	135°	180°	225°	270°	315°	All
Mean error (degree)	58.2	34.8	52.7	33.3	29.7	21.7	34.4	33.1	36.5
SD error (degree)	24.5	25.9	48.5	45.6	`17.6	30.5	37.0	22.0	35.9

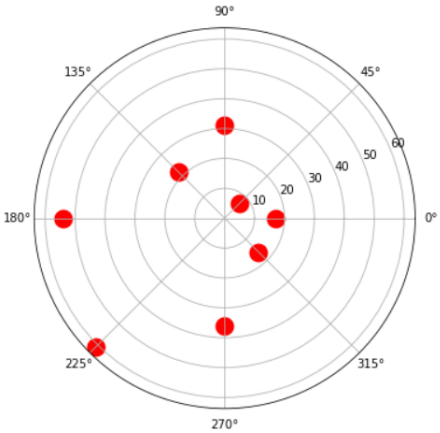
As seen from Table 13, the model achieved a mean error and standard deviation of 36 and 35 degrees, respectively. The mean error ranges from 21 degrees to 58 degrees for the orientation South (225°) to the East (0°), respectively. Since no previous research uses RSSI values between BLE beacons for human orientation prediction, it is unfair to compare the developed system performance with studies that deployed more sophisticated devices, such as cameras or Gyroscope sensors. However, the achieved results show the potential of the system to perform as close as other systems [99][100].

Secondly, the performance of the model is evaluated in the simulated operations. The performance of the body orientation detection module is examined for the painter and plasterer since their body orientation was nearly the same during their work. The mason was excluded from the evaluation since their chest- and hardhat-mounted receiving beacons mostly did not have LoS with the reference transmitting beacons. The records whose productivity state was detected as direct work and support work states were considered for the evaluation analysis. The painter worked on the wall “AB” while the plasterer worked on the three sides of the column, named “Side A”, ”Side B”, and “Side C”, Since the body orientation often changes during the working time, the workers' focus orientation during the working time is also considered as an evaluation criterion to

assess the module performance. Hence, the most frequent body orientation in a time window of eight minutes is determined and considered the orientation values of that window. Figure 27 shows the statistics of the predicted body orientations during the experiment and the estimated locations overlayed with body orientation. The accuracy of the predicted body orientations is presented in Table 10.

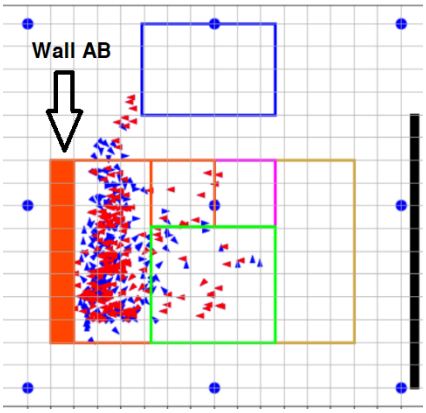


Scatter plot of estimated locations and orientations

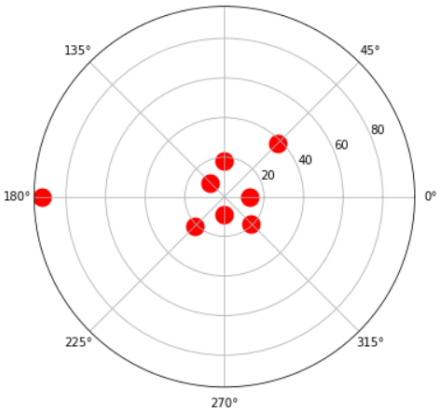


Statistics of the body orientation on the wall AB

(a) Single-crew Painting Experiments

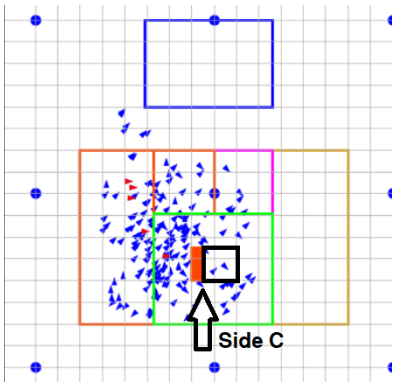


Scatter plot of estimated locations and orientations

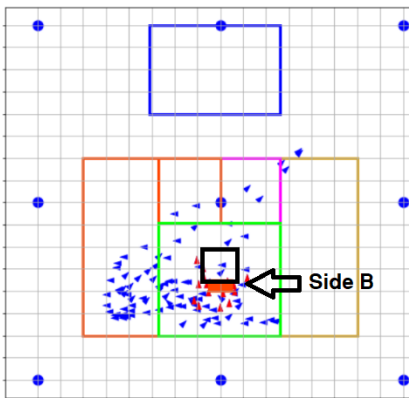


Statistics of the body orientation on the wall AB

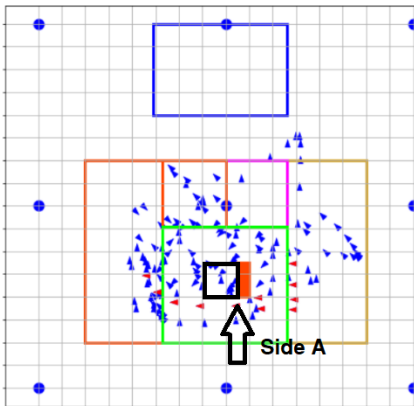
(b) Full-operation Painting Experiments



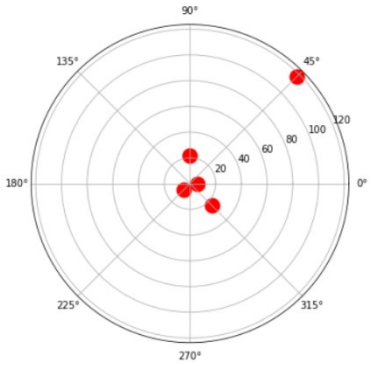
Scatter plot of estimated locations and orientations on side "C" of the column



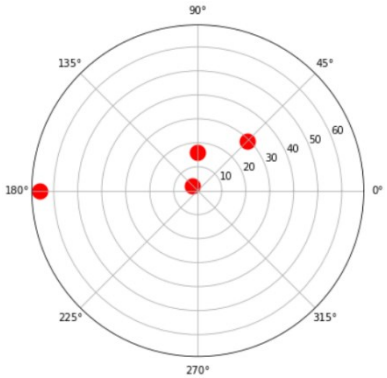
Scatter plot of estimated locations and orientations on side "B" of the column



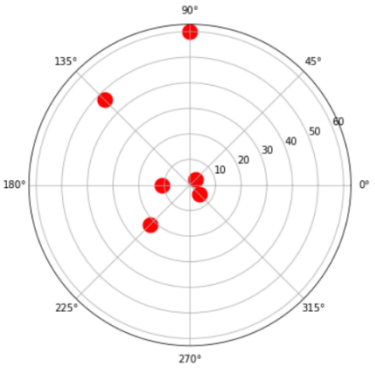
Scatter plot of estimated locations and orientations on side "A" of the column



Statistics of the body orientation *on side "C" of the column*

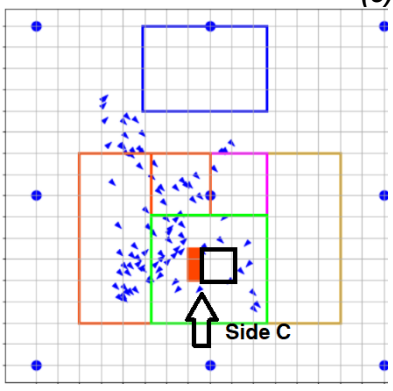


Statistics of the body orientation *on side "B" of the column*

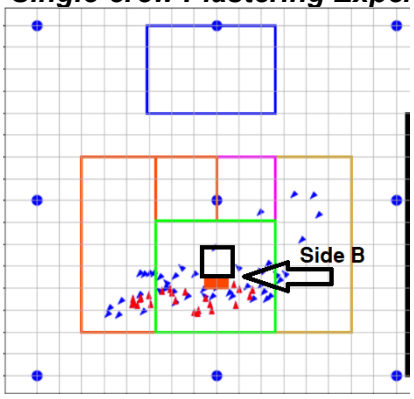


Statistics of the body orientation *on side "A" of the column*

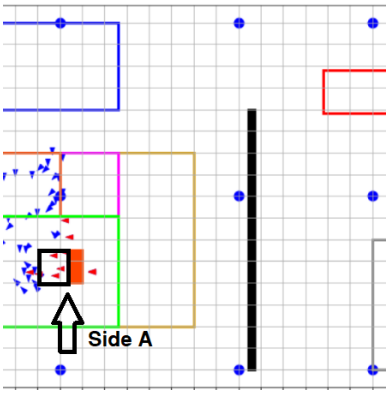
(c) Single-crew Plastering Experiments



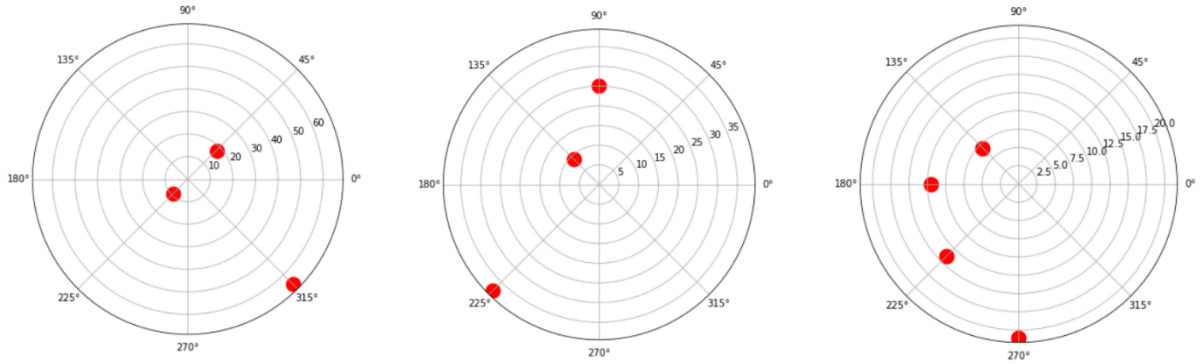
Scatter plot of estimated locations and orientations on side "C" of the column



Scatter plot of estimated locations and orientations on side "B" of the column



Scatter plot of estimated locations and orientations on side "A" of the column



Statistics of the body orientation *on side "C" of the column*

Statistics of the body orientation *on side "B" of the column*

Statistics of the body orientation *on side "A" of the column*

(d) Full operation Plastering Experiments

Figure 27: Performance of the body orientation detection module. The red and blue triangles represent the correct and incorrect predicted body orientations.

Table 14: Body orientation detection module accuracy with the raw and the most frequent body orientations

Worker	Body Orientation	Raw predicted body orientation				Predicted focus orientation			
		Single-crew Experiments		*Full-Operation Experiments*		*Single-crew Experiments*		*Full-Operation Experiments*	
		Mean error	*SD error*	*Mean error*	*SD error*	*Mean error*	*SD error*	*Mean error*	*SD error*
Painter	West (180°)	62°	44°	65°	54°	37°	21°	22°	0°
Plasterer	West (180°)	57°	48°	93°	52°	87°	11°	135°	0°
	North (90°)	63°	33°	57°	48°	90°	0°	52°	24°
	East (0°)	53°	26°	54°	36°	45°	0°	45°	0°

As seen in Figure 27 (i-ii), the body orientation of the painter was frequently predicted to the West and West-South orientations (180° and 225°) in the experiments. The module achieved a mean error of 37° and 22° in the predicted focus orientation of the painter in the single-crew and full-operation experiments, respectively. Unlike plastering work located in the middle of the simulated site, the painter worked on the edge of the site where the presence of the other crews did not cause signal interference for the body orientation detection module. In addition, the painter's chest-mounted receiving beacon generally had LoS with the reference transmitting beacons, resulting in the expected performance of the body orientation detection module. Hence, the totality of the predicted body orientations of the full-operation experiments was relatively similar to the single-crew experiments.

On the other hand, the model could not function normally for the plasterer due to the presence of the column that blocked the LoS of the chest-mounted receiving beacon from the reference transmitting beacons. Specifically for the North orientation (90°), the column was located in the sightline between the chest-mounted receiving and reference transmitting beacons. The mean error of the predicted body orientations for the three sides of the column ranges from 53° to 63° for the single-crew experiments. While the plasterer was working in the East direction (0°), the module typically detected the body orientation toward North-East (45°) in the single-crew experiments. In contrast, the predicted orientations were typically shifted toward South-East (315°) in the full-operation experiments. The most likely reason is the presence of the mason helper in the upper left of the plasterer in the full-operation experiments, verifying the impacts of the signal blockage caused by the worker body on the module performance. Due to the scatteredness of the predicted body orientations, the focus orientation of the plasterer was not accurate compared to the painter. Overall, the results show that the predicted body orientations can be used to identify building component(s) that a worker is working over a specific time period, on the condition that the LoS exists between the chest-mounted and the reference beacons.

3.7.3) Productivity State Detection Module

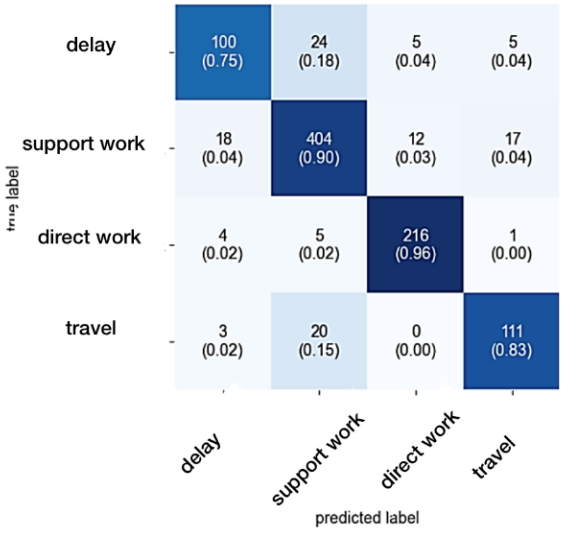
The performance of the productivity state detection module is evaluated both on the testbed and the simulated operations. As per the testbed, the classification performance of the six models was evaluated by using metrics, including Precision, Recall, and F_1-Score. Precision and Recall are the most frequently used evaluation metrics reported in the deep learning literature [101]. Precision is a positive predictive value, whereas recall is also known as sensitivity [72]. F_1-Score metric is described as the harmonic mean of Recall and Precision [102]. The average accuracy of five-fold cross-validation was used as the final result of the classification of the productivity state detection module. Table 15 shows the classification report of the models' performance.

Table 15: Classification report showing the productivity state model performance.

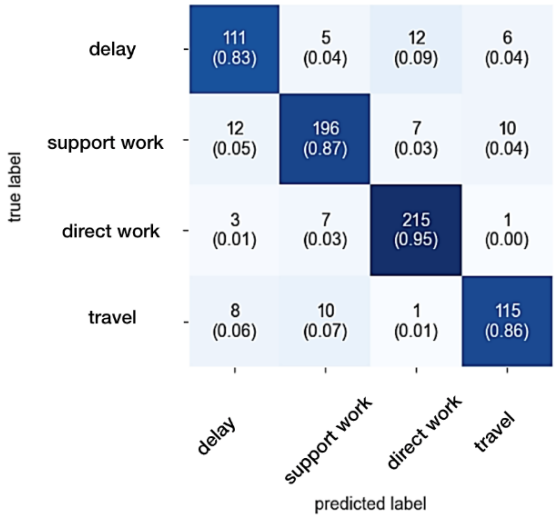
Worker	Precision	Recall	F1-Score	Support
Painter	0.87	0.87	0.87	945
Plasterer	0.89	0.89	0.89	719
Plastering helper	0.84	0.84	0.84	470
Mason	0.87	0.87	0.87	537
Masonry helper	0.84	0.83	0.83	675

Runner	0.83	0.83	0.83	577

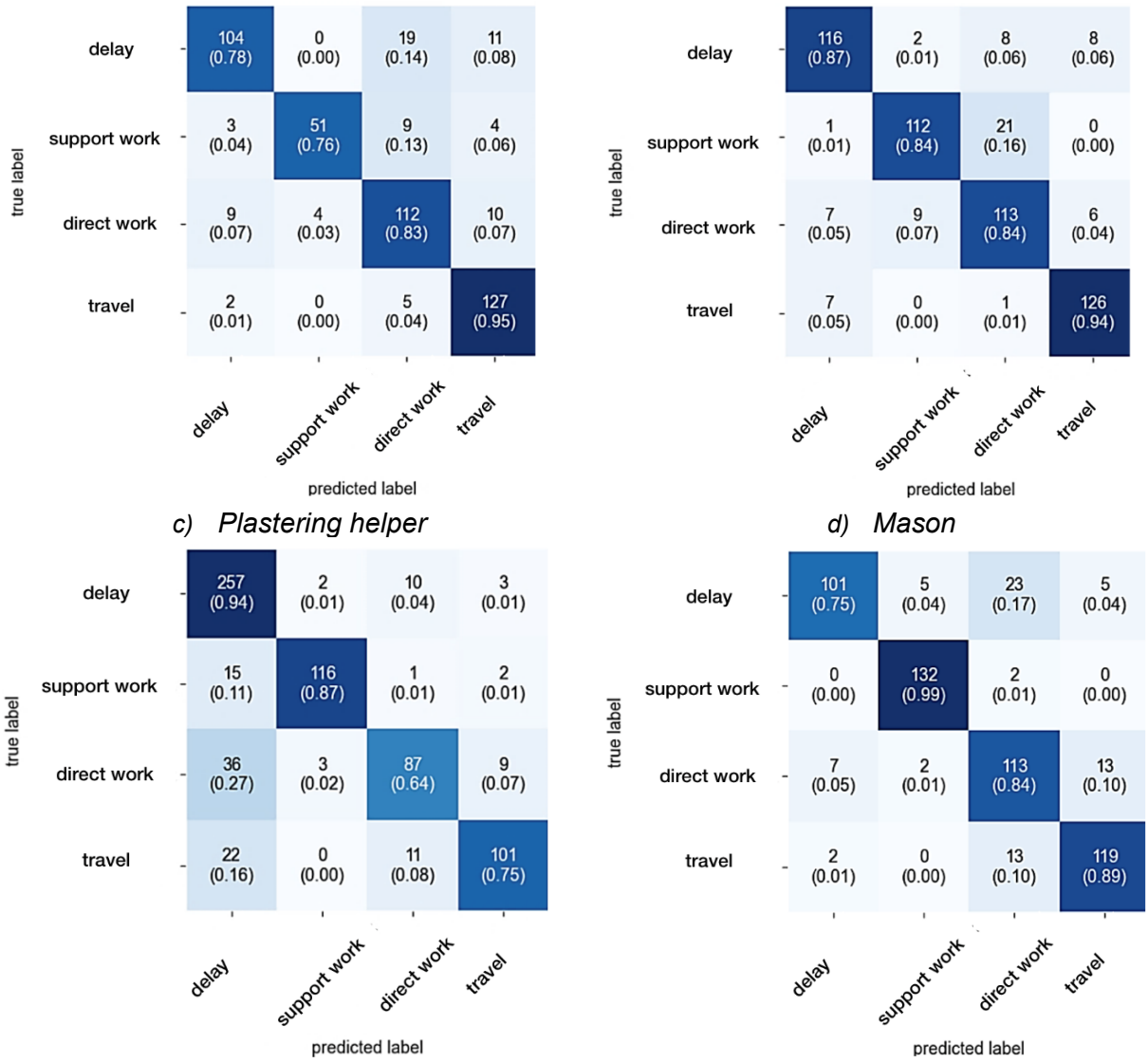
Since the number of supports (test samples) for the models is different, as shown in Table 15, F1-score is a better metric for evaluating the trained models. The module achieved an F1-score of at least 0.84 in predicting the productivity state for the worker. The support work state is predicted with higher accuracy than other classes, except for the painter and mason crews. The model achieved a prediction accuracy of 89%, 87%, and 87% for the plasterer, painter, and mason, respectively. On the other hand, the plastering helper and the runner are the more challenging crews for the model to detect productivity. The reason might be the subtle differences in body movement in performing tasks that the classifier models could not well measure. Overall, the results show that the model can differentiate the productivity states for all the workers. Figure 28 shows the confusion matrix, highlighting instances where the models do not recognize productivity states correctly. The right coordinate represents the actual class of the productivity state, and the bottom coordinate represents the prediction results of the states. Besides, the values in the main diagonal show the recall of the model's predictions.



a) Painter



b) Plasterer



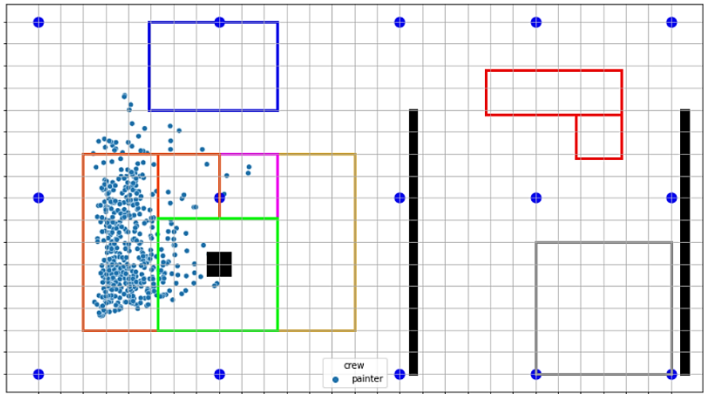
c) *Plastering helper* d) *Mason*

e) *Masonry helper* f) *Runner*

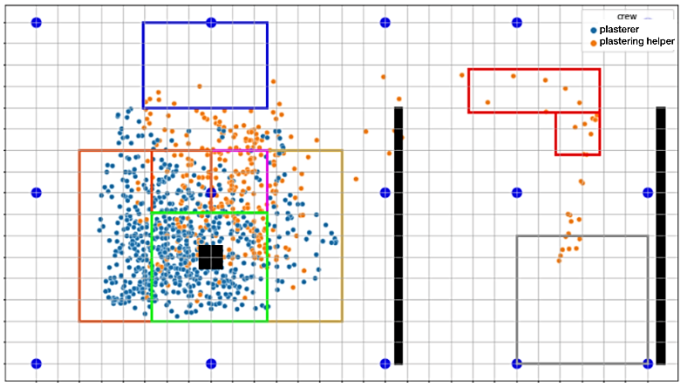
Figure 28: Confusion Matrix showing the productivity state model performance.

As seen in Figure 28, the delay state was rarely confused with the other classes except for the painter. However, the direct work state for the painter was predicted with a recall of as high as 0.96 since its direct work state is more distinguishable than its support work. The same distinction between direct work and support work can be observed for the runner, where the F1-score of 0.99 and 0.84 were achieved, respectively. On the other hand, the model trained for the mason occasionally confused direct work and support work states. The reason might be the similarity of the body posture for the productivity states, generating similar acceleration values that are challenging for the model to differentiate.

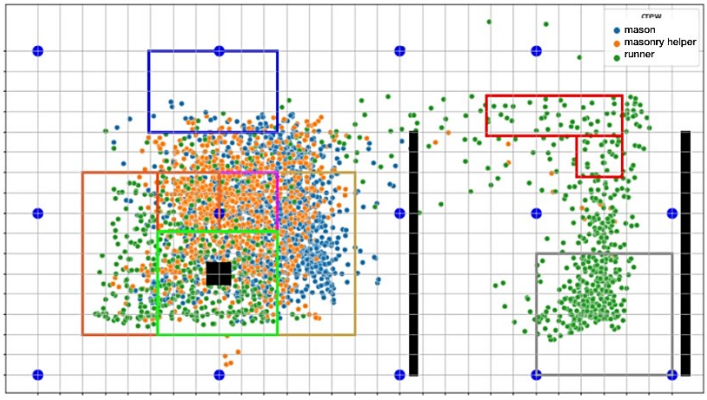
Furthermore, the model's performance is evaluated in the simulated operation experiments. Figure 29 shows the estimated location of the workers over the workspaces during the single-crew and full-operation experiments.



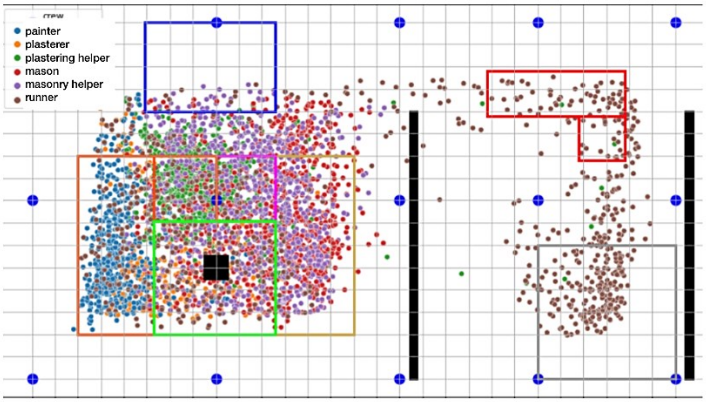
a) Single-crew Painting Experiment



b) Single-crew Plastering Experiment



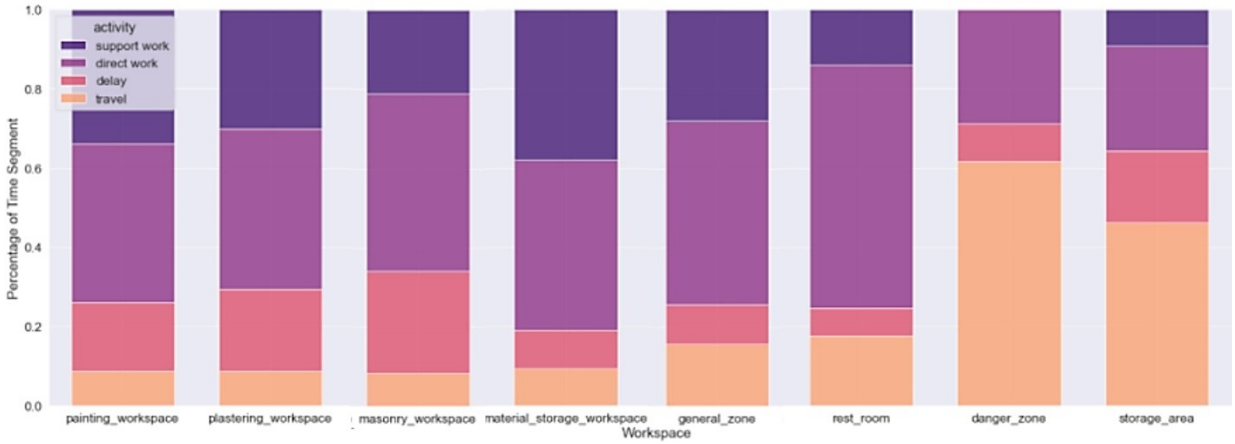
c) Single-crew Masonry Experiment



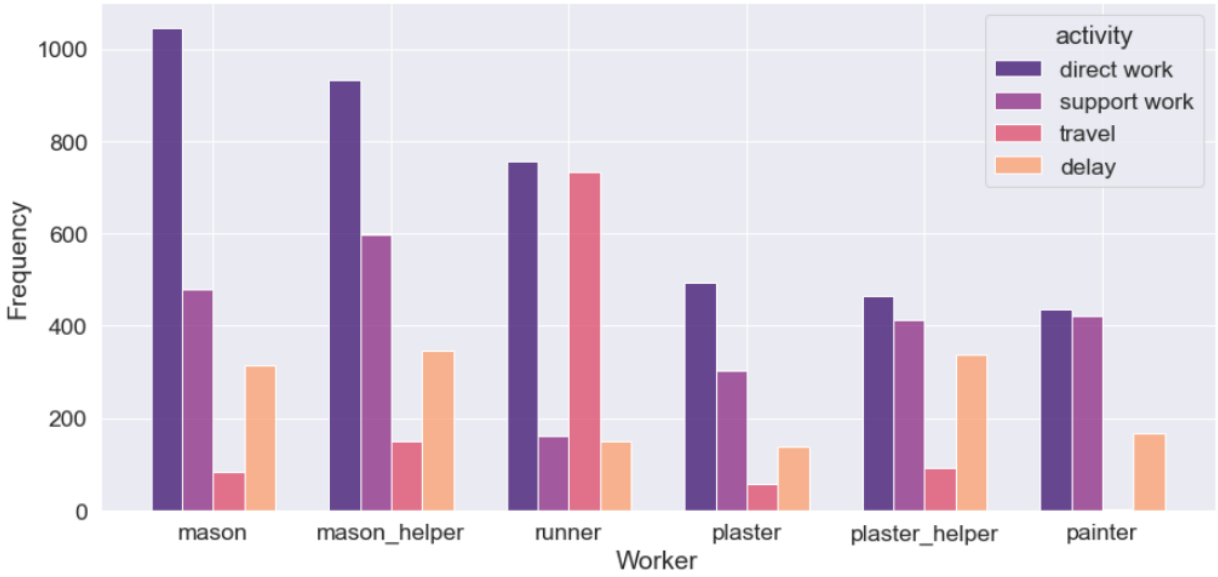
d) Full-Operation Experiments

Figure 29: Workspaces and estimated locations of the crews

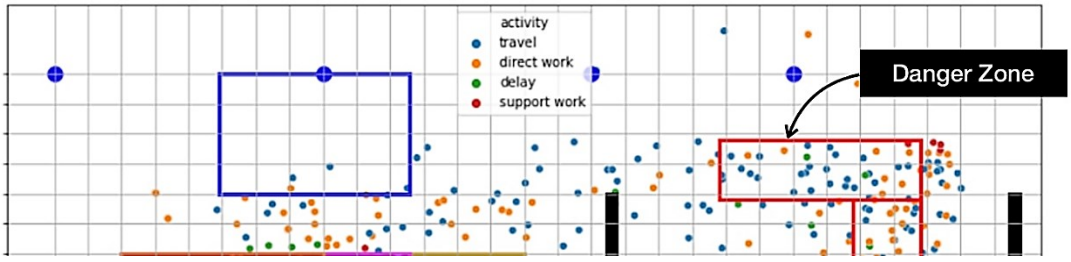
As seen from Figure 29 (a), the painter did not travel to the storage area, whereas for the plastering and masonry activities, the plastering helper and runner traveled to the storage area to take the materials, respectively (Figure 29 (b-c)). The plastering helper went to the storage area only once (at the beginning of the activity); however, the runner had to go to the storage area several times to bring the material. Figure 30 (a) shows the productivity states ratio of the workers during the full-operation experiments. Figure 30 (b) shows the predicted productivity state of the workers in the workspaces. It is noted that the provided analysis is based on a portion of the time that workers started and finished their construction tasks.



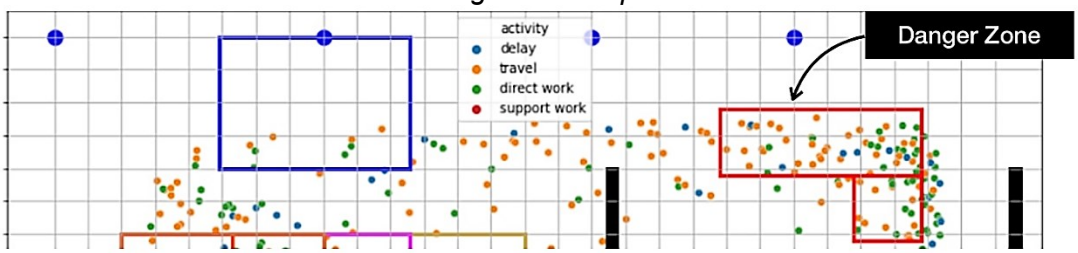
a) Overlaying worker's productivity state over the workspaces



b) Overlaying worker's productivity state over the crews



i. Single-crew experiments



ii. Full-operation experiments

c) Overlaying the runner's productivity state over the estimated locations in the traveling path

Figure 30: Worker's productivity states over workspaces

As seen from Figure 30 (a), the predominant productivity state (65 percent to be exact) of the records in the danger zone was travel since it was located in the traveling path, and the runner had to carry the wheelbarrow between the storage area and the masonry workspace. Likewise, 48 percent of the productivity state of the generated records in the storage area was travel since almost half of the storage area was used as a traveling path by the runner. The minority of the productivity states was predicted as a delay in the rest area since only the working time was considered for the analysis. The direct work state of the generated records in the rest area could be the productivity states of masonry helper and plastering helper, whose locations were wrongly predicted in the rest area, as depicted in Figure 29 (d). In addition, only four records of the painter were predicted as travel state; however, about eight percent of productivity states predicted in painting workspace was travel due to the RTLS location inaccuracy associated with plastering crew (plasterer and plastering helper). In general, the productivity states generated in the painting, plastering, masonry, and general zone follow almost the same pattern.

It is clear from Figure 30 (b) that the painter, plasterer, and plastering helper rarely walked during the painting activity. The runner had the highest ratio of travel to other states among workers, which was about 40 percent. Among other workers, the masonry helper had the highest walking ratio to other states, which was about 9 percent. The reason is that the worker had to pour mortar over the wall bricks in the masonry workspace and return to his workspace (material storage area). The level of accuracy of the productivity state module is verified by comparing the observed productivity state of crews from the video recordings and the predicted states by the module. The manual productivity assessment was made while the workers worked at the workspaces, except for the runner. Table 16 compares the values obtained from the automated and manual productivity state analysis.

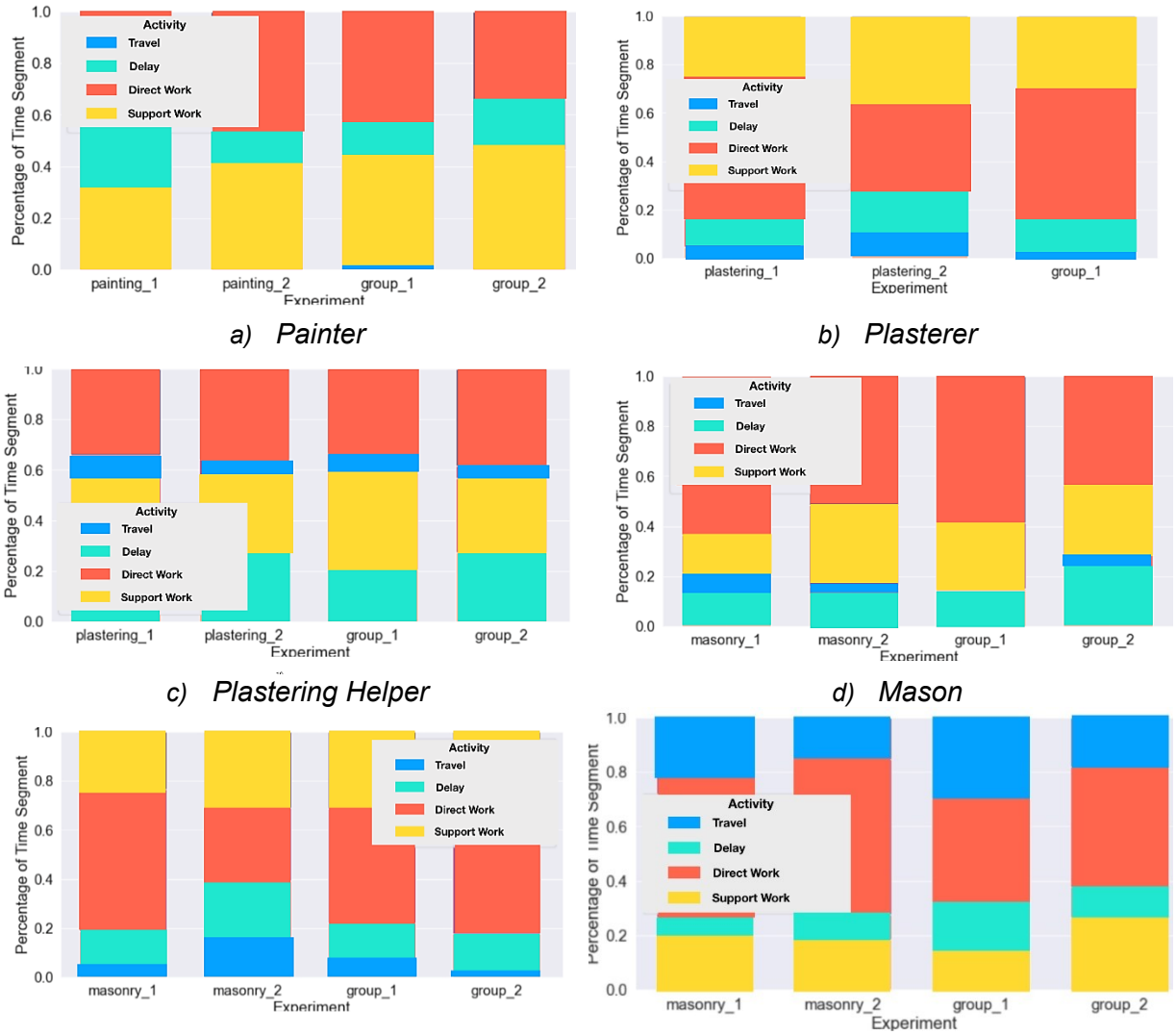
Table 16: Average difference and standard deviation of the differences between the automated and manual activity analysis

	Delay	Support work	Direct work	Travel
		Painter		
Mean actual duration	38 (sec)	362 (sec)	156 (sec)	n/a
Mean error	24%	12%	19%	n/a
SD error	12%	5%	11%	n/a
		Plasterer		
Mean actual duration	120 (sec)	215 (sec)	377 (sec)	n/a
Mean error	17%	13%	15%	n/a

SD error	12%	5%	9%	n/a
		Plastering Helper		
Mean actual duration	244 (sec)	266 (sec)	196 (sec)	n/a
Mean error	26%	28%	37%	n/a
SD error	18%	9%	28%	n/a
		Mason		
Mean actual duration	249 (sec)	283 (sec)	491 (sec)	n/a
Mean error	33%	14%	19%	n/a
SD error	14%	10%	21%	n/a
		Masonry Helper		
Mean actual duration	202 (sec)	394 (sec)	477 (sec)	n/a
Mean error	30%	15%	22%	n/a
SD error	14%	10%	14%	n/a
		Runner		
Mean actual duration	82 (sec)	206 (sec)	367 (sec)	293 (sec)
Mean error	22%	53%	17%	32%
SD error	25%	7%	11%	22%

As seen from Table 16, the model identified the direct work state of the plasterer and painter with a mean error of less than 15% and 19% for the work states, respectively. In contrast, the model had the worse performance in predicting the direct work state of the plastering helper that was associated with a mean error of as high as 37%. It verifies the relatively poor performance of the model trained for the plastering helper, as discussed in the previous section. Typically, the delay state of the workers was predicted with lower accuracy than other states, especially for the mason and masonry helper. The reason may be that the hand motion of the workers confused the productivity state detection model to detect the delay state accurately. The travel state was identified with a high mean error of 34% for the runner. The reason might be the different level of speed at which the runner carried the wheelbarrow during his/her traveling path. Since the runner had to maneuver the wheelbarrow at the end of the danger zone to reach the storage area, he paid more attention once he reached the corner. Accordingly, his productivity state was sometimes detected as the direct work state. Figure 30 (b) shows the overlay of the runner's productivity state over its estimated locations in the traveling path.

As previously mentioned, the participant who performed the second experiment of the single-crew experiments was different from the one who performed the first experiment of the first round and the full-operation experiments. Figure 31 shows the distribution of the productivity state of different workers over the experiments. The results for the plasterer in the second full-operation experiment were excluded due to the defection in the chest-mounted receiving beacon.



a) *Painter* b) *Plasterer*

c) *Plastering Helper* d) *Mason*

e) *Masonry Helper* f) *Runner*

Figure 31: Detected Productivity State of Workers over the Experiments

As seen from Figure 31, for the plasterer and masonry helper, the productivity performance of the different participants who performed the activities is distinguishable from one another. The productivity performance of the workers (except for the plastering helper) followed almost the same pattern for the last two experiments. It shows the impacts of the learning curve of activities by the workers that was stabilized after performing the activity after the second time. The direct work was the dominant productivity state for the plasterer, masonry helper, mason, and runner. By contrast, direct work and support work had a roughly similar share in the predicted productivity state of the painter and plastering helper. The painter had to perform a relatively higher number of secondary construction tasks, including mixing the paint and staining a paint roller with paint. Besides, the plastering helper had the highest ratio of delay productivity state compared to other

workers. The reason is that this worker had to frequently wait for the plasterer until their plaster material was finishes and refilled the plaster bucket. Oppositely, the runner had the highest ratio of travel state in comparison with other workers, which was about 24 percent of the total predicted states during the four experiments. The automated and manual activity analysis results for the experiments are shown in Figure 32. The time difference is calculated based on the difference in the actual and predicted time spent for each productivity state. While the actual time spent is calculated based on the video recordings, the predicted time spent is computed by adding the time difference between executive records for each productivity state.

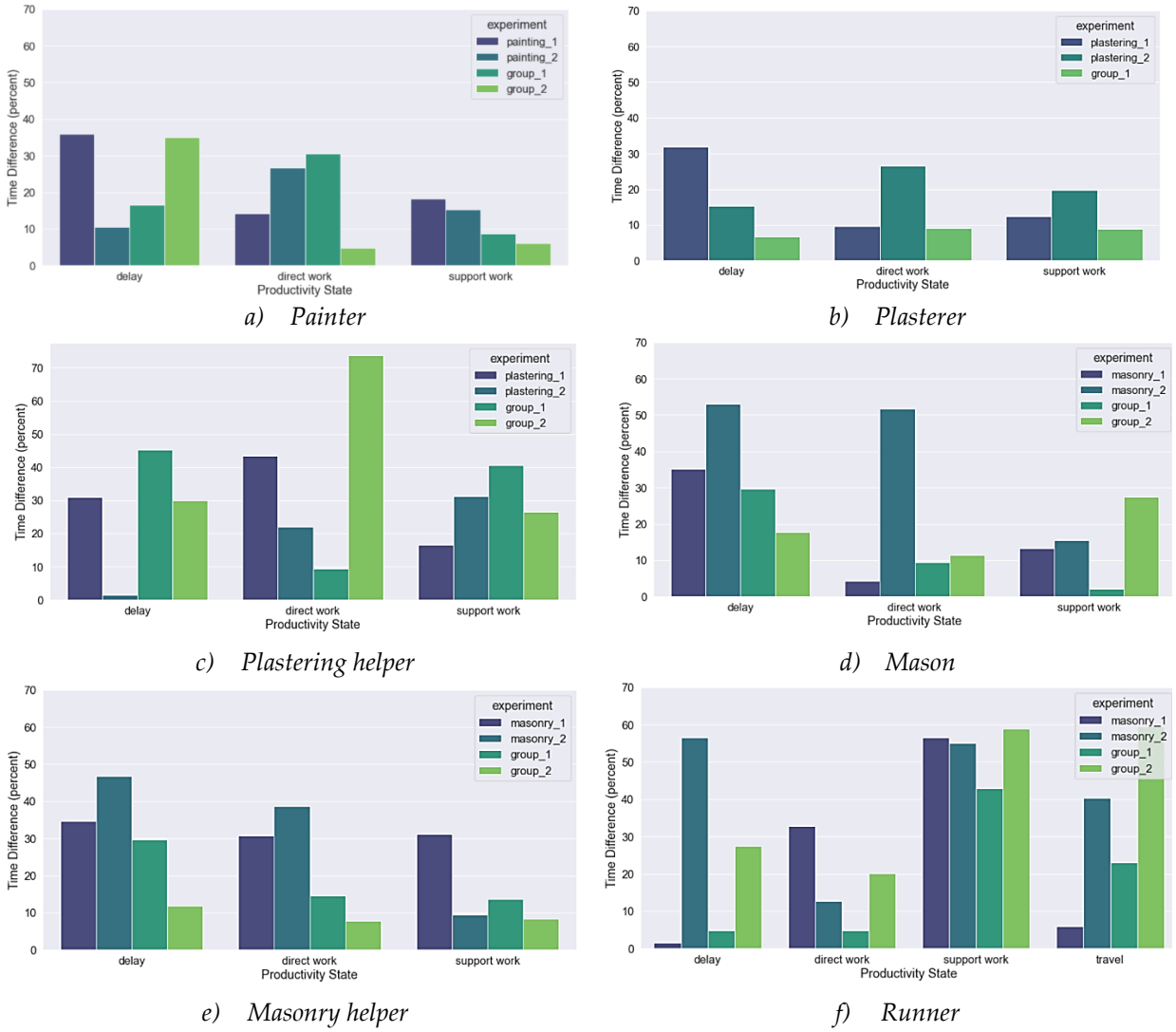


Figure 32: Time difference between the predicted and actual productivity states of the workers over the experiments

As seen from Figure 32, for the plasterer and mason, the time difference error of the second single-crew experiment generally differs from other experiments. In the second single-crew experiment, the time difference was roughly 280 and 510 percent higher for the plasterer and mason, respectively. As mentioned, the second single-crew experiment was conducted by a participant who is different from the participant who conducted the other experiments. It confirms the impacts of the different working styles of the participants on the productivity state model's performance. The worse performance of the module occurred for the second full-operation experiment for the plastering helper followed by the plastering helper, which was as high as 31 and 23 percent, respectively. Both workers performed the construction task of mixing the material, which might have confused their classifier models for productivity state detection. The time difference error for the plastering helper was unusually high in the second full-operation experiment since the wrist-mounted receiving beacon was removed several times during that experiment.

3.8) Recommendations for Construction Sites

Based on the level of accuracy achieved and the man-hours requirement for installing the RTLS infrastructure, the authors conclude that deploying the proposed BLE-based RTLS is feasible for tracking workers on construction job sites. Deploying BLE beacons as the reference and tracking BLE devices help to minimize the interference to the workflow and safety implications resulting from cables of the wired sensors, and it also makes the RTLS infrastructure more resistant to fall damages. Since the location data are not collected nor transferred through the worker's smartphone, the data privacy issue is resolved in the developed RTLS. This issue is regarded as the most critical concern by workers in adopting tracking devices on job sites [59]. Unlike other Locating Systems, which performed better in specific zones, such as areas closer to the fixed transmitting/receiving beacons ([50], [52], the accuracy of our proposed system has no bias toward a specific area of the test-bed including sub-module edges or areas close to the transmitting beacons (see Figure 22). This supports the scalability of the proposed infrastructure and suggests that placing modules beside one another is not expected to negatively affect the localization's accuracy. It is noted that the proposed RTLS is highly dependent on a Line-of-Sight (LOS) between the beacons, which makes the system not ideal for the applications that require the receiving beacon to be mounted at a significantly lower height from the reference transmitting beacons, such as material tracking. However, the system is highly recommended for construction applications that require tracking the workers' location and construction equipment.

Moreover, various intensity levels of the post-processing were found to be efficient for different applications, depending on the level of the worker's movement that each application requires to monitor. For instance, productivity assessment of crews involved in static operations, e.g., masonry work, carpentry welding, plastering, etc., requires an intense level of smoothing. Since the workers do not move (in short intervals), the filtering techniques can take advantage of previous states' records to interpret the current state, which minimizes the error in the estimated locations. By contrast, mild smoothing is necessary for applications associated with workers' dynamic behavior, such as safety-related applications, including the deployment of safety alert systems for hazardous zones avoidance when approaching dangerous areas on the job site. A combination of mild and intense smoothing will be efficient for other applications in which the workers are both static and dynamic, such as automated workspace identification. As per the filtering type, Moving Average is the ideal alternative for safety-related applications due to its acceptable performance in 95% of the time and the shorter computation time than others, which will be essential for deploying real-time solutions. Although the Kalman filter roughly outperforms

other filtering techniques in all the evaluation metrics considered in this study, it takes a significantly higher computation time, making it less appropriate for safety management applications compared to the Moving Average. However, the Kalman filter can be deployed successfully for productivity monitoring applications, including path planning and workspace identification. Besides, the results of the effect of the receiving beacon placement on a human body show the workers' hardhat is the best place for localization since the human body blockage cannot cause signal blockage.

The body orientation detection module could provide an acceptable level of accuracy in detecting the building component on which a worker performs construction activity, specifically for the building component located on the sub-modules boundary edge. Like RTLS, the body orientation detection module is highly dependent on a LOS between the chest-mounted and reference beacons, which makes the module not ideal for applications requiring a worker to work on building component that blocks the LOS between the beacons. The module detected the wall on which a painter was painting, whereas the module produced a relatively large error in predicting body orientation for the plasterer due to the signal blockage caused by the column. The productivity state detection module's performance was independent of RTLS location data, resulting in more reliable productivity state data regardless of LOS between body-mounted and reference beacons. It is particularly advantageous in detecting productivity states for construction activities requiring workers to have body posture where the LOS cannot be maintained. For instance, the mason's hardhat- and chest-mounted beacons mostly have NLOS with reference beacons when the worker lays bricks at low altitude, so the developed productivity state detection module can be superior to the RTLS location displacement method. In addition, the module classified the working productivity state into direct work and support work. Such classification produced insightful information for productivity assessment of workers, which could not be provided through the RTLS location displacement method.

For a successful RTLS deployment on job sites, the following recommendations are made to enable a proper setup: (i) deploy the sub-module on the boundary of the area of interest (no need for a buffer area outside the target zone); (ii) distribute the sub-modules according to the site layout and keep on changing the position of the sub-module as the layout of the construction site changes during the project; (iii) consider the minimum size of the sub-module that is the maximum distance in which the transmitting beacon can send BLE packets reliably for RSSI-distance prediction throughout the localization; (iv) keep the transmitting beacons at a minimum of height that is equal to the average of the worker's height, i.e. 2 (m), or more in order for the beacons to

have LOS with one another (increasing the height beyond that results in a shorter sub-module size); (v) turn the orientation of the transmitting beacons towards the ground to provide evenly transmission coverage area (particularly for the job sites with many obstacles).

The proper setup of RTLS positively affects the performance of the body orientation module by providing accurate location data. In addition, the following recommendations are made for the efficient deployment of the body orientation detection module: (i) place the boundary edge of sub-module(s) on the wall on which a worker is supposed to work; (ii) do not use the module for workers who are supposed to work on building components not located in boundaries of sub-modules; (iii) reduce the size of sub-modules in workspaces where many workers are supposed to work in close proximity; and (iv) consider deploying the module for workers who can mostly keep their bodies upright during the working time. Unlike RTLS and body orientation detection modules, the performance of the productivity state detection module is independent of the reference transmitting beacons (sub-modules). However, the following suggestions are made for a successful deployment of the productivity state detection module: (i) consider a reliable installation of the body-mounted beacons for workers who are supposed to perform intense physical construction activities and (ii) pay attention to keep the orientation of the body-mounted beacons similar to the body-mounted beacons on which the classifier models are trained.

Chapter 4 – Conclusion

4.1) Research Summary

Tracking workers and objects on construction job sites is essential for various applications, including safety, progress monitoring, on-site coordination, and geographical mapping of worker locations and trajectories. Potentially, higher knowledge of worker performance can be achieved by integrating worker location data with additional information, such as body orientation and productivity state data. The main goals of this study included: (i) designing a Real-Time Locating System for construction job sites by considering the aspects affecting the system deployability in the construction domain, including portability, affordability, scalability, and localization accuracy; (ii) developing a body orientation detection model to predict the worker's body orientation in eight ordinal orientations using the RSSI values and geometrical relationship data between reference transmitting and chest-mounted receiving beacons; and (iii) developing a productivity state detection model to predict the productivity state of the worker using acceleration signals processing provided by the accelerometer-embedded receiving beacons mounted on the worker's hardhat, chest, and writs. It is noted that the RTLS and body orientation detection modules are

developed for the indoor environment. Besides, the productivity state detection module is developed for repetitive constriction activities, including painting, plastering, and masonry.

A novel BLE-based RTLS was proposed, in which BLE beacons replaced the commonly used Bluetooth-enabled devices. For this purpose, the beacons' configurations were tuned, and a record correction algorithm was developed to solve the shortage of BLE packets limited payload that the receiving beacon relies on to broadcast the location data to the gateway. A modular system infrastructure placement was proposed based on the effective devices' range to strategically place the beacons on job sites. The proposed localization estimation model categorized and subsequently localized the records based on the reference transmitting beacons' position and estimated the distance between the receiving and transmitting beacons based on the RSSI. Finally, the locations were post-processed by filtering techniques to mitigate the effect of environmental noises. The performed experimental work indicated a localization error of 0.56 (m) and 0.64 (m) in a middle-size room in cases of dynamic and static targets, respectively. As per the localization precision, the system demonstrated no bias toward a specific coverage area and achieved, in 90% of the time, an error of less than 1.15 (m) and 1.17 (m) for static and dynamic test scenarios, respectively.

The body orientation detection module used the RSSI values and geometrical relationship data between reference transmitting and chest-mounted receiving beacons to predict the worker's body orientation. To this end, in-lab experiments were carried out to collect data between the chest-mounted receiving beacon and reference transmitting beacons for eight ordinal orientations. Finally, an ANN model was trained to predict the worker body orientation in eight ordinal orientations. Last but not least, the productivity state detection module predicted the productivity state of the worker using acceleration signals processing provided by the accelerometer-embedded receiving beacons mounted on the worker's hardhat, chest, and writs. With that goal, firstly, a frequency image generator segmented the tri-axial accelerometer data from the receiving beacons and converted them into frequency images. Secondly, six two-dimensional Convolutional Neural Networks (CNN) were trained to detect the workers' productivity states by taking the generated frequency images as input. The system performance was validated through experiments conducted on an in-lab simulated construction site by six volunteers, who played the role of various construction workers. The experimental results indicated that the productivity state detection module achieved a less than 20 percent time difference error in detecting the direct work and support work productivity states for the painter, plasterer, and mason. The body orientation detection module achieved a mean error of less than

30° in detecting the focus orientation on the condition that the LoS exists between the reference transmitting and chest-mounted receiving beacons.

4.2) Research Contributions

The main contributions of this work summarized as (i) proposing an RTLS architecture with minimal dependency on wiring and electricity outlets; (ii) developing an algorithm and configuring receiving and transmitting beacons to minimize the effect of signal interference caused by a network of transmitters ; (iii) categorizing the measurements of positions and distances of the (fixed) transmitting beacons from the (moving) receiver and developing localization algorithms for each category; (iv) examining the performance of various post-processing mechanisms on the estimated locations to find the best solutions for mitigating the system's incoherence in computed locations when the target is static and dynamic; (v) training a DNN model that uses RSSI values between chest-mounted receiving and reference transmitting beacons that are used for RTLS to detect body orientation; and (vi) training CNN models that take low-frequency acceleration data of the accelerometer embedded in the hardhat- and chest- and wrist-mounted beacons to detect worker’s productivity states.

In the proposed RTLS, the substitution of the commonly used Bluetooth-enabled devices with BLE beacons made the distributed infrastructure of the system, except for gateways (one every 900 m2 in a square-form layout), independent from wiring work and DC power supply. This substantially increased the system's deployability and flexibility of application on construction sites. The proposed modular system infrastructure placement strategy minimized the number of required on-site devices and improved the RTLS scalability and efficiency in terms of cost and power consumption. In traditional Bluetooth-based RTLS architecture, receiving nodes that require wiring are usually used as fixed benchmarks, and the moving objects are tracked through transmitting devices.

This study also proved that placing the receiving beacon on top of workers’ hardhat is ideal, compared to other placements such as the chest or pocket, since the receiving beacon had a Line-of-Sight (LOS) with the reference transmitting beacons. However, mounting the receiving beacon on the chest was beneficial for body orientation prediction due to the signal blockage caused by the human body. The body orientation detection module used the existing RTLS reference transmitting beacons and a chest-mounted receiving beacon to continuously predict workers' body orientation. Unlike IMU sensors that require other equipment for recalibration due to the accumulation of the sensor orientation errors, the performance of the developed body

orientation detection module remained stable during its operation. Additionally, deploying BLE beacons can substantially reduce the system implementation cost since they are more affordable than IMU-equipped BLE beacons. The productivity state detection module utilized the accelerometer embedded in the hardhat- and chest-mounted beacons deployed for the RTLS and body orientation detection modules and additional wrist-mounted beacon. Although the record generation frequency of the BLE beacon is typically lower than IMU sensors, they reduce the implementation cost of the system. The developed productivity state detection module classified workers' productivity states into direct work and support work states, which could not be distinguished by the workers' RTLS location displacement method.

The innovation of the developed BLE-based worker monitoring system is that its modules can make most of the infrastructure required for RTLS, including the reference transmitting beacon and gateways. It can satisfy requirements for widespread on-site adoption, including cost efficiency and deployability to the construction sites. However, the system modules can perform independently and be deployed as required for different applications except for the body orientation detection module that relies on RTLS. The real-time worker location data can be used in various applications, including (i) visualizing congested workspaces occupied by different subcontractors and planning optimized traveling paths for workers and equipment; (ii) determining time spent in various workspaces; and (iii) automating log of workers' site attendance, arrival times, and exit times. The real-time worker body orientation data can be used in various applications, such as (i) measuring workers' awareness of danger zones and moving equipment on job sites; (ii) identifying the building component(s) on which a worker is working; and (iii) identifying the construction social networks among the workers of the different crews to control social distancing in pandemic situations. Last but not least, the real-time worker productivity states data can be used in applications, including (i) facilitating the calculation of workers' realistic productivity rate by detecting actual direct work time (work input); (ii) detecting the unusual ratio of delay to working time to understand the potential body fatigue associated with construction tasks; (iii) detecting the unusual ratio of travel to working time to detect unoptimized traveling paths required for construction activities on construction sites; (iv) providing a detailed breakdown of working time to train workforce by influencing actions on workforce training to increase productivity; and (v) detecting unusual high support work time or low direct work time to defect inefficiency/defect in tools and equipment.

4.3) Limitations and Future Study

Although this study highlighted the feasibility of deploying BLE beacon technology as a worker monitoring system, it had several limitations that require further investigation. Firstly, since the experiments were conducted in a controlled laboratory environment, the effect of distractions and noise, which generally exist in a construction environment, was minimized. Thus, the system's performance should be tested against the effects of other parameters such as weather conditions and proximity to the construction equipment. Secondly, for measuring the localization accuracy, the body position of the worker wearing the receivers was assumed to be almost vertical in this experiment. The assumption was valid for tracking workers in most activities; however, if the work involved physical movements of the worker's head, the accuracy of the RTLS could be affected due to the Non-Line of-Sight (NLoS) between the receiving and transmitting beacons. Moreover, the body orientation detection module was negatively affected once the chest-mounted beacon did not have a LoS with the reference beacons. Especially in places where it was impossible to install the reference beacons on building components (e.g., column) and in places where many workers were supposed to work simultaneously. Although the accuracy of the predicted body orientations sufficed for productivity monitoring applications, a higher level of accuracy might have been required for safety-related applications.

Furthermore, the accuracy of the productivity state detection module could be improved by using IMU sensors to measure angular velocity and magnetic fields data. More sophisticated sensors and infrastructure settings might also be required to improve the performance of the productivity state detection module by increasing the frequency of acceleration records. Collecting data from multiple participants and increasing the size of the training set could also improve the performance of the productivity state detection module. Although the data labeled as support work productivity state of workers were not necessarily related to their secondary tasks, collecting specific data based on the secondary tasks of workers can improve the productivity state detection module's performance. The productivity state data linkage to 4D BIM could generate in-depth insights into workers' construction activities on job sites. For example, workers' direct work productivity could be translated into construction activities by knowing the workspace and crew of workers provided by the 4D BIM (schedule and BIM model). Although only three repetitive construction activities were considered for training the productivity state detection module, the number of activities can be expanded in future studies to incorporate other construction activities, such as plumbing and carpentry.

The impact of various beacon configurations on the battery life was a practical factor for deployment that requires investigation. In addition, deploying the system in real conditions of the construction site with actual workers would be necessary to understand the deployability and issues such as cultural and possible behavioral changes. The productivity state detection can be further improved to detect construction activity by increasing the frequency of record generation. This can help solve the privacy issues of tracking workers on job sites since the module can detect the performed activities without knowing workers' names and body-mounted beacon IDs. Last but not least, incorporating other data acquisition techniques, such as computer vision, is beneficial not only to improve the performance of productivity state data but also to provide the work output of workers, resulting in a more realistic productivity assessment of workers. Additionally, the fuzed workers' data can be used to develop an alerting system for safety management on job sites. The system can be particularly advantageous to assessing workers' awareness of the danger zone by using the workers' body orientation and productivity state data, resulting in a low false alarm system.

References

[1] A. H. Behzadan, Z. Aziz, C. J. Anumba, and V. R. Kamat, "Ubiquitous location tracking for context-specific information delivery on construction sites," *Automation in Construction*, vol. 17, no. 6, pp. 737–748, 2008, doi: 10.1016/j.autcon.2008.02.002.

[2] W. Umer and M. K. Siddiqui, "Use of ultra wide band real-time location system on construction jobsites: Feasibility study and deployment alternatives," *International Journal of Environmental Research and Public Health*, vol. 17, no. 7, 2020, doi: 10.3390/ijerph17072219.

[3] M. W. Park and I. Brilakis, "Continuous localization of construction workers via integration of detection and tracking," *Automation in Construction*, vol. 72, pp. 129–142, 2016, doi: 10.1016/j.autcon.2016.08.039.

[4] J. Park, K. Kim, and Y. K. Cho, "Framework of Automated Construction-Safety Monitoring Using Cloud-Enabled BIM and BLE Mobile Tracking Sensors," *Journal of Construction Engineering and Management*, vol. 143, no. 2, pp. 1–12, 2017, doi: 10.1061/(ASCE)CO.1943-7862.0001223.

[5] B. Becerik-Gerber *et al.*, "Civil Engineering Grand Challenges: Opportunities for Data Sensing, Information Analysis, and Knowledge Discovery," *Journal of Computing in Civil Engineering*, vol. 28, no. 4, p. 04014013, 2014, doi: 10.1061/(asce)cp.1943-5487.0000290.

[6] T. Cheng, M. Venugopal, J. Teizer, and P. A. Vela, "Performance evaluation of ultra wideband technology for construction resource location tracking in harsh environments," *Automation in Construction*, vol. 20, no. 8, pp. 1173–1184, 2011, doi: 10.1016/j.autcon.2011.05.001.

[7] R. Maalek and F. Sadeghpour, "Accuracy assessment of ultra-wide band technology in locating dynamic resources in indoor scenarios," *Automation in Construction*, vol. 63, pp. 12–26, 2016, doi: 10.1016/j.autcon.2015.11.009.

[8] J. Teizer, T. Cheng, and Y. Fang, "Location tracking and data visualization technology to advance construction ironworkers' education and training in safety and productivity," *Automation in Construction*, vol. 35, pp. 53–68, 2013, doi: 10.1016/j.autcon.2013.03.004.

[9] W. Umer and M. K. Siddiqui, "Use of ultra wide band real-time location system on construction jobsites: Feasibility study and deployment alternatives," *International Journal of Environmental Research and Public Health*, vol. 17, no. 7, 2020, doi: 10.3390/ijerph17072219.

[10] M. Park, A. M. Asce, C. Koch, I. Brilakis, and M. Asce, "Three-Dimensional Tracking of Construction Resources Using an On-Site Camera System," vol. 26, no. August, pp. 541–549, 2012, doi: 10.1061/(ASCE)CP.1943-5487.0000168.

[11] T. Cheng, J. Teizer, G. C. Migliaccio, and U. C. Gatti, "Automated task-level activity analysis through fusion of real time location sensors and worker's thoracic posture data," *Automation in Construction*, vol. 29, pp. 24–39, 2013, doi: 10.1016/j.autcon.2012.08.003.

[12] D. Calvetti, P. Mêda, M. C. Gonçalves, and H. Sousa, "Worker 4.0: The future of sensored construction sites," *Buildings*, vol. 10, no. 10, pp. 1–22, 2020, doi: 10.3390/BUILDINGS10100169.

[13] S. J. Ray and J. Teizer, “Coarse head pose estimation of construction equipment operators to formulate dynamic blind spots,” *Advanced Engineering Informatics*, vol. 26, no. 1, pp. 117–130, 2012, doi: 10.1016/j.aei.2011.09.005.

[14] A. Sattineni and T. Schmidt, “Implementation of Mobile Devices on Jobsites in the Construction Industry,” *Procedia Engineering*, vol. 123, pp. 488–495, 2015, doi: 10.1016/j.proeng.2015.10.100.

[15] H. Tong, N. Xin, X. Su, T. Chen, and J. Wu, “A robust PDR/UWB integrated indoor localization approach for pedestrians in harsh environments,” *Sensors (Switzerland)*, vol. 20, no. 1, pp. 1–20, 2020, doi: 10.3390/s20010193.

[16] S. Sadowski and P. Spachos, “RSSI-Based Indoor Localization with the Internet of Things,” *IEEE Access*, vol. 6, pp. 30149–30161, 2018, doi: 10.1109/ACCESS.2018.2843325.

[17] L. Xia, G. Retscher, H. Tian, and H. Tian, “A case study on the feasibility and performance of an UWB-AoA real time location system for resources management of civil construction projects,” *Journal of Applied Geodesy*, vol. 4, no. 1, pp. 23–32, 2010, doi: 10.1515/jag.2010.003.

[18] F. Topak, M. K. Pekeriçli, and A. M. Tanyer, “Technological Viability Assessment of Bluetooth Low Energy Technology for Indoor Localization,” *Journal of Computing in Civil Engineering*, vol. 32, no. 5, pp. 1–13, 2018, doi: 10.1061/(ASCE)CP.1943-5487.0000778.

[19] J. Kunhoth, A. Karkar, S. Al-Maadeed, and A. Al-Attiyah, “Comparative analysis of computer-vision and BLE technology based indoor navigation systems for people with visual impairments,” *International Journal of Health Geographics*, vol. 18, no. 1, 2019, doi: 10.1186/s12942-019-0193-9.

[20] F. Zafari, A. Gkelias, and K. K. Leung, “A Survey of Indoor Localization Systems and Technologies,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019, doi: 10.1109/COMST.2019.2911558.

[21] H. Liu, H. Darabi, P. Banerjee, and J. Liu, “Survey of wireless indoor positioning techniques and systems,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 37, no. 6, pp. 1067–1080, 2007, doi: 10.1109/TSMCC.2007.905750.

[22] K. Huang, K. He, and X. Du, “A Hybrid Method to Improve the BLE-Based Indoor Positioning in a Dense Bluetooth Environment,” 2019, doi: 10.3390/s19020424.

[23] D. Kim, M. Liu, S. H. Lee, and V. R. Kamat, “Remote proximity monitoring between mobile construction resources using camera-mounted UAVs,” *Automation in Construction*, vol. 99, no. November 2018, pp. 168–182, 2019, doi: 10.1016/j.autcon.2018.12.014.

[24] N. Alishahi, M. Nik-Bakht, and M. M. Ouf, “A framework to identify key occupancy indicators for optimizing building operation using WiFi connection count data,” *Building and Environment*, vol. 200, no. May, p. 107936, 2021, doi: 10.1016/j.buildenv.2021.107936.

[25] S. Zhuang, “Real-Time Indoor Location Tracking in Construction Site Using BLE Beacon Trilateration,” 2020.

[26] J. Zhao, O. Seppänen, A. Peltokorpi, B. Badihi, and H. Olivieri, “Real-time resource tracking for analyzing value-adding time in construction,” *Automation in Construction*, vol. 104, no. January, pp. 52–65, 2019, doi: 10.1016/j.autcon.2019.04.003.

[27] Y. Fang, Y. K. Cho, S. Zhang, and E. Perez, “Case Study of BIM and Cloud–Enabled Real-Time RFID Indoor Localization for Construction Management Applications,” *Journal of Construction Engineering and Management*, vol. 142, no. 7, p. 05016003, 2016, doi: 10.1061/(asce)co.1943-7862.0001125.

[28] H. Bardareh and O. Moselhi, “Automated Data Acquisition for Indoor Localization and Tracking of Materials Onsite,” *Proceedings of the 37th International Symposium on Automation and Robotics in Construction, ISARC 2020: From Demonstration to Practical Use - To New Stage of Construction Robot*, no. Isarc, pp. 765–772, 2020, doi: 10.22260/isarc2020/0106.

[29] H. Kim and S. Han, “Accuracy improvement of real-time location tracking for construction workers,” *Sustainability (Switzerland)*, vol. 10, no. 5, pp. 1–16, 2018, doi: 10.3390/su10051488.

[30] O. Moselhi, H. Bardareh, and Z. Zhu, “Automated data acquisition in construction with remote sensing technologies,” *Applied Sciences (Switzerland)*, vol. 10, no. 8, pp. 1–31, 2020, doi: 10.3390/APP10082846.

[31] A. Motamedi, M. M. Soltani, and A. Hammad, “Localization of RFID-equipped assets during the operation phase of facilities,” *Advanced Engineering Informatics*, vol. 27, no. 4, pp. 566–579, 2013, doi: 10.1016/j.aei.2013.07.001.

[32] A. Montaser and O. Moselhi, “RFID indoor location identification for construction projects,” *Automation in Construction*, vol. 39, pp. 167–179, 2014, doi: 10.1016/j.autcon.2013.06.012.

[33] G. Li, E. Geng, Z. Ye, Y. Xu, J. Lin, and Y. Pang, “Indoor positioning algorithm based on the improved rssi distance model,” *Sensors (Switzerland)*, vol. 18, no. 9, pp. 1–15, 2018, doi: 10.3390/s18092820.

[34] J. Park and Y. K. Cho, “Development and Evaluation of a Probabilistic Local Search Algorithm for Complex Dynamic Indoor Construction Sites,” *Journal of Computing in Civil Engineering*, vol. 31, no. 4, p. 04017015, 2017, doi: 10.1061/(asce)cp.1943-5487.0000658.

[35] M. Namian, S. M. Asce, A. Albert, A. M. Asce, and J. Feng, “Effect of Distraction on Hazard Recognition and Safety Risk Perception,” vol. 144, no. 4, pp. 1–11, 2018, doi: 10.1061/(ASCE)CO.1943-7862.0001459.

[36] N. Mohsin, S. Payandeh, D. Ho, and J. P. Gelinas, “Study of Activity Tracking through Bluetooth Low Energy-Based Network,” *Journal of Sensors*, vol. 2019, 2019, doi: 10.1155/2019/6876925.

[37] D. Won, S. Chi, and M. W. Park, “UAV-RFID Integration for Construction Resource Localization,” *KSCE Journal of Civil Engineering*, vol. 24, no. 6, pp. 1683–1695, 2020, doi: 10.1007/s12205-020-2074-y.

[38] Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy, “Smartphone-based indoor localization with bluetooth low energy beacons,” *Sensors (Switzerland)*, vol. 16, no. 5, pp. 1–20, 2016, doi: 10.3390/s16050596.

[39] A. Mackey, P. Spachos, L. Song, and K. N. Plataniotis, “Improving BLE Beacon Proximity Estimation Accuracy Through Bayesian Filtering,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3160–3169, 2020, doi: 10.1109/JIOT.2020.2965583.

[40] H. Park, J. Noh, and S. Cho, “Three-dimensional positioning system using Bluetooth low-energy beacons,” *International Journal of Distributed Sensor Networks*, vol. 12, no. 10, 2016, doi: 10.1177/1550147716671720.

[41] S. Subedi and J. Y. Pyun, “Practical Fingerprinting Localization for Indoor Positioning System by Using Beacons,” *Journal of Sensors*, vol. 2017, 2017, doi: 10.1155/2017/9742170.

[42] C. K. Ke, M. Y. Wu, Y. W. Chan, and K. C. Lu, “Developing a BLE beacon-based location system using location fingerprint positioning for smart home power management,” *Energies*, vol. 11, no. 12, 2018, doi: 10.3390/en11123464.

[43] S. N. Karabtcev, T. A. Khorosheva, and N. R. Kapkov, “BLE beacon interaction module and mobile application in the indoor-navigation system,” *2019 International Science and Technology Conference “EastConf”, EastConf 2019*, pp. 1–6, 2019, doi: 10.1109/Eastonf.2019.8725420.

[44] Y. Shi, W. Shi, X. Liu, and X. Xiao, “An RSSI classification and tracing algorithm to improve trilateration-based positioning,” *Sensors (Switzerland)*, vol. 20, no. 15, pp. 1–17, 2020, doi: 10.3390/s20154244.

[45] D. Sun, E. Wei, Z. Ma, C. Wu, and S. Xu, “Optimized cnns to indoor localization through ble sensors using improved pso,” *Sensors*, vol. 21, no. 6, pp. 1–20, 2021, doi: 10.3390/s21061995.

[46] M. Li, L. Zhao, D. Tan, and X. Tong, “BLE fingerprint indoor localization algorithm based on eight-neighborhood template matching,” *Sensors (Switzerland)*, vol. 19, no. 22, 2019, doi: 10.3390/s19224859.

[47] S. I. Sou, W. H. Lin, K. C. Lan, and C. S. Lin, “Indoor location learning over wireless fingerprinting system with particle markov chain model,” *IEEE Access*, vol. 7, no. January, pp. 8713–8725, 2019, doi: 10.1109/ACCESS.2019.2890850.

[48] T. M. T. Dinh, N. S. Duong, and K. Sandrasegaran, “Smartphone-Based Indoor Positioning Using BLE iBeacon and Reliable Lightweight Fingerprint Map,” *IEEE Sensors Journal*, vol. 20, no. 17, pp. 10283–10294, 2020, doi: 10.1109/JSEN.2020.2989411.

[49] X. Sun, H. Ai, J. Tao, T. Hu, and Y. Cheng, “BERT-ADLOC: A secure crowdsourced indoor localization system based on BLE fingerprints,” *Applied Soft Computing*, vol. 104, p. 107237, 2021, doi: 10.1016/j.asoc.2021.107237.

[50] M. Castillo-Cara, J. Lovón-Melgarejo, G. Bravo-Rocca, L. Orozco-Barbosa, and I. García-Varea, “An Analysis of Multiple Criteria and Setups for Bluetooth Smartphone-Based Indoor Localization Mechanism,” *Journal of Sensors*, vol. 2017, 2017, doi: 10.1155/2017/1928578.

[51] A. K. Taşkan and H. Alemdar, “Obstruction-aware signal-loss-tolerant indoor positioning using bluetooth low energy,” *Sensors (Switzerland)*, vol. 21, no. 3, pp. 1–28, 2021, doi: 10.3390/s21030971.

[52] C. Information, “A Robust Indoor Positioning Method based on Bluetooth Low Energy with Separate”.

[53] S. H. Baek and S. H. Cha, “The trilateration-based BLE Beacon system for analyzing user-identified space usage of New Ways of Working offices,” *Building and Environment*, vol. 149, no. December 2018, pp. 264–274, 2019, doi: 10.1016/j.buildenv.2018.12.030.

[54] V. Cantón Paterna, A. Calveras Augé, J. Paradells Aspas, and M. A. Pérez Bullones, “A Bluetooth Low Energy Indoor Positioning System with Channel Diversity, Weighted Trilateration and Kalman Filtering,” *Sensors (Basel, Switzerland)*, vol. 17, no. 12, 2017, doi: 10.3390/s17122927.

[55] S. Sadowski, P. Spachos, and K. N. Plataniotis, “Memoryless Techniques and Wireless Technologies for Indoor Localization with the Internet of Things,” *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 10996–11005, 2020, doi: 10.1109/JIOT.2020.2992651.

[56] Y. Huang, A. Hammad, and Z. Zhu, “Providing proximity alerts to workers on construction sites using Bluetooth Low Energy RTLS,” *Automation in Construction*, vol. 132, no. July, p. 103928, 2021, doi: 10.1016/j.autcon.2021.103928.

[57] J. Park, K. Kim, and Y. K. Cho, “Framework of Automated Construction-Safety Monitoring Using Cloud-Enabled BIM and BLE Mobile Tracking Sensors,” *Journal of Construction Engineering and Management*, vol. 143, no. 2, p. 05016019, 2017, doi: 10.1061/(asce)co.1943-7862.0001223.

[58] K. Chan, J. Louis, and A. Albert, “Incorporating worker awareness in the generation of hazard proximity warnings,” *Sensors (Switzerland)*, vol. 20, no. 3, 2020, doi: 10.3390/s20030806.

[59] L. Mohanty, S. Chae, and Y. Yang, “Identifying productive working patterns at construction sites using BLE sensor networks,” *Developments in the Built Environment*, vol. 4, no. March, p. 100025, 2020, doi: 10.1016/j.dibe.2020.100025.

[60] J. Zhao, H. Olivieri, O. Seppänen, A. Peltokorpi, B. Badihi, and P. Lundström, “Data analysis on applying real time tracking in production control of construction,” *IEEE International Conference on Industrial Engineering and Engineering Management*, vol. 2017-Decem, pp. 573–577, 2018, doi: 10.1109/IEEM.2017.8289956.

[61] J. Zhao, O. Seppänen, and A. Peltokorpi, “Applying heat maps to define workspace in construction based on real-time tracking system with coordinate positioning information,” *IGLC 28 - 28th Annual Conference of the International Group for Lean Construction 2020*, pp. 841–852, 2020, doi: 10.24928/2020/0014.

[62] C. Zhou and L. Y. Ding, “Safety barrier warning system for underground construction sites using Internet-of-Things technologies,” *Automation in Construction*, vol. 83, no. May, pp. 372–389, 2017, doi: 10.1016/j.autcon.2017.07.005.

[63] W. Wu, H. Yang, D. A. S. Chew, S. hua Yang, A. G. F. Gibb, and Q. Li, “Towards an autonomous real-time tracking system of near-miss accidents on construction sites,” *Automation in Construction*, vol. 19, no. 2, pp. 134–141, 2010, doi: 10.1016/j.autcon.2009.11.017.

[64] S. Chae and T. Yoshida, “Application of RFID technology to prevention of collision accident with heavy equipment,” *Automation in Construction*, vol. 19, no. 3, pp. 368–374, 2010, doi: 10.1016/j.autcon.2009.12.008.

[65] Y. Fang, Y. K. Cho, S. Zhang, and E. Perez, “Case Study of BIM and Cloud–Enabled Real-Time RFID Indoor Localization for Construction Management Applications,” *Journal of Construction Engineering and Management*, vol. 142, no. 7, p. 05016003, 2016, doi: 10.1061/(asce)co.1943-7862.0001125.

[66] P. Kriz, F. Maly, and T. Kozel, “Improving Indoor Localization Using Bluetooth Low Energy Beacons,” *Mobile Information Systems*, vol. 2016, 2016, doi: 10.1155/2016/2083094.

[67] S. Aguilar, R. Vidal, and C. Gomez, “Opportunistic sensor data collection with bluetooth low energy,” *Sensors (Switzerland)*, vol. 17, no. 1, 2017, doi: 10.3390/s17010159.

[68] J. M. Gómez-de-Gabriel, J. A. Fernández-Madrigal, A. López-Arquillos, and J. C. Rubio-Romero, “Monitoring harness use in construction with BLE beacons,” *Measurement: Journal of the International Measurement Confederation*, vol. 131, pp. 329–340, 2019, doi: 10.1016/j.measurement.2018.07.093.

[69] H. Zhu and T. Alsharari, “An Improved RSSI-Based Positioning Method Using Sector Transmission Model and Distance Optimization Technique,” *International Journal of Distributed Sensor Networks*, vol. 2015, 2015, doi: 10.1155/2015/587195.

[70] S. Lee, J. Kim, and N. Moon, “Random forest and WiFi fingerprint-based indoor location recognition system using smart watch,” *Human-centric Computing and Information Sciences*, vol. 9, no. 1, 2019, doi: 10.1186/s13673-019-0168-7.

[71] C. H. Hing *et al.*, “Map-Based Localization Indoor Environment for Object Tracking using RF Trackers,” *IOP Conference Series: Materials Science and Engineering*, vol. 705, no. 1, pp. 0–7, 2019, doi: 10.1088/1757-899X/705/1/012036.

[72] Z. Iqbal *et al.*, “Accurate real time localization tracking in a clinical environment using Bluetooth Low Energy and deep learning,” *PLoS ONE*, vol. 13, no. 10, pp. 1–13, 2018, doi: 10.1371/journal.pone.0205392.

[73] R. R. Tirumalareddy, “BLE Beacon Based Indoor Positioning System in an Office Building using Machine Learning,” no. June, 2020.

[74] T. Morita, K. Taki, M. Fujimoto, H. Suwa, Y. Arakawa, and K. Yasumoto, “Beacon-Based Time-Spatial Recognition toward Automatic Daily Care Reporting for Nursing Homes,” *Journal of Sensors*, vol. 2018, 2018, doi: 10.1155/2018/2625195.

[75] T. Wattananavin, K. Sengchuai, N. Jindapetch, and A. Booranawong, “A Comparative Study of RSSI-Based Localization Methods: RSSI Variation Caused by Human Presence and Movement,” *Sensing and Imaging*, vol. 21, no. 1, pp. 1–20, 2020, doi: 10.1007/s11220-020-00296-1.

[76] A. Booranawong, K. Sengchuai, N. Jindapetch, and H. Saito, “An investigation of min-max method problems for rssi-based indoor localization: Theoretical and experimental studies,” *Engineering and Applied Science Research*, vol. 47, no. 3, pp. 313–325, 2020, doi: 10.14456/easr.2020.34.

[77] C. Takenga, T. Peng, and K. Kyamakya, “Post-processing of fingerprint localization using Kalman filter and map-matching techniques,” *International Conference on Advanced Communication Technology, ICACT*, vol. 3, pp. 2029–2034, 2007, doi: 10.1109/ICACT.2007.358771.

[78] S. W. Smith, “Digital Signal Processing, Chapter 15,” *California Technical Publishing*, pp. 277–284, 1999.

[79] H. v. Ravinder, “Determining The Optimal Values Of Exponential Smoothing Constants – Does Solver Really Work?,” *American Journal of Business Education (AJBE)*, vol. 9, no. 1, pp. 1–14, 2016, doi: 10.19030/ajbe.v9i1.9574.

[80] P. Ji, D. Xiong, P. Wang, and J. Chen, “A study on exponential smoothing model for load forecasting,” *Asia-Pacific Power and Energy Engineering Conference, APPEEC*, no. 2, 2012, doi: 10.1109/APPEEC.2012.6307555.

[81] S. Gupta, A. P. Singh, D. Deb, and S. Ozana, “Kalman filter and variants for estimation in 2dof serial flexible link and joint using fractional order pid controller,” *Applied Sciences (Switzerland)*, vol. 11, no. 15, 2021, doi: 10.3390/app11156693.

[82] A. Hosny, M. Nik-Bakht, and O. Moselhi, “Workspace planning in construction: non-deterministic factors,” *Automation in Construction*, vol. 116, no. April, 2020, doi: 10.1016/j.autcon.2020.103222.

[83] M. S. Milgram, “Does a point lie inside a polygon?,” *Journal of Computational Physics*, vol. 84, no. 1, pp. 134–144, 1989, doi: 10.1016/0021-9991(89)90185-X.

[84] K. O. Keefe, “Detecting and Correcting for Human Obstacles in,” no. 1, 2020.

[85] M. A. al Mamun, D. V. Anaya, F. Wu, and M. R. Yuce, “Landmark-assisted compensation of user’s body shadowing on rssi for improved indoor localisation with chest-mounted wearable device,” *Sensors*, vol. 21, no. 16, 2021, doi: 10.3390/s21165405.

[86] W. R. Hydoxdwlqj, G. Rswlrqv, D. Q. G. Pdnlqj, D. G. Frqvlghulqj, and H. Sulqflsohv, “3Urfhhglqjv Ri Wkh :Lqwhu 6Lpxodwlrq &Rqihuhqfh : . 9 &Kdq $ ’ $Peurjlr * =Dfkduhzlf] 1 0Xvwdihh * :Dlqhu Dqg (3Djh Hgv,” pp. 4220–4227, 2017.

[87] M. Kim, S. Jung, and J. W. Kang, “Artificial neural network-based residential energy consumption prediction models considering residential building information and user features in South Korea,” *Sustainability (Switzerland)*, vol. 12, no. 1, 2020, doi: 10.3390/su12010109.

[88] P. Jamal, M. Ali, R. H. Faraj, P. J. M. Ali, and R. H. Faraj, “1-6 Data Normalization and Standardization: A Technical Report,” *Machine Learning Technical Reports*, vol. 1, no. 1, pp. 1–6, 2014, [Online]. Available: https://docs.google.com/document/d/1x0A1nUz1WWtMCZb5oVzF0SVMY7a_58KQulqQVT8LaVA/edit#

[89] H. Li, J. Wang, M. Tang, and X. Li, “Polarization-dependent effects of an Airy beam due to the spin-orbit coupling,” *Journal of the Optical Society of America A: Optics and Image Science, and Vision*, vol. 34, no. 7, pp. 1114–1118, 2017, doi: 10.1002/ecs2.1832.

[90] C. Corbière, N. Thome, A. Bar-Hen, M. Cord, and P. Pérez, “Addressing failure prediction by learning model confidence,” *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, pp. 1–12, 2019.

[91] O. Moselhi, R. Aly, and A. Hassanein, “Labour productivity in building construction: A field study,” *Proceedings, Annual Conference - Canadian Society for Civil Engineering*, vol. 3, no. August 2009, pp. 2399–2408, 2011.

[92] N. Sharma, V. Jain, and A. Mishra, “An Analysis of Convolutional Neural Networks for Image Classification,” *Procedia Computer Science*, vol. 132, no. Iccids, pp. 377–384, 2018, doi: 10.1016/j.procs.2018.05.198.

[93] L. Nanni, S. Ghidoni, and S. Brahnam, “Handcrafted vs. non-handcrafted features for computer vision classification,” *Pattern Recognition*, vol. 71, pp. 158–172, 2017, doi: 10.1016/j.patcog.2017.05.025.

[94] I. A. Lawal and S. Bano, “Deep Human Activity Recognition with Localisation of Wearable Sensors,” *IEEE Access*, vol. 8, pp. 155060–155070, 2020, doi: 10.1109/ACCESS.2020.3017681.

[95] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[96] B. Mele and G. Altarelli, “Lepton spectra as a measure of b quark polarization at LEP,” *Physics Letters B*, vol. 299, no. 3–4, pp. 345–350, 1993, doi: 10.1016/0370-2693(93)90272-J.

[97] H. Zeng *et al.*, “A LightGBM-Based EEG Analysis Method for Driver Mental States Classification,” *Computational Intelligence and Neuroscience*, vol. 2019, 2019, doi: 10.1155/2019/3761203.

[98] M. O. al Kalaa, W. Balid, N. Bitar, and H. H. Refai, “Evaluating Bluetooth Low Energy in realistic wireless environments,” *IEEE Wireless Communications and Networking Conference, WCNC*, vol. 2016-Septe, no. April, 2016, doi: 10.1109/WCNC.2016.7564809.

[99] J. Choi, B.-J. Lee, and B.-T. Zhang, “Human Body Orientation Estimation using Convolutional Neural Network,” 2016, [Online]. Available: http://arxiv.org/abs/1609.01984

[100] V. Marotto, M. Sole, and T. Dessì, “Orientation Analysis through a Gyroscope Sensor for Indoor Navigation Systems,” *The Fourth International Conference on Sensor Device Technologies and Applications*, no. c, pp. 85–90, 2013, [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=sensordevices_2013_6_40_20221

[101] A. E. Maxwell, T. A. Warner, and L. A. Guillén, “Accuracy assessment in convolutional neural network-based deep learning remote sensing studies—part 2: Recommendations and best practices,” *Remote Sensing*, vol. 13, no. 13, 2021, doi: 10.3390/rs13132591.

[102] A. Tharwat, “Classification assessment methods,” *Applied Computing and Informatics*, vol. 17, no. 1, pp. 168–192, 2018, doi: 10.1016/j.aci.2018.08.003.

Appendices

Appendix 1) RTLS's Python Code

The results

Importing the Required Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
pd.set_option('max_colwidth', 20000000)
from sklearn import metrics
from pykalman import KalmanFilter
import matplotlib.pyplot as plt
import math
```

Importing the Raw Dataset from Elasticsearch

```
df=pd.read_excel(r'G:\Localization\210418\210418.xlsx')
df=df.sort_values(by=['timestamp'])
df.head()
```

```
                    _id          _index  _score _type  \
```

8206	1618760836758-0000a7c0	diract-proximity	NaN	_doc
8205	1618760838159-0000a7c0	diract-proximity	NaN	_doc
8204	1618760839729-0000a7c0	diract-proximity	NaN	_doc
8203	1618760840946-0000a7c0	diract-proximity	NaN	_doc
8202	1618760842341-0000a7c0	diract-proximity	NaN	_doc

	acceleration	batteryPercentage	cyclicCount	instanceId
8206	['-0.125','-0.062','1']	57	0	0000a7c0
8205	['-0.125','-0.062','1']	57	1	0000a7c0
8204	['-0.125','-0.062','1']	57	2	0000a7c0
8203	['-0.125','-0.062','1']	57	3	0000a7c0
8202	['-0.125','-0.062','1']	57	5	0000a7c0

	nearest
8206	[{'instanceId':'00000058','rssi':-68},{'instanceId':'00000060','rssi':-73}]
8205	[{'instanceId':'00000060','rssi':-73},{'instanceId':'0000004e','rssi':-75}]
8204	[{'instanceId':'00000058','rssi':-68},{'instanceId':'00000060','rssi':-73}]
8203	[{'instanceId':'00000061','rssi':-69},{'instanceId':'00000060','rssi':-73}]
8202	[{'instanceId':'00000061','rssi':-71},{'instanceId':'0000004e','rssi':-75},{'instanceId':'00000060','rssi':-75}]

	timestamp
8206	Apr 18, 2021 @ 11:47:16

8205 Apr 18, 2021 @ 11:47:18

8204 Apr 18, 2021 @ 11:47:19

8203 Apr 18, 2021 @ 11:47:20

8202 Apr 18, 2021 @ 11:47:22

Pre-Processing the Elasticsearch Raw Dataset

The three instances (detected transmitters) and their corresponding RSSI values are defined. Also, a column for the Puck is added.

```
instance_1=df['nearest'].str.slice(16,24,1)
instance_2=df['nearest'].str.slice(53,61,1)
instance_3=df['nearest'].str.slice(90,98,1)

rssi_1=df['nearest'].str.slice(33,36,1)
rssi_2=df['nearest'].str.slice(70,73,1)
rssi_3=df['nearest'].str.slice(107,110,1)
Puck=df['instanceId']

Processed_dataset=pd.DataFrame(df['timestamp'])
Processed_dataset['instance_1']=instance_1
Processed_dataset['instance_2']=instance_2
Processed_dataset['instance_3']=instance_3
```

```
    Processed_dataset['rssi_1']=rssi_1
    Processed_dataset['rssi_2']=rssi_2
    Processed_dataset['rssi_3']=rssi_3
    Processed_dataset['rssi_3']=rssi_3
    Processed_dataset['Puck']=Puck

    Processed_dataset=Processed_dataset.reset_index(drop=True)
    Processed_dataset=Processed_dataset[Processed_dataset!='']
    Processed_dataset.dropna(inplace=True)
    Processed_dataset=Processed_dataset.reset_index(drop=True)

    # Rarely, the dataset coming from the Kibana has a couple of string values in the RSSI numeric columns.
    # Thus, in order for the RSSI columns to be readable for the ML models, we need to remove string characters from them and convert them to integers.
    # Since by applying str.extract function the negative sign of RSSI were removed, we need to multiply them by (-1).

    Processed_dataset['rssi_1']=pd.DataFrame(Processed_dataset['rssi_1'])['rssi_1'].str.extract(r'(\d+)', expand=False)
    Processed_dataset['rssi_2']=pd.DataFrame(Processed_dataset['rssi_2'])['rssi_2'].str.extract(r'(\d+)', expand=False)
    Processed_dataset['rssi_3']=pd.DataFrame(Processed_dataset['rssi_3'])['rssi_3'].str.extract(r'(\d+)', expand=False)

    Processed_dataset.dropna(inplace=True)
```

```
Processed_dataset['rssi_1']=Processed_dataset['rssi_1'].astype(int)
Processed_dataset['rssi_2']=Processed_dataset['rssi_2'].astype(int)
Processed_dataset['rssi_3']=Processed_dataset['rssi_3'].astype(int)

Processed_dataset['rssi_1']=Processed_dataset['rssi_1']*-1
Processed_dataset['rssi_2']=Processed_dataset['rssi_2']*-1
Processed_dataset['rssi_3']=Processed_dataset['rssi_3']*-1

Processed_dataset.head()
```

	timestamp	instance_1	instance_2	instance_3	rssi_1	rssi_2	rssi_3	Puck
0	Apr 18, 2021 @ 11:47:22	00000061	0000004e	00000060	-71	-75	-75	0000a7c0
1	Apr 18, 2021 @ 11:47:23	00000061	0000004e	00000060	-71	-75	-75	0000a7c0
2	Apr 18, 2021 @ 11:47:25	00000058	00000061	00000060	-68	-74	-75	0000a7c0
3	Apr 18, 2021 @ 11:47:45	00000058	00000061	00000060	-68	-70	-74	0000a7c0
4	Apr 18, 2021 @ 11:47:48	00000058	00000061	00000060	-67	-71		

```
4    -73  0000a7c0
```

Converting the Reference Transmitters ID's to (x,y) Coordinates.

```
    Processed_dataset["X1"]=Processed_dataset["instance_1"]
    Processed_dataset["X2"]=Processed_dataset["instance_2"]
    Processed_dataset["X3"]=Processed_dataset["instance_3"]

    Processed_dataset.rename(columns={"instance_1":"Y1"},inplace=True)
    Processed_dataset.rename(columns={"instance_2":"Y2"},inplace=True)
    Processed_dataset.rename(columns={"instance_3":"Y3"},inplace=True)

    # This requires the user to input the (x,y) coordinates for the fixed transmitters on-site.

    x_axis={'00000058':3,'00000059':0,'00000060':6,'00000061':3,'0000004d':0,'0000004e':6}
    y_axis={'00000058':0,'00000059':3,'00000060':0,'00000061':3,'0000004d':0,'0000004e':3}

    Processed_dataset["X1"]=pd.DataFrame(Processed_dataset["Y1"].replace(x_axis))
    Processed_dataset["X2"]=pd.DataFrame(Processed_dataset["Y2"].replace(x_axis))
    Processed_dataset["X3"]=pd.DataFrame(Processed_dataset["Y3"].replace(x_axis))

    Processed_dataset["Y1"]=pd.DataFrame(Processed_dataset["Y1"].replace(y_axis))
```

```
Processed_dataset["Y2"]=pd.DataFrame(Processed_dataset["Y2"].replace(y_axis))
Processed_dataset["Y3"]=pd.DataFrame(Processed_dataset["Y3"].replace(y_axis))

# Converting string values of coordiantes to float
df=Processed_dataset
df[['Y1', 'Y2', 'Y3','X1', 'X2', 'X3']]=df[['Y1', 'Y2', 'Y3','X1', 'X2', 'X3']].apply(lambda x: pd.to_numeric(x, errors = 'coerce')).dropna()

df.head()
```

	timestamp	Y1	Y2	Y3	rssi_1	rssi_2	rssi_3	Puck	\
0	Apr 18, 2021 @ 11:47:22	3.0	3.0	0.0	-71	-75	-75	0000a7c0	
1	Apr 18, 2021 @ 11:47:23	3.0	3.0	0.0	-71	-75	-75	0000a7c0	
2	Apr 18, 2021 @ 11:47:25	0.0	3.0	0.0	-68	-74	-75	0000a7c0	
3	Apr 18, 2021 @ 11:47:45	0.0	3.0	0.0	-68	-70	-74	0000a7c0	
4	Apr 18, 2021 @ 11:47:48	0.0	3.0	0.0	-67	-71	-73	0000a7c0	

	X1	X2	X3
0	3.0	6.0	6.0
1	3.0	6.0	6.0
2	3.0	3.0	6.0
3	3.0	3.0	6.0
4	3.0	3.0	6.0

Determining the Types of the Records

```
# Calculating the distance between the three detected transmitters
df['dis_12']=np.sqrt(((df['X1']-df['X2'])**2)+((df['Y1']-df['Y2'])**2))
df['dis_13']=np.sqrt(((df['X1']-df['X3'])**2)+((df['Y1']-df['Y3'])**2))
df['dis_23']=np.sqrt(((df['X2']-df['X3'])**2)+((df['Y2']-df['Y3'])**2))

# Type (1) is the Logical recaords

# Condition1
type1=df[(df['dis_12']<4.25)&(df['dis_13']<4.25)&(df['dis_23']<4.25)]
type1=type1.reset_index(drop=True).drop(['dis_12','dis_13','dis_23'],axis=1)

# Type (2) is the Semi-Logical recaords

# Condition1
type2=df[~((df['dis_12']<4.25)&(df['dis_13']<4.25)&(df['dis_23']<4.25))]
# Condition2
type2=type2[type2['dis_12']<4.25]
type2=type2.reset_index(drop=True).drop(['dis_12','dis_13','dis_23'],axis=1)
```

Semi-Logical to Logical Converter Model

```
    # Determining the clue point
    type2['Xfin']=((((type2['X1']+type2['X3'])/2)+((type2['X2']+type2['X3'])/2)))/2
    type2['yfin']=((((type2['Y1']+type2['Y3'])/2)+((type2['Y2']+type2['Y3'])/2)))/2

    # Finding the distance of the clue point from the transmitters
    type2['tr_60']=np.sqrt(((type2['Xfin']-6)**2)+((type2['yfin']-0)**2))
    type2['tr_4e']=np.sqrt(((type2['Xfin']-6)**2)+((type2['yfin']-3)**2))
    type2['tr_61']=np.sqrt(((type2['Xfin']-3)**2)+((type2['yfin']-3)**2))
    type2['tr_58']=np.sqrt(((type2['Xfin']-3)**2)+((type2['yfin']-0)**2))
    type2['tr_4d']=np.sqrt(((type2['Xfin']-6)**2)+((type2['yfin']-0)**2))
    type2['tr_59']=np.sqrt(((type2['Xfin']-6)**2)+((type2['yfin']-3)**2))

    #### The already detected transmitters shoud be omitted
    def my_fun(x,X1,Y1,X2,Y2,X3,Y3,tr_60,tr_4e,tr_61,tr_58,tr_4d,tr_59):

        if ((x[X1]==6)&(x[Y1]==0))|((x[X2]==6)&(x[Y2]==0))|((x[X3]==6)&(x[Y3]==0)):
            x[tr_60]=999999
        else:
            x[tr_60]=x[tr_60]
        return x
```

```
type2=type2.apply(lambda x:my_fun(x,'X1','Y1','X2','Y2','X3','Y3','tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59'), axis=1)

def my_fun(x,X1,Y1,X2,Y2,X3,Y3,tr_60,tr_4e,tr_61,tr_58,tr_4d,tr_59):

    if ((x[X1]==6)&(x[Y1]==3))|((x[X2]==6)&(x[Y2]==3))|((x[X3]==6)&(x[Y3]==3)):
        x[tr_4e]=999999
    else:
        x[tr_4e]=x[tr_4e]
    return x
type2=type2.apply(lambda x:my_fun(x,'X1','Y1','X2','Y2','X3','Y3','tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59'), axis=1)

def my_fun(x,X1,Y1,X2,Y2,X3,Y3,tr_60,tr_4e,tr_61,tr_58,tr_4d,tr_59):

    if ((x[X1]==3)&(x[Y1]==3))|((x[X2]==3)&(x[Y2]==3))|((x[X3]==3)&(x[Y3]==3)):
        x[tr_61]=999999
    else:
        x[tr_61]=x[tr_61]
    return x
type2=type2.apply(lambda x:my_fun(x,'X1','Y1','X2','Y2','X3','Y3','tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59'), axis=1)

def my_fun(x,X1,Y1,X2,Y2,X3,Y3,tr_60,tr_4e,tr_61,tr_58,tr_4d,tr_59):
```

```
    if ((x[X1]==3)&(x[Y1]==3))|((x[X2]==3)&(x[Y2]==0))|((x[X3]==3)&(x[Y3]==0)):
        x[tr_58]=999999
    else:
        x[tr_58]=x[tr_58]
    return x
type2=type2.apply(lambda x:my_fun(x,'X1','Y1','X2','Y2','X3','Y3','tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59'), axis=1)

def my_fun(x,X1,Y1,X2,Y2,X3,Y3,tr_60,tr_4e,tr_61,tr_58,tr_4d,tr_59):

    if ((x[X1]==6)&(x[Y1]==0))|((x[X2]==6)&(x[Y2]==0))|((x[X3]==6)&(x[Y3]==0)):
        x[tr_4d]=999999
    else:
        x[tr_4d]=x[tr_4d]
    return x
type2=type2.apply(lambda x:my_fun(x,'X1','Y1','X2','Y2','X3','Y3','tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59'), axis=1)

def my_fun(x,X1,Y1,X2,Y2,X3,Y3,tr_60,tr_4e,tr_61,tr_58,tr_4d,tr_59):

    if ((x[X1]==6)&(x[Y1]==3))|((x[X2]==6)&(x[Y2]==0))|((x[X3]==6)&(x[Y3]==0)):
        x[tr_59]=999999
    else:
        x[tr_59]=x[tr_59]
```

```
    return x
type2=type2.apply(lambda x:my_fun(x,'X1','Y1','X2','Y2','X3','Y3','tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59'), axis=1)


# The transmitter whose distance from the clue point is the minimum should replace the wrong detected transmitter along with its RSSI value.
type2['corrected_tr_X']=type2[['tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59']].idxmin(axis = 1)
type2['corrected_tr_Y']=type2[['tr_60','tr_4e','tr_61','tr_58','tr_4d','tr_59']].idxmin(axis = 1)

corrected_tr_X={'tr_58':'00000058','tr_59':'00000059','tr_60':'00000060','tr_61':'00000061','tr_4d':'0000004d','tr_4e':'0000004e'}
corrected_tr_Y={'tr_58':'00000058','tr_59':'00000059','tr_60':'00000060','tr_61':'00000061','tr_4d':'0000004d','tr_4e':'0000004e'}

type2['corrected_tr_X']=pd.DataFrame(type2['corrected_tr_X'].replace(corrected_tr_X))
type2['corrected_tr_Y']=pd.DataFrame(type2['corrected_tr_Y'].replace(corrected_tr_Y))

type2['corrected_tr_X']=pd.DataFrame(type2['corrected_tr_X'].replace(x_axis))
type2['corrected_tr_Y']=pd.DataFrame(type2['corrected_tr_Y'].replace(y_axis))

type2[['corrected_tr_X','corrected_tr_Y']]=type2[['corrected_tr_X','corrected_tr_Y']].apply(lambda x: x.astype(float))

type2['X3']=type2['corrected_tr_X']
type2['Y3']=type2['corrected_tr_Y']
type2=type2.drop(['corrected_tr_X','corrected_tr_Y'],axis=1)
```

```
type2=type2[['timestamp', 'Y1', 'Y2', 'Y3', 'rssi_1', 'rssi_2', 'rssi_3',
            'Puck','X1', 'X2', 'X3']]

all_data=pd.DataFrame(Processed_dataset['timestamp'])
data=pd.merge(all_data, pd.concat([type1,type2],axis=0), on="timestamp",how='outer').ffill().dropna()
data.head()
```

	timestamp	Y1	Y2	Y3	rssi_1	rssi_2	rssi_3	Puck	\
0	Apr 18, 2021 @ 11:47:22	3.0	3.0	0.0	-71.0	-75.0	-75.0	0000a7c0	
1	Apr 18, 2021 @ 11:47:23	3.0	3.0	0.0	-71.0	-75.0	-75.0	0000a7c0	
2	Apr 18, 2021 @ 11:47:25	0.0	3.0	0.0	-68.0	-74.0	-75.0	0000a7c0	
3	Apr 18, 2021 @ 11:47:45	0.0	3.0	0.0	-68.0	-70.0	-74.0	0000a7c0	
4	Apr 18, 2021 @ 11:47:48	0.0	3.0	0.0	-67.0	-71.0	-73.0	0000a7c0	

	X1	X2	X3
0	3.0	6.0	6.0
1	3.0	6.0	6.0
2	3.0	3.0	6.0
3	3.0	3.0	6.0
4	3.0	3.0	6.0

RSSI-distance Prediction Model

Importing the training dataset to train the RF model to predict the diatance from the RSSI values. The train set consistes four orientation sof the Receiving beacon (Puck) with respect to the transmitter.

```
Back_dataset=pd.read_csv(r"G:\Final RSSI-distance-1.80 m Height\Back\Back-dataset.csv")
Front_dataset=pd.read_csv(r"G:\Final RSSI-distance-1.80 m Height\Front\Front-dataset.csv")
Right_dataset=pd.read_csv(r"G:\Final RSSI-distance-1.80 m Height\Right\Right-dataset.csv")
Left_dataset=pd.read_csv(r"G:\Final RSSI-distance-1.80 m Height\Left\Lefr-Dataset.csv")

All_dataset=pd.DataFrame()
All_dataset=All_dataset.append([Back_dataset,Front_dataset,Right_dataset,Left_dataset])

"""
We need to filter our desired records based on their distnaces according to the Max diameter of one module.
e.g. here, the records with distnace from 0 to 4.25m are selected.
"""
Sub_dataset=All_dataset[All_dataset['distance']<=4.25]

"""
As a rule X shoud have a two dimensions (2D) and y should have one dimension (1D). If the taring set has only one attribute,
we should apply .to_numpy().reshape(-1,1) to the X to solve the problem.
```

```
    """
    X=pd.DataFrame(Sub_dataset['rssi'])
    #y=pd.DataFrame(Sub_dataset['distance'])
    y=Sub_dataset['distance']
```

Applying the RF model for the RSSI-distance prediction

```
    from sklearn.ensemble import RandomForestRegressor
    rfc = RandomForestRegressor(n_estimators=130,criterion='mse',max_depth=25,min_samples_leaf=8,bootstrap = True,max_features = 'sqrt')
    rfc.fit(X,y)

    distance_1=rfc.predict(pd.DataFrame(data['rssi_1']))
    distance_2=rfc.predict(pd.DataFrame(data['rssi_2']))
    distance_3=rfc.predict(pd.DataFrame(data['rssi_3']))

    distance_1=pd.DataFrame(distance_1).rename({0:'distance_1',}, axis=1)
    distance_2=pd.DataFrame(distance_2).rename({0:'distance_2',}, axis=1)
    distance_3=pd.DataFrame(distance_3).rename({0:'distance_3',}, axis=1)

    data=pd.concat([data, distance_1.reindex(Processed_dataset.index)], axis=1)
    data=pd.concat([data, distance_2.reindex(Processed_dataset.index)], axis=1)
```

```
data=pd.concat([data, distance_3.reindex(Processed_dataset.index)], axis=1)

data=data.drop(['rssi_1','rssi_2','rssi_3'], axis=1).dropna()
data=data[['timestamp','Puck','X1','Y1','X2','Y2','X3','Y3','distance_1','distance_2','distance_3']]
df = data

df.head()
```

	timestamp	Puck	X1	Y1	X2	Y2	X3	Y3	\
0	Apr 18, 2021 @ 11:47:22	0000a7c0	3.0	3.0	6.0	3.0	6.0	0.0	
1	Apr 18, 2021 @ 11:47:23	0000a7c0	3.0	3.0	6.0	3.0	6.0	0.0	
2	Apr 18, 2021 @ 11:47:25	0000a7c0	3.0	0.0	3.0	3.0	6.0	0.0	
3	Apr 18, 2021 @ 11:47:45	0000a7c0	3.0	0.0	3.0	3.0	6.0	0.0	
4	Apr 18, 2021 @ 11:47:48	0000a7c0	3.0	0.0	3.0	3.0	6.0	0.0	

	distance_1	distance_2	distance_3
0	3.471563	3.250526	3.250526
1	3.471563	3.250526	3.250526
2	3.361573	3.424589	3.250526
3	3.361573	3.384148	3.424589
4	2.908053	3.471563	3.427018

```
#data.to_excel(r"C:\Users\ali\Desktop\test_triangulation.xlsx",index=False)
```

Estimating the Locations through the Traingualtrion Models

```
#### Calculating the distance between the three detected transmitters
df['dis_12']=np.sqrt(((df['X1']-df['X2'])**2)+((df['Y1']-df['Y2'])**2))
df['dis_13']=np.sqrt(((df['X1']-df['X3'])**2)+((df['Y1']-df['Y3'])**2))
df['dis_23']=np.sqrt(((df['X2']-df['X3'])**2)+((df['Y2']-df['Y3'])**2))

#### Removing the outliers that a pair of identical transmitters exits in a record
df=df[(df['dis_12']!=0)&(df['dis_13']!=0)&(df['dis_23']!=0)]
```

Defining rules to check if a pair of circles intersect each other or are isolated.

```
#check1_1,check1_2
def my_fun(x):

    if x['dis_12']>(x['distance_1']+x['distance_2']):
        x['check1_1']=1
    else:
        x['check1_1']=0
    return x
```

```
df=df.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if x['dis_12']<abs(x['distance_1']-x['distance_2']):
        x['check1_2']=1
    else:
        x['check1_2']=0
    return x
df=df.apply(lambda x:my_fun(x), axis=1)



#check2_1,check2_2
def my_fun(x):

    if x['dis_23']>(x['distance_2']+x['distance_3']):
        x['check2_1']=1
    else:
        x['check2_1']=0
    return x
df=df.apply(lambda x:my_fun(x), axis=1)
```

```
def my_fun(x):

    if x['dis_23']<abs(x['distance_2']-x['distance_3']):
        x['check2_2']=1
    else:
        x['check2_2']=0
    return x
df=df.apply(lambda x:my_fun(x), axis=1)



#check3_1,check3_2
def my_fun(x):

    if x['dis_13']>(x['distance_1']+x['distance_3']):
        x['check3_1']=1
    else:
        x['check3_1']=0
    return x
df=df.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if x['dis_13']<abs(x['distance_1']-x['distance_3']):
        x['check3_2']=1
    else:
        x['check3_2']=0
    return x
df=df.apply(lambda x:my_fun(x), axis=1)
```

Categorizing the placements of the circles with respect to each other

Type1 (one circle is isolated and the other two intersect each other)

```
# In type1, we have 3 different sub-types based on which circle is isolated.

type1_1=df[(df['check1_1']==0)&(df['check1_2']==0)&
(df['check2_1']==1)&(df['check2_2']==0)&
(df['check3_1']==1)&(df['check3_2']==0)]

type1_2=df[(df['check1_1']==1)&(df['check1_2']==0)&
(df['check2_1']==0)&(df['check2_2']==0)&
(df['check3_1']==1)&(df['check3_2']==0)]

type1_3=df[(df['check1_1']==1)&(df['check1_2']==0)&
```

```
    (df['check2_1']==1)&(df['check2_2']==0)&
    (df['check3_1']==0)&(df['check3_2']==0)]
```

Triangulation Algorithm for Type1

type1_1

```
    #### Finding the segment a named 'intersections_a'

    def my_fun(x):

        if  x['distance_1']>x['distance_2']:
            x['intersections_a']=((x['distance_1']**2)-(x['distance_2']**2)+(x['dis_12']**2))/(2*x['dis_12'])
        else:
            x['intersections_a']=((x['distance_2']**2)-(x['distance_1']**2)+(x['dis_12']**2))/(2*x['dis_12'])
        return x
    type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

    #### Finding the segment h named 'intersections_h'

    def my_fun(x):
```

```
    if  x['distance_1']>x['distance_2']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_12'])*(x['X2']-x['X1'])
    else:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_12'])*(x['X1']-x['X2'])
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_12'])*(x['Y2']-x['Y1']))
```

```
    else:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
```

```
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle
```

```
def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X3']-x['X4_1'])**2)+((x['Y3']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X3']-x['X4_2'])**2)+((x['Y3']-x['Y4_2'])**2))
    return x
type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type1_1['location_X']=0
type1_1['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_X']=x['X4_2']
    else:
        x['location_X']=x['X4_1']
```

```
        return x
    type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

    def my_fun(x):

        if  x['intersection1_distance']>x['intersection2_distance']:
            x['location_Y']=x['Y4_2']
        else:
            x['location_Y']=x['Y4_1']
        return x
    type1_1=type1_1.apply(lambda x:my_fun(x), axis=1)

    type1_1=type1_1[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]

type1_2

    #### Finding the segment a named 'intersections_a'

    def my_fun(x):

        if  x['distance_2']>x['distance_3']:
            x['intersections_a']=((x['distance_2']**2)-(x['distance_3']**2)+(x['dis_23']**2))/(2*x['dis_23'])
```

```
    else:
        x['intersections_a']=((x['distance_3']**2)-(x['distance_2']**2)+(x['dis_23']**2))/(2*x['dis_23'])
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_23'])*(x['X3']-x['X2'])
```

```
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_23'])*(x['X2']-x['X3'])
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
```

```
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_2']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X1']-x['X4_1'])**2)+((x['Y1']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X1']-x['X4_2'])**2)+((x['Y1']-x['Y4_2'])**2))
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.
```

```
type1_2['location_X']=0
type1_2['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_X']=x['X4_2']
    else:
        x['location_X']=x['X4_1']
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_Y']=x['Y4_2']
    else:
        x['location_Y']=x['Y4_1']
    return x
type1_2=type1_2.apply(lambda x:my_fun(x), axis=1)

type1_2=type1_2[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]
```

type1_3

```
    #### Finding the segment a named 'intersections_a'

    def my_fun(x):

        if  x['distance_1']>x['distance_3']:
            x['intersections_a']=((x['distance_1']**2)-(x['distance_3']**2)+(x['dis_13']**2))/(2*x['dis_13'])
        else:
            x['intersections_a']=((x['distance_3']**2)-(x['distance_1']**2)+(x['dis_13']**2))/(2*x['dis_13'])
        return x
    type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

    #### Finding the segment h named 'intersections_h'

    def my_fun(x):

        if  x['distance_1']>x['distance_3']:
            x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
        else:
            x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
```

```
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_13'])*(x['X3']-x['X1'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_13'])*(x['X1']-x['X3'])
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)
```

Getting the pair of points

```
def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_1']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X2']-x['X4_1'])**2)+((x['Y2']-x['Y4_1'])**2))
```

```
    x['intersection2_distance']=np.sqrt(((x['X2']-x['X4_2'])**2)+((x['Y2']-x['Y4_2'])**2))
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type1_3['location_X']=0
type1_3['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_X']=x['X4_2']
    else:
        x['location_X']=x['X4_1']
    return x
type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
        if  x['intersection1_distance']>x['intersection2_distance']:
            x['location_Y']=x['Y4_2']
        else:
            x['location_Y']=x['Y4_1']
        return x
    type1_3=type1_3.apply(lambda x:my_fun(x), axis=1)

    type1_3=type1_3[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]
```

Type2 (all the circles intersect each other)

```
    # In type2, the three circles intersect each other

    type2=df[(df['check1_1']==0)&(df['check1_2']==0)&
    (df['check2_1']==0)&(df['check2_2']==0)&
    (df['check3_1']==0)&(df['check3_2']==0)]

    #### Finding intersections of circles of 1 and 2

    ######## Finding the segment a named 'intersections_a'
```

```
def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['intersections_a']=((x['distance_1']**2)-(x['distance_2']**2)+(x['dis_12']**2))/(2*x['dis_12'])
    else:
        x['intersections_a']=((x['distance_2']**2)-(x['distance_1']**2)+(x['dis_12']**2))/(2*x['dis_12'])
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)
```

```
def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_12'])*(x['X2']-x['X1'])
    else:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_12'])*(x['X1']-x['X2'])
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):
```

```
    if  x['distance_1']>x['distance_2']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
```

```
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X3']-x['X4_1'])**2)+((x['Y3']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X3']-x['X4_2'])**2)+((x['Y3']-x['Y4_2'])**2))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle
```

```
#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type2['location_X']=0
type2['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_X']=x['X4_2']
    else:
        x['location_P1_X']=x['X4_1']
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_Y']=x['Y4_2']
    else:
        x['location_P1_Y']=x['Y4_1']
```

```
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding intersections of circles of 2 and 3

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['intersections_a']=((x['distance_2']**2)-(x['distance_3']**2)+(x['dis_23']**2))/(2*x['dis_23'])
    else:
        x['intersections_a']=((x['distance_3']**2)-(x['distance_2']**2)+(x['dis_23']**2))/(2*x['dis_23'])
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
```

```
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_23'])*(x['X3']-x['X2'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_23'])*(x['X2']-x['X3'])
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)
```

```
#### Getting the pair of points

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_2']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X1']-x['X4_1'])**2)+((x['Y1']-x['Y4_1'])**2))
```

```
    x['intersection2_distance']=np.sqrt(((x['X1']-x['X4_2'])**2)+((x['Y1']-x['Y4_2'])**2))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type2['location_X']=0
type2['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_X']=x['X4_2']
    else:
        x['location_P2_X']=x['X4_1']
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_Y']=x['Y4_2']
    else:
        x['location_P2_Y']=x['Y4_1']
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding intersections of circles of 1 and 3

#### Finding the segment a named 'intersections_a'

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['intersections_a']=((x['distance_1']**2)-(x['distance_3']**2)+(x['dis_13']**2))/(2*x['dis_13'])
    else:
        x['intersections_a']=((x['distance_3']**2)-(x['distance_1']**2)+(x['dis_13']**2))/(2*x['dis_13'])
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'
```

```
def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_13'])*(x['X3']-x['X1'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_13'])*(x['X1']-x['X3'])
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_1']>x['distance_3']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
```

```
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
```

```
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X2']-x['X4_1'])**2)+((x['Y2']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X2']-x['X4_2'])**2)+((x['Y2']-x['Y4_2'])**2))
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type2['location_X']=0
type2['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
```

```
        x['location_P3_X']=x['X4_2']
    else:
        x['location_P3_X']=x['X4_1']
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P3_Y']=x['Y4_2']
    else:
        x['location_P3_Y']=x['Y4_1']
    return x
type2=type2.apply(lambda x:my_fun(x), axis=1)

#### Taking an average of the intersection points

def my_fun(x):

    x['location_X']=(x['location_P1_X']+x['location_P2_X']+x['location_P3_X'])/3
    x['location_Y']=(x['location_P1_Y']+x['location_P2_Y']+x['location_P3_Y'])/3
    return x
```

```
    type2=type2.apply(lambda x:my_fun(x), axis=1)

    type2=type2[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]
```

Type3 (two pairs of circles inrtersect with each other)

```
    # In type3, we have 3 different sub-type based on which circle intersects with the other circles.

    type3_1=df[(df['check1_1']==0)&(df['check1_2']==0)&
    (df['check2_1']==1)&(df['check2_2']==0)&
    (df['check3_1']==0)&(df['check3_2']==0)]

    type3_2=df[(df['check1_1']==0)&(df['check1_2']==0)&
    (df['check2_1']==0)&(df['check2_2']==0)&
    (df['check3_1']==1)&(df['check3_2']==0)]

    type3_3=df[(df['check1_1']==1)&(df['check1_2']==0)&
    (df['check2_1']==0)&(df['check2_2']==0)&
    (df['check3_1']==0)&(df['check3_2']==0)]

    ### type3_1
```

```
#### Finding intersections of the first pair

######## Finding the segment a named 'intersections_a'

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['intersections_a']=((x['distance_1']**2)-(x['distance_2']**2)+(x['dis_12']**2))/(2*x['dis_12'])
    else:
        x['intersections_a']=((x['distance_2']**2)-(x['distance_1']**2)+(x['dis_12']**2))/(2*x['dis_12'])
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    return x
```

```
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_12'])*(x['X2']-x['X1'])
    else:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_12'])*(x['X1']-x['X2'])
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)
```

```
#### Getting the pair of points

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_1']>x['distance_2']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X3']-x['X4_1'])**2)+((x['Y3']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X3']-x['X4_2'])**2)+((x['Y3']-x['Y4_2'])**2))
```

```
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type3_1['location_X']=0
type3_1['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_X']=x['X4_2']
    else:
        x['location_P1_X']=x['X4_1']
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_Y']=x['Y4_2']
    else:
        x['location_P1_Y']=x['Y4_1']
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Finding intersections of the second pair

#### Finding the segment a named 'intersections_a'

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['intersections_a']=((x['distance_1']**2)-(x['distance_3']**2)+(x['dis_13']**2))/(2*x['dis_13'])
    else:
        x['intersections_a']=((x['distance_3']**2)-(x['distance_1']**2)+(x['dis_13']**2))/(2*x['dis_13'])
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'
```

```
def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_13'])*(x['X3']-x['X1'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_13'])*(x['X1']-x['X3'])
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_1']>x['distance_3']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
```

```
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)
```

```
#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X2']-x['X4_1'])**2)+((x['Y2']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X2']-x['X4_2'])**2)+((x['Y2']-x['Y4_2'])**2))
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type3_1['location_X']=0
type3_1['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_X']=x['X4_2']
```

```
    else:
        x['location_P2_X']=x['X4_1']
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_Y']=x['Y4_2']
    else:
        x['location_P2_Y']=x['Y4_1']
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)

#### Taking an average of the intersection points

def my_fun(x):

    x['location_X']=(x['location_P1_X']+x['location_P2_X'])/2
    x['location_Y']=(x['location_P1_Y']+x['location_P2_Y'])/2
    return x
type3_1=type3_1.apply(lambda x:my_fun(x), axis=1)
```

```
    type3_1=type3_1[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]

type3_2

    #### Finding intersections of the first pair

    ######## Finding the segment a named 'intersections_a'

    def my_fun(x):

        if  x['distance_1']>x['distance_2']:
            x['intersections_a']=((x['distance_1']**2)-(x['distance_2']**2)+(x['dis_12']**2))/(2*x['dis_12'])
        else:
            x['intersections_a']=((x['distance_2']**2)-(x['distance_1']**2)+(x['dis_12']**2))/(2*x['dis_12'])
        return x
    type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

    #### Finding the segment h named 'intersections_h'

    def my_fun(x):
```

```
    if  x['distance_1']>x['distance_2']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_12'])*(x['X2']-x['X1'])
    else:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_12'])*(x['X1']-x['X2'])
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_12'])*(x['Y2']-x['Y1']))
```

```
    else:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
```

```
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y2']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_12'])*(x['Y1']-x['Y2']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_2']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X2']-x['X1']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_12'])*(x['X1']-x['X2']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle
```

```
def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X3']-x['X4_1'])**2)+((x['Y3']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X3']-x['X4_2'])**2)+((x['Y3']-x['Y4_2'])**2))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type3_2['location_X']=0
type3_2['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_X']=x['X4_2']
    else:
        x['location_P1_X']=x['X4_1']
```

```
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_Y']=x['Y4_2']
    else:
        x['location_P1_Y']=x['Y4_1']
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Finding intersections of the second pair

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['intersections_a']=((x['distance_2']**2)-(x['distance_3']**2)+(x['dis_23']**2))/(2*x['dis_23'])
    else:
        x['intersections_a']=((x['distance_3']**2)-(x['distance_2']**2)+(x['dis_23']**2))/(2*x['dis_23'])
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)
```

```
#### Finding the segment h named 'intersections_h'

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_23'])*(x['X3']-x['X2'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_23'])*(x['X2']-x['X3'])
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)
```

```
def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):
```

```
    if  x['distance_2']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
```

```
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X1']-x['X4_1'])**2)+((x['Y1']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X1']-x['X4_2'])**2)+((x['Y1']-x['Y4_2'])**2))
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type3_2['location_X']=0
type3_2['location_Y']=0

def my_fun(x):
```

```
    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_X']=x['X4_2']
    else:
        x['location_P2_X']=x['X4_1']
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_Y']=x['Y4_2']
    else:
        x['location_P2_Y']=x['Y4_1']
    return x
type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

#### Taking an average of the intersection points

def my_fun(x):

    x['location_X']=(x['location_P1_X']+x['location_P2_X'])/2
```

```
        x['location_Y']=(x['location_P1_Y']+x['location_P2_Y'])/2
        return x
    type3_2=type3_2.apply(lambda x:my_fun(x), axis=1)

    type3_2=type3_2[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]

type3_3

    #### Finding intersections of the first pair

    def my_fun(x):

        if  x['distance_2']>x['distance_3']:
            x['intersections_a']=((x['distance_2']**2)-(x['distance_3']**2)+(x['dis_23']**2))/(2*x['dis_23'])
        else:
            x['intersections_a']=((x['distance_3']**2)-(x['distance_2']**2)+(x['dis_23']**2))/(2*x['dis_23'])
        return x
    type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

    #### Finding the segment h named 'intersections_h'

    def my_fun(x):
```

```
    if  x['distance_2']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_2']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['P3_X']=x['X2']+(x['intersections_a']/x['dis_23'])*(x['X3']-x['X2'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_23'])*(x['X2']-x['X3'])
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
```

```
        x['P3_Y']=x['Y2']+((x['intersections_a']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
```

```
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y3']-x['Y2']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_23'])*(x['Y2']-x['Y3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_2']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X3']-x['X2']))
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_23'])*(x['X2']-x['X3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)
```

```
#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X1']-x['X4_1'])**2)+((x['Y1']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X1']-x['X4_2'])**2)+((x['Y1']-x['Y4_2'])**2))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type3_3['location_X']=0
type3_3['location_Y']=0

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_X']=x['X4_2']
    else:
```

```
        x['location_P1_X']=x['X4_1']
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P1_Y']=x['Y4_2']
    else:
        x['location_P1_Y']=x['Y4_1']
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment a named 'intersections_a'

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['intersections_a']=((x['distance_1']**2)-(x['distance_3']**2)+(x['dis_13']**2))/(2*x['dis_13'])
    else:
        x['intersections_a']=((x['distance_3']**2)-(x['distance_1']**2)+(x['dis_13']**2))/(2*x['dis_13'])
    return x
```

```
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the segment h named 'intersections_h'

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['intersections_h']=np.sqrt((x['distance_1']**2)-(x['intersections_a']**2))
    else:
        x['intersections_h']=np.sqrt((x['distance_3']**2)-(x['intersections_a']**2))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the point P3 (in vector form of X and Y)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['P3_X']=x['X1']+(x['intersections_a']/x['dis_13'])*(x['X3']-x['X1'])
    else:
        x['P3_X']=x['X3']+(x['intersections_a']/x['dis_13'])*(x['X1']-x['X3'])
    return x
```

```
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['P3_Y']=x['Y1']+((x['intersections_a']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['P3_Y']=x['Y3']+((x['intersections_a']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Getting the pair of points

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_1']=x['P3_X']+((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)
```

```
def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
    else:
        x['Y4_1']=x['P3_Y']-((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y3']-x['Y1']))
    else:
        x['X4_2']=x['P3_X']-((x['intersections_h']/x['dis_13'])*(x['Y1']-x['Y3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['distance_1']>x['distance_3']:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X3']-x['X1']))
```

```
    else:
        x['Y4_2']=x['P3_Y']+((x['intersections_h']/x['dis_13'])*(x['X1']-x['X3']))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Finding the distance of the intersection points from the center of the isolated circle

def my_fun(x):

    x['intersection1_distance']=np.sqrt(((x['X2']-x['X4_1'])**2)+((x['Y2']-x['Y4_1'])**2))
    x['intersection2_distance']=np.sqrt(((x['X2']-x['X4_2'])**2)+((x['Y2']-x['Y4_2'])**2))
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Determining the nearest intersection point to the isolated circle

#Just in case, if there is no record in this type, so we need manually create columns of 'location_X'
# and 'location_Y' in order for the model not to break.

type3_3['location_X']=0
type3_3['location_Y']=0
```

```
def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_X']=x['X4_2']
    else:
        x['location_P2_X']=x['X4_1']
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

def my_fun(x):

    if  x['intersection1_distance']>x['intersection2_distance']:
        x['location_P2_Y']=x['Y4_2']
    else:
        x['location_P2_Y']=x['Y4_1']
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

#### Taking an average of the intersection points
def my_fun(x):

    x['location_X']=(x['location_P1_X']+x['location_P2_X'])/2
```

```
    x['location_Y']=(x['location_P1_Y']+x['location_P2_Y'])/2
    return x
type3_3=type3_3.apply(lambda x:my_fun(x), axis=1)

type3_3=type3_3[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]
```

Type4 (the circles don't intersect each other)

```
type4=df[(df['check1_1']==1)&(df['check1_2']==0)&
(df['check2_1']==1)&(df['check2_2']==0)&
(df['check3_1']==1)&(df['check3_2']==0)]

# Min-Max method:
def my_fun(x):

    x['Xmin']=max((x['X1']-x['distance_1']),(x['X2']-x['distance_2']),(x['X3']-x['distance_3']))
    x['Xmax']=min((x['X1']+x['distance_1']),(x['X2']+x['distance_2']),(x['X3']+x['distance_3']))
    x['Ymin']=max((x['Y1']-x['distance_1']),(x['Y2']-x['distance_2']),(x['Y3']-x['distance_3']))
    x['Ymax']=min((x['Y1']+x['distance_1']),(x['Y2']+x['distance_2']),(x['Y3']+x['distance_3']))
    x['location_X']=(x['Xmin']+x['Xmax'])/2
    x['location_Y']=(x['Ymin']+x['Ymax'])/2
    return x
```

type4=type4.apply(lambda x:my_fun(x), axis=1)

type4=type4[['timestamp', 'Puck', 'X1', 'Y1', 'distance_1','distance_2', 'distance_3','location_X', 'location_Y']]

Concat all the estimated locations of the four categories

all_locations=pd.concat([type1_1,type1_2,type1_3,type2,type3_1,type3_2,type3_3,type4],axis=0).sort_values('timestamp').reset_index(drop=True)

Post-Processing the Estimated Locations

Shifting the estimated locations toward to the strongest transmitter

In rare cases, the estimated distance is zero. So, this can affects the divison lines bellow. Hence, we add 0.000001 to the distances.

def my_fun(x):

x['distance_1']=x['distance_1']+0.000001

x['distance_2']=x['distance_2']+0.000001

x['location_X']=((x['location_X']+x['X1']*(0.2*(x['distance_2'])/x['distance_1']))/(1+(0.2*(x['distance_2'])/x['distance_1'])))

x['location_Y']=((x['location_Y']+x['Y1']*(0.2*(x['distance_2'])/x['distance_1']))/(1+(0.2*(x['distance_2'])/x['distance_1'])))

return x

all_locations=all_locations.apply(lambda x:my_fun(x), axis=1)

all_locations=all_locations[['timestamp', 'Puck','location_X', 'location_Y']]

Applying Kalman Filtering

dff=all_locations[['location_X','location_Y']]

measurements =np.asarray(dff)

initial_state_mean = [measurements[0, 0],

```
                    0,
                    measurements[0, 1],
                    0]
transition_matrix = [[1, 1, 0, 0],
                    [0, 1, 0, 0],
                    [0, 0, 1, 1],
                    [0, 0, 0, 1]]
observation_matrix = [[1, 0, 0, 0],
                    [0, 0, 1, 0]]

kf1 = KalmanFilter(transition_matrices = transition_matrix,
                observation_matrices = observation_matrix,
                initial_state_mean = initial_state_mean)

kf1 = kf1.em(measurements, n_iter=5)
(smoothed_state_means, smoothed_state_covariances) = kf1.smooth(measurements)
kf2 = KalmanFilter(transition_matrices = transition_matrix,
                observation_matrices = observation_matrix,
                initial_state_mean = initial_state_mean,
                observation_covariance = 5 *kf1.observation_covariance,
                em_vars=['transition_covariance', 'initial_state_covariance'])
kf2 = kf2.em(measurements, n_iter=5)
```

```
(smoothed_state_means, smoothed_state_covariances)  = kf2.smooth(measurements)
plt.figure(1)
times = range(measurements.shape[0])
plt.plot(times, measurements[:, 0], 'bo',         times, measurements[:, 1], 'ro',
        times, smoothed_state_means[:, 0], 'b--',
        times, smoothed_state_means[:, 2], 'r--',)
plt.show()
Smooth=pd.DataFrame(smoothed_state_means[:]).rename(columns={0:'location_X',2:'location_Y'})[['location_X','location_Y']]
```

Appendix 2) Head Orientation Detection Module' Python Code

```
import numpy as np
import pandas as pd
import seaborn as sns
pd.set_option('max_colwidth', 20000000)
from sklearn import metrics
import math
import pygame

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import pickle

pygame 2.0.1 (SDL 2.0.14, Python 3.8.5)
Hello from the pygame community. https://www.pygame.org/contribute.html

df1=pd.read_excel(r"G:\Localization\211015\211015.xlsx")
df2=pd.read_excel(r'G:\Localization\211016\211016.xlsx')
```

```
df3=pd.read_excel(r'G:\Localization\211017\211017.xlsx')
df4=pd.read_excel(r'G:\Localization\211023\211023.xlsx')
df5=pd.read_excel(r'G:\Localization\211024\211024.xlsx')
df6=pd.read_excel(r'G:\Localization\211026\211026.xlsx')
df7=pd.read_excel(r'G:\Localization\211027\211027.xlsx')
df8=pd.read_excel(r'G:\Localization\211028\211028.xlsx')

df=pd.concat([df1,df2,df3,df4,df5,df6,df7,df8],axis=0)
df=df.sort_values(by=['timestamp'])
df=df.drop_duplicates(subset=['timestamp'])
```

The Puck ID should be chosen.

```
df=df[df['instanceId']=='0000a7c0']

alk=pd.DataFrame(df['timestamp'].apply(lambda x: x.replace('@','')))
df['timestamp']=pd.DataFrame(pd.to_datetime(alk['timestamp'], infer_datetime_format=True))
df['diff_seconds'] = pd.DataFrame(df['timestamp'].diff(1))['timestamp'].dt.total_seconds()

df=df[(df['diff_seconds']<=5)].sort_values('timestamp').reset_index().drop('index',axis=1)

a1=df['acceleration'].str.split(",", n = 2, expand = True).rename({0:'acc_x',1:'acc_y',2:'acc_z'},axis=1)[['acc_x','acc_y','acc_z']]
```

```
    acc=pd.concat([a1,df],axis=1)


    acc['acc_x']=acc['acc_x'].str.replace("[", '').str.replace("'", '').str.replace("'", '').apply(lambda x: pd.to_numeric(x, errors = 'coerce')).astype(float)
    acc['acc_y']=acc['acc_y'].str.replace("'", '').str.replace("'", '').apply(lambda x: pd.to_numeric(x, errors = 'coerce')).astype(float)
    acc['acc_z']=acc['acc_z'].str.replace("]", '').str.replace("'", '').apply(lambda x: pd.to_numeric(x, errors = 'coerce')).astype(float)


    acc = acc[['timestamp','acc_x','acc_y','acc_z']]


    """
    Three instances and their corresponding rssi values are defined.Also, a column for the Puck is added.
    """


    instance_1=df['nearest'].str.slice(16,24,1)
    instance_2=df['nearest'].str.slice(53,61,1)
    instance_3=df['nearest'].str.slice(90,98,1)
    rssi_1=df['nearest'].str.slice(33,36,1)
    rssi_2=df['nearest'].str.slice(70,73,1)
    rssi_3=df['nearest'].str.slice(107,110,1)
    Puck=df['instanceId']


    Processed_dataset=pd.DataFrame(df['timestamp'])
```

```
Processed_dataset['instance_1']=instance_1
Processed_dataset['instance_2']=instance_2
Processed_dataset['instance_3']=instance_3
Processed_dataset['rssi_1']=rssi_1
Processed_dataset['rssi_2']=rssi_2
Processed_dataset['rssi_3']=rssi_3
Processed_dataset['rssi_3']=rssi_3
Processed_dataset['Puck']=Puck


#Processed_dataset=Processed_dataset.set_index('timestamp')

Processed_dataset=Processed_dataset.reset_index(drop=True)

Processed_dataset=Processed_dataset[Processed_dataset!='']

Processed_dataset.dropna(inplace=True)

Processed_dataset=Processed_dataset.reset_index(drop=True)

"""
In order for the rssi columns to be ML model readable, we need to remove string records from them. Beacause rarely the dataset
```

coming from the Kibana has couple of string values.

The missing records should be removed.

Since by applying str.extract function, the negetaavie sign of rssi were removed, we should multiply them by -1.

"""

```
Processed_dataset['rssi_1']=pd.DataFrame(Processed_dataset['rssi_1'])['rssi_1'].str.extract(r'(\d+)', expand=False)
Processed_dataset['rssi_2']=pd.DataFrame(Processed_dataset['rssi_2'])['rssi_2'].str.extract(r'(\d+)', expand=False)
Processed_dataset['rssi_3']=pd.DataFrame(Processed_dataset['rssi_3'])['rssi_3'].str.extract(r'(\d+)', expand=False)

Processed_dataset.dropna(inplace=True)

Processed_dataset['rssi_1']=Processed_dataset['rssi_1'].astype(int)
Processed_dataset['rssi_2']=Processed_dataset['rssi_2'].astype(int)
Processed_dataset['rssi_3']=Processed_dataset['rssi_3'].astype(int)

Processed_dataset['rssi_1']=Processed_dataset['rssi_1']*-1
Processed_dataset['rssi_2']=Processed_dataset['rssi_2']*-1
Processed_dataset['rssi_3']=Processed_dataset['rssi_3']*-1

Processed_dataset["X1"]=Processed_dataset["instance_1"]
```

```
Processed_dataset["X2"]=Processed_dataset["instance_2"]
Processed_dataset["X3"]=Processed_dataset["instance_3"]

Processed_dataset.rename(columns={"instance_1":"Y1"},inplace=True)
Processed_dataset.rename(columns={"instance_2":"Y2"},inplace=True)
Processed_dataset.rename(columns={"instance_3":"Y3"},inplace=True)

x_axis={'0000004d':0,'00000058':4,'00000060':8,'0000004e':8,'00000061':4,'00000059':0,'000000a9':0,'000000ae':4,'000000af':8}
y_axis={'0000004d':0,'00000058':4,'00000060':0,'0000004e':4,'00000061':4,'00000059':4,'000000a9':8,'000000ae':8,'000000af':8}

Processed_dataset["X1"]=pd.DataFrame(Processed_dataset["Y1"].replace(x_axis))
Processed_dataset["X2"]=pd.DataFrame(Processed_dataset["Y2"].replace(x_axis))
Processed_dataset["X3"]=pd.DataFrame(Processed_dataset["Y3"].replace(x_axis))

Processed_dataset["Y1"]=pd.DataFrame(Processed_dataset["Y1"].replace(y_axis))
Processed_dataset["Y2"]=pd.DataFrame(Processed_dataset["Y2"].replace(y_axis))
Processed_dataset["Y3"]=pd.DataFrame(Processed_dataset["Y3"].replace(y_axis))

df=Processed_dataset

# Converting string values of coordiantes of the transmitters to float
df[['Y1', 'Y2', 'Y3','X1', 'X2', 'X3']]=df[['Y1', 'Y2', 'Y3','X1', 'X2', 'X3']].apply(lambda x: pd.to_numeric(x, errors = 'coerce')).dropna()
```

```
    # reorder by column name
    df2 = df.reindex(["timestamp","X1","Y1","X2", "Y2","X3","Y3","rssi_1","rssi_2","rssi_3","Puck"], axis=1)
    df2=df2.sort_values(by=['timestamp'])
```

Importing location dataset and applying Fuzzy matching

```
    # Correcting the timestamp format
    df1=pd.read_excel(r"G:\Localization\211015\all-location.xlsx")
    df12=pd.read_excel(r"G:\Localization\211016\all-location.xlsx")
    df13=pd.read_excel(r"G:\Localization\211017\all-location.xlsx")
    df14=pd.read_excel(r"G:\Localization\211023\all-location.xlsx")
    df15=pd.read_excel(r"G:\Localization\211024\all-location.xlsx")
    df16=pd.read_excel(r"G:\Localization\211026\all-location.xlsx")
    df17=pd.read_excel(r"G:\Localization\211027\all-location.xlsx")
    df18=pd.read_excel(r"G:\Localization\211028\all-location.xlsx")

    df1=pd.concat([df1,df12,df13,df14,df15,df16,df17,df18],axis=0)
    df1=df1.sort_values(by=['timestamp'])
    df1=df1.drop_duplicates(subset=['timestamp'])

    # Correcting the timestamp format
```

```
    alk=pd.DataFrame(df1['timestamp'].apply(lambda x: x.replace('@','')))
    df1['timestamp']=pd.DataFrame(pd.to_datetime(alk['timestamp'], infer_datetime_format=True))

    df1['diff_seconds'] = pd.DataFrame(df1['timestamp'].diff(1))['timestamp'].dt.total_seconds()
    df1=df1[(df1['diff_seconds']<=5)].sort_values('timestamp').reset_index().drop('index',axis=1)

    #Fuzzy Matching
    fuzzy_matched = pd.merge_asof(df1, df2, left_on='timestamp',right_on='timestamp', direction='nearest')
```

Calculating the angle between the TARGET and First Transmitter

```
    def angle_of_vector(x, y):
        return pygame.math.Vector2(x, y).angle_to((1, 0))

    def angle_of_line(x):
        return angle_of_vector(x['X1']-x['location_X'], x['Y1']-x['location_Y'])
    fuzzy_matched['angle_1'] = fuzzy_matched.apply(lambda x:angle_of_line(x), axis=1)
```

Correcting the angles

```
    def angle_correction(x):
        if x['angle_1'] == 0:
```

```
        return x
    elif 0>x['angle_1']>=-180:
        return -1 * x['angle_1']
    elif 0<x['angle_1']<180:
        return 360 - x['angle_1']
fuzzy_matched['angle_1'] = fuzzy_matched.apply(lambda x:angle_correction(x), axis=1)
```

Calculating the angle between the TARGET and Second Transmitter

```
def angle_of_vector(x, y):
    return pygame.math.Vector2(x, y).angle_to((1, 0))

def angle_of_line(x):
    return angle_of_vector(x['X2']-x['location_X'], x['Y2']-x['location_Y'])
fuzzy_matched['angle_2'] = fuzzy_matched.apply(lambda x:angle_of_line(x), axis=1)
```

Correcting the angles

```
def angle_correction(x):
    if x['angle_2'] == 0:
        return x
    elif 0>x['angle_2']>=-180:
```

```
        return -1 * x['angle_2']
    elif 0<x['angle_2']<180:
        return 360 - x['angle_2']
fuzzy_matched['angle_2'] = fuzzy_matched.apply(lambda x:angle_correction(x), axis=1)
```

Calculating the angle between the TARGET and Third Transmitter

```
def angle_of_vector(x, y):
    return pygame.math.Vector2(x, y).angle_to((1, 0))

def angle_of_line(x):
    return angle_of_vector(x['X3']-x['location_X'], x['Y3']-x['location_Y'])
fuzzy_matched['angle_3'] = fuzzy_matched.apply(lambda x:angle_of_line(x), axis=1)
```

Correcting the angles

```
def angle_correction(x):
    if x['angle_3'] == 0:
        return x
    elif 0>x['angle_3']>=-180:
        return -1 * x['angle_3']
    elif 0<x['angle_3']<180:
```

```
        return 360 - x['angle_3']
    fuzzy_matched['angle_3'] = fuzzy_matched.apply(lambda x:angle_correction(x), axis=1)
```

Performing feature engineering process

```
    def boundries(x):

        Xmin=min(x['X1'],x['X2'],x['X3'])
        Xmax=max(x['X1'],x['X2'],x['X3'])
        Ymin=min(x['Y1'],x['Y2'],x['Y3'])
        Ymax=max(x['Y1'],x['Y2'],x['Y3'])

        if (Xmin<=x['location_X']<=Xmax)&(Ymin<=x['location_Y']<=Ymax):
            x['check']=1

        else:

           x['check']=0

        return x

    fuzzy_matched = fuzzy_matched.apply(lambda x:boundries(x), axis=1)
```

Pre-processing the Dataset

```
final = fuzzy_matched.drop(['diff_seconds','Puck'],axis=1)

def boundries(x):

    x['dis1'] = np.sqrt(((x['location_X']-x['X1'])**2) + ((x['location_Y']-x['Y1'])**2))
    x['dis2'] = np.sqrt(((x['location_X']-x['X2'])**2) + ((x['location_Y']-x['Y2'])**2))
    x['dis3'] = np.sqrt(((x['location_X']-x['X3'])**2) + ((x['location_Y']-x['Y3'])**2))
    return x



final = final.apply(lambda x:boundries(x), axis=1)

def boundries(x):

    x['dis1_1'] = np.sqrt(((x['X2']-x['X1'])**2) + ((x['Y2']-x['Y1'])**2))
    x['dis2_1'] = np.sqrt(((x['X3']-x['X2'])**2) + ((x['Y3']-x['Y2'])**2))
    x['dis3_1'] = np.sqrt(((x['X1']-x['X3'])**2) + ((x['Y1']-x['Y3'])**2))
    return x
```

```
final = final.apply(lambda x:boundries(x), axis=1)

# Putting labels

# Creating Orientation attribute
final['or']=np.nan

# # # #Experiment1
final.loc[('2021-10-15 08:55:45'<=final['timestamp'])&(final['timestamp']<='2021-10-15 09:11:23'),'or']='right'
final.loc[('2021-10-15 08:37:20'<=final['timestamp'])&(final['timestamp']<='2021-10-15 08:55:05'),'or']='down_right'
final.loc[('2021-10-14 17:40:00'<=final['timestamp'])&(final['timestamp']<='2021-10-14 17:54:54'),'or']='down'
final.loc[('2021-10-14 17:14:25'<=final['timestamp'])&(final['timestamp']<='2021-10-14 17:31:22'),'or']='down_left'
final.loc[('2021-10-14 16:42:04'<=final['timestamp'])&(final['timestamp']<='2021-10-14 17:00:54'),'or']='left'
final.loc[('2021-10-14 16:13:20'<=final['timestamp'])&(final['timestamp']<='2021-10-14 16:31:28'),'or']='up_left'
final.loc[('2021-10-14 15:56:25'<=final['timestamp'])&(final['timestamp']<='2021-10-14 16:11:38'),'or']='up'
final.loc[('2021-10-15 18:16:30'<=final['timestamp'])&(final['timestamp']<='2021-10-15 18:33:40'),'or']='up_right'


# # # #Experiment2
```

```
# final.loc[('2021-10-16 16:30:30'<=final['timestamp'])&(final['timestamp']<='2021-10-16 16:54:00'),'or']='right'
# final.loc[('2021-10-16 16:54:50'<=final['timestamp'])&(final['timestamp']<='2021-10-16 17:09:19'),'or']='down_right'
# final.loc[('2021-10-16 15:41:20'<=final['timestamp'])&(final['timestamp']<='2021-10-16 16:02:40'),'or']='down'
# final.loc[('2021-10-16 17:10:15'<=final['timestamp'])&(final['timestamp']<='2021-10-16 17:26:20'),'or']='down_left'
# final.loc[('2021-10-16 17:27:20'<=final['timestamp'])&(final['timestamp']<='2021-10-16 17:32:22'),'or']='down_left'
# final.loc[('2021-10-16 15:26:50'<=final['timestamp'])&(final['timestamp']<='2021-10-16 15:40:32'),'or']='left'
# final.loc[('2021-10-16 18:53:20'<=final['timestamp'])&(final['timestamp']<='2021-10-16 19:08:55'),'or']='up_left'
# final.loc[('2021-10-16 18:36:20'<=final['timestamp'])&(final['timestamp']<='2021-10-16 18:52:37'),'or']='up'
# final.loc[('2021-10-16 18:17:59'<=final['timestamp'])&(final['timestamp']<='2021-10-16 18:35:23'),'or']='up_right'



# # # #Experiment3
# final.loc[('2021-10-17 11:39:59'<=final['timestamp'])&(final['timestamp']<='2021-10-17 11:57:00'),'or']='right'
# final.loc[('2021-10-17 15:34:30'<=final['timestamp'])&(final['timestamp']<='2021-10-17 15:51:40'),'or']='down_right'
# final.loc[('2021-10-17 15:11:15'<=final['timestamp'])&(final['timestamp']<='2021-10-17 15:31:45'),'or']='down'
# final.loc[('2021-10-17 14:11:04'<=final['timestamp'])&(final['timestamp']<='2021-10-17 14:26:00'),'or']='down_left'
# final.loc[('2021-10-17 14:26:40'<=final['timestamp'])&(final['timestamp']<='2021-10-17 14:31:55'),'or']='down_left'
# final.loc[('2021-10-17 13:53:25'<=final['timestamp'])&(final['timestamp']<='2021-10-17 14:09:18'),'or']='left'
# final.loc[('2021-10-17 13:36:40'<=final['timestamp'])&(final['timestamp']<='2021-10-17 13:51:15'),'or']='up_left'
# final.loc[('2021-10-17 13:18:50'<=final['timestamp'])&(final['timestamp']<='2021-10-17 13:34:55'),'or']='up'
# final.loc[('2021-10-17 11:59:04'<=final['timestamp'])&(final['timestamp']<='2021-10-17 12:13:33'),'or']='up_right'
```

```
#Experiment4
final.loc[('2021-10-23 09:40:20'<=final['timestamp'])&(final['timestamp']<='2021-10-23 09:56:25'),'or']='right'
final.loc[('2021-10-23 12:37:59'<=final['timestamp'])&(final['timestamp']<='2021-10-23 12:54:15'),'or']='down_right'
final.loc[('2021-10-23 12:17:35'<=final['timestamp'])&(final['timestamp']<='2021-10-23 12:36:40'),'or']='down'
final.loc[('2021-10-23 11:52:15'<=final['timestamp'])&(final['timestamp']<='2021-10-23 12:07:05'),'or']='down_left'
final.loc[('2021-10-23 11:32:55'<=final['timestamp'])&(final['timestamp']<='2021-10-23 11:51:25'),'or']='left'
final.loc[('2021-10-23 10:49:35'<=final['timestamp'])&(final['timestamp']<='2021-10-23 11:06:25'),'or']='up_left'
final.loc[('2021-10-23 10:19:20'<=final['timestamp'])&(final['timestamp']<='2021-10-23 10:35:45'),'or']='up'
final.loc[('2021-10-23 09:57:25'<=final['timestamp'])&(final['timestamp']<='2021-10-23 10:14:45'),'or']='up_right'

# #Experiment5
#final.loc[('2021-10-24 19:23:40'<=final['timestamp'])&(final['timestamp']<='2021-10-24 19:32:55'),'or']='right'
# final.loc[('2021-10-24 20:53:45'<=final['timestamp'])&(final['timestamp']<='2021-10-24 21:03:35'),'or']='down_right'
# final.loc[('2021-10-24 20:42:50'<=final['timestamp'])&(final['timestamp']<='2021-10-24 20:52:40'),'or']='down'
# final.loc[('2021-10-24 20:29:04'<=final['timestamp'])&(final['timestamp']<='2021-10-24 20:39:05'),'or']='down_left'
# final.loc[('2021-10-24 20:16:35'<=final['timestamp'])&(final['timestamp']<='2021-10-24 20:26:00'),'or']='left'
# final.loc[('2021-10-24 20:05:30'<=final['timestamp'])&(final['timestamp']<='2021-10-24 20:15:15'),'or']='up_left'
# final.loc[('2021-10-24 19:53:45'<=final['timestamp'])&(final['timestamp']<='2021-10-24 20:04:10'),'or']='up'
# final.loc[('2021-10-24 19:34:55'<=final['timestamp'])&(final['timestamp']<='2021-10-24 19:44:35'),'or']='up_right'

#Experiment6
final.loc[('2021-10-26 08:33:04'<=final['timestamp'])&(final['timestamp']<='2021-10-26 08:49:05'),'or']='right'
```

```
final.loc[('2021-10-26 10:50:45'<=final['timestamp'])&(final['timestamp']<='2021-10-26 11:07:05'),'or']='down_right'
final.loc[('2021-10-26 10:33:25'<=final['timestamp'])&(final['timestamp']<='2021-10-26 10:48:45'),'or']='down'
final.loc[('2021-10-26 10:16:05'<=final['timestamp'])&(final['timestamp']<='2021-10-26 10:31:55'),'or']='down_left'
final.loc[('2021-10-26 09:59:30'<=final['timestamp'])&(final['timestamp']<='2021-10-26 10:15:10'),'or']='left'
final.loc[('2021-10-26 09:30:30'<=final['timestamp'])&(final['timestamp']<='2021-10-26 09:47:10'),'or']='up_left'
final.loc[('2021-10-26 09:08:03'<=final['timestamp'])&(final['timestamp']<='2021-10-26 09:23:25'),'or']='up'
final.loc[('2021-10-26 08:50:20'<=final['timestamp'])&(final['timestamp']<='2021-10-26 09:07:05'),'or']='up_right'

final.loc[('2021-10-27 18:45:45'<=final['timestamp'])&(final['timestamp']<='2021-10-27 18:56:00'),'or']='right'
final.loc[('2021-10-27 20:23:10'<=final['timestamp'])&(final['timestamp']<='2021-10-27 20:33:10'),'or']='down_right'
final.loc[('2021-10-27 20:10:45'<=final['timestamp'])&(final['timestamp']<='2021-10-27 20:22:25'),'or']='down'
final.loc[('2021-10-27 19:59:20'<=final['timestamp'])&(final['timestamp']<='2021-10-27 20:09:33'),'or']='down_left'
final.loc[('2021-10-27 19:48:25'<=final['timestamp'])&(final['timestamp']<='2021-10-27 19:58:33'),'or']='left'
final.loc[('2021-10-27 19:20:45'<=final['timestamp'])&(final['timestamp']<='2021-10-27 19:31:25'),'or']='up_left'
final.loc[('2021-10-27 19:09:35'<=final['timestamp'])&(final['timestamp']<='2021-10-27 19:19:35'),'or']='up'
final.loc[('2021-10-27 18:56:50'<=final['timestamp'])&(final['timestamp']<='2021-10-27 19:06:55'),'or']='up_right'

final.loc[('2021-10-28 10:22:50'<=final['timestamp'])&(final['timestamp']<='2021-10-28 10:28:35'),'or']='right'
final.loc[('2021-10-28 11:11:05'<=final['timestamp'])&(final['timestamp']<='2021-10-28 11:17:05'),'or']='down_right'
final.loc[('2021-10-28 11:04:05'<=final['timestamp'])&(final['timestamp']<='2021-10-28 11:10:05'),'or']='down'
final.loc[('2021-10-28 10:56:25'<=final['timestamp'])&(final['timestamp']<='2021-10-28 11:03:15'),'or']='down_left'
final.loc[('2021-10-28 10:49:50'<=final['timestamp'])&(final['timestamp']<='2021-10-28 10:55:05'),'or']='left'
```

```
final.loc[('2021-10-28 10:43:05'<=final['timestamp'])&(final['timestamp']<='2021-10-28 10:48:55'),'or']='up_left'
final.loc[('2021-10-28 10:36:15'<=final['timestamp'])&(final['timestamp']<='2021-10-28 10:42:15'),'or']='up'
final.loc[('2021-10-28 10:29:30'<=final['timestamp'])&(final['timestamp']<='2021-10-28 10:35:25'),'or']='up_right'

final = final.dropna()
from sklearn.utils import shuffle
final = shuffle(final)

left = final[final['or']=='left'].head(1917).copy()
right = final[final['or']=='right'].head(1917).copy()
up_left = final[final['or']=='up_left'].head(1917).copy()
up_right = final[final['or']=='up_right'].head(1917).copy()
down_left = final[final['or']=='down_left'].head(1917).copy()
down = final[final['or']=='down'].head(1917).copy()
up = final[final['or']=='up'].head(1917).copy()
down_right = final[final['or']=='down_right'].head(1917).copy()

final_dataset = pd.DataFrame()
final_dataset = final_dataset.append([left, right, up_left,up_right,down_left,down,up,down_right])



#final_dataset = final_dataset[['rssi_1', 'rssi_2', 'rssi_3', 'angle_1', 'angle_2','angle_3','or']]
```

```
final_dataset = final_dataset.rename({'or':'label'},axis=1)

final_dataset=final_dataset.dropna().sort_values('timestamp')

#final_dataset = pd.merge(final_dataset, acc, on=['timestamp'])
final_dataset = pd.merge_asof(final_dataset, acc, left_on='timestamp',right_on='timestamp', direction='nearest')

#### Encoding the lables

from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()
final_dataset['label'] = label.fit_transform(final_dataset['label'])
final_dataset

label.classes_

array(['down', 'down_left', 'down_right', 'left', 'right', 'up',
       'up_left', 'up_right'], dtype=object)
```

Normalizing the training set

```
from sklearn.preprocessing import MinMaxScaler
### Standardized data

X = final_dataset.drop(['timestamp','label','acc_x','acc_y','acc_z','location_X', 'location_Y', 'X1', 'Y1', 'X2', 'Y2', 'X3', 'Y3',],axis=1)
y = final_dataset['label']

scaler = MinMaxScaler()
X[['angle_1','angle_2','angle_3']] = scaler.fit_transform(X[['angle_1','angle_2','angle_3']])

scaler = StandardScaler()
X[['rssi_1', 'rssi_2', 'rssi_3','dis1','dis2','dis3','dis1_1','dis2_1','dis3_1']] = scaler.fit_transform(X[['rssi_1', 'rssi_2', 'rssi_3','dis1','dis2','dis3','dis1_1','dis2_1','dis3_1']])

# scaled_X = pd.DataFrame(data = X, columns = ['rssi_1', 'rssi_2', 'angle_1', 'angle_2'])
# scaled_X['label'] = y.values

# scaled_X

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

Trianing the model

Creating the Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout
from tensorflow.keras.constraints import max_norm
from tensorflow.keras.optimizers import Adam

model = Sequential()

# input layer
model.add(Dense(13,  activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(32,  activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(64,  activation='relu'))
model.add(Dropout(0.1))
```

```
# output layer
model.add(Dense(8, activation='softmax'))

# Compile model
model.compile(optimizer=Adam(learning_rate = 0.001),
              loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

history = model.fit(X_train, y_train, batch_size=16 ,epochs = 200,validation_data= (X_test, y_test), verbose=1)

def plot_learningCurve(history, epochs):
  # Plot training & validation accuracy values
  epoch_range = range(1, epochs+1)
  plt.plot(epoch_range, history.history['accuracy'])
  plt.plot(epoch_range, history.history['val_accuracy'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

  # Plot training & validation loss values
  plt.plot(epoch_range, history.history['loss'])
```

```
    plt.plot(epoch_range, history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()
plot_learningCurve(history,200)

from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
plot_learningCurve(history,200)
y_pred=model.predict(X_test)
y_pred=np.argmax(y_pred,axis=1)
mat = confusion_matrix(y_test, y_pred)

mat = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(conf_mat=mat, class_names=label.classes_, show_normed=True, figsize=(7,7))
```

Adding confidence level to the predicted classes

```
pred_conf = pd.DataFrame((model.predict(X_test) > 0.5).astype("int32"))
```

```
dictionary = {False: "low", True: "high"}
pred_conf['pred_conf'] = pd.DataFrame(pred_conf.sum(axis=1) != 0).replace({0: dictionary})

pred_conf = pd.DataFrame(pred_conf)

dic_label = {0:'down',1:'down_left',2:'down_right',3:'left',4:'right',5:'up',6:'up_left',7:'up_right'}

final_predictions = pd.DataFrame(y_pred).replace(dic_label)
```

Appendix 3) Productivity State Detection Module' Python Code

```
import tensorflow as tf
    from tensorflow.keras import Sequential
    from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization
    from tensorflow.keras.layers import Conv2D, MaxPool2D
    from tensorflow.keras.optimizers import Adam
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler, LabelEncoder
    from sklearn.utils import shuffle

Load the Dataset
    data=pd.read_csv(r"G:\Localization\211027\labelled_dataset.csv")
    data=data[['timestamp', 'acc_x_h', 'acc_y_h', 'acc_z_h', 'acc_x_w', 'acc_y_w','acc_z_w', 'acc_x_c', 'acc_y_c', 'acc_z_c','activity']]
    data=data.dropna()
Balance this data

    data['acc_x_h'] = data['acc_x_h'].astype('float')
    data['acc_y_h'] = data['acc_y_h'].astype('float')
```

```
data['acc_z_h'] = data['acc_z_h'].astype('float')
data['acc_x_w'] = data['acc_x_w'].astype('float')
data['acc_y_w'] = data['acc_y_w'].astype('float')
data['acc_z_w'] = data['acc_z_w'].astype('float')
data['acc_x_c'] = data['acc_x_c'].astype('float')
data['acc_y_c'] = data['acc_y_c'].astype('float')
data['acc_z_c'] = data['acc_z_c'].astype('float')

Fs = 1
activities = data['activity'].value_counts().index

def plot_activity(activity, data):
    fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(13, 5), sharex=True)
    plot_axis(ax0, data['timestamp'], data['acc_x_h'], 'X-Axis')
    plot_axis(ax1, data['timestamp'], data['acc_y_h'], 'Y-Axis')
    plot_axis(ax2, data['timestamp'], data['acc_z_h'], 'Z-Axis')
    plt.subplots_adjust(hspace=0.2)
    fig.suptitle(activity)
    plt.subplots_adjust(top=0.90)
    plt.show()

def plot_axis(ax, x, y, title):
```

```
    ax.plot(x, y, 'g')
    ax.set_title(title)
    ax.xaxis.set_visible(False)
    ax.set_ylim([min(y) - np.std(y), max(y) + np.std(y)])
    ax.set_xlim([min(x), max(x)])
    ax.grid(True)

for activity in activities:
    data_for_plot = data[(data['activity'] == activity)][:Fs*100]
    plot_activity(activity, data_for_plot)
def plot_activity(activity, data):
    fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(13, 5), sharex=True)
    plot_axis(ax0, data['timestamp'], data['acc_x_c'], 'X-Axis')
    plot_axis(ax1, data['timestamp'], data['acc_y_c'], 'Y-Axis')
    plot_axis(ax2, data['timestamp'], data['acc_z_c'], 'Z-Axis')
    plt.subplots_adjust(hspace=0.2)
    fig.suptitle(activity)
    plt.subplots_adjust(top=0.90)
    plt.show()

def plot_axis(ax, x, y, title):
    ax.plot(x, y, 'r')
```

```
    ax.set_title(title)
    ax.xaxis.set_visible(False)
    ax.set_ylim([min(y) - np.std(y), max(y) + np.std(y)])
    ax.set_xlim([min(x), max(x)])
    ax.grid(True)

for activity in activities:
    data_for_plot = data[(data['activity'] == activity)][:Fs*100]
    plot_activity(activity, data_for_plot)
df = data.drop(['timestamp'], axis = 1).copy()
df.head()

#jogging = df[df['activity']=='jogging'].head(560).copy()
#walking = df[df['activity']=='walking'].head(560).copy()
#idling = df[df['activity']=='idling'].head(560).copy()

idling = df[df['activity']=='idling'].tail(1342).copy()
walking = df[df['activity']=='walking'].tail(1342).copy()



brick_laying = df[df['activity']=='brick_laying'].tail(1127).copy()
plastering = df[df['activity']=='plastering'].head(1127).copy()
```

```
mortar = df[df['activity']=='mortar'].tail(1127).copy()
shoveling = df[df['activity']=='shoveling'].tail(1127).copy()
painting = df[df['activity']=='painting'].tail(2255).copy()

others1 = pd.DataFrame()
others1 = others1.append([shoveling,mortar,brick_laying,painting,plastering])
others1 = shuffle(others1).tail(1342)

others = pd.DataFrame()
others = others.append([others1])

others['activity'] = 'others'

others = others[others['activity']=='others'].tail(1127).copy()



value_add_work = pd.DataFrame()
value_add_work = value_add_work.append([painting])
value_add_work = value_add_work.tail(2255)
value_add_work['activity'] = 'value_add_work'
```

```
non_value_add_work = pd.DataFrame()
non_value_add_work = non_value_add_work.append([shoveling,mortar,brick_laying,others])
non_value_add_work = non_value_add_work.tail(4510)
non_value_add_work['activity'] = 'non_value_add_work'
non_value_add_work = non_value_add_work[non_value_add_work['activity']=='non_value_add_work'].tail(4510).copy()

balanced_data = pd.DataFrame()
balanced_data = balanced_data.append([idling,walking,value_add_work,non_value_add_work])
balanced_data.shape

from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()
balanced_data['label'] = label.fit_transform(balanced_data['activity'])
[9447 rows x 11 columns]
```

Standardized data

```
balanced_data['acc_x_h'] = balanced_data['acc_x_h'].diff(1)
balanced_data['acc_y_h'] = balanced_data['acc_y_h'].diff(1)
balanced_data['acc_z_h'] = balanced_data['acc_z_h'].diff(1)
balanced_data['acc_x_w'] = balanced_data['acc_x_w'].diff(1)
```

```
    balanced_data['acc_y_w'] = balanced_data['acc_y_w'].diff(1)
    balanced_data['acc_z_w'] = balanced_data['acc_z_w'].diff(1)
    balanced_data['acc_x_c'] = balanced_data['acc_x_c'].diff(1)
    balanced_data['acc_y_c'] = balanced_data['acc_y_c'].diff(1)
    balanced_data['acc_z_c'] = balanced_data['acc_z_c'].diff(1)

    balanced_data = balanced_data.fillna(0)

    X = balanced_data.drop(['activity','label'],axis=1)
    y = balanced_data['label']

    scaled_X = pd.DataFrame(data = X, columns = ['acc_x_h', 'acc_y_h', 'acc_z_h', 'acc_x_w', 'acc_y_w','acc_z_w', 'acc_x_c', 'acc_y_c',
'acc_z_c'])
    scaled_X['label'] = y.values
```

Frame Preparation

```
    import scipy.stats as stats

    Fs = 1
    frame_size = Fs*4
    hop_size = Fs*1
```

```
def get_frames(df, frame_size, hop_size):

    N_FEATURES = 9

    frames = []
    labels = []
    for i in range(0, len(df) - frame_size, hop_size):
        acc_x_h = df['acc_x_h'].values[i: i + frame_size]
        acc_y_h = df['acc_y_h'].values[i: i + frame_size]
        acc_z_h = df['acc_z_h'].values[i: i + frame_size]
        acc_x_w = df['acc_x_w'].values[i: i + frame_size]
        acc_y_w = df['acc_y_w'].values[i: i + frame_size]
        acc_z_w = df['acc_z_w'].values[i: i + frame_size]
        acc_x_c = df['acc_x_c'].values[i: i + frame_size]
        acc_y_c = df['acc_y_c'].values[i: i + frame_size]
        acc_z_c = df['acc_z_c'].values[i: i + frame_size]



        # Retrieve the most often used label in this segment
        label = stats.mode(df['label'][i: i + frame_size])[0][0]
        frames.append([acc_x_h, acc_y_h, acc_z_h,acc_x_w, acc_y_w, acc_z_w,acc_x_c,acc_y_c,acc_z_c])
```

```
        labels.append(label)

    # Bring the segments into a better shape
    frames = np.asarray(frames)
    labels = np.asarray(labels)

    return frames, labels

X, y = get_frames(scaled_X, frame_size, hop_size)

X.shape, y.shape

((9443, 9, 4), (9443,))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0, stratify = y)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).reshape(X_train.shape)
X_test = scaler.transform(X_test.reshape(-1, X_test.shape[-1])).reshape(X_test.shape)

import pickle
scalerfile = 'scaler_Dec_painter.sav'
```

```
pickle.dump(scaler, open(scalerfile, 'wb'))
```

```
X_train.shape, X_test.shape
```

```
((8498, 9, 4), (945, 9, 4))
```

```
X_train[0].shape, X_test[0].shape
```

```
((9, 4), (9, 4))
```

```
X_train = X_train.reshape(8498, 9, 4,1)
X_test = X_test.reshape(945, 9, 4,1)
```

```
X_train[0].shape, X_test[0].shape
```

```
((9, 4, 1), (9, 4, 1))
```

```
label.classes_
```

```
array(['idling', 'non_value_add_work', 'value_add_work', 'walking'],
      dtype=object)
```

```
X_train.min()
```

2D CNN Model

```
model = Sequential()
model.add(Conv2D(32, (2, 2), activation = 'relu', input_shape = X_train[0].shape))
model.add(Dropout(0.2))

model.add(Conv2D(32, (2, 2), activation='relu'))
model.add(Dropout(0.2))

model.add(Conv2D(32, (2, 2), activation='relu'))
model.add(Dropout(0.2))


model.add(Flatten())

model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))


model.add(Dense(4, activation='softmax'))
```

```
model.compile(optimizer=Adam(learning_rate = 0.001),
              loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

history = model.fit(X_train, y_train,epochs = 50, validation_data= (X_test, y_test), verbose=1)
```