IMPROVING FIRE SAFETY SYSTEMS BASED ON INTERNET OF THINGS AND DEEP LEARNING

By
Mohamed Gamaleldin
Bachelor of Communication and Electronics Engineering, Mansoura University, Egypt, 2013

A thesis
presented to Ryerson University
in partial fulfillment of
the requirements for the degree of
Master of Applied Science
in the program of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2020

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Abstract

Mohamed Gamaleldin
Master of Applied Science
Electrical and Computer Engineering
Ryerson University, Toronto, Canada, 2020

Structure fires are one of the main concerns for fire safety systems. The actual fire safety of a building depends on not only how it is designed and constructed, but also on how it is operated. Computational fluid dynamics software is the current solution to reduce the casualties in the fire circumstances. However, it consumes hours to provide the results in some cases that makes it hard to run in real-time. It also does not accept any changes after starting the simulation, which makes it unsuitable for running in the dynamic nature of the fire. On the other hand, the current evacuation signs are fixed, which might guide occupants and firefighter to dangerous zones.

In this research, we present a smoke emulator that runs in real-time to reflect what is happening on the ground-truth. This system is achieved using a light-weight smoke emulator engine, deep learning, and internet of things. The IoT sensors are sending the measurements to correct the emulator from any deviation and reflect events such as fire starting, people movement, and the door's status. This emulator helps the firefighter by providing them with a map that shows the smoke development in the building. They can take a snapshot from the current status of the building and try different virtual evacuation and firefighting plans to pick the best and safest for them to proceed. The system will also control the exit signs to have adaptive exit routes that guide occupants away from fire and smoke to minimize the exposure time to the toxic gases.

Acknowledgments

Firstly, I would like to thank Prof. Lian Zhao and Prof. Zaiyi Liao for their support, guidance, and encouragement for my research. I am incredibly grateful for their comments and feedback that help me to develop, correct, and provide this research. Secondly, I would like to thank my research partner Mohamed Asfour for his dedication and assistance in my research. This thesis would not otherwise be completed without Prof. Zhao, Prof. Liao, and Asfour. I also would like to thank all the graduate students I have met who provided me with the support and this remarkable learning experience in my studies. Finally, I would like to thank my family for all the love and support they have provided throughout the years.

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Introduction

Structure fires are one of the main concerns for building management systems (BMS). According to the Canadian government reports, there are 19,000 structural fires yearly that caused in 2014, for instance, 109 casualties and 170 non-evacuation situations[1][2][3]. It is reported in 2015, that nearly 3000 firefighters had injuries caused by smoke and fire [4]. Another statistic shows that smoke inhalation is the leading cause of deaths in cases of building fires leading at 34% of the fatalities, even more than burns, and contributing to other causes as well[5][6].

Accordingly, many studies have been conducted to investigate how to reduce casualties by using smoke simulators such as Fire Dynamics Simulator (FDS)[7][8]. It solves the Computational Fluid Dynamics (CFD) model that is depicted by Navier-Stokes equations, which relying on numerical solving methods[9]. The model requires predetermined environment boundaries with a knowledge of all of its events to provide a very accurate prediction of the smoke propagation in the building. The smoke propagation can be affected by different elements such as layout, ventilation, and building materials [10] [11][12]. The building life cycle includes four phases, the design phase, the construction, the operation and the demolishing. The building fire safety is included in the design phase and the operation phase of the building life cycle. Like many other existing fire and smoke simulator, FDS can be used in the design phase to evaluate the building design, which helps to optimize the fire safety of buildings. FDS assists in developing effective methods to control the smoke path using different active and passive techniques. The active techniques such as pressurization that depends on changing the pressure in the pressure control zones to have a clear evacuation path and prevent smoke from spreading inside the building. The passive techniques are based on different structural components; for instance, smoke barriers that reduce smoke propagation through stairwells [13].

However, running an accurate CFD model can be computationally demanding where it takes hours in simulation to provide minutes of simulated data. It can not adapt to sudden changes in the environment after the simulation is commenced, which is against the dynamic nature of the fire scene. Therefore, it can not be used to assist firefighting process during the operation phase of buildings. In a normal firefighting process, firefighters are deployed into a building with duties such as searching for survivors, extinguishing fire, coordinating evacuation, and emergent operation of building equipment. While moving through the buildings, they often are exposed to dangerous situations, potentially resulting in major injuries even death. Among the many factors contributing to this dangerous situations, inability to accurately predict the development of fire and smoke is a key problem. In most cases, firefighters need to decide quickly between actions with varying risks, while facing very dynamic fire and smoke conditions. If an accurate prediction of how fire and smoke will develop can be provided, the firefighting process can be much safer and more efficient. Another important aspect of survival in construction fires is egress schemes.

The shortest evacuation path problem has been an ongoing research topic, specifically in evacuation scenarios [14], and it mainly depends on the representation of the search environment. Search algorithms can be categorized into uninformed [15] and informed algorithms [16]. The latter ones know the target position and are able to leverage that knowledge to find complete optimal paths in fewer iterations. This coincides with the nature of the evacuation task since the exits positions are known.

Some evacuation strategies including path planning algorithms [17], such as Dijkstra [18] [19] and A* [20], have already been implemented in BMS. These are used to search for the safest simultaneous evacuation routes in firefighting process during the operation phase of buildings. The performance and optimality of these algorithms depend on the instantaneously collected data from the deployed sensors [21]. They can measure smoke concentration and temperature [14] [22]; then these data are sent to a gateway that coordinates between the nodes and the cloud. The cloud handles data processing and storing, and provides the user with the optimal exit route in the case of fire. The smoke sensors are the only way for these systems to depict the smoke propagation on ground-truth, but lacking the ability to illustrate the propagation of smoke.

The new revolution of IoT has enormous impacts on many technologies and fields [23], such as environment [24], agriculture, and building management system (BMS). It helps to develop seamlessly integrated wireless sensor networks (WSN) with the cloud, which eases the data acquisition [25] [26], covering different scales from small houses to urban areas. The collected data helps to build powerful analysis models that give efficient and reliable solutions for wide range of complex problems such as space monitoring [27] [28] and navigation [29] [30]. BMS has beneficial improvement in HVAC by reducing power consumption [31] and its integration with firefighting [32], which can have a direct application on evacuation systems in burning buildings [33]. The statistics show that the fatalities caused by fire are reduced by almost 50% when the smoke alarms are used [34], which indicates that the usage of the new technologies such as IoT to build real-time smart systems helps to reduce fatalities.

1.2 Motivation

A fire emulator is to be developed to overcome these problems. The emulator is initially configured according to the physical condition of the building and is continuously updated using the data sent from a sensor network. It has low latency while predicting realistic smoke behavior taking into account the changing environment's events such as doors state. The result is compiled to produce instructions for firefighters in action so that they can properly assess the risks associated with all possible actions and be more efficient. It becomes possible to quickly find the optimal exit paths because both the current sensor measurements and the smoke spreading in the near future predicted by the emulator are available.

1.3 Literature Review

To employ simulation tools such as FDS in the design phase, Building Information Model (BIM) is used to allow for seamless data sharing between architects and fire engineers [35] in an Integrated Design Process (IDP) [36]. Based on the architectural design, smoke control strategies, egress themes, structural performance, and fire protection strategies can be optimized using fire simulation. It is promising that the advancement of relevant technologies and availability of engineering tools have largely enhanced the application of fire simulation technology in the design of buildings. However, this process does not take into account the dynamic nature of the fire scene, which can dramatically change the spread of smoke throughout the building and affect the egress routes. As an example, static exit signs might guide people to hazardous areas or could be very confusing especially when they point to where smoke is dominant and the egress routes do not seem tenable.

To obtain relevant data from a construction fire that can be used by a fire emulator, a sensor network needs to be deployed such that the construction is adequately covered. Timely and reliable detection of fire is critical and depends on the ability to recognize the changes of the environmental variables likely caused by fire, for instance, abnormal temperature increase, visibility obstruction due to flames [37] and smoke [38], radiation, and combination of chemical compounds resulting from combustion.

Most of the proposed solutions in the literature is relying on three major steps, monitor the environment, compare with a reference then report the situation to the user. The solutions use different softwares that are not fully integrated or orchestrated to provide reflect the fire situation on the ground-truth.

Choi *et al.* [39] use prediction data from the FDS model's results and informed search algorithm A* [40] to obtain the safest evacuation routes. The searching algorithm aims to minimize the exposure to the toxic smoke during the evacuation. The similarity between FDS model and the fire scene is a must in order to obtain a realistic evacuation path. Chen *et al.* [41] developed a system integrating BIM and FDS. This system is using an old saved historical data generated by FDS for different fire scenarios and play it back in fire scenario in order to skip the heavy consumed time in simulation. It can be used to find a safe egress plan for firefighters and evacuees in case of fire during the operation of a building. This system helps both firefighters and normal building occupants to evacuate safely. The firefighting coordinator needs to be familiar with all the FDS results that are produced previously through simulation and be able to retrieve the most similar one to the actual fire scenario in order to deploy rescue teams into the building. This process depends on the firefighters' experience and the similarity between the fire scene initial condition and the ones that were simulated in FDS. Another issue with this implementation, if the building structure and the fire situation is too complex, the firefighter coordinator will have a problem to track all the fire aspects and the firefighter coordinates which will introduce human error to the equation.

Al-Nabhan *et al.* [21] have developed an IoT-based conceptual solution that consists of two levels of sensor nodes for data collection and processing. The system includes main nodes that handles the heavy processing and some sub-nodes that collects the data. The main nodes report the data to the cloud, which will determine the safest egress route. The issue with this approach is, the accuracy of the system is relying on deploying more sensors to have a high resolution grid, which is not cost efficient. Lossing any of these sensors in the fire, will cause information losses for this particular zone that can not be interpolated and lead to system error as the sensor network does not understand the physical property of the smoke. In addition to that, the evacuation path in this system is based on the current condition of the scene, which might not be efficient or accurate for long-term evacuation plans.

Peng *et al.* [42] proposed a method to obtain a fast and dynamic path for evacuation by using a neural network to fit a scoring function f or the paths to be able to predict the best evacuation routes. Their solution relies on generating a simple analysis model to represent the information in the space, then generating and evaluating different evacuation paths. The generated paths will be used to train a policy neural network to get the best evacuation route that helps the occupant and the fire safety team to make better decisions.

1.4 Objectives

In this thesis, a new smoke emulator is proposed to assist firefighters and evacuees in order to reduce the casualties and asset losses. The proposed emulator will fulfill the drawbacks of the existing solution that has a considerable latency and has no feedback to update the simulation to match the dynamic nature of the fire scene. The emulator will run in real-time based on a VFX smoke engine that has a very light processing time, and the deviation from the ground truth scene will be corrected using deployed sensors in the building. The sensors will emit the data to the cloud using an MQTT broker and the cloud will do the computations and update the smoke map. The system is connected with Long Short Term Memory (LSTM) that tracks the data and provides prediction when the data is temporally missed. The map assists the firefighter and will also update the exit sign based on a two-leveled A* algorithm in order to increase the survival rate and reduce the economical losses.

1.5 Contributions

The contributions of this work are as follows:

- A real-time smoke emulator that runs faster than the FDS and can interact with the events in the fire scene.
- A correction method that is relying on deployed sensors in the building in order to match the emulator results with the ground-truth status.
- LSTM layer that is used to predict the missing data in order to keep the simulator valid for a longer time in case the sensors' data are missing.
- Two-leveled A* algorithm that mimics human behavior in solving the routing problem and its usage in the fire scenario.

1.6 Thesis Outline

The reminder of this thesis is organized as follows:

Chapter 2 discusses the smoke engine and how to prepare the building model to the emulator. It explores the engine internal component, the drawbacks of the engine and the actions taken to solve these problems.

Chapter 3 explores the hardware different iteration and compares between all the designs to achieve the best performance hardware. It includes a comparison between the LoRa and WiFi and different operation mode achieved by ESP32 micro-controller SoC.

Chapter 4 discusses the usage of the deep learning and how the prediction model is built and its importance for the emulator stability. It compares between the ordinary neural network, the recurrent neural network and the long short term memory architecture. It compares the different architecture of the model and the error in each of them.

Chapter 5 introduces the routing algorithm which is used to control the exit signs to have adaptive egress route that is much safer than the current used fixed ones.

Chapter 6 discusses the results of the integrating all the previous layer and the outcomes of the new emulator with comparison to FDS.

Chapter 7 concludes the work and provides the future steps that can be followed to upgrade the current system.

Chapter 2

Smoke Engine

2.1 Introduction

The goal of this smoke emulator is to have a real-time behaviour, this can not be achieved unless the emulator includes a fast smoke engine, where it is the bottleneck of the system. To achieve this purpose, this emulator is built using different integrated components, as shown in Fig. 2.1, this guarantees having a reliable solution. The environment is prepared for the smoke engine, which is built to give real-time emulation results. The proposed system uses the IoT technology and the power of LSTM to correct any deviation between the emulator and the real situation. This design helps to have a perfect digital twin that reflects the fire scene with all its events and smoke development, which has a huge impact on firefighter's decision making. In the following sections, the system components will be explained in more detail. To speed the smoke emulation, the smoke engine is dealing with a projected space, where the smoke zone's pressure will be represented as smoke saturation. The saturation of the zone is calculated based on the amount of smoke in the zone. The incoming pressure values from the sensors, will be mapped to match the zone's saturation value. The 2D projected of the space will ignore some of the smoke's 3D properties, but it improves the speed of the emulator, and the correction layer will handle any missing information. This research focuses on a single building layout; when dealing with a 3D building, more components will be used to deal with the rest of the smoke properties.

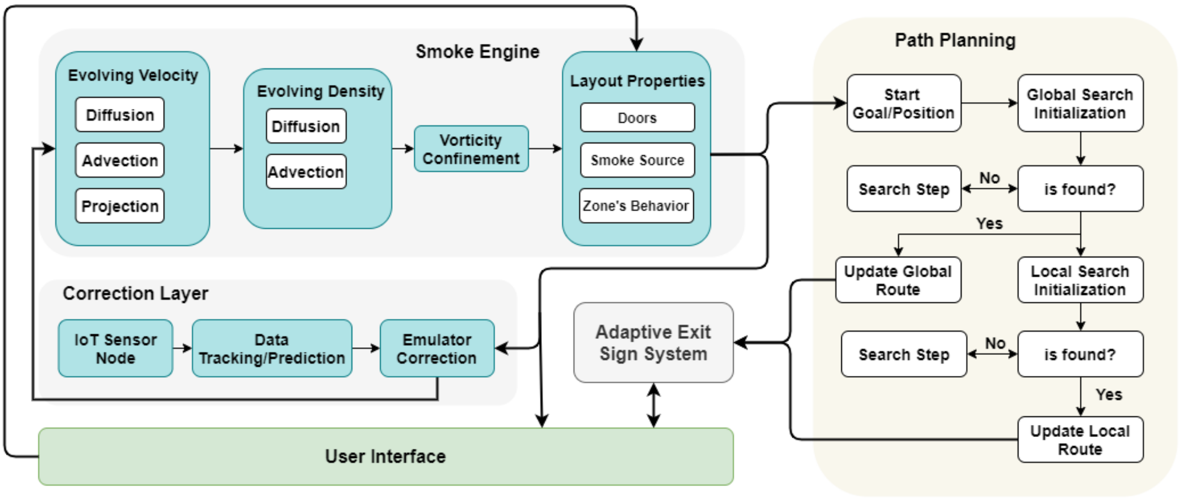


FIGURE 2.1: The Emulator Architect design

2.2 Building Layout Cases

Three different layout cases are used to test and evaluate the performance of the proposed system. Case 0 is the toy model, which is used as a playground to investigate and get the early insights about the system. It is used mainly to achieve the very basic initial stable smoke solver by tuning the hyper-parameters which will be explained in more detail in the following sections. Case 1 is used as the very basic introduction to the zone and the multi-wall layout structure to study and improve the advection algorithm. Case 2 is the major layout that includes multiple walls, doors and much more complicated structure for testing the correction algorithm. It is also used to evaluate the routing and evacuation algorithm as it has more doors and zonal relations. Figure 2.2 shows all the three layout cases.

2.3 Building Components

In comparison with existing methods, this proposed emulator deals with the building components differently. These components contain embedded information that affects the smoke properties, the evacuation plan, and the emulator correction algorithm. The emulator works with three elemental categories; doors, walls, and open space. These elements have information that can affect the density and the velocity parameters in the simulation. The Doors are special elements in this emulator acting as bridges between zones. They can increase the

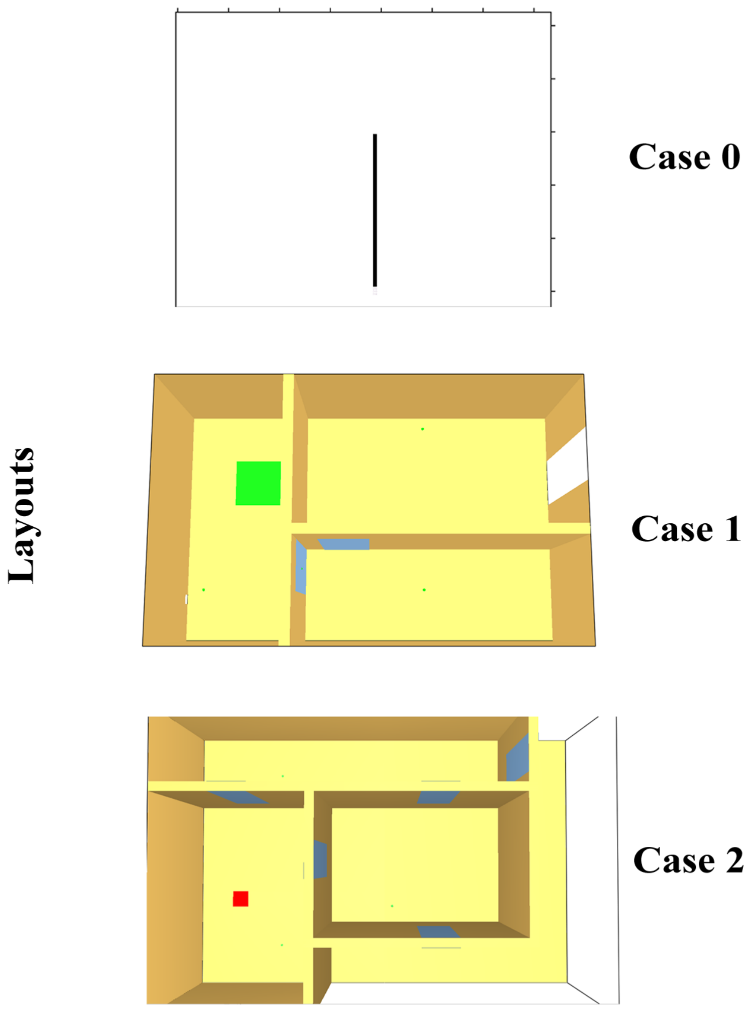


FIGURE 2.2: Layout cases

pressure of a zone over any adjacent zones when it is closed. It affects the smoke propagation where the smoke can leak through, and also it is a default exit point. The second component is the walls that acts as a rigid body, where it stops the smoke and defines the zones as well. The exit signs play an essential role during the evacuation, in this system, they are part of the IoT layer and can be controlled by the routing algorithm to guide evacuees to the safest route.

2.3.1 Space Discretization

In order to speed the emulator, the space is projected from 3D into 2D. The pressure in the zone can be obtained from the density grid, where the amount of smoke can reflect the amount of pressure in this zone. The emulator can not deal with all the points in the 2D space; it needs to discretize the space into cells such that the processing is fast enough. Two techniques have been adopted a fixed cell and adaptive cell sizes for decomposition, as seen in Fig. 2.3. The former is known as Exact Cell Decomposition [43], through which a space is divided into cells of similar squares to simplify the space to be used in the calculations. The latter, Approximate Cell Decomposition[44], may have different sizes for each cell. This method gives the most minimal representation of the discretized space. It works on grouping the fixed cells which belongs to the same zone into one big cell per zone. The zone is defined as anything that can be contained by four walls such as rooms or can be represented by regular four-linkage shape such as corridors.

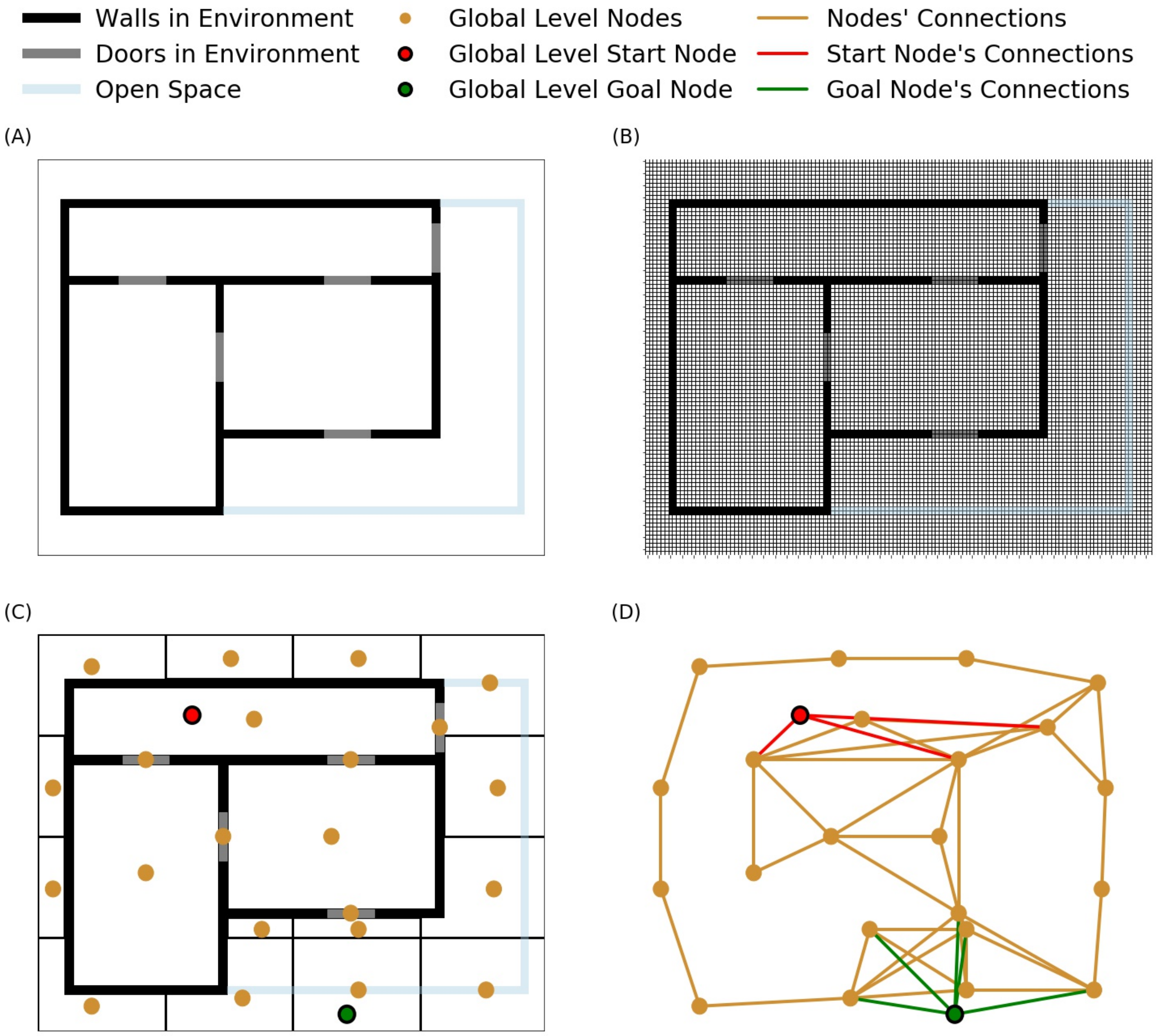


FIGURE 2.3: Two Levels Space Discretization

The requirements of both the smoke solver and the path planning processes of the emulator are kept in mind as well as maintaining a low processing cost for both of them while not giving up their performance.

2.3.2 Smoke Solver

Smoke is a computational fluid dynamics problem that can be solved using two different approaches. The first approach is the Euler representation where it assumes that the particles are living inside grid and it will move from cell to another. The second approach is the Lagrangian approach where it assume that the particles are moving around, this approach is harder to implement as the data structures that store the particles are always changing which is time and memory consuming. There are many formulas that is used to describe the smoke as a fluid dynamic problem such as Navier-Stokes equations.

Fluid dynamics software (FDS) is commonly used for fire and smoke simulation. If used properly, it can accurately model construction fires[45]. As any other CFD solver, it uses numerical methods to solve Navier-Stokes equations, and it requires pre-computation of the simulation cases thus taking long time before the start of the simulation. It is also necessary to pre-define all the events in the case such as which door will be opened at what time, the amount of energy in the fire and where it should start. The FDS is meant for the designing phase to change the design in order to have a safer building; however, it is used currently by most of the researcher in the operation phase. It does not fit in the fire situations due to the computational overhead of CFD based solvers along with their inability to adapt to dynamic events not known beforehand.

From literature, smoke simulation does not concern only BMS field but also the video effect industry (VFX) in order to simulate the smoke effects in videos games and movies. In VFX industry, it is required to have algorithms that run fast with low processing power, which pushes the industry to develop many techniques to achieve the illusion of the smoke performance. Some of the algorithms are based on alternating fixed smoke images to give the feel of the moving smoke. Some other applications require more advanced algorithms to interact with the smoke and these are derived from the CFD literature, mostly based on the Navier-Stokes Equations that demands a huge computational power. The proposed emulator is using one of these algorithm, which developed by Stam[46] as its basic building block. The basic algorithm is combined with other different layers to obtain a realistic smoke behaviour at a speed that allows for real-time emulation. This is made possible by additional components collaborate with the solver to deal with rigid bodies and adjust simulated results based on a correction layer that utilizes the feedback from the physical environment, provided by the IoT sensor network.

Stam introduced a simplified version of the Navier-Stokes Equations of fluid dynamics, by applying a two-step linearization. The first is the progression of the fluid's density over the simulation domain and the second is progression of the velocity vector field that affects the density. The algorithm would alternate between solving these two main steps alongside simpler steps to guarantee convergence in the simulation. The equation governing the flow's density ρ from Navier-Stokes Equations can be simplified to

$$\frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + k\nabla^2\rho + S, \tag{2.1}$$

where u is the velocity in an arbitrary dimension, k denotes diffusivity property of the fluid, and S represents smoke source injecting density into the simulation space. Whereas the equation governing the velocity fields can be written as

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \mu\nabla^2 u + F + \text{ Conservation Term}, \tag{2.2}$$

where μ is the viscosity property of fluid , and F is the injected force in the dimension .

2.4 Emulator Smoke Procedures

2.4.1 Smoke Diffusion

Smoke diffusion is the first layer that is used to diffuse the smoke particles in the space. The diffusion process depends mainly on smoke density where the smoke is moving physically from high concentration to low concentration even if there is no moving fluid around. The diffusion depends on the time step, the density grid, cell size and the diffusion rate hyperparameters. List 2.1 shows the diffusion implementation algorithm and figure 2.4 is showing the diffusion effect without adding any other layers such as the advection layer. The smoke is able to move by itself without any external forces. The diffusion is different when dealing with rigid bodies such as walls which will be explained later. This layer was originally proposed by Stam's solver, but it is modified in the proposed system to deal with the rigid bodies in the environment using the permitivity factor.

```
def diffuse(prop, prop0, diff, wall_flag):
    """
    Calculates the central difference coeffecients and calls the stable linear solver
    :param prop: the property at this moment
    :param prop0: the property at the previous state
    :param diff: the rate the property diffuses to neighbouring cells
    :param wall_flag: identifies the property at hand
    """
    # c1 , c2 : scales
    c1 = dt * diff * nx * ny
    c2 = 1 + 4 * c1
    stable_lin_sol(prop, prop0, c1, c2, wall_flag)
```

LISTING 2.1: Smoke diffusion algorithm

2.4.2 Smoke Velocity Advection

Advection is the process of moving smoke in the direction of the fluid, this process requires to generate velocity vector field developing overtime to move the smoke particles. The velocity vector in the emulator is generated by list 2.2 and the vector field changes at the rigid bodies boundaries which will be explained in the physical component section. Figure 2.5 shows the development of the velocity vector field at different time-step of the simulator, and it is obvious that the rigid body stops the velocity vector horizontal component and prevent the smoke from spreading through. Figure 2.6 shows the smoke advection process; however, the smoke does not include any turbulence, which is against the nature of the smoke fluid.

```
def advect(prop, prop0, u_x, v_y, wall_flag):
    """
    moves the property cells locations back in time to calculate the values
    of the current cells
    :param prop: the property at hand at the current moment
    :param prop0: the property at the previous moment
```

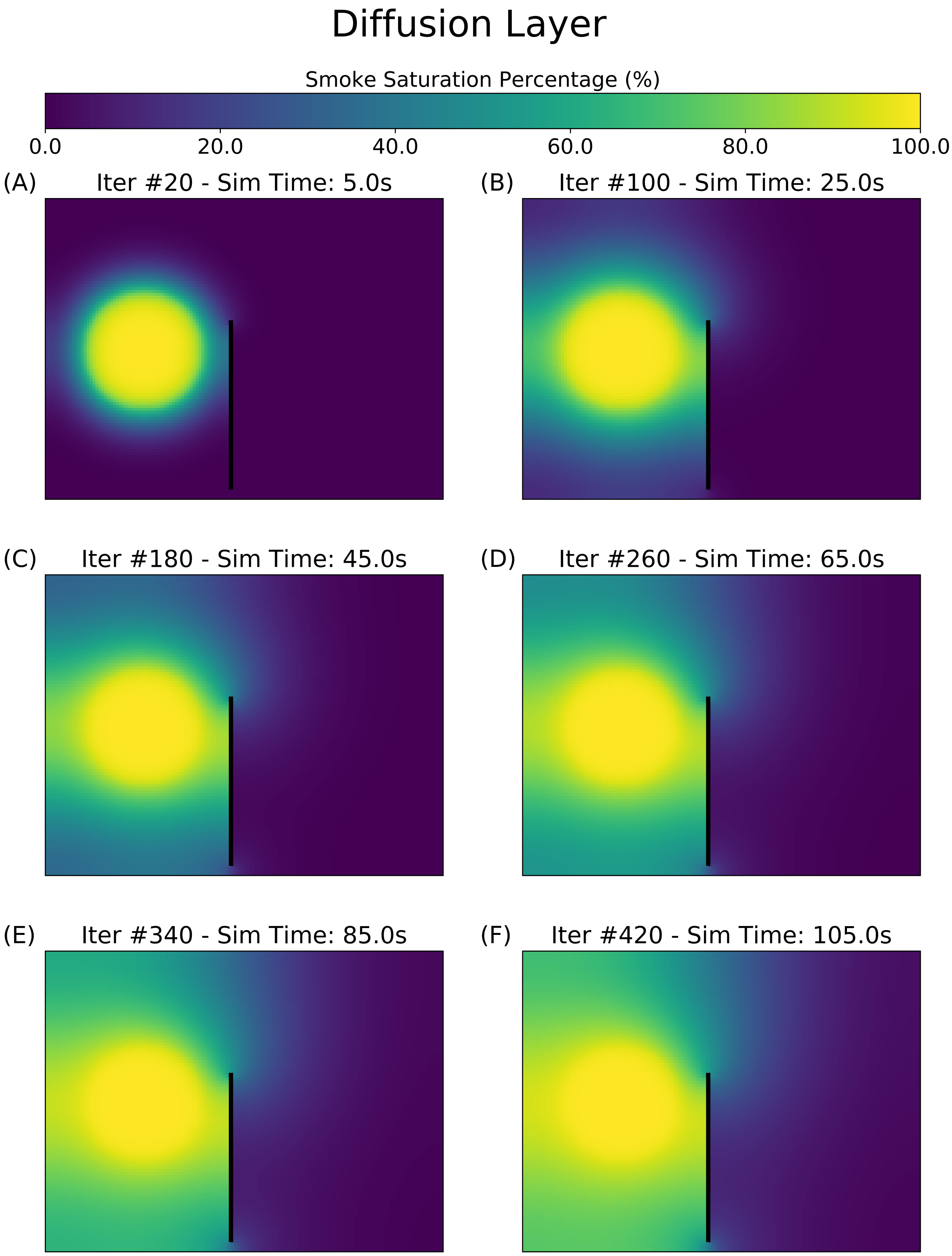


FIGURE 2.4: Smoke diffusion layer

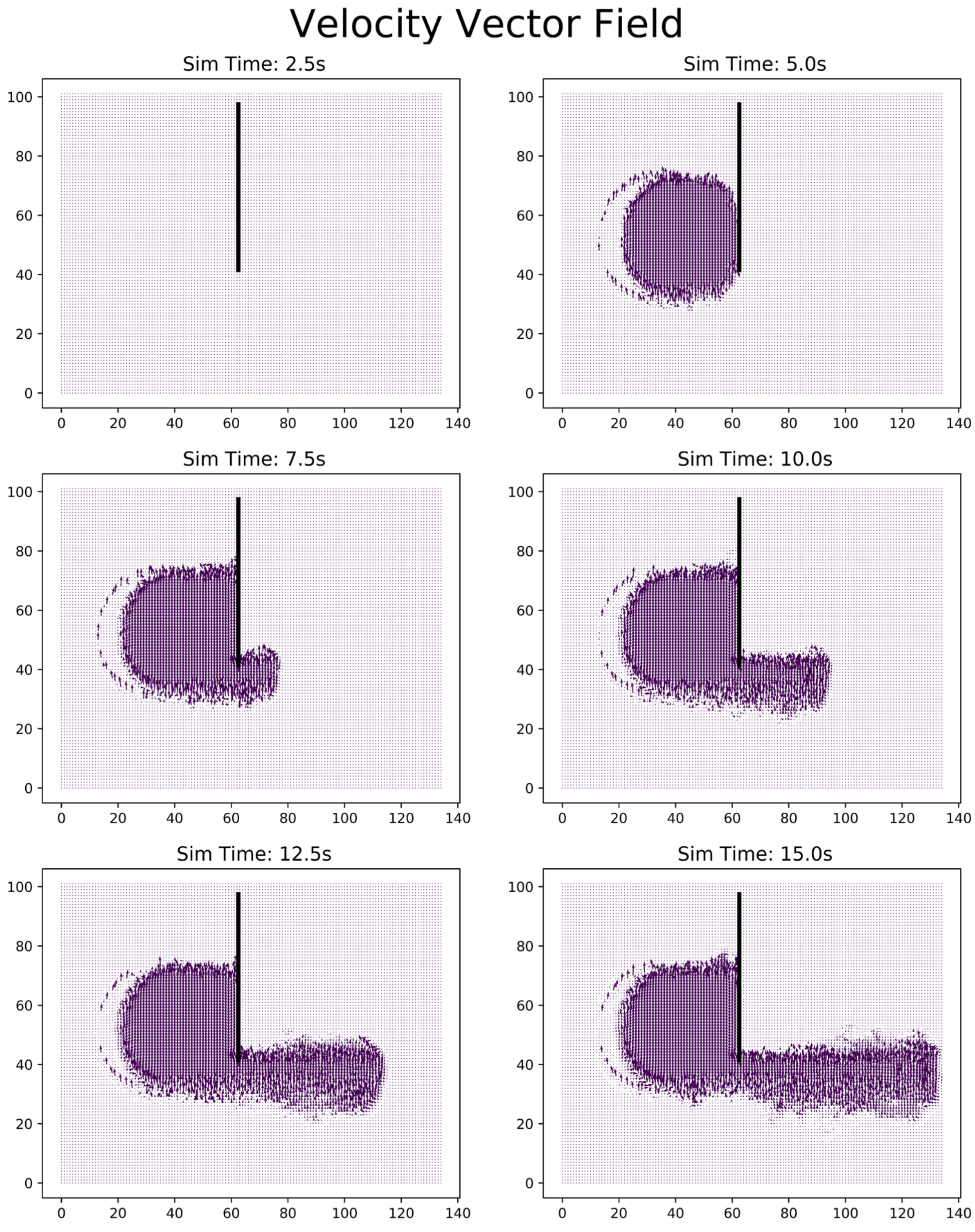


FIGURE 2.5: Velocity vector field

```
    :param u_x: the field that affects motion in the x-dir
    :param v_y: the field that affects motion in the y-dir
    :param wall_flag: identifies the property
    """
    # obtain the indices of the grid except for the BC cells outside the screen
    j, i = np.indices((ny, nx)) + 1
    x = i - dt * nx * v_y[j, i]  # the corresponding x-pos back in time
    y = j - dt * ny * u_x[j, i]  # the corresponding y-pos back in time

    # remove indices outside the x range of the window
    x_exact = np.minimum(x, nx + 0.5)
    x_exact = np.maximum(x_exact, 0.5)

    x0 = x_exact.astype(dtype=int)  # the x-positions prior to the exact x-value
    x1 = x0 + 1  # the x-positions post the exact x-value
    dx = x_exact - x0  # the ratio of inheritance

    # remove indices outside the y range of the window
    y_exact = np.minimum(y, ny + 0.5)
    y_exact = np.maximum(y_exact, 0.5)

    y0 = y_exact.astype(dtype=int)  # the y-positions prior to the exact y-value
    y1 = y0 + 1  # the y-positions post the exact y-value
    dy = y_exact - y0  # the ratio of inheritance

    # calculates the portion (the back in time cell) inherits from the nearest four cells
    y0_x0 = (1 - dy) * (1 - dx) * prop0[y0, x0]
    y0_x1 = (1 - dy) * dx * prop0[y0, x1]
    y1_x0 = dy * (1 - dx) * prop0[y1, x0]
    y1_x1 = dy * dx * prop0[y1, x1]

    # reset the cells densities after calculation
    if wall_flag == 'd':
        _reset_prop(prop)

    # the new cell density = the density at its location back in
    # time = sum of portions from the nearest four cells
    prop[idx(i, j, 0)] = y0_x0 + y0_x1 + y1_x0 + y1_x1

    bound_cond(prop, wall_flag)
```

LISTING 2.2: Advection algorithm

2.4.3 Vorticity Confinement Layer

The basic smoke solver has some drawbacks such as the gradual dissipation, which prevent it from depicting the smoke behavior realistically. The gradual dissipation happens because the

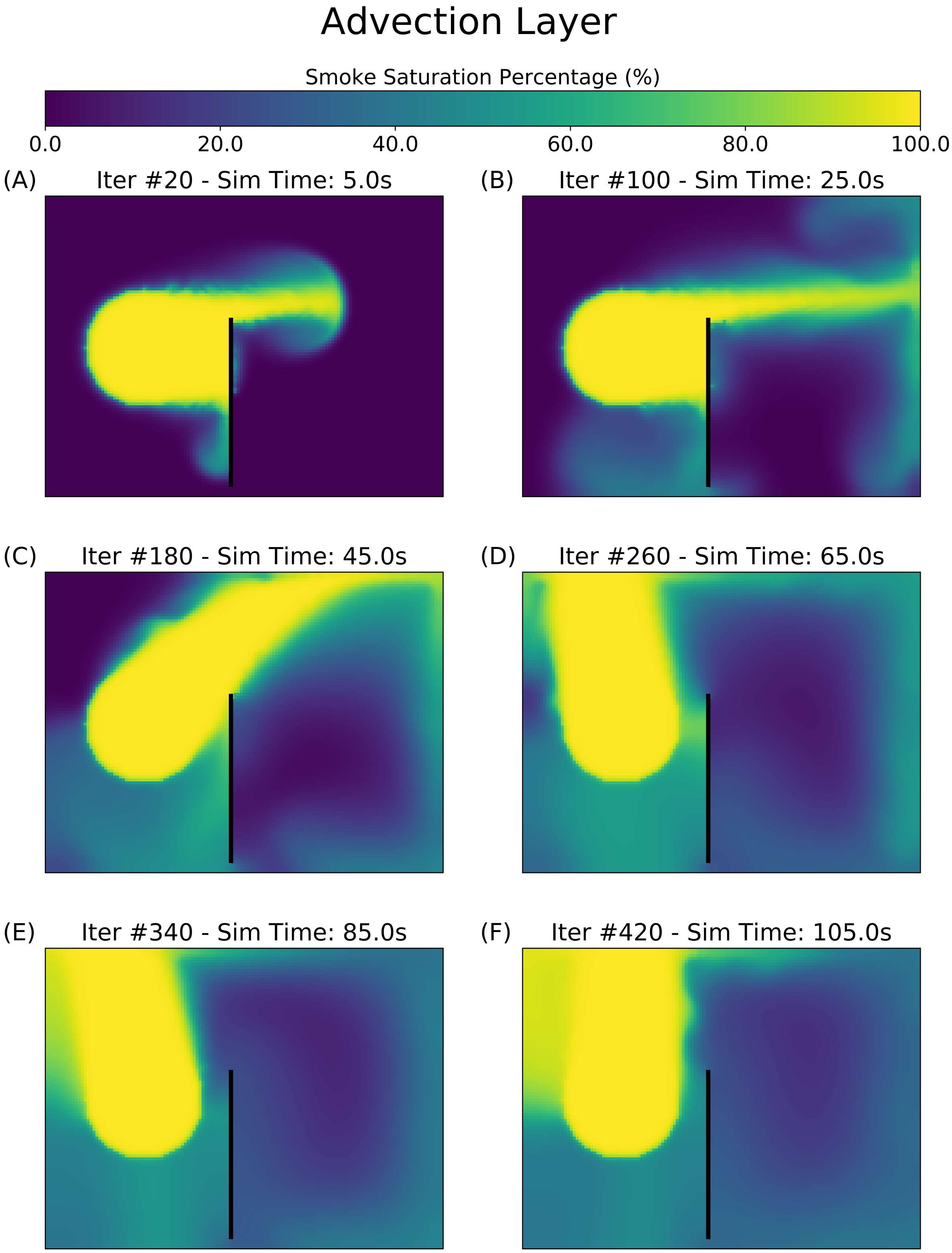


FIGURE 2.6: Advection layer

iterations of the linear solver that removes turbulent vortexes behavior of smoke propagation thus making it unable to simulate smoke accurately. In order to overcome the gradual dissipation, vorticity confinement technique [47] [48] is used to provide a solution for the vorticity dampening by injecting back the energy dissipated into the simulation. By modifying and integrating the vorticity confinement layer (VCL) in list 2.3 into the solver, it can simulate the vorticity of smoke by calculating the curl of the velocity vector using

$$\omega = \nabla \times u, \tag{2.3}$$

where ω denotes the vorticity value. The normalized force F is calculated and injected back to the velocity vector field using

$$F = \varepsilon h(N \times \omega), \tag{2.4}$$

where ε is the injection hyper-parameter, and h is the length of cell's edge, N which are normalized vectors from higher vorticity areas to lower vorticity ones.

```
def vorticity_confinement(vel_y, vel_x):
    """
    adds back in the dissipated energy from the diffusion step
    in the shape of smoke vortices
    :param vel_y: the velocity in the y-direction
    :param vel_x: the velocity in the x-direction
    :return: None
    """
    y, x = np.indices(vel_x.shape)

    # generate neighbours indexes
    y_1 = y + 1
    y_1[y_1 == y_1.shape[0]] = -1
    x_1 = x + 1
    x_1[x_1 == x_1.shape[1]] = -1

    curl = (vel_x[y_1, x] - vel_x[y - 1, x]) + \
           (vel_y[y, x - 1] - vel_y[y, x_1])

    curl_abs = abs(curl)

    vortex_x = curl_abs[y, x_1] - curl_abs[y, x - 1]
    vortex_y = curl_abs[y - 1, x] - curl_abs[y_1, x]

    vortex_abs = np.sqrt(vortex_x ** 2 + vortex_y ** 2)
    norm_vortex_x = vortex_x / vortex_abs
    norm_vortex_y = vortex_y / vortex_abs

    # to remove info or nan coming from devision by vortex_abs in case 0
    np.nan_to_num(norm_vortex_x, copy=False)
    np.nan_to_num(norm_vortex_y, copy=False)

    # (nx / (nx + ny))  = hyper-parameter curl:
    vel_x += (nx / (nx + ny)) * curl * norm_vortex_x
    vel_y += (ny / (nx + ny)) * curl * norm_vortex_y
```

LISTING 2.3: Vorticity confinement layer algorithm

Fig. 2.7 shows the effect of the vorticity confinement layer that boosts the small dampened vortices from the iterations of the linear approximation solver. Even though it adds little details compared to the results post integration of advection layer, these details are key behavior of smoke and can modify its propagation characteristics significantly with time.

2.4.4 Emulator Physical Components

The engine gives the user the ability to tune many parameters to achieve a realistic scenario close to FDS.

2.4.4.1 Fire Source Implementation

The user can control the fire properties such as the location on the map, the amount of the released smoke, the fire's life-time, and the fire's size. For instance, the system can measure the Oxygen level using IoT sensors and damp the fire accordingly if the oxygen is low. The fire source class in listing 2.4 shows the main methods and properties of the fire. The propel method is used to randomly add velocity component around the fire location in order to move the generated smoke that is assigned in the ignite method.

```
class Source:  # Smoke Source
    def __init__(self, pos, radius, fade, volatile, power):
        self.pos = pos
        # radius of the source (range of pixels that are randomized to be full)
```

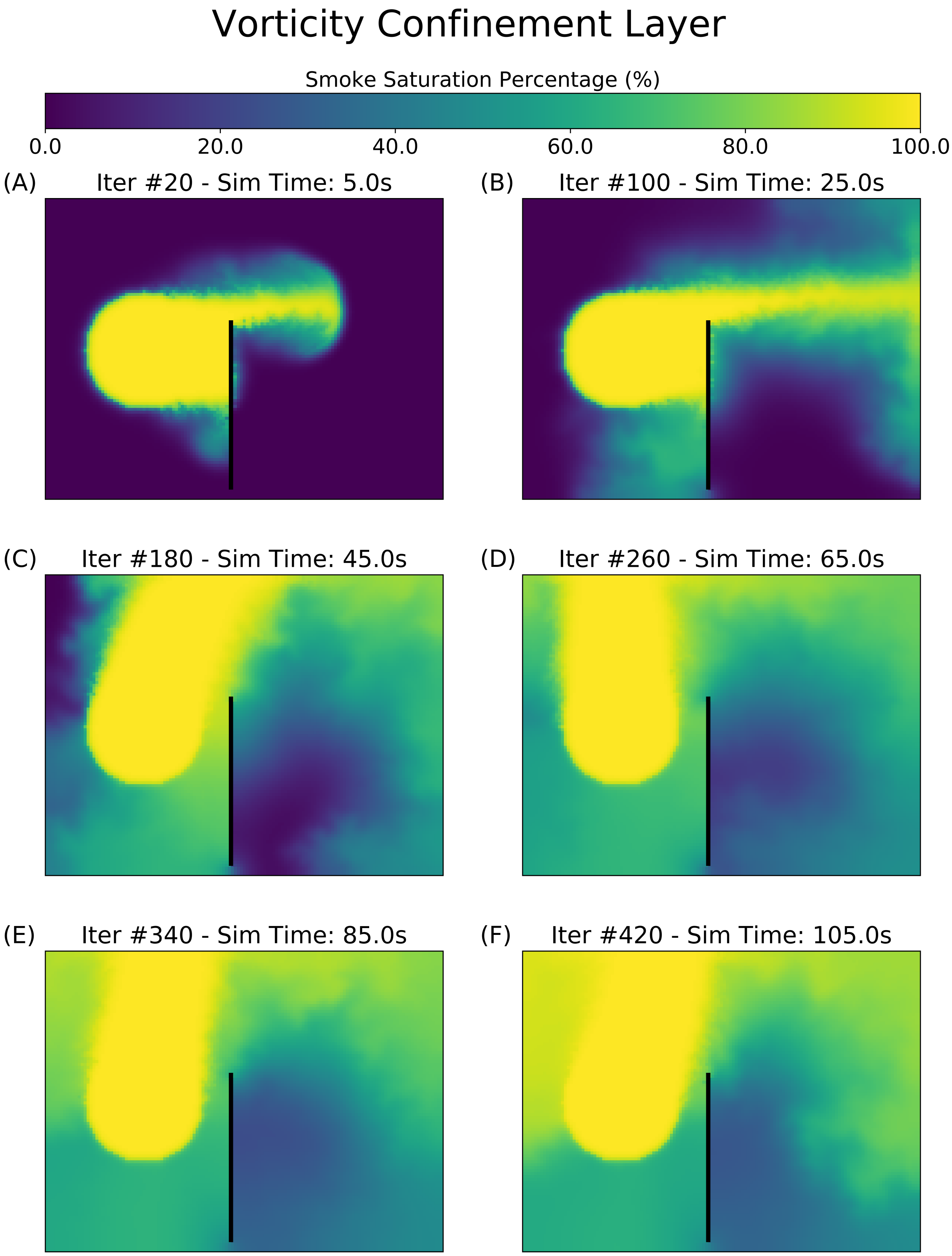


FIGURE 2.7: Vorticity confinement layer

```
        self.radius = radius
        # no. of pixels saturated per cycle
        self.intensity = power if power <= radius ** 2 else radius ** 2
        self.count = 0  # count down timer for putting off the source
        self.fade = fade  # make the source able to die out
        self.volatile = volatile

        shape_y, shape_x = np.indices((2 * self.radius, 2 * self.radius))
        idx_x = shape_x + (self.pos[0] - self.radius)
        idx_y = shape_y + (self.pos[1] - self.radius)
        distance = np.sqrt((idx_x - self.pos[0]) ** 2 + (idx_y - self.pos[1]) ** 2)

        self.indexes = idx_y[distance <= self.radius].ravel(), \\
        idx_x[distance <= self.radius].ravel()

    def propel(self):
        u_x, v_y = np.mean(vel_x[self.indexes]), np.mean(vel_y[self.indexes])
        theta = np.arctan2(u_x, v_y)
        vel_x[self.indexes] = 0.25 * np.sin(theta)
        vel_y[self.indexes] = 0.25 * np.cos(theta)

    def ignite(self):
        """
        Fills the density grid cells within the source radius with smoke randomly
        """
        # allows overlapping - simple approach
        #for counter in range(self.intensity):
        if self.volatile:
            rand_indexes = random.choices(range(len(self.indexes[0])), k=self.intensity)
            y = self.indexes[0][rand_indexes]
            x = self.indexes[1][rand_indexes]
        else:
            y, x = self.indexes

        if self.count <= 150:
            dens[y, x] = 255 - self.count  # value decreases gradually
            self.count += 1 if self.fade else 0
        else:
            sources.remove(self)
```

LISTING 2.4: Source Class

2.4.4.2 Walls Implementation

The implement_walls function works the same way as the bound_cond function. Simply giving the walls' cells opposite velocities to those of their neighbors to reflect the smoke when it hits the wall. The wall's cells will have the same density as their neighbours to prevent smoke from swapping between the walls and the zones with additional conditions to prevent that as well.

```
def _implement_walls(prop, flag):
    """
     Keeps the behaviour of the walls consistent
    :param prop: the property to correct the conditions of
    :param flag: the property flag - distinguish different properties
    """
    # if speed in x-dir make the velocity of wall cells the reverse of the neighbour velocities
    if flag == ’x’:
        grid.vel_x[dpkg(grid.wall_indexes_v)] = -grid.vel_x[dpkg(grid.neigh_prime_v)]
    elif flag == ’y’:
        grid.vel_y[dpkg(grid.wall_indexes_h)] = -grid.vel_y[dpkg(grid.neigh_prime_h)]
    elif flag == ’c’:  # if gradient of velocity field (called by energy conservation step)
        # copy property to keep behaviour consistent
        prop[dpkg(grid.wall_indexes)] = prop[dpkg(grid.neigh_prime)]
    else:  # if the property is the density
        # copy property to keep behaviour consistent
        prop[dpkg(grid.wall_indexes)] = prop[dpkg(grid.neigh_prime)]
        prop[dpkg(grid.lines_edges)] = prop[dpkg(grid.edges_neigh)]
```

LISTING 2.5: Wall algorithm

2.4.4.3 Doors Implementation

Doors are not ordinary obstacles as they normally block the smoke when the pressure difference between the door’s sides are small and allows portion of smoke to leak at high pressure difference. They have two concepts in our implementation. The first is the proportionality of a leak’s volume to the gaps in the door frame. These gaps are represented as cells with a permittivity factor that allows a portion of the smoke’s particles traversing them to pass while blocking the rest.

To implement this concept, another property is added to the emulator holding the permittivity values. This layer is integrated with the density advection step of the basic solver. The original solver density advection step[46] is replaced with the following proposed density advection equation

$$d^{t}_{i,j} = |d^{t-\Delta t}_{i,j} + (1-\varepsilon_{i,j})(d^{t-\Delta t}_{i,j} - \sum_{k=0}^{1}\sum_{l=0}^{1} \delta x_k \, \delta y_l \, d^{t-\Delta t}_{k,l})|, \tag{2.5}$$

where $d^{t}_{i,j}$ denotes the cell’s density post advection, $d^{t-\Delta t}_{i,j}$ denotes the cell’s density prior to advection, Δt denotes the time step value, $\varepsilon_{i,j}$ denotes the cell’s permittivity, δ_{x_k} denotes the neighbor cell’s distance to the prior center in x-axis, δ_{y_l} denotes the neighbor cell’s distance to the prior center in y-axis, and $d^{t-\Delta t}_{k,l}$ denotes the neighbor cell’s density prior to advection. The factor of the cross-section can take values from 0 to 2, to control the cell’s permittivity as the cell acts as a rigid body at 0, and as an open space at 2.

This proposed step simply updates the cell’s density as the density prior to the advection with the addition of a density change controlled through the permittivity hyperparameter.

The second concept is leaking velocity based on the pressure difference, it is implemented in list 2.6. It alters the boundary condition of the original solver for the velocity advection step to control the amount of perpendicular reflected velocities by the door. The proposed step for implementing leakages fully depends on the neighboring cells on the side with higher pressure, which is computed from

$$w_{i,j} = -\left(1 - c_{i,j}\right)\, w_{P_{high}}, \tag{2.6}$$

where $w_{i,j}$ represents the velocity component normal to the door, c_{i_j} represents the leakage hyper-parameter, and $w_{P_{high}}$ represents the cell’s neighbor on the higher pressure side. $c_{i,j}$ is determined in terms of the cross section of the door.

```
def _implement_doors(prop, flag):
    """
    Keeps the behaviour of the walls consistent
    :param prop: the property to correct the conditions of
    :param flag: the property flag - distinguish different properties
    """
     # get any obstacle cells that should leaks
    leaking_indexes, leaking_neigh = _find_leaks()
    # getting the cells ont the other side in order to leak to
    shift_leak = leaking_indexes - leaking_neigh

    if flag == ’x’ or flag == ’y’:  # if speed in x-dir or not density > mirror
        # make the velocity of wall cells the reverse of the neighbour velocities
        grid.vel_x[dpkg(grid.doors_indexes)] = - 0.99 *\\
        grid.vel_x[dpkg(grid.doors_neighbours)]
        grid.vel_y[dpkg(grid.doors_indexes)] = - 0.99 * \\
        grid.vel_y[dpkg(grid.doors_neighbours)]
        if any(leaking_indexes[0]):  # if any cells should leak
            # copy the leaking indexes velocities from neighbouring indexes
            grid.vel_x[dpkg(leaking_indexes)] = grid.vel_x[dpkg(leaking_neigh)]
            grid.vel_y[dpkg(leaking_indexes)] = grid.vel_y[dpkg(leaking_neigh)]
            # copy the same velocities to the cells on the other side
            # as well (create leaking channels/tubes)
            # control the leaking speed (0.0 to 1.0)
            grid.vel_x[dpkg(leaking_indexes + shift_leak)] = 0.4 * \\
            grid.vel_x[dpkg(leaking_neigh)]
            grid.vel_y[dpkg(leaking_indexes + shift_leak)] = 0.4 *\\
            grid.vel_y[dpkg(leaking_neigh)]

    # if gradient of velocity field (called by energy conservation step)
    elif flag == ’c’:
      # copy property to keep behaviour consistent
        prop[dpkg(grid.doors_indexes)] = prop[dpkg(grid.doors_neighbours)]
    else:  # if the property is the density
        # copy property to keep behaviour consistent
        prop[dpkg(grid.doors_indexes)] = prop[dpkg(grid.doors_neighbours)]
        # if the leakage memory has elements (the sim includes walls)
        if len(grid.leak_memory):
            grid.leak_memory += 0.05 * prop[dpkg(grid.leak_neigh)]
```

LISTING 2.6: Doors algorithm

2.4.5 Emulator Hyper-parameters Tuning

The following section explains the process of calibrating and tuning the emulator’s hyper-parameters, which affects the smoke behavior in the proposed emulator. Figure 2.8 is showing the performance of two different simulation values when the diffusion rate equals 0.02 on the right and when the value is 0.00002. The higher value of the diffusion rate will cause the source to dampen quickly as shown in figure 2.8, where a bigger portion of the smoke will be distributed from the cell to the neighbours.

The viscosity is a material property, which identify the friction between the layers of the fluid. Figure 2.9 shows that the fluid can adapt to the viscosity where on the left side the fluid is moving slower than the right side because of its high viscosity value.

The VCL works on boosting the dampened vorticity to increase the smoke turbulence, it uses a vorticity hyper-parameter which can effect the fluid performance as shown in figure 2.10.

Figure 2.11 shows the effect of the iteration that the solver takes to to produce a single time-step frame, more iteration will give a more detailed simulation; however, it takes longer time which is not allowed in this system.

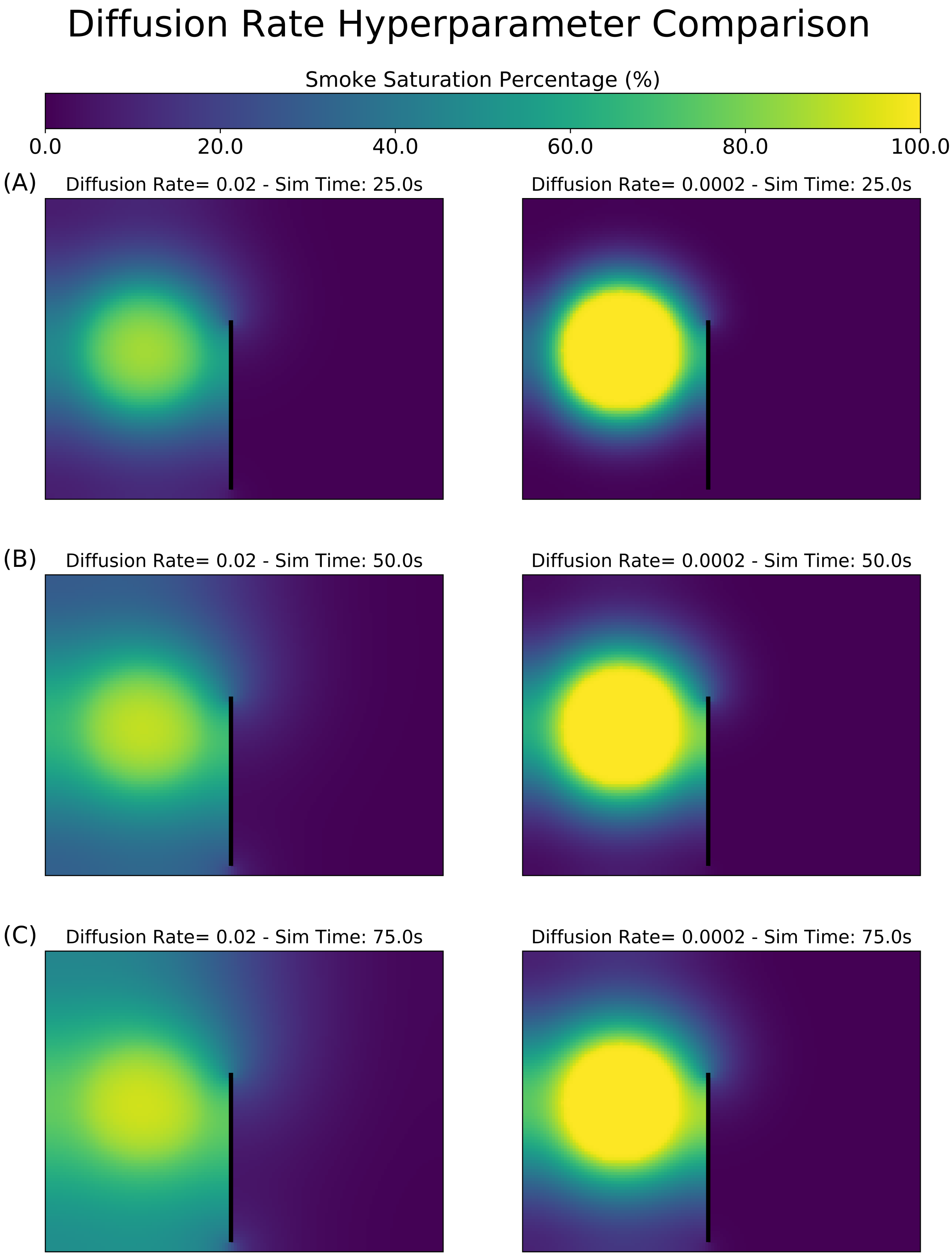


FIGURE 2.8: Diffusion rate hyper-parameter comparison

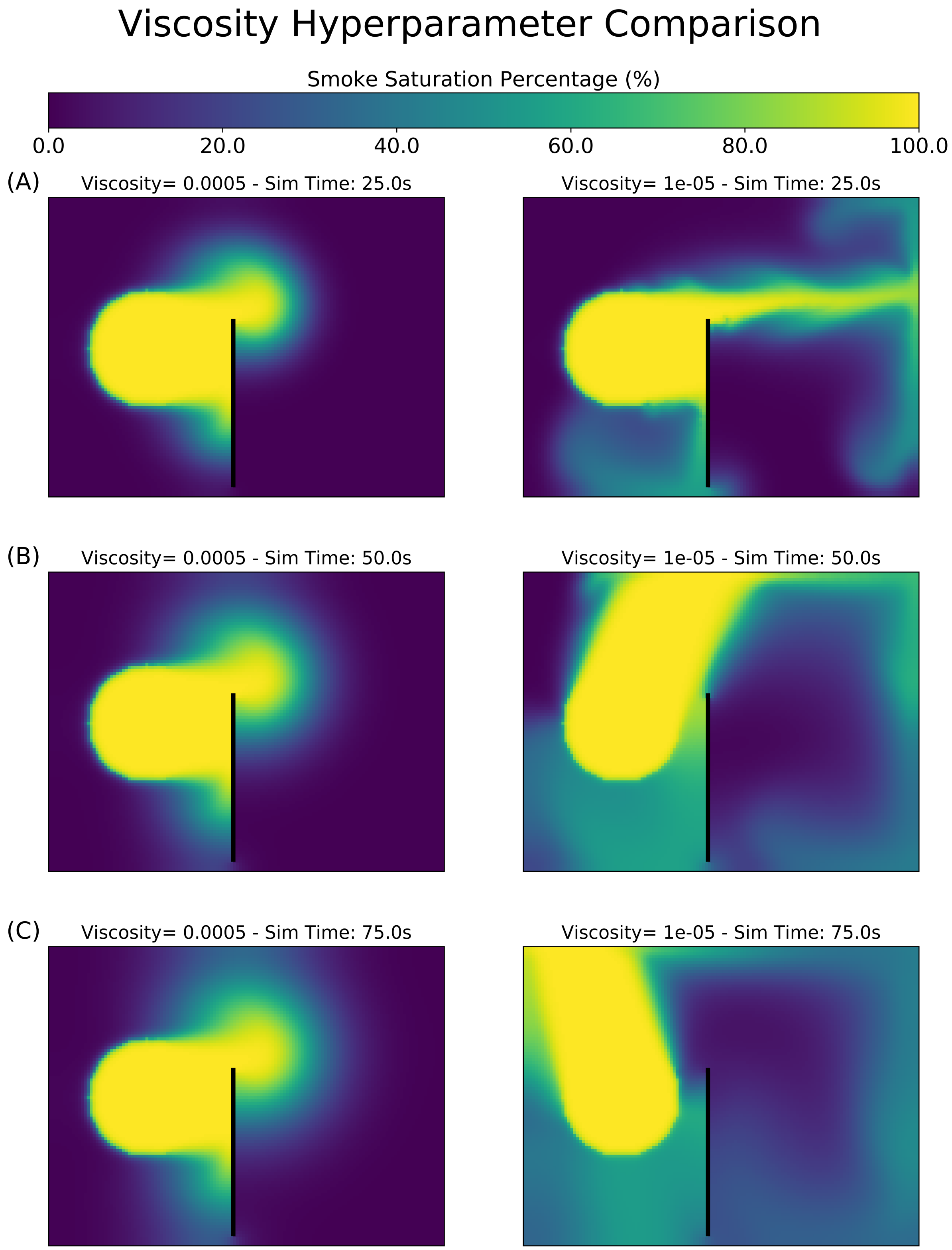


FIGURE 2.9: Viscosity hyper-parameter comparison

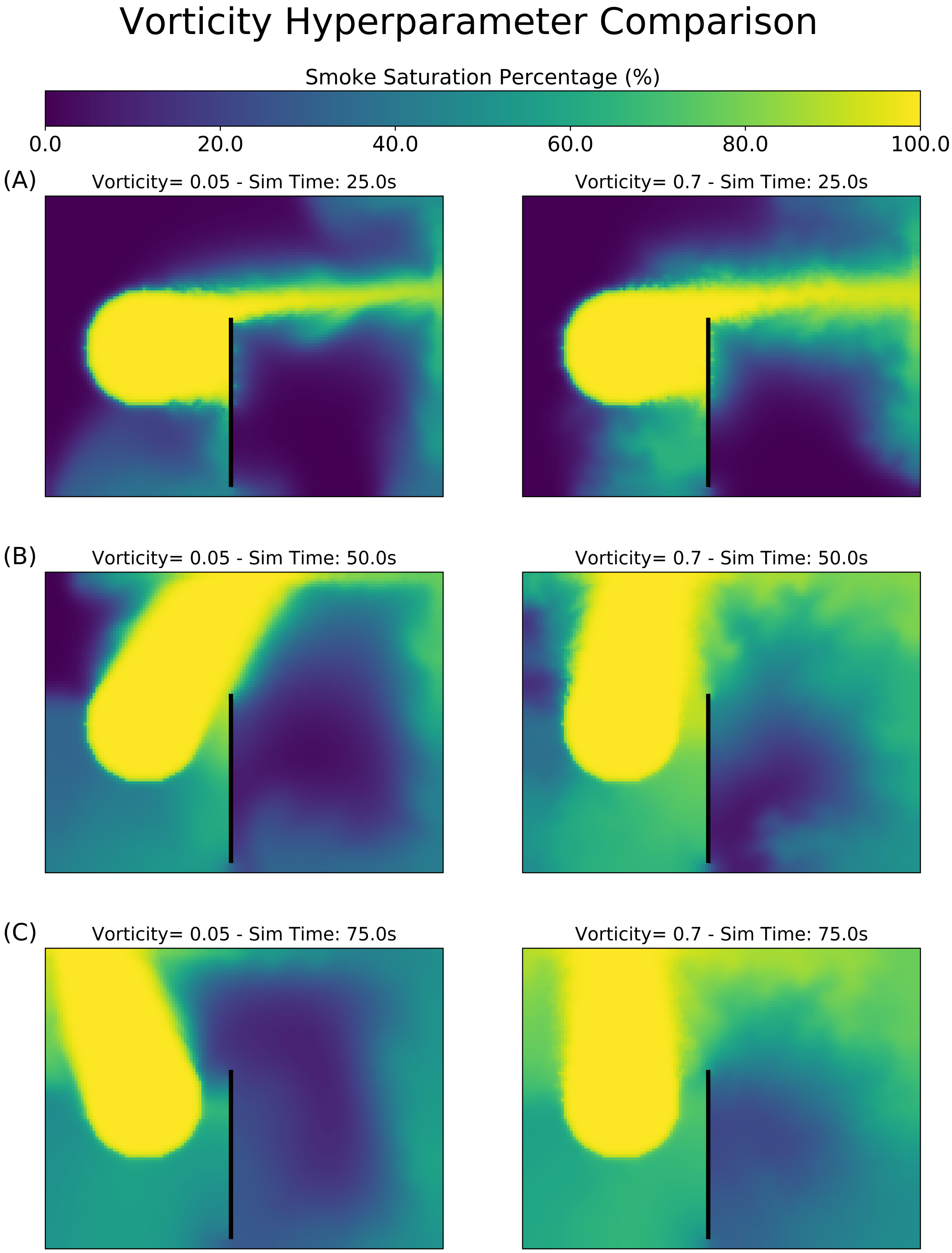


FIGURE 2.10: Vorticity hyper-parameter comparison

TABLE 2.1: Final emulator hyper-parameters

Parameter	Value
time step	0.25
cell size	6
viscosity	.0000001
diffusion rate	0.0002
vorticity rate	0.7
number of iterations	12

Table 2.1 shows the final tuned hyper-parameters that is used for the emulator and gives behaviour close to FDS after further analysis.

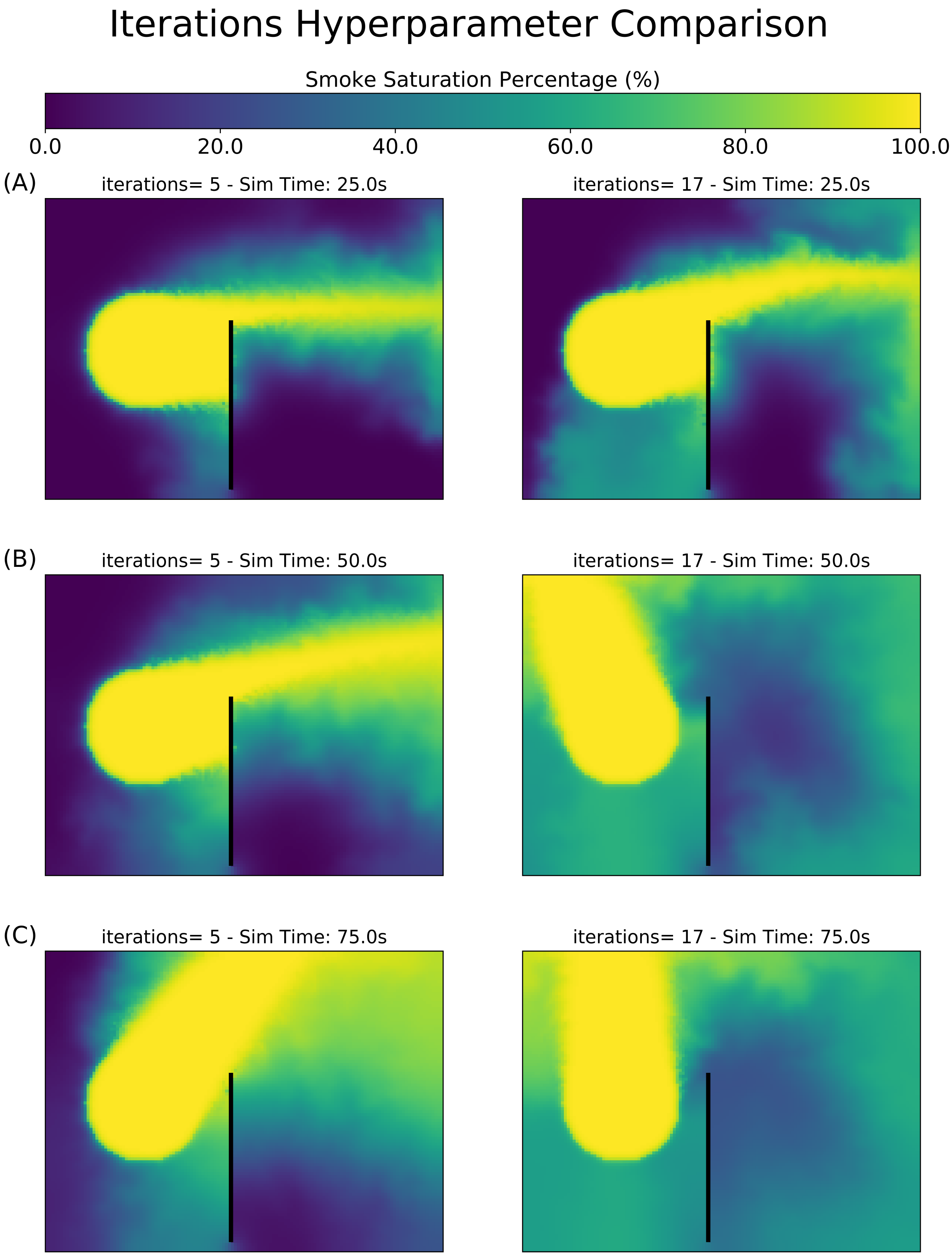


FIGURE 2.11: Iterations hyper-parameter comparison

Chapter 3

Hardware Nodes

3.1 Introduction

The Internet of things has a remarkable impact on many fields includes, but not limited to, agriculture, environment, smart cities, smart building, and health systems. It helps seamlessly integrating wireless sensors network (WSN) with the cloud, which eases data acquisition. The collected data help to build powerful analysis models to solve a wide range of complicated problems. The new WSN covers different scale from smart house to large urban area that makes it able to cover many fields such as space monitoring, tracking assets, navigation systems, and more. The growth of IoT is facing many challenges to make it more reliable such as security, power consumption, communication range, and latency. These challenges requires a new hardware implementation to run faster with less power, and new communication technologies to cover vast distances with high encryption light-weight algorithm. It also needs low latency cloud protocols to transfer the data in real-time to enhance the system response in order to get data or take actions. In the following sections, the iterations of the hardware implementation will be discussed to illustrate the steps that were taken to ensure low latency and low power consumption without giving up the performance.

3.2 Sensors Node

The emulator uses the sensors' data to correct the results from any deviation that might occurs due to the light-weight smoke engine. This will make digital twins [49], which helps in understand the fire scene better. The nodes also help to detect different events in the environment more than the pressure and temperature. It can detect the door status, which will cause smoke to spread if it is opened. The nodes help in monitoring the fire development better to make better decisions. The IoT devices measure the temperature, pressure, humidity, motion activity ,and door's status, which is used by the emulator engine and the evacuation plan. The nodes went through different iterations in order to reach the optimal design that fits the problem.

The optimal design should fit the data type. Normally, the data has two types, where it might change with a low rate ,such as the temperature inside the building. it can be measured each minute instead of each second, which is acceptable in this context. However, the temperature in the fire scenario is crucial, where the emulator needs it at a higher rate than usual. The emulator should receive an update each on 250 millisecond . The hardware needs to adapt to the system's requirements to reach the optimal design.

The node has three design iterations. the first iteration uses LoRa communication that supports low power consumption for data transmission and can cover big areas. The second iteration uses WiFi to transfer the data with low latency to fit the emulator's needs. The final iteration combines the advantages of the first two designs in order to save more power, which is a huge key factor for a reliable IoT solution.

3.2.1 First Design: LoRa Communication

Iteration one uses LoRa for communication as it covers long-range up to 15 km and has a low power consumption[50][51][52]. The typical LoRa architecture contains an end-device with LoRa transceiver, a gateway with LoRa transceiver, and an internet connection and a cloud, as shown in figure 3.3. LoRa communication uses the chirp spreading spectrum figure 3.1, which modulates the chirps to encode the data. By using Software-defined radio setup and listening to frequency 902.3 MHz, the graph in figure 3.1 is plotted as a demodulated FM signal. The chirps are a linear frequency change that represents different symbols by different frequencies sequences to describe the payload, as shown in figure 3.1. LoRa has two different transmis-

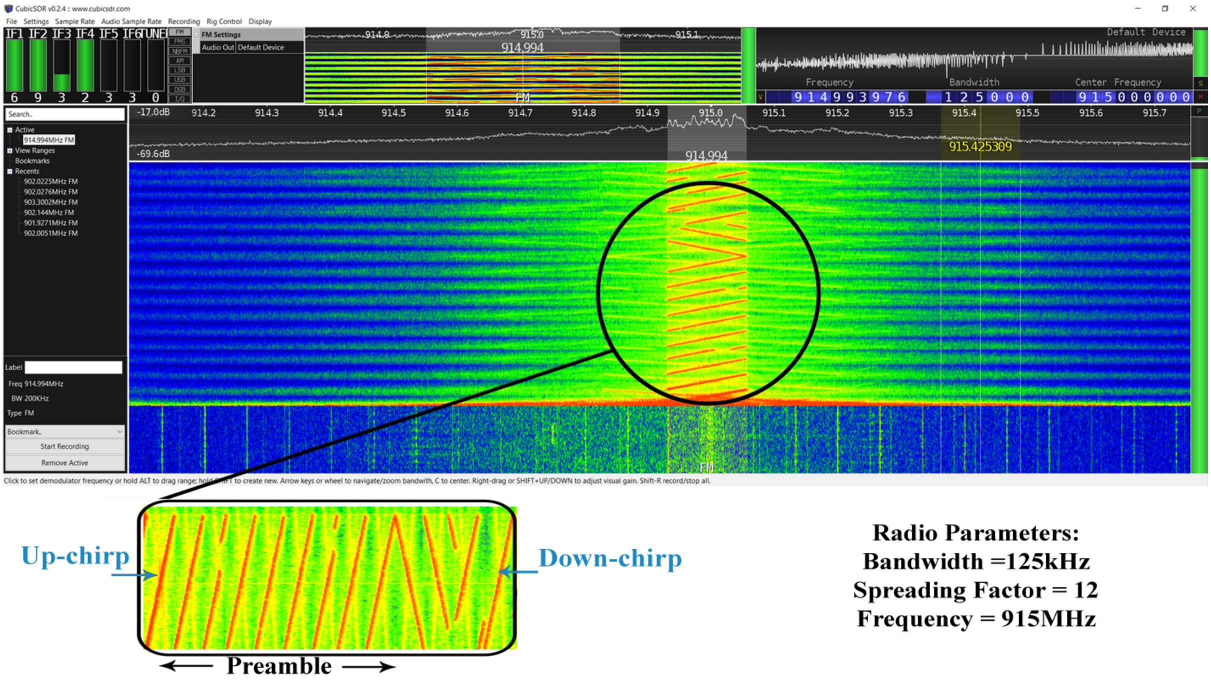
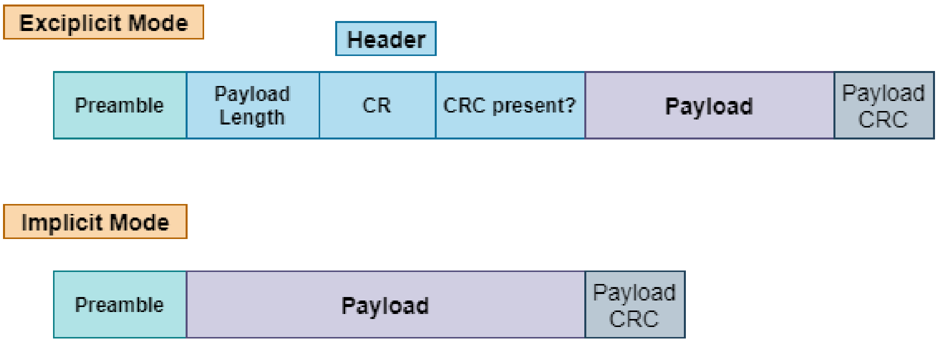


FIGURE 3.1: Chirp Spreading Spectrum Modulation



Exciplicit Mode

	Header				
Preamble	Payload Length	CR	CRC present?	Payload	Payload CRC

Implicit Mode

Preamble	Payload	Payload CRC

FIGURE 3.2: LoRa Packet Structure

sion mode packets, as shown in figure 3.2. The explicit mode contains a header that helps to recover the data if it has some corruption , and the implicit mode does not include the header in the package which is faster than the first mode. The LoRa transmission starts the message with a preamble that contains eight up-chirps to inform the receiver for start reception. LoRa's radio parameters can control the latency, the data rate, the time-on-air, and the coverage of the device. The radio parameters need to be matched between the transmitter and receiver to have data transfer. For instance, the spreading factor must be the same for the transmitter and receiver. The chips construct the symbols that form the message. The spreading factor parameter (SF) can control the number of chips in the symbol where

$$chips/symbol = 2^{SF}; \{SF = 7,8,9,10,11,12\}. \tag{3.1}$$

The bandwidth parameter control the time, that the chip takes on air where

$$T_{\text{on-air}}/chip = \frac{1}{BW}; \{BW = 125,250,500KHz\}. \tag{3.2}$$

Another important parameter in the LoRa payload is the code rate, which defines the ratio between the cyclic redundant check (CRC) and the data in the message. The CRC value can be calculated from

$$CRC = \frac{4}{4+crc}; \{crc = 1,2,3,4,5\}. \tag{3.3}$$

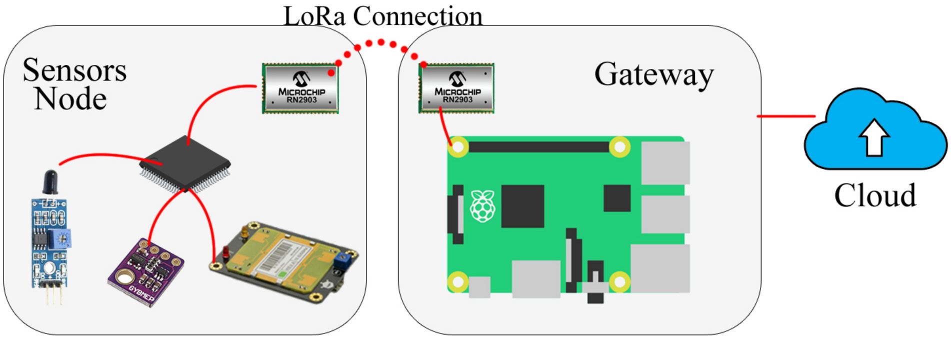


FIGURE 3.3: LoRa System Architect

By considering a message that has 10 bytes of data, preamble = 8 bytes, CRC = 4/5, BW = 125 KHz, and SF = 7; this message will take 41.22 milliseconds , which is a significantly considerable number. This node must be silent for 4.08 seconds before the next transmission according to the ISM regulation that requires a duty cycle equals .1-1% for each device on the channel. Tuning the radio parameters can reduce the latency, as shown in figure 3.4 and table

3.1. The only advantage of this technology is transmitting a low data rate that covers a wide range with low power consumption. Figure 3.4 shows the performance of the LoRa with different radio parameters. Changing the bandwidth and spreading factor can control the coverage distance, the data rate, and the on-air-time, which can specify the number of nodes in the WSN. Increasing the SF will increase the time-on-air, for instance, SF 8 will take twice the time of SF 7 when bandwidth is the same. By comparing case 2 and case 4 from table 3.1, it is clear that the RSSI signal is almost the same in these two conditions, but in case 2, it takes four times more time-on-air, which consumes more power and reduce the number of the devices in this channel. Selecting a good operating parameters for LoRa node can help to save more power, reduce the data latency, and have more nodes in the network.

TABLE 3.1: LoRa Radio Parameters Comparison the data represents average of 100 payload measurements for indoor operation

Case	SF	BW (KHz)	Time-on-Air(ms)	RSSI
case1	7	125	41.2	-48
case2	12	125	991.3	-65
case3	7	500	3.4	-59
case4	12	500	247	-65

The LoRa design uses RFM95 LoRa module, BME280 pressure sensor, Atmega 32u4 micro-controller. The current consumption is measured to optimize all the designs by using INA219 module that allows to measure the current using I2C interface with a micro-controller. Figure 3.5 shows that the current consumption of the node has average equals 42.5 mA, which will consume a 3000 mAh battery in 70.59 hours.

3.2.2 Second Design: WiFi Communication

The second method scenario uses WiFi. It is consuming more current, as shown in figure 3.6, where the average current consumption equals 75.3 mA. This design will allow a 3000 mAH battery to run this node for 39.84 hours, which is a concise life span. However, it is suitable for short-range and high data-rate. ESP32 micro-controller orchestrates the sensors in the hardware node and handles the wireless communication with its on-board WiFi peripheral. It uses WiFi to communicate with the emulator using MQTT, which is a very light-weight messaging protocol [53][54]. MQTT has two main components, the broker and the end-devices. The devices are connected to the broker with an open connection to transfer data both ways

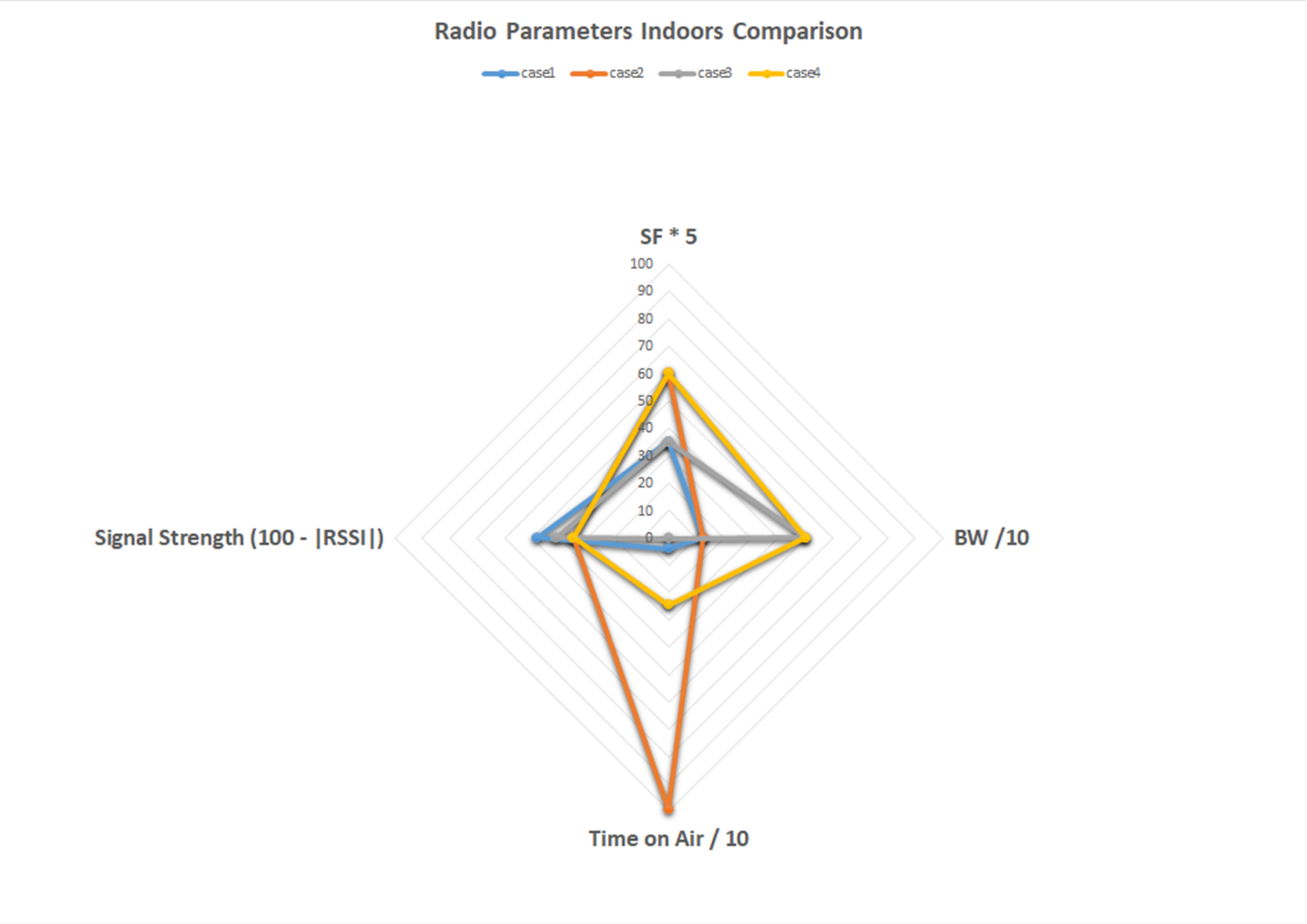


FIGURE 3.4: LoRa Radio Indoors Comparison

at any time. The devices send list of the subscribed topics to the broker when it initiates the communication for the first time . The node publishes messages on a specific topics according to the data type; then the broker routes these messages to all devices that are subscribing to this topic. MQTT has parameters that guarantees high data delivery named quality of service (QoS) according to table 2. This protocol helps to connect all the sensors nodes and exit signs to the emulator seamlessly to get the data and control the evacuation routes. Figure 3.7 shows the structure of this scenario. The motion sensor uses microwaves to detect the motion based on Doppler effect: however, it has glitches coming from the WiFi signals which requires connecting the sensor to interrupt service route to count the frequency of the incoming pulses to distinguish between the glitches and the actual motion.

The node uses QoS equals two as the data delivery is a must to correct the emulator.

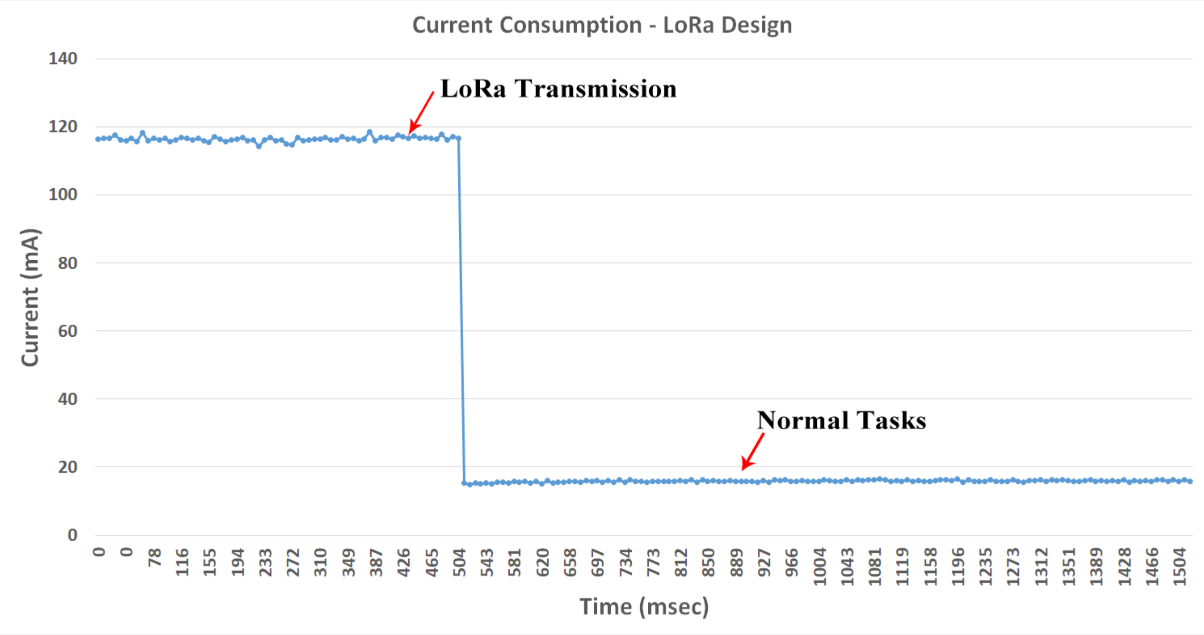


FIGURE 3.5: LoRa Design Current Consumption

TABLE 3.2: Quality of Service Parameter

QoS Value	Description
0	at most once delivery
1	at least once delivery
2	Exactly once delivery

3.2.3 Final Design: Dual Communication

The core concept in this implementation is the data type. When the data has slow-rate in normal circumstances, it will be reported at a lower rate, for instance, every 60 seconds. When there is a fire, the data should be reported at a high rate to update the emulator faster. The architecture of the ESP32 micro-controller allows combining these to methods using its SoC peripherals. ESP32 has two cores, ultra low power processor, and RTC unit. The ESP32 can go to sleep mode and be programmed to wake-up after certain time using the RTC, which has also interrupt pins that can wake-up the ESP32 at any time from any external source such as fire sensor or motion detector.

Figure 3.8 shows the state operation of the low power design, wherein normal operation, ESP32 wakes up to measure pressure, temperature, and humidity. After the data measurements, it initiates the LoRa module, then sends the data and goes back to sleep mode for 60

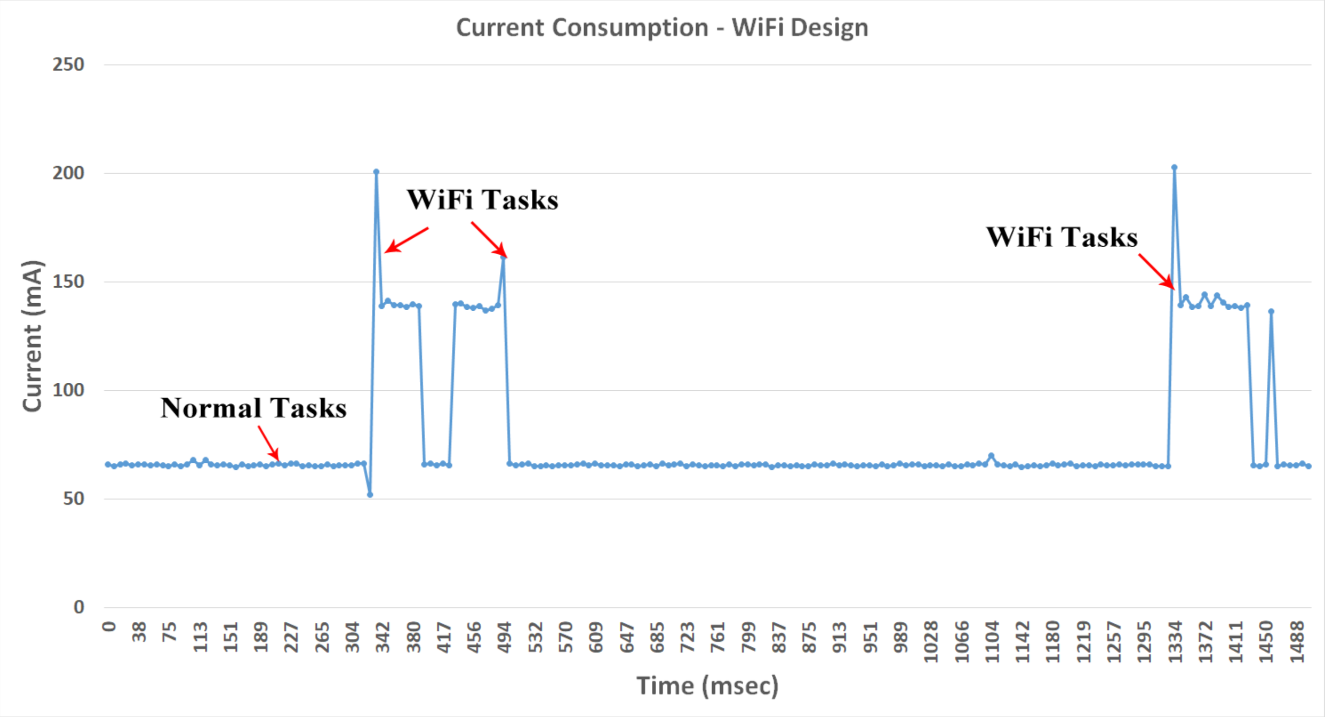


FIGURE 3.6: WiFi Current Consumption

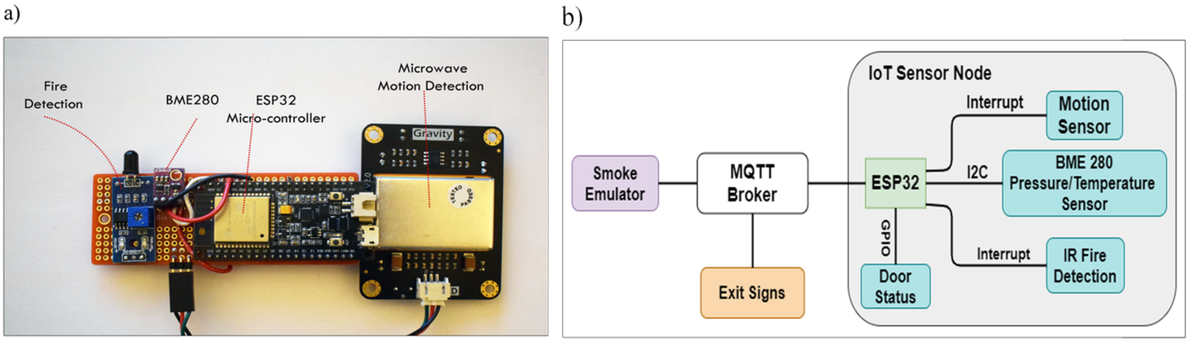


FIGURE 3.7: WiFi Node

seconds. When there is a fire, the interrupt pin will kick the RTC to wake up the ESP32. In this case, ESP32 operates with WiFi, it will connect to the access point and the MQTT broker, and it will start publishing the data every 250 milliseconds.

The normal operation consumes about 46.3 mA as shown in figure 3.9 and the cycle of operation is about 1.5 seconds.

Figure 3.9 shows the debugging serial monitor and the execution time that is taken per cycle. The difference between the measured cycle in figure 3.9 and the reported time in figure 3.10

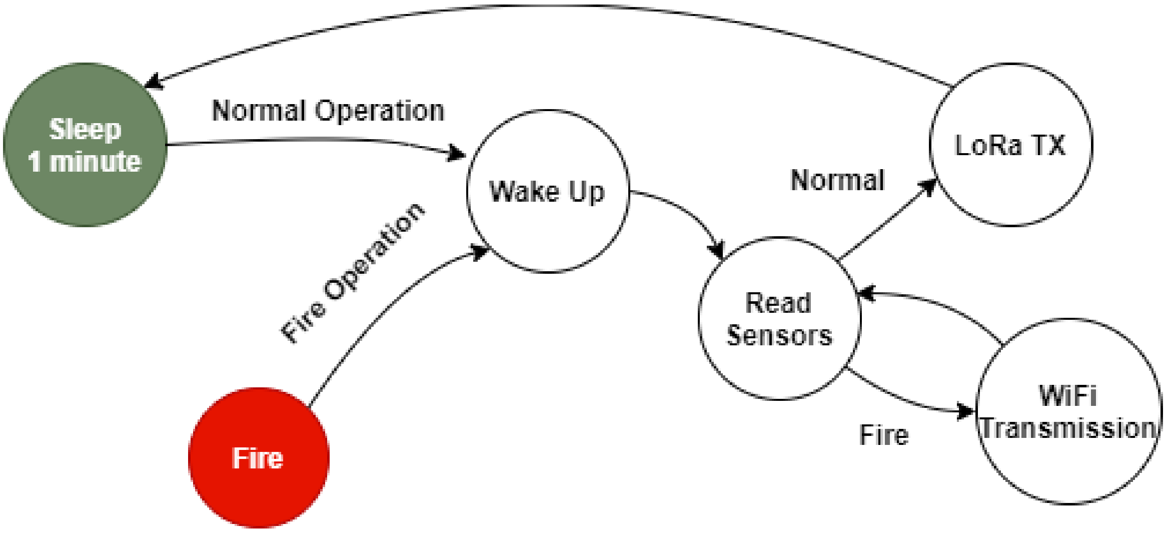


FIGURE 3.8: Low Power Design State Machine

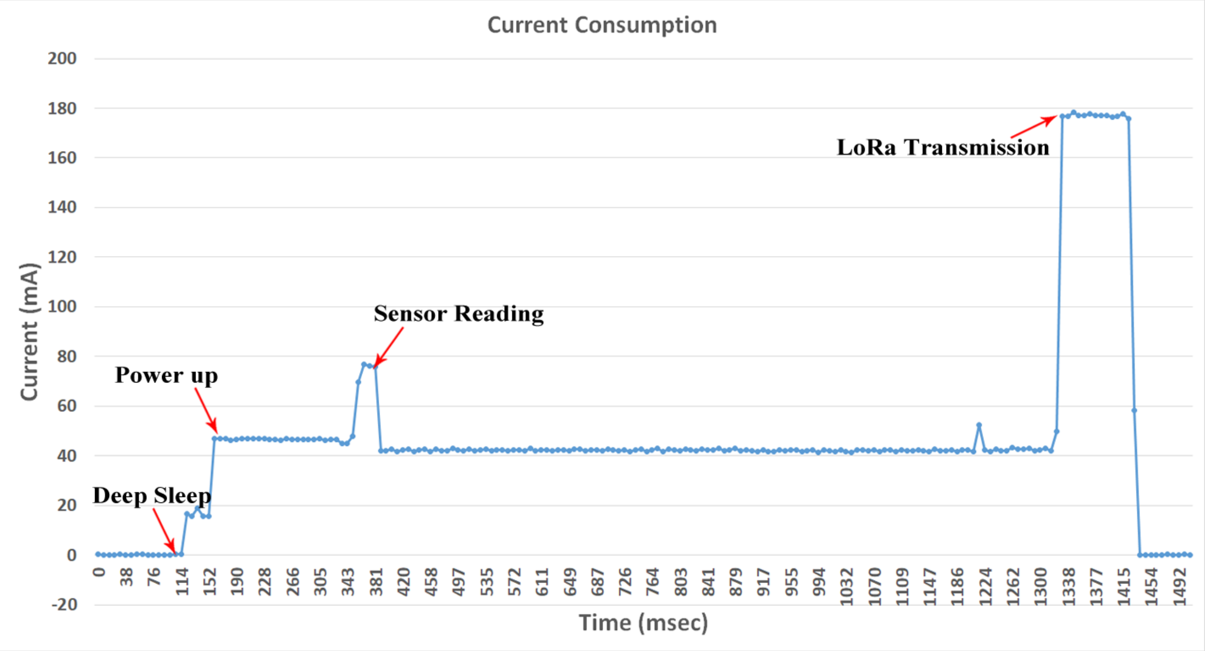
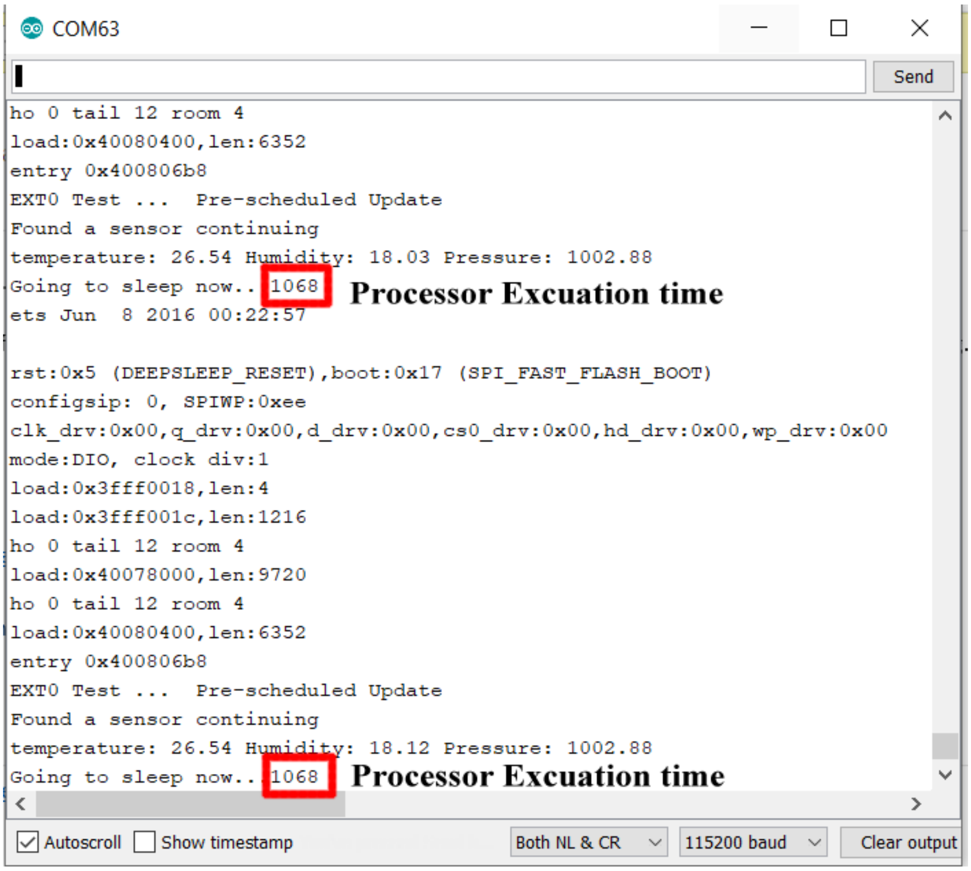


FIGURE 3.9: Current Consumption for Dual Communication Node

equals 436 millisecond, which is the transmission time between the sleep mode and operation mode. This latency is acceptable in the node operation to get the data. This implementation allows the node to operate on a 3000 mAh battery for 88 days with a reporting rate equals one report per minute.

The first response time in the fire scenario will be more as the nodes consumes more time to

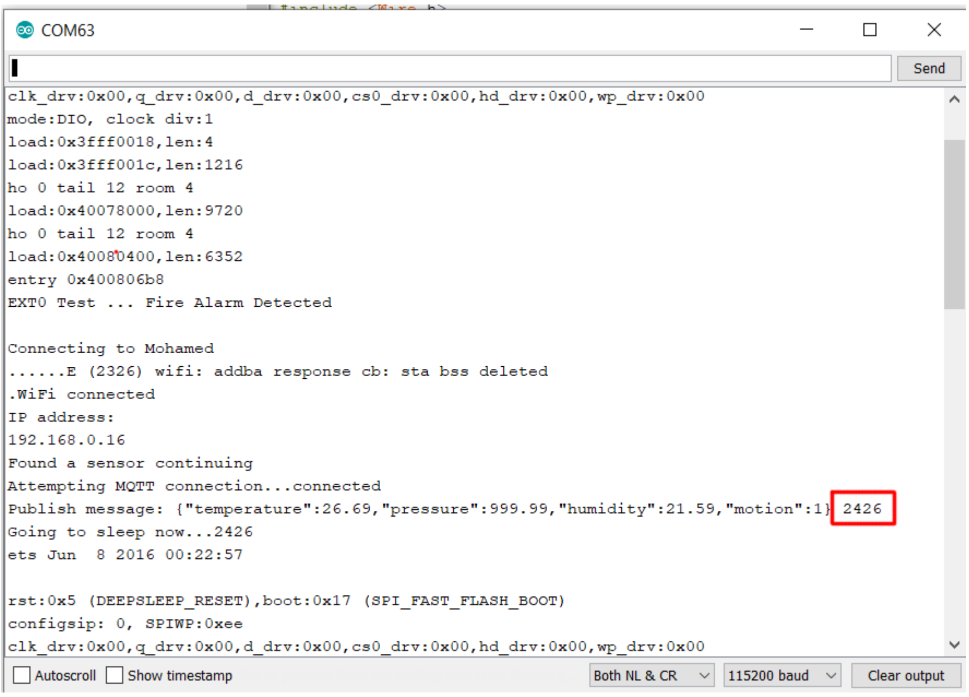


```
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
EXT0 Test ...  Pre-scheduled Update
Found a sensor continuing
temperature: 26.54 Humidity: 18.03 Pressure: 1002.88
Going to sleep now.. 1068
ets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
EXT0 Test ...  Pre-scheduled Update
Found a sensor continuing
temperature: 26.54 Humidity: 18.12 Pressure: 1002.88
Going to sleep now.. 1068
```

FIGURE 3.10: ESP32 Normal Operation Terminal

connect to the access point and the MQTT-Broker. Figure 3.11 shows the response time that the board takes from fire interrupt signal to the first payload transmission that equals 2.426 seconds, which is a reasonable time to get the very first fire alert. After getting the first response, the data latency will be around 5 ms.



```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
EXT0 Test ... Fire Alarm Detected

Connecting to Mohamed
......E (2326) wifi: addba response cb: sta bss deleted
.WiFi connected
IP address:
192.168.0.16
Found a sensor continuing
Attempting MQTT connection...connected
Publish message: {"temperature":26.69,"pressure":999.99,"humidity":21.59,"motion":1} 2426
Going to sleep now...2426
ets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
```

FIGURE 3.11: ESP32 Fire Operation Terminal

Chapter 4

Deep Learning for The Emulator

This chapter provides the usage of deep learning in the Emulator in order to cover the temporary data missing and to provide future prediction window.

4.1 Data Streaming over a Congestion Network

Data delivery is crucial to the emulator, as mentioned previously. The absence of the data caused by sensor damage, offline, resetting, or the data latency will cause the emulator to diverge over time. A developed algorithm evaluates the performance of MQTT latency over a congested network, as shown in figure 4.1. The algorithm creates different clients, which operates in parallel to transmit messages to simulate a busy system. The subscriber receives the messages and calculates the data latency based on the transmission and reception time, as shown in figure 4.2. The analysis of the received data shows that over a congestion network, the latency of the messages might reach to four seconds. To solve the latency problem, a Long Short Term Memory is used to provide a prediction window in order to have a consistent stream of data. This solution will be explained in more detail in the following sections.

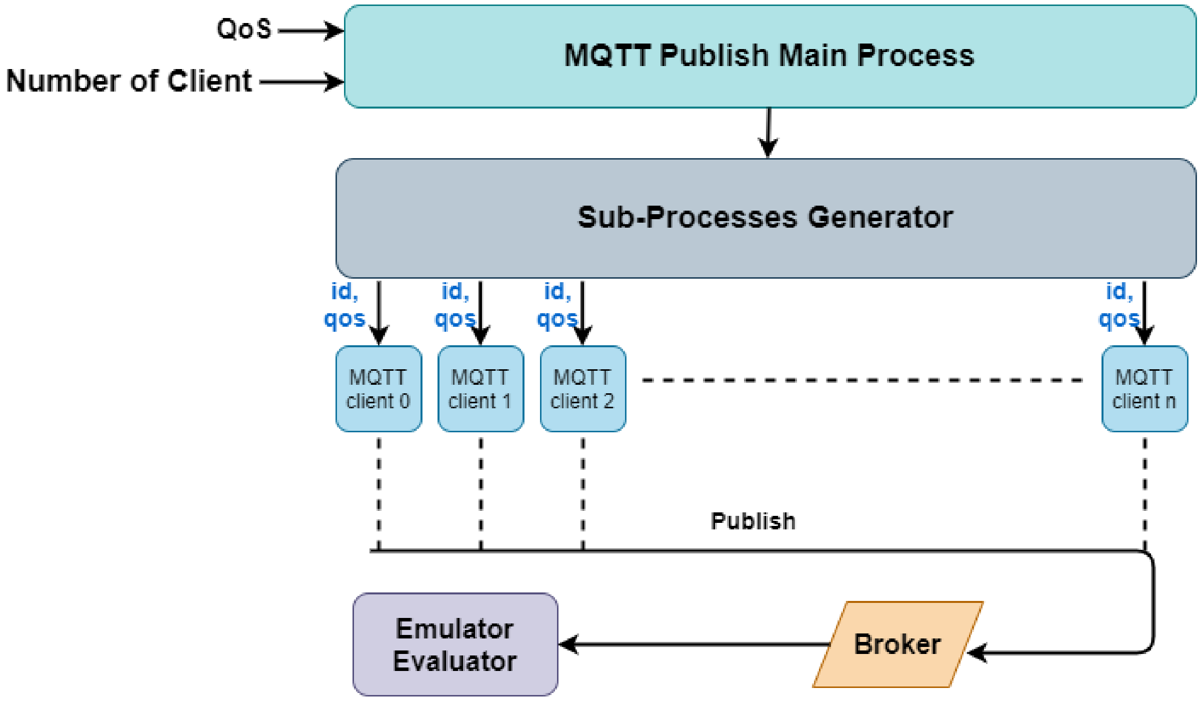


FIGURE 4.1: MQTT Evaluation Structure

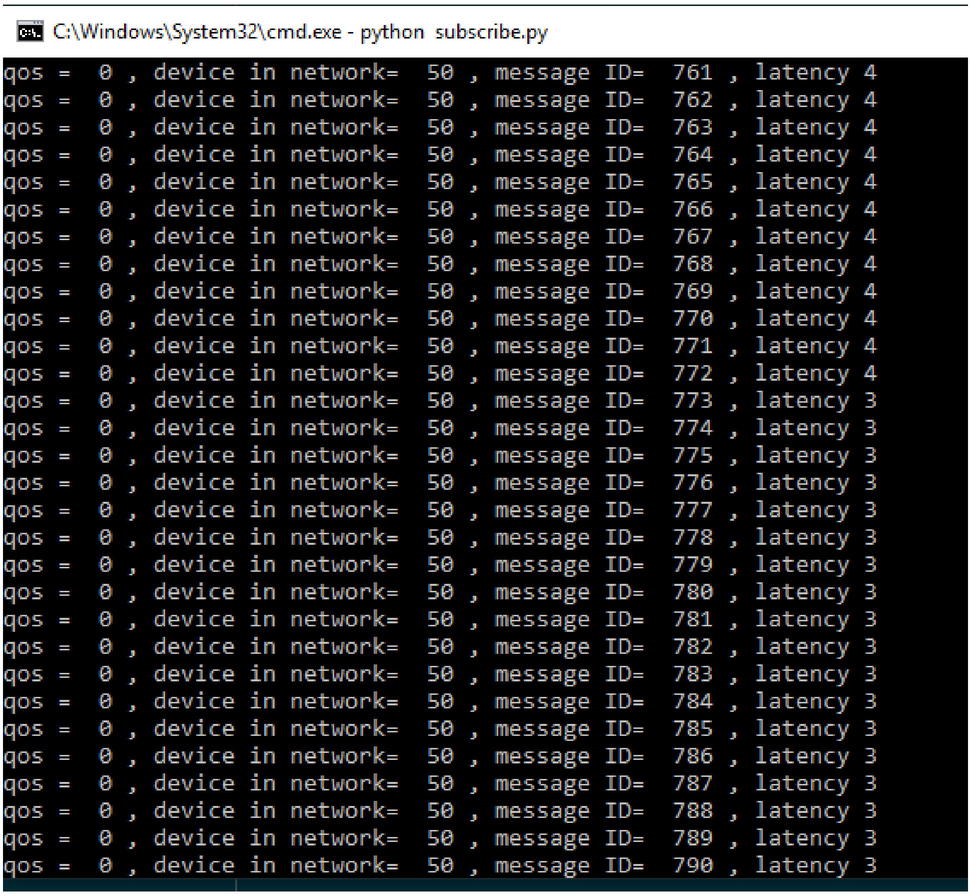
4.2 Artificial Neural Network

Artificial Neural Network (ANN) is inspired by the way of processing different types of information in the human brain [55]. The basic building block of ANN is the node which acts similar to the biological neuron. The node joins the weighted inputs together and applies an activation function on them. The weighted connections represent the strength of the connection between the nodes and the important of the features for desired output. The basic neuron can separate two simple classes as shown in figure 4.3. The neuron is represented as:

$$y = W \cdot X + b, \tag{4.1}$$

where y is the neural output, W is the weights of the inputs, X is the input vector, and b is the bias.

Different activation functions are represented in Fig. 4.4. These activation functions help process the data efficiently and add some non-linearity to the output. For instance, it helps in separating the non-linear boundary classes.



```
qos =  0 , device in network=  50 , message ID=  761 , latency 4
qos =  0 , device in network=  50 , message ID=  762 , latency 4
qos =  0 , device in network=  50 , message ID=  763 , latency 4
qos =  0 , device in network=  50 , message ID=  764 , latency 4
qos =  0 , device in network=  50 , message ID=  765 , latency 4
qos =  0 , device in network=  50 , message ID=  766 , latency 4
qos =  0 , device in network=  50 , message ID=  767 , latency 4
qos =  0 , device in network=  50 , message ID=  768 , latency 4
qos =  0 , device in network=  50 , message ID=  769 , latency 4
qos =  0 , device in network=  50 , message ID=  770 , latency 4
qos =  0 , device in network=  50 , message ID=  771 , latency 4
qos =  0 , device in network=  50 , message ID=  772 , latency 4
qos =  0 , device in network=  50 , message ID=  773 , latency 3
qos =  0 , device in network=  50 , message ID=  774 , latency 3
qos =  0 , device in network=  50 , message ID=  775 , latency 3
qos =  0 , device in network=  50 , message ID=  776 , latency 3
qos =  0 , device in network=  50 , message ID=  777 , latency 3
qos =  0 , device in network=  50 , message ID=  778 , latency 3
qos =  0 , device in network=  50 , message ID=  779 , latency 3
qos =  0 , device in network=  50 , message ID=  780 , latency 3
qos =  0 , device in network=  50 , message ID=  781 , latency 3
qos =  0 , device in network=  50 , message ID=  782 , latency 3
qos =  0 , device in network=  50 , message ID=  783 , latency 3
qos =  0 , device in network=  50 , message ID=  784 , latency 3
qos =  0 , device in network=  50 , message ID=  785 , latency 3
qos =  0 , device in network=  50 , message ID=  786 , latency 3
qos =  0 , device in network=  50 , message ID=  787 , latency 3
qos =  0 , device in network=  50 , message ID=  788 , latency 3
qos =  0 , device in network=  50 , message ID=  789 , latency 3
qos =  0 , device in network=  50 , message ID=  790 , latency 3
```

FIGURE 4.2: MQTT Latency in a congested network

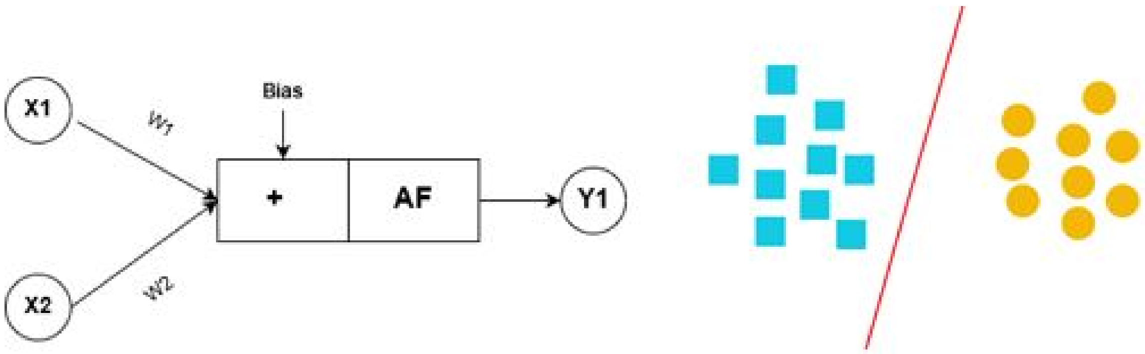


FIGURE 4.3: Basic Neuron

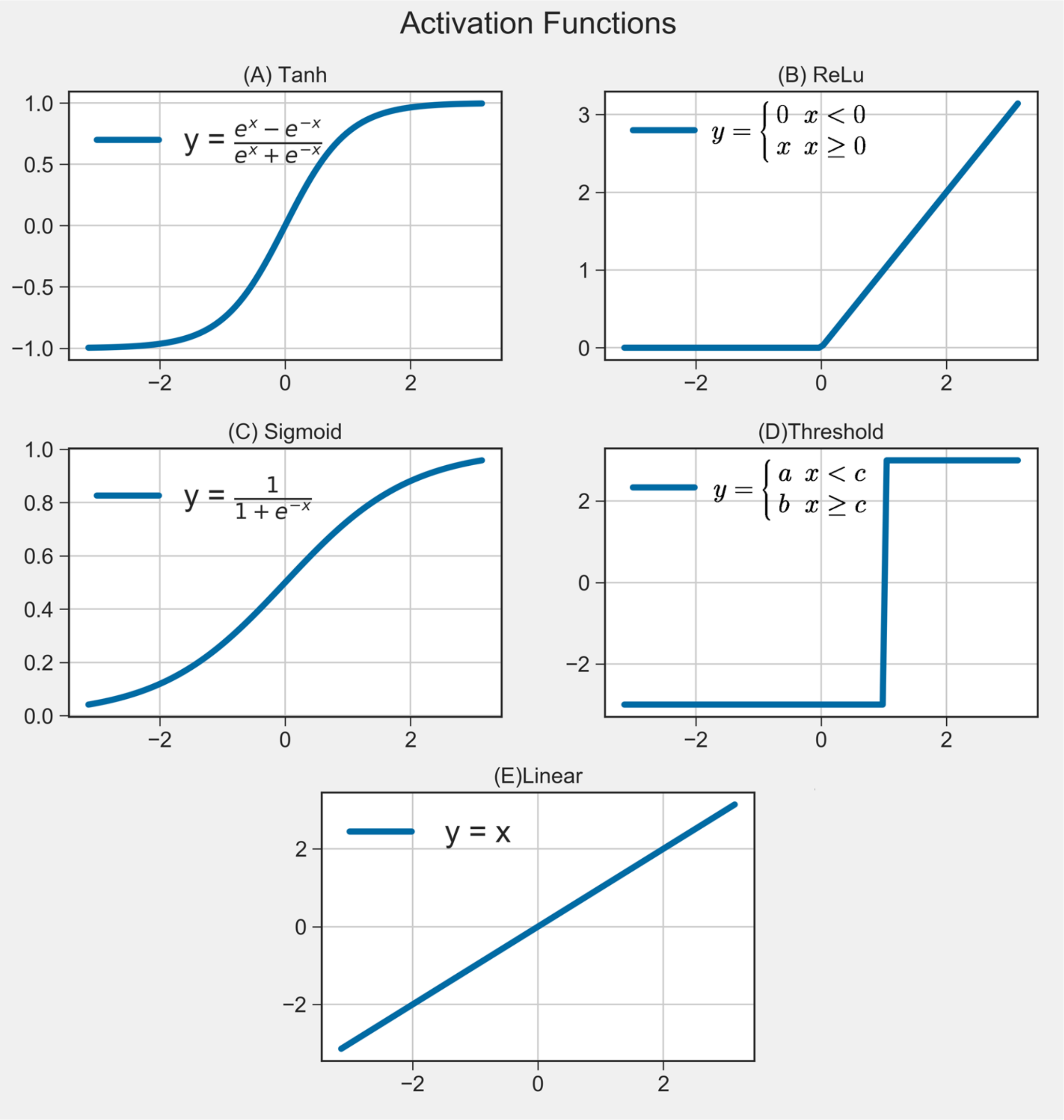


FIGURE 4.4: Neural Network Activation Functions

ANN has different forms such as feed-forward networks and feedback networks. The feedback network, also known as recurrent neural network RNN, will be explained later.

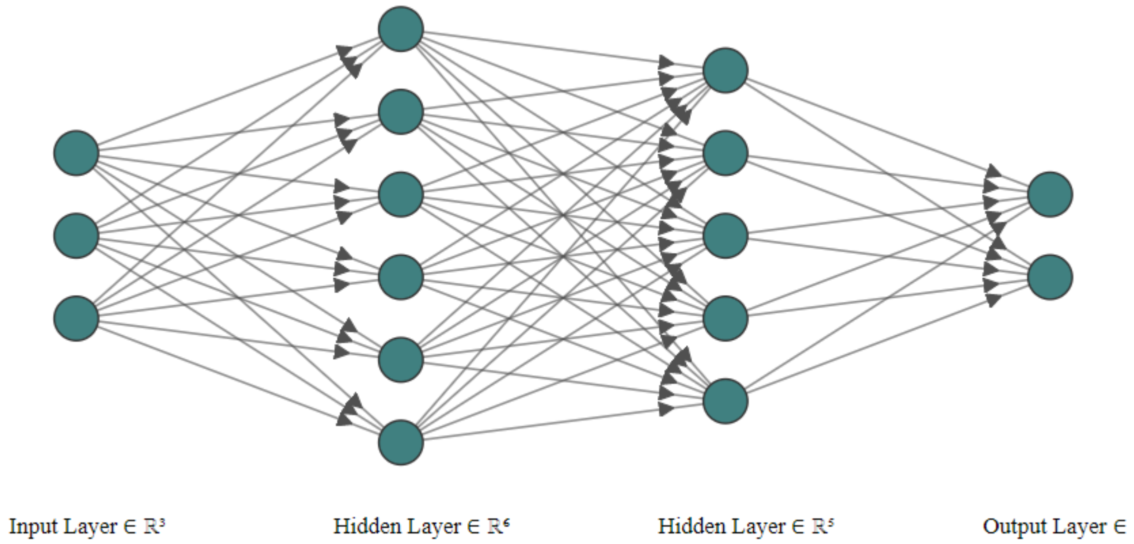

Input Layer $\in \mathbb{R}^3$
Hidden Layer $\in \mathbb{R}^6$
Hidden Layer $\in \mathbb{R}^5$
Output Layer $\in$

FIGURE 4.5: Neural Network Architecture

4.3 Neural Network Structure

In order to give the ANN, the ability to solve a complicated problem, we need to use a multi-layer perceptron (MLP). The nodes can be arranged in stacked layers that form the overall ANN architecture as shown in figure 4.5.

There are three types of layers:

- the input layer holds the input independent data features;
- the hidden layers work on combing the input features and try to find the pattern in the data.
- the output layer provides the output according to the task and number of classes.

4.4 Network Training

After designing the ANN, the weights of the nodes need to be adjusted to achieve their task. The process of getting the new weights is called training and it starts by splitting the data into training data-set and testing data-set, which will be used later to evaluate the model performance. Random weights will initiate the model, and the training data-set will be used to calculate the error function,

$$F(e) = y - \hat{Y} \tag{4.2}$$

where e is the error, y is the actual output, Ŷ is the predicted output.

The goal of the training process is to minimize the error function by finding the global minimum, which can be achieved using gradient descent. The gradient descent is an optimization algorithm that calculates the maximum gradient of each weight to the error function and propagates in the other direction to get the minimum error. The back-propagation algorithm is using gradient descent to optimize the weights as

$$w_i := w_i + 2.\alpha.\frac{\partial f_e}{\partial w_i}, \tag{4.3}$$

where w_{i} is the weight to optimize, α is the learning rate. The error function can be calculated for all data record, which is known as gradient descent. It has a single update step that is calculated after one cycle of the entire training data-set, which makes it a very slow algorithm [56]. The solution to this problem is using stochastic gradient descent [57], where the error is calculated for each data point in the training set. This will update the weights after every training data point. Batch gradient descent is another technique that calculates the error after certain amount of training points then it updates the weights.

4.5 Network Over-fitting

One of the neural network problems is data over-fitting as shown in figure 4.6-a, where the network is fitting only to a set of input data. Dropout [58] is one of the regularization techniques that are used to solve this problem as it forces all the weights to update in the training process. This method is based on randomly deactivating some neurons as shown in figure 4.6-b in the training steps to give the network the ability to recognize more features and relations in the input data.

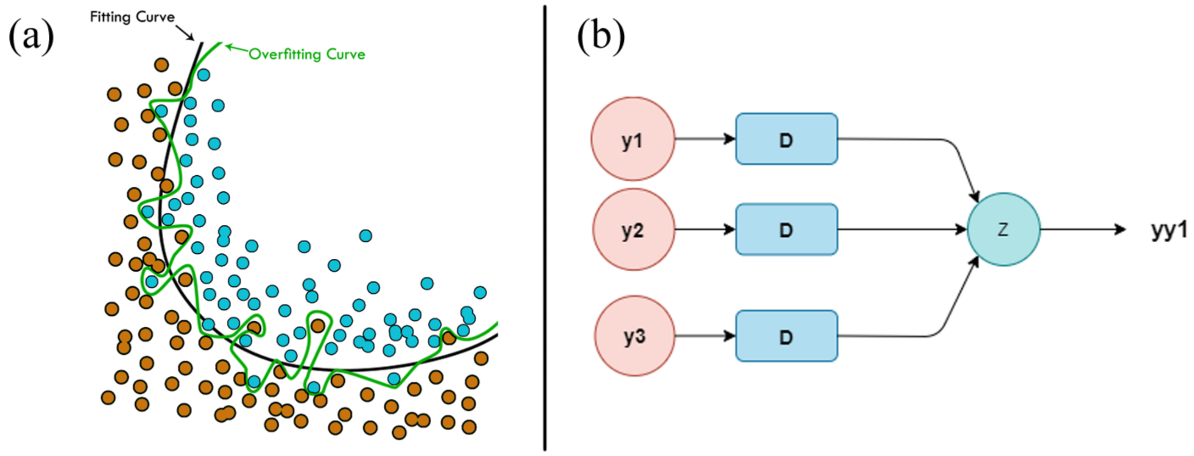


FIGURE 4.6: Over-fitting problem and solution

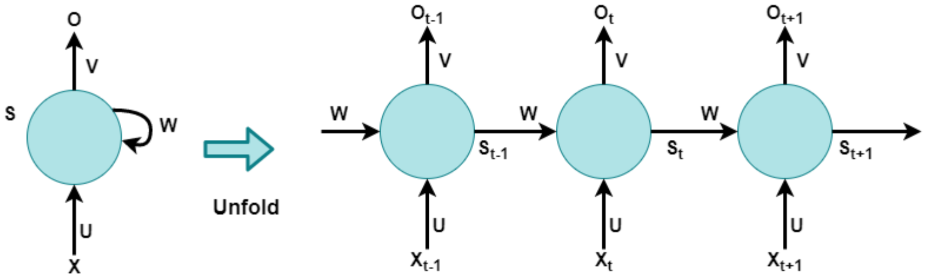


FIGURE 4.7: Recurrent neural network architecture

4.6 Recurrent Neural Network

A feed-forward neural network works to classify objects using the input data at the current moment only. In our system, we need to predict the future data [59] from the historical data in order to keep the emulator stable. This solution helps in case the data are temporary missed, which requires the neural network to have a feedback to predict the next value. This can be achieved using Recurrent Neural Network (RNN). Figure 4.7 shows the main structure of the RNN, where the training process is used to find the values of V, W, and U, which are shared in the entire network. S represents the hidden state that is calculated using non-linear function such as Tanh and ReLu and it depends on the current input and the last hidden state. The hidden state represents the historical information of the data and can be calculated as,

$$S_t = f(W \cdot S_{t-1}, X_t \cdot U), \tag{4.4}$$

where S_{t} denotes the current hidden state of the cell,W and U denote the weights and X_{t} denotes the cell input. The output can be written as,

$$O_t = V \cdot S_t, \tag{4.5}$$

where O_{t} denotes the current output and V denotes the weight.

The non-linear functions make the RNN have a shorter memory that is known as a vanishing gradient[60], which occurs by multiplying small weights of the layers together. The non-linear functions contribute to this problem by squeezing the input data to smaller values between -1 and 1 when using tanh and 0 and 1 when using sigmoid. As a result of the vanishing gradient, the network will have a shallow memory and can not use old data efficiently.

4.7 Long Short Term Memory

To solve the vanishing gradient, a new cell structure of Long Short Term Memory (LSTM) [61] is introduced, as shown in figure 4.8.

LSTM has four main components forget gate, input gate, cell state, and output gate. The new input and the old hidden state are combined to form the combined input. The forget gate applies Sigmoid function on the combined input to decide to use the cell state line or not. It has output between 0 and 1 where 0 means blocking the old cell state and 1 means allowing all the cell state to go through the line. The input gate applies Sigmoid and Tanh function on the combined input and uses the output to update the cell state. The cell state is updated by the

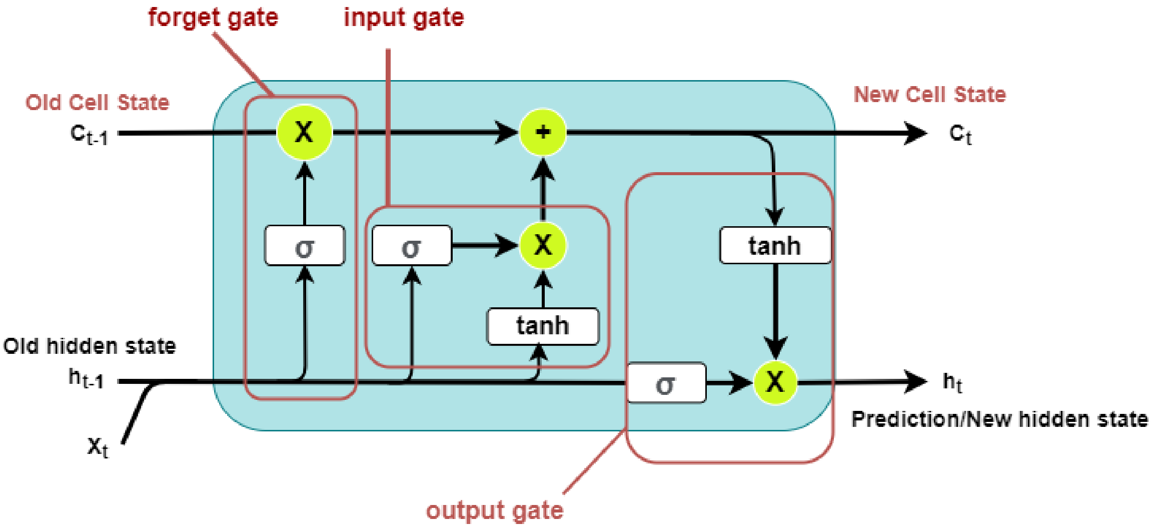


FIGURE 4.8: Recurrent Neural network Architecture

output of the input gate and the forget gate. The output gate uses the combined input with the new cell state to predict the pressure in our case which is also the hidden state for the next time step. This design gives the network the ability to use older sequential data.

4.8 Data Pre-Processing and Training

The data pre-processing is a crucial step to have a good deep learning model. There are many techniques that can be followed such as data normalization. Data normalization scales each data input category from 0 to 1, thus avoiding a biased model to certain features. For the proposed LSTM network, the input data includes 50 readings for the current time and the past 49 each reading have two value, the pressure and the temperature. Keras library [62] is used to configure and train the LSTM as shown in list 4.1 and figure 4.9. It includes 3 input layers each of them has dropout equals 0.2 and dense layer at the end to output the next predicted value. It uses mean square error [63] to evaluate the model in the training process. The data are splited into training data-set with 70% of the data and 30% for the testing. The evaluation of the model is done for case 1 on the testing data-set and the entire output of FDS for case 2 in order to validate that the model does not overfit to a certrain case .

```
    def configure_model(self,):
```

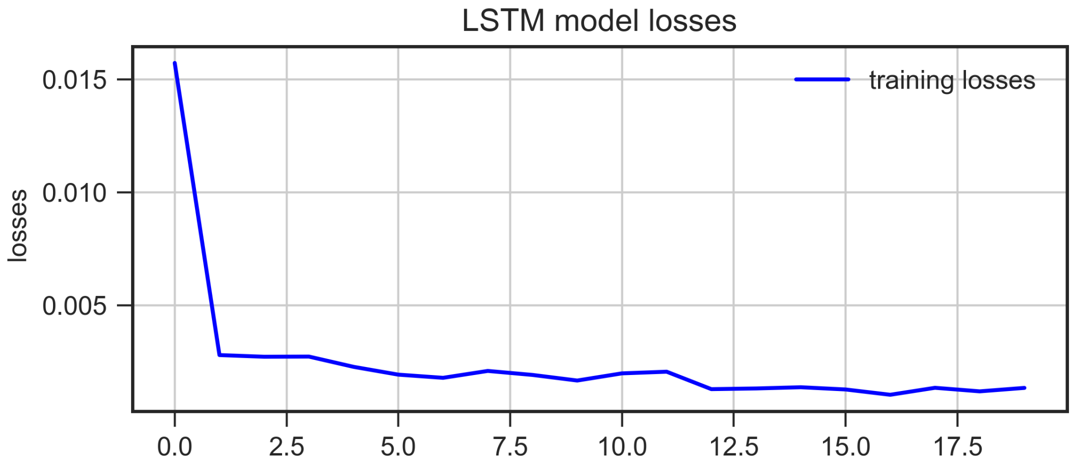


FIGURE 4.9: LSTM training losses

```
self.model.add(LSTM(units=50, return_sequences= True,\\
input_shape=(self.X.shape[1],2)))
self.model.add(Dropout(.2))
for i in range(2):
    self.model.add(LSTM(units=50, return_sequences=True))
    self.model.add(Dropout(.2))

self.model.add(LSTM(units=50))
self.model.add(Dropout(.2))


self.model.add(Dense(units=1))
self.model.summary()
self.model.compile(optimizer=’adam’, loss=’mean_squared_error’)
```

LISTING 4.1: LSTM model configuration

4.9 Data Prediction

In the fire environment, the data is critical to update the emulator; missing data caused by the high data traffic or the noisy environment might cause trouble for the emulator. LSTM network predicts the pressure in order to have a stable data stream. The pressure plays an essential rule in moving the air and smoke inside the building. In order to get the training dataset, the building model was simulated on FDS software, and its output data were used for the training and testing.

The LSTM uses multi-variant input data that contains the temperature and pressure of the zone to predict the upcoming pressure. In order to evaluate the models, the Mean Absolute Percentage Error (MAPE) is calculated for all the models as,

$$\text{MAPE} = \frac{1}{N} \cdot \sum_{n=1}^{N} \left| \frac{\text{Original Reading} - \text{Predicted Reading}}{\text{Original Reading}} \right| \tag{4.6}$$

The prediction buffer in Fig. 4.10 receives the sensors' readings from the hardware nodes over MQTT. This data fill the LSTM buffer first when the buffer has the last 50 readings. It can predict the upcoming pressure values in case there is no sudden event. If the data is still available, the last reading will be added to the buffer, and the oldest value will be removed. When the data is missed, the model will predict the next data and will send it to the correction layer and buffer. Different architecture had been tested with different hyper-parameters such as number of LSTM layers, sequence length, epochs, batch sizes and different missing data rate the performance of these models appears in table 4.1.

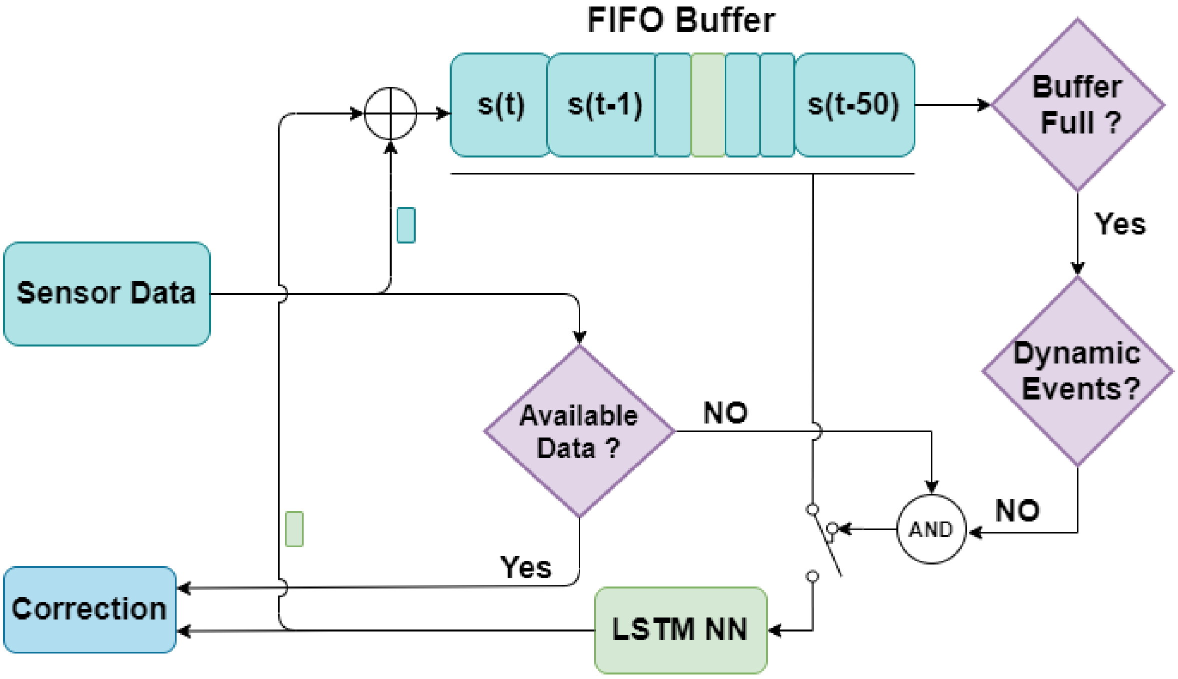


FIGURE 4.10: Data Tracking and Prediction

TABLE 4.1: LSTM different hyper-parameters results

Model ID	LSTM layers	Batch size	Epochs	Input size	Model Error	missing Samples	Error at missing data
1	2	5	20	50	2.54	50 100 150	2.8 3.16 3.58
2	2	5	100	50	2.05	50 100 150	2.44 3.04 4.01
3	2	5	100	20	1.75	50 100 150	2.07 2.5 3.17
4	2	5	200	80	1.55	50 100 150	1.84 2.24 2.8
5	2	15	50	20	3.06	50 100 150	3.38 3.78 4.34
6	2	20	100	20	2.57	50 100 150	2.84 3.17 3.6
7	3	5	100	50	.96	50 100 150	1.16 1.42 1.76
8	3	15	50	50	6.53	50 100 150	7.17 8 9.02
9	5	15	50	50	4.2	50 100 150	4.4 4.73 5.14
10	5	15	200	50	1.89	50 100 150	2.15 2.46 2.94

Model 7 is selected as it has the best performance and the lowest error on the testing data and when the data are temporarily missed as shown in figure 4.11. Figure 4.12 shows the prediction on case 1 even though the model is not trained in this case, it can predict the pressure with low error. This indicates that the LSTM network can perform well on different layouts without new training.

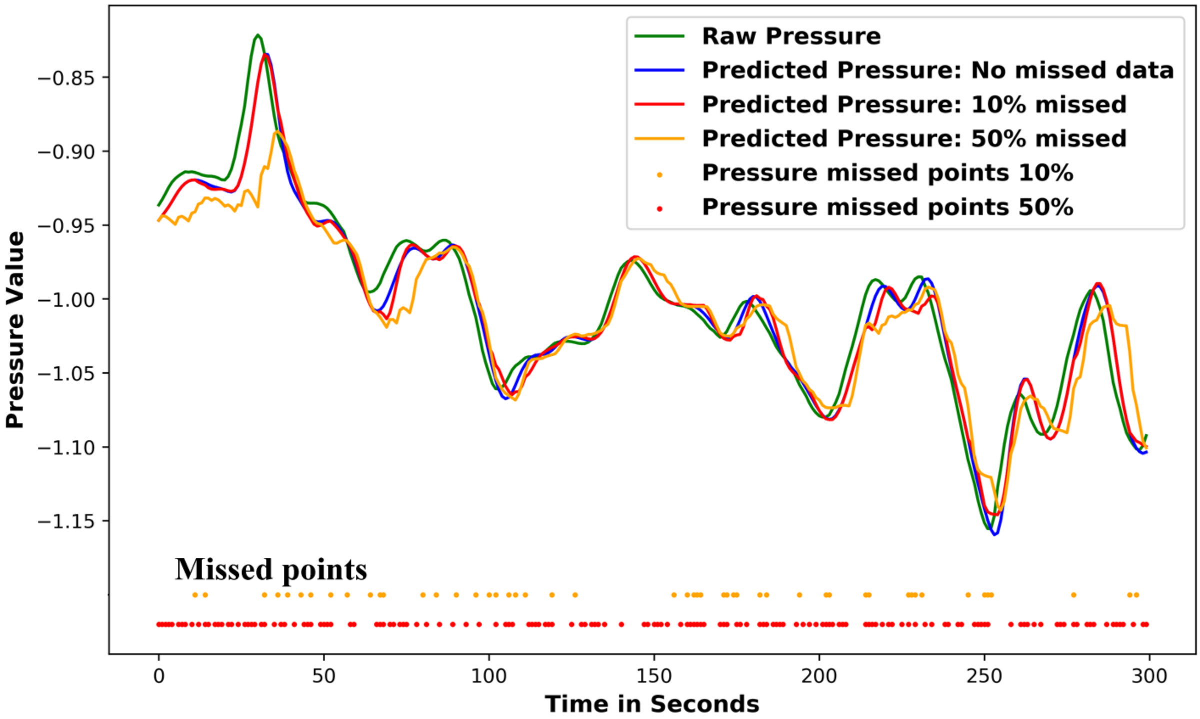


FIGURE 4.11: Recurrent Neural network Architecture

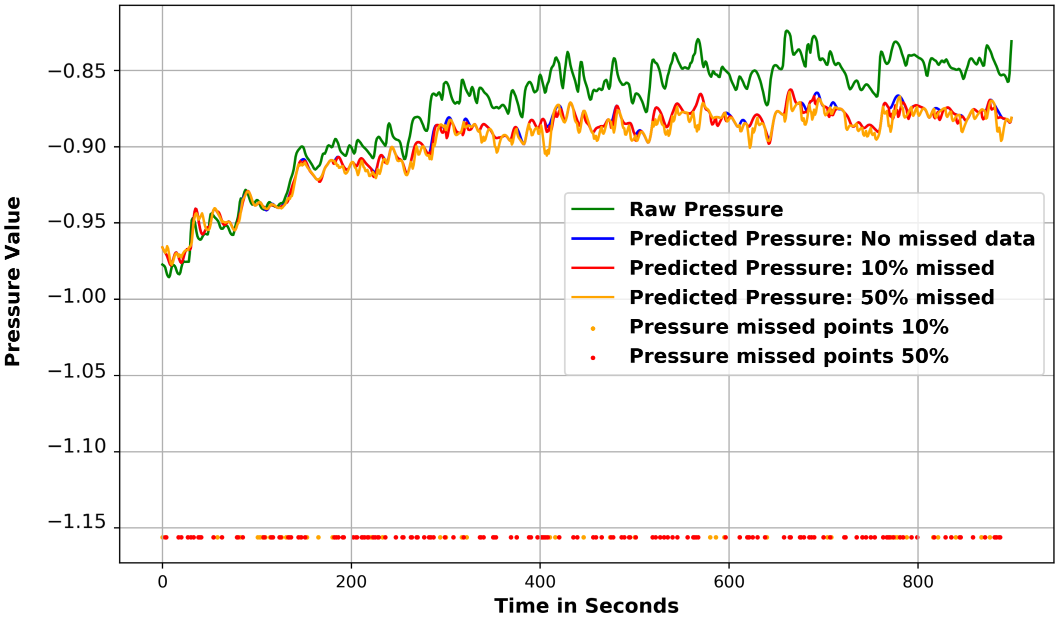


FIGURE 4.12: Recurrent Neural network Architecture

Chapter 5

Path Planning

Path planning plays an important role in evacuation and assisting firefighters to evacuate entrapped occupants[64]. Our proposed emulator uses a nested searching space that can be seen as two integrated processes, global level search and local level search. The former handles the high level details for evacuation routes such as which rooms and doors form the route. The latter is enabled to integrate with the former level search to provide precise information about the evacuation routes such as the route's turns in a small corridor. This is most useful to prioritize evacuations for people with disabilities, fire fighters navigating the building, or rescue robots through means of position tracking[65]. This method is inspired by how humans solve relatively simple mazes. If a person glances over a simple maze, they can find a path from start to goal without details about how it turns corners or which wall side should be traced for the shortest way out. They can see that an abstract connection of empty spaces from start to goal exists and further detailed considerations could be taken afterwards. Different search algorithms can be used for each level separately, but to compare results with 1 level search, A* search algorithm[40] is used for both the global and local levels.

5.1 Global Level

The connections between nodes that were created from the approximate cell decomposition are based on a connection check that ensures there's no rigid bodies separating the spaces they represent. This level consists of irregularly connected nodes that form a graph search problem, removing any spatial correlation between nodes.

This also is the most abstract representation of the real space with the least number of nodes making the search problem much easier. A global level path determines which doors and rooms to use.

Instead of using an extension of the A* algorithm or its original heuristic[40], a unique heuristic suitable for the problem definition at hand and the discretization technique is developed. The heuristic function used for the A* algorithm on the global space is

$$H_p^{(Global)} = D_p^{(Global)} + k_G \sum_{i=1}^{p} \sum_{n=0}^{N} \gamma^n \left(C_i^{t_0+n} + T_{i-1,i}^{t_0+n}|_{i \neq 1} \right), \tag{5.1}$$

where $H_p^{(Global)}$ denotes the p_{th} node's heuristic, $D_p^{(Global)}$ denotes its global level distance, k_G denotes the global cost hyper-parameter, N denotes the size of future simulation window for cost, γ denotes the cost discount hyper-parameter, $C_i^{t_0+n}$ denotes the global cost of the i^{th} node at $t_0 + n$, $T_{i-1,i}^{t_0+n}$ denotes the transition cost from $i^{th} - 1$ to i^{th} node at $t_0 + n$, and t_0 denotes the current real time.

The global level distance of an arbitrary node is defined as the smallest number of connections from that node to the goal node, whereas the cost of a node is the smoke concentration in that node's entity relative to the space it represents so that nodes representing larger spaces are not penalized more. A transition cost term is added to represent the transition between two doors, which includes the smoke in the calculations.

The coefficient of this weighted sum and the global cost hyper-parameter k_G are used to tune the global search algorithm, enabling it to search for the least number of nodes connecting start to goal, with the lowest percentage of smoke. The cost term of the heuristic elevates the fact that the smoke simulator is faster than real time and takes the simulated future cost of the nodes to stay away from smoke propagation in the future. The impact of the simulated cost of future time steps is controlled by the hyper-parameter γ which takes a value larger than 0 and smaller than 1. The impact of future simulation is included by raising γ to the power of n, where n is the order of the simulated step in the future. Setting N to 0 will disregard any simulated future cost information and the heuristic will use the cost of the current simulated time step only.

For each of the nodes corresponding to rooms, a global search process continuously searches for a global evacuation route, namely the Prime Evacuation Route. All the exit and emergency signs on this route are controlled to indicate the updated direction towards safety so that it is possible to achieve a complete evacuation solution for the entire building.

5.2 Local Level

At the local level, the search algorithm chosen will traverse a regularly spaced discretized space consisting of fixed size cells with spatial connections to their neighbors. The search is enhanced by refining the allowed pool of cells to search, the frontier, by preventing previously checked cells from being checked again, then the heuristic for the cells in the pool is calculated from

$$H_p^{(Local)} = \frac{D_p^{(Local)} + k_L \sum_{i=1}^{p} \sum_{n=0}^{N} \gamma^n \, C_i^{t_0+n}}{1 + k_{len} \, Q}, \tag{5.2}$$

where $H_p^{(Local)}$ denotes the p_{th} cell's heuristic, $D_p^{(Local)}$ denotes its local level distance, k_L denotes the local cost hyper-parameter, N denotes the size of future simulation window for cost, γ denotes the cost discount hyper-parameter, $C_i^{t_0+n}$ denotes the local cost of the i^{th} cell at t_0+n, k_{len} denotes the local path length hyper-parameter, Q denotes the number of cells in the path from the starting cell to the current one, and t_0 denotes the current simulation time. The local level cost of a cell is the smoke saturation in the cell, whereas the local level distance is simply the euclidean distance to the goal cell. The cost term of the heuristic uses the hyper-parameter γ with a value larger than 0 and less than 1, similar to the global level search heuristic, to control the impact of simulated future cost of cells.

The k_{len} hyper-parameter, when tuned properly, does not make the search deviate from finding the shortest path with the lowest cost, but allows it to use longer paths, in case it searches for a short path to goal without finding it, for an amount of time before using different cells to lead to other directions from frontier. In most cases setting k_{len} to 0 is sufficient to obtain viable paths but in some extreme cases in which the path needs to take wide turns to goal, setting k_{len} to a positive value smaller than 0.01 will speed up the search.

The ability to perform a search on the local level allows the emulator to give evacuation paths that go around concentrations of smoke in an area such as a room. Even though local search processes can be independent of global search processes, integrating the two yields better and faster results. For this reason, any local search problem is translated to a global search then delegated back to the local level. After finding a global level path, the positions of each two subsequent nodes, each segment of that path, in the local level will serve as a start and a goal for one of many sub-paths in the local level. Fig. 5.1 shows how the integrated 2 levels search is much superior to the 1 level search in terms of adaptability and number of iterations as the integrated search manages to find an evacuation route faster than the 1 level search. It looks for a safer evacuation route at the time the 1 level search finds its first, and not the safest, evacuation route. Although the nodes in the global level do not carry any spatial information, they have information about their initial position at the local level to serve as a medium for transforming the search problem between the two levels back and forth.

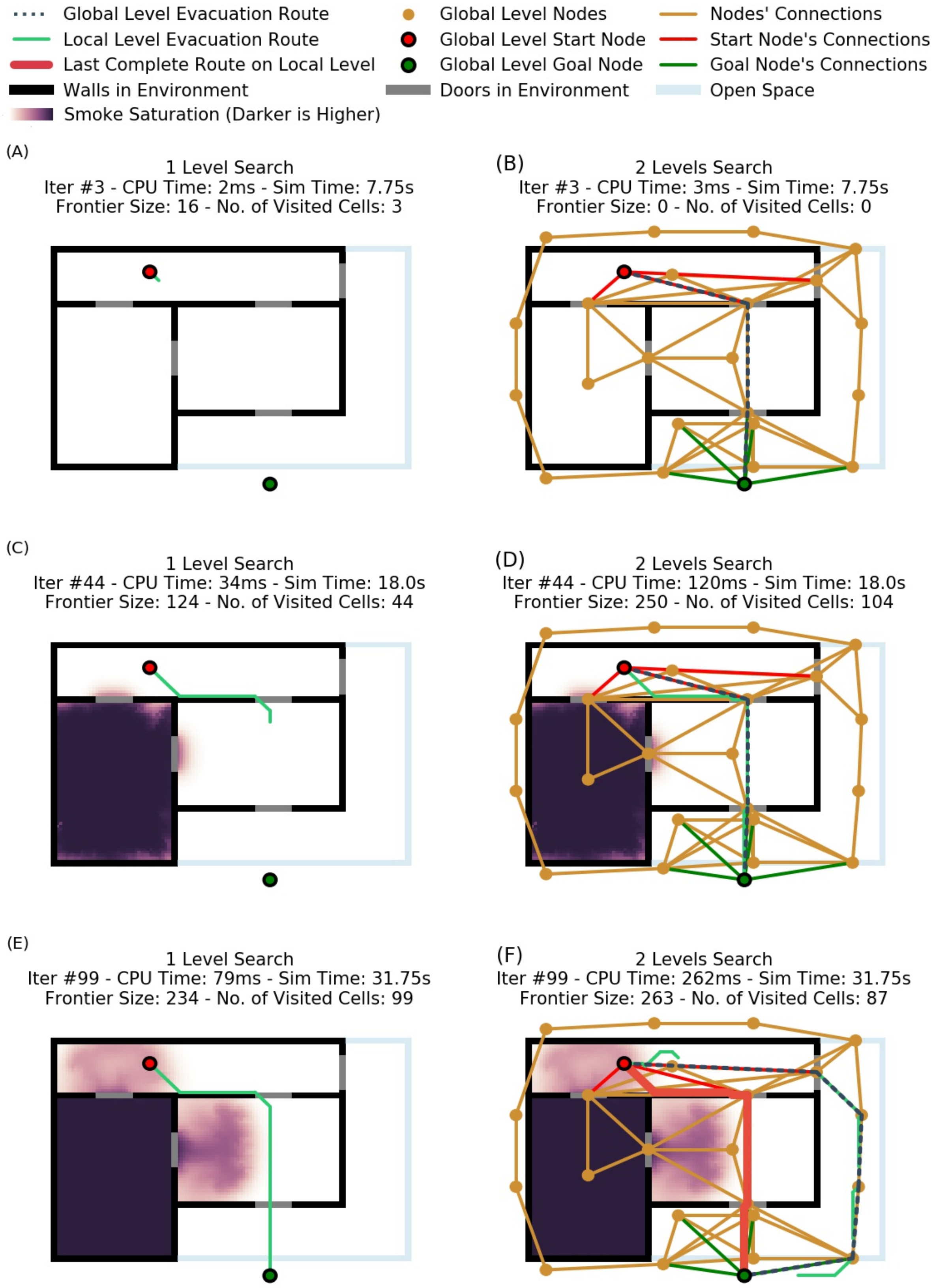

Global Level Evacuation Route
Local Level Evacuation Route
Last Complete Route on Local Level
Walls in Environment
Smoke Saturation (Darker is Higher)
Global Level Nodes
Global Level Start Node
Global Level Goal Node
Doors in Environment
Nodes' Connections
Start Node's Connections
Goal Node's Connections
Open Space
(A)
1 Level Search
Iter #3 - CPU Time: 2ms - Sim Time: 7.75s
Frontier Size: 16 - No. of Visited Cells: 3
(B)
2 Levels Search
Iter #3 - CPU Time: 3ms - Sim Time: 7.75s
Frontier Size: 0 - No. of Visited Cells: 0
(C)
1 Level Search
Iter #44 - CPU Time: 34ms - Sim Time: 18.0s
Frontier Size: 124 - No. of Visited Cells: 44
(D)
2 Levels Search
Iter #44 - CPU Time: 120ms - Sim Time: 18.0s
Frontier Size: 250 - No. of Visited Cells: 104
(E)
1 Level Search
Iter #99 - CPU Time: 79ms - Sim Time: 31.75s
Frontier Size: 234 - No. of Visited Cells: 99
(F)
2 Levels Search
Iter #99 - CPU Time: 262ms - Sim Time: 31.75s
Frontier Size: 263 - No. of Visited Cells: 87

FIGURE 5.1: Integrated 2 Levels Search and 1 Level Search Comparison

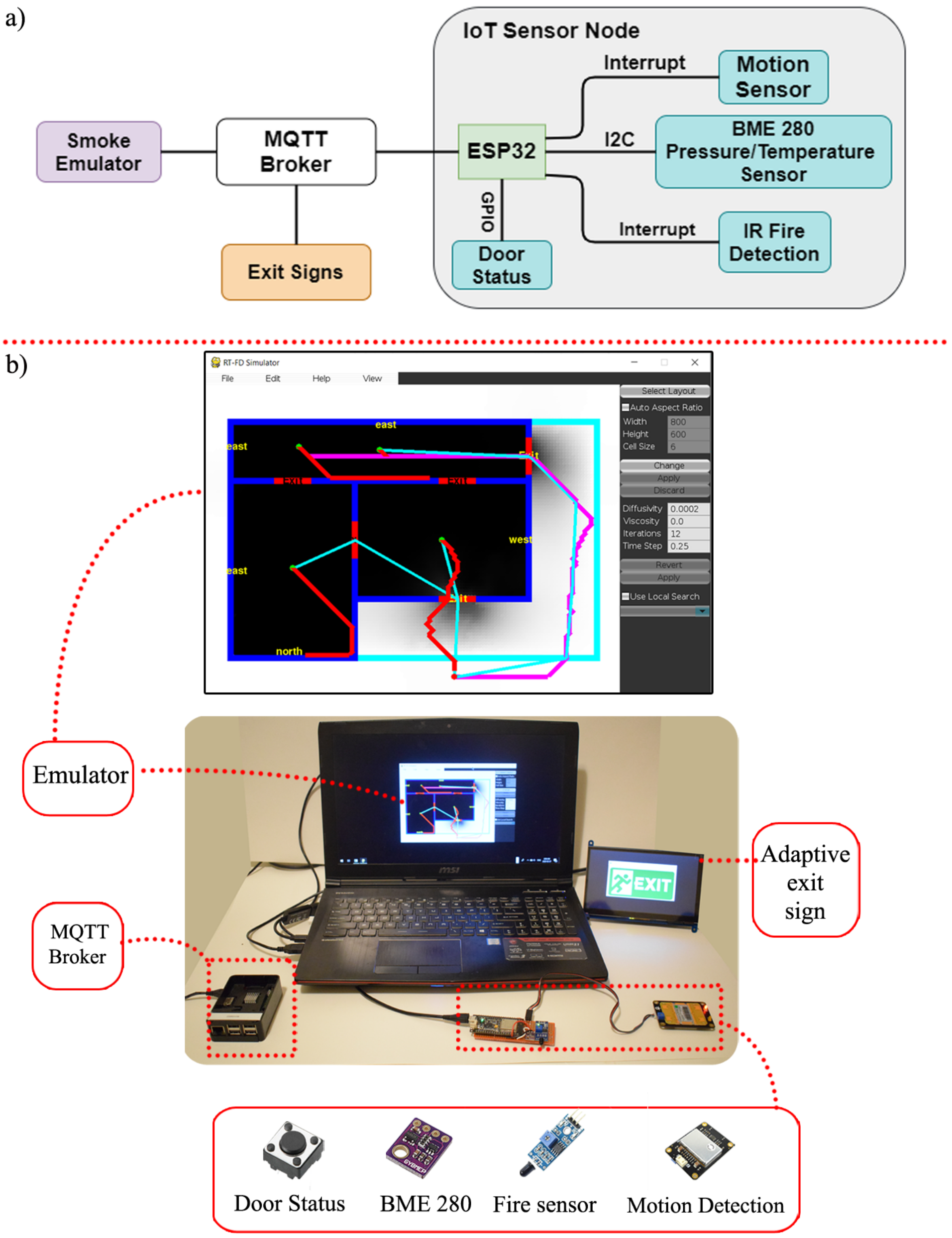
Chapter 6

Results and Discussion

6.1 Correction Layer

The deployed IoT nodes in the building are capable of monitoring the space parameters, which will help make a digital twin with the emulator environment. Each of the nodes has a pressure sensor, a temperature sensor, a door status sensor, a fire detection[66] sensor, and motion sensor seen in Fig. [6.1]. The correction layer provides the emulator with momentary feedback, that will be used by many components in the emulator. It also has a display that acts as a dynamic exit sign to visualize the best exit route for the occupant. The pressure and temperature data will correct the emulator's results from deviations and make the emulator runs similar to the ground-truth. The door status reading interacts with the rigid bodies layer that changes the door's status to control the smoke flow through the door. The motion sensor will allow the firefighters to detect the occupants' location faster to minimize the searching and evacuation time. The routing algorithm controls the exit signs in order to provide the best adaptive exit route, to minimize smoke exposure and evacuation time[64]. The current implementation uses the hybrid ESP32 design that collects the data and control the exit signs.

After integrating the above-mentioned layers, the solver can simulate key behaviors of fires in buildings, but to make it closer to the physical environment state rather than relying only on the simulated values. Another feedback layer is added to validate and correct the emulator's output. The emulator uses the readings of the deployed IoT sensor nodes and correct the emulation output to produce better results for the rooms where the sensors are installed.



□ Auto Aspect Ratio
Width 800
Height 600
Cell Size 6
Diffusivity 0.0002
Viscosity 0.0
Iterations 12
Time Step 0.25
□ Use Local Search

FIGURE 6.1: a) A circuit diagram for the hardware sensor node and its connection with the system. b) The actual system and the emulator interface

Each space in the simulation, such as a room, has a timestamped readings' buffer from its node. At each iteration of the solver, it performs a correction step to converge gradually to the ground-truth.

The correction algorithm remains the same regardless of the data transfer scheme is data pooling or data streaming. The correction step for each of the zones uses the most relevant reading, which has the nearest timestamp to the simulation time and has a time offset less than a 5 seconds threshold and is given by

$$d_{corr} = (1 - k_r \cdot k_c)\, d_{sim} + k_r \cdot k_c\, (d_{sim} - \mu_{sim} + d_{fb}), \tag{6.1}$$

where d_{corr} denotes the density array of zone's cells post correction, k_r denotes feedback value relevancy factor, k_c denotes correction factor, d_{sim} denotes the density array of zone's cells prior to correction, μ_{sim} denotes mean of zone's cells density array prior to correction, and d_{fb} denotes the density value from buffer's most relevant reading.

The smoke density value of a zone d_{fb} is a scaled value of the pressure reading for this zone to represent the pressure projection into 2D.

The relevancy factor k_r is a normalized value of the time offset between the most relevant reading in the buffer and the simulation time, whereas the correction factor k_c is a hyper-parameter that controls the responsiveness and speed of the correction.

Fig. 6.2 shows the smoke saturation of a room in a simulation case with a noise subtracted to represent a case where simulation diverges immensely from the true state inferred from sensor's readings.

As seen from Fig. 6.2, the higher k_r, the faster the simulation converges to the same saturation state the sensor indicates, but the high responsiveness of the correction has its own drawback. The correction step shifts the values of the density of a zone's cells equally at the moment of correction so that their mean is closer to the mean of the most relevant reading. The high correction factor will compromise the consistency of the simulation through time steps. The correction will boost or dampen the simulated values rapidly, giving the fluid solver too large of a shift in the density values suddenly. This sudden shift might alter the behavior of the flow that will not match the ground-truth nor the solver's result without correction, which will lead to unreliable simulation.

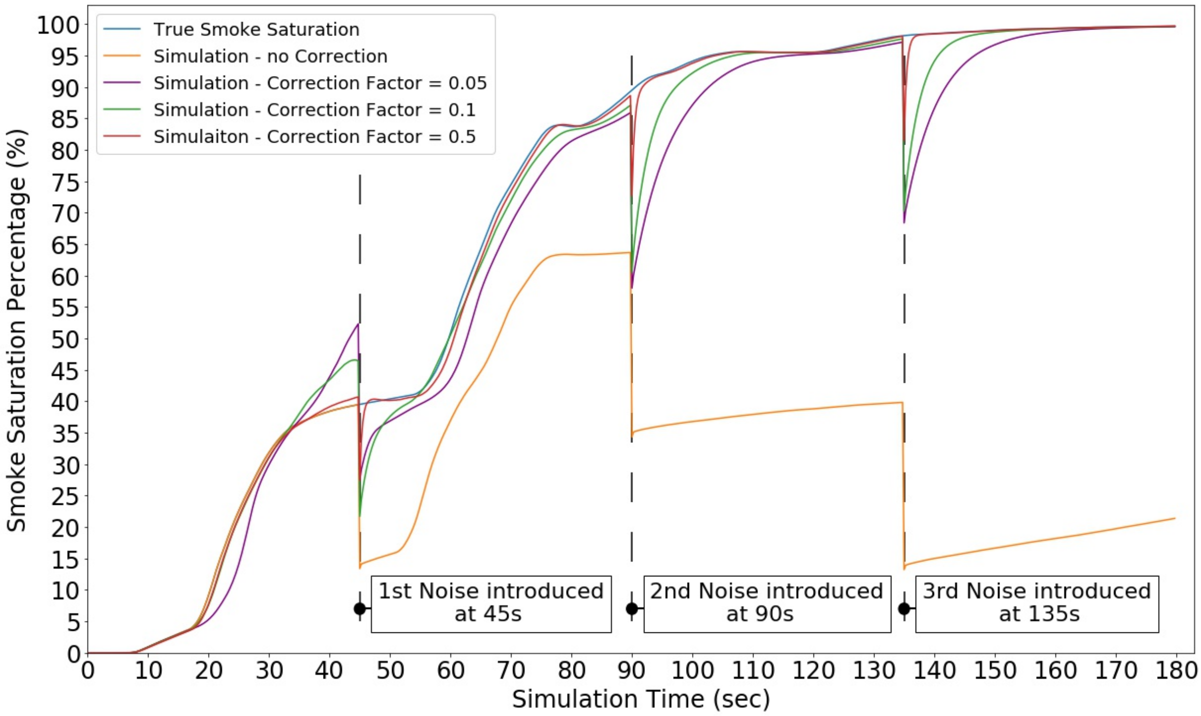


FIGURE 6.2: Correction Convergence Comparison

Gaussian noise is introduced to the zone's cells to randomly change the simulated values in Fig.6.3, shifting the simulated mean and emulating divergence from ground-truth without giving bias to a certain area of the simulated space.

The trade-off between correction responsiveness and the realism of the corrected simulation results is shown in Fig. 6.3, where using a high value for k_r shifts the mean by a large positive value and amplifies the characteristics of the flow, which is already deemed divergent due to the relatively large difference between it and the sensors readings, thus making its characteristics less reliable even though they have a mean very close to the reading after the correction. In comparison, using smaller values for k_r managed to keep the simulation consistent through time but took longer to converge to 95% of the reading's value.

The correction step adjusts all cells in a zone with the same value, but in case of a spatial information about the sensor location or using multiple sensor nodes in a zone, the correction step can be adjusted to accommodate the spatial relations between the position of the simulated cells and the location of the sensors. The spatial information can be introduced as an array of weights inversely related to the sensors' euclidean distance.

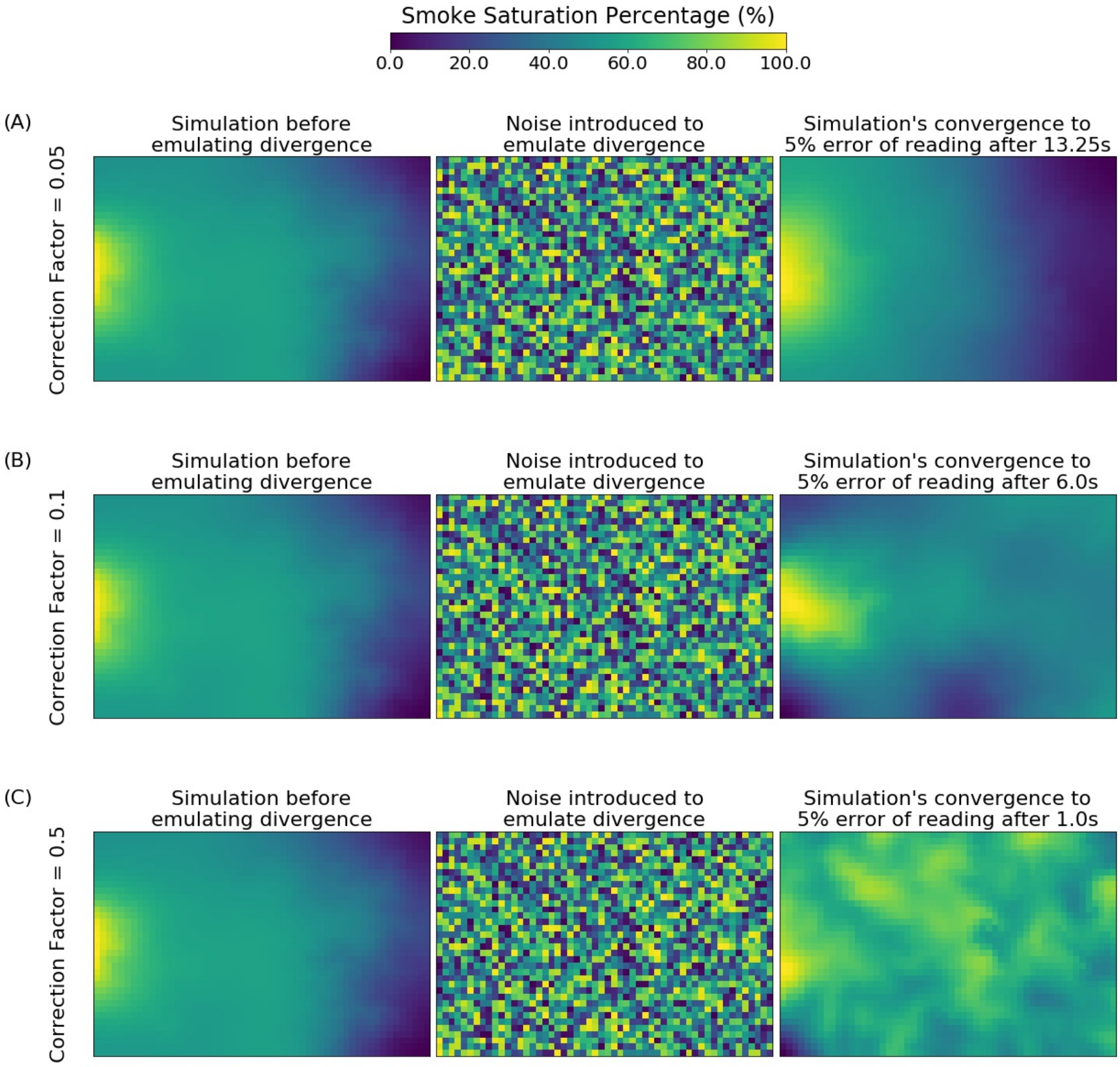


FIGURE 6.3: Correction Behavior Comparison

This ability to forecast possible sensor outputs allows for robust and temporally efficient path planning, even if it is for the very near future. Moreover, the low computation cost of the solver allows for the update of the smoke and path planning process almost instantaneously in case of sudden dynamic events in the environment.

TABLE 6.1: Processing time comparison with FDS

Time Simulated (Seconds)	Our Emulator while displaying results (Seconds)	FDS without displaying results (Seconds)
60	12.5	86.2
120	26.8	175.5
240	57.6	191

6.2 Comparison with FDS

The comparison between the FDS and this developed emulator has shown similar results in multiple cases specially after adding the hardware correction layer. However, our emulator runs much faster so it can be used to run real time prediction whilst FDS has a large computation overhead, as comparison Table 6.1. The emulator is also capable of handling the dynamic events better than FDS which requires predefined conditions. Figure 6.4 and figure 6.5 6.2 show that the final solver is able to provide smoke propagation that is similar to FDS results numerically and visually, and faster than real time in different fire cases. Figure 6.2 shows that the emulator gives the same saturation that matches the pressure changing in the 3D space and it has noise immunity that helps to recover in case of any error occurs.

Figure 6.6 shows that performance of the emulator when the door is suddenly opened. The performance of the proposed emulator on the left is very close to FDS, which indicates that the emulator can handle sudden events. The sudden events handling in such system is a must to have a real-time and reliable solution.

Figure 6.5 shows that the interactions with bodies dampen the flow by comparing the smoke flow characteristics after 40 and 170 simulated seconds from the start of the fire and the results from FDS for the same moments. It's extremely visible that the smoke flow in the upper right room in the simulator has less turbulence and energy whereas FDS simulated flow didn't lose any notable amount of the energy source.

6.3 Search Process

The single iteration of the integrated search algorithm requires more processing than 1 level search, yet it uses fewer iterations to find the goal in most cases. The 1 level search works better in cases where evacuation start and goal points are close to each other and not in contact with smoke. The two levels search is used mainly to simplify the solution the algorithm needs to obtain, thus reducing the number of visited cells by the algorithm as well as the number of iterations to find the goal. The integrated 2 levels search also has a great advantage by making use of zone information directly at the layout of the building. Fig. 6.7 shows how the 2 levels search process quickly responds to sudden smoke changes in a very short time while showing the last complete evacuation route during the search for a new safer egress. The fact that the smoke solver operates faster than real time enables it to provide future paths away from smoke inhabited zones predicted by the solver.

6.4 Environment Evacuation

The proposed system gives real-time integration with the physical environment through the IoT layer. As mentioned in the correction layer, the solver corrects its results and updates the environment state to depict the ground-truth as much as possible, then the simulated results are used to find prime evacuation routes. The results update the exit and directional signs on the emulator and the signs in the physical environment through the IoT layer also. Fig. 6.8 shows how the proposed system updates the signs states along with updates of prime evacuation routes. The signs labeled exit are used on each door, while directional signs are used on zones' walls to guide evacuees to the nearest exit of safety.

6.5 Use of the Emulator

This system has several features to help firefighters and occupants. The system provides the firefighters with a fire scene map that has the fire's location, temperature, the pressure of different zones, and the motion-activity map of occupants on the map. The firefighters can grab a snapshot of the current scene and try to test different evacuation scenarios in the simulator, which will reduce hazards such as back-draft [67]. They can speed up the simulation to see the future smoke prediction as the emulator is five times faster than real-time. Based on that, they may decide to change their location or even withdraw the team from the building due to upcoming hazards. They can use the system to allocate trapped occupants on the map then use the routing algorithm to reach them faster.

6.6 Deployment

To implement the proposed emulator in a real building, two simple steps need to be followed. The first step is to install and calibrate the sensors, to add control to the evacuation signs, and to set the MQTT clients for each of them. The second step is to synchronize the sensors and door nodes with their counterparts in the emulator at initialization only. Either the MQTT clients in physical environment or the emulator client would have a predefined parameters and messages formats to be able to communicate.

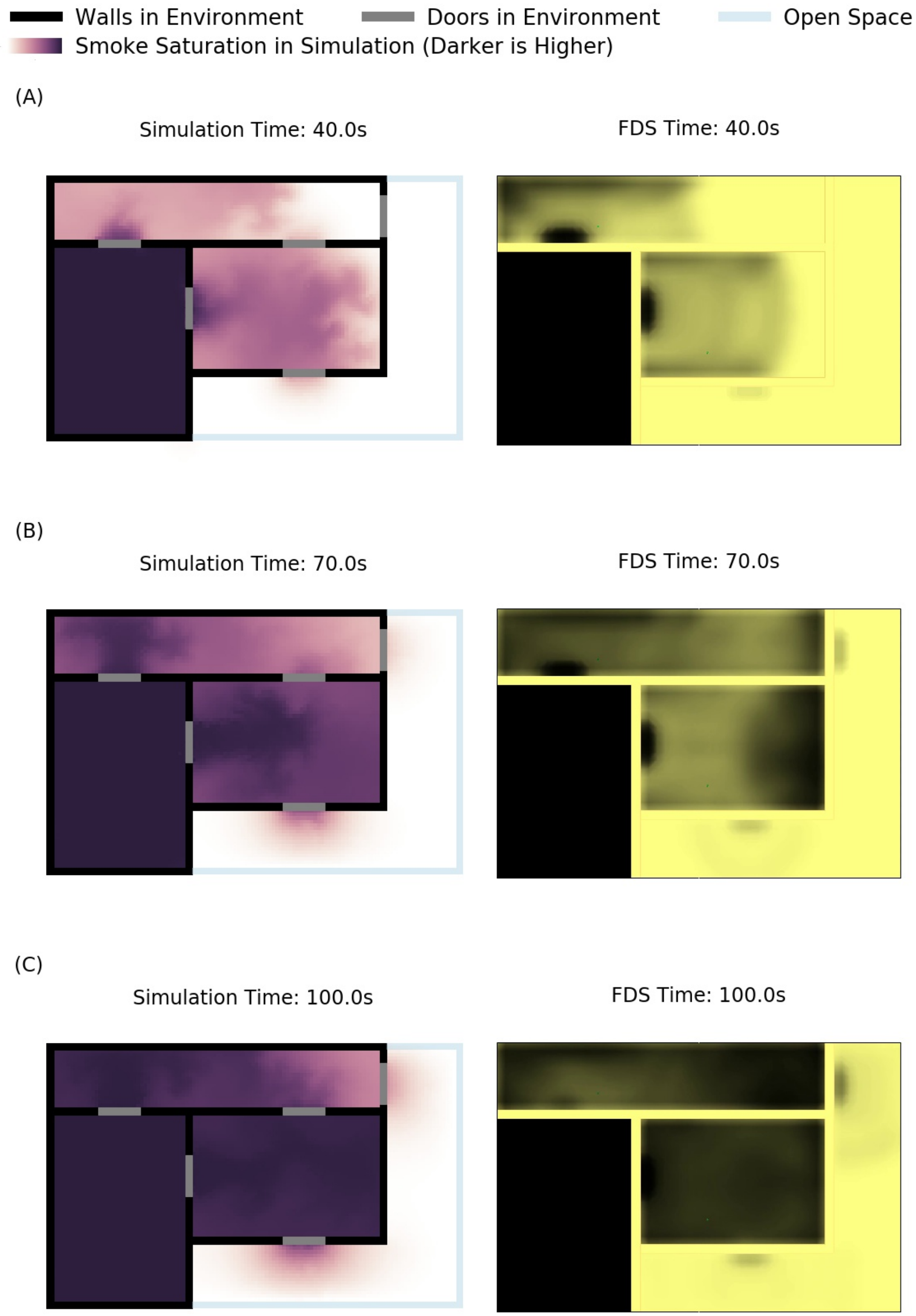


FIGURE 6.4: Simulator and FDS Results Comparison Case 2

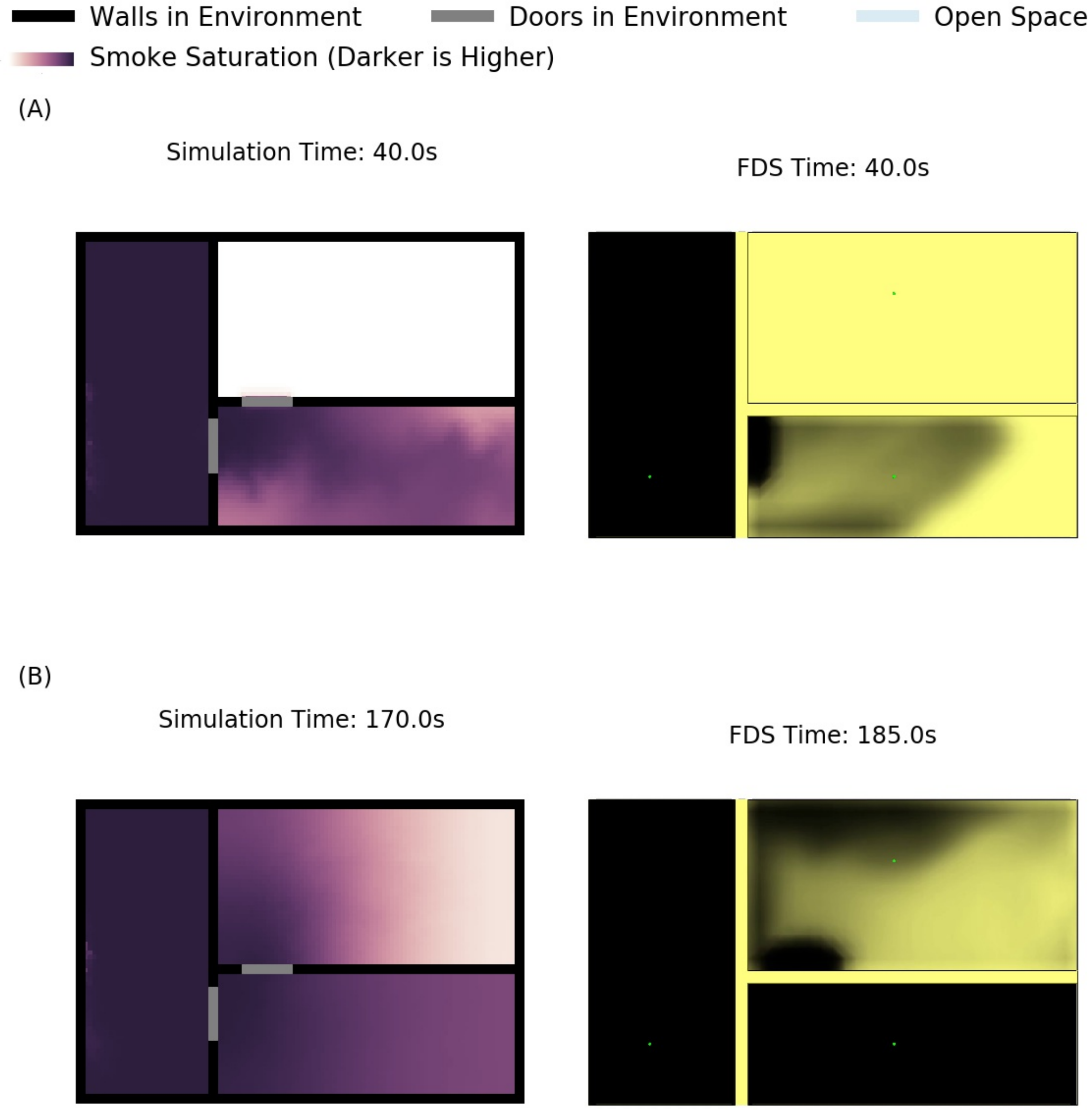


FIGURE 6.5: Simulator and FDS Results Comparison Case 1

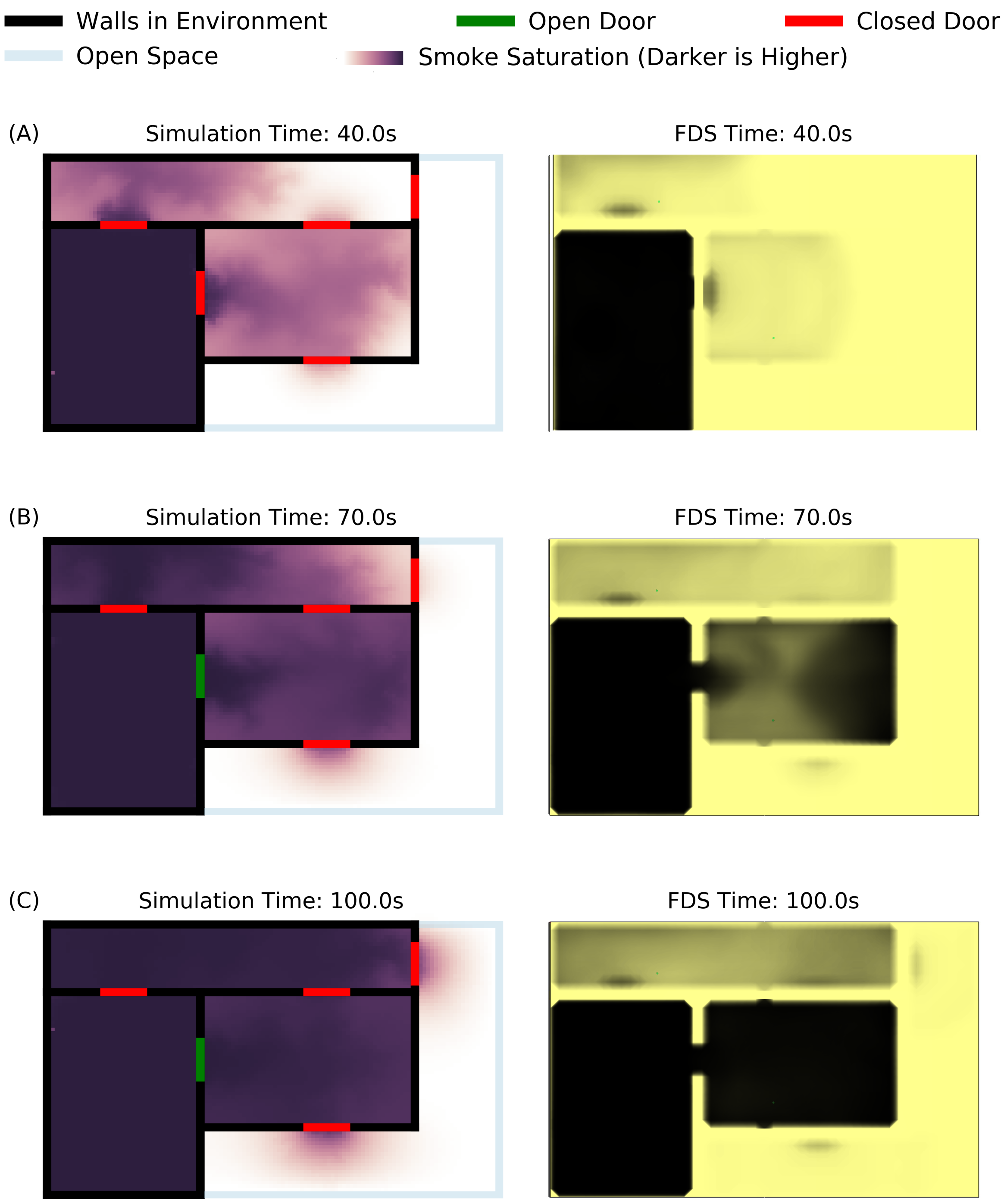
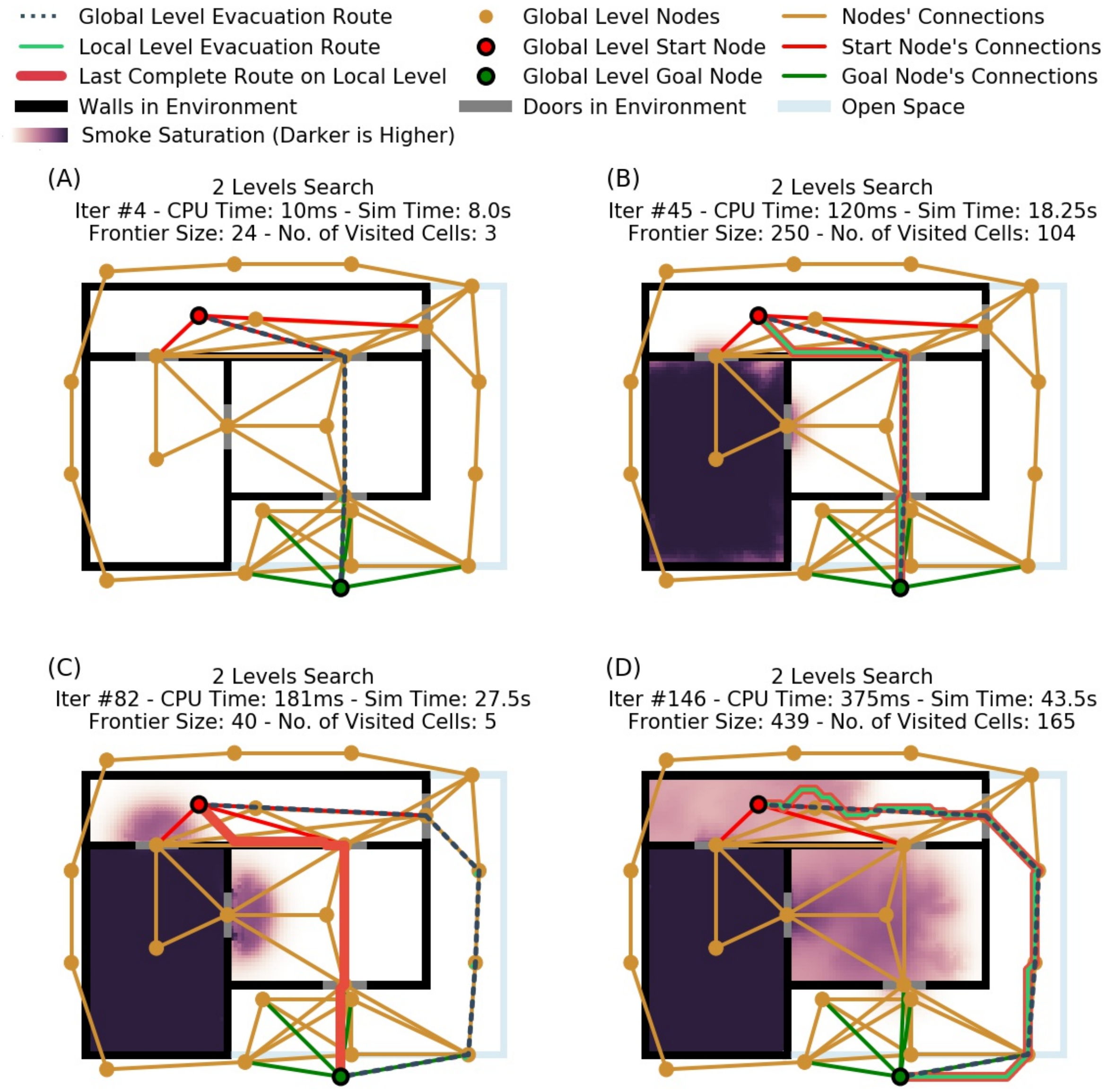


FIGURE 6.6: Simulator and FDS Results Comparison Case 3 - opened door



Global Level Evacuation Route
Local Level Evacuation Route
Last Complete Route on Local Level
Walls in Environment
Smoke Saturation (Darker is Higher)
Global Level Nodes
Global Level Start Node
Global Level Goal Node
Doors in Environment
Nodes' Connections
Start Node's Connections
Goal Node's Connections
Open Space
(A)
2 Levels Search
Iter #4 - CPU Time: 10ms - Sim Time: 8.0s
Frontier Size: 24 - No. of Visited Cells: 3
(B)
2 Levels Search
Iter #45 - CPU Time: 120ms - Sim Time: 18.25s
Frontier Size: 250 - No. of Visited Cells: 104
(C)
2 Levels Search
Iter #82 - CPU Time: 181ms - Sim Time: 27.5s
Frontier Size: 40 - No. of Visited Cells: 5
(D)
2 Levels Search
Iter #146 - CPU Time: 375ms - Sim Time: 43.5s
Frontier Size: 439 - No. of Visited Cells: 165

FIGURE 6.7: Evolving of Integrated Global and Local Search Processes

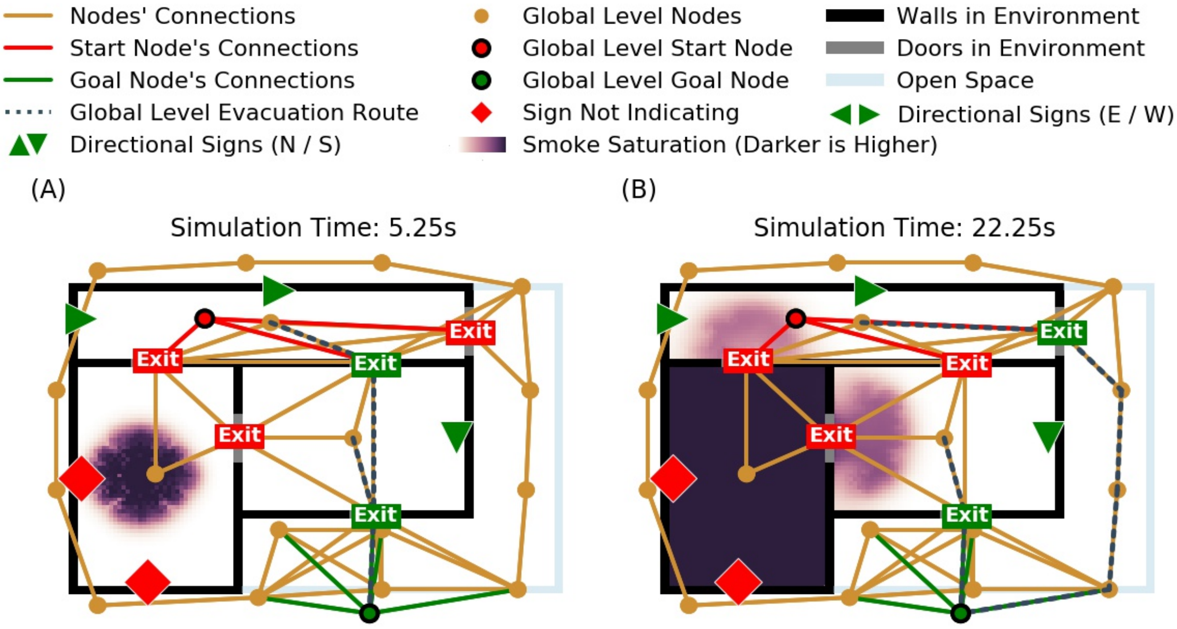


FIGURE 6.8: Prime Evacuation Routes Evolution with Simulation

Chapter 7

Conclusion

This work introduced a new system that can assist firefighters and occupants in a fire situation. The proposed system is using a light-weight algorithm that can provide data in real-time with high accuracy. The system creates digital twins using the internet of things and the LSTM network to correct the emulator from any deviation. This system overcomes two of the FDS problem. Firstly, it runs faster and can provide data in real-time, not with a playback such as FDS. Secondly, the emulator is better than FDS, where it can interact with the environment to change the emulation performance. The interaction is a critical factor in the fire situation due to the dynamic nature of the scene. Getting feedback from the environment helps to avoid problems that might occur, such as back-draft. The emulator will assist firefighter by showing them the fire development in real-time, which will ultimately help them set better evacuation and fire extinguishing plans. The firefighter can take a snapshot from the system and try different evacuation techniques before proceeding, which will be safer and more efficient for them. The system will control the exit signs in order to guide people away from the smoke to increase their survival rate. The exit signs will be assigned based on a developed two-leveled A* routing algorithm. The IoT hardware is designed and optimized to operate in dual mode. The first mode is using LoRa in regular operation, as it has a low power consumption. In the fire situation, the hardware will switch to WiFi for fast and low latency data transfer. The hybrid designs can run the node for 88 days against 39.84 hours for WiFi design and 70.59 hours for LoRa design. After adding all the layers and tuning the emulator hyper-parameters, the proposed emulator gives a result very close to FDS in the two different testing scenarios.

More features can be added to the proposed emulator to depict other dynamics of the fire propagation in buildings and how the materials of the rigid bodies represented in the emulator such as walls, affect the propagation of smoke directly. This is an ongoing research topic and there are recent contributions to fire simulations to depict material properties' effects on the spreading of fire in buildings[10][11].

The current state of the proposed emulator focuses on a single floor fire emulation and evacuation The emulator can handle emulate multiple floors in parallel, however, some extra components needs to be designed to deal with the 3D structure of the building.[13]. In the future, the system can also relies on 5G to communicate with the server in order to overcome any infrastructure failure such as the access point during the fire . The system can also represented as nodes that is connected through a budget link. This implementation will help to scale up the environment and reduce the computational overload. The smoke engine will run on each single zone separately for more accuracy and the budget link will handle the smoke movement behaviour from zone to another. This will help to focus the CPU resources on the nodes that need the resources the most and ignore the idle nodes that does not contribute to the fire scene.

Bibliography

[1] Statistics Canada, “Table 35-10-0194-01 fire-related deaths and persons injured, by cause of death or injury and reason for non-evacuation,” 2015.

[2] Statistics Canada, “Table 35-10-0195-01 fire-related deaths and persons injured, by type of structure,” 2015.

[3] Statistics Canada, “Table 35-10-0197-01 incident-based fire statistics, by performance of sprinkler system, structural fires,” 2015.

[4] S. A. Kahn, C. Leonard, Y. G. Lee, R. Boatwright, T. Flamm, and J. Woods, “A pilot survey of southeastern firefighters: Safety practices, use of protective gear, and injury,” *Burns*, vol. 46, pp. 298 – 302, March 2020.

[5] Department for Communities and Local Government, United Kingdom Government, “Fire statistics great britain 2010 to 2011,” 2011.

[6] T. Grabowska, R. Skowronek, J. Nowicka, and H. Sybirska, “Prevalence of hydrogen cyanide and carboxyhaemoglobin in victims of smoke inhalation during enclosed-space fires: a combined toxicological risk,” *Clinical Toxicology*, vol. 50, pp. 759–763, Aug. 2012.

[7] G. Rein, A. Bar-Ilan, A. Fernandez-Pello, and N. Alvares, “A comparison of three models for the simulation of accidental fires,” *Journal of Fire Protection Engineering*, vol. 16, pp. 183–209, Aug. 2006.

[8] D. Yang, L. Hu, Y. Jiang, R. Huo, S. Zhu, and X. Zhao, “Comparison of FDS predictions by different combustion models with measured data for enclosure fires,” *Fire Safety Journal*, vol. 45, pp. 298 – 313, Aug. 2010.

[9] M. Adam aigo, “Numerical solution of navier stokes equation using control volume and finite element method,” *International Journal of Applied Mathematical Research*, vol. 5, pp. 63–68, Feb. 2016.

[10] P. McKeen and Z. Liao, “A three-layer macro-scale model for simulating the combustion of PPUF in CFD,” *Building Simulation*, vol. 9, pp. 583–596, Oct. 2016.

[11] P. McKeen and Z. Liao, “Pyrolysis model for predicting the fire behavior of flexible polyurethane foam,” *Building Simulation*, vol. 12, pp. 337–345, Nov. 2019.

[12] L. Shi, M. Y. L. Chew, X. Liu, X. Cheng, B. Wang, and G. Zhang, “An experimental and numerical study on fire behaviors of charring materials frequently used in buildings,” *Energy and Buildings*, vol. 138, pp. 140 – 153, March 2017.

[13] P. Mckeen and Z. Liao, “The influence of building airtightness on airflow —in stairwells,” *Buildings*, vol. 9, Sept. 2019.

[14] F. Mirahadi and B. McCabe, “A real-time path-planning model for building evacuations,” in *ISARC Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 36, pp. 998–1004, Jan. 2019.

[15] G. R, “Comparative analysis of various uninformed searching algorithms in ai,” *International Journal of Computer Science and Mobile Computing*, vol. 8, pp. 57–65, June 2019.

[16] M. Panda and A. Mishra, “A survey of shortest-path algorithms,” *International Journal of Applied Engineering Research*, vol. 13, no. 9, pp. 6817–6820, 2018.

[17] M.-Y. Cheng, K.-C. Chiu, Y.-M. Hsieh, I.-T. Yang, J.-S. Chou, and Y.-W. Wu, “BIM integrated smart monitoring technique for building fire prevention and disaster relief,” *Automation in Construction*, vol. 84, pp. 14 – 30, Dec. 2017.

[18] W. Shu-Xi, “The improved dijkstra’s shortest path algorithm and its application,” *Procedia Engineering*, vol. 29, pp. 1186 – 1190, 2012.

[19] P. Singal and R. Chhillar, “Dijkstra shortest path algorithm using global positioning system,” *International Journal of Computer Applications*, vol. 101, pp. 12–18, Sept. 2014.

[20] K. Kim and Y.-C. Lee, “Automated generation of daily evacuation paths in 4d bim,” *Applied Sciences*, vol. 9, Jan. 2019.

[21] N. Al-Nabhan, N. Al-Aboody, and A. A. A. Islam, “A hybrid IoT-based approach for emergency evacuation,” *Computer Networks*, vol. 155, pp. 87 – 97, May 2019.

[22] C.-H. Wu and L.-C. Chen, “3d spatial information for fire-fighting search and rescue route analysis within buildings,” *Fire Safety Journal*, vol. 48, pp. 21 – 29, Feb. 2012.

[23] “Internet of things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, pp. 1645 – 1660, Sept. 2013.

[24] A. Abdelsalam, H. Islam, P. Song, M. Gamaleldin, A. Radhi, N. Panwar, S. C. Tjin, A. Y. Desoky, D. Sinton, K.-T. Yong, and J. Zu, “Self-adaptive bioinspired hummingbird-wing stimulated triboelectric nanogenerators,” *Scientific Reports (Nature Publisher Group)*, vol. 7, pp. 1–9, Dec. 2017.

[25] K. Kang, J. Lin, and J. Zhang, “BIM- and IoT-based monitoring framework for building performance management,” *Journal of structural integrity and maintenance*, vol. 3, pp. 254–261, Oct. 2018.

[26] H. Alawad and S. Kaewunruen, “Wireless sensor networks: Toward smarter railway stations,” *Infrastructures*, vol. 3, Sept. 2018.

[27] B. P. L. Lau, N. Wijerathne, B. K. K. Ng, and C. Yuen, “Sensor fusion for public space utilization monitoring in a smart city,” *IEEE Internet of Things Journal*, vol. 5, pp. 473–481, April 2018.

[28] A. Pastor-Aparicio, J. Segura-Garcia, J. Lopez-Ballester, S. Felici-Castell, M. García-Pineda, and J. J. Pérez-Solano, “Psychoacoustic annoyance implementation with wireless acoustic sensor networks for monitoring in smart cities,” *IEEE Internet of Things Journal*, vol. 7, pp. 128–136, Jan. 2020.

[29] C. Huang, Z. Liao, and L. Zhao, “Synergism of ins and pdr in self-contained pedestrian tracking with a miniature sensor module,” *IEEE Sensors Journal*, vol. 10, pp. 1349–1359, Aug. 2010.

[30] Q. Li, M. De Rosa, and D. Rus, “Distributed algorithms for guiding navigation across a sensor network,” (New York, NY, USA), Association for Computing Machinery, 2003.

[31] X. Zhang, M. Pipattanasomporn, T. Chen, and S. Rahman, “An IoT-based thermal model learning framework for smart buildings,” *IEEE Internet of Things Journal*, vol. 7, pp. 518–527, Jan. 2020.

[32] G. Gai and P. Cancelliere, “Design of a pressurized smokeproof enclosure: CFD analysis and experimental tests,” *Safety*, vol. 3, June 2017.

[33] M. Barnes, H. Leather, and D. K. Arvind, “Emergency evacuation using wireless sensor networks,” in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, pp. 851–857, Nov. 2007.

[34] National Fire Protection Association, “Smoke alarms in U.S. home fires,” 2019.

[35] A. Pavan, C. Bolognesi, F. Guzzetti, E. Sattanino, E. Pozzoli, L. D’Abrosio, C. Mirarchi, and M. Mancini, “BIM digital platform for first aid: Firefighters, police, red cross,” in *Digital Transformation of the Design, Construction and Management Processes of the Built Environment* (B. Daniotti, M. Gianinetto, and S. Della Torre, eds.), pp. 279–289, Springer International Publishing, 2020.

[36] H. Park, B. J. Meacham, N. A. Dembsey, and M. Goulthorpe, “Integration of fire safety and building design,” *Building Research and Information*, vol. 42, pp. 696–709, Nov. 2014.

[37] F. Cui, “Deployment and integration of smart sensors with IoT devices detecting fire disasters in huge forest environment,” *Computer Communications*, vol. 150, pp. 818 – 827, Jan. 2020.

[38] B. Dave, A. Buda, A. Nurminen, and K. Främling, “A framework for integrating BIM and IoT through open standards,” *Automation in Construction*, vol. 95, pp. 35 – 45, Nov. 2018.

[39] M. Choi and S. Chi, “Optimal route selection model for fire evacuations based on hazard prediction data,” *Simulation Modelling Practice and Theory*, vol. 94, pp. 321 – 333, July 2019.

[40] M. Nosrati, R. Karimi, and H. A. Hasanvand, “Investigation of the * (star) search algorithms: Characteristics, methods and approaches,” *World Applied Programming*, vol. 2, pp. 251–256, April 2012.

[41] X.-S. Chen, C.-C. Liu, and I.-C. Wu, “A BIM-based visualization and warning system for fire rescue,” *Advanced Engineering Informatics*, vol. 37, pp. 42 – 53, Aug. 2018.

[42] Y. Peng, S.-W. Li, and Z.-Z. Hu, “A self-learning dynamic path planning method for evacuation in large public buildings based on neural networks,” *Neurocomputing*, vol. 365, pp. 71 – 85, July 2019.

[43] J. C. Latombe, “Exact cell decomposition,” in *Robot Motion Planning*, vol. 142, pp. 200–247, Springer, 1991.

[44] J. C. Latombe, “Approximate cell decomposition,” in *Robot Motion Planning*, vol. 142, pp. 248–294, Springer, 1991.

[45] J. Dimyadi, M. Spearpoint, and R. Amor, “Generating fire dynamics simulator geometrical input using an IFC-based building information model,” *Journal of Information Technology in Construction*, vol. 12, pp. 443–457, Oct. 2007.

[46] J. Stam, “Real-Time Fluid Dynamics for Games,” in *Proceedings of the Game Developer Conference*, vol. 18, March 2003.

[47] R. Fedkiw, J. Stam, and H. W. Jensen, “Visual simulation of smoke,” in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, p. 15–22, Aug. 2001.

[48] J. Steinhoff and D. Underhill, “Modification of the euler equations for “vorticity confinement”: Application to the computation of interacting vortex rings,” *Physics of Fluids*, vol. 6, no. 8, pp. 2738–2744, 1994.

[49] M. Schluse, L. Atorf, and J. Rossmann, “Experimentable digital twins for model-based systems engineering and simulation-based development,” in *2017 Annual IEEE International Systems Conference (SysCon)*, pp. 1–8, 2017.

[50] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, “Understanding the limits of LoRaWAN,” *IEEE Communications Magazine*, vol. 55, pp. 34–40, Sept 2017.

[51] D. Magrin, M. Capuzzo, and A. Zanella, “A thorough study of LoRaWAN performance under different parameter settings,” *IEEE Internet of Things Journal*, vol. 7, pp. 116–127, Jan. 2020.

[52] W. Xu, J. Y. Kim, W. Huang, S. S. Kanhere, S. K. Jha, and W. Hu, “Measurement, characterization, and modeling of LoRa technology in multifloor buildings,” *IEEE Internet of Things Journal*, vol. 7, pp. 298–310, Jan. 2020.

[53] R. A. Atmoko, R. Riantini, and M. K. Hasin, “IoT real time data acquisition using MQTT protocol,” *Journal of physics. Conference series*, vol. 853, May 2017.

[54] S. Pal, S. Ghosh, and S. Bhattacharya, “Study and implementation of environment monitoring system based on MQTT,” *Environmental and Earth Sciences Research Journal*, vol. 4, pp. 23–28, March 2017.

[55] L. I. Burke, “Introduction to artificial neural systems for pattern recognition,” *Computers Operations Research*, vol. 18, no. 2, pp. 211 – 220, 1991.

[56] H. H. Yang and S.-i. Amari, “Complexity issues in natural gradient descent method for training multilayer perceptrons,” *Neural Computation*, vol. 10, no. 8, pp. 2137–2157, 1998.

[57] S. ichi Amari, “Backpropagation and stochastic gradient descent method,” *Neurocomputing*, vol. 5, no. 4, pp. 185 – 196, 1993.

[58] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[59] W. Chen, Z. Zhao, J. Liu, P. Chen, and X. Wu, “Lstm network: A deep learning approach for short-term traffic forecast,” *IET Intelligent Transport Systems*, vol. 11, 01 2017.

[60] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[61] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[62] F. Chollet *et al.*, “Keras.” `https://keras.io`, 2015.

[63] Z. Wang and A. C. Bovik, “Mean squared error: Love it or leave it? a new look at signal fidelity measures,” *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[64] J.-S. Chou, M.-Y. Cheng, Y.-M. Hsieh, I.-T. Yang, and H.-T. Hsu, “Optimal path planning in real time for dynamic building fire rescue operations using wireless sensors and visual guidance,” *Automation in Construction*, vol. 99, pp. 1–17, March 2019.

[65] W. Yan, C. Culp, and R. Graf, “Integrating BIM and gaming for real-time interactive architectural visualization,” *Automation in Construction*, vol. 20, pp. 446 – 458, July 2011.

[66] S. Bhattacharjee, P. Roy, S. Ghosh, S. Misra, and M. S. Obaidat, “Wireless sensor network-based fire detection, alarming, monitoring and prevention system for bord-and-pillar coal mines,” *Journal of Systems and Software*, vol. 85, pp. 571 – 581, March 2012.

[67] D. Myilsamy, C. B. Oh, and B.-I. Choi, “Large eddy simulation of the backdraft dynamics in compartments with different opening geometries,” *Journal of Mechanical Science and Technology*, vol. 33, pp. 2189–2201, May 2019.