要約

自然を相手にする農業従事者が抱える問題として獣害がある．獣害被害の中でも夜間に発生するものは，害獣の侵入に気づくことが困難であり，日中と異なり農業従事者が直接追い払うことが難しい．そのため基本的には獣害被害は事後に痕跡から経路や動物の種類を推定して電気柵や対策が取られている．しかし，害獣の正確な侵入経路がわからないため，柵の隙間といった不備のある位置がわからないことやそもそも推定した動物の種類が間違っているなど対策が機能しないことも少なくない．そのため適切な対策を取るために様々な対策を試す必要があり，大きな手間と時間・費用を要している．そこで我々はこれらの問題を解決するために，畑への侵入を検出・通知し，その動物の行動や大きさを推定することで，農業従事者の獣害対策を効果的に行えるよう支援するシステムを提案する．特に本論文ではクラウドサーバを用いて，畑への侵入したという情報および推定して得られた結果を保存するデータベース，畑への侵入情報から農業従事者へ通知する Web アプリケーション，推定された行動や大きさを地図を用いて行動履歴を表示する Web アプリケーションの設計を行った．機能としては，リバースプロキシ，WebPush を用いた通知サーバ，Leaflet.js による地図を用いた行動履歴表示を行う Web アプリケーション，データベースおよびデータベースアクセス制御用の内部 API があり，それぞれを Docker Compose を駆使して連携させたクラウドサーバを設計した．設計・実装後，クラウドサーバがやり取りする畑のゲートウェイに搭載される無線モジュールとの疎通確認および通信形式の確認，クラウドサーバ内の通知サーバからブラウザへの Push 通知の確認，クラウドサーバ内の Web アプリケーションによる行動履歴表示の確認を行った．さらなる実際の使用環境に合わせた改善をするには，全体を連携させて実際の畑で得られたセンサーデータで解析，通知・表示を行うことが必要である．展望としては，本論文担当であるクラウドサーバについてセキュリティや UI・UX の向上の他にリバースプロキシとデータベースアクセス制御用の内部 API が通信量のボトルネックになるといった懸念点が挙げられる．通信量のボトルネックは，本論文のシステム構成ではリバースプロキシと内部 API にデータが集中するのに対し 1 つずつしか用意していないことが根本的な原因である．そのため，kubernetes といった分散管理システムの導入することで，更なる冗長性とスケーラビリティが実現することができる．

目 次

1　まえがき

自然を相手にする農業従事者が抱える問題として獣害がある．令和 4 年度の野生鳥獣による全国の農作物被害は約 156 億円で，特にシカの被害額は約 65 億円で前年の被害額と比べて増加傾向にある [1].

獣害被害の中でも夜間に発生するものは，害獣の侵入に気づくことが困難であり，日中と異なり農業従事者が直接追い払うことが難しい．そのため基本的には獣害被害は事後に痕跡から経路や動物の種類を推定して電気柵や対策が取られている．しかし，害獣の正確な侵入経路がわからないため柵の隙間といった不備のある位置がわからないことやそもそも推定した動物の種類が間違っているなど対策が機能しないことも少なくない．そのため適切な対策を取るために様々な対策を試す必要があり，大きな手間と時間・費用を要している．

我々はこれらの問題を解決するために，畑への侵入を検出・通知し，その動物の行動や大きさを推定することで農業従事者の獣害対策を効果的に行えるよう支援するシステムを提案する．具体的な流れとしては，畑の周囲に赤外線センサを搭載したセンサノードを規則的に配置する．赤外線センサノードが検出した侵入した動物のデータを LoRa 通信および LTE 通信を用いてクラウドサーバへ送信する．クラウドサーバでは収集されたセンサデータを用いて農業従事者に通知やセンサデータの解析で得られた畑内に侵入した動物についての行動履歴を地図と合わせて農業従事者に提示する．

本論文では著者が担当するクラウドサーバの解析機能以外の機能群について述べる．クラウドサーバの著者担当部分は大きく分けて，データの保存，農業従事者への通知，行動履歴の表示の 3 つがある．データの保存では畑に設置した赤外線センサノードから得られたセンサデータ，センサデータから解析して得られた行動履歴のデータ，登録されている畑についてのデータについて保存し，管理する．農業従事者への通知ではセンサデータの送信に応答して農業従事者の個人端末へ通知を行う Web アプリケーションを実現する．行動履歴の表示ではクラウドサーバ内に保存された行動履歴のデータを用いて農業従事者の個人端末で確認可能な Web アプリケーションを実現する．以上の 3 つの機能について設計・実装をし，動作確認を行う．

2　理論

本章では，本研究の前提となる知見や技術について説明する．

2.1　Docker

Docker はコンテナ技術の一種で，システムの環境ごとに隔離した空間を用意して独自のディレクトリツリーを提供する．Docker コンテナはそれ単体でシステムが完結しているため，Docker コンテナを別のサーバにコピーして起動させることも容易であり，ポータビリティ性を有している [2]．本論文では Docker を用いて機能ごとに Docker コンテナを作成する．

2.1.1　Docker イメージ

Docker イメージは Docker コンテナ作成を支援するために用意されたアーカイブパッケージである．Docker コンテナは独立したシステム実行環境であるため，一から構築するには OS やライブラリなど全てを作り込む必要があり非常に手間がかかる．Docker コンテナ作成に必要なファイルを纏めたアーカイブパッケージが Docker イメージである [2]．Docker イメージは Docker 社が運営している Docker Hub で登録・公開されており，Docker Hub に公開されている Docker イメージをダウンロードして Docker コンテナを作成することが可能である．

Docker イメージには以下の 2 種類ある．

- 基本的な Linux ディストリビューションだけの Docker イメージ
- アプリケーション入りの Docker イメージ

本論文では後者のアプリケーション入りの Docker イメージを用いてシステムの Docker コンテナを作成する．

2.1.2　Dockerfile

Dockerfile は自作のイメージを作るもので，ベースとなるイメージに対してどのような変更指示をするかをまとめたファイルである．Dockerfile は「docker build」コマンドを用いてビルドすることで Docker イメージを作成できる [2]．

2.1.3 Docker ネットワーク

Docker コンテナは独立しており，1 つの Docker コンテナがサーバマシンのような役割を持っている．Docker コンテナ間が通信するために Docker 内に仮想的なネットワークを構築する必要がある．Docker には以下の 3 種類のネットワークが用意されている．

- bridge ネットワーク
- host ネットワーク
- none ネットワーク

ここでは，本論文で利用する bridge のみ説明し，他 2 種類については説明を省く．

bridge ネットワークでは IP マスカレードを利用してホスト PC が所属する新たな内部ネットワークを構築している． 図 1 に Apache の Docker コンテナを 2 つ起動し，それぞれのポートをホスト PC の 8080 と 8081 に接続した時の例を示す．

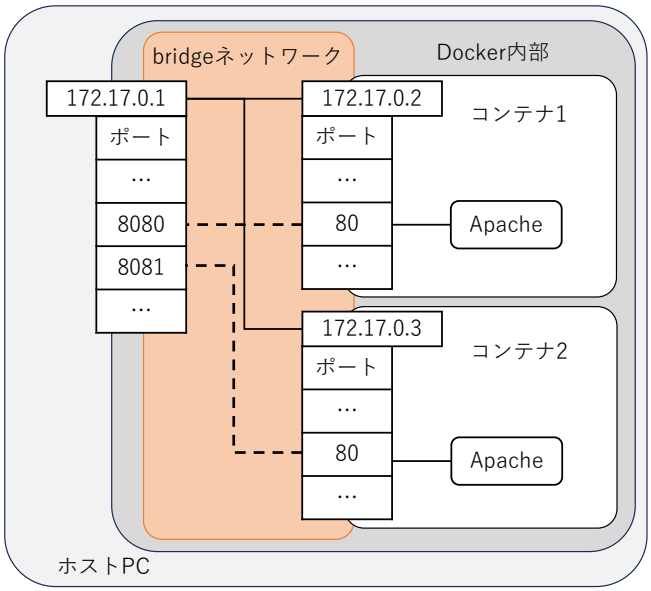

図 1: bridge ネットワーク例

2.1.4 Docker Compose

Docker Compose は複数の Docker コンテナの作成および Docker コンテナ間のネットワークの作成など，複数の Docker コンテナの作成・停止・破棄の一連の操作

をまとめて実行する仕組みのことである [2]．一連の Docker コンテナやシステムについての定義ファイル (Compose ファイル) は既定では「docker-compose.yml」というファイル名になっている．Compose ファイルを「docker-compose(docker compose)」コマンドで実行するとボリュームやネットワークが作成され，まとめて Docker コンテナを起動できる．停止や削除するときも同様のコマンドで行うことができる．本論文で実装する通知や行動履歴表示といった一連のシステムをまとめて起動・制御するために Docker Compose を用いる．

2.2 Node.js

Node.js は JavaScript のランタイム環境であり，Web ブラウザの中でなく単独で JavaScript のプログラムを実行できるようになっているものである [3]．ランタイム環境であることで，従来の JavaScript ではクライアント側の実装でしか使えなかったが，サーバ側でも JavaScript を利用でき，Web 開発の全てを JavaScript のみで完結することができるようになっている．Node.js は Web サーバのための機能を備えており，容易にサーバ側の実装を行うことができる．

2.2.1 npm

npm は Node.js 専用のパッケージマネージャの一種である [3]．Node.js では Node.js の機能を拡張する様々なプログラムがパッケージとして配布されている．その多種多様なパッケージをインストール・管理するシステムとして npm が用意されている．

2.2.2 Express

Express は Node.js のフレームワークの一種で，Node.js の機能をわかりやすくし基本的な Web アプリケーション機能を比較的軽量で提供できる [3].

2.3 Leaflet.js

Leaflet.js は Volodymyr Agafonkin 氏によって作成された JavaScript のマップライブラリである．マーカの表示やポップアップ，地図の拡大縮小など基本的な地図アプリの機能を提供しており，Edge や Chrome，Firefox，Safari などの基本的なブラウザをサポートしている [5].

本論文では畑の地図表示および害獣の行動経路を線として，センサノードの位置をマーカとして図示することに用いる．

2.4 WebPush

WebPush は Web アプリケーションがサーバからメッセージを受信できる仕組みのことである．Web アプリケーションがフォアグランド状態や読み込まれているかどうかに関わらず，リアルタイムな通知を提供することができる．Web アプリケーションが Push 通知のメッセージを送信するには受信側が「サービスワーカー」を登録している必要がある [6].

WebPush の一般的なモデルを 図 2 に示す．WebPush はまず利用者が Push サーバに登録を行い，同時にアプリケーションサーバに登録情報を共有する．この時にサービスワーカーを利用者側に登録する．アプリケーションサーバは共有された登録情報を用いて Push メッセージを Push サーバを経由して利用者に送り，Push 通知を実現する [7].

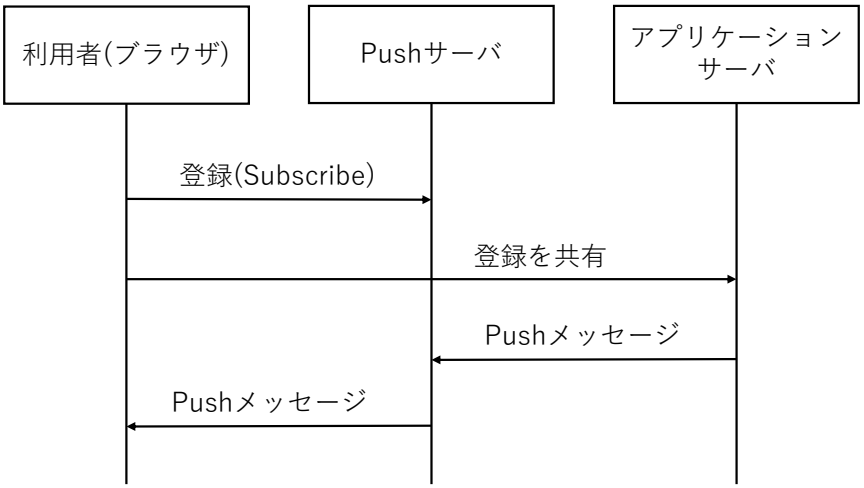

図 2: WebPush の模式図

WebPush は基本的なブラウザで保証されていて， 表 1 に示すブラウザでは保証が確認されている [6].

2.4.1 サービスワーカー (Service Worker)

サービスワーカーはあるオリジン (プロトコル・ポート番号・ホストの 3 つの組) とパスに対して登録されたイベント駆動型の JavaScript ファイルである [8].

表 1: WebPush が保証されているブラウザ [6]

パソコン	スマートフォン
Chrome	Chrome Android
Firefox	Firefox for Android
Opera	Opera Android
Safari	Safari on iOS
Edge	Samsung Internet

2.4.2 VAPID(Voluntary Application Server Identification)

VAPID はアプリケーションサーバを Push サービスに認証・識別してもらうための仕組みである [9]．RFC8030[7] に記載されている WebPush にはアプリケーションサーバを認証する構造が含まれておらず，アプリケーションサーバになりすました通信が行われる問題点がある．ここで VAPID によってアプリケーションサーバの識別をすることで，アプリケーションサーバになりすました通信を防ぐことができるようになる．

2.5 Nginx

Nginx は元々 Igor Sysoev 氏が作成した HTTP サーバやリバースプロキシサーバ，メールプロキシサーバといった汎用的な TCP/UDP プロキシサーバである [10]．本論文においてはクラウドサーバ内のリバースプロキシサーバとして活用する．

3　システム構成

3.1　全体構成

本研究で作成するシステム全体のシステム図を 図 3 に示す．まず，畑内に設置した赤外線センサノード群で動物の侵入を検知し，そのデータをワイヤレス通信の LoRa 通信を用いてゲートウェイに集約する．ゲートウェイは集約されたデータを LTE 通信を用いて Internet 経由でクラウドサーバにデータを送信する．クラウドサーバではデータの管理やデータの解析，農業従事者への通知および解析結果の表示を行う．本論文ではクラウドサーバの解析アルゴリズム以外を担うため，次節でクラウドサーバの構成と詳細について述べる．

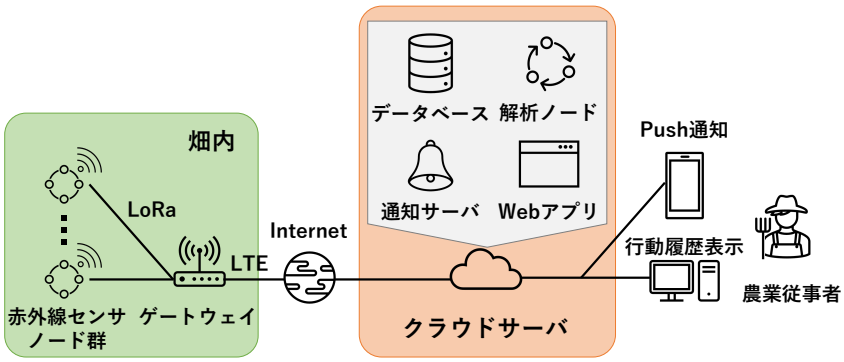

図 3: 全体のシステム図

3.2　サーバ構成

本システムのクラウドサーバは 図 3 にある通り，以下の 4 つの役割の担い，それぞれの役割が連携しながら処理を行う．

- データベース
- 解析ノード
- 通知サーバ
- Web アプリケーション

クラウドサーバの役割が 4 つあり，それぞれが連携するため処理の流れが複雑になっている．クラウドサーバの処理を整理するために DFD(Data Flow

Diagram) を 図 4 と 図 5 に示す．クラウドサーバの処理には畑のデータを送ってくるゲートウェイ，本システムに登録して通知や解析結果を受け取る農業従事者，センサデータや解析データ，農業従事者の登録者情報を保存するデータベースが関わっている．ゲートウェイからは畑で検知したセンサデータがクラウドサーバへ一方的に送信される．クラウドサーバでは送信されてきたセンサデータをデータベースに一時的に保存する．保存したセンサデータを用いて解析を行い，通知と解析後データの保存を行う．通知では事前に登録した農業従事者の登録者情報から Push 通知を行う．また，解析後データから行動履歴表示を行う Web アプリケーションを農業従事者へ提供する．

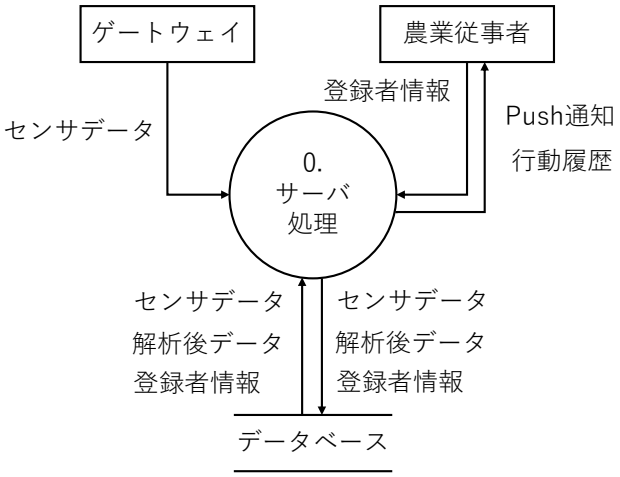

図 4: クラウドサーバの DFD(レベル 0)

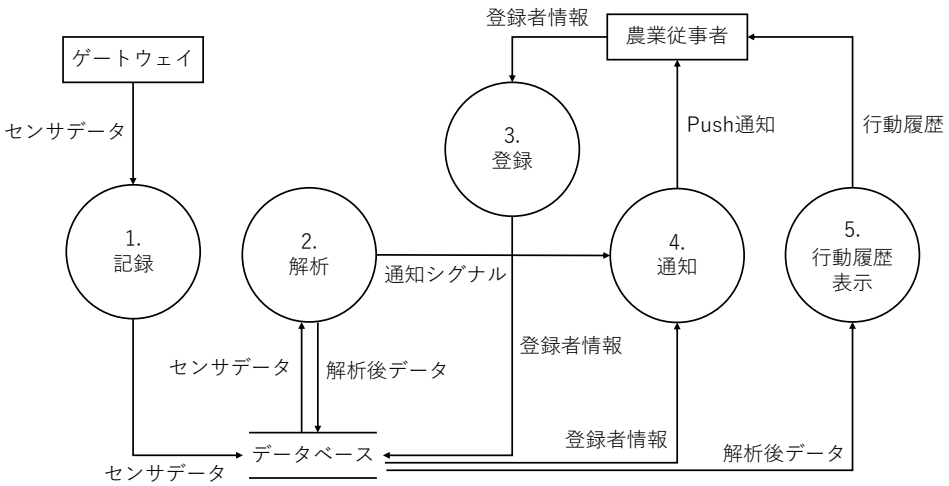

図 5: クラウドサーバの DFD(レベル 1)

クラウドサーバの DFD(図 4 と 図 5) を参考にクラウドサーバ上の構成を決める．本システムのクラウドサーバは文字通り，クラウド上にシステムを設置するため，開発環境 (手元の PC) との OS の違いといった環境差が生じる．そういった環境差を最小限にするために Docker を用いて Docker コンテナ内に各システムを構築する．また，クラウド内の 4 つのシステムを連携させるために Docker Compose を用いる．

Docker Compose を用いて作成するクラウドサーバの内部ネットワークを 図 6 に示す．クラウドサーバ内部に作成する Docker コンテナは以下の 6 つである．また，Docker コンテナごとの詳細については後述する．

- リバースプロキシ (Reverse Proxy)
- 通知サーバ (Notification-app)
- Web アプリケーション (Visualizer-app)
- データベース
  - データベースのアクセス制御 (Database-API)
  - データベース本体 (Database)
- 解析ノード (Analyzer-node)

また，Docker コンテナ間などの通信を行うために Docker ネットワークを以下の 3 つ用意する．

- surface
- inside
- db-bus

surface ネットワークでは Internet と接続する Docker コンテナのみ所属させる．inside ネットワークには基本的な Docker コンテナ全て所属させる．db-bus ネットワークにはデータベース管理の処理を担う Docker コンテナのみ所属させる．

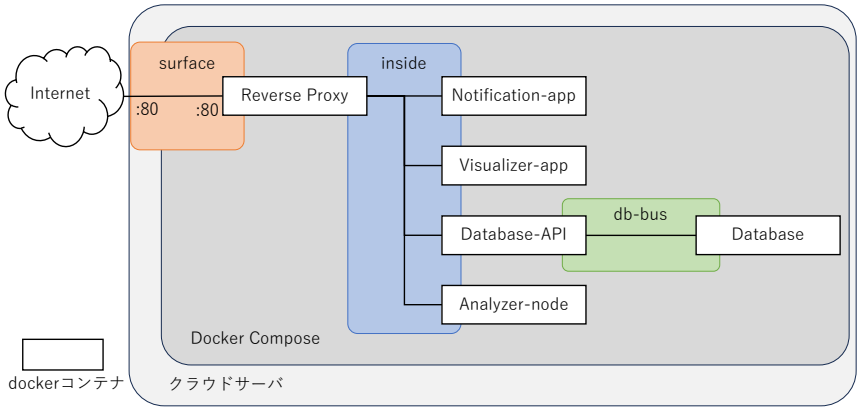

図 6: クラウドサーバのネットワーク図

3.2.1 リバースプロキシ (Reverse Proxy)

クラウドサーバには前述の 4 つの役割があり，それぞれについて Docker コンテナが作成される．しかし，クラウドサーバ内に 4 つのサーバを作成するとなると，ポートの衝突などを回避するために Docker コンテナごとに設定を細かく変える必要が出てくる．細かい設定が増えるとクラウドサーバ全体の設定を把握しづらくなり管理保守することが難しくなる．そこでリバースプロキシを用いて Internet とクラウドサーバ内部の Docker コンテナを仲介・制御する．リバースプロキシは外からのアクセスの種類によって内部のどのサーバと通信するかを制御する機能を持ったサーバのことである．リバースプロキシによって，クラウドサーバ内部に存在する複数機能を担う各 Docker コンテナへのアクセスをまとめて管理する．

実装するリバースプロキシの具体的な仕様としては，クラウドサーバにくる全てのデータアクセス (HTTP 通信) を役割ごとに適当な Docker コンテナに通信するため，surface ネットワークと inside ネットワークに所属する．

Nginx の Docker イメージをベースとし，/etc/nginx/nginx.conf を編集することでリバースプロキシサーバとして Docker コンテナを作成する．

リバースプロキシで行う転送処理は以下の通りである．

/notification： 通知サーバ (Notification-app)

/maps： Web アプリケーション (Visualizer-app)

/nodegw (POST)： データベース (Database-API)

3.2.2 通知サーバ (Notification-app)

通知サーバではレベル 1 の DFD 図 5 にある「4. 通知」で行うため．Push 通知を用いて農業従事者への通知を実現する．まず，Push 通知はスマートフォンやパソコンなどに搭載されている通知方法の一つで，画面を付けていない状態や別のアプリを利用中といった場面でも通知音やバナーといった方法で即時通知することができる．このような常に通知用の管理画面を開く必要がないことや即時性に着目して，通知を Push 通知を用いて実現することにした．しかし基本的な Push 通知の実装はその端末にあったアプリケーションを作成する必要があり，Android や iOS，Windows，MacOS といった対応させたい OS の数だけアプリケーションを作成する必要がある．内部構造が基本的に同じになるとは言っても OS ごとのアプリケーションを作成するのはコストが高く，バージョンアップといった保守の手間も非常に大きい．

そこで, WebPush というものを活用する．WebPush はブラウザを用いた Push 通知を行う仕組みであり, 一つの Web サーバを作成するだけで対応するブラウザを介して複数種類の端末で Push 通知を実現することができる．また WebPush は基本的なブラウザに対応しているため，簡単に複数種類の端末への Push 通知が実現することが可能である．

したがって，通知サーバでは登録された畑に動物の侵入があったことを Web-Push を用いて農業従事者に通知するサーバを実現する．具体的には WebPush で VAPID を用いてサーバの登録を行い，Node.js の Docker イメージをベースとして WebPush を行うサーバを実現する．

通知サーバはレベル 1 の DFD(図 5) においては「4. 通知」を主に担う．そのため「2. 解析」からの通知シグナルとデータベースから登録者情報を取得するために Docker ネットワークで Analyzer-node および Database-API と同じ inside に所属する．

3.2.3 Web アプリケーション (Visualizer-app)

Web アプリケーションではレベル 1 の DFD 図 5 にある「3. 登録」と「5. 行動履歴表示」を行う．行動履歴表示では特に侵入した動物の侵入経路の分かりやすさが重要である．そこで Leaflet.js というマップライブラリを用いる．Lealfet.js は地図の拡大縮小やマーカの表示など基本的な地図アプリの機能に加えて，線や多角形領域，ピンといったマーカを地図に投影することができるため採用する．

加えて，Web ページとして JavaScript(Leaflet.js) を利用するため，JavaScript の Node.js を用いることで利用する言語を統一し，管理を容易にする．

また，本システムは複数の農業従事者および畑を対象としてサービスを提供するため，どこの畑で，誰が管理者であるかを登録する必要がある．登録ページについて開発中のページを 図 7 に示す．

図 7: 開発中の登録ページ

Web アプリケーションの具体的な実装としては，Node.js の Docker イメージをベースとして HTML や CSS，Leaflet.js を含めた JavaScript を用いて実現する．Web アプリケーションはレベル 1 の DFD(図 5) においては「3. 登録」と「5. 行動履歴表示」を主に担う．そのためデータベースへの登録者情報の保存および解析後データの取得するため，Docker ネットワークで Database-API と同じ inside に所属する．

3.2.4 データベース (Database, Database-API)

データベースでは MySQL を用いたデータベースコンテナと Node.js を用いたデータベースコンテナ管理用の API コンテナの 2 つで構成する．API コンテナでデータベースアクセスを仲介することで，データベースアクセスの簡易化や誤操作の抑制を実現する．

Docker ネットワークとしては db-bus ネットワークに所属して，他の Docker コンテナからデータベースアクセスを受けるために Database-API コンテナのみ inside ネットワークにも所属する．

3.2.5 データ解析ノード (Analyzer-node)

データ解析ノードでは python イメージを主体としてデータベースに蓄えられる赤外線センサのデータを解析する役割を持つ．本コンテナでは Database-API

コンテナと通信してデータの取得および解析後のデータの保存を行い，通知サーバに通知シグナルを送信する．そのため，Docker ネットワークとしては inside ネットワークに所属する．

解析の詳細に関しては本論文範囲外であるため割愛する．

4　検証実験

4.1　実験方法

4.1.1　ゲートウェイの LTE モジュールとの疎通実験

ゲートウェイが使用する LTE 通信モジュールとクラウドサーバ間での POST 形式での通信が可能か確認する．

実験手順としてはゲートウェイに搭載予定の LTE モジュールから Internet を経由させて用意したサーバにデータを送信し，通信を確認する．LTE モジュールから送信は POST 形式で HTTP 通信を用いて行う．また，送るデータは JSON 形式で送信し，データ内容としては 表 2 に示す．

表 2: LTE 通信する JSON データ

データ名	データフォーマット	注釈
timestamp	文字列 (YYYYMMDDHHMMSSSSS)	検出時刻 (年月日, 時分秒 [ms])
node	数値	センサノード番号
dir	数値	ノード内のセンサ番号
sd	(任意長)	センサデータ (サンプリングデータ)

4.1.2　通知サーバの WebPush 実験

クラウドサーバの構成要素である通知サーバ (Notification-app) から WebPush による通知ができるかどうかを確認する．

実験手順としては以下の通りである．

1. Docker Compose でサーバを起動する．
2. ブラウザアプリで起動したサーバにアクセスする．
3. アクセス時に通知を受け取るかの確認が出るため，それを許可する．
4. サーバ側から WebPush で Push 通知を送る．
5. ブラウザアプリを通じて通知がくるかを確認する．

4.1.3　Web アプリケーションの行動履歴表示実験

クラウドサーバの構成要素である Web アプリケーション (Visualizer-app) の行動履歴表示ができるかを確認する．

実験内容としては，ダミーで用意した畑の位置情報，センサノード位置情報，経路情報について正しく表示できているかを Web ページにアクセスして確認する．ダミーで用意したデータについては 表 3 に示す．

表 3: ダミーデータ

データ種類	緯度	経度	注釈
畑位置	34.64801074823137	135.75705349445346	奈良高専の校庭
センサ 1	34.64746793595177	135.75796544551852	畑の南東
センサ 2	34.64801074823137	135.75796544551852	畑の東
センサ 3	34.64852707856505	135.75796544551852	畑の北東
センサ 4	34.64852707856505	135.75705349445346	畑の北
センサ 5	34.64852707856505	135.75623810291293	畑の北西
センサ 6	34.64801074823137	135.75623810291293	畑の西
センサ 7	34.64746793595177	135.75623810291293	畑の南西
センサ 8	34.64746793595177	135.75705349445346	畑の南
経路情報	34.64879186210419	135.75744509696963	1 番目の位置
	34.64826229418025	135.75744509696963	2 番目の位置
	34.64828435957796	135.75665116310122	3 番目の位置
	34.64780333257661	135.75666189193728	4 番目の位置
	34.64724286640332	135.75664043426517	5 番目の位置

4.2　実験結果

4.2.1　ゲートウェイの LTE モジュールとの疎通実験

ゲートウェイとの LTE 通信を受信した結果を 図 8 と 図 9 に示す．

```
--- server up ---
{"timestamp":"20240123135017200","node":"1","dir":"1","sd":"1234567890"}
```

図 8: サーバの受信ログ

図 9: サーバの受信データ (body 情報の一部抜粋)

図8 にあるように,「{“timestamp”:“20240123135017200”,“node”:“1”,“dir”:“1”,“sd”:“1234567890”}」というデータを受け取ったことがわかる．また，図9 より「methods: { post: true }」のように POST 形式で送られていることがわかる．加えて,「[Symbol(kHesaders)]」の項目から「content-length」は 54,「content-type」は ‘application/x-www-form-urlencoded’ であることがわかる．(「host」の伏せてある部分はクラウドサーバのホスト名である．)

「content-length」は body の波括弧 ({}) と空白スペース，クォーテーションを除いた数である 54 文字と一致するため，body のデータサイズと考えられる．また,「content-type」の ‘application/x-www-form-urlencoded’ は HTML フォームでデータを送信した時と同じタイプであるため [11]，ゲートウェイからの通信は HTML フォームによるアクセスと同義に扱えると考えられる．

4.2.2　通知サーバの WebPush 実験

通知サーバの通知結果を 図 10 と 図 11　に示す．今回の実験では PC 版の Google Chrome(バージョン:120.0.6099.234（Official Build）（x86_64）) をブラウザとして利用した．

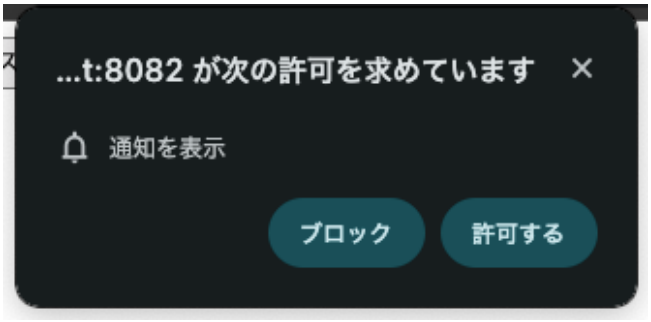

図 10: WebPush の通知許可への確認ダイアログ

図 11: WebPush の通知

図 10 は WebPush を実施するアプリケーションサーバにアクセスした時のダイアログである． 図 10 の選択肢で「許可する」を選択することで，WebPush の通知が受け取れるようになる．「許可する」を選択後，サーバに WebPush 送信用のメソッドを起動するようにシグナルを送ると，図 11 がデスクトップに表示された． 図 11 の通り，Google Chrome からの通知であることがわかり，Push 通知をクリックすると Google Chrome に遷移させられたため，Google Chrome による Push 通知だと判断できる．

4.2.3　Web アプリケーションの行動履歴表示実験

Web アプリケーションの行動履歴表示について 表 3 を表示した Web ページを 図 12 に示す．

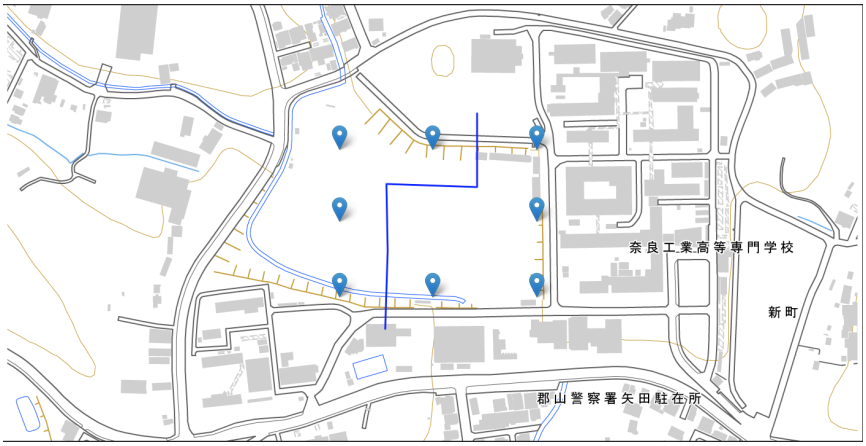

図 12: Web アプリケーションの行動履歴表示 (ダミーデータ)

図 12 より，奈良高専の校庭を中心とした 8 つのセンサノード (ピン) と，行動経路 (青い折れ線) が描画されていることがわかる．そのため正しくダミーデータ (表 3) 通りの描画が行えていると判断できる．

4.3 まとめ

本章の検証実験により，ゲートウェイとの疎通確認および通信方法の特定，通知サーバによる WebPush の通知の確認，Web アプリケーションによる行動履歴表示の確認が行うことができた．

本論文ではできなかったがシステムとして効果を確認するためには，赤外線センサノードおよびゲートウェイを実際の畑に設置し，動物が畑に侵入した時の実際のセンサデータを取得，クラウドサーバへ送信したのち，行動解析アルゴリズムを用いて侵入した動物の行動を推定，そうして得られたデータから PC やスマートフォンに対して WebPush による通知，ブラウザを通した行動履歴の表示する，という一連の流れを検証する必要がある．

5　あとがき

本論文では害獣検出と行動解析を行うシステムの中でクラウドサーバに関する部分の設計・実装・動作確認について述べてきた．畑に設置予定のゲートウェイとのデータ通信や通知アプリの動作検証，Web アプリケーションによる行動履歴の表示といった，クラウドサーバとして必要な要素についての設計・検証を行ってきたが，まだ試験的な部分が大きく，実際に利用するには至らない部分が多い．特に優先的に対処しなければならないこととして以下に示すことが挙げられる．

- セキュリティ対策
- UI・UX の向上
- サーバの冗長性

5.1　セキュリティ対策

クラウドサーバではゲートウェイからの受信に HTTP 通信を用いている．そのため，データ通信が十分に暗号化されておらず，通信中に盗み見られることや悪意を持った攻撃者がデータを装って POST 送信をしてくることが考えられる．これに関してはゲートウェイに搭載できるモジュールやマイコンの性能に左右されるため，ゲートウェイ設計担当である共同研究者とも話し合っていく必要があると考えられる．

また，クラウドサーバへのアクセスは基本的にクラウドサーバ自体のファイアウォールや Docker Compose 内のリバースプロキシ，利用するクラウドサービス独自のセキュリティシステムを考えている．本論文のシステムではクラウドサーバ内で農業従事者の情報を入力し，登録するフォームを設けるなどそれらの個人情報を取り扱うこととなるため，十分に堅牢なシステムが必要だと考えられる．そのためデータ管理や通信手段を HTTPS などで暗号化するなど，登録方法を別の手段を利用して本クラウドサーバ上で個人情報のやり取りを最小限にするなどの対策することが重要である．

5.2 UI・UX の向上

本論文のクラウドサーバは実験結果の通り，機能を確かめるための仮組みの設計でしかない．そのため対象ユーザが農業従事者であり，基本的にパソコンなどの操作に慣れていない可能性を考慮する必要がある．

5.3 サーバの冗長性

本クラウドサーバは常時，畑に設定したゲートウェイからセンサ情報を受け取ることが予想される．そのため，登録される畑が増えればクラウドサーバへのデータ送信は肥大化していく．特に対象としているのが夜間の獣害被害であるため，そのデータ送信も夜間に集中する可能性がある．そうなったときに，本論文のクラウドサーバ構成は通信制御を行うリバースプロキシとデータベースがボトルネックとなることが考えられる．

そのような場合に備えて将来的にはボトルネックとなる，Docker コンテナを複数個並列して用意することや負荷を分散させる機構 (kubernetes のようなロードバランサー) を用いることなどが必要になると考えられる．

参考文献

[1] 農林水産省, “野生鳥獣による農作物被害状況の推移”, https://www.maff.go.jp/j/seisan/tyozyu/higai/hogai_zyoukyou/attach/pdf/index-31.pdf, 2024 年 1 月 18 日参照.

[2] 大澤文孝, 浅井尚, “触って学ぶクラウドインフラ docker 基礎からのコンテナ構築”, 日経 BP マーケティング, 2020 年.

[3] 掌田津耶乃, “Node.js 超入門”, 株式会社 秀和システム, 2017 年.

[4] StrongLoop, IBM, “Express - Node.js web application framework”, https://expressjs.com/, 2024 年 1 月 18 日参照.

[5] Volodymyr Agafonkin, “Leaflet - a JavaScript library for interactive maps”, https://leafletjs.com, 2024 年 1 月 18 日参照.

[6] Mozilla Foundation, “プッシュ API - Web API | MDN”, https://developer.mozilla.org/ja/docs/Web/API/Push_API, 2024 年 1 月 18 日参照.

[7] M. Thomson, E. Damaggio, B. Raymor, Ed., “Generic Event Delivery Using HTTP Push”, https://datatracker.ietf.org/doc/html/rfc8030, RFC8030, December 2016, 2024 年 1 月 21 日参照.

[8] Mozilla Foundation, “サービスワーカー API - Web API | MDN”, https://developer.mozilla.org/ja/docs/Web/API/Service_Worker_API, 2024 年 1 月 18 日参照.

[9] M. Thomson, P. Beverloo, “Voluntary Application Server Identification (VAPID)”, https://datatracker.ietf.org/doc/html/rfc8292, RFC8292, November 2017, 2024 年 1 月 21 日参照.

[10] Nginx, “nginx”, https://nginx.org/en/, 2024 年 1 月 18 日参照.

[11] Mozilla Foundation, “POST - HTTP | MDN”, https://developer.mozilla.org/ja/docs/Web/HTTP/Methods/POST, 2024 年 1 月 22 日参照.