Análise de campos agrículas feito por drones

Universidade de Aveiro

João Torrinhas, Diogo Torrinhas,

Análise de campos agrículas feito por drones

Departamento de Electrónica, Telecomunicações e Informática

Universidade de Aveiro

João Torrinhas, Diogo Torrinhas

(98435) joao.torrinhas@ua.pt, (98440) diogotorrinhas@ua.pt

June 20, 2023

Contents

Chapter 1

Introdução

Este relatório tem como objetivo descrever todos os passos e processos realizados durante o desenvolvimento deste projeto no âmbito da unidade curricular de Redes e Sistemas Autónomos.

O código desenvolvido para o projeto encontra-se em:

https://github.com/JoaoTorrinhas/Drones-Scanning

Segue em anexo com este relatório um **vídeo** que mostra o resultado da simulação.

1.1 Objetivo

O principal objetivo deste trabalho é desenvolver uma rede veicular que permita a troca de informação, entre drones/OBUs, de zonas livres/ocupadas no campo agrícula, posição dos drones/OBUs com o objetivo de os drones procurarem por zonas livres no campo agrícula para as analisarem e posteriormente passarem essa informação aos restantes drones. No fundo, os drones, através da troca de mensagens, vão ter a informação sobre as posições de cada um deles e vão conseguir interpretar quais zonas estão livres/ocupadas, de maneira a que caso um dos drones esteja a ir para um zona ocupada ele imediatamente vai procurar uma zona livre para, posteriormente, ser analisada.

Esta ideia iria evitar, principalmente em dias de grande calor, que os agricultores saissem de casa para ver quais as áreas do campo agrícula precisam de ser regadas, facilitando os seus trabalhos e até mesmo preservando um pouco das suas saúdes. Deste modo, eles apenas tinham de aguardar o resultado da pesquisa, feita pelos drones, e em seguida irem até às áreas, obtidas pela pesquisa, regar o solo agrícula.

No desenvolvimento do projeto identificou-se uma identidade:

- **OBUs (On Board Units)-** No contexto do projeto, são equipamentos para comunicação wireless colocados nos drones para enviarem mensagens para outras **OBUs** a informar as suas posições e outra informação adicional.

A partir da identidade referida acima, identica-se um tipo de comunicação no contexto de redes *VANET* (Vehicular Ad Hoc Networks):

- **Vehicle-to-Vehicle(V2C)-** Comunicação direta entre veículos (drones no contexto deste projeto) que contém informação como velocidade, direção e as suas respetivas características.

1.2 Mensagens

Para comunicar entre os vários drones presentes na rede foi necessário usar dois tipos de mensagens com formatos pré-definidos: **CAMs** e **DENMs**.

1.2.1 CAMs

CAMs (Cooperative Awareness Messages) são mensagens usadas para informar os drones sobre a posição de cada um deles e, com essa informação, conseguirem interpretar quais são as áreas do campo agrícula livres/ocupadas. Contém informações tais como velocidade, posição do drone, id da estação, direção, tipo de veículo, etc. São enviadas periodicamente enquanto o drone está em movimento.

1.2.2 DENMs

DENMs (Decentralized Environmental Notification Messages) são mensagens usadas para sinalizar outros nós da rede veicular sobre eventos presentes na estrada tais como, um engarrafamento ou um acidente. Não são geradas periodicamente, são geradas na ocorrência de um evento e têm um tempo de validade. No caso deste projeto, elas foram usadas com o propósito de notificar os outros drones com o resultado da análise de uma área do campo agrícula.

Para adequar a **DENM** à situação deste projeto, foi usado um novo valor de *CauseCode* para indicar se a zona precisa de ser regada: *CauseCode=1* significa que a zona do campo precisa de ser regada, *CauseCode=0* significa que a zona do campo **não** precisa de ser regada.

1.3 Tecnologias

Para a criação desta rede veicular foi utilizada a extensão *NAP-Vanetza* (https://code.nap.av.it.pt/mobility-networks/vanetza) que integra *MQTT* e *JSON*. Esta extensão permite implementar a comunicação com mensagens no formato *ETSI C-ITS* entre os vários nós da rede, lidando com processamento, envio e receção destas mensagens.

A linguagem de programação usada para criar o processo de emulação e para integrar cada um dos scripts foi *Python*.

Foi utilizada a biblioteca *SQLite3* para a criação das bases de dados por ser uma ferramenta simples e ideal para este projeto.

Para a consttrução do mapa da simulação na página web foram usadas as bibliotecas *Javascript*, *Leaflet*. Além dessas bibliotecas, foram usadas as tecnologias *HTML* e *CSS* para servirem como base ao mapa apresentado e *Flask* para o servidor

1.4 Metodologia Seguida

O primeiro passo para implementarmos este projeto foi estudar o sistema *vanetza* assim como o *mqtt* para perceber como é que iam ser feitas as comunicações entre as *OBUs* presentes na rede.

Depois de termos percebido como é que o *vanetza* funciona, criámos a primeira simulação simples onde apenas tinhamos duas *OBUs* que, à medida que iteravam até um ponto destino, enviavam *CAMs* com a posição de cada *OBU*.

A partir desta simulação inicial, adicionámos mais uma *OBU*, quatro zonas no mapa para aumentar a complexidade e criámos um programa para analisar as imagens captadas pelos drones. Em seguida, foi criado uma espécie de algoritmo de decisão onde as *OBUs* sempre que se aproximam de uma zona verificam, numa base de dados geral, (base de dados para todas as *OBUs* com a informação das zonas, entre outras) se já está ocupada por outra *OBU* e, caso esteja, procura uma nova zona mais próxima de si. Ao mesmo tempo, foi também criada uma interface web onde é possível observar todo o processo durante a simulação.

Por fim, adicionámos mais uma zona no mapa e, de maneira ao processo da escolha de zonas por parte das *OBUs* não estar tão centralizado, eliminámos essa base de dados geral e associámos a cada *OBU* uma base de dados, de maneira a cada uma delas ser indepedente das outras e a obrigar as *OBUs* a comunicarem entre si para atualizarem a informação das suas bases de dados.

Chapter 2

Sistema/Implementação

2.1 Arquitetura Geral do Sistema

Depois de várias discussões e estudos realizados para compreender o projeto, desenvolvemos a arquitetura representada na Figura 2.1, com a seguinte análise:

- A parte mais importante do projeto, está relacionada com a estrutura do protocolo de rede veicular que implementa toda parte relacionada com a comunicação entre os drones e que foi desenvolvida usando tecnologias *Python* em conjunto com o *Vanetza* e *MQTT*.
- Uma aplicação web desenvolvida em *HTML*, *CSS* e com *Leaflet* para aprensentação do mapa. Envia periodicamente um conjunto de requests ao servidor web, desenvolvido em flask, para atualizar/mostrar informação na sua página web.
- Contrução de várias bases de dados (uma relaciona com as posições das *OBUs* e três relacionadas com as zonas presentes no mapa, para cada *OBU*) que suportam todo o processo de emulação e que ajudam os drones a tomar decisões.
- Scripts desenvolvidos em *Python* para gerar o processo de simulação e definir o comportamente das entidades presentes na rede.

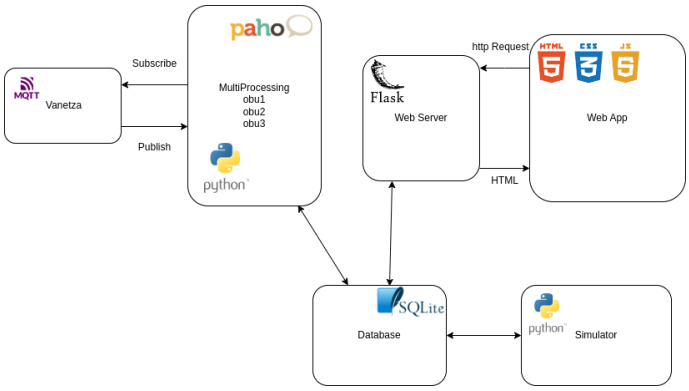

Figure 2.1: Arquitetura Geral do Sistema.

2.2 Arquitetura dos Drones/OBUs

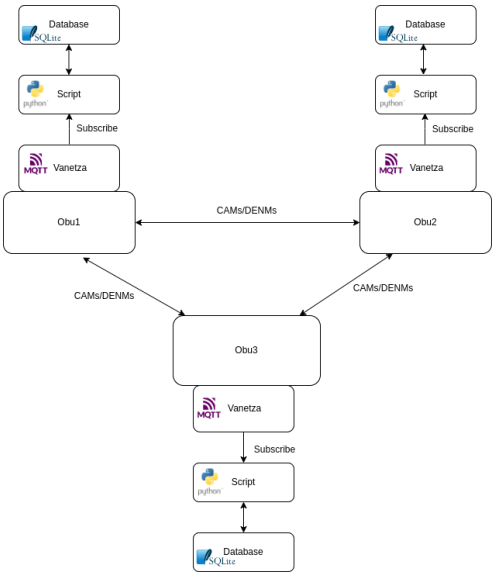

Figure 2.2: Arquitetura das OBUs do Sistema.

Na Figura 2.2, está presente a arquitetura de cada *OBU*/Drone. Cada uma vai ter a ela associada um script: para tomar decisões (relativamente a quais zonas estão livres, qual a próxima zona a ser analisada, quais zonas estão ocupadas entre outras), *Vanetza*, com a ajuda do *MQTT*, para ouvir/enviar as

CAMs e *DENMs* de outras *OBUs*.

Além do referido acima, cada *OBU* tem uma base de dados, totalmente independente das outras, onde armazena e atualiza a informação das zonas (zonas livres/zonas ocupadas, coordenadas das zonas, zonas que precisam ou não de ser regadas) durante todo o processo de simulação.

2.3 Brokers e Vanetza

O protocolo de rede veicular corre sobre vanetza. São lançados um conjunto de containers *Docker* onde cada um representa uma das entidades de rede (três containers para três *OBUs*). Para a comunicação entre containers foi usada uma rede virtual do *Docker* denominada por *vanetzalan0* com ip 192.168.98.0/24.

No ficheiro *docker-compose.yml* conseguimos definir o IP, ID, endereço MAC e o tipo de entidade para cada *OBU*. Em abaixo, são apresentados os IP, ID's e endereço MAC para cada *OBU*:

- **OBU1 -> ip-** 192.168.98.20, **id-** 2, **MAC-** 6e:06:e0:03:00:02
- **OBU2 -> ip-** 192.168.98.30, **id-** 3, **MAC-** 6e:06:e0:03:00:03
- **OBU3 -> ip-** 192.168.98.40, **id-** 4, **MAC-** 6e:06:e0:03:00:04

Cada *OBU* faz subscrição dos tópicos *"vanetza/out/cam"* e *"vanetza/out/denm"* para tomar decisões e atualizar cada uma das suas bases de dados. Além disso, cada uma também vai publicar mensagens no tópico *"vanetza/in/cam"* e *"vanetza/in/denm"* para enviar *CAMs* às outras *OBUs* com a informação da sua posição atual e para trocarem *DENMs* entre eles com a informação de rega dos campos.

2.4 Ficheiros usados na simulação

Os seguintes ficheiros foram desenvolvidos para a criação de todo o processo de emulação:

- **Obu_simulation.py -** Contêm a inicialização dos Drones/OBUs. Foram utilizadas threads para cada *OBU* que são lançadas concorrentemente e efetuam várias chamadas ao ficheiro *driving.py* (iterar os drones até um determinado destino), e efetua também várias chamadas ao script de cada OBU (tomar decisões). Além disso é neste ficheiro que são enviadas DENMs quando um drone termina de analisar uma determinada zona.
- **driving.py -** Contêm a função que permite a iteração das posições de um drone até um determinado destino.

- **script_obu*.py -** Contêm um conjunto de funções que permite tomar decisões (vê se uma zona está livre/ocupada, procura novas zonas, atualiza informações da BD) de acordo com as *CAMs* e *DENMs* que recebe de outras *OBUs*.

- **zones_obu*_db.py -** Criação das bases de dados das zonas e gerenciamente das mesmas (Adicionar, alterar, retornar, eliminar informação da base de dados).

- **obu_db.py -** Criação da base de dados obu (contêm a informação das coordenadas para cada *OBU*) e gerenciamente das mesmas.

- **utils.py -** Contêm uma função usada para verificar se uma determinada *OBU* está próxima de uma zona. Tem como parâmetros as coordenadas da OBU, as coordenadas da zona destino, e um *threshold* para comparar com a distância da OBU a essa zona que é calculada com recurso à biblioteca *geographiclib*. Por exemplo, se o *threshold* é de 15 metros e a *OBU* está 13 metros da zona, então a função retorna *True* a indicar que a *OBU* está próxima dessa zona. A Figura 2.3 representa esta função.

- **image_analyzer.py -** Contêm um conjunto de funções usadas para analisar uma imagem. A análise baseia-se na percentagem de pixeis verdes, amarelos e castanhos. Quanto maior for a percentagem de pixeis verdes significa que a probabilidade de o drone ter capturado uma imagem de um campo verde é grande e assim sucessivamente.

```
def check_proximity(obu_lat, obu_lon, zone, proximity_threshold):
    geod = Geodesic.WGS84
    result = geod.Inverse(obu_lat, obu_lon, zone[0], zone[1])
    distance = result['s12']
    return distance <= proximity_threshold
```

Figure 2.3: Função que verifica se uma OBU está próxima de uma Zona.

2.5 Processamento de Imagem

No processamento das imagens captadas pelos drones, cada imagem é passada como parâmetro numa função. Essa mesma função vai procurar por pixeis com a cor verde, amarela e castanha (cores mais abundantes num campo agrícula).

Em seguida, é calculado o número total de pixeis e o número de pixeis para cada cor com o objetivo de calcular a percentagem de pixeis verdes, amarelos e castanhos na imagem.

De acordo com o resultado dessa percentagem, para cada cor referida acima, é determinado se essa zona do campo, captada pelos drones, precisa ou não de ser regada. Na Figura 2.4 está presente um exemplo de imagem a ser analisada.

Figure 2.4: Imagem a ser analisada.

E na Figura 2.5 está presente o resultado da análise. Tal como era esperado, a percentagem de pixeis verdes é muito superior comparado com a percentagem dos pixeis das outras cores, por isso é possível concluir que a região do campo captada pelo drone não precisa de ser regada.

```
The percentage of green pixels in the image is:  85.28960905349794
The percentage of yellow pixels in the image is:  14.512345679012345
The percentage of brown pixels in the image is:  2.9495884773662553
```

Figure 2.5: Resultado da análise

2.6 Decisões dos Drones

Cada drone no ínicio da simulação vai para a zona mais próxima a ele e à medida que eles estão a iterar até à zona, eles comunicam entre si para saberem as posições de cada um. Caso um deles esteja mais próximo de uma zona (15 metros), através das mensagens trocadas entre eles, os outros drones vão ter conhecimento que ele é o que está mais próximo dessa zona e vão atualizar a sua base de dados ($zones\_db$) e, caso outro drone também tivesse a ir para essa zona, além de atualizar a sua base de dados vai mudar a sua rota e escolhe outra zona que esteja mais próximo de sí.

No fim, depois de todas as zonas terem sido analisadas, os drones retornam às suas posições iniciais.

O algoritmo da Figura 2.6 abaixo mostra a explicação do processo de decisão dos drones.

Algorithm 1 Algorithm for drones decision

Drones Decision:

```
1: While (still_zones_to_analyse) do
2:     While (reached_a_zone == False) do
3:         choose closest zone
4:         reached_a_zone =iterate to chosen zone  //return true if reach's a zone or false if it is already occupied
5:     end While
6:     if (reached_a_zone) then
7:         analyse the zone()
8:         send DENM()
9:     end if
10:    if (obu_check_zones_not_occupied() == []) then
11:        still_zones_to_analyse = False
12:    else
13:        reached_a_zone = False
14:    end if
15: end While
```

Figure 2.6: Algoritmo de Decisão.

2.7 Aplicação Web

Para correr a aplicação web, apenas é preciso correr o programa *app.py* e abrir o URL *http://127.0.0.1:5000/*.

A estrutura e estilo da página foram construídas com *HTML*, *CSS* e *Javascript*, sendo que o mapa interativo foi desenvolvido com recurso à biblioteca *Leaflet*.

O acesso às base de dados é feito do lado do servidor para obter informação das posições periodicamente atualizadas das OBUs e também informação das zonas que precisam de ser regadas. Com ajuda da tecnologia *Ajax* são feitos periodicamente *http requests* ao servidor, de forma a obter as tais posições atualizadas que posteriormente vão ser mostradas no mapa através de um ícone (um para cada drone) e também para obter a informação das zonas que precisam de ser regadas que posteriormente vão ser mostradas em *box's* na página web (três *box's* para cada um dos três drones).

No mapa também vai aparecer cinco marcadores, um para cada zona. No capítulo **Simulação** é possível ver como é o design da aplicação web.

2.8 Bases de Dados

A base de dados é composta por 4 tabelas: OBU e três tabelas Zone para cada um dos Drones/*OBUs*.

A tabela OBU vai armazenar o endereço ip da *OBU*(chave primária) e a latitude e longitude da mesma que periodicamente vão sendo atualizadas à medida que as posições do drone vão sendo iteradas até um determinado destino. Estas posições vão servir para mostrar os movimentos dos drones na aplicação web.

A tabela *Zone* vai armazenar o ip e broker da *OBU* que analisou essa zona, a informação se a zona precisa ou não de ser regada, a latitude e longitude da zona e, por fim, o id da zona (chave primária). Cada *OBU* vai ter uma tabela *Zone* associada a si, para guardar as informações sobre ela e as informações que recebe das restantes *OBUs* (troca de mensagens).

Na Figura 2.7 está presente um diagrama das bases de dados usadas neste projeto.

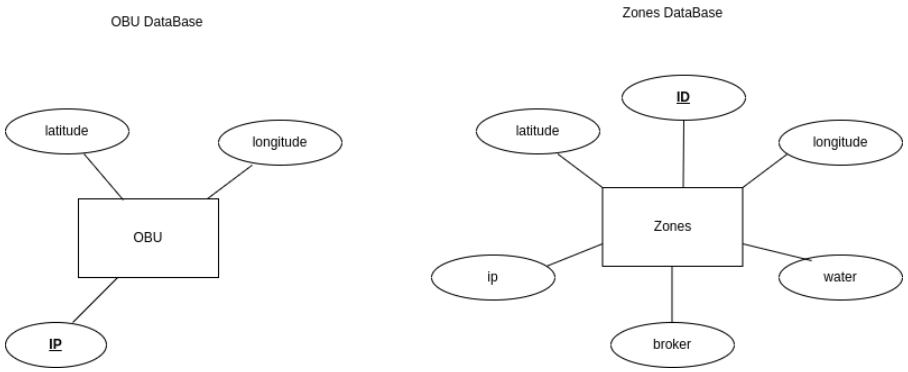

Figure 2.7: Diagrama das Bases de Dados.

Chapter 3

Simulação

Neste capítulo vai ser explicado todo o processo, decisões e mensagens trocadas durante a simulação.

3.1 Condições de simulação

3.1.1 Condições Iniciais

Cada drone vai ter uma posição inicial e vai haver 5 zonas no mapa (pode haver mais zonas) onde a cada zona vai estar associado uma imagem predefinida de um campo agrícula de maneira a simular as imagens capturadas por um drone numa situação real. Tal como mostra a figura 3.1, no mapa estão representandos os drones e as zonas que podem ser analisadas (marcadores vermelhos). É possível alterar as coordenadas iniciais dos drones de maneira a ser possível ver trajetos diferentes dos mesmos sempre que se inicie uma nova simulação.

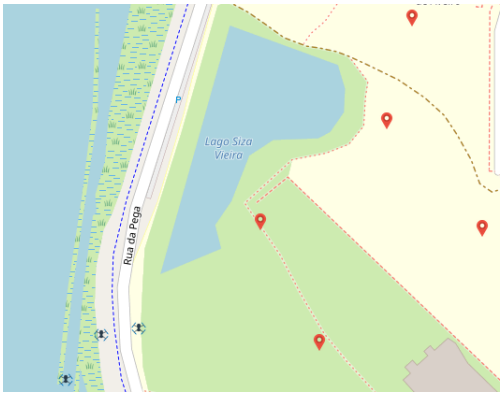

Figure 3.1: Mapa da simulação.

3.1.2 Condições Esperadas

É esperado que ao longo da simulação todos os drones comuniquem entre si de maneira a não visitarem zonas que já tenham sido analisadas e que não analisem a mesma zona ao mesmo tempo. No fim, é expectável que todos os drones voltem à posição inicial quando já não houver mais zonas para serem analisadas e também que nenhuma zona tenha sido analisada por mais do que um drone.

3.2 Interação e processamento de mensagens

As *OBUs*, enquanto estão a ir para um certo destino, vão enviar (publish) *CAMs* periodicamente a informar a sua posição atual para as outras *OBUs* e ouvem (subscribe) atentamente as mensagens das outras *OBUs* para posteriormente processarem essa informação recebida.

Sempre que uma *OBU* estiver a pelo menos 15 metros de uma zona (função *check_proximity* com um *threshold* de 15 metros (pode ser mudado), ver capítulo anterior), ela vai atualizar a sua base de dados *Zones* a indicar que esta zona pertence a ela.

Uma *OBU*, ao receber uma *CAM* vai ver a posição da *OBU* que enviou essa *CAM* (a latitude, longitude e ID da *OBU* que a enviou estão presentes na *CAM* que é representada num *JSON object*) e vai comparar essa posição com as coordenadas das zonas (todas as *OBUs* têm guardado na base de dados todas as posições das zonas a serem analisadas). Caso essa posição esteja a pelo menos 15 metros de uma zona, a *OBU* que recebeu essa *CAM* vai atualizar a base de dados *Zones* a indicar que essa zona está ocupada pela *OBU* que a enviou.

Após a análise da zona, a *OBU* atualiza a sua base de dados *Zones*, envia uma *DENM* com o resultado da análise e procura uma nova zona livre para analisar, caso não haja mais zonas para análise volta à posição inicial.

As outras *OBUs* ao receberem uma *DENM* vão ver qual foi a *OBU* que a enviou e procuram na sua base de dados a zona que está a ser analisada por essa *OBU* (sabem com antecedência qual é a zona que a *OBU* está a analisar por causa das *CAMs* e do *threshold*, tal como foi explicado anteriormente) e atualizam o campo *water* com o resultado da análise para, posteriormente, ser mostrado na aplicação web o resultado da análise das zonas feita por cada *OBU*.

Na Figura 3.2 está presente um exemplo onde é possível verificar o que foi explicado acima. Os três drones dirigiam-se para a mesma zona mas, através das *CAMs* trocadas entre eles, tomam conhecimento que o drone mais à frente está a pelo menos 15 metros da zona para onde eles iam, então eles individualmente atualizam a sua base de dados *Zones* e procuram uma nova zona mais próxima a eles para analisarem.

Figure 3.2: Exemplo de movimento das OBUs.

3.3 Análise das Zonas

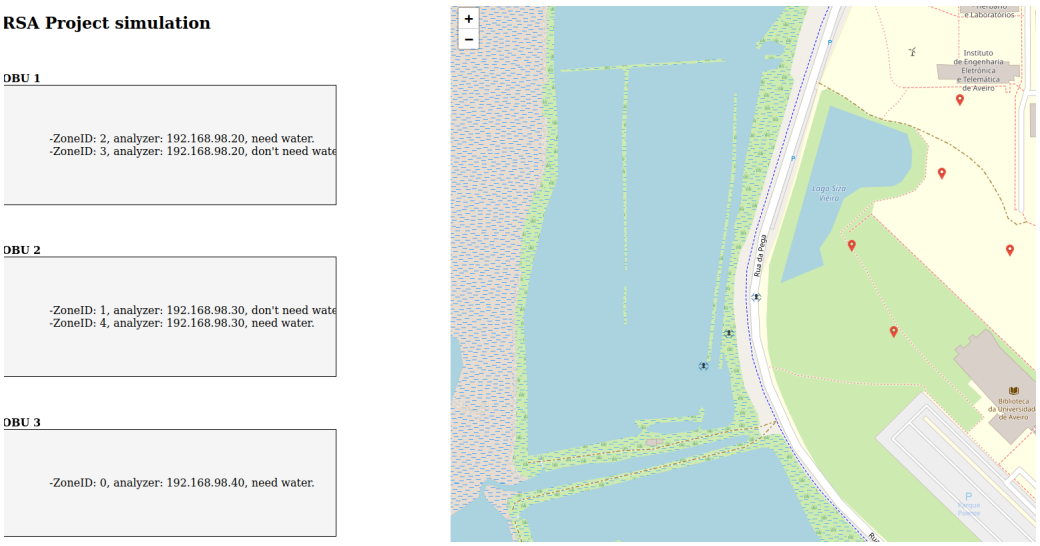

Figure 3.3: Resultado final após simulação (Exemplo).

Como já foi dito anteriormente, durante a simulação as *OBUs* vão escolher uma zona para analisar e a essa zona vai estar associada uma imagem (suposta imagem da zona capturada pelo drone), tal como mostra a Figura 3.4. O ID da zona vai estar associado ao index na lista evidenciada na imagem abaixo.

```
obus_cameras = ["static/seca.png", "static/verde2.png", "static/seca3.png", "static/verde1.jpg", "static/seca2.jpg"]
```

Figure 3.4: Lista com as imagens de cada zona.

Olhando para o exemplo da Figura 3.3, observamos que a *OBU3* analisou a zona com o ID 0, logo se formos à lista *obus_cameras* vemos que a imagem no index 0 é *seca.png*, que representa a imagem associada à zona e capturada pela respetiva *OBU*.

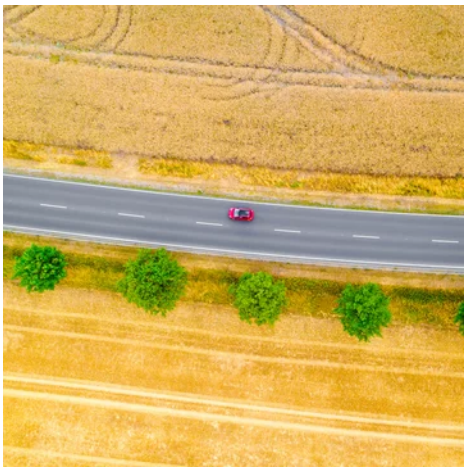

Figure 3.5: Imagem seca.png.

Através da imagem capturada, vemos claramente que o terreno precisa de ser regado, daí o resultado da análise da *OBU3* para a zona com ID 0 ter sido *"need water"*.

Chapter 4

Resultados

Em termos de resultados obtidos, foi possível verificar que os drones efetuam o processamento correto das *CAMs* recebidas uma vez que, eles se comportam como o esperado e também processam corretamente as *DENMs* entre eles porque as bases de dados *Zones* são iguais em todas as *OBUs* e por sua vez a informação é passada corretamente para a aplicação web.

Além disso, foi possível realizar várias simulações com posições diferentes para cada *OBU* e adição de mais uma zona (no ínicio tinhamos apenas 4) e verificar que nunca uma zona era analisada por mais do que uma *OBU*, que as *OBUs* nunca retornavam à posição inicial sem que todas as zonas do campo tivessem sido analisadas e que a base de dados *Zones* era igual para todas *OBUs*, fruto da comunicação entre eles.

Chapter 5

Conclusão

5.1 Conclusões

No nosso ponto de vista, achamos que conseguímos atingir o objetivo inicial do projeto com sucesso e também implementámos tudo o que foi pedido por parte do professor. Durante todo o desenvolvimento do mesmo ficámos a entender melhor o quão importante é a troca de mensagens entre entidades da rede, como por exemplo *CAMs* e *DENMs*, para processamento de informação e tomada de decisões.

Além disso, temos noção que numa situação real deveria existir um alcance de comunicação ideal capaz de englobar todos os drones independentemente da zona onde eles estejam de forma a garantir que não haja atrasos na receção das mensagens ou até mesmo a não receção das mensagens levando a um comportamento totalmente diferente do esperado.

Concluindo, achamos que este projeto foi muito importante no sentido em que ganhamos um grande conhecimento sobre redes veiculares, conseguimos perceber o funcionamento do *vanetza* e entendemos melhor toda a interação e tomadas de decisão entre entidades da rede através de mensagens trocadas. Temos a certeza que o conhecimento ganho no desenvolvimento deste projeto irá ser muito importante no nosso futuro.

5.2 Trabalho Futuro

Como trabalho futuro, passaria por melhor a maneira como é feita a análise dos campos agrículas uma vez que a maneira que estamos a usar para fazer a análise dos mesmos não é 100% viável porque, por exemplo, imaginando que o drone capta uma imagem onde além de ter uma zona do campo também tem uma casa ou uma estrada, a percentagem de píxeis verdes (imaginando que o campo estaria em boas condições) já não seria muito alta, devido a haver outras

cores dos píxeis na imagem além do verde/amarelo/castanho (por causa da casa/estrada) e, até mesmo se a casa tivesse um telhado com tons castanhos, a percentagem de píxeis castanhos resultante da análise iria ser um pouco grande indicando que o campo precisasse de ser regado quando na verdade o campo está em perfeitas condições, o drone apenas captou na imagem o telhado de uma casa.

A nossa solução passaria por usar um sensor para analisar os campos agrículas de maneira a tornar a análise dos mesmos mais viável.

Podería-se também adaptar o código a um cenário mais real por exemplo, usar apenas *docker containers* para conseguirmos integrar com o hardware dos drones, de forma a no futuro ser possível testar o programa desenvolvido numa situação mais real.

Ainda como trabalho futuro, facilmente podería-se alterar a aplicação web para quando os drones chegassem a uma zona aparecer, numa *html box* por exemplo, a imagem que foi capturada pelos mesmos e que posteriormente vai ser analisada, de maneira a tornar a aplicação web mais intuitiva para quem é de "fora".