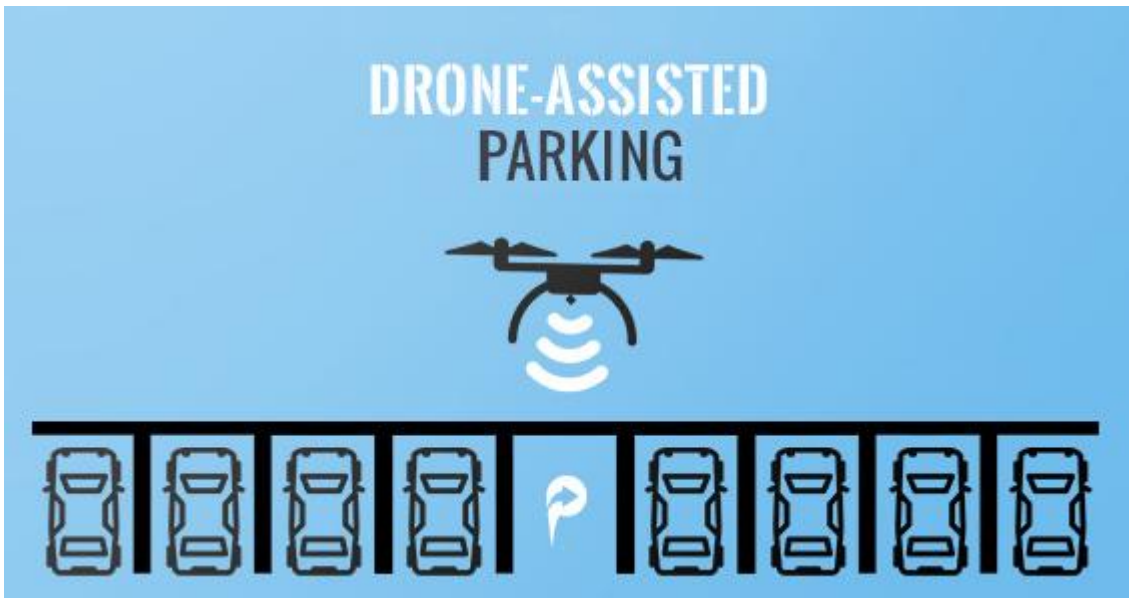
universidade de aveiro

LICENCIATURA EM ENGENHARIA INFORMÁTICA

DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

Monitorização de Parque de Estacionamento Utilizando Drones



Aveiro 15 de Junho de 2016

Elementos do grupo:

André Cardoso, 65069

André Carvalho, 64257

Tiago Coelho, 52029

Josimar Cassandra, 66109

Orientador:

Prof. António J. R. Neves

Coordenador:

Prof. José Moreira

Resumo

Este projeto visa o desenvolvimento de um sistema para gestão dos parques de estacionamento da Universidade de Aveiro com recurso a imagens aéreas capturada por um *drone*.

O sistema é composto por um *drone* que, de forma autónoma e quando solicitado por um utilizador credenciado, sobrevoa um determinado parque e captura vídeo que é depois processado autonomamente para a contagem de veículos.

O sistema engloba também um *website* usado como interface para o utilizador iniciar novas capturas e disponibilizar a informação sobre os lugares ocupados, obtida pela análise de imagem.

Índice

Índice de Tabelas 4
Índice das imagens 4
Introdução 5
1.1. Conteúdo 6
1.3. Motivação 6
1.4. Objetivos 7
1.5. Estrutura do relatório 7
2. Estado da Arte 8
2.2. Escolha do drone 9
2.3. Tecnologias 11
2.2.1. Java Spring Framework 11
2.2.2. Node Bebop 12
2.2.3. OpenCV 12
2.3.4 RabbitMQ 13
2.3.5 Base de Dados 14
2.3.6 MAVLink 15
2.3.7. Python 15
2.3.8. C++ 15
3. Requisitos do sistema e Arquitetura 16
3.1. Requisitos do sistema 16
3.1.2. Actores 16
3.1.3. Casos de Uso 17
3.2. Requisitos Funcionais 19
3.2.1. Regras de negócio 20
3.2.2. Autenticação e níveis de autorização 20
3.2.3. Características do sistema 21
3.4. Arquitetura do Sistema 22
3.4.1. Modelo de domínio 22
3.4.2. Modelo de instalação 23
3.4.3. Modelo da Tecnologia 24
3.4.4. Modelo de Atividades 25
3.4.5. Modelo de Componentes 26
4. Android 28

4.2. Dependências e Versões 28
4.3. Aplicação Móvel 28
5. Análise de imagem 30
5.1. Watch Folder 31
5.2. Partição em frames 32
5.3. Algoritmo de imagem 33
5.4. Serviços RESTful 34
6. Relação entre componentes 35
6.1. Inicialização do projeto 35
6.2. Comunicação com o Drone 35
6.2.1. Detalhes de utilização das Tecnologias 36
6.3. Estrutura do repositório 38
7. Aplicação web 39
7.1. Base de Dados 40
7.2. Serviços RESTful 41
7.3. RabbitMQ 41
Discussão 42
Conclusão 43
Referências 44

Índice de Tabelas

Tabela 2: Descrição consulta de lugares disponíveis 17
Tabela 3: Descrição da consulta de estatísticas 18
Tabela 4: Descrição da visualização de fotos do parque 18
Tabela 5: Descrição do envio do drone para nova captura 18
Tabela 6: Descrição do Serviço de comunicação com o drone 26
Tabela 7: Descrição do Serviço de Processamento de Imagens 27
Tabela 8: Descrição do Servidor Aplicacional 27

Índice das imagens

Tabela 1: Comparação dos Drones 10
Figura 1: Modelo de caso de uso 17
Figura 2: Modelo de Domínio 22
Figura 3: Modelo de instalação 23
Figura 4: Modelo da Tecnologia 24
Figura 5: Modelo de atividades 25
Figura 6: Modelo de componentes 26
Figura 7: Diagrama UML Aplicação Móvel 29
Figura 8: Modelo de base de dados 40

Introdução

Atualmente as tecnologias têm evoluído a um ritmo impressionante sendo de notar a área dos *drones*, visto que é o ponto fulcral do nosso projeto. Diariamente novas experiências e ideias surgem à volta das possíveis utilidades a dar aos mesmos.

Este projeto teve como objetivo utilizar um *drone* para obter informações relativas a parques de estacionamento. É necessário que várias componentes se interliguem para que seja possível construir uma solução exequível.

Inicialmente foi necessário interagir com o *drone* para definir rotas de voo autônomas e capturar imagens dos parques, sendo que no mercado atual existem vários *drones* programáveis com bibliotecas disponíveis ao público que ajudam neste tipo de objetivos. Uma vez efetuada a captura o vídeo é descarregado e processado por um algoritmo de imagem capaz de fazer uma contagem do número de lugares ocupados no parque em questão. Para finalizar existe uma última componente que é a parte em que se compilam e disponibilizam todas as informações ao público através de um *website*.

1.1. Conteúdo

Este é o projeto final da Licenciatura em Engenharia Informática cujo tema é a monotorização dos parques de estacionamento utilizando *drones*. O projeto foi desenvolvido no Departamento de Eletrónica Telecomunicações e Informática (DETI) e no Instituto de Engenharia Eletrónica e Telemática de Aveiro (IEETA).

Recentemente os departamentos estão a entrar em projetos na área dos *drones* sendo este um dos projetos piloto que tem como objetivo interligar as várias tecnologias e técnicas que adquirimos durante o período acadêmico de forma a criar um produto final funcional.

1.3. Motivação

Atualmente existem várias soluções para fazer controlo de entradas e saídas de parques de estacionamento, como sensores e portões de acesso. Mas estas soluções possuem alguns problemas como: o custo de implementação, a sua flexibilidade e o facto de que não serem soluções ótimas.

Assim a motivação pela qual escolhemos o nosso projeto deve-se não só à recente evolução dos *drones* mas também à possibilidade de implementar outra solução viável, com menos custos e mais flexível. Esta solução motivou-nos pelo facto de existir a possibilidade de trabalhar com várias tecnologias para construir o produto final e pelo facto de que a solução se poderá adaptar facilmente a qualquer parque ao ar livre. Outro dos grandes motivos foi o facto de existir a possibilidade de programar os *drones*, o que nos despertou bastante interesse.

1.4. Objetivos

Para que o projeto seja realizado da melhor forma possível foi subdividido em três objetivos principais:

1) A parte da navegação autônoma aos parques, em que cada *drone* terá de ser capaz de receber as rotas programáveis com as fronteiras do parque e executar uma rotina. Durante esta rotina é necessário que capture um vídeo que possua todos os lugares do parque.

2) A segunda parte é relativa à análise de imagem, em que terá de ser desenhado um algoritmo que seja capaz de contar o número de carros com o menor número de erro possível. O algoritmo terá de conseguir receber o vídeo, calcular e armazenar os resultados para posterior uso.

3) Por fim a criação de um sistema de informação *online*, para fornecer aos utilizadores as informações de cada parque, e através de uma área reservada dar a possibilidade de iniciar novas capturas.

1.5. Estrutura do relatório

Para além da introdução, este documento tem mais 7 capítulos. No capítulo 2, é descrito o estado da arte em sistemas de monitorização de parques de estacionamentos usando *drones* e as tecnologias que usamos no nosso projeto. O Capítulo 3 apresenta o processo de listagem de requisitos e exigências e arquitetura do sistema. No capítulo 4, apresentamos a aplicação que desenvolvemos em *Android* e a finalidade da mesma no âmbito do nosso projeto. Em seguida, no capitulo 5, descrevemos o algoritmo de imagem, como é crucial para o nosso projeto e como se relaciona com as restantes componentes que constituem a nossa solução. O capítulo 6 é relacionado com a comunicação com o *drone*. O capítulo 7 descreve todas as componentes que são necessárias e constituem a base do *website*, como a base de dados o uso dos serviços *RESTful* e *RabbitMQ*. Por fim, o capítulo 8, faz um resumo do trabalho realizado neste projeto, os principais resultados, conclusões e trabalhos futuros.

2. Estado da Arte

Atualmente existem vários estudos relativos à deteção de objetos em imagens aéreas, mais precisamente deteção de veículos. Assim vários algoritmos e formas de processar as imagens aéreas foram criados para melhorar cada vez mais a sua percentagem de sucesso.

No mercado atual existem poucos projetos públicos com implementações detalhadas destes algoritmos, sendo que é uma área bastante recente e pouco estudada. Nestes estudos o principal objetivo é reduzir o número de falsos positivos para ter uma boa percentagem de sucesso. O que os difere, para além do algoritmo de contagem, são as tecnologias usadas para capturar as imagens e as processar. Atualmente a maior parte das soluções implementadas utilizam imagens de CCTV, para controlo de tráfego e também para monitorização dos parques.

Relativamente ao nosso projeto, existem outros semelhantes em desenvolvimento na área dos *drones*. Um projeto bastante conhecido nesta área foi desenvolvido em 2015 pela empresa DJI, líder de mercado na construção de *drones*, que juntamente com a Shanghai's Fudan University criou um sistema [1] capaz de detetar veículos mal estacionados, em parques de estacionamento.

O que diferencia o nosso projeto dos restantes é a forma como inter-relacionamos todos os componentes e apresentamos um produto inovador. Uma vez que o nosso sistema não só usa *drones* para detetar veículos como faz uso dessa informação para construir um sistema de informação para possíveis clientes. Em termos de análise de imagem o nosso projeto também é uma novidade visto que existem poucos projetos de deteção de carros utilizando imagens aéreas capturas por *drones*. Em termos de automação do *drone* os métodos que utilizamos são recentes o que nos torna pioneiros na área.

2.2. Escolha do drone

Decidimos optar pelo *drone Parrot Bebop 2* [8], sendo que tanto *Parrot ar drone 2.0* versão GPS como *Parrot Bebop* 2 pareciam ser os mais indicados para implementar a solução. Esta decisão teve como fatores principais não só as suas características, que vamos detalhar em seguida, mas também a documentação e bibliotecas online que encontramos em relação aos mesmos.

A grande diferença entre ambos é a qualidade da câmara e a autonomia, pelo que preferimos o *Parrot Bebop 2* uma vez que a deslocação por mais parques e ao possuir mais resolução de imagem possibilita que o algoritmo de detenção de imagem tenha melhores recursos para trabalhar.

Para além destas opções também foram considerados os *drones Phantom 3 Professional* e *Phantom Advanced* que também seriam escolhas viáveis. A preferência pelo *Parrot Bebop 2* e no *Parrot ar drone 2.0* versão GPS devem-se ao facto de serem fornecidas bibliotecas para sistemas Unix e a relação qualidade preço.

Várias revisões desfavoreciam o *Parrot ar drone 2.0* em relação à distância, fragilidade e resistência ao vento. Em seguida apresentamos uma tabela com todos os detalhes mais importantes relativos aos *drones* mencionados, para que seja possível comparar as suas características.

Drone	Parrot Bebop 2	Parrot ar drone 2.0	Phantom 3 advanced
SDK	O SDK permite conectar, pilotar, receber streams, guardar ficheiros media, enviar e correr planos de voo automáticos e fazer updates ao drone. Este SDK é escrito principalmente em C e fornece librarias para sistemas Unix, Android e iOS. Existem também bibliotecas externas disponíveis para integração com outras plataformas.		SDK disponível para Android e iOS. Obrigatório criar uma conta para obter um app key, que permite a utilização do SDK.
Documentação	Explicita e rica em conteúdo.		Difícil de ler e com menos conteúdos.
Bateria	2700mAh (25 minutos)	1000mAh (12 minutos)	4480mAh - 23 minutos
Resolução de vídeo	1080p (30fps)	720p (30fps)	2.7k (30fps)
Resolução de imagem	14MP	Informação não fornecida	12MP
Radio feed	2 km	Informação não fornecida	5km
Wireless feed	0.3km	0.05 km	Informação não fornecida
Resistência a ventos	até 65km/h	Informação não fornecida	até 32km/h
Velocidade max	47km/h	40km/h	18km/h

Tabela 1: Comparação dos Drones

2.3. Tecnologias

Neste capítulo enumeramos todas as tecnologias (linguagens, *frameworks*, dependências) utilizadas no projeto. O objetivo é descrever quais são e como se instalam para o sistema operativo *Ubuntu*, depois em cada tópico da solução, é detalhada a sua aplicação.

2.2.1. Java Spring Framework

Spring [7] é uma *framework open source* para desenvolver aplicações em *Java* (é necessário a instalação do Java 8). Atualmente possui diversos módulos para tratar de problemas como a persistência de dados, tratar da segurança das aplicações entre outros. Tem como principais funcionalidades a injeção de dependências e a programação orientada a objetos.

Uma das grandes vantagens desta *framework* deriva de ter embebido o *Tomcat* e *Jetty*, que permite a sua execução sem necessidade de instalar um servidor aplicacional (Ex. *Glassfish*). Esta plataforma foi bastante importante na estrutura do nosso projeto uma vez foi utilizada para o desenvolvimento dos serviços *RESTful*.

Como criamos um projecto "*maven web application*", a instalação da *Spring framework* é feita através de uma dependência no *pom.xml*.

Exemplo de execução "*java -jar drone.jar --static.path=public/*". O *--static.path* é especialmente importante, pois define uma pasta de recursos estáticos fora do *Java archive*, que podem ir sendo adicionados, como as imagens dos parques.

2.2.2. Node Bebop

Optamos por utilizar uma *framework* de terceiros, desenvolvida em *Node.js*, para comunicar com o *drone*. Esta biblioteca [9] implementada em *Node.js* era no entanto limitada em termos das funcionalidades do *drone*, nomeadamente a navegação autônoma através de um ficheiro *mavlink*.

Esta limitação, levou-nos numa fase inicial para uma implementação em *Android* utilizando a biblioteca oficial [10] da *Parrot*. No entanto os programadores envolvidos no projeto *node bebop*, após nossa solicitação acabaram por desenvolver novas funcionalidades, nomeadamente suporte ao *mavlink*, que nos permitiu aplicar esta biblioteca na solução final do projeto.

2.2.2.1. Instalação

Para fazer uso do *Node.js* e realizar esta comunicação é necessário seguir os seguintes passos de forma a instalar o *Node.js* e usar a biblioteca:

1) Instalar o *Node.js* em: https://nodejs.org/en/download/package-manager/
2) Instalar a biblioteca recorrendo ao comando: $ **npm install node-bebop**

2.2.3. OpenCV

É uma biblioteca multiplataforma totalmente livre para uso acadêmico. Utilizamos esta biblioteca que fornece módulos de processamento de imagem e vídeo.

A ferramenta tornou-se ideal para esta componente do nosso projeto de forma a integrar o algoritmo de imagem para contagem de número de lugares ocupados em cada estacionamento com recurso ao algoritmo de imagem já desenvolvido. Inicialmente a escolha foi feita com vista a desenvolver o nosso próprio algoritmo de imagem.

2.2.3.1. Instalação

A versão que utilizamos foi a 3.1, pelo que versões mais antigas ou mais recentes poderão não funcionar na totalidade com o nosso projeto. A instalação foi feita em sistemas *unix* e vamos apresentar os passos para a sua instalação:

1) Inicialmente é necessário instalar várias componentes como: *GCC, CMake, GTK, pkg-config, python, numpy, ffmpeg e libav packages*. Para isto são necessários dois comandos, sendo eles:
    a) $ **sudo apt-get install build-essential**
    b) $ **sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev**

2) A forma mais fácil de fazer download da versão do *OpenCV* é recorrer ao *GIT* e fazer clone do repositório, seguindo os seguintes comandos:
    a) $ **cd ~/<my_working_directory>**
    b) $ **git clone https://github.com/Itseez/opencv.git**
    c) $ **git clone https://github.com/Itseez/opencv_contrib.git**

2.3.4 RabbitMQ

O *RabbitMQ Server* [11] é um servidor para suportar a troca de mensagens, sendo concebido para suportar *AMQP* (*Advanced Message Queuing Protocol*). Foi escrito em *Erlang*, é robusto, fácil de usar, funciona nos principais sistemas operativos, suporta um enorme número de plataformas de desenvolvimento, é *open source* e pode ter suporte comercial. Utilizamos este serviço para facilitar a troca de mensagens entre o *website* e o *node controller* (*laptop*). O *RabbitMQ server* foi instalado no servidor virtual.

2.2.4.1. Instalação

Para instalar basta seguir um conjunto de comandos:

1) Atualizar o sistema: $ **sudo apt-get update**
2) Ativação do repositório da aplicação *RabbitMQ*: $ **echo "deb http://www.rabbitmq.com/debian/ testing main" >> /etc/apt/sources.list**
3) Adicionar a chave de verificação para a pasta: $ **curl http://www.rabbitmq.com/rabbitmq-signing-key-public.asc | sudo apt-key add**
4) Atualização do sistema: $ **sudo apt-get update**
5) Finalmente, fazer o *download* e instalar o *RabbitMQ*: $ **sudo apt-get install rabbitmq-server**

2.3.5 Base de Dados

MySQL é um sistema gestor de base de dados relacionais de código aberto usado em muitas aplicações para gerir as bases de dados. O serviço utiliza a linguagem *SQL* (*Structure Query Language*), que é a linguagem mais popular para as operações *CRUD*.

2.3.5.1 Instalação

Para instalação da mesma será necessário executar o seguinte comando: $ **sudo apt-get install mysql-server**

2.3.6 MAVLink

MAVLink ou *Micro Air Vehicle Link* é um protocolo de comunicação com pequenos veículos não tripulados. Contêm uma estrutura muito simples com várias linhas de comandos, que depois são interpretados pelo dispositivo de forma sequencial.

Os comandos variam conforme os equipamentos, mas podem significar o início de gravação em vídeo pela câmara, deslocação do veículo para uma coordenada, indicando a velocidade e altitude. O *drone* da *Parrot* que utilizamos suporta este protocolo de navegação autônoma, através de colocação ficheiro **.mavlink* numa diretoria especifica.

2.3.7. Python

Para que o conteúdo *Python* do nosso projeto funcione é necessário instalar a versão 3.5. Esta linguagem é usada no script de *startDrone.py*. Para instalar, seguir os seguintes passos:

1) Para instalação do *Python* 3.5 usar:
    a) $ **sudo apt-get install python3.5**
    b) $ **alias python=python3**

2) Para instalação das dependências necessárias usar:
    a) Instalação do *pika*, $ **sudo apt-get install python3-pip** e $ **pip3 install pika**
    b) Instalação do *Naked*, $ **pip3 install Naked**

2.3.8. C++

Utilizamos a linguagem C++ no algoritmo de análise de imagem. Uma vez que o trabalho foi realizado no sistema operativo *Ubuntu* este já nos oferece o *GCC standard*. E todas as tecnologias necessárias para compilar o projeto com o algoritmo de imagem já foram previamente instaladas no comando: $ **sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev**

3. Requisitos do sistema e Arquitetura

Neste capítulo fizemos um levantamento dos requisitos e da arquitetura do sistema.

3.1. Requisitos do sistema

RS 1: O sistema deve guardar toda a informação obtida pelo *drone*.

RS 2: O sistema deve ser instalado num servidor a correr uma distribuição *Linux*.

RS 3: O interface *web* do sistema, será compatível com os *browsers* mais recentes. Versões anteriores poderão ter anomalias no seu funcionamento.

RS 4: O sistema deve ser facilmente adaptável a um novo *drone* e a novos parques.

3.1.2. Actores

Temos o utilizador, este ator é qualquer individuo que pretenda consultar informação sobre os parques disponíveis, através do *website*.

E o gestor, este ator é o responsável pela gestão do parque com permissões para gerir os voos do *drone*.

3.1.3. Casos de Uso

Diagrama de casos de uso, com os dois atores existentes na plataforma e suas principais funcionalidades.

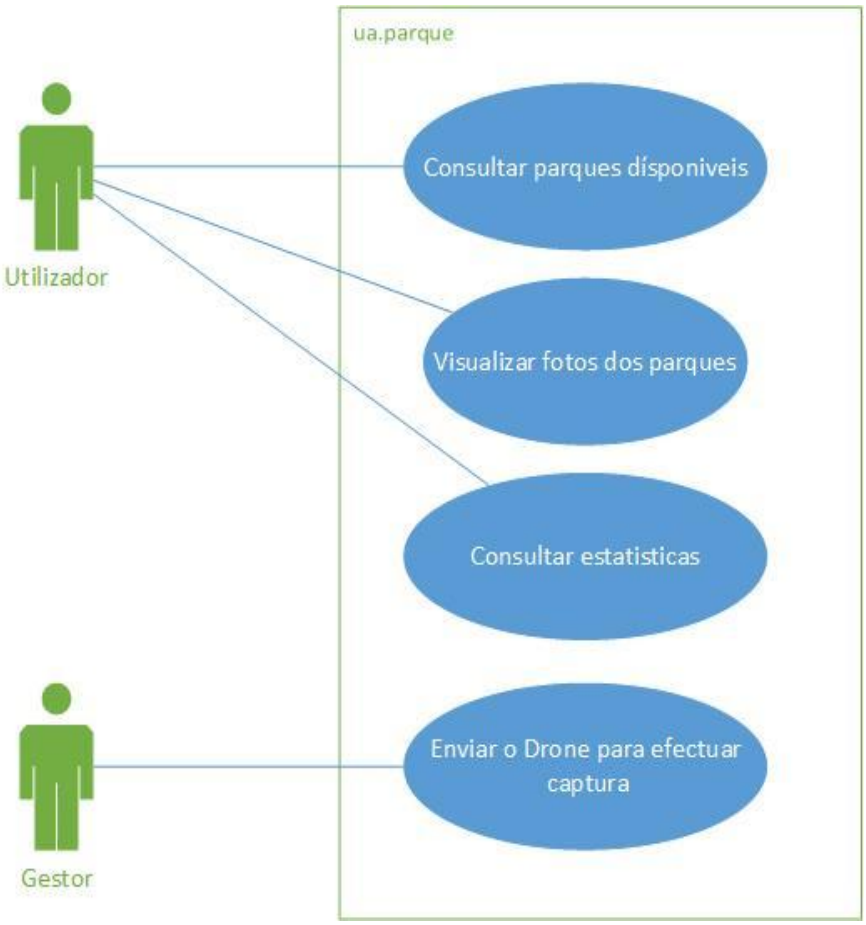


Figura 2: Modelo de caso de uso

Nome:	Consultar lugares disponíveis
Descrição	O utilizador do parque acede ao website, entra na área de parques, e verifica quantos lugares estão disponíveis por parque
Exceções	N/A
Prioridade	Alta

Tabela 1: Descrição consulta de lugares disponíveis

Nome:	Consultar estatísticas
Descrição	O utilizador do parque acede ao website, entra na área de estatísticas, e escolhe os filtros que pretende consultar (parque e dia)
Exceções	N/A
Prioridade	Alta

Tabela 2: Descrição da consulta de estatísticas

Nome:	Ver fotos do parque
Descrição	O utilizador do parque acede ao website, entra na área de parques, e clica no parque que pretende visualizar as fotos
Exceções	N/A
Prioridade	Alta

Tabela 3: Descrição da visualização de fotos do parque

Nome:	Enviar drone para nova captura
Descrição	O Gestor do parque, através da área reservada do website, seleciona o parque que pretende e clica em enviar *drone*.
Exceções	As condições climatéricas não permitem o *drone* efetuar um voo e a ordem é cancelada.
Prioridade	Alta

Tabela 4: Descrição do envio do drone para nova captura

3.2. Requisitos Funcionais

Neste tópico apresentamos a lista de requisitos com uma pequena descrição.

REQ 1: O *website* deve permitir iniciar novas capturas, para os utilizadores devidamente autenticados.

REQ 2: O website deve fornecer por parque, o número de lugares ocupados e imagens da última captura, estatísticas e a localização do parque mapeado no google maps.

REQ 3: A área reservada deve também permitir a gestão de parques (criação, edição e eliminação).

REQ 4: Deve ser criado/obtido um algoritmo para deteção de veículos com base nas imagens áreas obtidas pelo *drone*.

REQ 5: O algoritmo de deteção de veículos, deve ser eficiente na grande maioria dos parques da Universidade de Aveiro.

REQ 6: O *drone* deve efetuar uma navegação autónoma, com instruções de voo pré-programadas para cada parque.

3.2.1. Regras de negócio

Neste tópico apresentamos a lista de regras do sistema.

RNE 1: O *drone* apenas pode efetuar uma captura de cada vez, sem regressar à base.

RNE 2: Se as condições climatéricas não forem as ideais, ou seja, vento e/ou chuva, o *drone* não deve aceitar novas capturas.

RNE 3: O *drone* não deve efetuar capturas senão existir luz natural.

3.2.2. Autenticação e níveis de autorização

Existem dois tipos de utilizadores que podem interagir com o nosso *website*.

Os utilizadores comuns, ou seja, o público para a qual se destina a nossa aplicação, podem consultar informações relativas aos parques de estacionamento como: ocupação dos parques a determinada hora, localização geográfica dos parques através do *google maps* como também um histórico com recurso a um gráfico ocupação/hora de cada parque.

O outro nível de utilizador é o de gestor, que pode aceder à área reservada do *website*, através das suas credenciais. Nesta área de administrador é possível consultar o histórico de ordens de captura, gerir os parques, e iniciar uma nova captura.

3.2.3. Características do sistema

O sistema é composto por várias componentes que interligadas estabelecem uma solução, estas componentes têm as seguintes funcionalidades:

1) Uma base de dados que permite armazenar a informação relativa aos parques.
2) Uma forma de interligar a base de dados com o website.
3) Um website que possibilita:
    a) Fornecer ao utilizador informações relativas ao número de estacionamentos livres, ocupados e totais, por parque.
    b) Galeria de imagens, por cada captura efetuada ao parque.
    c) Google maps a limitar a zona do parque.
    d) Relatório de atividades do drone.
    e) Gráfico de ocupação/hora.
    f) Área reservada a utilizadores com credenciais, para efetuar operações diretas com o drone.
4) Um algoritmo de imagem capaz de processar os vídeos recebidos pelo drone e fazer uma contagem dos carros.
5) Uma solução que permita receber os vídeos do drone quando ele termina o seu percurso.
6) Uma aplicação capaz de traçar rotas para cada parque de estacionamento, para ser usada pelo drone.
7) E por fim uma forma de apresentar os resultados obtidos pela contagem dos carros no website.

3.4. Arquitetura do Sistema

Neste capítulo são fornecidos os vários diagramas que retratam a arquitetura do sistema.

3.4.1. Modelo de domínio

Para guardar a informação, desenhamos o seguinte modelo. Neste diagrama UML temos as entidades e os atributos de cada uma delas.

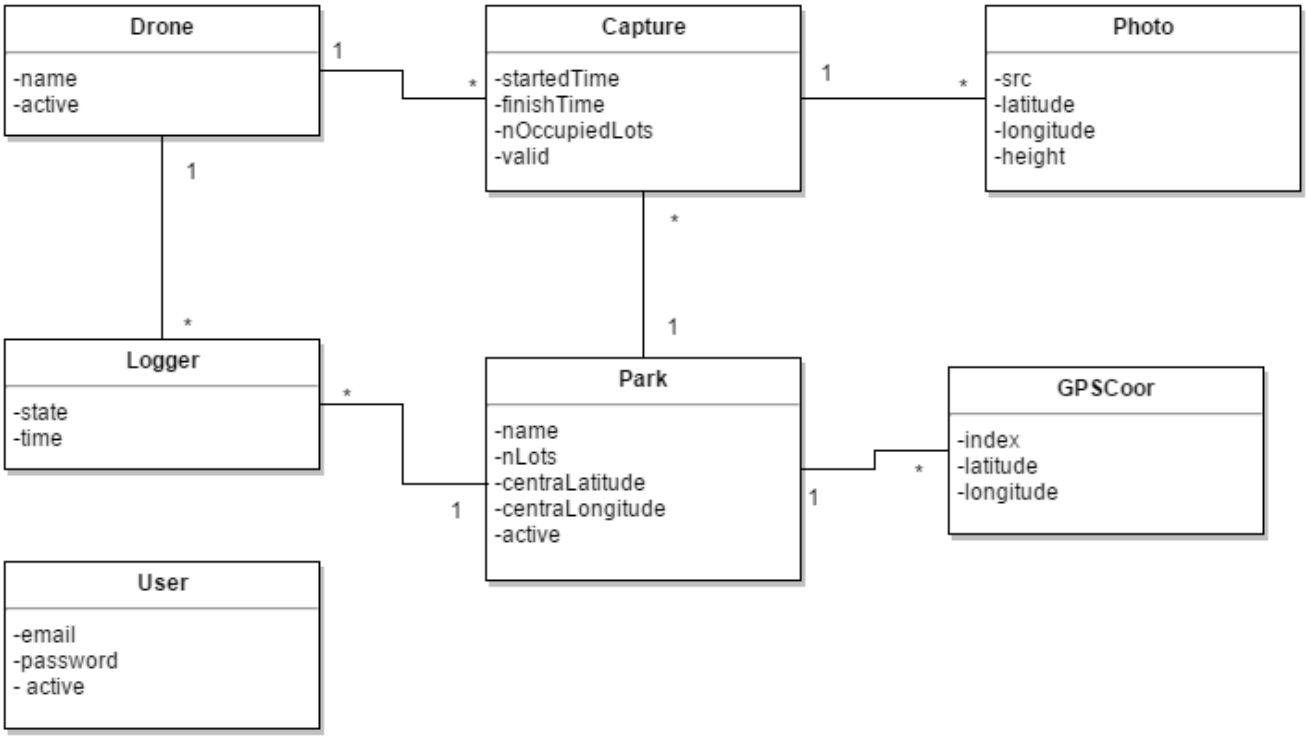


Figura 3: Modelo de Domínio

3.4.2. Modelo de instalação

Apresentamos um diagrama que explica os vários componentes instalados nos diferentes dispositivos. O *node controller*, refere-se a um equipamento portátil, a componente pessoal é um utilizador que utiliza o browser para aceder ao nosso site e a máquina virtual é onde está instalado o *website* a base de dados e serviços *RESTful*.

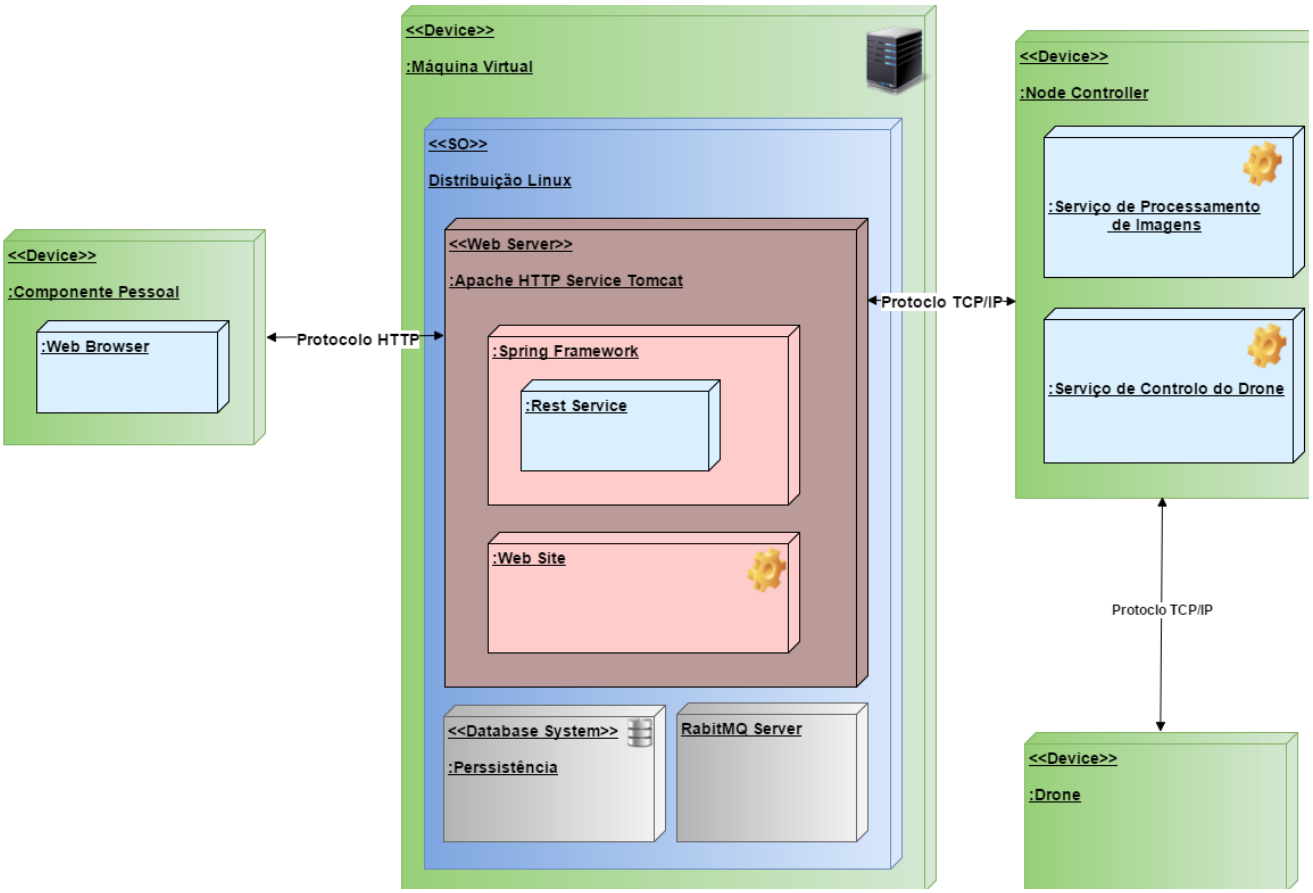


Figura 4: Modelo de instalação

3.4.3. Modelo da Tecnologia

Neste diagrama conseguimos visualizar as várias camadas da nossa arquitetura. O *website*, invoca serviços *REST* que estão disponíveis no servidor, que por sua vez comunica com o equipamento externo (*node controller*), que está ligado ao *drone*.

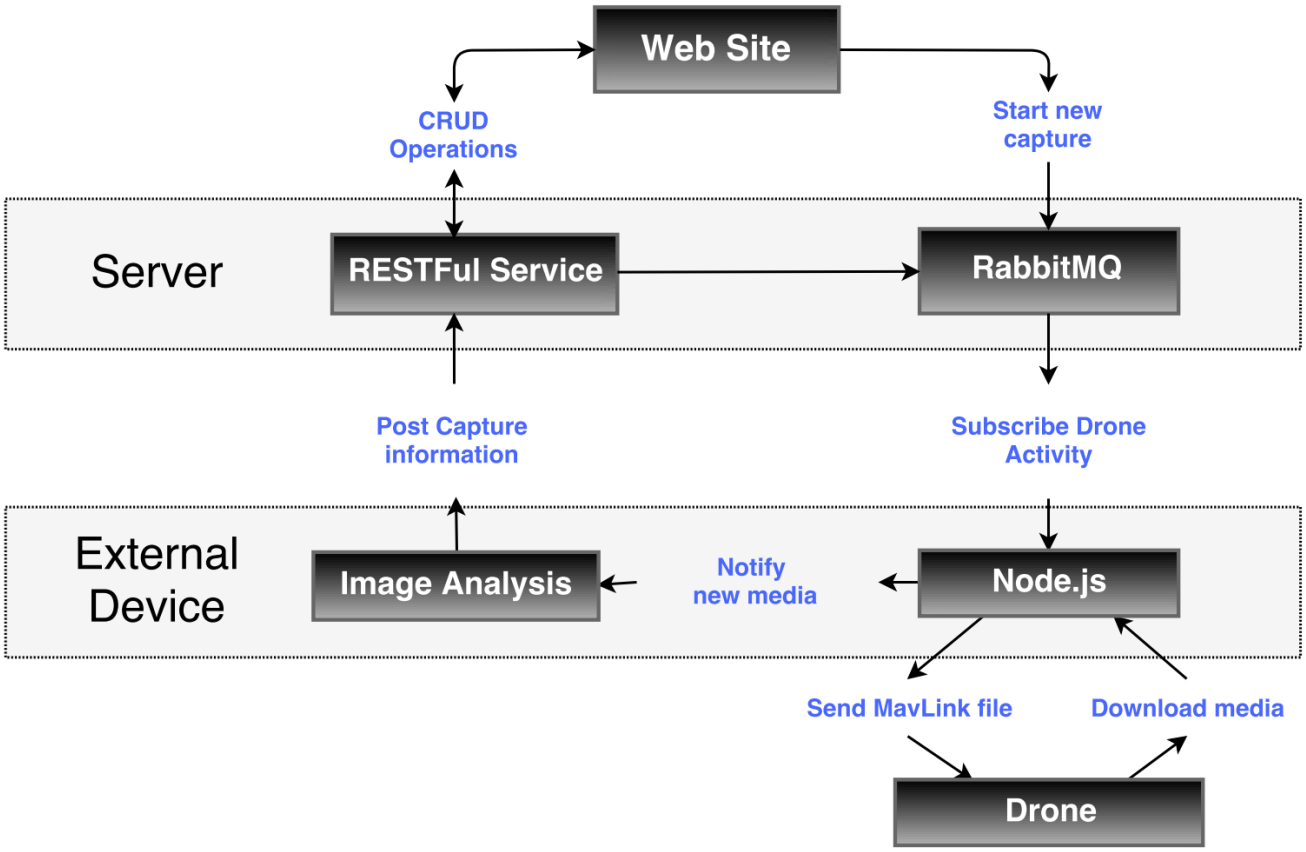


Figura 5: Modelo da Tecnologia

3.4.4. Modelo de Atividades

Neste diagrama descrevemos o fluxo de uma nova captura que inicia no *website*, e termina no envio da informação para a base de dados por *REST*.

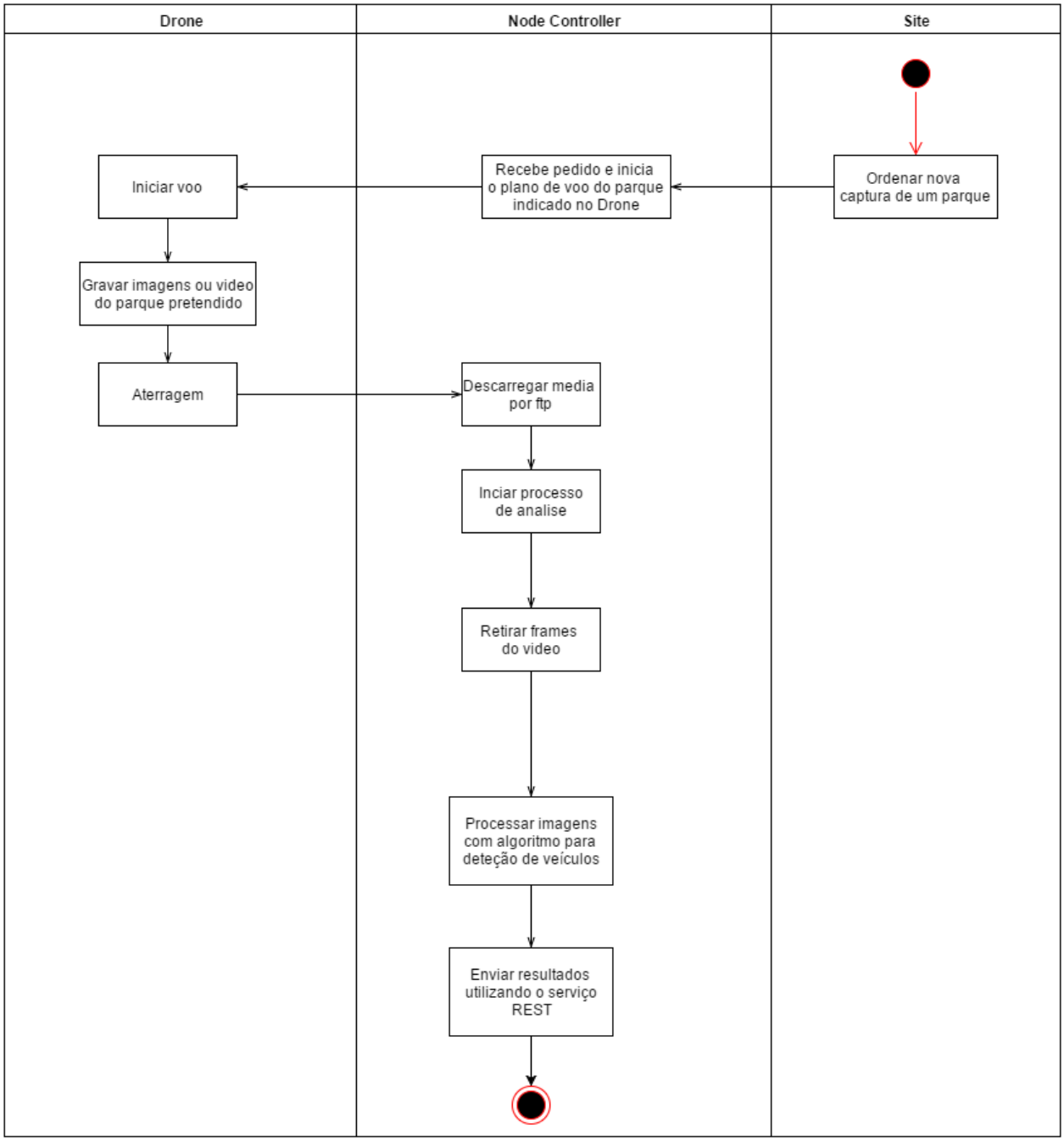


Figura 6: Modelo de atividades

3.4.5. Modelo de Componentes

Neste diagrama descrevemos as várias componentes do nosso projeto.

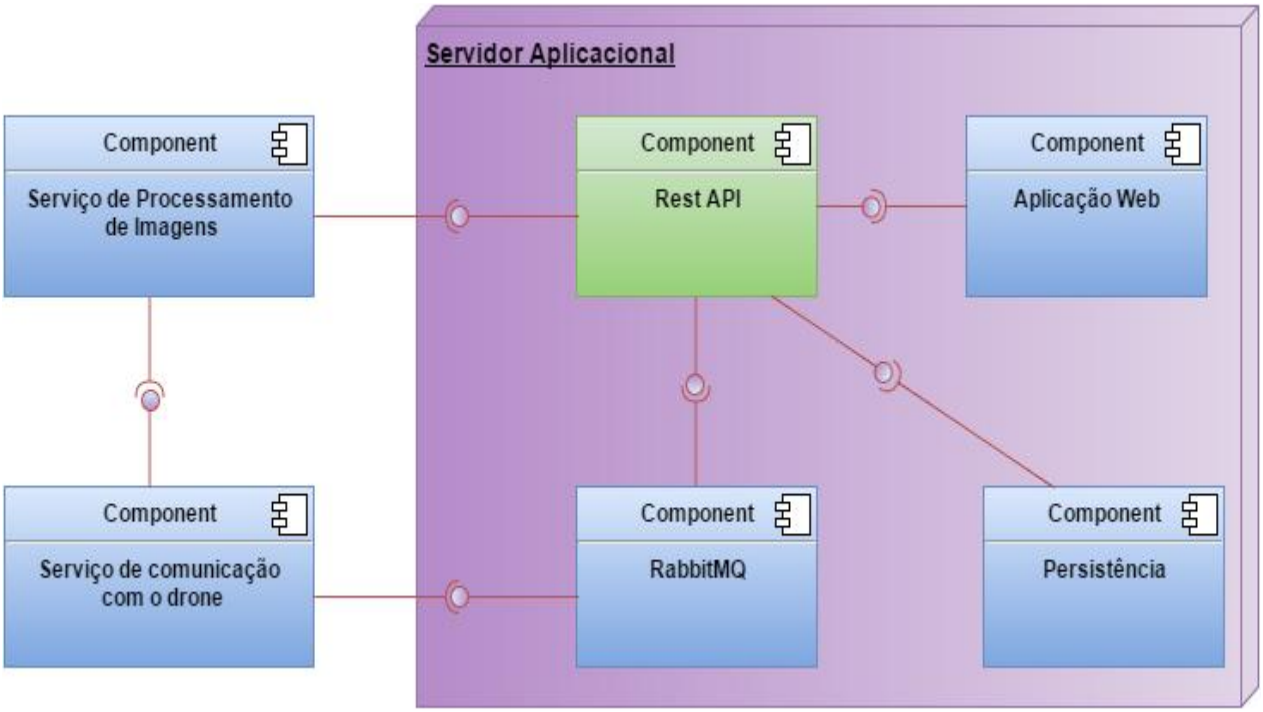


Figura 7: Modelo de componentes

Nome:	Serviço de comunicação com o drone
Descrição	Dispositivo preparado com duas interfaces de rede, sendo este responsável pela comunicação com o drone e a aplicação web em simultâneo. Este comunica também com o Serviço de processamento de Imagens.

Tabela 5: Descrição do Serviço de comunicação com o drone

Nome:	Serviço de Processamento de Imagens
Descrição	Responsável por retirar do vídeo as frames necessárias, analisar as imagens utilizado o algoritmo de detecção de veículos e enviar os resultados para o a aplicação web.

Tabela 6: Descrição do Serviço de Processamento de Imagens

Nome:	Servidor Aplicacional	
	Rest API	Aplicação Web
Descrição	Repositório de informação do sistema que comunica com: Aplicação Web, Persistência, RabbitMQ, Serviço de comunicação com o drone e Serviço de Processamento de Imagens.	Responsável por disponibilizar informações sobre os parques de estacionamento.
Nome:	Persistência	RabbitMQ
Descrição	Responsável por armazenar e fornecer os dados.	Responsável pela troca de mensagens entre a Rest API e o Serviço de comunicação com o drone.

Tabela 7: Descrição do Servidor Aplicacional

4. Android

A componente *Android* no nosso projeto representa uma introdução ao desenvolvimento de uma aplicação. Inicialmente o objetivo era usar a aplicação como dispositivo externo com duas interfaces de rede que estabeleceria a ligação entre o site e o *drone*. Isto porque existe muita documentação fornecida pela *Parrot* relativamente ao *SDK* para *Android*. No entanto melhores soluções foram encontradas como já explicado anteriormente (com a utilização do *node-bebop* em *Node.js*).

4.2. Dependências e Versões

A aplicação foi desenvolvida para dispositivos com a versão *SDK* (*Software Developer Kit*) da *API* 23, sendo que se pode utilizar até a *API* 15 como requisito mínimo para que a aplicação funcione. Bibliotecas externas usadas no desenvolvimento da aplicação encontram-se listadas abaixo:

1) Para transferências *ftp* usou-se: **ftp4j-1.7.2.jar**
2) Para possibilitar a receção de mensagens em *realtime*: **rabbitmq-client.jar**
3) A biblioteca oficial da *Parrot* para dispositivos móveis: **arsdk-3.8.3**

4.3. Aplicação Móvel

Começamos por explorar as funcionalidades a partir de um *sample* fornecido pela *Parrot* [2] para *developers* na plataforma *GitHub*.

O *sample* disponibiliza várias classes para o uso de uma variada gama de *drones*, no entanto no contexto deste projeto usamos apenas as relativas ao *Bebop* 2. Para simplificar a arquitetura foram removidas as classes das quais não fazemos uso.

A aplicação apresenta um menu inicial onde cada opção tem uma funcionalidade diferente. O principal objetivo era descentralizar funcionalidades e isolar as mesmas de forma a serem testadas e usadas como orientação. O desenvolvimento das funcionalidades foi baseado no site oficial onde estão descritos comandos e eventos, sendo que a aplicação envia comandos e recebe eventos.

A aplicação original engloba todos os processos necessários para procurar e conectar o dispositivo ao *drone*, controlar o movimento em todos os seus eixos, *stream* e transferência de ficheiros do *drone* para o dispositivo. Estas funcionalidades foram separadas para uma melhor compreensão da arquitetura da aplicação. Foram adicionadas as funcionalidades de leitura de GPS, código exemplo de como criar e executar ficheiros MAVLink e leitura de mensagens em tempo real.

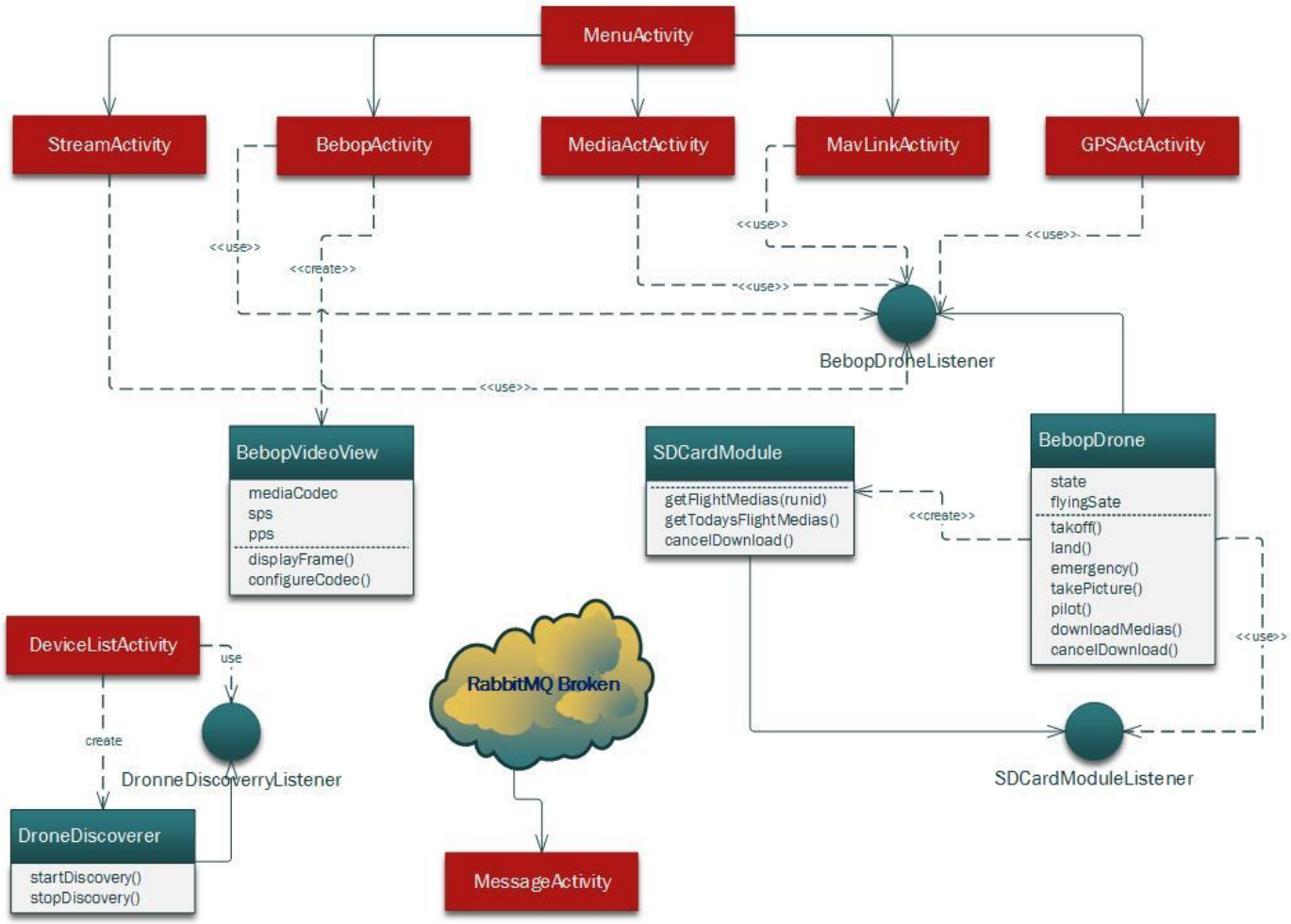


Figura 8: Diagrama UML Aplicação Móvel

5. Análise de imagem

Durante este capítulo vamos detalhar as tecnologias utilizadas na construção da solução de processamento de imagem. Esta é uma componente essencial da nossa solução que tem como finalidade ser o ponto de transição entre a leitura do vídeo recebido pelo *drone* e do *POST* dos resultados processados no website. Todos os processos envolvidos foram testados e abordados de diferentes formas para tentar solucionar o problema da melhor forma. O algoritmo de imagem foi fornecido por um aluno de mestrado [4] e integrado no nosso projeto para funcionar com as restantes componentes.

Os principais objetivos foi atingir um processamento rápido e intuitivo, onde é possível que as diferentes tecnologias envolvidas tomem partido umas das outras e tornem o processo autônomo.

O algoritmo tem como base a utilização das bibliotecas *OpenCV*, apesar de unificado o nosso algoritmo é composto por 4 componentes distintas. Um *watch folder*, para iniciar o script sempre que um ficheiro é colocado na pasta, uma parte dedicada a dividir o vídeo em *frames*, um algoritmo de imagem que utiliza as *frames* para detetar os veículos e por fim a invocação dos serviços *REST* para envio de informação no final do processo.

5.1. Watch Folder

O nosso algoritmo de imagem faz uso de um *watch folder* que desperta eventos sempre que recebe um vídeo com uma extensão predefinida (*.mp4*) num diretório específico. Desta forma o intuito do *watch folder* foi tornar o mecanismo de análise de imagem automático sempre que o *drone* envia um vídeo.

Para que isto fosse possível utilizámos o *inotify(7)* [3] que é uma API que fornece mecanismos de monotorização de eventos no sistema de ficheiros. Permite que ações sejam despertadas quando um diretório sofre modificações. Isto tornou-se uma mais-valia para o nosso projeto porque possibilitou predefinir uma diretoria com a qualidade de *watch folder*.

Desta forma sempre que executamos o nosso ficheiro *roads.cpp* ele já engloba a componente de análise de imagem e o evento do *watch folder* "*IN_MOVE*". Assim ao receber um vídeo caso este tenha um formato e nome especifico como "*dronex_parky(…).mp4*", irá inicializar o algoritmo de análise de imagem. Dividindo o vídeo por *frames* e executando em seguida a contagem dos carros.

Para inicializar o algoritmo com o *watch folder*, e desta forma fazer com que a diretoria do projeto de análise de imagem esteja à escuta de novos vídeos, há que seguir os seguintes passos no terminal:

a) Entrar na diretoria do projeto $ **cd ../ProjectAlgorithm**
b) Criar uma diretoria vazia "/*build*" e outra "/*bin*" dentro da diretoria do projeto.
c) Entrar na diretoria build: $ **cd build**
d) Compilar o projeto utilizando os comandos: $ **cmake ..** e $ **make**
e) Entrar na pasta */bin* onde os executáveis foram criados: $ **cd ../bin**
f) E por fim correr o executável *roads:* $ **./roads**

5.2. Partição em frames

Uma vez que o *watch folder* desperta um evento a primeira instrução a ser executada é a parte da divisão do vídeo em *frames*. Isto acontece porque o algoritmo de imagem está pronto para receber imagens de forma a conseguir processar as mesmas e fazer a contagem dos carros.

No processo de obtenção de *frames*, o vídeo é cortado de meio em meio segundo, e as *frames* são armazenadas numa diretoria para serem depois utilizadas no algoritmo de imagem. Para que o algoritmo de imagem funcione é necessário que as imagens respeitem normas específicas como:

1) A estrada dos parques de estacionamento tem de estar precisamente no centro da imagem.
2) O angulo da câmara no processo de filmagem tem de estar a uma inclinação de 83º (máximo possível).
3) A velocidade do *drone* tem de ser de 2m/s.
4) E para finalizar a altitude do *drone* em relação ao pavimento deve de ser de 10 metros para que a área de interesse esteja totalmente representada na imagem.

Estas normas são facilmente respeitadas ao utilizar a aplicação para traçar as rotinas do *drone*.

5.3. Algoritmo de imagem

O algoritmo de imagem [4] para deteção de veículos baseia-se na densidade de arestas para determinar a posição dos carros estacionados. Este *software* assume que o *drone* se encontra centrado com a estrada e que os estacionamentos se localizam à direita, esquerda ou ambos.

Após a deteção da estrada e escolha das regiões de interesse em seu redor, o algoritmo avalia linha a linha a densidade de arestas. Zonas de baixa densidade representam automóveis e tejadilhos, habitualmente sem arestas. O algoritmo é composto por três partes distintas:

1) Encontrar as bordas das estradas:
    a) Converte a imagem para escala de cinza.
    b) Aplica-se o algoritmo de deteção de bordas em canny.
    c) Faz-se uma leitura da imagem do centro para fora, para que assim se encontre as fronteiras com a estrada.
    d) E guarda-se para cada linha, borda direita e à esquerda e largura da estrada

2) Selecionar regiões de interesse:
    a) Seleção de regiões entre fronteiras das estradas e bordas da imagem.
    b) Selecionar o lado direito, esquerdo ou ambos dependendo do que se quer processar.

3) Detetar carros na imagem:
    a) Fazer uma leitura das regiões de interesse e calcular o rácio de bordas em cada linha.
    b) Aplicar um filtro no resultado da relação de bordas.
    c) Encontrar as zonas de baixa densidade próximas dos veículos.
    d) Gravar a posição do carro e a média de *Hue-Saturation-Value* numa lista.

5.4. Serviços RESTful

Ao ser finalizada a contagem dos carros, o *roads.cpp* também engloba serviços *REST* que permitem que ao terminar o processo de análise de imagem seja feito um *POST* no servidor com os resultados, bem como algumas imagens da captura efetuada pelo drone. Deste modo irá atualizar informações na base de dados, sendo que ficam disponíveis no website.

6. Relação entre componentes

Este capítulo tem a finalidade de explicar como se interligam na prática todos os processos/componentes do nosso projeto, para que seja possível compreender como utilizar a nossa solução.

Engloba também a comunicação com o *drone*. Este ponto é bastante importante uma vez que explica de forma precisa como o *drone* efetua as suas comunicações.

6.1. Inicialização do projeto

Para o sistema funcionar com todos os seus componentes é necessário:

1) Instalar todas as dependências necessárias [2.3]

2) Inicializar o site e serviços *REST* no servidor virtual [7.1]
    a) **java -jar drone.jar --static.path=public/**". O *--static.path* é especialmente importante, pois define uma pasta de recursos estáticos fora do *java archive*, que podem ir sendo adicionados, como as imagens dos parques.

3) Inicializar no *node controller* os serviços
    a) Correr o ficheiro python: **$ python3 startDrone.py** [6.2.1]
    b) Correr o executável *roads:* **$ ./roads** [5.1.1]

6.2. Comunicação com o Drone

A ligação ao *drone* está disponível através de um *access point* do próprio *drone*. Como precisamos de uma ligação entre o servidor e o *drone*, utilizamos um portátil com dois interfaces de rede, um para o *drone* outro para a rede da Universidade de Aveiro. Assim conseguimos ter um equipamento que faz de *bridge* entre o servidor e o *drone*.

A *Parrot* disponibiliza uma *API* de comunicação com o *drone* para *Android* e *iOS*. Como mencionado no capítulo 2.3.7, optamos por utilizar a biblioteca de terceiros *node-*

bebop, implementada em *Node.js*, permitindo comunicar com o *drone* utilizando o sistema operativo *Linux*.

Na solução que implementamos utilizamos a versão *14.04LTS* do *ubuntu*, mas pode ser aplicada em Windows ou MAC, dado que estes sistemas operativos suportam todas as tecnologias utilizadas na comunicação com o *drone*.

Para navegação autônoma do *drone* utilizamos o protocolo *MAVLink*, que é suportado pelos *drones* da *Parrot*. Os ficheiros *mavlink*, foram criados utilizando a aplicação *mobile* [5] para *apple/Android Flight Plan*, da *Parrot*. Este ficheiro pode ser gerado manualmente, mas optamos por utilizar a aplicação pois o interface gráfico facilita a sua criação, é bastante intuitivo e previne erros. Para descarregar esse ficheiro, é necessário iniciar a navegação do *drone* através da aplicação, e no final efetuar uma ligação anónima por *ftp* ao *drone* e descarregar o ficheiro *flightPlan.mavlink* dentro da diretória */data/ftp/internal_000/flightplans/*, que corresponde à última navegação autônoma.

6.2.1. Detalhes de utilização das Tecnologias

Para esta comunicação ser possível implementamos várias tecnologias, que são descritas nos seguintes passos:

1. Efetuar a ligação wifi ao *drone* e à rede da Universidade de Aveiro. Note-se que é necessário dois interfaces de rede, no nosso caso utilizamos o interface de rede do portátil e uma antena externa usb.

2. Iniciar o script em *startDrone.py*, para ficar à escuta de mensagens através do *RabbitMQ*, que são enviadas pelo site. Ao efetuar um *POST* no *REST*, através do website, para início de uma nova captura, o servidor envia uma mensagem para a *queue* "*droneActivities*", que se encontra à escuta pelo *startDrone*.py. Optamos por utilizar este método de comunicação porque achamos que é o meio mais rápido para iniciar o processo.

3. Ao receber uma mensagem com o *id* do parque e o *id* do *drone*, o *startDrone*.py efetua um *POST* no serviço *REST* para o *log* com indicação que recebeu um novo

pedido de captura de um parque. Este *POST* permite ao *website* fornecer algum feedback sobre o estado do pedido.

4. Caso a ligação ao *drone* através do respetivo interface de rede seja bem-sucedida inicia o *script flightplan*.js em *Node.js*, que utiliza a biblioteca *node-bebop* para iniciar um novo plano de voo para o respetivo parque. Esse plano de voo, consiste num ficheiro *MavLink*, pré-gravado no disco do *drone*, dentro da diretória /data/ftp/internal_000/flightplans/, que contêm as coordenadas *GPS*, para onde o *drone* deve voar, qual a altitude, e que deve iniciar uma gravação em vídeo. Para cada parque foi criado um ficheiro *mavlink* diferente, que é enviado como parâmetro e invocado no *flightplan.js*.

5. Após conclusão da navegação com sucesso, o *startDrone.py* acede através do protocolo *ftp*, à diretoria /internal_000/Bebop_2/media, e efetua o download de todos os ficheiros *.mp4, que lá se encontram. Cada vídeo corresponde a uma captura, portanto o disco do *drone*, não deve conter vídeos antigos.

6. Após descarregar o ficheiro de vídeo *.*mp4*, com o nome original é acrescentado ao nome do ficheiro o *id* do parque, e do *drone*. No final deste processo o ficheiro é eliminado do *drone*, de forma a não entrar em conflito com uma próxima captura.

7. O ficheiro é gravado com um novo nome (id do parque e *drone*) temporariamente na pasta /*temp* e só após concluído o *download* é transferido para a diretoria que irá disputar o início da análise de imagem através de um *watch folder*. Os passos seguintes são descritos no capítulo 5.

6.3. Estrutura do repositório

Apesar do repositório [6] possuir vários conteúdos que não estão a ser utilizados, achamos por bem deixar alguns, uma vez que suportam as pesquisas feitas para chegar à solução final. Desta forma o repositório do *code.ua* tem a seguinte estrutura de diretorias:

1) Drone:
    a) **Android**, com todo o conteúdo relativo ao desenvolvimento da aplicação *Android*.
    b) **NodeJS**, contendo as pesquisas e exemplos de utilização desta *framework* com o nosso *drone*.
2) **Drone-NodeJS**, com os ficheiros *python* necessários para inicializar o *log* de comunicação entre o *drone* e o *node controller*.
3) **OpenCV**, onde está localizada a versão completa do nosso algoritmo de imagem e *watch folder*.
4) **Restful**, contem o projeto java de criação dos nossos serviços.
5) **RabbitMQ-Client**, exemplo de comunicação em *Java*, através de mensagens utilizando o *broker RabbitMQ*.
6) **drone-client,** exemplo em *Java*, de um *POST* no serviço *REST* do projeto com imagens de uma captura.
7) **mySQL**, contem a base de dados do nosso projeto.
8) **pappers**, fornecidos para consulta de possíveis soluções de análise de imagem.
9) **restcpp**, contem exemplos de utilização de serviços *REST* em c++.
10) **watchfolder**, onde são fornecidos exemplos de criação de possíveis *watch folders*.
11) **Projecto**, versão final do nosso projeto, que contem todos os ficheiros necessários para execução de todas as tarefas.

Utilizamos também a wiki [6] do code.ua.pt para o nosso projeto com uma estrutura bem concebida pelo que a pesquisa na mesma é intuitiva.

7. Aplicação web

A aplicação web foi desenvolvida com recurso a *html, css* e *javascript*, e utiliza a f*ramework angularjs* para invocar os serviços *REST*. Para algumas funcionalidades mais específicas recorremos também à *jquery framework*, que simplifica as operações em *javascript*. A aplicação web divide-se nas seguintes áreas:

1) Área dos parques. Listagem de parques monitorizados, com indicação visual da ocupação. Ao clicar em cada parque, é possível consultar mais detalhes sobre o parque, nomeadamente um gráfico de ocupação, a sua área de ocupação com recurso ao google maps, e uma galeria de fotos com imagens da última captura efetuada ao parque pelo drone.
2) Área das estatísticas. Listagem de capturas efetuadas, com filtro por data e parque. Ao clicar numa captura podemos consultar o detalhe da captura com a galeira de fotos.
3) Área reservada com:
    a) Área protegida por credenciais, que permite ao gestor, efetuar a gestão de parques e gerir as atividades do *drone*.
    b) Gerir atividades do *drone*: escolhe o parque e o *drone* disponível, e clica em iniciar captura.
    c) Gestão de parques: listagem dos parques, edição do parque, e criação de novos parques.

7.1. Base de Dados

Para gestão de informações existentes na nossa aplicação, utilizamos a base de dados *mysql*.

Dado que o projeto utiliza o modelo de *Object-relational mapping*, a base de dados, pode ser gerada automaticamente pelo projeto "*Maven Web Application*" colocando no ficheiro *persistence.xml* a seguinte *tag*:

<property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>

Para o projeto criado foi gerado por Java Persistence API o seguinte modelo:

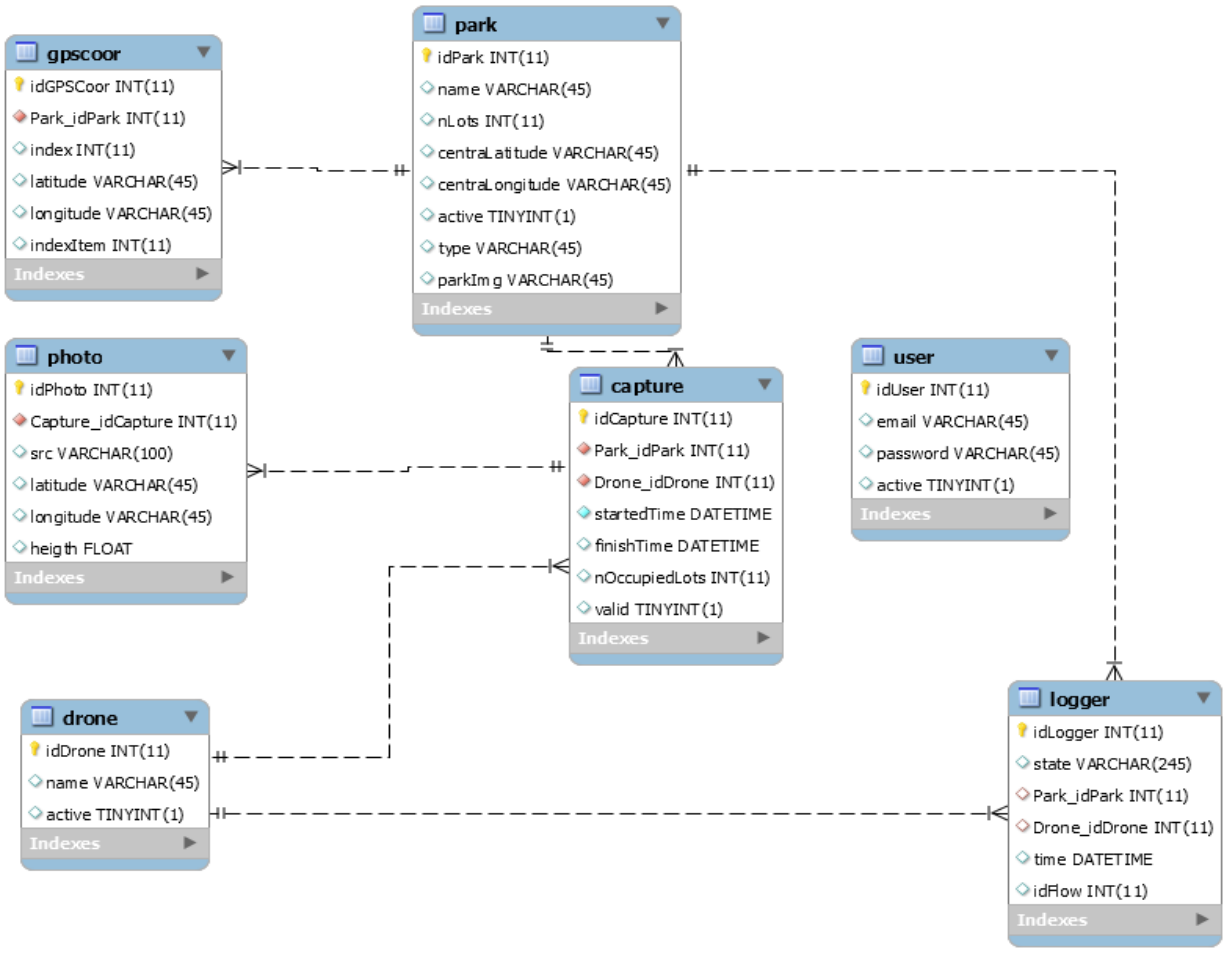


Figura 9: Modelo de base de dados

7.2. Serviços RESTful

Os serviços *REST*, que são utilizados pelas diferentes componentes do projeto foram desenvolvidos em *Java*, utilizando a *framework spring*.

Estes serviços são utilizados pelo *website* para disponibilizar informação, e efetuar a gestão dos parques. Pelo serviço de análise de imagem que após conclusão envia a informação através dos serviços *REST* referente as capturas “imagens e número de lugares ocupados nos parques de estacionamentos analisados” efetuadas pelo *drone*. O *script* de comunicação com o *drone*, utiliza o serviço *REST*, para fazer um *log* das atividades do *drone*, que são divulgadas na área reservada do site.

7.3. RabbitMQ

Para troca de mensagens entre o servidor e o computador portátil que utilizamos para comunicação com o drone, utilizamos o software rabbitMQ. Como não é possível dentro da Universidade de Aveiro, a utilização de uma conta na cloud, tivemos que instalar um servidor do rabbitMQ localmente. Este software está instalado no servidor virtual.

Discussão

Durante o projeto tivemos que investigar bastante sobre os caminhos a percorrer. Por vezes as decisões não foram as mais corretas o que nos levou a explorar melhores opções e assim obter melhores resultados. A solução a que chegamos é do nosso agrado visto que conseguimos alcançar as metas definidas.

Futuramente achamos que o projeto poderá ser melhorado. A integração de um *Raspberry PI* no *drone*, seria uma ideia a ter em conta. Uma vez que poderia tornar o processo de análise da imagem em tempo real. O *drone* poderia atualizar os valores do nosso *website* quase que instantaneamente e desta forma melhorar o *feedback* aos utilizadores.

Conclusão

Com o desenvolvimento deste projeto tivemos a oportunidade de aplicar os conhecimentos adquiridos ao longo do curso bem como explorar novas tecnologias e outras linguagens de programação existentes no mundo da informática, sendo que foi uma mais valia para nós.

Tivemos algumas dificuldades na construção do projeto que nos foi proposto, bem como a junção de todas as componentes. Mas depois de muito trabalho e pesquisa foi possível resolver todos os desafios encontrados, e consequentemente, por em prática uma estratégia mais elaborada para chegar à solução.

A exploração e programação do *drone* foi algo interessante, tendo em conta que é algo recente no mundo da tecnologia. Gostamos bastante do trabalho desenvolvido e estamos confiantes que é um projeto a que se deverá dar continuidade.

Referências

[1] Dji. (2016). *Parking Inspection*. Retrieved 24 Fevereiro, 2016, from https://developer.dji.com/showcase/parking-inspection/

[2] Parrot. (2016). *Developer Samples*. Retrieved 2 Março, 2016, from https://github.com/Parrot-Developers/Samples

[3] Man7org. (2016). *INOTIFY(7), Linux Programmer's Manual* . Retrieved 20 Abril, 2016, from http://man7.org/linux/man-pages/man7/inotify.7.html

[4] Manuel Camarneiro, "Deteção *de Veículos em Imagens Obtidas por Drones*", MsC Thesis, University of Aveiro, 2016

[5] Parrot. (2015). *Flight plan update*. Retrieved 15 Março, 2016, from http://blog.parrot.com/2015/10/21/flight-plan-update-available/

[6] Codeua. (2016). *Monitorização usando drones*. Retrieved 20 Fevereiro, 2016, from https://code.ua.pt/git/monitorizacao-dos-parques-de-estacionamento-da-ua-usando-drones

[7] Spring. (2016). *Spring*. Retrieved 8 Março, 2016, from https://spring.io

[8] Parrot. (2016). *Parrot Bebop 2*. Retrieved 2 Março, 2016, from http://www.parrot.com/usa/products/bebop2

[9] Hybridgroup. (2016). *Node-bebop*. Retrieved 18 Março, 2016, from https://github.com/hybridgroup/node-bebop

[10] Parrot. (2016). *Parrot Developers*. Retrieved 2 Março, 2016, from http://developer.parrot.com/

[11] Rabbitmq. (2016). *Downloading and Installing RabbitMQ*. Retrieved 6 Abril, 2016, from https://www.rabbitmq.com/download.html