IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Romeu Fernando Temudo Oliveira

dezembro | 2015

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

RELATÓRIO DE PROJECTO

FIREPROTECTION (SISTEMA DE MONITORIZAÇÃO DE VEÍCULOS EM TEMPO REAL)

ROMEU FERNANDO TEMUDO OLIVEIRA

RELATÓRIO PARA A OBTENÇÃO DO GRAU DE LICENCIADO

EM Engenharia Informática

Dezembro/2015

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

RELATÓRIO DE PROJECTO

FIREPROTECTION (SISTEMA DE MONITORIZAÇÃO DE VEÍCULOS EM TEMPO REAL)

ROMEU FERNANDO TEMUDO OLIVEIRA

RELATÓRIO PARA A OBTENÇÃO DO GRAU DE LICENCIADO

EM Engenharia Informática

Dezembro/2015

Coordenador : António Fernandes Gerente da empresa Explorinova

Orientador :Professor Doutor Carlos Carreto

Elementos Identificativos

Aluno

Nome: Romeu Fernando Temudo Oliveira

Número: 1009082

Curso: Engenharia Informática

Estabelecimento de Ensino

Nome: Escola Superior de Tecnologia e Gestão – Instituto Politécnico da Guarda

Morada: Av. Dr. Francisco Sá Carneiro 50, 6300-559 Guarda

Telefone: 271220120 | Fax: 271220150

Instituição Acolhedora do Estágio

Nome: Explorinova, Sociedade Unipessoal Limitada

Morada: Avenida Dr. Francisco Sá Carneiro, Nº 50, Instituto Politécnico da Guarda, Policasulo Nº 5, 6300-559 Guarda

Duração do Estágio

Início: 15 de Junho de 2015

Fim: 15 de Setembro de 2015

Coordenador do Estágio:

Nome: António Fernandes

Título: Gerente

Orientador do Estágio

Nome: Carlos Carreto

Grau: académico: Doutor

Resumo

Este relatório descreve o trabalho realizado no âmbito da unidade curricular Projeto de Informática, na Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda. Este projeto foi desenvolvido em contexto de estágio na empresa Explorinova, com o tema FireProtection.

O principal objetivo da aplicação é desenvolver um sistema automatizado para simplificar as tarefas das corporações de bombeiros no combate a incêndios. Esta aplicação pretende implementar um sistema de segurança com a possibilidade de ser calibrado em qualquer viatura e alertar o condutor de forma a evitar acidentes no percurso das viaturas das corporações, um sistema capaz recolher dados presentes no ambiente e apresenta-los de forma automatizada, um sistema de monitorização em tempo real e coordenadas geográficas, a capacidade de armazenar todos os dados recolhidos, e a possibilidade de o sistema apresentar mobilidade e escolha na instalação.

Esta aplicação foi desenvolvida com a linguagem Java na plataforma Android Studio IDE para o dispositivo móvel e com a linguagem C na plataforma Arduino IDE para o sistema embebido.

Palavras-chave: Android, Arduino, App, Monitorização de veículos

Abstract

This report describes the work developed for the course Projeto de Informática, for the bachelor's degree in Computer Engineering of the Escola Superior de Tecnologia e Gestão in the Instituto Politécnico da Guarda. This project was developed in context of internship for the "Explorinova" company, with the theme "FireProtection".

The application main objective is to develop an automated system to simplify the corporations firefighters tasks when dealing with fires. This application intends to implement an active security system with the possibility of calibration on any vehicle and to alert the driver and, consequently, avoid accidents on the routes of the vehicles, a system capable of gathering data present in the environment and display it in an automated fashion, a real-time monitoring system and geographical coordinates, the ability to storage every data collected, and the system's possibility to display mobility and choice in the installation

This application was developed using Java programming in the Android Studio IDE platform for mobile devices and using C programming in the Arduino IDE platform for the embedded system.

Keywords : Android,Arduino,App,Vehicle monitoring

Agradecimentos

Gostaria de agradecer:

Ao António Fernandes fundador e coordenador da empresa Explorinova por toda a sua disponibilidade e por todo o material facultado durante a realização do projeto.

Ao Professor Doutor Carlos Carreto por ter aceitado o desafio de me orientar neste projeto, os seus conhecimentos foram importantes para o desenvolvimento do projeto.

A todos os docentes não referidos pelos conhecimentos transmitidos ao longo de todo o curso.

Também um agradecimento á minha família, em especial aos meus pais por todo o esforço e ajuda ao longo destes anos.

Glossário

Acess point – É um dispositivo presente numa rede sem fios que realiza a ligação entre dispositivos.

Array – É um objeto que detêm um determinado número de valores do mesmo tipo, o tamanho do array é determinado quando é criado.

Buffer – Tem a função de guardar dados para serem editados ou processados posteriormente.

Checksum – É um algoritmo usado para verificar a integridade de dados transmitidos.

Cloud – Sistema de armazenamento de dados na internet, o acesso a programas, serviços.

GPS - É a sigla para *Global Positioning System*, o GPS é um sistema de navegação por satélite.

Hanlder – É uma rotina utilizada para gerir *inputs* recebidos por um determinado programa.

IDE – É a sigla para Integrated Development Environment , o IDE é um conjunto de ferramentas de programação para o desenvolvimento de aplicações.

Interrupt – É um sinal enviado ao processador que indica um evento.

Launcher – É o ecran inicial de uma aplicação em Android.

LCD – É a sigla para *Liquid Crystal Display*, o LCD é um aparelho de transmissão de imagem através de cristais líquidos.

Manifesto Android - É um ficheiro com a informação essencial em relação à aplicação desenvolvida para com o sistema Android.

Main activity – É o ecran principal de uma aplicação em Android.

Shield – É uma placa que pode ser adicionada ao Arduino para acrescentar funcionalidades.

Sockets – É uma interface de comunicação bidirecional entre processos através de uma rede de computadores, os sockets são a base de comunicação em redes TCP/IP.

String Builder – É um objeto capaz de modificar internamente uma *string*.

Thread – É um fluxo sequêncial de controlo dentro de um programa.

USART – É a sigla para Universal Synchronous/Asynchronous Receiver/Transmitter, o USART é uma interface de comunicação.

UUID – É a sigla para Universal Unique Identifier, o UUID é um identificador padrão usado na construção de software. O UIID é um código formado por 128 bits

Zigbee – É um protocolo de comunicação sem fios de baixo custo e potência.

Índice Geral

Índice de Figuras

1. **Introdução**

Este capítulo pretende apresentar em que âmbito este projeto pode ser enquadrado, quais os problemas que levaram á necessidade da elaboração de um projeto, bem como todos os objetivos e soluções que pretendem ser implementadas.

O presente relatório descreve o projeto desenvolvido no âmbito da unidade curricular Projeto de Informática, na Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda, em parceria com a empresa Explorinova.

A explorinova é uma empresa situada nos poli-casúlos da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda que desenvolve *software* para solucionar problemas presentes na área da agricultura e indústria. A empresa utiliza sempre tecnologia (hardware e software) desenvolvida na própria empresa. A marca FireProtection engloba vários projetos já desenvolvidos, o projeto desenvolvido é um módulo da FireProtection e pretende solucionar os problemas encontrados pelas viaturas de reconhecimento dos bombeiros.

Com a evolução da Informática e a expansão das tecnologias móveis é pretendido criar uma aplicação capaz de monitorizar vários fatores ambientais e geográficos de forma automatizada.

Neste caso particular pretende-se equipar um veículo com um dispositivo móvel e um sistema embebido para recolha de dados de interesse nas atividades das corporações de bombeiros.

O projeto consiste numa aplicação simples e *user-friendly*. Esta pretende dar segurança ativa às viaturas das corporações em relação ao declive da viatura, bem como recolher e armazenar dados de interesse no combate a incêndios, que potencialmente podem ser usados para criar um serviço de alerta para corporações de bombeiros.

1.1 Enquadramento e motivação do projeto

Este projeto enquadra-se em situações de emergência, na necessidade de ter acesso e avaliar todas as condições envolventes no ambiente para garantir uma avaliação rápida e eficaz destas.

Esta aplicação foi desenvolvido para a empresa Explorinova após o contacto da corporação de bombeiros da Guarda em que foi exposto a dificuldade sentida em certas situações de emergência.

A motivação pessoal para o desenvolvimento desta aplicação é a preocupação cívica de facultar melhorias no desempenho deste tipo de serviços.

1.2 Definição e Solução do Problema

Tipicamente, em uma rota de um serviço de emergência, neste caso específico para a viatura de reconhecimento, várias dificuldades podem surgir:

1. Em certas situações de emergência onde as condições do terreno são precárias e os veículos estão propícios a declives íngremes e posteriormente ao despiste ou a capotarem;
2. Na chegada da viatura e por motivos de segurança e eficácia no desempenho das corporações de bombeiros é imperativo haver uma recolha precisa e automatizada de valores de dióxido de carbono, humidade, temperatura e vento.
3. As potenciais localizações do sistema na viatura requerem que este tenha em atenção a mobilidade e logística;
4. No combate a incêndios a colaboração é um elemento chave para uma melhor organização e gestão de recursos. A obtenção das coordenadas geográficas de todas as viaturas seria ideal na decisão estratégica ao combate de incêndios.

Para resolver estes problemas desenvolveu-se uma aplicação que consiste num sistema embebido e uma aplicação móvel que aplicam os seguintes métodos:

1. Alertar o utilizador da estabilidade do veículo;
2. Recolha de dados de monóxido de carbono, humidade, temperatura e vento. Os dados são recolhidos de forma cíclica e automatizada.

3. Utilização da comunicação sem fios, para permitir a instalação do dispositivo em qualquer lugar;
4. Monitorização da posição geográfica da viatura em tempo real.

1.4 Objetivos

A solução desenvolvida pretende implementar as seguintes funcionalidades:

- Um sistema embebido capaz de recolher concentrações de monóxido de carbono;
- Recolher a percentagem de humidade no ar;
- Recolher a temperatura em graus;
- Recolher a velocidade e direção do vento;
- Transmitir os dados recolhidos pelos sensores;
- Comunicação através de um módulo Bluetooth;
- Apresentar os dados recebidos numa trama de dados;
- Monitorização em tempo real recorrendo à API da Google;
- Segurança da viatura recorrendo ao sensor acelerómetro incorporado no dispositivo móvel;
- A possibilidade de calibrar o sistema de segurança ativa;
- Armazenar os dados tratados no cartão SD do dispositivo móvel.

1.5 Organização do Relatório

O documento compreende 5 capítulos, para além da Introdução e de um capítulo de conclusões. No capítulo2 é apresentado algum trabalho relacionado, com referência a algumas aplicações já existentes no mercado com objetivos semelhantes ao sistema desenvolvido. No capítulo 3 é apresentado o processo de desenvolvimento de *software* que foi seguido para desenvolver o sistema. No capítulo 4 é apresentado o *software* e *hardware* utilizado para o desenvolvimento do projeto e a implementação dos seus módulos. No capítulo 5 é apresentado os testes efetuados em alguns módulos e os resultados obtidos. Finalmente, no capítulo 6 é feita uma análise do projeto bem como as conclusões mais importantes do projeto e implementações futuras.

2. Trabalho Relacionado

Este capítulo apresenta projetos semelhantes ao projeto desenvolvido. Como este projeto pretende responder a necessidades específicas e engloba vários módulos, não é possível apresentar projetos semelhantes de forma global, sendo então apresentados projetos com módulos semelhantes.

2.1 Sistema de monitorização remota de temperatura e humidade de câmaras de frio

O projeto MTH (sistema de monitorização remota de temperatura e humidade de câmaras de frio) cuja arquitetura é apresentada na Fig 1 tem o objetivo de monitorizar a temperatura e humidade de câmaras de frio, através de sensores incorporados num controlador Arduino [1]. Para uma melhor compreensão o projeto será denominado MTH.

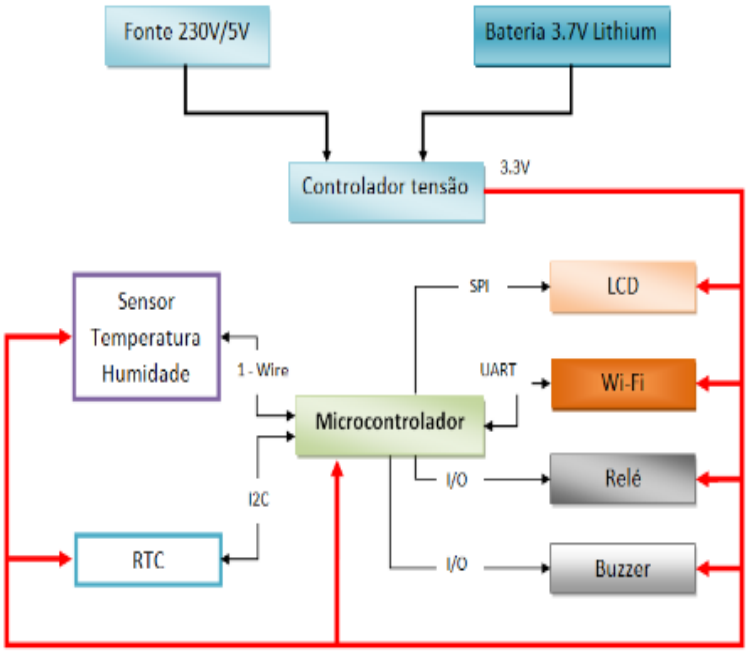

Figura 1- Arquitetura MTH (adaptado de [1])

O projeto MTH apresenta vários módulos idênticos à nossa aplicação desenvolvida. O sensor de humidade e temperatura usado no projeto MTH é o sensor RHT03, este sensor é mais antigo que o sensor DHT22 usado na nossa aplicação, mas o seu funcionamento é praticamente igual. Para a visualização em tempo real dos valores obtidos pelo sensor, o projeto MTH usa um LCD incorporado no controlador, enquanto a nossa aplicação desenvolvida apresenta os valores em um dispositivo móvel devido às necessidades do projeto e a evolução das tecnologias móveis. O projeto MTH implementa um módulo de comunicação WI-FI para enviar os dados para um servidor, a nossa aplicação implementa um módulo de comunicação mas a comunicação é feita via Bluetooth. A comunicação Wi-Fi é uma comunicação mais robusta e com maior alcance que a comunicação Bluetooth, mas é uma comunicação que tem mais significado quando nos ligamos a um *ponto de acesso* com ligação á internet para enviar dados para servidores via web, como a viatura dos bombeiros não está equipada com um *router* com acesso à internet e o controlador Arduino não se encontra a uma grande distancia do dispositivo móvel optou-se pela comunicação Bluetooth.

2.2 Sistema de aquisição de dados para monitorização de transportes de carga

O projeto MTC (sistema de aquisição de dados para monitorização de transportes de carga) cuja arquitetura é apresentada na Fig 2 tem o objetivo de monitorizar o transporte de cargas frágeis, através de sensores incorporados no controlador Arduino [2]. Para uma melhor compreensão o projeto será denominado MTC.

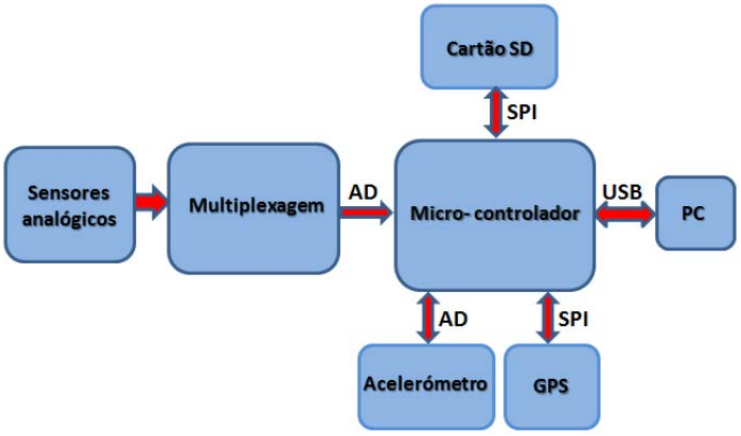

Figura 2-Arquitetura MTC (adaptado de [2])

As cargas frágeis estão sujeitas a danos devido as condições das vias de comunicação ou á condução do veículo. O projeto MTC usa sensores de força resistiva, GPS e acelerómetro, esses sensores estão incorporados no controlador Arduíno. A nossa aplicação é extremamente semelhante ao projeto MTC devido ao tipo de componentes utilizados, a nossa aplicação não tem a necessidade de incorporar um sensor GPS ou um acelerómetro no Arduíno, porque o dispositivo móvel já tem estes sensores incorporados. O projeto MTH trata todos os dados no controlador Arduíno enquanto a nossa aplicação trata os dados dos sensores no controlador Arduíno e os dados do GPS e do Acelerómetro no dispositivo móvel. Para armazenamento de dados o projeto MTH integra um *shield* para permitir o armazenamento da informação num cartão SD, a nossa aplicação armazena igualmente os dados recolhidos num cartão SD mas esse armazenamento é tratado no dispositivo móvel, esses dados são enviados através de um módulo Bluetooth, tratados e posteriormente guardados no cartão SD do dispositivo.

2.3 Sistema de aquisição e monitorização de dados para industria alimentar e laboratorial

O projeto MIA (sistema de aquisição e monitorização de dados para a industria alimentar e laboratoriais) tem o objetivo de adquirir e monitorizar as condições de armazenamento de produtos, sobretudo nas áreas laboratoriais de análises clínicas e restauração[3]. Para facilitar a leitura do texto este projeto será denominado MIA. O projeto MIA pretende monitorizar as condições ambientais e de conservação de amostras de análises clinicas bem como produtos alimentares. O projeto MIA usa sensores de temperatura, humidade e de luminosidade, esses sensores são incorporados no Arduino. O sensor de temperatura e humidade usado no projeto MIA é o sensor DH11, este sensor é uma versão antiga do sensor DH22 usado na nossa aplicação. O projeto MIA tem também a funcionalidade de enviar dados através de uma comunicação sem fios, a comunicação sem fios utilizada foi *Zigbee* enquanto a comunicação na nossa aplicação é feita através de Bluetooth, como já foi referido. Os dados no projeto MIA são armazenados num sistema de cloud e posteriormente podem ser acedidos em uma página web. Este projeto armazena a informação no cartão SD, mas a implementação de um método de armazenamento de dados na *cloud* foi tido em conta para trabalho realizado no futuro.

3. Processo de Desenvolvimento de *software*

Este capítulo apresenta todos os processos necessários para o desenvolvimento do projeto. É definido neste capítulo a necessidade de optar por um processo de desenvolvimento de *software*, em que consiste cada etapa e as tarefas realizadas em cada uma dessas etapas.

3.1 Etapas e Atividades

O processo de desenvolvimento de *software* consiste na estruturação de um conjunto de atividades pretendidas para o desenvolvimento do sistema de *software [4]*. Devido à complexidade e vários módulos envolvidos no projeto, não foi seguida uma metodologia específica, mas sim uma estrutura de atividades necessárias para o desenvolvimento do sistema. A estrutura de atividades pode ter diferentes processos de *software* mas, todas estão envolvidas nas etapas e iterações apresentadas na, Figura 3.

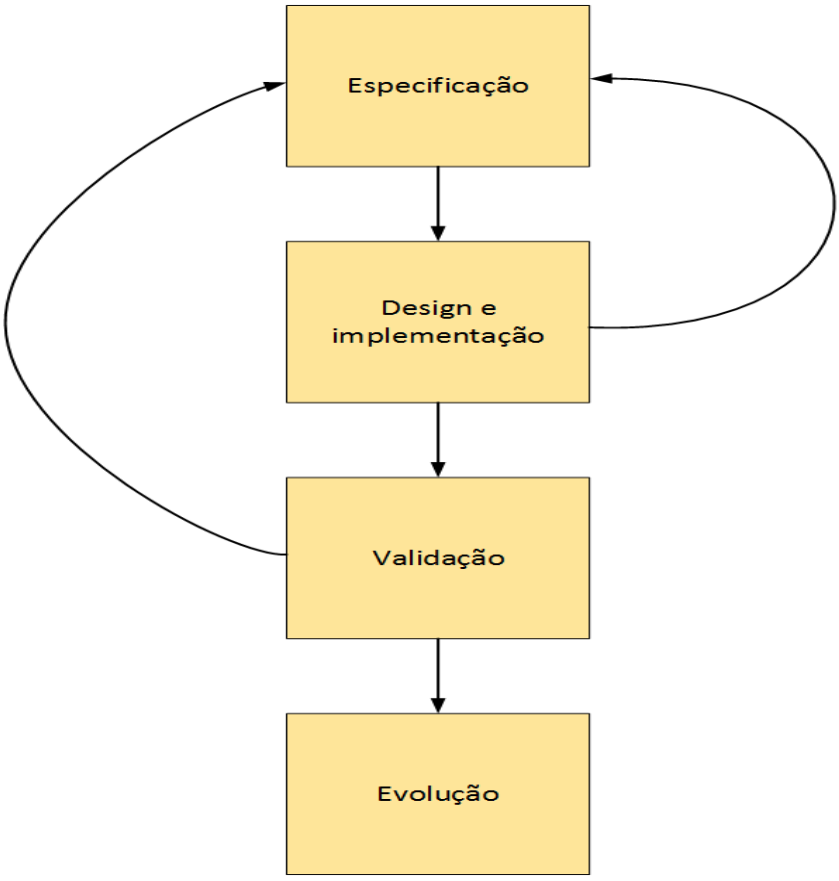

Figura 3 - Etapas do processo de software

Especificação – Esta etapa pretende definir as funcionalidades e os objetivos do sistema. As reuniões com a empresa Explorinova foram muito importantes para entender o que era pretendido pela empresa e também para debater ideias e que passos deveriam ser seguidos para a aplicação ir de encontro às necessidades do cliente.

Design e Implementação – Esta etapa define a organização do sistema e a implementação do sistema. Esta etapa foi a mais importante e a mais longa do projeto devido aos vários módulos que este projeto envolve, para isso foi necessário dividir o projeto em 2 arquiteturas distintas sendo elas a arquitetura de *software* e *hardware*. Para definir a organização e implementação da arquitetura de *hardware* foi necessário definir que sensores iam ser usados, que Arduíno se adaptava as nossas necessidades, qual a melhor forma de comunicação entre o Arduíno e o dispositivo móvel e para que plataforma seria desenvolvida a aplicação móvel. Esta etapa foi desenvolvida em conjunto com a empresa Explorinova. A organização e implementação da arquitetura de *software* foca-se na parte lógica do projeto, para isso foi desenvolvido pseudo código para implementar as funcionalidades de todo o *hardware*. Foi necessário implementar bibliotecas para as leituras dos sensores, como essas leituras seriam enviadas, definir como seria feita a comunicação Bluetooth, tratar os dados recebidos e apresentá-los na forma de uma trama de dados em formato usado na indústria, definir de que forma seria implementado o módulo GPS e como seria implementada a segurança ativa da viatura.

Validação – Esta etapa verifica se o sistema preenche todos os requisitos da etapa de especificação. A validação do projeto foi aplicada individualmente a cada componente implementado e só numa fase mais tardia o projeto foi validado num todo. Devido à interação com a empresa Explorinova durante a especificação e implementação, as alterações ocorridas na validação não foram significativas.

Evolução – Esta etapa pretende garantir a manutenção e evolução do sistema consoante as necessidades do cliente. Como este projeto foi desenvolvido em contexto de estágio para a empresa Explorinova a manutenção e evolução do sistema serão realizados pela empresa.

4.Desenvolvimento do projeto

Neste capítulo vai ser apresentada toda a arquitetura do sistema desenvolvido. Como o sistema desenvolvido envolve módulos físicos e lógicos será feita uma apresentação individual desses módulos para uma melhor compreensão. Este capítulo pretende apresentar todos os elementos, inicialmente de uma forma geral e posteriormente de uma forma detalhada A implementação de todos os componentes é também apresentada neste capítulo.

4.1 Arquitetura do sistema desenvolvido

O projeto desenvolvido foi divido em duas arquiteturas, a arquitetura de *software* e a arquitetura de *hardware*. A arquitetura de *software* é composta por todas as partes lógicas do projeto, enquanto a arquitetura de *software* é composta por todas as componentes físicas envolvidas no projeto. A figura 4 apresenta a arquitetura do sistema.

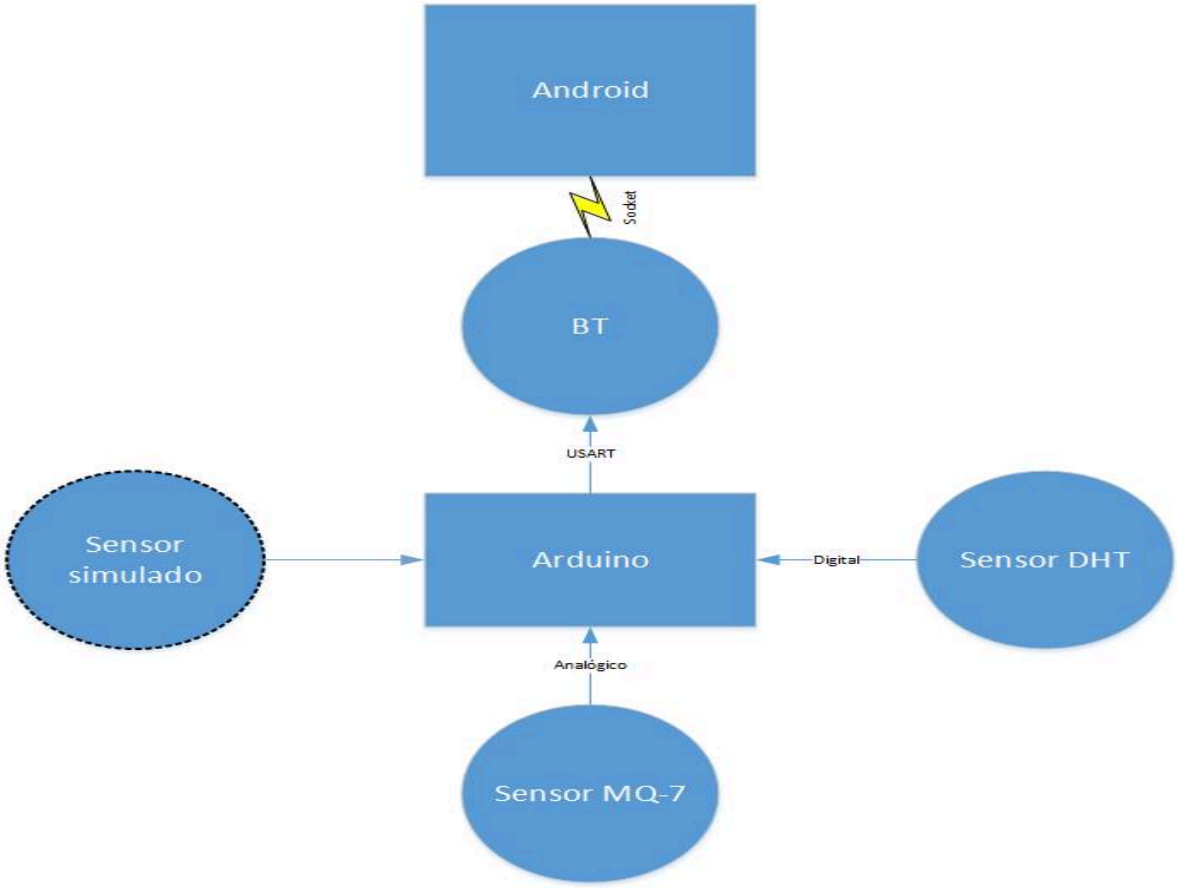

Figura 4 - Arquitetura do sistema

Para uma melhor compreensão da arquitetura geral vai ser feita uma pequena introdução sobre as funcionalidades de cada componente.

Arduíno

O Arduíno é uma plataforma de prototipagem eletrónica de *hardware* livre projetada num microcontrolador com suporte a entrada/saída de portas digitais ou analógicas embutidas. O Arduíno tem um IDE próprio e uma linguagem de programação padrão. O Arduíno pretende recolher e tratar toda a informação recolhida pelos sensores.

No anexo A encontra-se todo o código utilizado pelo Arduino

Sensor MQ-7

O Sensor MQ-7 é responsável por medir os níveis de dióxido de carbono presentes no ambiente [5]. Este sensor pretende solucionar o problema encontrado pelas

corporações de bombeiros em detetar os níveis de dióxido de carbono presentes no ambiente.

Sensor DHT

O sensor DHT é responsável por medir a temperatura e humidade do ambiente [6]. Este sensor tem a particularidade de medir duas variáveis de ambiente num único sensor. Este sensor pretende solucionar o problema encontrado pelas corporações de bombeiros em detetar os níveis de temperatura e humidade presentes no ambiente.

Sensor simulado

O sensor simulado pretende simular um sensor de vento. Devido à falta de um sensor de vento os dados apresentados vão ser simulados através de valores aleatórios mas muito próximos aos valores obtidos por um sensor de vento real.

BT

O bloco BT, implementa a comunicação entre o controlador Arduino e o dipositivo móvel Android. A comunicação entre os dispositivos é feita sem fios e é implementada através de um módulo Bluetooth através da norma USART. O Bluetooth é um protocolo de comunicação projetado para consumos baixos de energia e para comunicação entre distâncias pequenas. O sistema Bluetooth possibilita a comunicação entre dispositivos quando estes se encontram dentro de um raio de alcance e a sua comunicação é feita via rádio. A implementação da comunicação Bluetooth permite garantir mobilidade ao sistema.

Android

O bloco Android é a representação do dispositivo móvel. Este dispositivo móvel pode representar um *smartphone* ou *tablet* com sistema operativo Android. Android é um sistema operativo baseado em Linux e que permite desenvolver aplicações de forma livre. A aplicação desenvolvida pretende ser instalada e executada em qualquer dispositivo Android.

4.2 Circuito do sistema desenvolvido

Para uma melhor análise e compreensão da componente física do sistema embebido será analisada a Figura 5 que representa todo o circuito do sistema

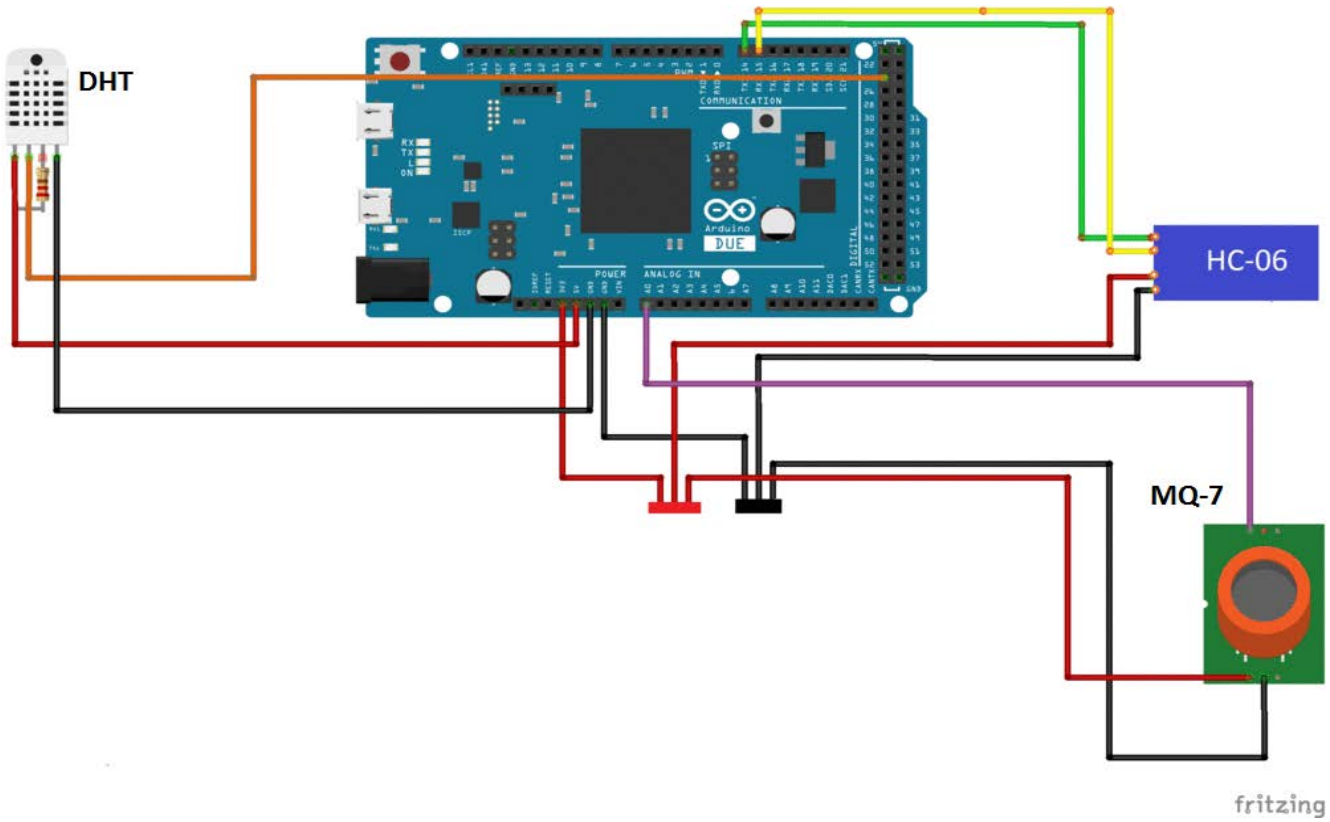

Figura 5 - Arquitetura Arduino

Controlador Arduino Due

O controlador Arduino Due possui um microcontrolador ATMEL SAM3SX8E com capacidade para operar a velocidades de processamento até 84 MHz. O Arduino Due possui 512KB de memória Flash e 96KB de memória SDRAM que aumentando os recursos para programas maiores e mais complexos. O Arduino Due possui também uma grande diversidade e quantidade de portas, garantindo a prospeção do projeto. Ao contrário da maior parte dos Arduinos, a placa do Arduino Due funciona a 3.3V em vez de 5V [7].

Sensor DHT

O sensor DHT é composto por 4 pinos; 2 pinos responsáveis pela alimentação, sendo 1 de voltagem 5V (ligação a vermelho) e o outro de 0V (ligação a preto). Este sensor tem a particularidade de usar uma resistência ligada no pino 3 diretamente ao

pino de 5V para um correto funcionamento do sensor. O pino 2 (ligação a laranja) está ligado ao pino digital 24, este pino é responsável por receber os dados do sensor.

Sensor MQ-7

O sensor MQ-7 é composto por 4 pinos, mas apenas são usados 3 pinos para o funcionamento do sensor, em que 2 pinos são responsáveis pela alimentação, sendo 1 de voltagem 3.3V (ligação a vermelho) e o outro de 0V (ligação a preto).O pino 3 (ligação violeta) é ligado ao pino analógico A0 responsável por receber os dados do sensor.

Bluetooth HC-06

O módulo Bluetooth HC-06 é composto por quatro pinos; 2 pinos responsáveis pela alimentação do módulo, sendo 1 de voltagem 3.3V (ligação a vermelho) e o outro de 0V (ligação a preto). Os dois pinos restantes especificamente RX (ligação a verde) e TX (ligação a amarelo) são responsáveis pela transmissão de dados, estes pinos são ligados aos pinos Tx14 e Rx15 no controlador Arduino com a particularidade de serem ligados cruzados [8].

4.3 Arquitectura de *software*

A arquitetura de *software* do sistema está dividida em duas partes, a parte referente ao Arduino e a parte referente ao Android, estas duas partes complementam-se e fazem parte da arquitetura geral de *software*, mas para uma melhor compreensão vão ser analisadas separadamente. A seguinte Figura 6 apresenta a arquitetura de *software* do sistema:

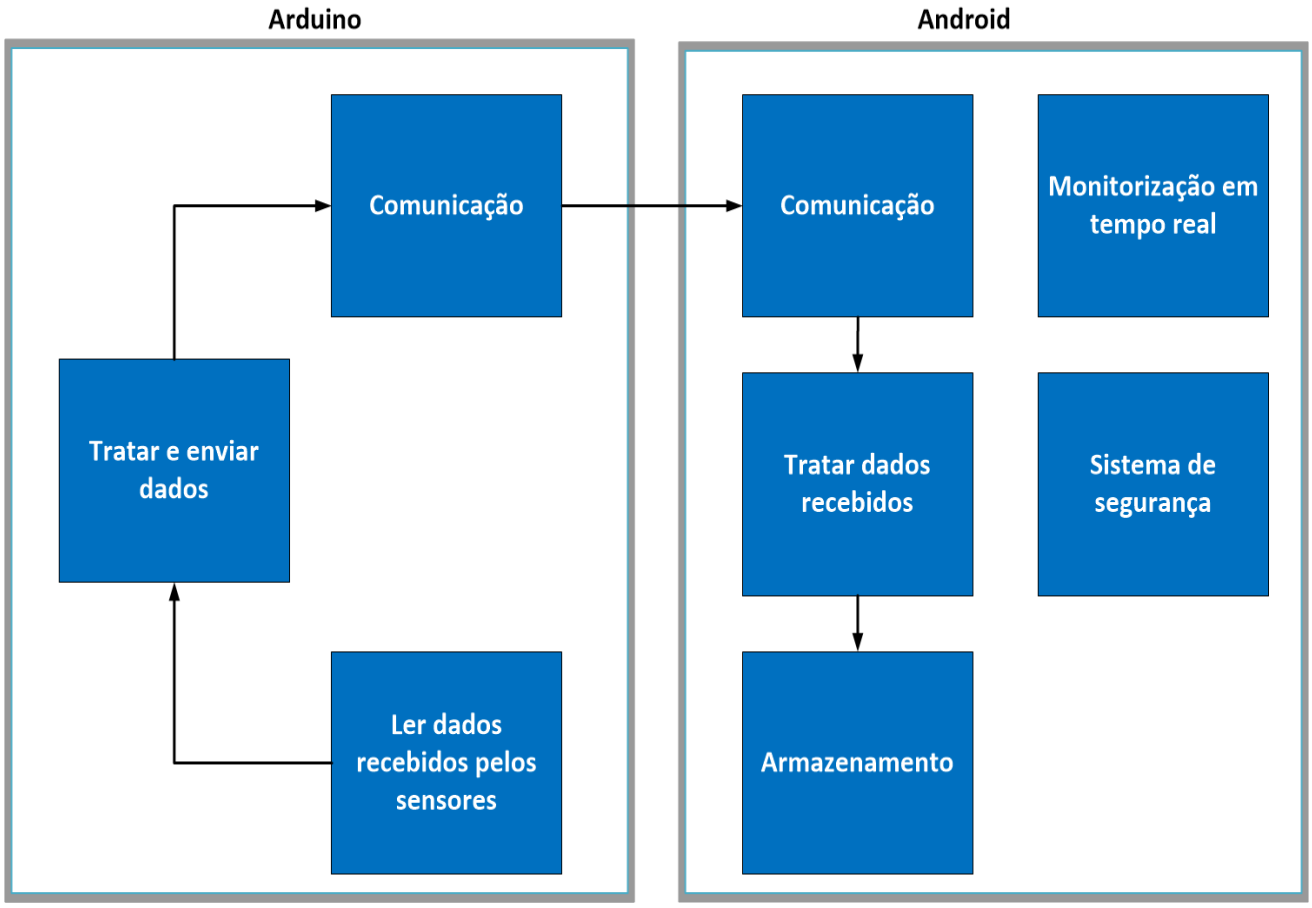

Figura 6 - Arquitetura de software

4.3.1 Arquitetura do Arduino

Todo o código desenvolvido no Arduino IDE encontra-se no Anexo A.

O bloco Comunicação é responsável pela comunicação sem-fios Bluetooth como já foi referido. A comunicação entre o Arduino e o dispositivo móvel é feita através da interface USART, o Arduino Due dispõe de 3 portas USART, neste caso a porta configurada para a comunicação será a porta USART3.

O bloco Ler dados recebidos pelos sensores é responsável por ler todos os dados recebidos dos sensores de CO2, humidade, temperatura e vento. Ao sensor de CO2 é atribuída uma variável para guardar todos os valores recebidos pelo sensor. Este sensor tem a particularidade de apenas apresentar leituras corretas depois de passar por um período de aquecimento, para evitar leituras erradas este sensor apresenta valores com uma determinada frequência. A biblioteca CS_MQ7.h disponibiliza as funções necessárias para o funcionamento do sensor e o seu código encontra-se no anexo B. O sensor de humidade e temperatura tem a particularidade de recolher os valores da humidade e temperatura em simultâneo, a mesma função envia dois valores distintos que são guardados em duas variáveis. A biblioteca DHT.h disponibiliza as funções necessárias para o funcionamento do sensor, o código da biblioteca encontra-se no anexo C. Finalmente, devido há falta de um sensor de vento, serão criados valores aleatórios de forma a simular potências valores recolhidos por um sensor real. Estes sensores recolhem normalmente a velocidade e a direção do vento, cujos valores são guardados em duas variáveis.

O bloco Tratar e enviar dados é responsável por o tratamento de todos os dados recebidos através dos sensores, bem como o envio dos mesmos. Em primeiro lugar todos os dados são guardados num *array* de dados para serem enviados todos ao mesmo tempo através de um ciclo. Como os dados pretendem ser apresentados individualmente, posteriormente é necessário definir um método para identificar quando começa e acaba cada valor único. Para isso todos os valores são separados por um caracter especial. Como o programa Arduino envia tramas de forma cíclica é necessário também definir quando cada trama única é finalizada, o método utilizado para separar as tramas foram os seguintes caracteres especiais: # e ~.

O sistema pretende implementar um elemento de validação para confirmar se os dados enviados não resultaram em perdas durante a transmissão Bluetooth. Esse elemento consiste em somar todos os valores dos sensores e guardá-los numa variável e enviar o valor na trama.

4.3.2 Arquitetura do Android

Todo o código desenvolvido no Android Studio encontra-se no Anexo D.

O bloco Comunicação pretende criar todas as funcionalidades para a comunicação via Bluetooth. O bloco de comunicação BT no Arduino apenas envia os dados através da porta configurada, enquanto no Android é necessário implementar os procedimentos para a receção dos dados. O procedimento passa por várias etapas, essas etapas são:

- Encontrar e criar uma lista de dispositivos para emparelhar; caso o dispositivo não tenha o Bluetooth ligado será enviada uma mensagem para o utilizador permitir ligar o Bluetooth.
- Guardar o endereço do dispositivo emparelhado.
- Criar uma ligação segura através de *sockets* e UUID.
- Criar uma thread em *background* para estar sempre a receber dados e enviar os dados para a thread principal através de um *handler*.

Como um dos principais objetivos da aplicação é comunicar com o Arduino e receber os valores enviados dos sensores, a aplicação móvel é composta por um *launcher* que apenas permite o acesso à *main activity* caso a ligação Bluetooth tenha sido efetuada com sucesso.

O bloco Monitorização em tempo real implementa a aplicação Google Maps V2 da Google. Para usar esta API desenvolvida pela Google é necessário criar uma conta Google *console* e agregar o projeto a essa conta. Depois de agregado o projeto à conta é obtida uma chave de autenticação para usar a API Google Maps V2. Este bloco tem as funcionalidades de apresentar a localização exata da viatura em um mapa bem como apresentar as coordenadas geográficas longitude e latitude. O dispositivo móvel necessita obrigatoriamente de estar equipado com um sistema de GPS interno para implementar este bloco.

O bloco Armazenamento de dados tem a funcionalidade de guardar os dados dos sensores, a data e hora da leitura, as coordenadas geográficas e os parâmetros de início e fim da trama de dados. O armazenamento de dados é efetuado no cartão SD do dispositivo móvel num ficheiro de texto. A trama de dados guardada no cartão SD apresenta o seguinte formato:

EXS#sensor1#sensor2#sensor3#sensor4#longitude#latitude#data&hora#EXE

O bloco Sistema de segurança implementa a segurança da viatura. Este tem a funcionalidade de informar através de um aviso sonoro e de uma mensagem enviada para a aplicação o utilizador sobre o declive em que a viatura se encontra para prevenir acidentes. O sistema pode ser calibrado para que o dispositivo esteja em qualquer lugar da viatura. O sistema está configurado para que uma inclinação de 70º no eixo vertical e 45º no eixo horizontal representem uma situação de perigo para a viatura.

O bloco Tratar dados implementa a funcionalidade de tratar os dados recebidos e apresentá-los de forma personalizada. Os dados são recebidos no formato de uma trama de dados e é necessário definir quando começa e termina uma trama de dados, para evitar tratar várias tramas em simultâneo. Após obter a trama de dados é necessário também separar os valores de cada sensor e apresentá-los individualmente. A trama de dados é personalizada no formato usado na indústria. Os dados apenas são apresentados no sistema se o elemento de validação passar a condição que consiste em somar todos os valores recebidos e comparar se é igual à soma dos valores enviados.

4.4 Implementação

4.4.1 Comunicação BT

Este módulo implementa todas as funcionalidades necessárias para a comunicação via Bluetooth [8]. Na primeira utilização do dispositivo móvel com o módulo Bluetooth é necessário emparelhar os dispositivos manualmente. Para efetuar a comunicação é necessário em primeiro lugar verificar se o dispositivo móvel tem o Bluetooth ligado e se essa condição não se verificar é pedido ao utilizador se permite a ligação do Bluetooth. O seguinte código apresenta essa verificação.

```
private void checkBTState() {

        mBtAdapter=BluetoothAdapter.getDefaultAdapter();
        if(mBtAdapter==null) {
            Toast.makeText(getBaseContext(), "O dispositivo nao
suporta bluetooth", Toast.LENGTH_SHORT).show();
        } else {
            if (mBtAdapter.isEnabled()) {
                Log.d(TAG, "...Bluetooth ligado...");
            } else {

                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);

            }
        }
    }
```

Depois de o Bluetooth estar ligado é necessário criar um *array* de dados para apresentar todos os dispositivos Bluetooth existentes e apresentar a lista desses dispositivos disponíveis para se ligar. O seguinte código mostra como é criada a lista dos dispositivos disponíveis para ligação.

```
       mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this,
R.layout.device_name);

        ListView pairedListView = (ListView)
findViewById(R.id.paired_devices);

        pairedListView.setAdapter(mPairedDevicesArrayAdapter);

        pairedListView.setOnItemClickListener(mDeviceClickListener);

        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
        Set<BluetoothDevice> pairedDevices =
mBtAdapter.getBondedDevices();

        if (pairedDevices.size() > 0) {


findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);

            for (BluetoothDevice device : pairedDevices) {

                mPairedDevicesArrayAdapter.add(device.getName() + "\n"
+ device.getAddress());

      }
```

Após obtermos a lista de todos os dispositivos é possível escolher a qual deles a aplicação se vai ligar e guardar o endereço do dispositivo e prosseguir para a *main activity* da aplicação. O seguinte código demonstra como podemos retirar o endereço do dispositivo e prosseguir para *a main activity*.

```
private OnItemClickListener mDeviceClickListener = new
OnItemClickListener() {
            public void onItemClick(AdapterView<?> av, View v, int
arg2, long arg3) {

                textView1.setText("Connecting...");

                String info = ((TextView) v).getText().toString();
                String address = info.substring(info.length() - 17);


                Intent i = new Intent(DeviceListActivity.this,
MainActivity.class);
                i.putExtra(EXTRA_DEVICE_ADDRESS, address);
                startActivity(i);
            }
    };
```

O endereço é guardado numa variável e enviado para *a main activity* para ser criada a ligação através de *sockets*. O seguinte código apresenta como é criado o *socket* de comunicação com o módulo Bluetooth através do endereço do dispositivo.

```
public void onResume() {
        super.onResume();

        Intent intent = getIntent();

        address =
intent.getStringExtra(DeviceListActivity.EXTRA_DEVICE_ADDRESS);

        BluetoothDevice device = btAdapter.getRemoteDevice(address);
```

```
        try {
            btSocket = createBluetoothSocket(device);
        } catch (IOException e) {
            Toast.makeText(getBaseContext(), "Socket creation failed",
Toast.LENGTH_LONG).show();
        }

        try {
            btSocket.connect();
        } catch (IOException e) {
            try {
                btSocket.close();
            } catch (IOException e2) {

            }
        }
        mConnectedThread = new ConnectedThread(btSocket);
        mConnectedThread.start();
 }
```

A ligação é criada com segurança através da implementação do método UUID. O seguinte código mostra a inicialização do método UUID e a criação da ligação segura.

```
private static final UUID BTMODULEUUID = UUID.fromString("00001101-
0000-1000-8000-00805F9B34FB");

  private BluetoothSocket createBluetoothSocket(BluetoothDevice
device) throws IOException {

        return device.createRfcommSocketToServiceRecord(BTMODULEUUID);

    }
```

O dispositivo Arduino envia os dados recolhidos de forma cíclica, devido a isso é necessário criar uma *thread* responsável apenas por receber esses valores. Esta *thread* é executada em *background* em vez de ser executada na *thread* principal . O seguinte código apresenta a criação dessa *thread*.

```
private class ConnectedThread extends Thread {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;


        public ConnectedThread(BluetoothSocket socket) {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            try {

                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
            }
```

```
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }
```

Os dados recebidos através da comunicação Bluetooth são representados por *bytes*. Para ter acesso a todos os bytes é necessário criar uma variável para guardar a informação que é enviada através de um *buffer*. Como a informação pretendida não são bytes é necessário converter esses *bytes* numa *string*. O seguinte código apresentar essa conversão.

```
public void run() {
            byte[] buffer = new byte[256];
            int bytes;

            while (true) {
                try {
                    bytes = mmInStream.read(buffer);
                    String readMessage = new String(buffer, 0, bytes);
```

Finalmente é necessário enviar os dados recebidos para serem tratados na *thread* principal, para isso é necessário criar um *handler* que tem como função enviar mensagens de uma *thread* a ser exetuada em *background* para a thread principal. O seguinte código demonstra o envio da mensagem para a *thread* principal.

```
bluetoothIn.obtainMessage(handlerState, bytes, -1,
readMessage).sendToTarget();
```

Neste momento a *thread* principal da aplicação pode proceder ao tratamento dos dados.

4.4.2 Sistema de segurança

Este módulo implementa um sistema de segurança da viatura, esse sistema é implementado através de um sensor interno disponível no dispositivo móvel denominado por acelerómetro [10]. O acelerómetro é um sensor capaz de medir a aceleração de um corpo em relação à gravidade num sistema de eixos. O seguinte código apresenta como podemos ter acesso a esse sensor.

```
public void Acelerometro() {

        sm = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensor = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sm.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
```

Depois de termos acesso ao sensor é necessário usar uma função do acelerometro para obter informação sobre os valores em cada um dos eixos x,y e z respetivamente. A obtenção dos valores é feita através de um evento que guarda o valor de cada eixo num *array*. O seguinte código apresenta com podemos guardar numa variável o valor de cada eixo.

```
public void onSensorChanged(SensorEvent event) {

        ValorX = event.values[0];
        ValorY = event.values[1];
        ValorZ = event.values[2];
```

Como foi referenciado anteriormente, a aplicação tem a funcionalidade de calibrar o dispositivo. Para isso é necessário criar uma nova escala sempre que o dispositivo é calibrado. O método implementado consiste em subtrair os valores lidos no momento da calibração em todos os eixos e apresentá-los com o valor 0. Durante a implementação deste método surgiu um problema para definir a escala e foi necessário trabalhar com o eixo x e o eixo z em conjunto. Os valores obtidos pelo sensor acelerómetro estão compreendidos entre 0 e 10, em que 0 equivale a 0º e 10 equivale a 90º. Como já foi referido anteriormente, a viatura encontra-se em perigo iminente quando tiver uma inclinação no eixo vertical superior a 65 º. Através de uma regra de 3 simples esse valor representa aproximadamente o valor 7. Em relação ao eixo horizontal, a viatura encontra-se em perigo iminente quando tiver uma inclinação superior a 45º. Também através de uma regra de 3 simples esse valor representa o valor 5. O sistema de segurança têm a funcionalidade de alertar o utilizador através de um

efeito sonoro e uma mensagem visual de perigo. O seguinte código apresenta como foi implementado o sistema de segurança da viatura.

```
if (event.values[0] - x > 7)
        {
            mPlayer.start();
            textViewPerigo.setVisibility(View.VISIBLE);

        } else if (ValorZ < 0) {
            float soma = 10 - x;
            soma = soma + (10 - ValorX);

            if (soma > 7) {
                mPlayer.start();
                textViewPerigo.setVisibility(View.VISIBLE);
            }
            else{
                mPlayer.pause();
                textViewPerigo.setVisibility(View.INVISIBLE);
            }
        } else {

            if (mPlayer.isPlaying()) {
                mPlayer.pause();
                textViewPerigo.setVisibility(View.INVISIBLE);
            }

        }
        if (event.values[1] - y > 5 || event.values[1] - y  < -5){

            mPlayer.start();
            textViewPerigo.setVisibility(View.VISIBLE);
        }
```

4.4.3 Monitorização em tempo real

Este módulo implementa um sistema de monitorização em tempo real. Este sistema é implementado através de uma API da Google denominada Google MapsV2. Como já foi referido, é necessário criar conta na Google e obter uma chave para usar esta API num determinado projeto. Toda a informação necessária para utilizar esta API pode ser encontrada na documentação da Google [10].

Depois de criar conta na Google e obter a chave para o projeto é necessário alterar o manifesto no projeto Android e acrescentar algumas permissões e a chave obtida. O seguinte código apresenta essas alterações necessárias.

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVIC
ES" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />

<meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />
```

Para implementar as funcionalidades da API Google Maps V2 é necessário criar um procedimento para ligar o serviço de GPS e atualizar essa ligação de forma periódica. As coordenadas geográficas, latitude e longitude, são fornecidas pela operadora a que foi criada a ligação e são atualizadas sempre que a localização do dispositivo é alterada. As coordenadas geográficas são guardadas na trama de dados em conjunto com todos os outros valores. A visualização do mapa na aplicação é realizada através de um fragmento responsável por apresentar o mapa e a posição em tempo real do dispositivo. A Figura 7 representa esse fragmento.

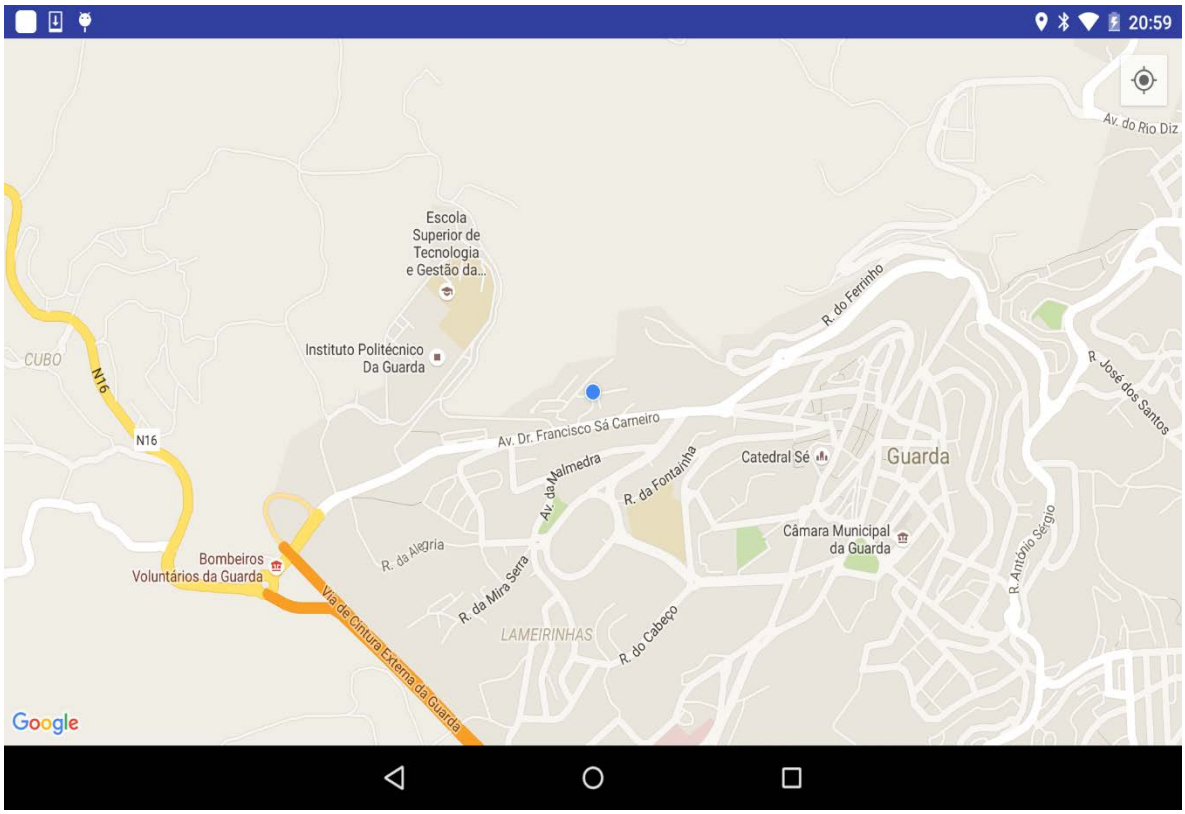

Figura 7 - GoogleMaps

4.4.4 Armazenamento de dados

Este módulo pretende implementar um sistema de armazenamento de dados. Esta funcionalidade permite ao utilizador guardar todos os dados já referidos anteriormente ao premir o botão “Guardar dados” presente na aplicação (Figura 9). Os dados armazenados são os dados que são visualizados no momento que o utilizador prime o botão, e são guardados no cartão *SD* do dispositivo móvel, num ficheiro de texto denominado “FireProtection”. Caso esse ficheiro de texto não exista, será criado um ficheiro com esse nome. Como o sistema pretende guardar inúmeras tramas e não sobrepor tramas sempre que é guardada uma trama nova, é necessário, sempre que seja guardado uma trama, abrir o ficheiro e guardar todas as tramas existentes no ficheiro e guardar o ficheiro com a nova trama. A Figura 8 apresenta os dados guardados no cartão SD do dispositivo móvel.

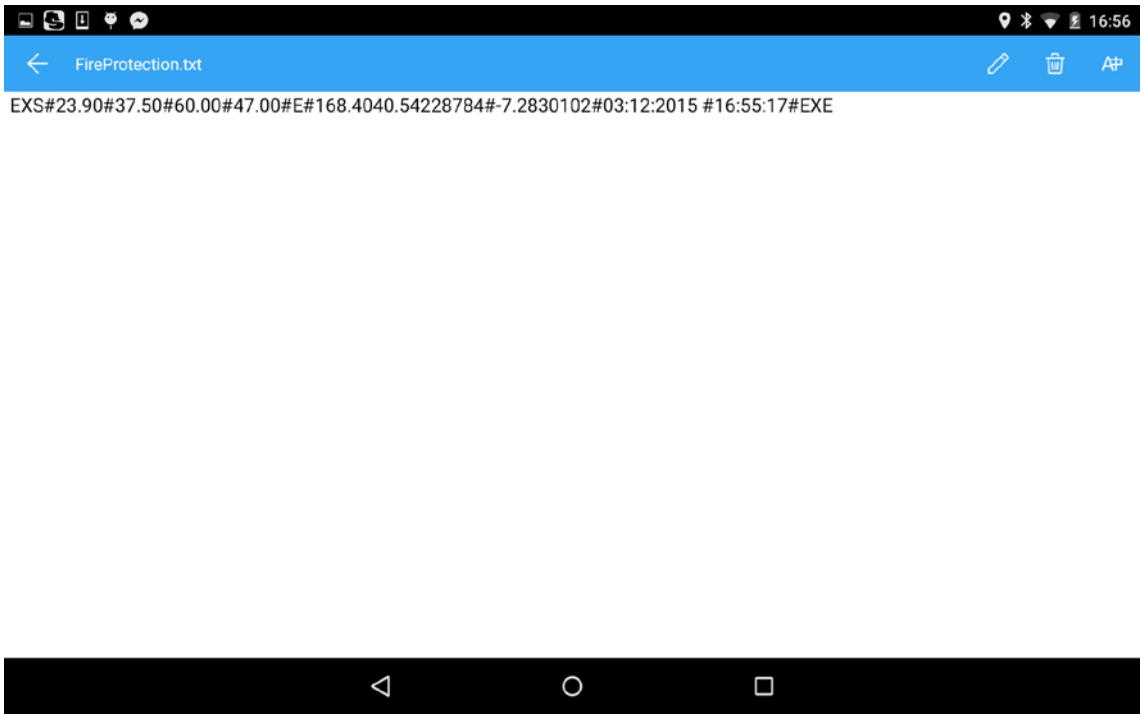

Figura 8 - Trama de dados

4.4.5 Checksum

Este módulo implementa um sistema de validação de dados. A solução encontrada para garantir a integridade dos dados e garantir que os dados enviados pelo dispositivo Arduino são os dados recebidos no dispositivo Android sem que haja perda de informação, foi criar uma variável responsável pela validação dos dados. Essa variável é a soma do valor de todos os sensores e é enviada em conjunto com a trama de dados. A seguinte figura apresenta como é enviada através do Arduino essa variável.

```
float sensorValueTotal = sensorValue[0] + sensorValue[1] +
sensorValue[2] + sensorValue[3];
Serial.println(sensorValueTotal);

Serial3.print(sensorValueTotal);
```

A validação dos dados é efetuada posteriormente no Android, essa validação consiste em somar todos os valores recebidos de cada sensor e comparar se esse valor é igual ao valor enviado pelo Android. Os dados apenas serão apresentados na aplicação se essa condição for respeitada, caso essa condição não seja respeitada é enviada uma mensagem de erro. O seguinte código apresenta essa validação.

```
  String[] aux = TramaDados.toString().split("#");
```

```
  Double total, sensor1, sensor2, sensor3, sensor4;

   sensor1 = Double.parseDouble(aux[1]);
   sensor2 = Double.parseDouble(aux[2]);
   sensor3 = Double.parseDouble(aux[3]);
   sensor4 = Double.parseDouble(aux[4]);

   total = Double.parseDouble(aux[6]);

 if (total == (sensor1 + sensor2 + sensor3 + sensor4)) {

     textViewTE.setText("Temperatura Exterior: " + aux[1] + "º");
     textViewHR.setText("Humidade Relativa: " + aux[2] + "%");
     textViewCO2.setText("Co2:" + aux[3]+ "ppm");
     textViewVV.setText("Velocidade Vento: " + aux[4]+ "m/s");
     textViewDV.setText("Direcção Vento: " + aux[5]);

             } else {
                  textViewTE.setText("Temperatura Exterior: Erro");
                  textViewHR.setText("Humidade Relativa: Erro " );
                  textViewCO2.setText("Co2: Erro" );
                  textViewVV.setText("Velocidade Vento: Erro" );
                   textViewDV.setText("Direcção Vento: Erro" );
                    }
      }
```

4.4.6 Tratamento de dados

Este módulo tem a funcionalidade de tratar os dados depois de serem recebidos via Bluetooth. Os dados recebidos são tratados através de um *handler* criado na comunicação Bluetooth , depois de recebida essa mensagem é necessário anexar todos os elementos dessa mensagem numa variável do tipo *string builder*. O método usado para definir quando começa e termina uma trama de dados é o envio do caracter “#” e “~” respetivamente. O seguinte código demonstra o envio desses caracteres para iniciar e terminar a trama de dados.

```
Serial3.print ("#");
for(int k=0;k<4;k++){

Serial3.print(sensorValue[k]);

Serial3.print('#');

}

Serial3.print(direcaoVento);
Serial3.print('#');

float sensorValueTotal = sensorValue[0] + sensorValue[1] +
sensorValue[2] + sensorValue[3];
Serial.println(sensorValueTotal);

Serial3.print(sensorValueTotal);
Serial3.print("~");
```

Com isto, é necessário agora tratar os dados recebidos. A solução implementada consiste em definir quando começa e termina a trama e retirar esses caracteres, bem como os caracteres que separam os valores de cada sensor e tratá-los de forma individual. Todos os valores são guardados num *array* de dados. O seguinte código demonstra como são tratados os dados recebidos.

```
recDataString.append(readMessage);

 int endOfLineIndex = recDataString.indexOf("~");

  if (endOfLineIndex > 0) {
     String TramaDados = recDataString.substring(0, endOfLineIndex);
     TramaTotal = "EXS" + TramaDados + Coordenadas + strdate + "#EXE";


          if (recDataString.charAt(0) == '#'){


              String[] aux = TramaDados.toString().split("#");
```

De seguida é necessário apresentar os dados de forma individual e nas respetivas textviews. O seguinte código demonstra como os dados são apresentados de forma individual.

```
textViewTE.setText("Temperatura Exterior: " + aux[1]);
textViewHR.setText("Humidade Relativa: " + aux[2]);
textViewCO2.setText("Co2:" + aux[3]);
textViewVV.setText("Velocidade Vento: " + aux[4]);
textViewDV.setText("Direcção Vento: " + aux[5]);

  } else {
      textViewTE.setText("Temperatura Exterior: Erro");
      textViewHR.setText("Humidade Relativa: Erro " );
      textViewCO2.setText("Co2: Erro" );
      textViewVV.setText("Velocidade Vento: Erro" );
      textViewDV.setText("Direcção Vento: Erro" );
}
```

A Figura 9 apresenta o *layout* principal onde é possível visualizar todos os dados de forma personalizada, esses dados são atualizados de forma automática.

Figura 9 - Layout principal

4.4.7 Sensor simulado

Este módulo pretende simular valores hipotéticos de um sensor de vento. O sensor de vento tem a funcionalidade de recolher duas componentes diferentes, sendo elas, a velocidade do vento e a direção do vento. Os valores simulados para a componente velocidade do vento são gerados aleatoriamente e estão compreendidos entre os 0 e 60 m/s, enquanto a componente direção do vento é gerada de forma aleatória e apresenta os pontos cardiais : N,S,E e O. O seguinte código apresenta as variáveis para essa simulação.

```
char direcao[4] = {'N','S','E','O'};
char direcaoVento;
long velocidadeVento =0;

velocidadeVento = random(60);
Serial.println(velocidadeVento);

char direcaoVento = direcao[rand()%4];
Serial.println(direcaoVento);
```

As variáveis criadas para simular o sensor de vento são enviadas em conjunto com o resto dos valores dos sensores.

4.4.8 Sensor MQ-7

Este módulo pretende implementar todas as funcionalidades necessárias para obter os valores capturados pelo sensor MQ-7. Como já foi referido, este sensor deteta os níveis de dióxido de carbono presentes no ambiente e tem a particularidade de funcionar por ciclos. A Figura 10 apresenta o sensor usado.

Figura 10 - Sensor MQ-7

Os valores do sensor são recebidos através de uma entrada analógica e são guardados numa variável para mais tarde serem enviados. O seguinte código mostra a entrada analógica usada e a variável que vai guardar os valores recolhidos por esse sensor.

```
int Co2SensorOutput = 0;
int Co2Dados = 0;
```

Este sensor necessita aquecer para fazer leituras corretas de valores, para isso é necessário implementar um ciclo para funcionar faseadamente. O ciclo implementado tem o objetivo de obrigar o sensor a aquecer durante 60 segundos a uma voltagem de 3.3V e apresentar valores durante 90 segundos a uma voltagem de 1.4V. Os intervalos de tempo são implementados na biblioteca CS_MQ7.h,através da função MQ7.CoPwrCycler bem como o estado *high* ou *low* em que se encontra o sensor através da função CurrentState. Como apenas pretendemos leituras corretas do sensor vamos impor a condição de ler os valores analógicos e guardá-los numa variável apenas quando a voltagem estiver a 1.4V e apresentar o valor 0 enquanto o sensor estiver a aquecer à voltagem de 3.3V. O seguinte código exemplifica essa condição.

```
if(MQ7.CurrentState() == LOW){

    Co2Dados = analogRead(Co2SensorOutput);
    Serial.println(Co2Dados);
  }

  else{

    Serial.println("sensor heating!");
    Co2Dados= 0;
  }
```

Os dados guardados na variável Co2Dados são enviados mais tarde em conjunto com todos os restantes sensores.

4.4.9 Sensor DHT

Este módulo pretende implementar todas as funcionalidades necessárias para obter os valores capturados pelo sensor DHT. Como já foi referido, este sensor deteta os níveis de humidade e a temperatura presente no ambiente. A Figura 11 apresenta o sensor usado.

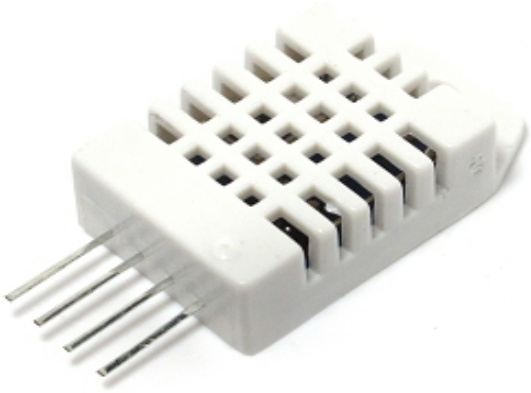

Figura 11 - Sensor DHT22

Os valores do sensor são recebidos através de uma entrada digital. Ao contrário do sensor MQ-7, os valores do sensor são lidos através de pulsos em que esses pulsos podem ser zeros ou uns. A conversão de zeros e uns é implementada pela biblioteca do sensor. Os valores do sensor DHT são recebidos através de uma função implementada na biblioteca e essa função requer dois parâmetros de entrada: a entrada digital a que está ligada no Arduino e qual o tipo do sensor DHT. O seguinte código apresenta a entrada digital escolhida, o tipo de sensor e a função implementada na biblioteca.

```
#define DHTPIN 24
#define DHTYPE DHT22
```

As leituras dos valores deste sensor são feitas em tempos muito rápidos e para isso é necessário desligar os *interrupts* do microcontrolador temporariamente. O sensor envia 40 bits, estes bits são pulsos que vão ser examinados e convertidos em uns e zeros. Como esta leitura é feita em microssegundos os pulsos são guardados num *array* de dados. Por fim, é necessário confirmar que lemos os 40 bits e validá-los através de um *cheksum* implementado pela biblioteca do sensor. Para ter acesso aos dados recolhidos pelo sensor é necessário usar a função implementada pela biblioteca do sensor, e guardar esses dados. O seguinte código apresenta o uso da função dht para ler os dois componentes do sensor e guardá-los em 2 varáveis.

```
float humidade = dht.readHumidity();
float temperatura = dht.readTemperature();

DHT dht(DHTPIN,DHTYPE);
```

Os dados guardados nas variáveis humidade e temperatura são enviados mais tarde em conjunto com todos os restantes sensores.

5.Testes

Neste capítulo vamos apresentar os testes efetuados a alguns módulos do sistema para entender o nível de precisão e eficácia desses módulos.

5.1 Comunicação

O sistema implementa um método de comunicação via Bluetooth. Como todos os métodos de comunicação sem fios, é natural que possa existir perdas de dados durante a comunicação. Essas perdas podem surgir devido a efeitos externos ao sistema ou a problemas relacionados com a implementação do módulo de comunicação. Para testar a comunicação foram efetuados dois testes sendo eles, um teste em ambiente isolado e um ambiente sujeito a interferências externas.

O envio dos dados recolhidos pelo dispositivo Arduino são feitos ciclicamente de dois em dois segundos. Através do elemento de validação implementado no sistema é possível visualizar quando uma trama enviada não é a correta. Através de um teste visual foram analisadas 200 tramas. Como cada trama tem um período de 2s foram realizados 3 testes com a duração de 6 minutos e 60 segundos para cada situação. Os resultados obtidos para as duas situações foram os seguintes:

- Ambiente isolado: média de leituras 2 erradas;
- Ambiente sujeito a interferências externas: média de 3 leituras erradas.

Com estes resultados é possível dizer que o sistema tem falhas na ordem dos 1% em um ambiente isolado e 2 % em um ambiente sujeito a interferências externas para as 200 leituras efetuadas.

É possível determinar que o sistema é estável e como era de esperar o sistema iria ter perdas maiores em ambientes sujeitos a interferências externas.

5.2 Acelerómetro

O sistema implementa um método de segurança ativa através do sensor acelerómetro incorporado no dispositivo móvel. Este sensor determina a posição em que se encontra o dispositivo. Como o dispositivo apresenta valores numa escala de 1 a 10 foi necessário fazer a conversão para uma escala em graus. Os testes efetuados pretendem demonstrar se o sistema é preciso e eficaz, para isso, e através de um esquadro uma ferramenta própria para medir ângulos foram efetuados dois testes, um teste sem calibrar o sistema e um teste depois de calibrar o sistema, este processo foi repetido 10 vezes O teste consistiu em verificar se os ângulos apresentados por o acelerómetro eram os ângulos presentes no esquadro. Os resultados obtidos para as duas situações foram os seguintes:

- Sistema sem estar calibrado: média de 2 graus errados
- Sistema calibrado: média de 2 graus errados

Depois de analisados estes dados é possível dizer que as falhas do sistema são insignificantes e não prejudicam nem representam perigo para as necessidades do cliente.

6.Conclusões e Trabalho futuro

6.1 Conclusões

Ao longo da realização deste projeto, em contexto de estágio, foram utilizadas várias ferramentas e tecnologias novas que não foram introduzidas na licenciatura de Engenharia Informática. Devido a isso foi necessário um estudo e uma pesquisa mais aprofundada sobre essas ferramentas e tecnologias.

Todos os módulos analisados na etapa de especificação do projeto foram implementados. Devido á quantidade de módulos envolvidos no sistema foi dada mais importância ao módulo da comunicação e ao tratamento dos dados. É necessário então, realçar a importância da implementação da comunicação Bluetooth. Sem esta seria impossível apresentar os dados no dispositivo móvel. Como os dados devem respeitar o formato das tramas usadas pela indústria foi necessário implementar um método para armazenar esses dados com esse formato.

Alguns módulos foram implementados apenas com as funcionalidades pretendidas e devido à base do sistema ser sólida é possível implementar novas funcionalidades úteis ou otimizar as funcionalidades já implementadas.

A diversidade de tecnologias utilizadas no desenvolvimento do projeto criou imensas dificuldades. Estas dificuldades consistiram na programação em duas plataformas diferentes, que caminho seguir quando surgiram problemas, ligação dos circuitos, compreender a comunicação Bluetooth entre outras.

Por concluir, é de salientar os conhecimentos adquiridos a nível pessoal e profissional em várias tecnologias. O projeto ter sido desenvolvido em contexto de estágio para a empresa Explorinova foi importante para adquirir experiencia e métodos de trabalho.

6.2 Trabalho futuro

A estrutura base da aplicação encontra-se completa e implementa todas funcionalidades definidas na etapa de especificação. A alguns módulos do sistema é possível acrescentar funcionalidades de modo a enriquecer a aplicação. A segurança da viatura pode ser melhorada através da implementação de um algoritmo mais eficaz para garantir a segurança da viatura em dois eixos em simultâneo. Os dados armazenados no cartão SD do dispositivo podem ser enviados para todas as corporações de bombeiros se fosse criada uma comunicação com o exterior. Adquirir um sensor de vento para obter valores reais dessas medidas. Para finalizar, como esta aplicação pretende ser introduzida no mercado seria interessante efetuar testes no terreno para uma melhor análise da eficácia e precisão da aplicação.

Bibliografia

1. Fernandes, António Jorge de Almeida, (2012), "*Sistema de Monitorização Remota de Temperatura e Humidade de Câmaras de Frio"*. Licenciatura em Engenharia Informática. Instituto Politécnico da Guarda,Guarda.

2. Preto, Ana Cláudia Ferreira**,** (2013), "*Sistema de Aquisição de Dados para Monitorização de Transportes de Carga". Tese de Mestrado em Engenharia Industrial. Escola Superior de Tecnologia e Gestão de Bragança, Bragança.*

3. Araújo, João Carlos Dias Abrunhosa,(2012), "*Sistema de Aquisição e Monitorização de Dados para a Industria Alimentar e Laboratorial". Tese de Mestrado em Engenharia Electrotécnica. Escola Superior de Tecnologia e Gestão de Viseu, Viseu.*

4. Sommersville,(2010). " *Software engineering 9".*

5." HANWEI ELECTRONICS CO, LTD. TECNHICAL DATA MQ-7 GAS SENSOR", *Sparkfun,(2010).Acedido em 25 de Junho de 2015, em:* https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf.

6. "Digital relative humidity & temperature sensor AM2302/DHT22", *Adafruit, (2013). Acedido em 2 de julho de 2015, em :* https://www.adafruit.com/datasheets/Digital%20humidity%20and%20temperature%20sensor%20AM2302.pdf.

7. Arduino Due. *Arduino.Acedido em 28 de junho de Junho de 2015 em:* https://www.arduino.cc/en/Main/ArduinoBoardDue.

8. "Configuração do módulo Bluetooth HC-06", *Buildbot, (2014). Acedido em 14 de julho de 2015 em:* http://buildbot.com.br/blog/configuracao-do-modulo-bluetooth-hc-06-com-arduino/.

9. "WINGOODHARRY. Android send/receive data wth Arduino", *HARRY'S DEVELOPER BLOG, (2013). Acedido em 18 de julho de 2015 em:* https://wingoodharry.wordpress.com/2014/04/07/anrdroid-sendreceive-data-with-arduino-via-bluetooth-part-1/.

10. Motion Sensors. *Android. Acedido em 4 de julho de 2015 em:* http://developer.android.com/guide/topics/sensors/sensors_motion.html.

11. "Google Maps Android API", *Google Developers. Acedido em 29 de Junho de 2015 em:* https://developers.google.com/maps/documentation/android-api/start.

Anexos

Anexo A-
Código da aplicação Arduino

```
#include <CS_MQ7.h>
#include"DHT.h"

#define Serial_ArdAnd Serial3
#define Serial_Debug Serial
#define Serial_Baud 115200

#define DHTPIN 24
#define DHTYPE DHT22

float sensorValue[4] = {0,0,0,0};
long velocidadeVento =0;
float floatVal = 14.5;
String stringVal = "";

char direcao[4] = {'N','S','E','O'};
char direcaoVento;
```

```
CS_MQ7 MQ7(12, 13);

int Co2SensorOutput = 0;
int Co2Dados = 0;
DHT dht(DHTPIN,DHTYPE);

void setup(){

  Serial_ArdAnd.begin(Serial_Baud);
  Serial_Debug.begin(Serial_Baud);

  Serial3.begin(9600);
  dht.begin();

  randomSeed(500);

}

void loop(){


if (Serial_ArdAnd.available()) {

Serial_Debug.write(Serial_ArdAnd.read());
}
Serial_Debug.flush();

if (Serial_Debug.available()) {
```

```
Serial_ArdAnd.write(Serial_Debug.read());
}
Serial_ArdAnd.flush();

delay(2000);

float humidade = dht.readHumidity();
float temperatura = dht.readTemperature();

  float f = dht.readTemperature(true);

  if (isnan(humidade) || isnan(temperatura) ) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  float hif = dht.computeHeatIndex(f, humidade);

  float hic = dht.computeHeatIndex(temperatura, humidade,
false);

  Serial.print("Humidity: ");
  Serial.print(humidade);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(temperatura);
  Serial.print(" *C ");
  //Serial.print(f);
  Serial.print(" *F\t");
  Serial.print("Heat index: ");
```

```
  Serial.print(hic);
  Serial.print(" *C ");
  Serial.print(hif);
  Serial.println(" *F");


  MQ7.CoPwrCycler();

  if(MQ7.CurrentState() == LOW){
    Co2Dados = analogRead(Co2SensorOutput);
    Serial.println(Co2Dados);
  }
  else{
    Serial.println("sensor heating!");
    Co2Dados= 0;
  }

velocidadeVento = random(60);
Serial.println(velocidadeVento);

char direcaoVento = direcao[rand()%4];
Serial.println(direcaoVento);

sensorValue[0] = temperatura;
sensorValue[1] = humidade;
sensorValue[2] = Co2Dados;
sensorValue[3] = velocidadeVento;

Serial3.print ("#");
for(int k=0;k<4;k++){
```

```
Serial3.print(sensorValue[k]);


Serial3.print('#');


}


Serial3.print(direcaoVento);
Serial3.print('#');


float sensorValueTotal = sensorValue[0] + sensorValue[1] +
sensorValue[2] + sensorValue[3];
Serial.println(sensorValueTotal);


Serial3.print(sensorValueTotal);
Serial3.print("~");
Serial3.println();


}
```

Anexo B-
Biblioteca do sensor MQ-7

B.1 MQ7.h

```
/*
     CS_MQ7_02.h - Library for reading the MQ-7 Carbon
Monoxide Sensor
     Breakout, as part of the Citizen Sensor project.
     http://citizensensor.cc

     Released into the public domain.

     Created by J Saavedra, October 2010.
     http://jos.ph

*/

#ifndef CS_MQ7_h
#define CS_MQ7_h

#include "Arduino.h"

class CS_MQ7{

     public:

          CS_MQ7(int CoTogPin, int CoIndicatorPin);
          void CoPwrCycler();
          boolean CurrentState();

          unsigned long time;
```

```
        unsigned long currTime;
        unsigned long prevTime;
        unsigned long currCoPwrTimer;

        boolean CoPwrState;

    private:
        int _CoIndicatorPin;
        int _CoTogPin;

    };

#endif
```

B.2 MQ7.cpp

```
/*
     CS_MQ7_02.h - Library for reading the MQ-7 Carbon
Monoxide Sensor
     Breakout, as part of the Citizen Sensor project.
     http://citizensensor.cc
     Released into the public domain.
     Created by J Saavedra, October 2010.
     http://jos.ph
*/

#include "Arduino.h"
#include "CS_MQ7.h"

CS_MQ7::CS_MQ7(int CoTogPin, int CoIndicatorPin){

     pinMode(CoIndicatorPin, OUTPUT);
     pinMode(CoTogPin, OUTPUT);

     _CoIndicatorPin = CoIndicatorPin;
     _CoTogPin = CoTogPin;

     time = 0;
     currTime = 0;
     prevTime = 0;
     currCoPwrTimer = 0;
     CoPwrState = LOW;
     currCoPwrTimer = 500;
```

```
    }

void CS_MQ7::CoPwrCycler(){

  currTime = millis();

  if (currTime - prevTime > currCoPwrTimer){
    prevTime = currTime;

    if(CoPwrState == LOW){
      CoPwrState = HIGH;
      currCoPwrTimer = 60000;  //60 seconds at 5v
    }
    else{
      CoPwrState = LOW;
      currCoPwrTimer = 90000;  //90 seconds at 1.4v
    }
    digitalWrite(_CoIndicatorPin, CoPwrState);
    digitalWrite(_CoTogPin, CoPwrState);
  }
}

boolean CS_MQ7::CurrentState(){
    if(CoPwrState == LOW){
        return false;
    }
    else{
        return true;
    }
}
```

Anexo C-
Biblioteca do sensor DHT

C.1 DHT.h

```
#define DEBUG_PRINTER Serial


// Setup debug printing macros.
#ifdef DHT_DEBUG
  #define DEBUG_PRINT(...) {
DEBUG_PRINTER.print(__VA_ARGS__); }
  #define DEBUG_PRINTLN(...) {
DEBUG_PRINTER.println(__VA_ARGS__); }
#else
  #define DEBUG_PRINT(...) {}
  #define DEBUG_PRINTLN(...) {}
#endif


// Define types of sensors.
#define DHT11 11
#define DHT22 22
#define DHT21 21
#define AM2301 21



class DHT {
  public:
   DHT(uint8_t pin, uint8_t type, uint8_t count=6);
   void begin(void);
   float readTemperature(bool S=false, bool force=false);
   float convertCtoF(float);
   float convertFtoC(float);
```

```
   float computeHeatIndex(float temperature, float
percentHumidity, bool isFahrenheit=true);
   float readHumidity(bool force=false);
   boolean read(bool force=false);


 private:
  uint8_t data[5];
  uint8_t _pin, _type;
  #ifdef __AVR
    // Use direct GPIO access on an 8-bit AVR so keep track
of the port and bitmask
    // for the digital pin connected to the DHT.  Other
platforms will use digitalRead.
    uint8_t _bit, _port;
  #endif
  uint32_t _lastreadtime, _maxcycles;
  bool _lastresult;


  uint32_t expectPulse(bool level);


};
class InterruptLock {
  public:
   InterruptLock() {
    noInterrupts();
   }
   ~InterruptLock() {
    interrupts();
   }
};


#endif
```

C.2 DHT.cpp

```
/* DHT library

MIT license
written by Adafruit Industries
*/

#include "DHT.h"

#define MIN_INTERVAL 2000

DHT::DHT(uint8_t pin, uint8_t type, uint8_t count) {
  _pin = pin;
  _type = type;
  #ifdef __AVR
    _bit = digitalPinToBitMask(pin);
    _port = digitalPinToPort(pin);
  #endif
  _maxcycles = microsecondsToClockCycles(1000);  // 1
millisecond timeout for
                                                 // reading
pulses from DHT sensor.
  // Note that count is now ignored as the DHT reading
algorithm adjusts itself
  // basd on the speed of the processor.
}

void DHT::begin(void) {
  // set up the pins!
```

```
  pinMode(_pin, INPUT_PULLUP);
  // Using this value makes sure that millis() -
lastreadtime will be
  // >= MIN_INTERVAL right away. Note that this assignment
wraps around,
  // but so will the subtraction.
  _lastreadtime = -MIN_INTERVAL;
  DEBUG_PRINT("Max clock cycles: ");
DEBUG_PRINTLN(_maxcycles, DEC);
}

//boolean S == Scale.  True == Fahrenheit; False == Celcius
float DHT::readTemperature(bool S, bool force) {
  float f = NAN;

  if (read(force)) {
    switch (_type) {
    case DHT11:
      f = data[2];
      if(S) {
        f = convertCtoF(f);
      }
      break;
    case DHT22:
    case DHT21:
      f = data[2] & 0x7F;
      f *= 256;
      f += data[3];
      f *= 0.1;
      if (data[2] & 0x80) {
        f *= -1;
      }
```

```
      if(S) {
        f = convertCtoF(f);
      }
      break;
    }
  }
  return f;
}

float DHT::convertCtoF(float c) {
  return c * 1.8 + 32;
}

float DHT::convertFtoC(float f) {
  return (f - 32) * 0.55555;
}

float DHT::readHumidity(bool force) {
  float f = NAN;
  if (read()) {
    switch (_type) {
    case DHT11:
      f = data[0];
      break;
    case DHT22:
    case DHT21:
      f = data[0];
      f *= 256;
      f += data[1];
      f *= 0.1;
      break;
```

```
    }
  }
  return f;
}


//boolean isFahrenheit: True == Fahrenheit; False ==
Celcius
float DHT::computeHeatIndex(float temperature, float
percentHumidity, bool isFahrenheit) {
  // Using both Rothfusz and Steadman's equations
  //
http://www.wpc.ncep.noaa.gov/html/heatindex_equation.shtml
  float hi;


  if (!isFahrenheit)
    temperature = convertCtoF(temperature);


  hi = 0.5 * (temperature + 61.0 + ((temperature - 68.0) *
1.2) + (percentHumidity * 0.094));


  if (hi > 79) {
    hi = -42.379 +
              2.04901523 * temperature +
             10.14333127 * percentHumidity +
             -0.22475541 * temperature*percentHumidity +
             -0.00683783 * pow(temperature, 2) +
             -0.05481717 * pow(percentHumidity, 2) +
              0.00122874 * pow(temperature, 2) *
percentHumidity +
              0.00085282 * temperature*pow(percentHumidity,
2) +
             -0.00000199 * pow(temperature, 2) *
pow(percentHumidity, 2);
```

```
    if((percentHumidity < 13) && (temperature >= 80.0) &&
(temperature <= 112.0))

      hi -= ((13.0 - percentHumidity) * 0.25) * sqrt((17.0
- abs(temperature - 95.0)) * 0.05882);


    else if((percentHumidity > 85.0) && (temperature >=
80.0) && (temperature <= 87.0))

      hi += ((percentHumidity - 85.0) * 0.1) * ((87.0 -
temperature) * 0.2);

  }


  return isFahrenheit ? hi : convertFtoC(hi);

}


boolean DHT::read(bool force) {

  // Check if sensor was read less than two seconds ago and
return early

  // to use last reading.

  uint32_t currenttime = millis();

  if (!force && ((currenttime - _lastreadtime) < 2000)) {

    return _lastresult; // return last correct measurement

  }

  _lastreadtime = currenttime;


  // Reset 40 bits of received data to zero.

  data[0] = data[1] = data[2] = data[3] = data[4] = 0;


  // Send start signal.  See DHT datasheet for full signal
diagram:

  //
http://www.adafruit.com/datasheets/Digital%20humidity%20and
%20temperature%20sensor%20AM2302.pdf
```

```
  // Go into high impedence state to let pull-up raise data
line level and
  // start the reading process.
  digitalWrite(_pin, HIGH);
  delay(250);

  // First set data line low for 20 milliseconds.
  pinMode(_pin, OUTPUT);
  digitalWrite(_pin, LOW);
  delay(20);

  uint32_t cycles[80];
  {
    // Turn off interrupts temporarily because the next
sections are timing critical
    // and we don't want any interruptions.
    InterruptLock lock;

    // End the start signal by setting data line high for
40 microseconds.
    digitalWrite(_pin, HIGH);
    delayMicroseconds(40);

    // Now start reading the data line to get the value
from the DHT sensor.
    pinMode(_pin, INPUT_PULLUP);
    delayMicroseconds(10);  // Delay a bit to let sensor
pull data line low.

    // First expect a low signal for ~80 microseconds
followed by a high signal
    // for ~80 microseconds again.
    if (expectPulse(LOW) == 0) {
```

```
      DEBUG_PRINTLN(F("Timeout waiting for start signal low
pulse."));
      _lastresult = false;
      return _lastresult;
    }
    if (expectPulse(HIGH) == 0) {
      DEBUG_PRINTLN(F("Timeout waiting for start signal
high pulse."));
      _lastresult = false;
      return _lastresult;
    }


    // Now read the 40 bits sent by the sensor.  Each bit
is sent as a 50
    // microsecond low pulse followed by a variable length
high pulse.  If the
    // high pulse is ~28 microseconds then it's a 0 and if
it's ~70 microseconds
    // then it's a 1.  We measure the cycle count of the
initial 50us low pulse
    // and use that to compare to the cycle count of the
high pulse to determine
    // if the bit is a 0 (high state cycle count < low
state cycle count), or a
    // 1 (high state cycle count > low state cycle count).
Note that for speed all
    // the pulses are read into a array and then examined
in a later step.
    for (int i=0; i<80; i+=2) {
      cycles[i]   = expectPulse(LOW);
      cycles[i+1] = expectPulse(HIGH);
    }
  } // Timing critical code is now complete.
```

```
  // Inspect pulses and determine which ones are 0 (high
state cycle count < low
  // state cycle count), or 1 (high state cycle count > low
state cycle count).
  for (int i=0; i<40; ++i) {
    uint32_t lowCycles  = cycles[2*i];
    uint32_t highCycles = cycles[2*i+1];
    if ((lowCycles == 0) || (highCycles == 0)) {
      DEBUG_PRINTLN(F("Timeout waiting for pulse."));
      _lastresult = false;
      return _lastresult;
    }
    data[i/8] <<= 1;
    // Now compare the low and high cycle times to see if
the bit is a 0 or 1.
    if (highCycles > lowCycles) {
      // High cycles are greater than 50us low cycle count,
must be a 1.
      data[i/8] |= 1;
    }
    // Else high cycles are less than (or equal to, a weird
case) the 50us low
    // cycle count so this must be a zero.  Nothing needs
to be changed in the
    // stored data.
  }


  DEBUG_PRINTLN(F("Received:"));
  DEBUG_PRINT(data[0], HEX); DEBUG_PRINT(F(", "));
  DEBUG_PRINT(data[1], HEX); DEBUG_PRINT(F(", "));
  DEBUG_PRINT(data[2], HEX); DEBUG_PRINT(F(", "));
  DEBUG_PRINT(data[3], HEX); DEBUG_PRINT(F(", "));
  DEBUG_PRINT(data[4], HEX); DEBUG_PRINT(F(" =? "));
```

```
  DEBUG_PRINTLN((data[0] + data[1] + data[2] + data[3]) &
0xFF, HEX);


  // Check we read 40 bits and that the checksum matches.
  if (data[4] == ((data[0] + data[1] + data[2] + data[3]) &
0xFF)) {
    _lastresult = true;
    return _lastresult;
  }
  else {
    DEBUG_PRINTLN(F("Checksum failure!"));
    _lastresult = false;
    return _lastresult;
  }
}


// Expect the signal line to be at the specified level for
a period of time and
// return a count of loop cycles spent at that level (this
cycle count can be
// used to compare the relative time of two pulses).  If
more than a millisecond
// ellapses without the level changing then the call fails
with a 0 response.
// This is adapted from Arduino's pulseInLong function
(which is only available
// in the very latest IDE versions):
//
https://github.com/arduino/Arduino/blob/master/hardware/ard
uino/avr/cores/arduino/wiring_pulse.c
uint32_t DHT::expectPulse(bool level) {
  uint32_t count = 0;
  // On AVR platforms use direct GPIO port access as it's
much faster and better
```

```
  // for catching pulses that are 10's of microseconds in
length:
  #ifdef __AVR
    uint8_t portState = level ? _bit : 0;
    while ((*portInputRegister(_port) & _bit) == portState)
{
      if (count++ >= _maxcycles) {
        return 0; // Exceeded timeout, fail.
      }
    }
  // Otherwise fall back to using digitalRead (this seems
to be necessary on ESP8266
  // right now, perhaps bugs in direct port access
functions?).
  #else
    while (digitalRead(_pin) == level) {
      if (count++ >= _maxcycles) {
        return 0; // Exceeded timeout, fail.
      }
    }
  #endif


  return count;
}
```

Anexo D-
Código do Android

D.1 MainActivity

```
package com.example.temudo.fireprotection;

import android.Manifest;
import android.annotation.TargetApi;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.media.MediaPlayer;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.provider.MediaStore;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.view.WindowManager;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.maps.model.LatLng;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.List;
import java.util.UUID;

public class MainActivity extends AppCompatActivity
implements LocationListener, SensorEventListener {

    Sensor sensor;
    SensorManager sm;

    MediaPlayer mPlayer;
    Handler bluetoothIn;
    TextView textViewHR, textViewTE, textViewCO2,
textViewLat, textViewLong, textViewVV, textViewDV,
textViewDecliveX, textViewDecliveY, textViewDecliveZ,
textViewPerigo;
    Button btGuardarSD;

    final int handlerState = 0;
//used to identify handler message
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private StringBuilder recDataString = new
StringBuilder();
    private StringBuilder Antigos = new StringBuilder();


    private String TramaTotal;
    private String Coordenadas;
    private float ValorX;
    private float ValorY;
    private float ValorZ;
    private float ValorXX;
    private float x, y, z;
    private Double total;


    private ConnectedThread mConnectedThread;

    // SPP UUID service - this should work for most devices
    private static final UUID BTMODULEUUID =
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    // String for MAC address
    private static String address;
```

```
    private static Button button_sbm;
    private static Button button_sair;
    private static Button button_calibrar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_S
CREEN_ON); //forçar a aplicação a manter-se acordada

        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar)
findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        OnClickButtonListener();
        initLocationListener();
        Acelerometro();
        GuardarSD();


        textViewHR = (TextView)
findViewById(R.id.textViewHR);
        textViewTE = (TextView)
findViewById(R.id.textViewTE);
        textViewCO2 = (TextView)
findViewById(R.id.textViewCO2);
        textViewVV = (TextView)
findViewById(R.id.textViewVV);
        textViewDV = (TextView)
findViewById(R.id.textViewDV);
        textViewPerigo = (TextView)
findViewById(R.id.textViewPerigo);


        bluetoothIn = new Handler() {
            public void handleMessage(android.os.Message
msg) {
                if (msg.what == handlerState) {
//if message is what we want
                    String readMessage = (String) msg.obj;

                    Calendar c1 = Calendar.getInstance();
                    SimpleDateFormat sdf = new
SimpleDateFormat("dd:MM:yyyy #HH:mm:ss");
                    String strdate =
sdf.format(c1.getTime());
// msg.arg1 = bytes from connect thread

                    recDataString.append(readMessage);
```

```
                    int endOfLineIndex =
recDataString.indexOf("~");
//keep appending to string until ~

                    if (endOfLineIndex > 0) {
// make sure there data before ~
                        String TramaDados =
recDataString.substring(0, endOfLineIndex);             //
extract string


                        TramaTotal = "EXS" + TramaDados +
Coordenadas + strdate + "#EXE";



                        if (recDataString.charAt(0) == '#')
//if it starts with # we know it is what we are looking for
                        {


                            String[] aux =
TramaDados.toString().split("#");
                           Double sensor1, sensor2,
sensor3, sensor4;

                            sensor1 =
Double.parseDouble(aux[1]);
                            sensor2 =
Double.parseDouble(aux[2]);
                            sensor3 =
Double.parseDouble(aux[3]);
                            sensor4 =
Double.parseDouble(aux[4]);

                            total =
Double.parseDouble(aux[6]);

                            if (total == (sensor1 + sensor2
+ sensor3 + sensor4)) {



textViewTE.setText("Temperatura Exterior: " + aux[1] +
"º");

textViewHR.setText("Humidade Relativa: " + aux[2] + "%");
                                textViewCO2.setText("Co2:"
+ aux[3]+ "ppm");
```

```
textViewVV.setText("Velocidade Vento: " + aux[4]+ "m/s");

textViewDV.setText("Direcção Vento: " + aux[5]);

                            } else {

textViewTE.setText("Temperatura Exterior: Erro");

textViewHR.setText("Humidade Relativa: Erro " );
                                textViewCO2.setText("Co2:
Erro" );

textViewVV.setText("Velocidade Vento: Erro" );

textViewDV.setText("Direcção Vento: Erro" );
                                }
                            }
                            recDataString.delete(0,
recDataString.length());                    //clear all
string data

                            // strIncom =" ";
                            TramaDados = " ";
                        }
                    }
                }
            };

            btAdapter = BluetoothAdapter.getDefaultAdapter();
// get Bluetooth adapter
            checkBTState();


    }


    private BluetoothSocket
createBluetoothSocket(BluetoothDevice device) throws
IOException {

            return
device.createRfcommSocketToServiceRecord(BTMODULEUUID);
            //creates secure outgoing connecetion with BT
device using UUID
    }

    @Override
    public void onResume() {
            super.onResume();
```

```
        //Get MAC address from DeviceListActivity via
intent
        Intent intent = getIntent();

        //Get the MAC address from the DeviceListActivty
via EXTRA
        address =
intent.getStringExtra(DeviceListActivity.EXTRA_DEVICE_ADDRE
SS);

        //create device and set the MAC address
        BluetoothDevice device =
btAdapter.getRemoteDevice(address);

        try {
            btSocket = createBluetoothSocket(device);
        } catch (IOException e) {
            Toast.makeText(getBaseContext(), "Socket
creation failed", Toast.LENGTH_LONG).show();
        }
        // Establish the Bluetooth socket connection.
        try {
            btSocket.connect();
        } catch (IOException e) {
            try {
                btSocket.close();
            } catch (IOException e2) {
                //insert code to deal with this
            }
        }
        mConnectedThread = new ConnectedThread(btSocket);
        mConnectedThread.start();

        //I send a character when resuming.beginning
transmission to check device is connected
        //If it is not an exception will be thrown in the
write method and finish() will be called
        mConnectedThread.write("x");
    }

    @Override
    public void onPause () {
        super.onPause();
        try {
            //Don't leave Bluetooth sockets open when
leaving activity
            btSocket.close();
        } catch (IOException e2) {
            //insert code to deal with this
        }
    }
```

```
    private void checkBTState() {

        if (btAdapter == null) {
            Toast.makeText(getBaseContext(), "Device does
not support bluetooth", Toast.LENGTH_LONG).show();
        } else {
            if (btAdapter.isEnabled()) {
            } else {
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);
            }
        }
    }

    private class ConnectedThread extends Thread {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        //creation of the connect thread
        public ConnectedThread(BluetoothSocket socket) {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            try {
                //Create I/O streams for connection
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            byte[] buffer = new byte[256];
            int bytes;

            // Keep looping to listen for received messages
            while (true) {
                try {
                    bytes = mmInStream.read(buffer);
//read bytes from input buffer
                    String readMessage = new String(buffer,
0, bytes);

                    Log.v("mensagem:", readMessage + "");
```

```
                    // Send the obtained bytes to the UI
Activity via handler
                    bluetoothIn.obtainMessage(handlerState,
bytes, -1, readMessage).sendToTarget();

                } catch (IOException e) {
                    break;
                }
            }
        }

        //write method
        public void write(String input) {
            byte[] msgBuffer = input.getBytes();
//converts entered String into bytes
            try {
                mmOutStream.write(msgBuffer);
//write bytes over BT connection via outstream
            } catch (IOException e) {
                //if you cannot write, close the
application
                Toast.makeText(getBaseContext(),
"Connection Failure", Toast.LENGTH_LONG).show();
                finish();

            }
        }
    }


    public void GuardarSD() {

        btGuardarSD = (Button)
findViewById(R.id.btGuardar);
        btGuardarSD.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {

                try {
                    File myFile = new
File("/sdcard/FireProtection.txt");
                    FileInputStream fIn = new
FileInputStream(myFile);
                    BufferedReader myReader = new
BufferedReader(
                            new InputStreamReader(fIn));
                    String aDataRow = "";
                    String aBuffer = "";

                    while ((aDataRow = myReader.readLine())
```

```
!= null) {
                        Antigos.append(aDataRow + "\n");

                    }
                    Antigos.append(aBuffer);
                    myReader.close();

                } catch (Exception e) {
                    Toast.makeText(getBaseContext(),
e.getMessage(),
                            Toast.LENGTH_SHORT).show();
                }

                try {
                    File myFile = new
File("/sdcard/FireProtection.txt");
                    myFile.createNewFile();
                    FileOutputStream fOut = new
FileOutputStream(myFile);
                    OutputStreamWriter myOutWriter =
                            new OutputStreamWriter(fOut);
                    myOutWriter.write(Antigos.toString());
                    myOutWriter.write(TramaTotal);
                    myOutWriter.close();
                    fOut.close();
                    Toast.makeText(getBaseContext(),
                            "Informacao Gravada",
                            Toast.LENGTH_SHORT).show();
                } catch (Exception e) {
                    Toast.makeText(getBaseContext(),
e.getMessage(),
                            Toast.LENGTH_SHORT).show();
                }


            }//
        });


    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action
bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
```

```
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action
bar will
        // automatically handle clicks on the Home/Up
button, so long
        // as you specify a parent activity in
AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @TargetApi(Build.VERSION_CODES.M)
    private void initLocationListener() {


        // Getting LocationManager object from System
Service LOCATION_SERVICE
        LocationManager locationManager = (LocationManager)
getSystemService(LOCATION_SERVICE);

        // Creating a criteria object to retrieve provider
        Criteria criteria = new Criteria();

        // Getting the name of the best provider
        String provider =
locationManager.getBestProvider(criteria, true);

        Location location =
locationManager.getLastKnownLocation(provider);

        if (location != null) {
            onLocationChanged(location);
        }

        locationManager.requestLocationUpdates(provider,
2000, 0, this);



}
    @Override
    public void onLocationChanged(Location location) {

        textViewLat = (TextView)
findViewById(R.id.textViewLat);
```

```
        textViewLong = (TextView)
findViewById(R.id.textViewLong);

        double latitude = location.getLatitude();
        String CLat = Double.toString(latitude);

        double longitude = location.getLongitude();
        String CLong = Double.toString(longitude);

        Coordenadas = CLat +"#"+ CLong+"#";

        textViewLat.setText("Latitude: " + latitude);
        textViewLong.setText("Longitude: " + longitude);

    }

    @Override
    public void onStatusChanged(String provider, int
status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {

    }

    @Override
    public void onProviderDisabled(String provider) {

    }

    public void OnClickButtonListener() {

        button_sbm = (Button) findViewById(R.id.btGps);
        button_sair = (Button) findViewById(R.id.btSair);
        button_calibrar = (Button)
findViewById(R.id.btCalibrar);


        button_sair.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
                System.exit(0);
            }
        });


        button_calibrar.setOnClickListener(new
```

```
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                x= ValorX;
                y=ValorY;
                z=ValorZ;

            }
        });


        button_sbm.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        Intent intent = new
Intent("com.example.temudo.fireprotection.MapsActivity");
                        startActivity(intent);
                    }
                }

        );
    }

    public void Acelerometro() {

        sm = (SensorManager)
getSystemService(SENSOR_SERVICE);
        sensor =
sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sm.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
        textViewDecliveX = (TextView)
findViewById(R.id.textViewDecliveX);
        textViewDecliveY = (TextView)
findViewById(R.id.textViewDecliveY);
        textViewDecliveZ = (TextView)
findViewById(R.id.textViewDecliveZ);
        mPlayer = MediaPlayer.create(this, R.raw.sword);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {

        ValorX = event.values[0];
        ValorY = event.values[1];
        ValorZ = event.values[2];


        textViewDecliveX.setText("X: " + (event.values[0] -
x));
```

```
        textViewDecliveY.setText("Y: " + (event.values[1] -
y));
        textViewDecliveZ.setText("Z: " + (event.values[2] -
z));



        if (event.values[0] - x > 7)
        {
            mPlayer.start();
            textViewPerigo.setVisibility(View.VISIBLE);

        } else if (ValorZ < 0) {
            float soma = 10 - x;
            soma = soma + (10 - ValorX);

            if (soma > 7) {
                mPlayer.start();
                textViewPerigo.setVisibility(View.VISIBLE);
            }
            else{
                mPlayer.pause();

textViewPerigo.setVisibility(View.INVISIBLE);
            }
        } else {

            if (mPlayer.isPlaying()) {
                mPlayer.pause();

textViewPerigo.setVisibility(View.INVISIBLE);
            }

        }
        if (event.values[1] - y > 5 || event.values[1] - y
< -5){

            mPlayer.start();
            textViewPerigo.setVisibility(View.VISIBLE);
        }


    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int
accuracy) {

    }
}
```

D.2 MapsActivity

```
package com.example.temudo.fireprotection;

import android.Manifest;
import android.app.Dialog;
import android.content.pm.PackageManager;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.common.ConnectionResult;
importcom.google.android.gms.common.GooglePlayServicesUtil;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity
implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified
when the map is ready to be used.
        SupportMapFragment mapFragment =
(SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

    }
  @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
         mMap.setMyLocationEnabled(true);


    }
```

D.3 DeviceListActivity

```
package com.example.temudo.fireprotection;

import java.util.Set;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;


public class DeviceListActivity extends Activity {

    private static final String TAG = "DeviceListActivity";
    private static final boolean D = true;
    private static Button button_sair;



    Button tlbutton;
    TextView textView1;




    public static String EXTRA_DEVICE_ADDRESS =
"device_address";


    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String>
mPairedDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.device_list);
        OnClickButtonListener();
```

```
    }

    public void OnClickButtonListener(){

        button_sair = (Button)
findViewById(R.id.SairInicial);

        button_sair.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
                System.exit(0);
            }
        });







    }


    @Override
    public void onResume()
    {
        super.onResume();

        checkBTState();

        textView1 = (TextView)
findViewById(R.id.connecting);
        textView1.setTextSize(40);
        textView1.setText(" ");


        mPairedDevicesArrayAdapter = new
ArrayAdapter<String>(this, R.layout.device_name);


        ListView pairedListView = (ListView)
findViewById(R.id.paired_devices);

pairedListView.setAdapter(mPairedDevicesArrayAdapter);

pairedListView.setOnItemClickListener(mDeviceClickListener)
;


        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
        Set<BluetoothDevice> pairedDevices =
mBtAdapter.getBondedDevices();


        if (pairedDevices.size() > 0) {

findViewById(R.id.title_paired_devices).setVisibility(View.
VISIBLE);
            for (BluetoothDevice device : pairedDevices) {

mPairedDevicesArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
            }
        } else {
        }
    }


    private OnItemClickListener mDeviceClickListener = new
OnItemClickListener() {
            public void onItemClick(AdapterView<?> av, View
v, int arg2, long arg3) {

                textView1.setText("Connecting...");

                String info = ((TextView)
v).getText().toString();
                String address =
info.substring(info.length() - 17);


                Intent i = new
Intent(DeviceListActivity.this, MainActivity.class);
                i.putExtra(EXTRA_DEVICE_ADDRESS, address);
                startActivity(i);
            }
    };

    private void checkBTState() {

        mBtAdapter=BluetoothAdapter.getDefaultAdapter();
        if(mBtAdapter==null) {
            Toast.makeText(getBaseContext(), "Device does
not support Bluetooth", Toast.LENGTH_SHORT).show();
        } else {
            if (mBtAdapter.isEnabled()) {
                Log.d(TAG, "...Bluetooth ligado...");
            } else {
```

```
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);

            }
        }
    }
}
```

D.4 AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.temudo.fireprotection" >


    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.R
EAD_GSERVICES" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.BLUETOOTH"/>
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="landscape"
            >


            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.DEFAULT" />
```

```
            </intent-filter>
        </activity>

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action
android:name="com.example.temudo.fireprotection.MapsActivit
y" />

                <category
android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

        <activity

android:name="com.example.temudo.fireprotection.DeviceListA
ctivity"
            android:label="@string/app_name"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```