



**Miguel Filipe Batista Prego**

Licenciado em Engenharia Eletrotécnica e de Computadores

## **Deteção e registo automático de irregularidades no asfalto**

Relatório intermédio para obtenção do Grau de Mestre em  
**Engenharia Eletrotécnica e de Computadores**

Orientador: José Manuel Matos Ribeiro da Fonseca, Professor Associado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

**Setembro, 2016**



# ÍNDICE

<b>Lista de Figuras</b>	<b>v</b>
<b>Lista de Tabelas</b>	<b>vii</b>
<b>1 Trabalho desenvolvido</b>	<b>1</b>
1.1 Funcionamendo do Arduino . . . . .	1
1.2 Funções adicionais . . . . .	3
1.2.1 SD_init . . . . .	3
1.2.2 WriteSD . . . . .	3
1.2.3 ReadSD . . . . .	3
1.2.4 AccelInit . . . . .	3
1.2.5 Accel . . . . .	4
1.2.6 displayGPS . . . . .	4
1.2.7 FTOA - Float to ASCII . . . . .	4
1.3 Funcionamento do Android . . . . .	5
1.4 Base de dados . . . . .	8
1.5 WebSite . . . . .	8
1.5.1 home.php . . . . .	8
1.5.2 store.php . . . . .	8
1.5.3 listLocations.php . . . . .	9
1.5.4 showLocation.php . . . . .	9
<b>Bibliografia</b>	<b>11</b>



## LISTA DE FIGURAS

1.1	Fluxograma referente ao código Arduino . . . . .	2
1.2	Formatos de apresentação de coordenadas . . . . .	5
1.3	Fluxograma referente ao código Android . . . . .	7
1.4	Página <i>web</i> home.php . . . . .	9
1.5	informação tipo adicionada na base de dados . . . . .	9
1.6	Página <i>web</i> listlocations.php . . . . .	10



## LISTA DE TABELAS





## TRABALHO DESENVOLVIDO

A solução proposta para a resolução do problema apresentado em XXX é um sistema separado em três fases, cada uma delas ligada a um elemento físico: o Arduino, o telemóvel e a base de dados (alocada num computador). Assim surgem as seguintes secções, referentes às três fases do sistema.

### 1.1 Funcionamendo do Arduino

Tal como descrito na secção XXX do capítulo XXX, o código Arduino está separado em três grupos chave específicos, sendo descrita a sua utilização específica na presente secção:

- Na zona de declarações são incluídas as bibliotecas referentes ao acelerómetro, ao cartão de memória e ao GPS. São também definidos os modos de escrita no cartão de memória, os portos de ligação de comunicação do GPS e criadas variáveis globais referentes ao GPS e ao acelerómetro. Além disso, são criadas variáveis globais de controlo do código, variáveis de passagem de parâmetros entre vários ciclos da função *loop* e declaração de constantes que serão utilizadas diversas vezes ao longo do código, facilitando a alteração do seu valor em todo o código, caso tal se mostre necessário.
- Na função *setup* são declaradas as bandas de comunicação entre o Arduino e o telemóvel e entre o GPS e o Arduino. São também inicializados dois *leds* como elementos de saída de informação e um botão como entrada de informação e é atribuído o estado de “desligado” a ambos os *leds*. Por fim, são executadas as funções de inicialização do leitor do cartão de memória e do acelerómetro, *SD\_init* e *AccelInit* respectivamente, ambas explicadas mais á frente.

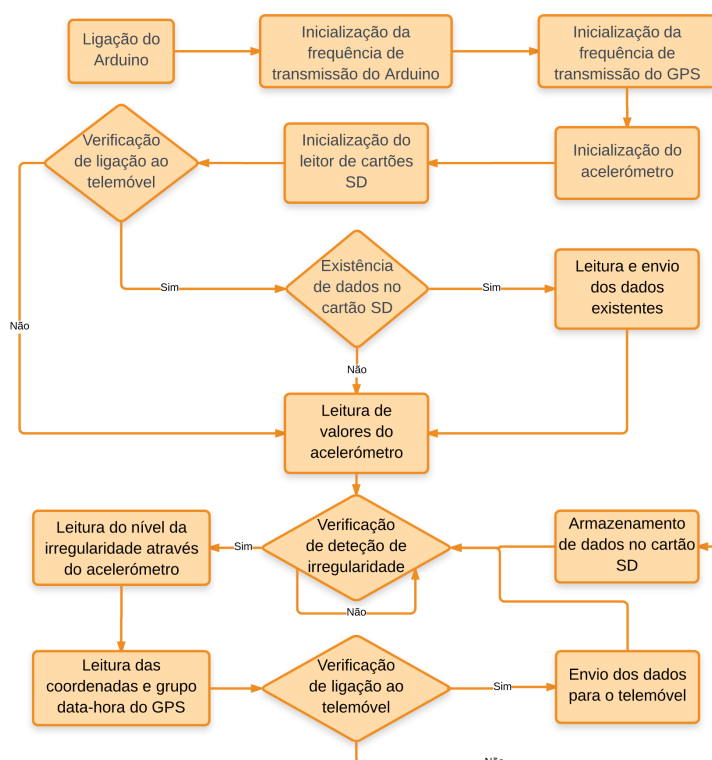


Figura 1.1: Fluxograma referente ao código Arduino

- A função *loop*, embora executada de uma forma constante, contém um código bastante simples mas importante. A função começa por verificar se existe informação disponível para leitura no módulo Bluetooth. Caso tal se verifique, essa informação é lida e comparada, para que seja possível determinar se a ligação ao módulo Bluetooth do Arduino sofreu alguma alteração, nomeadamente, quanto à ligação com a aplicação desenvolvida para esta dissertação. Na hipótese desta alteração se verificar, é avaliada se a ligação foi estabelecida ou cortada, sendo que, no caso de ser estabelecida, é executada a função *ReadSD* para leitura do cartão de memória e um *led* de controlo é aceso. Caso contrário, o mesmo *led* de controlo é desligado. De seguida são lidos os valores de aceleração dos eixos X, Y e Z, provenientes do acelerómetro, armazenados e comparados com a leitura anterior para verificação da existência de alguma irregularidade no asfalto. A maneira como esta comparação será explicada mais à frente. Caso seja detectada alguma irregularidade, um dos *leds* de controlo pisca e são armazenados, numa variável *S*, os valores do nível da irregularidade, bem como o grupo data-hora proveniente do GPS. Seguidamente é verificada a ligação à aplicação e caso esta exista, os valores de *S* são enviados directamente para o telemóvel. Se a ligação não existir, os valores de *S* são transferidos para o cartão de memória para que possam ser transmitidos para o telemóvel assim que exista uma ligação estável com o mesmo. Por fim, os valores actuais do acelerómetro são armazenados para que possam ser comparados na próxima execução da função *loop*.

## 1.2 Funções adicionais

### 1.2.1 SD\_init

Esta função serve para inicializar o leitor de cartões SD. Nela é chamada a função *begin* da biblioteca SD e verificada a resposta dessa mesma função. Caso a resposta seja o valor 10, um *led* de controlo pisca para que o utilizador saiba que a inicialização ocorreu com sucesso e é verificada a existência de um ficheiro com informação previamente armazenada sobre irregularidades detetadas anteriormente. Caso exista esse ficheiro, este é aberto de modo a que se possam juntar novos valores de irregularidades detetadas, caso contrário, o ficheiro é criado para futuros armazenamentos.

### 1.2.2 WriteSD

Para que o cartão de memória possa armazenar dados é apenas necessário abrir um ficheiro em modo escrita. A função *open* da biblioteca do SD está desenhada de forma a que, quando é pedido para abrir um ficheiro, este seja procurado no cartão de memória e caso não exista nenhum ficheiro com esse nome, então é criado um novo ficheiro vazio, com o nome inserido e posteriormente aberto. Para que possa ser mais fácil ao utilizador saber se a abertura e escrita do ficheiro foi feita com sucesso, um *led* pisca duas vezes. No final resta apenas fechar o ficheiro para que tudo fique guardado devidamente.

### 1.2.3 ReadSD

Semelhante à função de escrita, a função de leitura tenta abrir um ficheiro com o nome pretendido embora, neste caso, se o ficheiro desejado não existir, não é criado um novo pois se este não existe significa que não existem dados pendentes para envio. Quando um ficheiro é lido com sucesso, os seus valores são enviados pela porta série do Arduino para o telemóvel, o ficheiro é fechado e eliminado, de modo a que não sejam enviados dados duplicados. O código está protegido de modo a que os dados sejam apenas enviados para um telemóvel com a aplicação desenvolvida pois previamente foi recebida uma mensagem de controlo enviada pela própria aplicação, informando o estado da ligação, evitando que os dados sejam enviados para ligações desconhecidas.

### 1.2.4 AccelInit

Nesta função o acelerómetro é ligado e são determinados os valores de sensibilidade do mesmo. É também feita uma inicialização de valores para queda livre e batida que não são utilizados para esta dissertação. Esta funcionalidade foi mantida para possíveis aplicações futuras e uma deteção mais pormenorizada das irregularidades. Seguidamente, um *led* de controlo pisca para o utilizador saber que a inicialização foi bem sucedida.

### 1.2.5 Accel

Na função *Accel* é feito o processamento dos valores de aceleração dos três eixos do acelerómetro para determinar a classificação da irregularidade detetada. Embora o processo seja simples, é eficaz e semelhante ao método **STDEV(Z)** apresentado em XXX, mas aplicado aos três eixos de aceleração. O método baseia-se em detetar a existência de desvios superiores a um limite previamente determinado, tendo sido considerados valores múltiplos de 100 nesta dissertação. Assim, é feita uma subtração de valores sucessivos de aceleração dos eixos e avaliado o valor absoluto dessa diferença. O valor atribuído à irregularidade é o mais baixo dos três desvios.

### 1.2.6 displayGPS

A função *displayGPS* é sem dúvida a mais elaborada nesta secção do trabalho devido aos dados enviados pelo GPS, muito extensos e, neste caso, não necessários, além de se encontrarem num formato não muito fácil de separar. A função é chamada dentro da função *loop* de modo a registar tanto o conjunto data-hora como as coordenadas em que a irregularidade foi detetada. Quando é chamada, faz uma leitura dos valores que o sensor GPS recebe dos satélites e percorre-os até encontrar o texto "GPRMC", a partir do qual vem toda a informação necessária e armazena todos esses valores num vetor de strings (sequências de caracteres). O passo seguinte consiste em percorrer este e retirar apenas os segmentos relevantes, pela ordem desejada, neste caso, começando pela latitude e longitude, seguindo-se a data e por fim a hora. Quando estes quatro parâmetros são armazenados, a função muda o valor de uma variável de controlo e o código prossegue.

### 1.2.7 FTOA - Float to ASCII

Dada a forma em que os valores das coordenadas do GPS são recebidos, é necessária fazer uma conversão para um formato mais simples de transmitir e como o processo de conversão é executado diversas vezes, foi criada uma função que evita repetição do código. Os dados enviados pelo GPS, relativos às coordenadas vêm no formato DDS – Degree Decimal Minutes (Graus e Minutos Decimais em Português) e apresentam o formato (1) da figura 1.2. Tal como se pode verificar, existem seis parâmetros neste formato, nomeadamente graus, minutos e hemisfério, três para a latitude e outros três para a longitude. Um formato mais simples é o DD – Decimal Degree (Graus Decimais em Português), também apresentado na figura 1.2, no ponto (2). Neste formato existem apenas quatro parâmetros (graus e hemisfério) que podem ser reduzidos a dois, se forem considerados graus negativos, representando o hemisfério a que a latitude ou longitude se referem, sendo assim necessário enviar menos dados para representar a mesma quantidade de informação, tornando o processo mais rápido. A conversão de DDS para DD é simples, sendo apenas necessário dividir a parte dos minutos decimais por 60 (convertendo minutos para graus) e somar esse valor aos graus já existentes. Como os valores que são recebidos do GPS

vêm em strings, é necessário convertê-los para formato numérico, através da função `atoi` (ASCII to Integer, ASCII para inteiro em português) que já existe nas bibliotecas do Arduino e posteriormente voltar a converter para formato de string. Apesar de ser possível escrever um valor numérico numa string, quando esta operação é feita, apenas duas casas decimais são utilizadas e para que as coordenadas possam apresentar um elevado grau de precisão são necessárias quatro casas decimais, sendo assim necessária a criação desta nova função de modo a reter tantas casas decimais quanto as desejadas.

**DDS - DD° MM.MMMM (1)**

**DD - DD.DDD° (2)**

Figura 1.2: Formatos de apresentação de coordenadas

### 1.3 Funcionamento do Android

Tal como explicado na secção XXX do capítulo XXX, o método escolhido para a programação Android foi a MIT App Inventor 2, no website <http://ai2.appinventor.mit.edu> devido à facilidade apresentada, graças a ser uma programação por blocos bem como ao facto de correr num web browser e fazendo o processamento e compilação de código no servidor em vez de ser no cliente, diminuindo bastante o processamento feito no computador local, tornando o processo de criação mais rápido.

A base do código da aplicação é um ciclo `while` que corre desde que a aplicação esteja aberta, semelhante à função `loop` do Arduino. Dentro deste ciclo existem duas condições: uma que verifica se o GPS do telemóvel está a detetar coordenadas e outra que verifica a ligação ao módulo *Bluetooth* do Arduino, bem como um procedimento que verifica a existência de dados armazenados no telemóvel para futuro envio para a base de dados. O GPS do telemóvel é utilizado para que seja possível reportar irregularidades existentes no asfalto sem que se esteja a conduzir, possibilitando todos os transeuntes adicionar informações sobre o local em que se encontram. Sempre que são detetadas coordenadas disponíveis, um botão com o texto "*Shock*" fica ativo e sempre que é pressionado são armazenadas as coordenadas atuais, bem como a data e hora presentes para um posterior envio para a base de dados. Para que estas ocorrências possam ser diferenciadas das detetadas pelo acelerómetro montado no Arduino, o código que lhe é atribuído tem um valor único, facilitando assim a diferenciação entre os dois tipos de irregularidades. No que toca à ligação a um módulo *Bluetooth*, sempre que esta existe, é enviado um código para o dispositivo a que foi feita a ligação. Este código serve para o Arduino saber que o dispositivo que se encontra ligado tem a aplicação a correr, evitando assim que os dados recolhidos por este sejam enviados para um dispositivo desconhecido. Depois de ser feita a ligação, o código fica a "escutar" o Arduino até que ele envie alguma informação para

o telemóvel e quando tal acontece, os vários conjuntos de dados recebidos são divididos para uma lista temporária. Após a divisão ser feita, a lista é percorrida e os dados são enviados para o cartão de memória do telemóvel ou para a base de dados, dependendo da existência de ligação Wi-Fi. Esta ligação é verificada a partir de um erro produzido automaticamente pelo sistema Android, sendo emitida uma mensagem de erro com um código específico. O envio de dados para a base de dados é feito através do método *POST*, sendo desta forma possível ocultar os dados enviados, sendo esta uma vantagem no caso de no futuro serem desenvolvidas contas de utilizador em que certas informações devem permanecer confidenciais. Se a ligação à *internet* não existir, os dados são armazenados no telemóvel para um futuro envio, sendo necessário pressionar um botão existente na aplicação. Este botão serve também para informar o utilizador que tem dados armazenados no dispositivo, uma vez que o botão nem sempre se encontra ativo. No que toca à seleção de dispositivos *Bluetooth*, é mostrada uma lista dos dispositivos com que o telemóvel está emparelhado. No caso de ser a primeira vez que um determinado telemóvel se liga ao Arduino, é necessário fazer o emparelhamento fora da aplicação, como para qualquer outro dispositivo. Quando a aplicação é terminada, é enviada uma mensagem para o Arduino de modo a que este saiba que não é possível enviar mais informação sobre irregularidades e armazene todas as deteções nesse cartão de memória.

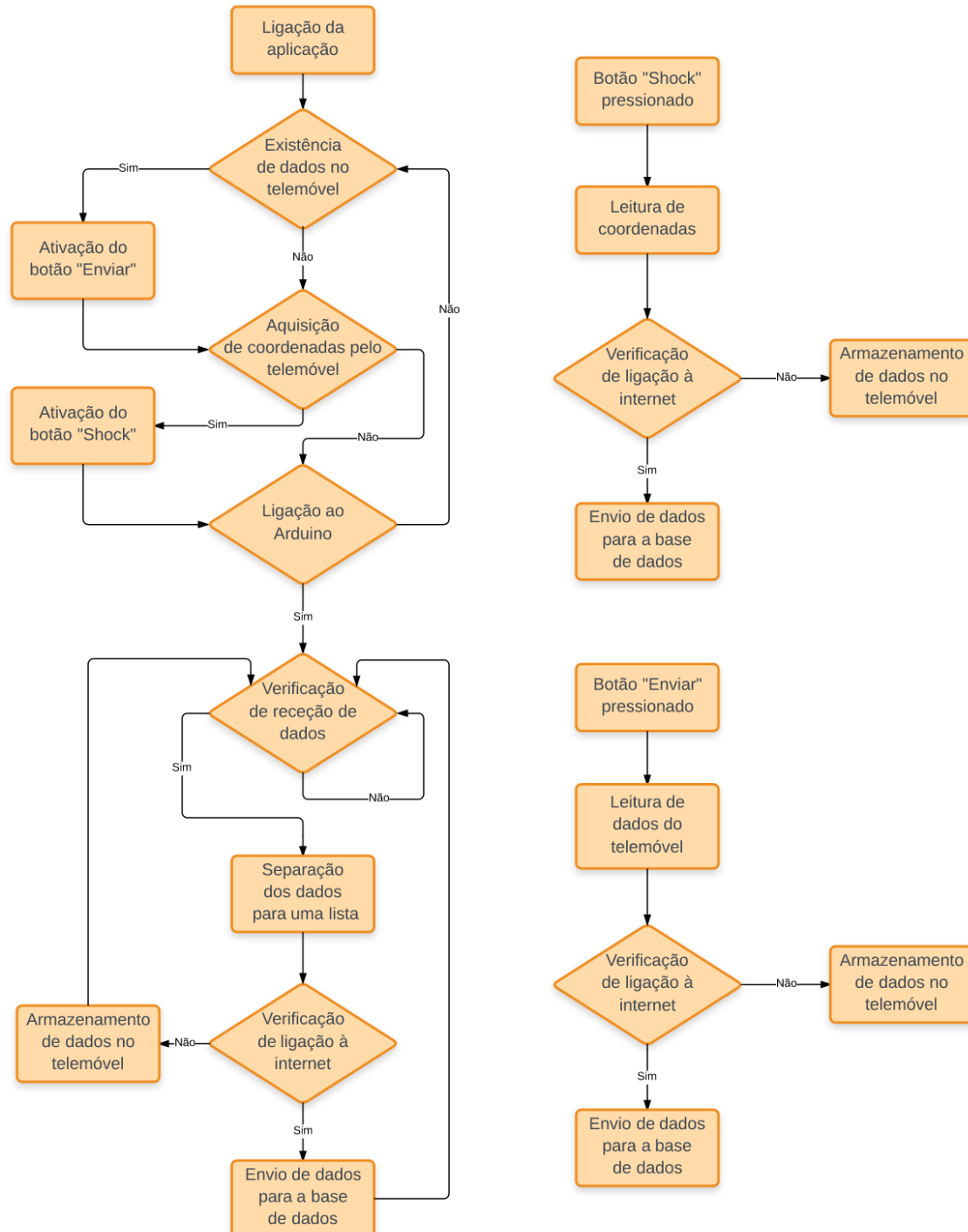


Figura 1.3: Fluxograma referente ao código Android

## 1.4 Base de dados

De modo a que seja possível consultar as irregularidades, foi necessário desenvolver uma base de dados onde ficassem armazenadas as coordenadas, data, hora e intensidade da irregularidade detectadas. Esta base de dados foi desenvolvida em MySQL e contém cinco campos, sendo eles o ID (campo de identificação de cada entrada), a intensidade da irregularidade, a latitude e a longitude da irregularidade e também o conjunto data-hora, utilizando apenas uma variável. Esta base de dados pode ainda ser consultada e alterada por um administrador da mesma, caso este ache necessário de modo a inserir novas funcionalidades no projeto.

## 1.5 WebSite

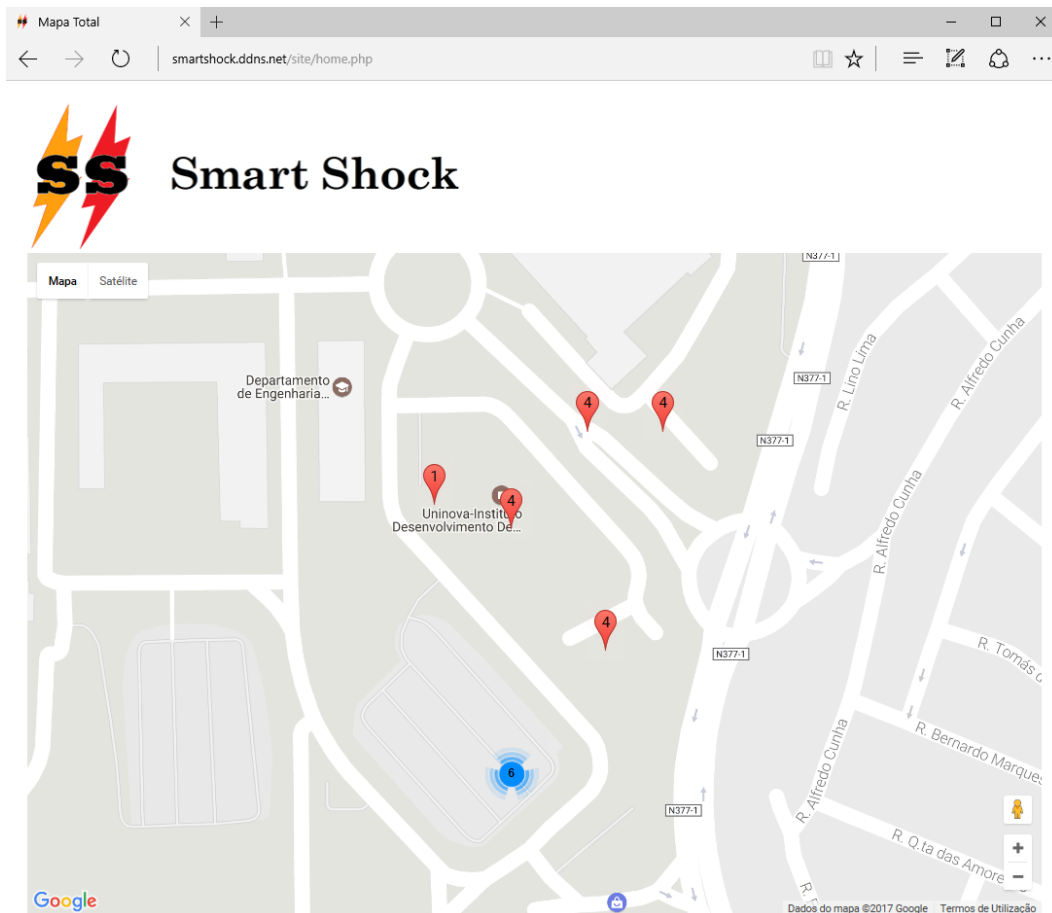
### 1.5.1 home.php

Esta é a página principal da parte *web* desta dissertação. Nela é possível ver-se o logótipo e o nome utilizados neste projeto e também um mapa que contém todas as irregularidades detetadas ou inseridas manualmente, referindo a intensidade das mesmas. Para que essas irregularidades possam ser mostradas, é percorrida toda a base de dados através de uma *query* (questão, em português) de MySQL em que são devolvidos os valores de latitude, longitude e intensidade. Depois, utilizando uma pequena interface desenvolvida pela Google, todos os pontos são marcados no mapa, sendo que no seu sinalizador está indicada a intensidade da irregularidade, como se pode ver na figura 1.4. No caso de existirem várias irregularidades muito próximas fisicamente, a interface agrupa-as automaticamente, informando quantas se encontram nessa zona. A cor destes agrupamentos é alterada consoante o número de itens aglomerados, tornando assim mais fácil a deteção de locais com elevado número de ocorrências.

### 1.5.2 store.php

A página store não contém nenhuma componente gráfica para consulta de informação pois destina-se apenas ao armazenamento de dados na base de dados. Quando o telemóvel envia dados, é este o destino e como tal, tem que existir algum processamento dos dados recebidos. A informação recebida tem o aspecto da figura 1.5 e vem em quatro strings: força, latitude, longitude e data-hora, sendo portanto necessário fazer uma conversão para o formato correcto e consequente armazenamento em variáveis locais. De seguida é feita uma query MySQL para que seja possível fazer o armazenamento na base de dados, sendo emitida uma mensagem de controlo para o telemóvel de modo a dar informação sobre o sucesso ou falha deste armazenamento.



Figura 1.4: Página *web* home.php

### 1.5.3 listLocations.php

Esta página *web* é bastante semelhante à *home.php*, sendo que aqui apenas é mostrada a localização da irregularidade selecionada na página *listlocations.php* de modo a ser mais fácil descobrir a localização da irregularidade quando esta se encontra agrupada com outras irregularidades existentes nas proximidades, como acontece na figura 1.4.

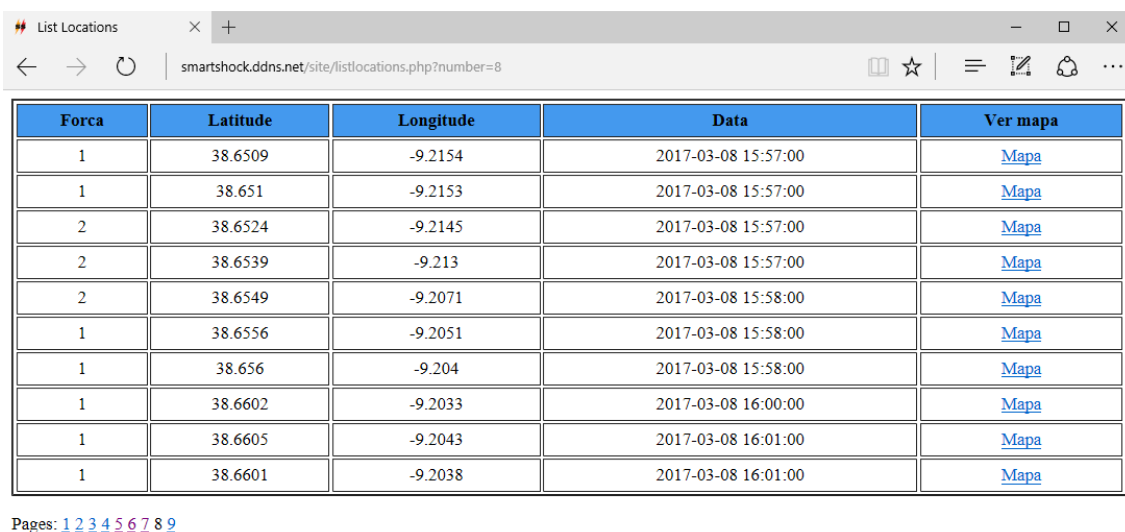
3 +38.6601 -09.2047 2017-02-15 14:37

Figura 1.5: informação tipo adicionada na base de dados

### 1.5.4 showLocation.php

Esta página tem a finalidade de consultar todas as irregularidades num formato numérico, ao invés da página Home.php, em que as irregularidades são apresentadas num mapa.

Tal como apresentado na figura 1.6, nesta página é possível consultar a toda a informação referente a uma irregularidade, incluindo a data e hora em que foi detetada, ao invés da página Home.php. Para que seja necessário visualizar a lista de irregularidades, é necessário colocar no endereço o número da página que se deseja visualizar, sendo o valor pré-definido “1”. Cada uma destas páginas mostra dez entradas da lista de irregularidades e no fundo da página é mostrado um navegador para as diferentes páginas. Alinhado com cada irregularidade, existe uma hiperligação para a página showLocation.php em que é mostrado no mapa a localização da irregularidade seleccionada para uma melhor compreensão dos valores de latitude e longitude.



Força	Latitude	Longitude	Data	Ver mapa
1	38.6509	-9.2154	2017-03-08 15:57:00	<a href="#">Mapa</a>
1	38.651	-9.2153	2017-03-08 15:57:00	<a href="#">Mapa</a>
2	38.6524	-9.2145	2017-03-08 15:57:00	<a href="#">Mapa</a>
2	38.6539	-9.213	2017-03-08 15:57:00	<a href="#">Mapa</a>
2	38.6549	-9.2071	2017-03-08 15:58:00	<a href="#">Mapa</a>
1	38.6556	-9.2051	2017-03-08 15:58:00	<a href="#">Mapa</a>
1	38.656	-9.204	2017-03-08 15:58:00	<a href="#">Mapa</a>
1	38.6602	-9.2033	2017-03-08 16:00:00	<a href="#">Mapa</a>
1	38.6605	-9.2043	2017-03-08 16:01:00	<a href="#">Mapa</a>
1	38.6601	-9.2038	2017-03-08 16:01:00	<a href="#">Mapa</a>

Pages: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#)

Figura 1.6: Página *web* listlocations.php

## BIBLIOGRAFIA

- Chan, C. K., Y. Gao, Z. Zhang e N. Dahnoun (2014). "Implementation and evaluation of a pothole detection system on TI C6678 digital signal processor". Em: *EDERC 2014 - Proceedings of the 6th European Embedded Design in Education and Research Conference*.
- Chen, K., M. Lu, X. Fan, M. Wei e J. Wu (2011). "Road condition monitoring using on-board three-axis accelerometer and GPS sensor". Em: *Proceedings of the 2011 6th International ICST Conference on Communications and Networking in China, CHINACOM 2011*.
- Fouad, M. M., M. A. Mahmood, H. Mahmoud, A. Mohamed e A. E. Hassanien (2014). "Intelligent road surface quality evaluation using rough mereology". Em: *2014 14th International Conference on Hybrid Intelligent Systems*.
- He, Y., J. Wang, H. Qiu, W. Zhang e J. Xie (2011). "A research of pavement potholes detection based on three-dimensional projection transformation". Em: *Proceedings - 4th International Congress on Image and Signal Processing, CISP 2011*.
- Hegde, S., H. Mekali e G. Varaprasad (2015). "Pothole detection and inter vehicular communication". Em: *2014 IEEE International Conference on Vehicular Electronics and Safety, ICVES 2014*.
- Jang, J., A. W. Smyth, Y. Yang e D. Cavalcanti (2015). "Road surface condition monitoring via multiple sensor-equipped vehicles". Em: *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*.
- Kattan, A. e M. F. Aboalmaaly (2014). "A smartphone-cloud application as an aid for street safety inventory". Em: *Proceedings of the 11th International Conference on Electronics, Computer and Computation, ICECCO 2014*.
- Madli, R., S. Hebbar, P. Pattar e V. Golla (2015). "Automatic Detection and Notification of Potholes and Humps on Roads to Aid Drivers". Em: *IEEE Sensors Journal*.
- Mednis, A., G. Strazdins, R. Zviedris, G. Kanonirs e L. Selavo (2011). "Real time pothole detection using Android smartphones with accelerometers". Em: *2011 International Conference on Distributed Computing in Sensor Systems and Workshops, DCOSS'11*.
- Moazzam, I., K. Kamal, S. Mathavan, S. Usman e M. Rahman (2013). "Metrology and visualization of potholes using the microsoft kinect sensor". Em: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*.
- Yu, X. e E. Salari (2011). "Pavement pothole detection and severity measurement using laser imaging". Em: *IEEE International Conference on Electro Information Technology*.

## BIBLIOGRAFIA

---

Zhang, Z., X. Ai, C. K. Chan e N. Dahnoun (2014). “An efficient algorithm for pothole detection using stereo vision”. Em: *IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*.