FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Sistemas de Computação

Móvel e Ubíqua

16/17

CARUINO

Final Report

Turno P2

Prof. Hervé Paulino

Prof. Carmen Morgado

Ana Simão 42978

Pedro Vieira 42610

Índice

1. Introdução

O projeto desenvolvido no âmbito da cadeira de Sistemas de Computação Móvel e Ubíqua consiste na construção de um dispositivo móvel que assume a forma de um pequeno carro e que pode ser controlado através de uma aplicação para *smartphones Android*.

Tendo em conta as definições de "computação móvel", que referem um ambiente em que os componentes de *hardware* podem ser transportados de um lado para outro durante uma utilização regular e que o *software* deve ser construído com o intuito de mitigar os problemas derivados por essa mobilidade, e de "computação ubíqua", que por sua vez indica um modelo que integra a própria computação em objetos e atividades do quotidiano, o grupo considerou que o desenvolvimento de um carro totalmente controlável seria altamente desafiante.

Devido à elevada mobilidade intrínseca à componente deste projeto, aspetos como *energy awareness*, *mobile networking*, garantia de *high-availability* e *location sensitivity and context awareness*, que serão detalhados posteriormente, foram tidos especialmente em conta de forma a desenvolver um sistema inteligente e tolerante a falhas.

2. Descrição Geral

Como descrito anteriormente, o sistema apresenta dois elementos principais: a componente de *hardware* que funciona como um protótipo de um carro, simulando os principais componentes mecânicos do mesmo; a componente de software, apresentada sob a forma de uma aplicação móvel, que serve como controlador do carro, influenciando desta forma os seus comportamentos físicos (movimento frontal, à retaguarda e para os lados).

Por este mesmo motivo, faz sentido que a solução desenvolvida seja utilizada por apenas um utilizador de cada vez, ou seja, a componente física e de *software* estabelecem uma relação de 1:1 (um para um).

O sistema dispensa a utilização de um servidor, pois não necessita nem de obter informação externa ao seu contexto nem de preservar informação entre sessões de utilização.

Para o "a componente física, será utilizado uma placa de *Arduino*, de *Wi-Fi* e todos os componentes periféricos que enumeramos de seguida:

<u>Sensores</u>:

- Sensor ultrassónico

- Fotocélula

Atuadores:

- 2 motores DC
- 3 LEDs

Como referido, o utilizador poderá utilizar a aplicação para controlar a direção na qual o carro se deve movimentar, interagindo desta forma com os motores DC; é também possível ligar e desligar 2 dos LEDs (faróis) através da aplicação. O sistema apresenta também funcionalidades nas quais os atuadores se comportam dependendo dos valores recebidos pelos sensores e interagem de seguida com a aplicação para informar o componente de software das ações que ocorreram.

3. Aplicação Móvel

É através da aplicação móvel que o utilizador vai interagir com o sistema. Esta será desenvolvida com um *design* simples que permitirá o controlo do movimento do carro tanto a nível manual, através de controlos *joystick*, como através da utilização do sensor de giroscópio integrado no aparelho. Para além disso será também possível controlar os faróis do carro através de um botão *on*/*off.*

Desta forma, o projeto cumpre com os requisitos relativos à interface gráfica da aplicação, bem como a utilização de um sensor disponível no telemóvel.

3.1. Descoberta do carro na rede

Desde a idealização do projeto, foi decidido que apenas faria sentido o utilizador controlar o carro se estivesse fisicamente próximo dos seus componentes de *hardware*. Para garantir isto, a componente de *Wi-Fi* do *Arduino*, ao ser inicializada, emula um *Access Point*. Desta forma, utilizamos o processo de descoberta de redes *Wi-Fi built-in* no próprio sistema operativo *Android* para verificar se o utilizador já se encontra na cobertura do sistema: se estiver próximo, o *AP* "CARUINO" aparecerá na lista do *Wi-Fi*; caso contrário, o utilizador ainda não se encontra no local correto e a aplicação continuará à procura.

Toda esta lógica foi construída utilizando a classe *WifiManager* do *Android*, nomeadamente o *intent* "*SCAN_RESULTS_AVAILABLE_ACTION*", que possibilita a obtenção das redes *Wi-Fi* atualmente disponíveis. Assim, se nos resultados devolvidos existir uma entrada com o *SSID* "CARUINO", a aplicação enviará uma chamada ao sistema para se conectar programaticamente a essa rede. Após este pedido de associação ao *Basic Service Set* do *AP* terminar com sucesso, a

aplicação irá estabelecer uma conexão *TCP* com o servidor criado no *Arduino*, usando o *IP* e *port* usados aquando da sua inicialização, para que possa efetivamente enviar e ler comandos do carro. Caso não seja emitida qualquer exceção durante a criação do *socket*, considera-se que a aplicação se encontra corretamente ligada ao *Arduino*.

Figura 1 Página inicial onde é apresentado o loading spinner

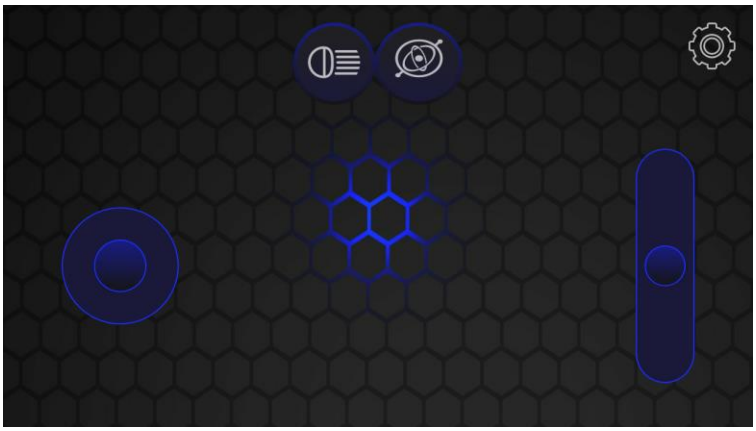

Figura 2 Vista principal onde estão presentes os controladores

Durante a totalidade deste processo, a aplicação mostrará ao cliente uma *View* simples, informando-o que se encontra ativamente à procura do carro (fazendo uso de um *loading spinner* - fig. 1), e após oficializar a conexão, uma nova vista, com todos os controlos, será apresentada (fig. 2).

3.2. Controlo dos faróis

Como foi referido anteriormente, dois dos LEDs conectados ao *Arduino* têm o objetivo de simular os faróis do carro. De forma a controlá-los, foi disponibilizado ao utilizador, um botão on/off que, quando clicado, envia um pacote pela rede com a ação (ligar ou desligar) associada. De forma a dar *feedback* visual ao utilizador sobre qual o estado das luzes, a cor do ícone é alterada quando estas estão desligadas (fig. 3) e ligadas (fig. 4).

Figura 3 Aspeto do ícone quando luzes estão desligadas

Figura 4 Aspeto do ícone quando as luzes estão ligadas

Para além do controlo manual dos faróis, a aplicação também disponibiliza uma configuração que permite automatizá-los consoante a intensidade de luz presente no espaço atual (figs. 5, 6 e 7). Ou seja, se existir muita luminosidade os faróis do carro permanecerão desligados e, quando a intensidade de luz diminuir, os LEDs serão ligados automaticamente, sem qualquer interação por parte do utilizador.

Figura 3 Botão das definições

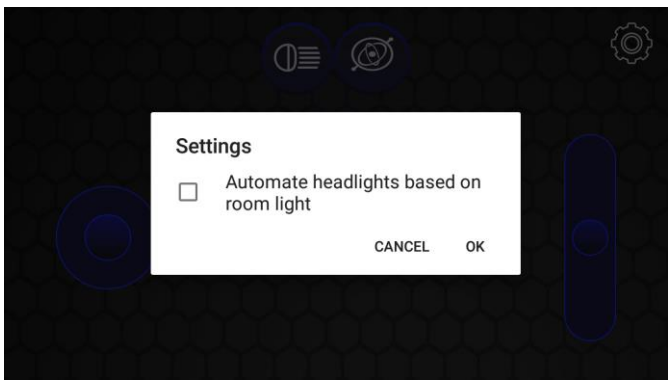

Figura 5 Opção de automatização de luzes desativada

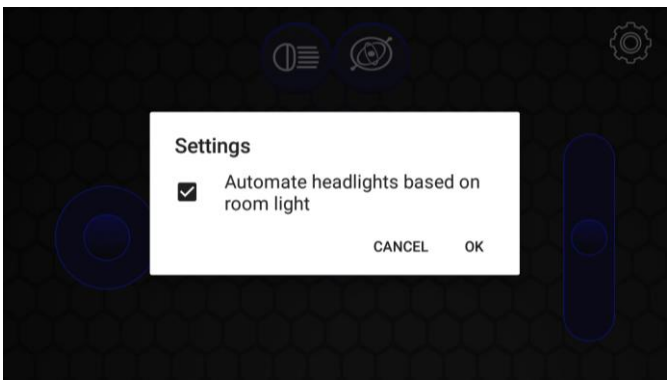

Figura 4 Opção de automatização de luzes ativada

Toda esta automatização é processada pela componente do *Arduino* com o auxílio de sensores, como será descrita na secção sequente, e é esta que notifica o cliente quando existem alterações, para que este possa atualizar a sua *UI*, com o intuito de dar *feedback* ao utilizador, quando as luzes se ligam após uma diminuição da intensidade de luz no local atual (fig. 9), e quando estas passam para o estado contrário quando existe luz suficiente (fig. 8). De notar que quando esta automatização se encontra ativada, a letra 'A' (de "*automatic*") é embutida no topo do botão dos faróis e que o utilizador não poderá interagir manualmente com este botão até que esta definição seja desativada.

Figura 7 Aspeto do botão quando luzes são automaticamente ligadas

Figura 6 Aspeto do botão quando luzes são automaticamente ligadas

3.3. Controlo do movimento do carro

A aplicação disponibiliza duas formas de o utilizador interagir com os motores do carro, de forma a controlar o seu movimento: através dos clássicos comandos *joystick* e através do próprio movimento do telemóvel, usando o giroscópio.

Relativamente ao movimento através dos *joysticks*, a aplicação disponibiliza um para o movimento lateral do carro, do lado esquerdo, e outro para o movimento frontal (frente e trás) do lado direito (fig. 10).

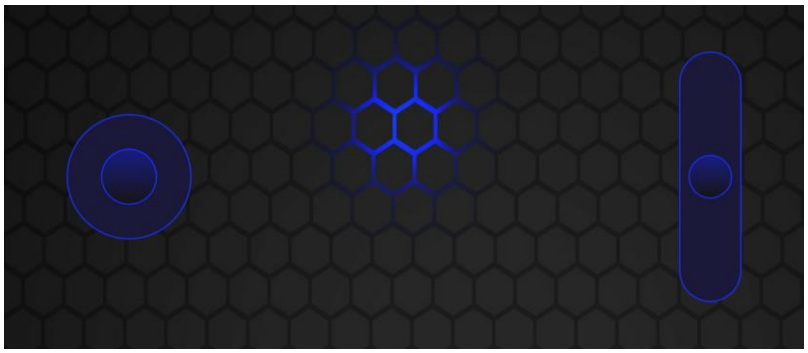

Figura 8 Controlos joystick

Assim, com estes *joysticks*, o utilizador poderá interagir com eles e movimentar o carro para a frente e para trás (fazendo uso apenas do *joystick* direito), avançar com o carro para a direita ou para a esquerda e fazendo marcha-atrás para a direita ou esquerda (usando ambos os *joysticks* em simultâneo). Desta forma, sempre que o utilizador arrasta estes controlos, a aplicação enviará o respetivo comando pela rede de forma a que o *Arduino* se movimente. Por outro lado, como suprarreferido, também é disponibilizado um outro mecanismo de controlo que usa os sensores do próprio dispositivo *smartphone* para que o utilizador possa controlar o carro como se o seu telemóvel fosse um volante digital. Para aceder a este modo de interação, o cliente poderá premir o botão presente no topo do ecrã (figs. 11 e 12).

Figura 10 Botão do giroscópio

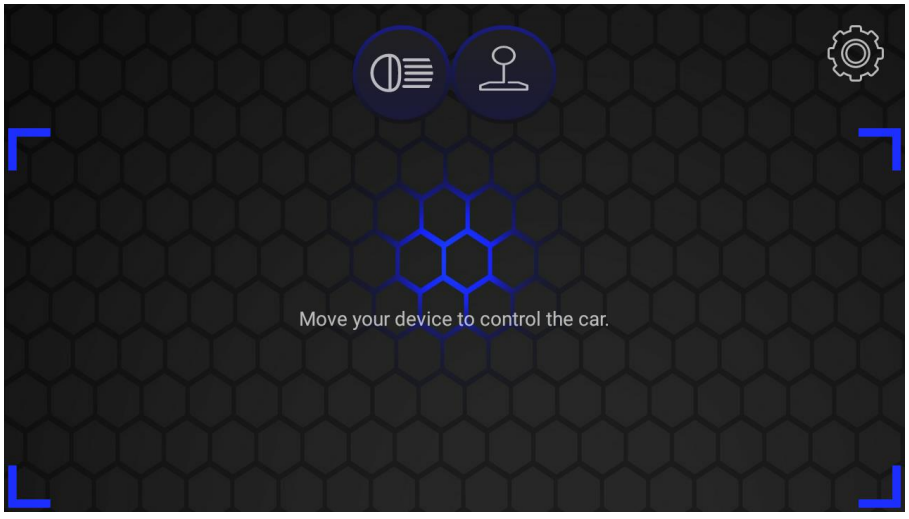

Figura 9 Vista do controlo do giroscópio

Inicialmente, a ideia seria utilizar diretamente o sensor de giroscópio do telemóvel, obter os valores referentes à orientação do dispositivo e agir consoante essa informação. No entanto, devido a algumas limitações do telemóvel usado pelos membros do grupo, nomeadamente a ausência deste sensor, foi necessário repensar a abordagem ao problema. Após alguma pesquisa, foi verificado ser possível simular um giroscópio com o auxílio de dois outros sensores, nomeadamente o acelerómetro e o sensor do campo magnético. Assim, a cada leitura, a aplicação combina os valores de cada sensor e gera a matriz de rotação e orientação de um sistema de coordenadas cartesiano de três dimensões, em que os eixos denominam-se *pitch* (*x*), *roll* (*y*) e *azimuth* (*z*), e que é utilizada para estimar a orientação do dispositivo (fig. 13).

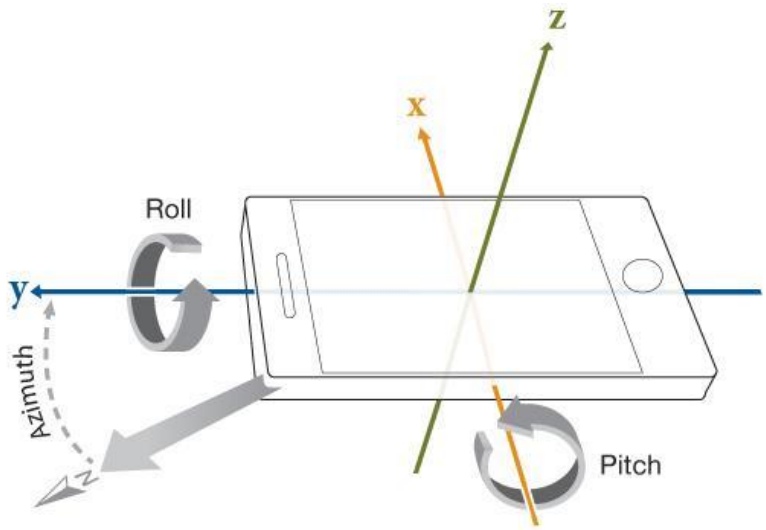

Figura 11 Representação do sistema de eixos e coordenadas utilizado

Desta forma, a aplicação utiliza os valores de rotação no eixo do *roll* para movimentar o carro para a frente e para trás e no eixo do *pitch* para virá-lo para a esquerda e para a direita. A razão pela qual usamos o eixo do *pitch* para o movimento lateral do carro ao invés do eixo do *azimuth*, que simularia uma utilização similar à rotação de um volante verídico, reside no facto de, após múltiplos testes, os valores obtidos nesse eixo apresentarem variações profundamente inconsistentes.

No que toca ao sistema de coordenadas referido, quando o utilizador pressiona no botão do giroscópio a aplicação inicia o seu processo de *sampling* dos dados de rotação e orientação, lendo os dez primeiros valores obtidos dos sensores. Este processo permite-nos obter uma média dos valores de cada eixo e calcular o ponto (0, 0, 0) - ou base - para a orientação e rotação atual do dispositivo. Desta forma, ao invés de estabelecermos a partir de que graus

específicos e constantes consideramos que o dispositivo se encontra orientado para a frente, para trás, etc., utilizamos *offsets* (em graus) a partir da posição base, obtida anteriormente, do telemóvel.

Assim, à medida que utilizador roda o seu dispositivo, a *user interface* mostrará dicas visuais, através de barras, que indicarão qual o sentido do movimento do carro (figs. 14-19).

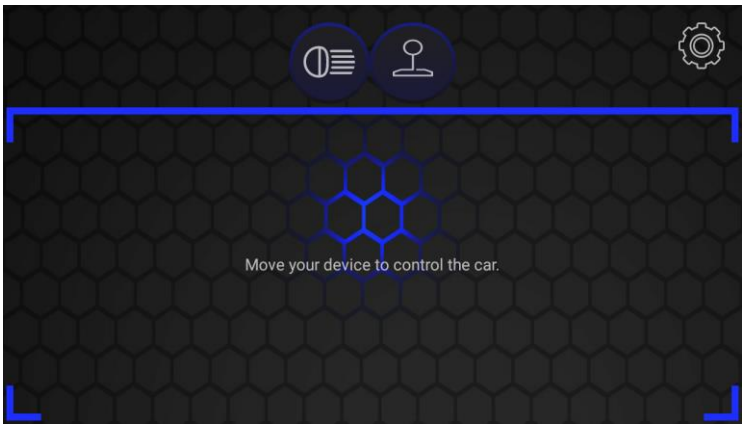

Figura 17 Representação do comando frente

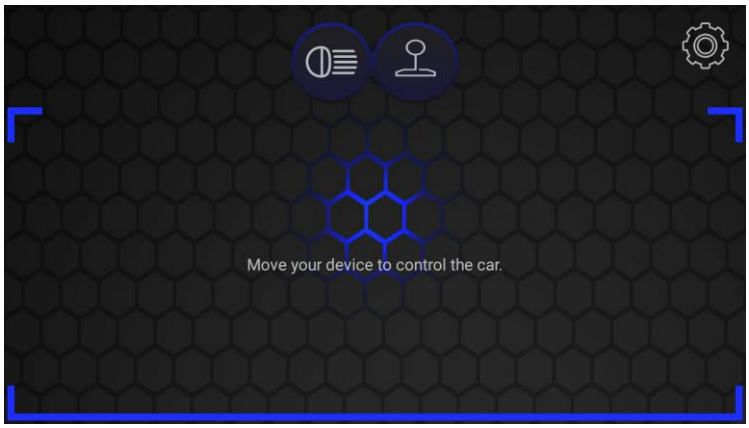

Figura 16 Representação do comando trás

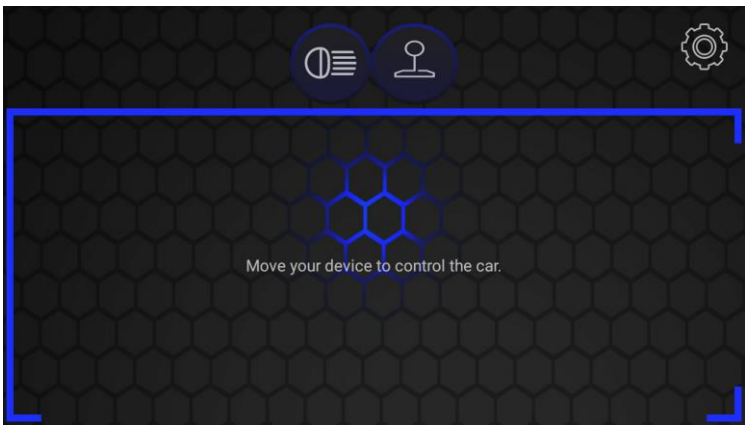

Figura 15 Representação do comando frente e esquerda

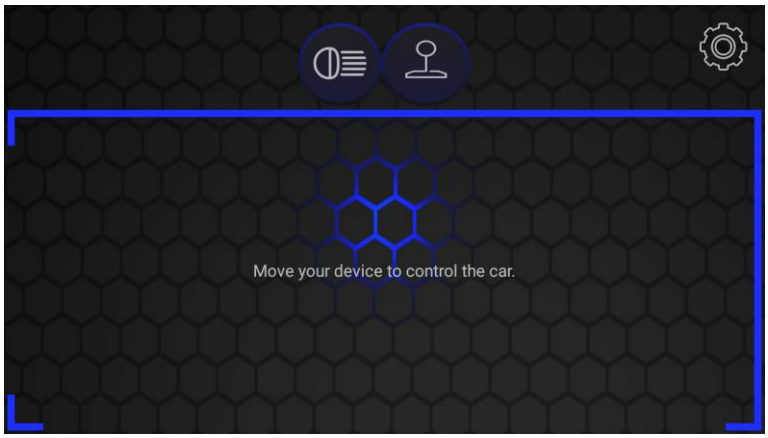

Figura 14 Representação do comando frente e direita

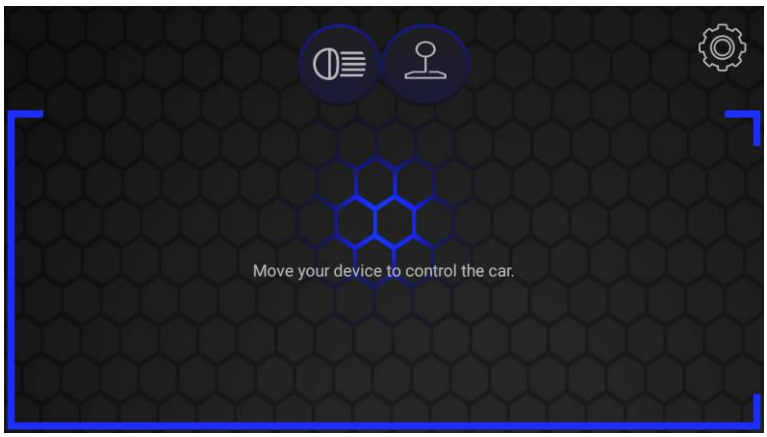

Figura 13 Representação do comando trás e esquerda

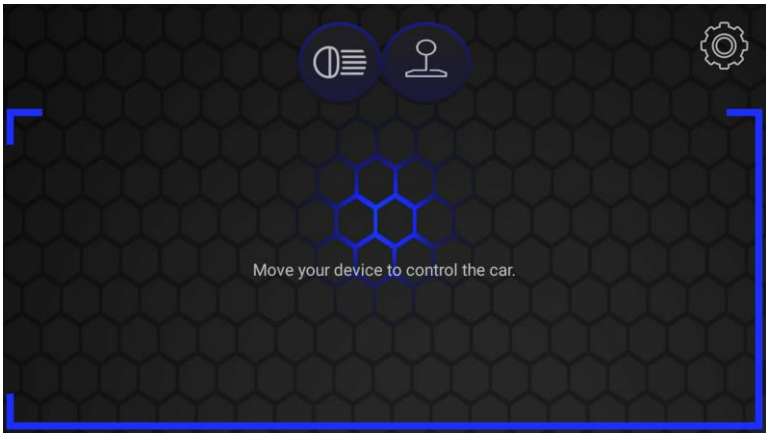

Figura 12 Representação do comando trás e direita

Para voltar ao controlo clássico do movimento do carro, o utilizador poderá pressionar no botão com o ícone do *joystick*, situado no topo do ecrã (fig. 20).

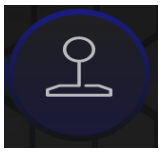

Figura 18 Botão de joystick

3.4. Notificação de colisão iminente

À semelhança do que se sucede durante a automatização dos faróis do carro consoante a intensidade de luz atual, o *Arduino* também notifica a aplicação quando os valores obtidos do seu sensor ultrassónico indicam que este se encontra próximo de uma parede ou obstáculo, impossibilitando o carro de progredir (este aspeto será descrito em maior pormenor, nomeadamente como os valores deste sensor são obtidos e calculados). Assim, quando a aplicação verifica que um pacote foi recebido pelo *socket* de comunicação e que este pacote representa uma paragem completa, um pequeno alerta é apresentado ao utilizador a informar da presença do obstáculo (fig. 21).

Figura 19 Representação do alerta de obstáculo

3.5. Envio dos comandos pela rede

Sempre que o utilizador interage com um controlo através da aplicação, quer seja com os *joysticks*, botão dos faróis ou a automatização dos *LEDs*, existe um comando associado a essa ação que é enviado para o *Arduino* para que este saiba o que é necessário executar. De seguida, apresentamos a tabela que contém todos os comandos disponíveis e a sua representação programática.

Comando	Representação
Parar motores (*STOP*)	‘0’

Avançar	‘w’
Marcha-atrás	‘s’
Avançar para a esquerda	‘q’
Avançar para a direita	‘e’
Marcha-atrás para a esquerda	‘z’
Marcha-atrás para a direita	‘x’
Ligar faróis	‘o’
Desligar faróis	‘f’
Automatizar faróis	‘n’
Desativar automatização de faróis	‘m’

Tendo em conta o contexto de utilização deste sistema, é expectável para o utilizador que, enquanto um *joystick* é arrastado, o carro permaneça em movimento. Para tal, espera-se que aplicação esteja de forma sensivelmente contínua a enviar o comando de avançar para a frente, por exemplo, para o carro.

Baseado nesse comportamento, foram implementados diversos mecanismos adicionais, maioritariamente no âmbito de *energy awareness*, de forma a minimizar os recursos utilizados e a ocorrência de congestionamentos na rede, potenciados por esta interação continuada entre a aplicação e a componente de *hardware*.

1. Envio periódico de comandos: ao invés de se enviar instantaneamente um comando para o *Arduino* sempre que alguma ação é despoletada pelo utilizador, a aplicação contém uma *queue* onde vai colocando comandos no fim. Por sua vez, existe uma *background thread* que tem o único propósito de consumir o próximo comando à cabeça da estrutura de dados, enviá-lo para o *Arduino* através do *socket* previamente construído, realizar *sleep* durante um pequeno *timeout* e assim sucessivamente. Para além disto, também foi desenvolvido um mecanismo de deteção e rejeição de comandos repetidos: sempre que um comando é executado, este é guardado; se o utilizador executar a mesma ação num intervalo de tempo inferior a 100ms (melhor valor encontrado após a realização de múltiplos testes), considera-se que o envio desse comando é desnecessário e, por sua vez, é ignorado; caso o comando seja o mesmo mas o *timeout* tenha passado ou o comando seja diferente, este será automaticamente

inserido na estrutura de dados para ser enviado num *loop* posterior. Tendo em conta que o envio de pacotes pela rede é um processo que utiliza uma quantidade energia considerável, uma das razões pelo qual é geralmente preferível usar um protocolo de *publish-subscribe* num ambiente de computação móvel e ubíqua pois o número de mensagens enviadas pela rede é reduzido e a energia gasta na receção de pacotes é extremamente mais baixa. Tendo em conta estes dois mecanismos *energy-wise* introduzidos, a aplicação encontra-se mais otimizada, pois o número de comandos semelhantes sequentes enviados diminuirá drasticamente e o envio de pacotes será faseado. Outras vantagens destes mecanismos passam pela diminuição da probabilidade de causar um congestionamento de pacotes na rede e de causar um *overflow* do *buffer* de receção de pacotes na componente de *Wi-Fi* do *Arduino*;

2. Tamanho reduzido dos pacotes enviados: como é possível verificar na tabela supra, todos os comandos são representados por um único caractere. Apesar desta escolha aparentar ser simples, aleatória e pouco legível (ao invés de se utilizar identificações extensas dos comandos, como por exemplo “forward” e “stop”), existe uma razão rigorosa para o sucedido. Ao enviar apenas 8 *bits*, o tamanho mínimo possível, os pacotes emitidos pela rede terão um tamanho extremamente reduzido, contribuindo, à semelhança do ponto anterior, para a diminuição de casos de congestionamento e *overflow* do *buffer*. Paralelamente, isto aumentará o tempo de resposta da componente de *hardware* pois esta apenas necessita de pedir um único *byte* ao *Serial* de receção proveniente da componente de *Wi-Fi*.

4. Arduino / *Sensing and Reacting*

Complementando as informações apresentados no início deste relatório relativamente aos sensores e atuadores, na seguinte listagem foram acrescentados alguns detalhes extra sobre o seu funcionamento e como estão integrados no sistema.

Sensores:

- Sensor ultrassónico para medição de distâncias que é usado para detetar e evitar obstáculos;
- Fotocélula para deteção de luz para ligar/desligar os faróis automaticamente consoante a intensidade da luz presente no local.

Atuadores:

- Motores DC para o movimento das rodas. Estes motores são controlados através da aplicação, quer usando *joysticks* quer usando o *giroscópio* do dispositivo;
- LEDs amarelos para simular o comportamento dos faróis que poderão ser ligados/desligados através do botão on/off presente na aplicação. Para além disso, o estado destes LEDs também poderá ser alterado relativamente consoante os valores obtidos no sensor de luminosidade;
- LED vermelho para indicar ao utilizador que está conectado ao *Arduino*. Este LED é ativado aquando a conexão de um cliente e é ligado/desligado intermitentemente com o intuito de informar o utilizador que este se conectou com sucesso ao carro.

O esquemático da ligação de todos os componentes *hardware* é o seguinte (fig. 22):

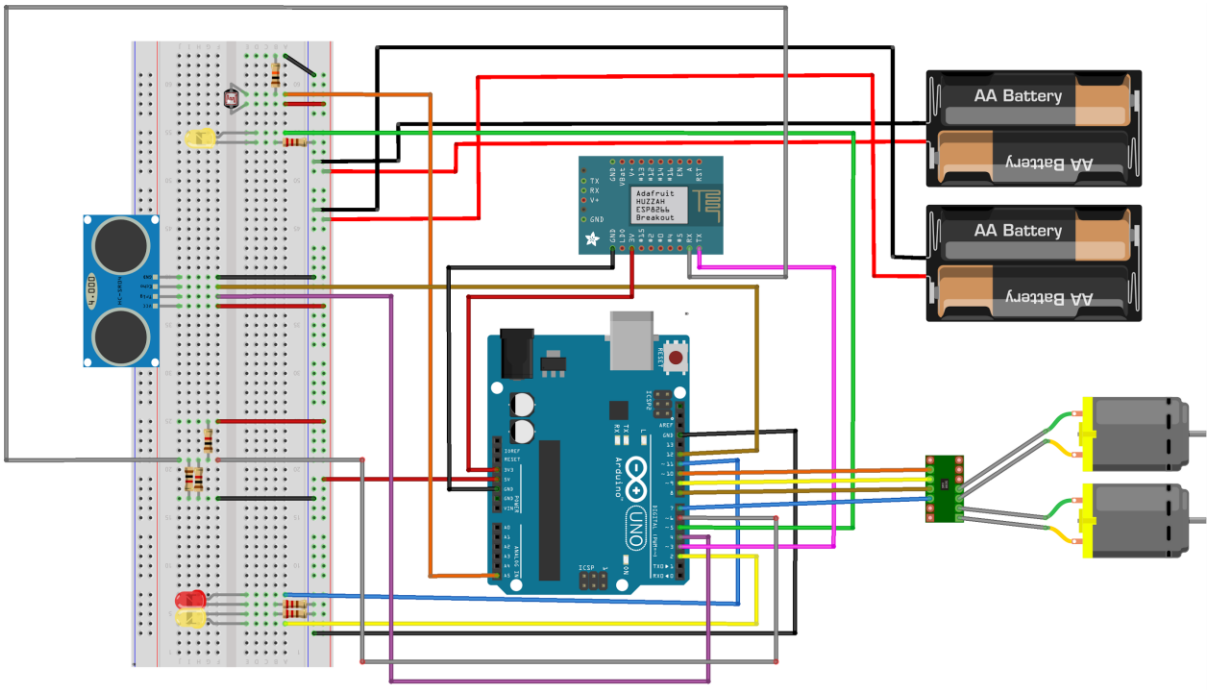

Figura 20 Esquemático da componente de hardware

De seguida, iremos enumerar várias ocasiões em que as componentes de *hardware*, nomeadamente os atuadores, reagem, *on the fly,* consoante os valores obtidos dos sensores. Para além disso, também entraremos em detalhe em

algumas funcionalidades adicionais que foram implementadas, pois acreditamos que estas contribuem, de uma ou outra forma, para o correto funcionamento geral do sistema.

4.1. Sensor de luminosidade (fotocélula)

Como foi referido na secção 3.2, o sistema disponibiliza uma configuração ao utilizador que permite automatizar todo o processo de ligar e desligar os faróis do carro tendo em conta a intensidade da luz presente no local em que o carro se encontra. Desta forma, quando o utilizador envia o comando com o intuito de iniciar este processo de automatização, nomeadamente o caractere 'n', o *Arduino* passa para um estado em que, para cada iteração da função *loop*, este encontra-se ativamente a obter valores do sensor de luminosidade para se autocalibrar.

Para obter estes dados provenientes da fotocélula, a componente de *hardware* executa cinco *samplings* seguidos ao respetivo sensor, de forma a obter uma média da luminosidade e decidir como agir a partir daí. Este tipo de cálculo de valores finais através da média de *x* leituras consecutivas é crítica para o correto funcionamento desta *feature*: após alguns testes iniciais, verificámos que os valores obtidos por este sensor apresentam, por vezes, algumas variações bruscas, variações essas que poderiam fazer com que os LEDs piscassem, passando do estado *HIGH* para *LOW* e vice-versa, momentaneamente sem qualquer razão aparente para o utilizador. Assim, ao realizar um processo de *smoothing* na *data set* obtido do sensor, é possível dar menos peso ao *noise* obtido.

Após ter sido realizado o processo acima descrito, iniciou-se um novo leque de experiências de forma a obter um *threshold* aceitável para considerarmos quais os intervalos de valores da intensidade luminosa para o qual é necessário ligar e desligar os LEDs. Após algumas experiências, como referidas na secção seguinte, verificámos que os faróis deverão ser/manter-se desligados quando o valor médio obtido da fotocélula situa-se no intervalo de *[0, 70[* e deverão ser/manter-se ligados quando este se encontra no intervalo de *[70, +∞[*.

Por fim, assim que o *Arduino* verifique que os valores do sensor tenham passado de um intervalo para o outro, este notifica a componente de *Wi-Fi* para que esta, por sua vez, informe a aplicação com o intuito de atualizar a sua *UI* (como descrito na secção 3.2).

4.2. Sensor ultrassónico

Uma das funcionalidades extra que foi considerada importante de ser implementada, e que não é costume existir nos sistemas gerais de controlo de carros telecomandados, é a adição de um sensor ultrassónico que será usado para verificar a existência de um obstáculo próximo da parte frontal do *chassis* do carro.

Caso exista algum objeto numa distância de *x* centímetros (distância essa que é dada e que representa um *threshold*), o *Arduino* imobilizará os motores DC de forma a evitar uma colisão. Esta *feature* encontra-se intrinsecamente associada ao âmbito de *Location Sensitivity and Context Awareness*.

Similarmente ao que foi descrito na subsecção anterior, os valores deste sensor são também obtidos através de cinco *samplings* consecutivos para que, no final, se consiga obter uma distância média, em centímetros, resultante deste processo de *smoothing*. Este processo torna-se ainda mais necessário ao lidar com os dados obtidos por este sensor do que pelo de luminosidade: este *sonar* apresenta alguns resultados erráticos quando os obstáculos não estão posicionados de forma frontal relativamente ao próprio sensor. Isto ocorre, principalmente, quando o carro está em movimento, pois existe uma maior probabilidade deste se deparar com obstáculos circulares e/ou diagonais, em ângulos pouco favoráveis para a deteção (ângulos superiores a 30º, como demonstra a fig. 23), os sons acabam por ser desviados e o sinal de eco pode não atingir o sensor de volta ou alcançá-lo de forma muito atenuada e, portanto, não ser detetado corretamente. Esta falta de consistência em relação aos valores lidos pelo sonar é a principal justificação para a importância de efetuar o processo de *sampling*.

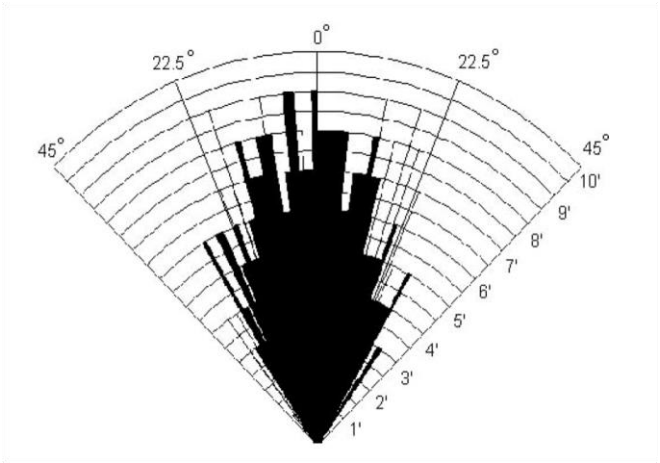

Figura 21 Representação da deteção de ondas relativamente ao ângulo

Inicialmente, a ideia seria ativar o *sonar* a cada iteração da função *loop* do *Arduino* durante a execução de um comando de movimentação frontal, para que a rotação dos motores fosse imediatamente interrompida assim que o carro estivesse prestes a colidir com um obstáculo. No entanto, verificou-se que todo o mecanismo de envio e receção de ondas de som de alta frequência, inerente ao sensor ultrassónico e em cada um dos 5 *samples* pedidos a cada *loop*, acrescentava um *overhead* considerável ao *Arduino*. Este *overhead* era especialmente notável pois

impossibilitava o *Arduino* de consumir os comandos provenientes do *Serial* da camada *Wi-Fi* tão rápido quanto estes eram produzidos pela aplicação móvel, algo que não se sucedia anteriormente. Por sua vez, esta problemática contribuiu para uma redução da *responsiveness* do sistema, aumentando o *delay* entre execuções de comandos ao ponto de ser percetível para o utilizador.

Assim, de forma a contrariar esta perda de *performance*, ambas as partes, a equipa de desenvolvimento em conjunto com os docentes da cadeira, chegaram a um compromisso e apenas é verificado se a componente do carro está prestes a colidir com um obstáculo uma única vez, antes da execução de cada comando de movimento. Apesar desta escolha não ser ideal, pois possibilita a colisão após um comando ter sido inicialmente executado, acreditamos ser uma solução aceitável tendo em conta os seus *trade-offs*.

Apesar de se ter enveredado pela implementação descrita, consideramos que existe uma outra alternativa extremamente mais eficiente e otimizada: quando um comando de movimentação fosse executado o valor do *sonar* seria obtido e armazenado localmente e, a cada *loop*, tendo em conta o tempo decorrido desde o início do comando, seria estimada a distância percorrida pelo carro e essa quantia subtraída ao valor previamente adquirido do sensor ultrasónico. Desta forma, já seria possível simular um *pseudo sonar* a cada iteração do *loop* sem qualquer *overhead* considerável, dado que estas são simples operações aritméticas. No entanto, devido à dececionante potência dos motores *DC* disponibilizados para a criação deste projeto, que por sua vez não tinham capacidade de movimentar o carro construído, não nos foi possível realizar testes de forma a computar um valor possível para a distância percorrida, por segundo, pelo carro para a realização dos cálculos estimativos descritos.

4.3. Mobile Networking e High-Availability

Um dos desafios intrínsecos a este paradigma de computação móvel e ubíqua é o *Mobile Networking* que, sucintamente, baseia-se no princípio de garantir *high-availability* ao cliente caso este esteja constantemente a ser desconectado. Por sua vez, uma das características dos ambientes de comunicação de redes sem fios é que estes têm comportamentos muito díspares comparativamente a redes *wired*, nomeadamente a sua fragilidade no envio de dados através do meio ambiente que potencia perdas de pacotes. Estas perdas de pacotes podem ser causadas por interferências, ou *noise*, no meio, que são geradas por outras tecnologias que usem a mesma frequência, por outros objetos que emitem radiação eletromagnética ou simplesmente porque o sinal é refletido por obstáculos antes de chegar ao destino. Desta forma, um dos focos da equipa de desenvolvimento desde a conceção do projeto foi permitir a continuidade da execução do carro mesmo que, por algum motivo, existam falhas de ligação

intermitentes entre a aplicação e a componente de *hardware*, fornecendo ao utilizador uma melhor experiência, de forma totalmente transparente.

Para garantir esta funcionalidade, foi desenvolvido um pequeno mecanismo adicional, ao qual denominamos de "replicação sequencial". Assim que o *Arduino* recebe um pedido de execução de um comando, proveniente da aplicação móvel, este executa-o, guarda localmente esse mesmo comando e o *timestamp* do início da sua execução. A partir daí, o sistema irá replicar, de forma contínua, a execução dessa mesma operação até que receba pacote que contenha um comando diferente do anteriormente executado ou até que já tenha decorrido pelo menos um segundo (valor ótimo obtido através de diversos testes de conexão) do *timestamp* obtido anteriormente, e assim sucessivamente. Comandos de caráter atómico, que necessitem de ser executados instantaneamente, como o "STOP" ou a interação com os LEDs, não estão, propositadamente, abrangidos pela replicação sequencial.

Esta propriedade possui duas grandes vantagens:

1. Possibilita uma utilização com perdas de pacotes intermitentes sem que esta ocorrência cause uma degradação na experiência do utilizador;
2. Caso a componente de *hardware* receba múltiplos comandos idênticos seguidos num curto espaço de tempo, *energy e resource-wise*, o sistema ignorá-los-á fazendo com que não necessite de utilizar recursos computacionais para decifrar, preparar os atuadores e sensores e inicializar um comando que já se encontra atualmente a ser executado.

4.4. Operação de desconexão

De modo a proporcionar um *feedback* visual, assim como uma funcionalidade mais significativa do que apenas o ligar e desligar de um LED, como ocorre aquando da conexão do sistema, quando o utilizador termina a aplicação ou por algum outro motivo sai da cobertura da componente *Wi-Fi* do sistema e a conexão entre o *Arduino* e a aplicação é desfeita, o atuadores procedem a um conjunto de operações que correspondem a todos os comandos anteriormente efetuados pelo utilizador durante a sessão, executados pela ordem inversa à da original (o primeiro comando é executado por último e vice-versa), com o intuito de voltar, teoricamente, à posição inicial do carro aquando da conexão do utilizador.

Para tornar esta funcionalidade possível foi necessário guardar os comandos efetuados, tal como o tempo de execução de cada um destes. Para tal foram utilizados dois *arrays* contíguos com 300 entradas cada (devido à capacidade de memória limitada do *Arduino*), em que num dos *arrays* serão

armazenados os comandos concluídos em formato *char* e no outro serão adicionados os tempos de execução, em milissegundos, de cada um destes em formato *int*. Em relação aos comandos, estes serão inseridos quando o *Arduino* recebe da aplicação um comando diferente do que está atualmente a ser executado e, no caso dos comandos contínuos, quando o tempo de execução máximo estabelecido é atingido.

Antes da implementação desta solução, foi considerada uma outra abordagem que passava pela utilização da estrutura de dados *LinkedList*. À primeira vista esta seria uma solução mais adequada para o problema em questão e ofereceria uma solução mais simplificada e legível conceptualmente. No entanto, ao considerar as características próprias deste sistema, nomeadamente o facto de ser utilizado equipamento *hardware* relativamente limitado em termos de recursos, verificou-se que esta abordagem iria acrescentar um *overhead* desnecessário à placa *Arduino* e teria uma maior necessidade de utilização de memória, pois ao invés de cada entrada da *LinkedList* utilizar 8 *bits* (que representam os comandos efetuados) + 16 *bits* para os inteiros (que representam o tempo de execução de cada comando) respetivamente — de notar que um inteiro na plataforma *Arduino* ocupa 2 *bytes* —, como ocorre na solução previamente descrita, cada entrada necessitaria de 8 *bits* + 16 *bits* + 16 *bits* (tempo de execução) respetivamente. Os 16 bits adicionais devem-se à necessidade de existir um apontador para o próximo item guardado, comportamento específico desta estrutura de dados.

5. Experiências

Os testes conduzidos no sistema foram feitos de forma progressiva, acompanhando o desenvolvimento do projeto, de forma a garantir que cada componente tinha o comportamento esperado antes de se proceder à implementação da componente seguinte. Numa fase final foram desenvolvidas experiências mais complexas que envolvem componentes de *hardware* e de *software* e a respetiva comunicação e interação.

1. Teste do funcionamento do sensor de luminosidade e comportamento dos faróis

Como referido anteriormente, existem 2 formas distintas de ligar/desligar os LEDs que representam os faróis do carro, automaticamente e de forma manual através da aplicação. Em relação ao comportamento automático, é esperado que a luzes permaneçam desligadas num ambiente de elevada luminosidade e, num

ambiente com pouca luminosidade, os 2 LEDs devem acender, dando o *feedback* correspondente à aplicação e consequentemente ao utilizador.

Para este teste assume-se que a definição para gerir o comportamento dos faróis consoante a luminosidade do espaço está ativa. Desta forma, o teste inicia-se com os faróis desligados e o sensor num ambiente de elevada luminosidade, de seguida o sensor foi encoberto de forma a que a luminosidade à volta deste diminuísse e, como consequência disso, verificou-se então a ligação dos faróis e a atualização da *UI* correspondente. Para terminar, o sensor foi novamente colocado nas condições de luminosidade iniciais e os faróis foram então desligados.

Para testar a funcionalidade manual assume-se que a definição para gerir o comportamento dos faróis consoante a luminosidade do espaço encontra-se desativada. Para este teste basta simplesmente utilizar o botão dos faróis na aplicação e verificar se o comportamento das luzes corresponde ao *feedback* mostrado pela aplicação, ou seja, quando o ícone se encontra branco os LEDs devem permanecer desligados e quando o ícone se encontra amarelo, os LEDs devem estar ligados.

2. Teste do funcionamento do sonar

Nesta experiência, e tendo em conta as limitações de hardware existentes, assume-se que não aparecem obstáculos externos ao ambiente de teste. Para este teste é esperado que o carro não se desloque caso seja detetado um obstáculo próximo do carro (e consequentemente do *sonar*) e que o utilizador receba um alerta visual através da interface da aplicação. Caso contrário, o carro deve executar o seu movimento conforme as indicações do utilizador. É importante também referir que devido à utilização de apenas um sonar, só é possível executar este teste apenas quando o carro se desloca para a frente (direção para a qual o sonar existente aponta).

Assim sendo, na primeira experiência a aplicação foi utilizada para controlar o movimento do carro (para a frente), sendo que não existia nenhum obstáculo próximo e tal como esperado, ambos os motores rodaram na direção esperada.

Numa segunda tentativa foi colocado um obstáculo a poucos centímetros do sensor e, mais uma vez, como esperado, os motores não executaram qualquer movimento e o utilizador foi notificado da existência de um obstáculo através da aplicação.

Um teste que consideramos também importante, mas que não foi possível executar devido a atrasos na implementação do comportamento adequado, devido a problemas com o *hardware* disponível, consistiria em iniciar a experiência com um obstáculo a uma distância suficientemente grande do sensor para que os motores iniciassem o seu movimento mas que, ao ocorrer uma aproximação, o

alerta fosse enviado e os motores parassem a sua rotação de forma a que o carro se imobilizasse.

3. Teste do controlo dos motores

Para esta experiência assume-se que a conexão entre a aplicação e o *Arduino* foi bem-sucedida e que os valores lidos pelo sonar estão a ser levados em conta, ou que por e simplesmente não existem obstáculos no caminho do dispositivo móvel.

Nesta experiência as consequências e *feedback* das ações executadas são unilaterais, ou seja, o utilizador envia as suas ordens através da aplicação e o hardware corresponde com o comportamento apropriado. De forma a colocar o sistema à prova em todos os cenários possíveis, os testes foram executados em todas as direções utilizando as duas alternativas de controlo, os *joysticks* e o movimento do *smartphone*.

De seguida são listadas as várias direções tomadas a cada teste e o respetivo resultado.

- O comando dado é unicamente para a frente: ambos os motores rodam para a frente;
- O comando dado é unicamente para trás: ambos os motores rodam para trás;
- O comando dado é unicamente para o lado esquerdo: nenhum dos motores executa qualquer movimento (foi estabelecido que para o dispositivo virar é necessário que o utilizador desloque os controlos na direção que pretende virar e numa das duas direções frontais);
- O comando dado é unicamente para o lado direito: nenhum dos motores executa qualquer movimento
- O comando dado é simultaneamente para a frente e para o lado esquerdo: o motor direito vai rodar para a frente;
- O comando dado é simultaneamente para a frente e para o lado direito: o motor esquerdo vai rodar para a frente;
- O comando dado é simultaneamente para trás e para o lado esquerdo: o motor direito vai rodar para trás;
- O comando dado é simultaneamente para trás e para o lado direito: o motor esquerdo vai rodar para a frente;

Todos os comportamentos atrás descritos estão de acordo com o esperado do sistema proposto.

4. Teste da operação de conexão/desconexão

Para este teste assume-se que a placa de *Wi-Fi* está corretamente ligada e programada, bem como toda a lógica por trás da conexão por parte do *Arduino* e da aplicação (explicada em detalhe anteriormente). Quanto ao *hardware,* foi destacado um LED cuja única função é piscar (acender e apagar repetidas vezes) aquando de uma conexão bem-sucedida. Para além disto, também um LED da própria placa de *Wi-Fi* é ligado quando a conexão se estabelece; em simultâneo, na aplicação, o utilizador é transferido da página inicial (onde estava presente o *loading spinner*) para a *View* de controlo do dispositivo móvel, indicando, portanto, uma conexão bem sucedida.

Quando o utilizador termina a aplicação móvel é procedida a uma desconexão entre o *Arduino* e a aplicação, simbolizada pelo desligar dos 2 LEDs anteriormente referidos e por uma "reversão" dos comandos efetuados até então, ou seja, o inverso do último comando fornecido pelo utilizador será o primeiro a ser executado aquando de uma desconexão, da mesma forma que o inverso do primeiro comando fornecido será o último a ser replicado. Este comportamento foi verificado executando testes em que os comandos fornecidos pelo utilizador e a ordem pela qual devem ser realizados estava previamente definida. Quando a ligação entre a aplicação e o *Arduino* é então terminada, as ações executadas pelo *hardware* são comparadas com a lista anteriormente referida, verificando a sua correta execução.

Este foi o teste mais vezes executado tendo em conta que este comportamento de conexão e desconexão é chave para a comunicação de todo o sistema e, portanto, essencial para a execução de todas as outra experiências anteriormente relatadas.

6. Conclusão

Tendo em conta que este projeto possuía uma forte componente prática que passou pela montagem e implementação de componentes de *hardware*, foi possível desenvolver competências até então não muito exploradas. Para além disso, possibilitou que o *mindset* com que as decisões de implementação foram tomadas estivesse mais direcionado para problemas que são típicos de um sistema que utiliza a interação entre vários elementos e que existe fisicamente, nomeadamente as preocupações com os problemas de conexão inerentes a uma comunicação sem fios, bem como a preocupações com os gastos de energia, visto que ambas as fontes de energia utilizadas (a do *smartphone* e as baterias externas do *Arduino*) são fontes móveis e consequentemente finitas.

Observou-se que um dos aspetos que deveria ser mais explorado caso este fosse um projeto de um produto real, desenhado e implementado para ser integrado no mercado, prende-se pela segurança do veículo. Nomeadamente, numa situação real seria importante possuir um sonar que detetasse obstáculos posicionado na parte traseira do carro e mesmo nessa situação, teriam de ser analisadas opções concorrentes para assegurar a utilização do sensor mais sensível e que assim garantisse maior fidelidade nos dados recolhidos. Nessa situação hipotética, também os dados deveriam ser analisados de forma mais cuidadosa de forma a ter em conta obstáculos que se atravessassem inadvertidamente no caminho do carro.

Considera-se que este foi um projeto desafiante que permitiu aprender conceitos e aplicações da informática que fogem um pouco do habitual. Permitiu também uma visão mais aprofundada de um ramo da tecnologia que está cada vez mais em voga, ao mesmo tempo que o sistema foi criado a partir do nada. É importante referir que o projeto também permitiu criar uma responsabilidade e consciencialização em relação aos cuidados e forma de manusear *hardware*

De notar que durante a construção deste sistema, era tido como objetivo implementar funcionalidades adicionais, mas que, por falta de tempo derivado a alguns problemas na componente de *hardware* do projeto, não chegaram a ser totalmente programadas para a versão final. Algumas destas funcionalidades seriam:

- *Feedback* estatístico da distância total percorrida pelo carro;
- *Feedback* estatístico do tempo decorrido em ambientes de baixa luminosidade, bem como da quantidade de tempo que os faróis permaneceram ligados;
- Contabilização da quantidade de obstáculos encontrados e evitados.

Como aspetos negativos do desenvolvimento do projeto, é necessário referir as limitações de material *hardware* que impossibilitaram a implementação de algumas funcionalidades, nomeadamente a demora de acesso a uma placa de *Wi-Fi*, que condiciona o desenvolvimento de uma parte fulcral do projeto que é a interação e comunicação entre os componentes principais do sistema; também ocorreram problemas com os motores utilizados, a falta de potência inerente ao modelo de motor que foi utilizado, que condiciona enormemente o movimento do carro (objetivo principal do sistema), sucederam também complicações relativamente ao micro controlador utilizado para controlar o movimento dos motores e que obrigará a uma substituição do equipamento. Todas estas nuances acabaram por atrasar bastante o progresso do desenvolvimento e gerar algumas das falhas e limitações já referidas ao longo deste documento.