Universidade do Minho
Escola de Engenharia

Ambientes Inteligentes

Relatório de Ambientes Inteligentes

Trabalho Prático de Grupo nº1

Docentes:

Paulo Novais

Discentes:

A87240, Ciarán Mcevoy

PG50788, Tomas Lima

PG50416, Hugo Silva

PG50392, Gonçalo Carvalho

Ano Letivo de 2022/2023

1 Introdução

A inteligência artificial é, atualmente, uma das áreas do conhecimento com uma evolução e crescimento mais explosivos. Por sua vez é também uma área que confere uma automatização, personalização e reatividade a diversas funções diferentes, o que, aliado ao referido anteriormente, a torna extremamente atrativa para a criação de novas formas de interagir com o ambiente que nos rodeia.

O uso destas ferramentas é já comum no nosso dia-a-dia, apesar de muitas vezes passar despercebido. Um exemplo disto são as mensagens que muitas vezes recebemos em dias de maior calor a alertar acerca do risco elevado de incêndios. Isto nada mais é do que um conjunto de sensores que detetam a informação, aliados a um programa com uma inteligência artificial (geralmente regras de se A então B) muito rudimentar, que monitoriza estes dados e age autonomamente para nos informar caso estes dados ultrapassem um determinado nível, num dado parâmetro.

Este tipo de ferramentas pode também ser associado à saúde. De facto, é comum este tipo de sensores serem utilizados por atletas, diabéticos ou doentes crónicos, não só para melhorar a sua *performance* e qualidade de vida, como para alertar caso determinadas situações limite sejam ultrapassadas.

Partindo deste princípio, foi decidido que seria interessante criar uma aplicação semelhante ao acima referido, e que nos permitisse monitorizar e aconselhar os atletas em questão. Para este efeito recorreu-se aos sensores utilizados pelo *smartwatch Forerunner 245* da *Garmin*, em combinação com um conjunto de métodos para reunir e comunicar a informação, e, por fim, enviando um aviso caso determinadas condições se verifiquem através do *Adafruit IO* e IFTTT.

Este equipamento foi escolhido pois um dos elementos do grupo, que é atleta, o utiliza, o que facilitaria o acesso a uma grande quantidade de dados recolhidos ao longo dos últimos anos.

2 Sensores e dados recolhidos

O *smartwatch* em questão contém vários sensores, como sensores de localização (GPS - Americano, GLONASS - Russo e GALILEO - Europeu), monitor de frequência

cardíaca, bússola, acelerómetro e monitor de saturação de 0_2 no sangue.(1)

A partir destes sensores, é possível obter vários tipos de dados diferentes. Para a nossa aplicação, entendemos que os dados pertinentes, obtidos diretamente da API da *Garmin*, são: frequência cardíaca, respiração, movimento durante o sono, stress, bateria corporal, saturação de 0_2 no sangue, número de passos e distância percorrida diária.

3 Garmin Access Menu

Este foi possivelmente o passo mais difícil, o de tentar perceber como se faria o acesso aos dados. A Garmin possui uma aplicação, a Garmin Connect, na qual apresenta todos os dados registados pelos dispositivos do utilizador. As primeiras tentativas passaram por descarregar toda a informação num ficheiro ZIP, mas o volume de dados era excessivamente grande para este tipo de aplicação, para não falar no estado caótico em que estavam. Em seguida tentou-se o acesso através dos *Software Development Kits* (SDK's) da Garmin, tendo em vista o acesso em direto aos sensores. Contudo, para além de alguns destes SDK's estarem apenas disponíveis para empresas parceiras da Garmin, os de acesso livre apresentavam uma estrutura demasiado complexa, onde se teria de fazer a programação da aplicação em *Monkey C* e a criação de *watch faces*, algo completamente fora do âmbito deste projeto. Ao fim de várias tentativas para tentar implementar uma aplicação em *Monkey C*, optou-se pela adaptação do código disponibilizado por *cyberjunky*, que permite o acesso à API da Garmin. (2) Foi criado um ficheiro *garmin_access_menu.py* com o objetivo de estabelecer uma ligação com a API da *Garmin*, que retira dados de um utilizador específico, e cria ficheiros JSON a partir desses dados que vão ser lidos mais tarde pelo ficheiro *json_readers.py*.

O *garmin_access_menu.py* também age como um *frontend*, no sentido em que demonstra na consola vários menus com opções que o utilizador pode escolher dependendo do que estiver à procura. Ao correr este ficheiro, a primeira coisa que aparece na consola é um input em que o utilizador tem que escolher o dia a que quer obter os dados, sendo que as opções são 0-hoje, 1-ontem, 2-anteontem, etc.

De seguida, é então feita a ligação com o *Garmin Connect*, usando a função *init_api*, que recebe como parâmetros o email e password do utilizador a que se quer conectar, e inicializa a API, neste caso sendo com a conta do Tomás Lima. Cada pedido de ligação

efetuado é guardado um *sessions.json*, que serve como forma de cache para guardar a sessão anterior.

Se os dados inseridos estiverem todos corretos, é apresentado ao utilizador um menu com 10 opções que podem ser escolhidas dependendo das métricas físicas que se quer, com duas adicionais sendo elas "Z"para terminar a sessão do *Garmin Connect*, ou "q"para sair da aplicação. Com a opção introduzida na consola, é chamada a função *switch* que recebe como parâmetros a API e a opção escolhida. A função *switch* começa por criar um JSON referente à opção escolhida pelo utilizador, e a escrever nele os dados que foi buscar ao API. Algumas opções contêm métricas físicas com valores nulos e, portanto, estes são eliminados do ficheiro JSON criado.

4 Leitores dos ficheiros JSONs

Para a leitura dos JSONs guardados anteriormente, foi criado o ficheiro *json_readers.py*. Este ficheiro implementa diversas funções diferentes que serão utilizadas para representar a informação, em forma de gráficos, que será depois acedida através da nossa interface.

4.1 Leitor da frequência cardíaca (hr_reader)

A função *hr_reader* é utilizada para obter informação acerca dos batimentos cardíacos do utilizador. Esta função usa como parâmetro uma determinada data, sendo que é assim que conseguimos escolher de que dia queremos ver os nossos dados.

Tendo esta informação em conta, os ficheiros JSON que contêm a informação relevante vão ser abertos, e vamos começar a iterar pelos valores presentes nestes.

Se o valor de batimentos cardíacos fôr não nulo (algo que pode acontecer em períodos em que o *smartwatch* não esteve ligado ou em contacto com o utilizador) vamos adicioná-lo a uma lista, assim como o momento em que esta medição ocorreu.

De forma a calcular os momentos do dia em que os batimentos cardíacos estiveram altos (definiu-se um limite de 100 para este efeito), quando um valor dos batimentos cardíacos seja igual ou superior a 100 e a variável booleana *high* tenha como valor *false*, que é o seu valor inicial, passa-se esta variável para *true*, e colocamos a variável *start_time* igual ao *timestamp* onde esse valor ocorreu.

Quando o valor dos batimentos cardíacos descer abaixo de 95, passamos a variável booleana de volta para *false* e associamos o momento a que isto aconteceu a outra variável, o *end_time*. De seguida, os valores de *start_time* e *end_time* são adicionados na forma de tuplo à lista *high_values*. A diferença entre os 100 batimentos por minuto para iniciar o período *high* e os 95 para o terminar serve para criar um *threshold*. Isto é, evita-se que em momentos em que a FC esteja próxima de 100, não estejam sempre a ser iniciados e terminados momentos de FC elevada.

Depois disto, converteu-se todos os *timestamps* no ficheiro JSON para o formato *datetime*, e procedeu-se à criação de um gráfico com os valores das horas no eixo do X, a frequência cardíaca no eixo do Y, e em que a função é a frequência cardíaca por intervalo de tempo.

Através deste gráfico, consegue-se observar toda a informação relevante dos batimentos cardíacos ao longo do dia.

Por fim, será também mostrada uma mensagem ao utilizador que contém todos os períodos nos quais os seus batimentos cardíacos ultrapassaram os 100 por minuto, assim como uma mensagem adicional.

4.2 Leitor do sono (sleep_reader)

A função *sleep_reader* cria um gráfico um pouco mais complexo do que o gráfico anterior. Cada ficheiro JSON criado quando se acede aos dados do sono através da API contém informação útil sobre os movimentos, a respiração e o stress noturnos. Optou-se pelo acesso aos dados dos movimentos e respiração por serem medidos diretamente a partir dos sensores (e não uma métrica estimada pelo *software* da Garmin, como o stress). Refira-se que, durante a noite, a medição dos valores da respiração é bastante estável e precisa, ao contrário do que acontece durante o dia por haver interferência do movimento.

Com estes dados, é criado um gráfico com duas linhas diferentes, uma com a informação referente aos movimentos ao longo da noite, e a outra com as respirações. Este gráfico irá permitir ao utilizador ter uma perceção geral das horas que dormiu, assim como da qualidade do seu sono.

De uma forma geral, quanto menor o movimento e mais constante e baixo o ritmo

das respirações, melhor o sono.

4.3 Leitor do número de passos, objetivo de passos e distância percorrida diárias dos últimos 7 dias (week_steps_reader)

A função *week_steps_reader* é outro caso que, apesar de semelhante, tem algumas variações. Primeiro, acede-se à informação dos 7 dias anteriores à data que se está a analisar. Além dos *timestamps*, teremos também a quantidade de passos diários efetuados, a distância diária percorrida (calculada pelo número de passos multiplicado pela distância da passada), assim como um objetivo diário de passos definidos pelo próprio *smartwatch* com uso dos seus algoritmos internos.

Depois de recolher esta informação, e igualmente aos casos anteriores, será criado um gráfico com estes valores. Este gráfico vai conter os valores de passos e distância percorrida por dia, assim como o objetivo diário de passos, permitindo assim ao utilizador verificar se está a cumprir os objetivos ou não.

Note-se que as linhas do número de passos e a distância percorrida por estes dependem uma da outra. Contudo, serão tanto mais diferentes quanto mais diferente a distância da passada fôr do normal 1 metro. Isto quer dizer que, durante uma corrida, onde a distância da passada é maior, a diferença entre as duas linhas será maior.

4.4 Leitor da bateria corporal (body_battery_reader)

A função *body_battery_reader* abre o JSON que contém a informação da bateria corporal nos últimos 7 dias, a contar a partir da data que se está a analisar. Recolhe 3 dados diferentes: a bateria corporal instantânea (medida com um curto intervalo de tempo), a quantidade de bateria recarregada no dia (geralmente durante a noite/sono), e a quantidade de bateria utilizada no dia.

Estes dados são recolhidos de forma indireta, com recurso, por exemplo, aos batimentos cardíacos, às horas de sono, ao nível de stress, entre outros.

Com estes valores, vão ser criados dois gráficos diferentes: um gráfico de barras, que contém a informação referente aos valores de bateria carregada e descarregada. Com recurso a esta informação, será então possível avisar o utilizador se estiver a perder mais bateria do que aquela que está a recuperar, assim como conselhos associados para

auxiliar na correção disto.

O outro gráfico contém a evolução da percentagem de *body battery* que o utilizador tem ao longo dos dias da semana.

4.5 Leitores de passos (steps_reader), respiração (resp_reader), saturação de O_2 no sangue (spo2_reader) e stress (stress_reader)

Ao contrário dos leitores anteriores, que utilizavam vários parâmetros, tinham um processamento dos dados mais complexo, ou utilizavam dados de vários dias, estes leitores apenas se limitam a criar um gráfico com o seu parâmetro no eixo dos Y, e o *timestamp* relativo no eixo dos X, referentes a apenas um dia.

Desta forma, é possível que o utilizador verifique estes dados de uma maneira fácil de interpretar.

5 Notificações

De seguida, foram criados alguns ficheiros *feedback.py* para algumas das medições, que serão responsáveis por calcular e verificar os valores de algumas variáveis medidas e informar o utilizador no caso de estas serem, de certa forma, anormais ou prejudiciais ao mesmo. Todos estes ficheiros correm a seguir ao *reader* do seu parâmetro.

Apenas os dados do batimento cardíaco e do sono é que foram enviados para o *Adafruit*, de forma a criar uma notificação no telemóvel via IFTTT, devido a haver um limite de dois *applets* que podem ser criados na sua versão gratuita.

5.1 Frequência cardíaca

A frequência cardíaca é uma das duas métricas físicas em que são enviadas notificações pelo *IFTT* e pela consola.

Para criação da notificação pela consola foi criada uma função *feedbach_hr* que recebe dois parâmetros, sendo eles a frequência cardíaca média diária, e a data do dia que se está a analisar. A função lê os ficheiros JSON dos últimos 30 dias de modo a calcular a média e o desvio padrão da frequência cardíaca em repouso. Se a rHR (*Resting Heart Rate* - FC durante o sono) for maior que a média + desvio padrão então é porque está

muito alta em relação aos últimos 30 dias. Por outro lado, se a rHR for menor que a média − desvio padrão então é porque está anormalmente baixa. Se nenhum destes for o caso então é porque a rHR esteve normal. Para ambos estes casos é enviada uma mensagem para a consola a notificar o utilizador.

Foi também testada a combinação de dados recolhidos pelo relógio e por uma fonte externa, sendo ela o *openweathermap*. Contudo não foi possível obter acesso ao histórico do *openweathermap* e portanto se a data introduzida como parâmetro na função for igual à data de hoje é que se cruza os dados. Para este caso vai ser pedido ao utilizador que introduza a cidade onde está, e com base nessa informação é efetuado o pedido ao *openweathermap*. Mais uma vez tendo o desvio padrão e a média da frequência cardíaca nos últimos 30 dias em conta, foi concluído que se a frequência cardíaca média diária for maior que 72 é porque está anormalmente alta. Foi considerado que valores de temperatura e humidade mais elevados podem levar a ritmos cardíacos mais elevados. Com isto em mente, se o ritmo cardíaco for de facto maior que 72, pode ser que a temperatura e a humidade tenham tido um impacto. Dito isto, se a temperatura for maior que 30ºC e humidade maior que 95%, é então enviada um mensagem pela consola a dizer que houve um aumento na frequência cardíaca média diária, mas que esta pode ter sido causada por estes dois fatores.

Para a notificação pelo IFTT, se a frequência cardíaca média diária subir acima de 72 é enviada uma notificação a dizer que está anormal.

5.2 Sono

O *feedback.py* do sono realiza alguns cálculos para informar o utilizador sobre a qualidade do seu sono.

A medida mais fácil para avaliar o sono é o número de horas. Para calcular este valor, consideramos os valores de movimento do sono de 2.0 para a entrada no sono e 3.0 para saída do sono, e a notificação é feita por via da consola e também no telemóvel, através do IFTTT. Desta forma, caso o utilizador durma 8 ou mais horas, então é informado que dormiu uma quantidade saudável, caso durma entre 7 a 8 horas, é informado que dormiu uma quantidade ligeiramente inferior à recomendada, e caso durma menos de 7 horas, então é informado que teve uma quantidade de sono insuficiente, e que esse

comportamento, no caso de repetitivo, pode ser prejudicial.

Para além da quantidade, também é importante avaliar a qualidade do sono. Para tal, calculou-se a média de movimento durante o sono do último mês, bem como o desvio-padrão. Desta forma, se a média do movimento durante a noite em questão estiver compreendida entre a média - desvio-padrão e a média + desvio-padrão, então o utilizador é informado de que teve uma atividade durante o sono dentro dos seus parâmetros normais, caso contrário, é informado de que teve mais ou menos atividade do que o normal.

5.3 Bateria corporal

O *feedback.py* da bateria corporal calcula as médias da bateria recarregada e gasta durante a semana, informando o utilizador dessas quantidades, mas também informa se, de uma forma geral, o saldo de bateria foi positivo, ligeiramente negativo (entre -20% a 0%) ou se foi muito negativo (inferior a -20%), mostrando também todos os dias da semana em que este foi negativo.

5.4 Passos

O *feedback.py* dos passos calcula o número de passos em intervalos de 15 minutos. Para cada intervalo vai classificar se o sujeito esteve "Muito Ativo", "Ativo", ou "Sedentário", dependendo da quantidade de passos que deu. No fim de cada dia vai contar o número total de passos dados e o tempo que o sujeito esteve em cada uma das três categorias. Classifica-se os intervalos como, "Sedentário"se der menos de 300 passos, "Ativo"se der entre 300 a 1000 passos, e por fim "Muito Ativo"se der mais de 1000 passos. Para o número total de passos, se este for menor que 10,000, envia uma mensagem a dizer que o sujeito não alcançou esta meta diária de passos. Caso tenha alcançado, é enviada uma mensagem de incentivo.

6 Melhorias

Analisando o estado final da aplicação desenvolvida, verifica-se que existem vários pontos em que podia ser melhorada.

O ponto de melhoria principal é o *frontend*, com o desenvolvimento de uma interface mais adequada e visualmente agradável que substituísse a interação por parte do utilizador com a consola/linha de comando, por esta se tratar, nos dias de hoje, uma forma muito rudimentar de interação pessoa-máquina.

Para além disso, também houve uma limitação por parte do *Adafruit*, apenas permitindo que houvesse 2 *feeds* simultâneos para fazer notificações. Porém, numa aplicação real, seria de interesse fazer uma versão móvel da aplicação, e dessa forma as notificações poderiam ser feitas diretamente a partir dessa aplicação, sem necessidade de intermediários.

Outra das melhorias possíveis é o funcionamento em tempo real da aplicação. Neste momento, os dados só são recolhidos, analisados e as notificações enviadas quando o utilizador está a utilizar a aplicação. Numa aplicação real isto seria tudo feito em segundo plano, sendo o utilizador apenas necessário para ver as notificações e a evolução dos parâmetros medidos. Claro que, numa aplicação móvel, esta melhoria seria mais facilmente implementada.

Outra melhoria seria também utilizar valores de *threshold* baseados na ciência e personalizados para o utilizador. Alguns destes valores, como os 100 bpm para determinar frequência cardíaca alta, o movimento do sono de 2.0 e 3.0 para determinar se o utilizador adormece ou acorda, ou as médias $+/-$ o desvio-padrão, são valores que utilizamos, mas sem base na ciência. Estes valores foram escolhidos pois, analisando os dados explorados, nos pareceram serem ponderados para a aplicação em questão, porém, como os dados são apenas de uma pessoa, podem não funcionar corretamente se aplicados a outros utilizadores.

Também é de referir que as análises que foram feitas, bem como as notificações, apesar de serem diversas, analisam, quase todas, apenas uma variável de cada vez. Seria interessante conseguir estabelecer uma plataforma que fizesse a análise conjunta dos vários parâmetros disponíveis. Por exemplo, poderia ser feita a análise se um aumento da FC implica sempre um aumento da frequência respiratória ou se os níveis de stress são afetados pelo sono.

Como já referido, não se procedeu à análise profunda de algumas das métricas, as calculadas indiretamente pelo *software* da Garmin, por não resultarem da medição direta de sensores e as mais instáveis (casos da respiração e stress durante o dia). Con-

tudo, talvez com ferramentas mais avançadas de *machine learning* seja possível extrair informação útil destas métricas.

Por fim, algumas das métricas disponibilizadas pela API da Garmin foram postas de parte logo à partida, como a hidratação, já que o utilizador em estudo nunca procedeu ao registo deste parâmetro no *Garmin Connect*. Outro fator sobre o qual se decidiu não atuar foi o da informação presente nas atividades desportivas registadas pelo utilizador (como corridas ou treinos de bicicleta) com os dados de GPS, já que o grau de complexidade que isto acarretaria seria demasiado grande. Contudo, como é óbvio, muita informação valiosa poderá ter sido perdida com esta decisão.

7 Conclusão

Apesar de todas as propostas de melhoria, foi possível o desenvolvimento de um ambiente inteligente que integrasse sensores físicos (conseguidos a partir do *smartwatch Garmin Forerunner 245*) e sensores virtuais (do *openweathermap*) que monitorizasse a saúde, neste caso, de um atleta, e notificasse o utilizador de acordo com os dados lidos.

De uma forma geral, foi possível realizar um *proof of concept*, que serve para provar a utilidade deste tipo de dispositivos quando combinados com a inteligência artificial.

As melhorias propostas seriam assim úteis para aumentar a eficiência do aparelho em questão, mas no entanto, é importante salientar que o uso deste tipo de dispositivos e software poderá, no futuro, permitir um tratamento personalizado para cada utilizador, podendo esta área ser vista como uma área em emergência, que poderá ter um elevado contributo para a melhoria da qualidade de vida e desempenho de atletas, mas também do cidadão comum.

Referências

[1] *Garmin Forerunner 245*. Garmin. Acedido a 4 de abril de 2023. [https://www.garmin.com/en-US/p/628939#specs].

[2] Ron (2023). *python-garminconnect*. GitHub. Acedido a 27 de março de 2023. [https://github.com/cyberjunky/python-garminconnect/blob/master/example.py]