
# MANUAL DE UTILIZADOR

**Gestão de Projetos**

**2020/2021**

# Raspberry Pi e Software

O *software* deverá iniciar assim que o Raspberry Pi iniciar. No entanto para o caso de algo correr mal, apresentaremos aqui o que terá de ser feito para voltar a colocar o sistema a funcionar.

Uma vez que o Raspberry utiliza um cartão SD para correr sistema, será disponibilizado um clone de todo o sistema para ser reinstalado. Isto é necessário uma vez que se o Raspberry for desligado à força, existe a probabilidade do cartão ficar corrompido e assim é muito mais fácil reinstalar todo o sistema de uma vez.

Para o caso de o sistema entrar em modo de manutenção, recomenda-se primeiramente reiniciar todo o sistema e verificar se o mesmo continua funcional. Caso tal não aconteça, poderá ser problema com alguma parte do *hardware*. Todas as ligações estarão apresentadas adiante.

Todo o código fonte do sistema está comentado na maior parte para o caso de se querer alterar alguma coisa. Fica aqui também o link para o *Github* com todo o código: https://github.com/GP-2020-9L/9Lives/tree/main/src.

A imagem abaixo demonstra a estrutura lógica do *software* desenvolvido. Todos os scripts foram desenvolvidos em linguagem *Python* e, como se vê abaixo, fazem as respetivas ligações entre eles. Para mais informação sobre cada uma das classes poderá verificar cada um dos ficheiros no *link* acima indicado ou na página seguinte.

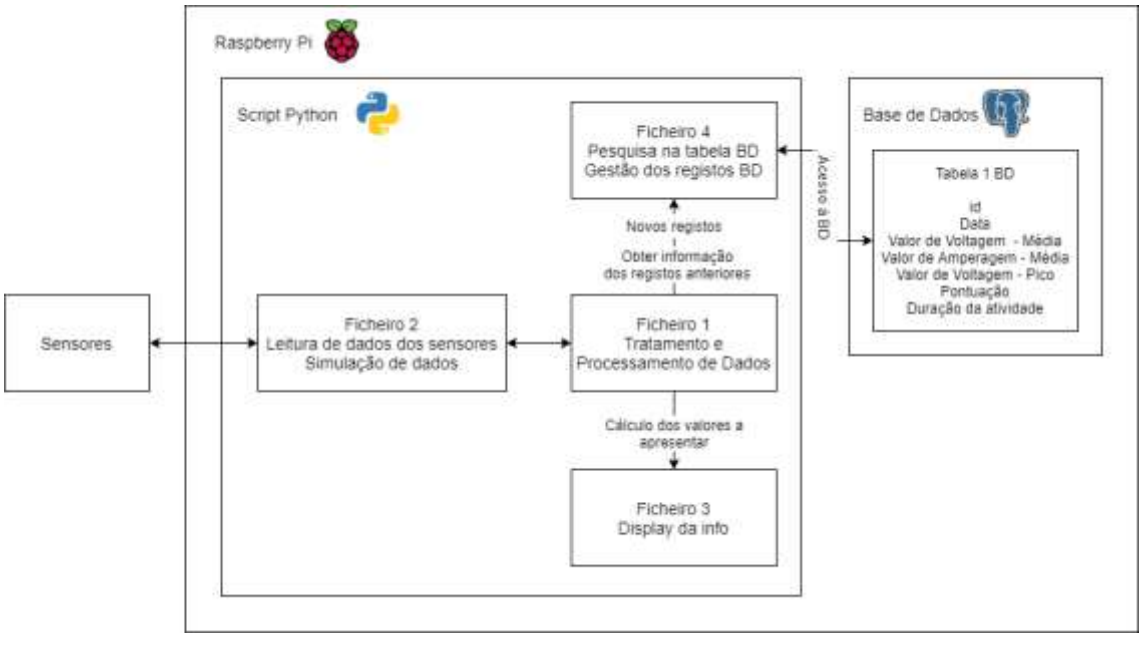



*Figura 1 - Diagrama lógico do software*

# Especificação de ficheiros

Os pontos que se seguem indicam as classes e métodos presentes em cada ficheiro do software bem como uma breve explicação do que cada um deles faz. Mais uma vez, todos os ficheiros estão presentes no link do *Github* disponibilizado acima e com o código fonte comentado.

**Ficheiro 1 – main.py:**

Métodos:

- dataThread – classe;
  - QThread.__init__() – inicia a classe numa nova *thread* e declara variáveis;
  - run() – coloca a *thread* a correr;
  - calcBatteryCharge() – calcula a percentagem de bateria possível de ser carregada a partir da energia gerada pela atividade;
  - endActivity() – envia um sinal de fim de atividade para o *display* juntamente com a informação a apresentar;
  - resetActivity() – envia um sinal para voltar ao ecrã de repouso juntamente com a informação a apresentar;

**Ficheiro 2 – dataInput.py:**

Métodos:

- dataGeneration – classe;
  - __init__() – inicia a classe e declara variáveis;
  - generateData() – gera os dados para testes de simulação;
- dataRead – classe;
  - __init__() – inicia a classe e declara variáveis;
  - readData() – recebe os valores provenientes dos sensores;

**Ficheiro 3 – display.py:**

Métodos:

- displayClasse – classe:
  - __init__() – inicia a classe e declara variáveis;
  - __startThread() – inicia a classe dataThread e coloca-a a correr
  - closeEvent() – declara o procedimento aquando o utilizador tenta fechar o programa
  - keyPressEvent() – declara várias funcionalidades ao utilizar certas teclas
  - setResultScreen() – muda o ecrã a ser mostrado para o ecrã de resultados
  - startActivity() – faz chamada aos dois métodos responsáveis por alterar o ecrã atual e atualizar a informação mostrada para o início de atividade
  - setActivityScreen() – muda o ecrã a ser mostrado para o ecrã de atividade
  - updateActivityScreen() – atualiza a informação que está a ser apresentada no ecrã de atividade
  - endActivity() – muda o ecrã para a ser mostrado para o ecrã de transição e termina a atividade internamente
  - scoreActivity() – muda o ecrã para a ser mostrado para o ecrã de resultados, enviando toda a informação necessária a apresentar e termina a atividade internamente
  - setIdleScreen() – muda o ecrã para a ser mostrado para o ecrã de repouso e atualiza a informação a ser apresentada
  - setErrorScreen() – muda o ecrã para a ser mostrado para o ecrã de manutenção e atualiza a informação a ser apresentada
- idleScreen – classe;
  - __init__() – inicia a classe e declara variáveis;
  - __setText() – declara, modifica e posiciona todas as caixas de texto a apresentar;
  - updateText() – atualiza as caixas de texto com novos dados;
  - errorScreen() – muda o ecrã de repouso para o ecrã de manutenção;
  - startTimer() – inicia o contador para mudar o ecrã de repouso;
  - stopTimer() – para o contador para mudar o ecrã de repouso;
  - __changeBackground – chamado pelo contador, muda o ecrã de repouso apresentado;

- activityScreen – classe;
    - __init__() – inicia a classe e declara variáveis;
    - __setText() – declara, modifica e posiciona todas as caixas de texto a apresentar;
    - updateText() – atualiza as caixas de texto com novos dados;
- transitionScreen – classe;
    - __init__() – inicia a classe e declara variáveis;
    - startAnimation() – inicia a animação do ecrã de transição com a duração dada;
- resultsScreen – classe;
    - __init__() – inicia a classe e declara variáveis;
    - __setText() – declara, modifica e posiciona todas as caixas de texto a apresentar;
    - updateText() – atualiza as caixas de texto com novos dados;
    - setStandard() – apresenta o ecrã de resultados no caso de a atividade não obter o melhor resultado até ao momento;
    - setUpdateBg() – apresenta o ecrã de resultados no caso de a atividade ter obtido o melhor resultado até ao momento;

**Ficheiro 4 – dbAccess.py:**

Métodos:

- dbAccess – classe;
    - __init__() – inicia a classe e declara variáveis;
    - __config() – lê o ficheiro de configuração para acesso à base de dados;
    - insertNewRide() – insere na base de dados uma nova entrada com os dados da última atividade;
    - isNewHighScore() – verifica se foi atingido um novo *highscore*;
    - getBestScore() – busca na base de dados o maior valor de *score* atingido;
    - getEnergyDay() – busca na base de dados pelo valor de energia produzido naquele dia;
    - getEnergyMonth() – busca na base de dados pelo valor de energia produzido naquele mês;
    - getSimilarScore() – busca na base de dados por valores de score semelhantes ao da última atividade.

# Montagem de hardware

As imagens abaixo representam os compentes utilizados na construção do *hardware* para este projeto. Como se pode ver cada uma das imagens apresenta o seu nome por cima seguido de uma letra. Esta letra será relevante para as tabelas de ligações, as quais serão seguidas do respetivo nome, por exemplo V S será o sensor de tensão V pino S.

Em relação aos pinos do Raspberry Pi utilizamos os números, começanco por 1, para identificar cada um deles, por exemplo o pino de 3.3 Volts será representado por R 1.

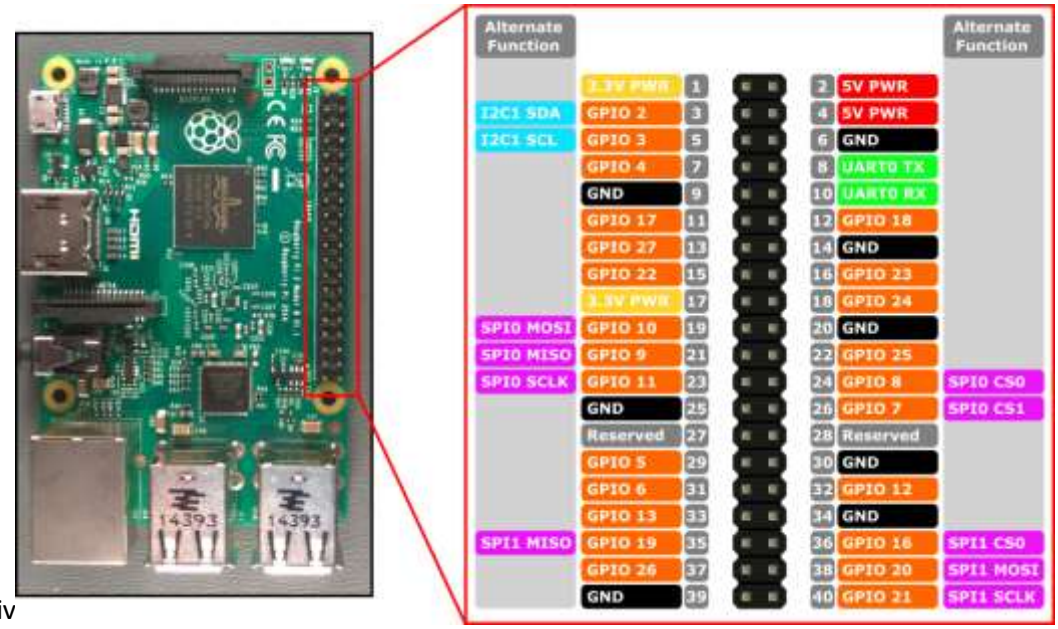
Por fim a *breadboard* não está aqui representada uma vez que a sua utilização é relativamente simples e as ligações abaixo indicadas terão todas de passar pela *breadboard*.

Sensor de tensão - V


Sensor de amperagem - A


Conversor analógico digital - C


Raspberry Pi - R



As tabelas abaixo apresentam todas as ligações necessárias para o sistema funcionar. Estão aqui representadas todas as 14 ligações existentes e a cor do cabo utilizado para mais fácil compreensão. O facto de uma célula estar vazia indica a ausência de ligação.

| | *V Negativo* | *V Positivo* | *V S* |
|---|---|---|---|
| *Breadboard negativo* | Castanho | | |
| *C A0* | | | Roxo |

| | *C VDD* | *C GND* | *C SCL* | *C SDA* | *C ADDRESS* | *C ALERT* | *C A2* | *C A3* |
|---|---|---|---|---|---|---|---|---|
| *Breadboard negativo* | | Preto | | | Amarelo | | | Preto |
| *Breadboard positivo* | Branco | | | | | | Branco | |
| *R 5* | | | Verde | | | | | |
| *R 3* | | | | Azul | | | | |

| | *A GND* | *A OUT* | *A VCC* |
|---|---|---|---|
| *Breadboard negativo* | Amarelo | | |
| *Breadboard positivo* | | | Vermelho |
| *C A1* | | Cinzento | |

| | *R 2* | *R 6* |
|---|---|---|
| *Breadboard negativo* | | Preto |
| *Breadboard positivo* | Branco | |

Para as ligações de entrada nos sensores serão utilizados os cabos positivos e negativos. O **sensor de tensão necessita obrigatoriamente** de ter o **positivo ligado à entrada VCC** (ver na entrada do sensor) e o **negativo à entrada GND**, caso contrário o sensor deixará de funcionar. No entanto isto não se aplica ao sensor de amperagem. Neste os cabos positivo e negativo podem estar ligados por qualquer ordem. Por fim esta é a disposição final do *hardware*.

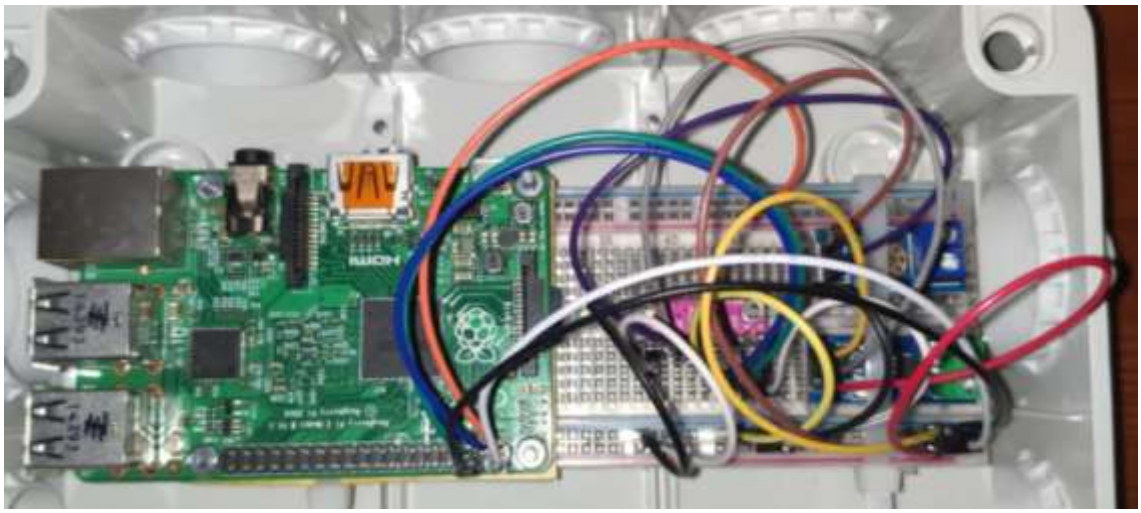


*Figura 2 - Sistema físico completo*

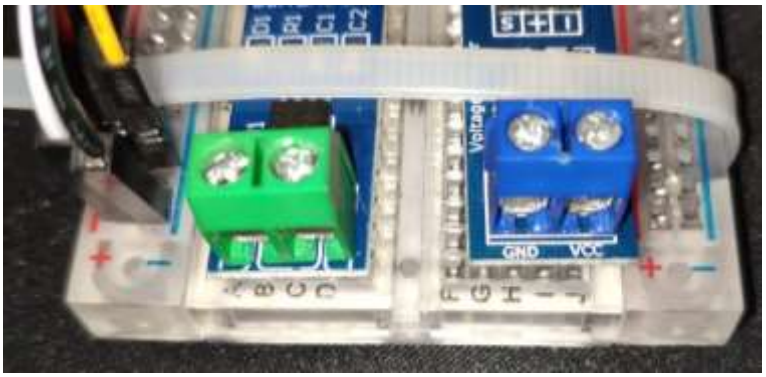



*Figura 3 - Entradas dos sensores*