

# Documentação do software do sistema embarcado - requisitos e modelagem

Modelo machine learning (TinyML) de reconhecimento de movimento usando Raspberry Pi Pico

Jonas Lendzion Schultz II

**link GitHub:**

<https://github.com/jonas-ii/jonasii-pico-motion.git>

**link demonstração no YouTube:**

<https://youtu.be/0jky7azLvT0>

Mestrando em Engenharia Elétrica

202304977

jonasschultz1309@gmail.com

## 1 Introdução

O presente relatório tem como objetivo documentar o software utilizado no sistema embarcado e no hospedeiro (PC), bem como os requisitos de modelagem do projeto final da disciplina EEL510265 - Programação de Sistemas Embarcados ministrada pelo Prof. Eduardo Augusto Bezerra.

Fazendo um breve resumo do projeto final, a ideia é implementar um modelo de machine learning embarcado (TinyML) em uma Raspberry Pi Pico para a identificação de movimentos.

Para se captar esses movimentos, utiliza-se um acelerômetro MMA845X. Diferentemente da proposição original, esse é um acelerômetro digital (e não analógico), logo, para o software embarcado na Raspberry Pi Pico, foi adicionalmente necessário implementar a comunicação via i2C aumentando um pouco mais a complexidade do projeto.

## 2 Coleta de dados - 1<sup>a</sup> versão do software embarcado (sem TinyML)

A própria empresa Raspberry Pi possui um repositório no GitHub, no qual existe um exemplo de utilização de um acelerômetro MMA8451. Ao implementar o exemplo '.c' em conjunto com uma 'CMakeLists', percebeu-se que alterações são necessárias nesses arquivos com o intuito de proporcionar o funcionamento do acelerômetro digital em específico - com a árvore de projeto inicialmente proposta. Percebe-se que a maioria do exemplo está escrito em *C*, no entanto, esta versão é somente uma preliminar e a utilizada no projeto está em *C++* como requisitado.

Finalmente, a estrutura em árvore do arquivo do projeto de coleta de dados é a seguinte:

```
.  
└── pico/  
    ├── final_project_digital/  
    │   ├── mma845X_i2c.c  
    │   ├── CMakeLists.txt  
    │   ├── pico_sdk_import.cmake  
    │   └── build/  
    └── pico-sdk/  
        ├── cmake  
        └── external  
            ...
```

Figure 1: Árvore do projeto (sem ML)

Uma vez feito o 'cross compiler', envia-se o arquivo com a extensão '.uf2' para a Raspberry Pi Pico.

O set-up utilizado é o seguinte:

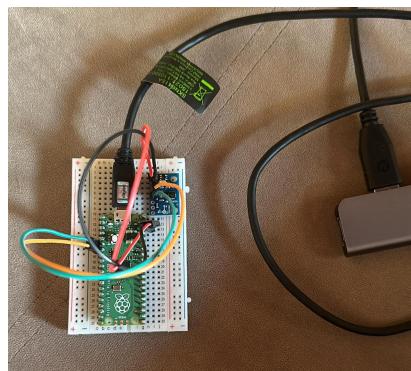


Figure 2: Set-up do projeto

Alguns movimentos foram testados utilizando recursos do Arduino (Serial Monitor e Serial Plotter) para verificar o bom funcionamento do sistema embarcado. Aqui está um exemplo ao realizar alguns movimentos aleatórios somente com o intuito de analisar se o sensor acelerômetro estava variando seus eixos (x,y,z) de acordo com os movimentos realizados:

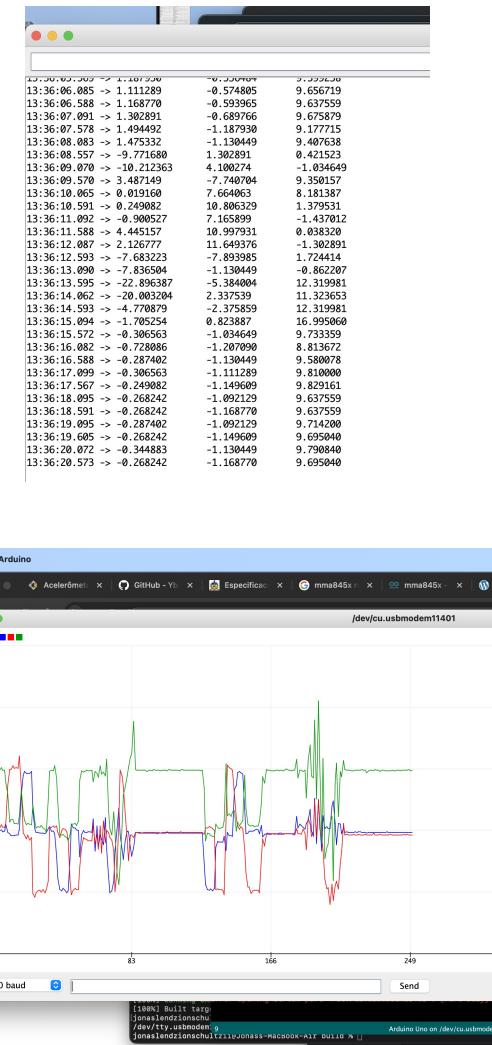


Figure 3: Monitor serial e plotter ao realizar movimentos com o set-up

Após a validação dessa etapa, parte-se para a realização do modelo de machine learning embarcado (TinyML).

### 3 Software embarcado versão final - com TinyML

Para tanto, será utilizado o programa 'Edge Impulse Studio'.

Depois que o projeto for criado e a CLI instalada, a maneira mais fácil de obter dados do Pico é usando o encaminhador Edge Impulse Data. Isso permite encaminhar dados coletados por uma interface serial para o estúdio. Este método só funciona perfeitamente em sensores com frequências de amostragem mais baixas, como no caso do projeto em questão (gestos humanos).

O encaminhador de dados é usado para retransmitir facilmente dados de qualquer dispositivo para o Edge Impulse via serial. Os dispositivos gravam os valores dos sensores por meio de uma conexão serial e o encaminhador de dados coleta os dados, assina-os e envia-os ao serviço de ingestão.

Dentro desse programa, pode-se plotar os dados (movimentos humanos realizados) para verificar se eles estão bem definidos e com isso será realizada separação do dataset em treino/validação e teste.

A imagem abaixo mostra a divisão dos clusters dentro da plataforma:



Figure 4: Divisão dos clusters - movimentos

Uma rede neural simples foi implementada para prever o movimento humano realizado. Ela possui 4 neurônios de saída - já que temos 4 classes possíveis de movimentos.

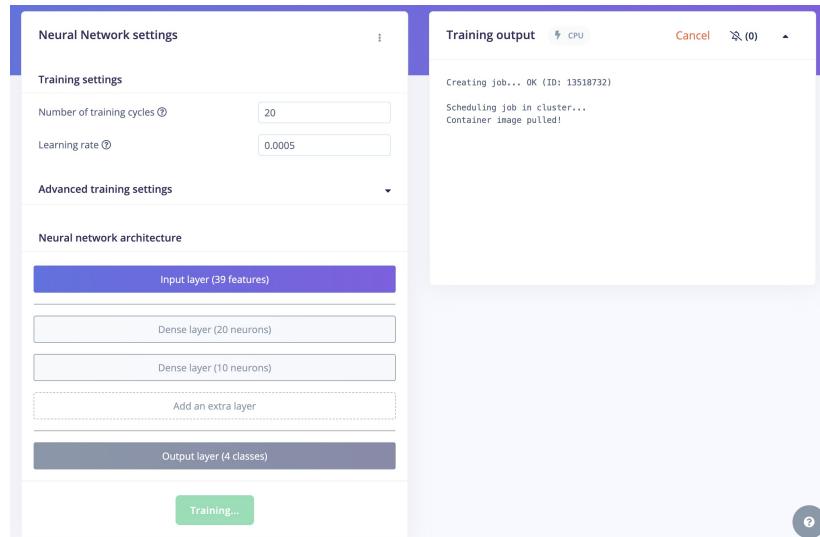


Figure 5: Rede neural implementada no treinamento do modelo

Após o seu treinamento, o seguinte desempenho foi alcançado:



Figure 6: Desempenho da rede neural com o conjunto de treinamento

E ao aplicar o conjunto de teste, o resultado obtido foi o seguinte:

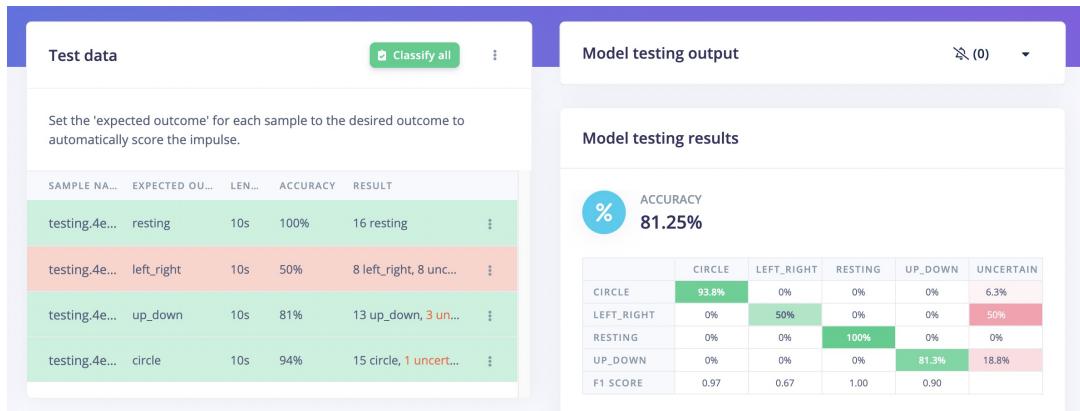


Figure 7: Desempenho com o conjunto de teste

Ressalta-se que os resultados obtidos poderiam ainda ser melhorados ao aumentar o conjunto de treinamento. No caso do projeto em questão, capturou-se 60 segundos de cada movimento para o conjunto de treinamento.

Ao final de toda a previsão e desenvolvimento do modelo, deve-se realizar a conversão e modificação dos arquivos gerados para o envio à Pico. O Edge Impulse possui uma seção de Deployment para auxiliar em que pode ser escolhida a extensão C++ para os arquivos.

O requisito principal dessas modificações é atender a proposição de utilizar uma fila de eventos (nesse caso um buffer com um ID específico para cada evento) dos gestos humanos e adicionar a função de leitura do rtc interno para a adição da data/hora do evento.

Finalmente, a estrutura final do projeto é a seguinte:

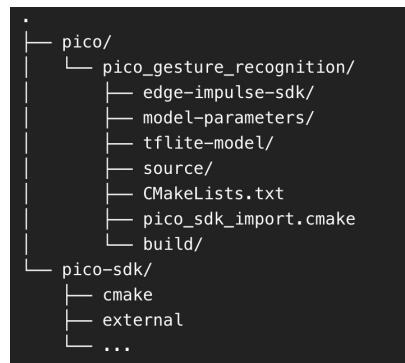
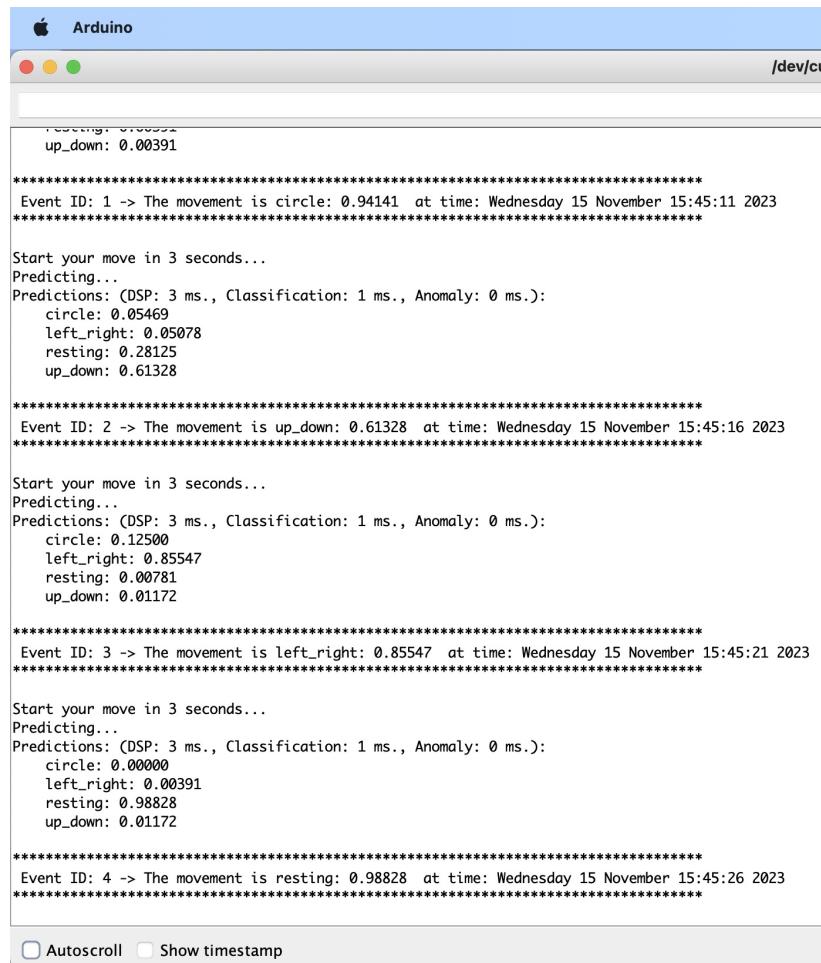


Figure 8: Árvore do projeto (com ML)

Após a criação de um arquivo 'main.cpp' dentro da pasta *sources/* e das respectivas mudanças para o envio da mensagem via USB/Serial, tem-se o seguinte resultado:



The screenshot shows the Arduino Serial Plotter window. The title bar says "Arduino". The top right corner shows the port name "/dev/cu". The main text area displays the following serial data:

```

  up_down: 0.00391
*****
Event ID: 1 -> The movement is circle: 0.94141 at time: Wednesday 15 November 15:45:11 2023
*****
Start your move in 3 seconds...
Predicting...
Predictions: (DSP: 3 ms., Classification: 1 ms., Anomaly: 0 ms.):
  circle: 0.05469
  left_right: 0.05078
  resting: 0.28125
  up_down: 0.61328
*****
Event ID: 2 -> The movement is up_down: 0.61328 at time: Wednesday 15 November 15:45:16 2023
*****
Start your move in 3 seconds...
Predicting...
Predictions: (DSP: 3 ms., Classification: 1 ms., Anomaly: 0 ms.):
  circle: 0.12500
  left_right: 0.85547
  resting: 0.00781
  up_down: 0.01172
*****
Event ID: 3 -> The movement is left_right: 0.85547 at time: Wednesday 15 November 15:45:21 2023
*****
Start your move in 3 seconds...
Predicting...
Predictions: (DSP: 3 ms., Classification: 1 ms., Anomaly: 0 ms.):
  circle: 0.00000
  left_right: 0.00391
  resting: 0.98828
  up_down: 0.01172
*****
Event ID: 4 -> The movement is resting: 0.98828 at time: Wednesday 15 November 15:45:26 2023
*****

```

At the bottom of the window, there are two checkboxes: "Autoscroll" and "Show timestamp".

Figure 9: Serial Plotter do software embarcado

O evento, inicialmente presente em um buffer do sistema embarcado, é enviado via USB/Serial para o hospedeiro (já que, conforme relatado pelo professor, não há mais a necessidade de se usar uma UART). Nessa etapa utilizou-se somente o Serial Plotter da interface Arduino, porém, para a próxima seção, essa fila (buffer) será enviada para uma outra fila no hospedeiro de acordo com as especificações propostas.

Observa-se que esses eventos possuem um ID e uma data/hora específico. Após a validação dessa etapa, parte-se para o desenvolvimento do software do hospedeiro.

## 4 Software do hospedeiro (PC)

A primeira modificação foi feita no software embarcado na Pico, pois visa-se reduzir o tamanho da mensagem já que a mesma será copiada para um 'log' em forma de fila no host (PC).

Dante disso, somente a mensagem presente entre os asteriscos (\*) da Figura 10 é enviada via USB/Serial para o host.

O host possui um sistema operacional MacOS, logo, foi utilizado o 'Xcode' em uma forma de *Xcode project* para a realização do programa em *C++*.

O programa acessa a porta USB correspondente à conexão da Raspberry Pi Pico - consequentemente também seta a *Baud Rate* necessária e o caminho até essa porta - e define as funções correspondentes para realizar a leitura dos eventos enviados pelo setup embarcado. Cabe, então, ao usuário escolher entre as diversas opções disponibilizadas o que ele deseja executar, conforme a imagem abaixo:

```
*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option:
```

Figure 10: Menu do Software do Host (PC)

Alguns casos do funcionamento desse software do host serão mostrados em seguida.

Inicialmente, mostra-se a funcionalidade de adquirir um evento (movimento) do sistema embarcado e adicioná-lo a uma fila, além disso, como requisitado, é possível ao 'admin' visualizar os eventos presentes nessa fila:

```
*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 1

The event is: Event ID: 444 -> The movement is resting: 0.97266 at time:
Wednesday 15 November 16:22:12 2023

*****  

Welcome 'admin'  

Press 1 if you want to read a movement (5 seconds)  

Press 2 if you want to check the log (queue)  

Press 3 if you want to clear the log (queue)  

Press 4 if you want to see the active time of Pico and close the serial port  

*****  

Option: 2

The event 0 in the queue of the host is:  

Event ID: 411 -> The movement is resting: 0.97266 at time:  

Wednesday 15 November 16:19:26 2023  

The event 1 in the queue of the host is:  

Event ID: 430 -> The movement is resting: 0.98047 at time:  

Wednesday 15 November 16:21:01 2023  

The event 2 in the queue of the host is:  

Event ID: 432 -> The movement is resting: 0.98438 at time:  

Wednesday 15 November 16:21:11 2023  

The event 3 in the queue of the host is:  

Event ID: 444 -> The movement is resting: 0.97266 at time:  

Wednesday 15 November 16:22:12 2023

*****  

Welcome 'admin'  

Press 1 if you want to read a movement (5 seconds)  

Press 2 if you want to check the log (queue)  

Press 3 if you want to clear the log (queue)  

Press 4 if you want to see the active time of Pico and close the serial port  

*****  

Option: 3

The queue is now empty
```

Figure 11: Caso de funcionamento do software do host para adicionar eventos na fila e visualizá-la

Em seguida, mostra-se que o usuário 'admin' também pode reiniciar a fila ao apagar todos os eventos previamente listados nela:

```
*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 2

The event 0 in the queue of the host is:
Event ID: 472 -> The movement is resting: 0.95703 at time:
Wednesday 15 November 16:24:32 2023
The event 1 in the queue of the host is:
Event ID: 475 -> The movement is resting: 0.98047 at time:
Wednesday 15 November 16:24:47 2023
The event 2 in the queue of the host is:
Event ID: 484 -> The movement is resting: 0.97656 at time:
Wednesday 15 November 16:25:32 2023

*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 3

The queue is now empty

*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 2

*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
```

Figure 12: Caso de funcionamento do software do host ao esvaziar a fila

Além disso, conforme a especificação do projeto, o usuário 'admin' também pode acessar o tempo em que a Raspberry Pi Pico esteve ativa, nesse caso, o tempo é dado em  $\mu$ s.

```
*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 1

The event is: Event ID: 502 -> The movement is resting: 0.91016 at time:
Wednesday 15 November 16:27:02 2023

*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 1

The event is: Event ID: 504 -> The movement is resting: 0.95703 at time:
Wednesday 15 November 16:27:12 2023

*****
Welcome 'admin'
Press 1 if you want to read a movement (5 seconds)
Press 2 if you want to check the log (queue)
Press 3 if you want to clear the log (queue)
Press 4 if you want to see the active time of Pico and close the serial port
*****
Option: 4

The active time of Pico is: 2537015234 (us)

Serial port closed

Program ended with exit code: 0
```

Figure 13: Caso de funcionamento do software do host ao requisitar o tempo em que o micro-controlador esteve ativo

Finalmente, após as explicações do funcionamento tanto do software embarcado e o software do host (PC), tem-se o seguinte diagrama de classes:

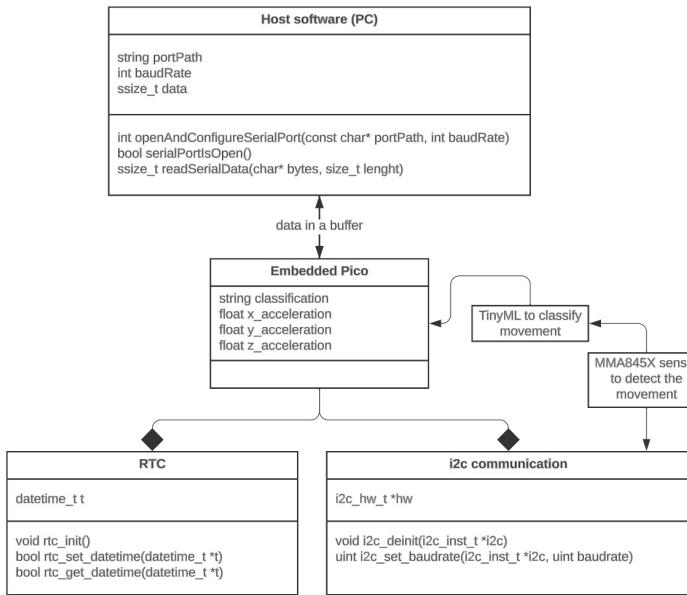


Figure 14: Diagrama de classes simplificado do projeto

## 5 Software do smartphone (versão preliminar)

Desenvolver um software para smartphone requer, inicialmente, uma breve pesquisa para analisar quais são as melhores linguagens de programação de acordo com sistema operacional do smartphone alvo.

Visto que possuo um iPhone e, como já relatado anteriormente, também possui um computador macOS, foi feita a escolha pela linguagem Swift, pois ela possui compatibilidade perfeita com o ecossistema iOS.

A linguagem Swift, criada pela Apple, oferece um ambiente altamente intuitivo e eficiente para a criação de aplicativos nativos, garantindo um desempenho otimizado. Além disso, ela possui inspirações na programação orientada a objetos - como foi requisitado pela disciplina para a realização do software na linguagem C++.

Swift, embora tenha sua própria sintaxe e abordagem, herda muitos conceitos fundamentais da orientação a objetos presentes em linguagens como C++: classes, estruturas, protocolos e métodos, permitindo a modelagem de entidades e o encapsulamento de dados e comportamentos.

Após essa breve introdução da linguagem e de apresentar as suas inúmeras semelhanças com C++ orientado à objetos, visto que esse é meu primeiro contato com um software realizado para celular, procurei realizar um simples exemplo de 'HelloWorld' como proposto na especificação da entrega 3.

Detalharei brevemente o código empregado nessa aplicação. O projeto se divide em dois arquivos principais, 'HelloWorldApp.swift' e 'ContentView.swift'.

O 'HelloWorldApp.swift' é o arquivo principal. Ele serve como ponto de entrada para a execução do código em um aplicativo. Quando o programa é executado, o sistema começa a execução a partir desse arquivo.

Com relação ao 'ContentView.swift', temos a declaração das seguintes **structs** - como em C++.

1. **ContentView**: Essa struct define a view principal (View) do aplicativo. Ela cria uma estrutura de visualização que consiste em um VStack (um recipiente vertical que organiza as visualizações nele) contendo uma imagem e um texto.
  - 'VStack': Empilha visualizações verticalmente. No código fornecido, contém uma imagem e um texto.
  - 'Image': Exibe uma imagem usando o sistema de ícones dos Symbols do iOS, nesse caso, utilizou-se o símbolo "globe".
  - 'imageScale': Define o tamanho da imagem como grande (large).
  - 'foregroundColor': Define a cor da imagem como a cor de destaque (accentColor) definida no tema do aplicativo.

- (e) '*Text*': Exibe um texto com o conteúdo "Hello world from the smartphone app!". O padding() aplicado no VStack adiciona um preenchimento ao redor do conteúdo dentro do recipiente.
2. ContentView\_Previews: Esta struct é um PreviewProvider, que oferece uma visualização de pré-visualização do ContentView durante o desenvolvimento no ambiente de desenvolvimento da Apple (o Xcode nesse caso). A função previews retorna uma instância de ContentView para mostrar uma representação visual do layout e aparência do aplicativo enquanto você o desenvolve.
- (a) '*previews*': Retorna uma instância de ContentView() para exibição na interface de pré-visualização do Xcode.

A utilização da struct logo acima é muito interessante pois permite ao programador emular o funcionamento do código (feito para um smartphone iOS) em seu computador (macOS).

Com isso, tem-se o seguinte resultado:

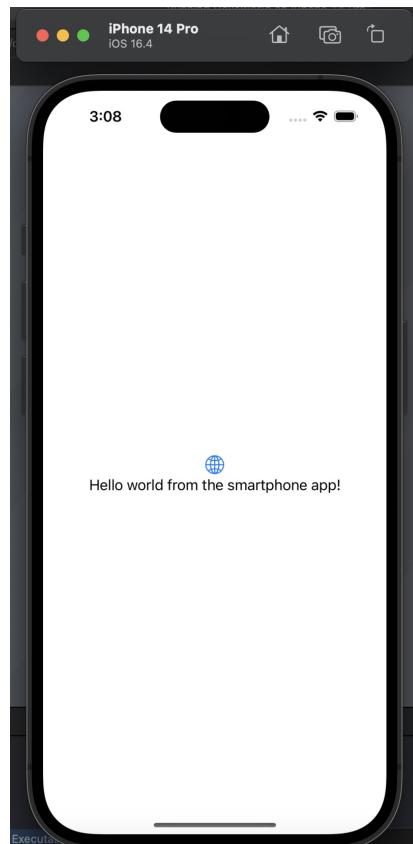


Figure 15: Software 'Hello world' do smartphone

Apesar de ser muito semelhante à linguagem *C++*, na Swift as visualizações são construídas como estruturas que estão conforme ao protocolo chamado View.

Diante disso, efetuou-se um diagrama de classes conceitual, indicando as relações entre as estruturas:

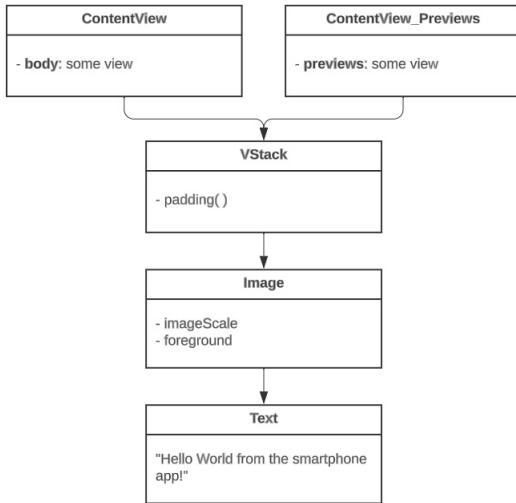


Figure 16: Diagrama das estruturas presentes no software do smartphone

Com relação ao próximos passos e plano de testes, com a finalidade de facilitar o desenvolvimento do software, será suposto que o smartphone já está recebendo a mensagem recebida entre (\*) da figura 9 (via bluetooth, WiFi, etc - à definir). Será realizada uma tentativa de programar a interface desse recebimento e, com isso, pode-se realizar a exibição da mensagem do movimento que o modelo de TinyML está prevendo e notificar o usuário.

## 6 Entrega e considerações finais

Visando o planejamento proposto na seção anterior, buscou-se, inicialmente, analisar qual o hardware necessário para o modelo proposto de projeto, visto que a *Raspberry Pi Pico* não possui um módulo de comunicação interessante para a interface com o smartphone (Bluetooth, WiFi, etc). Optou-se, então, pela futura utilização de uma *Raspberry Pi Pico W* que possui módulos Bluetooth e WiFi.

Além disso, como o projeto do aplicativo para smartphone está sendo desenvolvido na linguagem Swift, buscou-se também na literatura como acessar o módulo bluetooth de um smartphone iOS.

### 6.1 Requisitos e plano de desenvolvimento

Não possuo o hardware *Raspberry Pi Pico W*, no entanto, isso não impede a continuação do desenvolvimento da forma que serão apresentados todos os códigos que seriam utilizados para alcançar o resultado final esperado.

Após a validação dessa etapa e visando também a implementação da comunicação via UART inicialmente presente nos requisitos, o seguinte plano de desenvolvimento foi pensado:

- Adicionar uma comunicação UART (no presente código em C++) que enviará uma mensagem contendo o movimento realizado pelo usuário para a *Raspberry Pico W*
- Visto que a *Pico W* será utilizada somente como intermediária na comunicação e o seu desenvolvimento não é o alvo do presente projeto, ela seria programada em *MicroPython* visando maior comodidade (com o código chamado '`bluetooth(picow.py)`'.
- Desenvolvimento do aplicativo para smartphone iOS visando os seguintes pontos:
  - Autorizar o acesso ao módulo Bluetooth
  - Ativar Bluetooth
  - Buscar dispositivos presentes na redondeza
  - Conexão à um dispositivo
  - Função que execute a leitura de dados recebidos via Bluetooth
  - A cada evento recebido, o aplicativo deverá notificar o usuário

Diante do exposto, parte-se então para a execução.

## 6.2 Desenvolvimento

### 6.2.1 Comunicação via UART

Inicialmente, a comunicação via UART deverá ser declarada da seguinte forma no código 'main.cpp':

```
31 #define UART_ID uart1
32 #define BAUD_RATE 115200
33 #define DATA_BITS 8
34 #define STOP_BITS 1
35 #define PARITY    UART_PARITY_NONE
36
37 // Using pins 8 and 9
38 #define UART_TX_PIN 8
39 #define UART_RX_PIN 9
```

Figure 17: Constantes da comunicação via UART

```
63
64 int main() {
65     stdio_init_all();
66
67     // Set up our UART with a basic baud rate.
68     uart_init(UART_ID, 2400);
69
70     // Set the TX and RX pins by using the function select on the GPIO
71     // Set datasheet for more information on function select
72     gpio_set_function(UART_TX_PIN, GPIO_FUNC_UART);
73     gpio_set_function(UART_RX_PIN, GPIO_FUNC_UART);
74
75     // Actually, we want a different speed
76     // The call will return the actual baud rate selected, which will be as close as
77     // possible to that requested
78     int actual = uart_set_baudrate(UART_ID, BAUD_RATE);
79
80     // Set UART flow control CTS/RTS, we don't want these, so turn them off
81     uart_set_hw_flow(UART_ID, false, false);
82
83     // Set our data format
84     uart_set_format(UART_ID, DATA_BITS, STOP_BITS, PARITY);
85
86     // Turn off FIFO's - we want to do this character by character
87     uart_set_fifo_enabled(UART_ID, false);
88
```

Figure 18: Implementação da comunicação UART

```
212     int len1 = snprintf(NULL, 0, "%f", result.classification[jx].value);
213     int len2 = snprintf(NULL, 0, "%d", id);
214     char *value_char = (char *)malloc(len1 + 1);
215     char *id_char = (char *)malloc(len2 + 1);
216     snprintf(value_char, len1 + 1, "%f", result.classification[jx].value);
217     snprintf(id_char, len2 + 1, "%d", id);
218     //char *value_char;
219
220     if (result.classification[jx].label=="circle"){
221         if (uart_is_writable(UART_ID)) {
222             uart_puts(UART_ID, "ID: ");
223             uart_puts(UART_ID, id_char);
224             uart_puts(UART_ID, " -- > Circle with probability of: ");
225             uart_puts(UART_ID, value_char);
226         }
227     } else if (result.classification[jx].label=="left_right") {
228         uart_puts(UART_ID, "ID: ");
229         uart_puts(UART_ID, id_char);
230         uart_puts(UART_ID, " -- > Left-right with probability of: ");
231         uart_puts(UART_ID, value_char);
232     } else if (result.classification[jx].label=="resting") {
233         uart_puts(UART_ID, "ID: ");
234         uart_puts(UART_ID, id_char);
235         uart_puts(UART_ID, " -- > Resting with probability of: ");
236         uart_puts(UART_ID, value_char);
237     }
238     else {
239         uart_puts(UART_ID, "ID: ");
240         uart_puts(UART_ID, id_char);
241         uart_puts(UART_ID, " -- > Up_down with probability of: ");
242         uart_puts(UART_ID, value_char);
243     }
244
245     free(value_char);
246     free(id_char);
247 }
```

Figure 19: Envio de mensagens via UART

### 6.2.2 Placa Raspberry Pi Pico W - Micropython

Como já relatado anteriormente, o alvo do projeto não é o desenvolvimento desse programa e por conta disso optou-se por não descrevê-lo em detalhe.

O arquivo chamado 'bluetooth\_picow.py' executaria tanto o recebimento da comunicação UART quanto o envio dessas informações via Bluetooth (presente no GitHub disponibilizado).

### 6.2.3 Aplicativo para Smartphone

O aplicativo para smartphone iOS, como relatado na sessão 5, será realizado na linguagem Swift - que possui inspirações na programação orientada à objetos.

O projeto é baseado no seguinte repositório <https://github.com/adafruit/Basic-Chat>. Para atender aos requisitos propostos, foram realizadas alterações no programa para realizar a implementação de um aplicativo que acessa o módulo Bluetooth do smartphone, realiza somente a leitura de informações recebidas e notifica o usuário. O projeto chama-se 'final\_app\_bluetooth'.

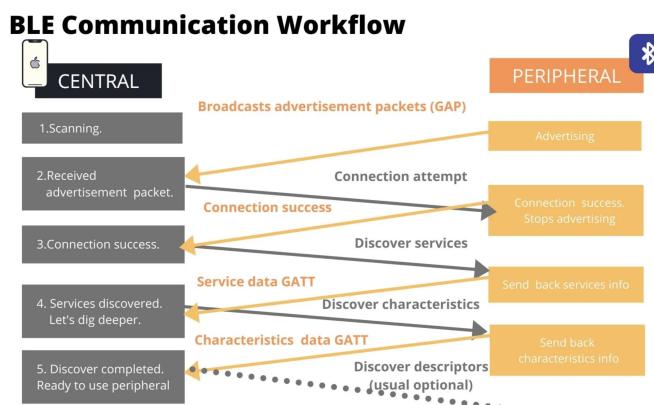


Figure 20: Workflow da comunicação BLE

Vale ressaltar, como já descrito anteriormente, que não foi possível realizar o teste do programa, uma vez que não possuo a *Raspberry Pi Pico W*.

O código (orientado à objetos) fornece as classes necessárias para que o aplicativo se comunique com tecnologia sem fio de baixa energia (LE) e taxa básica/taxa de dados aprimorada (BR/EDR) equipada com Bluetooth.

Brevemente, o programa é baseado nas seguintes classes:

- CBCentralManager e CBCentralManagerDelegado: São responsáveis por verificar se o Bluetooth está ligado e, em seguida, verificar, descobrir, conectar e gerenciar periféricos.
- CBPeripheral e CBPeripheralDelegado: Representa dispositivos BLE físicos conforme foram descobertos pelo CBCentralManager. Eles são identificados por UUID (identificador universalmente exclusivo) que contém um ou mais serviços.

Um serviço encapsula a forma como parte do dispositivo se comporta: no caso do projeto em questão, um serviço representará um movimento realizado pelo usuário. Esse serviço pode, então, conter diversas características, logo:

- CBServiço: Representa o dispositivo BLE físico de serviço e fornece comportamentos e características associados aos dados determinados no dispositivo BLE.
- Características do CB: Representa os dados de serviço do dispositivo e contém um único valor. É aqui que podemos ler, escrever e assinar os dados do dispositivo (ex: nível da bateria, temperatura, luz LED).

Relação entre objetos 'CBPeripheral', 'CBService' e 'CBCharacteristic':

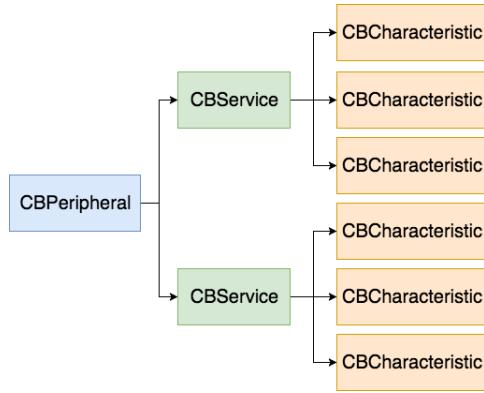


Figure 21: Diagrama simplificado das classes

Novamente, emulando o código no 'XCode project', observa-se a seguinte tela inicial:

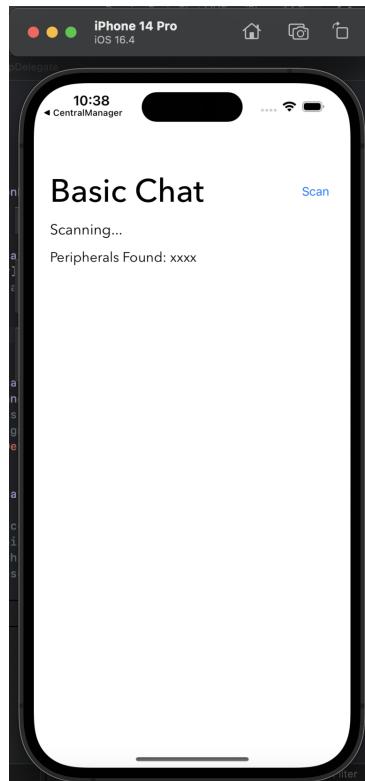


Figure 22: Tela inicial do aplicativo

## 7 Conclusão

Inicialmente, gostaria de relatar que o desenvolvimento do projeto foi um grande desafio, pois é a primeira vez que sou confrontado com a linguagem orientada à objetos.

A ideia de utilizar TinyML foi muito interessante, pois foi possível aplicar conhecimentos de outras disciplinas na realização do projeto. Aproveito aqui para ressaltar questões de desempenho: ao habilitar o 'Edge Optimized Neural Compiler (EON™)', permite-se executar redes neurais com 25-55% menos RAM e até 35% menos flash, mantendo a mesma precisão, em comparação com o TensorFlow Lite para microcontroladores, como pode-se observar abaixo:

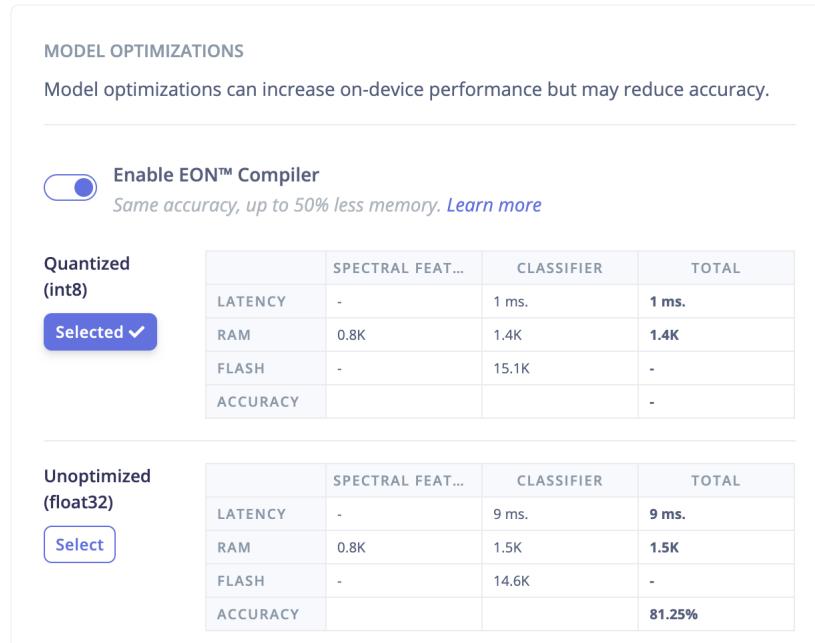


Figure 23: Utilização do compilador otimizado

Como já relatado na introdução, programar a Pico em C++ também foi um desafio, visto que normalmente se programa esse microcontrolador com MicroPython.

O software do hospedeiro poderia ser melhorado para atender à todos os requisitos, principalmente com relação à listagem de eventos ao ser inserido uma data/hora de intervalo, no entanto, a fila atendeu à especificação.

Considero também muito interessante a especificação de realizar um aplicativo para smartphone, e a utilização da linguagem Swift (que se assemelha à C++) deixou a tarefa ainda mais complexa.

A comunicação via UART me demandou um tempo maior que o previsto, pois estava utilizando a UART0 da placa e essa não pode funcionar para a comunicação (pois possui a função REPL).

Por fim, o fato de não possuir uma Raspberry Pi Pico W impossibilitou inúmeros testes finais, principalmente com relação ao desenvolvimento do aplicativo final. É uma pena não conseguir testar propriamente a versão final.

Como pôde-se observar, aparentemente o aplicativo funciona de forma correta e, caso existisse uma Pico W devidamente configurada, a mesma estaria presente na lista ao scanear os dispositivos disponíveis.

Poderia ser implementado, entre as leituras de movimento, a utilização dos estados de SLEEP e DORMANT, presentes na *Raspberry Pi Pico* para melhorar o consumo de energia e tornar a sua utilização mais viável para um ambiente embarcado.