Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Humberto de Brito Rangel Neto

Uso de Aprendizado de Máquina Visando Maior Eficiência Energética de Aplicações com Microcontroladores Embarcados

Campina Grande
2019

Humberto de Brito Rangel Neto

Uso de Aprendizado de Máquina Visando Maior Eficiência Energética de Aplicações com Microcontroladores Embarcados

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Título de Engenheiro Eletricista.
Orientador: Prof. Dr. Danilo Freire de Souza Santos

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Campina Grande
2019

Humberto de Brito Rangel Neto

Uso de Aprendizado de Máquina Visando Maior Eficiência Energética de Aplicações com Microcontroladores Embarcados

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Campina Grande como requisito parcial para a obtenção do título de Engenheiro Eletricista.

Comissão Examinadora

Prof. Dr. Danilo Freire de Souza Santos
Universidade Federal de Campina Grande
Orientador

Prof. Dr. Gutemberg Gonçalves dos Santos Júnior
Universidade Federal de Campina Grande

Campina Grande, 10 de julho de 2019

Dedico este trabalho primeiramente a Deus, que por Sua graça salva pecadores, e aos pais que Ele me deu, que desde muito cedo se esforçam em me pavimentar os caminhos da vida.

Agradecimentos

Junto a este trabalho aproxima-se a conclusão de um longo ciclo. Nada mais adequado que agradecer a todos os que me permitiram realizá-lo.

Primeiramente agradeço a Deus pelo dom da vida e por ser Ele quem dá toda a boa dádiva e todo o dom perfeito para a expressão do seu poder. A Ele seja a glória.

Agradeço aos meus pais, Emanuel e Verônica. Foram eles que ao longo desses anos se encarregaram dos incontáveis sacrifícios, que me ensinaram a valorizar as oportunidades, e que continuam me instruindo para a vida até hoje, como se a fadiga não lhes ocorresse. Agradeço também e em especial aos meus irmãos, Emanuella e Victor, e à Ellen, pelo suporte nos últimos meses e pela graça de tê-la como namorada.

Agradeço sinceramente aos professores e funcionários da Universidade Federal de Campina Grande (UFCG) que me instruíram nos últimos anos. Em especial, agradeço aos professores cuja empatia para com os alunos e zelo pelo conhecimento andam lado a lado: Damásio Júnior, Danilo Santos, Ana Paula Basso, Marcos Morais, Gutemberg Júnior, Edmar Gurjão, Mário Araújo, Carlos de Araújo, Roberto Siqueira e Luís Reyes. Meu agradecimento também aos mais que queridos Tchai, Adail e Lúcia.

Registro aqui também minha gratidão aos colegas e amigos que dividiram fardos e partilharam alegrias ao longo dos últimos anos, mais especialmente Flávio Moreira, Hugo Gayoso, Matheus Leor, Michael Douglas, Niago Nobre em Campina Grande; e em João Pessoa Diogo Barbosa, Mateus Cavalcanti e Rennan Dias.

"Engineering is the art of modelling materials we do not wholly understand, into shapes we cannot precisely analyse so as to withstand forces we cannot properly assess, in such a way that the public has no reason to suspect the extent of our ignorance."

Dr. Archie Reece Dykes

Resumo

Sistemas embarcados podem ser encontrados nas mais variadas aplicações e em vasta quantidade, de aparelhos celulares a satélites, de brinquedos a controle de usinas. Por serem fisicamente embutidos em outros sistemas maiores, são chamados embarcados. Pela mesma razão apresentam requisitos próprios quanto ao tamanho, peso e, consequentemente, limitações claras quanto à capacidade de processamento e memória, custo final e consumo de energia.

Nas últimas décadas, indústria e academia têm somado esforços para o desenvolvimento de métodos que aumentem a eficiência energética dos sistemas embarcados, em todas as camadas, de hardware a software, tendo em vista os impactos positivos em custo, autonomia, usabilidade, confiabilidade etc.

O objetivo do projeto descrito neste relatório foi de aplicar técnicas de aprendizado de máquinas no processamento de dados de sistemas embarcados, visando o desenvolvimento de aplicações de baixo consumo a partir da redução quantitativa dos dados transmitidos.

Este trabalho apresenta um estudo de caso como guia do desenvolvimento do trabalho. São também apresentados um resumo das tecnologias usadas e as atividades feitas nos últimos meses, sendo estas: a obtenção, limpeza e criação de banco de dados; a definição, implementação e treino de rede neural em CPU; a conversão do modelo de rede neural de arquitetura de CPU para MCU; e a proposta e implementação de rotina adaptada ao baixo consumo em um MCU comercial, o NXP QN9080.

Palavras-Chave: 1. Sistemas Embarcados. 2. Aprendizado de Máquina. 3.Baixo Consumo Energético. 4.Processamento embarcado.

Abstract

Embedded systems can be easily found in many domains of technology, from mobile phones to satellites, from toys to industrial plants. As the name suggests, these systems are embedded in more complex ones. And for this reason, many concerns arise from their strict requirements of small size and low weight. When developing embedded systems, these requirements are translated into constraints on memory, processing performance, cost, and low energy consumption.

In the last decades, industry has joined forces with academic research for developing new methods to increase energy efficiency of embedded systems, from hardware to software, and to improve usability, cost, autonomy and reliability altogether.

The goal this project was to reduce the amount of transmitted data by applying machine learning techniques for processing data on the edge. This strategy should increase the autonomy of low power wireless applications.

This work presents a study case as a guide. A review of the technologies and of the developed activities in the last months is presented. Among other topics, the process of gathering, preprocessing and creating a database; defining, implementing and training a neural network on CPU; the conversion from the CPU model to a model adapted to MCU architecture; and the proposal and implementation of a solution adapted to low power in a commercial MCU (NXP QN9080)

Keywords: 1. Embedded Systems. 2. Machine Learning. 3. Low Power. 4.Edge computing.

Lista de figuras

Lista de Siglas e Abreviaturas

ADC	*Conversor Analóogico Digital*
BLE	*Bluetooth Low Energy*
CI	*Circuito Integrado*
CNN	*Convolutional Neural Network*
CMSIS	*Cortex Microcontroller Software Interface Standard*
CPU	*Central Processing Unit*
DAC	*Conversor Digital Analóogico*
DC	*Corrente Direta*
DK	*Development Kit*
FFT	*Transformada Rápida de Fourier*
GPIO	*Porta genérica de entrada e saída*
I2C	*Inter-Integrated Circuit*
MCU	*Microcontroller Unit*
NN	*Neural Network*
PWM	*Modulação por Largura de Pulso*
RAM	*Random Access Memory*
RF	*Radiofrequência*
SDK	*Software Development Kit*
UFCG	*Universidade Federal de Campina Grande*
USART	*Universal Synchronous and Assynchronous Receiver Transmitter*
USB	*Universal Serial Bus*

Sumário

1 Introdução

Um sistema embarcado é uma combinação de hardware e software desenvolvida para executar uma ou algumas tarefas específicas. Diferentemente de computadores, que são de uso genérico e baseados em microprocessadores, os sistemas embarcados possuem função preconcebida e são, normalmente, baseados em microcontroladores. Por serem fisicamente embutidos em outros sistemas maiores, são chamados embarcados. Pela mesma razão apresentam requisitos próprios quanto ao tamanho, peso e, consequentemente, limitações claras quanto à capacidade de processamento e memória, quanto ao custo final e, de maneira mais importante, quanto ao consumo de energia por utilizarem baterias pequenas.

Sistemas embarcados podem ser encontrados nas mais variadas aplicações e em vasta quantidade: em aparelhos celulares, eletro-eletrônicos, máquinas industriais, automóveis, câmeras, dispositivos de automação residencial, aviões, brinquedos, dispositivos médicos etc. Em 2011, foi estimado que um automóvel bem equipado utilizava pelo menos 50 unidades de microcontroladores, dentre eles o sistema de air-bag, de freios ABS, sensores de estacionamento, câmeras, rádio etc. Oito anos depois, a VolksWagen anunciou sua estimativa de que 90% da inovação da indústria automotiva virá por meio do software embarcado nas unidades de microcontroladores, i.e., dos sistemas embarcados[5] [6].

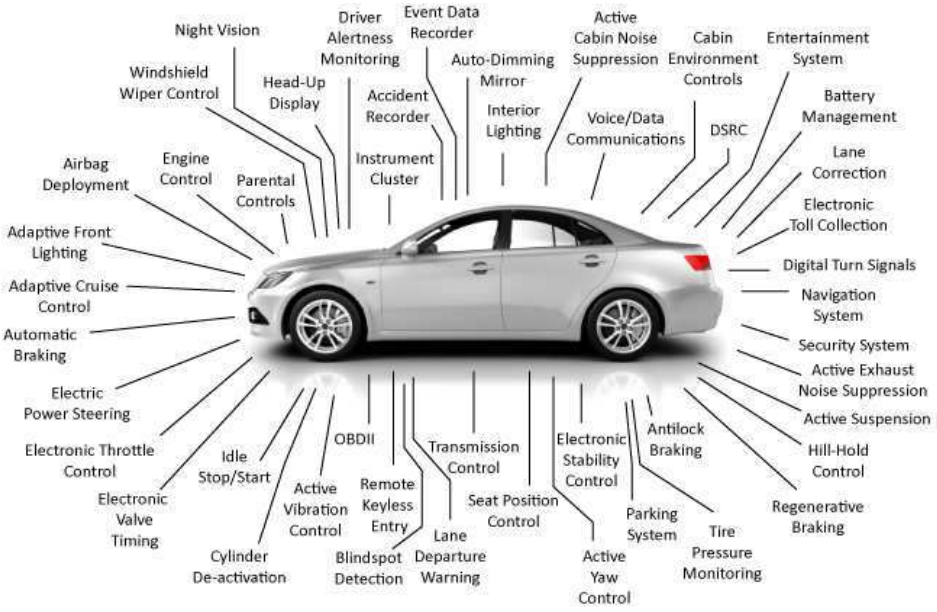

Figura 1 – Indicação de sistemas embarcados em um automóvel [1].

Grande parte dos sistemas embarcados desempenha a função específica de sensoriamento, i.e., a obtenção de informações sobre o sistema ou ambiente no qual se encontram. É o caso de um monitor de cardíaco que captura dados da atividade elétrica do coração através das variações de tensão percebidas na pele do usuário, eletrocardiograma, ou tam-

bém de sensores de estacionamento que obtém dados de distância de um carro, ou ainda de um sensor de vibração de um motor industrial[7][3]. Nessas arquiteturas as limitações energéticas são ainda mais manifestas, visto que o sistema embarcado é responsável pela captura, geração e comunicação, geralmente por radiofrequência, desses dados a um nó central. Após a recepção dos dados através da rede de comunicação, o nó central do sistema de monitoramento utiliza algoritmos lógicos ou de aprendizado de máquina para processá-los e extrair deles a informação que interessa ao usuário. Tal arquitetura é ilustrada no diagrama da Figura 2, onde em uma ponta encontra-se o sistema embarcado equipado de sensores para monitorar o ambiente, gerar dados. Esse nó periférico do sistema, limitado em memória, energia, processamento e custo, conecta-se à uma infraestrutura de comunicação sem fios e transfere os dados ao nó central. O nó central, sendo fixo, não possui as mesmas limitações do nó embarcado, por isso é geralmente o nó responsável por armazenar e processar os dados. O mesmo nó central poderia também recolher os dados de uma malha maior de sensores, a depender da aplicação, o que permitiria obter informações de maior valor para o usuário sobre o sistema monitorado[7].

Figura 2 – Arquitetura simplificada de um sistema de monitoramento com dois nós periféricos.

De todas as limitações, vale salientar o impacto que a limitação energética causa na desempenho, na autonomia, na manutenção, na experiência do usuário e, por consequência, na viabilidade prática e econômica das aplicações de sistemas embarcados [8]. Para que um sistema embarcado cumpra seu objetivo, é necessário que tenha pouca ou nenhuma necessidade de manutenção, troca de componentes, em especial de troca de bateria. A manutenção constante dos nós periféricos, por exemplo, aumentaria os custos da aplicação, podendo torná-la inviável se considerada em larga escala. Aplicações atuais como dispositivos médicos (marca-passo, bomba de infusão, aparelho auditivo etc), monitoramento de plantações (malha de sensores de umidade, temperatura, condições do solo), monitoramento industrial (pressão de oleodutos, velocidade de turbinas, temperatura de fornos)

têm sido objeto de estudo durantes as últimas décadas, na indústria e na academia, para o desenvolvimento de técnicas de redução do consumo de energia dos sistemas embarcados. Como resultado, os sistemas têm sido melhorados em todas suas camadas: de transdutores eficientes [9] a redes de comunicação de baixa potência [10],em sistemas mais eficientes da segurança da informação [11] a sistemas operacionais de baixo consumo [12] etc. Nos últimos anos, os sistemas embarcado tem se tornado mais populares e de menor custo, e somado a isso novas capacidades de processamento em tempo real tem sido adicionadas a controladores de baixo custo. Tais condições tem propiciado aos microcontroladores executar algoritmos antes destinados apenas aos sistemas fixos. Tais circunstâncias têm permitido também a exploração de mais uma face no desenvolvimento de sistemas embarcados de baixo consumo: a utilização de algoritmos mais complexos no nó periférico a fim de reduzir a quantidade de dados enviada para o nó central. Tendo também o conhecimento de que a unidade de audiofrequência de um sistema embarcado concentra grande parte do consumo de energia, espera-se que a redução do número de pacotes e a redução da frequência de envios traga vantagens em termos de eficiência energética.

1.1 Objetivos

O objetivo deste trabalho foi de identificar e aplicar técnicas atuais de processamento embarcado, utilizando aprendizado de máquina, para redução do envio de dados e verificar, ainda que preliminarmente, os impactos no consumo de energia de um nó periférico.

Como um caso de estudo, foi considerado uma aplicação simulada de uma turbina industrial. Utilizou-se um acelerômetro para medir as variações de aceleração de um pequeno ventilador e o processamento para identificar o seu estado de funcionamento em: desligado, ligado, obstruído e desbalanceado.

O trabalho está organizado na seguinte sequência: Revisão bibliográfica, com apresentação dos detalhes sobre tecnologias utilizadas no trabalho; Proposta de Solução, com apresentação do projeto realizado, nível de arquitetura do sistema; Desenvolvimento e Validação, com detalhes da implementação do trabalho e validação; e Conclusões, com um resumo do que foi realizado e os resultados obtidos e pontos que podem guiar trabalhos futuros.

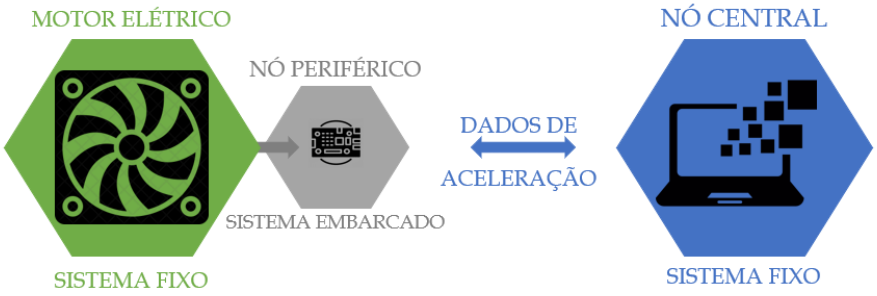

Figura 3 – Arquitetura simplificada de um sistema de monitoramento simples.

2 Revisão Bibliográfica e Tecnológica

Neste capítulo são apresentados os principais conceitos e tecnologias utilizadas neste trabalho.

2.1 Estudo de Caso: Monitorando o Motor DC

O caso de estudo brevemente descrito na seção anterior, monitoramento de estado de motor DC, será também usado aqui como ponto inicial para apresentação dos tópicos mais importantes no desenvolvimento deste trabalho. Abaixo encontra-se um diagrama mais detalhado da sua arquitetura, comum em ambientes industriais.

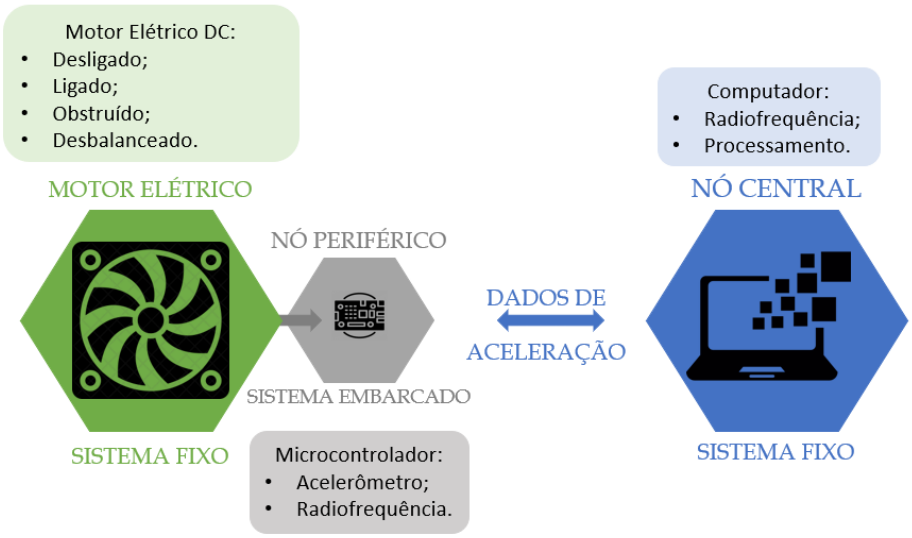

Figura 4 – Arquitetura do sistema simples de monitoramento com detalhes.

O sistema de monitoramento mostrado acima fundamenta-se em dois nó: o nó periférico e o nó central.

O nó periférico, ou embarcado, encontra-se instalado, ou mesmo embutido, no motor. Ele baseia-se em um microcontrolador, i.e., um microprocessador equipado de memórias e periféricos e, se comparado com um computador, apresenta claras limitações de capacidade de processamento, consumo de energia, assim também como evidentes vantagens, tamanho e peso reduzido. O microcontrolador utilizado para o monitoramento do motor é também equipado de um acelerômetro, com o qual medirá as perturbações físicas geradas pelo motor no ambiente, e com um módulo de radiofrequência, com o qual acessará a infraestrutura de uma rede de comunicação sem fios para envio dos dados brutos obtidos.

O nó central, i.e. o computador, baseado em um microprocessador, também utilizará um módulo de radiofrequência e acessará a infraestrutura de comunicação para receber os dados brutos referentes às perturbações do motor. Além disso, por não ter as mesmas limitações de um microcontrolador, o computador será inicialmente o responsável por processar os dados brutos de aceleração recebidos. Diversas técnicas e algoritmos de processamento estão disponíveis na literatura, uma delas, porém, é o apredizado de máquina por redes neurais. As redes neurais, nesse estudo de caso, utilizam amostras identificadas do sinal de aceleração de cada um dos estados (desligado, ligado, obstruído e desbalanceado) para construir um modelo matemático capaz de inferir o estado de uma amostra não identificada como o menor erro possível.

Depois de definido tal modelo, ele seria utilizado como uma função no computador todas as vezes que uma nova amostra fosse enviada pelo nó periférico, embutido no motor, e a informação sobre a planta seria supervisionada pelo usuário. A seguir cada um pontos relevantes a este estudo de caso é prefaciado.

2.2 Nó Periférico: Sistema Embarcado

2.2.1 Microcontrolador: NXP QN9080

Um microcontrolador, MCU ou Unidade de Microcontrolador, trata-se de um circuito integrado (CI) geralmente usado para uma aplicação específica e desenvolvido para implementar um conjunto determinado de tarefas.

De maneira sintética, um microcontrolador coleta sinais de entrada, processa a informação do sinal e produz uma determinada ação como saída, baseado na informação obtida inicialmente. Microcontrontroladores normalmente operam em baixas frequências de clock, de 1MHz a 200 MHz, e são desenvolvidos para baixo consumo de energia.

Um microcontrolador pode ser visto como um pequeno computador, pois possui uma Unidade Central de Processamento (CPU), memória de Random-Access (RAM) e Flash, interface de barramento serial, portas de entrada e saída etc [13]. No caso de estudo inicial, o microcontrolador seria responsável por estabelecer a comunicação com um acelerômetro e requerer os dados. Após serem recebidos, organizá-los em pacotes e formatos adequados à transmissão por radiofrequência, de acordo com o protocolo utilizado, e comunicar-se com o módulo de radiofrequência para configurar e realizar a transmissão dos dados.

Um dos MCU de baixo consumo de energia disponíveis no mercado atual é o NXP QN9080. Ele possui um processador ARM Cortex-M4F de 32-bits, e vários periféricos, como transceptor RF (Bluetooth Low Energy), memórias (ROM, Flash de 512KB, RAM de 128 KB e porta quad SPI), interfaces seriais (USART, I2C, SPI, USB 2.0), até 35 GPIO, 8 ADCs de 16-bit e DACs de 8-bits, timers, PWM, decodificadores de quadratura, unidades de gerenciamento de clock (osciladores internos e externos) e de potência,

sensor de temperatura, entre outros. A comunicação entre todos os subsistemas é feita pelo padrão ARM Advanced Microcontroller Bus Architecture (AMBA) através de multi-camadas do Advanced High-performance Bus (AHB). O núcleo de processamento conta ainda com uma unidade adicional de processamento de sinal (FSP).

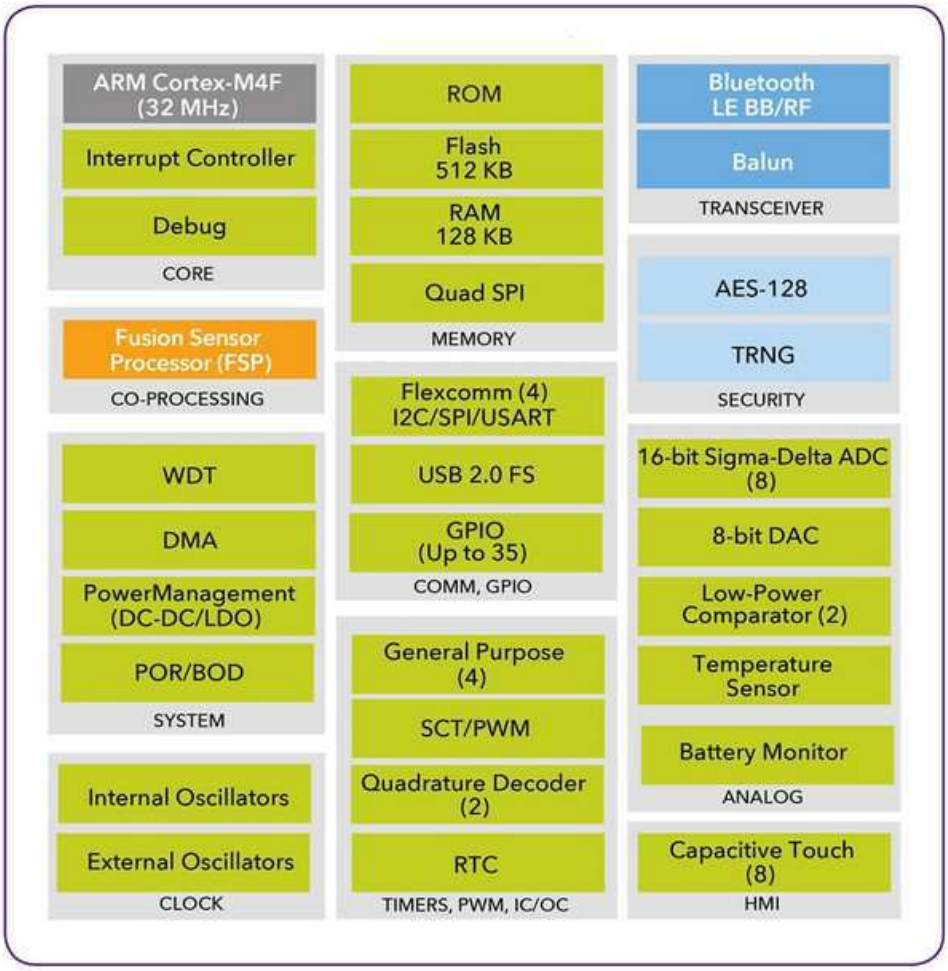

Figura 5 – Diagrama de blocos do microcontrolador NXP QN9080.

Para tornar mais simples o desenvolvimento de aplicações com seus MCUs, fabricantes oferecem gratuitamente soluções em software para programação e depuração (debugging), Kits de Desenvolvimento de Software (SDK) contendo projetos de uso com demonstrações de uso de periféricos, assim como kits de desenvolvimento de baixo custo dos microcontroladores (DK), úteis para prototipagem e validações de propostas de soluções.

Para o QN9080 a NXP oferece o *MCUXpresso IDE*, um software baseado em Eclipse e GCC capaz de editar, compilar e depurar (debugging) projetos em microcontroladores e permite configurar de forma simples pinos, clocks e periféricos. É também disponível no mercado o Kit de Desenvolvimento QN9080-DK, que trata-se de uma placa concebida para a avaliação das funções e performance do MCU, contendo tudo o que é necessário para isso, i.e., botões, circuitos auxiliares, programador e depurador, interfaces, permitindo o rápido desenvolvimento de aplicações.

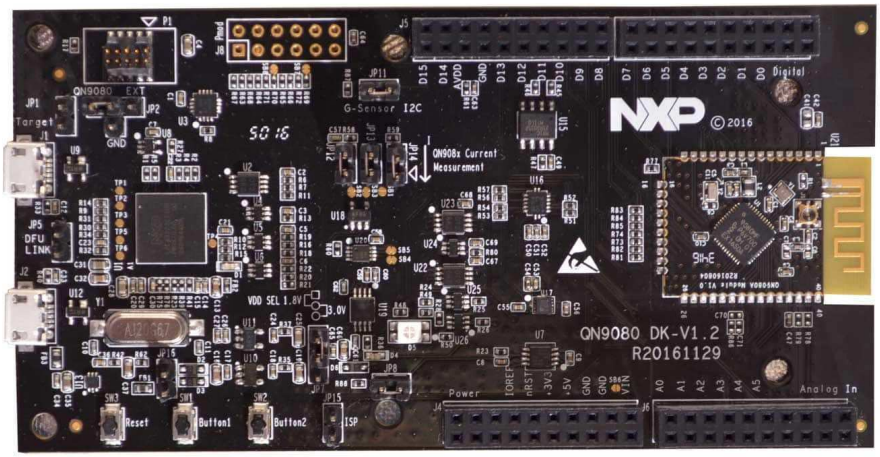

Figura 6 – NXP QN9080 Development Kit.

2.2.2 Acelerômetro: MMA8452Q

Outro componente essencial ao caso proposto é o acelerômetro. Um acelerômetro é um dispositivo que mede a vibração, ou aceleração do movimento de uma estrutura. A força causada pela vibração ou mudança de movimento produz também movimento do material piezoelétrico que o compõe, e, por consequência, produz uma variação na carga elétrica no material. Com a carga variando proporcionalmente à aceleração aplicada é possível calcular a aceleração sobre o dispositivo [14].

O MMA8452Q é um acelerômetro capacitivo de baixo consumo de energia, com capacidade de processamento interno, de três eixos com 12 bits de resolução. Além disso fornece também a possibilidade de seleção entre escalas de +-2 g, +-4 g e +-8g [15]. Na placa de desenvolvimento QN9080-DK há um acelerômetro MMA8452Q embarcado que se comunica via I2C com o microcontrolador.

Figura 7 – Acelerômetro embarcado na QN9080-DK.

2.2.3 Radiofrequência: Bluetooth Low Energy

Há também, para a aplicação de monitoramento de um motor, a necessidade de uma infraestrutura de comunicação sem fios. Por meio dessa rede será feita a transmissão dos

dados obtidos a partir do acelerômetro.

Várias tecnologias se propõem a atender às especificações mínimas de sistemas de monitoramento e acessórios conectados, como comunicação de curta/média distância, baixa taxa de dados, longa autonomia de baterias (escala de meses e anos) e longa autonomia da aplicação. Destacam-se na literatura e na indústria o Bluetooth Low Energy, Zigbee, ANT etc. Elas, porém, diferem quanto ao nicho de aplicações. Cabe aos desenvolvedores, de acordo com as especificações e requisitos de seus projetos, decidir a arquitetura a ser usada com protocolos, parâmetros, modos de conexão etc.

O Bluetooth Low Energy (BLE) especificamente tem como diferencial um menor consumo de enrgia. Como mostrado na seção anterior, um dos módulos do MCU QN9080 é um SoC BLE, que contém todas as camadas necessárias à conectividade sem fio via BLE do sistema.

2.3 Nó Central: Computador

Uma vez que os dados sejam recebidos, o computador será responsável pelo processamento.

2.3.1 Radiofrequência: Bluetooth Low Energy

Assim como o nó periférico, o nó central precisa ser capaz de acessar à rede de comunicação sem fios. No caso do trabalho desenvolvido, o computador acessou à rede usando também uma placa de desenvolvimento QN9080-DK.

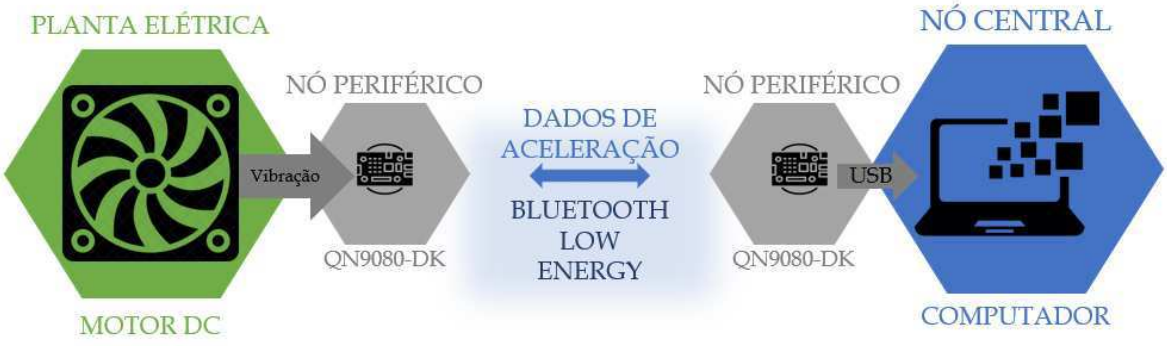

Figura 8 – Sistema com infraestrutura BLE definida.

Neste caso, o MCU QN9080 teve a função específica de receber os dados via módulo Bluetooth Low Energy e enviá-los via UART para o circuito depurador presente na placa QN9080-DK, esta converteu os dados e os enviou para o computador via USB.

2.3.2 Processamento: Redes Neurais

Como dito, o processamento feito no computador pode ser feito de diversas maneiras, a depender do desenvolvedor. Uma das maneiras é a partir de redes neurais artificiais.

Redes neurais artificiais podem ser entendidas como uma tentativa de modelar o processamento de informação que acontece em redes neurais reais, i.e., em sistemas nervosos biológicos. A descrição matemática do processamento de informação pelo cérebro humano de consenso de neurocientistas é que sistemas nervosos são compostos por milhões de células interconectadas. Cada uma delas é um arranjo interligado complexo na forma de tratar o sinal que recebe [16].

Sendo assim, redes neurais artificiais são geralmente organizadas em camadas interligadas. Cada camada é composta por certo número de nós interconectados, que seriam as células, contendo cada nó uma função de ativação, activation function. Padrões são apresentados na camada de entrada da rede, que comunicará a informação recebida às camadas ocultas e internas do sistema, onde o processamento é realmente feito por um sistema de pesos, weights, para cada conexão entre nós. As camadas ocultas convergem para a camada de saída onde a resposta do processamento é obtida [17].

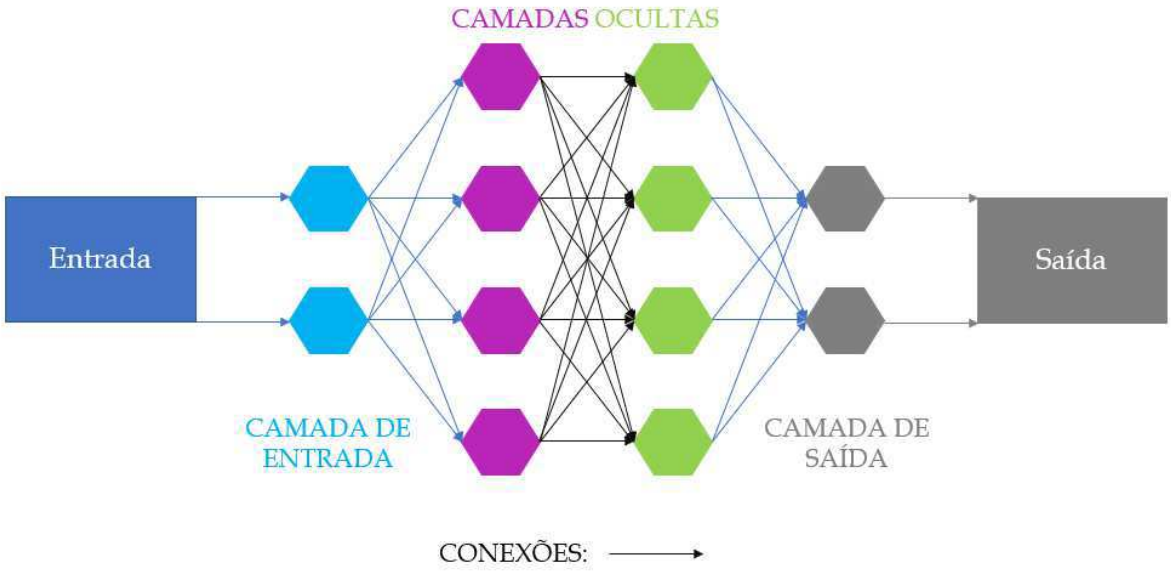

Figura 9 – Ilustração de rede neural de quatro camadas.

A maior parte das ANN possuem a chamada regra de aprendizagem, **'learning rule'**, que modificará os pesos das conexões de acordo com os padrões que são apresentados, passando a reconhecer a foto de um cachorro pela repetição de entradas de exemplos de cachorros e os acertos e erros do modelo. Percebe-se que nessa fase há o treinamento da rede neural, seus parâmetros são modificados de acordo com a sua capacidade de reconhecer um padrão identificado. Quanto mais exemplares identificados de cada padrão

forem colocados na entrada do modelo, mais ele corrigirá seus próprios parâmetros, ou seja, melhor treinará a si. E quanto melhor treinado, mais provavelmente acertará a inferência quando um padrão não identificado for posto em sua entrada.

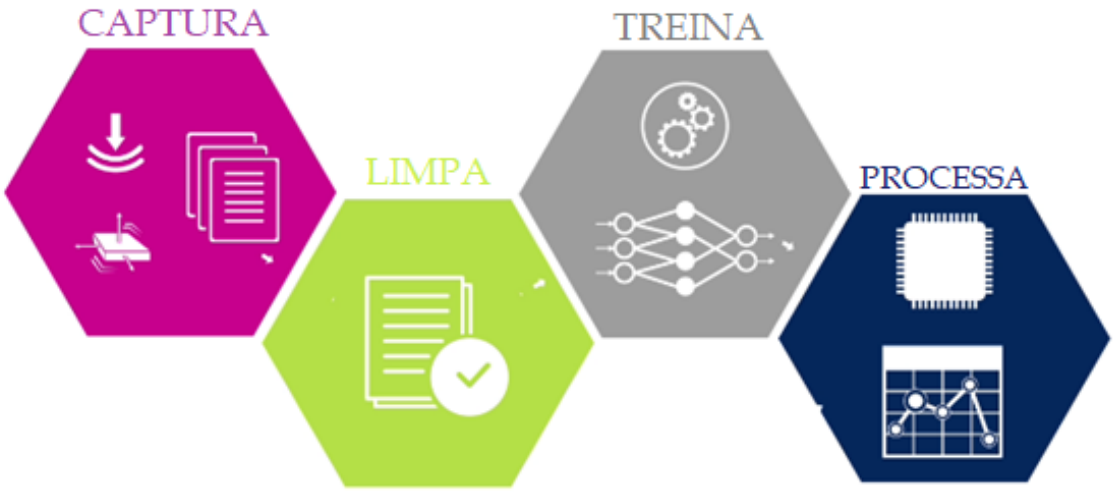

Figura 10 – Sequência de atividades para implementação de sistema de monitoramento simples.

A sequência de atividades é tal como ilustrado na Figura 10. Inicialmente obtém-se dados identificados de cada padrão previsto, que no caso de estudo seriam amostras de dados de aceleração para cada um dos quatro estados do motor e em seguida a limpeza dos dados. Logo após, o treinamento da arquitetura de redes neurais com os dados identificados, para que seus parâmetros se adaptassem aos padrões conhecidos da aplicação. E, por fim, com a obtenção do modelo treinado, o uso deste modelo para inferência do estado do motor, tendo amostras de aceleração não identificadas como sinal de entrada.

2.3.2.1 Preprocessamento: MATLAB

O sinais reais são geralmente incompletos, podendo estar ora incompletos, ora inconsistentes, conter ruídos, erros ou valores fora do intervalo imaginado etc. A fase de preprocessamento existe para que tais dados sejam corrigidos e não causem erros, ou ainda para que favoreçam um melhor desempenho, quando aplicados a um modelo de ANN. Nessa etapa estão presentes as tarefas de importar bibliotecas de tratamento de dados, importar os dados, ler os dados e verificar a falta de variáveis, a consistência dos

dados com a variável esperada, normalizá-los e dividi-los para treinamento e validação do treinamento. [1]

2.3.2.2 Framework para Redes Neurais: Keras

Dada a necessidade de desenvolvimento dos sistemas de processamento em curto prazo, assim como a complexidade das funções matemáticas utilizadas em algoritmos de aprendizado de máquina, não seria recomendável que para cada aplicação cada desenvolvedor criasse sua própria biblioteca, ou ainda que cada empresa construísse do zero sua própria biblioteca de processamento. Bibliotecas open-source, testadas têm surgido e se consolidado na academia e indústria, dentre eles The Microsoft Cognitive Toolkit, ou CNTK, da Microsoft, o TensorFlow, do Google e o Theano da University of Montreal. Outras ferramentas são construídas sob tais bibliotecas, elevando o nível de abstração e tornando a programação com tais funções menos árdua e mais rápida, é o caso da ferramenta Keras.

Keras é um API de alto nível para o desenvolvimento de redes neurais, é escrito em Python e capaz de rodar utilizando TensorFlow, CNTK ou Theano como backend. Foi desenvolvido com foco em permitir uma rápida prototipagem e experimentação, tornando possível aplicar uma ideia e ver seus resultados com o menor tempo possível. Trata-se de uma biblioteca de funções que suporta redes convolucionais e recorrentes e combinações, em CPU ou GPU. Os pontos mais relevantes e característicos do Keras são a interface amigável, programação em Python e modularidade e extensibilidade, que facilitam na organização, adição e exclusão de módulos [2].

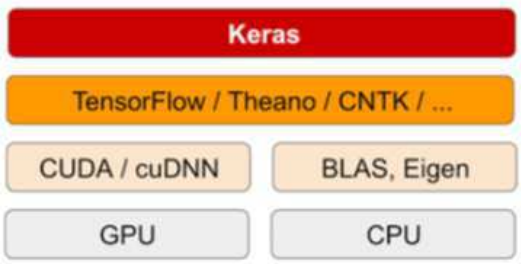

Figura 11 – Camadas inferiores ao Keras, hardware e software [2].

[1] (https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa)

3 Proposta de Solução

A proposta de solução deste trabalho é a construção um sistema de monitoramento remoto tradicional, com processamento por redes neurais, e transferir o software de processamento de dados da arquitetura de computadores para a arquitetura de microcontroladores.

Sendo assim, para esta proposta, o microcontrolador terá por função: (i) obter os dados do ambiente, (ii) processá-los e (iii) enviar para a rede sem fios apenas a informação extraída dos dados. Objetiva-se uma redução drástica na quantidade de dados a serem enviados do nó periférico ao nó central e com isso, consequentemente, menor uso do módulo de radiofrequência, menos picos de corrente demandados da bateria e menor consumo médio de energia. Para o caso inicial, a arquitetura do sistema seria como ilustrado na Figura 12.

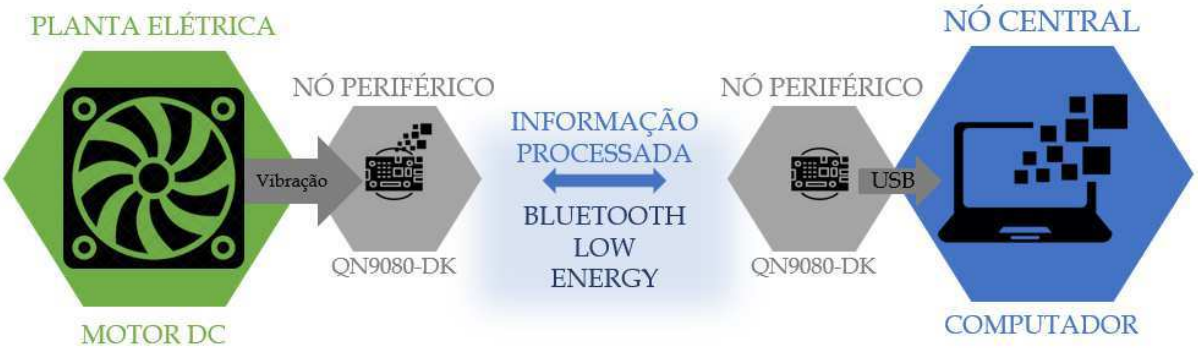

Figura 12 – Sistema com infraestrutura BLE.

A aceleração gerada pelo funcionamento do motor DC é capturada pelo acelerômetro que está integrado ao microcontrolador. O microcontrolador por sua vez recebe os dados de aceleração e os guarda, em seguida processa esses dados em uma rede neural previamente programada em sua memória. A informação sobre o estado do motor (i.e., desligado, ligado, obstruído ou desbalanceado) é enviada para a infraestrutura BLE. A informação do estado do motor é recebida pelo computador e em seguida supervisionada via nuvem ou localmente.

A sequência de atividades para execução de tal solução é similar à da solução de monitoramento comum, descrita anteriormente na Figura 10. Porém, vale ressaltar que para o caso tradicional de monitoramento, com processamento em computadores, as considerações sobre tamanho de memória, tempo de execução, consumo de energia são menos

importantes do que em microcontroladores, dadas as capacidades superiores do computador. A diferença na capacidade de processamento entre as duas arquiteturas faz com que funções, bibliotecas e frameworks que definem e treinam redes neurais e executam modelos em computadores não sejam adequadas ou possíveis para uso em microcontroladores. Daí a necessidade da adição de uma nova atividade ao fluxo de desenvolvimento, conforme mostrado abaixo. Para a solução com processamento embarcado, além das atividades de captura e limpeza de dados, definição e treino de uma arquitetura de rede neural e implementação, há também a necessidade de conversão do modelo para a arquitetura de microcontroladores.

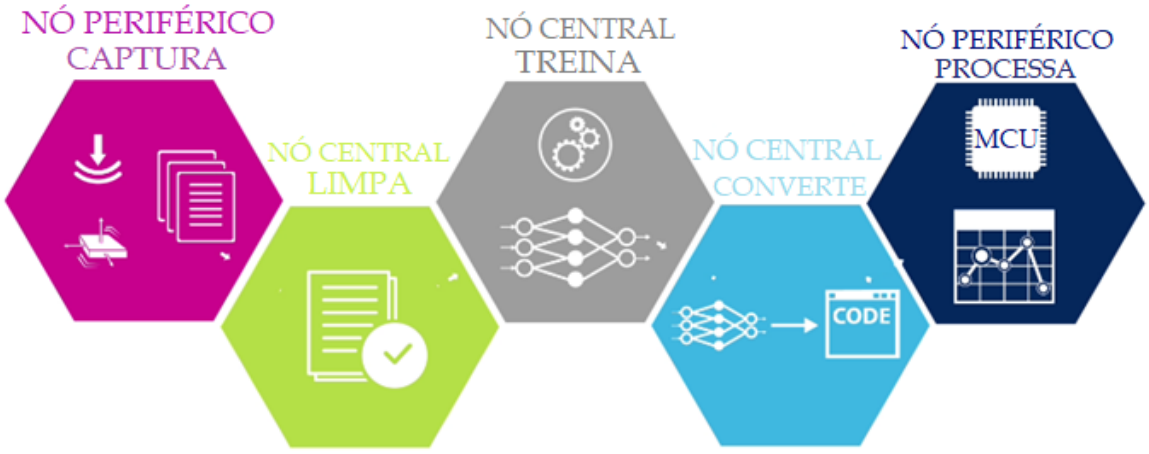

Figura 13 – Sequência de atividades prevista para execução da proposta.

O projeto de desenvolvimento desta solução, e cada etapa prevista para a execução, é apresentado a seguir, com referências aos dispositivos e tecnologias descritos na Revisão Tecnológica.

3.0.1 Captura dos Dados de Aceleração

Um pequeno ventilador DC de 5V de tensão de entrada, normalmente usado no resfriamento de fontes de computadores, foi utilizado como o motor DC e gerador da perturbação física. Os estados do motor seriam definidos como (i) desligado: com tensão de alimentação zero, (ii) ligado: com o motor funcionando em regime permanente, (iii) obstruído: com a adição de um pequeno obstáculo, menor que 25% da área total, do fluxo de ar gerado pelas hélices do ventilador, e (iv) desbalanceado: com a adição de um adesivo a uma das hélices do ventilador.

Na estrutura metálica externa desse ventilador, a placa QN9080-DK seria acoplada e parafusada. Como descrito na revisão tecnológica, a placa de desenvolvimento do QN9080 possui, dentre outros circuitos auxiliares, um acelerômetro MMA8452, capaz de se comunicar com o MCU QN9080 via I2C. Logo, com a placa de desenvolvimento acoplada ao motor, o acelerômetro seria capaz de capturar as perturbações de movimento geradas pelo motor em diferentes estados.

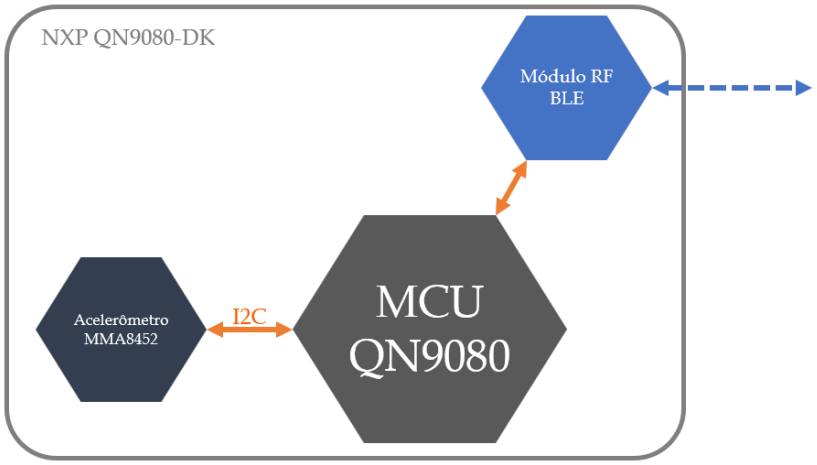

Figura 14 – Previsão de funcionamento do MCU.

Na outra ponta da rede BLE, o computador receberia os dados, por meio do software MATLAB, com comunicação serial com outra placa de desenvolvimento, e os armazenava em uma só variável. O mesmo procedimento seria feito para todos os estados, recolhendo-se e organizando os dados de aceleração por um tempo considerável. Dessa maneira seria possível compor o banco de dados para futuro treinamento.

3.0.2 Limpeza dos Dados de Aceleração

Após a obtenção de longos arrays identificados com cada estado, seria necessário aplicar as medidas descritas na seção de revisão tecnológica, separando, verificando, normalizando e identificando os dados de treinamento e os de teste.

Figura 15 – Separação, limpeza e organização dos dados.

3.0.3 Definição de Arquitetura e Treinamento da Rede Neural

Planejou-se que diferentes arquiteturas seriam implementadas e consideradas para a aplicação, tendo como base artigos que descrevessem problemáticas de identificação de estado por dados de acelerômetro e similares. Vale salientar que as arquiteturas pesquisadas não se tratavam especificamente de redes neurais para MCUs.

Considerou-se a implementação de uma rede neural convolucional, CNN, semelhante à feita por [3] que aplicou uma CNN para identificação de atividade humana, entre andar, correr ou parado, baseado em dados de acelerômetros de telefones celulares; ou das estratégias de [18] ou ainda de outra abordagem ao problema, como proposto por [7], que utilizou Support Vector Machine, SVM para processamento do sinal de acelerômetro.

As arquiteturas possíveis seriam implementadas em Python utilizando o framework Keras, e bibliotecas Tensorflow. Após verificação de quantidade de parâmetros, simplificações possíveis, resultados, uma arquitetura seria escolhida para ser implementada no MCU.

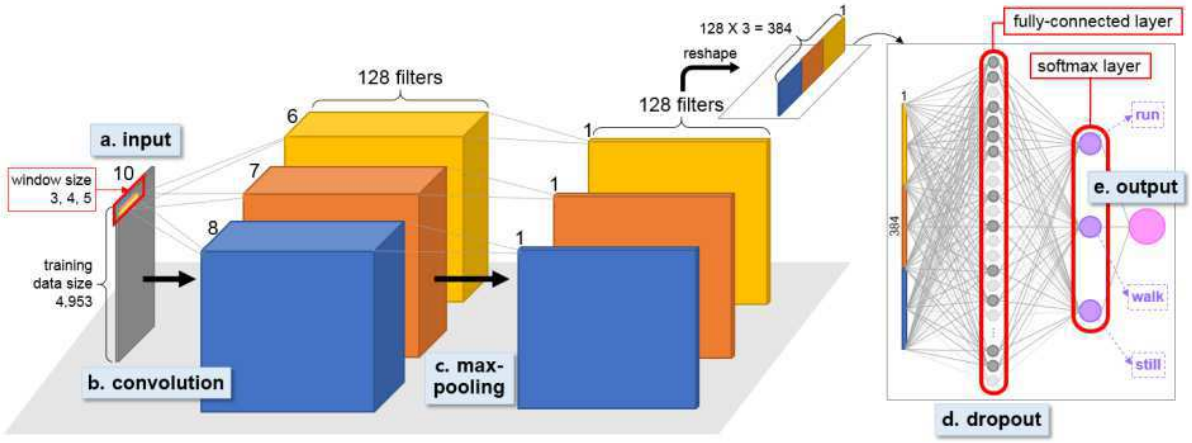

Figura 16 – Rede neural convolucional encontrada em [3].

3.0.4 Conversão do Modelo de Rede Neural para MCU

Como introduzido na revisão tecnológia, o QN9080 é baseado na arquitetura Cortex-M4 da ARM. A implementação de um modelo de rede neural em um MCU seria feita por uso de bibliotecas CMSIS, Cortex Microcontroller Software Interface Standard, providas pela ARM. De forma sucinta, o CMSIS é um pacote de bibliotecas validadas pela ARM e disponibilizadas para facilitar e acelerar o desenvolvimento de aplicações com seus processadores.

Dentre os pacotes oferecidos, dois deles foram entendidos como necessário para o desenvolvimento da proposta: (i) A biblioteca de software CMSIS-DSP, que contém funções de processamento de dados, como de matemática básica, complexa, filtros, funções matriciais, transformadas etc; e também a (ii) CMSIS-NN, que é uma coleção de funções

de redes neurais otimizadas para melhor performance e menor uso de memória em arquiteturas de núcleo Cortex-M. E de forma similar ao TensorFlow, o CMSIS-NN fornece uma série de funções consolidadas para o processamento de redes neurais convolucionais, funções de ativação, redes neurais densas, funções softmax etc.

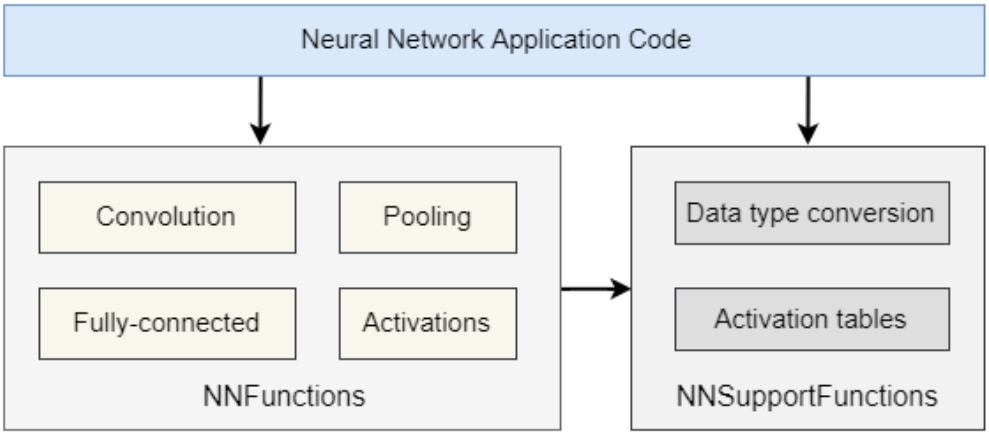

Figura 17 – Estrutura da biblioteca CMSIS-NN.

Com base em [19], [20],[4], [21] foi identificado o procedimento necessário para a conversão. As funções usadas para a construção da rede neural, importadas do TensorFlow, deveriam ser substituídas por funções do CMSIS-NN. E os parâmetros da rede neural, provenientes do modelo já treinado, deveriam ser quantizados para diminuição do consumo de memória. A ARM disponibiliza exemplos e algoritmos para conversão de modelos do framework Caffe para modelos baseados no CMSIS-NN, [22], [23]. Há também disponível, em open-source, algoritmos de conversão de modelos entre frameworks.

3.0.5 Implementação do Modelo no MCU

Assim sendo feito, o código de processamento seria implementado e depurado por meio da IDE disponibilizada pela NXP. Usando também o MCUXpresso e circuito auxiliar já integrado à placa de desenvolvimento, seria feita a medição de consumo de corrente instantânea do MCU para a estimação de autonomia da aplicação. A rotina do MCU seria programada como ilustrado abaixo.

A expectativa era de obter uma redução considerável no consumo de energia, dada a redução da quantidade de bytes passariam a ser enviadas por transação, assim como a redução da quantidade de transações. Por outro lado, houve também a expectativa de aumento de consumo, menos significante, dada a adição do processamento.

Para validar a proposta, previu-se a medição do consumo médio de corrente do MCU para duas situações.

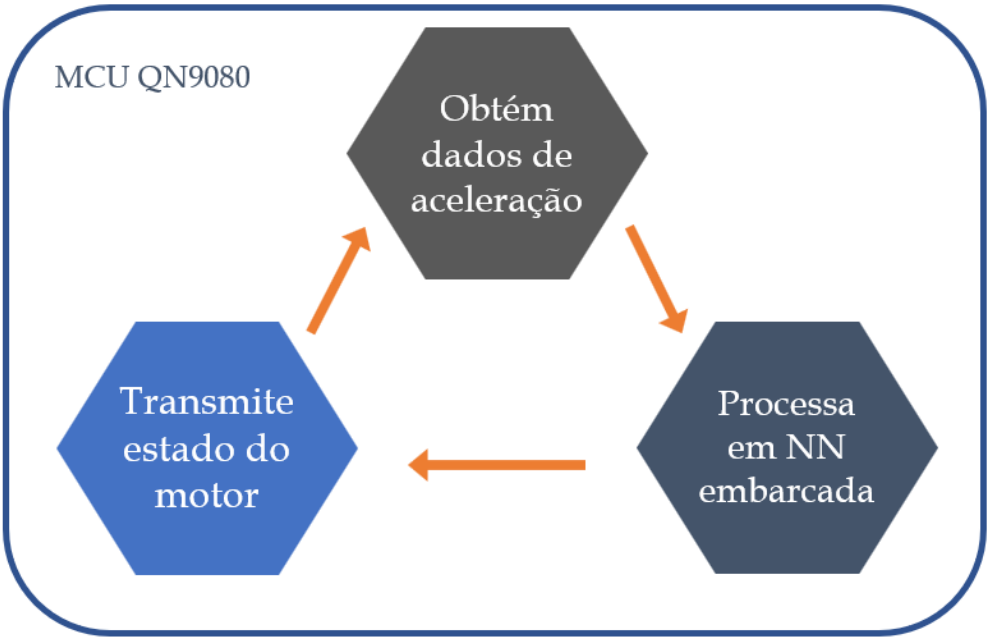

Figura 18 – Rotina proposta para o MCU. [3]

Para a primeira situação, os dados do acelerômetro seriam enviados diretamente do QN9080-DK, nó periférico, ao nó central central via BLE, sem processamento embarcado. A informação do estado do motor contida nos dados deveria ser obtida pelo sistema central a partir da rede neural desenvolvida. A autonomia da aplicação seria estimada a partir da alimentação de uma pilha de 220mAh.

Para a segunda situação avaliada, os dados do acelerômetro seriam processados no nó periférico, utilizando redes neurais embarcada para obtenção da informação do estado do motor. Em seguida, o QN9080 enviaria a informação ao sistema central via BLE. A autonomia da aplicação seria estimada a partir da alimentação de uma pilha de 220mAh.

4 Desenvolvimento e Validação

4.1 Captura dos Dados de Aceleração

Um código foi implementado usando o MCUXpresso e baseado no SDK disponibilizado pela NXP para o QN9080-DK. Foi feita a definição dos pinos do MCU como interface I2C, estabelecendo assim como a interface com o MMA8452. Funções de inicialização dos pinos e de configuração foram escritas. Os registradores do acelerômetro foram configurados para captura de aceleração nos eixos X, Y e Z, na faixa de -4g a +4g, com frequência de amostragem de 400Hz e uso da FIFO interna para armazenamento das amostras até que fossem requeridas via I2C.

```
555 /*! *********************************************************************************
556 * \brief    Initializes application specific functionality before the BLE stack init.
557 *
558 ********************************************************************************** */
559 void BleApp_Init(void)
560 {
561     /* Initialize application support for drivers */
562     BOARD_InitAdc();
563     BOARD_InitPMod_SPI_I2C(); //NN_APP: Had to initialize I2C and change pin_mux.c
564     POWER_DisablePD(kPDRUNCFG_PD_FSP); /* Power enable */
565     FSP_Init(FSP); //NN_APP: /* FSP initialization */
566     BOARD_InitDebugConsole(); //NN_APP
567
568 #ifdef MULTICORE_HOST
569     /* Init eRPC host */
570     init_erpc_host();
571 #endif
572 }
```

Figura 19 – Destaque para inicialização de pinos para uso em interface I2C com acelerômetro.

```
732     mAdvTimerId = TMR_AllocateTimer();
733     mHidDemoTimerId = TMR_AllocateTimer();
734     mBatteryMeasurementTimerId = TMR_AllocateTimer();
735
736 #if gAppUsePrivacy_d
737     mPrivacyDisableTimerId = TMR_AllocateTimer();
738 #endif
739
740     //NN_APP: Configure Accel
741     bool isThereAccel = LPI2C_ConfigureMasterAndAccel();
742         if (true == isThereAccel)
743         {
744             LPI2C_SetAccelUp();
745         }
746
747
748 }
```

Figura 20 – Destaque para código de configuração do acelerômetro.

O motor foi posto em cada um dos estados, enquanto o código em execução no MCU da placa QN9080-DK capturava os dados de aceleração e enviava via BLE para o computador. Abaixo é mostrada a estrutura de dados que era reportada do nó periférico ao nó central.

```
typedef struct Report_tag{
    uint8_t AccelStatus;
    uint8_t xAxisM;
    uint8_t xAxisL;
    uint8_t yAxisM;
    uint8_t yAxisL;
    uint8_t zAxisM;
    uint8_t zAxisL;
}Report_t; //NN_APP: Modified for sending the 3-axis data from accelerometer

```

Figura 21 – Struct criado e usado para envio de dados de aceleração nos três eixos.

Esta etapa isoladamente representa o funcionamento de um nó periférico, embarcado, para o caso tradicional de monitoramento. Por isso, foi medido o consumo de corrente médio do MCU para estimativa de autonomia da situação, definida no fim da última seção.

O computador recebia os dados através do software MATLAB, por comunicação serial, e os armazenava em uma só variável. O mesmo procedimento foi feito para todos os quatro estados, recolhendo-se dados de aceleração durante aproximadamente vinte minutos para cada estado e obtendo-se assim o primeiro conjunto de dados do sistema.

4.2 Limpeza dos Dados de Aceleração

Primeiramente os longos arrays, de em média vinte minutos, foram divididos em amostras menores, de um segundo, e a diferença de quantidades de amostras disponíveis para cada estado foi corrigida. Em seguida foi feita também uma normalização dos dados e criação de um estoque de amostras para teste, 10% do total, e outra para o treinamento da rede neural. Todos esses procedimentos foram executados no MATLAB.

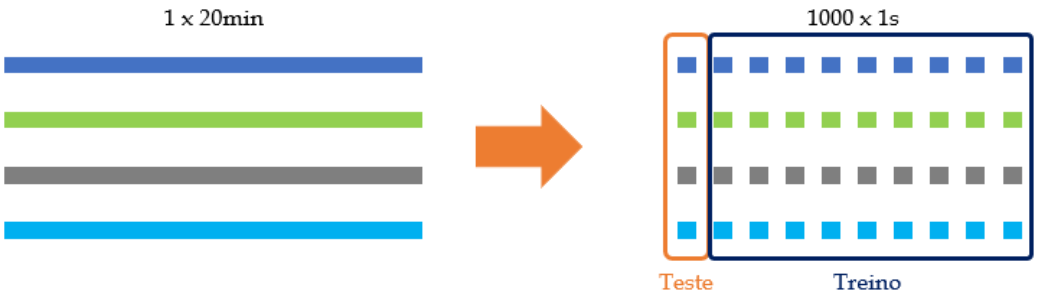

Figura 22 – Separação e organização dos dados.

```
clear data;
clear sample
clear all;

if ~isempty(instrfind)
 fclose(instrfind);
 delete(instrfind);
end
number_of_samples = 100;
sec = 10;
s = serial('COM8','BaudRate',115200);
set(s,'BaudRate',115200);
fopen(s);
data = zeros(400*sec+1,3);
sample = zeros(1,3);

for count = 1 : number_of_samples
    for t = 1:(400*sec)+1
        sample = str2num((fscanf(s)));
        data(t,:) = sample;
    end
    file_name = sprintf('sample_%i', count+100);
    csvwrite(file_name,data);
end
```

Figura 23 – Código que define a comunicação serial do MATLAB com MCU.

```
clear all;
%divides 10s into 10 pieces of 1s
number_of_samples = 200; %check the workspace
Fs = 400; %Hz
time_len_sub_samples = 1; %seconds

for count_samples = 101 : number_of_samples
    file_name = sprintf('sample_%i', count_samples);
    sample = csvread(file_name);

    for sub_count = 0 : 9
        data = sample((sub_count*Fs +1) : (sub_count+1)*Fs,:);
        sub_sample_name = sprintf('Offsample_%i%i',count_samples-101 , sub_count);
        csvwrite(sub_sample_name, data);
    end
end
```

Figura 24 – Código de separação de amostras em MATLAB.

```
Fs = 400; %Hz

OnStd  = (On  - mean(On))  ./std(On);
OffStd = (Off - mean(Off)) ./std(Off);
ObsStd = (Obs - mean(Obs)) ./std(Obs);
UnbStd = (Unb - mean(Unb)) ./std(Unb);
```

Figura 25 – Código de normalização em MATLAB.

4.3 Definição de Arquitetura e Treinamento da Rede Neural

Dispondo da base de dados de quatro mil amostras, mil para cada estado do motor, foi definida uma arquitetura de CNN, baseada em [3]. Porém, a fim de reduzir a quantidade de parâmetros da rede e facilitar a implementação e desempenho em arquiteturas de hardware mais simples, optou-se por reduzir as dimensões do sinal de entrada. Apenas

o sinal do eixo X foi considerado para geração do modelo, pois foi identificado que este seria suficiente. A redução do tamanho do sinal de entrada foi feita também a partir da transformada de Fourier. Dessa forma, o sinal de entrada da rede neural passou a ser o conjunto de 128 coeficientes de componentes de frequência que caracterizam as amostras, de um segundo, de aceleração para cada estado. Os valores dos coeficientes foram obtidos a partir da função FFT do MATLAB. Isso permitiu simplificar a rede neural, de uma CNN para uma Feedforward de três camadas e menos parâmetros, como mostrada abaixo.

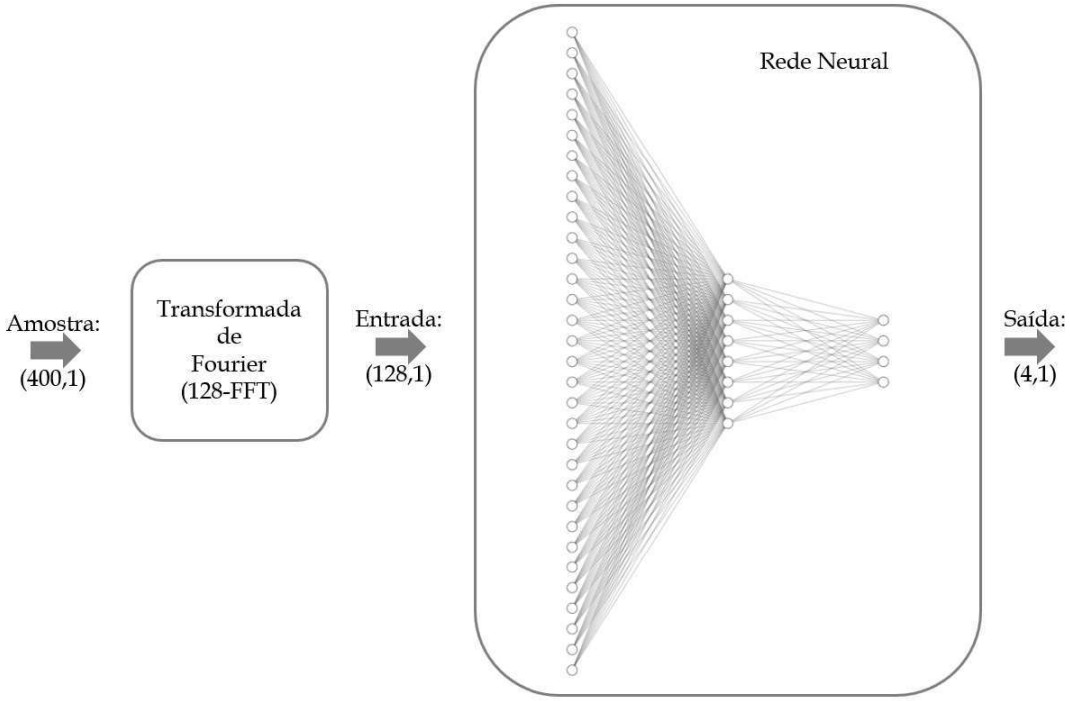

Figura 26 – Fluxograma dos subsistemas de processamento.

Em Python e usando o framework Keras, e TensorFlow como backend, foi feita a importação do banco de dados, a definição da arquitetura, seu modo de compilação e treino. Por fim o modelo foi exportado, como mostrado na Figura 27 e Figura 28 a seguir.

```
In [7]: modelnn = tf.keras.models.Sequential()
        modelnn.add(tf.keras.layers.Dense(128, use_bias = True))
        modelnn.add(tf.keras.layers.Activation('relu'))
        modelnn.add(tf.keras.layers.Dense(32))
        modelnn.add(tf.keras.layers.Activation('relu'))
        modelnn.add(tf.keras.layers.Dense(4))
        modelnn.add(tf.keras.layers.Activation('softmax'))

In [8]: modelnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        modelnn.fit(XTrainD, XTrainD, epochs=5, shuffle = True)
        modelnn.summary()
        modelnn.evaluate(XTestD, yTestD)
        modelnn.save('./models/modelcnn/model.keras')
```

Figura 27 – Rede neural definida usando o framework Keras.

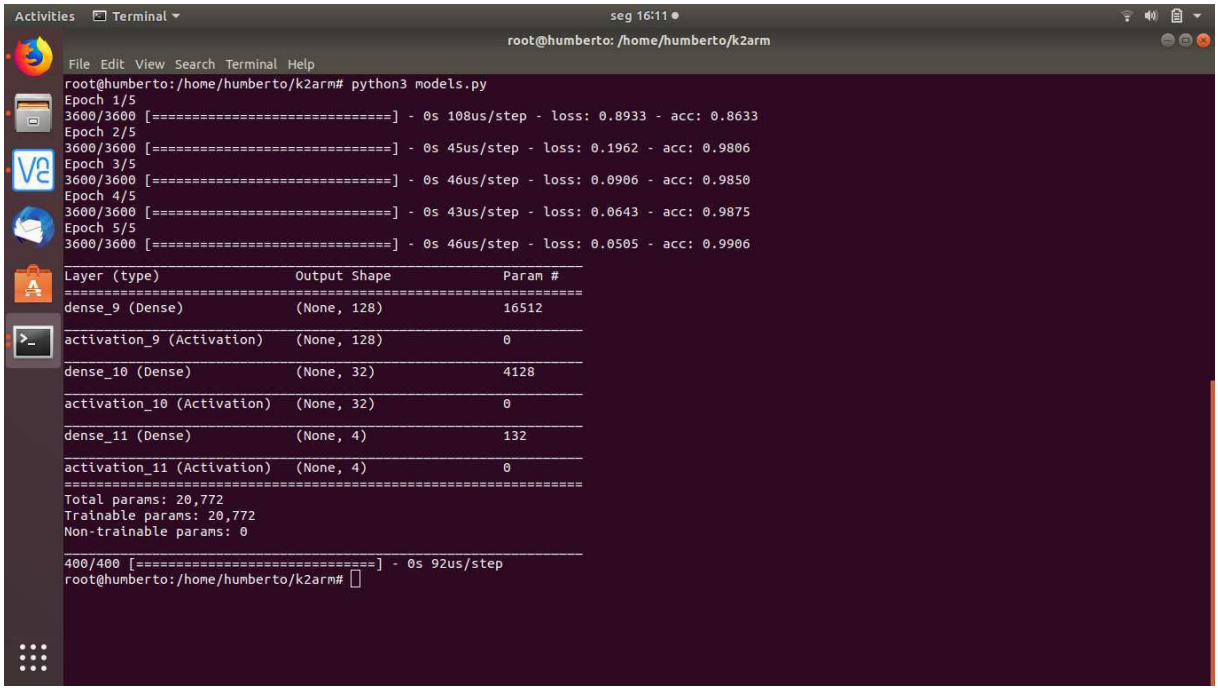

Figura 28 – Resultado da compilação e treino do modelo de rede neural proposto.

O modelo final da rede neural foi composto por 20.772 parâmetros foi capaz de acertar em mais de 99% das inferências.

Neste momento, o sistema tradicional de monitoramento foi feito completo. Tem-se (i) o nó periférico, capaz de obter os dados de aceleração do sistema físico e enviar via BLE para a rede BLE, assim como (ii) o nó central, capaz de receber via BLE os dados de aceleração e processá-los para obtenção da informação desejada, o estado do motor.

Na próxima seção será mostrado o necessário para transferir um modelo de rede neural de arquitetura de computador para um MCU.

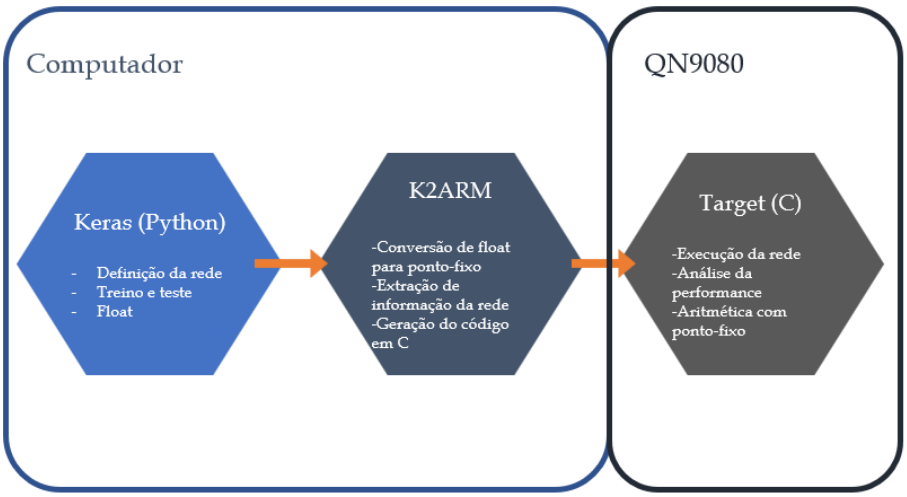

Figura 29 – Fluxograma e descrição de atividades para conversão. Retirado de [4]

4.4 Conversão do Modelo de Rede Neural: K2ARM

Dentre os vários algoritmos que automatizam o procedimento de conversão, descrito na seção anterior, foi utilizado o K2ARM [4] para reescrever automaticamente as funções do modelo do Keras em funções nativas do CMSIS-NN.

In [1]:
```
# ------------------------------------------------------------------------------------------------
#                                     load pre-trained model
# ------------------------------------------------------------------------------------------------
classifier = k.models.load_model(modelPath)

# ------------------------------------------------------------------------------------------------
#                                     create k2arm parser
# ------------------------------------------------------------------------------------------------
parser = k2arm(outputFilePath='../target/', fixPointBits=qFormat,
               model=classifier, evalData=VmnistEvalData)

# ------------------------------------------------------------------------------------------------
#                       parse the model from keras to CMSIS-NN parameters
# ------------------------------------------------------------------------------------------------
parser.quantizeWeights()
parser.findOutputFormat()

# ------------------------------------------------------------------------------------------------
#                                     write the C-code files
# ------------------------------------------------------------------------------------------------
parser.storeWeights()
parser.storeDimension()
parser.storeOutShiftParams()
parser.storeNetFunction()
parser.storeBitSize()
print('Model successfully parsed and C-code generated!')
```

Figura 30 – Algoritmo conversor de modelo Keras para CMSIS.

In [1]:
```
# ------------------------------------------------------------------------------------------------
#                                     load pre-trained model
# ------------------------------------------------------------------------------------------------
classifier = k.models.load_model(modelPath)

# ------------------------------------------------------------------------------------------------
#                                     create k2arm parser
# ------------------------------------------------------------------------------------------------
parser = k2arm(outputFilePath='../target/', fixPointBits=qFormat,
               model=classifier, evalData=VmnistEvalData)

# ------------------------------------------------------------------------------------------------
#                       parse the model from keras to CMSIS-NN parameters
# ------------------------------------------------------------------------------------------------
parser.quantizeWeights()
parser.findOutputFormat()

# ------------------------------------------------------------------------------------------------
#                                     write the C-code files
# ------------------------------------------------------------------------------------------------
parser.storeWeights()
parser.storeDimension()
parser.storeOutShiftParams()
parser.storeNetFunction()
parser.storeBitSize()
print('Model successfully parsed and C-code generated!')
```

Figura 31 – Resultados da execução do conversor.

O K2ARM, escrito em Python, também quantifica os valores de peso (weight) e viés (bias) do modelo para o o melhor formato de ponto fixo possível, no sistema de representação Q. Após verificar cada possibilidade de representação, atem-se àquela que produz o resultado de melhor precisão.

Ao fim da execução, quatro arquivos foram gerados arquivos baseados no modelo desenvolvido no Keras.

4.5 Implementação do Modelo no MCU

A rotina do MCU foi definida tal como ilustrado abaixo: primeiramente o MCU acessa os registradores do MMA8452 para obter os dados de aceleração em tempo real e salva em um array 512 amostras sequenciais. Depois disso, o array é aplicado em funções que retornam 128 coeficientes da FFT. Os coeficientes são aplicados ao modelo embarcado e para inferência do estado do motor. O resultado de 1 a 4 é enviado via BLE.

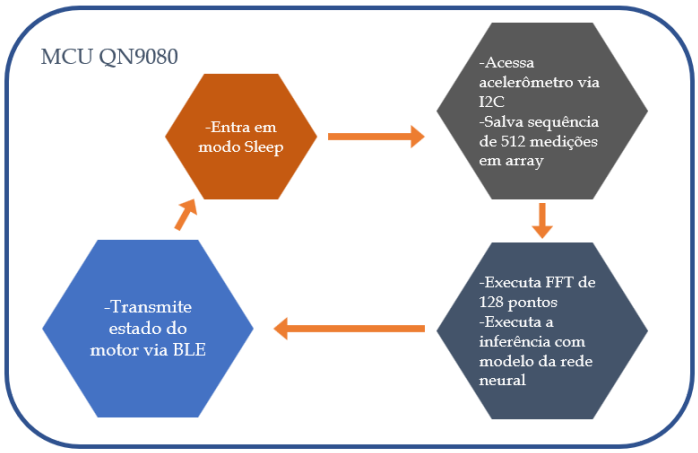

Figura 32 – Rotina final a ser executada pelo MCU.

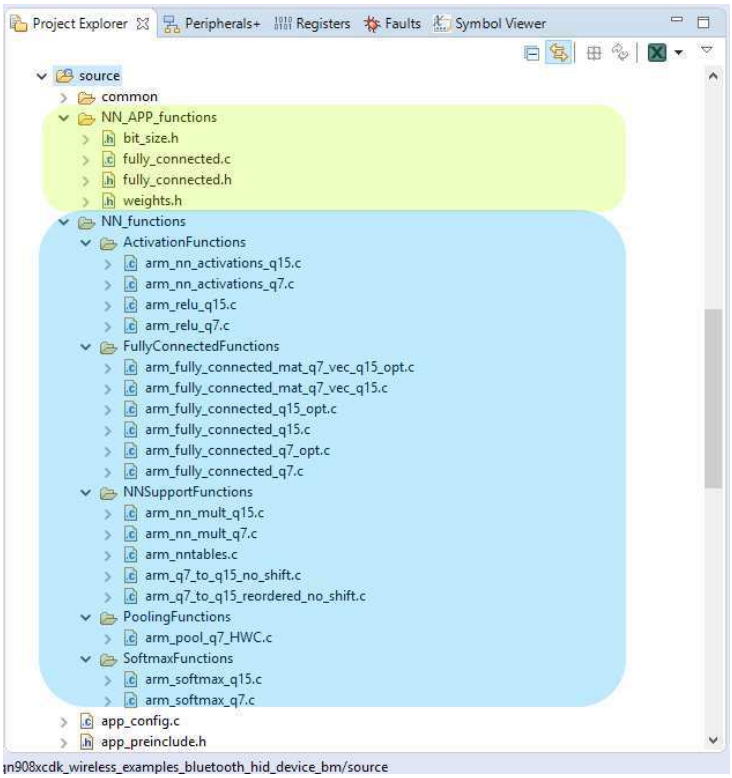

Figura 33 – Destaque das bibliotecas e arquivos que foram adicionados ao projeto.

Para que as funções pudessem ser compiladas, as bibliotecas de funções DSP e NN foram incluídas ao projeto como destacado na Figura 33.

Em destaque de cores estão as três camadas da rede neural que foram geradas automaticamente. Os valores dos pesos também foram salvos em um arquivo .c.

```c
/**
 * @brief       Autogenerated module by Keras2arm.py
 *              This module contains a function which can be used to
 *              classify MNIST images using the ARM-CMSIS-NN lib
 * @Note        This module uses the q15 implementation
 * @date        2019-06-19 00:59
 * @file        fully_connected.c
 * @author      Raphael Zingg zing@zhaw.ch
 * @copyright   2018 ZHAW / Institute of Embedded Systems
 */
#include "arm_nnfunctions.h"
#include "weights.h"
#include "fully_connected.h"
#include <stdint.h>

q15_t aq15_out_Buf[128];

uint8_t fully_connected_run(q15_t * aq15_input_data)
{
    int16_t i16_max_val = 0x8000, i = 0;
    uint8_t u8_max_prediction = 0;

    arm_fully_connected_q15(
        aq15_input_data,
        aq15_layer_1_weights,
        LAYER_1_IN_DIM,
        LAYER_1_OU_DIM,
        LAYER_1_IN_SHIFT,
        LAYER_1_OU_SHIFT,
        aq15_layer_1_bias,
        aq15_out_Buf,
        NULL);
    arm_relu_q15(
        aq15_out_Buf,
        LAYER_1_OU_DIM);
    memcpy(aq15_input_data, aq15_out_Buf, sizeof(aq15_out_Buf));
    arm_fully_connected_q15(
        aq15_input_data,
        aq15_layer_2_weights,
        LAYER_2_IN_DIM,
        LAYER_2_OU_DIM,
        LAYER_2_IN_SHIFT,
        LAYER_2_OU_SHIFT,
        aq15_layer_2_bias,
        aq15_out_Buf,
        NULL);
    arm_relu_q15(
        aq15_out_Buf,
        LAYER_2_OU_DIM);
    memcpy(aq15_input_data, aq15_out_Buf, sizeof(aq15_out_Buf));
    arm_fully_connected_q15(
        aq15_input_data,
        aq15_layer_3_weights,
        LAYER_3_IN_DIM,
        LAYER_3_OU_DIM,
        LAYER_3_IN_SHIFT,
        LAYER_3_OU_SHIFT,
        aq15_layer_3_bias,
        aq15_out_Buf,
        NULL);
    memcpy(aq15_input_data, aq15_out_Buf, sizeof(aq15_out_Buf));
    arm_softmax_q15(
        aq15_input_data,
        LAYER_3_OU_DIM,
        aq15_out_Buf);
    for(i = 0; i < LAYER_3_OU_DIM; i++) {
        if(i16_max_val < aq15_out_Buf[i]) {
            i16_max_val = aq15_out_Buf[i];
            u8_max_prediction = i;
        }
    }
    return u8_max_prediction;
}

```

Figura 34 – Destaque das três camadas da rede neural em C.

```
#include "arm_nnfunctions.h"
q15_t aq15_layer_1_weights[] = {-825, 655, 1044, 99, -98, -366, -15, -866, 338, -332, -27, -50, -6, 60, -642, -736, -1290, -492
q15_t aq15_layer_1_bias[] = {898, 880, 728, 839, 700, -136, 258, 788, 853, -130, 258, -72, 925, -15, 94, -447, -124, 29, 712, 7
q15_t aq15_layer_2_weights[] = {-2219, -1389, 1242, -2192, 1426, -37, -483, -2079, 501, -1195, 2115, -1299, -537, 1458, 1466, -
q15_t aq15_layer_2_bias[] = {-57, 47, 788, -49, 144, 697, 152, 13, 84, 737, -156, 474, 705, 635, -161, -93, 430, 579, 605, 1, -
q15_t aq15_layer_3_weights[] = {-2669, -2519, 869, 649, 2121, 452, -2714, 183, -2034, -650, 436, -2251, -261, -1348, -1180, -19
q15_t aq15_layer_3_bias[] = {145, 712, -572, -349};
#define LAYER_1_IN_DIM 128
#define LAYER_1_OU_DIM 128
#define LAYER_2_IN_DIM 128
#define LAYER_2_OU_DIM 32
#define LAYER_3_IN_DIM 32
#define LAYER_3_OU_DIM 4
#define LAYER_1_IN_SHIFT 15
#define LAYER_1_OU_SHIFT 14
#define LAYER_2_IN_SHIFT 14
#define LAYER_2_OU_SHIFT 13
#define LAYER_3_IN_SHIFT 13
#define LAYER_3_OU_SHIFT 11
```

Figura 35 – Arquivo com valores de pesos (weights) quatizados.

4.6 Validação

Para a última fase do projeto, a validação, foi planejado colocar o motor em cada um dos estados por longos períodos de tempo e o salvar em um array a sequência de resultados das inferências. A inferência seria feita por pelo menos mil vezes e o resultado seria enviado ao nó central para futuro processamento. A partir da quantidade de erros e do universo de amostras, seria possível quantificar a rede a performance da rede e desenhar uma matriz de confusão. Não sendo feita a validação, também não foi possível demostrar a economia de energia, dado que não foi atestada a precisão da inferência. Até a data de entrega deste relatório, não foi possível concluir o trabalho.

5 Conclusões

Através deste trabalho foi possível abordar diversas tecnologias atuais para a especificação de um sistema de monitoramento remoto simples, baseado em dados de aceleração enviados por comunicação BLE e processados em CPU. Foi possível implementá-lo por completo, utilizando um MCU moderno na aquisição de dados, e aplicar processamento de sinais e redes neurais à solução, apesar de ter sido o primeiro contato do autor com técnicas de aprendizado de máquina.

A partir dessa primeira solução foi possível explorar quais são as limitações e exigências que surgem no deslocamento do processamento para nós mais periféricos do sistema, os nós embarcados. Foi possível experimentar diferentes estratégias e algoritmos para definição do processamento, em função das limitações da arquitetura, e converter o modelo final de CPU para MCU, buscando a forma mais simples e direta para tal tarefa.

Por fim foi possível especificar um sistema de processamento em nó periférico e implementá-lo em código por completo no dispositivo. Foram validadas as etapas de (i) captura de dados, (ii) o estabelecimento e funcionamento da comunicação via BLE, (iii) o envio de dados relativos ao sistema mecânico para todos os quatro estados, (iv) o desempenho da rede em processar tais dados e extrair a informação de estado em CPU, (v) a conversão do modelo de rede neural de CPU a MCU, (vi) a implementação da transformada rápida de fourier (FFT) [24] no MCU, (vii) e o (viii) funcionamento da rotina proposta em execução (captura, tratamento em FFT, inferência com rede neural embarcada), restando apenas a fase final de validação a ser feita que seria a verficação dos resultados inferidos e medições de consumo energétivo.

Por isso, o autor considera ainda atividades futuras para melhoria da solução. Seriam estas a implementação do código para validação descrita acima do modelo atual; realização de medição de consumo e estimações de autonomia; aplicação de modos de baixo consumo de forma mais crítica nas sub-rotinas do MCU; a otimização da rede neural em arquitetura, quantidade de nós e de parâmetros; a utilização dos coeficientes de FFT obtidos diretamente por funções nativas do MCU como banco de dados, e não do MATLAB; adição de técnicas de de controle de riscos e redução de erros nas inferências, como feito em casos de aplicação industrial, espacial; finalização do modelo físico reduzido; e desenvolvimento de aplicativo móvel com indicação do estado do motor em tempo real, substituindo completamente o CPU por soluções embarcadas.

Referências Bibliográficas

1 SITE. https://cecas.clemson.edu/cvel/auto/systems/auto-systems.html. Acessado em junho de 2019. 8, 12

2 SITE. https://keras.io/. Acessado em junho de 2019. 8, 23

3 LEE SANG MIN YOON, H. C. S.-M. Human activity recognition from accelerometer data using convolutional neural network. 2017. 8, 13, 27, 29, 32

4 SITE. https://github.com/InES-HPMM/k2arm. Acessado em junho de 2019. 8, 28, 34, 35

5 FLEMINGS, B. Microcontroller units in automobiles. *IEEE Vehicular Technology Magazine*, 2011. 12

6 SITE. ttps://uk.reuters.com/article/uk-volkswagen-results-software/volkswagen-ceo-expects-software-to-make-up-90-percent-of-auto-industry-innovation-idukkbn1qt1zk. *UK Reuters*, Acessado em junho de 2019. 12

7 CRADDOCK, X. F. L. M. A. E. P. R. P. I. Extending the battery lifetime of wearable sensors with embedded machine learning. IEEE, 2018. 13, 27

8 CHAN D. ESTEVE, J.-Y. F. C. E. M.; CAMPO, E. Smart wearable systems: Current status and future challenges. *Artificial Intell. in Medicine*, v. 56, n. 3, p. 137–156, 2012. 13

9 KUMAR, R. J. S. P. Ultra-low power digitally operated tunable mems accelerometer. *IEEE Sensors*, v. 16, n. 24, p. 8715–8721, 2016. 14

10 VERGADOS, N. A. P. S. A. N. D. D. Energy-efficient routing protocols in wireless sensor networks: A survey. *Commun. Surveys Tuts*, v. 15, n. 2, p. 551–591, 2013. 14

11 MAURO X. FAFOUTIS, S. M. A. D.; DRAGONI, N. Detecting and preventing beacon replay attacks in receiver-initiated mac protocols for energy efficient wsns. *Proc. 18th Nordic Conf. Secure IT Systems, (NordSec)*, n. 2, p. 1–16, 2013. 14

12 DUNKELS, B. G. A.; VOIGT, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. *IEEE Int. Conf. on Local Computer Networks*, p. 455–462, 2004. 14

13 SITE. https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller. Acessado em junho de 2019. 17

14 SITE. https://www.omega.com/en-us/resources/accelerometers. Acessado em junho de 2019. 19

15 DATASHEET. Mma8452q, 3-axis, 12-bit/8-bit digital accelerometer technical data. 2019. 19

16 ROJAS, R. Neural networks: A systematic approach. In: SPRINGER. [S.l.], 1996. 21

17 SITE. http://pages.cs.wisc.edu/ bolo/shipyard/neural/local.html. Acessado em junho de 2019. 21

18 GARCIA, L. M. N. S.-P. J. M. M. A. C. B. Anomaly detection based on sensor data in petroleum industry application. *Sensors*, v. 15, p. 2774–1797, 2015. 27

19 ZHANG, N. S. e. a. Y. Hello edge: Keyword spotting on microcontrollers. *ArXiv*, 2017. 28

20 LAI NAVEEN SUDA, V. C. L. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *ArXiv*, 2018. 28

21 ARM. Deploying a convolutional neural network on cortex-m with cmsis-nn. Acessado em junho de 2019. 28

22 ARM. Tutorial: Deploying a caffe mnist model using the arm nn sdk. Acessado em junho de 2019. 28

23 SITE. Deploying a caffe model on openmv using cmsis-nn. 28

24 NXP. Application note: An12144 qn908x fft application. 2018. 39