

Prótese Eletrônica Auxiliar Infantil

Gabriel Genari Carmona

Programa de Engenharia Eletrônica

Universidade de Brasília - FGA

Brasília, Brasil

gabrielgcarmona@gmail.com

Tiago Rodrigues Pereira

Programa de Engenharia Eletrônica

Universidade de Brasília - FGA

Brasília, Brasil

tiagorodriguesp2@gmail.com

Resumo—Este documento apresenta uma proposta de projeto final para a matéria de Sistemas Operacionais Embarcados, cujo objetivo é de apresentar o desenvolvimento de sistemas embarcados utilizando sistemas operacionais. Consequentemente, o presente relatório, tem a finalidade apresentar uma prótese eletrônica de membro superior projetada para faixa etária de 3 a 7 anos. Com funcionalidade de abrir e fechar a mão com base em sinais eletromiográficos capturados no antebraço de um indivíduo de mão amputada.

I. INTRODUÇÃO

A utilização das próteses ortopédicas são datadas dos tempos mais antigos, sendo o registro mais antigo na Índia Antiga entre 3800 a.C e 1400 a.C [1]. O desenvolvimento delas evoluiu de forma exponencial no último século com a evolução da área biomédica utilizando componentes eletrônicos, alcançando funções de alta complexidade como mecanismos telemétricos eletrônicos de alta complexidade baseados nos joelhos humanos [2].

Conforme a Constituição Brasileira é instituída a Lei Nº 13.146 de Inclusão da Pessoa com Deficiência (Estatuto da Pessoa com Deficiência), destinada a assegurar e a promover, em condições de igualdade, o exercício dos direitos e das liberdades fundamentais por pessoa com deficiência, visando à sua inclusão social e cidadania [3]. Entretanto, grande parte das próteses com funções eletrônicas tendem a preços inacessíveis para a maior parte da população brasileira, onde uma prótese de alta capacidade esteja entre os valores de R\$46.000,00 e R\$190.000,00 [4].

Atualmente as próteses de alta capacidade possuem controle mio-elétrico obtido dos músculos presentes nos membros superiores do usuário, onde normalmente uma cirurgia de reinervação muscular é realizada para aumentar a precisão do controle mio-elétrico. Dessa forma, a prótese auxilia nas funções de segurar objetos e em apoio, onde os modelos mais avançados é possível trocar mão da prótese para realizar tarefas específicas como andar de bicicleta ou praticar ginástica.

Dentre os modelos mais avançados disponíveis no mercado temos o sistema MyoBock, feito pela empresa Ottobock para crianças entre 3 a 5 anos. É uma combinação da Mão Elétrica 2000 (Fig. 1), o MyolitoWrist 2000 (Fig. 2) e a luva MyolitoSkin Natural (Fig. 1); utilizando a tecnologia de captura de sinais neuromusculares desenvolvido para próteses de adulto. O sistema de controle (Fig. 3) desse modelo realiza o processamento dos sinais de EMG e envia comandos de controle para a

mão protética, possuindo 7 programas de controle que devem ser escolhidos manualmente usando aplicativo desenvolvido para prótese (necessário conexão cabo ou wireless caso tenha acessório bluetooth). Esse programas foram desenvolvidos para ser ajustado a preferencia e a qualidade do sinal muscular capturado pelos eletrodos (Exemplo na Fig. 4, como exemplos existem rotinas específicas para pacientes com dois sinais fortes, dois sinais fracos, apenas um músculo e um sinal fraco, um sinal fraco ou sem sinal, etc. No melhor dos casos, onde são capturados dois sinais fortes, essa prótese pode variar a velocidade de abertura e fechamento da mão e a força de pegada, ambos com base nos dois sinais de EMG adquiridos [5].



Figura 1: Luva MyolitoSkin Natural com Mão Elétrica 2000



Figura 2: Componente MyolitoWrist 2000

Especificamente a população brasileira de deficientes motores com grande dificuldade é constituída por 3.698.929 pessoas, ou cerca de 2% da população brasileira total [6].

Representando um número expressivo de pessoas onde apenas 3%, cerca de 110.000, teriam alguma condição para



Figura 3: Controlador 7 em 1 para sistema Myobock



Figura 4: Exemplo de eletrodo para sistema Myobock

utilizar alguma prótese de funções eletrônicas [7]. Acrescido a estes fatos, a população infantil, entre 0 e 9 anos, tem-se um número muito menor de deficientes motores que a população adulta, devido principalmente a uma parte grande dos deficientes não nascem com ela mas adquirem após algum trauma que compromete as funções motoras [6]. Ocasionando uma eventual diminuição no investimento em próteses complexas nesta faixa etária, mesmo sendo as mais críticas para desenvolvimento das funções básicas.

Tornando assim a proposta de uma prótese de baixo custo com função de abrir e fechar para faixa etária de 3 a 7 anos, sendo a movimentação dos dedos realizada por servomotores. Sendo implementado um sistema embarcado que realiza o processamento do sinal de eletromiografia (EMG) capturado no antebraço e controla o gesto de abertura e fechamento da mão.

Uma das funções mais importantes para desenvolvimento de uma prótese é a velocidade de pegada (fechamento da mão), sendo necessário ser rápido o suficiente para uso suave do usuário. Para atividades cotidianas que usam o gesto de pegada a velocidade de flexão varia entre 170 graus/segundo e 300 graus/segundo para usuário normal e para próteses comerciais uma velocidade entre 60-103 graus/segundo. Ou seja, para próteses comerciais o tempo para fechar a mão completamente varia entre 1 e 1,5 segundos [14].

O espectro de EMG inclui sinais que variam de 10 Hz a 1 kHz, principalmente entre 50 e 150 Hz, enquanto a voltagem possui valores de 50 μ V a 9 mV. O sinal de EMG possui níveis elevados de interferência e ruído exigindo um circuito de condicionamento cuidadosamente projetado para permitir a análise do sinal.

O sinal EMG de cada músculo envolve muitas ações potenciais resultando em várias MUAPs (Motor Unit Action Potential) de cada unidade de motor. Desta forma, é possível distinguir o espectro do músculo em função da distância entre

os eletrodos e as contrações com níveis de intensidade.

O estímulo para a contração muscular é geralmente um impulso nervoso, que chega à fibra muscular através de um nervo. O impulso nervoso propaga-se pela membrana das fibras musculares (sarcolema) e atinge o retículo sarcoplasmático, fazendo com que o cálcio ali armazenado seja liberado no hialoplasma. Ao entrar em contato com as miofibrilas, o cálcio desbloqueia os sítios de ligação da actina e permite que está se ligue à miosina, iniciando a contração muscular. Assim que cessa o estímulo, o cálcio é imediatamente rebombeado para o interior do retículo sarcoplasmático, o que faz cessar a contração. Dessa forma, esses sinais podem ser captados na superfície da pele por meio de eletrodos, uma vez que geram diferenças de potencial e consequentemente formam campos elétricos.

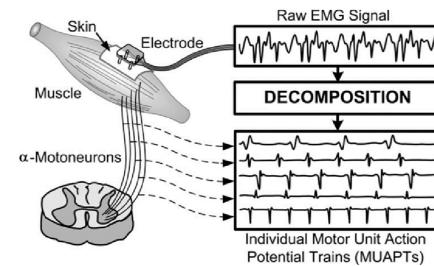


Figura 5: Representação da aquisição de um EMG

Os músculos do antebraço que movimentam o punho, a mão e os dedos são muitos e variados. Esses músculos que integram esse grupo que atua nos dedos são conhecidos como músculos extrínsecos da mão, pois se originam fora da mão e se inserem nela.

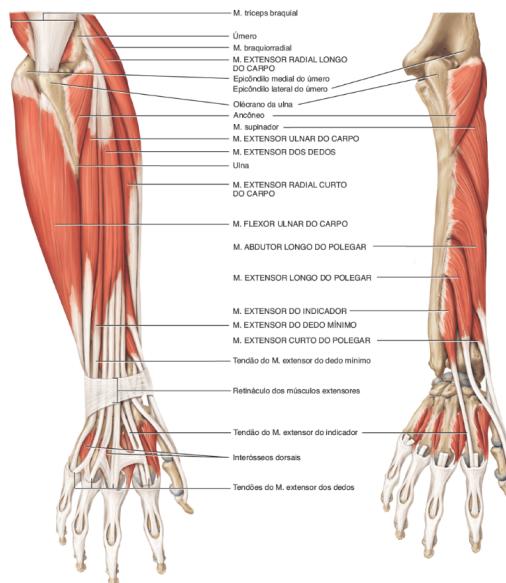


Figura 6: Musculatura do antebraço

Conforme a figura 6 é perceptível que com a grande quantidade de músculos adjacentes para a aquisição das MUAPs

existe a influência do *crosstalk*, em que o tem-se o sinal captado em um músculo entretanto o sinal é gerado em outro músculo. Sendo assim, a localização dos eletrodos é de grande importância, pois a localização incorreta pode causar resultados indesejados, assim como aumento de interferências no sistema. Consequentemente, foi analisado que para a melhor aquisição do sinal eletromiográfico, os eletrodos positivo e negativo devem estar adequadamente posicionados enfileirados na parte interna do antebraço, com a mesma distância entre eles conforme a figura 7. Sendo necessário eletrodos como o de tipo de gel sólido com sensor de prata (Ag/AgCl) MP43 circular de 43mm da marca MedPex.



Figura 7: Posicionamento dos eletrodos no antebraço

Existem algumas considerações importantes a serem feitas em relação a qualidade do sinal além do posicionamento dos eletrodos, que são: tipos de fibras musculares, diâmetro das fibras, distâncias entre as fibras, tipo de tecido em cada fibra, ponto de captação do sinal, distribuição espacial das unidades motoras, quantidade de unidades motoras recrutadas e propriedades dos eletrodos utilizados para detecção do sinal. Dessa forma, todos esses fatores podem alterar o sinal de EMG.

II. DESENVOLVIMENTO

A solução proposta de implementar uma prótese de baixo custo para faixa etária de 3 a 7 anos controlada por sinais de eletromiográfica (EMG) temos que os principais objetivos desse projeto são:

- Desenvolver prótese de mão de baixo custo com função de pegar objetos de diferentes tamanhos e formatos;
- Classificar sinais eletromiográficos dos membros superiores do usuário final referentes a movimentos da mão;
- Automatizar processo de abertura e fechamento por meio de software embarcado.

O principal benefício de ser uma prótese com tecnologia de controle mais avançada que realiza a função de pegar e segurar objetos é a abertura de novas possibilidade para quantidade de tarefas que uma criança entre 3 e 7 anos pode fazer com autonomia. Durante essa idade são desenvolvidos a base do convívio social com outras crianças e melhor autonomia motora dos membros superiores em esportes, escrita e outros casos. Como a mecânica do protótipo de prótese é limitado e não é o foco do desenvolvimento atual, a ideia é que a criança

se acostume a usar uma prótese com funcionalidade básica de pegar objetos de tamanho médio de até 100 mm de espessura, como usar para andar de bicicleta segurando o guidão.

Onde o escopo proposto é de implementar com base em uma plataforma utilizada de prótese feita em impressão 3D para tamanho reduzido para o público-alvo. Sendo este crianças amputadas de parte de um dos membros superiores na faixa etária entre 3 a 7 anos de idade.

Tendo em vista o escopo do produto e adicionando a limitações de ser um produto de baixo custo, variando entre 800-1500 reais para implementação completa, o principal foco desse projeto é na solução eletrônica com software embarcado, deixando melhorias mecânicas da prótese de lado.

Além disso, por se tratar de um protótipo inicial e as necessidades burocráticas para fazer testes de equipamentos biomédicos, ou seja, necessidade da liberação para testes externos com pessoas do público alvo seja feita por um conselho de ética os testes serão feitos em escala reduzida. Onde o intuito é de demonstrar que a tecnologia implementada funciona para pessoas sem dificuldades motoras.

A. Requisitos funcionais:

- Abrir e fechar mão robótica com tempo variável;
- Capturar sinais de eletromiografia (EMG);
- Classificar sinais de EMG em aberto e fechado.

B. Requisitos não-funcionais:

- Módulo de controle e processamento utilizar um computador de placa única com sistema operacional;
- Utilizar sistema operacional Linux;
- Módulo de controle e processamento portátil;
- Alimentação dos sistemas deve ser portátil
- Módulo de controle e processamento deve funcionar de forma offline;
- Adquirir sinais de EMG com componentes de frequência de até 800Hz;
- Janelamento do sinal de EMG ser de 500 ms com sobreposição ajustável;
- Programa de processamento e controle em linguagem C/C++;
- Realizar gesto de pegar-e-segurar com velocidade fixa máxima de 1,5 segundos
- Atraso entre recebimento do sinal de EMG para fechamento da mão e o inicio do gesto pegar-e-segurar menor ou igual 2 segundos.

Analizando os requisitos funcionais e não funcionais levantados, as limitações aplicadas pelo escopo geral e os objetivos gerais pode-se afirmar que a principal vantagem do projeto comparado com a solução existente de estado da arte é de ser um produto de baixo custo onde são aplicadas tecnologias semelhantes (controle por sinais de EMG).

C. Descrição de Hardware

O hardware necessário para o projeto é dividido em dois grupos: Mecânico, onde o principal é a prótese e acompanha

de todos os componentes mecânicos necessários para movimentação da mão; Eletrônico, que inclui todos os componentes necessários para movimentação da mão, captura e instrumentação do sinal de EMG, processamento e controle usando uma Raspberry Pi e fontes de energia para alimentar todos os componentes.

1) Hardware Mecânico:

- Dimensionamento e impressão 3D da prótese:

Primeiramente é necessário determinar as dimensões médias do braço de uma criança para escolha dos componentes e modelagem 3D da prótese final. Conforme amostragem realizada no desenvolvimento da prótese Cyborg Beast [9], gráfico da figura 8, podemos definir as dimensões aproximadas para o braço de uma criança com base em um adulto entre 16-20 anos.

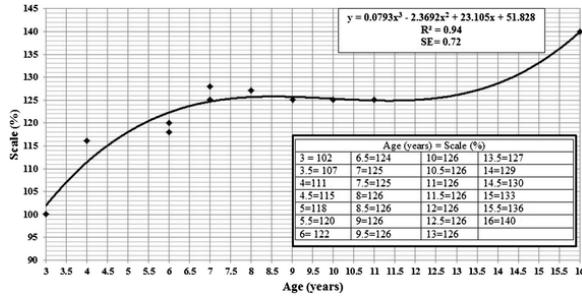


Figura 8: Gráfico mostrando a relação de dimensões da prótese Cyborg Beast em porcentagem conforme a idade [9]

Utilizando como base dos cálculos do gráfico na figura 8 obtemos os valores necessários para a prótese entre as idades almejadas do projeto com base nas dimensões de um dos participantes do projeto. Estas dimensões, tabela I, representam aproximadamente 140% dos tamanhos médios para uma criança entre 3-4 anos.

Tabela I: Dimensões medidas em um participante do projeto com idade de 20 anos, legenda 9

Dimensão	Medição Participante (cm)	Medida final Criança com 4 anos (cm)	Escala (%)
Comp. A	09,00	06,89	76
Comp. B	19,60	14,98	76
Comp. C	21,30	16,28	76
Comp. D	06,50	04,97	76
Área do corte transversal no comp. C	11,93	09,12	76
Área do corte transversal no comp. E	72,70	55,57	76

Conforme obtido na tabela I as dimensões para a prótese de uma criança com idade de 4-5 anos deve ser de aproximadamente 76% da dimensão de um adulto de 20 anos. Como o intuito deste projeto é demonstrar principalmente a implementação do sistema embarcado que realiza o processamento dos dados vindos do circuito instrumentação e controla os servomotores, não cabendo uma análise detalhada do desenho



Figura 9: Legenda dos comprimentos medidos na tabela I

e criação 3D do design de um braço completo, foi utilizado o projeto design de braço do projeto MyPo 2.0 [10] com escala reduzida para 75%.

A diferença de escala de 76% para 75% na impressão 3D final é devido as limitações de comprimento que a impressora 3D usada delimitava para as maiores peças. Abaixo temos a foto da prótese impressa em 3D em sua primeira versão realizada em outra disciplina, figura 10.

- Componentes mecânicos extras:

Para melhorar a aderência no procedimento de fechamento da mão são adicionados borrachas aderentes nas pontas e na peça proximal de cada dedo. Adicionalmente são usados elásticos de ligadura modular de látex empregados na ortodontia como ligamentos e articulações dos dedos na mão.

A movimentação dos dedos é feita a partir de atuadores tracionando os fios de linha localizados entre as articulações. Sendo assim, são utilizados fios de costura normalmente aplicadas para costura com couro profissional devido as suas propriedades de ótimo acabamento final e resistência à tração



Figura 10: Prótese em 3D para um criança entre 3-5 anos, versão V0 e com base no design MyPo 2.0 [10] (Trabalho realizado em outra disciplina)

mais elevadas, suficientes para suportar a força necessária para movimentação dos dedos realizada por um atuador. Na figura 11 temos uma foto da prótese mostrando em detalhes esses componentes mecânicos na mão.



Figura 11: Enfase das articulações e a borrachas nas pontas dos dedos

Como não existe espaço suficiente de se utilizar um atuador para cada dedo da mão é necessário adicionar um mecanismo que transfere toda a força de um atuador para os 5 dedos simultaneamente. Portanto, foi adicionado a prótese um mecanismo que transforma um atuador, como servomotores, em atuadores lineares. Na figura 12 temos uma foto da prótese mostrando em detalhes o mecanismo utilizado, foi impresso em 3D usando como material PLA branco.

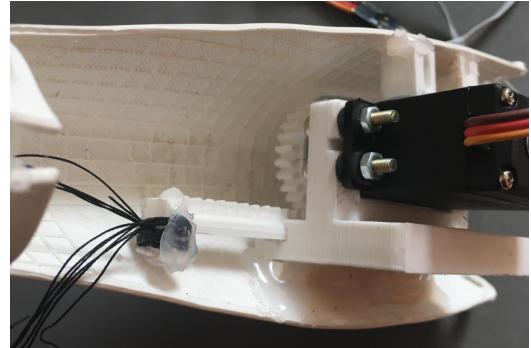


Figura 12: Detalhe do mecanismo ligado ao atuador e os fios de cada dedo

2) *Hardware Eletrônico:*

Para a parte de *hardware* da solução relacionada a eletrônica se segue o diagrama de blocos geral na figura 24 no apêndice A. Na figura 24 no apêndice A pode ser visualizado o diagrama de blocos geral da solução de *hardware* proposto para solução. Onde está demonstrado de forma geral tanto os principais componentes utilizados como a conexão entre eles.

Na tabela II, apêndice C, está a relação de todos os componentes utilizados para implementação do *hardware* descrito abaixo. Incluindo os custos totais com listagem de preço unitário, preço final (sem frete), e fabricantes onde foram consultados os preços (atualizado para data 14/05/2021).

• Circuito Condicionador:

Para aquisição dos sinal eletromiográfico será necessário a realização de um circuito analógico, o qual amplifica e filtra este com base em 3 pontos específicos que são medidos por eletrodos de contato. O circuito proposto está representado na figura 25, no apêndice A.

O circuito proposto é dividido em 6 subcircuitos da seguinte forma:

1) Proteção de entrada e Amplificador de instrumentação

O estágio de proteção de entrada tem como principal funcionalidade proteger o paciente de choques fatais durante a realização do exame de EMG. Seu funcionamento tem como objetivo utilizar diodos de pequenos sinais para deixar o sinal de baixa amplitude advindo dos eletrodos passar mas evitar que possíveis correntes advindas do amplificador operacional, devido alguma falha elétrica, voltem pelo eletrodo e atinjam o paciente. Enquanto que o estágio com amplificador de instrumentação tem como principal funcionalidade amplificar em grande escala sem adicionar ruído e componente DC o sinal advindo dos eletrodos, e os quais passaram pelo estágio de proteção. Este amplificador é ideal devido sua característica de ser um tipo de amplificador diferencial, tem saída como a diferença de dois sinais de entrada retirando o sinal DC, sem a necessidade da compatibilidade de impedância nos terminais + e -. Adicionalmente tem outras características como tensão DC offset muito baixa, pouca adição de ruído e um ganho de malha

fechada muito elevado, sendo está ultima a principal função deste estágio.

2) Amplificador Operacional

Esse estágio é formado por um amplificador operacional na configuração diferencial que tem como funcionalidade aumentar a amplitude do sinal de entrada em relação ao sinal de saída dependendo do valor do resistor.

3) Filtro Ativo Passa Alta

O estágio de filtro passa altas tem funcionalidade de filtrar o sinal advindo do estágio de amplificação anterior. Dessa forma possui o ganho na banda passante (H_0) de -1 e frequência de corte (fc) de 39,79 Hz.

4) Filtro Ativo Passa Baixa

Consequentemente, após o filtro as frequências estão mais altas que a faixa de operação típica de um sinal de EMG. Ou seja, teoricamente o intuito é filtrar todas as frequências acima de 600Hz, retirando assim ruídos e interferências que estão afetando o sistema. No caso do filtro projetado tem uma frequência de corte (fc) de 610,33 Hz e ganho na banda passante (H_0) de 2.

5) Integrador

É utilizado um integrador, uma vez que quando a tensão fixa for aplicada como entrada, a tensão de saída cresce sobre um período de tempo, fornecendo uma tensão em forma de rampa.

6) Saída - circuito de correção do Offset

O estágio final de saída requer a adição de resistores externos no terminal inversor para reduzir a tensão de offset. Entretanto, esse método depende do amplificador operacional e da precisão dos resistores que serão utilizados.

A lista dos materiais utilizada para o circuito proposto de aquisição do sinal de EMG implementado em placa de circuito impresso (PCB) para componentes de encapsulamento SMD está na tabela II, apêndice C.

Para verificação e aprovação da compra dos componentes acima foi realizado um teste do circuito utilizando o software LTSpice. Como as características de amplitude e frequência, assim como o grau de atenuação do sinal amplificado pelos filtros, são as mais importantes para verificação do circuito o mesmo foi testado utilizando um exemplo de sinal de EMG retirado do conjunto de dados de [11].

O sinal utilizado é oriundo de um dos 30 experimentos realizados no candidato 1 (masculino com 22 anos) segurando um objeto esférico apenas para teste com sinal real sem necessidade de validação utilizando todo o conjunto de dados da referência, especificamente canal 1 - experimento 1 - candidato "male-1". Este sinal foi normalizado para $V_{max} \approx 300mV$ e convertido como arquivo áudio utilizando MatLab para simulação no LTSpice. Além disso, o sinal de entrada no circuito, conforme representado na figura 25 localizado no apêndice A, foi condicionado para $V_{max} \approx 6mV$ e $V_{offset} \approx 10mV$ para simula-lo com a características de amplitude e deslocamento DC adquiridos superficialmente.

O gráfico da entrada V_{in} e saída após a passagem do filtro passa-banda $V_{out,lp1}$ na figura 13.

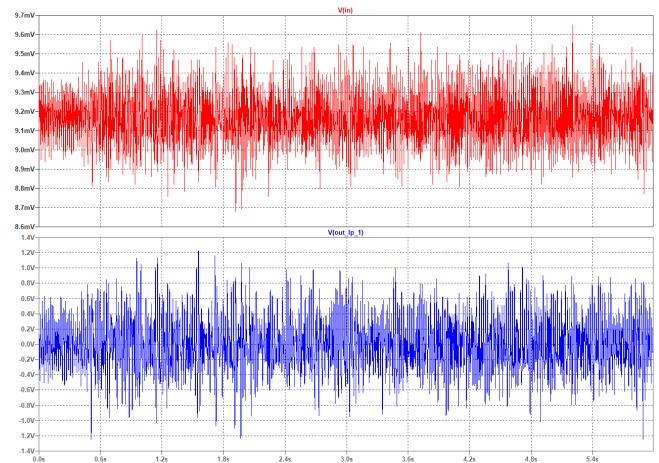


Figura 13: Resultado da simulação

• Conversor A/D:

O resultado do circuito condicionador é um sinal amplificado em tempo contínuo. Para ser possível o processamento do sinal resultante é necessário a conversão para um sinal em tempo discreto. Conforme os grandes especialistas na área de processamento sugerem o sinal de EMG tem componentes até a faixa de 1Khz sendo necessário sua captura com, no mínimo, 2kHz de taxa de amostragem e resolução 12 Bits. Entretanto, como filtra-se o sinal em todas as frequências acima de 600Hz, se utilizará um taxa de amostragem de 1600Hz, que é mais do que o suficiente para se adequar ao teorema de Nyquist. Como a Raspberry não inclui conversor analógico digital que cumpre com requisito de amostragem, resolução e compatibilidade de comunicação foi realizado a compra do módulo ADS1015, esquemático na figura 14 e representação física na figura 15. Este possui as seguintes características:

- Fabricado por Texas Instruments;
- Conversor AD de baixo consumo;
- 12 bits de resolução *noise-free*;
- Faixa de alimentação entre 2.0V a 5.5V;
- Pode operar entre 128 SPS à 3.3 kSPS (*samples per second* - amostras por segundo);
- Compatível com o protocolo I_2C - 4 pinos de interface;
- Oscilador interno e baixa tensão de *offset* interna;
- Tem um MUX interno que possibilita 4 entradas analógicas ou 2 entradas diferenciais;
- Entradas tem um faixa entre $\pm 256mV$ e $\pm 6.144V$;
- Inclui um amplificador de ganho programável e um comparador programável;
- Modos de conversão: Contínuo e *Single-Shot* (baixo consumo quando em repouso);

Entretanto o sinal de EMG analógico proveniente do circuito condicionador inclui componentes de tensão negativas, não sendo convertidas pelo conversor A/D por trabalhar apenas com tensões positivas. Com isso, é necessário adicionar um pequeno divisor de tensão ajustável por potênciometro tipo trimpot para adicionar DC *offset* ao sinal analógico entre 2,5

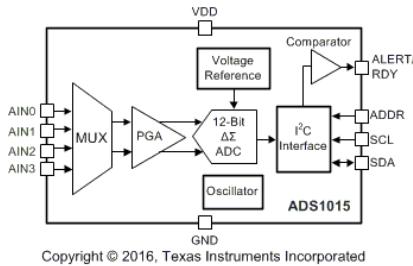


Figura 14: Esquemático do módulo ADS1015



Figura 15: Foto do módulo ADC ADS1015

V e 2,8 V.

• Sistema embarcado:

A funcionalidade do sistema embarcado é realizar pre-processamento digital do sinal advindo do conversor A/D, extrair características do sinal e predizer se o sinal de EMG é para mão aberta ou fechada e enviar o sinal de controle para atuador escolhido que realiza o movimento dos dedos da mão. Em especial, a etapa de extração de características e predição, onde se utiliza o SVM e explicada na seção II-D, requer poder de processamento elevado por ser realizado em tempo real junto ao pre-processamento e controle do atuador.

Como um dos objetivos da disciplina é demonstrar a implementação de sistemas operacionais embarcados onde se usa como exemplo a Raspberry Pi, o modelo escolhido para desenvolvimento inicial como prova de conceito da solução foi a Raspberry Pi 4 B com 4GBs de memória RAM. Esse modelo em específico foi escolhido por ter sido adquirido por um dos projetistas antes do desenvolvimento desse projeto e ter processador mais rápido que a Raspberry Pi 1, outro modelo que os projetistas já possuem que é equivalente a raspberry pi zero em nível de processamento.

Adicionalmente pode ser citado que esse projeto foi desenvolvido e implementado em um notebook com processador intel i7 e placa de vídeo em um disciplina anterior pois não rodou com velocidade mínima para funcionar em tempo real em uma Raspberry Pi 1.

O armazenamento deve ser suportado em hardware pela *Raspberry Pi*, sendo assim empregado o cartão de memória chamado em inglês de *Secure Digital Card* (SD Card). Dentro diversos modelos SD Card foi escolhido o **SanDisk Extreme** com **32 GBs** de armazenamento e especificação de leitura em até 100 MB/s e escrita de 90 MB/s. Essa escolha foi baseada em um comparativo de velocidade realizado pela imprensa especializada aliada ao preço em fornecedores locais [13].

O esquemático na figura 31, apêndice B, ilustra todas as

conexões realizadas na Raspberry Pi 4 e outros componentes como pinos para controle dos servomotores e conexão por I²C com o conversor A/D ADS1015.

• Atuador:

O atuador a ser escolhido precisa aplicar força mínima necessária para mover os cinco dedos de forma simultânea, sendo também compatível com as formas de alimentação disponíveis nos componentes selecionados (entre 3.3V e 8.6V). Também existem diversos tipos de atuadores como motor de passo, motor DC e servomotor onde cada um tem vantagens e desvantagens em relação ao outro.

O tipo de atuador escolhido foi o servomotor por não necessitar de circuitos extras para alimentação e controle, como ponte H para motor DC e *driver* específico para motor de passo. Neste caso, os ruídos elétricos gerados pela adição de mais circuitos elétricos são o maior preocupação devido a alta sensibilidade que o circuito condicionador tem devido ao sinal capturado ser de baixa amplitude de entrada (varia entre 50 μV a 9 mV).

Dentre os servomotores o escolhido foi o engrenagens metálicas MG995 por ser compatível com os requisitos de força e alimentação elétrica, representação real na figura 16. As especificações técnicas deste componente são descritas a seguir:

- **Peso:** 55 gramas
- **Dimensões:** 40.7 x 19.7 x 42.9 mm (aproximadamente)
- **Stall Torque:** 8.5 kgf.cm (4.8 V), 10 kgf.cm(6V)
- **Velocidade de Operação:** 0.2 s/60° (4.8 V), 0.16 s/60° (6V)
- **Tensão de Operação:** 4.8 V á 7.2 V
- **Largura de Banda Morta:** 5 μs

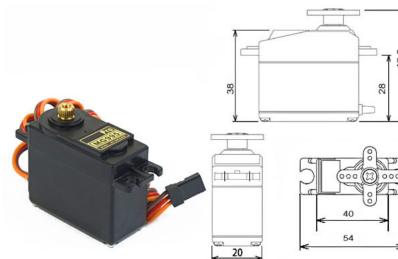


Figura 16: Servomotor MG995 com engrenagens metálicas

O esquemático na figura 30, apêndice B, ilustra as conexões realizadas para alimentação do atuador MG995. Incluindo os pinos de conexão que são conectados por conector P2 para ligar o atuador com a Raspberry Pi.

• Circuito Desacoplamento:

Existem ruídos que podem alterar e prejudicar os sinais adquiridos por meio do circuito condicionador, entre eles temos os gerados pela movimentação de motores elétricos. No caso dos servomotores esse ruído é de menor amplitude por serem atuadores que são construídos com um circuito interno de controle do motor DC que interpreta o sinal de controle por PWM para movimentação do motor. Este circuito de controle

interno normalmente contém algum tipo de isolamento elétrico porém ele não suficiente para evitar em interferência com o circuito condicionador caso estejam na mesma referência.

Dessa forma foi projetado um circuito de desacoplamento entre a referência de alimentação do servomotor e a referência entre os outros componentes como o circuito condicionador. O circuito proposto está representado na figura 26, apêndice A.

Seu funcionamento se baseia no uso de um optoacoplador para isolar as referências. Onde ele se utiliza de luz gerada por um fotodiodo para estimular um fototransistor em um caminho direto, assim isolando o circuito conectado ao fotodiodo e o circuito conectado ao fototransistor. A forma em que os componentes conectados faz com que o sinal de controle por PWM funcione mesmo quando as referências do servomotor e da Raspberry Pi não estão diretamente conectadas.

- **Alimentação:**

Para alimentação de todos os componentes deve ser respeitado a faixa de operação e potência consumida de cada um. Também deve se evitar o uso de fontes chaveadas que usam rede elétrica alternada 220/110VAC 50/60Hz pois essas fontes adicionam ruído nas frequências de 50/60Hz (ruído ambiente) no sinal de EMG capturado.

Sendo assim, serão usados 3 fontes de alimentação, onde a primeira é especificamente para a Raspberry Pi, outra para alimentar do restante dos circuitos e a última especificamente o servomotor devido necessidade referência exclusiva para o mesmo. A fonte de alimentação que alimenta a Raspberry Pi 4 é um *powerbank* comercial saída USB C 5V/3A da GeoNav e capacidade 20.000 mAh, modelo PB20KBK. A conexão entre essa fonte e a Raspberry Pi é feita por um cabo USB C macho/USB C macho.

Uma fonte de alimentação foi projetada para alimentar o circuito condicionador e o conversor A/D. Nela se utiliza 2 pilhas de lítio polímero 3,7V em paralelo conectada ao módulo elevador tensão ajustável XL6009 para elevar a tensão de 3,7V para 8,6V, tensão elétrica necessária para funcionamento correto do circuito condicionador. Como a faixa de operação do conversor A/D ADS1015 é entre 3V e 5V é utilizado um regulador de tensão 5V/1A LM7805 para regular a tensão 8,6V para 5V. Para recarregamento das pilhas de lítio sem necessidade de mover as pilhas é utilizado o módulo carregador bateria lítio TP4056, que carrega pilhas de lítio 3,7V aplicando 5V/1A em sua entrada micro USB.

A última fonte de alimentação foi projetada especificamente para faixa de operação do servomotor, ou seja, entre 4.8V e 7.2. Segundo o fabricante quanto maior é a tensão de alimentação do servomotor maior é o torque e a velocidade de operação portanto a saída dessa fonte é de 7,1 V. Ela se assemelha a fonte de alimentação anteriormente descrita, pilhas lítio e módulo XL6009, porém a saída do módulo elevador é regulado para 7,1V. O recarregamento é feito da mesma forma usando o módulo TP4056.

D. Software

Para desenvolvimento inicial da primeira versão foi utilizada a linguagem python 3 e o framework Robot Operating System (ROS) pois foram utilizadas anteriormente para desenvolvimento da primeira versão desse projeto implementada em PC (notebook), assim possibilitando o desenvolvimento e depuração rápida de cada parte do código para versão embarcada.

Para ter maior compatibilidade as últimas versões das bibliotecas do Python 3 e do ROS, o Noetic Ninjemys, foi instalado como sistema operacional o Ubuntu Server 20.04 para a Raspberry Pi4.

O framework ROS utiliza um sistema de nós e tópicos, os nós são programas independentes que são conectados por tópicos de dados. Na figura 28, apêndice A, temos o diagrama completo de todos os nós e tópicos que os conectam.

Para a segunda e terceira versão foi utilizado as linguagens C e C++ e utilizado o MQTT, sigla do inglês *Message Queueing Telemetry Transport*, para substituir o ROS por possuir estrutura semelhante de tópicos em que os dados são publicados. A escolha de uso do MQTT foi devido a ser um dos tópicos especiais ensinados na disciplina e, com implementação correta, uma alternativa mais leve para integração entre os subprogramas.

As etapas gerais que um ciclo de processamento segue está representado no fluxograma na figura 27, apêndice A.

1) *Programa ADC*: O programa foi escrito em C e tem a função de capturar os dados do conversor A/D ADS1015 são capturados e enviados ao tópico dados. O código fonte utilizado está disponível na subseção D-A do apêndice D.

Pseudo-código:

- Abrir conexão I²C com *baudrate* de 500000;
- Declarar tópico "dados"em que os dados recebidos serão publicados;
- Estipular frequência de publicação para 1600Hz com *signal* e *ualarm*;
- Loop até o programa ser desligado:
 - Enviar byte de comando e receber dados I²C;
 - Converter inteiro em string;
 - Publicar valor no tópico "dados".

2) *Programa predictor*: O programa foi escrito em C++ e possui as funções de acumular uma janela de dados, extrair as características dessa janela e utilizá-las para classificar o sinal em aberto e fechado. O código fonte utilizado está disponível na subseção D-B do apêndice D.

Pseudo-código:

- Declarar vetor de tamanho 800 para dados;
- Declarar tópico "*prediction*"em que a classificação será publicada;
- Declarar tópico "dados"que será lido pelo programa;
- Carregar SVM por meio de xml;
- Função quando dados são recebidos:
 - Descarta a primeira posição do vetor e desloca o vetor;
 - Atribui o valor recebido a ultima posição do vetor;
 - Soma no contador;

- Se o contador atingir metade da janela:
 - * Extrai características da janela
 - * Classifica as características.
 - * Publica resultado da classificação no tópico *prediction*.
- Caso contrário, soma um ao contador.

3) *Função de extração de características*: Esta função é responsável por extrair características do sinal recebidas e está contida dentro do programa preditor. As seguintes características são extraídas:

- Valor médio absoluto

$$\sum_n^N \frac{|x|}{N} \quad (1)$$

- Valor RMS

$$\sqrt{\sum_n^N \frac{x^2}{N}} \quad (2)$$

- Variância

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (3)$$

- Média de mudança de amplitude

$$\sum_n^N \frac{|x_n - x_{n-1}|}{N} \quad (4)$$

- Comprimento de forma de onda

$$\sum |x'| \quad (5)$$

- Energia em bandas

Compuestos por três filtros digitais que dividem as frequências do sinal em três bandas e determinam sua energia.

Essas características são então retornadas ao código.

Pseudo-código:

- Declarar vetor *float* com parâmetros a serem extraídos
- Função quando dados são recebidos;
- Retirar valor DC do sinal;
- Calcular parâmetros II-D3;
- Retornar parâmetros.

4) *Treinamento SVM*: Para treinar a SVM foram usados arquivos CSV com os dados obtidos. O código fonte utilizado está disponível na subseção D-C do apêndice D.

Pseudo-código:

- Carregar CSV com matriz de dados e etiquetas;
- Declarar matriz de parâmetros;
- Para todos os vetores de dados:
 - Extrair características;
 - Atribuir a coluna na matriz de parâmetros;
- Treinar SVM com matriz de parâmetros;
- Salvar SVM em XML.

5) *Programa de controle*: Esse programa foi escrito em C e tem a função de controlar o servomotor da prótese, abrindo e fechando a mão por meio dos ângulos de 60 graus e -35 graus, respectivamente. O código fonte utilizado está disponível na subseção D-D do apêndice D.

Pseudo-código:

- Declarar tópico "prediction" que será lido pelo programa;
- Declarar o pino GPIO que o sinal de controle PWM é gerado;
- Declarar ângulos máximo e mínimo necessários para fechar e abrir a mão;
- Declarar características do sinal de controle PWM a ser gerado;
- Inicializar configuração do GPIO por acesso a registradores;
- Inicializar o pino GPIO especificado para modo de saída;
- Função quando algum dado é recebido no tópico (*control_callback*):
 - Caso o dado recebido seja inteiro '1' o servomotor é movimentado para ângulo mínimo (abrir mão);
 - Caso o dado recebido seja inteiro '-1' o servomotor é movimentado para ângulo máximo (fechar mão);
 - Caso o dado recebido seja inteiro '3' o servomotor é movimentado para ângulo médio de 0° (mão meio fechada);
- Função movimentar servomotor para ângulo de entrada (*dutyCyclechange*):
 - Calcula *duty_cycle* do sinal PWM nas variáveis t1 e t2;
 - Gera onda quadrada com os tempos t1 e t2 calculados com sequencia "10" aplicando nos registradores especificados;

E. Execução automática dos programas

Para funcionamento autônomo da prótese é necessário que todos os programas descritos acima sejam executados quando a Raspberry Pi é ligada. Para isso, foi implementado um script bash que é acionado periodicamente pelo agendador crontab. O código fonte utilizado está disponível na subseção D-E do apêndice D.

O crontab checa se o PID gravado, específico para o *script* de ativação automática, está ativo no sistema. Se não estiver ativo ele executa o *script*, assim ativando todos os programas necessários para processamento e controle da prótese.

III. RESULTADOS

A. Circuito condicionador

O circuito condicionador foi implementado em uma placa de circuito impresso (PCB), versão em 3D na figura 18 e versão real na figura 19. Como foi uma implementação realizada na disciplina de Instrumentação Eletrônica antes da pandemia foi possível realizar testes e validar o funcionamento do circuito implementado no osciloscópio. Eles revelaram que o circuito realiza a amplificação do sinal de entrada, assim como reduz o ruído existente. Na figura 17 demonstra o teste realizado no

osciloscópio, onde temos como entrada uma senóide $V_{pp} = 21,6$ mV a frequência de 75,30 Hz e como saída uma senóide amplificada e filtrada para $V_{pp} = 6,6$ V a frequência 75,19 Hz.

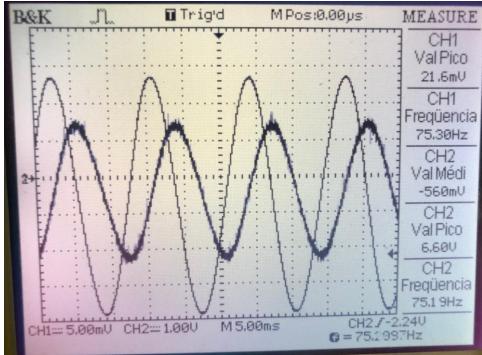


Figura 17: Teste do circuito com uma entrada senoidal

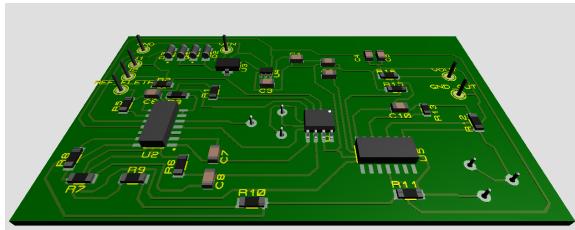


Figura 18: Resultado final da PCB no Proteus

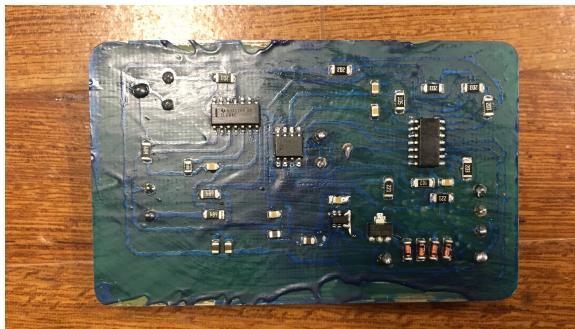


Figura 19: Resultado final da PCB

B. Conversor A/D

Para obter os sinais de EMG com o módulo ADS1015 é necessária uma conexão I²C de alta frequência para atingir a frequência de amostragem de 1600Hz. Entretanto, o código implementado na primeira versão em python 3 concordava com outros processos de processamento, não alcançando a frequência necessária para o critério de Nyquist. Neste caso a frequência de amostragem quando se rodava apenas esse nó foi de aproximadamente 1430 Hz e com os outros nós de processamento e controle abaixava para frequência de aproximadamente 1000 Hz. A frequência de amostragem de

um sinal de EMG abaixo do critério de Nyquist causou *aliasing* e impediu o processamento e predição adequada do sinal.

Para solucionar o problema foi utilizado um Arduino Pro mini para capturar os sinais no conversor A/D e enviá-los ao sistema embarcado por meio de protocolo UART. Com isso, a frequência de amostragem atingiu o valor estipulado de 1600 Hz, adquirindo sinais como o representado na figura 20.

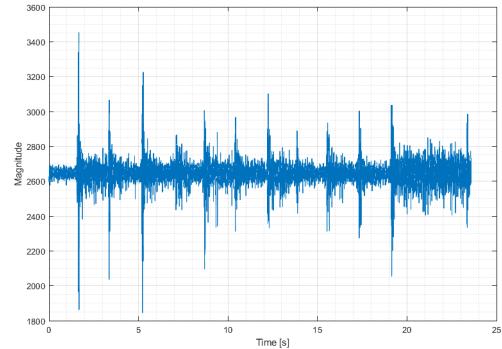


Figura 20: Visualização de sinal de EMG com $f_s = 1600\text{Hz}$ utilizando o eletrodo de superfície descartável - Versão 1 com microcontrolador e comunicação serial UART

No caso do código para captura dos dados diretamente do conversor A/D via I²C, implementado em C, a velocidade de aproximada de 1600Hz foi atingida quando rodando os outros programas simultaneamente. Sendo assim foi removido a necessidade de usar um microcontrolador Arduino Pro mini como necessário na primeira versão. Porém foi observado outro erro onde o sinal capturado era diferente do esperado, ocasionando a falha na predição do processamento e não funcionamento na segunda versão ao integrar os outros programas.

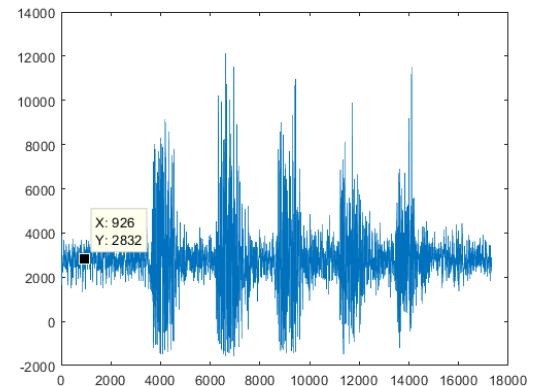


Figura 21: Visualização de sinal de EMG com erros no conversor A/D - Versão 2 com comunicação I²C.

Na terceira versão foram corrigidos os erros de captura do

sinal e um exemplo de sinal capturado pode ser observado na figura 22.

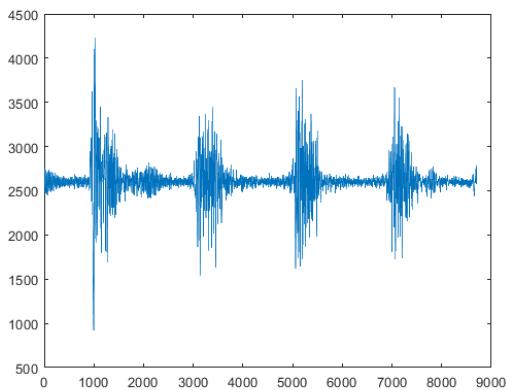


Figura 22: Visualização de sinal de EMG sem erros no conversor A/D - Versão 3 com comunicação I²C.

Os erros na captura solucionados na versão 3 foram os seguintes:

- 1) Mal contato na conexão da referência (*Ground*) entre a Raspberry Pi e os circuitos de captura;
- 2) Local onde estava sendo realizados os testes com ruído elevado na frequência de 60 Hz (advindo de um carregador indução para celular próximo a prótese durante os testes);
- 3) Erro no código de captura dos dados por comunicação I²C:
 - Necessário deslocar o valor dos bytes recebidos em 4 para descartar os de teste (4 menos significativos).

C. Processamento de Sinais

Para implementação dos nós de processamento de sinais (accumulator, extraction e predictor) foi necessário atualizar bibliotecas da versão por python 2 previamente utilizada para python 3. Consequentemente foi necessário gerar novamente o arquivo de treinamento de SVM para python 3, devido a não funcionar por python 2.

Para testar e validar que o código atualizado e novo arquivo de treinamento de SVM gerado funcionam como anteriormente foi aplicado um sinal de EMG previamente adquirido e gravado no framework *ROS* em um arquivo .bag. Durante o teste é solicitado no terminal a visualização do tópico *servo/action*, observando se os resultados estão condizentes com o esperado ('1' caso mão aberta e '-1' caso mão fechada).

Utilizando os dados obtidos dos integrantes foi possível treinar uma *SVM* com os dados extraídos. Após rodar um teste de classificação baseado em k-fold foi obtida uma acurácia de 90.13% e uma precisão de 91.54% para versão em Python e uma acurácia de 90.82% e uma precisão de 92.11% para versão em C/C++. Essa diferença vêm da diferença entre a metodologia que o Python e o OpenCV em C++ usam para gerar o modelo SVM treinado para base de dados utilizada.

Para que esses resultados melhorem, se torna necessária a utilização de mais sinais para treino.

D. Controle do Servomotor

Para teste e validação do código implementado primeiro temos a necessidade de validar se o sinal PWM gerado condiz para controle de um servomotor, teste realizado usando microservo 9g SG90. Esse teste é necessário fazer uma pequena alteração no código do nó *control* comentando a leitura do tópico *servo/action* e descomentando o *Node loop*. Nesse código é requisitado qual ângulo que se deseja movimentar o servo (entre 0 e 180 graus), é aplicado, solicita novo ângulo após servomotor ter sido movimentado. Os ângulos testados foram de ordem crescente e decrescente de 0-180 graus, avaliando com o ângulo gerado no servomotor.

Após validação que os parâmetros de geração do PWM funcionam é restaurado a funcionalidade anterior do tópico. No segundo teste é validado se o nó aplica os ângulos máximos e mínimos ao receber respectivo dado no tópico *servo/action*. Foram feitos diversos envios de dados para o tópico e o comportamento do servomotor é compatível com a aplicação.

O terceiro teste integra o nó de controle com os nós de processamento de sinais, aplicando um sinal de EMG previamente adquirido conforme descrito em um dos testes dos nós de processamento descritos na seção III-C. O resultado desse teste cumpriu o requisito de, a partir de um sinal de EMG, o código conseguiu processar e controlar o servomotor para posição de angulo que a mão é fechada ou aberta.

Após confirmação que todas as rotinas de controle funcionam conforme esperado foi substituído o microservo 9g SG90 pelo o servomotor que será utilizado e foi previamente instalado na prótese, servomotor MG995. Todos os 3 testes anteriores foram repetidos para garantir que os ângulos previamente obtidos funcionam para o servomotor definitivo alimentado com tensão maior de 7.1 V, observando a movimentação dos dedos da mão ao invés do ângulo. Durante os testes não foram encontrados problemas mas foi alterado o ângulo mínimo de 45° pra 60° para evitar a aplicação de força excessiva pelo servomotor.

E. Circuito de Desacoplamento

Para simulação e validação que o circuito funciona foi utilizado o PROTHEUS 8.9 que inclui internamente um simulador de microcontroladores Arduino 328p. Usando na simulação do circuito proposto na figura 26 no apêndice e código exemplo compilado para controle de servos *Sweep* foi possível observar a movimentação do servomotor simulado entre os ângulos de 90° e -90°.

Para teste e validação do circuito primeiro é necessário validar se o sinal PWM aplicado na entrada gera a mesma resposta na saída, ao isolar as referências pelo uso do optoacoplador. Neste caso, foi montado o circuito usando os seguintes componentes:

O primeiro teste foi montado conforme circuito da figura 26, nos apêndices, sendo conectado o pino de saída do PWM (GPIO 17) da Raspberry Pi na entrada, saída o pino de dados

do servomotor SG90, e usado optocoplador EL817. Para alimentação é conectado entrada positiva +5V e GND da raspberry pi no lado em que o sinal é gerado e fonte de 5V/1A no lado em que o servomotor recebe os dados.

O resultado do teste foi falho e o servomotor não se movimentava. Foi adicionado o led vermelho em série com resistor de $1k\Omega$ no lugar do servomotor, observando se a intensidade de cor mudava e indicando que estaria funcionando. O novo teste usando led foi observado mudança de intensidade porém não funcionou trocando pelo servomotor novamente. Em seguida, foram realizados diversos testes trocando o optoacoplador, o transistor e os resistores em ambos os lados não obtendo sucesso.

Para o próximo ponto de controle será testado novamente o circuito porém analisando a entrada e saída em um osciloscópio que os integrantes tem acesso ocasional. Caso não seja obtido sucesso será removido o uso desse circuito da montagem final pois não seu uso não é necessário para funcionamento do projeto. Devido a forma que os componentes são montados atualmente, em que servomotor e o circuito condicionador são alimentados por fontes de alimentação diferentes, o ruído é mínimo e o preditor está funcionando de forma normal.

F. Integração

Com todos integração dos componentes foram todos conectados entre seguindo o diagrama de bloco geral, figura 24 nos apêndices, obtendo como resultado o sistema na figura 23.



Figura 23: Sistema Quiro com todos os componentes integrados

Para teste de integração todo o sistema é ligado e são colocados e conectados os eletrodos na posição do antebraço, conforme explicado anteriormente e visualizado na figura 7. Em seguida, é ativado por terminal SSH o programa de processamento e controle na Raspberry Pi e realizado uma série de movimentos de abertura e fechamento da mão com o braço apoiado em uma superfície para evitar geração de outros sinais de EMG que não estão relacionados a movimentação da mão.

O teste de integração foi aplicado em ambos integrantes desse projeto, adultos do sexo masculino de 22 e 23 anos, e

observado o bom funcionamento caso eletrodos forem colocados recentemente e na posição correta. No caso em que são usados eletrodos reutilizados é observado a ocorrência de comportamentos falso-positivo no caso de fechar a mão. Primeiro é realizado um teste apenas com a abertura e fechamento da mão em intervalos de 4 segundos e, em seguida, testes para segurar uma bola com diâmetro de 40mm e 4g de peso.

No caso da primeira versão usando Python 3 e o ROS foi obtido sucesso no funcionamento da prótese por sinal de EMG e na segundo versão, em C, C++ e MQTT, não funcionou devido erro anteriormente descrito na captura dos dados do conversor A/D. Na terceira versão, onde foram corrigidos os erros da versão 2, foi obtido sucesso no funcionamento da prótese.

G. Comparação de Performance

Com o sucesso na transferência do código de Python/ROS para C/C++/OpenCV foi realizado um teste comparando o uso de recursos e melhorias de performance entre as duas versões. O teste utilizado foi igual a metodologia do teste de integração anteriormente descrito, onde é observado as seguintes características: Frequência de operação do programa de captura dos dados do conversor A/D, total de memória RAM em uso pelo Raspberry Pi, média entre do uso de *thread* da CPU da Raspberry Pi, atraso do processamento (diferença em milisegundos entre fechar/abrir a mão do aluno e o momento que o servomotor ativa para fechar/abrir a mão da prótese).

- Sistema em Python

- **Frequência do conversor A/D:** Variação entre 1560 Hz e 1600 Hz;
- **Uso total de Memória RAM:** 452 MB;
- **Atraso do Processamento:** Entre 700 ms a 2000 ms (dependendo do gesto - gesto de abrir mais demorado);

- Sistema em C/C++

- **Frequência do conversor A/D:** Constante em 1600 Hz;
- **Uso total de Memória RAM:** 203 MB;
- **Atraso do Processamento:** Entre 400 ms a 1000 ms (dependendo do gesto - gesto de abrir mais demorado)

No caso do atraso do processamento, é utilizado a quantidade quadros, em um vídeo de 30 segundos a 120 fps, para calcular o atraso entre o momento que o aluno fecha/abre a mão e a ativação do servomotor para executar o gesto correspondente.

Comparando os dados fica claro que a implementação em C/C++ é mais eficiente. Pois foi retirada a necessidade de um microcontrolador para capturar dos dados do conversor A/D, tornando a frequência constante, diminuiu em mais pela metade o uso de memória RAM e ocorreu uma diminuição em quase 100% para atraso do processamento.

H. Teste de Equivalência

O último teste realizado é para verificar a possibilidade de troca da Raspberry Pi 4 B por uma Raspberry Pi Zero. Essa troca é interessante para aplicação pois a Raspberry Pi Zero é muito menor, tornando o sistema mais portátil.

Para esse teste foi utilizado uma Raspberry Pi 1 pois se tem o mesmo processamento da Zero e os alunos já tinham disponível entre seus componentes. Realizando o teste completo, igual o de integração, o sistema não funcionou na Raspberry Pi 1.

Verificando cada processo foi observado que o programa de captura dos dados A/D roda a frequência média de 400Hz. Sendo uma amostragem muito menor que o mínimo recomendado pela frequência de Nyquist, distorcendo o sinal de EMG e tornando impossível sua predição

IV. CONCLUSÕES

Foi possível observar que o desenvolvimento do sistema de forma completamente embarcada no raspberry pi é viável. Entretanto é necessário o uso de linguagem C para otimizar a captura de dados e todo o processamento no sistema, pois a concorrência de recursos de processamento no sistema entre os programas escritos em python atrapalharam a frequência de amostragem tornando o processamento inviável.

Todos os programas antes implementados em python 3 foram bem sucedidos na tradução para C e C++ com uso do opencv para SVM. Após correção dos erros da versão 2 foi implementado com sucesso uma versão funcional em C e C++ para a Raspberry Pi 4 atendendo as funcionalidades propostas.

Considerando os testes de performance, foi demonstrado que a implementação em C/C++ é mais eficiente. Porém não o suficiente para rodar em sistemas com menor potência de processamento e maior portabilidade, como uma Raspberry Pi Zero.

Para trabalhos futuros é interessante a expansão dos tipos de movimento que a prótese pode fazer; Sendo necessário um novo projeto mecânico que comporte mais motores para controle individual dos dedos da mão. Em relação ao *software*, a implementação de novos gestos requer mínimo de refatoração de código porque o uso da SVM se demonstrou viável como preditor de movimentos. Sendo necessário o uso de mais sinais de eletromiografia para aplicações mais avançadas e com mais tipos de movimentos.

Os códigos finais implementados estão disponíveis para consulta em [Repositório GitHub - Quiro - embedded_emg_prosthesis](#).

REFERÊNCIAS

- [1] PARATLETISMO, Braskem. Evolução das próteses na Linha do Tempo. Disponível em: <<https://www.braskem.com.br/paratletismo-infografico>>. Acesso em: 22 fev. 2021.
- [2] Morris, Beverly A., et al. e-Knee: evolution of the electronic knee prosthesis: telemetry technology development. JBJS 83.2_{suppl 1} (2001): S62-66
- [3] Lei Brasileira 13.146 de Inclusão da Pessoa com Deficiência. Disponível em: <<https://www2.camara.leg.br/legin/fed/lei/2015/lei-13146-6-julho-2015-781174-normaactualizada-pl.html>>. Acesso em: 22 fev. 2021
- [4] OTTOBOCK. Empresa de próteses. Próteses de Membro Inferior. Disponível em: <<https://www.ottobock.com.br/prosthetics/membros-superiores/>>. Acesso em: 22 fev. 2021.
- [5] OTTOBOCK. Empresa de próteses. Catálogo de Próteses de Membro Inferior (inglês). 2020. Disponível em: <<https://shop.ottobock.us/media/pdf/UpperLimbCatalogue20192020.pdf>>. Acesso em: 12 maio 2021.
- [6] IBGE. Censo Demográfico 2010. <Disponível em: <http://www.ibge.gov.br>> Acesso em 23 fev. 2021.
- [7] ABOTEC. Associação Brasileira de Ortopedia Técnica. Avaliação de acessibilidade da população para próteses. Disponível em: <<http://www.abotec.org.br/novosite/index.html>>. Acesso em: 23 fev. 2021.
- [8] DUFF, Susan V. et al. Innovative evaluation of dexterity in pediatrics. Journal of Hand Therapy, v. 28, n. 2, p. 144-150, 2015.
- [9] ZUNIGA, Jorge et al. Cyborg beast: a low-cost 3d-printed prosthetic hand for children with upper-limb differences. BMC research notes, v. 8, n. 1, p. 10, 2015.
- [10] Myo & PO, MyPo 2.0 . "Disponível em: <<https://www.thingiverse.com/thing:2409406>>". Acesso em 23 fev. 2021 .
- [11] C. Sapsanis, G. Georgoulas, A. Tzes, D. Lymberopoulos, e.g. Improving EMG based classification of basic hand movements using EMD in 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society 13 (EMBC 13), July 3-7, pp. 5754 - 5757, 2013. "Disponível em: <<https://archive.ics.uci.edu/ml/datasets/EMG+for+Basic+Hand+movements>>". Acesso em 24 fev de 2021 .
- [12] ALKAN, Ahmet; GÜNEY, Mücahit. Identification of EMG signals using discriminant analysis and SVM classifier. Expert systems with Applications, v. 39, n. 1, p. 44-47, 2012.
- [13] DRAMBLE, R. P. microSD Card Benchmarks. 2020. [Online; Acessado em 01 março 2021]. Disponível em: <<https://www.pidramble.com/wiki/benchmarks/microsd-cards>>.
- [14] P. Wattanasiri, P. Tangpornprasert and C. Virul Sri, "Design of Multi-Grip Patterns Prosthetic Hand With Single Actuator," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 26, no. 6, pp. 1188-1198, June 2018, doi: 10.1109/TNSRE.2018.2829152.

APÊNDICE A

IMAGENS

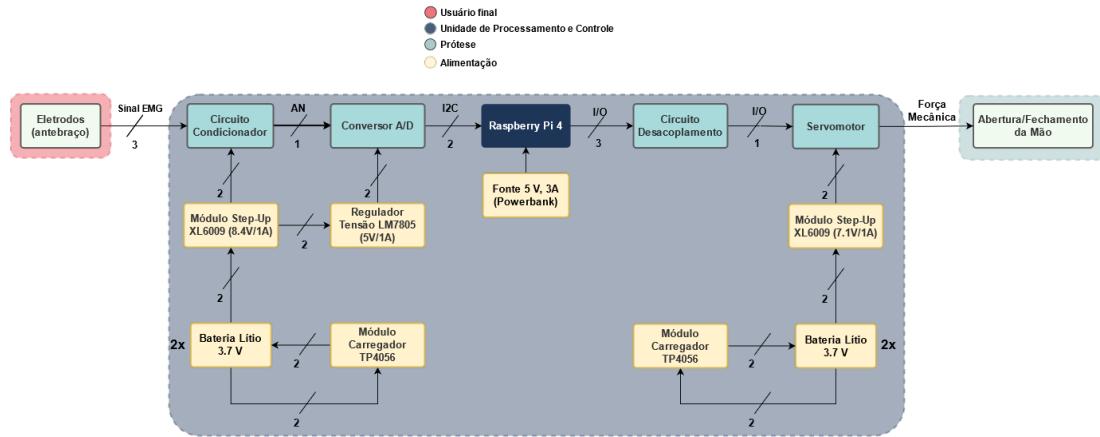


Figura 24: Diagrama de Blocos da Solução

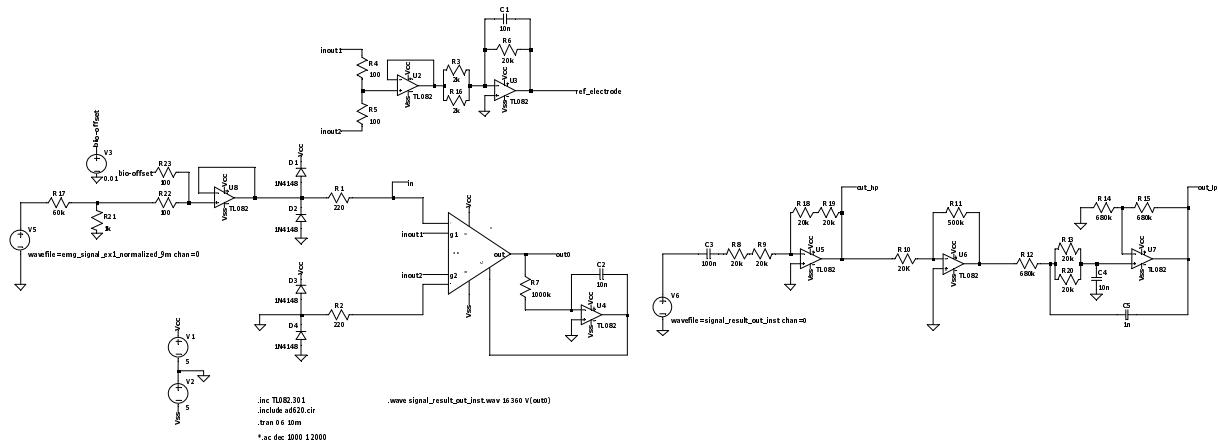


Figura 25: Circuito de captura e condicionamento proposto - LTSpice

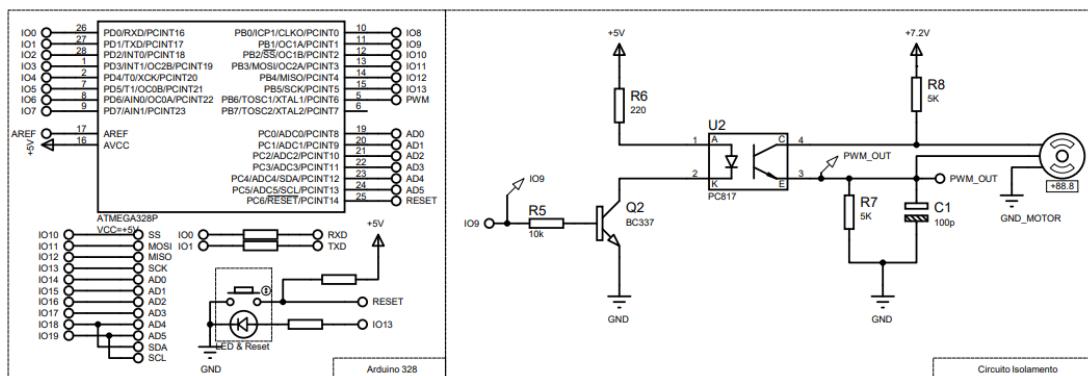


Figura 26: Circuito de desacoplamento proposto

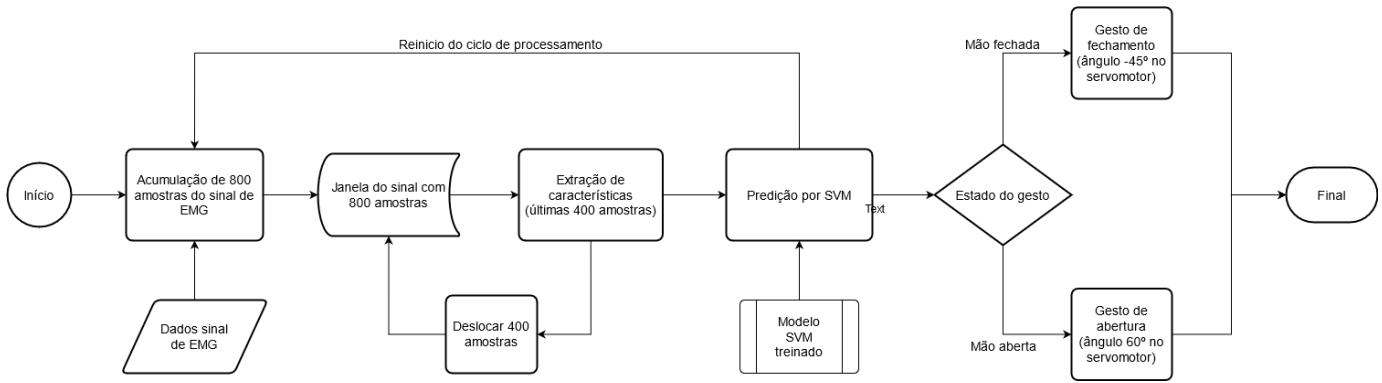


Figura 27: Fluxograma do processamento e controle



Figura 28: Diagrama de nós e conexões do projeto - Versão ROS

APÊNDICE B ESQUEMÁTICOS

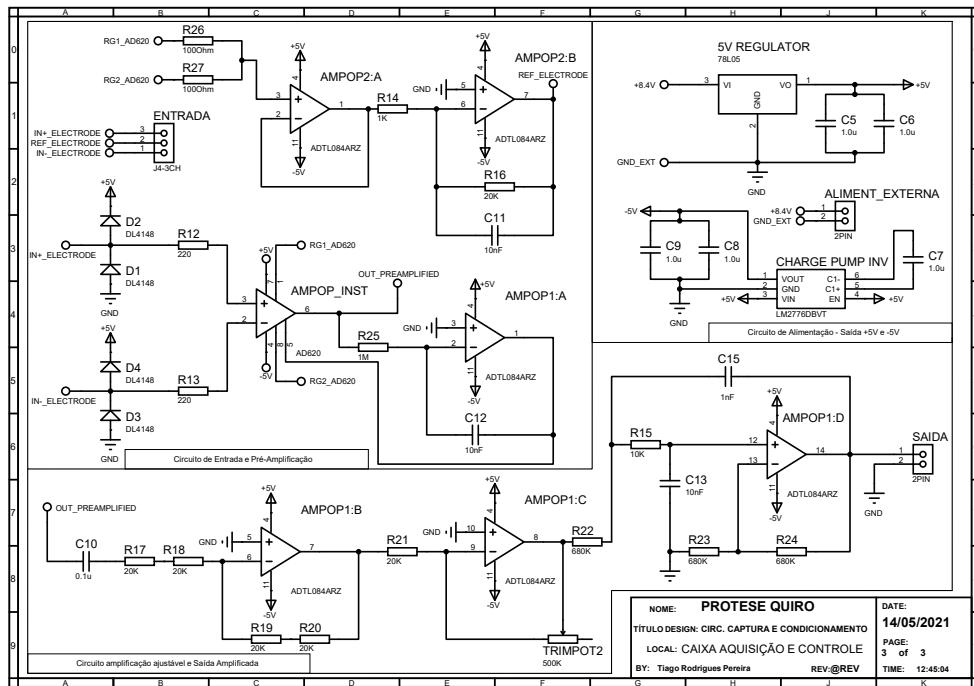


Figura 29: Esquemático 1 - Circuito de Aquisição e Condicionamento do Sinal de EMG

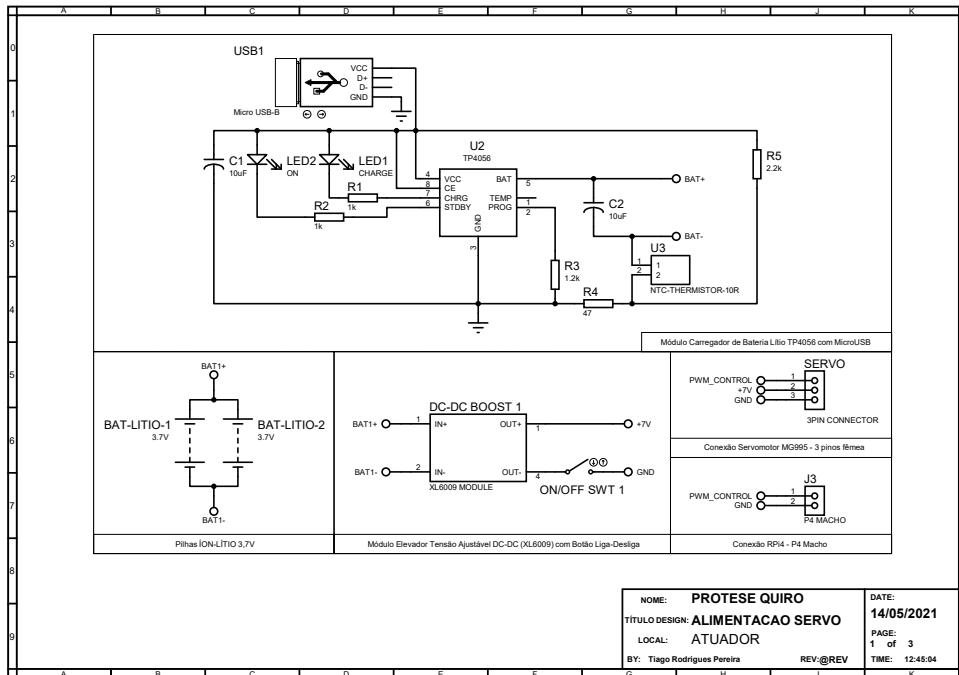


Figura 30: Esquemático 2 - Alimentação e Conexão do Servomotor da Prótese

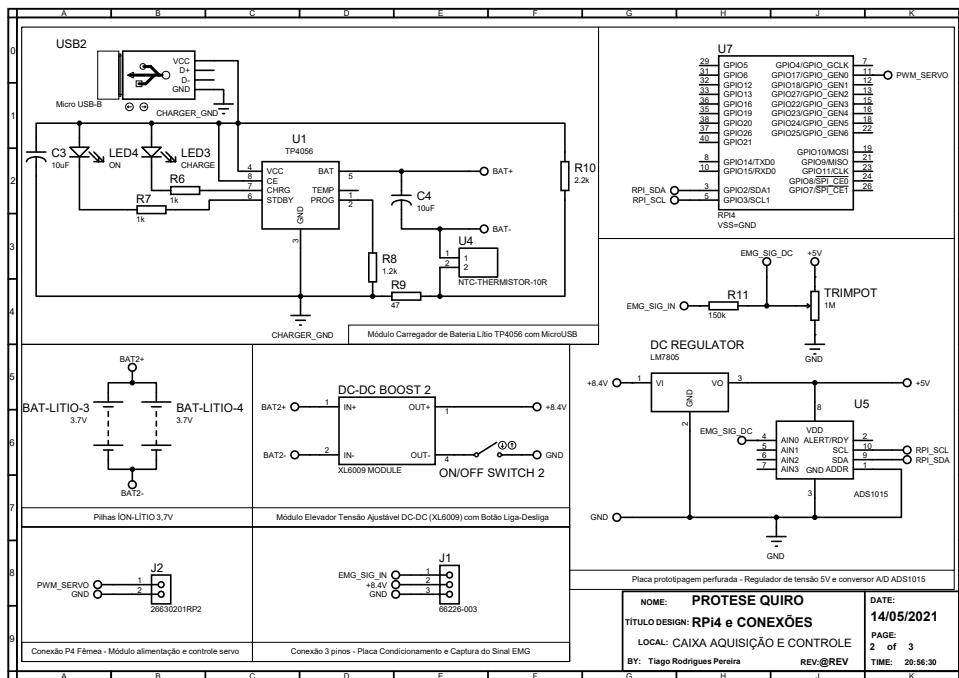


Figura 31: Esquemático 3 - Alimentação do conversor A/D, Adição DC ao sinal EMG capturado e conexão da RPi4 com componentes

APÊNDICE C
LISTA DE MATERIAIS (BOM)

Tabela II: Custos Totais

ID	Componente	Modelo	Fornecedor	Qtd.	Preço R\$	
					Unitário	Final
C1-C4	Capacitor 10uF	Cerâmico - SMD - 50V - MLCC	Mercado Livre	4 (10)	-	R\$ 18,90
C5-C9	Capacitor 1uF	Cerâmico - SMD - 50V - MLCC	Mercado Livre	4 (100)	-	R\$ 18,50
C10	Capacitor 100nF	Cerâmico - SMD - 50V - MLCC	Mercado Livre	1 (100)	-	R\$ 18,50
C11-C13	Capacitor 10nF	Cerâmico - SMD - 50V - MLCC	Mercado Livre	3 (10)	-	R\$ 18,50
C15	Capacitor 1nF	Cerâmico - SMD - 50V - MLCC	Mercado Livre	1 (10)	-	R\$ 7,75
R12-R13	Resistor 220Ω	SMD - 5% - 1/4W	Mercado Livre	2 (10)	-	R\$ 10,88
R14	Resistor 1KΩ	SMD - 5% - 1/4W	Mercado Livre	1 (100)	-	R\$ 30,00
R15	Resistor 10kΩ	SMD - 5% - 1/8W	Mercado Livre	1 (100)	-	R\$ 11,90
R16-R21	Resistor 20kΩ	SMD - 5% - 1/4W	Mercado Livre	6 (100)	-	R\$ 11,90
R22-R24	Resistor 680kΩ	SMD - 5% - 1/4W	Mercado Livre	3 (5)	-	R\$ 7,91
R25	Resistor 220Ω	SMD - 5% - 1/4W	Mercado Livre	1 (10)	-	R\$ 10,88
R26-R27	Resistor 100Ω	SMD - 5% - 1/4W	Mercado Livre	2 (100)	-	R\$ 22,87
TRIMPOT2	Resistor Variável Linear 500k	Trimpot Horizontal - 3361P-504	HU Infinito	1	R\$ 1,45	R\$ 1,45
AMPOP_INST	Amp. Instrumentação	INA AD620BRZ - SMD	Mercado Livre	1	R\$ 19,50	R\$ 19,50
AMPOP1-AMPOP2	Amp. Operacional	ADTL084ARZ - SMD	Mercado Livre	2	R\$ 10,00	R\$ 20,00
D1-D4	Díodo de Silício	DL4148-TP - SMD	Mouser (I)	4	R\$ 3,18	R\$ 12,72
5V REGULADOR	Regulador DC 5V	78L05 - SMD - 5V/100mA	Mercado Livre	1 (20)	-	R\$ 21,78
CHARGE PUMP INV	Inversor Comutado a Capacitor	LM2776 - SMD	Texas Instruments (I)	1	R\$ 4,21	R\$ 4,21
-	Placa de cobre Virgem para PCB	Fibra de Vidro - Dupla Face - 200x200mm	HU Infinito	1	R\$ 26,17	R\$ 26,17
Circuito de Aquisição e Condicionamento - Esquemático 1						R\$ 294,32
RPI4	Microprocessador embarcado	Raspberry Pi 4 B 4Gbs	FilipeFlop	1	R\$ 639,99	R\$ 639,99
-	SD Card	SanDisk 32 GBs Extreme	Mercado Livre	1	R\$ 58,50	R\$ 58,50
-	Dissipador ativo RPi4	Alumínio - Preto - 2 fans 5V	Mercado Livre	1	R\$ 129,90	R\$ 129,90
-	Carregador portátil para RPi4	Geonav - 16000mAh - PB16KWT	Kabum	1	R\$ 103,55	R\$ 103,55
RPi e Acessórios - Esquemático 3						R\$ 931,94
U5	Conversor A/D	Módulo ADS1015 - 12Bits - 3300bps	Mercado Livre	1	R\$ 42,79	R\$ 42,79
R11	Resistor 150kΩ	Filme de Carbono 5% 1/4W	Mercado Livre	1	R\$ 0,04	R\$ 0,04
TRIMPOT	Resistor Variável Linear 1M	Trimpot Horizontal F105-124C	HU Infinito	1	R\$ 1,43	R\$ 1,43
-	Placa perfurada para prototipagem	Fibra de Vidro - Dupla Face - 40x60mm	HU Infinito	1	R\$ 1,90	R\$ 1,90
Conversor A/D e Adiciona DC - Esquemático 3						R\$ 46,16
DC-DC BOOST 1-DC-DC BOOST 2	Módulo elevador tensão DC-DC	XL6009 2-24V 2A	HU Infinito	2	R\$ 14,07	R\$ 28,14
DC REGULADOR	Regulador DC 5V	LM7805 - 5V/1A	HU Infinito	1	R\$ 1,29	R\$ 1,29
BAT_LITIO_1-BAT_LITIO_4	Pilhas ión lítio	3,7V - Modelo 18650	HU Infinito	4	R\$ 12,40	R\$ 49,60
U1,U2	Módulo Carregador de Bateria Lítio	TP4056	HU Infinito	2	R\$ 3,50	R\$ 7,00
-	Suporte Bateria Lítio Paralelo	Modelo 18650 tipo canoa	HU Infinito	2	R\$ 5,90	11,80
Fontes de Alimentação - Esquemático 2 e 3						R\$ 92,83
ON/OFF SWITCH 1-2	Chave ON/OFF	Gangorra 2T S/LED	HU Infinito	2	R\$ 1,99	R\$ 3,98
3PIN,2PIN,SAIDA	Conector barra de pino macho	40 pinos - 1x15mm - 180 graus	HU Infinito	1	R\$ 0,77	R\$ 0,77
-	Conector barra de pino fêmea	10 pinos - 1x20mm	HU Infinito	1	R\$ 1,19	R\$ 1,19
-	Jumper fêmea-fêmea	Comprimento de 100mm	HU Infinito	10 (40)	-	R\$ 10,90
-	Jumper macho-fêmea	Comprimento de 100mm	HU Infinito	10 (40)	-	R\$ 10,90
ENTRADA	Conector P2 Fêmea (J2)	Mono - 2T - sem rosca	HU Infinito	1	R\$ 0,92	R\$ 0,92
-	Conector Garra Jacaré (com proteção)	Médio - Vermelho/Preto (2/1) - 35mm	HU Infinito	3	R\$ 0,60	R\$ 1,80
-	Pormenores	x	x	x	x	R\$ 20,00
Pinos, Conectores, Fios, Extras - Esquemáticos 1, 2 e 3						R\$ 50,46
Total						R\$ 1.415,71

Informações adicionais:

• Coluna Fornecedores:

- Fornecedores com (I) indica que o componente foi importado, valor gerado é uma conversão direta usando Dólar Comercial na cotação \$1=R\$5,53;
- Site dos fornecedores: Mercado Livre ([Link](#)), HU Infinito ([Link](#)), Mouser ([Link](#)), Texas Instruments ([Link](#)), FilipeFlop ([Link](#)), Kabum ([Link](#));

• Coluna Qtd: Números entre aspas () indica a quantidade de compra para o respectivo valor final, por isso coluna preço unitário está vazia nesses casos;

• Coluna Preços: Valor para cada componente foi pesquisado no dia 14/05/2021; Valor para cada componente não inclui frete, apenas preço de varejo.

APÊNDICE D CÓDIGO FONTE

A. Programa ads - /Final/ads.c - [Link](#)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <string.h>
5 #include <signal.h>
6 #include <linux/i2c-dev.h>
7 #include <sys/ioctl.h>
8 #include <fcntl.h>
9 #include <time.h>
10 #include <MQTTClient.h>
11 #include "ads1115_rpi.h"
12
13 #define TEMPO 625
14
15 #define MQTT_ADDRESS      "tcp://127.0.0.1:1883"
16
17 #define CLIENTID          "ads"
18
19
20 #define MQTT_PUBLISH_TOPIC      "dados"
21 #define MQTT_SUBSCRIBE_TOPIC    "MQTTClientSubTopic"
22
23 MQTTClient client;
24 int i=0;
25 int k=0;
26 int t=0;
27 int ant=0;
28
29 void publish(MQTTClient client, char* topic, char*
30 payload) {
31   MQTTClient_message pubmsg =
32     MQTTClient_message_initializer;
33
34   pubmsg.payload = payload;
35   pubmsg.payloadlen = strlen(pubmsg.payload);
36   pubmsg.qos = 0;
37   pubmsg.retained = 0;
38   MQTTClient_deliveryToken token;
39   MQTTClient_publishMessage(client, topic, &pubmsg
40 , &token);
41   MQTTClient_waitForCompletion(client, token, 1000
42 L);
43 }
44
45 void capturaDados(){
46   char n[10];
47   unsigned int dados;
48
49   dados = readVoltage(0);
50   sprintf(n, "%d", dados);
51   publish(client, MQTT_PUBLISH_TOPIC, n);
52   k++;
53   t=time(NULL);
54   if(t-ant == 1){
55     printf("Freq: %dHz\n",k);
56     k=0;
57   }
58   ant=t;
59 }
60
61 int main(void) {
62   int rc;
63   int lel;
64   MQTTClient_connectOptions conn_opts =
65     MQTTClient_connectOptions_initializer;
66   MQTTClient_create(&client, MQTT_ADDRESS, CLIENTID,
67     MQTTCLIENT_PERSISTENCE_NONE, NULL);

```

```

64   rc = MQTTClient_connect(client, &conn_opts);
65
66   if (rc != MQTTCLIENT_SUCCESS)
67   {
68     printf("\n\rFalha na conexao ao broker MQTT.
69     Erro: %d\n", rc);
70     exit(-1);
71   }
72
73   if(openI2CBus("/dev/i2c-1") == -1)
74   {
75     return EXIT_FAILURE;
76   }
77   setI2CSlave(0x48);
78   signal(SIGALRM,capturaDados);
79   alarm(TEMPO,TEMPO);
80   while(1)
81   {
82
83   }
84
85   return EXIT_SUCCESS;
86 }

```

B. Programa predictor - /Final/predictor.cpp - [Link](#)

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <MQTTClient.h>
5 #include <time.h>
6 #include <iostream>
7 #include <armadillo>
8 #include <opencv2/core.hpp>
9 #include <opencv2/imgproc.hpp>
10 #include "opencv2/imgcodecs.hpp"
11 #include <opencv2/highgui.hpp>
12 #include <opencv2/ml.hpp>
13 #include "lib/sigpack/sigpack.h"
14
15 // using namespace std;
16 using namespace arma;
17 using namespace sp;
18 using namespace cv;
19 using namespace cv::ml;
20
21 #define WINDOW 800
22
23 #define MQTT_ADDRESS      "tcp://127.0.0.1:1883"
24
25 #define CLIENTID          "test"
26
27
28 #define MQTT_PUBLISH_TOPIC      (char*) "prediction"
29 #define MQTT_SUBSCRIBE_TOPIC    (char*) "dados"
30
31
32 MQTTClient client;
33 Ptr<SVM> svm = SVM::create();
34
35 // int window[WINDOW]={0};
36 colvec window(WINDOW, fill::zeros);
37 int counter=0;
38
39
40 double energia(vec b, vec a, colvec x){
41   IIR_filt<double, double, double> iir_filt;
42   iir_filt.set_coeffs(b, a);
43   colvec Y1 = iir_filt.filter(x);
44   double y1 = norm(Y1,2);
45   y1 = y1*y1;
46   return y1;

```

```

47 }
48
49
50 colvec extraction(colvec x){
51     colvec parameters(8, fill::zeros);
52
53     x=x-mean(x);
54     //Valor medio absoluto
55     parameters(0)=mean(abs(x));
56     //Valor RMS
57     parameters(1)=sqrt(sum(square(x))/WINDOW);
58     //Waveform length
59     parameters(2)=sum(abs(diff(x)));
60     //Variancia
61     parameters(3)=var(x);
62     //Average Amplitude Change
63     double Y = 0;
64     for (int i = 0; i < WINDOW - 2; i++)
65     {
66         Y = Y + abs(x(i+1)-x(i));
67     }
68     parameters(4)=Y;
69     //Energia em bandas
70     double y1;
71     double tot = norm(x,2);
72     tot = tot*tot;
73
74
75     vec b = {2.44897128e-09, 3.67345692e-08,
76     2.57141984e-07, 1.11428193e-06,
77     3.34284579e-06, 7.35426075e-06, 1.22571012e
78     -05, 1.57591302e-05,
79     1.57591302e-05, 1.22571012e-05, 7.35426075e
80     -06, 3.34284579e-06,
81     1.11428193e-06, 2.57141984e-07, 3.67345692e
82     -08, 2.44897128e-09};
83
84     vec a = { 1.00000000e+00, -8.94097666e+00,
85     3.81274598e+01, -1.02595239e+02,
86     1.94421048e+02, -2.74396921e+02,
87     2.97560724e+02, -2.52180163e+02,
88     1.68238453e+02, -8.82785479e+01,
89     3.61095963e+01, -1.12999558e+01,
90     2.61723273e+00, -4.23343031e-01,
91     4.27414269e-02, -2.02960157e-03};
92     y1=energia(b,a,x);
93     parameters(5)=y1/tot*100;
94
95     vec b1 = { 1.31247236e-05, 0.00000000e+00,
96     -1.18122513e-04, 0.00000000e+00,
97     4.72490051e-04, 0.00000000e+00,
98     -1.10247679e-03, 0.00000000e+00,
99     1.65371518e-03, 0.00000000e+00,
100    -1.65371518e-03, 0.00000000e+00,
101    1.10247679e-03, 0.00000000e+00,
102    -4.72490051e-04, 0.00000000e+00,
103    1.18122513e-04, 0.00000000e+00,
104    -1.31247236e-05};
105
106     vec b2 = { 9.17785774e-09, 0.00000000e+00,
107     -1.28490008e-07, 0.00000000e+00,
108     8.35185054e-07, 0.00000000e+00,
109     -3.34074022e-06, 0.00000000e+00,
110     9.18703559e-06, 0.00000000e+00,
111     -1.83740712e-05, 0.00000000e+00,
112     2.75611068e-05, 0.00000000e+00,
113     -3.14984077e-05, 0.00000000e+00,
114     2.75611068e-05, 0.00000000e+00,
115     -1.83740712e-05, 0.00000000e+00,
116     9.18703559e-06, 0.00000000e+00,
117     -3.34074022e-06, 0.00000000e+00,
118     8.35185054e-07, 0.00000000e+00,
119     -1.28490008e-07, 0.00000000e+00,
120     9.17785774e-09};
121
122     vec a2 = { 1.00000000e+00, -3.75618537e+00,
123     1.49160032e+01, -3.64590663e+01,
124     8.62006102e+01, -1.58929766e+02,
125     2.79121488e+02, -4.14409137e+02,
126     5.84695352e+02, -7.23468409e+02,
127     8.51082988e+02, -8.94437527e+02,
128     8.94426960e+02, -8.06151007e+02,
129     6.91695461e+02, -5.36216565e+02,
130     3.95715767e+02, -2.63000210e+02,
131     1.66317800e+02, -9.38008369e+01,
132     5.03045394e+01, -2.35910290e+01,
133     1.05214957e+01, -3.94854217e+00,
134     1.41484447e+00, -3.92253746e-01,
135     1.05899399e-01, -1.73361510e-02,
136     3.06405453e-03};
137     y1=energia(b2,a2,x);
138     parameters(7)=y1/tot*100;
139     return parameters;
140 }
141
142 cv::Mat<double> to_cvmat(const mat &src)
143 {
144     return cv::Mat<double>{int(src.n_cols), int(src.n_rows), const_cast<double*>(src.memptr())};
145 }
146
147 void publish(MQTTClient client, char* topic, char* payload)
148 {
149     MQTTClient_message pubmsg =
150     MQTTClient_message_initializer;
151
152     pubmsg.payload = payload;
153     pubmsg.payloadlen = strlen((char*)pubmsg.payload);
154     pubmsg.qos = 0;
155     pubmsg.retain = 0;
156     MQTTClient_deliveryToken token;
157     MQTTClient_publishMessage(client, topic, &pubmsg,
158     &token);
159     MQTTClient_waitForCompletion(client, token, 1000
160 L);
161 }
162
163 int on_message(void *context, char *topicName, int
164 topicLen, MQTTClient_message *message) {
165     char n[10];
166
167     for (int i = 0; i < WINDOW-2; i++)
168     {
169         window(i)=window(i+1);
170     }
171     char* payload = (char*)message->payload;
172     window(WINDOW-1)= atoi(payload);
173
174     if(counter == WINDOW/2){
175         counter = 0;
176         colvec parametros = extraction(window);
177         cv::Mat param_cv= to_cvmat(parametros);
178     }
179 }

```

C. Programa de treinamento - /Final/treino.cpp - [Link](#)

```

1 #include <iostream>                                67    vec b1 = { 1.31247236e-05,  0.00000000e+00,
2 #include <opencv2/core.hpp>                         68    -1.18122513e-04,  0.00000000e+00,
3 #include <opencv2/imgproc.hpp>                      69    4.72490051e-04,  0.00000000e+00,
4 #include "opencv2/imgcodecs.hpp"                     70    -1.10247679e-03,  0.00000000e+00,
5 #include <opencv2/highgui.hpp>                      71    1.65371518e-03,  0.00000000e+00,
6 #include <opencv2/ml.hpp>                           72    -1.65371518e-03,  0.00000000e+00,
7 #include <armadillo>                               73    1.10247679e-03,  0.00000000e+00,
8 #include "lib/sigpack/sigpack.h"                    74    -4.72490051e-04,  0.00000000e+00,
9 #include <time.h>                                 75    1.18122513e-04,  0.00000000e+00,
10 using namespace cv;                                76    -1.31247236e-05};

11 using namespace cv::ml;                            77    vec a1 = { 1.00000000e+00, -9.68593127e+00,
12 using namespace std;                            78    4.69901672e+01, -1.50739975e+02,
13 using namespace arma;                           79    3.56909439e+02, -6.60502621e+02,
14 using namespace sp;                            80    9.88073702e+02, -1.21982331e+03,
15                                         1.25851221e+03, -1.09246913e+03,
16 #define WINDOW 800                                81    7.99555480e+02, -4.92238937e+02,
17                                         2.53142229e+02, -1.07374086e+02,
18                                         3.67952917e+01, -9.85240606e+00,
19 double energia(vec b,vec a,colvec x){          82    1.94788749e+00, -2.55100758e-01,
20     IIR_filt<double, double, double> iir_filt;  83    1.68137718e-02};
21     iir_filt.set_coeffs(b, a);
22     colvec Y1 = iir_filt.filter(x);

```

```

y1=energia(b1,a1,x);
parameters(6)=y1/tot*100;
vec b2 = { 9.17785774e-09, 0.00000000e+00,
-1.28490008e-07, 0.00000000e+00,
8.35185054e-07, 0.00000000e+00,
-3.34074022e-06, 0.00000000e+00,
9.18703559e-06, 0.00000000e+00,
-1.83740712e-05, 0.00000000e+00,
2.75611068e-05, 0.00000000e+00,
-3.14984077e-05, 0.00000000e+00,
2.75611068e-05, 0.00000000e+00,
-1.83740712e-05, 0.00000000e+00,
9.18703559e-06, 0.00000000e+00,
-3.34074022e-06, 0.00000000e+00,
8.35185054e-07, 0.00000000e+00,
-1.28490008e-07, 0.00000000e+00,
9.17785774e-09};

vec a2 = { 1.00000000e+00, -3.75618537e+00,
1.49160032e+01, -3.64590663e+01,
8.62006102e+01, -1.58929766e+02,
2.79121488e+02, -4.14409137e+02,
5.84695352e+02, -7.23468409e+02,
8.51082988e+02, -8.94437527e+02,
8.94426960e+02, -8.06151007e+02,
6.91695461e+02, -5.36216565e+02,
3.95715767e+02, -2.63000210e+02,
1.66317800e+02, -9.38008369e+01,
5.03045394e+01, -2.35910290e+01,
1.05214957e+01, -3.94854217e+00,
1.41484447e+00, -3.92253746e-01,
1.05899399e-01, -1.73361510e-02,
3.06405453e-03};
y1=energia(b2,a2,x);
parameters(7)=y1/tot*100;
return parameters;
}

mat multi_extraction(mat M){
mat E(8,M.n_cols);
for (unsigned int i = 0; i < M.n_cols; i++)
{
    E.col(i)=extraction(M.col(i));
}
return E;
}

cv::Mat<double> to_cvmat(const mat &src)
{
    return cv::Mat<double>{int(src.n_cols), int(src.n_rows), const_cast<double*>(src.memptr())};
}

void fitScaler(const cv::Mat matriz, cv::Mat *media,
cv::Mat *desvio){
cv::Mat meanValue, stdValue;
cv::Mat colSTD(1, matriz.rows, CV_64FC1);
cv::Mat colMEAN(1, matriz.rows, CV_64FC1);

for (int i = 0; i < matriz.rows; i++){
    cv::meanStdDev(matriz.row(i), meanValue,
stdValue);
    colSTD.at<double>(i) = stdValue.at<double>(0);
    colMEAN.at<double>(i) = meanValue.at<double>(0);
}
*media = colMEAN;
*desvio = colSTD;
}

cv::Mat transformScaler(const cv::Mat matriz, cv::Mat
media , cv::Mat desvio){
cv::Mat escalonado;
matriz.copyTo(escalonado);
for (int i = 0; i < matriz.rows; i++){
    escalonado.row(i)=(escalonado.row(i) - media
.at<double>(i))/desvio.at<double>(i);
}
return escalonado;
}

int main()
{
    mat M_close;
    colvec t_close;
    mat M_open;
    colvec t_open;
    mat M_opened;
    colvec t_opened;

M_close.load("M_close.csv");
t_close.load("t_close.csv");
M_open.load("M_open.csv");
t_open.load("t_open.csv");
t_open.fill(1);
M_opened.load("M_opened.csv");
t_opened.load("t_opened.csv");
mat E1 = multi_extraction(M_close);
mat E2 = multi_extraction(M_opened);
mat E3 = multi_extraction(M_open);
mat E = join_rows(E1,E2);
E = join_rows(E,E3);
colvec t_total = join_cols(t_close,t_opened);
t_total = join_cols(t_total,t_open);

cv::Mat E1_cv = to_cvmat(E1);
cv::Mat E2_cv = to_cvmat(E2);
cv::Mat E3_cv = to_cvmat(E3);
cv::Mat t1 = to_cvmat(t_close);
cv::Mat t2 = to_cvmat(t_opened);
cv::Mat t3 = to_cvmat(t_open);
E1_cv.convertTo(E1_cv,CV_32F);
E2_cv.convertTo(E2_cv,CV_32F);
E3_cv.convertTo(E3_cv,CV_32F);
t1.convertTo(t1,CV_32S);
t2.convertTo(t2,CV_32S);
t3.convertTo(t3,CV_32S);
E1_cv = E1_cv.t();
E2_cv = E2_cv.t();
E3_cv = E3_cv.t();
cv::Ptr<cv::ml::TrainData> E1_data=cv::ml::
TrainData::create(E1_cv,1,t1);
cv::Ptr<cv::ml::TrainData> E2_data=cv::ml::
TrainData::create(E2_cv,1,t2);
cv::Ptr<cv::ml::TrainData> E3_data=cv::ml::
TrainData::create(E3_cv,1,t3);
cv::theRNG().state = time(NULL);
E1_data->setTrainTestSplitRatio(0.8 ,true );
E1_data->shuffleTrainTest ();
E2_data->setTrainTestSplitRatio(0.8 ,true );
E2_data->shuffleTrainTest ();
E3_data->setTrainTestSplitRatio(0.8 ,true );
E3_data->shuffleTrainTest ();
cv::Mat E_f;
cv::vconcat(E1_data->getTrainSamples() ,E2_data->
getTrainSamples() ,E_f);
cv::vconcat(E_f,E3_data->getTrainSamples() ,E_f);
E_f=E_f.t();
cv::Mat t_f;
cv::hconcat(E1_data->getTrainResponses() ,E2_data-
>getTrainResponses() ,t_f);
cv::hconcat(t_f ,E3_data->getTrainResponses() ,t_f );
}

```

```

200
201     cv::Mat E_cv = to_cvmat(E);
202     cv::Mat t_cv = to_cvmat(t_total);
203     E_cv.convertTo(E_cv,CV_32F);
204     t_cv.convertTo(t_cv,CV_32S);
205     E_cv=E_cv.t();
206
207     cv::Mat media , desvio ;
208     fitScaler(E_f,&media,&desvio);
209     E_f = transformScaler(E_f,media , desvio );
210
211     cv::FileStorage fs( " scaler.yml" , FileStorage::WRITE);
212     fs << " media " << media;
213     fs << " desvio " << desvio;
214     fs.release();
215
216     cout << " Starting training process " << endl;
217     Ptr<SVM> svm = SVM::create();
218     svm->setType(SVM::C_SVC);
219     svm->setC(1.0);
220     svm->setKernel(SVM::RBF);
221     svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, (int)1e7, 1e-6));
222
223     //svm->train(E_cv , COL_SAMPLE, t_cv );
224     svm->train(E_f , COL_SAMPLE, t_f );
225     svm->save("maquina.xml");
226     cout << " Finished training process " << endl;
227     cv::Mat E1_idx = E1_data->getTestSampleIdx();
228     E1_cv = transformScaler(E1_cv,media , desvio );
229     cv::Mat E2_idx = E2_data->getTestSampleIdx();
230     E2_cv = transformScaler(E2_cv,media , desvio );
231     cv::Mat E3_idx = E3_data->getTestSampleIdx();
232     E3_cv = transformScaler(E3_cv,media , desvio );
233     int Tn=0;
234     int Tp=0;
235     int Fn=0;
236     int Fp=0;
237     for (int i = 0; i < E1_idx.cols; i++)
238     {
239         Tp += svm->predict(E1_cv.col(E1_idx.at<int>(i)).t());
240     }
241     for (int i = 0; i < E3_idx.cols; i++)
242     {
243         Tp += svm->predict(E3_cv.col(E3_idx.at<int>(i)).t());
244     }
245     Fn = (E1_idx.cols + E3_idx.cols)-Tp;
246     for (int i = 0; i < E2_idx.cols; i++)
247     {
248         Fp += svm->predict(E2_cv.col(E2_idx.at<int>(i)).t());
249     }
250     Tn = E2_idx.cols - Fp;
251     //cv::Mat E4_idx = E1_data->getTestSamples();
252     double accuracy = (double)(Tp+Tn) / (double) (Tp+Fp+Tn+Fn);
253     double precision = (double)(Tp) / (double) (Tp+Fp);
254     double recall = (double)(Tp) / (double)(Tp+Fn);
255     double f1_score = 2 * (precision*recall) / (precision+recall);
256
257     cout << "Acuracia: " << accuracy << endl;
258     cout << "Precisao: " << precision << endl;
259     cout << "Recall: " << recall << endl;
260     cout << "F1_score: " << f1_score << endl;
261     cout << "Matriz: " << endl;
262     cout << Tp << " << Fn << endl;
263     cout << Fp << " << Tn << endl;
264     cout << E1_cv.col(5) << endl;
265

```

```

266     return 0;
267 }
```

D. Programa de controle - /Final/control.c - [Link](#)

```

1 // Access from ARM Running Linux - updated for RPi4
2 #include "gpio_dev_mem.h"
3 #include <sys/poll.h>
4 #include <pthread.h>
5 #include <signal.h>
6 #include <string.h>
7 #include <MQTTClient.h>
8 #include <time.h>
9
10 #define MQTT_ADDRESS "tcp://127.0.0.1:1883"
11
12 #define CLIENTID "control"
13
14 #define MQTT_PUBLISH_TOPIC "MQTTCCClientPubTopic"
15 #define MQTT_SUBSCRIBE_TOPIC "prediction"
16
17 MQTTClient client;
18 int t=0;
19 int ant=0;
20
21 void publish(MQTTClient client , char* topic , char*
payload);
22 int predictor_callback(void *context , char *
topicName , int topicLen , MQTTClient_message * message);
23
24
25 #define servo_pin 17
26 #define minAngle -45 // 60
27 #define maxAngle 60 //150
28
29 #define Nlim 30
30 #define f 50
31
32 /* dutyCyclechange function - using device memory */
33 void dutyCyclechange(int pin , int degree , int N)
34 {
35     int t1 = (f*degree+4)/9+1500;
36     int t2 = 20000-t1;
37     int i;
38     for(i=0; i<N; i++)
39     {
40         GPIO_SET = 1<<pin;
41         usleep(t1);
42         GPIO_CLR = 1<<pin;
43         usleep(t2);
44     }
45     // printf("Duty cycle = %.2f %%\n" , (double)
46     // (100*t1)/(double)period);
47 }
48
49 int n=0;
50 int control_callback(void *context , char *topicName ,
51 int topicLen , MQTTClient_message *message) {
52     char* payload = message->payload;
53     int N = Nlim;
54
55     if (atoi(payload) == 1)
56         dutyCyclechange(servo_pin , minAngle , N);
57     else if (atoi(payload) == 0)
58         dutyCyclechange(servo_pin , maxAngle , N);
59     else if (atoi(payload) == 3)
60         dutyCyclechange(servo_pin , 0 , N);
61     MQTTClient_freeMessage(&message);
62     MQTTClient_free(topicName);
63 }
```

```

64 int main(int argc, char *argv[])
65 {
66     int rc;
67
68     /* GPIO setup - device memory */
69     setup_io();
70     OUT_GPIO(servos_pin);
71
72     /* MQTT */
73     MQTTClient_connectOptions conn_opts =
74         MQTTClient_connectOptions_initializer;
75
76     MQTTClient_create(&client, MQTT_ADDRESS,
77         CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
78     MQTTClient_setCallbacks(client, NULL, NULL,
79         control_callback, NULL);
80     rc = MQTTClient_connect(client, &conn_opts);
81
82     if (rc != MQTTCLIENT_SUCCESS)
83     {
84         printf("\n\rFalha na conexao ao broker MQTT.
85             Erro: %d\n", rc);
86         exit(-1);
87     }
88
89     MQTTClient_subscribe(client,
90         MQTT_SUBSCRIBE_TOPIC, 0);
91
92 }

```

E. Programa de ativação automática - /Final/automatico.sh

- [Link](#)

```

1 #!/usr/bin/env bash
2 programas=(ads preditor controle)
3 for i in ${programas[@]}
4 do
5     PID=`cat /home/ubuntu/git/embedded_emg_prosthesis/
6         codigo-fonte/PC4/$i.pid`
7
8     if ! ps -p $PID > /dev/null
9     then
10        rm /home/ubuntu/git/embedded_emg_prosthesis/codigo
11            -fonte/PC4/$i.pid
12        sudo ./${i}.out & echo $! >>/home/ubuntu/git/
13            embedded_emg_prosthesis/codigo-fonte/PC4/$i.pid
14    fi
15 done

```