

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.

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 [5].

Representando um número expressivo de pessoas onde apenas 3%, cerca de 110.000, teriam alguma condição para utilizar alguma prótese de funções eletrônicas [6]. 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 [5]. 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 envia o comando de abertura e fechamento da mão para o servomotor.

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 hialoplasmoma. Ao entrar em contato com as miofibrillas, 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.

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.

Conforme a figura 2 é perceptível que com a grande quan-

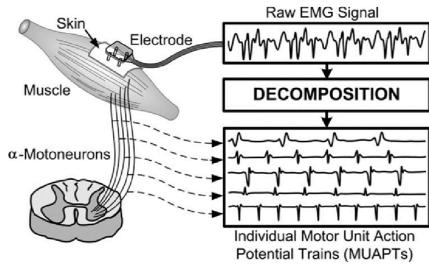


Figura 1: Representação da aquisição de um EMG

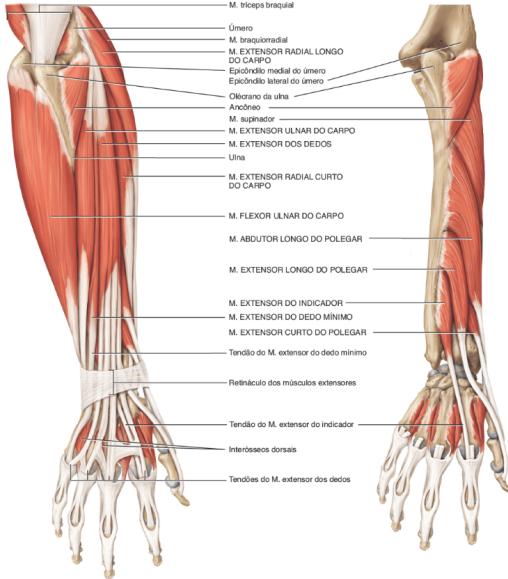


Figura 2: Musculatura do antebraço

tidade 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 3. Sendo necessário eletrodos como o de tipo de gel sólido com sensor de prata (Ag/AgCl) MP43 circular de 43mm da marca MedPex.

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.



Figura 3: Posicionamento dos eletrodos no antebraço

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.

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;
- Detectar mau contato nos eletrodos;

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++;

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 [8], gráfico da figura 4, podemos definir as dimensões aproximadas para o braço de uma criança com base em um adulto entre 16-20 anos.

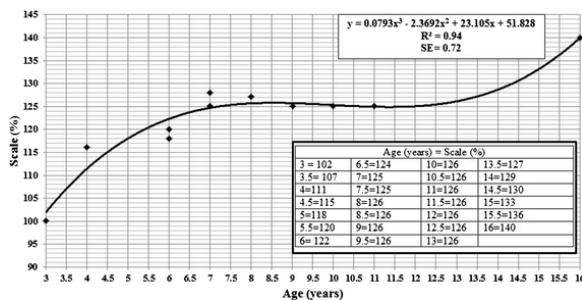


Figura 4: Gráfico mostrando a relação de dimensões da prótese Cyborg Beast em porcentagem conforme a idade [8]

Utilizando como base dos cálculos do gráfico na figura 4 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 5

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

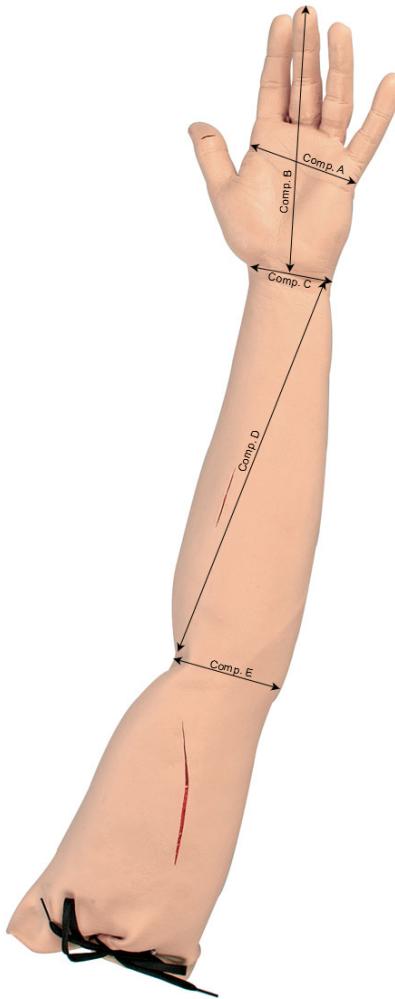


Figura 5: Legenda dos comprimentos medidos na tabela I

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 e criação 3D do design de um braço completo, foi utilizado o projeto design de braço do projeto MyPo 2.0 [9] 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 6.

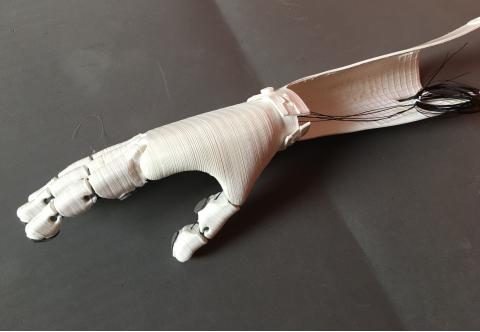


Figura 6: Prótese em 3D para um criança entre 3-5 anos, versão V0 e com base no design MyPo 2.0 [9] (Trabalho realizado em outra disciplina)

- 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 mais elevadas, suficientes para suportar a força necessária para movimentação dos dedos realizada por um atuador. Na figura 7 temos uma foto da prótese mostrando em detalhes esses componentes mecânicos na mão.



Figura 7: 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 8 temos uma foto da prótese mostrando em detalhes o mecanismo utilizado, foi impresso em 3D usando como material PLA branco.

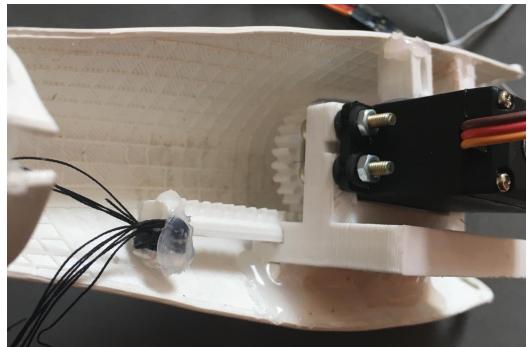


Figura 8: Detalhe do mecanismo ligado ao atuador e os fios de cada dedo

2) Hardware Eletrônico:

Para a parte de *hardware* da solução relacionada a eletrônica se segue o diagrama de blocos geral na figura 18 nos apêndices. Na figura 18 nos apêndices 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.

- 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 19 no apêndice.

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 esta 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.

Abaixo temos a lista dos materiais que foram utilizados para o circuito proposto de aquisição do sinal de EMG implementado em placa de circuito impresso (PCB) para componentes de encapsulamento SMD, com inclusão de ferramentas e componentes químicos para confecção da PCB.

- 2 - Resistor de 220Ω 5% - 1/4W;
- 1 - Resistor de $1k\Omega$ 5% - 1/4W;
- 1 - Resistor de $10k\Omega$ 5% - 1/8W;
- 6 - Resistor de $20k\Omega$ 5% - 1/4W;
- 3 - Resistor de $680k\Omega$ 5% - 1/4W;
- 1 - Resistor de $1M\Omega$ 5% - 1/4W;
- 1 - Trimpot linear $500k\Omega$ (3361P-504);
- 4 - Diodo de Silício DL4148;
- 1 - Capacitor cerâmico 50V/1nF MLCC;
- 3 - Capacitor cerâmico 50V/10nF MLCC;
- 1 - Capacitor cerâmico 50V/100nF MLCC;
- 5 - Capacitor cerâmico 50V/ $1\mu F$ MLCC;
- 1 - C.I. INA AD620 (1 Amplificador de Instrumentação);
- 2 - C.I. ADTL084ARZ (4 Amplificadores Operacionais);
- 1 - C.I. 78L05 SMD (Regulador de tensão 5V/140mA);

- 1 - C.I. LM2776 SMD (*Switched Capacitor Inverter*);
- 7 - Conector pino macho (1x15mm - 180 graus)
- 2 - Bateria de 3.8V/2480mah;
- 1 - Placa de fibra de vidro virgem dupla face 40x70mm.

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 [10].

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 19 localizado no apêndice, 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 9.

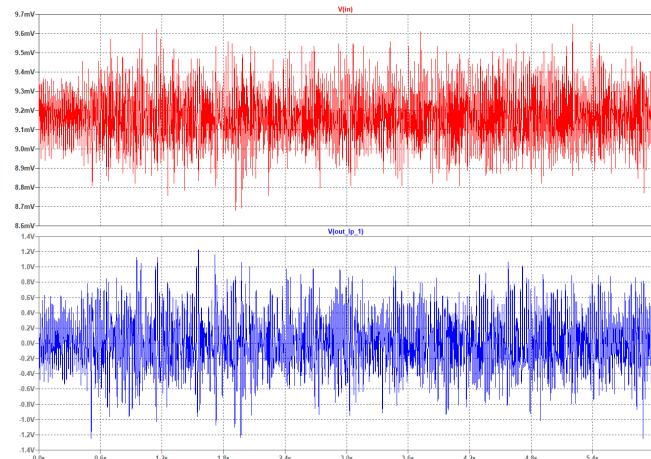


Figura 9: Resultado da simulação

• 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 10 e representação física na figura 11. 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 *I₂C* - 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);

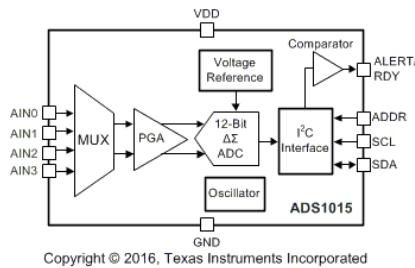


Figura 10: Esquemático do módulo ADS1015



Figura 11: Foto do módulo ADC ADS1015

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 V e 2,8 V.

• Sistema embarcado:

A funcionalidade do sistema embarcado é realizar pre-processamento digital do sinal advindo do conversor A/D, extraír 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 e 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). Dentre 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 [12].

• 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 μ V 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 12. As especificações técnicas deste componentes 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 μ s

• Circuito Desacoplamento:

Existem ruídos que podem alterar e prejudicar os sinais adquiridos por meio do circuito condicionador, entre eles

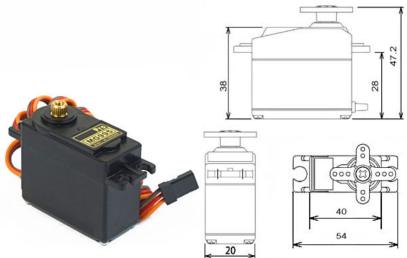


Figura 12: Servomotor MG995 com engrenagens metálicas

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 20 no apêndice.

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 phototransistor em um caminho direto, assim isolando o circuito conectado ao fotodiodo e o circuito conectado ao phototransistor. 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.

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 20 no apêndice e código exemplo compilado para controle de servos *Sweep* foi possível observar a movimentação do servomotor simulado entre os angulos de 90° e -90°.

• 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 de prova de conceito 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 21 no apêndice temos o diagrama completo de todos os nós e tópicos que os conectam.

1) *Nó adc*: Nesse programa os dados do conversor A/D ADS1015 são capturados e enviados ao tópico dados. O código fonte utilizado está disponível no apêndice código fonte, seção A.

Etapas:

- Abrir conexão serial com *baudrate* de 115200;
- Declarar tópico em que dados serão publicados;
- Estipular frequência do nó para 1600Hz;
- Loop até o programa ser desligado;
 - Ler linha do serial;
 - Converter string em inteiro;
 - Publicar valor no tópico dados.
 - Dormir para obedecer frequência de execução de 1600Hz

2) *Nó accumulator*: Esse programa é responsável em criar um vetor de tamanho 800 com dados para processamento. Os dados enviados possuem uma sobreposição de metade da janela criada e são enviados ao tópico *data_window*. O código

fonte utilizado está disponível no apêndice código fonte, seção A.

Etapas:

- Declarar vetor de tamanho 800 para dados;
- Declarar tópico *data_window* em que o vetor será publicado;
- Declarar tópico *dados* que será lido pelo programa;
- 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, publica vetor no tópico *data_window*;
 - Caso contrário, soma um ao contador.

3) *Nó extraction*: Esse programa é responsável em extraír características do sinal disponíveis no tópico *data_window*. O código fonte utilizado está disponível no apêndice código fonte, seção A. 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

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

Essas características são então mandadas ao tópico *extracted*.

Etapas:

- Declarar vetor float com parâmetros a serem extraídos
- Declarar tópico *extracted* em que o vetor será publicado
- Declarar tópico *data_window* que será lido pelo programa
- Função quando dados são recebidos;
 - Retirar valor dc do sinal;
 - Calcular parâmetros II-D3;
 - Publicar vetor de parâmetros no tópico *extracted*;

4) *Nó predictor*: Esse programa classifica as características extraídas do sinal, disponível no tópico *extracted*, por meio de uma SVM treinada com dados obtidos dos autores do projeto e então mandam a classificação para o tópico *servo/action*. O código fonte utilizado está disponível no apêndice código fonte, seção A.

Etapas:

- Declarar tópico *servo/action* em que o vetor será publicado
- Declarar tópico *extracted* que será lido pelo programa
- Carregar arquivo de treinamento de SVM
- Carregar arquivo de escalonamento de dados
- Função quando dados são recebidos:
 - Escalona dados recebidos;
 - Classifica dados recebidos na SVM;
 - Converte classificação para inteiro;
 - Publica classificação no tópico *servo/action*

5) *Nó control*: Esse programa controla o servomotor presente na mão robótica abrindo e fechando por meio dos ângulos de 150 graus e 60 graus, respectivamente. O código fonte utilizado está disponível no apêndice código fonte, seção A.

Etapas:

- 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;
 - Período para pulso relacionado a posição 0 graus;
 - Período para pulso relacionado a posição 180 graus;
 - Frequência do sinal PWM a ser gerado;
- Calcular parâmetros relacionados ao *duty_cycle* do PWM;
- Inicializar o pino GPIO configurado;
- Declarar o tópico *servo/action* que será lido pelo programa;
- Função quando algum dado é recebido no tópico *servo/action*:
 - Caso o dado recebido seja int32 '1' o servomotor é movimentado para ângulo mínimo (abrir mão);
 - Caso o dado recebido seja int32 '-1' o servomotor é movimentado para ângulo máximo (fechar mão);
 - Caso o dado recebido seja int32 '3' o servomotor é movimentado para ângulo médio de 90° (mão meio fechada);
- Função movimentar servomotor para ângulo de entrada:
 - Calcula *duty_cycle* do sinal PWM;
 - Configura o pino GPIO configurado para o modo envio de dados (*output*);
 - Solicita mudança do sinal PWM enviado pelo pino GPIO para novo ângulo.

III. RESULTADOS

A. Circuito condicionador

O circuito condicionador foi implementado em uma placa de circuito impresso (PCB), versão em 3D na figura 14 e versão real na figura 15. 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 13 demonstra o teste realizado no osciloscópio, onde temos como entrada uma senóide $V_{pp} = 21,6 \text{ mV}$ a frequência de 75,30 Hz e como saída uma senóide amplificada e filtrada para $V_{pp} = 6,6 \text{ V}$ a frequência 75,19 Hz.

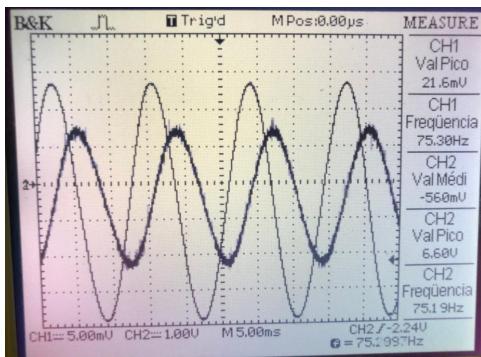


Figura 13: Teste do circuito com uma entrada senoidal

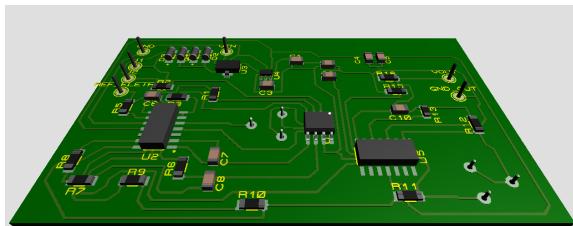


Figura 14: Resultado final da PCB no Proteus

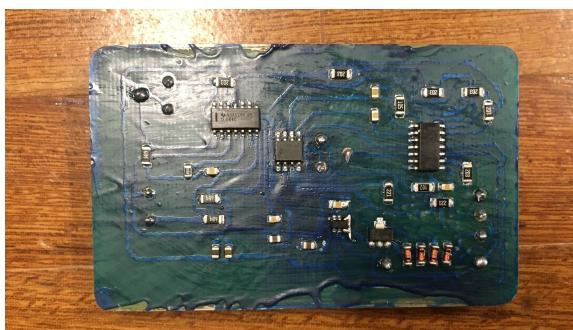


Figura 15: Resultado final da PCB

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 em python 3 diretamente no sistema embarcado concordava com outros processos da implementação, 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 16.

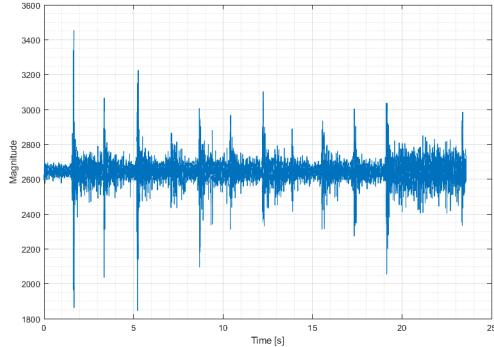


Figura 16: Visualização de sinal de EMG com $f_s = 1600\text{Hz}$ utilizando o eletrodo de superfície descartável

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 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. Integração

Com todos integração dos componentes foram todos conectados entre seguindo o diagrama de bloco geral, figura 18 nos apêndices, obtendo como resultado o sistema na figura 17.

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 3. 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



Figura 17: Sistema Quiro com todos os componentes integrados

observado o bom funcionamento caso eletrodos forem colocados recentemente e na posição correta. No caso em que são usados eletrodos usados anteriormente são observados comportamentos falso-positivo no caso de fechar a mão.

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.

Para otimização e comprimento dos requisitos exigidos para o projeto também deve ser implementado o uso de pipes, threads, sockets e etc para substituir o uso do framework ROS. Dependendo do nível de otimização final alcançado pode ser interessante a mudança da Raspberry Pi 4 para uma Raspberry Pi 1 para diminuir ainda mais os componentes utilizados por essa se assemelha em nível de processamento a uma Raspberry Pi Zero.

Os códigos atualmente implementados e futuras alterações 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 83.2s uppl (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] IBGE. Censo Demográfico 2010. Disponível em: <http://www.ibge.gov.br> Acesso em 23 fev. 2021.

- [6] 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.
- [7] DUFF, Susan V. et al. Innovative evaluation of dexterity in pediatrics. *Journal of Hand Therapy*, v. 28, n. 2, p. 144-150, 2015.
- [8] 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.
- [9] Myo & PO, MyPo 2.0 . "Disponível em: <https://www.thingiverse.com/thing:2409406>". Acesso em 23 fev. 2021 .
- [10] 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/sEMG+for+Basic+Hand+movements>". Acesso em 24 fev de 2021 .
- [11] ALKAN, Ahmet; GÜNEY, Mücahid. Identification of EMG signals using discriminant analysis and SVM classifier. *Expert systems with Applications*, v. 39, n. 1, p. 44-47, 2012.
- [12] DRAMBLE, R. P. microSD Card Benchmarks. 2020. [Online; Acessado em 01 março 2021]. Disponível em: <<https://www.pidramble.com/wiki/benchmarks/microsd-cards>>.

APÊNDICE IMAGENS

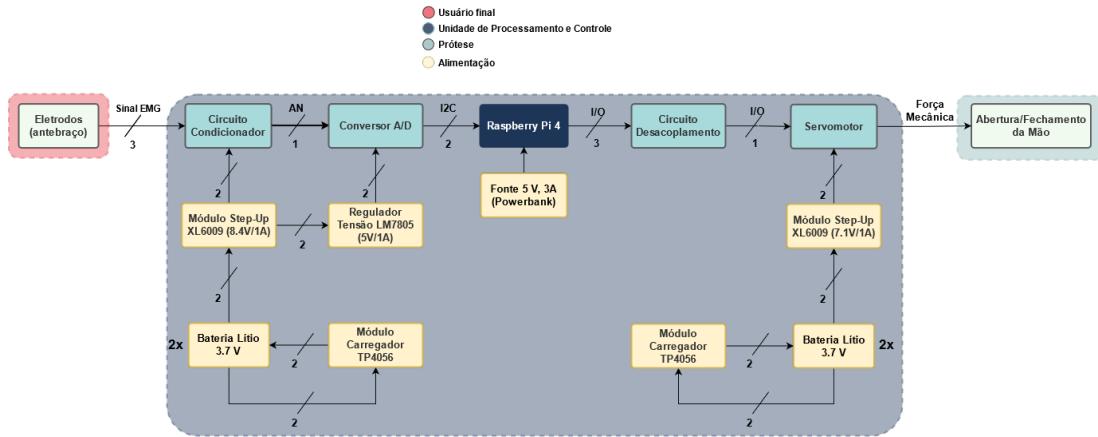


Figura 18: Diagrama de Blocos da Solução

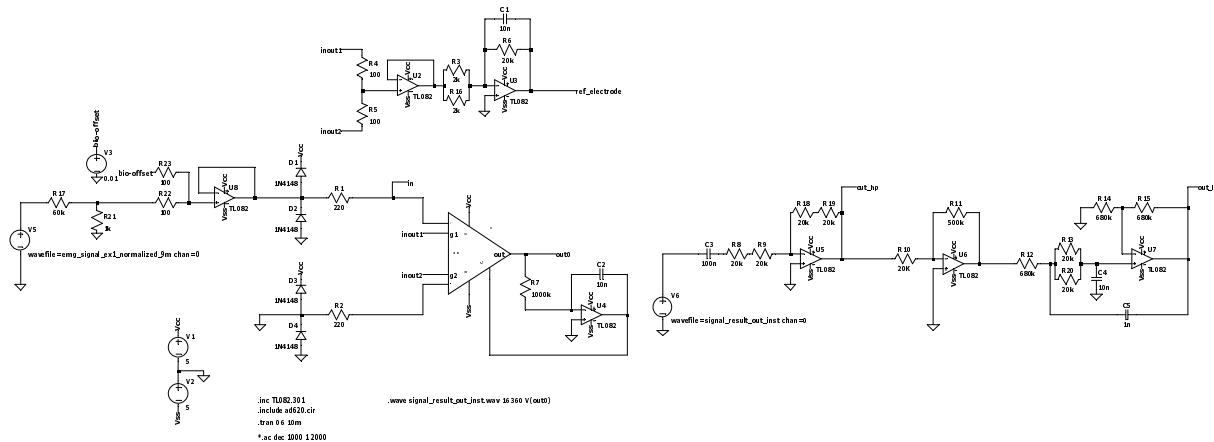


Figura 19: Circuito de captura e condicionamento proposto

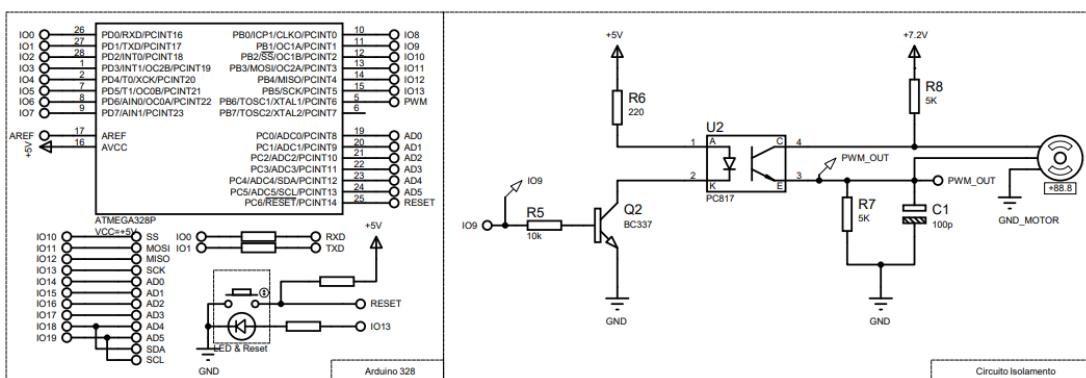


Figura 20: Circuito de desacoplamento proposto



Figura 21: Diagrama de nós e conexões do projeto

APÊNDICE

CÓDIGO FONTE

• Nô adc - /PC2/scripts/adc.py - [Link](#)

```

1 #!/usr/bin/env python3
2 import rospy
3 import serial
4 import struct
5 from std_msgs.msg import Int32
6
7 # def receive(ser):
8 #     k=ser.read(2)
9 #     v=struct.unpack('H', k)
10 #     h=v[0] * 3
11 #     return h
12 global ser
13 ser = serial.Serial('/dev/serial0', 115200) #rfcomm0
14 #ttyUSB0
15 def adc():
16
17     # ser = serial.Serial('/dev/ttyUSB0', 115200) # rfcomm0 #ttyUSB0
18     global ser
19     pub = rospy.Publisher('dados', Int32, queue_size=10)
20     rospy.init_node('adc', anonymous=True)
21     rate = rospy.Rate(1600)
22
23     while not rospy.is_shutdown():
24         try:
25             #k=ser.read(2)
26             k=ser.readline()
27             #v=struct.unpack('H', k)
28             #print(k)
29             v=int(k)
30             dados=v#v[0] * 3
31             #print(dados)
32             if dados > 13000:
33                 print("Dados Incorretos")
34                 rospy.signal_shutdown('Quit')
35             pub.publish(Int32(dados))
36         except serial.SerialException:
37             print("Erro captura")
38             rospy.signal_shutdown('Quit')
39             #rate.sleep()
40     if __name__ == '__main__':
41         try:
42             adc()
43         except rospy.ROSInterruptException:
44             ser.close()

```

• Nô accumulator - /PC2/scripts/accumulator.py - [Link](#)

```

1 #!/usr/bin/env python3
2
3 import rospy
4 from std_msgs.msg import Int32
5 from std_msgs.msg import Int32MultiArray
6
7 global window
8 global counter
9 global pub
10 global len_window
11 len_window = 800
12 window=Int32MultiArray()
13 window.data = len_window * [0]
14 counter = 0
15 pub = rospy.Publisher('data_window', Int32MultiArray,
16                         queue_size=10)
17 def callback(data):
18     global window
19     global counter
20     global pub
21     global len_window

```

```

21
22     window.data.pop(0)
23     window.data.append(data.data)
24
25     if counter == len_window/2:
26         counter = 0
27         pub.publish(window)
28     else:
29         counter += 1
30
31 def accumulator():
32     rospy.init_node('accumulator', anonymous=True)
33     rospy.Subscriber('dados', Int32, callback)
34     rospy.spin()
35
36 if __name__ == '__main__':
37     try:
38         accumulator()
39     except rospy.ROSInterruptException:
40         pass

```

• Nô extraction - /PC2/scripts/extraction.py - [Link](#)

```

1 #!/usr/bin/env python3
2
3 import rospy
4 import numpy as np
5 from statsmodels.tsa.ar_model import AR
6 from scipy.signal import *
7 from std_msgs.msg import Float64MultiArray
8 from std_msgs.msg import Int32MultiArray
9
10
11 global pub
12 global parameters
13
14 parameters=Float64MultiArray()
15 parameters.data = 4 * [0]
16 counter = 0
17 pub = rospy.Publisher('extracted', Float64MultiArray,
18                         queue_size=10)
19 def callback(data):
20     global pub
21     global parameters
22     parameters.data = 8 * [0]
23     x=data.data - np.mean(data.data)
24     N = len(x)
25     # Valor medio absoluto
26     parameters.data[0] = np.mean(np.abs(x))
27     # Valor RMS
28     parameters.data[1] = np.sqrt(np.sum(x**2)/len(x))
29     # Waveform length
30     parameters.data[2] = np.sum(np.abs(np.diff(x)))
31     # Variancia
32     parameters.data[3] = np.var(x)
33     # Average Amplitude Change
34     Y=0;
35     X=np.array(x);
36     for i in range(N-1):
37         Y=Y+np.abs(X[i+1]-X[i])
38     parameters.data[4]=Y/N;
39     #Energia em bandas
40     tot = np.linalg.norm(X)**2
41     #print tot
42     num=[2.44897128e-09, 3.67345692e-08, 2.57141984e
43     -07, 1.11428193e-06,
44     3.34284579e-06, 7.35426075e-06, 1.22571012e
45     -05, 1.57591302e-05,
46     1.57591302e-05, 1.22571012e-05, 7.35426075e
47     -06, 3.34284579e-06,
48     1.11428193e-06, 2.57141984e-07, 3.67345692e
49     -08, 2.44897128e-09]

```

```

45 den=[ 1.00000000e+00, -8.94097666e+00,
46     3.81274598e+01, -1.02595239e+02,
47     1.94421048e+02, -2.74396921e+02,
48     2.97560724e+02, -2.52180163e+02,
49     1.68238453e+02, -8.82785479e+01,
50     3.61095963e+01, -1.12999558e+01,
51     2.61723273e+00, -4.23343031e-01,
52     4.27414269e-02, -2.02960157e-03]
53 zi = lfilter_zi(num, den)
54 y, _ = lfilter(num, den, X, zi=zi*X[0])
55 y1 =np.linalg.norm(y)**2
56 parameters.data[5]=y1/tot*100
57
58 num = [ 1.31247236e-05, 0.00000000e+00,
59     -1.18122513e-04, 0.00000000e+00,
60     4.72490051e-04, 0.00000000e+00,
61     -1.10247679e-03, 0.00000000e+00,
62     1.65371518e-03, 0.00000000e+00,
63     -1.65371518e-03, 0.00000000e+00,
64     1.10247679e-03, 0.00000000e+00,
65     -4.72490051e-04, 0.00000000e+00,
66     1.18122513e-04, 0.00000000e+00,
67     -1.31247236e-05]
68
69 den = [ 1.00000000e+00, -9.68593127e+00,
70     4.69901672e+01, -1.50739975e+02,
71     3.56909439e+02, -6.60502621e+02,
72     9.88073702e+02, -1.21982331e+03,
73     1.25851221e+03, -1.09246913e+03,
74     7.99555480e+02, -4.92238937e+02,
75     2.53142229e+02, -1.07374086e+02,
76     3.67952917e+01, -9.85240606e+00,
77     1.94788749e+00, -2.55100758e-01,
78     1.68137718e-02]
79
80 zi = lfilter_zi(num, den)
81 y, _ = lfilter(num, den, X, zi=zi*X[0])
82 y1 =np.linalg.norm(y)**2
83 parameters.data[6]=y1/tot*100
84
85 num = [ 9.17785774e-09, 0.00000000e+00,
86     -1.28490008e-07, 0.00000000e+00,
87     8.35185054e-07, 0.00000000e+00,
88     -3.34074022e-06, 0.00000000e+00,
89     9.18703559e-06, 0.00000000e+00,
90     -1.83740712e-05, 0.00000000e+00,
91     2.75611068e-05, 0.00000000e+00,
92     -3.14984077e-05, 0.00000000e+00,
93     2.75611068e-05, 0.00000000e+00,
94     -1.83740712e-05, 0.00000000e+00,
95     9.18703559e-06, 0.00000000e+00,
96     -3.34074022e-06, 0.00000000e+00,
97     8.35185054e-07, 0.00000000e+00,
98     -1.28490008e-07, 0.00000000e+00,
99     9.17785774e-09]
100
101 den = [ 1.00000000e+00, -3.75618537e+00,
102     1.49160032e+01, -3.64590663e+01,
103     8.62006102e+01, -1.58929766e+02,
104     2.79121488e+02, -4.14409137e+02,
105     5.84695352e+02, -7.23468409e+02,
106     8.51082988e+02, -8.94437527e+02,
107     8.94426960e+02, -8.06151007e+02,
108     6.91695461e+02, -5.36216565e+02,
109     3.95715767e+02, -2.63000210e+02,
110     1.66317800e+02, -9.38008369e+01,
111     5.03045394e+01, -2.35910290e+01,
112     1.05214957e+01, -3.94854217e+00,
113     1.41484447e+00, -3.92253746e-01,
114     1.05899399e-01, -1.73361510e-02,
115     3.06405453e-03]
116
117 zi = lfilter_zi(num, den)
118 y, _ = lfilter(num, den, X, zi=zi*X[0])
119 y1 =np.linalg.norm(y)**2
120 parameters.data[7]=y1/tot*100
121
122 # Autorregressive Model
123 # model = AR(np.array(x)/1.)
124 # model_fit = model.fit()
125 # parameters.data = parameters.data + model_fit.
126 # params.tolist()
127 #print len(parameters.data)
128 pub.publish(parameters)
129
130
131 def extraction():
132     rospy.init_node('extraction', anonymous=True)
133     rospy.Subscriber('data_window', Int32MultiArray,
134                      callback)
135     rospy.spin()
136
137 if __name__ == '__main__':
138     try:
139         extraction()
140     except rospy.ROSInterruptException:
141         pass

```

• **Nó predictor** - /PC2/scripts/predictor.py - Link

```

1 #!/usr/bin/env python3
2 import rospy
3 import rospkg
4 import numpy as np
5 import pickle
6 from statsmodels.tsa.ar_model import AR
7 from sklearn.datasets import make_blobs
8 from sklearn.model_selection import train_test_split
9 from matplotlib import pyplot as plt
10 from sklearn.svm import *
11 from numpy import convolve as conv
12 from scipy.signal import *
13 from sklearn.svm import SVC
14 from matplotlib.colors import ListedColormap
15 from sklearn.model_selection import train_test_split
16 from sklearn.preprocessing import StandardScaler
17 from std_msgs.msg import Int32
18 from std_msgs.msg import Float64MultiArray
19
20
21 global pub
22 global svm
23 global sc
24 rospack = rospkg.RosPack()
25 svm = pickle.load(open(rospack.get_path("embedded_emg_prosthesis")+"/scripts/maquina.sav",
26                         'rb'))
27 sc = pickle.load(open(rospack.get_path("embedded_emg_prosthesis")+"/scripts/scaler.sav",
28                         'rb'))
29 pub = rospy.Publisher('servo/action', Int32,
30                       queue_size=10)
31
32 def callback(data):
33     global pub
34     global svm
35     global sc
36     dados =np.array(data.data).reshape(1,-1)
37     params = sc.transform(dados)
38     prediction = svm.predict(params)
39     prediction = int(prediction)
40     pub.publish(Int32(prediction))
41
42 def predictor():
43     rospy.init_node('predictor', anonymous=True)

```

```

42    rospy.Subscriber('extracted', Float64MultiArray, callback)
43    rospy.spin()
44
45 if __name__ == '__main__':
46     try:
47         predictor()
48     except rospy.ROSInterruptException:
49         pass

```

• Nó control - /PC2/scripts/servo_node.py - Link

```

1 #!/usr/bin/env python3
2
3 import rospy
4
5 # Import ROS msgs
6 from std_msgs.msg import Float64
7 #from std_msgs.msg import UInt8
8 from std_msgs.msg import String
9 from std_msgs.msg import Int32
10
11 # Import utilities
12 import RPi.GPIO as GPIO
13 import time
14 import rosnode
15
16 # Parameters
17 global servoPIN # servomotor pin in raspberry GPIO
18 global minAngle # minimun angle
19 global maxAngle # maximum angle
20 global deg_0_pulse # pulse period for 0 degree
21 global deg_180_pulse # pulse period for 180 degree
22 global f # frequency
23
24 # Calculation parameters
25 global period # period
26 global k # constant
27 global deg_0_duty # 0 degree position
28 global pulse_range # pulse total range
29
30 # Servopin
31 servoPIN = 17
32
33 # Max/min Angle for open/close prothesys action
34 minAngle = 60
35 maxAngle = 150
36
37 # Ajuste estes valores para obter o intervalo
# completo do movimento do servo
38 deg_0_pulse = 0.5
39 deg_180_pulse = 2.5
40 f = 50.0
41
42 # Servo parameters calculation
43 period = 1000/f
44 k = 100/period
45 deg_0_duty = deg_0_pulse*k
46 pulse_range = deg_180_pulse - deg_0_pulse
47 duty_range = pulse_range * k
48
49
50 def SetAngle(pwm, angle):
51     duty = deg_0_duty + (angle/180.0) * duty_range
52     GPIO.output(servoPIN, True)
53     pwm.ChangeDutyCycle(duty)
54     # time.sleep(1)
55     # GPIO.output(servoPIN, False)
56     # pwm.ChangeDutyCycle(0)
57
58 def control_callback(data):
59
60     # Prothesis hand close
61     if data == Int32(1):
62         SetAngle(p, float(minAngle))

```

```

63
64     # Prothesis hand open
65     elif data == Int32(-1):
66         SetAngle(p, float(maxAngle))
67
68     # Prothesis hand half open
69     elif data == Int32(3):
70         SetAngle(p, float(90))
71
72
73 def main():
74     global p
75
76     # Init servo node
77     rospy.loginfo('Inicializing node')
78     rospy.init_node('servo', anonymous=False)
79
80     # Initializing GPIO pin
81     GPIO.setmode(GPIO.BCM)
82     GPIO.setup(servoPIN, GPIO.OUT)
83     p = GPIO.PWM(servoPIN, 50) # GPIO 17 for PWM
# with 50Hz
84     p.start(0) # Initialization
85
86     # list subscribed topics
87     rospy.loginfo('Setting up topics')
88     sub = rospy.Subscriber('servo/action', Int32,
89                           callback=control_callback)
90
91     rospy.spin()
92     p.stop()
93     GPIO.cleanup()
94
95     # # loop rate (in hz)
# rate = rospy.Rate(5)
96
97     # # Node Loop
98     # while not rospy.is_shutdown():
99     #     angle = input("Enter angle (0 a 180): ")
# SetAngle(p, float(angle))
100
101     #     # Wait for the next cycle
#     rate.sleep()
102
103     #     # Wait for the next cycle
#     rate.sleep()
104
105
106 if __name__ == '__main__':
107     try:
108         main()
109     except rospy.ROSInterruptException:
110         pass

```