GABRIEL CESARIO GOMES

CONTROLE DE MANIPULADOR ROBÓTICO ARTICULADO 3DOF POR MEIO DE APLICATIVO DESENVOLVIDO EM NODE.JS E REACT NATIVE

LAVRAS – MG

2021

GABRIEL CESARIO GOMES

CONTROLE DE MANIPULADOR ROBÓTICO ARTICULADO 3DOF POR MEIO DE APLICATIVO DESENVOLVIDO EM NODE.JS E REACT NATIVE

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do curso de Engenharia de Controle e Automação, para obtenção do título de Bacharel.

Prof. Dr. Leonardo Silveira Paiva
Orientador

LAVRAS-MG

2021

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Gomes, Gabriel Cesario.

Controle de manipulador robótico articulado 3DOF por meio de aplicativo desenvolvido em Node.js e React Native / Gabriel Cesario Gomes. - 2021.

60 p.

Orientador(a): Leonardo Silveira Paiva.

TCC (graduação) - Universidade Federal de Lavras, 2021.

Bibliografia.

1. Manipuladores. 2. Nodejs. 3. React Native. I. Paiva, Leonardo Silveira. II. Título.

GABRIEL CESARIO GOMES

CONTROLE DE MANIPULADOR ROBÓTICO ARTICULADO 3DOF POR MEIO DE APLICATIVO DESENVOLVIDO EM NODE.JS E REACT NATIVE
CONTROL OF ROBOTIC MANIPULATOR ARTICULATED 3DOF BY MEAND OF APPLICATION DEVELOPED IN NODE.JS AND REACT NATIVE

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do curso de Engenharia de Controle e Automação, para obtenção do título de Bacharel.

APROVADO EM 07 de junho de 2021

Dr. Leonardo Silveira Paiva UFLA

Dr. Carlos Renato Borges dos Santos IFTM

MSc. Antonio Mendes Magalhães Junior UFLA

Prof. Dr. Leonardo Silveira Paiva

Orientador

LAVRAS-MG

2021

AGRADECIMENTOS

Durante toda a minha trajetória acadêmica, enfrentei diversos desafios que me ajudaram a evoluir como pessoa e como profissional. A finalização do presente trabalho vem com a sensação de dever cumprido apesar de todos os empecilhos encontrados no caminho.

Agradeço primeiramente aos meus pais, Aloísio Cupertino Gomes e Juliana Cesário Gomes pela compreensão e amor durante todo o meu progresso, vocês são o meu início, meio e fim e eu simplesmente não posso descrever em palavras o meu amor por vocês. À minha irmã, Ana Júlia Gomes, agradeço o apoio nos momentos difíceis e por estar ao meu lado nos meus piores momentos, você merece o mundo e nada pode te derrubar. Sem me esquecer dos demais membros da família, meu avô e avós, tios e tias, primos e primas, eu agradeço pelo carinho e exemplo, obrigado pela presença e carinho, vocês significam muito para mim e sempre é uma honra estar na presença de vocês.

Aos meus amigos sou grato por me ajudarem a trilhar o caminho que escolhi. Agradeço aos meus amigos de longa data e aos que encontrei pelo caminho, aos meus companheiros da República 401, aos membros da Robótica Jr. e meus amigos e parceiros de trabalho da Gafit, espero encontrar todos vocês novamente em breve para que possamos trocar boas notícias e colocar as conversas em dia. Sinto saudades de todos, sem exceções.

Ao Prof. Dr. Leonardo Paiva agradeço por me permitir aprimorar meus conhecimentos e pela paciência durante essa jornada.

À Deus agradeço por minha vida e existência.

EPÍGRAFE

“A persistência é o caminho do êxito.”
(Charles Chaplin)

RESUMO

Considerando a expansão de indústrias e de empresas, um passo importante a ser dado para sua modernização será a possibilidade de controle e monitoramento de suas atividades de quaisquer locais do mundo. Para tal, novas tecnologias são criadas e novos *softwares* são desenvolvidos a fim de possibilitar tal feito. O presente trabalho utiliza as linguagens *Node.js* e *React Native* para criação de um aplicativo *Android* capaz de controlar um manipulador robótico 3DOF simulando assim um equipamento industrial de uso mais complexo. Para que o desenvolvimento consiga atender às expectativas, primeiro são feitos os cálculos de modelagem do manipulador, a partir dos parâmetros de Denavit-Hartenberg para que se possa comparar os resultados encontrados com as posições que serão configuradas no aplicativos. O aplicativo também apresentará outras funcionalidades para demonstrar a capacidade que um simples *software* tem de atender as demandas da indústria e simular situações de trabalho de um manipulador industrial.

Palavras-chave: Arduino. Protocolo Firmata. Processo unificado.

ABSTRACT

Considering the expansion of industries and companies, an important step to be taken for its modernization will be the possibility of controlling and monitoring its activities from any location in the world. To this end, new technologies are created and new softwares are developed in order to make this possible. The present work uses the Node.js and React Native languages to create an Android application capable of controlling a 3DOF robotic manipulator, simulating more complex industrial equipment. In order for the development to meet the expectations, first the modeling calculations of the manipulator are made, based on the Denavit-Hartenberg parameters so that we can compare the results found with the positions that will be configured in the application. The application will also present other features to demonstrate the ability that a simple software has to meet the demands of the industry and to simulate work situations of an industrial manipulator.

Keywords: Arduino. Firmata Protocol. Rational Unified Process.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CSS – *Cascading Style Sheets*

DOF – *Degrees of Freedom*

HTML – *HiperText Markup Language*

IOT – *Internet of Things*

JSX – JavaScript XML

MVC – *Model-View-Controller*

PLC – *Programmable Logic Controller*

PWM – *Pulse Width Modulation*

RUP – Processo Unificado

UI – *User Interface*

XML – *eXtensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

Foi em 2011 na Alemanha que o termo "*Industrie* 4.0" (ou em português Indústria 4.0) foi apresentado pela primeira vez, referindo-se ao que seria a Quarta Revolução Industrial (DRATH; HORCH, 2014). Ela tem seu alicerce em tecnologias como a Internet das Coisas *(Internet of Things – IOT*) e objetos inteligentes, que objetiva construir sistemas com maior capacidade de autogestão e com maior possibilidade de customização dos produtos sem que haja perdas das vantagens da produção em massa (LASI et al., 2014). Por fim, a Indústria 4.0 prevê a integração entre humanos e máquinas, mesmo que em locais distantes, formando grandes redes e fornecendo serviços e produtos de forma autônoma (SILVA; SANTOS FILHO; MIYAGI, 2015).

Uma maneira interessante de se formar essa integração é com o auxílio de um aplicativo para *smartphone*. Segundo dados divulgados pela Anatel[1] (Agência Nacional de Telecomunicações), em Dezembro de 2020, já são mais assinaturas de serviços de telefonia móvel do que pessoas no Brasil, com aproximadamente 234,1 milhões de assinaturas no total. Entretanto, o que mais surpreende não são os números, mas sim as possibilidades e oportunidades que o aparelho traz ao reconfigurar as dimensões de espaço e tempo ao toque de um botão, cooperando para a criação de uma "Sociedade em Rede" (CASTELLS, 2013).

Pensando nas possiblidades que o *smartphone* traz, o seu uso no ambiente educacional criou pesquisas como a "*Teaching via mobile phone: a case study on Malaysian teachers' technology acceptance and readiness" e a* "*The current perspectives, theories and practices of mobile learning"* (Ismail et al., 2013; Keskin e Metcalf, 2011) que apresentam formas de utilização dos dispositivos em salas de aula pelos alunos, criando assim um primeiro passo para se pensar em sua utilização no ambiente industrial. Dito isso, o mercado está cada vez mais imerso na automação de suas atividades. Segundo a *Forrester,* em matéria de O Globo vinculada em 2018, até 2021, a automação substituirá o equivalente a 4,3 milhões de profissionais em todo o mundo.

A ideia do presente trabalho é apresentar um protótipo que visa integrar um aplicativo de celular *Android* a um manipulador robótico, simulando, a utilização do mesmo a um componente industrial utilizado amplamente em cadeias de produção.

[1] Disponível em <https://www.anatel.gov.br/paineis/acessos/telefonia-movel>. Acesso 24 abr. 2021.

2. OBJETIVOS

Tendo em vista a expansão do uso dos telefones celulares e o potencial de trabalho dos manipuladores robóticos, este trabalho tem como objetivo desenvolver um manipulador com 3 graus de liberdade que seja capaz de alcançar e mover objetos entre diversos pontos dentro do seu espaço de trabalho por meio de um aplicativo em *Node*.*js* no seu *back-end* e *React Native* em seu *front-end*.

Dentre os objetivos específicos, estão:

- Apresentar os benefícios da integração de equipamentos mecânicos a um *software* controlado por aplicativo;
- Apresentar o estudo de um manipulador serial e elementos finais de controle;
- Permitir ao usuário o controle visual de um manipulador;
- Apresentar a interface criada com o objetivo de movimentar o manipulador;
- Apresentar a metodologia utilizada para o desenvolvimento do aplicativo;
- Apresentar o código *back-end* e *front-end* em *Node*.js e *React Native*, respectivamente, para a criação da interface;
- Comparar os resultados calculados e os resultados medidos com o aplicativo.

3. FUNDAMENTAÇÃO TEÓRICA

No levantamento bibliográfico realizado sobre manipuladores foram observadas diversas pesquisas sobre o uso deles em sistemas industriais auxiliando na produção de diversos equipamentos e produtos. Entretanto, quanto a sua integração a um aplicativo de celular teve-se uma enorme redução nos resultados encontrados, trazendo essa integração como um fator diferencial para o desenvolvimento do presente projeto.

3.1. Manipuladores

Foi com o robô de *Stanford* (1969), um dos precursores dos manipuladores que são utilizados atualmente na indústria, mostrado na Figura 1, que se deu início a sua utilização em massa na indústria. Sua principal característica era ter motores que controlam seu movimento conectados diretamente nas suas junções. Isso possibilitou movimentos mais rápidos e precisos que eram demandados em tarefas mais complexas (GAMERO, 2018).

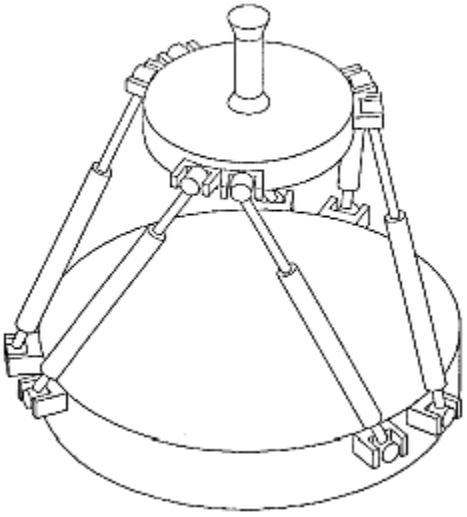
Figura 1 – Robô de *Stanford* (1969)

Fonte: *The Stanford Infolab,* 2021.

No que diz respeito à estrutura física dos robôs, ele são formados por um conjunto de partes interligados por meio de juntas e a configuração dessas junções dita a configuração de sua estrutura, que pode ser serial ou paralela (LOPES, 2002).

Segundo Faveri (2013), na estrutura paralela as juntas formam uma cadeia cinemática fechada e independente, se posicionam em uma base fixa e utilizam atuadores lineares que permitem a mudança do comprimento das juntas para conexão com uma plataforma móvel, como mostrado na Figura 2. De modo a orientar o atuador final de maneira desejada, suas principais características incluem grande rigidez, elevada capacidade de carga e boa capacidade de posicionamento.

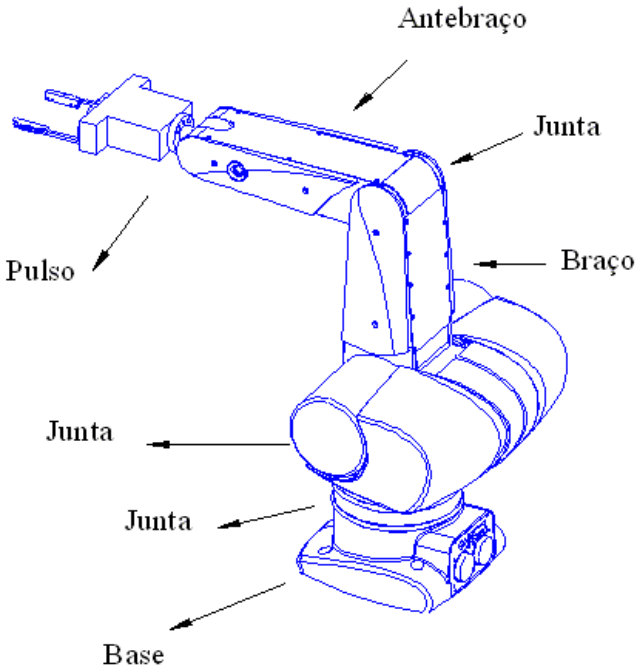
Figura 2 – Plataforma de *Stewart*, estrutura de manipulador paralelo



Fonte: LOPES, 2002.

Lopes (2002) afirma que na estrutura serial, as juntas formam uma cadeia cinemática aberta e dependente, uma das extremidades é ligada a uma base fixa, enquanto outra extremidade possui um atuador final, chamado de "*end-effector",* que está diretamente ligado a algum tipo de garra ou ferramenta que realizará a tarefa exigida pelo sistema. Pimenta (2009) salienta que esta estrutura se assemelha a um braço humano, visto na Figura 3, e suas características envolvem grande espaço de trabalho, baixa rigidez e fraca capacidade de posicionamento se comparada com a estrutura paralela.

Figura 3 – Estrutura do manipulador serial

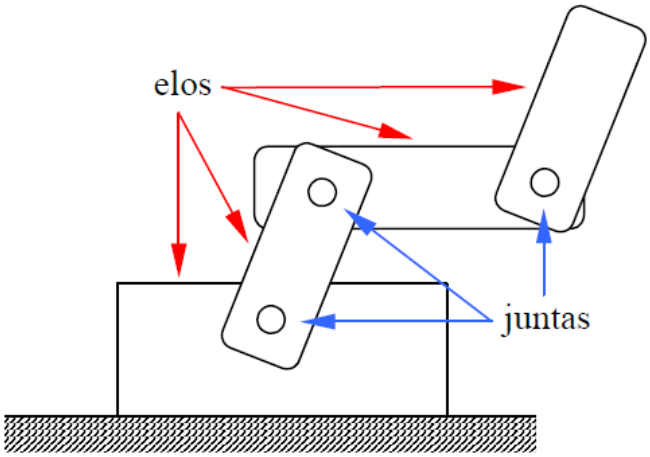


Fonte: CARRARA, 2009.

Por essa característica de se assimilar com a anatomia do braço humano, o manipulador serial também é conhecido como braço robótico. De acordo com Siciliano et al (2010) consiste em uma sequência de corpos rígidos chamados de elos que são unidos por juntas de movimento relativo, onde são posicionados os atuadores que são instruídos por um controlador.

Segundo Pimenta (2009), a junta pode ser do tipo Rotacional (R), ou seja, gira em torno de um eixo de rotação ou Prismática (P), com o movimento linear composto por duas hastes que deslizam entre si e são a combinação dessas juntas seriais que ditam a classificação do seu tipo, também chamado de estrutura cinemática. A Figura 4 mostra um esquema sequencial de elos e juntas de um braço robótico.

Figura 4 – Esquema sequencial de elos e juntas



Fonte: CARRARA, 2009.

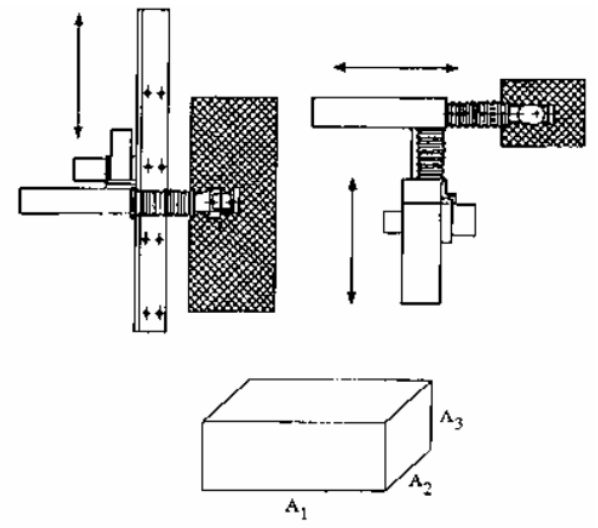
Além das juntas e elos, se tem mais um atributo essencial a construção de um manipulador, sendo ele os graus de liberdade (Degrees of Freedom - DOF). O número de DOF de um manipulador serial determina a quantidade de movimento dele, e está associado ao número de variáveis posicionais independentes que determinam a posição dos elos de forma única. Esse número é dado pela somatória dos DOF das juntas (CARRARA, 2009).

Segundo Siciliano et al. (2010), a estrutura de elos e juntas determinam o espaço de trabalho do braço robótico, que segundo o autor é a denominação da porção do ambiente onde o *end-effector* alcança, respeitando suas restrições. Em acréscimo, Nof (1999) afirma existir o espaço de junta, ele é formado por todos os possíveis vetores dos valores da junta e a dimensão deste vetor é igual ao número de DOF do manipulador.

Aproveitando a definição das juntas de Pimenta (2009) e a descrição do espaço de trabalho de Siciliano et al. (2010) e Nof (1999), as Figuras 5 a 9 apresentam uma visualização detalhada de sua movimentação.

Começando pela Figura 5 que apresenta o manipulador Cartesiano, cuja configuração das juntas se dá por três juntas prismáticas, ou seja, PPP (SANTOS, 2003).

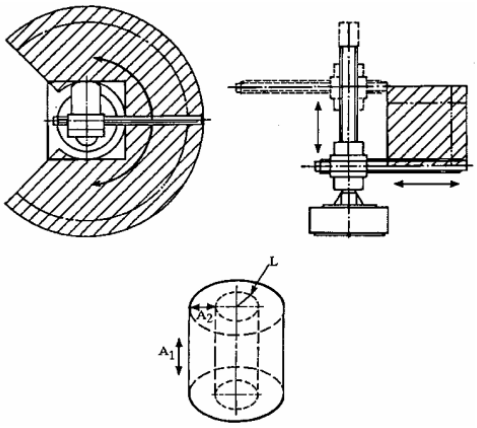
Figura 5 – Espaço de trabalho da estrutura cartesiana



Fonte: SANTOS, 2003.

Em seguida, na Figura 6, se tem o manipulador de estrutura cilíndrica, composto a partir da base por duas juntas do tipo prismática e uma do tipo rotacional, configuração chamada de PPR (SANTOS, 2003)

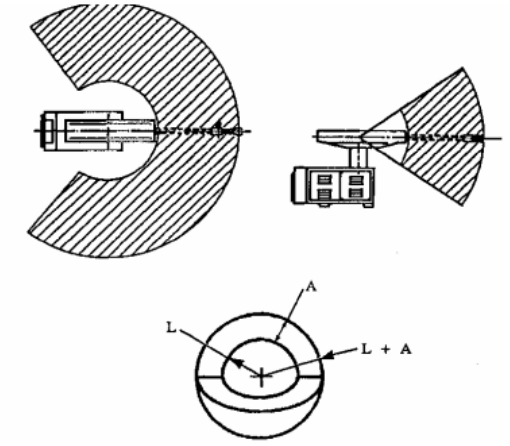
Figura 6 – Espaço de trabalho da estrutura cilíndrica



Fonte: SANTOS, 2003.

Na Figura 7, com a configuração dadas por uma junta prismática e duas rotacionais, ou seja, configuração PRR, tem-se o manipulador esférico, também chamado de manipulador polar (SANTOS, 2003)

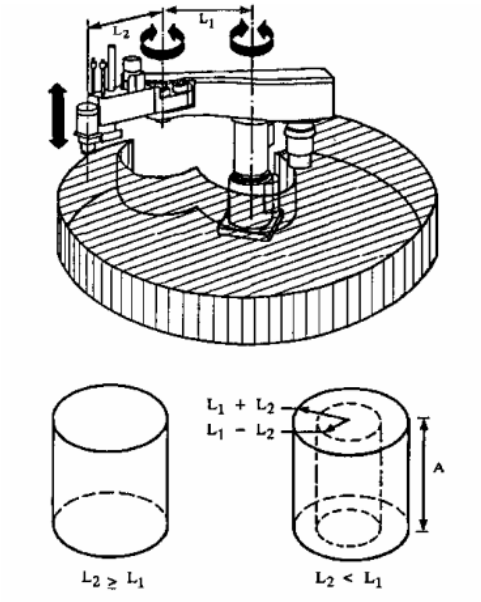
Figura 7 – Espaço de trabalho da estrutura esférica



Fonte: SANTOS, 2003.

Assim como o manipulador esférico, o manipulador SCARA, mostrado na Figura 8, possui uma junta prismática e duas rotacionais, porém sua configuração traz as juntas rotacionais partindo da base, sendo sua configuração dada por RRP (SANTOS, 2003).

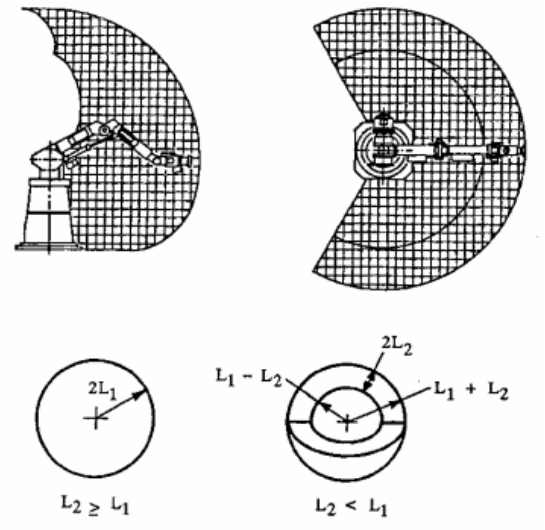
Figura 8 – Espaço de trabalho da estrutura SCARA



Fonte: SANTOS, 2003.

Por último, apresentado na Figura 9, o manipulador articulado formado por três juntas rotacionais e alvo de estudo do presente trabalho, sua configuração é dada por RRR (SANTOS, 2003).

Figura 9 – Espaço de trabalho da estrutura articulada



Fonte: SANTOS, 2003.

A utilidade dos manipuladores é tamanha que, em relatório apresentado em 2019 na conferência de Shangai, a Federação Internacional de Robótica afirma que em 2018 existiam 2,4 milhões de robôs operando mundialmente e a previsão é que até 2022 esse valor tenha aumentado para 4 milhões (*International Federation of Robotics*, 2019).

3.2. Cinemática

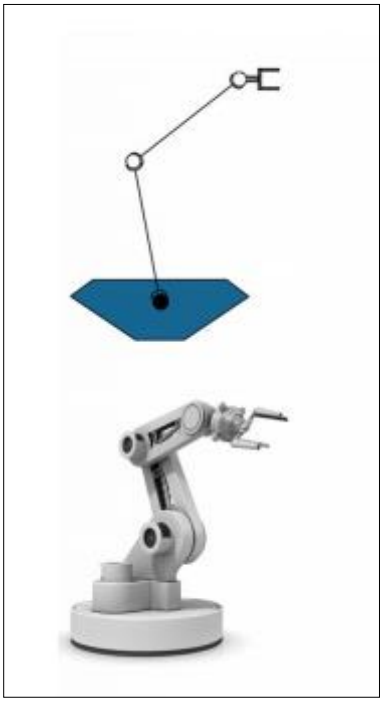
Segundo Mohammed e Sunar (2015), um passo decisivo em qualquer sistema robótico é a análise e modelagem da cinemática do manipulador, que pode ser dividida em cinemática direta e inversa. A primeira se refere a encontrar a postura e posição do *end-effector* para as coordenadas das juntas dadas e a segunda é a determinação das variáveis das juntas em termos da postura e posição dele. Nos tópicos seguintes serão explicados um pouco mais sobre elas.

3.2.1. Cinemática Direta

Segundo Carrara (2006), a cinemática direta constitui na necessidade de ir do espaço de junta para o espaço cartesiano. Dessa forma o autor salienta o que foi dito anteriormente; será necessário determinar, a partir do ângulo das juntas, a posição cartesiana da extremidade do braço, ou seja, do *end-effector*.

Dombre e Khalil (2007) explicam que a modelagem sistemática de robôs requer um método para descrição de sua estrutura, sendo que o mais utilizado é o de Denavit-Hartenberg (DH), desenvolvido para estruturas simples de robôs seriais, demonstrado na Figura 10.

Figura 10 – Exemplo de robô serial de cadeia aberta



Fonte: YAVUZ, 2009.

Segundo Ranky e Ho (1985), o método DH introduz o conceito de matriz de transformação homogênea, que segundo Yoshikawa (2003) é um conjunto de elementos que indicam rotações e translações necessárias para que um ponto localizado em um conjunto de eixos coordenados seja expresso em outro conjunto de eixos coordenados.

A matriz de translação entre o ponto n e o ponto seguinte n + 1 é apresentada na Equação 3.1, dada por Ranky e Ho (1985), na qual os elementos da matriz p_x, p_y e p_z, são as representações respectivas das coordenadas cartesianas x, y e z.

$$p_n^{n+1} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{3.1}$$

Pensando em um ponto $p = (p_x, p_y, p_z)$ a ser transladado para o ponto $p' = (p'_x, p'_y, p'_z)$, com um deslocamento de $\Delta p = (\Delta x, \Delta y, \Delta z)$, então, segundo McCloy (2013) basta somar os valores do deslocamento à coordenada p, descrita na Equação 3.2.

$$p' = \begin{bmatrix} p_x + \Delta x \\ p_y + \Delta y \\ p_z + \Delta z \end{bmatrix} \tag{3.2}$$

Para rotação, o processo se torna um pouco mais complexo. De acordo com Ranky e Ho (1985), para encontrar a matriz de rotação R de dimensão 3x3, será aplicado a transformação na matriz identidade com relação ao respectivos eixos x, y e z.

Slabaugh (1999) salienta que um passo necessário é a determinação dos ângulos de Euler, dessa forma pode-se estabelecer as colunas da matriz R em relação aos três principais eixos citados por Ranky e Ho.

Dessa forma, o autor apresenta a Equação 3.3a representando R para o ângulo ψ em relação ao eixo x. Para o eixo y e ângulo θ adquire-se R na Equação 3.3b. E R aplicado no eixo z e ângulo ∅ é dado pela Equação 3.3c.

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\psi & -sin\,\psi \\ 0 & sin\,\psi & cos\,\psi \end{bmatrix} \tag{3.3a}$$

$$R_y(\theta) = \begin{bmatrix} cos\,\theta & 0 & sin\,\theta \\ 0 & 1 & 0 \\ -sin\,\theta & 0 & cos\,\theta \end{bmatrix} \quad (3.3b)$$

$$R_z(\emptyset) = \begin{bmatrix} cos\,\emptyset & -sin\,\emptyset & 0 \\ sin\,\emptyset & cos\,\emptyset & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.3c)$$

Os ângulos ψ, θ e ∅ são os ângulos de Euler.

Niku (2015) complementa que diversas transformações lineares podem ser combinadas em uma única matriz que consiste na preservação da distância entre pontos distintos e em sua colinearidade mostrada na Equação 3.4. Siciliano et al. (2010) reforça que conhecendo as matrizes de translação e rotação, pode-se definir a matriz de transformação homogênea H que mapeia a cinemática direta.

$$H = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} & p_x \\ R_{yx} & R_{yy} & R_{yz} & p_y \\ R_{zx} & R_{zy} & R_{zz} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

Spong et al. (2005) apresenta uma forma simplificada da equação apresentada na Equação 3.5, na qual R é a matriz 3x3 de rotação e p é a matriz 3x1 de translação.

$$H = \begin{bmatrix} R & p \\ [0] & 1 \end{bmatrix} \quad (3.5)$$

Tsai (1999) demonstra que cada combinação de rotações e translações ocorridas em uma junta dá origem a uma matriz H. Dessa forma pode-se deduzir que o movimento de um manipulador provoca n transformações homogêneas, na qual n é o número de juntas dele.

A Equação 3.6, dada por Crane e Duffy (2008), demonstra o movimento total do manipulador, indo da base (numerada como 0) ao *end-effector*, ao multiplicar-se as matrizes H.

$$H_0^n = \prod_{i=0}^{n-1} H_i^{i+1} \tag{3.6}$$

Siciliano et al. (2010), Spong et al. (2005) e Ranky e Ho (1985) apresentam de maneira detalhada o método proposto por Denavit e Hatenberg que consiste em descrever o eixo de coordenadas da junta em relação ao eixo anterior, o que resulta em apenas quatro parâmetros $a, \alpha, d \; e \; \theta$, sendo eles:

- a: Comprimento da normal comum;
- α: Ângulo em torno da normal comum de z_{i-1} a z_i;
- d: Distância ao longo de z_{i-1} até a normal comum;
- θ: Ângulo em torno de z_{i-1}, do x_{i-1} até x_i

Segundo Spong et al. (2005), pode-se representar a posição e orientação de cada junta a respeito da junta anterior com uma matriz de transformação homogênea, mostrada na Equação 3.7, que utiliza os quatro parâmetros citados acima.

$$A_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\text{sen}(\theta_i)\cos(\alpha_i) & \text{sen}(\theta_i)\text{sen}(\alpha_i) & a_i\cos(\theta_i) \\ \text{sen}(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\text{sen}(\alpha_i) & a_i\text{sen}(\theta_i) \\ 0 & \text{sen}(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Como já dito por Crane e Duffy (2008) e agora reforçado por Spong et al. (2005) na Equação 3.8, o produto das matrizes A_i^{i-1} demonstra o movimento total do manipulador indo da base até o *end-effector* em função dos parâmetros Denavit-Hartenberg.

$$T_i^0 = \prod_{j=1}^{i} A_j^{j-1} \tag{3.8}$$

3.2.2. Cinemática Inversa

Segundo Penha, Queiroz e Rossini (2018), a cinemática direta leva as posições no espaço de junta para o espaço cartesiano, enquanto a cinemática inversa faz o

caminho contrário. Niku (2015), em conformidade com os autores, diz que é preciso saber quais serão os ângulos das juntas do manipulador, de forma que o *end-effector* se encontre na posição e orientação desejada.

Spong et al. (2005) e Siciliano et al. (2010) entram em conformidade ao discutir que a cinemática inversa costuma ser bem mais complexa que a cinemática direta, pois em sua maioria as equações da mesma são não-lineares, podendo assim existir múltiplas ou infinitas soluções. Siciliano (2010) ainda reitera que existem soluções que não são admissíveis devido à estrutura do robô. Dessa forma Craig (2009) diz que a primeira ação necessária para a cinemática inversa é restringir os pontos ao espaço de trabalho, garantindo que seja alcançável.

Craig (2009) propõe que há dois tipos de solução para a cinemática inversa, analíticas e numéricas. Segundo o autor, os métodos numéricos costumam demandar um custo computacional elevado por se tratar de problemas mais complexos, enquanto os métodos analíticos podem ser algébricos ou geométricos e, mesmo sendo aplicados em problemas mais simples, tem dificuldade em encontrar uma aproximação mais precisa.

Dombre e Kalil (2007) também apresentam o método de Paul, que analisa cada robô individualmente e é bastante utilizado em robôs industriais, já o método de Pieper permite a solução para robôs com seis graus de liberdade, dos quais três juntas (rotacional ou prismática) possuem eixos que se interceptam.

3.3. Dispositivos móveis

De acordo com Mantovani (2005), o primeiro aparelho celular comercial do mundo, o *DynaTAC* 8000X da Motorola, surgiu apenas em 1983. Foi fruto de décadas de pesquisa, do barateamento de microprocessadores e da digitalização das linhas de comunicação que permitiram sua massificação. Com o advento da internet, uma nova comunicação interativa surge, com a oportunidade de enviar mensagens no padrão muitos para muitos, em tempo real ou não (CASTELLS, 2011).

Silva e Santos (2014) apontam que, com a evolução da tecnologia, a funcionalidade do uso de ligações e mensagens via SMS diminuiu drasticamente em decorrência do desenvolvimento de *softwares* mais avançados que permitem o

acesso à internet, com mais recursos e funções. Segundo os autores, o crescimento rápido do mercado de aplicativos vem em conjunto com uma disputa de diferentes plataformas tecnológicas, mesmo que haja limitações em sua distribuição, custo, tempo para desenvolvimento e complexidade.

Dito isso, as fabricantes de celulares implementaram cada uma um sistema operacional, sendo os principais: *Symbian*, como o primeiro a ser utilizado, *Android*, *Windows Mobile* e *iOS* (DAPPER, 2017). Segundo um levantamento da Tecmundo (2018) visto na Figura 11, os sistemas mais utilizados em 2016 eram o Android, mais utilizado em *smartphone* de custo mais acessível e o iOS, exclusivo para uso em iPhones que são produzidos pela Apple.

Figura 11 – Tabela de sistemas operacionais mais vendidos

Operating System	2Q16 Units	2Q16 Market Share (%)	2Q15 Units	2Q15 Market Share (%)
Android	296,912.8	86.2	271,647.0	82.2
iOS	44,395.0	12.9	48,085.5	14.6
Windows	1,971.0	0.6	8,198.2	2.5
Blackberry	400.4	0.1	1,153.2	0.3
Others	680.6	0.2	1,229.0	0.4
Total	**344,359.7**	**100.0**	**330,312.9**	**100.0**

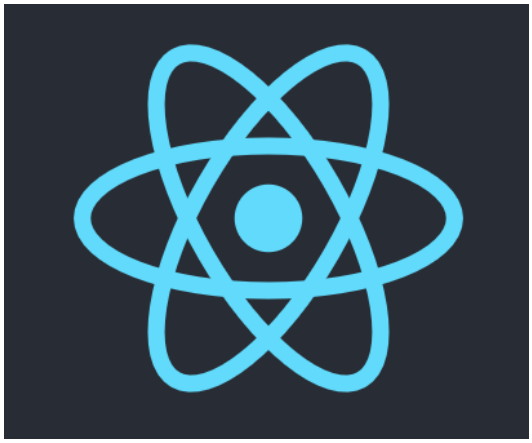
Fonte: Tecmundo, 2018.

3.3.1. Linguagem *React Native*

O *React Native,* cujo logotipo aparece na Figura 12*,* é um *framework JavaScript* para escrever aplicações *Android* e *iOS* renderizando de maneira nativa, na qual é baseado no *React*, uma biblioteca para construção de interfaces que foi desenvolvida pelo Facebook. Em outras palavras, desenvolvedores *Web* agora podem escrever

aplicações móveis que parecem e são verdadeiramente “nativas”, tudo isso com a segurança de uma biblioteca *JavaScript* conhecida (EISENMAN, 2016).

Figura 12 – Logotipo do React Native



Fonte: React Native[2].

Suas aplicações são escritas usando uma mistura de *JavaScript* e XML (*eXtensible Markup Language*), conhecida como JSX (*JavaScript XML*), ela renderizará usando componentes UI (*user interface*), utilizados no desenvolvimento nativo, e poderá acessar os recursos do dispositivo fazendo-a parecer como qualquer outra aplicação *mobile* (EISENMAN, 2016).

A sua principal vantagem em relação aos outros *frameworks* híbridos está no aproveitamento dos meios existentes de renderizar as views, ou seja, ele não se utiliza de *webviews,* ele traduz a linguagem de marcação para elementos UI nativos. Em adição a isso, o *React Native* trabalha separado da *thread* principal, de forma a manter a alta performance sem sacrificar sua aptidão (EISENMAN, 2016).

3.3.2. Linguagem *Node.js*

O *Node.js,* logo na Figura 13, desenvolvido originalmente em 2009 por Ryan Dahl, a linguagem interpreta o *JavaScript* usando o *engine* V8 do *Google Chrome* e evoluiu como um verdadeiro ambiente *runtime cross-plataform* para desenvolvimento de uma ampla variedade de aplicações. Do ponto de vista arquitetural, caracteriza-se por utilizar um modelo direcionado ao eventos, onde todas as operações I/O (*i.e,*

[2] Disponível em <https://reactnative.dev/>. Acesso 12 abr. 2021.

entradas/saídas) são efetuadas de forma assíncrona para evitar bloqueios desnecessários. Atualmente, o seu ecossistema de pacotes disponibiliza o maior número de bibliotecas *open-source* existentes no mundo, que podem ser utilizadas em qualquer aplicação por meio do utilitário *npm* (*Node.js Package Manager*) (ABREU, 2016).

Figura 13 – Logotipo do *Node.js*

Fonte: *Node.js*[3].

Embora interprete o *JavaScript* usando o *engine* V8 do Google, a biblioteca não se limita somente a isso e recorre a algumas das técnicas mais avançadas na área de compiladores. Com isso, a plataforma aproveitou para efetuar otimizações extremamente úteis quando se pensa em sua execução fora do *browser*, como por exemplo a manipulação de dados binários. Internamente, a sua arquitetura recorre ao chamado ciclo de eventos (*event loop*) para permitir a fácil criação de aplicações seguras e escaláveis, optando assim pela redução da complexidade inerente à criação de aplicações sem que isso implique em uma penalização no nível da sua performance (ABREU, 2016).

Com a explosão da Web nos dias de hoje, o *JavaScript* vem se tornando a linguagem da moda, as aplicações mais complexas e a possibilidade de reutilização de conceitos, recomendações e código entre cliente e servidor tornou-se uma mais-valia que chamou a atenção dos programadores. (ABREU, 2016).

[3] Disponível em <https://nodejs.org/en/>. Acesso 12 abr. 2021.

4. MATERIAIS E MÉTODOS

Após entender os fundamentos que definem os manipuladores e as linguagens de programação utilizadas, foi necessário decidir qual manipulador utilizar e como realizar a integração entre o *hardware* e o *software* do projeto. Por isso esse capítulo apresenta os materiais e métodos utilizados no desenvolvimento, tais como detalhes da construção do protótipo, bibliotecas utilizadas no desenvolvimento e comunicação entre as partes desenvolvidas.

4.1. Construção do protótipo

Como o principal elemento de automação e controle do projeto se trata de um aplicativo de celular, a escolha do manipulador foi realizada por meio de uma pesquisa de mercado em busca de um kit didático básico para manipuladores robóticos. Sendo assim, foi levado em consideração a sua capacidade de integração com o projeto e o seu custo financeiro.

4.1.1. Escolha do microcontrolador

O microcontrolador será responsável pelo código que integrará o manipulador ao aplicativo e define, basicamente, toda a estrutura do projeto, pois suas características, sejam elas computacionais ou físicas, podem se tornar potenciais limitadores dependendo de sua aplicação.

Figura 14 – Placa do controlador Arduino Uno

Fonte: Arduino. [4]

Dito isso, o Arduino é uma plataforma de desenvolvimento de microcontrolador de código aberto e contida em uma única placa com *software* e *hardware* fácil de trabalhar e amplamente utilizada. O Arduino Uno tem como base o microcontrolador Atmel Atmega328 e tem clock de 16 MHz, visto na Figura 14. Possui 6 entradas analógicas e 14 I/O digitais (FERDOUSH, XINRONG, 2014). Dentre as digitais, 6 delas possuem PWM (*Pulse Width Modulation*) necessários para o controle dos motores e as entradas 0 e 1 são reservadas para a comunicação RX/TX que foram utilizadas para comunicação com o back-end. Para o presente trabalho, foram utilizados as portas 11, 10, 9 e 6, que respectivamente controlam a junta superior, junta inferior, base e o movimento de agarrar no braço robótico.

4.1.2. Escolha do manipulador

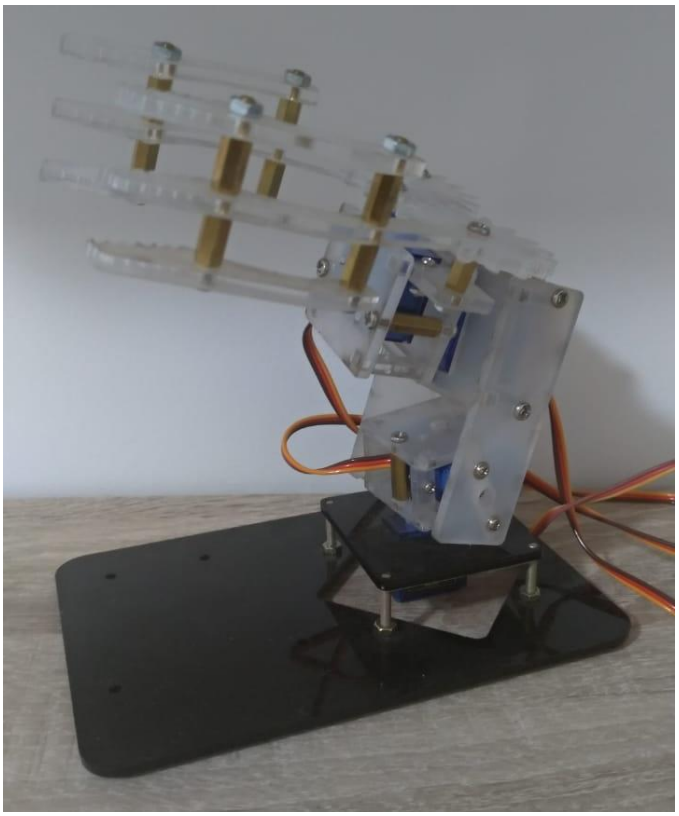
Para a escolha do manipulador a ser utilizado, foi necessário focar no número de graus de liberdade. Para fins didáticos, existem diversas estruturas com diferentes características, podendo variar em tamanho, estrutura da base e tipo de material.

Para fazer a escolha correta foi primeiro necessário decidir quantos graus de liberdade o manipulador teria, pois isso é impactado diretamente pelo número de portas disponíveis no microcontrolador. O motor DC, responsável pelo movimentação das articulações, consome somente uma porta digital com PWM, o que não oferece

[4] Disponível em <https://www.arduino.cc/>

uma limitação para o protótipo, que se optou por 3 DOF, e mais um motor para acionar a garra, apresentado na Figura 15.

Figura 15 – Estrutura do protótipo do manipulador



Fonte: Do autor, 2021.

Um ponto a ser lembrado é que nem todos os modelos de manipulador disponíveis no mercado podem ser utilizados para o protótipo desenvolvido, visto que alguns possuem limitações em sua movimentação que não podem ser tratadas em *software* o que torna o controle de posição impreciso e inconstante.

4.2. Modelagem da cinemática do manipulador

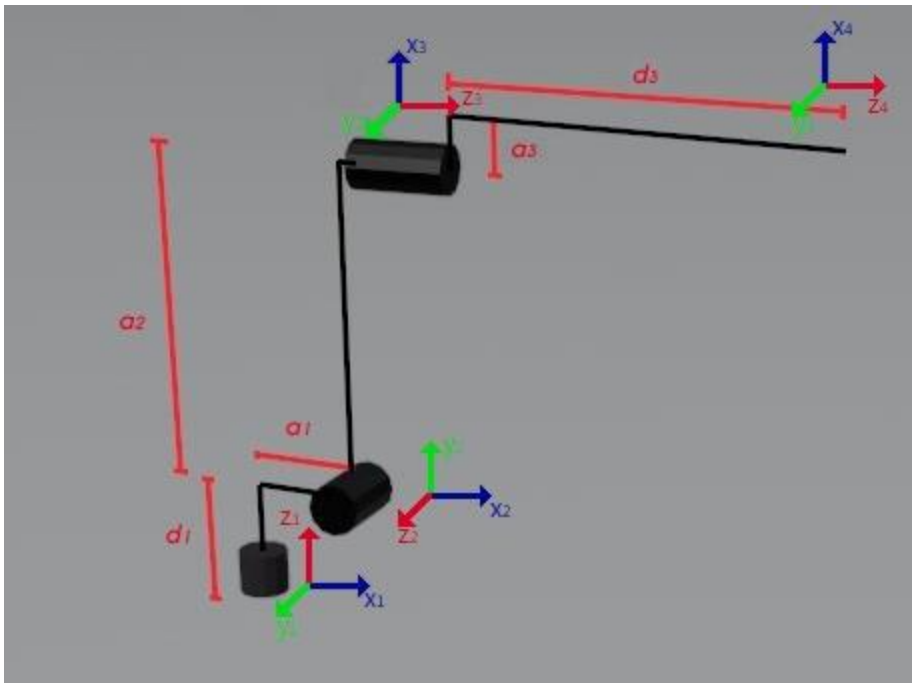
Como visto no capítulo 3, para calcular a cinemática do manipulador, primeiro é necessário definir os eixos de coordenadas para cada junta do manipulador apresentando na Tabela 1. Para isso a base será considerada como elo 0, pois se tratará de um ponto fixo, ilustrado na Figura 16.

Tabela 1 – Parâmetros Denavit-Hartenberg para o manipulador

Elos	a (cm)	α (rad)	d (cm)	θ (rad)
0 – 1	$a_1 = 0{,}7$	$\pi/2$	$d_1 = 3{,}1$	θ_1
1 – 2	$a_2 = 5{,}5$	$-\pi/2$	0	θ_2
2 – 3	$a_3 = 2{,}4$	0	$d_3 = 9{,}7$	θ_3

Fonte: Do autor, 2021.

Figura 16 – Identificação dos elos e juntas do manipulador



Fonte: Do autor, 2021.

Como se trata de um robô que possui todas as juntas rotativas, as variáveis do manipulador serão sempre em função de θ.

Com as variáveis definidas, substitui-se os valores na Equação 3.7 e obtém-se as matrizes de transformação homogênea representadas nas Equações 4.1 a 4.3.

$$A_0^1 = \begin{bmatrix} c\theta_1 & 0 & s\theta_1 & a_1 c\theta_1 \\ s\theta_1 & 0 & -c\theta_1 & a_1 s\theta_1 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$A_1^2 = \begin{bmatrix} \mathrm{c}\theta_2 & 0 & -\mathrm{s}\theta_2 & a_2\mathrm{c}\theta_2 \\ \mathrm{s}\theta_2 & 0 & \mathrm{c}\theta_2 & a_2\mathrm{s}\theta_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

$$A_2^3 = \begin{bmatrix} \mathrm{c}\theta_3 & -\mathrm{s}\theta_3 & 0 & a_3\mathrm{c}\theta_3 \\ \mathrm{s}\theta_3 & \mathrm{c}\theta_3 & 0 & a_3\mathrm{s}\theta_3 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Tendo as matrizes de transformação homogênea para cada elo, é aplicada a Equação 3.8 e se obtém a Equação 4.4, que demonstra o movimento total do manipulador.

$$T_0^3 = \tag{4.4}$$

$$\begin{bmatrix} c\theta_1c\theta_2c\theta_3 - s\theta_1s\theta_3 & -(c\theta_1c\theta_2s\theta_3 + s\theta_1c\theta_3) & -c\theta_1s\theta_2 & a_1c\theta_1 + a_2c\theta_1c\theta_2 + a_3(c\theta_1c\theta_2c\theta_3 - s\theta_1s\theta_3) - d_3c\theta_1s\theta_2 \\ s\theta_1c\theta_2c\theta_3 + c\theta_1s\theta_3 & c\theta_1c\theta_3 - s\theta_1c\theta_2s\theta_3 & -s\theta_1s\theta_2 & a_1s\theta_1 + a_2s\theta_1c\theta_2 + a_3(s\theta_1c\theta_2c\theta_3 + c\theta_1s\theta_3) - d_3s\theta_1s\theta_2 \\ s\theta_2c\theta_3 & -s\theta_2s\theta_3 & c\theta_2 & d_1 + d_3c\theta_2 + a_2s\theta_2 + a_3s\theta_2c\theta_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A partir das Equações 3.4 e 4.4, consegue-se deduzir o vetor que representa a orientação do *end-effector* em relação a base mostrado na Equação 4.5.

$$p_0^3 = \begin{bmatrix} a_1c\theta_1 + a_2c\theta_1c\theta_2 + a_3(c\theta_1c\theta_2c\theta_3 - s\theta_1s\theta_3) - d_3c\theta_1s\theta_2 \\ a_1s\theta_1 + a_2s\theta_1c\theta_2 + a_3(s\theta_1c\theta_2c\theta_3 + c\theta_1s\theta_3) - d_3s\theta_1s\theta_2 \\ d_1 + d_3c\theta_2 + a_2s\theta_2 + a_3s\theta_2c\theta_3 \end{bmatrix} \tag{4.5}$$

4.3. Desenvolvimento de software

A fim de apresentar um diferencial para um manipulador robótico, ele foi integrado a um *software* a ser controlado por um aplicativo Android. Esta seção trata como a integração com o protótipo foi solucionada e quais as tecnologias foram utilizadas para alcançar essa finalidade.

4.3.1. Comunicação entre o microcontrolador e o *back-end*

Para ser possível fazer a integração do microcontrolador ao *back-end*, foi necessário encontrar uma maneira de fazer a comunicação entre as partes. Dito isso, foi verificado a disponibilidade da utilização do protocolo *Firmata* e seu *firmware* para Arduino, o uso do mesmo visa simplificar o processo reduzindo a necessidade de criar um do zero. O protocolo é implementado na biblioteca, disponível de forma gratuita na própria IDE (*Integrated Development Environment*), *StandardFirmata*, o que permite a comunicação direta entre o microcontrolador Arduino e um computador *host*, ou seja, o servidor que processará o *back-end* do projeto (DOS SANTOS, R. M., FLÔR, D. E., 2018). Outro ponto positivo da mesma é a já inclusão da biblioteca Servo, utilizada para a movimentação dos motores do manipulador. Desta forma, a dinâmica de comunicação das partes é exemplificada na Figura 17.

Figura 17 – Comunicação Arduino/Servidor

Fonte: Do autor, 2021.

4.3.2. O *back-end* em *Node.js*

Visando agilizar a criação do servidor, utilizou-se o *Node.js* em conjunto do *johnny-five*, um framework de programação para robótica e *IoT* com código aberto baseado no protocolo *Firmata* que é responsável pela comunicação com o microcontrolador (*Johnny-Five,* 2012*)*.

Outros conjuntos de bibliotecas, apresentadas abaixo, para o gerenciamento das rotas de acesso da servidor, validação dos *inputs* de requisição e comunicação com o banco de dados.

- express: Framework web baseado no módulo núcleo do *Node.js* “http” e componentes Connect. É utilizado para resolver problemas como análise de corpo de requisição http, administração de sessões, organização de rotas e determinação de respostas corretas para cabeçalhos baseados em tipos de dados (MARDANOV, 2013).
- *celebrate*: Função *middleware* utilizada em conjunto do *Express* que engloba a biblioteca de validação “joi”. Isto permite que o *middleware* seja utilizado em uma única rota, ou globalmente, e garanta que todos os seus inputs estejam corretos antes de qualquer função do controlador (*arb/celebrate*).
- knex: Construtor de *query* SQL para bancos de dados relacionais, ele apresenta retornos de chamada, interface para controle de fluxo assíncrono, construtores de *query* e *schema* com recursos completos e respostas padronizadas entre diferentes clientes (Knex.js).

A utilização dessas bibliotecas irá ajudar na proposta de construir o servidor na arquitetura MVC (*Model-View-Controller*), que segundo Krasner e Pope (1998) é a aplicação de uma divisão de três formas, separando (1) as partes que representam o modelo do domínio de aplicação subjacente do (2) o método como o modelo é apresentado ao usuário e do (3) método como o usuário interage com ela. Desta forma, o modelo comportará a implementação central do aplicativo responsável pela divisão (1), a *view* irá lidar com todo conteúdo gráfico requisitando dados do modelo e apresentando os mesmos para o usuário, como explicado na divisão (2) e o controlador fará a interface entre os modelos e *views* associadas fechando a função da divisão (3).

4.3.3. O *front-end* em React Native

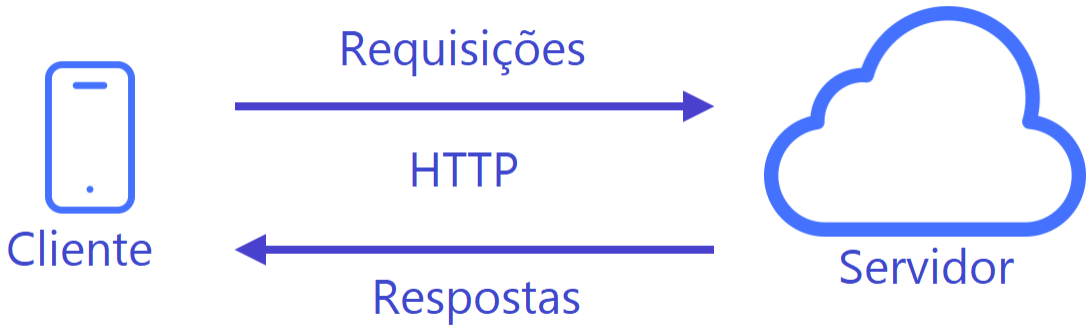
O *Node.js* é o responsável por lidar com os modelos e controladores da aplicação, enquanto o responsável pelas *views* é o *React Native*.

E como no *back-end,* também são utilizadas algumas bibliotecas para auxiliar na implementação, as principais delas são:

- axios: Biblioteca para criação do cliente HTTP baseado em Promises para requisições ao navegador ou ao *Node.js* (axios/axios).
- @react-navigation: Utiliza de utilitários chave para que os navegadores criem a estrutura de navegação no aplicativo (react-navigation)
- expo: *Framework* e plataforma para aplicações *React* universais. Uma gama de ferramentas e serviços construídos em torno de plataformas nativas e do *React Native* que auxiliam no desenvolvimento, construção, implantação e rápida iteração de aplicações iOS, Android e web para a mesma base de código JavaScript/TypeScript (expo).

Um ponto a ser destacado é a comunicação entre o aplicativo e o servidor, na qual foi utilizado o HTTP (Hypertext Transfer Protocol), exemplificado na figura 18.

Figura 18 – Comunicação Aplicativo/Servidor



Fonte: Do autor, 2021.

4.3.4. Processo Unificado (RUP)

Para Guedes (2018), mesmo um simples sistema deve ser modelado antes de sua implementação, pois os *softwares* costumam expandir em tamanho, complexidade e abrangência. E para colocar o desenvolvimento da aplicação em prática foi adotado o Processo Unificado, que consiste em modelar e implementar um sistema utilizando ciclos iterativos e incrementais. Larman (2004), conclui que a implementação do *software* se dá em um tempo justo por não se tratar de um longo e demorado projeto que tenta resolver todos os problemas, pois por se tratar de um processo iterativo, ao final de cada iteração se terá uma versão executável, mas que não está pronta para ser colocada em produção, ou seja, ser liberada para o público.

O processo começa pelo Levantamento e Análise de Requisitos, nessa fase foi preciso criar uma concepção básica das funcionalidades que o aplicativo tem, qual a finalidade e quais as principais dificuldades que os usuários poderiam ter ao utilizar o mesmo. Com uma base de ideias criadas, pode-se descartar os casos de uso que não entrariam no projeto por se tratar de um protótipo inicial, porém algumas das ideias foram separadas para uma possível expansão do projeto e serão apresentadas como possíveis melhorias no futuro.

Tendo uma imagem do aplicativo em mente, segue-se para a fase de Prototipação**,** mesmo que o projeto se trate de um protótipo inicial, essa fase se deu como uma forma de rascunho, na qual está implementado algumas das ideias levantadas na fase 1. E foi durante os testes dessa fase que foram cogitadas a implementação de mais funcionalidades para o aplicativo que o tornaram mais interessantes para o usuário final.

5. RESULTADOS E DISCUSSÕES

Este capítulo apresenta os resultados do desenvolvimento do aplicativo responsável por controlar o manipulador. A ideia é que se possa substituir os cálculos de cinemática inversa pelo aplicativo, pois ele já armazena a posição das juntas e retorna um *feedback* visual ao usuário.

Para tal, foi necessário dividir o desenvolvimento em duas partes, começando pelo *front-end* do aplicativo, no qual foram definidos os elementos de controle de cada junta e os elementos de ação para a movimentação do manipulador. O segundo foi o *back-end* que executa as funções que comunicam com o microcontrolador de controle e movimentação a partir das requisições enviadas pelo *front-end* e comunica ao microcontrolador as ações que devem ser tomadas pelo manipulador.

5.1. Descrição e funcionalidades do sistema

Este aplicativo visa apresentar uma ideia de integração entre os elementos de controle industriais de modo a aumentar a efetividade das ações de seus trabalhadores que poderão executar o serviço estando em casa.

Para a utilização do aplicativo, o microcontrolador deverá estar conectado a um servidor. Para fins de testes foi utilizado um Notebook Dell *Inspiron* 14, pois dessa forma ele se conecta via USB (*Universal Serial Bus*), uma vez que manipuladores são elementos que têm a sua base fixada no local que será utilizado, fazer essa conexão em ambiente industrial também não seria um empecilho para o protótipo.

Entre as funcionalidades da aplicação, está a capacidade de controlar individualmente cada junta do motor e seu *end-effector* e de salvar duas posições para que o robô possa trabalhar em *loop* entre elas sobre o comando de iniciar e pausar o *loop* sobre requisição do usuário.

5.2. Apresentação do design do sistema

O *layout* do aplicativo, chamado de *manipulator-control*, é composto por uma tela *splash* (i.e uma tela de apresentação) de fundo branco e com a logo do aplicativo ao centro, mostrada na Figura 19.

Figura 19 – Tela *splash* do aplicativo

Fonte: Do autor, 2021.

Na Figura 20 é apresentada a tela utilizada para controle do manipulador que possui a logo do aplicativo ao topo e em seguida os inputs do sistema que serão responsáveis por fazer as requisições ao servidor. São eles os três *sliders* ao centro, responsáveis pela movimentação individual dos motores e o botão para acionar e liberar o *end-effector*, logo em seguida estão os botões para salvar as posições que o usuário configurou utilizando os *sliders,* e por fim os botões para que iniciam o movimento de loop entre as duas posições salvas pelo usuário.

Figura 20 – Tela para controle do manipulador

Fonte: Do autor, 2021.

5.3. Implementação do sistema

Para implementar a tela de controle foram utilizados as classes *View, Image, Text, Slider* e *TouchableOpacity* importadas diretamente da biblioteca *react-native*, como mostrado na Figura 21. Elas serão utilizadas para criar a estrutura inicial do aplicativo e seu uso é extremamente similar ao HTML (*HyperText Markup Language*) utilizado em desenvolvimento de sites, um exemplo de seu uso é apresentado na Figura 22.

Figura 21 – Adição das classes para criação da estrutura

```
import {View, Image, Text, Slider, TouchableOpacity} from 'react-native';
```

Figura 22 – Exemplos de uso das classes estruturais

```
<View style={styles.sliderView}>
  <Text style={styles.description}>Pulso</Text>

  <View style={{borderRadius: 50, overflow: 'hidden', borderWidth: 1, 'borderColor': '#737380'}}>
    <View style={{flexDirection: 'row', position: 'absolute'}}>
      <View style={{ ... styles.sliderDummy, width: (pulse/180) * 300}}/>
      <View style={styles.sliderReal}/>
    </View>

    <Slider
      style={{width: 300, height: 30, borderRadius: 50}}
      minimumValue={1}
      maximumValue={180}
      value={pulse}
      onValueChange={val ⇒ setPulse(val) }
      onSlidingComplete={moveArm}
      minimumTrackTintColor='transparent'
      maximumTrackTintColor='transparent'
      thumbTintColor='transparent'
    />
  </View>
</View>

<View style={{ ... styles.actions, marginTop: 30, justifyContent: 'center'}}>
  <TouchableOpacity style={{ ... styles.action,  backgroundColor: '#119EC2'}} onPress={grab}>
    <Text style={styles.actionText}>Agarrar</Text>
  </TouchableOpacity>
</View>
```

Fonte: Do autor, 2021.

Para a estilização do sistema foi utilizado outra classe da biblioteca *react-native* chamada *StyleSheet*, ela permite criar uma folha de estilo similar ao CSS (*Cascading Style Sheets*) também utilizado em desenvolvimento de sites, na Figura 23 é apresentado um exemplo de sua utilização.

Figura 23 – Exemplo do uso da Classe *StyleSheet*

```
export default StyleSheet.create({
  container: {
    flex: 1,
    paddingHorizontal: 24,
    paddingTop: Constants.statusBarHeight + 20,
  },

  header: {
    flexDirection: 'column',
    justifyContent: 'center',
    alignItems: 'center',
  },

  headerText: {
    fontSize: 20,
    lineHeight: 24,
    color: '#737380',
    marginTop: 15,
  },

  description: {
    fontSize: 16,
    lineHeight: 24,
    color: '#737380',
  },

  sliderView: {
    marginTop: 30,
  },
```

Fonte: Do autor, 2021.

Seguindo para as funcionalidades do aplicativo foi utilizada a já apresentada biblioteca *axios* e *@react-navigation*, com implementação mostrada nas Figuras 24 a 26, a biblioteca *axios* facilita as chamadas ao servidor, pois ela automatiza as requisições http feitas pelo aplicativo, enquanto a *@react-navigation* cuidará da navegação entre as telas do aplicativo.

Figura 24 – Implementação do *axios*

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://192.168.1.215:3333'
});

export default api;
```

Figura 25 – Implementação do *@react-navigation*

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const AppStack = createStackNavigator();

import Main from "./pages/main";

export default function Routes() {
  return (
    <NavigationContainer>
      <AppStack.Navigator screenOptions={{ headerShown: false }}>
        <AppStack.Screen name="Main" component={Main}/>
      </AppStack.Navigator>
    </NavigationContainer>
  );
}
```

Figura 26 – Requisições ao servidor utilizando o axios

```
import api from "../../services/api";

export default function Main() {
  const [shoulder, setShoulder] = useState( initialState: 90);
  const [elbow, setElbow] = useState( initialState: 75);
  const [pulse, setPulse] = useState( initialState: 90);

  function moveArm() {
    api.post( url: '/mover-braco',  data: {shoulder, elbow, pulse});
  }

  function grab() {
    api.get( url: '/agarrar');
  }
```

Fonte: Do autor, 2021.

Para receber as requisições vindas do *front-end*, foram utilizadas duas bibliotecas principais o *express* e o *celebrate*, demonstradas na Figura 27 e 28, o *express* com a classe *Router* facilita a criação de rotas, pois, como o *axios* no *front-end*, ela automatiza as requisições recebidas pelo servidor, enquanto o *celebrate*, utilizando as classes *celebrate*, *Segments* e *Joi* faz o tratamento dos *inputs* enviados para a rota impedindo algum mal funcionamento das funções que movimentam o manipulador.

Figura 27 – Implementação do gerenciamento de rotas e tratamento de *inputs*

```
const { Router } = require('express');
const { celebrate, Segments, Joi } = require('celebrate');

const ArmController = require('./controllers/ArmController');

const routes = Router();

routes.post('/mover-braco', celebrate( requestRules: {
  [Segments.BODY]: Joi.object().keys({
    shoulder: Joi.number().required(),
    elbow: Joi.number().required(),
    pulse: Joi.number().required(),
  }),
}), ArmController.move);

routes.get( path: '/agarrar', ArmController.grab);

routes.post('/salvar-posicao-1', celebrate( requestRules: {
  [Segments.BODY]: Joi.object().keys({
    shoulder: Joi.number().required(),
    elbow: Joi.number().required(),
    pulse: Joi.number().required(),
  }),
}), ArmController.savePosition1);
```

Figura 28 – Inicialização dos módulos *express* e *celebrate*

```
const app = require('./app');

app.listen( port: 3333);
```

```
const express = require('express');
const { errors } = require('celebrate');
const routes = require('./routes');

const app = express();

app.use(express.json());
app.use(routes);
app.use(errors());

module.exports = app;
```

Fonte: Do autor, 2021.

Para executar as ações requisitadas, utiliza-se duas bibliotecas o *knex* para conexão com o banco de dados que salvará as posições da funcionalidade de *loop* e a *johnny-five* para comunicação com o microcontrolador, ambas implementações e usos apresentados nas Figuras 29 a 32.

Figura 29 – Importação e inicialização do módulo *knex*

```
const  knex = require('knex');
const configuration = require('../../knexfile');

const config = configuration.development;

const connection = knex(config);

module.exports = connection;
```

Figura 30 – Implementação das funções relacionadas ao banco de dados

```
async savePosition2(request, response) {
    if (!isReady || isRunning) return response.status(404).send();

    const {shoulder, elbow, pulse} = request.body;

    await connection('positions')
      .where( columnName: 'id', operator: '=', value: 2)
      .update( data: {shoulder, elbow, pulse});

    response.status(200).send();
},
async play(request, response) {
    if (!isReady || isRunning) return response.status(404).send();

    const point1 = await connection('positions').where( columnName: 'id', value: 1).first();
    if (!point1) return response.status(204).send();

    const point2 = await connection('positions').where( columnName: 'id', value: 2).first();
    if (!point2) return response.status(204).send();
```

Figura 31 – Importação do módulo *johnny-five* e inicialização da placa e motores

```
const connection = require('../database/connection');
const arduino = require('johnny-five');

let board = new arduino.Board({ port: "COM3" });
let isReady = false;
let isRunning = false;
let isOn = false;
let shoulderServo;
let elbowServo;
let pulseServo;
let grabServo;
let runFunction;
const timeout = 1000;

board.on( event: 'ready', listener: function() {
    shoulderServo = new arduino.Servo({
        pin: 9,
        type: "standard",
        range: [0, 180],
        fps: 100,
        center: true,
    });
```

Figura 32 – Implementação das funções de movimentação do manipulador

```
move(request, response) {
    if (!isReady || isRunning) return response.status(404).send();

    const {shoulder, elbow, pulse} = request.body;

    pulseServo.to(pulse, Math.abs( x: pulseServo.last.degrees - pulse) > 75 ? 500 : 250);
    elbowServo.to(elbow, Math.abs( x: elbowServo.last.degrees - elbow) > 75 ? 500 : 250);
    shoulderServo.to(shoulder, Math.abs( x: shoulderServo.last.degrees - shoulder) > 75 ? 500 : 250);

    response.status(200).send();
},
grab(request, response) {
    if (!isReady || isRunning) return response.status(404).send();

    toggleGrab();

    response.status(200).send();
},
```

Figura 33 – Implementação das funções *setInterval* e *clearInterval*

```
    runFunction = setInterval( callback: () => {
        shoulderServo.to(point1.shoulder, timeout);
        setTimeout( callback: function() { ... }, ms: 1000);
        setTimeout( callback: function() { ... }, ms: 2000);
        setTimeout( callback: function() { ... }, ms: 3500);
        setTimeout( callback: function() { ... }, ms: 4000);
        setTimeout( callback: function() { ... }, ms: 5000);
        setTimeout( callback: function() { ... }, ms: 6000);
        setTimeout( callback: function() { ... }, ms: 7500);
    }, ms: 9000);

    response.status(200).send();
},
stop(request, response) {
    if (!isReady) return response.status(404).send();

    isRunning = false;
    clearInterval(runFunction);

    response.status(200).send();
},
```

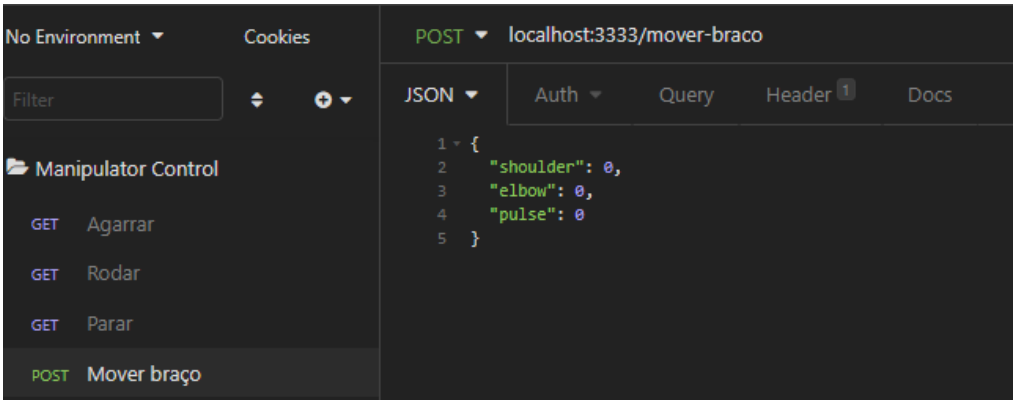
Fonte: Do autor, 2021.

Para a implementação da movimentação em *loop*, vale salientar a necessidade das funções *setInverval* e *clearInterval*, mostradas na Figura 33. Segundo o *MDN Web Docs,* o método *setInterval* repete chamadas de funções com um tempo de espera fixo entre cada chamada, isso permite que a função se mantenha executando após a finalização da execução da requisição feita pelo *front-end* e o *loop* só será finalizado quando a função *clearinterval* for chamada por outra requisição feita pelo *back-end*.

5.4. Análise de resultados

Para os testes de performance do aplicativo, foi utilizada a equação 4.5 em conjunto com a configuração das posições pelo aplicativo. Para garantir a precisão da posição do software durante os testes foi utilizado o Insomnia, exemplo na Figura 34, que é uma plataforma *desktop* gratuita para interagir e projetar APIs baseadas em HTTP, ele combina uma interface fácil de usar com funcionalidades avançadas como geração de código, autenticação e variáveis de ambiente (Insomnia).

Figura 34 – Interface Insomnia



Fonte: Do autor, 2021.

Os testes ocorreram da seguinte maneira, foram escolhidos três ângulos de forma aleatória para posicionamento das juntas, utilizando a equação citada acima obtêm-se um vetor de três posições que serão utilizados para comparar com as posições medidas. Para as medições serão feitas três simulações utilizando os ângulos escolhidos, os resultados apresentam a média dos valores encontrados com

o auxílio de uma régua, que possui um erro de $\pm$ 0,5cm, e foram calculados, por meio das Equações 5.1 e 5.2, seu desvio absoluto e seu desvio relativo percentual, respectivamente.

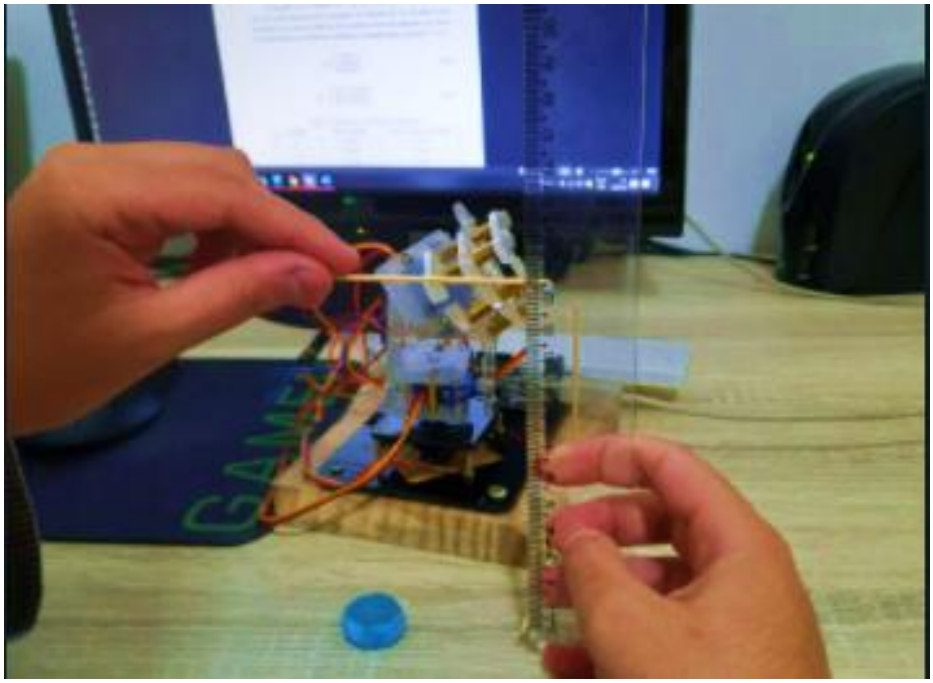
$$\bar{\delta}_{abs} = \frac{1}{N}\sum_{i=1}^{n}|x_i - \bar{x}| \tag{5.1}$$

$$\bar{\delta}_{\%} = 100\frac{\bar{\delta}_{abs}}{\bar{x}} \tag{5.2}$$

n é o número de medições realizadas.

Começando com os ângulos, $\theta_1 = -30°$, $\theta_2 = 30°$ e $\theta_3 = -60°$, se obtém o vetor de posição calculado visto na Equação 5.3a. Enquanto na 5.3b, se obtém o vetor de posição para a média medida de três simulações feitas pelo aplicativo, cujo método de medição é mostrado na Figura 35. Na Tabela 2 são apresentados os parâmetros estatísticos calculados pelas Equações 5.1 e 5.2.

Figura 35 – Medição da posição do eixo z para simulação 1



Fonte: Do autor, 2021.

$$p_0^3 = \begin{bmatrix} 2{,}47cm \\ -3{,}83cm \\ 12{,}45cm \end{bmatrix} \tag{5.3a}$$

$$\bar{p_0^3} = \begin{bmatrix} 2{,}55 \ \pm 0{,}17cm \\ -3{,}65 \ \pm 0{,}15cm \\ 12{,}45 \ \pm 0{,}20cm \end{bmatrix} \tag{5.3b}$$

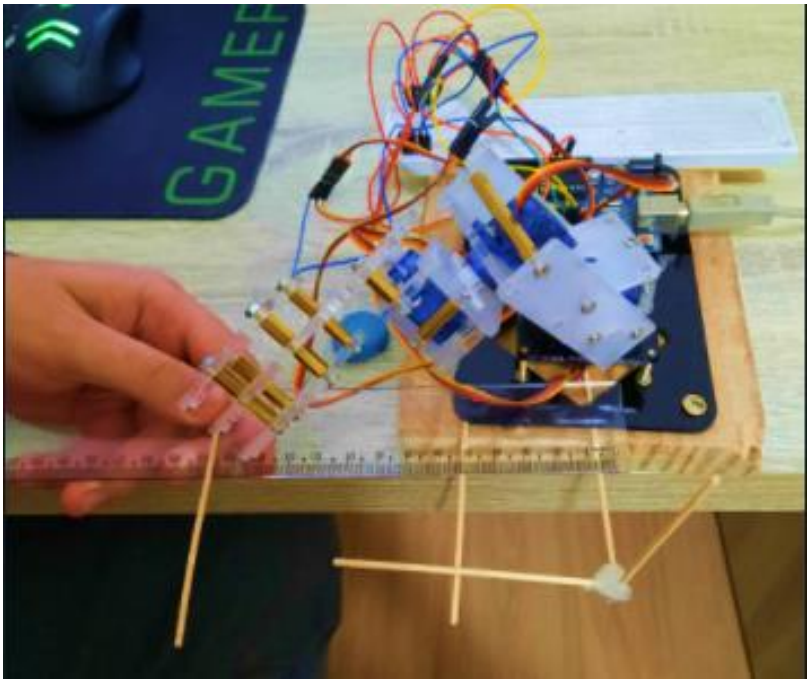
Tabela 2 – Parâmetros estatísticos da simulação 1

Posição	Desvio absoluto	Desvio relativo percentual
p_x	0,17cm	6,67%
p_y	0,15cm	3,91%
p_z	0,20cm	1,61%

Fonte: Do autor, 2021.

O segundo teste será feito para os ângulos $\theta_1 = 60°$, $\theta_2 = -45°$ e $\theta_3 = 45°$, desta forma se obtém os vetores nas Equações 5.4a e 5.4b, a Tabela 3 apresenta seus parâmetros estatísticos e na Figura 36 é mostrado sua medição.

Figura 36 – Medição da posição do eixo y para simulação 2



Fonte: Do autor, 2021.

$$p_0^3 = \begin{bmatrix} 6{,}05cm \\ 13{,}88cm \\ 2{,}47cm \end{bmatrix} \tag{5.4a}$$

$$\bar{p_0^3} = \begin{bmatrix} 6{,}25 \pm 0{,}16cm \\ 13{,}85 \pm 0{,}12cm \\ 2{,}35 \pm 0{,}16cm \end{bmatrix} \tag{5.4b}$$

Tabela 3 – Parâmetros estatísticos da simulação 2

Posição	Desvio absoluto	Desvio relativo percentual
p_x	0,21cm	3,36%
p_y	0,12cm	0,86%
p_z	0,16cm	6,81%

Fonte: Do autor, 2021

O último teste utilizará os ângulos $\theta_1 = 45°$, $\theta_2 = 60°$ e $\theta_3 = -30°$ e seus resultados são apresentados nas equações 5.5a, 5.5b, na Tabela 4 e na Figura 37.

Figura 37 – Medição da posição do eixo x para simulação 3

Fonte: Do autor, 2021.

$$p_0^3 = \begin{bmatrix} -0{,}22cm \\ -1{,}9cm \\ 12{,}11cm \end{bmatrix} \tag{5.5a}$$

$$\bar{p_0^3} = \begin{bmatrix} -0{,}45 \ \pm 0{,}21cm \\ -1{,}85 \ \pm 0{,}08cm \\ 12{,}35 \ \pm 0{,}23cm \end{bmatrix} \tag{5.5b}$$

Tabela 4 – Parâmetros estatísticos da simulação 3

Posição	Desvio absoluto	Desvio relativo percentual
p_x	0,21cm	46,67%
p_y	0,08cm	4,32%
p_z	0,23cm	1,28%

Fonte: Do autor, 2021.

6. CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi implementar um aplicativo para controle visual de um manipulador robótico 3DOF a fim de demonstrar a integração de um dispositivo de uso popular que é o telefone celular com um equipamento de uso industrial, sendo este o manipulador. Para tal, foram utilizados principalmente o microcontrolador Arduino com a comunicação feita por meio do protocolo *Firmata* e as linguagens de programação *React Native* para o aplicativo e *Node.js* para o *back-end*. O aplicativo tem como principal função controlar individualmente cada junta do manipulador. Como forma de simular o movimento entre posições no espaço de trabalho do manipulador, ele também possui a capacidade de salvar duas posições e criar um loop de movimentação entre os dois pontos. Diversas bibliotecas foram necessárias para tal, com destaque para a *johnny-five* que possibilitou a integração do microcontrolador com o aplicativo.

Devido à indisponibilidade de um manipulador industrial e um servidor de grande escala, foi necessário adaptar as soluções para menores dimensões. Apesar da redução escalar terem facilitado o processo de desenvolvimento, isso

impossibilitou um processo de integração completo do aplicativo, porém deixou um espaço de crescimento enorme para melhorias e novas aplicações.

Além das melhorias ligadas as limitações de *hardware*, como a troca do microcontrolador de um Arduino para um PLC (*Programmable Logic Controller*) e do manipulador protótipo para um industrial, o *software* também apresenta alguns pontos para serem desenvolvidos, entre eles estão:

- Incorporação de uma plataforma *web* para monitoramento das ações do manipulador;

- Um cadastro de usuários para que possa ser implementado uma ferramenta de login para aumentar a segurança de utilização do aplicativo e o acesso as funções do manipulador.

Um último ponto a ser considerado se trata da pandemia da Covid-19, segundo a OMS, mesmo com o uso de máscaras e a frequente higienização das mãos é de extrema importância o distanciamento físico (OMS, 2020). Levando isso em consideração, o presente trabalho também é apresentado como uma alternativa viável para o *home-office,* com a possibilidade de controlar equipamentos e monitorar os dados da empresa, um trabalhador poderia ser mantido em casa. Nesse caso, sua presença na empresa só seria necessária em casos de urgência, reduzindo, para ele e demais funcionários, o risco de contaminação.

O código utilizado para o *back-end* e *front-end* estão disponíveis no repositório *Github* no *link:* https://github.com/gcgomes/manipulator-control.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, L. **Node.js**: Construções de aplicações Web. 2016. p. 19

arb/celebrate. **A joi validation middleware for Express**. Disponível em <https://github.com/arb/celebrate>. Acesso 10 mar. 2021.

axios/axios. **Promise based HTTP client for the browser and node.js**. Disponível em <https://github.com/axios/axios>. Acesso em 14 mar. 2021.

CARRARA V. et al. Estudo da cinemática inversa aplicada num braço robótico. **Anais do 12º Encontro de Iniciação Científica e Pós-Graduação do ITA – XII ENCITA**. São José dos Campos, 2006.

CARRARA, V. Apostila de robótica. **Universidade Braz Cubas, Área de Ciências Exatas Engenharia Mecânica, Engenharia de Controle e Automação**. Brasília, 2009. 10 – 26 p.

CASTELLS, M. **A Sociedade em Rede**. 8. ed. Paz e Terra. São Paulo, 2013.

CASTELLS, M. **Communication Power**. 2nd ed. OUP Oxford. Oxford, 2011.

CRAIG, J. J.; **Introduction to robotics**: mechanics and control. 3rd ed. Pearson Education India. New Jersey, 2009.

DAPPER, A. R. **Aplicativo mobile para localização de farmácias**. Florianópolis, 2017. 20 – 21 p. Monografia (Bacharel em Sistemas de informação) – Universidade do Sul de Santa Catarina, SC, 2017.

DOMBRE, E.; KHALIL, W. **Robot Manipulators**: Modeling, Performance, Analysis and Control. 2nd ed. Wiley-ISTE. Wiltshire, 2007.

DOS SANTOS, R. M., FLÔR, D. E. **RASPCAR**: Carro guiado remotamente utilizando o Raspberry Pi e Arduino. Paranavaí, 2018. 4 p. Trabalho de Iniciação Científica – Instituto Federal do Paraná, PR, 2018.

DRATH, R.; HORCH, A. Industrie 4.0: Hit or hype? **IEEE industrial electronics magazine**. New Jersey, 2014.

EISENMAN, B. **Learning React Native**. 1st ed. O'Reilly Media, Inc. Massachussets, 2016. 17 p.

Expo. **Introduction to expo**. Disponível em <https://docs.expo.io>. Acesso em 14 mar. 2021.

FAVERI, G. de. **Cinemática inversa de robôs paralelos: aplicações e teleoperação de manipuladores planares e espaciais**. Florianópolis, 1998. 19 – 20 p. Monografia (Bacharel em Engenharia Mecânica) – Universidade do Estado de Santa Catarina, SC, 1998.

FERDOUSH S.; XINRONG L. **Wireless Sensor Network System Design using Raspberry Pi and Arduino for Environmental Monitoring Applications**. Department of Electrical Engineering, University of North Texas, Denton, Texas, USA 2014. 3 p.

GAMERO, I. **Robôs Industriais**: tudo o que você precisa saber! Pollux, mar. 2018. Disponível em: <https://www.pollux.com.br/blog/robos-industriais-tudo-o-que-voce-precisa-saber/>. Acesso em abr. 2019.

GUEDES, G. T. A. **UML 2**: Uma Abordagem Prática, 3 ed. Novatec. São Paulo, 2018.

Insomnia. **Getting Started with Insomnia**. Disponível em: <https://support.insomnia.rest/article/157-getting-started-with-insomnia>. Acesso em 22 mai. 2021.

International Federation of Robotics. **IFR World Robotics Presentation**. IFR Press Conference. Shanghai, 2019. Disponível em: <https://ifr.org/downloads/press2018/IFR%20World %20Robotics%20Presentation%20-%2018%20Sept%202019.pdf>. Acesso em abr. 2021.

ISMAIL, I.; BOKHARE, S. F.; AZIZAN; S. N.; AZMAN, N. Teaching via mobile phone: A case study on Malaysian teachers' technology acceptance and readiness. **Journal of Educators Online**, Dothan, Jan. 2013. 1-38 p.

Johnny-Five. **The JavaScript Robotics Programming Framework**, 2012. Disponível em <https://www.npmjs.com/package/johnny-five>. Acesso em 10 mar. 2021.

KESKIN, N. O.; METCALF, D. The current perspectives, theories and practices of mobile learning. The Turkish Online **Journal of Educational Technology**, Sakarya, 2011. 202-208 p.

Knex.js. **Um construtor de consultas SQL para Javascript**. Disponível em: <http://knexjs.org/>. Acesso em: 10 mar. 2021.

KRASNER, G.; POPE, S. A cookbook for using the model view controller user interface paradigm in smalltalk-80. **Journal of Object-Oriented Programming**, 1998. 2-3 p.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado. 2 ed. Bookman. Porto Alegre, 2004.

LASI, H.; FETTKE, P.; KEMPER, H.-G.; FELD, T.; HOFFMANN, M. Industry 4.0. **Business & Information Systems Engineering**. 2014. 239–242 p.

LOPES, A. M. **Modelação cinemática e dinâmica de manipuladores de estrutura em série**. Porto, 2002. 4 – 17 p. Tese (Mestrado em Automação, Instrumentação e Controle) – Faculdade de Engenharia. Universidade do Porto, Portugal, 2002.

MANTOVANI, C. M. Telefonia Celular: Informação e Comunicação em Novo Espaço de Fluxo. Intercom. **Sociedade Brasileira de Estudos Interdisciplinares da Comunicação XXVIII Congresso Brasileiro de Ciências da Comunicação**. Rio de Janeiro, 2005. Disponível em: <http://www.portcom.intercom.org.br/pdfs/79903392067139223359944593220619405378.pdf>. Acesso em 20 mai. 2021.

MARDANOV A. **Express.js Guide**: The Comprehensive Book on Express.js. 1st ed. Createspace Independente Publishing Platform. North Charleston, 2013.

MCCLOY, D.; **Robotics**: an introduction. Springer Science & Business Media, 2013.

MDN Web Docs. **WindowOrWorkerGlobalScope.setInterval()**. Disponível em <https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval>. Acesso em 05 mai. 2021.

MOHAMMED, A. A.; SUNAR, M. Kinematics modeling of a 4-DOF robotic arm. **International Conference on Control, Automation and Robotics**. Singapore, 2015.

NIKU, S. B. **Introdução à robótica**: análise, controle, aplicações. 2. ed. Tradução e revisão técnica: Sérgio Gilberto Taboada. Rio de Janeiro, 2015

NOF, S. Y. **Handbook of industrial robotics**. John Wiley & Sons, 1999. Disponível em: <https://onlinelibrary.wiley.com/doi/book/10.1002/9780470172506>. Acesso em 26 de abr. 2021

O Globo. Até 2021, robôs farão o trabalho de 4,3 milhões de humanos, mostra pesquisa. **Globo Economia**. 2018. Disponível em: <https://oglobo.globo.com/economia/ate-2021-robos-farao-trabalho-de-43-milhoes-de-humanos-mostra-pesquisa-22953268>. Acesso em mar. 2019

OMS (Organização Mundial da Saúde); **Recomendações sobre o uso de máscaras no contexto da COVID-19**. 2020. Disponível em: <https://apps.who.int/iris/bitstream/handle/10665/332293/WHO-2019-nCov-IPC_Masks-2020.4-por.pdf>. Acesso em 23 mai. 2021.

PENHA, B. S.; QUEIROZ, M. E.; ROSSINI, F. L. Modelagem e Análise da Cinemática Direta e Inversa de Manipulador Robótico com Cinco Juntas Rotativas. **III Simpósio de Tecnologia e Engenharia Eletrônica - III SIMTEEL**. Campo Mourão, 2018.

PIMENTA, T. T. **Controle de Manipuladores Robóticos**. Rio de janeiro, 2009. 10 – 18 p. Monografia (Bacharel em Engenharia de Controle e Automação) – PUC, RJ, 2009.

RANKY, P. G.; HO, C. Y. **Robot modelling**: control and applications with software. United Kingdom, 1985.

React-navigation. **Documentação do @react-navigation**. Disponível em <https://reactnavigation.org/docs/getting-started>. Acesso em 18 mar. 2021

SANTOS, V. F. S. **Robótica Industrial**. Departamento de Engenharia Mecânica. Universidade de Aveiro. Aveiro, 2003. 20 – 24 p.

SICILIANO, B. et al. **Robotics: modelling, planning and control**. 1st ed. Springer Science & Business Media, 2010.

SILVA, M. M. da. SANTOS, M. T. P.; Os Paradigmas de Desenvolvimento de Aplicativos para Aparelhos Celulares. **T.I.S - Tecnologias, Infraestrutura e Software – UFSCar**. São Carlos, 2014. 162– 170 p.

SILVA, R. M. da; SANTOS FILHO, D. J.; MIYAGI, P. E. Modelagem de Sistema de Controle da Indústria 4.0 Baseada em Holon, Agente, Rede de Petri e Arquitetura Orientada a Serviços. **XII Simpósio Brasileiro de Automação Inteligente**. Natal, 2015.

SLABAUGH G.G. **Computing euler angles from a rotation matrix**. Technical report. University of London, London, 1999.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. **Robot Modeling and Control**. 1ª ed. John Wiley & Sons, INC. New Jersey, 2005.

TECMUNDO. **Os 5 sistemas operacionais mobile mais vendidos de 2016**. Disponível em: <https://www.tecmundo.com.br/mercado/108748-5-sistemas-operacionais-mobile-vendidos-2016.htm>. Acesso em: 08 abr. 2021.

The Stanford Infolab. **Robots and Theirs Arms. Stanford Arm – 1969**. Disponível em <http://infolab.stanford.edu/pub/voy/museum/pictures/display/1-Robot.htm>. Acesso em 20 abr. 2021.

TSAI, L.W. **Robot analysis: the mechanics of serial and parallel manipulators**. John Wiley & Sons. New Jersey, 1999.

YAVUZ, S. Ç. **Kinematic Analysis For Robot Arm**. Electrical and electronics faculty department of computer engineering Yildiz Technical University. Istanbul, 2009.

YOSHIKAWA, T. **Foundations of Robotics:** Analysis and Control. MIT Press. Massachussets, 2003.