UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Definição de extremos do valor limite utilizando robô triângulo delta

Paulo Henrique Araújo Munhoz

Manaus – AM
Novembro/2021

Paulo Henrique Araújo Munhoz

Definição de extremos do valor limite utilizando robô triângulo delta

Monografia de Graduação apresentada à Faculdade de Tecnologia da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de bacharel em Engenharia da Computação.

Orientador: Prof. Francisco de Assis Pereira Januário, M.Sc.

Universidade Federal do Amazonas – UFAM
Faculdade de Tecnologia – FT

Manaus-AM
Novembro/2021

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

M966d

Munhoz, Paulo Henrique Araújo
Definição de Extremos do Valor Limite utilizando Robô Triângulo Delta / Paulo Henrique Araújo Munhoz . 2021
74 f.: il. color; 31 cm.

Orientador: Francisco de Assis Pereira Januário
TCC de Graduação (Engenharia da Computação) - Universidade Federal do Amazonas.

1. Robô triângulo delta. 2. Teste de Software. 3. Testes Automatizados. 4. Teste de aplicativos móveis. 5. Análise do Valor Limite. I. Januário, Francisco de Assis Pereira. II. Universidade Federal do Amazonas III. Título

Monografia de Graduação sob o título Definição de extremos do valor limite utilizando robô triângulo delta apresentada por Paulo Henrique Araújo Munhoz e aceita pela Faculdade de Tecnologia da Universidade Federal do Amazonas, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Francisco de Assis Pereira Januário

Orientador: Prof. Francisco de Assis Pereira Januário, M.Sc.
Departamento de Eletrônica e Computação
Universidade Federal do Amazonas

Frederico da Silva Pinagé

Prof. Frederico da Silva Pinagé, D. Sc.
Departamento de Eletrônica e Computação
Universidade Federal do Amazonas

José Reginaldo Hughes Carvalho

Prof. José Reginaldo Hughes Carvalho, D. Sc.
Instituto de Computação
Universidade Federal do Amazonas

Manaus-AM, 22 de novembro de 2021

À minha família, à minha esposa e ao meu irmão, os meus heróis.

Agradecimentos

Agradeço primeiramente à Deus todo poderoso e as entidades abaixo Dele aos quais sempre me ajudaram espiritualmente. À minha mãe Maria do Socorro que me gerou e criou com todo amor e carinho, ao meu pai Josiel Pereira que sempre garantiu o conforto e conselhos aos quais sempre levarei em meu coração e ao meu irmão João Paulo, meu herói que sempre se colocou nos momentos árduos e corriqueiros, meu muito obrigado.

À minha esposa Kariny Oliveira, graças a seus conselhos e perseverança sempre me deu forças a nunca desistir de nada. À toda família Araújo e Munhoz, cada parentesco contribuiu com ensinamentos dos quais foram a soma da minha formação e em especial minhas tias Fátima e Nazaré que tenho como minha segunda mãe, vocês sempre foram a minha fonte de inspiração.

A Universidade Federal do Amazonas, a Faculdade de Tecnologia e aos professores: Celso Carvalho, Waldir Sampaio, Carlos Cruz, Florindo Ayres, Helder Cruz e ao IComp carinhosamente os professores: Arilo Claúdio, Eduardo Nakamura, Eulanda Miranda, Rosiane Rodrigues, Leandro Carvalho e José Pio. Especialmente ao professor Francisco Januário, responsável pela orientação do meu trabalho. Obrigado por esclarecer tantas dúvidas e ser tão paciente.

A todos do grupo Experts: Awdren Fontão, Karina Villanes, Josias Gomes e Oswald. Onde, por meio das reuniões realizadas semanalmente pude aprender a ter gosto pela área de engenharia de software.

A todos do grupo Bughackers: Jonathas Kerber, Sarah Silva, João Danilo, Amanda de Paula e Rebeca Rodrigues, que compartilharam de seu tempo e conhecimento, criando um espaço onde podíamos trocar ideias que pudessem ser alavancadas para entusiasmar novos alunos a compreender e ter gosto por sistemas embarcados.

Aos meus amigos da graduação: Suzana Rita, Aurélio Freire, Felipe André, William Barreiros, Geraldo Rabelo, João Mota, Lucas Lauria, Valdenise

Costa, Emanuela Magalhães (Manu), Marcel Loebens, Ailton da Silva, Tainá Gonçalves, Michael Marlon, Juliana Assunção e a todos os amigos que adquiri nessa trajetória.

Aos meus amigos Felipe Lourenço (notridame), Carol (Yuna), Kayque Damasceno (Kaaayy), Akio Shishido (Shishido), Gabriel (Dzerol), Bruno Matthaus (Gato Marshmallow), Luis Gustavo (LGugs), Ana Paula (Kitana) que proporcionaram um tempo de lazer por meio dos jogos eletrônicos, sem vocês não seria tão divertido os finais de semana.

Sou grato a empresa INDT, que me concedeu a chance de conhecer um pouco mais da minha área de formação. Em especial ao meu gestor Edissandro Bessa e meus colegas William Freitas, Victor Lopes e Galilleu Souza, que me auxiliaram em todas as etapas do meu crescimento profissional. Obrigado por confiarem nos conhecimentos que adquiri durante a faculdade.

Aos meus avós Orzila Reis, Manoel Gonçalves, Nicolau Munhoz e Hetelvina Pereira (in memoriam), por ter me ensinado valores que carrego comigo em todos os momentos. Obrigado por me olhar de algum lugar.

Ao professor Dr. Ayres Mardem Almeida do Nascimento (in memoriam), que sempre disponibilizou seu tempo ao ensino e sempre auxiliou os alunos em matérias que eram consideradas extremamente difíceis, mestre onde estiver serei eternamente grato pelos seus conselhos. Ao professor Heitor José Ferreira de Carvalho (in memoriam), suas aulas puderem abrir possibilidades para compreender a importância de outras áreas, carinhosamente guardarei o livro que o senhor me presenteou. E por fim ao meu ilustrador mangaká favorito Kentaro Miura (in memoriam), seu legado nunca será esquecido, todas as páginas de Berserk são lições que podemos refletir no nosso cotidiano.

Por fim, agradeço todas as pessoas que de alguma forma estiveram envolvidas na realização deste trabalho.

Somos todos estrelas de uma história que está longe de acabar.

Griffith

Definição de extremos do valor limite utilizando robô triângulo delta

Autor: Paulo Henrique Araújo Munhoz
Orientador: Prof. Francisco de Assis Pereira Januário, M.Sc.

RESUMO

O objetivo geral deste trabalho é apresentar uma base de dados extraída contendo os valores limites de uma aplicação simples de dispositivo móvel. Inicialmente foi definida a aplicação móvel a ser estudada como base para o estudo, após isso foi realizada a impressão e montagem do robô a ser utilizado como meio para extração dos dados. Esse mapeamento foi realizado por meio de um robô triângulo delta *Tapster* que possui todas as características propícias para a realização de testes automatizados em dispositivos móveis: estrutura com tamanhos para acoplagem de dispositivos, interação com a tela, conexão do telefone com o computador pelo modo depuração própria do sistema Android e conexões e juntas para fixação física do dispositivo. Utilizando a linguagem *JavaScript*, foi possível estabelecer a comunicação entre o sistema embarcado Arduino e os servomotores de forma síncrona ou assíncrona, extrair a árvore XML da aplicação por meio de *Dump* com o recurso *UIAutomatorViewer*, utilizando chamadas de serviços do sistema operacional e a linguagem *Python.* Após a extração obteve-se um conjunto de dados, dos quais foram classificados e analisados os valores máximos e mínimos de um elemento da aplicação, por exemplo, o quanto um botão pode ser clicado em suas extremidades ou bordas.

Palavras-chave: Tapster, *JavaScript*, Arduino, *UIAutomatorViewer*, *python*.

Defining limit value extremes using delta triangle robot

Author: Paulo Henrique Araújo Munhoz
Advisor: Francisco de Assis Pereira Januário, Msc.

ABSTRACT

The general objective of this work is to present an extracted database containing the limit values of a simple mobile device application. Initially, the mobile application to be used as the basis for the study was defined, after which the printing and assembly of the robot to be used as a means for data extraction were performed. This mapping was carried out using a delta Tapster triangle robot that has all the characteristics suitable for performing automated tests on mobile devices: structure with sizes for device coupling, interaction with the screen, the connection between the phone and the computer using debug mode own Android system and connections and joints for physical fixation of the device. Using the JavaScript language, it was possible to establish communication between the Arduino embedded system and the servomotors synchronously or asynchronously, extracting the application's XML tree through Dump with the UIAutomatorViewer resource, using service calls from the operating system and the Python language. After extraction, a set of data was obtained, from which the maximum and minimum values of an element of the application were classified and analyzed, for example, how much a button can be clicked on its ends or edges.

Keywords: Tapster, JavaScript, Arduino, UIAutomatorViewer, python.

Sumário

Lista de abreviaturas e siglas

CEO	*Chief Executive officer*
GDL	Grau de Liberdade
NPM	*Node Package Manager*
ADB	*Android Debug Bridge*
ADBD	*Android Debug Bridge Daemon*
UI	*User Interface*
STL	*Standard Triangle Language*
CAD	*Computer Aided Design*
3D	*Three-dimensional*
ABS	Acrilonitrila Butadieno Estireno
XML	*eXtensible Markup Language*
Kb	*Kilobytes*
Mb	*Megabytes*

Lista de figuras

Lista de tabelas

1 Introdução

Uma aplicação móvel é um software projetado para executar em smartphones, tablets e outros dispositivos móveis, considerando informações contextuais de entrada (CHEN & KOTZ, 2000). Ao mesmo tempo em que as aplicações móveis estão se tornando cada vez maiores e mais complexas, a exigência de qualidade nestes produtos também tem aumentado. O mercado de testes automatizados é considerado um mercado novo e em constante evolução. Diversas ferramentas foram desenvolvidas com esse objetivo. No entanto, muitas delas têm a dificuldade em simular os gestos[1] realizados pelo usuário. Uma das formas de suprir essa necessidade é a utilização de robôs.

Um robô é um dispositivo, ou grupo de dispositivos, eletromecânico capaz de realizar trabalhos de maneira autônoma ou pré-programada (GONÇALVES, 2007). Os robôs são classificados em três tipos (USATEGUI & LEÓN, 1990):

- **Robôs Industriais** - Formados por uma estrutura mecânica articulada, que se move adaptando-se a diferentes configurações pelas ordens recebidas de um equipamento e controle baseado normalmente em um microprocessador. Os robôs industriais são os de maior difusão em tarefas de alcance econômico. (SECCHI, 2008).

- **Robôs Médicos** - Segundo (USATEGUI & LEÓN, 1990), dá-se o nome de robôs médicos às próteses de braços, pernas ou mãos biônicas aos que possuem motorização própria e que tenham capacidade de funcionamento com autonomia e com ações reflexas. Os robôs médicos de cooperação ou de reabilitação são concebidos como próteses inteligentes para as pessoas com necessidades físicas que se diferenciam do resto em sua forma procurando ter a aparência da correspondente extremidade humana.

- **Robôs Móveis** – São dispositivos de transporte automático, ou seja, são dotados de navegar através de um determinado ambiente de

[1] O gesto é a capacidade de o indivíduo expressar uma variedade de sentimentos e pensamentos com a comunicação não-verbal. É realizada por meio das partes do corpo, como mão, braços e expressões fisionômicas.

trabalho, tendo nível de autonomia para sua locomoção. Suas aplicações estão relacionadas com tarefas que normalmente são arriscadas ou nocivas para a saúde humana. (SECCHI, 2008). Um tipo de robô muito eficiente, geralmente utilizado na indústria é o robô paralelo tipo delta, definido como um manipulador multifuncional reprogramável. Este robô foi projetado para mover materiais, peças, ferramentas ou dispositivos através de movimentos que podem ser programáveis e variados para o desenvolvimento de diversas tarefas (PANDILOV & DUKOVSKI, 2004). Esse robô delta possui os seguintes equipamentos constituintes (OTTONI, 2010):

- Manipuladores: São as partes mecânicas que realizam os movimentos;
- Atuadores: São os motores que movimentam os manipuladores;
- Controlador: É o computador;
- Fonte de energia: Podem ser por baterias ou rede elétrica;
- Drivers: Controla o motor de passo;
- Arduino: Faz a interpretação dos dados e envia os comandos para os motores;
- Transmissão de energia e dados: São as fiações do sistema.

A ideia básica do robô delta é o uso de três paralelogramos que mantém constante a orientação da plataforma móvel, mesmo com três graus de liberdades puramente de translação. Os três paralelogramos giram sobre três juntas de rotação. A grande vantagem do robô delta é que os atuadores estão na base fixa, as hastes e os manipuladores têm baixo peso permitindo grandes velocidades de operação (SALABARRIA, 2007).

Segundo Jason Huggins (FINLEY, 2013), um dos robôs que imita o toque do dedo humano, *Tapster*, é considerado uma nova forma de testar aplicações móveis em dispositivos que tenham tela sensível ao toque. Normalmente, desenvolvedoras de aplicativos contam com pessoas reais para realizar testes que não podem ser facilmente reproduzidos por ferramentas ou emuladores, mas trata-se de uma tarefa exaustiva e de alto custo. De acordo

com Bryce Day (FINLEY, 2013), CEO de uma empresa de testes, o teste manual é visto como a forma menos eficiente no ciclo de testes.

1.1 A problemática da Automação de Teste de Software

Na automatização de testes de software existem certas dificuldades já conhecidas na literatura técnica e em âmbito industrial, como uma equipe despreparada, os altos custos, a mão de obra não qualificada, dificuldades quanto ao funcionamento dos casos lógicos de programação e os casos de teste criados para a aplicação.

As empresas apresentam certa resistência na implantação da automatização de testes, pois é necessário um alto investimento inicial e sendo necessário um longo prazo para obtenção de retorno financeiro. Os custos elevados são oriundos da necessidade de compra de ferramentas para a criação e execução dos testes, treinamento de equipe, contratação de profissionais qualificados, entre outros fatores. Sendo necessária então uma estratégia para implantação do teste automatizado com menos custos e maior produtividade. Apesar dos testes automatizados serem desenvolvidos com ferramentas já existentes no mercado, cada vez mais os testes automatizados, utilizando robôs, vem sendo requisitado para atuar em testes de desempenhos, exaustivos, repetições, UI e entre outros. O teste apresentado nesse trabalho, classifica um conjunto de valores que podem ser mapeados para cada tipo de dispositivo, simulando o toque no botão da aplicação calculadora em um *smartphone*, e gera uma base de dados, com um conjunto de máximo e mínimos das coordenadas da tela.

1.2 Objetivos

O objetivo geral do trabalho é extrair os valores limites de uma aplicação de calculadora, de um *smartphone*, por meio de um robô triângulo delta, utilizando uma caneta que simula o toque humano a cada interação de abscissa, coordenada e profundidade no espaço.

Os objetivos específicos deste trabalho visam o seguinte:

a) Levantar técnicas existentes em teste de *software* aplicadas a robotização;

b) Replicação e montagem de robô triângulo delta (*Tapster*);

c) Teste de repetibilidade e precisão com o robô triângulo delta;

d) Adquirir e analisar os valores limites dos botões da aplicação.

1.3 Organização do trabalho

O Capítulo 1 apresentou uma visão geral da proposta de trabalho, contextualizando o tema com áreas como a robótica e a engenharia de software. Apresentou a problemática, a motivação e objetivo geral, assim como a estruturação do processo de trabalho.

O Capítulo 2 apresentam o referencial teórico com ênfase no robô triângulo delta, suas aplicações, componentes e descrição das fórmulas responsáveis pela cinemática do equipamento. Teste de Software, sua importância, fundamentos e análise do valor limite.

O Capítulo 3 apresenta o procedimento de montagem do robô, desde seus arquivos fonte até a manufatura das peças.

O Capítulo 4 descreve a aplicação do robô triângulo delta para extrair valores limite em forma de coordenadas de uma aplicação móvel.

O Capítulo 5 apresenta as conclusões e trabalhos futuros.

2 Fundamentação teórica

2.1 Robô triângulo delta

Sistemas mecânicos que permitem que um corpo rígido se mova em relação a uma base fixa, desempenham um papel muito importante em inúmeras aplicações. Um corpo rígido no espaço pode se mover de diversas maneiras, com movimentos rotativos ou de translação. Estes são denominados seus graus de liberdade, entretanto, não se pode exceder o total de seis graus de liberdade de um corpo rígido no espaço (três movimentos translacionais ao longo dos eixos do plano ortogonal mútuo e três movimentos rotativos em torno desses eixos). A posição e a orientação do efetuador[2] podem ser descritas por suas coordenadas generalizadas, estas na maioria das vezes são as coordenadas de um ponto final específico do efetuador e os ângulos que definem sua orientação, mas pode ser qualquer outro conjunto de parâmetro que permita definir a posição e a orientação do efetuador. Quando for possível controlar os vários graus de liberdade do efetuador através de um sistema mecânico, este sistema se chamará de robô (MERLET, 2006)

Professor Reymond Clavel, no Instituto Federal Suíço de Tecnologia de Lausanne em 1988, desenvolveu a ideia do uso de robôs paralelos com três graus de liberdade translacional e um grau de liberdade rotacional, criando os robôs delta, como mostrado na Figura 1. Sua ideia passa pelo uso de paralelogramos que permitem que um elo[3] de saída permaneça em uma posição fixa em relação ao elo de entrada. A função dos paralelogramos é restringir o efetuador, permanecendo apenas com três graus de liberdade (BONEV, 2001)

[2] Termo utilizado para a base móvel.

[3] elos de robôs são estruturas rígidas interligadas por meio de juntas (CRAIG, 2005). Dependendo da literatura os elos podem ser denominados links ou braços.

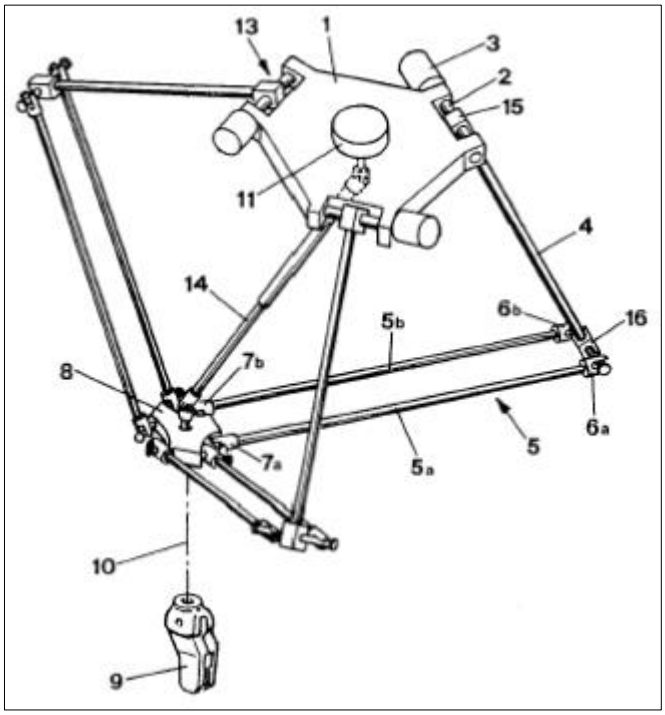

Figura 1. Robô Delta. *(BONEV, 2001)*

Abaixo segue a especificação dos componentes da estrutura do robô Delta:

Número	Componentes
1	Base superior
2	Conexão com atuador
3	Atuador
4	Braço superior (elo)
5	Braço inferior (paralelogramo)
6	Terminais rotulares (juntas esféricas)
7	Terminais rotulares (juntas esféricas)

8	Base inferior (efetuador)
9	Ferramenta
10	Conexão com o quarto eixo
11	Atuador do quarto eixo

Os três elos superiores idênticos e simétricos estão conectados à plataforma fixa através de juntas[4] rotacionais, estas juntas permitem a rotação de apenas um dos corpos em relação a outro, portanto possui apenas um grau de liberdade. Estes, por sua vez, são conectados aos *links* inferiores através de juntas esféricas. A posição do efetuador é alterada produzindo um movimento de translação à medida que são alteradas as posições angulares na base fixa. Em alguns robôs pode ainda existir o quarto grau de liberdade, utilizado para transmitir o movimento rotatório (PREMPRANEERACH, 2014).

A velocidade é a principal característica do robô delta. A diferença entre os elos dos robôs série e dos delta, é que robôs série além de mover a carga útil tem que movimentar todos os servos em cada elo e um delta apenas move sua estrutura de material mais leve, devido a essas vantagens, os robôs delta atingem acelerações em todos os seus 12 graus de liberdade, em aplicações industriais, tornando-se um candidato perfeito para aplicações de *pick and place*[5] *(*Termo usual na robótica para aplicações de pegar um objeto e reposicionar em outro local*)* de pequenas massas (BONEV, 2001).

Além da velocidade, os robôs paralelos apresentam outras vantagens em relação aos tradicionais robôs de arquitetura série, o processo de montagem, fabricação dos componentes mecânicos que o compõe e o controle de movimento são mais manifestos. Existem vários exemplos de robôs paralelos, especialmente em áreas de montagem e aplicações médicas. Entretanto, essa arquitetura possui um espaço de trabalho menor e de formato irregular além

[4] Termo pode ser encontrado como articulação em algumas literaturas.

[5] Termo usual na robótica para aplicações de pegar um objeto e reposicionar em outro local.

de menos destreza (HADIAN & FATTAH, 2008).

2.1.1 Cinemática

A cinemática de um robô envolve a análise do movimento de sua estrutura com seus respectivos graus de liberdade (GDL). Geralmente a estrutura de um robô é constituída por peças rígidas chamadas *links* e juntas com movimentos rotacionais ou translacionais. Cinemática é o estudo do movimento do sistema robótico sem considerar as forças que são causadas. Na robótica, a cinemática pode ser dividida em duas categorias: a cinemática inversa e a cinemática direta, além disso também é utilizada para gerar o espaço de trabalho do robô (PRANAV, MUKILAN, & GANESH, 2016).

Um robô delta é um manipulador de cadeia cinemática fechada. É constituído por uma base fixa e efetuador, os quais são interconectados por três links que formam um paralelogramo, portanto o efetuador está restrito a movimentos translacionais e limitado pelas dimensões de seus links (ARRAZATE, 2017).

O robô triangulo delta é capaz de controlar a translação XYZ de sua plataforma móvel dentro de seu espaço de trabalho, as juntas de rotação superiores acionado por meio de atuadores rotacionais fixos na base, $\theta_1, \theta_2 \, e \, \theta_3$, como mostrado na Figura 2.

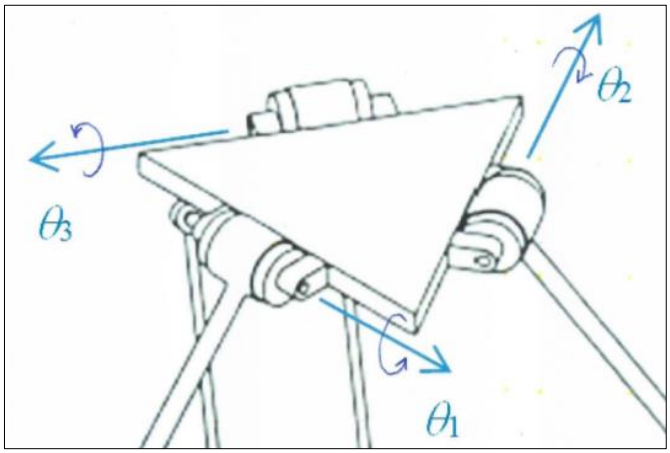

Figura 2. Variáveis da articulação. *(WILLIAMS II, 2016)*

É possível observar na Figura 3 e Figura 4 que a referência da base fixa no plano cartesiano é dada por B, e da base móvel por P respectivamente. Devido a ambas terem como origem o centro de cada triângulo a orientação de P é sempre igual a de B, assim a matriz rotacional ${}^{B}_{P}R = I_3$ é constante, isso ocorre pelo fato de a arquitetura delta não permitir rotações. As variáveis das articulações são θ_i $(\theta_1, \theta_2\, e\, \theta_3,)$ e as variáveis cartesianas ${}^{B}P_p = \{x\ y\ z\}^T$ (WILLIAMS II, 2016).

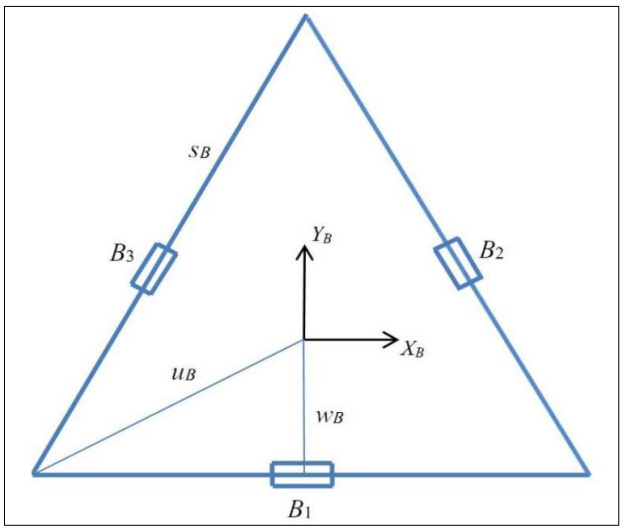

Figura 3. Detalhes da Base Superior. *(WILLIAMS II, 2016)*

A distância do centro do triângulo até a base é W_b e a distância entre o centro e o vértice, denomina-se U_b. A partir do uso da trigonometria é possível determinar as posições das juntas de revolução em relação ao centro da base fixa utilizando a Equação 1 (WILLIAMS II, 2016)

$$B_{\boldsymbol{B_1}} = \begin{Bmatrix} 0 \\ -W_b \\ 0 \end{Bmatrix} \qquad B_{\boldsymbol{B_2}} = \begin{Bmatrix} \sqrt{\frac{3}{2}}\, W_b \\ \frac{1}{2} W_b \\ 0 \end{Bmatrix} \quad B_{\boldsymbol{B_3}} = \begin{Bmatrix} 0 \\ -W_b \\ 0 \end{Bmatrix} \tag{1}$$

A mesma relação pode ser feita para a base móvel, segundo a Equação 2, pois os pontos de fixação das juntas esféricas também são constantes, ou seja, possuem conexão fixa na estrutura da base (WILLIAMS II, 2016).

$$P_{\boldsymbol{P_1}} = \begin{Bmatrix} 0 \\ -u_p \\ 0 \end{Bmatrix} \qquad P_{\boldsymbol{P_2}} = \begin{Bmatrix} \frac{SP}{2} \\ W_p \\ 0 \end{Bmatrix} \quad P_{\boldsymbol{P_3}} = \begin{Bmatrix} -\frac{SP}{2} \\ W_p \\ 0 \end{Bmatrix} \tag{2}$$

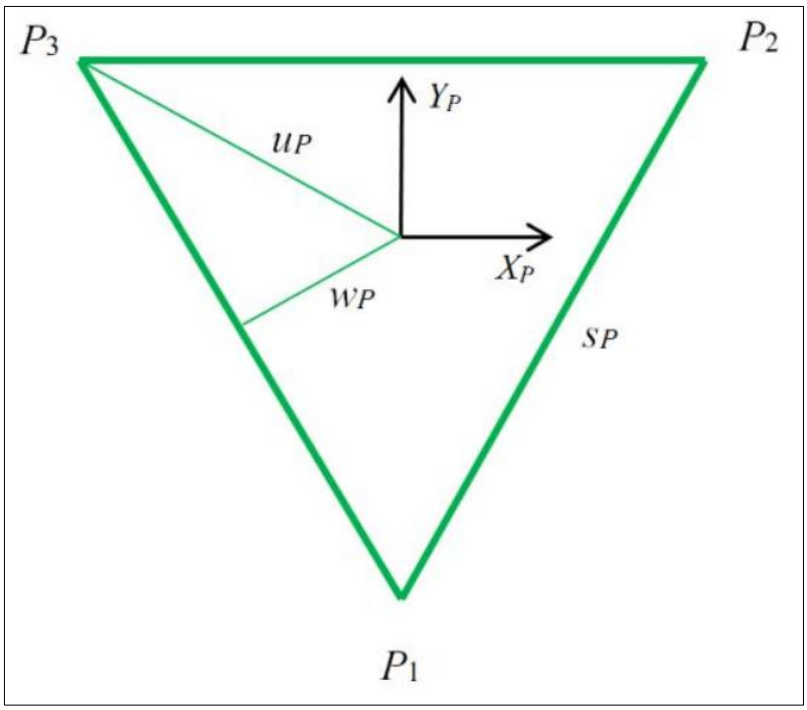

Figura 4. Base móvel. *(WILLIAMS II, 2016)*

Os vértices da base fixam são extraídos a partir da Equação 3:

$$B_{b_1} = \begin{Bmatrix} \frac{sb}{2} \\ -W_b \\ 0 \end{Bmatrix} \qquad B_{b_2} = \begin{Bmatrix} 0 \\ ub \\ 0 \end{Bmatrix} \quad B_{b_3} = \begin{Bmatrix} -\frac{sb}{2} \\ -W_b \\ 0 \end{Bmatrix} \tag{3}$$

Onde:

$$W_b = \frac{\sqrt{3}}{6}\, x\, S_b \qquad ub = \frac{\sqrt{3}}{3}\, x\, S_b \qquad w_p = \frac{\sqrt{3}}{3}\, x\, S_p \qquad up = \frac{\sqrt{3}}{6}\, x\, S_p$$

2.1.2 Cinemática inversa

Para expressar o movimento de um robô delta, dada a localização do seu efetuador é necessário conhecer o princípio da cinemática inversa e as características do robô delta. O princípio da cinemática inversa diz que só será possível obter a posição final do efetuador após obter os parâmetros das juntas que fornecem a posição desejada do efetuador (ARRAZATE, 2017). O robô delta é formado por três elos idênticos em paralelo entre a base superior (fixa) e a inferior (móvel). A primeira articulação é constituída por atuadores fixos a base que permitem apenas movimentos rotacionais, formando as

juntas de revolução. Os movimentos dessas juntas podem ser mensurados com o uso da regra da mão direita, onde é definido o ângulo zero quando o elo do atuador[6] se encontra no plano horizontal. As variáveis utilizadas para controle são definidas como, θi e i = 1,2,3 que são variáveis dos eixos mostrados, conforme Figura 5. (ARRAZATE, 2017)

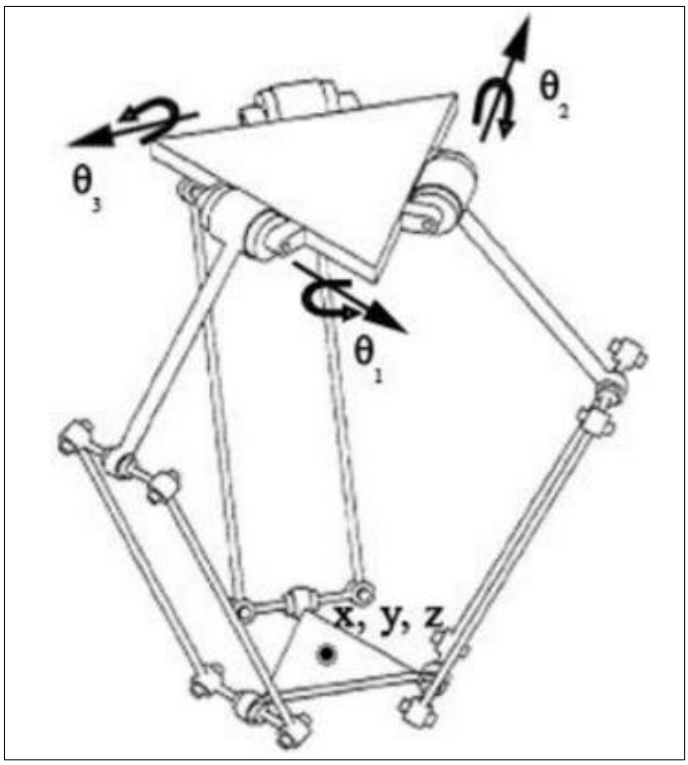

Figura 5. Representação das juntas de revolução. (ARRAZATE, 2017)

A base superior e a plataforma móvel inferior são formadas por triângulos equiláteros, sendo Sb definido como comprimento lateral da base superior e Sp o comprimento lateral da base móvel inferior. Conforme Figura 6, a plataforma móvel é invertida em relação a base fixa (WILLIAMS II, 2016).

[6] Elo de atuador é a articulação na qual está conectado o mecanismo que irá movimentar o robô.

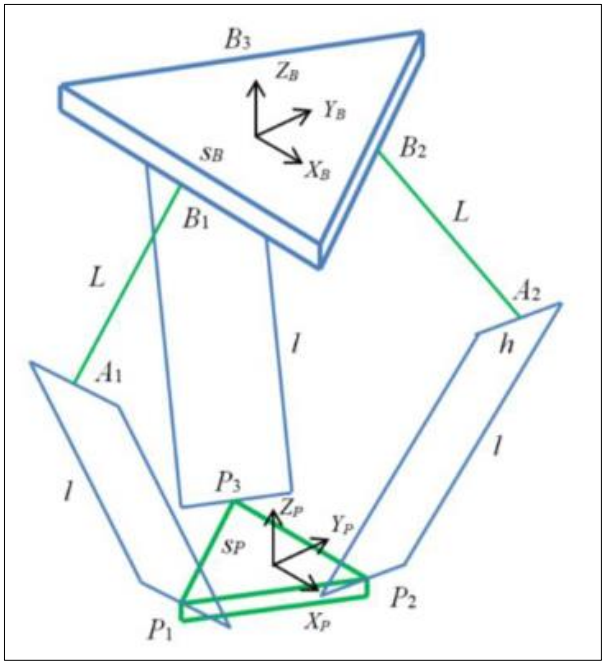

Figura 6. Diagrama cinemático. *(WILLIAMS II, 2016)*

De acordo com (WILLIAMS II, 2016), para produzir a solução da cinemática inversa, é necessário definir variáveis intermediárias referenciadas a base fixa a partir do ponto do efetuador (x, y, z).

As variáveis intermediárias são definidas conforme as equações a, b e c abaixo:

$$a = Wb - Up \qquad b = \frac{sp}{2} - \frac{\sqrt{3}}{2}Wb \qquad c = Wp - \frac{1}{2}Wb \tag{4}$$

As equações cinemáticas para o robô delta são produzidas a partir das equações de limitação, mostradas em Equação 5, 6 e 7:

$$2L(y+a)cos\theta_1 + 2zLsen\theta_1 + x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 = 0 \tag{5}$$

$$-L\left(\sqrt{3}(x+b) + y + c\right)cos\theta_2 + 2zLsen\theta_2 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2xb + 2yc - l^2 = 0 \tag{6}$$

$$L\left(\sqrt{3}(x-b) - y - c\right)cos\theta_3 + 2zLsen\theta_3 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 - 2xb + 2yc - l^2 = 0 \tag{7}$$

Escrevendo as equações conforme a Equação 8 para i =1 ,2 ,3.

$$E_i cos\theta_i + F_i sen\theta_i + G_i = 0 \tag{8}$$

Em que E_i, F_i, G_i, mostrados nas Equações 9-15 são os complementos das equações cinemáticas:

$$\boldsymbol{E_1} = 2L(y+a) \tag{9}$$

$$\boldsymbol{F_1} = \boldsymbol{F_2} = \boldsymbol{F_3} = 2zL \quad (10)$$

$$\boldsymbol{G_1} = x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 \quad (11)$$

$$\boldsymbol{E_2} = -L(\sqrt{3}(x + b) + y + c) \quad (12)$$

$$\boldsymbol{G_2} = x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2xb + 2yc - l^2 \quad (13)$$

$$\boldsymbol{E_3} = L(\sqrt{3}(x - b) - y - c) \quad (14)$$

$$\boldsymbol{G_3} = x^2 + y^2 + z^2 + b^2 + c^2 + L^2 - 2xb + 2yc - l^2 \quad (15)$$

Usando a substituição da Tangente de meio ângulo e as definindo conforme mostrada nas Equações 16-18:

$$t_i = \tan\frac{\theta_i}{2} \quad (16)$$

$$cos\theta_i = \frac{1-t_i^2}{1+t_i^2} \quad (17)$$

$$sen\theta_i = \frac{2t_i}{1+t_i^2} \quad (18)$$

Nas Equações 19-21 é apresentada a aplicação das Equações 17 e 18 na equação 8:

$$E_i\frac{1-t_i^2}{1+t_i^2} + F_i\frac{2t_i}{1+t_i^2} + G_i = 0 \quad (19)$$

$$E_i(1 - t_i^2) + F_i(2t_i) + G_i(1 + t_i^2) = 0 \quad (20)$$

$$(G_i - E_i)t_i^2 + (2F_i)t_i + (G_i + E_i) = 0 \quad (21)$$

O resultando será uma equação quadrática como demonstrado na Equação 22:

$$t_{i1,2} = \frac{-F_i \pm \sqrt{E_i^2 + F_i^2 - G_i^2}}{G_i - E_i} \quad (22)$$

Retornando para a equação em relação a θ_i, invertendo a definição de tangente de meio ângulo, temos como resultado a Equação 23:

$$\theta_i = 2tan^{-1}(t_1(1,2,3)) \quad (23)$$

Ambas soluções apresentadas pela equação quadrática são soluções válidas para cada um dos ângulos das juntas, sendo uma solução positiva e outra negativa. No entanto, deve-se optar pela solução que esteja dentro do espaço de trabalho do robô. A solução positiva da raiz quadrada, apresenta

fisicamente características com os elos do robô posicionados para fora, representados conforme Figura 7 com os pontos P1 e P2 sendo seus limites. Esta deve ser a opção escolhida, pois utilizando a outra solução da raiz, resultaria em possíveis colisões com a estrutura do robô (WILLIAMS II, 2016) (ARRAZATE, 2017).

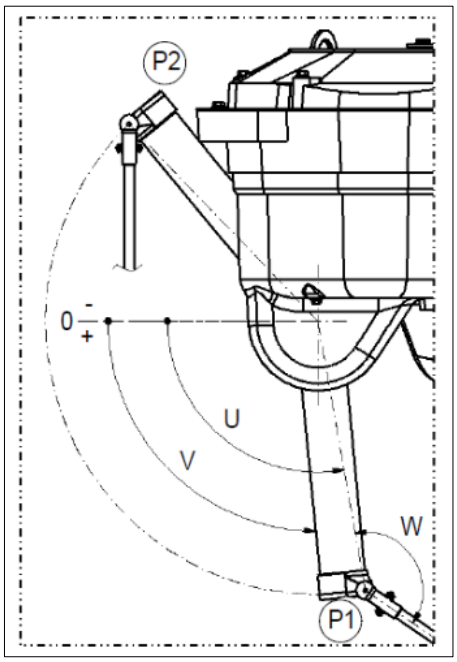

Figura 7. Posição extremas do robô. *(ARRAZATE, 2017)*

2.1.3 Cinemática direta

A cinemática direta determina a partir do ângulo θ_j (j= 1, 2, 3) das juntas rotativas atuadas[7] as coordenadas cartesianas (x, y, z) para qualquer configuração do robô delta, conforme Figura 8. (LARIBI, ROMDHANE, & ZEGHLOUL, 2008).

[7] Juntas rotativas atuadas são as juntas ativas. Normalmente compostos por servo motores ou motores de passo.

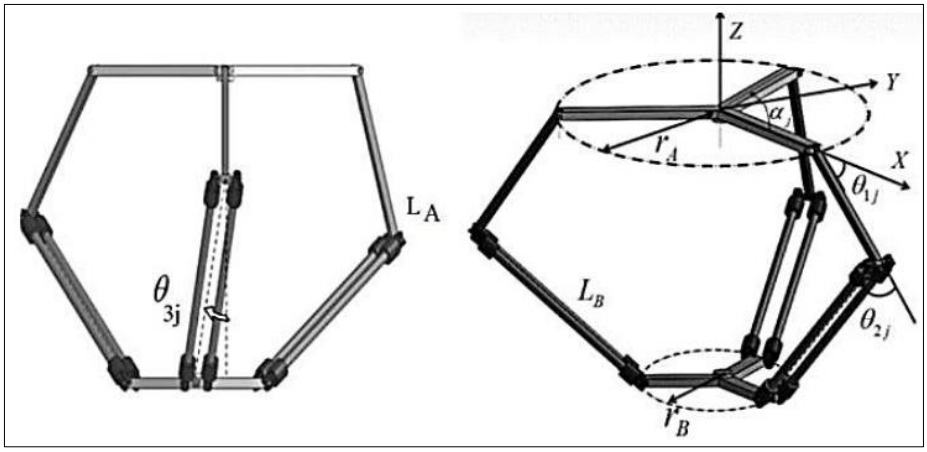

Figura 8. Representação das variáveis utilizadas na cinemática direta. *(GHARAHSOFLOO & RAHMANI, 2015)*

Os parâmetros geométricos do robô delta são dados como: $L_A, L_B, r_A, r_B, \theta_j$ para (j= 1, 2, 3), sendo L_A o comprimento do elo superior, L_B o comprimento do elo inferior e r_A e r_B o raio da base superior e inferior respectivamente, conforme definido nas Figura 8 e Figura 9, assim como para os ângulos das juntas articuladas θ_i(i= 1, 2, 3) que definem a configuração de cada braço[8] (GHARAHSOFLOO & RAHMANI, 2015).

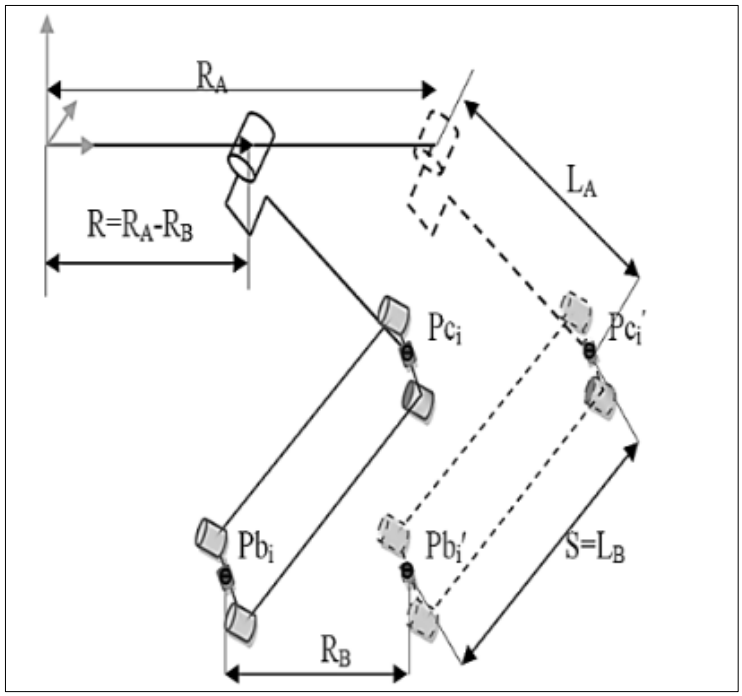

Figura 9. Parâmetros geométricos. *(GHARAHSOFLOO & RAHMANI, 2015)*

[8] O braço do robô é formado por elos e juntas.

Seja P um ponto localizado no centro da base móvel, o modelo geométrico pode ser descrito conforme as Equações 24, 25 e 26. (LARIBI, ROMDHANE, & ZEGHLOUL, 2008):

$$X_p = cosa_j\left(r_A + L_A cos\theta_{1j} + L_B cos\theta_{3j}\cos\left(\theta_{1j} + \theta_{2j}\right) - r_B\right) - L_A sena_j sen\theta_{3j} \quad (24)$$

$$Y_p = \cos a_j\left(r_A + L_A cos\theta_{1j} + L_B cos\theta_{3j}\cos\left(\theta_{1j} + \theta_{2j}\right) - r_B\right) + L_B \cos a_j\, sen\ \theta_{3j} \quad (25)$$

$$Z_p = L_A sen\theta_{1j} + L_B cos\theta_{3j} sen(\theta_{1j} + \theta_{2j}) \quad (26)$$

Onde $[X_p, Y_p, Z_p]$ são coordenadas no ponto P para um determinado ângulo de articulação $\theta_{1j}, \theta_{2j}, \theta_{3j}$ (j = 1,2,3).

De maneira a eliminar as variáveis das juntas passivas das Equações 24, 25 e 26 deve-se elevá-las ao quadrado, resultando em:

$$\left(x_j - X_p\right)^2 + \left(Y_j - Y_p\right)^2 + \left(Z_j - Z_p\right)^2 = L_B^2$$

Em que j=1,2,3 e r é a diferença entre os raios das bases mostradas nas Equações 27, 28, 29 e 30.

$$r = r_A - r_B \quad (27)$$

$$X_j = \left(r + L_A \cos\theta_{1j}\right)\cos a_j \quad (28)$$

$$Y_j = \left(r + L_A \cos\theta_{1j}\right) sen\ a_j \quad (29)$$

$$Z_j = -L_A sen\ \theta_{1j} \quad (30)$$

O resultado representa três esferas com centros (X_j, Y_j, Z_j)para j = 1,2,3 e com raio LB. A solução da equação é o ponto de interseção das três esferas, como pode ser observado na Figura 10. Percebe-se que duas soluções são possíveis para o problema da cinemática direta do robô delta. As soluções estão localizadas na parte superior e inferior a base fixa do robô e devem ser escolhidas de acordo com a alocação do espaço de trabalho selecionado (GHARAHSOFLOO & RAHMANI, 2015).

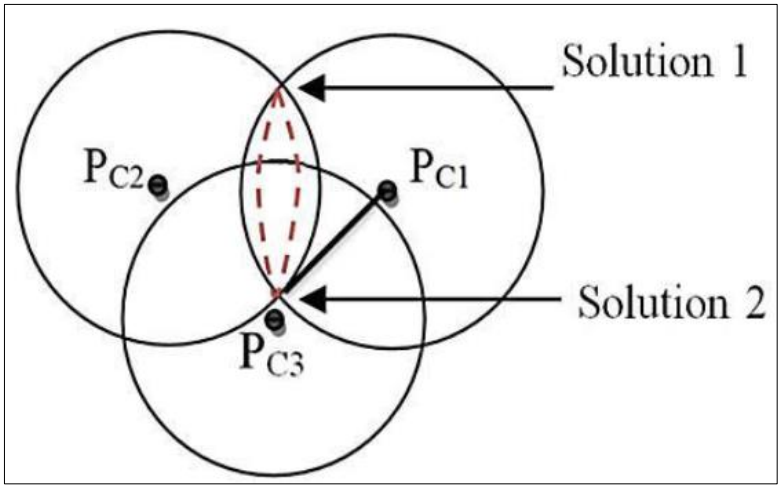

Figura 10. Duas soluções para cinemática direta. (GHARAHSOFLOO e RAHMANI, 2015)

2.2 Gerenciador de Pacotes do Node (NPM)

O código do robô triângulo delta está disponível na linguagem Javascript e pode ser facilmente configurado utilizando o repositório de pacotes do Node, ele serve para compartilhar ferramentas, instalar vários módulos e gerenciar suas dependências. Segundo o autor (LONGEN, 2021), o npm baseia-se em:

- Repositório amplamente usado para a publicação de projetos Node.js de código aberto (*open-source*). Isso significa que ele é uma plataforma online onde qualquer pessoa pode publicar e compartilhar ferramentas escritas em JavaScript.

- Ferramenta de linha de comando que ajuda a interagir com plataformas online, como navegadores e servidores. Essa utilidade auxilia na instalação e desinstalação de pacotes, gerenciamento das versões e gerenciamento de dependências necessárias para executar um projeto.

2.3 Android Debug Bridge

O Android Debug Bridge é uma ferramenta de linha de comando versátil que permite a comunicação com um dispositivo móvel, tais comandos adb facilitam ações que se queira inferir a um dispositivo, detalhado na Figura 11, como por exemplo: instalar aplicações, depurar aplicações e fornecer acesso a um *shell Unix* que pode ser usado para executar diversos comandos em um

telefone, (GOOGLE, 2021). Ele é um programa cliente-servidor com três componentes:

- O cliente, que é executado no computador de desenvolvimento, o cliente envia comandos para invocar o cliente de um terminal pelo terminal de linha de comando.
- Um *daemon* (adbd), que executa comandos em um dispositivo. O *daemon* é executado como um processo em segundo plano em cada dispositivo.
- Um servidor, que gerencia a comunicação entre o cliente e o *daemon*. O servidor é executado como um processo em segundo plano na máquina de desenvolvimento.

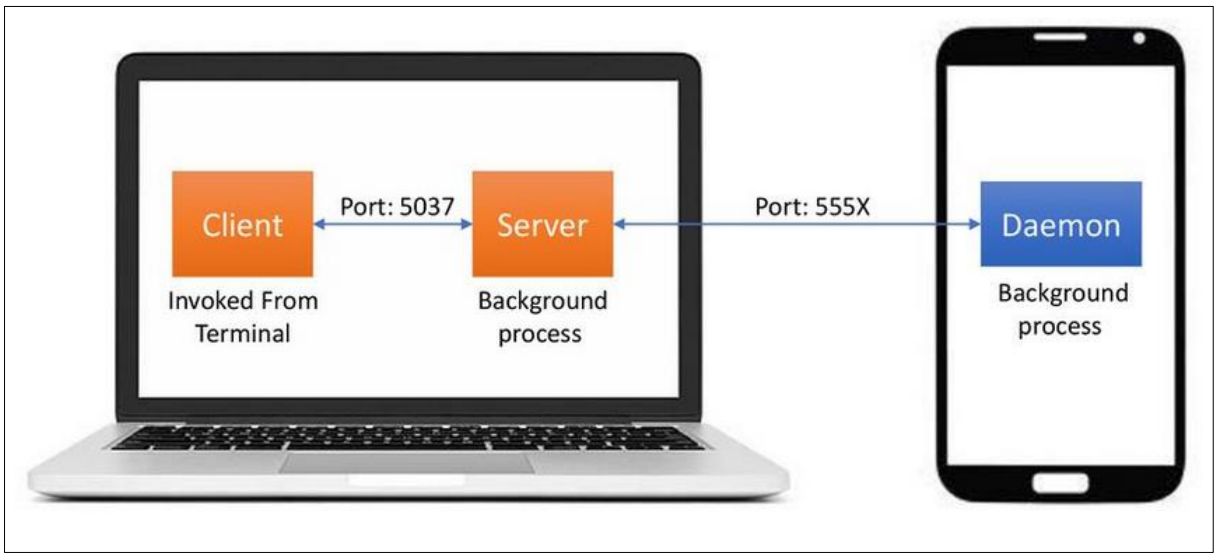

Figura 11. Arquitetura do Android Debug Bridge, *(MATTHEWS, s.d.)*

2.4 UIAutomatorviewer

Segundo a (GOOGLE, 2021), o UIAutomator é um framework de testes de UI recomendado para testes funcionais, ou seja, testa a funcionalidade, mas não tendo o acesso ao código fonte de UI das aplicações instaladas, incluindo os do sistema.

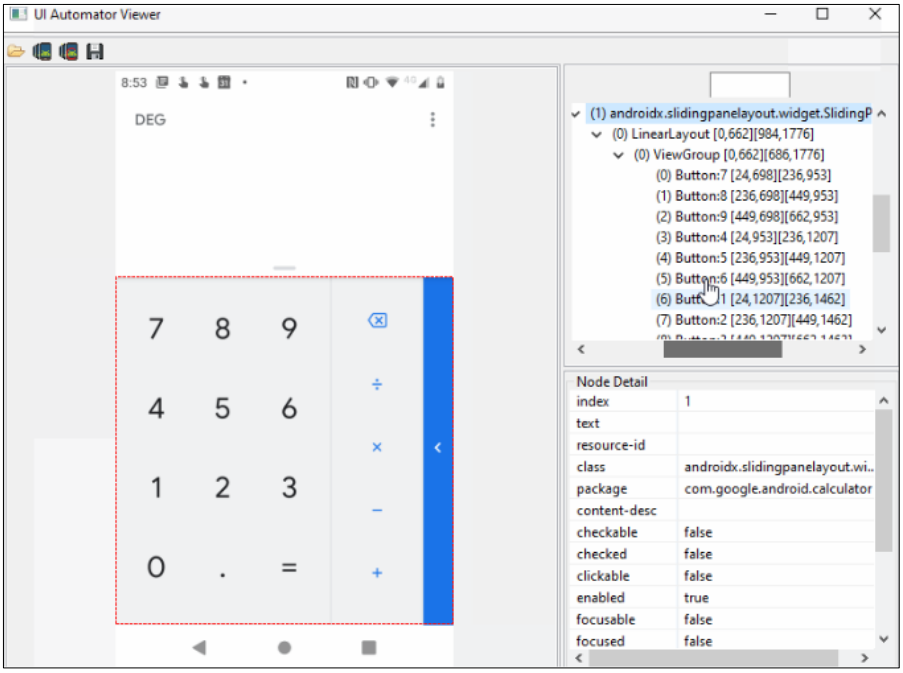

Figura 12. *Dump* da aplicação Calculadora

Sendo assim, o UIAutomator é recomendado na programação para testes automatizados, em que o código do teste não se baseia em detalhes de implementação das aplicações a serem testados, estilo caixa preta.

Essa ferramenta tem um visualizador para inspecionar a hierarquia de *layouts* e as propriedades de cada componente de UI que estão presentes na tela mostrada na Figura 12, podendo dá ao testador níveis de detalhes para a criação de testes.

As principais classes utilizadas para a automatização de testes são:

- UiObject: representa um elemento de UI visível no dispositivo. Exemplos de métodos: click(), getText(), isClickable()
- UiSelector: representa uma consulta para um ou mais elementos de UI de destino em um dispositivo. Exemplos de métodos: resourceId(String id), text(String text), textContains(String text)
- UiScrollable: simula rolagens verticais ou horizontais em uma tela. Exemplos de métodos: scrollForward(), scrollToEnd(int maxSwipes)

Além disso, existe uma classe chamada de UiDevice, em que você consegue acessar propriedades do dispositivo, como orientação atual, pressionar os botões “Voltar”, “Início” ou “Menu” por meio de seus métodos.

2.5 Teste de Software

Segundo (OLSEN, POSTHUMA, & ULRICH, 2018), os sistemas de software são parte integrante do cotidiano, desde aplicações comerciais (p. ex., serviços bancários) até produtos de consumo (p. ex., carros). A maioria das pessoas já teve uma experiência com software que não funcionou como o esperado. O software que não funciona corretamente pode levar a muitos problemas, incluindo a perda de dinheiro, de tempo ou da reputação comercial, e até mesmo ferimentos ou morte. O teste de software é uma maneira de avaliar a qualidade do software e reduzir o risco de falha do software em operação.

É muito comum pensar que o papel do teste seja uma atividade de apenas executar testes, ou seja, executar o software e verificar os resultados. Porém o processo de teste também inclui atividades como planejamento de teste, análise, modelagem e implementação dos testes, relatórios de progresso e resultados de testes e avaliação da qualidade de um objeto de teste.

Alguns testes envolvem a execução de um componente ou sistema que está sendo testado, sendo esse teste chamado de teste dinâmico. Outros testes não envolvem a execução do componente ou sistema que está sendo testado, sendo chamados de teste estático. Assim, o teste também inclui a revisão de produtos de trabalho, como requisitos, histórias de usuários e código-fonte.

É comum ter a impressão que o papel do teste é que ele se concentra inteiramente na verificação de requisitos, histórias de usuários ou outras especificações. Embora o teste envolva verificar se o sistema atende aos requisitos especificados, ele também contempla a validação, que está verificando se o sistema atenderá às necessidades do usuário e stakeholders em seu(s) ambiente(s) operacional(is). As atividades de teste são organizadas e executadas de forma diferente em diferentes ciclos de vida.

2.5.1 Objetivos típicos do teste

Para qualquer projeto, os objetivos do teste podem incluir informações para:

- Evitar defeitos, avaliar os produtos de trabalho, como requisitos, histórias de usuários, modelagem e código;
- Verificar se todos os requisitos especificados foram cumpridos;
- Verificar se o objeto de teste está completo e validar se funciona como os usuários e stakeholders esperam;
- Criar confiança no nível de qualidade do objeto de teste;
- Encontrar defeitos e falhas reduz o nível de risco de qualidade inadequada do software;
- Fornecer informações suficientes aos stakeholders para que tomem decisões especialmente em relação ao nível de qualidade do objeto de teste;
- Cumprir os requisitos ou normas contratuais, legais ou regulamentares, ou verificar a conformidade do objeto de teste com esses requisitos ou normas.
- Os objetivos do teste podem variar, dependendo do contexto do componente ou sistema que está sendo testado, do nível de teste e do modelo de ciclo de vida de desenvolvimento de software.

2.5.2 Erros, defeitos e falhas

Segundo (OLSEN, POSTHUMA, & ULRICH, 2018), uma pessoa pode cometer um erro (engano), que pode levar à introdução de um defeito (falha ou bug) no código do software ou em algum outro produto de trabalho relacionado. Um erro que leva à introdução de um defeito em um produto de trabalho pode acionar outro erro que leva à introdução de um defeito em um outro produto de trabalho relacionado. Por exemplo, um erro de elicitação de requisitos pode levar a um defeito de requisitos, o que resulta em um erro de programação que leva a um defeito no código.

A execução de um código defeituoso, pode causar uma falha, mas não necessariamente em todas as circunstâncias. Por exemplo, alguns defeitos exigem entradas ou pré-condições muito específicas para desencadear uma falha, o que pode ocorrer raramente ou nunca. Erros podem ocorrer por vários motivos, como:

- Pressão do tempo;
- Falha humana;
- Participantes do projeto inexperientes ou insuficientemente qualificados;
- Falta de comunicação entre os participantes do projeto, incluindo falta de comunicação sobre os requisitos e a modelagem;
- Complexidade do código, modelagem, arquitetura, o problema a ser resolvido ou as tecnologias utilizadas;
- Mal-entendidos sobre interfaces intra-sistema e entre sistemas, especialmente quando tais interações são em grande número;
- Tecnologias novas, ou desconhecidas.

Além das falhas causadas devido a defeitos no código, falhas também podem ser causadas por condições ambientais. Por exemplo, radiação, campos eletromagnéticos e poluição podem causar defeitos no firmware ou influenciar a execução do software alterando as condições do hardware.

Nem todos os resultados inesperados de teste são considerados falhas. Falsos positivos podem ocorrer devido a erros na forma como os testes foram executados ou devido a defeitos nos dados de teste, no ambiente de teste ou em outro *testware* ou por outros motivos. A situação inversa também pode ocorrer, onde erros ou defeitos similares levam a falsos negativos. Falsos negativos são testes que não detectam defeitos que deveriam ser detectados; falsos positivos são relatados como defeitos, mas na verdade não são defeitos.

2.5.3 Defeitos, causas-raiz e efeitos

As causas-raiz dos defeitos são as primeiras ações ou condições que contribuíram para a criação dos defeitos. Os defeitos podem ser analisados para identificar suas causas-raiz, de modo a reduzir a ocorrência de defeitos similares no futuro. Ao realçar as causas-raiz mais significativas, a análise de causa-raiz pode levar à melhoria de processo que evitam a introdução de um número significativo de defeitos futuros (OLSEN, POSTHUMA, & ULRICH, 2018).

2.5.4 Teste caixa branca

O teste caixa-branca é derivado de testes com base na estrutura interna ou na implementação do sistema. A estrutura interna pode incluir código, arquitetura, fluxos de trabalho e fluxos de dados dentro do sistema.

A eficácia dos testes caixa-branca pode ser medida através da cobertura estrutural. A cobertura estrutural é a extensão em que algum tipo de elemento estrutural foi testado e é expresso como uma porcentagem do tipo de elemento a ser coberto. (OLSEN, POSTHUMA, & ULRICH, 2018)

2.5.5 Teste caixa preta

As técnicas de teste caixa-preta (também chamadas de técnicas comportamentais ou baseadas no comportamento) são fundamentadas em uma análise da base de teste apropriada (por exemplo: documentos de requisitos formais, especificações, casos de uso, histórias de usuários ou processos de negócios). Essas técnicas são aplicáveis a testes funcionais e não funcionais. As técnicas de teste caixa-preta se concentram nas entradas e saídas do objeto de teste sem referência a sua estrutura interna.

As técnicas de teste baseadas na experiência aproveitam o conhecimento dos desenvolvedores, testadores e usuários para projetar, implementar e executar os testes. Estas técnicas são frequentemente combinadas com técnicas de teste caixa-preta e caixa-branca (JORGENSEN, 2014).

As características comuns das técnicas de teste caixa-preta incluem o seguinte:

- As condições de teste, os casos de teste e os dados de teste são derivados de uma base de teste que pode incluir requisitos de software, especificações, casos de uso e histórias de usuários;
- Os casos de teste podem ser usados para detectar lacunas entre os requisitos e as suas implementações, bem como desvios nos requisitos;
- A cobertura é medida com base nos itens testados na base de teste e na técnica aplicada nesta base.

Esse conhecimento e experiência inclui o uso esperado do software, seu ambiente, possíveis defeitos e a distribuição desses defeitos.

2.5.6 Análise de valor limite

A análise do valor limite é uma técnica de criação de casos de teste que complementa a técnica de partição de equivalência. Ao invés de selecionar qualquer elemento em uma partição de equivalência, a técnica de análise do valor limite conduz para a seleção de casos de teste nas "bordas" da partição. Ao invés de focar somente nas condições de entrada, a análise do valor limite deriva casos de teste dos resultados dos mesmos (MYERS, 2011). Os valores mínimo e máximo (ou primeiro e último valores) de uma partição são seus valores limites (BEIZER, 1990).

Por exemplo, suponha que um campo de entrada aceite um único valor inteiro como uma entrada, usando um teclado para limitar que entradas não inteiras sejam impossíveis. O intervalo válido é de 1 a 5, inclusive. Portanto, existem três partições de equivalência: inválidas (muito baixas); válido; inválido (muito alto). Para a partição de equivalência válida, os valores limites são 1 e 5. Para a partição inválida (muito alta), os valores limites são 6 e 9. Para a partição inválida (muito baixa), existe apenas um valor limite, 0, porque esta é uma partição com apenas um membro.

No exemplo acima, identificamos dois valores limites por limite. O limite entre inválido (muito baixo) e válido fornece os valores de teste 0 e 1. A

fronteira entre válido e inválido (muito alto) fornece os valores de teste 5 e 6. Algumas variações dessa técnica podem identificar três valores de limite por limite de dados: o valor anterior ao limite, o próprio valor limite e o imediatamente superior a ele. No exemplo anterior, usando valores limites de três pontos, os valores do limite inferior são 0, 1 e 2, e os valores dos limites superiores são 4, 5 e 6 (JORGENSEN, 2014).

O comportamento nos limites das partições de equivalência é mais provável que seja incorreto do que o comportamento dentro das partições. É importante lembrar que os limites especificados e implementados podem ser deslocados para posições acima ou abaixo de suas posições pretendidas, podem ser omitidos por completo ou podem ser complementados com limites adicionais indesejados. A análise e o teste do valor-limite revelarão quase todos esses defeitos forçando o software a mostrar comportamentos de uma partição diferente daquela à qual o valor limite deveria pertencer.

A análise de valor limite pode ser aplicada em todos os níveis de teste. Essa técnica é geralmente usada para testar requisitos que exigem um intervalo de números (incluindo datas e horas). A cobertura de limite para uma partição é medida como o número de valores limites testados, dividido pelo número total de valores de teste de limite identificados, normalmente expressos como uma porcentagem.

2.6 Teste exploratório

Nesta técnica, os testes informais (não pré-definidos) são modelados, executados, registrados e avaliados dinamicamente durante a execução do teste. Os resultados do teste são usados para aprender mais sobre o componente ou sistema e para criar testes para as áreas que podem precisar de mais testes.

Às vezes, o teste exploratório é realizado usando testes baseados em sessões para estruturar as atividades de teste. No teste baseado em sessão, o teste exploratório é conduzido dentro de uma janela de tempo definida, e o testador usa um termo de teste contendo objetivos de teste para orientar o

teste. O testador pode usar folhas de sessão de teste para documentar as etapas seguidas e as descobertas feitas.

O teste exploratório é mais útil quando há poucas ou inadequadas especificações ou pressão de tempo significativa nos testes. O teste exploratório também é útil para complementar outras técnicas de teste mais formais (MYERS, 2011).

3 Montagem do robô

O robô triângulo possui toda sua estrutura para impressão em 3D com o formato STL no repositório Github de (HUGGINS, 2020) na seção de hardware, onde todas as partes montáveis estão disponíveis.

3.1 Formato STL

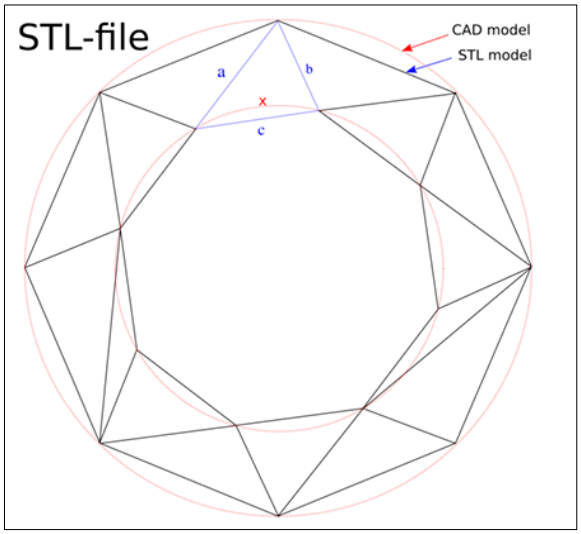

Figura 13. CAD representando arquivo STL.

A criação de um arquivo STL mostrado na Figura 13 consiste em converter sua casca externa em uma infinidade de triângulos para tornar o arquivo possível de ser impresso. A escolha dos triângulos se dá pelo fato de ser a figura geométrica mais próxima de um vetor. O triângulo possui intensidade, direção e sentido, propriedades fundamentais para a impressão 3D.

3.2 Peças em formato STL

A peça Placa Arduino mostrada na Figura 14 é onde fixa-se o Arduino uno. Seu tamanho é proporcional a placa com os furos para parafusar o *hardware* embarcado e leva aproximadamente 1h32min para ser impressa.

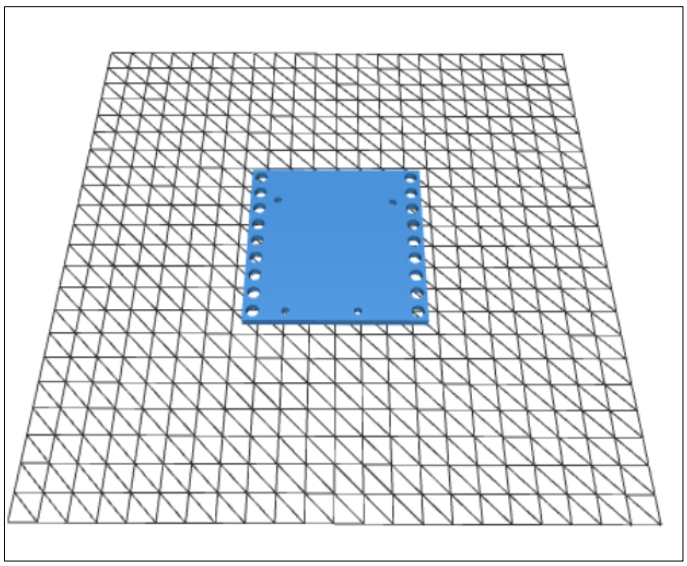

Figura 14. Esquemático da placa Arduino

A peça Base mostrada na Figura 15, é a parte onde temos toda a estrutura do robô triângulo delta. Possuindo três entradas para os alicerces da estrutura e também possíveis entradas para parafusar dispositivos móveis como smartphones e leva aproximadamente 6h25min para ser impressa.

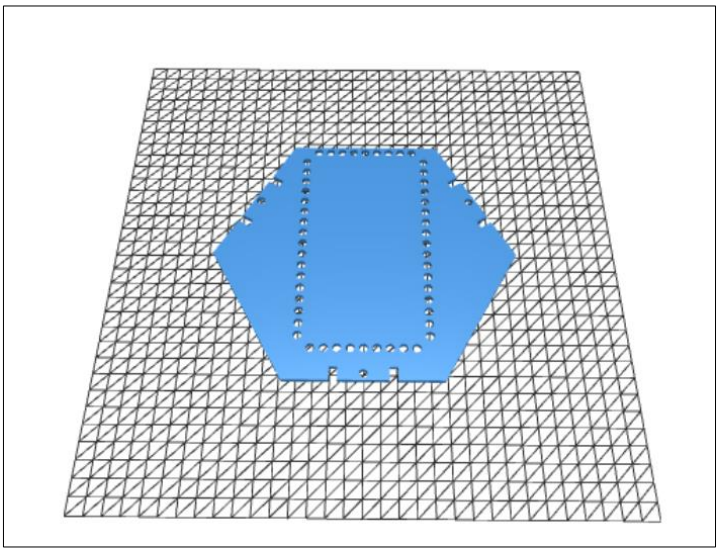

Figura 15. Esquemático da base

A peça Feixe de 3x1 mostrada na Figura 16, serve de junta para peças maiores, é ela que faz a junção das peças na base do robô, fixando os dispositivos móveis e leva aproximadamente 14min para ser impressa.

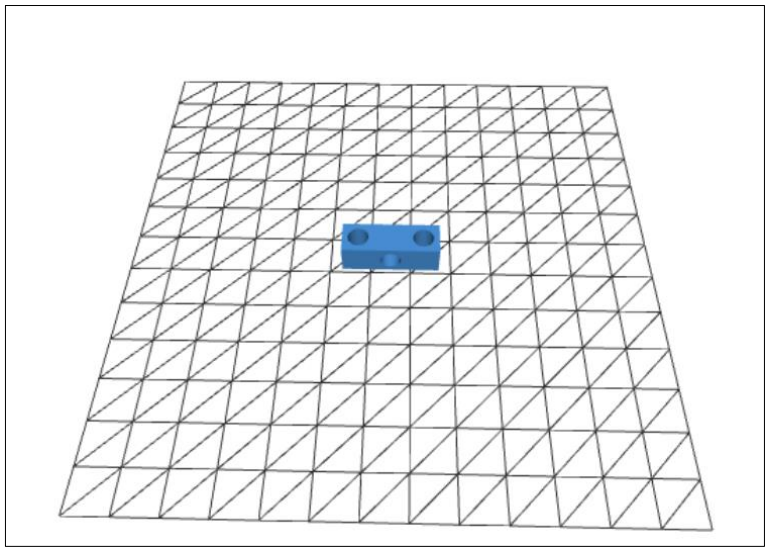

Figura 16. Esquemático do feixe de 3x1

A peça Feixe de 9x1 mostrada na Figura 17. É a plataforma para parafusar a estrutura base com outras estruturas e leva aproximadamente 26min para ser impressa.

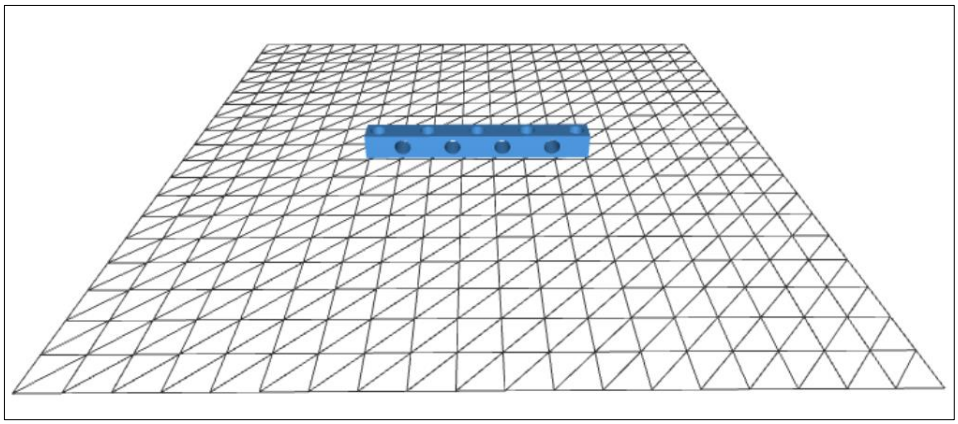

Figura 17. Esquemático do feixe de 9x1

A peça acopladora mostrada na Figura 18 interliga os imãs com os suportes de metal que simulam os braços robóticos tridimensionais e leva aproximadamente 14min para ser impressa.

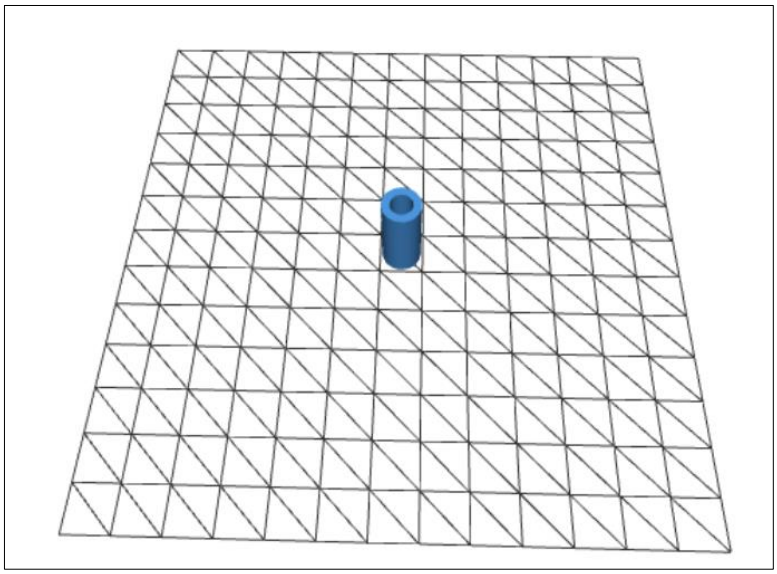

Figura 18. Esquemático da acopladora

A peça Lâmina mostrada na Figura 19 faz a interligação das junções e outras peças maiores e leva aproximadamente 21min para ser impressa.

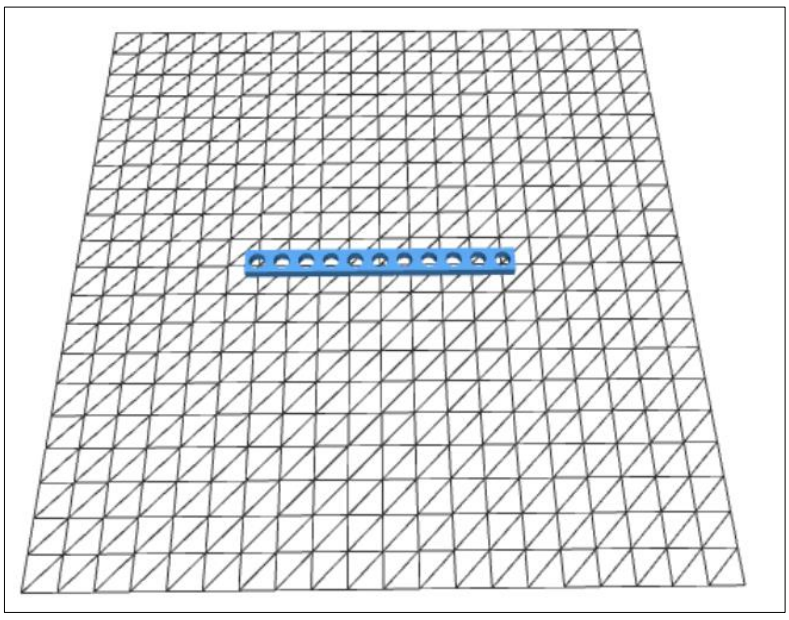

Figura 19. Esquemático da lâmina 11x1

A peça Servo mostrada na Figura 20 é onde acopla-se o motor responsável pela movimentação dos braços do robô e leva aproximadamente 26min para ser impressa.

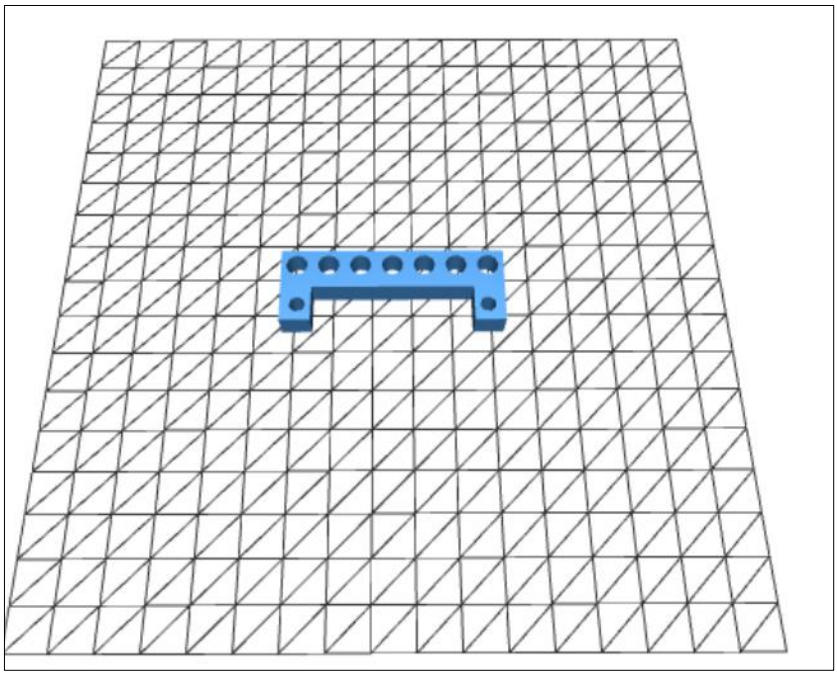

Figura 20. Esquemático da montagem servo

A peça Lado mostrada na Figura 21, sustenta a base inferior e a base superior do robô, no total são 3 lados e leva aproximadamente 2h34min para ser impressa.

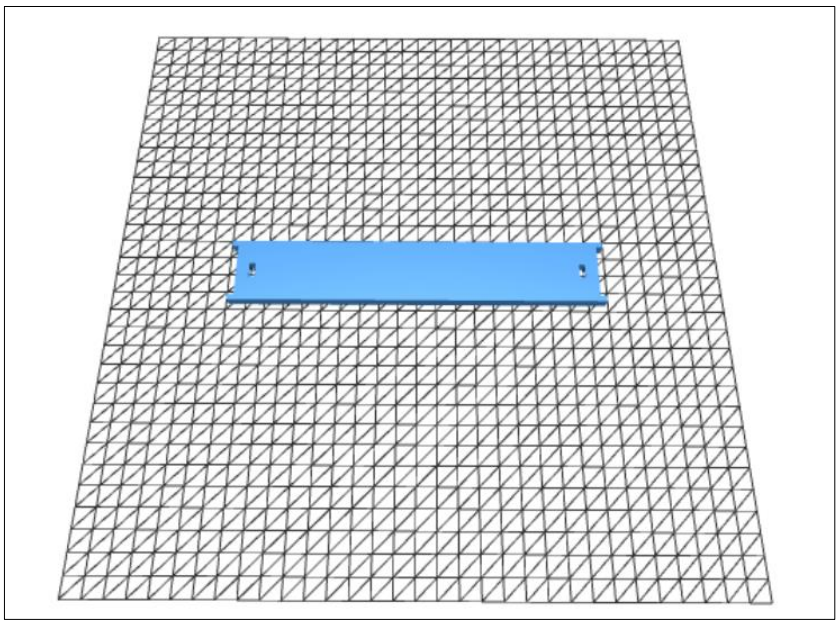

Figura 21. Esquemático do lado

A peça Porta Caneta mostrada na Figura 22 é onde fixa-se a caneta *stylus*, sua característica é ter a ponta que simula o toque de um dedo humano e leva aproximadamente 21min para ser impressa.

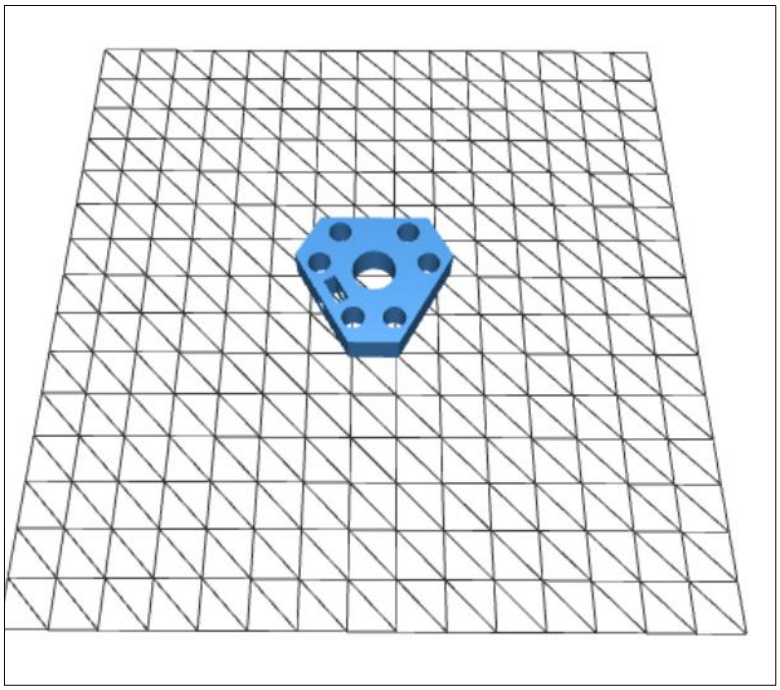

Figura 22. Esquemático do porta-caneta

A peça Topo mostrada na Figura 23, é a base superior do robô, onde fica o sistema embarcado e leva aproximadamente 6h18min para ser impressa.

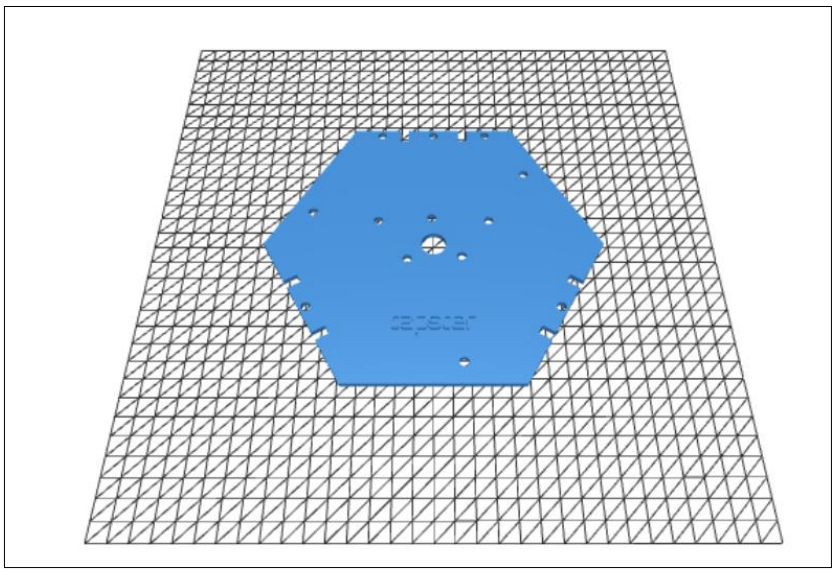

Figura 23. Esquemático do topo

A peça Braço mostrada na Figura 24 é onde fixa-se o motor para que haja a movimentação do braço robótico e leva aproximadamente 29min para ser impressa.

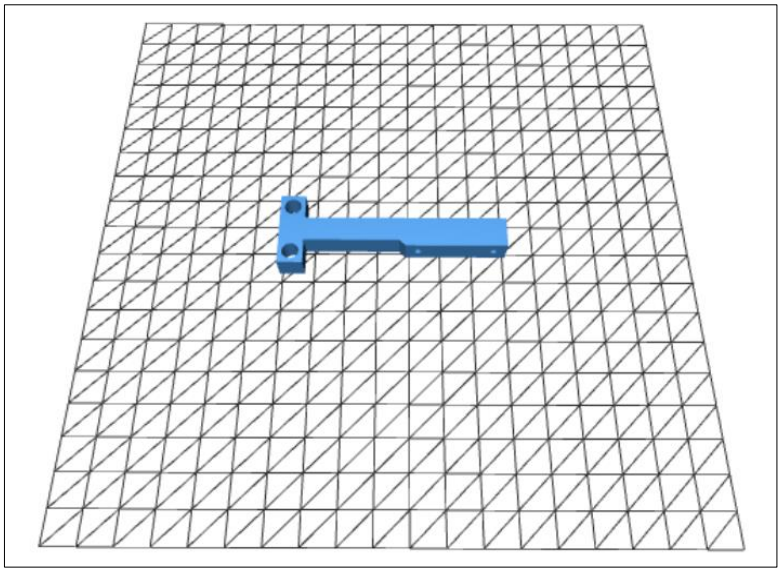

Figura 24. Esquemático do braço

3.3 Peças Impressas

Toda as peças foram impressas utilizando a impressora 3D Creality Ender 3 Pro, tendo como material base o filamento ABS de cor cinza, bastante utilizado por conta de suas boas propriedades de resistência.

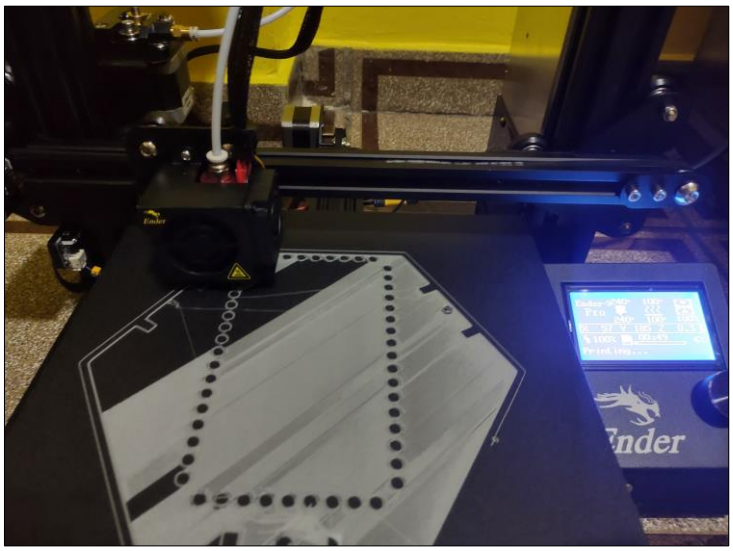

Figura 25. Impressão da base

A base do *Tapster* sendo impressa é mostrada na Figura 25, a impressão é feita camada por camada, por meio de processo aditivo, até formar uma estrutura rígida e sólida.

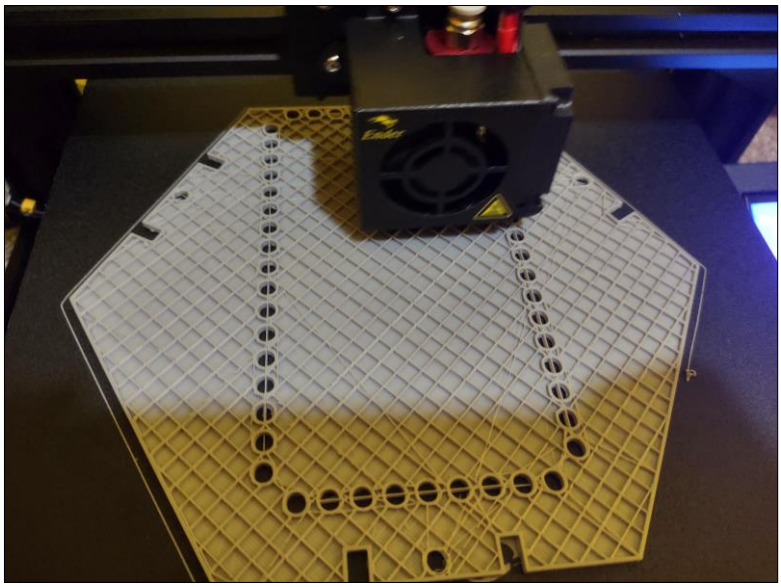

Figura 26. Impressão de camadas da base

A solidez das peças é construída a partir do entrelaçamento de ponto a ponto das peças conforme Figura 26.

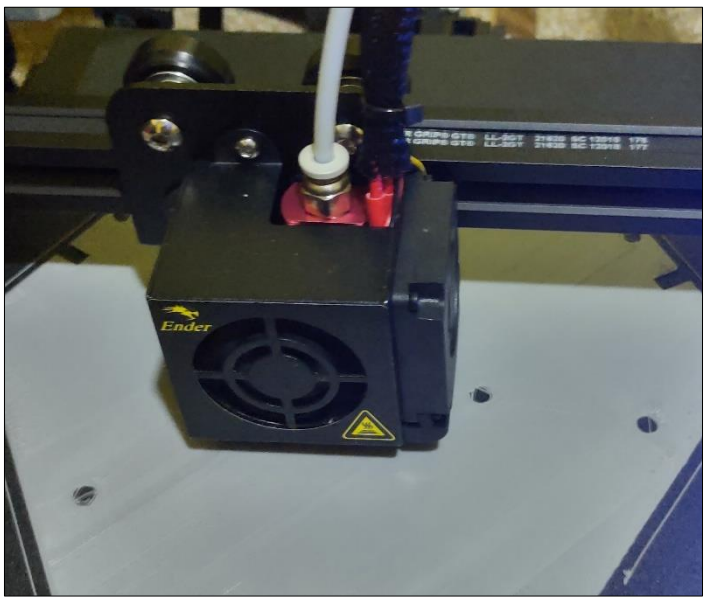

Figura 27. Impressão do topo

O topo mostrado na Figura 27 possui as mesmas dimensões que a base, porém com menos nível de detalhe.

Figura 28. Junção da placa de Arduino e feixe 9x1

Analisando a Figura 28 pode-se observar a junção da peça de placa de Arduino com o feixe 9x1 com parafuso 9mm.

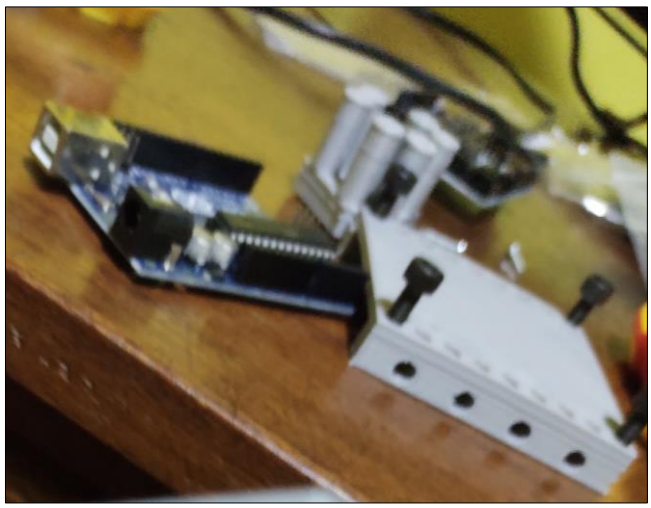

Figura 29. Peça placa Arduino

Conforme a Figura 29 o Arduino uno foi parafusado na placa de Arduino.

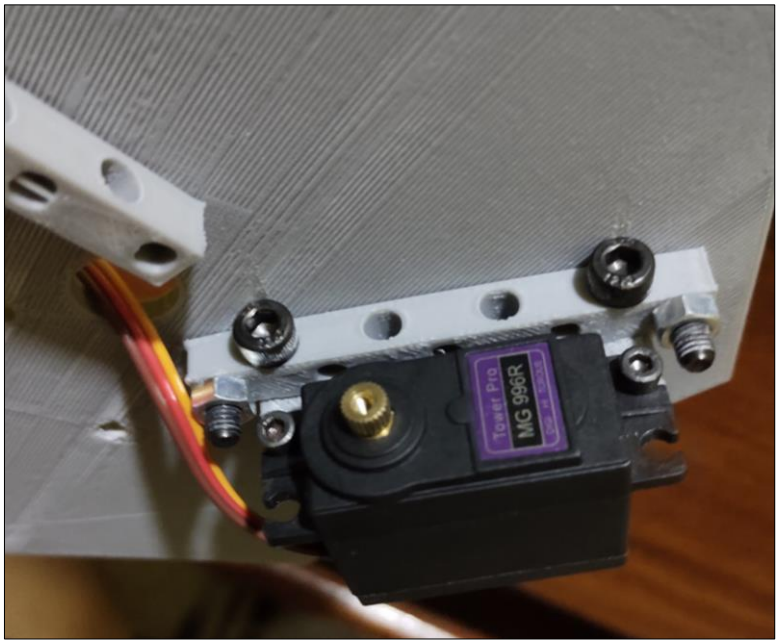

Figura 30. Acoplagem dos servomotores na peça topo

O servo motor MG996R foi acoplado a peça Servo e ao topo conforme Figura 30.

Figura 31. Junção da acopladora e braço

Foram colocados os ímãs Neodímio 5 x 9 mm na acopladora e no braço conforme Figuras 31 e 32.

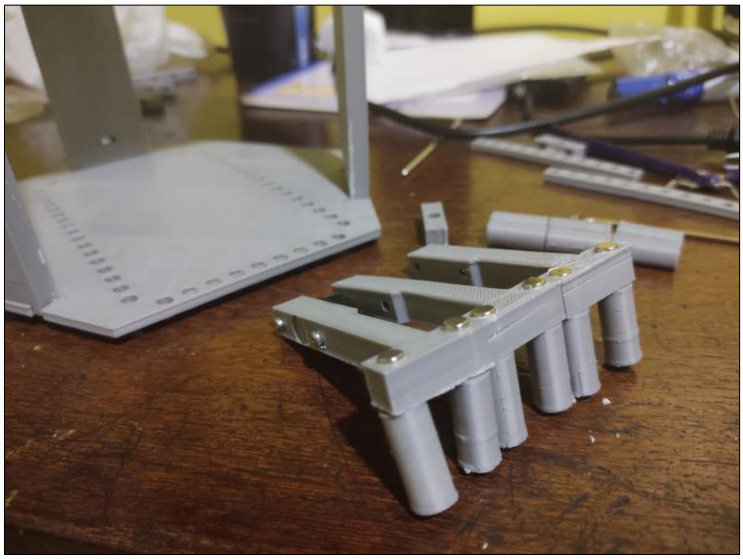

Figura 32. Acopladores e braços

Os três lados foram colocados junto com a base e o topo, sendo fixados com os parafusos conforme Figura 33.

Figura 33. Os três lados

Na Figura 34 pode-se observar a caneta *stylus* sendo fixada na porta caneta já integrada aos ímãs Neodímio 5 x 9.

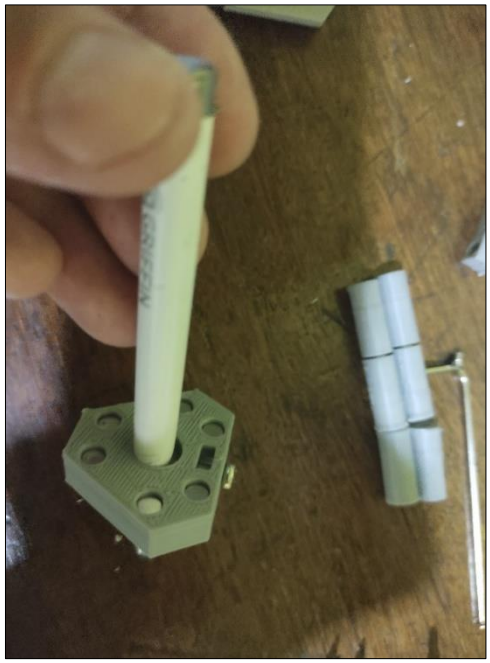

Figura 34. Porta caneta

No total há 6 braços robóticos, contendo duas acopladoras em cada um e um imã em cada extremidade conforme Figura 35.

Figura 35. Acopladores dos braços

3.4 Configuração do Tapster

O *Tapster* padrão atualmente usa 3 servos. Esses servos são conectados aos pinos 9, 10 e 11 na placa-filha mostrados na Figura 36. Se, por algum motivo, eles forem desconectados ou conectados incorretamente, o *Tapster* não funcionará corretamente.

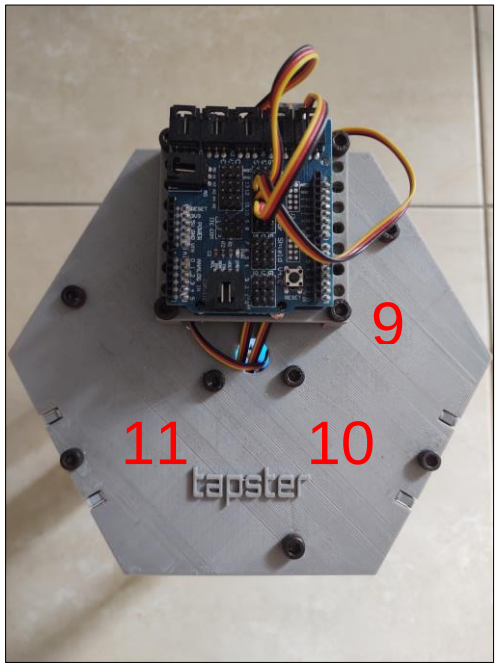

Figura 36. Correspondências dos servomotores na pinagem

Os servos *Hi-Tec* 311, utilizados no *design* atual do *Tapster*, requerem uma calibração prévia para garantir o mais alto nível de precisão possível. Para isso, inicialmente deve-se remover os braços para começar a calibrar os servos. O objetivo é descobrir em quais valores as montagens do servo são completamente horizontais e completamente verticais, e então inserir esses valores no arquivo de configuração para serem usados em uma função de mapeamento.

3.4.1 Arquivo de configuração

O arquivo de configuração do *Tapster* é chamado config.js e está localizado no diretório *software*. As variáveis e, f, re e rf mostradas na Figura 37 são os comprimentos dos componentes do *Tapster*. Os valores são necessários para o cálculo dos ângulos de cinemática inversa.

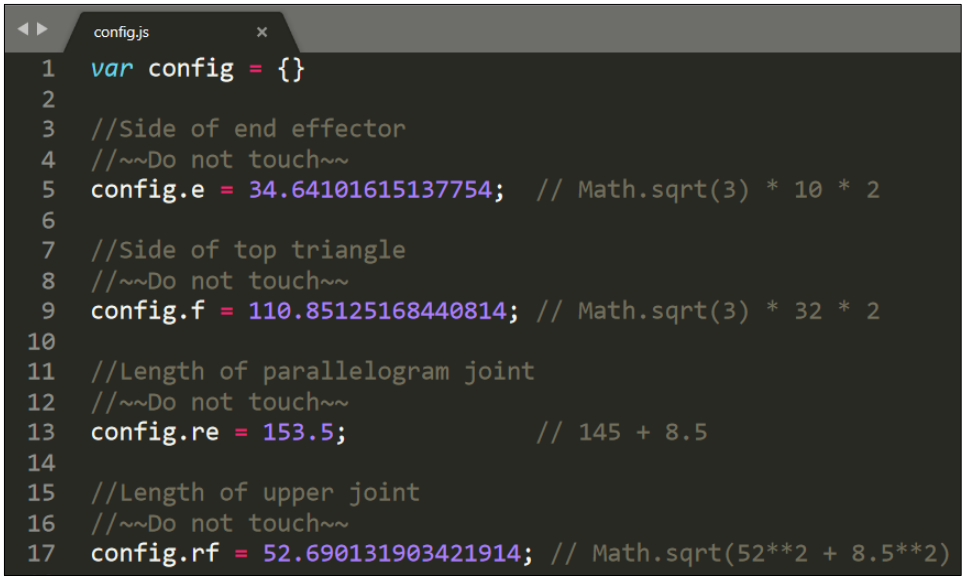

Figura 37. Arquivo de configuração config.js

Os valores *in_min* e *in_max* para todos os três servos são utilizados para mapear pontos.

O *baseWidth* e *baseHeight* são as dimensões da área de desenho da placa de base. O *Tapster* só pode alcançar parte da placa de base de forma confiável devido à forma como foi projetado. Esses valores são usados para mapear pontos e não devem ser alterados.

Os valores *out_min* e *out_max*, representados na Figura 38, são utilizados para mapear pontos; no entanto, eles mudam de servo para servo e, portanto, devem ser alterados após a calibração de cada servo onde o valor padrão é 0, 90.

```
//Input ranges for servos
//~~Do not touch~~
config.servo1 = {in_min: 0, in_max: 90};
config.servo2 = {in_min: 0, in_max: 90};
config.servo3 = {in_min: 0, in_max: 90};

//Default output ranges for servos
//CHANGE THESE
config.servo1.out_min = 12;
config.servo1.out_max = 93;
config.servo2.out_min = 8;
config.servo2.out_max = 90;
config.servo3.out_min = 14;
config.servo3.out_max = 96;

//Dimensions of the base plate
config.baseHeight = 95;
config.baseWidth = 80;

```

Figura 38. Valores máximo e mínimo presentes no arquivo config.js

O *penHeight* é o nível z padrão para o qual o robô deve se mover. Um valor mais alto suspende a caneta e um valor mais baixo rebaixa a mesma, onde o valor padrão é -140.

O *delay* é a quantidade de atraso a ser usada nos comandos SVGReader. Cada comando demora milissegundos ou mais para ser concluído. Um valor mais alto de atraso resultará em desenhos mais precisos e suaves, ao custo de torná-los mais lentos; um valor mais baixo resultará em um desenho menos preciso e menos suave, mas será muito mais rápido onde o valor padrão é 150.

O *defaultEaseType* é o algoritmo de atenuação padrão a ser usado quando nenhum é especificado. Para obter a lista de algoritmos de atenuação, é necessário verificar o arquivo motion.js. Um valor "nenhum" significa que, a menos que seja especificado, nenhum algoritmo de atenuação deve ser usado, por padrão é linear.

```
//Default Z-Level of the pen
config.penHeight = -140;

//Default drawing height of the pen
config.drawHeight = -152.75;

//Delay for commands in SVGReader
//Note that some commands will take longer than this
//Default value is 150
config.delay = 200;

//The default easing type to be used
//When no easing is specified, this is the type that will be used
//"none" means that if no easing is specified, do not ease
//For a list of possible easing types, look in motion.js
config.defaultEaseType = "linear";

module.exports = config;
```

Figura 39. Parâmetros de altura e largura no arquivo config.js

É possível especificar um arquivo de configuração diferente do config.js padrão. Ao iniciar o bot.js, deve-se inserir o caminho para o arquivo de configuração customizado como um parâmetro de *string*. Todos os valores padrões foram estipulados pelo próprio Jason, conforme seu repositório no github, (HUGGINS, 2020).

3.4.2 Comunicação do Hardware Embarcado

O StandardFirmata é o elemento responsável pela comunicação do Arduino e os servomotores com o *javascript*. O código fonte disponibilizado do robô *Tapster* possui um arquivo package.json, conforme Figura 40, que descreve os módulos e pacotes necessários para utilização do robô.

```
package.json
{
   "name": "tapster",
   "description": "Mobile automation robot",
   "version": "2.0.0",
   "homepage": "",
   "author": [
      {
         "name": "Jason Huggins",
         "email": "jrhuggins@gmail.com"
      }
   ],
   "repository": {
      "type": "git",
      "url": "https://github.com/tapsterbot/tapsterbot.git"
   },
   "engines": {
      "node": "*"
   },
   "dependencies": {
      "body-parser": "*",
      "express": "*",
      "johnny-five": "^2.0.0",
      "svg-path-parser": "*",
      "xml2js": "*"
   }
}
```

Figura 40. Arquivo de pacotes e módulos package.json

Portanto todo o fluxo de funcionamento do robô acontece pelo *scratch* (*StandardFirmata*). Na Figura 41 pode-se perceber o fluxo de comunicação do *scratch* com o JavaScript e o Node.

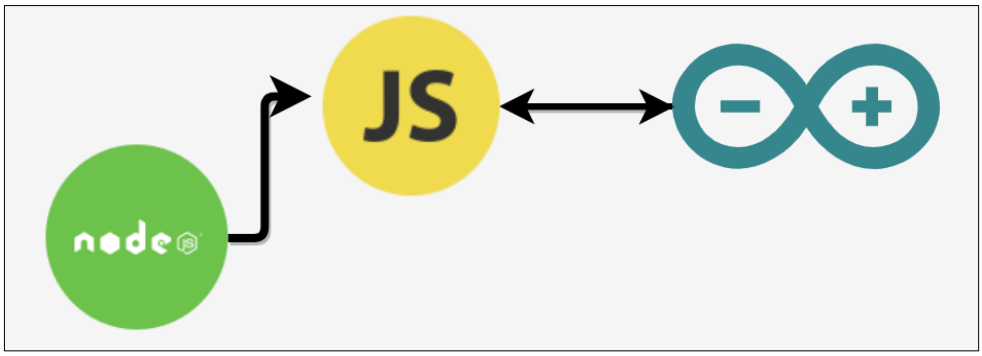

Figura 41. Comunicação Arduino, JavaScript e Node

Para visualizar a comunicação, alguns passos devem ser seguidos:

1. Abra o esboço *StandardFirmata* utilizando o Arduino IDE (Arquivo> Exemplos> Firmata), conforme Figura 42.
2. Embarque o código para o dispositivo Arduino.
3. Após embarcado, a comunicação entre o Javascript e o Arduino é estabelecida.

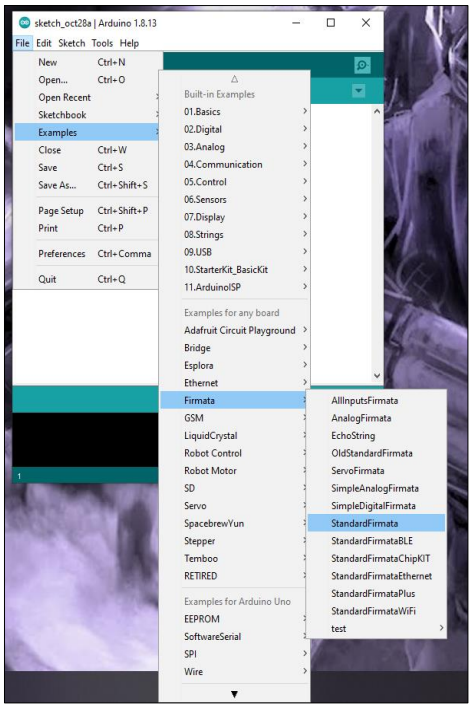

Figura 42. Scatch padrão do StandardFirmata

4 Valor limite na aplicação Calculadora

A aplicação calculadora é utilizada para fazer desde cálculos básicos como adição, subtração, multiplicação como também fazer operações científicas, como funções exponenciais e trigonométricas. Os testes dessa aplicação com um robô triângulo delta que tem como função principal simular o toque do dedo humano teve um fator importante para extração dos valores limites de cada botão.

4.1 Codificação

A codificação foi realizada utilizando o código bot.js, que tem funções especificas para o robô como fazer desenhos como quadrado, círculo e inserir uma coordenada x, y e z para deslocar o braço robótico conforme os valores limites configurados. No entanto não existia uma função que fizesse toda o caminho de cliques ao longo do tempo, ou em outras palavras não existia um mapeamento de cliques. Então para fazermos esse mapeamento escolhemos um modelo de dispositivo que foi o Motorola Moto G8 Plus, fixamos na base do robô conforme Figura 43. Tendo assim uma fixação padrão para este dispositivo, caso eventualmente precisássemos retira-lo e colocá-lo novamente e verificar suas coordenadas mapeadas, elas não se perderiam, pois temos o ponto de referência fixo.

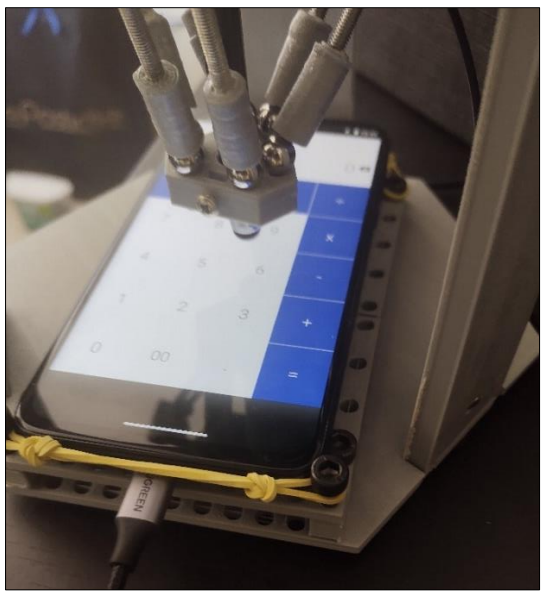

Figura 43. Fixação do dispositivo móvel com o robô

Do ponto de vista do robô temos as coordenadas x e y conforme um plano cartesiano, conforme Figura 44, onde o ponto inicial, são as coordenadas x = 0 e y = 0, diferenciando apenas o z que tem por profundidade padrão z = -140.

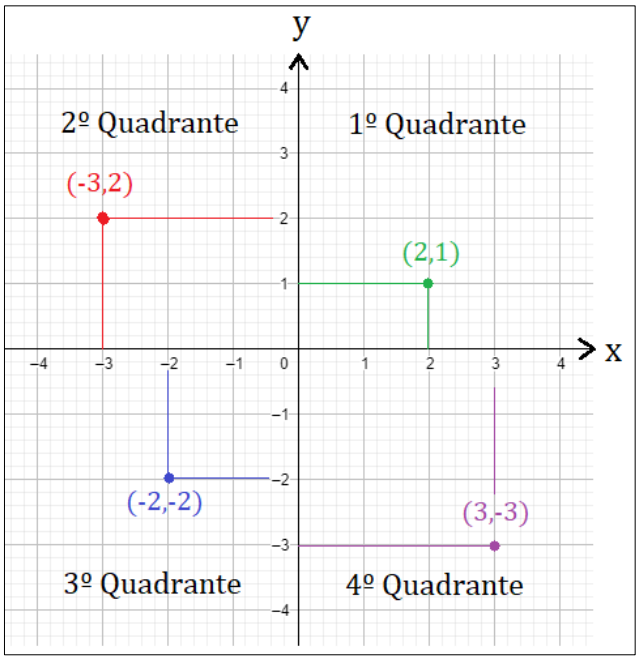

Figura 44. Representação de quadrantes no plano cartesiano

Para definir o ponto inicial do nosso mapeamento foi necessário utilizar o código bot.js e assim testar de forma empírica as coordenadas correspondentes, tanto de abcissas, coordenadas e profundidade. Sendo assim a variação de x foi definida de x = -30 até x= 15, y =15 até y= -30 e z possuindo uma profundidade de z = -130 (sem toque na tela) e z =-145 (com toque na tela). Após o mapeamento dos valores foi criado o método *runftw*, mostrado na Figura 45, responsável pela movimentação do braço do robô a uma coordenada estipulada e realizar a ação de toque na tela.

```
289
290   runftw = function(){
291     for (var y = 15; y > -31; y--){
292         for (var x = -30; x < 16; x++){
293             doSetTimeout(x,y,-130,1000);
294             doSetTimeout(x,y,-145,300);
295             doSetTimeout(x,y,-130,300);
296             doSetTimeout2(x,y,-130,5000);
297             doSetTimeout3(x,y,-130,1000);
298         }
299     }
300     return;
301   }
```

Figura 45. Método de rotina para extração dos valores x e y.

Para cada interação do laço de repetição temos instruções especificas para o braço robótico, Dump da tela da aplicação e armazenamento dos valores em formato XML.

O método doSetTimeout, mostrado na Figura 46, tem como parâmetros as coordenadas informadas, o tempo de *delay* e uma flag de verificação. Caso essa flag de verificação seja falsa ela recebe o valor padrão, chamado de defaultEaseType. O método setTimeout move o braço do robô até a coordenada informada pelo usuário, definindo um tempo para a execução dessa instrução.

```
329   doSetTimeout = function(x, y, z, timeDelay, easing) {
330     if (!easing)
331       easing = defaultEaseType;
332
333     setTimeout(function() {
334       go(x, y, z, easing);
335     }, timer);
336     timer = timer + timeDelay;
337   };
```

Figura 46. Método de movimento padrão do robô

O método doSetimeout2, mostrado na Figura 47, é responsável por extrair o dump da tela e renomear o arquivo. Inicialmente são exibidas as coordenadas x, y no terminal pelo método console.log. O shell.exec é um comando via shell para fazer rotinas de chamadas do sistema operacional, dessa forma utilizamos o adb para fazer o dump pelo Uiautomator e salvar essa extração no diretório Dumps. Como o arquivo possui o mesmo nome a cada dump, precisamos sempre renomear o arquivo para as respectivas coordenadas, por isso abrimos o arquivo, renomeamos o arquivo de dump

original de window_dump para window_dump_coord_x_y e salvamos, todo esse processo leva em torno de 5000 ms.

```
doSetTimeout2 = function(x, y, z, timeDelay, easing) {
  if (!easing)
    easing = defaultEaseType;

  setTimeout(function() {
    go(x, y, z, easing);
    console.log("coord:",x,y);
    shell.exec('adb shell uiautomator dump');
    shell.exec('adb pull /sdcard/window_dump.xml  "/home/araunhoz/Documents/Robot/tapsterrobot/software/src/Dumps"');

var fs = require('fs');
var caminho = '/home/araunhoz/Documents/Robot/tapsterrobot/software/src/Dumps/window_dump';
var caminho_novo = caminho.concat('_coord_',x.toString());
var caminho_novo_novo = caminho_novo.concat('_',y.toString());

console.log(caminho_novo_novo);

fs.rename('/home/araunhoz/Documents/Robot/tapsterrobot/software/src/Dumps/window_dump.xml',caminho_novo_novo, function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});

  }, timer);
  timer = timer + timeDelay;
};
```

Figura 47. Método de extração de dump da tela

O método Setimeout3, mostrado na Figura 48, faz a ação de clicar no botão *clear* da aplicação. Após algumas tentativas foi inferido que enviar o comando via adb seria menos custoso do que movimentar o braço do robô para realizar essa ação, devido a instrução aumentar drasticamente o tempo de extração dos dumps. Ao final o arquivo de dump é fechado e salvo.

```
doSetTimeout3 = function(x, y, z, timeDelay, easing) {
  if (!easing)
    easing = defaultEaseType;

  setTimeout(function() {
    go(x, y, z, easing);
    shell.exec('adb shell input tap 0 581');
  }, timer);
  timer = timer + timeDelay;
};


var fs = require("fs");

resetTimer = function() {
  timer = 0;
}
```

Figura 48. Método para disparar ação de toque na tela

4.2 Extração de Dados

A extração dos dados foi realizada da esquerda para direta começando da coordenadas x, y do botão 7 finalizando no botão 9, repetindo novamente esse laço de coordenada do botão 4 até o botão 6 e por fim coordenada do botão 1 até o 3 finalizando todas a extração dessas coordenadas, conforme

Figura 49. Ou seja, fixando o y e variando o x, tudo isso foi feito com o código Read.js. O mapeamento foi feito apenas nas teclas [7,8,9], [4,5,6] e [1,2 ,3] conforme definido no escopo de trabalho.

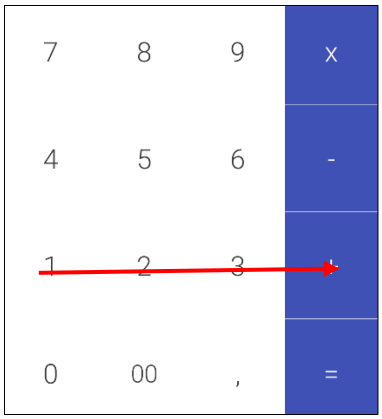

Figura 49. Fluxo da extração dos dados

Portanto a cada dump extraído era gerado um arquivo XML no diretório Dumps com as respectivas coordenadas, conforme Figura 50.

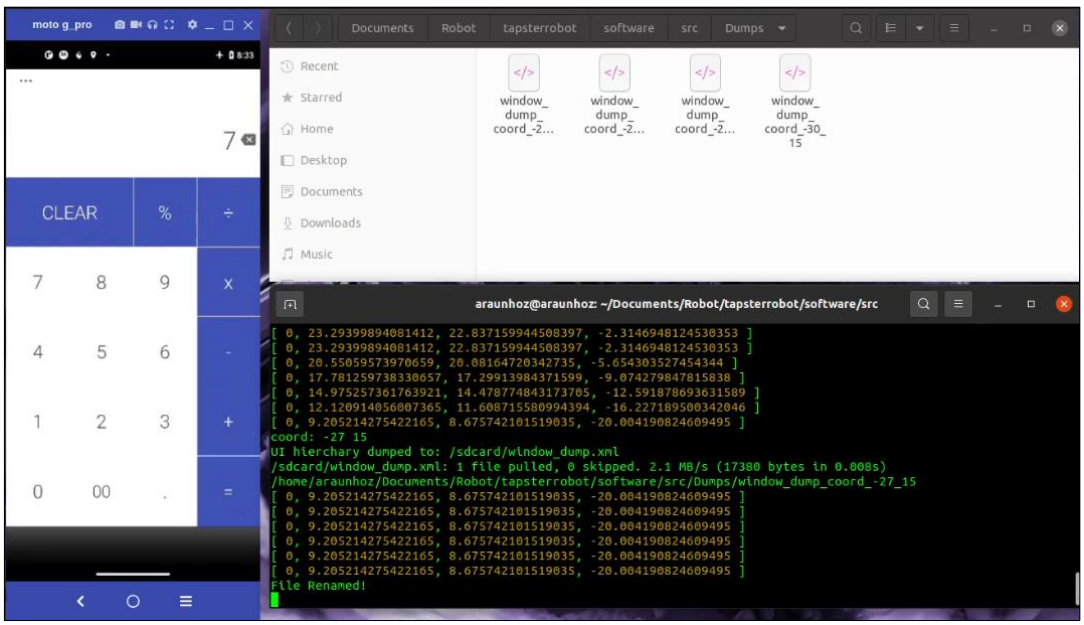

Figura 50. Extração das coordenadas e geração de arquivo xml

No total foram extraídos 2.116 arquivos, cada arquivo tendo 17,4 kB, totalizando 36,8 MB, como mostrado na Figura 51.

Figura 51. Arquivos XML gerados após o *dump* da tela

O XML extraído possui informações de todos os componentes da tela, mostrado na Figura 52, porém tais informações é extraída em um bloco único de informação da tela de aplicação, o que dificulta a análise.

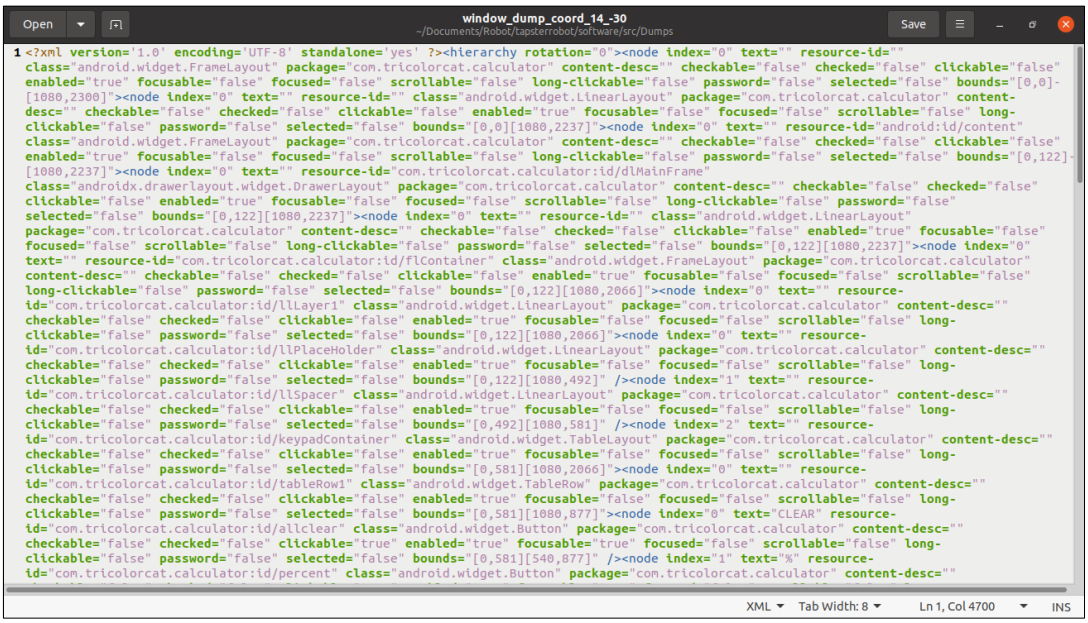

Figura 52. Arquivo XML gerado pelos *dumps*

Para cada movimento do braço robótico com método doSetTimeout levou em torno de 1,6 s. E o tempo de armazenamento desses arquivos com os métodos doSetTimeout2 e doSetTimeout3 levou 6,0 s, logo temos que o total de tempo de instrução de movimentação do braço robótico e o salvamento destes arquivos é de 7,6 s. Então podemos estipular o tempo total com os arquivos gerados, sendo assim 7,6 s * 2.110 arquivos nos resulta um tempo de 16.0816 s. Convertendo para horas temos aproximadamente 4 h. Então para cada tecla temos que o método doSetTimeout, ou melhor a movimentação do braço até o ponto desejado tem uma porcentagem de 21,07% e a geração do arquivo é a soma dos métodos doSetTimeout2 e doSetTimeout3 nos resultando em 78,93%, conforme tabela 1.

Tabela 1. *Tempo de execução por instrução.*

Método	Tempo [s]	Tecla [%]
doSetTimeout	1,6	21,07
doSetTimeout2	5,0	65,78
doSeTimeout3	1,0	13,15
Total	7,6	100

Toda a extração dos arquivos XML demorou em torno de 4 horas, 28 minutos e 7 segundos. Foi utilizado uma aplicação de cronômetro para averiguação do tempo de execução, conforme Figura 53.

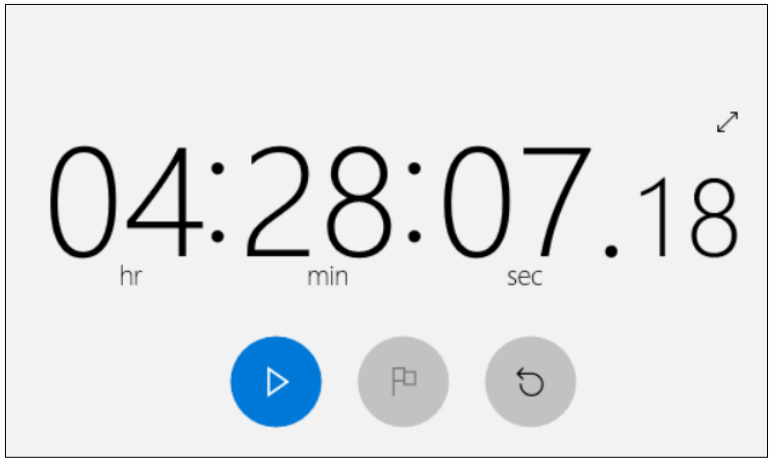

Figura 53. Tempo de extração dos dados

Uma maneira de visualizar um XML é transformá-lo em uma árvore de hierarquia, ficando mais legível como apresentado na Figura 54.

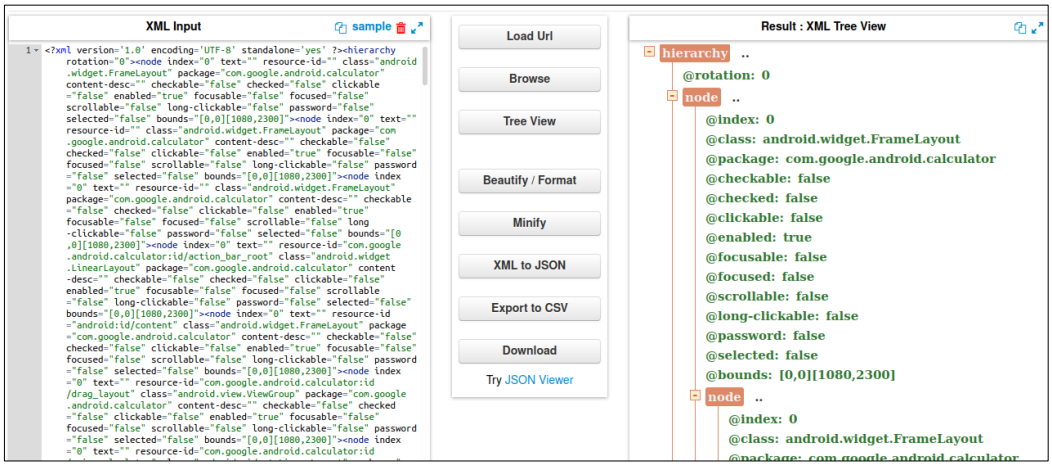

Figura 54. Árvore de hierarquia XML

4.3 Tratamento dos Dados

Após a extração dos dados era necessário validar as coordenadas e os elementos presentes na aplicação. Para isso foi necessário associar a coordenada que estávamos requisitando ao robô com o valor que aparecia na tela de aplicação, conforme Figura 55.

Figura 55. Componente que exibe o valor pressionado

Foi então desenvolvido um método para realizar o mapeamento e classificação dos dados, conhecido como parser, onde foram filtrados em cada nó, //node, os *id's* e *text's* com o código Read.js, mostrado na Figura 56. Identificando assim se o valor clicado correspondia a posição enviada ao robô.

```
Read.js
1  var shell = require('shelljs');
2  var select = require('xpath.js')
3  , dom = require('xmldom').DOMParser
4  var fs = require("fs");
5  var z = -145;
6  for (var y = 15; y > -31; y--) {
7      for (var x = -30; x < 16; x++) {
8          var caminho = '/home/araunhoz/Documents/Robot/tapsterrobot/software/src/Dumps/window_dump';
9          var caminho_novo = caminho.concat('_coord_',x.toString());
10         var caminho_novo_novo = caminho_novo.concat('_',y.toString());
11         console.log(caminho_novo_novo);
12         var data= fs.readFileSync(caminho_novo_novo, 'utf8');
13
14             var doc = new dom().parseFromString(data)
15             var nodes = select(doc, "//node")
16             function print_a(node) {
17                 return {
18                     "id":node.getAttribute('resource-id'),
19                     "text":node.getAttribute('text'),
20                 }
21             }
22
23             const orderIds = nodes.map(print_a);
24             const text = nodes.map(print_a);
25
26             var choice  = text[40].text
27             var tecla;
```

Figura 56. Método para mapeamento e validação de valor pressionado

Ao final do arquivo Read.js, foi criada uma estrutura de decisão, *switch case* para filtrar os valores encontrados ou não encontrados, conforme Figura 57.

Read.js

```
case '4':
    var acerto = 'S';
    var tecla = '4';
    shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);
    break
case '5':
    var acerto = 'S';
    var tecla = '5';
    shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);
    break
case '6':
    var acerto = 'S';
    var tecla = '6';
    shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);
    break
case '7':
    var acerto = 'S';
    tecla ='7';
    shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);
    break
case '8':
    var acerto = 'S';
    var tecla = '8';
    shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);
    break
case '9':
    var acerto = 'S';
    var tecla = '9';
    shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);
    break
default:
            var acerto = 'F'
            var tecla = 'x';
            shell.exec('python3 SaveDump.py ' +tecla + ' ' + x +' '+ y +' ' + z +' ' + acerto);

```

Figura 57. Estrutura de decisão para filtragem de dados

A cada caso de decisão era chamado um programa Python SaveDump.py, conforme Figura 58, que era responsável por salvar os valores de modelo do dispositivo num arquivo de formato TXT, tecla ao qual foi mapeado, valor de x, valor de y, valor de z e acerto. No acerto tínhamos F para falha e S para sucesso.

SaveDump.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
tecla = sys.argv[1]
x = sys.argv[2]
y = sys.argv[3]
z = sys.argv[4]
acerto = sys.argv[5]
arq = open('/home/araunhoz/Documents/Robot/tapsterrobot/software/src/Data/BD.txt', 'a')
texto = "Motorola_g6;"+ tecla +";"+ x +";"+ y +";"+ z +";"+ acerto
arq.write(texto)
arq.write('\n')
arq.close()
```

Figura 58. Arquivo de geração da base de dados

Ao final é gerado um arquivo com formato TXT com 2116 linhas, mostrado na Figura 59, com uma classificação do que foi clicado ou não para um determinado botão para cada conjunto de coordenadas x, y e z.

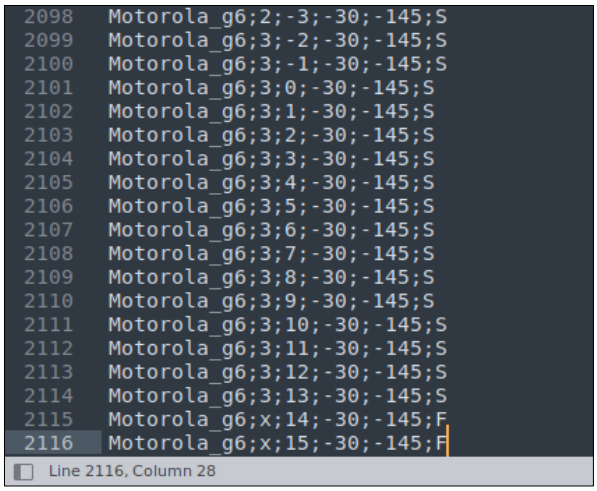

Figura 59. Trecho da base de dados

4.3 Limiar dos valores dos botões

De acordo com os resultados da base de dados alguns valores na variável tecla foram atribuídos como x. O valor x significa que o botão que foi clicado não pertencia ao esperado, sendo considerado um efeito de borda ou valor limite, exemplificado na Figura 60. Por exemplo, se o testador pretende apertar a tecla 6, porém o retorno do valor é o –, tal informação irá falhar seu caso de teste numa automatização.

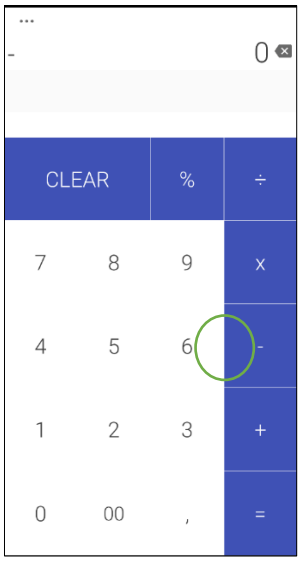

Figura 60. Exemplo do problema de borda

Portanto o mapeamento de onde os botões estão situados nos dispositivos é de suma importância para um teste automatizada, onde se quer utilizar a robotização para simular do toque humano. Os valores de máximos e mínimos reduzem a inconsistência de valores que podem acarretar em falhas que não são do cenário de teste. A definição de valores limite é similar a uma faixa para definir o conjunto de valores utilizados como parâmetros de entrada em um caso de teste, apresentado na Figura 61.

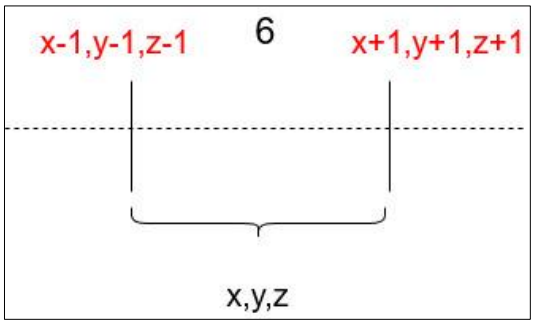

Figura 61. Faixa das coordenadas correspondentes ao valor

O impasse de valores máximos e mínimos dos botões é uma forma de classificar os conjuntos possíveis para a instrução informada, reduzindo o percentual de falhas ao enviar as coordenadas ao robô. Aumentando na qualidade dos parâmetros de entrada do caso de teste automatizado. Na Tabela 2 é mostrada parcialmente a base de dados.

Tabela 2. *Trecho da tabela de base de dados classificada*

Modelo	Tecla	x	y	z	Acerto
Motorola_g6	1	-30	-30	-145	S
Motorola_g6	1	-30	-29	-145	S
Motorola_g6	1	-30	-28	-145	S
Motorola_g6	1	-30	-27	-145	S
Motorola_g6	1	-30	-26	-145	S
Motorola_g6	1	-30	-25	-145	S
Motorola_g6	1	-30	-24	-145	S

5 Considerações finais e trabalhos futuros

5.1 Considerações finais

O trabalho proposto teve como finalidade apresentar o problema de valor limite numa aplicação simples de calculadora utilizando um robô triângulo delta, chamado de *Tapster,* que tem como propósito simular o toque humano. Podemos perceber pela extração dos dados que a complexidade de examinar uma aplicação por mais simples que seja é proporcional ao quanto a aplicação é rica em detalhes como: tamanho de botões, funcionalidades, o tempo de resposta e etc. Retiramos alguns fatores como a questão de tamanho de tela, tipo de borda (alguns dispositivos possuem borda infinita), tempo de resposta de hardware com a aplicação, o tipo de caneta que simula o toque, a limitação do robô em si, todas essas variáveis foram desprezadas para conseguirmos uma extração e termos uma base de dados sucinta. Concluindo que aplicações que tenham uma interface gráfica de característica simples podem ser utilizadas para extração de seus valores limites usados como entradas em testes automatizados como sanidade ou regressão, os valores são classificados em máximos e mínimos para cada botão.

5.2 Trabalhos futuros

- Utilização do *Raspberry Pi* com câmera acoplada, visando a captura de mais elementos e aumento de acurácia do robô;
- Realizar experimentos com mais modelos de dispositivos e outras aplicações para expandir a base de dados;
- Treinamento da base de dados, utilizando aprendizagem de máquina ou técnicas de predição a partir de imagem para ampliar a extração de dados.
- Expandir as ações gestuais e de interações da caneta *stylus* utilizando o robô triângulo delta, como por exemplo o uso de duas canetas para ser feito o gesto de pinça, muito utilizado nos dispositivos Android;
- Realizar teste automatizado de sanidade e de regressão com a base de dados dos valores limites como parâmetro de entrada e expandido o trabalho para o uso em problema como os valores limites por *pixel.*

Referências Bibliográficas

ARRAZATE, R. T. (2017). *Development of a URDF file for simulation and programming of a delta robot using ROS.* Santiago de Querétaro: -.

BEIZER, B. (1990). *Software Testing Techniques.* Van Nostrand Reinhold: -.

BONEV, I. (06 de 05 de 2001). *Parallemic*. Acesso em 01 de 11 de 2021, disponível em Parallemic: https://www.parallemic.org/Reviews/Review002p.html

CHEN, G., & KOTZ, D. (2000). *A Survey of Context-Aware Mobile Computing Research.* v. 3755: Technical Report TR2000-381.

CRAIG, J. J. (2005). *Introduction to Robotics: Mechanics and Control.* (3ª ed.). New Jersey: Pearson Prentice Hall.

FINLEY, K. (01 de 08 de 2013). *Robot With Long Finger Wants to Touch Your iPhone Apps*. Acesso em 27 de 10 de 2021, disponível em WIRED: https://www.wired.com/2013/08/tapster/

GHARAHSOFLOO, A., & RAHMANI, A. (- de Agosto de 2015). An Efficient Algorithm for Workspace Generation of Delta Robot. *International Journal of Robotics, 5*, pp. 48-53.

GONÇALVES, P. C. (2007). *Protótipo de um robô móvel de baixo custo para uso educacional.* Maringá: Universidade Federal de Maringá.

GOOGLE. (27 de 10 de 2021). *developers*. Acesso em 05 de 11 de 2021, disponível em DOCUMENTATION: https://developer.android.com/training/testing/ui-automator

GOOGLE. (09 de 07 de 2021). *developers*. Acesso em 01 de 11 de 2021, disponível em ANDROID STUDIO: https://developer.android.com/studio/command-line/adb

HADIAN, H., & FATTAH, A. (2008). *Genetic algorithms for workspace optimization of planar medical parallel robot.* -: -.

HUGGINS, J. (18 de 06 de 2020). *tapster/tapster*. Acesso em 09 de 09 de 2021, disponível em gihub: https://github.com/tapsterbot/tapsterbot

JORGENSEN, P. (2014). *Software Testing, A Craftsman's Approach .* USA: CRC Press.

LARIBI, M. A., ROMDHANE, L., & ZEGHLOUL, S. (- de 04 de 2008). Advanced Synthesis of the DELTA Parallel Robot for a Specified Workspace. *Parallel Manipulators, towards New Applications*, p. 506. doi:ISSN 978-3-902613-40-0

LONGEN, A. (19 de 08 de 2021). *O Que É npm? Introdução Básica para Iniciantes*. Acesso em 25 de 10 de 2021, disponível em Hostinger Tutoriais: https://www.hostinger.com.br/tutoriais/o-que-e-npm

MATTHEWS, R. (s.d.). *Lecture 9: ADB Topics: Basic ADB Commands*. Fonte: slideplayer: https://slideplayer.com/slide/13765739/

MERLET, J. P. (2006). *Parallel Robots* (2 ed., Vol. 128). France: Sophia-Antipolis.

MYERS, G. (2011). *The Art of Software Testing.* USA: John Wiley & Sons.

OLSEN, K., POSTHUMA, M., & ULRICH, S. (2018). *Certified Tester Foundation Level Syllabus.* International Software Testing Qualifications Board.

OTTONI, A. L. (2010). *Introdução à robótica.* São João DelRei: Universidade Federal de São João DelRei.

PANDILOV, Z., & DUKOVSKI, V. (2004). *Comparison of the Characteristics Between Serial and Parallel Robot* (Vol. 7). -: ACTA TEHNICA CORVINIENSIS – Bulletin of Engineering.

PRANAV, M., MUKILAN, A., & GANESH, C. S. (- de - de 2016). A novel design of delta robot. *International Journal of Multidisciplinary Research and Modern Education (IJMRME), II*, pp. 365-377. doi:ISSN 2454 - 6119

PREMPRANEERACH, P. (2014). *Workspace and Dynamic Trajectory Tracking of Delta Parallel Robot.* ICSEC: International Computer Science and Engineering Conference.

SALABARRIA, M. H. (2007). *Robô Hiper-Redundante com Módulo de Arquitetura Paralela.* São Paulo: Escola Politécnica da Universidade de São Paulo.

SECCHI, H. A. (2008). *Una Introducción a Los Robots Móviles.* San Juan : -.

USATEGUI, J. A., & LEÓN, J. S. (1990). *Manual Prático de Robótica* (Vol. 1). -: Hemus.

WILLIAMS II, L. (- de - de 2016). *The Delta Parallel Robot : Kinematics Solutions*. Acesso em 27 de 10 de 2021, disponível em Ohio University: https://www.ohio.edu/mechanical-faculty/williams/html/PDF/DeltaKin.pdf