**UNIVERSIDADE METODISTA DE PIRACICABA**
**FACULDADE DE ENGENHARIA, ARQUITETURA E URBANISMO**
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO**

# Identificação Automática de Rebarbas em Peças Fundidas: Uma Contribuição para a Automação do Processo de Rebarbação

**Eng. Bruno Brogio de Oliveira**
Orientador: Prof. Dr.-Ing. Klaus Schützer

Santa Bárbara d'Oeste, SP

2011

# UNIVERSIDADE METODISTA DE PIRACICABA

**FACULDADE DE ENGENHARIA, ARQUITETURA E URBANISMO**
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO**


# Identificação Automática de Rebarbas em Peças Fundidas: Uma Contribuição para a Automação do Processo de Rebarbação


**Eng. Bruno Brogio de Oliveira**
Orientador: Prof. Dr.-Ing. Klaus Schützer


Dissertação de Mestrado apresentada no Programa de Pós-Graduação em Engenharia de Produção da Faculdade de Engenharia, Arquitetura e Urbanismo, da Universidade Metodista de Piracicaba – UNIMEP.


Santa Bárbara d'Oeste, SP

2011

# Agradecimentos

Ao Prof. Dr.-Ing. Klaus Schützer pela orientação e incentivo para a conclusão deste trabalho.

A CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, pela concessão da bolsa de estudos.

A UNIMEP por ter me recebido como pesquisador.

A Universidade Técnica de Darmstadt, em especial ao DiK cujo coordenador é o Prof. Dr.-Ing. Reiner Anderl que me receberam e me aceitaram como estudante e pesquisador no período de 2008 a 2009, onde o conhecimento adquirido nessa experiência originou e possibilitou essa dissertação de mestrado.

A minha família, por todo apoio, ajuda e paciência.

Aos meus amigos que sempre estiveram presentes mesmo quando o tempo estava curto e eu não pude estar.

Aos colegas de trabalho do SCPM, por terem proporcionado um agradável ambiente de trabalho.

A RGB Sistemas que forneceram serviço, equipamento e conhecimento necessário para a realização do processo de escaneamento do modelo usado na validação desse trabalho.

# Sumário

# Índice de Figuras

# Resumo

Rebarbas estão presentes em processos de fabricação como estampagem, usinagem e fundição e sua remoção é sempre um problema complexo. Atualmente, muitas das operações de rebarbação são feitas manualmente em um ambiente com elevado nível de ruído, empoeirado e insalubre. Robôs industriais podem ser de grande contribuição para o processo de automação de fábricas e permitir uma redução na força de trabalho. Estes já foram introduzidos em processos como o de soldagem, manuseio de materiais e pintura com excelentes resultados. Robôs industriais estão hoje, também sendo aplicados em alguns tipos de usinagem e rebarbação. Esse trabalho visa contribuir com um método capaz de identificar automaticamente rebarbas em peças obtidas pelo processo de fundição para o uso principalmente em programação de robôs para esta aplicação. Por meios de cálculos matemáticos e a implementação e validação desse método em linguagem Java, este trabalho pretende contribuir com um aplicativo e um procedimento de cálculos que compare automaticamente duas malhas a qual uma se refere ao arquivo obtido pelo processo de escaneamento 3D da peça fundida, e outro obtido direto do modelo CAD, e identifique as regiões que possuem rebarbas e as separe, para que em um próximo passo, possa ser programados robôs para a realização desta tarefa.

# Abstract

Burrs are present in manufacturing processes as stamping, machining and casting, and their removal has always been a complex problem. Nowadays, many of deburring process are done manually in an extremely noisy, dusty and unhealthy environment. Industrial robots can contribute greatly to the process of factory automation and allow the reduction of the labor. These have been introduced to process such as welding, painting and material handling with excellent results. Industrial robots are today also being applied in some types of machining and deburring process. This work aims to contribute with a method which can automatically identify burrs on parts obtained by casting process in order to improve the programming of robot process for this application. Through mathematical calculations and the implementation of this method in Java language, this work intends to contribute with an application that automatically compare two meshes, one obtained by 3D scanning process of the casting part, and one obtained directly from CAD model, and identify regions that have burrs, and separate them, so in a next step, robots can be programmed to perform this task.

# 1. Introdução

Todo produto requer uma ou mais etapas de acabamento durante o seu processo de fabricação. Essas etapas são muitas vezes realizadas manualmente, processo que pode aumentar significativamente os custos de produção e aumentar o tempo necessário para sua manufatura. Dessa forma, processos que realizam a etapa de acabamento incluindo o uso de robôs industriais então em constante crescimento [1, 2].

Um processo que pode ocorrer na etapa de acabamento é o processo de rebarbação. Rebarbas são comuns estar presente em processos de fabricação como a usinagem, fundição entre outros. A remoção de rebarbas em materiais de alta resistência e com características que dificultam o acesso tornam muitos processos de rebarbação tão complexos que podem ser caracterizados como inviáveis [2].

Rebarbas são definidas como projeções indesejáveis de materiais além da borda de uma peça devido à deformação plástica durante a usinagem ou imperfeições surgidas durante o processo de fundição [3, 4].

A Figura 1.1 apresenta rebarbas provenientes do processo de usinagem,

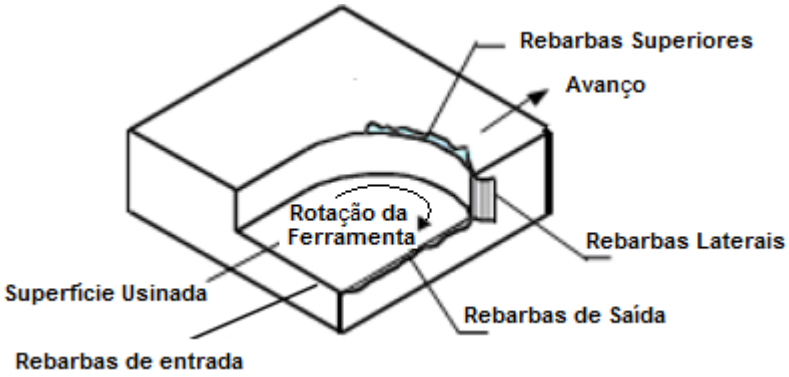



***Figura 1.1: Formação de rebarbas em peças usinadas [4].***

Os custos de mão de obra e o fato de que muitas das operações de rebarbação são feitas manualmente e na maioria das vezes em um ambiente com elevado nível de ruído, empoeirado e insalubre, fazem da rebarbação manual um processo caro e ineficiente. Como uma alternativa mais viável, um robô industrial pode ser utilizado para remover rebarbas de forma rápida e mais completa do que os métodos manuais e ainda, com melhor tolerância dimensional [4, 5].

Como exemplo, em processos de fabricação por fundição, as rebarbas e outros materiais indesejáveis que se formam precisam ser retirados [6]. Operações de limpeza e pré-usinagem são as principais atividades e representam um alto custo para as empresas. As operações de limpeza são responsáveis por cerca de 20% a 40% dos custos da manufatura por fundição. Operações de limpeza, usinagem, rebarbação e corte são aplicações promissoras para robôs industriais com a tendência da automação da indústria de fundição. [5, 7].

O processo de rebarbação pode ser aplicado em peças que exigem qualidade superficial para atender exigências como estética. Como exemplo, após a fundição de rodas de alumínio, essas precisam ser necessariamente rebarbadas e polidas antes do processo de cromagem. Caso contrário, toda e qualquer parte que não foi adequadamente suavizada será muito notável após essa operação de cromagem [8].

A Figura 1.2 ilustra uma roda obtida pelo processo de fundição e suas rebarbas.

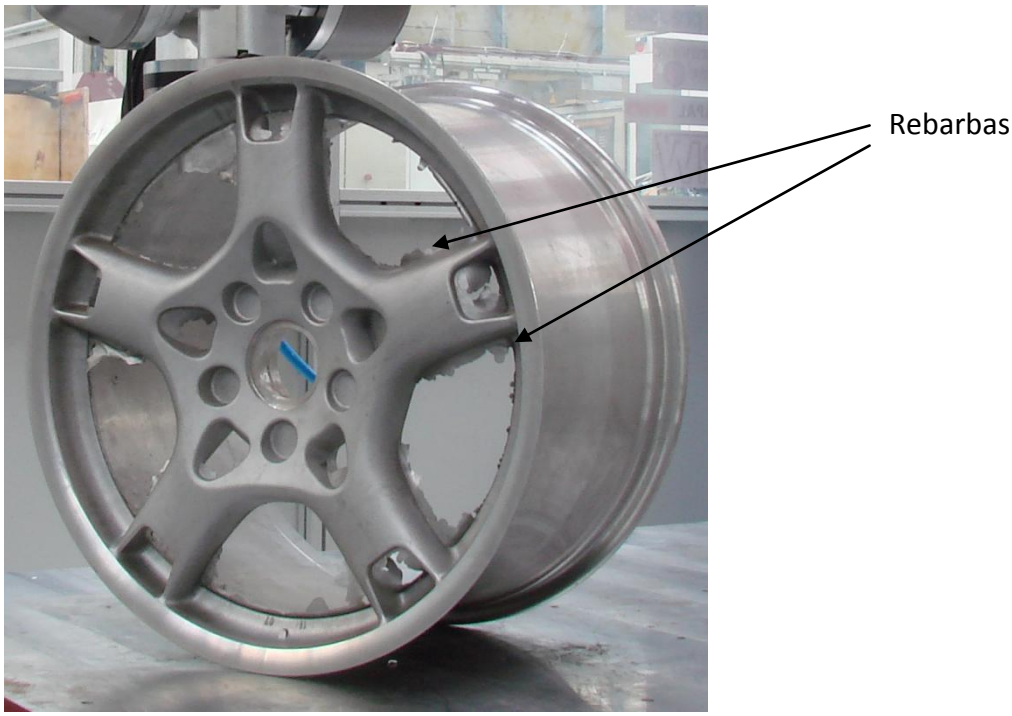



***Figura 1.2: Roda de alumínio obtida pelo processo de fundição***

O processo de rebarbação também pode ser aplicado em peças que exigem qualidade superficial para atender exigências como a resistência mecânica. Como exemplo, em componentes de motores aeronáuticos, os quais são muitas vezes submetidos a altos níveis de vibrações durante sua operação, a integridade mecânica desses componentes usinados ou fundidos pode ser comprometida pela

presença de rebarbas ou arestas cortantes. Portanto, a remoção de rebarbas é necessário para melhorar a resistência à fadiga térmica e mecânica desses componentes submetidos a um alto nível de estresse mecânico [9].

Em alguns casos, rebarbas podem ser difíceis de serem detectadas, mas principalmente difíceis de serem mensuradas devido às suas formas e dimensões variáveis. A remoção de rebarbas chega a ser um gargalo na produção e montagem das peças de subconjuntos e conjuntos [10, 11].

O método manual de geração da trajetória da ferramenta de rebarbação para robôs requer que operadores movam o robô para sucessivas posições e orientações baseados no método de tentativa e erro [8]. A desvantagem do uso desse método conhecido como *teaching* é que o método assume que após a passagem da ferramenta do robô pelas áreas indicadas, a peça está perfeita, não havendo um retorno de informação sobre a superfície e nenhuma comparação da peça rebarbada com o projeto inicial da peça [9].

Segundo uma empresa de fundição na região de Santa Bárbara D´Oeste, em São Paulo, além do ambiente com elevado nível de ruído, empoeirado e insalubre, a rebarbação é também um processo que exige muito esforço físico dos trabalhadores, uma vez que essa produz peças de proporções elevadas e para o acabamento dessas, utiliza também de lixadeiras manuais robustas. Segundo essa empresa, uma inviabilidade atual de se tentar automatizar principalmente com o uso de robôs, é a diversidade de peças que produzem e, portanto, o sistema teria que ser simples e rápido de configurar.

Existem vários estudos para facilitar a geração manual da trajetória e ainda, até para automatizar esse processo. Um dos métodos estudados é o uso de câmeras de visão para guiar os robôs por uma trajetória pré determinada. O trabalho realizado por H. Zhang at all [12] estuda um mecanismo onde uma câmera de visão faz a leitura da trajetória do robô, e sensores de força aplicados no robô garantem que o mesmo esteja o tempo todo em contato com a peça a ser rebarbada. Nessa proposta de processo, a trajetória da ferramenta é marcada manualmente na peça onde essa trajetória é reconhecida através da câmera de visão para ser seguida pelo robô [5].

Para a utilização de um sistema robótico automatizado em processos de rebarbação é necessário um mecanismo de reconhecimento das geometrias que possuem rebarbas. Em uma peça com esses contornos desconhecidos, o planejamento de uma trajetória para um robô industrial pode se tornar inviável [5].

Tendo em vista as dificuldades relacionadas ao processo manual de rebarbação apresentadas assim com a necessidade de um reconhecimento preciso da geometria e localização das rebarbas, esse trabalho apresenta como contribuição para o processo de rebarbação utilizando principalmente de robôs industriais, uma possível solução para o reconhecimento dessas geometrias através de um método onde através de cálculos matemáticos se torna possível esse reconhecimento. Esse trabalho ainda apresenta uma implementação e validação desse método em um aplicativo. Como necessidade de dados sobre a peça com rebarbas, se faz necessário então o escaneamento do modelo físico, para que assim, possa se obter um modelo matemático da peça física e suas rebarbas. Esse modelo deve então, ser alinhado computacionalmente com o modelo CAD e então, fazer os cálculos necessários para identificar as rebarbas.

**Objetivo proposto**

O objetivo deste trabalho é desenvolver um método capaz de identificar automaticamente rebarbas em peças fundidas por meio da comparação dos dados no formato de arquivo STL da peça fundida com o modelo CAD.

**Estrutura do Trabalho**

Esse trabalho foi estruturado em 7 capítulos, sendo estes:

Capítulo 1: **Introdução** – Apresenta o processo de rebarbação de superfícies complexas assim como suas dificuldades relacionadas ao processo atualmente.

Capítulo 2: **Revisão Bibliográfica** – Apresenta uma revisão bibliográfica sobre o uso e a importância de robôs, fundamentos matemáticos necessários para a realização desse trabalho e os métodos aplicados.

Capítulo 3: **Objetivos e Metodologias** – Apresenta os objetivos deste trabalho e a metodologia utilizada para alcançar os objetivos propostos.

Capítulo 4: **Descrição e Desenvolvimento do Método** – Nesse capítulo está descrito todo o método, assim como ele deve ser aplicado, os problemas e soluções para sua realização.

Capítulo 5: **Implementação e Validação** – Esse capítulo apresenta a implementação prática, e demonstra a sua validação e descrição dos experimentos, assim como os resultados obtidos.

Capítulo 6: **Conclusões** – Apresenta as conclusões obtidas por intermédio das pesquisas e da realização do método de reconhecimento de rebarbas.

Capítulo 7: **Referências Bibliográficas**

# 2 Revisão Bibliográfica

## 2.1 Utilização de Robôs e Máquinas-Ferramenta

Diversos processos de usinagem ainda são realizados manualmente devido à falta de um conceito econômico de máquinas-ferramenta, ou seja, para alguns processos de usinagem ainda não é viável economicamente o uso de uma máquina para realizar o trabalho. Isso significa precisam ser desenvolvidos processos mais flexíveis e com redução de custos em máquinas-ferramentas para substituir esses processos de usinagem que ainda são realizados de forma manual [13].

Existem diferentes conceitos de máquinas-ferramentas capazes de substituir trabalhos manuais [13]. São esses: máquinas-ferramenta convencionais, máquinas-ferramenta de cinemática paralela e robôs industriais (cinemática seqüencial). Sobre esses conceitos podem ser analisados os fatores como espaço de trabalho, utilização do espaço, investimento rigidez e exatidão da trajetória da ferramenta Observa-se a partir desses conceitos que Robôs Industriais (IRs - Industrial Robots) possuem vantagens significativas em relação a espaço, investimento necessário e o grau de utilização do espaço. Além disso, eles possuem alto grau de flexibilidade permitindo também a usinagem de peças complexas. Devido a essas vantagens, apresentam grande potencial para o uso em processos de usinagem. Entretanto, apenas 3% do uso de robôs são destinados para a usinagem. Esta situação pode estar relacionada basicamente com várias desvantagens como falta de precisão de posicionamento, rigidez mecânica, baixa precisão da trajetória da ferramenta e a complexidade da programação quando comparado a uma máquina-ferramenta convencional [13, 14].

De acordo com o roteiro de robótica CARE "*Coordination Action for Robotics in Europe*", na Europa o custo de produção eficiente e a produção de alta qualidade só são possíveis com a utilização da robótica, devido aos custos de mão de obra elevados. Além disso, no mundo ocidental, é esperada uma escassez de trabalhadores qualificados como resultado do envelhecimento da sociedade. Isso não afeta exclusivamente o alto volume de fabricação onde a robótica industrial já é aplicada, mas principalmente as produções de pequena escala onde um grau maior e mais flexível de automação é exigido para que a Europa continue a ser competitiva e mantenha o seu padrão de vida atual [15].

Cada vez mais as manufaturas de pequena escala e manufaturas flexíveis estão incluindo IRs devido a que a vantagem de robôs vem principalmente de sua rápida

adaptação e menor custo se comparado as máquinas-ferramenta convencionais [15].

Segundo alguns sites comerciais, o Brasil possui qualidade em pesquisas teóricas sobre o uso de robôs, mas a aplicação industrial ainda é pequena. Um dos motivos é a falta de investimento do governo e a falta de políticas de desenvolvimento sobre o setor.

Inovações na área da engenharia têm cada vez mais envolvido robôs na área da usinagem. Nos dias de hoje, IRs destinados à usinagem, comandados por sistemas computacionais, são empregados para produzir superfícies complexas para suprir as necessidades de manufaturas como as das indústrias automobilística, aeronáutica e naval. Esses estão se tornando mais e mais populares devido a sua capacidade de lidar com peças de geometria complexa compostas dos mais diversos materiais como borracha, madeira, pedra, metal, plástico etc. Além disso, cada vez mais robôs são caracterizados pela sua alta taxa de remoção de material, e um acabamento superficial eficiente quando comparados a processos tradicionais principalmente quando realizado manualmente [16].

## 2.2 Aplicação de Robôs

Robôs estão cada vez mais inseridos em nossas vidas. Hoje em dia, a área de micro-robôs para o setor medicinal está em constante pesquisa e cada vez mais em utilização. Esses causam redução no trauma dos pacientes e ainda, permitem acesso a até então, áreas não acessíveis aos seres humanos como artérias do crânio. [17].

Robôs também podem ser encontrados na indústria alimentícia. Higiene e limpeza são obviamente importantes para a indústria de alimentos e os robôs atuais são projetados para atender a esse requisito [18, 19]. A indústria alimentar está também sob crescente pressão para reduzir custos de produção devido a elevados níveis de concorrência. A indústria está agora à procura cada vez mais para automação e robótica para ajudar a reduzir os custos de produção [20].

Na indústria de manufatura, um robô industrial pode ser de grande contribuição para o processo de automação de fábricas e permitir uma redução na força de trabalho. IRs já foram introduzidos no processo de soldagem e manuseio de materiais onde apresentaram excelentes resultados. [21].

A alta velocidade de um robô é um dos requisitos essenciais para minimizar o tempo de ciclo de produção em muitos sistemas de produção automatizados [22].

Uma das principais questões que os fabricantes de robôs devem abordar no futuro próximo, é que alguns fatores como a falta de agilidade por causa de sua automação ainda complexa ou o fato dos fabricantes não conseguirem expor a flexibilidade que robôs industriais possuem para os usuários podem torná-los menos interessantes [23].

A Figura 2.1 mostra um robô industrial da marca ABB, modelo IRB 660, destinado ao manuseio de produtos.


***Figura 2.1: Robô Industrial ABB IRB 660***

### 2.2.1 Aplicação em Usinagem

Nos últimos anos, vários estudos vêm sendo realizados sobre robôs articulados para aplicações em processos de usinagem e processos de rebarbação [21].

No entanto, existem ainda poucos estudos sobre as aplicações no processo de usinagem porque os robôs articulados apresentam baixa rigidez e apresentam deformações substanciais, reduzindo assim a precisão de sua trajetória [21].

A aplicação de robôs industriais, em operações de usinagem de peças complexas se apresenta como uma alternativa promissora quando comparada às máquinas-ferramentas tradicionais no que se refere a custos. Os principais fatores que

influenciam no resultado da aplicação de robôs na usinagem são os relacionados à rigidez estática e dinâmica de seus componentes [13].

A Figura 2.2 apresenta o IRB 6660, da ABB, para pré-usinagem. É o primeiro robô no mercado dedicado à pré-usinagem e sua aplicação é destinada principalmente à indústria de fundição. Este novo robô foi projetado para aplicações de alto desempenho, nos quais a estabilidade é um fator chave para o sucesso [24].


***Figura 2.2: Robô para Usinagem ABB IRB 6660***

#### 2.2.1.1 Furação

Sistemas de furação automática têm uma longa história, tanto na área da indústria quanto na comunidade de pesquisa. Em particular, o uso de robôs industriais para a furação é interessante devido à sua flexibilidade de programação e do custo relativamente baixo de sistemas com robôs industriais. No entanto, a furação utilizando robôs é uma tarefa muito desafiadora devido à rigidez mecânica relativamente baixa dos robôs industriais em uso hoje. O processo de furação gera forças radiais, tangenciais e axiais durante o processo que faz o robô desviar, às vezes até vários milímetros e também, devido à própria flexão do robô e da

elasticidade nas suas juntas. Os resultados dessas deformações são furos com baixa qualidade e posicionamento impreciso [25].

#### 2.2.1.2 Rebarbação

A rebarbação é normalmente o último processo do ciclo de produtivo de uma peça e apresenta grandes possibilidades de falha como, por exemplo, precisão geométrica. A tarefa de rebarbação é uma mão de obra intensiva e de baixa produtividade, altos custos e ambiente de trabalho perigoso. Estudos mostram que em processos de rebarbação manual, o tempo do processo aumenta exponencialmente em relação à espessura da rebarba e a dificuldade de acesso a elas [26].

Operações de rebarbação consistem em retirar as projeções de material além das bordas de uma peça que podem ser geradas pelos processos de estampagem, usinagem ou fundição. A formação de rebarbas ao longo da peça compromete a qualidade da superfície e conseqüentemente reduz a funcionalidade e durabilidade do produto. Rebarbas nas arestas e cantos podem contribuir para o aparecimento de trincas e ruptura da peça. Além disso, rebarbas podem ainda causar ferimentos, como por exemplo, corte na pele. Normalmente, as rebarbas são removidas manualmente pelo fato de ser um processo complexo e as soluções tradicionais de robôs não possuírem recursos para controle de força e sistemas de visão. Bordas irregulares têm que ser rebarbadas dentro de tolerâncias para manter a exatidão da peça. Além disso, o tamanho das rebarbas e sua distribuição podem variar significativamente, dependendo do processo de fabricação [27, 28].

Numerosos processos tradicionais e não-tradicionais de rebarbação têm sido desenvolvidos, mas muitos destes são inúteis para rebarbar matérias de alta resistência como ligas e compósitos com características específicas [3].

Em processos de fabricação por fundição, operações de limpeza e pré-usinagem são as principais atividades e representam um alto custo para as empresas [8].

A aplicação de robôs manipuladores automáticos para substituição da operação manual de rebarbação tem se tornado mais e mais importante devido ao alto custo de rebarbação para determinados tipos de peças fundidas. Em geral, tarefas de rebarbação são destinadas a remover as rebarbas das bordas de peças e manter a geometria final dentro das tolerâncias dimensionais permitidas. Para isso, uma ferramenta de corte necessita chanfrar as bordas enquanto se desloca ao longo da peça. Portanto, ao conduzir a ferramenta de corte para executar uma tarefa rebarbação, os robôs devem implementar dois movimentos principais: um para

aplicar a força adequada para chanfrar a borda da peça de modo a remover as rebarbas, e o outro é para executar um movimento de contorno seguinte para garantir que a ferramenta de corte mantenha contato com todas as rebarbas que se formaram ao longo do chanfro [29].

Os principais problemas no controle de robôs no processo de rebarbação surgem principalmente devido à incerteza dos robôs destinados a usinagem devido aos problemas de menor rigidez mecânica e o complexo processo de rebarbação [28].

Rebarbação por meio de robôs tem um mérito forte quando a peça é complexa para ser rebarbada manualmente [30]. Não há dúvida de que com um robô, a qualidade é aumentada e o tempo de ciclo é reduzido quando comparado a rebarbação manual. O uso de robôs para rebarbação é por outro lado um processo caro e por isso só se justifica quando os volumes são em grande escala [31].

Geralmente, uma ferramenta de rebarbar é montada em um centro de usinagem CNC ou em um robô manipulador. Máquinas CNC são indicadas por causa de sua rigidez e precisão. No entanto, o custo destas máquinas é demasiadamente alto para justificar seu uso em um processo de rebarbação além do fato de muitas dessas máquinas-ferramentas não serem indicadas devidos ao fato de rebarbas possuírem grãos abrasivos. Adicionalmente, máquinas CNC têm limitado o número de eixos e amplitude de movimentos limitados em funcao de aumentar a rigidez e precisão, tornando essas máquinas impróprias para muitas aplicações. Robôs, em contrapartida, são menos rígidos e precisos, mas oferecem um maior volume de trabalho e mais eixos controláveis a um custo menor do que as máquinas CNC. Além disso, um sistema de rebarbação com base em um robô manipulador é mais eficaz do um centro de usinagem CNC porque reduz o espaço de trabalho, gastos e tempo de trabalho [28].

As operações de rebarbação e polimento em uma empresa fabricante de rodas de alumínio, por exemplo, são realizadas manualmente e requerem grande números de trabalhadores. São operações tediosas e monótonas que respondem por muitas das ocorrências de lesões por esforços repetitivos. Devido à flexibilidade dos robôs industriais comparados as máquinas convencionais, esses são mais indicados para esse tipo de automação [12].

### 2.2.2 Aplicação em Soldagem

Soldagem é com certeza a área mais popular de aplicação de robôs industriais [23].

As áreas de aplicação tradicional de robôs industriais envolvem operações altamente repetitivas, como solda a ponto, solda por arco elétrico ou ainda, solda por fricção [25, 32].

Um dos principais problemas relacionados a soldagem é a qualidade do cordão de solda relacionada com a homogeneidade da microestrutura e suas características físicas e dimensionais. Esses fatores estão diretamente relacionados quando o processo é manual, isso porque o soldador não é capaz de soldar com boa repetibilidade todos os cordões de solda. Dessa forma, robôs estão cada vez mais viáveis para esse tipo de tarefa, devido principalmente à sua boa repetibilidade [33].

No entanto, quando os robôs são utilizados no chão de fábrica para executar tarefas de soldagem, há ainda muito trabalho adicional para garantir que eles realizem a tarefa com a qualidade exigida. O processo também é muito demorado, o que significa que os robôs demoram tempo para serem programados e configurados. Isso é um problema, uma vez que conceitos atuais como disponibilidade e agilidade são questões-chave da produção [23]. A maioria dos processos de programação de robôs na área de soldagem é feita pelo procedimento de *teaching* ou programação off-line [34].


***Figura 2.3: Robô ABB IRB 1600ID destinado a Soldagem***

Alta eficiência, controle digital e robotização tem sido a tendência crescente para solda elétrica, que segue na direção da inserção da integração da inteligência artificial, visão computacional, processamento digital e robôs industriais na

tecnologia de soldagem. Essa nova tendência pode influenciar a forma tradicional de fabricação envolvendo soldagem, e possibilitar o processo ser mais flexível e inteligente [34].

A Figura 2.3 mostra o IRB 1600ID, da empresa ABB. Todos os cabos e mangueiras estão localizados dentro do braço, tornando o robô perfeitamente adequado para a soldagem.

### 2.2.3 Manuseio e Montagem

Em operações que requerem repetibilidade e tolerâncias dimensionais, robôs industriais mostram vantagens em relação ao trabalho manual. No entanto, apesar de sua alta produtividade e confiabilidade, ainda não são capazes de executar operações complexas de montagem [27].

As operações de transporte e manuseio de materiais ("*materials handling*") são aquelas em que o robô move materiais ou componentes de uma posição para outra O transporte de componentes ou materiais é uma aplicação ideal para um robô industrial. É normalmente, uma tarefa repetitiva e que normalmente requer pouca complexidade. Para poder transportar os materiais ou componentes o robô é equipado com uma garra na sua parte terminal [35].

A Figura 2.4 apresenta um robô destinado à movimentação de carga, da marca ABB, modelo IRB 6620..


***Figura 2.4: Robô ABB IRB 6620 para manuseio***

A Figura 2.5 apresenta um robô com funcionalidades de montagem e transporte, da empresa KUKA.

***Figura 2.5: Robô KUKA KR 6-2 destinado à operação de manuseio***

### 2.2.4 Pintura

Robôs industriais são amplamente utilizados na indústria na área de pintura, especialmente na área automotiva. A uniformidade da espessura da pintura em um produto pode influenciar fortemente a sua qualidade, que é determinada pela trajetória da ferramenta acoplada ao robô. O planejamento da trajetória com a distribuição de material uniforme continua a ser um desafio nos processos de pintura. Atualmente, há dois métodos para o planejamento da trajetória para um robô de pintura: O Manual e o automático [27, 36].

Métodos manuais de planejamento da trajetória (métodos de *teaching*) são realizados com base na experiência dos engenheiros de produção e conhecimento de instalações de produção, equipamentos, suas capacidades, processos e ferramentas. Um engenheiro de produção tem que executar testes extensivos em uma célula de trabalho para otimizar um caminho de ferramenta para obter uma distribuição uniforme de pintura. Este processo é complexo, muito demorado e os resultados variam de acordo com habilidade de cada engenheiro de fabricação. Normalmente exige que os engenheiros utilizem uma abordagem de tentativa e erro para localizar uma trajetória considerada aceitável para a ferramenta do robô para

os processos de pintura. Existem sistemas de planejamento da trajetória da ferramenta com auxilio de computador (CATP - *Computer aided tool path planning*), no qual a trajetória da ferramenta é automaticamente correlacionada com o modelo CAD da peça a ser produzida, o que reduz o trabalho humano e evita que os operadores sejam expostos a ambientes de trabalho prejudiciais. Porém, ainda é essencial desenvolver metodologias de planejamento da trajetória para substituir o processo manual pelo automático [36].

A Figura 2.6 apresenta o Robô IRB 52, da empresa ABB, destinado à pintura.


***Figura 2.6: Robô IRB 52, destinado para pintura***

## 2.3 Fundamentos Matemáticos

Como neste trabalho as ferramentas matemáticas são fundamentais, são apresentadas aqui algumas propriedades matemáticas necessárias para uma melhor compreensão do trabalho.

### 2.3.1 Operações com Vetores

O termo vetor é usado pelos cientistas para indicar uma quantidade como a velocidade ou força, que tem tanto magnitude (módulo ou norma) quanto direção e sentido. Um vetor é freqüentemente indicado por uma flecha ou por um segmento de reta orientado. O comprimento da flecha representa o módulo do vetor, e a flecha, aponta a direção no sentido do vetor [37]. A Figura 2.7 mostra uma partícula se movendo ao longo de um caminho no plano e seu vetor velocidade **v** numa posição específica da partícula. Aqui o comprimento da flecha representa a velocidade da partícula, e ela indica a direção e o sentido em que a mesma está se movimentando.

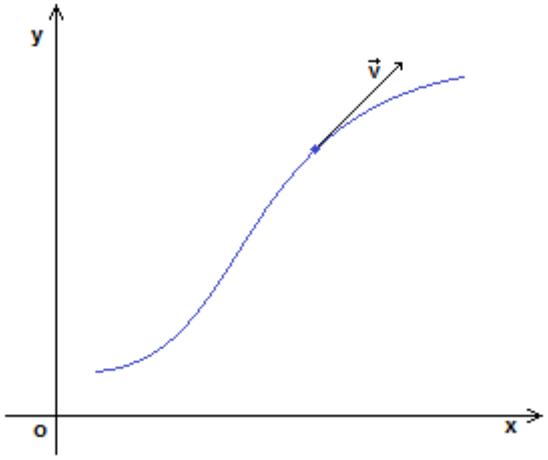



***Figura 2.7: Vetor Velocidade $\vec{V}$***

**Grandeza Vetorial**

Um vetor é composto por:

- Módulo

- Direção

- Sentido

O símbolo para representação de um vetor qualquer V pode ser $\vec{V}$

**Módulo de um Vetor**

Módulo de um vetor, ou norma, é o comprimento de um vetor, ou também, a distância da origem à sua extremidade [37].

O módulo de um vetor pode ser calculado utilizando a equação 2.1.

$$\left|\vec{V}\right| = \sqrt{x^2 + y^2 + z^2} \qquad \textbf{(2.1)}$$

**Propriedades de um Vetor**

Vetor oposto: Dado um vetor qualquer $\vec{V}$ , vetor oposto é o vetor $-\vec{V}$ de igual módulo e direção, só que com direção oposta ao vetor $\vec{V}$

Vetor Unitário: É todo vetor cujo módulo vale um. Ou seja, $|\vec{V}|$=1

Vetor Nulo: É o vetor cujo módulo vale zero e, portanto, sua direção e sentido não podem ser determinados [37].

Dado dois vetores, $A = (x_a, y_a, z_a)$ e $B = (x_b, y_b, z_b)\cdots$, são definidas as seguintes operações aritméticas:

**Soma Vetorial**

O vetor Resultante $R = (x_r, y_r, z_r)$ é o resultado da soma vetorial $\vec{A} + \vec{B}$, realizada na forma:

$$R = (x_a + x_b, y_a + y_b, z_a + z_b)$$

A Figura 2.8 ilustra essa propriedade.

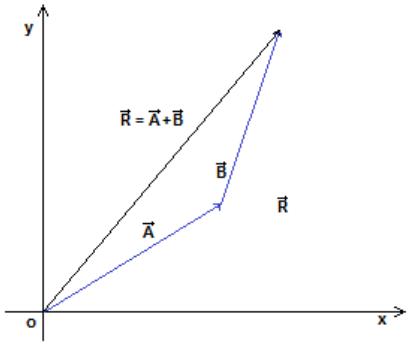



***Figura 2.8: Soma de dois vetores [37].***

Outra alternativa para a interpretação da soma de vetores é a Lei do Paralelogramo, conforme ilustrado na Figura 2.9:

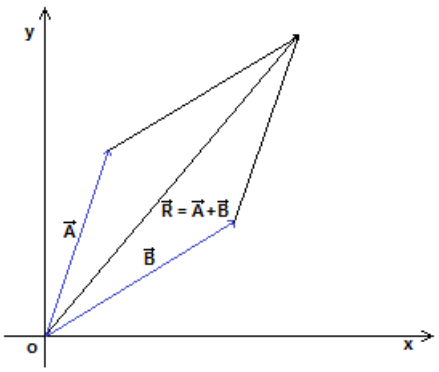



***Figura 2.9: Lei do Paralelogramo***

**Produto Escalar**

Produto escalar é o produto entre dois vetores, cujo resultado é um número escalar [37].

A Figura 2.10 demonstra graficamente um produto escalar. A equação 2.2 apresenta como esse produto pode ser calculado.

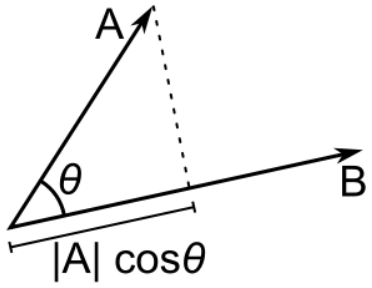



***Figura 2.10: Produto Escalar***

$$\vec{A}.\vec{B} = |\vec{A}|.|\vec{B}|.cos\,\theta \qquad \textbf{(2.2)}$$

Da definição acima, deduz-se que:

a) Se dois vetores são paralelos, $\theta = 0°$ e $\cos\theta = 1$ então o produto interno deles, coincidirá com o produto dos seus módulos.

$$\vec{A}.\vec{B} = (x_a.x_b + y_a.y_b + z_a.z_b)$$

b) O produto escalar de um vetor por ele mesmo será igual ao quadrado do seu módulo.

Portanto, $\theta = 0°$ e $\cos\theta = 1$ $\therefore$ $\vec{A}.\vec{A} = |\vec{A}|.|\vec{A}|.1 = |\vec{A}|^2$

c) Se dois vetores são perpendiculares, $\theta = 90°$ e $\cos\theta = 0$ então o produto escalar deles será nulo.

d) O produto escalar de dois vetores será sempre um número real.

e) O produto interno de vetores é também conhecido como produto escalar.

O $\cos\theta$ pode ser determinado pela equação 2.3:

$$\cos\theta = \frac{x_a.x_b + y_a.y_b + z_a.z_b}{|\vec{A}|.|\vec{B}|} \qquad \textbf{(2.3)}$$

Se $|\vec{A}| = |\vec{B}| = 1$, então o produto interno é o cosseno do ângulo entre os dois vetores.

**Produto Vetorial**

O produto vetorial **A x B** de dois vetores **a** e **b**, ao contrário do produto escalar, é um vetor, por isso seu nome [37].

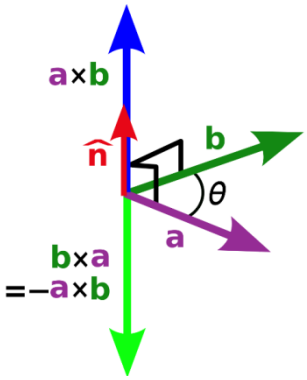



***Figura 2.11: Produto Vetorial***

Se $A = (x_a, y_a, z_a)$ e $B = (x_b, y_b, z_b)\cdot\cdot$, então o produto vetorial é o vetor dado pela equação 2.4 [37]:

$$\vec{A}x\vec{B} = y_a.z_b - z_a.y_b, z_a.y_b - x_a.z_b, x_a.y_b - y_a.x_b \qquad \textbf{(2.4)}$$

### 2.3.2 Matrizes

Sejam $m \geq 1$ e $n \geq 1$ dois números inteiros. Uma matriz $m\,x\,n$ real é uma dupla seqüência de números reais, distribuídos em m linha e n colunas, formando uma tabela que se indica do seguinte modo [38]:

$$\begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix}$$

Abreviadamente, essa matriz pode ser expressa por $(a_{ij})_{\substack{1\leq i\leq m\\ 1\leq j\leq n}}$ ou apenas $(a_{ij})$, se não houver possibilidade de confusão quanto à variação dos índices [38].

**Operações com Matrizes**

**Adição**

Sejam $A = (a_{ij})$ e $B = (b_{ij})$ matrizes m x n. Indicamos por A + B e chamamos soma de A com B a matriz m x n cujo termo geral é $a_{ij} + b_{ij}$, ou seja:

$$A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{pmatrix} \qquad \textbf{(2.5)}$$

**Multiplicação de uma Matriz por um número**

Dada uma matriz real $A = (a_{ij})$, m x n, e dado um número real b, o produto de A por b é uma matriz real m x n dada por:

$$b.A = \begin{pmatrix} b.a_{11} & b.a_{12} & \dots & b.a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ b.a_{m1} & b.a_{m2} & \dots & b.a_{mn} \end{pmatrix} \qquad \textbf{(2.6)}$$

**Multiplicação de Matrizes**

Sejam $A = (a_{ij})$ de tipo m x n e $B = (b_{jk})$ de tipo n x p. O produto AB é a matriz m x p cujo termo geral é dado por [38]:

$$c_{ik} = \sum_{j=1}^{n} a_{ij}.b_{jk} = a_{i1}.b_{1k} + \cdots + a_{in}.b_{nk}$$

**Matriz Identidade**

Matriz identidade é uma matriz quadrada e uma matriz diagonal, cuja função é de ser o elemento neutro na multiplicação de matrizes. É denotada por In (onde n é a ordem da matriz), ou simplesmente por I. A matriz é construída da seguinte forma: os elementos da diagonal principal têm valor um, e os demais elementos da matriz são zero, como mostrado na equação abaixo [38]:

$$I_n = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

**Matriz Inversa**

Uma matriz A de ordem n se diz inversível se, e somente se, existe uma matriz B, também de ordem n, de modo que [38]:

$$A.B = B.A = I_n$$

Esta matriz B, caso exista, é única, chama-se inversa de A e é indicada por $A^{-1}$.

Por exemplo, dada a matriz A:

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

É inversível uma vez que $det(A) \neq 0$:

$$B = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}$$

Resulta:

$$A.B = B.A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I_2 \qquad \textbf{(2.7)}$$

### 2.3.3 Autovalores e Autovetores

Dada uma transformação linear de um espaço vetorial nele mesmo, $T: V \to V$, pretende-se saber que vetores seriam levados neles mesmos por essa transformação.

**Autovalores e Autovetores de uma Matriz**

Dada uma matriz quadrada A de ordem n, entende-se por autovalor e autovetor de A o autovalor e o autovetor da transformação linar $T_A: R^n \to R^n$, associada à matriz A em relação a base canônica, isto é, $T_A(A) = A.v$ (na forma coluna). Assim, um autovalor $\lambda \in R$ de A, e um autovetor v ∈ $R^3$, são soluções da equação $A.v = \lambda v, v \neq 0$ [39].

Dada uma matriz diagonal:

$$A = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

E dado os vetores $e_1 = (1, 0, \dots, 0)$, $e_2 = (0, 1, \dots, 0)$, e $e_n = (0, 0, \dots, 1)$, tem-se:

$$A.e_1 = \begin{bmatrix} a_{11} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = a_{11}.e_1$$

E em geral, $A.e_i = a_{ii}.e_i$. Então, estes vetores da base canônica de $R^n$ são autovetores para A, e o autovetor $e_i$ é associado ao autovalor $a_{ii}$. [38].

**Polinômio Característico**

Conforme observado no item anterior desse capítulo, nas definições de autovalores e autovetores, para efetuar os cálculos que determinam seus autovalores, se faz necessário adotar um procedimento complexo. Dessa forma, é preciso então utilizar de um método prático para encontrar autovalores e autovetores de uma matriz real de ordem n [39].

Como exemplo, o caso de uma matriz de ordem n=3 e em seguida, o equacionamento para generalizar para qualquer ordem n [39].

Exemplo:

$$A = \begin{pmatrix} 4 & 2 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

Deseja-se conhecer vetores $v \in R^3$ e escalares $\lambda \in R$ tais que $A.v = \lambda.v$. Se I for a matriz identidade de ordem 3, então a equação descrita pode ser escrita na forma $A.v = (\lambda.I).v$, ou ainda $(A - \lambda.I).v = 0$.

Escrevendo explicitamente:

$$\left( \begin{bmatrix} 4 & 2 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \right) . \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

temos então a seguinte equação matricial:

$$\begin{bmatrix} 4-\lambda & 2 & 0 \\ -1 & 1-\lambda & 0 \\ 0 & 1 & 2-\lambda \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Escrevendo explicitamente o sistema de equações lineares equivalentes a esta equação matricial, obtem-se um sistema de três equações e três incógnitas. Se o determinante da matriz dos coeficientes for diferente de zero, sabendo que este sistema tem uma única solução, que é a solução nula, ou seja, x = y = z = 0. Porém, o interesse é calcular os autovetores de A, isto é, vetores $v \neq 0$, tais que $(A - \lambda.I.v=0$. Neste caso, $detA - \lambda I$ deve ser zero, ou seja:

$$\begin{bmatrix} 4-\lambda & 2 & 0 \\ -1 & 1-\lambda & 0 \\ 0 & 1 & 2-\lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

E, portanto: $-\lambda^3 + 7\lambda^2 - 16\lambda + 12 = 0$.

Observa-se, portanto, que $det(A - \lambda I)$ é um polinômio em $\lambda$. Este polinômio é chamado de polinômio característico de A. Continuando a resolução, tem-se:

$$(\lambda - 2)^2(\lambda - 3) = 0.$$

Logo, $\lambda = 2$ e $\lambda = 3$ são raízes do polinômio característico de A, e portanto os autovalores da matriz A são 2 e 3. Conhecendo os autovalores pode ser encontrados os autovetores correspondentes. Resolvendo a equação $A.v = \lambda.v$, para os casos:

1. $\lambda = 2$

$$\begin{bmatrix} 4 & 2 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 2. \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{cases} 4x + 2y = 2x \\ -x + y = 2y \\ y + 2z = 2z \end{cases}$$

A terceira equação implica que y=0 e por isso vemos na segunda que x=0. Como nenhuma equação impõe uma restrição em z, os autovetores associados a $\lambda = 2$ são do tipo$(0, 0, z)$, ou seja, pertencem ao subespaço $[(0, 0, 1)]$.

2. $\lambda = 3$

$$\begin{bmatrix} 4 & 2 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 3. \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{cases} 4x + 2y = 3x \\ -x + y = 3y \\ y + 2z = 3z \end{cases}$$

Tanto da primeira equação quento da segunda vemos que $x = -2y$ e da terceira vem $z = y$. Os autovetores associados ao autovalor $\lambda = 3$ são do tipo$(-2y, y, z)$, ou seja, pertencem ao subespaço $[(-2, 1, 1)]$.

O que foi feito para a matriz A de ordem 3, pode ser generalizado. Seja A uma matriz de ordem n. Os autovalores e autovetores correspondentes de A são exatamente aqueles que satisfazem a equação $[A - \lambda I]v$ =0.

Escrevendo essa equação explicitamente temos:

$$\begin{bmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Chamando a primeira matriz acima de A, então $A.v = 0$. Se $detA \neq 0$, sabe-se que o sistema de equações lineares homogêneo indicado tem uma única solução. Se $x_1 = x_2 = \cdots = x_3 = 0$, ou v = 0 é solução de um sistema homogêneo, então esta única solução é nula. Assim, a única maneira de encontrar autovetores v é termos $detA = 0$, ou seja:

$$det(A - \lambda I) = 0$$

Impondo essa condição, determina-se primeiramente os autovalores de $\lambda$ que satisfazem a equação e depois os autovetores a eles associados. Observa-se que:

$$P(\lambda) = det(A - \lambda I) = det \begin{bmatrix} a_{11} - \lambda & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} - \lambda \end{bmatrix}$$

É um polinômio de λ de grau n.

$\mathrm{P}(\lambda) = (\mathrm{a}_{11} - \lambda) \dots (\mathrm{a}_{\mathrm{nn}} - \lambda)$ termos de grau <n, e os autovalores procurados são as raízes desse polinômio. $\mathrm{P}(\lambda)$ é chamado polinômio característico [38].

### 2.3.4 Números Complexos

Quando os matemáticos encontram equações cujas soluções não podem ser expressas em termos de funções conhecidas, é comum que definam sua solução como nova função, e passem a determinar suas principais características. No passado, a não possibilidade dos números racionais existentes dar solução para $x^2 = 2$ levou a extensão dos conjuntos de números aos números reais, que incorporam os irracionais. Por sua vez, a não possibilidade dos reais para ajudar a resolver equações como $x^2 = -1$, $(x-3)^4 = -1$, levou a extensão dos reias aos números complexos [40].

Para resolver, portanto, a equação $x^2 = -1$, que resulta em $x = \pm\sqrt{-1}$, introduziu-se uma quantidade i que obedece a regra $i^2 = -1$.

Define-se, portanto, um número complexo como da forma $a + bi$, onde *a* e *b* são números reais.

Portanto, regras foram estabelecidas para as operações aritméticas envolvendo esses números [40].

Sendo dois números complexos: $z_1 = (a + bi)$ e $z_2 = (c + di)$

**Adição:**

$$z_1 + z_2 = (a + bi) + (c + di) = (a + c) + (b + d)i \qquad \textbf{(2.8)}$$

**Subtração:**

$$z_1 - z_2 = (a + bi) - (c + di) = (a - c) + (b - d)i \qquad \textbf{(2.9)}$$

**Multiplicação:**

$$z_1 . z_2 = (a + bi).(c + di) = (a.c - b.d) + (b.c + a.d)i \qquad \textbf{(2.10)}$$

**Conjugado:**

Cada número complexo $a + bi$ corresponde um único número complexo $a - bi$ chamado o conjugado. O conjugado de $z$ é denotado $\bar{z}$.

**Divisão:**

$$\frac{z_1}{z_2} = \frac{z_1}{z_2}.\frac{\overline{z_2}}{\overline{z_2}} = \frac{z_1\overline{z_2}}{z_2\overline{z_2}} \qquad \textit{(2.11)}$$

O que pode ser também expresso pela equação abaixo:

$$\frac{z1}{z2} = \frac{a+b.i}{c+d.i} = \frac{a.c+b.d}{c^2+d^2} + \frac{b.c-a.d}{c^2+d^2} \qquad \textit{(2.12)}$$

**Módulo:**

$$|z| = \sqrt{a^2+b^2} \qquad \textit{(2.13)}$$

**Ângulo:**

$$\varphi = tg^{-1}\left(\frac{b}{a}\right) \qquad \textit{(2.14)}$$

**Raiz Enésima:**

$$\sqrt[n]{z} = \sqrt[n]{|z|}.\left(cos\frac{\varphi+2.k.\pi}{n} + \left(sen\frac{\varphi+2.k.\pi}{n}\right)\right) \qquad \textit{(2.15)}$$

### 2.3.5 Polinômios Terceiro Grau

Para encontrar as raízes de polinômios de grau três, Cardano e Tagalia desenvolveram o seguinte método [41]:

Dado o polinômio de grau três na forma:

$$A.x^3 + B.x^2 + C.x + D = 0$$

Deve-se primeiro deixá-lo na forma harmônica:

$$x^3 + \frac{B}{A}.x^2 + \frac{C}{A}.x + \frac{D}{A} = 0$$

onde ficará na forma:

$$y^3 + a.y^2 + b.y + c = 0$$

São soluções desse polinômio a equação:

$$x_k = t_k - \frac{a}{3} \; ; k = 1, 2 \; e \; 3$$

onde:

$$t_1 = \sqrt[3]{-\frac{q}{2} + \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} + \sqrt[3]{-\frac{q}{2} - \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}$$

$$t_2 = \left(-\frac{1}{2} - \frac{\sqrt{3}}{2}.i\right).\sqrt[3]{-\frac{q}{2} + \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} + \left(-\frac{1}{2} + \frac{\sqrt{3}}{2}.i\right).\sqrt[3]{-\frac{q}{2} - \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}$$

$$t_3 = \left(-\frac{1}{2} + \frac{\sqrt{3}}{2}.i\right).\sqrt[3]{-\frac{q}{2} + \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} + \left(-\frac{1}{2} - \frac{\sqrt{3}}{2}.i\right).\sqrt[3]{-\frac{q}{2} - \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}$$

### 2.3.6 Polinômios Quarto Grau

O método proposto por Horn, para a resolução do alinhamento sugere o método proposto por Ludovico Ferrari.

Ludovico Ferrari (1522-1560) nasceu em Bologna. Foi o mais famoso dos discípulos de Cardano. De origem muito humilde foi trabalhar como servo na casa de Cardano quando tinha 15 anos. Sua inteligência foi logo reconhecida e logo ocupou o cargo de secretário. Seu gênio incontrolável gerava constantes atritos com Cardano, mas apesar disso, eram amigos e colaboradores. A partir dos 18 anos, Ferrari passou a ensinar por conta própria em Milão e sob a proteção do Cardeal de Mantôva, alcançou posições que lhe proporcionavam boa renda. Aos 38 anos tornar-se professor de matemática na Universidade de Bologna e em seguida veio a falecer, provavelmente envenenado pela própria irmã [42].

Como era costume na época os matemáticos proporem desafios uns aos outros. Zuanne de Tonini da Coi propôs uma questão que envolvia a equação:

$$x^4 + 6x^2 - 60x + 36 = 0$$

Após inúmeras tentativas sem êxito, Cardano passou a questão a Ferrari, que encontrou um método geral para a solução das equações do 4º grau. Tal método foi publicado por Cardano em Ars Magna.

**Resumo do Método**

Dado o polinômio de grau quatro:

$$A.x^4 + B.x^3 + C.x^2 + D.x + E = 0$$

O método proposto por Ferrari soluciona polinômios de grau quatro na forma harmônica, portanto, deve estar conforme o polinômio abaixo:

$$a.x^4 + \frac{b}{a}.x^3 + \frac{c}{a}.x^2 + \frac{d}{a}.x + \frac{e}{a} = 0$$

Em seguida, deve reduzir o polinômio de forma a eliminar o termo de grau três, conforme o polinômio descrito abaixo:

$$u^4 + \alpha.u^2 + \beta.u + \gamma = 0$$

em que:

$$\alpha = -\frac{3.b^2}{8.a^2} + \frac{c}{a}$$

$$\beta = \frac{b^3}{8.a^3} - \frac{b.c}{2.a^2} + \frac{d}{a}$$

$$\gamma = -\frac{3.b^4}{256.a^4} + \frac{c.b^2}{16.a^3} - \frac{b.d}{4.a^2} + \frac{e}{a}$$

O polinômio de grau três pode ser escrito a partir do polinômio de grau quatro reduzido, na forma:

$$y^3 + \frac{5}{2}.\alpha.y^2 + (2.\alpha^2 - \gamma).y + \left(\frac{\alpha^3}{2} - \frac{\alpha.\gamma}{2} - \frac{\beta^2}{8}\right) = 0$$

Sugere-se a solução do polinômio de grau três apresentado acima através do método de Cardano.

Obtidas as raízes, escolhe-se então uma delas e prossegue com a equação:

$$u = \frac{\pm_s\sqrt{\alpha + 2.y}\pm_t \sqrt{-\left(3.\alpha + 2.y \pm_s \frac{2.\beta}{\sqrt{\alpha + 2.y}}\right)}}{2}$$

Onde se obtém, portanto, quatro possíveis soluções para u.

### 2.3.7 Quarténios

Um quatérnio pode ser pensado como um vetor com quatro componentes, como um composto de um escalar e um vetor comum, ou como um número complexo com três partes imaginárias.

Dado dois números na forma quatérnios [43, 44]:

$$\dot{q} = q_0 + iq_x + jq_y + kq_z$$

e

$$\dot{r} = r_0 + ir_x + jr_y + ir_z$$

Onde $\dot{q}$ e $\dot{r}$ são os quatérnios, $q_0$ e $r_0$ é a parte escalar e os termos $iq_x, jq_y, kq_z$ e $ir_x, jr_y, ir_z$ são a parte complexa. Essa parte complexa também pode ser compreendida como um vetor, portanto, $q_v$ e $r_v$.

A **adição** pode ser dada na forma:

$$\dot{r} + \dot{q} = [q_0, q_v] + [r_0, r_v]$$

Portando:

$$\dot{r} + \dot{q} = (q_0 + r_0) + i(q_x + r_x) + j(q_y + r_y) + k(q_z + r_z)$$

A **subtração** pode ser dada na forma:

$$\dot{r} - \dot{q} = [q_0, q_v] - [r_0, q_v]$$

Portanto:

$$\dot{r} + \dot{q} = (q_0 - r_0) + i(q_x - r_x) + j(q_y - r_y) + k(q_z - r_z)$$

A **multiplicação** pode ser dada pela equação [43, 44]: (43) (44)

$$\dot{r}\dot{q} = (r_0 q_0 - r_x q_x - r_y q_y - r_z q_z)$$
$$+ i(r_0 q_x + r_x q_0 + r_y q_z - r_z q_y)$$
$$+ j(r_0 q_y - r_x q_z + r_y q_0 + r_z q_x)$$
$$+ k(r_0 q_z + r_x q_y + r_y q_x - r_z q_0)$$

A **multiplicação** de um quatérnio por um **escalar** *w*, pode ser dada na forma:

$$\dot{q}.w = [w.q_0, w.q_v]$$

Portanto:

$$\dot{q}.w = w.q_0 + i(w.q_x) + j(w.q_y) + k(w.q_z)$$

## 2.4 Manipulação Geométrica de Pontos

O modelo geométrico de um objeto deve ser de fácil manipulação ou transformação. Transformar um objeto implica alterá-lo em qualquer posição, orientação ou forma. Uma vez que um modelo geométrico deve ser armazenado em um computador como dados numéricos em relação a algum sistema de coordenadas, temos de descobrir maneiras de transformar os dados para representar mudanças como a posição do objeto e orientação [45].

A manipulação de pontos, como a translação ou rotação de um ou mais pontos podem ser dados por matrizes ou ainda, por transformação desses pontos em quatérnios. Três ou mais pontos podem, ainda, além das transformações rígidas, sofrer uma transformação denominada escala, a qual também pode ser manipulada e calculada por matrizes de transformação [45].

### 2.4.1 Matrizes de Transformação

As transformações mais simples também podem ser chamadas de transformações de corpo rígido. Podemos operar diretamente sobre a representação paramétrica de objetos geométricos, como pontos, vetores, curvas e superfícies, para efetivar essas alterações. Nesse tópico, é apresentado uma solução usando apenas cálculos envolvendo matrizes [45].

Equações lineares devem ser estabelecidas para a transformação de objetos geométricos como translação, escala e rotação. Essas equações lineares podem ser formuladas sob a forma de matrizes e utilizadas para cada ponto P de uma imagem e transformando o mesmo em um novo ponto P' [46].

Dado um ponto P(x, y, z), pode ser calculado o ponto P´(x, y, z) através da transformação dada pela equação 2.16

$$P'(x, y, z) = T.P(x, y, z) \qquad \textbf{(2.16)}$$

#### 2.4.1.1 Matriz de Translação

A translação pode ser calculada aplicando o ponto a ser deslocado, e o vetor de translação (dx, dy, dz) no cálculo envolvendo matriz descrita na equação 2.17:

$$\underbrace{\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}}_{P'} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{T_t} \cdot \underbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}_{P} \qquad \textbf{(2.17)}$$

Nesse cálculo, o ponto original *P* é multiplicado pela matriz de transformação $T_t$, onde $d_x$ é o valor de deslocamento no sentido do eixo *x*, $d_y$ é o valor de deslocamento no sentido do eixo *y*,e $d_z$ é o valor de deslocamento na direção do eixo z. $P'$ é o ponto final, ou seja, o ponto depois da transformação.

### 2.4.1.2 Matriz de Escala

A escala pode ser calculada aplicando o ponto a ser transformado, e o vetor de escala na matriz demonstrada na equação 2.18 [46, 45].

$$T_s = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \textbf{(2.18)}$$

Da mesma forma que na translação, o ponto original, *P*, é multiplicado pela matriz de transformação $T_s$, onde nessa, $S_x$ é o valor de escala na direção do eixo *x*, $S_y$ é o valor de escala na direção do eixo *y*,e $S_z$ é o valor de escala na direção do eixo z. $P'$ é o ponto final, ou seja, o ponto depois da transformação, onde:

$S_x$, $S_y$, $S_z$ > 1 significa ampliação.

$S_x$, $S_y$, $S_z$ < 1 significa redução.

$S_x$, $S_y$, $S_z$ = -1 significa espelhar, ou seja, inversão da imagem em relação a algum dos eixos x, y ou z.

### 2.4.1.3 Matriz de Rotação

A rotação pode ser calculada aplicando o ponto a ser transformado, na matriz referente ao eixo que se pretende girar (x, y ou z) aplicar o ângulo e então, calcular a transformação de acordo com a equação 2.16.

Da mesma forma que na translação, o ponto original, *P*, é multiplicado pela matriz de transformação $T_R$, onde nessa, $\vartheta_x$ é o valor do ângulo de rotação sobre o eixo *x*, $\vartheta_y$ é

o valor do ângulo de rotação sobre o eixo *y*,e $\vartheta_z$ é o valor do ângulo de rotação sobre o eixo *z*. $P'$ é o ponto final, ou seja, o ponto depois da transformação [45, 46].

Sobre o eixo x:

$$T_{Rx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\vartheta_x & -\sin\vartheta_x & 0 \\ 0 & \sin\vartheta_x & \cos\vartheta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Sobre o eixo y:

$$T_{Ry} = \begin{pmatrix} \cos\vartheta_y & 0 & \sin\vartheta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\vartheta_y & 0 & \cos\vartheta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Sobre o eixo z:

$$T_{Rz} = \begin{pmatrix} \cos\vartheta_z & -\sin\vartheta_z & 0 & 0 \\ \sin\vartheta_z & \cos\vartheta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

### 2.4.2 Quatérnios

Outra forma de realizar uma transformação, na forma de rotação é através de operações com quatérnios.

Dado um ponto P(x, y, z), podemos transformá-lo em um quatérnio na forma[43, 44]:

$$\dot{r} = r_0 + iz_x + jz_y + iz_z \qquad \textbf{(2.19)}$$

Podemos determinar um eixo de rotação qualquer (x, y, z), e um ângulo de rotação Θ, e calcular a partir desses, o quatérnio de rotação através da fórmula apresentada na equação 2.20. [43, 44]:

$$\dot{q} = \left(cos\left(\frac{\theta}{2}\right); sin\left(\frac{\theta}{2}\right).(i.x; i.y; i.z)\right) \qquad \textbf{(2.20)}$$

E o seu conjugado, pela fórmula:

$$\bar{q} = q_0 - iq_x - jq_y - ik_z \qquad \textbf{(2.21)}$$

A rotação pode ser dada pela fórmula:

$$\dot{r'} = \dot{q}.\dot{r}.\dot{q}^* \qquad \textbf{(2.22)}$$

O quatérnio resultante dessa multiplicação será também da forma:

$$\dot{r} = w + x.i + y.i + z.i \qquad \textbf{(2.23)}$$

O ponto rotacionado, será portanto, o ponto P´(x, y, z).

A multiplicação de quatérnios segue uma forma especifica e de certa complexidade. Essa matemática envolvendo quatérnios pode ser encontrada no Capítulo 2.3.7.

## 2.5 Alinhamento de Imagens 3D

Uma etapa importante desse trabalho é o alinhamento de imagens 3D. Não se percebe, mas nosso cérebro compara imagens no nosso dia a dia e reconhece regiões semelhantes e diferentes entre elas. O mesmo método é aplicado nesse trabalho. Para realizar os cálculos matemáticos de forma menos complexa, alinhamos previamente as imagens 3D, o que possibilita a próxima etapa, a de reconhecimento de regiões diferentes, o que interpreta como rebarbas. A Figura 2.12 ilustra o processo de alinhamento.

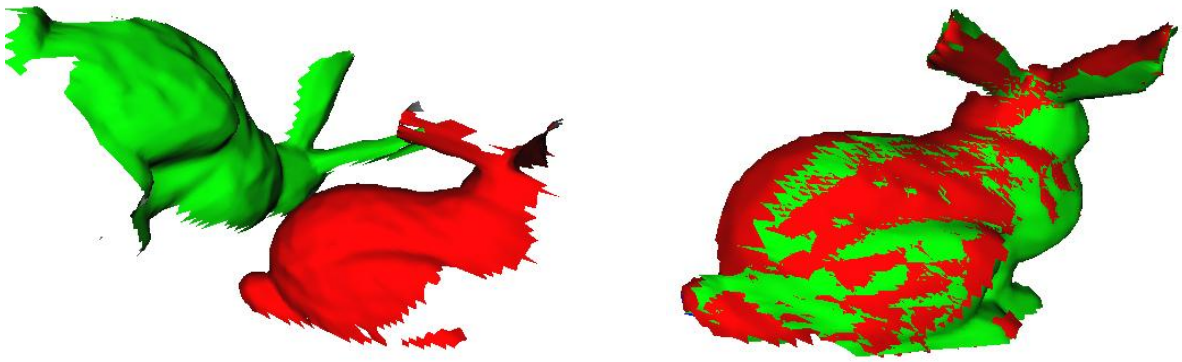


***Figura 2.12: Alinhamento de Imagens 3D***

Através de correspondência de superfície, um objeto pode ser reconhecido em uma cena comparando um escaneamento superficial a uma superfície de objeto armazenado na memória. Quando a superfície do objeto corresponde à superfície da cena, é feita uma associação entre algo conhecido (o objeto) e algo desconhecido (a cena). Outra aplicação da correspondente superfície é o alinhamento de duas superfícies representadas em sistemas de coordenadas diferentes. Para alinhar as superfícies, é necessário determinar uma transformação dos sistemas de coordenadas entre a superfície e a cena. Alinhamento de superfície tem inúmeras aplicações, incluindo a localização para navegação de robôs e de modelagem de

cenas complexas a partir de visões múltiplas, escaneamentos diversos de uma mesma superfície [47].

Alinhar imagens 3D consiste basicamente em reposicionar todos os pontos dessa imagem em relação a uma mesma origem conhecida, ou ainda, apenas como uma interpretação diferente, posicionarmos a origem em relação aos pontos em um mesmo lugar e direção conhecida.

Vários métodos são descritos na literatura, dentre eles destacam-se o algoritmo proposto por Horn, em seu trabalho em 1986 e outro método, é o conhecido com ICP (Iterative Closest Point) ou ponto interativo mais próximo. Como o método de Horn, como pode-se observar no próximo item desse trabalho, o método necessita termos já previamente conhecidos os pontos que se correlacionam entre as duas imagens 3D, um método descrito Johnson em 1997 como SpinImages, ou imagens geradas por rotação através de um eixo, o qual é possível identificar pontos semelhantes em duas imagens distintas 3D é também descrito e utilizado nesse trabalho.

### 2.5.1 Algoritmo de Horn

Berthold K. P. Horn, em 1986 publicou um trabalho chamado *Closed-form solution of absolute orientation using unit quartenions*, que seria uma solução fechada por assim dizer de orientação absoluta usando quarténio. A vantagem desse método para esse trabalho é que pode ser realizada a orientação de imagens 3D, não importando a sua posição ou orientação inicial. Portanto, as imagens podem estar totalmente desorientadas entre si, com direção e sentidos totalmente opostos, e em posições mais distantes imagináveis, porém o método proposto por Horn, e descrito e utilizado nesse trabalho, orienta as duas imagens de forma perfeita, salvo que precisão dessa orientação depende de alguns fatores que serão descritos mais adiante [43].

De forma inicial e simplificada, o processo consiste em um método que Horn desenvolveu para a solução do problema dos mínimos quadrados para três ou mais pontos, de forma simplificada utilizando unidade quarténio para representar rotação. Horn ressalta ainda que a propriedade de simetria é importante para o sucesso desse método. A melhor translação se dá pela diferença entre os centróides das malhas em um sistema, onde centróides são as coordenadas centrais de cada imagem ou malha de pontos. A rotação e escala são calculadas em outro sistema.

Como nesse trabalho não utiliza escala, fica aqui ressaltado que Horn também sugere uma solução para escala, porém não descrito nesse trabalho.

O quatérnio unitário que representa a melhor rotação é o autovetor associado ao maior autovalor positivo de uma matriz simétrica 4 X 4. Autovalores e autovetores de uma matriz simétrica são melhores descritos no capítulo 2.3.3. Os elementos desta matriz são combinações de somas de produtos de correspondentes às coordenadas dos pontos.

**Descrição do Método**

Suponha que são dadas as coordenadas de um número de pontos medidos em dois diferentes sistemas de coordenadas cartesianas c e s, como mostra a Figura 2.13. A solução dessa transformação é denominada como orientação absoluta [43].

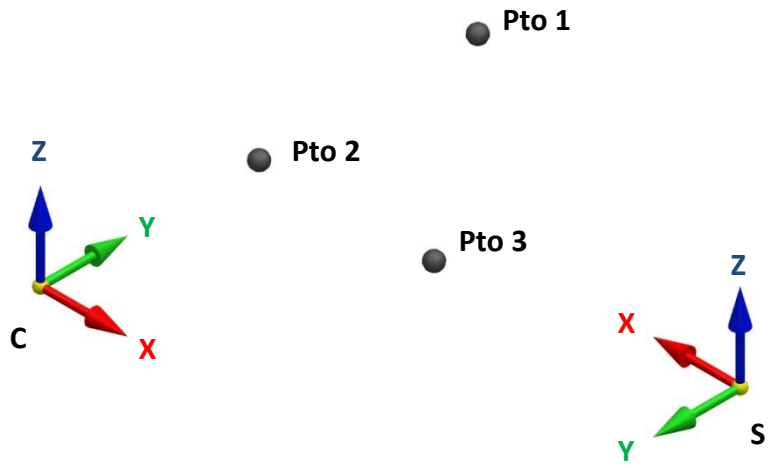



***Figura 2.13: Coordenadas dos pontos mensuradas em dois diferentes sistemas de coordenadas***

A transformação entre dois sistemas de coordenadas cartesianas pode ser pensada como o resultado de um movimento de corpo rígido e assim pode ser decomposto em uma rotação e uma translação [43].

Existem em relação à translação três graus de liberdade. Em relação a rotação, mais três graus, sendo a direção de cada eixo mais o ângulo de rotação. E a escala, acrescenta mais um grau de liberdade.

Três pontos conhecidos nos dois sistemas de coordenadas fornecem nove restrições (três coordenadas de cada), o que são mais do que suficientes para permitir a determinação das sete variáveis desconhecidas [43].

Portanto, pode se concluir que três é o número mínimo de pontos que cada malha deve possuir para que se possa determinar uma transformação em relação a essas malhas. Pode se concluir também que, três é o número suficiente de pontos necessários para calcular a transformação de quaisquer malhas, não importando o número de pontos delas [43].

Para determinar a translação, basta calcular os centróides de cada uma das imagens 3D e encontrar a diferença entre essas coordenadas. Essa diferença será a melhor translação. O problema está em descrever um método que encontre a melhor rotação.

Horn mostra em seu trabalho que para tornar o problema da rotação menos complexo, deve-se calcular os centróides das malhas, e então recalcular as coordenadas dos pontos a partir desses centróides.

Para encontrar os centróides, a equação a seguir descreve a solução do problema, onde essa consiste basicamente em somar as coordenadas dos pontos, e dividir o resultado pelo número de pontos da malha.

$$c_k = \frac{1}{n}\sum_{i=1}^{n} p_{k,i}$$

Sendo:

$c_{k:}$: é a coordenada X, Y ou Z do centróide

n: o número de pontos da malha

$p_{k,i,}$: O ponto onde k representa x, y ou z, e i o ponto a ser calculado.

Em seguida, devem ser calculadas as novas coordenadas de cada ponto em relação ao centróide da malha ao qual esse ponto corresponde. Dessa forma, os pontos da malha c devem ser subtraídos o valor do centróide da malha c e os pontos da malha s devem ser subtraídos o valor do centróide da malha s. A equação a seguir resolve o problema.

$$P'_{k,i} = P_{k,i} - c_k$$

Sendo as coordenadas expressas em termos dos centróides das malhas c e s, o nosso sistema pode se reduzir a determinar a rotação R das coordenadas P do referencial c para as coordenadas Q do referencial s dos pontos i:

$$Q_i = R(P_i)$$

Horn mostrou que ao expressar esta matriz de rotação como um quatérnio, pode-se reduzir o problema a determinar o autovetor associado ao maior autovalor da matriz simétrica 4x4. Para calcular os elementos dessa matriz, primeiro calcula-se a matriz M, cujo elemento (i, j) é a soma dos produtos das coordenadas das duas malhas, c e s.

$$M = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix}$$

Cada termo deve ser calculado como, por exemplo, na equação abaixo:

$$S_{kl} = \frac{1}{n}\sum_{i=1}^{n} P'_{k,i,c} \times P'_{k,i,s}$$

Sendo:

kl: Representam as coordenadas x, y e z.

n: Número de pontos da malha

$P'_{k,i,c}$: Ponto referencial na malha c.

$P'_{k,i,s}$: Ponto referencial na malha s.

Para calcular os demais termos da matriz M, basta substituir na equação acima os termos a serem calculados, como por exemplo, Sxy, o qual o primeiro termo da multiplicação da somatória será $X_{c,i}$ e o segundo $Y_{s,i}$ e assim por diante até o termo Szz.

Em seguida, deve ser calculada a matriz N, através dos elementos calculados na matriz M:

$$N = \begin{bmatrix} (S_{xx}+S_{yy}+S_{zz}) & S_{yz}-S_{zy} & S_{xx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & (S_{xx}-S_{yy}-S_{zz}) & S_{xy}-S_{yx} & S_{zx}-S_{xz} \\ S_{xx}-S_{xz} & S_{xy}-S_{yx} & (-S_{xx}+S_{yy}-S_{zz}) & S_{yz}-S_{zy} \\ S_{xy}-S_{yx} & S_{zx}-S_{xz} & S_{yz}-S_{zy} & (-S_{xx}-S_{yy}+S_{zz}) \end{bmatrix}$$

O autovetor associado ao maior autovalor dessa matriz N representa o quatérnio de melhor rotação.

Para calcular o maior autovalor dessa matriz, precisamos calcular os quatro autovalores possíveis dessa matriz.

Para determinar os autovalores, deve-se resolver a equação abaixo, onde N é a matriz N, $\lambda$ são os possíveis autovalores que desejamos calcular, e I é a matriz identidade 4x4.

$$det(N - \lambda I) = 0$$

O desenvolvimento dessa equação nos dá um polinômio de grau quatro:

$$x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = 0 \qquad \textbf{(2.24)}$$

A equação tem, portanto quatro raízes para γ. [Conforme descrito no capítulo 2.3.3, sobre autovalores e autovetores, a matriz N, por ser uma matriz simétrica 4 x 4, possui quatro autovalores].

O desenvolvimento dessa equação não é simples, porém, Horn descreve em seu trabalho equações prontas para cada coeficiente desse polinômio de grau quatro. São elas:

$$c_3 = a + b + c + d$$

$$c_2 = -2\left(S_{xx}{}^2 + S_{xy}{}^2 + S_{xz}{}^2 + S_{yx}{}^2 + S_{yy}{}^2 + S_{yz}{}^2 + S_{zx}{}^2 + S_{zy}{}^2 + S_{zz}{}^2\right)$$

$$c_1 = -8.det(M)$$

$$c_0 = det(N)$$

Onde em $c_3$ os coeficientes a, b, c e d são a representação da matriz N na seguinte forma:

$$N = \begin{bmatrix} a & e & h & j \\ e & b & f & i \\ h & f & c & g \\ j & i & g & d \end{bmatrix}$$

A solução do polinômio $x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = 0$ discutido na $x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = 0$ (2.24) pode ser resolvida pelo método de Ferrari, descrito no capitulo 2.3.6.

Obtidos esses quatro autovalores, encontra-se agora o maior autovalor positivo.

Para encontrar agora o autovetor $\dot{e}_m$ associado à esse maior autovalor positivo $\lambda_m$, desenvolve-se a equação abaixo:

$$[N - \lambda_m I]\dot{e}_m = 0$$

sendo:

N: Matriz N

$\lambda_m$: Maior autovalor positivo da matriz N

I: Matriz identidade

$\lambda_m$.I: É a multiplicação da matriz identidade pelo maior autovalor da matriz N

$\dot{e}_m$ : É o autovetor que se deseja encontrar.

Ao desenvolver essa equação, encontra-se em um sistema linear homogêneo de quatro equações. Lembrando que sistemas lineares homogêneo possuem sua solução trivial como sendo zero para todas as incógnitas, e o que se pretende calcular é uma solução não trivial.

O autovetor resultante desses cálculos será na forma:

$$\dot{e}_m = q_o + iq_x + jq_y + kq_z$$

O quatérnio que representa a melhor rotação é o vetor unitário no sentido do vetor $\dot{e}_m$. Deve-se, portanto, normalizar $\dot{e}_m$.

Calcula-se agora a rotação para cada ponto da malha que se deseja realizar as transformações. A equação que resolve matematicamente a rotação na forma de quatérnios é descrita abaixo, como se pode observar no item sobre quatérnios, na seção 2.3.7, essa multiplicação também não é simples. Deve se obedecer a regra de multiplicação envolvendo números na forma de quatérnios.

$$P' = q.P.q^{-1} \leftrightarrow P' = R(P)$$

O cálculo dessa rotação deve ser realizado com as novas coordenadas dos pontos, encontradas a partir dos centróides. Então, o procedimento para realizar agora a rotação, é subtrair o centróide do ponto que se pretende transformar, calcular a rotação aplicando o quatérnio encontrado e adicionar novamente o centróide no ponto transformado Após esses cálculos tem-se as malhas alinhadas no sentido da rotação, faltando apenas agora, a translação.

Para a realização da translação, deve-se agora calcular o vetor que indica essa translação, ou de uma forma mais simples, calcula-se a distância entre os centroides das duas malhas. A equação abaixo resolve o vetor de translação:

$$T_k = c_{k,s} - c_{k,c}$$

sendo:

$T_k$ : Vetor de translação

k: Os componentes x, y e z do vetor T

$c_{k,s}$ : Centróide da malha um.

$c_{k,s}$ : Centróide da malha dois.

Agora, para realizar a translação aos pontos já rotacionados, deve ser aplicada a fórmula abaixo:

$$P''_{k,i} = P'_{k,i} - T_k$$

Sendo:

P´´: Ponto após sua translação

P´: Ponto após sua rotação

T: Vetor de translação.

**Conclusão do Método**

Pode-se, portanto, resumir o algoritmo da seguinte forma:

Primeiro calcula-se os centróides $c_c$ e $c_s$ das duas malhas de pontos

Os centróides agora são subtraídos das coordenadas dos pontos das devidas malhas e agora, os cálculos são apenas com as coordenadas relacionadas aos centróides das malhas.

Para cada par de coordenadas conhecidas, computam-se os nove possíveis produtos dos dois vetores de pontos e adicionamos para se obter Sxx, Sxy ... Szz, que é a matriz M. Isso significa que $q^t Nq$ deve ser maximizado de forma a minimizar o erro da transformação.

Esses nove resultados contêm toda informação necessária para se encontrar a solução.

Agora calcula-se os 10 elementos independentes que formam a matriz 4 x 4 simétrica denominada de N através da combinação por soma dos valores obtidos na matriz M.

A partir destes elementos, por sua vez, calcula-se os coeficientes de o polinômio de quarta ordem que tem de ser resolvido para obter os autovalores da matriz N. Horn sugere o método de Ferrari.

O autovetor normatizado associado ao maior autovalor correspondente a essa matriz 4x4 é o quatérnio que representa a rotação.

Nesse ponto, caso as malhas estivessem em escalas diferentes, poderia ser calculado a escala das malhas, o qual não é utilizado nesse trabalho.

Finalmente é computada a translação das malhas, com o sendo a diferença entre os centróides das duas malhas.

Segundo Horn, embora estes cálculos são triviais, ou seja, conhecidos, não há aproximação envolvida, portanto sem necessidade de correção iterativa. Ou seja, não há a necessidade de repetir o processo para um melhor alinhamento.

Quanto à precisão, esse método é preciso quando são dadas malhas perfeitamente idênticas, onde os pontos correspondentes nas duas malhas são perfeitamente casados.

### 2.5.2 ICP – Interative Closest Point

Este algoritmo é responsável pelo refinamento do alinhamento entre duas superfícies previamente alinhadas. Nesse trabalho, devido à forma em que os pontos fornecidos para o método de Horn foram selecionados, não se faz necessário o uso do ICP.

O algoritmo ICP (Iterative Closest Point) é a técnica mais utilizada atualmente para realizar registro de superfícies. Dadas duas malhas pré-alinhadas, o ICP executa um processo iterativo para refinar a transformação inicial até convergir para um mínimo local. Ao final deste processo, as duas malhas com alinhamento estimado terão um alinhamento mais preciso [48].

O objetivo do ICP é das duas malhas de pontos C e S, encontrar o movimento rígido (R, t) que minimiza o erro de ajuste.

A Figura 2.14 ilustra o processo de alinhamento por ICP.

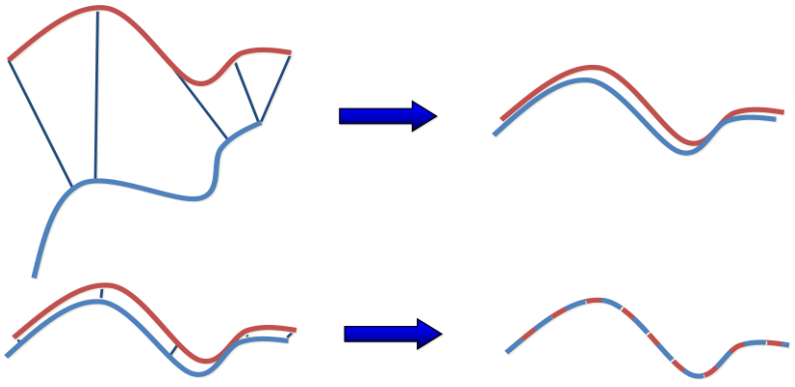


***Figura 2.14: ICP interatividade***

ICP funciona melhor quando todo ponto em C corresponde a algum ponto em S (não é o caso no alinhamento de malhas obtidas por fotografia 3D).

Nos últimos anos, muitas variantes desse algoritmo foram desenvolvidas, afetando todos os estágios do procedimento. Segundo [24] o algoritmo ICP pode ser dividido em seis estágios:

1. Seleção de pontos em uma ou nas duas malhas;
2. Geração de correspondências entre os pontos selecionados;
3. Atribuição de pesos nos pares de pontos;
4. Rejeição de pares de acordo com algum critério;
5. Determinação de uma métrica de erro;
6. Minimização da métrica de erro.

### 2.5.3 SpinImages

Correspondência de superfície é um processo que determina quando as formas de dois objetos diferentes estão congruentes. Correspondência de superfície é uma técnica de visão por computador em 3D que tem muitas aplicações na área de robótica e automação [47].

Johnson (1997) em seu trabalho, descreve um método para reconhecimento de pares de pontos em duas malhas distintas.

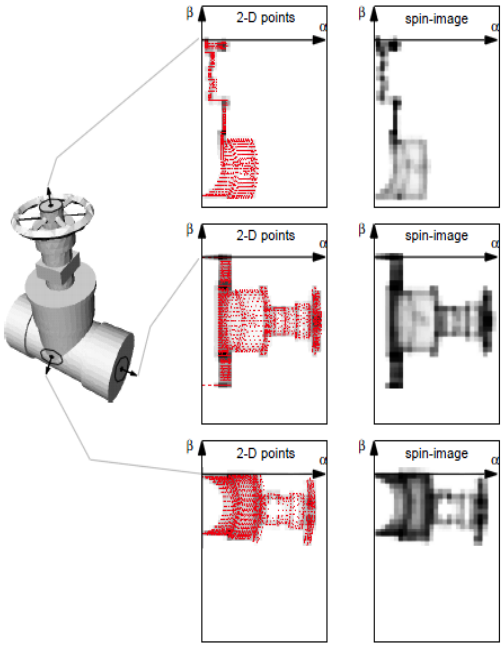



***Figura 2.15: SpinMaps e SpinImages***

De uma forma inicial e simplificada, o processo descrito por Johnson(1997) baseia-se em gerar imagens, igual a ilustração mostrada Figura 2.15. Nessa figura, o

modelo triangularizado, ou seja, o arquivo STL ou imagem 3D se encontra ao lado esquerdo. A seqüência de imagens no centro da figura, em vermelho, apresentam imagens geradas apenas por pontos denominadas como SpinMaps, e na direita, em preto, imagens geradas em escala de cinza, formadas por pixels onde sua escala de cinza, pode variar dependendo de quantos pontos contém cada pixel.

Após esse procedimento, é calculado um coeficiente de correlação entre as imagens da malha um com a malha dois.

Para a geração dos SpinMaps, a transformação abaixo deve ser aplicada:

$$S_0 : R^3 \rightarrow R^2$$

Essa transformação se resume em transformar as coordenadas dos pontos que estão em um espaço de três dimensões, para um de duas dimensões, ou seja, transformar os pontos em uma imagem.

Para essa transformação, Johnson sugere o equacionamento:

$$S_0(x) \rightarrow (\alpha, \beta) = \left( \sqrt{\|x - p\|^2 - \big(n.(x - p)\big)^2}, n.(x - p) \right) \qquad \textbf{(2.25)}$$

Sendo:

$S_0$: SpinMap correspondente ao ponto orientado p.

X: O ponto em que se pretende projetar no SpinMap

$\alpha$: Distância perpendicular ao vetor normal n

$\beta$: Distância perpendicular ao plano P

p: Ponto de referencia do SpinMap

n: Vetor normal ao ponto P

A equação 2.25 descreve que uma imagem será gerada pela rotação de um plano em torno do vetor normal à superfície, onde cada ponto deixará inscrito um novo ponto nesse plano.

A Figura 2.16 apresenta as variáveis da equação 2.25 e sua representação gráfica:

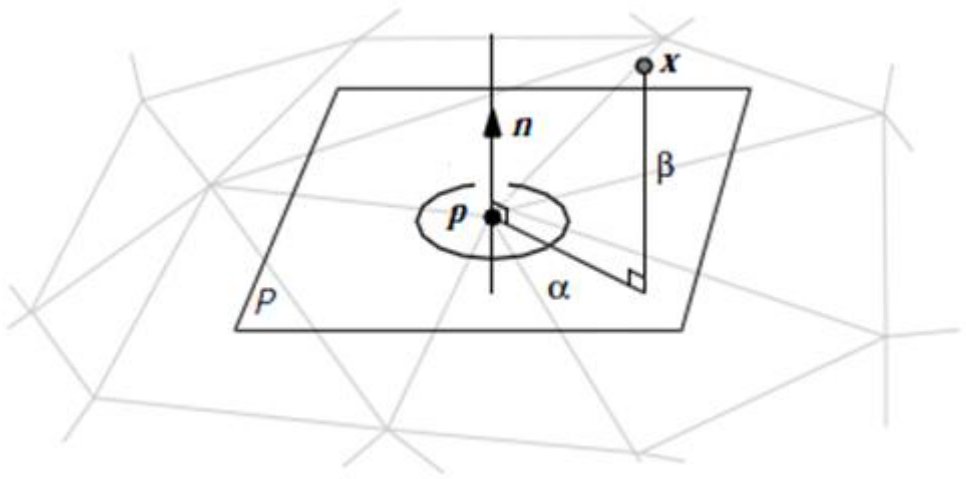



***Figura 2.16: SpinMaps [28]***

Para a transformação seguinte, dos SpinMaps em SpinImages, deve ser aplicado o método que se segue:

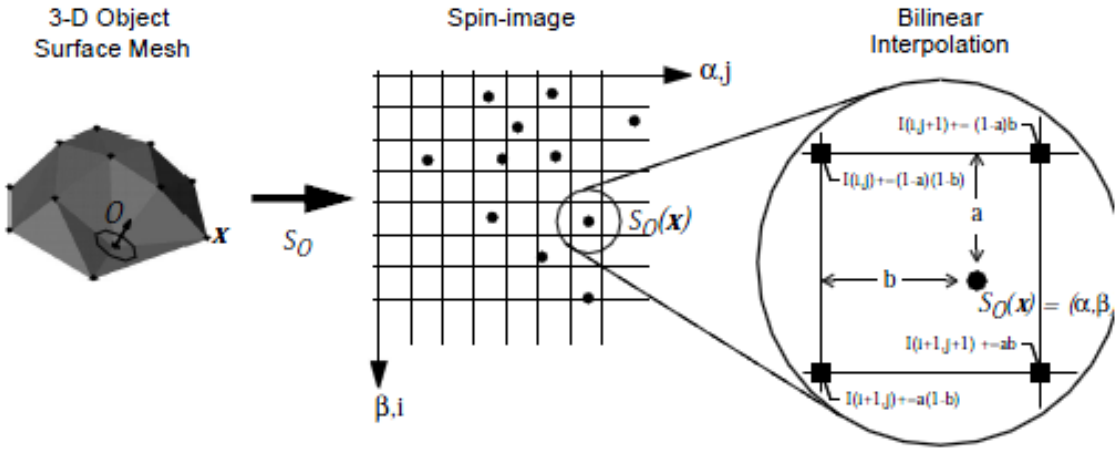



***Figura 2.17: SpimImages [47]***

As equações seguintes sevem para determinar em qual pixel cada ponto pertence:

$$i = \left\lfloor \frac{\frac{w}{2} - \beta}{b} \right\rfloor$$

$$j = \left\lfloor \frac{\alpha}{b} \right\rfloor$$

Sendo:

w: Tamanho da imagem.

b: Tamanho do pixel

Depois de calculadas as imagens, deve ser encontrado o fator de correlação de cada imagem da malha um em relação às imagens da malha dois.

A equação que realiza essa correlação é representada pela equação 2.26:

$$R(P,Q) = \frac{N.\sum pi.qi - \sum pi.\sum qi}{\sqrt{(N.\sum pi^2 - (\sum pi)^2).(N.\sum qpi^2 - (\sum qi)^2)}} \qquad \textbf{(2.26)}$$

Em que o coeficiente R(P,Q), que pode variar entre -1 e 1, onde -1 é menos correlativo e 1 é mais correlativo.

# 3 Objetivos e Metodologia

Modelos matemáticos gerados em sistemas CAD possuem um sistema de coordenadas com sua origem conhecida, e posicionada de forma a favorecer a sua modelagem, visualização e utilização do modelo matemático. Modelos matemáticos gerados por processos como scaner 3D, sua origem é conhecida, porém, em posicionamento diferente da do sistema CAD, o que torna inviável para cálculos matemáticos como comparações.

Dessa forma, antes do processo de reconhecimento de rebarbas, necessita ser feito um alinhamento das duas imagens, ou modelos matemáticos, de forma a que ambos possuam a mesma origem.

## 3.1 Objetivo

Desenvolver um método capaz de identificar automaticamente rebarbas em peças fundidas por meio da comparação dos dados no formato de arquivo STL da peça fundida com o modelo CAD.

## 3.2 Metodologia

Com o propósito de atingir aos objetivos neste trabalho, a seguinte metodologia foi estabelecida:

- Pesquisa bibliográfica para aprofundamento no tema, de forma a justificar a necessidade desse trabalho e poder contribuir de forma mais eficiente no processo de rebarbação.
- Seleção do método mais adequado para realizar o alinhamento das duas imagens 3D, onde através de pesquisa na literatura, a seleção do método descrito por Horn, 1987 foi o mais indicado. Para a aplicação desse método, aprofundamentos matemáticos em cálculos vetoriais, quatérnios e resolução de polinômios de grau três e quatro são essenciais.
- Pelo fato do método de Horn possuir algumas restrições, como a entrada de dados, que tem que ser pontos semelhantes nas duas imagens, e aos pares, uma pesquisa na literatura indicou o método descrito por Johnson, 1997 descrito como SpinImages, o qual é capaz de identificar pontos semelhantes entre as duas Imagens. Porém, problemas relacionados a peças simétricas e número de pontos das malhas tornaram o método de Johnson inviável. Esse método pode sim ser aplicado, mas precisam ser estudadas soluções para o

problema de simetria das malhas e número de pontos. Portanto, para a entrada de dados no método de Horn, foi escolhido uma seleção manual dos pontos destinados aos cálculos do alinhamento.

- Outro método apresentado é o ICP (Iterative Closest Point). Esse método não é utilizado como método inicial, pois para ser possível seus cálculos, os modelos matemáticos devem estar previamente alinhados. Ele serve para um refinamento do alinhamento. Nesse trabalho não houve essa necessidade, uma vez que a entrada de dados é uma seleção manual de pontos.
- Realizado o alinhamento, pode-se agora, realizar os cálculos que visam os reconhecimentos das possíveis irregularidades como rebarbas.
- Implementação do método em linguagem JAVA, possibilitando assim testes reais do método.

A Figura 3.1 apresenta uma visão geral do processo de reconhecimento de rebarbas após a verificação na literatura.

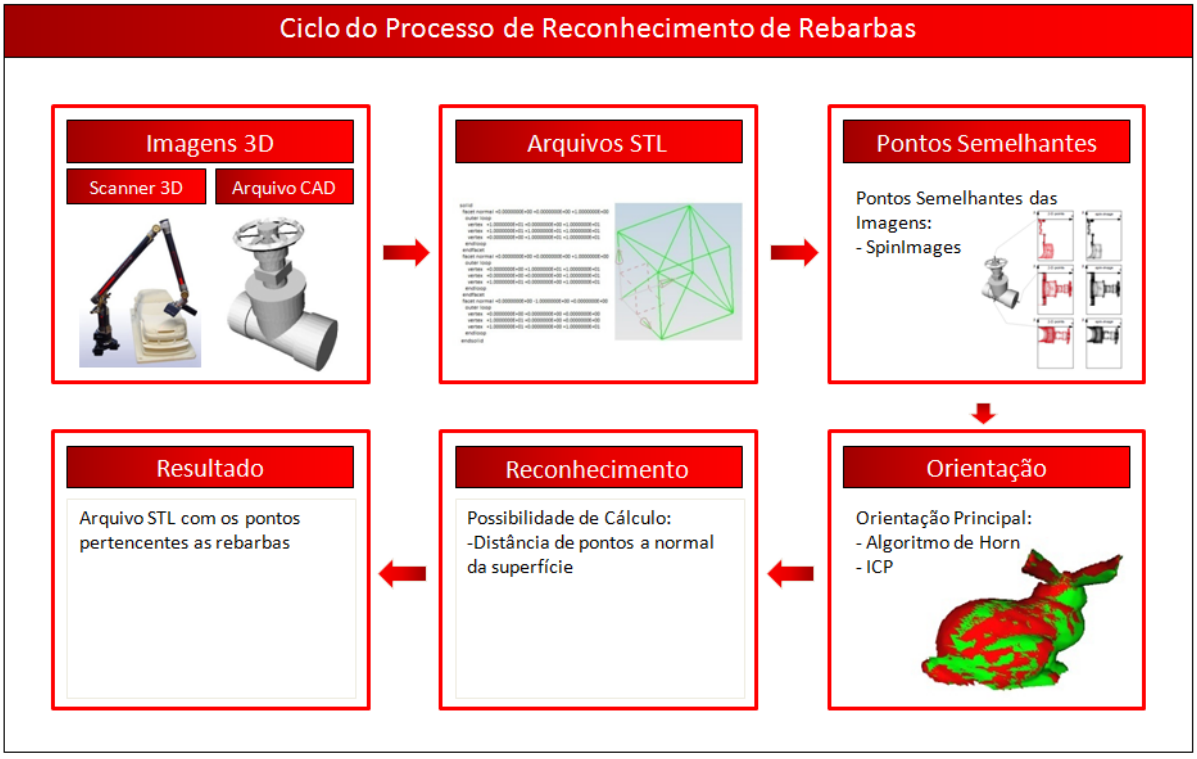
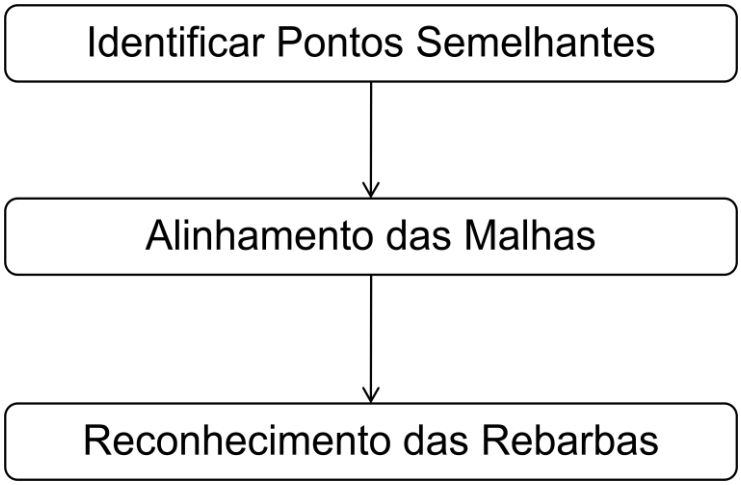



***Figura 3.1: Ciclo do Processo, visão inicial***

# 4 Descrição e Desenvolvimento do Método

Para o inicio do procedimento, deve-se ter previamente os arquivos em formato STL da peça projetada e da peça física escaneada. Para realizar o procedimento, esse pode ser dividido em 3 grandes fases:




Essas fases podem então ser subdivididas 6 em etapas, são elas:

Etapa 1: Identificar pontos semelhantes na malha proveniente do sistema de escaneamento 3D e na malha proveniente do modelo CAD.

Etapa 2: Calcular as transformações entre as malhas. O quatérnio de rotação e o vetor de translação entre as malhas.

Etapa 3: Aplicar a transformação de rotação através do quatérnio de rotação em todos os pontos do modelo escaneado.

Etapa 4: Aplicar a transformação de translação através do vetor de translação em todos os pontos do modelo escaneado.

Etapa 5: Calcular e separar as facetas do modelo escaneado que não pertencem ao modelo CAD.

Etapa 6: Salvar as facetas encontradas na forma de arquivo STL.

**Descrição detalhada das etapas:**

Etapa 1: O responsável pelo modelamento da peça deve identificar pontos estratégicos no modelo, preferencialmente de 5 a 10 pontos, de forma que os mesmos possam ser encontrados na peça fundida. Ou seja, esses pontos não podem ser definidos em regiões onde se esperam ou se encontram defeitos como regiões de fechamento do molde, regiões que possuam drenos ou machos, ou distantes dos canais de respiros ou de vazamento. Também devem ser evitadas

regiões onde possa não ocorrer o preenchimento completo do modelo. Pode-se também, para facilitar o processo, observar o modelo já fundido e encontrar regiões visualmente possíveis de encontrar pontos semelhantes, que não tenham imperfeições relevantes. Regiões prismáticas ou com encontro de planos ou arestas são indicadas, pois são mais fáceis de serem capturados a coordenadas desses pontos nos softwares utilizados pelos escanners. Apesar de ser descrita na revisão bibliográfica uma possibilidade automática através do método de SpinImages, esse não foi utilizado devido a alguns problemas que teriam que ser melhores estudados e propostas soluções. São alguns desses problemas o número diferente de pontos entre a malha proveniente do modelo CAD e da malha escaneada, e também, o problema da simetria, pois quando a peça é simétrica, um mesmo ponto pode gerar correspondências em um lugar diferente do esperado.

Etapa 2: Para realizar os cálculos das transformações, deve-se primeiro calcular os centróides dos pontos adquiridos na etapa 1. Os pontos provenientes do modelo escaneado devem possuir um centróide (x, y, z) e os pontos provenientes do modelo CAD devem possuir um segundo centróide (x, y, z). Encontrados os centróides, deve-se então subtrair as coordenadas de cada centróide do seu modelo específico. Dessa forma, cada modelo estará apenas com a questão da rotação. Deve-se então, aplicar o método de Horn. Deve-se calcular os elementos das matrizes M e N. Calcula-se então o autovetor relacionado ao maior autovalor positivo da matriz N. Esse autovetor é o quatérnio de rotação. Para o cálculo do vetor de translação, subtrai-se o centróide da malha do modelo CAD do centróide da malha escaneada.

Etapa 3: Para aplicar a transformação de rotação encontrada, deve-se para cada ponto do modelo escaneado, subtrair o centróide da malha escaneada, aplicar a rotação e novamente adicionar o centróide da mala escaneada. Como resultado desse procedimento, obtém-se as malhas alinhadas.

Etapa 4: Para aplicar a transformação de translação, subtraí-se agora, de cada pontos do modelo escaneado já aplicada a rotação o valor do vetor de translação. Como resultado, obtém-se a malha escaneada alinhada com a malha do modelo CAD. Apenas com o método de Horn as imagens ficaram alinhadas dentro de uma faixa de tolerância aceitável para a realização do cálculo das rebarbas. Portanto, o método descrito na revisão bibliográfica como ICP não se fez necessário implementar.

Etapa 5: Para calcular as facetas do modelo escaneado que não pertencem ao modelo CAD, deve-se primeiro isolar os pontos do modelo escaneado. Isto é,

originalmente eles estão na forma de triângulos, agrupados a cada três, e ainda, se repetindo em vários outros triângulos. Deve-se portanto, por meio de comparação de coordenadas, isolar apenas os pontos distintos do modelo escaneado.

Para encontrar quais são rebarbas, dois fatores devem ser levados em consideração. Se a projeção do ponto em questão está dentro da área do triangulo que se está comparando no modelo CAD, e no caso de pertencer, se esse ponto está a uma distância maior do que uma tolerância especificada e no sentido do vetor normal. Isso porque, quando contra o sentido do vetor normal, pode ser alguma região que ficou faltando material, e assim, é uma imperfeição na forma de falta de material, e não de sobra como uma rebarba.

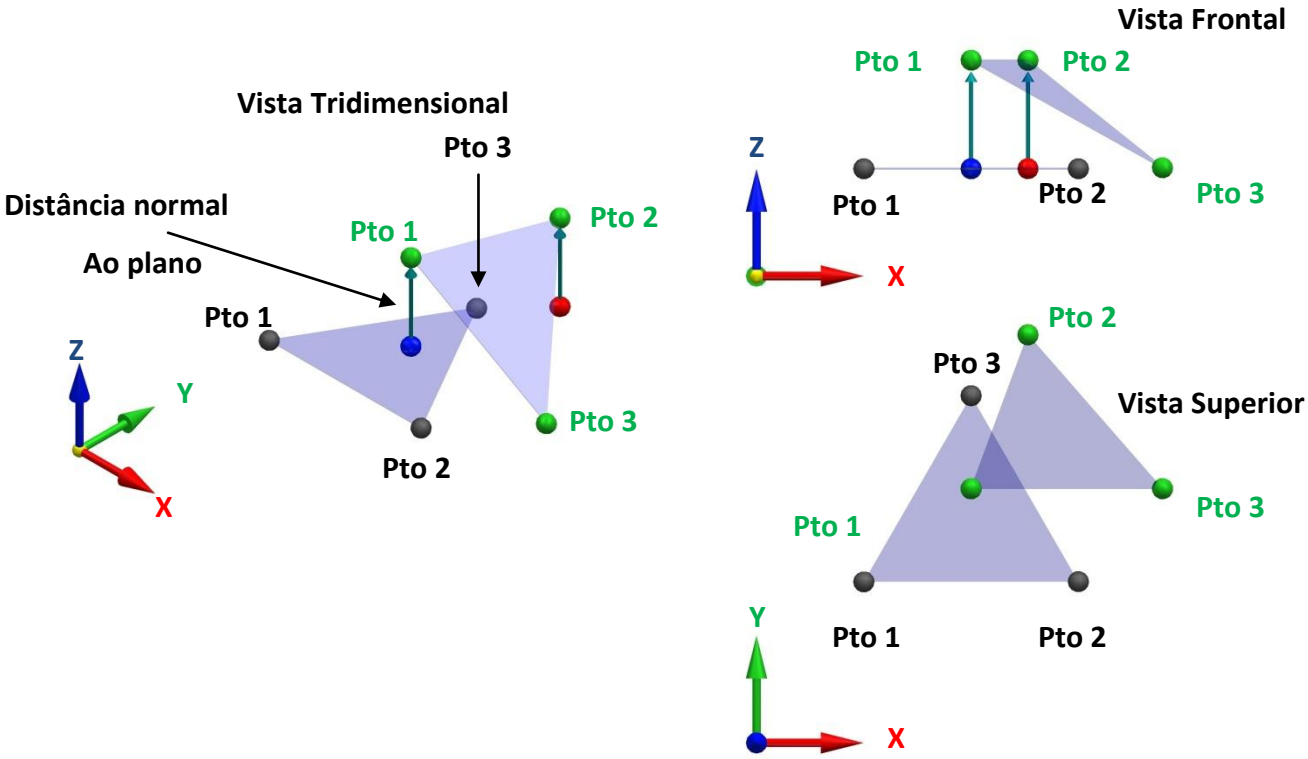



***Figura 4.1: Projeção de pontos modelo escaneado em uma faceta do modelo CAD***

Conforme mostra a Figura 4.1, raramente acontecerá dos três pontos do triângulo do modelo escaneado, em verde, terem projeções em um mesmo triângulo do modelo CAD, em preto. Dessa forma, os três pontos do modelo escaneado podem pertencer ao modelo CAD, em triângulos distintos. Essa também é mais uma razão para calcular os pontos do modelo escaneado individualmente. Pode-se observar também que a projeção do Ponto 1 do modelo escanado, ponto em azul, pertence a área triangular formada pelos pontos um, dois e três do modelo CAD. Portanto, se a distância normal ao plano não tivesse fora da tolerância especificada, esse ponto

para esse triângulo específico não seria uma rebarba. Já a projeção do ponto dois do modelo escaneado, ponto em vermelho, mesmo que a distância normal ao plano não tivesse fora da tolerância especificada, seria para esse triângulo do modelo CAD uma rebarba.

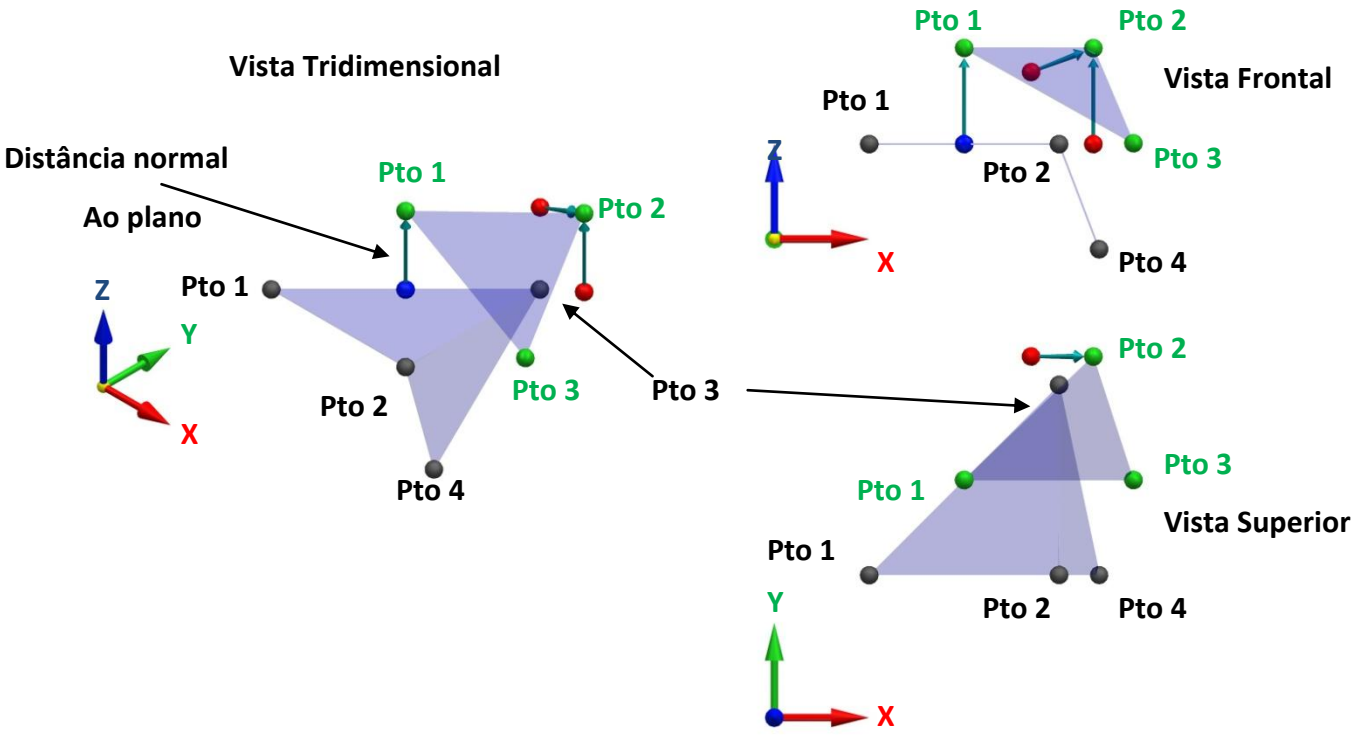



***Figura 4.2: Motivo pelo qual o ponto projetado deve estar dentro do triângulo***

A razão pela qual o ponto projetado deve estar dentro da área do triangulo para averiguar se é ou não uma rebarba, pode ser demonstrado na Figura 4.2. Em relação ao ponto três do modelo escaneado, em verde, sua distância normal ao plano formado pelos pontos do modelo CAD um, dois e três é zero, ou seja, está na tolerância. Mas quando comparamos ao triângulo formado pelos pontos dois, três e quatro, observamos que esse ponto é uma rebarba, pois o formato da peça se altera nesses dois triângulos. Outro caso que pode ser observado nessa figura, é o caso do ponto dois, do modelo escaneado, em verde. Esse não teria projeção em nenhum dois triângulos apresentados na figura, e ainda, supondo que a geometria da peça seria a continuidade desses triângulos, ele não teria projeção em nenhum triângulo do modelo CAD. Nesse caso, não é necessário analisar a distância normal ao plano formado pelos pontos do triângulo, pois uma vez que não há projeção em nenhum dos triângulos do modelo CAD, esse pontos já podem ser considerados como rebarba.

Dessa forma, podemos resumir esses casos da seguinte forma:

**É rebarba se:**

- O ponto do modelo escaneado não possuir nenhuma projeção dentro de algum dos triângulos do modelo CAD.
- O ponto do modelo escaneado possuir projeção em algum dos triângulos do modelo CAD, porém, nos que possuir essa projeção, a distância normal ao plano formado pelos pontos que definem esse triângulo é maior que a tolerância especificada.

**Não é rebarba se:**

- O ponto do modelo escaneado possuir em relação a algum dos triângulos do modelo CAD, a distância normal ao plano formado pelos pontos que definem o triângulo em análise.
- Para calcular a tolerância, devem ser levados em consideração dois parâmetros:
- A tolerância do equipamento usado no processo de escaneamento.
- A exatidão das coordenadas dos pontos fornecidos para o processo de alinhamento.

Para calcular a projeção de um ponto do modelo escaneado a um triângulo específico, deve-se primeiro calcular a distância normal desse ponto ao plano através da equação 4.1**Erro! Fonte de referência não encontrada.**, onde n é o vetor normal ao plano formado pelos três pontos do triângulo do modelo CAD, x é o ponto do modelo escaneado e p um dos três pontos do triângulo do modelo CAD.

$$d = n.(x - p) \qquad \textbf{(4.1)}$$

Nessa equação, ao subtrair do ponto do modelo escaneado qualquer um dos três pontos do modelo CAD, obtém-se um vetor que aponta do ponto do modelo CAD para o modelo escaneado. Ao multiplicar esse vetor pelo vetor normal, resulta no vetor que aponta da normal a faceta do modelo CAD ao ponto do modelo escaneado, não importando qual dos três pontos na faceta do modelo CAD.

Para calcular então, as coordenadas do ponto projetado, basta multiplicar o resultado da equação 4.1 com o vetor normal. Deve-se então, subtrair as coordenadas calculadas x, y e z do ponto do modelo escaneado. Essas novas

coordenadas x, y e z são as coordenadas do ponto projetado no plano. O resultado da equação 4.1 é também, o valor da distância normal ao plano formado pelos três pontos que determinam o triângulo em análise.

Para calcular se esse ponto projetado está ou não dentro do triângulo, podemos fazer uma analogia trigonométrica a Figura 4.3 e a Figura 4.4:

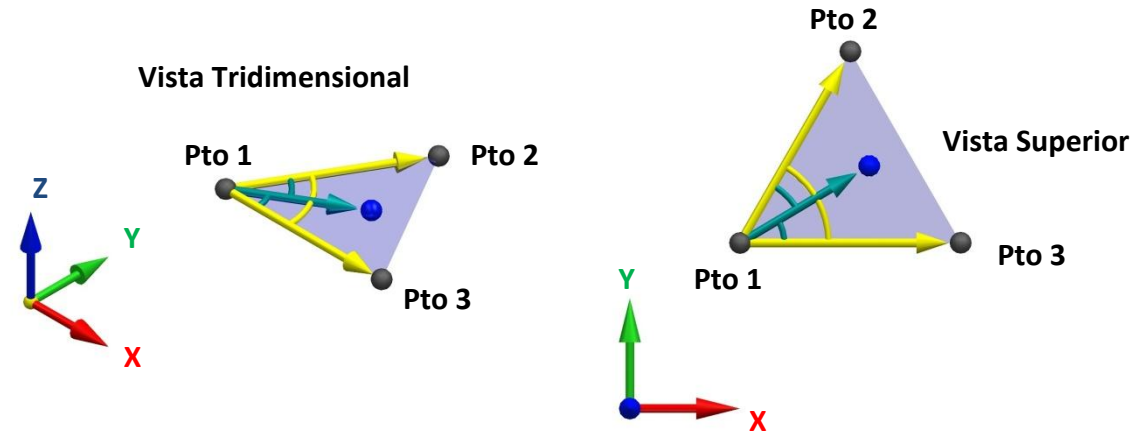



***Figura 4.3: Ponto pertencente ao triângulo***

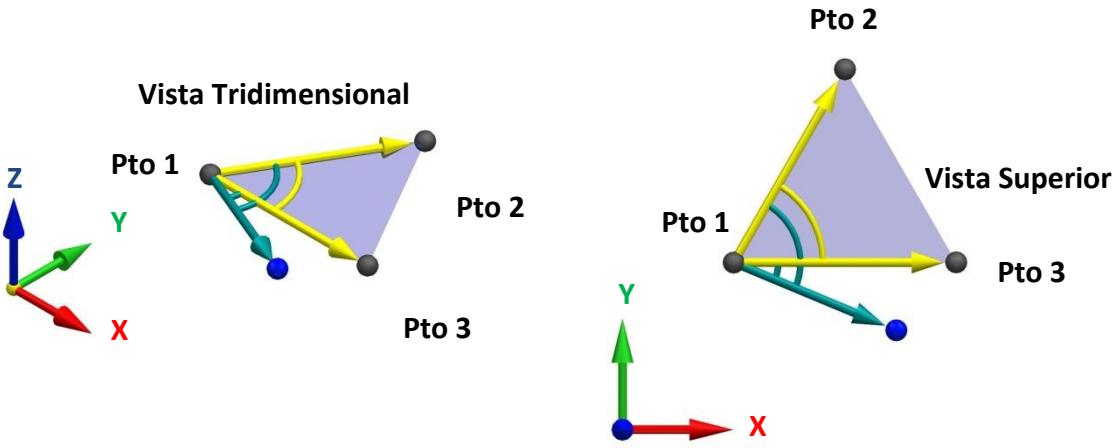



***Figura 4.4: Ponto não pertencente ao triângulo***

Se subtrair o ponto 1 do ponto 2, tem-se um vetor com a mesma direção e sentido ao vetor representado em amarelo nas figuras. Podemos fazer o mesmo em relação aos pontos 1 e 3. Tem-se então, dois vetores, um que aponta ponto 1 ao ponto 2 e outro que aponta ponto 1 ao ponto 3. Se subtrair agora o ponto 1 do ponto projetado em azul, tem-se então um terceiro vetor que aponta ponto 1 ao ponto projetado.

Se o ângulo entre o vetor ponto 1 ao ponto projetado e o ponto 1 ao ponto 2 for menor que o ângulo entre os vetores ponto 1 ao ponto 2 e ponto 1 ao ponto 3, e ainda, o ângulo entre o vetor ponto 1 ao ponto projetado e o ponto 1 ao ponto 3

também for menor que o ângulo entre os vetores ponto 1 ao ponto 2 e ponto 1 ao ponto 3, conforme pode ser observado na Figura 4.3, esse ponto pertence ao triângulo de acordo com o ponto 1.

Conforme pode-se observar na Figura 4.4, o ângulo entre o vetor ponto 1 ao ponto projetado em relação ao vetor ponto 1 ao ponto 3 é maior que o ângulo entre os vetores ponto 1 ao ponto 2 e ponto 1 ao ponto 3. Dessa forma, esse ponto não pertence ao triângulo de acordo com o ponto 1.

Esse procedimento deve ser repetido para os três pontos do triângulo em análise.

Pode-se concluir então que:

**Pertence ao triângulo em análise se:**

- Em relação aos três pontos, os ângulos envolvendo o ponto projetado for menor que o ângulo formado pelas arestas do triângulo.

**Não pertence ao triângulo em análise se:**

- Em relação a algum dos três pontos, os ângulos envolvendo o ponto projetado for maior que o ângulo formado pelas arestas do triângulo.

Para realizar o cálculo desses ângulos, a equação 4.2 deve ser usada, onde A e B são os vetores o qual se pretende calcular o ângulo.

$$arc\ cos\ \theta = \frac{x_a . x_b + y_a . y_b + z_a . z_b}{\left|\vec{A}\right| . \left|\vec{B}\right|} \qquad \textbf{(4.2)}$$

Com isso, pode-se então encontrar quais pontos não pertencem ao modelo CAD.

Para encontrar quais facetas não pertencem ao modelo CAD, basta apenas analisar e separar toda e qualquer faceta do modelo escaneado que possuam pelo menos um dos pontos encontrados como rebarba.

Etapa 6: Como conclusão do método, deve-se então salvar as facetas encontradas em um arquivo de texto no formato de arquivo STL, ou ainda, armazenar essas informações na forma binária como um arquivo STL binário.

# 5 Implementação e Validação

## 5.1 Implementação

A implementação do método proposto foi realizada em linguagem Java. O código segue na seção Anexos.

## 5.2 Validação

Para o processo de validação do método, foram feitas algumas comparações. Dentre elas, comparações de simulação de rebarba no próprio sistema CAD, onde se tem a modelagem da peça que se pretende obter, e a modelagem da peça contendo uma ou mais rebarbas. E também, validação por escaneamento de peça física comparada ao modelo CAD.

### 5.2.1 Experimento 1

A Figura 5.1 mostra o modelo matemático da peça projetada, ou seja, a peça que se pretende obter após o processo de fundição seguido do de rebarbação.

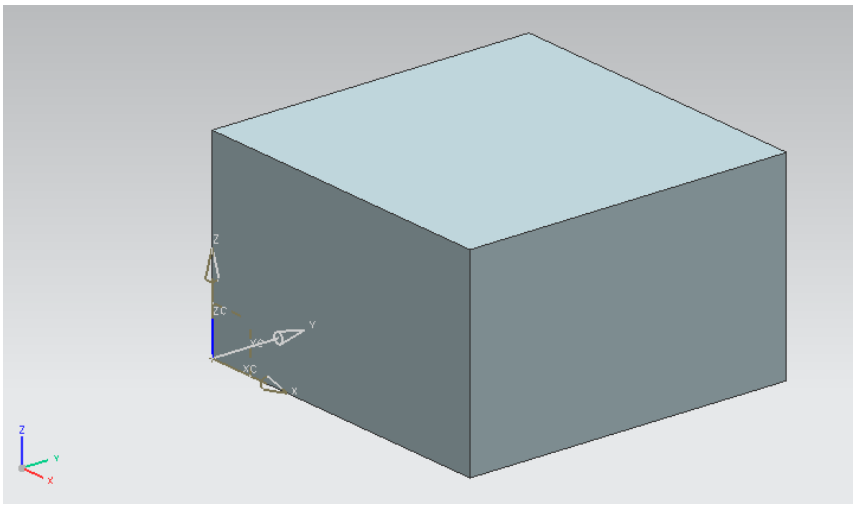


***Figura 5.1: Modelo CAD***

A Figura 5.2 mostra a transformação do modelo CAD em formato STL, composto por 12 triângulos.

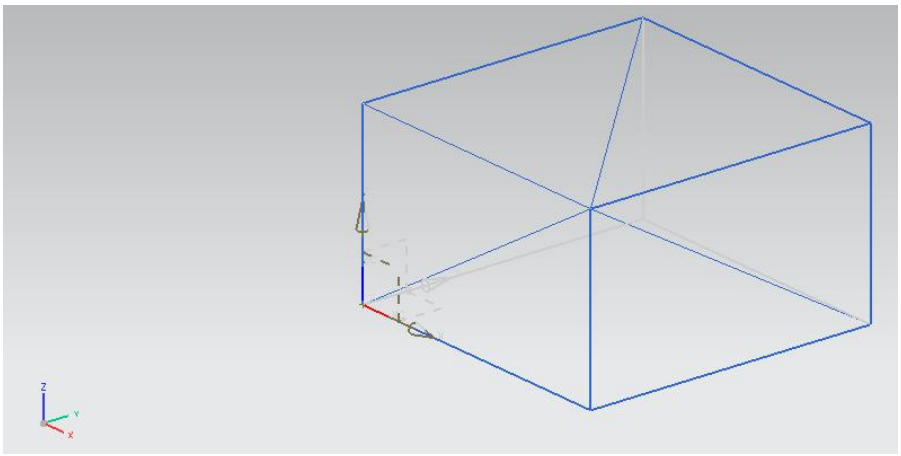


***Figura 5.2: STL do modelo CAD***

Nesse experimento, a Figura 5.3 mostra a modelagem de uma rebarba para uma das validações. Essa modelagem, além de simular a rebarba, também foi realizada em um lugar diferente no sistema de coordenadas do que o modelo CAD da peça projetada. Dessa forma, pode-se validar tanto o alinhamento quanto o reconhecimento de rebarbas.

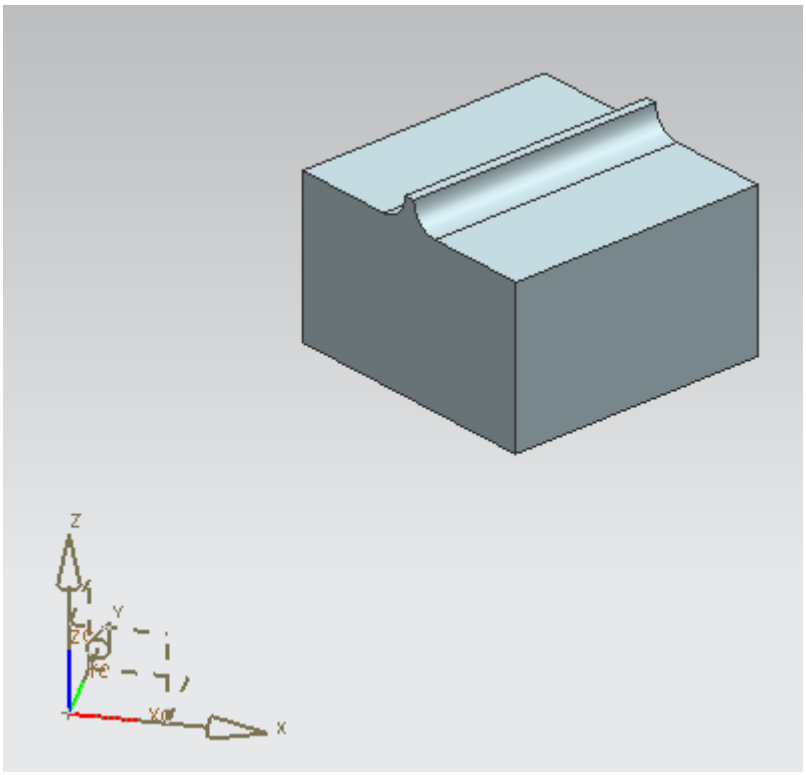



***Figura 5.3: Rebarba modelada***

A Figura 5.4 mostra o modelo matemático da rebarba transformado em um arquivo STL, composto por 60 triângulos.

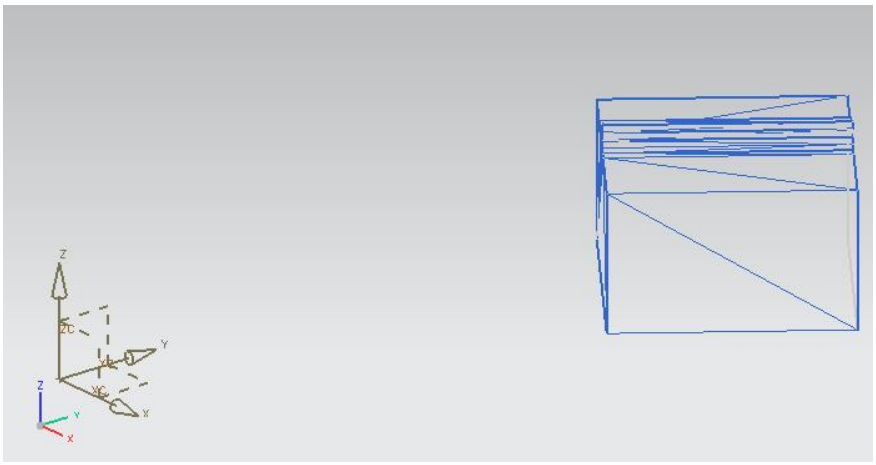


***Figura 5.4: STL Rebarba modelada***

Na Figura 5.5, os dois arquivos STL, o da peça projetada e o da peça contendo as rebarbas foram inseridos em um mesmo arquivo. Pode-se perceber que realmente, perante um mesmo sistema de coordenadas, as malhas não estão alinhadas, tornando o reconhecimento de rebarbas inviável.

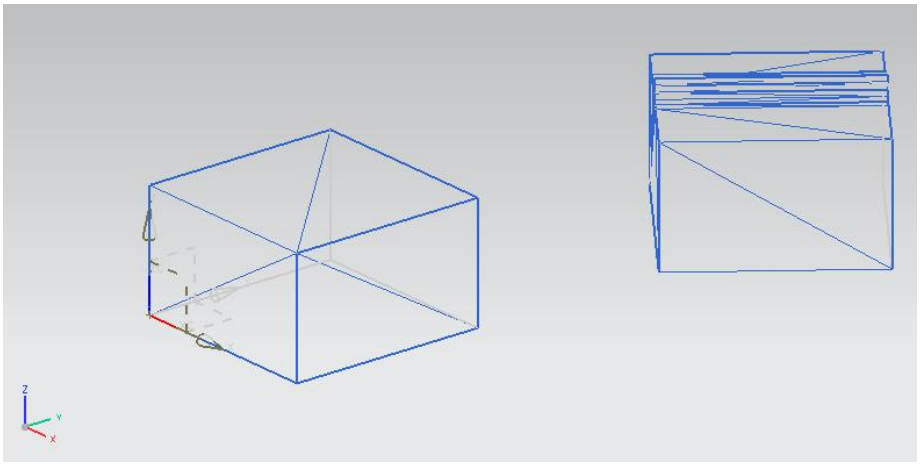


***Figura 5.5: STL modelo CAD e modelo com rebarbas***

Como entrada de dados para o procedimento do método para reconhecimento de rebarbas descrito nesse trabalho, foram escolhidos os 8 vértices da peça, tanto da peça projetada, quanto a da simulação das rebarbas. Conforme descrito no capítulo sobre descrição do método, o responsável pelo modelamento deve indicar esses pontos para serem coletados. É importante ressaltar que esses pontos têm que estar no mesmo lugar em ambas as peças, e na mesma seqüência.

A Figura 5.6 mostra como esses pontos foram selecionados de forma a evitar a rebarba.

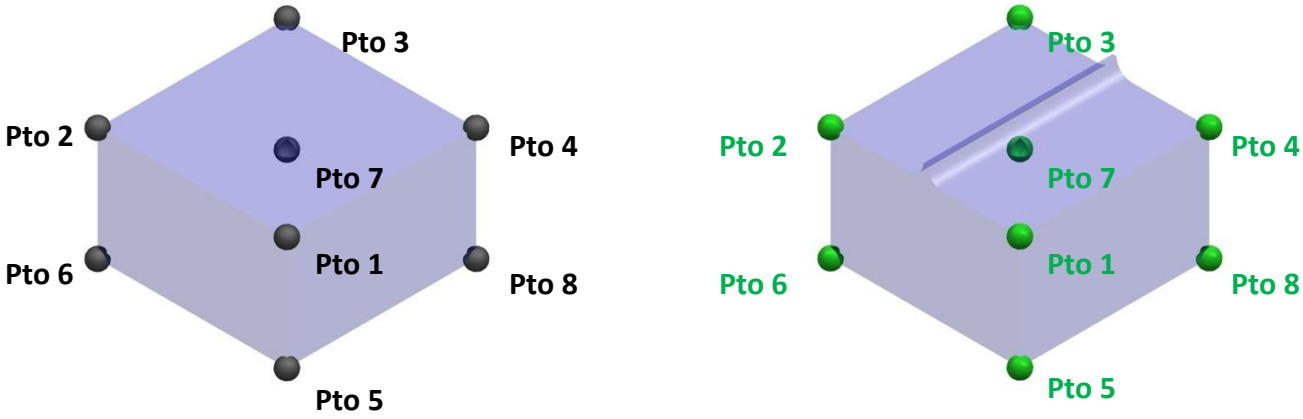



***Figura 5.6: Pontos para alinhamento***

A tabela a seguir mostra os valores das coordenadas de cada ponto do experimento:

| Pontos Para Cálculo do Alinhamento | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| Coordenadas STL Peça Projetada | | | | Coordenadas STL Peça com Rebarba | | | |
| | | | | | | | |
| Ponto | X | Y | Z | Ponto | X | Y | Z |
| 1 | 50 | 0 | 30 | 1 | 50 | 100 | 30 |
| 2 | 0 | 0 | 30 | 2 | 10 | 130 | 30 |
| 3 | 0 | 50 | 30 | 3 | 40 | 170 | 30 |
| 4 | 50 | 50 | 30 | 4 | 80 | 140 | 30 |
| 5 | 50 | 0 | 0 | 5 | 50 | 100 | 0 |
| 6 | 0 | 0 | 0 | 6 | 10 | 130 | 0 |
| 7 | 0 | 50 | 0 | 7 | 40 | 170 | 0 |
| 8 | 50 | 50 | 0 | 8 | 80 | 140 | 0 |

Como resultado para o processo de alinhamento, o valor do quatérnio de rotação é:

Q0: 0,9486; Qx: 0,0; Qy: 0,0; Qz: 0,3162

O vetor de translação é: X: 20,0; Y: 110,0; Z: 0,0.

Como resultado da transformação de rotação, a Figura 5.7 mostra as duas malhas inseridas em um mesmo sistema de coordenadas.

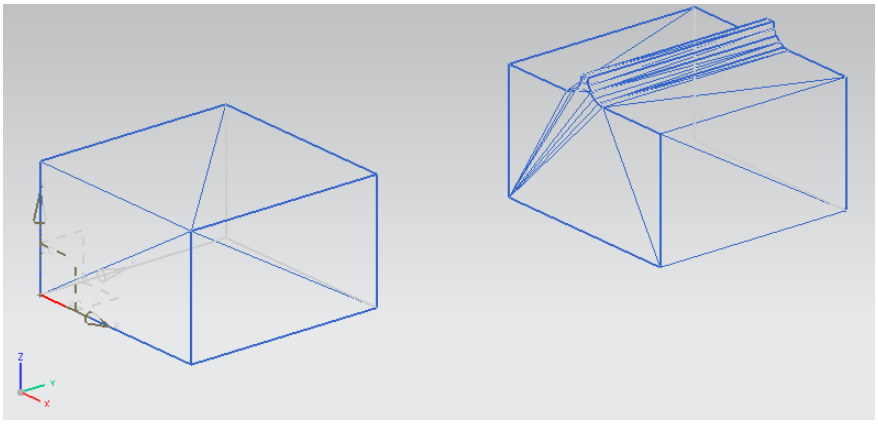


***Figura 5.7: Peça com rebarba aplicada a transformação de rotação***

Aplicando então o vetor de translação, obtém as malhas perfeitamente alinhadas conforme se pode observar na Figura 5.8.

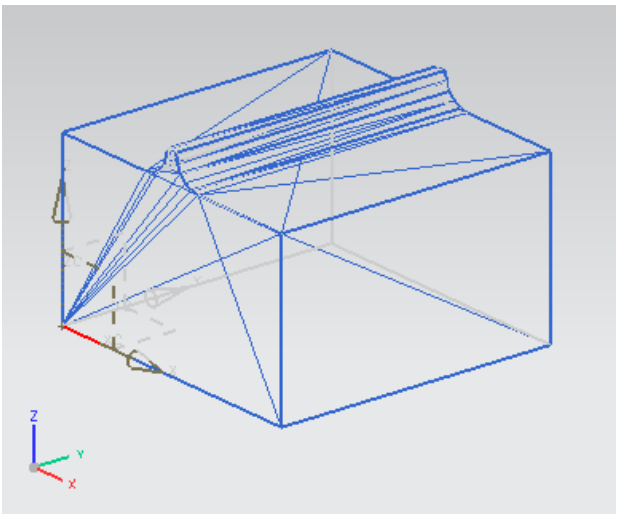


***Figura 5.8: Malhas alinhadas***

Aplicando então, a implementação dos cálculos referentes ao reconhecimento de rebarbas, se tem que 8 pontos do STL com rebarbas não pertencem ao STL do modelo da peça projetada. Então, dos 60 triângulos que fazem parte da malha contendo a rebarba, 42 são pertencentes à rebarba.

Na Figura 5.9 podemos ver apenas os triângulos que descrevem a rebarba sendo a tolerância de 3mm. Ou seja, todo triangulo que estiver afastado até 3 mm da peça não foi considerado e separado como rebarba.

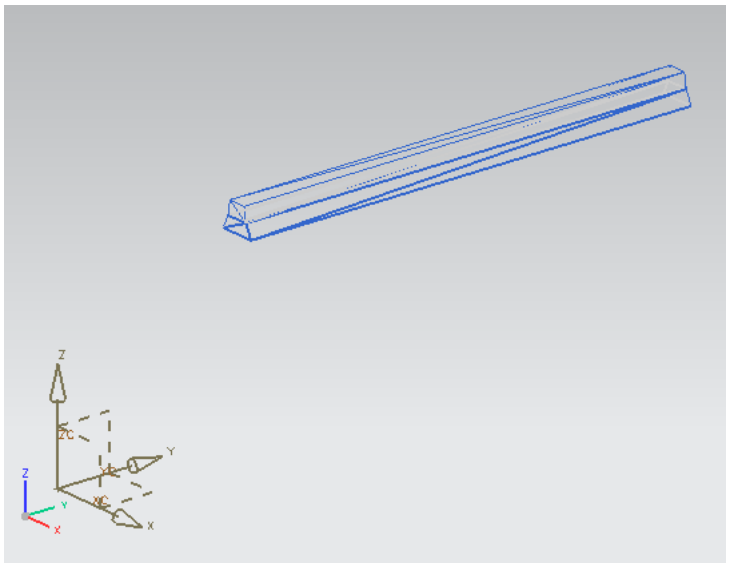


***Figura 5.9: Rebarba Tolerância 3 mm.***

Na Figura 5.10, tem-se o resultado do arquivo STL contendo as rebarbas separadas com tolerância de 0,5 mm. Os triângulos que se estendem pelo lado da peça são separados, pois um dos seus vértices é uma rebarba. Essa área separada que não é rebarba não ocorre quando a peça é escaneada, pois os lados dos triângulos além de serem uniformes, são menores e descrevem melhor a superfície da peça. Devido ao fato desse experimento ser a transformação de um arquivo CAD em arquivo STL, o sistema CAD procura sempre colocar o menor número de triângulos para descrever uma peça na tolerância solicitada. Como o lado dessa peça em teste é plano, sempre terão triângulos pegando toda a face lateral.

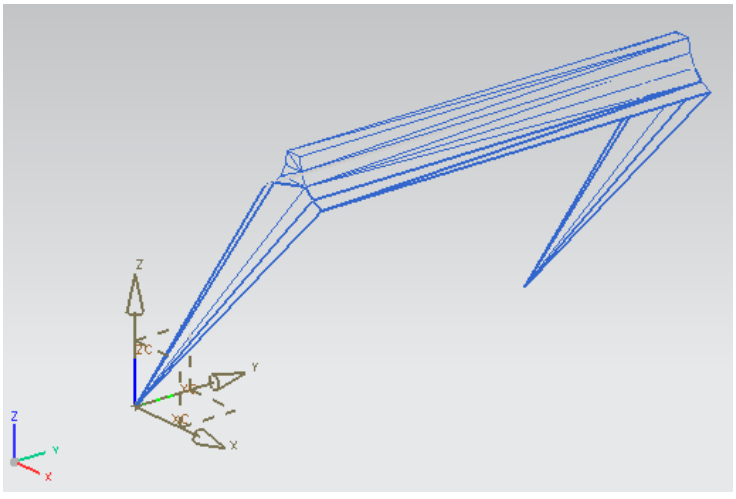


***Figura 5.10: Rebarbas Tolerância 0,5 mm***

A Figura 5.11 mostra ainda, as rebarbas com uma tolerância de 0,2 mm. Nessa figura, todas as facetas pertencentes as rebarbas foram separadas. Como esse exemplo é aplicado de um modelo CAD, o alinhamento é exato e, portanto, mesmo

que a tolerância usada seja de 0 mm, as rebarbas e somente as rebarbas serão separadas.

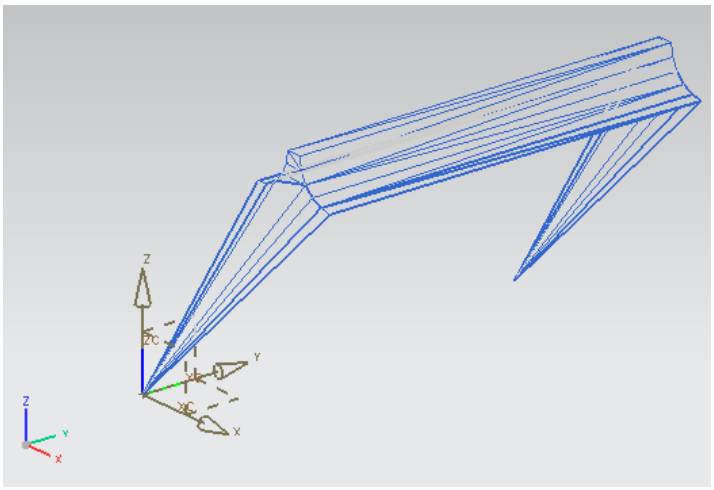


***Figura 5.11: Rebarbas Tolerância 0,2 mm***

### 5.2.2 Experimento 2

A Figura 5.1 mostra o modelo matemático de uma peça com rebarbas. Nesse experimento, no mesmo modelo com rebarba do experimento 1, está modelado duas rebarbas em localidades diferentes. Nesse experimento, se pretende simular uma rebarba com faces perpendiculares a peça, o que faz com que a projeção das rebarbas nas facetas do modelo CAD se encaixe nos casos especiais descritos no capítulo 4 e mostre que o método descrito funciona corretamente para toda e qualquer rebarba. Através do cilindro, pretende simular o canal por onde o metal é vazado na peça fundida, mostrando que o método não só reconhece rebarbas, mas também, toda e qualquer saliência de material que não pertence à peça projetada.

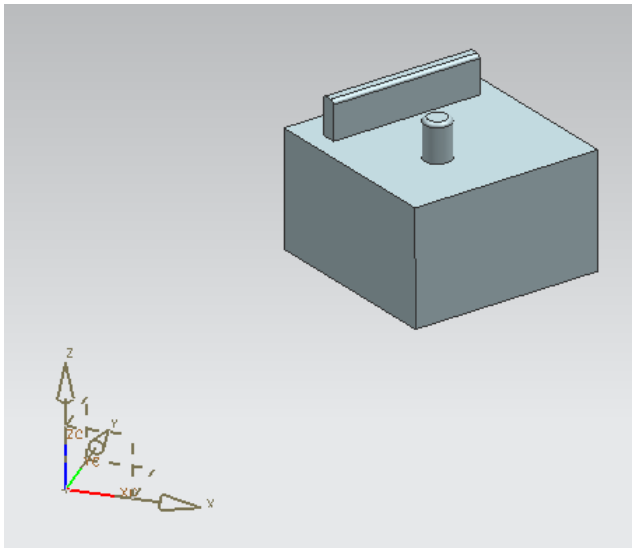


***Figura: 5.12: Rebarba Modelada***

A Figura 5.13 mostra o arquivo STL dessa peça com rebarbas.

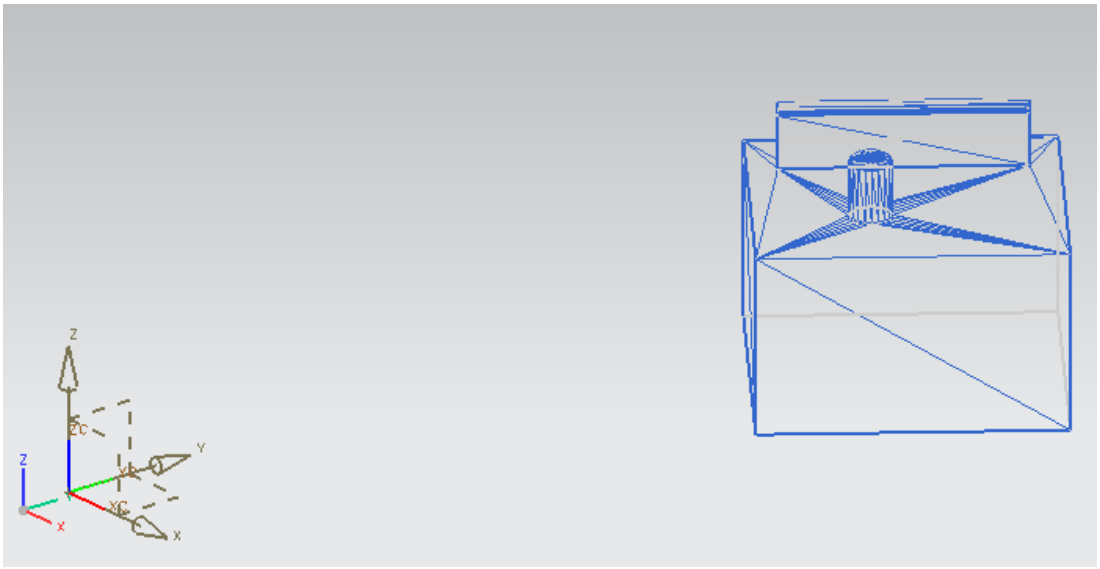


***Figura 5.13: STL Rebarba Modelada***

Os pontos a serem inseridos no cálculo das transformações assim como as transformações são as mesmas do experimento 1. Dessa forma, ilustrando as rebarbas separadas temos a Figura 5.14 contendo as rebarbas separadas com tolerância de 0,2 mm. Devido ao fato de que essa rebarbas não estão alinhadas com nenhuma das faces da peça projetada, como resultados temos apenas os triângulos que descrevem as rebarbas, diferentemente do experimento 1, que um mesmo triangulo descrevia o lado da peça e a rebarba.

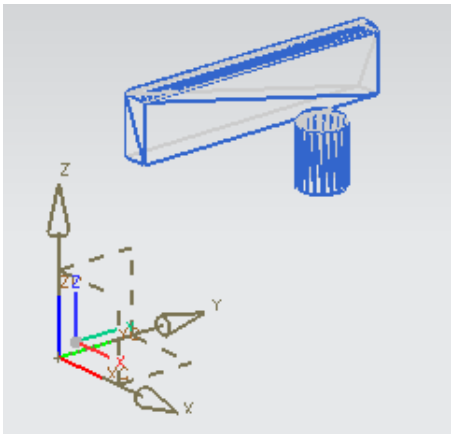


***Figura 5.14: STL Rebarbas***

### 5.2.3 Experimento 3

Nesse experimento, a peça projetada é a mesma do experimento 1 e 2. Porém o arquivo STL da peça com rebarbas, é um arquivo proveniente de um escaneamento

a laser de uma peça física contendo as rebarbas. Essa peça é a mesma do experimento 1, porém, real.

A peça foi obtida pelo processo de usinagem. Devido ao fato de que usinar uma peça simulando uma rebarba é mais viável para a validação desse trabalho do que a fundição da mesma. Isso significa que, essa usinagem simula em sua rebarba, de forma ampliada, a rebarba resultante do fechamento do molde de fundição, e, portanto, após o escaneamento, tem as mesmas funcionalidades e valida o trabalho da mesma maneira.

A Figura 5.15 mostra a peça do experimento 3, na mesa, a peça do experimento 4, na base preta com as referências, a câmera e o software usado no escaneamento.

Esse processo foi gentilmente realizado pela empresa RGB sistemas, em Santa Bárbara D´Oeste.

O sistema utilizado é o VX Elements – 3D Digitalizing Software 1.0.

A câmera é uma Hand Scan – ExaScan da Creaform, uma empresa Canadense.


***Figura 5.15: Processo de Escaneamento das Peças experimento 3 e 4***

Como resultado do escaneamento, obteve-se a malha mostrado na Figura 5.16.

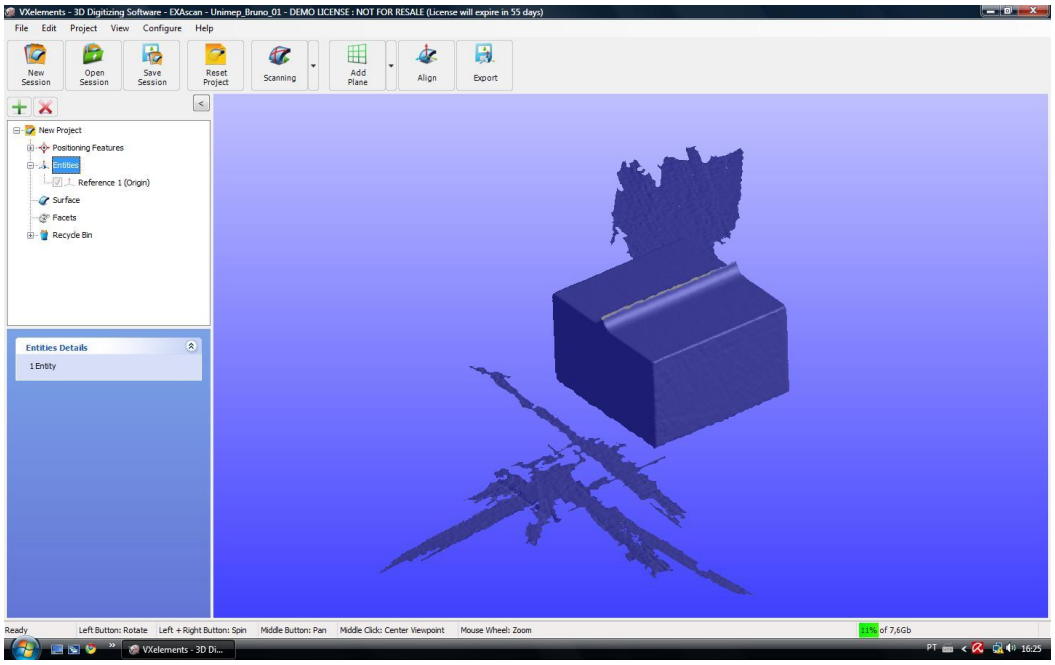



***Figura 5.16:Resultado do Escaneamento do Experimento 3,***

A cada face, foi atribuído um plano que minimizava o erro dos pontos perante esse plano, conforme mostrado na Figura 5.17.

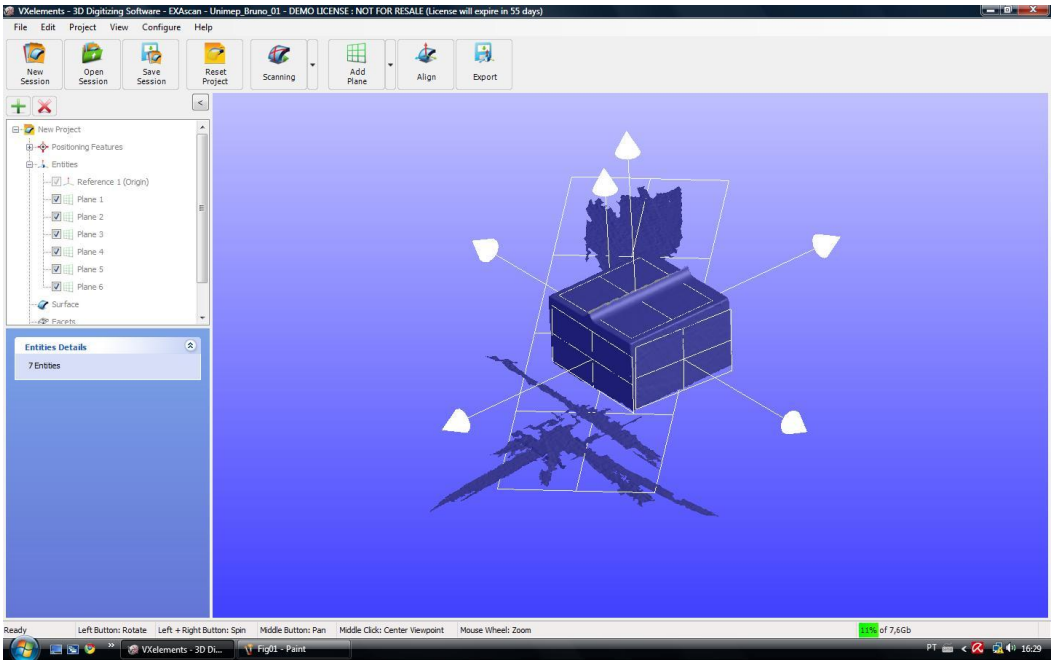



***Figura 5.17: Planos inseridos na face escaneada***

Pontos foram adicionados à intersecção desses planos. Tanto os planos quanto os pontos foram adicionados diretamente no software VX Elements, o qual tem ferramentas especificas para tratamento de malhas STL. Na Figura 5.18 pode ser

observados os pontos adicionados. As coordenadas desses pontos são fornecidas pelo próprio software.

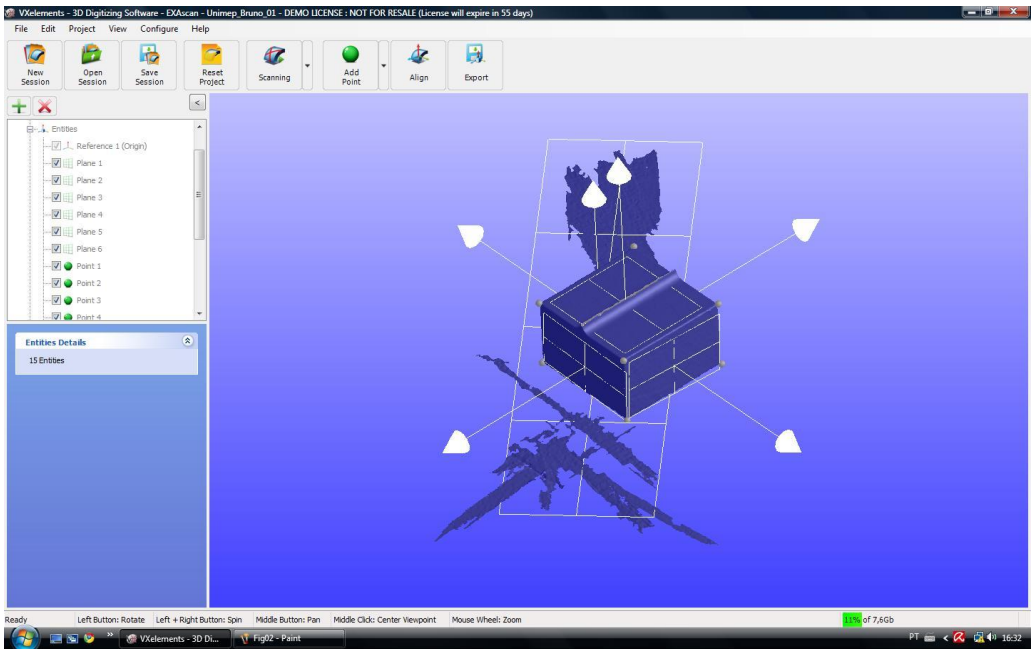



***Figura 5.18: Pontos adicionados a interseção dos planos***

A tabela a seguir mostra os valores das coordenadas de cada ponto do experimento:

| Pontos Para Cálculo do Alinhamento | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| Coordenadas STL Peça Projetada | | | | | Coordenadas STL Peça com Rebarba | | |
| | | | | | | | |
| Ponto | X | Y | Z | | Ponto | X | Y | Z |
| 1 | 50 | 0 | 30 | | 1 | -30,74 | 94,63 | 373,75 |
| 2 | 0 | 0 | 30 | | 2 | -23,34 | 142,57 | 362,99 |
| 3 | 0 | 50 | 30 | | 3 | -64,18 | 154,72 | 388.57 |
| 4 | 50 | 50 | 30 | | 4 | -71,57 | 106,82 | 399,31 |
| 5 | 50 | 0 | 0 | | 5 | -14,14 | 97,73 | 398,68 |
| 6 | 0 | 0 | 0 | | 6 | -6,80 | 145,59 | 387,86 |
| 7 | 0 | 50 | 0 | | 7 | -47,66 | 157,79 | 413,62 |
| 8 | 50 | 50 | 0 | | 8 | -54,99 | 109,97 | 424,42 |

Como resultado para o processo de alinhamento, o valor do quatérnio de rotação é:

Q0: 0,2577; Qx: -0,5990; Qy: 0,7452; Qz: 0,1388

O vetor de translação é: X: -64,18; Y: 101,23; Z: 378,65.

O arquivo STL do modelo CAD possui 12 triângulos, e o arquivo escaneado, possui 11296 triângulos.

A Figura 5.19 mostra o STL do modelo CAD e o STL da peça escaneada inseridas em um mesmo sistema de coordenadas. Existe essa diferença entre os posicionamentos devido ao fato de que o processo de escaneamento utilizado define o zero assim que o scaner é ativado, e portanto, a uma certa distância da peça.

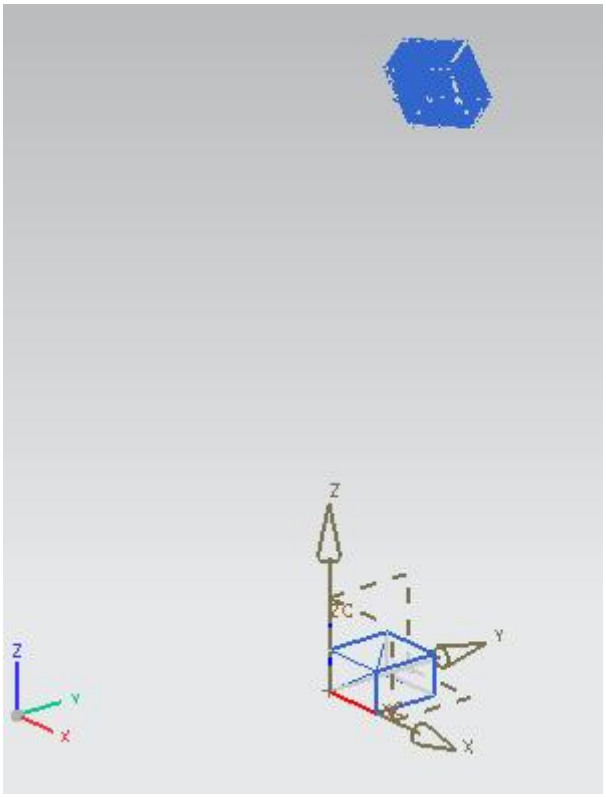



***Figura 5.19: STL Modelo CAD e STL Modelo Escaneado***

A Figura 5.20 mostra o modelo escaneado já aplicado as transformações de rotação e de translação.

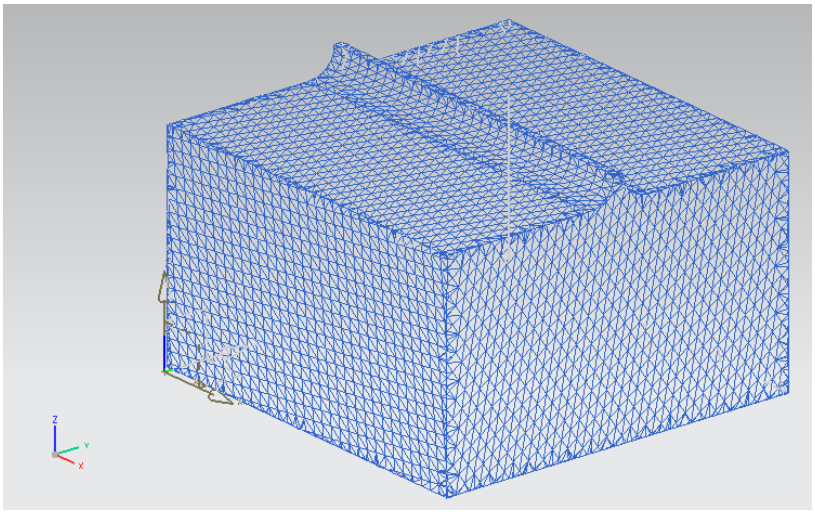


***Figura 5.20: Modelo escaneado já aplicado as transformações.***

Aplicado o método de identificação de rebarbas, a Figura 5.21 mostra o arquivo gerado pelo aplicativo. O tempo necessário entre ler o arquivo, alinhar, calcular as rebarbas e salvar novamente foi de 1 minuto e 35 segundos. Sendo que o aplicativo leva muito mais tempo lendo e salvando o STL do que calculando a transformação ou o reconhecimento de rebarbas.

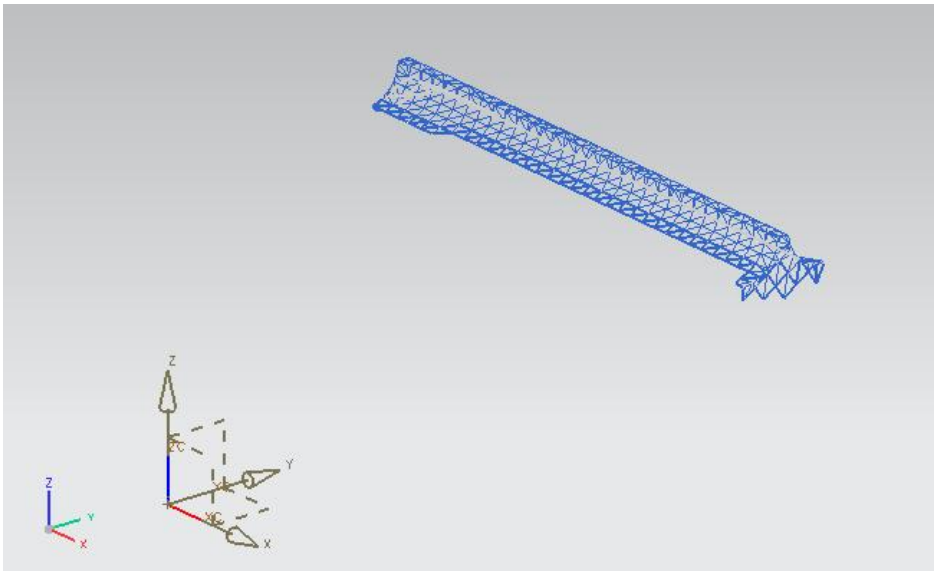



***Figura 5.21: Rebarbas***

### 5.2.4 Experimento 4

Nesse experimento, a peça projetada é a mesma do experimento 1, 2 e 3. O arquivo STL da peça com rebarbas, é um arquivo proveniente de um escaneamento a laser, obtido da mesma forma do experimento 3.

Como resultado do escaneamento, obteve-se a malha mostrado na Figura 5.16.

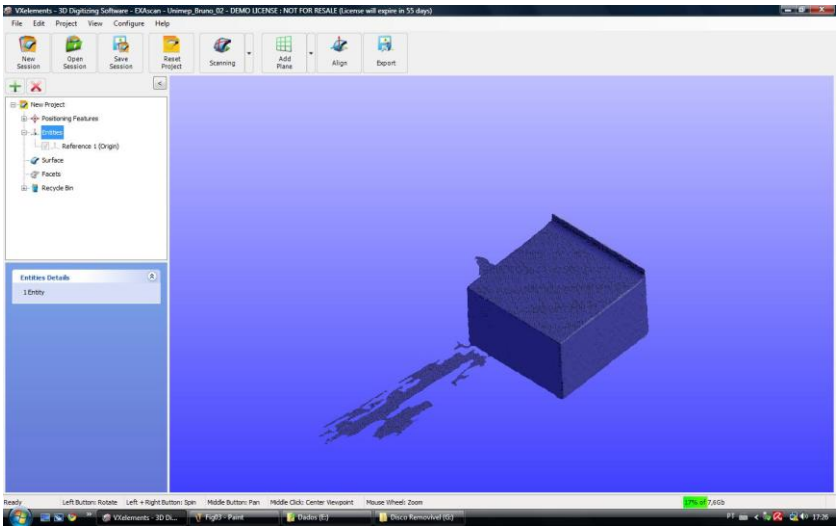


***Figura 5.22: Resultado do Escaneamento do Experimento 4,***

A cada face, foi atribuído um plano que melhor minimizava o erro dos pontos perante esse plano, da mesma forma do experimento 3, conforme mostrado na Figura 5.23.

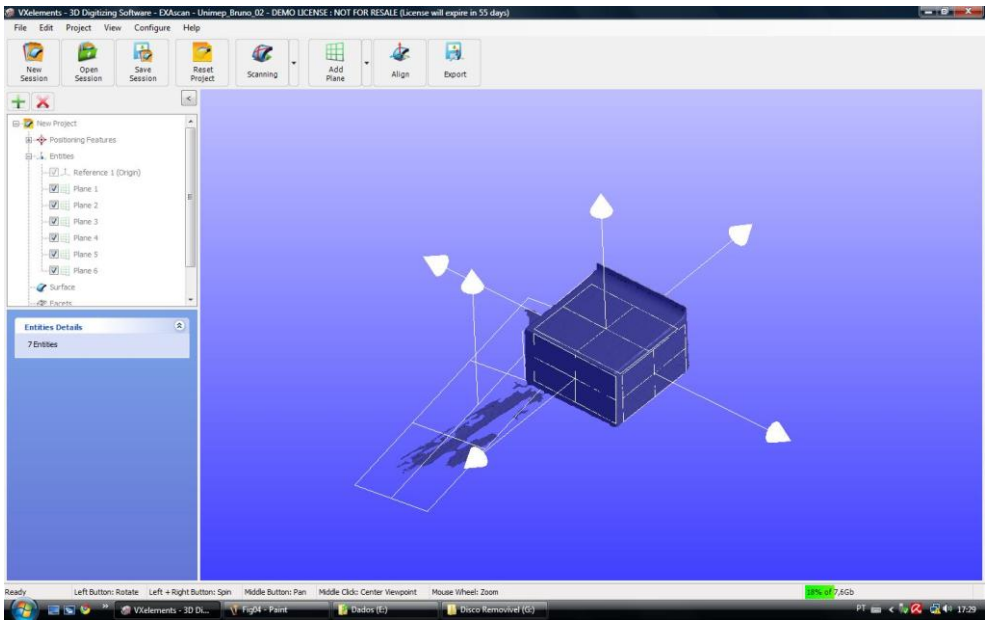


***Figura 5.23: Planos inseridos na face escaneada***

Pontos foram adicionados à intersecção desses planos.. Na Figura 5.24 pode ser observados os pontos adicionados. As coordenadas desses pontos são fornecidas pelo próprio software.

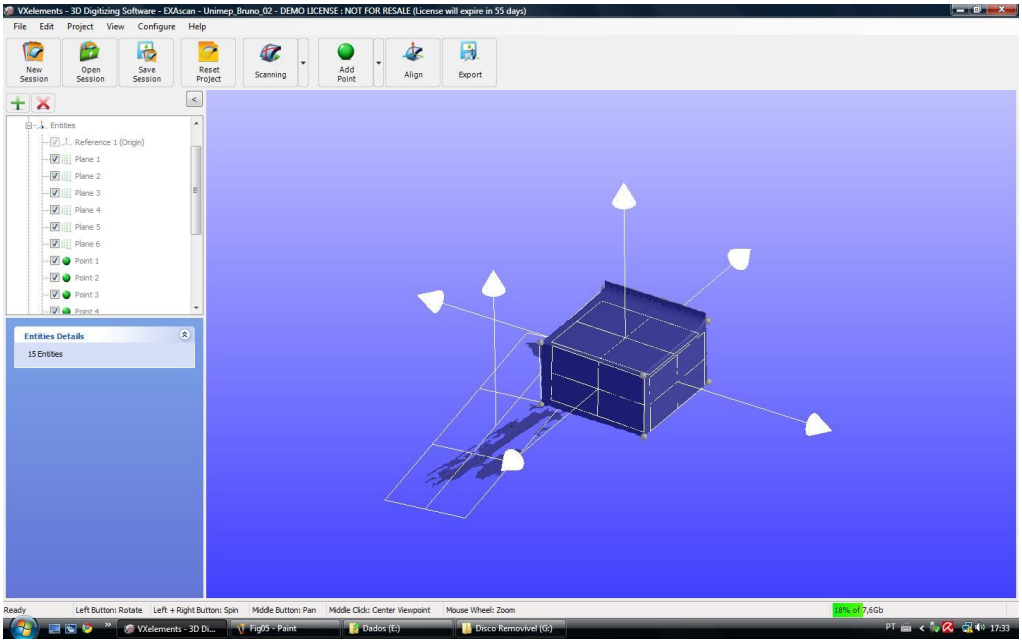


***Figura 5.24: Pontos adicionados a interseção dos planos***

A tabela a seguir mostra os valores das coordenadas de cada ponto do experimento:

| Pontos Para Cálculo do Alinhamento | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| Coordenadas STL Peça Projetada | | | | Coordenadas STL Peça com Rebarba | | | |
| | | | | | | | |
| Ponto | X | Y | Z | Ponto | X | Y | Z |
| 1 | 50 | 0 | 30 | 1 | 76,97 | 57,48 | 275,32 |
| 2 | 0 | 0 | 30 | 2 | 93,80 | 103,86 | 269,86 |
| 3 | 0 | 50 | 30 | 3 | 48,15 | 121,31 | 278,28 |
| 4 | 50 | 50 | 30 | 4 | 31,31 | 75,03 | 275,32 |
| 5 | 50 | 0 | 0 | 5 | 82,83 | 58,65 | 304,16 |
| 6 | 0 | 0 | 0 | 6 | 99,71 | 105,05 | 298,93 |
| 7 | 0 | 50 | 0 | 7 | 53,93 | 122,53 | 307,23 |
| 8 | 50 | 50 | 0 | 8 | 37,05 | 76,23 | 312,45 |

Como resultado para o processo de alinhamento, o valor do quatérnio de rotação é:

Q0: 0,0927; Qx -0,5672; Qy: 0,8173; Qz: 0,0389

O vetor de translação é: X: 40.47; Y: 65.02; Z: 276.24.

O arquivo STL do modelo CAD possui 12 triângulos, e o arquivo escaneado, possui 12098 triângulos.

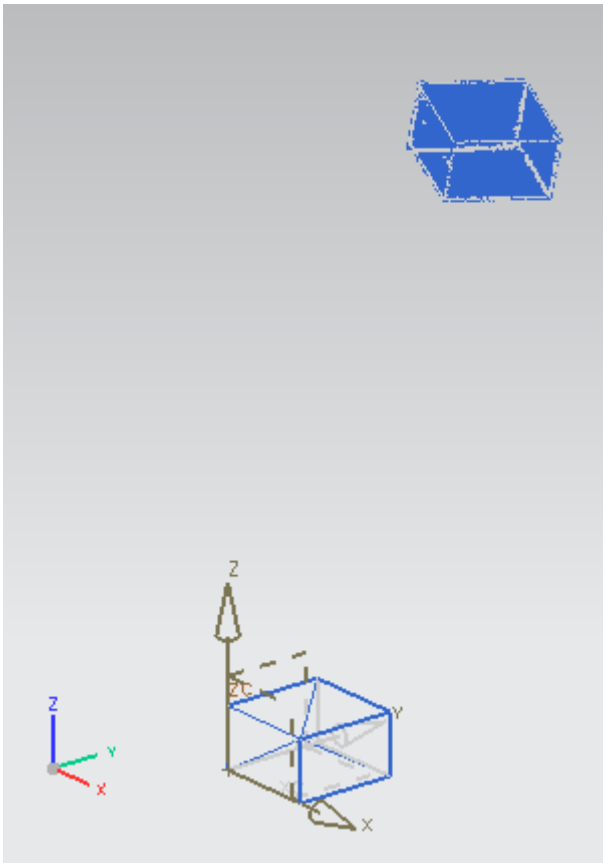



***Figura 5.25: STL Modelo CAD e STL Modelo Escaneado***

A Figura 5.25 mostra o STL do modelo CAD e o STL da peça escaneada inseridas em um mesmo sistema de coordenadas. Existe essa diferença entre as coordenadas, devido ao fato de que o processo de escaneamento utilizado define o zero assim que o scaner é ativado, e portanto, em um lugar distinto.

A Figura 5.20 mostra o modelo escaneado já aplicado às transformações de rotação e de translação.

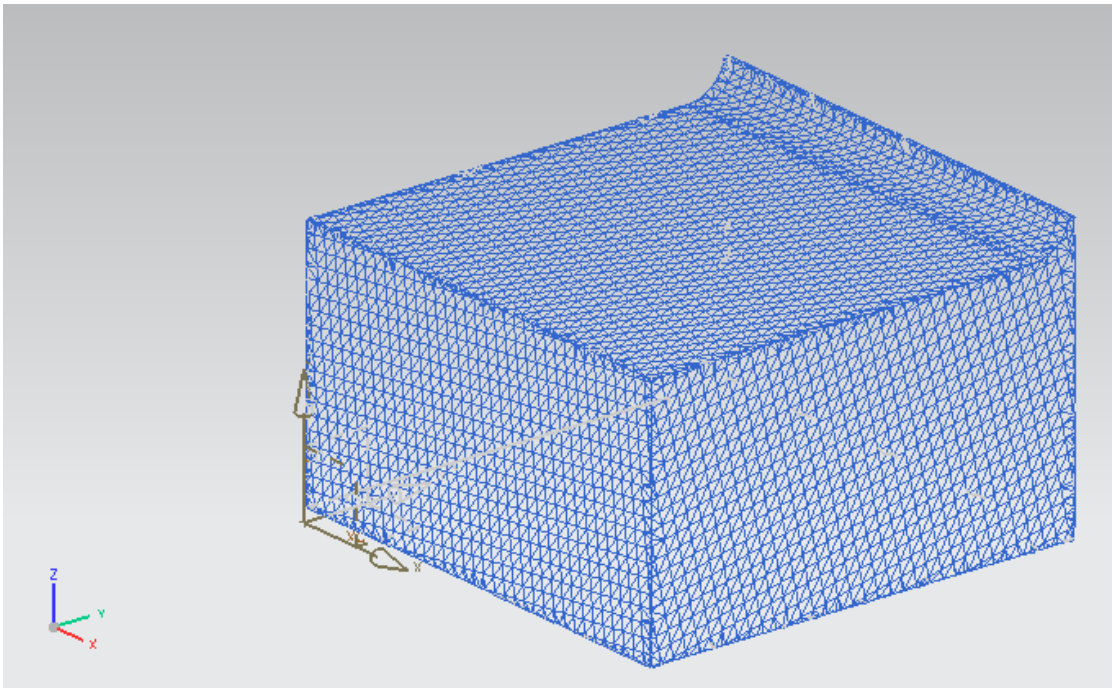


***Figura 5.26: Modelo escaneado já aplicado as transformações de rotação e translação***

Aplicado o método de identificação de rebarbas, a Figura 5.27 mostra o arquivo gerado pelo aplicativo. O tempo necessário entre ler o arquivo, alinhar, calcular as rebarbas e salvar novamente foi de 1 minuto e 53 segundos.

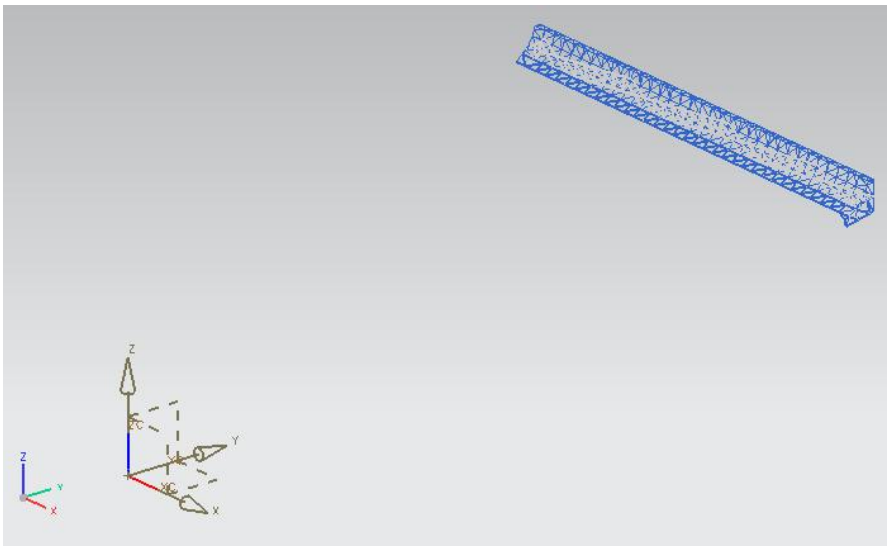


***Figura 5.27: Rebarbas***

# 6 Conclusões

Robôs industriais são uma realidade na indústria. Nas áreas de pintura, soldagem e manipulação, esses desempenham suas funções com competência e viabilidade.

Esta automação possibilita incrementos na produtividade, melhoram a qualidade dos produtos, evita-se perda e refugos, reduz tempo morto, permite operação em ambiente difíceis e perigosos ou tarefas desagradáveis ou repetitivas, não precisam de condições ambientais como ar condicionado, luz e silêncio além de muitas outras vantagens.

A aplicação de robôs em outras áreas como a usinagem, principalmente devido a suas características de baixa rigidez mecânica quando comparados a máquinas ferramenta faz com que ainda sejam poucos utilizados. Porém, esta aplicação vem se mostrando cada vez mais promissora. Em processos de usinagem como a rebarbação, por exemplo, onde as exigências de exatidão e qualidade superficial são menores do que em um processo de usinagem que necessita tolerâncias dimensionais mais precisas, os robôs já estão sendo aplicados e amplamente estudados.

Alguns processos consideram uma trajetória fixa para o robô não levando em consideração possíveis erros de fixação da peça. Alguns estudos consideram ainda o uso de câmeras para fazer uma leitura da geometria e atualizar a trajetória da ferramenta on-line. Mas como mostram os estudos, uma melhoria nesse processo seria um processo de reconhecimento automático dessas geometrias denominadas rebarbas. Esses mostram que uma possibilidade de melhoria do processo seria considerar a geometria da rebarba e a geometria que se pretende chegar e assim gerar uma trajetória mais eficiente para a ferramenta.

O objetivo deste trabalho foi desenvolver um método capaz de identificar automaticamente rebarbas em peças fundidas por meio da comparação dos dados no formato de arquivo STL da peça fundida com o modelo CAD.

Foi desenvolvido um método que possibilita essa melhoria. Esse consiste em um processo de alinhamento das malhas de triângulos de ambos modelos geométricos, seguido de um processo de comparação e por fim, a identificação das regiões denominadas de rebarbas. Dessa forma, o método desenvolvido e descrito nesse trabalho se mostrou promissor no reconhecimento automático das geometrias consideradas rebarbas em peças fundidas.

Para o alinhamento das malhas, o método de Horn foi estudado e implementado. O método de Horn consistiu em determinar uma matriz 4x4 onde o autovetor associado ao maior autovalor dessa matriz representou a melhor transformação de rotação entre as malhas. Para encontrar os autovalores dessa matriz 4x4 foi utilizado o método desenvolvido por Ferrari. Ao se calcular o determinante de uma matriz 4x4, esse resulta em um polinômio de grau 4. Portanto, para resolver esse polinômio foi aplicado o método de Ferrari. E por fim, para encontrar o autovetor foi utilizado o método de Gauss.

Para encontrar as geometrias que são consideradas rebarbas foi desenvolvido um método próprio que consistiu na comparação de cada ponto da malha escaneada com cada faceta da malha proveniente do modelo CAD. Basicamente o método verifica  esses pontos estão a uma distancia o dentro de uma faixa de tolerância de cada faceta do modelo CAD:

Nos experimentos realizados e através da análise dos resultados, pode-se comprovar a eficiência do processo assim como também justificar sua importância. O processo de alinhamento teve resultados positivos devido à forma de inserção dos pontos para o processo de alinhamento. Uma dificuldade dessa forma é a definição da localização e a inserção desses pontos manualmente. Isso também pode trazer erros como de digitação assim como deixa o processo mais demorado para realizar os cálculos.

Esse processo pode ser aprimorado através de um estudo mais aprofundado de um método para reconhecer regiões de similaridade entre as malhas. Esse poderia definir esses pontos automaticamente contribuindo para a automatização da inserção manual de pontos no método de alinhamento. Uma possibilidade é o uso de Spin Images. Alguns problemas desse método estudado para a aplicação do mesmo nesse trabalho seriam a diferença entre o número de pontos das malhas e a simetria das malhas. Os estudos mostraram que é possível solucionar esses problemas e portanto, usar esse método para essa automação.

Como proposta para trabalhos posteriores, pode ser feito uma total automação do processo permitindo ainda mais que o mesmo seja eficaz e promissor.

# 7 Revisão Bibliográfica

[1] BOGUE, Robert. Finishing robots: a review of technologies and applications. **Industrial Robot: An International Journal,** p. 6-12. 2009.

[2] BLOSS, Richard. Robots that are faster or perform in-mold labeling highlight plastics show. **Industrial Robot: An International Journal,** p. 13-16. 2010.

[3] YEO, S. H. et al. Ultrasonic Deburring. **Advanced Manufacturing Technology**, London, p. 333-341. 1997.

[4] YUNMING, Zhu et al. A Study on Milling Burr Expert System in Micro-Machining. **IEEE,** Zhenjiang, p. 965-969. 1 jan. 2008.

[5] YEO, S. H. et al. Trajectory planning for automated robotic deburring on an unknown contour. **International Journal Of Machine Tools & Manufacture,** Chung-li, p. 957-978. 2000.

[6] BROWN, Leslie. The development of software to assist in off-line programming for robotic fettling of cast components. **Industrial Robot,** p. 282-287. 1998.

[7] KOCHAN, Anna. Deburring: a joint operation. **Assembly Automation,** p. 29-30. 1996.

[8] PAN, Zengxi; ZHANG, Hui. Robotic machining from programming to process control: a complete solution by force control. **Industrial Robot: An International Journal,** p. 400-409. 2008.

[9] FAYAWEERA, Nirosh; WEBB, Phil. Robotic edge profiling of complex components. **Industrial Robot: An International Journal,** p. 38-47. 2011.

[10] BENATI, F. et al. A robot-based burr measurement system for the automotive industry. **IEEE - International Conference On Automation Science And Engineering**, Uxbridge, p. 1740-1744. 1999.

[11] CHEN, M.; LIU, G.; SHEN, Z.. Study on Active Process Control of Burr Formation in Al-Alloy Milling Process. **IEEE - International Conference On Automation Science And Engineering**, Shanghai, p. 431-436. 7 out. 2006.

[12] ABELE, Eberhard; KULOK, Michael; WEIGOLD, Matthias... Machining with industrial Robots: analysis of the system behavior. **Ciência e Tecnologia**, p. 72. 2006.

[13] PIRES, F. Noberto et al. Force/torque sensing applied to industrial robotic deburring. **Sensor Review,** p. 232-241. 2002.

[14] AN executive summary of the strategic research agenda for robotics in Europe **CARE - Coordination Action For Robotics In Europe**. jun. 2008.

[15] MAKHANOV, S.s. et al. On the tool-path optimization of a milling robot. **Computers & Industrial Engineering**, Thailand, p. 455-472. 20 mar. 2002.

[16] MATSUOKA, Shin-ichi et al. High-speed end milling of an articulated robot and its characteristics. **Journal of Materials Processing Technology**, Toyama, p. 83-89. 3 abr. 1998.

[17] BOGUE, Robert. The development of medical microrobots: a review of progress. **Industrial Robot: An International Journal**, p. 294-299. 2008.

[18] WILSON, Mike. Developments in robot applications for food manufacturing. **Industrial Robot: An International Journal**, p. 498-502. 2010.

[19] KUSUDA, Yoshihiro. The use of robots in the Japanese food industry. **Industrial Robot: An International Journal,** p. 503-508. 2010.

[20] MASEY, Rene F. Moreno et al. Guidelines for the design of low-cost robots for the food industry. **Industrial Robot: An International Journal,** p. 509-517. 2010.

[21] KIM, Foonyoung et al. A practical approach for minimum-time trajectory planning for industrial robots. **Industrial Robot: An International Journal,** p. 51-61. 2010.

[22] PIRES, F. Norberto; GODINHO, T.; FERREIRA, P.. CAD interface for automatic robot welding programming. **Industrial Robot: An International Journal**, Coimbra, p. 71-76. 2004.

[23] ABB Power and Productive Disponível em: <http://www.abb.com.br/product/seitp327/cb6581d257fb24ccc12573fe003d9436.aspx>. Acesso em: 15 nov. 2010.

[24] TOMASOLSSON et al. Cost-efficient drilling using industrial robots with high-bandwidth force feedback. **Robotics And Computer-integrated Manufacturing**, p. 24-38. 2010.

[25] MIN, Sangkee; KIM, Jinsoo; DORNFELD, David A.. Development of a drilling burr control chart for low ally steel, AISI 4118. **Journal Of Materials Processing Technology**, Berkeley, p. 4-9. 2001.

[26] GRONINGER, Rolf; KUS, Elzbieta; HUPPI, Richard. Market Study on Adaptive Robots for Flexible Manufacturing Systems. **IEEE - International Conference On Automation Science And Engineering**, Málaga, p. 978-984. abr. 2009.

[27] KIM, Changhoon; CHUNG, Jae H.. Application of an active pneumatic actuator to robotic deburring. **Industrial Robot: An International Journal**, New Jersey, p. 487-494. 2007.

[28] HSU, Feng-yih; FU, Li-chen. A New Adaptive Fuzzy Hybrid Force/Position Control for Intelligent Robot Deburring. **IEEE - International Conference On Automation Science And Engineering**, Detroit, p. 2476-2481. maio 1999.

[29] LEE, Young Dae; KANG, Byung Hun; PARK, Jong Oh. Robotic Deburring Strategy Using Burr Shape Recognition. **IEEE - International Conference On Automation Science And Engineering**, Seoul, p. 1513-1518. 1999.

[30] KOCHAN, Anna. ZF deburrs transmission housings with robots. **Industrial Robot**, p. 256-258. 1998.

[31] ZHANG, Hui et al. On-Line Path Generation for Robotic Deburring of Cast Aluminum Wheels. **IEEE - International Conference On Automation Science And Engineering**, Beijing, p. 2400-2405. 9 out. 2006.

[32] BRES, Antoine et al. Simulation of friction stir welding using industrial robots. **Industrial Robot: An International Journal,** p. 36-50. 2010.

[33] II, Eduardo José Lima; BRACARENSE, Alexandre Queiroz. Trajectory generation in robotic shielded metal arc welding during execution time. **Industrial Robot: An International Journal,** p. 19-26. 2009.

[34] CHEN, X.z.; CHEN, S.b.. The autonomous detection and guiding of start welding position for arc welding robot. **Industrial Robot: An International Journal,** p. 70-78. 2010.

[35] ABREU, Paulo. Robótica Industrial. 2002. 29 f. **Mestrado em Automação, Instrumentação e Controle** - Universidade do Porto, Porto, 2002.

[36] CHEN, Heping; FUHLBRIGGE, Thomas; LI, Xiongzi. A review of CAD-based robot path planning for spray painting. **Industrial Robot: An International Journal, Connecticut**, p. 45-50. 2009.

[37] STEWART, James. Cálculo Volume II. São Paulo: Pioneira, 2001.

[38] CALLIO, Carlos A. et al. Ágebra Linear e Aplicações. São Paulo: Atual Editora, 1981.

[39] BOLDRINI, José Luiz; COSTA, Rodrigues. Ágebra Linear. São Paulo: Harper, 1980.

[40] BAJPAI, A. C.; MUSTOE, L. R.; WALKER, D. Matemática para Engenheiros. São Paulo: Hemus, 1980.

[41] WIKIPEDIA ENCYCLOPEDIA. Wapidia. Disponível em: <http://wapedia.mobi/en/Cubic_function>. Acesso em: 15 jul. 2010.

[42] WIKIPEDIA ENCYCLOPEDIA. Wapidia. Disponível em: < http://wapedia.mobi/en/Quartic_function >. Acesso em: 15 jul. 2010.

[43] HORN, Berthold K. P. Closed-form solution of absolute orientation using unit quartenions. **Journal Of The Optical Society Of America**, p. 629-642. abr. 1987

[44] ANDREIS, Davide; CANUTO, Enrico S.. Orbit dynamics and kinematics with hll quaternions. **American Control Conference**, Boston, p. 3660-3665. 2 jul. 2004.

[45] MORTENSON, Michael E.. Geometric Modeling. Usa: Braun-Brumfield, 1985.

[46] ANDERL, Rainer. Grundlagen des CAE/CAD. Darmstadt: 2009. 145 p.

[47] JOHNSON, Andrew Edie. Spin-Images: A Representation for 3-D Surface Matching. 1997. 308 f. Doutorado - Carnegie Mellon University, Pittsburgh, 1997.

[48] VIEIRA, Thales Miranda de Almeida. Registro Automático de Superfícies Usando Spin-Image. 2007. 62 f. Mestrado - Universidade Federal de Alagoas, Maceió, 2007.

[49] BOLDRINI, José Luiz et al. Álgebra Linear. Sao Paulo: Haper, 1980.

[50] WIKIPEDIA ENCYCLOPEDIA. Wapidia. Disponível em: <http://wapedia.mobi/en/Cubic_function>. Acesso em: 15 jul. 2010.

[51] YUNMING, Zhu; GUICHENG, Wang. Simulation Model and Mechanism of Burr Formation Zhu Yunminga. **International Workshop On Modelling, Simulation And Optimization**, Zhenjiang, p. 350-354. 1 jan. 2008.

[52] ONWUBOLU, G. C.. Prediction of burr formation during face milling using a hybrid GMDH network model with optimized cutting conditions. **Int J Adv Manuf Technol**, London, p. 1083-1093. 1 jan. 2009

# ANEXOS

## Classe Main – Classe principal

```
package MainClass;

import Data.BurrRecognition;
import Data.HornAlignment;
import Data.DataManagement;

public class Main {
	/**
	 * Descricao do Programa
	 * O alinhamento é feito através de oito pontos inseridos manualmente no programa.
	 * Esse é feito pelo método de Horn
	 * O reconhecimento de rebarbas é feito pelo método de projecao de triangulos
	 */
	public static boolean Exception1=true;
	public static boolean Exception2=true;

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		System.out.println("Automatic Burr Recognition Program");

		System.out.println("\nStep 1 - Read the CAD and Scene models");

		DataManagement CADModel = new DataManagement();
		if(CADModel.readSTLFile("d:/STLCAD.stl")==false){
			Exception1=false;	}
		DataManagement SceneModel = new DataManagement();
		//if(SceneModel.readSTLFile("d:/STLCenaRebarba3.stl")==false){
		if(SceneModel.readSTLFile("d:/Peca1_22.stl")==false){
		//if(SceneModel.readSTLFile("d:/Peca2_22.stl")==false){
			Exception2=false;	}
		else{
		}

		System.out.println("\nException1:" +Exception1);
		System.out.println("\nException2:" +Exception2);

		if(Exception1==false&Exception2==true){
			System.out.println("\nInconsistent CAD Model");
			System.out.println("Please, check the file before restart");
		}
		if(Exception1==true&Exception2==false){
			System.out.println("\nInconsistent Scene Model");
			System.out.println("Please, check the file before restart");
		}
		if(Exception1==false&Exception2==false){
			System.out.println("\nInconsistent CAD and Scene Model");
```

```
                    System.out.println("Please, check both files before
restart");
              }
              if(Exception1==true&Exception2==true){
                                    //End of reading process
                    System.out.println("\nFiles read correctly.");
                    System.out.println("Proceding with the calculations.");

                    // The program starts here

                    System.out.println("Getting the transformations
points.");

                    CADModel.GetVertexesCAD();
                    SceneModel.GetVertexesScene();

                    System.out.println("\nCalculating the transformation by
Horn Method");
                    System.out.println("\nCAD Number of Triangles:
"+CADModel.Data.size());
                    System.out.println("\nScene Number of Triangles:
"+SceneModel.Data.size());

                    HornAlignment Alignment = new
HornAlignment(CADModel.Data,
                                    SceneModel.Data,
                                    CADModel.EigthVertexesCAD,
                                    SceneModel.EigthVertexesScene);

                    Alignment.Execute();

                    System.out.println("\nCalculating and separating the
burrs");

                    BurrRecognition Recognition = new
BurrRecognition(CADModel.Data,
                                    Alignment.TransformedSceneModel);

                    Recognition.Execute();

                    System.out.println("\nBurrs Recognition Complete");

                    System.out.println("\nBurrs Successfully Separated");

              }
        }
}
```

## Classe STLReader – Classe cuja função é ler um arquivo STL

```
package ReadSaveSTL;

import Objects.Triangle;
import Objects.Vertex;
//import Objects.Vertex;
//import java.io.BufferedReader;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
//import java.io.FileReader;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
//import java.util.StringTokenizer;
import java.util.StringTokenizer;
import java.util.Vector;

public class STLReader {

	private FileInputStream file;
	private DataInputStream data;
	private String pathname;
	private String header;

	Vector<Triangle> triangles = new Vector<Triangle>();

	public STLReader(String pathname) throws Exception  {// to open the
file
		try {
			this.pathname = pathname;
			file = new FileInputStream(pathname);
			data = new DataInputStream(file);
		} catch (FileNotFoundException e) {
			System.out.println(e.getMessage()); // FIXME: handle
Exception
			throw new Exception();
		}
	}

	public Vector<Triangle> ReadFile() {

		try {
		for (int i = 0; i < 80; ++i) {
		char ch = (char) data.readByte();
		if (ch != -1 && ch != '\0' && ch != '\n') {
		header += ch;
		} else
		break;
		}
		} catch (IOException e) {
		System.out.println(e.getMessage()); // FIXME: handle Exception
		}
		if (header.startsWith("nullsolid"))
		return asciiReader();
		//else
```

```
            else
            return binaryReader();
      }

      private Vector<Triangle> binaryReader() {
            // TODO Auto-generated method stub
            long tri_cnt = 0;
            try {
                  // reading an unsigned int
                  int fourthByte = data.readByte(); // endianess: STL seems
to store
                  // the 4 byte unsigned as little
                  // endian
                  int thirdByte = data.readByte();
                  int secondByte = data.readByte();
                  int firstByte = data.readByte();

                  tri_cnt = ((long) (firstByte << 24 | secondByte << 16
                              | thirdByte << 8 | fourthByte)) & 0xFFFFFFFFL;

                  System.out.println("Número de Triangulos: "+tri_cnt);

            } catch (IOException e) {
                  System.out.println(e.getMessage()); // FIXME: handle
Exception
            }

            //Vector<Triangle> triangles = new Vector<Triangle>((int) 1,
(int)tri_cnt);
            //Vector<Triangle> triangles = new Vector<Triangle>((int)
tri_cnt);

            int cnt = 0;
            while (cnt < tri_cnt) {
                  Triangle tria = new Triangle();
                  ++cnt;
                  //System.out.println("Atual: "+cnt);
                  try {
                        Vertex ver = new Vertex();
                        ver.setX(getLEFloat(data));
                        ver.setY(getLEFloat(data));
                        ver.setZ(getLEFloat(data));
                        tria.setNormal(ver);

                        ver = new Vertex();
                        ver.setX(getLEFloat(data));
                        ver.setY(getLEFloat(data));
                        ver.setZ(getLEFloat(data));
                        tria.setVertex1(ver);

                        ver = new Vertex();
                        ver.setX(getLEFloat(data));
                        ver.setY(getLEFloat(data));
                        ver.setZ(getLEFloat(data));
                        tria.setVertex2(ver);

                        ver = new Vertex();
                        ver.setX(getLEFloat(data));
                        ver.setY(getLEFloat(data));
                        ver.setZ(getLEFloat(data));
                        tria.setVertex3(ver);
```

```
                        @SuppressWarnings("unused")
                        char ch = data.readChar(); // currently not used,
but is:
                        // 'attribute byte count'
                  } catch (IOException e) {
                        System.out.println(e.getMessage()); // FIXME:
handle Exception
                  }
                  triangles.add(tria);
            }
            finalize();
            return triangles;
      }

      private Vector<Triangle> asciiReader() {
            // TODO Auto-generated method stub
            try {
                  data.close();
                  file.close();
            } catch (IOException e) {
                  System.out.println(e.getMessage()); // FIXME: handle
Exception
            }

            // ascii reading demands different reader
            BufferedReader reader;
            try {
                  reader = new BufferedReader(new FileReader(pathname));
            } catch (IOException e) {
                  System.out.println(e.getMessage()); // FIXME: handle
Exception
                  return null;
            }

            Vector<Triangle> triangles = new Vector<Triangle>();
            Triangle tria = null;
            String line = "";
            int cnt = 0;
            while (true) {
                  try {
                        line = reader.readLine();
                        if (line == null)
                              break;

                        // parsing
                        if (line.startsWith("solid")) // maybe .tolower()
is needed here
                              continue; // header
                        if (line.contains("endloop")) // maybe .tolower()
is needed here
                              continue; // header
                        if (line.contains("endfacet")) {// maybe .tolower()
is needed
                              // here
                              triangles.add(tria);
                              continue; // header
                        }
                        if (line.contains("endsolid"))
                              break;
```

```
                        if (line.contains("outer")) // maybe .tolower() is
needed here
                              continue; // header
                        if (line.contains("facet")) {
                              tria = new Triangle();
                              Vertex ver = new Vertex();
                              cnt = 0;

                              StringTokenizer token = new
StringTokenizer(line);
                              token.nextToken(); // facet string
                              token.nextToken(); // normal string
                              ver.setX(new
Float(token.nextToken()).floatValue());
                              ver.setY(new
Float(token.nextToken()).floatValue());
                              ver.setZ(new
Float(token.nextToken()).floatValue());
                              tria.setNormal(ver);
                              continue;
                        }
                        if (line.contains("vertex")) {
                              ++cnt;
                              Vertex ver = new Vertex();

                              StringTokenizer token = new
StringTokenizer(line);
                              token.nextToken(); // vertex string
                              ver.setX(new
Float(token.nextToken()).floatValue());
                              ver.setY(new
Float(token.nextToken()).floatValue());
                              ver.setZ(new
Float(token.nextToken()).floatValue());
                              switch (cnt) {
                              case 1:
                                    tria.setVertex1(ver);
                                    break;
                              case 2:
                                    tria.setVertex2(ver);
                                    break;
                              case 3:
                                    tria.setVertex3(ver);
                                    break;
                              default:
                                    continue;
                              }
                              continue;
                        }
                  } catch (IOException e) {
                        System.out.println(e.getMessage()); // FIXME:
handle exception
                  }
            }

            return triangles;
      }

      private float getLEFloat(DataInputStream stream) {
            // float f = ByteBuffer.wrap( array ).order(
ByteOrder.LITTLE_ENDIAN
```

```
            // ).getFloat(); little endian float reading
            byte[] array = new byte[4];
            for (int i = 0; i < 4; ++i) {
                  try {
                        array[i] = stream.readByte();
                  } catch (IOException e) {
                        System.out.println(e.getMessage()); // FIXME:
handle exception
                  }

            }
            return
ByteBuffer.wrap(array).order(ByteOrder.LITTLE_ENDIAN).getFloat();
      }

      protected void finalize() {// to close the file
            try {
                  data.close();
                  file.close();
            } catch (IOException e) {
                  System.out.println(e.getMessage()); // FIXME: handle
Exception
            }
      }
}
```

## Classe STLWriter – Classe cuja função é salvar um arquivo STL

```
package ReadSaveSTL;

import Objects.Triangle;
//import Objects.Vertex;
import java.io.DataOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Vector;

/**
 * @author
 */
public class STLWriter {
	private FileOutputStream file;
	private DataOutputStream data;
	//private String pathname;
	private String header;

	public final static int BINARY = 0;
	public final static int ASCII = 1;

	/**
	 * @param pathname
	 */
	public STLWriter(String pathname) {
		//this.pathname = pathname;
		try {
			file = new FileOutputStream(pathname);
			data = new DataOutputStream(file);
		} catch (FileNotFoundException e) {
			System.out.println(e.getMessage()); // FIXME: handle
Exception
		}
	}

	protected void finalize() {
		try {
			data.close();
			file.close();
		} catch (IOException e) {
			System.out.println(e.getMessage()); // FIXME: handle
Exception
		}
	}

	/**
	 * @param triangles
	 */
	public void writeSTL(Vector<Triangle> triangles) {
		writeSTL(triangles, BINARY);
	}

	/**
	 * @param triangles
	 * @param filetype
```

```
	 */
	public void writeSTL(Vector<Triangle> triangles, int filetype) {
		if (filetype == BINARY)
			writeBinarySTL(triangles);
		else
			writeAsciiSTL(triangles);
	}

	/**
	 * @param header
	 */
	public void setHeader(String header) {
		this.header = header;
	}

	/**
	 * @param triangles
	 */
	private void writeBinarySTL(Vector<Triangle> triangles) {
		try {
			// 80 byte String header
			if (header.length() == 0) {
				for (int i = 0; i < 10; ++i)
					data.writeBytes("header  ");
			} else
				data.writeBytes(header);

			// num of triangles
			int dummy = triangles.size() & 0xFF;
			byte by = (byte) (dummy & 0xFF);
			data.write(by);
			dummy = triangles.size() & 0xFF00;
			by = (byte) (dummy >> 4 & 0xFF);
			data.write(by);
			dummy = triangles.size() & 0xFF0000;
			by = (byte) (dummy >> 8 & 0xFF);
			data.write(by);
			dummy = triangles.size() & 0xFF000000;
			by = (byte) (dummy >> 12 & 0xFF);
			data.write(by);

			// triangles
			for (int i = 0; i < triangles.size(); ++i) {
				putLEFloat(data,
triangles.elementAt(i).getNormal().getX());
				putLEFloat(data,
triangles.elementAt(i).getNormal().getY());
				putLEFloat(data,
triangles.elementAt(i).getNormal().getY());

				putLEFloat(data,
triangles.elementAt(i).getVertex1().getX());
				putLEFloat(data,
triangles.elementAt(i).getVertex1().getY());
				putLEFloat(data,
triangles.elementAt(i).getVertex1().getZ());

				putLEFloat(data,
triangles.elementAt(i).getVertex2().getX());
				putLEFloat(data,
triangles.elementAt(i).getVertex2().getY());
```

```
                        putLEFloat(data,
triangles.elementAt(i).getVertex2().getZ());

                        putLEFloat(data,
triangles.elementAt(i).getVertex3().getX());
                        putLEFloat(data,
triangles.elementAt(i).getVertex3().getY());
                        putLEFloat(data,
triangles.elementAt(i).getVertex3().getZ());

                        data.writeChar(0);
                  }

            } catch (IOException e) {
                  System.out.println(e.getMessage()); // FIXME: handle
Exception
            }
      }

      /**
       * @param triangles
       */
      private void writeAsciiSTL(Vector<Triangle> triangles) {
            try {
                  // 80 byte String header
                  for (int i = 0; i < 10; ++i)
                        data.writeBytes("solid   ");
                  data.writeBytes(System.getProperty("line.separator"));

                  // triangle data
                  for (int i = 0; i < triangles.size(); ++i) {
                        data.writeBytes(" facet normal "
                                    +
triangles.elementAt(i).getNormal().getX() + " "
                                    +
triangles.elementAt(i).getNormal().getY() + " "
                                    +
triangles.elementAt(i).getNormal().getZ()
                                    + System.getProperty("line.separator"));
                        data.writeBytes("   outer loop"
                                    + System.getProperty("line.separator"));
                        data.writeBytes("     vertex "
                                    +
triangles.elementAt(i).getVertex1().getX() + " "
                                    +
triangles.elementAt(i).getVertex1().getY() + " "
                                    +
triangles.elementAt(i).getVertex1().getZ()
                                    + System.getProperty("line.separator"));
                        data.writeBytes("     vertex "
                                    +
triangles.elementAt(i).getVertex2().getX() + " "
                                    +
triangles.elementAt(i).getVertex2().getY() + " "
                                    +
triangles.elementAt(i).getVertex2().getZ()
                                    + System.getProperty("line.separator"));
                        data.writeBytes("     vertex "
                                    +
triangles.elementAt(i).getVertex3().getX() + " "
```

```
                                        +
triangles.elementAt(i).getVertex3().getY() + " "
                                        +
triangles.elementAt(i).getVertex3().getZ()
                                        + System.getProperty("line.separator"));
                            data.writeBytes("   endloop"
                                        + System.getProperty("line.separator"));
                            data.writeBytes(" endfacet"
                                        + System.getProperty("line.separator"));
                      }
                      data.writeBytes("endsolid");
                } catch (IOException e) {
                      System.out.println(e.getMessage()); // FIXME: handle
Exception
                }

          }

          private void putLEFloat(DataOutputStream stream, float f) {
                // float f = ByteBuffer.wrap( array ).order(
ByteOrder.LITTLE_ENDIAN
                // ).getFloat(); little endian float reading
                byte[] b =
ByteBuffer.allocate(4).order(ByteOrder.LITTLE_ENDIAN)
                            .putFloat(f).array();

                try {
                      stream.write(b);
                } catch (IOException e) {
                      System.out.println(e.getMessage()); // FIXME: handle
exception
                }
          }

}
```

**Classe burrRecognition – Classe cuja função é fazer o reconhecimento das rebarbas**

```
package Data;

import java.util.Vector;
import Objects.Triangle;
//import Objects.Vertex;
import Objects.Vertex;
import ReadSaveSTL.STLWriter;

public class BurrRecognition {

	public Vector<Triangle> CADModel;
	public Vector<Triangle> TransformedSceneModel = new
Vector<Triangle>();
	public Vector<Triangle> Burrs = new Vector<Triangle>();
	public Vector<Vertex> SceneModelAllPoints = new Vector<Vertex>();
	public Vector<Vertex> SceneModelPoints = new Vector<Vertex>();
	public Vector<Vertex> BurrsPoints = new Vector<Vertex>();
	public double Tolerance;

	public BurrRecognition(Vector<Triangle> CADModel,
                                    Vector<Triangle>
TransformedSceneModel){

		this.CADModel = CADModel;
		this.TransformedSceneModel = TransformedSceneModel;
		Tolerance = 0.6;

	}

	public void Execute(){

		System.out.println("\nCAD Model ");

		for (int ii = 0; ii < CADModel.size(); ++ii) {

			System.out.println("\nTriangle "+ii);
			System.out.print("Normal: ");

	System.out.print(CADModel.elementAt(ii).getNormal().getX());
			System.out.print(" ");

	System.out.print(CADModel.elementAt(ii).getNormal().getY());
			System.out.print(" ");

	System.out.println(CADModel.elementAt(ii).getNormal().getZ());

			System.out.print("Vertex1: ");

	System.out.print(CADModel.elementAt(ii).getVertex1().getX());
			System.out.print(" ");

	System.out.print(CADModel.elementAt(ii).getVertex1().getY());
			System.out.print(" ");

	System.out.println(CADModel.elementAt(ii).getVertex1().getZ());

			System.out.print("Vertex2: ");
```

```
        System.out.print(CADModel.elementAt(ii).getVertex2().getX());
                    System.out.print(" ");

        System.out.print(CADModel.elementAt(ii).getVertex2().getY());
                    System.out.print(" ");

        System.out.println(CADModel.elementAt(ii).getVertex2().getZ());

                    System.out.print("Vertex3: ");

        System.out.print(CADModel.elementAt(ii).getVertex3().getX());
                    System.out.print(" ");

        System.out.print(CADModel.elementAt(ii).getVertex3().getY());
                    System.out.print(" ");

        System.out.println(CADModel.elementAt(ii).getVertex3().getZ());
              }

              for (int i = 0; i < TransformedSceneModel.size(); ++i) {

                    Vertex TempVertex;
                    TempVertex = new Vertex();

        TempVertex.setX(TransformedSceneModel.elementAt(i).getVertex1().getX(
));

        TempVertex.setY(TransformedSceneModel.elementAt(i).getVertex1().getY(
));

        TempVertex.setZ(TransformedSceneModel.elementAt(i).getVertex1().getZ(
));
                    SceneModelAllPoints.add(TempVertex);

                    TempVertex = new Vertex();

        TempVertex.setX(TransformedSceneModel.elementAt(i).getVertex2().getX(
));

        TempVertex.setY(TransformedSceneModel.elementAt(i).getVertex2().getY(
));

        TempVertex.setZ(TransformedSceneModel.elementAt(i).getVertex2().getZ(
));
                    SceneModelAllPoints.add(TempVertex);

                    TempVertex = new Vertex();

        TempVertex.setX(TransformedSceneModel.elementAt(i).getVertex3().getX(
));

        TempVertex.setY(TransformedSceneModel.elementAt(i).getVertex3().getY(
));

        TempVertex.setZ(TransformedSceneModel.elementAt(i).getVertex3().getZ(
));
                    SceneModelAllPoints.add(TempVertex);

              }
```

```
            System.out.println("All points: "+SceneModelAllPoints.size());

            System.out.println("Separating the different points from Scene
Model");
            SceneModelPoints.add(SceneModelAllPoints.elementAt(0));

            int k;
            for (int i = 0; i < SceneModelAllPoints.size(); ++i) {
                  k = 0;
                  for (int j = 0; j < SceneModelPoints.size(); ++j) {

                        if (SceneModelAllPoints.elementAt(i).getX() ==
SceneModelPoints.elementAt(j).getX()
                                    &&
SceneModelAllPoints.elementAt(i).getY() ==
SceneModelPoints.elementAt(j).getY()
                                    &&
SceneModelAllPoints.elementAt(i).getZ() ==
SceneModelPoints.elementAt(j).getZ()) {
                        }
                        else {
                              k = k + 1;
                        }
                  }

                  if (k == SceneModelPoints.size()) {

      SceneModelPoints.add(SceneModelAllPoints.elementAt(i));
                  }
                  else {
                  }

            }

            System.out.println("\nScene Points ");

            for (int ii = 0; ii < SceneModelPoints.size(); ++ii) {

                  System.out.println("\nPoint"+ii);

                  System.out.print("Vertex: ");
                  System.out.print(SceneModelPoints.elementAt(ii).getX());
                  System.out.print(" ");
                  System.out.print(SceneModelPoints.elementAt(ii).getY());
                  System.out.print(" ");

      System.out.println(SceneModelPoints.elementAt(ii).getZ());
            }

            System.out.println("\nTolerance of Scene Model comparing to CAD
Model "+Tolerance);

            Triangle TempTria = new Triangle();
            Vertex TempVertex;

            int a, e;

            RecognitionAlgorithm Recognition = new
RecognitionAlgorithm(Tolerance);

            //for (int i = 0; i < 2; ++i){
```

```
            for (int i = 0; i < SceneModelPoints.size(); ++i){

                  e = 1;

                  for (int j = 0; j < CADModel.size(); ++j){

                        System.out.println("\nComparing Scene Triangle
"+i+" with CAD Triangle "+j);


      TempTria.setNormal(CADModel.elementAt(j).getNormal());

      TempTria.setVertex1(CADModel.elementAt(j).getVertex1());

      TempTria.setVertex2(CADModel.elementAt(j).getVertex2());

      TempTria.setVertex3(CADModel.elementAt(j).getVertex3());

                        TempVertex = new Vertex();
                        TempVertex = SceneModelPoints.elementAt(i);

                        a = Recognition.RecognitionCalculations(TempTria,
TempVertex);
                        int b;

                        if(a==1){
                              b = 0;
                        }
                        else{
                              b = 1;
                        }
                        e = e*b;
                        /**
                        System.out.print("\nScene Triangle "+i);
                        System.out.println("\nCAD Triangle "+j);
                        System.out.print("\nE =  "+e);
                        */

                  }
                  System.out.print("------------------------------");
                  System.out.print("\nE FINAL =  "+e);

                  if(e == 0){
                        System.out.println("\nPonto "+i+" nao é rebarba");

                  //    System.out.println("\nScene Triangle "+i+" add as
burr");
                  //    Burrs.add(TransformedSceneModel.elementAt(i));
                  }
                  else{
                        System.out.println("\nPonto "+i+" é rebarba");
                        BurrsPoints.add(SceneModelPoints.elementAt(i));
                  }
                  System.out.print("------------------------------");

            }

            System.out.println("\nBurr number of points =
"+BurrsPoints.size());
            if(BurrsPoints.size()==0){
            }
```

```
            else{

                  for (int ii = 0; ii < BurrsPoints.size(); ++ii) {

                        System.out.println("\nBurrPoints"+ii);

                        System.out.print("Vertex: ");
                        System.out.print(BurrsPoints.elementAt(ii).getX());
                        System.out.print(" ");
                        System.out.print(BurrsPoints.elementAt(ii).getY());
                        System.out.print(" ");

      System.out.println(BurrsPoints.elementAt(ii).getZ());
                  }
            }

            System.out.println("\nSeparating the Triangles that have the
burr points");

            for (int i = 0; i < TransformedSceneModel.size(); ++i) {

                  for (int j = 0; j < BurrsPoints.size(); ++j) {

                        int k1, k2, k3;
                        k=0;

                        if
(TransformedSceneModel.elementAt(i).getVertex1().getX() ==
BurrsPoints.elementAt(j).getX()
                                    &&
TransformedSceneModel.elementAt(i).getVertex1().getY() ==
BurrsPoints.elementAt(j).getY()
                                    &&
TransformedSceneModel.elementAt(i).getVertex1().getZ() ==
BurrsPoints.elementAt(j).getZ()) {
                              k1=1;
                        }
                        else{
                              k1=0;
                        }
                        if
(TransformedSceneModel.elementAt(i).getVertex2().getX() ==
BurrsPoints.elementAt(j).getX()
                                    &&
TransformedSceneModel.elementAt(i).getVertex2().getY() ==
BurrsPoints.elementAt(j).getY()
                                    &&
TransformedSceneModel.elementAt(i).getVertex2().getZ() ==
BurrsPoints.elementAt(j).getZ()) {
                              k2=1;
                        }
                        else{
                              k2=0;
                        }
                        if
(TransformedSceneModel.elementAt(i).getVertex3().getX() ==
BurrsPoints.elementAt(j).getX()
                                    &&
TransformedSceneModel.elementAt(i).getVertex3().getY() ==
BurrsPoints.elementAt(j).getY()
```

```
                                        &&
TransformedSceneModel.elementAt(i).getVertex3().getZ() ==
BurrsPoints.elementAt(j).getZ()) {
                                k3=1;
                        }
                        else{
                                k3=0;
                        }

                        k = k1+k2+k3;

                        if(k==1){
                                Burrs.add(TransformedSceneModel.elementAt(i));

                        }
                        else{
                        }
                }
        }

        System.out.println("\nBurr number of triangles =
"+Burrs.size());

        STLWriter writer3 = new STLWriter("d:/rebarbas.stl");
        writer3.writeSTL(Burrs, STLWriter.ASCII);

    }
}
```

**Classe Centroids_Calculations – Classe cuja função é calcular o centróide de cada malha**

```
package Data;

import Objects.Vertex;
import java.util.Vector;

public class Centroids_Calculations {
	/**
	 Centroids_Calculations Class

	 - Main Objective: Repositioning the coordinates of the points
according the centroid.
	   - Generate the Centroids of the Pointcloud
	   - Recalculation of the coordinates of the points according the
centroid calculated
	 */

	private Vector<Vertex> EigthVertexes;
	private Vector<Vertex> EigthVertexesCentroid = new Vector<Vertex>();
	private Vertex Centroid = new Vertex();

	public Centroids_Calculations(Vector<Vertex> EigthVertexes){

		this.EigthVertexes = EigthVertexes;
	}

	public Vertex GetCentroid(){

		float x=0, y=0, z=0;

		for (int i = 0; i < EigthVertexes.size(); ++i){
			x = x + EigthVertexes.elementAt(i).getX();
		}
		x = x/EigthVertexes.size();

		for (int i = 0; i < EigthVertexes.size(); ++i){
			y = y + EigthVertexes.elementAt(i).getY();
		}
		y = y/EigthVertexes.size();

		for (int i = 0; i < EigthVertexes.size(); ++i){
			z = z + EigthVertexes.elementAt(i).getZ();
		}
		z = z/EigthVertexes.size();

		Centroid.setX(x);
		Centroid.setY(y);
		Centroid.setZ(z);

		return Centroid;

	}

	public Vector<Vertex> GetRepositioningByCentrois(){

		Vertex TempPoint = new Vertex();

		for (int i = 0; i < EigthVertexes.size(); ++i){
```

```
                TempPoint = new Vertex();

                float x=0, y=0, z=0;

                x = EigthVertexes.elementAt(i).getX() - Centroid.getX();
                y = EigthVertexes.elementAt(i).getY() - Centroid.getY();
                z = EigthVertexes.elementAt(i).getZ() - Centroid.getZ();

                TempPoint.setX(x);
                TempPoint.setY(y);
                TempPoint.setZ(z);

                EigthVertexesCentroid.add(TempPoint);
            }
            return EigthVertexesCentroid;
        }
}
```

## Classe DataManagement – Classe cuja função é fazer a entrada de dados

```
package Data;

import Objects.Triangle;
import Objects.Vertex;
import ReadSaveSTL.STLReader;
import java.util.Vector;

public class DataManagement {

	public Vector<Triangle> Data;
	public Vector<Vertex> EigthVertexesCAD = new Vector<Vertex>();
	public Vector<Vertex> EigthVertexesScene = new Vector<Vertex>();
	public boolean Exception=true;

	public boolean readSTLFile(String pathname) {

		try {
		STLReader ReadSTL = new STLReader(pathname);
		Data = ReadSTL.ReadFile();
		} catch (Exception e) {
			//e.printStackTrace();
			Exception=false;
		}

		if(Exception==false){
			return false;
		}
		else{
			//Data = ReadSTL.ReadFile();
			return true;
		}
	}

	public Vector<Vertex> GetVertexesCAD() {

		Vertex TempVertex = new Vertex();

		//Pto 1
		TempVertex = new Vertex();
		TempVertex.setX((float)50); TempVertex.setY((float) 0);
TempVertex.setZ((float) 30);
		EigthVertexesCAD.add(TempVertex);

		//Pto 2
		TempVertex = new Vertex();
		TempVertex.setX((float)0); TempVertex.setY((float) 0);
TempVertex.setZ((float) 30);
		EigthVertexesCAD.add(TempVertex);

		//Pto 3
		TempVertex = new Vertex();
		TempVertex.setX((float)0); TempVertex.setY((float) 50);
TempVertex.setZ((float) 30);
		EigthVertexesCAD.add(TempVertex);
		//**
		//Pto 4
		TempVertex = new Vertex();
```

```
            TempVertex.setX((float)50); TempVertex.setY((float) 50);
TempVertex.setZ((float) 30);
            EigthVertexesCAD.add(TempVertex);

            //Pto 5
            TempVertex = new Vertex();
            TempVertex.setX((float)50); TempVertex.setY((float) 0);
TempVertex.setZ((float) 0);
            EigthVertexesCAD.add(TempVertex);

            //Pto 6
            TempVertex = new Vertex();
            TempVertex.setX((float)0); TempVertex.setY((float) 0);
TempVertex.setZ((float) 0);
            EigthVertexesCAD.add(TempVertex);

            //Pto 7
            TempVertex = new Vertex();
            TempVertex.setX((float)0); TempVertex.setY((float) 50);
TempVertex.setZ((float) 0);
            EigthVertexesCAD.add(TempVertex);

            //Pto 8
            TempVertex = new Vertex();
            TempVertex.setX((float)50); TempVertex.setY((float) 50);
TempVertex.setZ((float) 0);
            EigthVertexesCAD.add(TempVertex);
            //*/
            System.out.println("\nCADModel Tranformation Points:\n");

            for (int i = 0; i < EigthVertexesCAD.size(); ++i){
                  System.out.println("   Point "+i+":
X:"+EigthVertexesCAD.get(i).getX()+"  " +
                                    "Y:"+EigthVertexesCAD.get(i).getY()+"  " +
                                    "Z:"+EigthVertexesCAD.get(i).getZ());

            }
            return EigthVertexesCAD;

      }

      // Modelo com rebarbas CAD
      /**
      public Vector<Vertex> GetVertexesScene() {

            Vertex TempVertex = new Vertex();

            //Pto 1
            TempVertex = new Vertex();
            TempVertex.setX((float)50); TempVertex.setY((float) 100);
TempVertex.setZ((float) 30);
            EigthVertexesScene.add(TempVertex);

            //Pto 2
            TempVertex = new Vertex();
            TempVertex.setX((float)10); TempVertex.setY((float) 130);
TempVertex.setZ((float) 30);
            EigthVertexesScene.add(TempVertex);

            //Pto 3
            TempVertex = new Vertex();
```

```
            TempVertex.setX((float)40); TempVertex.setY((float) 170);
TempVertex.setZ((float) 30);
            EigthVertexesScene.add(TempVertex);
            //**
            //Pto 4
            TempVertex = new Vertex();
            TempVertex.setX((float)80); TempVertex.setY((float) 140);
TempVertex.setZ((float) 30);
            EigthVertexesScene.add(TempVertex);

            //Pto 5
            TempVertex = new Vertex();
            TempVertex.setX((float)50); TempVertex.setY((float) 100);
TempVertex.setZ((float) 0);
            EigthVertexesScene.add(TempVertex);

            //Pto 6
            TempVertex = new Vertex();
            TempVertex.setX((float)10); TempVertex.setY((float) 130);
TempVertex.setZ((float) 0);
            EigthVertexesScene.add(TempVertex);

            //Pto 7
            TempVertex = new Vertex();
            TempVertex.setX((float)40); TempVertex.setY((float) 170);
TempVertex.setZ((float) 0);
            EigthVertexesScene.add(TempVertex);

            //Pto 8
            TempVertex = new Vertex();
            TempVertex.setX((float)80); TempVertex.setY((float) 140);
TempVertex.setZ((float) 0);
            EigthVertexesScene.add(TempVertex);
            */


      // Modelo escaneado PECA 1
      //**
      public Vector<Vertex> GetVertexesScene() {

            Vertex TempVertex = new Vertex();

            //Pto 1
            TempVertex = new Vertex();
            TempVertex.setX((float)-30.7469);
TempVertex.setY((float)94.6378); TempVertex.setZ((float)373.7508);
            EigthVertexesScene.add(TempVertex);

            //Pto 2
            TempVertex = new Vertex();
            TempVertex.setX((float)-23.3467);
TempVertex.setY((float)142.5727); TempVertex.setZ((float)362.9974);
            EigthVertexesScene.add(TempVertex);

            //Pto 3
            TempVertex = new Vertex();
            TempVertex.setX((float)-64.1849);
TempVertex.setY((float)154.7267); TempVertex.setZ((float)388.5756);
            EigthVertexesScene.add(TempVertex);
            //**
            //Pto 4
```

```
            TempVertex = new Vertex();
            TempVertex.setX((float)-71.5736);
TempVertex.setY((float)106.8271); TempVertex.setZ((float)399.3171);
            EigthVertexesScene.add(TempVertex);

            //Pto 5
            TempVertex = new Vertex();
            TempVertex.setX((float)-14.1454);
TempVertex.setY((float)97.7383); TempVertex.setZ((float)398.6805);
            EigthVertexesScene.add(TempVertex);

            //Pto 6
            TempVertex = new Vertex();
            TempVertex.setX((float)-6.8037);
TempVertex.setY((float)145.5970); TempVertex.setZ((float)387.8697);
            EigthVertexesScene.add(TempVertex);

            //Pto 7
            TempVertex = new Vertex();
            TempVertex.setX((float)-47.6657);
TempVertex.setY((float)157.7952); TempVertex.setZ((float)413.6275);
            EigthVertexesScene.add(TempVertex);

            //Pto 8
            TempVertex = new Vertex();
            TempVertex.setX((float)-54.9961);
TempVertex.setY((float)109.9723); TempVertex.setZ((float)424.4264);
            EigthVertexesScene.add(TempVertex);
            //*/
      /**
            public Vector<Vertex> GetVertexesScene() {

                  Vertex TempVertex = new Vertex();

                  //Pto 1
                  TempVertex = new Vertex();
                  TempVertex.setX((float)76.9785);
TempVertex.setY((float)57.4868); TempVertex.setZ((float)275.3293);
                  EigthVertexesScene.add(TempVertex);

                  //Pto 2
                  TempVertex = new Vertex();
                  TempVertex.setX((float)93.8041);
TempVertex.setY((float)103.8641); TempVertex.setZ((float)269.8642);
                  EigthVertexesScene.add(TempVertex);

                  //Pto 3
                  TempVertex = new Vertex();
                  TempVertex.setX((float)48.1528);
TempVertex.setY((float)121.3148); TempVertex.setZ((float)278.2834);
                  EigthVertexesScene.add(TempVertex);
                  //**
                  //Pto 4
                  TempVertex = new Vertex();
                  TempVertex.setX((float)31.3221);
TempVertex.setY((float)75.0332); TempVertex.setZ((float)283.7452);
                  EigthVertexesScene.add(TempVertex);

                  //Pto 5
                  TempVertex = new Vertex();
```

```
                  TempVertex.setX((float)82.8388);
TempVertex.setY((float)58.6552); TempVertex.setZ((float)304.1601);
                  EigthVertexesScene.add(TempVertex);

                  //Pto 6
                  TempVertex = new Vertex();
                  TempVertex.setX((float)99.7175);
TempVertex.setY((float)105.0534); TempVertex.setZ((float)298.9353);
                  EigthVertexesScene.add(TempVertex);

                  //Pto 7
                  TempVertex = new Vertex();
                  TempVertex.setX((float)53.9376);
TempVertex.setY((float)122.5363); TempVertex.setZ((float)307.2301);
                  EigthVertexesScene.add(TempVertex);

                  //Pto 8
                  TempVertex = new Vertex();
                  TempVertex.setX((float)37.0548);
TempVertex.setY((float)76.2338); TempVertex.setZ((float)312.4518);
                  EigthVertexesScene.add(TempVertex);


      */

            System.out.println("\nSceneModel Tranformation Points:\n");

            for (int i = 0; i < EigthVertexesScene.size(); ++i){
                  System.out.println("   Point "+i+":
X:"+EigthVertexesScene.get(i).getX()+"  " +
                              "Y:"+EigthVertexesScene.get(i).getY()+"  "
+
                              "Z:"+EigthVertexesScene.get(i).getZ());

            }


            return EigthVertexesScene;

      }

}
```

### Classe HornAlignment– Classe cuja função é alinhar as malhas

```
package Data;

import java.util.Vector;

import Data.Centroids_Calculations;
import MathCalculations.Ferrari_Method;
import MathCalculations.Gauss_Method;
import MathCalculations.Horn_Algorithm;
import MathCalculations.Matrix_3x3_Determinant;
import MathCalculations.Matrix_4x4_Determinant;
import Objects.Quaternion;
import Objects.Triangle;
import Objects.Vertex;
import ReadSaveSTL.STLWriter;
//import PrintMethods.SystemPrint;

public class HornAlignment {

	public Vector<Triangle> CADModel;
	public Vector<Triangle> SceneModel;
	//CADModel, with the triangle formation for alignment

	public Vector<Vertex> EigthVertexesCAD = new Vector<Vertex>();
	public Vector<Vertex> EigthVertexesScene = new Vector<Vertex>();
	//Points for Horn Algorithm

	public Vector<Vertex> EigthVertexesCADCentroid = new
Vector<Vertex>();
	public Vector<Vertex> EigthVertexesSceneCentroid = new
Vector<Vertex>();
	//Points for Horn Algorithm

	public Vector<Triangle> RotatedSceneModel = new Vector<Triangle>();
	//CADModel, with the triangle formation for the rotation
	public Vector<Triangle> TransformedSceneModel = new
Vector<Triangle>();
	//CADModel, with the triangle formation final transformation results

	public Vertex TranslationVector = new Vertex();
	public Quaternion RotationQuaternion = new Quaternion();



	public HornAlignment(Vector<Triangle> CADModel,
	                     Vector<Triangle> SceneModel,
	                     Vector<Vertex> EigthVertexesCAD,
	                     Vector<Vertex> EigthVertexesScene){

		this.CADModel = CADModel;
		this.SceneModel = SceneModel;
		this.EigthVertexesCAD = EigthVertexesCAD;
		this.EigthVertexesScene = EigthVertexesScene;

	}

	public void Execute(){

		System.out.println("\nGetting CAD Model Centroid");
```

```
            Centroids_Calculations CADCentroid = new
Centroids_Calculations((Vector<Vertex>)EigthVertexesCAD);
            Vertex CADModelCentroid = new Vertex();
            CADModelCentroid = CADCentroid.GetCentroid();

            System.out.println("Getting Scene Model Centroid");

            Centroids_Calculations SceneCentroid = new
Centroids_Calculations((Vector<Vertex>)EigthVertexesScene);
            Vertex SceneModelCentroid = new Vertex();
            SceneModelCentroid = SceneCentroid.GetCentroid();

            TranslationVector.setX(SceneModelCentroid.getX()-
CADModelCentroid.getX());
            TranslationVector.setY(SceneModelCentroid.getY()-
CADModelCentroid.getY());
            TranslationVector.setZ(SceneModelCentroid.getZ()-
CADModelCentroid.getZ());


            System.out.println("CADModel Centroid:
X:"+CADModelCentroid.getX()+
                                                                    "
Y:"+CADModelCentroid.getY()+
                                                                    "
Z:"+CADModelCentroid.getZ());

            System.out.println("SceneModel Centroid:
X:"+SceneModelCentroid.getX()+
                                                                "
Y:"+SceneModelCentroid.getY()+
                                                                "
Z:"+SceneModelCentroid.getZ());

            System.out.println("Translation Vector:
X:"+TranslationVector.getX()+
                                                              "
Y:"+TranslationVector.getY()+
                                                            "
Z:"+TranslationVector.getZ());

            System.out.println("Getting the new coordinates for the Eight
Points");

            EigthVertexesCADCentroid =
CADCentroid.GetRepositioningByCentrois();
            EigthVertexesSceneCentroid =
SceneCentroid.GetRepositioningByCentrois();

            System.out.println("\nCalculationg The M and N Matrix");

            double MatrixM[][] = null;
            double MatrixN[][] = null;
            try {
                  Horn_Algorithm Horn_Transformation = new Horn_Algorithm(
                              (Vector<Vertex>)EigthVertexesSceneCentroid,
                              (Vector<Vertex>)EigthVertexesCADCentroid);

                  MatrixM=Horn_Transformation.getM();
                  MatrixN=Horn_Transformation.getN();
```

```
            } catch (Exception e) {
                  //e.printStackTrace();
            }

            System.out.println("\nMatrix M:");

            for (int i = 0; i < MatrixM[0].length; ++i){ //Colunas

                  System.out.println("");
                  for (int j = 0; j < MatrixM.length; ++j){//Linhas

                        System.out.print(" "+MatrixM[i][j]+"");

                  }
            }

            System.out.println("\n\nMatrix N:");

            for (int i = 0; i < MatrixN[0].length; ++i){ //Colunas

                  System.out.println("");
                  for (int j = 0; j < MatrixN.length; ++j){//Linhas

                        System.out.print(" "+MatrixN[i][j]+"");

                  }
            }

            //Calculating the EingenValues
            System.out.println("\nGenerating the Four Degree Polynomial
Equation:");

            double c0, c1, c2, c3;

            c3 = MatrixN[0][0]+MatrixN[1][1]+MatrixN[2][2]+MatrixN[3][3];

            c2 = -
2*(MatrixM[0][0]*MatrixM[0][0]+MatrixM[0][1]*MatrixM[0][1]+MatrixM[0][2]*Ma
trixM[0][2]+

MatrixM[1][0]*MatrixM[1][0]+MatrixM[1][1]*MatrixM[1][1]+MatrixM[1][2]*Matri
xM[1][2]+

MatrixM[2][0]*MatrixM[2][0]+MatrixM[2][1]*MatrixM[2][1]+MatrixM[2][2]*Matri
xM[2][2]);

            Matrix_3x3_Determinant MDeterminant = new
Matrix_3x3_Determinant(MatrixM);
            c1 = -8*MDeterminant.GetDeterminant();

            Matrix_4x4_Determinant NDeterminant = new
Matrix_4x4_Determinant(MatrixN);
            c0 = NDeterminant.GetDeterminant();


            System.out.println("\nSolving the Four Degree Polynomial
Equation");

            Ferrari_Method FourDegreePolynomial = new
Ferrari_Method((double) 1, (double) c3,
                        (double) c2, (double) c1, (double) c0);
```

```
            System.out.println("\nX1 = "+FourDegreePolynomial.GetZero1()+"
+ "
                        +FourDegreePolynomial.GetZero1i()+" i ");
            System.out.println("X2 = "+FourDegreePolynomial.GetZero2()+" +
"
                        +FourDegreePolynomial.GetZero2i()+" i ");
            System.out.println("X3 = "+FourDegreePolynomial.GetZero3()+" +
"
                        +FourDegreePolynomial.GetZero3i()+" i ");
            System.out.println("X4 = "+FourDegreePolynomial.GetZero4()+" +
"
                        +FourDegreePolynomial.GetZero4i()+" i ");


            //Finding the most positive EigenValue
            System.out.println("\nCalculating the most positive
eigenvalue");

            double Eigenvalue=0;


      if(FourDegreePolynomial.GetZero1()>Eigenvalue&FourDegreePolynomial.Ge
tZero1i()==0){
                  Eigenvalue = FourDegreePolynomial.GetZero1();}
            else{}

      if(FourDegreePolynomial.GetZero2()>Eigenvalue&FourDegreePolynomial.Ge
tZero2i()==0){
                  Eigenvalue = FourDegreePolynomial.GetZero2();}
            else{}

      if(FourDegreePolynomial.GetZero3()>Eigenvalue&FourDegreePolynomial.Ge
tZero3i()==0){
                  Eigenvalue = FourDegreePolynomial.GetZero3();}
            else{}

      if(FourDegreePolynomial.GetZero4()>Eigenvalue&FourDegreePolynomial.Ge
tZero4i()==0){
                  Eigenvalue = FourDegreePolynomial.GetZero4();}
            else{}

            System.out.println("\nEigenvalue = "+Eigenvalue);

            //Calculating the EigenVector
            System.out.println("\nCalculating the eigenvector corresponding
to he " +
                        "the most positive eigenvalue");

            double MatrixIxE[][]=new double [][] {
                        {Eigenvalue, 0, 0, 0},
                        {0, Eigenvalue, 0, 0},
                        {0, 0, Eigenvalue, 0},
                        {0, 0, 0, Eigenvalue}
            };

            double a = MatrixN[0][0]-MatrixIxE[0][0];
            double e = MatrixN[0][1]-MatrixIxE[0][1];
            double h = MatrixN[0][2]-MatrixIxE[0][2];
            double j = MatrixN[0][3]-MatrixIxE[0][3];
            double b = MatrixN[1][1]-MatrixIxE[1][1];
```

```
            double f = MatrixN[1][2]-MatrixIxE[1][2];
            double i = MatrixN[1][3]-MatrixIxE[1][3];
            double c = MatrixN[2][2]-MatrixIxE[2][2];
            double g = MatrixN[2][3]-MatrixIxE[2][3];
            double d = MatrixN[3][3]-MatrixIxE[3][3];

            double MatrixF[][]=new double [][] {
                        {a, e, h, j},
                        {e, b, f, i},
                        {h, f, c, g},
                        {j, i, g, d}
                        };

            Gauss_Method LinearSystem = new Gauss_Method(MatrixF);

            RotationQuaternion = LinearSystem.GetQuaternion();

            System.out.println("\nQuaternion of rotation:
Q0:"+RotationQuaternion.getQ0()+
                        "Qx:"+RotationQuaternion.getQx()+
                        "Qy:"+RotationQuaternion.getQy()+
                        "Qz:"+RotationQuaternion.getQz());

            System.out.println("\nRotation Quaternion and Translation
Vector Calculated Successfully");

            System.out.println("\nApplying the Transformations on the Scan
STL Data");

            // O CALCULO É REALIZADO AQUI

            Quaternion_Rotation Rotation = new
Quaternion_Rotation(RotationQuaternion);


            System.out.println("Processo de Transformacao");

            for (int i2 = 0; i2 < SceneModel.size(); ++i2) {

                  //System.out.println("Triangulo "+i2);

                  Triangle TempTria = new Triangle();
                  Vertex TempVertex;
                  Vertex TempVertex2;
                  Vertex TempVertex3;

                  TempVertex = new Vertex();
                  TempVertex3 = new Vertex();


      TempVertex.setX(SceneModel.elementAt(i2).getNormal().getX());

      TempVertex.setY(SceneModel.elementAt(i2).getNormal().getY());

      TempVertex.setZ(SceneModel.elementAt(i2).getNormal().getZ());

                  //System.out.println("Normal");
                  //System.out.println("X: "+TempVertex.getX());
                  //System.out.println("Y: "+TempVertex.getY());
                  //System.out.println("Z: "+TempVertex.getZ());
```

```
                    TempVertex3 = Rotation.GetRotation(TempVertex);

                    //System.out.println("Normal Rotacionada");
                    //System.out.println("X: "+TempVertex3.getX());
                    //System.out.println("Y: "+TempVertex3.getY());
                    //System.out.println("Z: "+TempVertex3.getZ());


                    TempTria.setNormal(TempVertex3);

                    //----Vertice 1

                    TempVertex = new Vertex();
                    TempVertex2 = new Vertex();
                    TempVertex3 = new Vertex();


      TempVertex2.setX(SceneModel.elementAt(i2).getVertex1().getX()-
SceneModelCentroid.getX());

      TempVertex2.setY(SceneModel.elementAt(i2).getVertex1().getY()-
SceneModelCentroid.getY());

      TempVertex2.setZ(SceneModel.elementAt(i2).getVertex1().getZ()-
SceneModelCentroid.getZ());

                    //System.out.println("Ponto Deslocado do Centroid");
                    //System.out.println("X: "+TempVertex2.getX());
                    //System.out.println("Y: "+TempVertex2.getY());
                    //System.out.println("Z: "+TempVertex2.getZ());

                    TempVertex = Rotation.GetRotation(TempVertex2);

                    //System.out.println("Ponto Rotacionado");
                    //System.out.println("X: "+TempVertex.getX());
                    //System.out.println("Y: "+TempVertex.getY());
                    //System.out.println("Z: "+TempVertex.getZ());

                    TempVertex3.setX(TempVertex.getX()+
SceneModelCentroid.getX());
                    TempVertex3.setY(TempVertex.getY()+
SceneModelCentroid.getY());
                    TempVertex3.setZ(TempVertex.getZ()+
SceneModelCentroid.getZ());

                    //System.out.println("Ponto Rotacionado + Centroid");
                    //System.out.println("X: "+TempVertex3.getX());
                    //System.out.println("Y: "+TempVertex3.getY());
                    //System.out.println("Z: "+TempVertex3.getZ());

                    TempTria.setVertex1(TempVertex3);

                    //----Vertice 2

                    TempVertex = new Vertex();
                    TempVertex2 = new Vertex();
                    TempVertex3 = new Vertex();


      TempVertex2.setX(SceneModel.elementAt(i2).getVertex2().getX()-
SceneModelCentroid.getX());
```

```
      TempVertex2.setY(SceneModel.elementAt(i2).getVertex2().getY()-
SceneModelCentroid.getY());

      TempVertex2.setZ(SceneModel.elementAt(i2).getVertex2().getZ()-
SceneModelCentroid.getZ());

                  //System.out.println("Ponto Deslocado do Centroid");
                  //System.out.println("X: "+TempVertex2.getX());
                  //System.out.println("Y: "+TempVertex2.getY());
                  //System.out.println("Z: "+TempVertex2.getZ());

                  TempVertex = Rotation.GetRotation(TempVertex2);

                  //System.out.println("Ponto Rotacionado");
                  //System.out.println("X: "+TempVertex.getX());
                  //System.out.println("Y: "+TempVertex.getY());
                  //System.out.println("Z: "+TempVertex.getZ());

                  TempVertex3.setX(TempVertex.getX()+
SceneModelCentroid.getX());
                  TempVertex3.setY(TempVertex.getY()+
SceneModelCentroid.getY());
                  TempVertex3.setZ(TempVertex.getZ()+
SceneModelCentroid.getZ());

                  //System.out.println("Ponto Rotacionado + Centroid");
                  //System.out.println("X: "+TempVertex3.getX());
                  //System.out.println("Y: "+TempVertex3.getY());
                  //System.out.println("Z: "+TempVertex3.getZ());

                  TempTria.setVertex2(TempVertex3);

                  //----Vertice 3

                  TempVertex = new Vertex();
                  TempVertex2 = new Vertex();
                  TempVertex3 = new Vertex();


      TempVertex2.setX(SceneModel.elementAt(i2).getVertex3().getX()-
SceneModelCentroid.getX());

      TempVertex2.setY(SceneModel.elementAt(i2).getVertex3().getY()-
SceneModelCentroid.getY());

      TempVertex2.setZ(SceneModel.elementAt(i2).getVertex3().getZ()-
SceneModelCentroid.getZ());

                  //System.out.println("Ponto Deslocado do Centroid");
                  //System.out.println("X: "+TempVertex3.getX());
                  //System.out.println("Y: "+TempVertex3.getY());
                  //System.out.println("Z: "+TempVertex3.getZ());

                  TempVertex = Rotation.GetRotation(TempVertex2);

                  //System.out.println("Ponto Rotacionado");
                  //System.out.println("X: "+TempVertex.getX());
                  //System.out.println("Y: "+TempVertex.getY());
                  //System.out.println("Z: "+TempVertex.getZ());
```

```
                    TempVertex3.setX(TempVertex.getX()+
SceneModelCentroid.getX());
                    TempVertex3.setY(TempVertex.getY()+
SceneModelCentroid.getY());
                    TempVertex3.setZ(TempVertex.getZ()+
SceneModelCentroid.getZ());

                    //System.out.println("Ponto Rotacionado + Centroid");
                    //System.out.println("X: "+TempVertex3.getX());
                    //System.out.println("Y: "+TempVertex3.getY());
                    //System.out.println("Z: "+TempVertex3.getZ());

                    TempTria.setVertex3(TempVertex3);

                    //----

                    RotatedSceneModel.add(TempTria);
              }

              System.out.println("\nApplying the Translation:");

              for (int i2 = 0; i2 < SceneModel.size(); ++i2) {

                    //System.out.println("Triangulo "+i2);

                    Triangle TempTria = new Triangle();
                    Vertex TempVertex;
                    Vertex TempVertex2;

                    TempVertex = new Vertex();


        TempVertex.setX(RotatedSceneModel.elementAt(i2).getNormal().getX());

        TempVertex.setY(RotatedSceneModel.elementAt(i2).getNormal().getY());

        TempVertex.setZ(RotatedSceneModel.elementAt(i2).getNormal().getZ());

                    TempTria.setNormal(TempVertex);

                    //----Vertice 1

                    TempVertex = new Vertex();
                    TempVertex2 = new Vertex();


        TempVertex2.setX(RotatedSceneModel.elementAt(i2).getVertex1().getX())
;

        TempVertex2.setY(RotatedSceneModel.elementAt(i2).getVertex1().getY())
;

        TempVertex2.setZ(RotatedSceneModel.elementAt(i2).getVertex1().getZ())
;

                    TempVertex.setX(TempVertex2.getX()-
TranslationVector.getX());
                    TempVertex.setY(TempVertex2.getY()-
TranslationVector.getY());
                    TempVertex.setZ(TempVertex2.getZ()-
TranslationVector.getZ());
```

```
                    TempTria.setVertex1(TempVertex);

                    //----Vertice 2

                    TempVertex = new Vertex();
                    TempVertex2 = new Vertex();


      TempVertex2.setX(RotatedSceneModel.elementAt(i2).getVertex2().getX())
;

      TempVertex2.setY(RotatedSceneModel.elementAt(i2).getVertex2().getY())
;

      TempVertex2.setZ(RotatedSceneModel.elementAt(i2).getVertex2().getZ())
;

                    TempVertex.setX(TempVertex2.getX()-
TranslationVector.getX());
                    TempVertex.setY(TempVertex2.getY()-
TranslationVector.getY());
                    TempVertex.setZ(TempVertex2.getZ()-
TranslationVector.getZ());

                    TempTria.setVertex2(TempVertex);

                    //----Vertice 3

                    TempVertex = new Vertex();
                    TempVertex2 = new Vertex();


      TempVertex2.setX(RotatedSceneModel.elementAt(i2).getVertex3().getX())
;

      TempVertex2.setY(RotatedSceneModel.elementAt(i2).getVertex3().getY())
;

      TempVertex2.setZ(RotatedSceneModel.elementAt(i2).getVertex3().getZ())
;

                    TempVertex.setX(TempVertex2.getX()-
TranslationVector.getX());
                    TempVertex.setY(TempVertex2.getY()-
TranslationVector.getY());
                    TempVertex.setZ(TempVertex2.getZ()-
TranslationVector.getZ());

                    TempTria.setVertex3(TempVertex);

                    //----

                    TransformedSceneModel.add(TempTria);

            }

            System.out.println("Saving the STL File ASCII");

            STLWriter writer = new STLWriter("d:/Rotacionado.stl");
            writer.writeSTL(RotatedSceneModel, STLWriter.ASCII);
```

```
            STLWriter writer2 = new STLWriter("d:/Transladado.stl");
            writer2.writeSTL(TransformedSceneModel, STLWriter.ASCII);

            System.out.println("\nScan STL Data alignment complete");

        }
}
```

**Classe Quaternion_Rotation– Classe cuja função é calcular a rotação ponto a ponto**

```
package Data;

import Objects.Quaternion;
import Objects.Vertex;

public class Quaternion_Rotation {


	private Vertex PointRotated;

	double Q0;
	double Qx;
	double Qy;
	double Qz;

	double P0;
	double Px;
	double Py;
	double Pz;

	double Q0i;
	double Qxi;
	double Qyi;
	double Qzi;

	double Q01;
	double Qx1;
	double Qy1;
	double Qz1;

	double P0f;
	double Pxf;
	double Pyf;
	double Pzf;

	public Quaternion_Rotation(Quaternion RotationQuaternion){

		this.Q0 = RotationQuaternion.getQ0();
		this.Qx = RotationQuaternion.getQx();
		this.Qy = RotationQuaternion.getQy();
		this.Qz = RotationQuaternion.getQz();

	}

	public Vertex GetRotation(Vertex Point){


		//System.out.println("\nQ: "+Q0+" ; "+Qx+" ; "+Qy+" ; "+Qz);

		double Q0i = Q0;
		double Qxi = -Qx;
		double Qyi = -Qy;
		double Qzi = -Qz;

		//System.out.println("\nQ´: "+Q0i+" ; "+Qxi+" ; "+Qyi+" ;
"+Qzi);
```

```
            //Getting the Points

            //for (int i = 0; i < Model.size(); ++i){

                  //System.out.println("Model Size:"+Model.size()+" time
"+i);

                  P0 = 0;
                  Px = Point.getX();
                  Py = Point.getY();
                  Pz = Point.getZ();

                  //System.out.println("P: "+P0+" ; "+Px+" ; "+Py+" ;
"+Pz);

                  //Step 1

                  double Q01 = Q0*P0 - Qx*Px - Qy*Py - Qz*Pz;
                  double Qx1 = Q0*Px + Qx*P0 + Qy*Pz - Qz*Py;
                  double Qy1 = Q0*Py - Qx*Pz + Qy*P0 + Qz*Px;
                  double Qz1 = Q0*Pz + Qx*Py - Qy*Px + Qz*P0;

                  //System.out.println("Step1: "+Q01+" ; "+Qx1+" ; "+Qy1+"
; "+Qz1);

                  //Step 2

                  //double P0f = Q01*Q0i - Qx1*Qxi - Qy1*Qyi - Qz1*Qzi;
                  double Pxf = Q01*Qxi + Qx1*Q0i + Qy1*Qzi - Qz1*Qyi;
                  double Pyf = Q01*Qyi - Qx1*Qzi + Qy1*Q0i + Qz1*Qxi;
                  double Pzf = Q01*Qzi + Qx1*Qyi - Qy1*Qxi + Qz1*Q0i;

                  //System.out.println("Step2: "+P0f+" ; "+Pxf+" ; "+Pyf+"
; "+Pzf);

                  PointRotated = new Vertex();
                  PointRotated.setX(((float) Pxf));
                  PointRotated.setY(((float) Pyf));
                  PointRotated.setZ(((float) Pzf));

            return PointRotated;
      }
}
```

**Classe RecognitionAlgorithm – Classe cuja função é implementar o algoritmo de reconhecimento de rebarba**

```
package Data;

import Objects.Triangle;
import Objects.Vertex;

public class RecognitionAlgorithm {

	public Triangle Triangle = new Triangle();
	public Vertex BurrPoint = new Vertex();
	public double Tolerance;

	public RecognitionAlgorithm(double Tolerance) {

		this.Tolerance=Tolerance;

	}

	public int RecognitionCalculations(Triangle Triangle, Vertex
BurrPoint){

		this.Triangle=Triangle;
		this.BurrPoint=BurrPoint;

		//System.out.print("\nCAD Triangle Normal:  X:
"+Triangle.getNormal().getX()+" Y: "+Triangle.getNormal().getY()+" Z:
"+Triangle.getNormal().getZ());
		//System.out.print("\nCAD Triangle V1:  X:
"+Triangle.getVertex1().getX()+" Y: "+Triangle.getVertex1().getY()+" Z:
"+Triangle.getVertex1().getZ());
		//System.out.print("\nCAD Triangle V2:  X:
"+Triangle.getVertex2().getX()+" Y: "+Triangle.getVertex2().getY()+" Z:
"+Triangle.getVertex2().getZ());
		//System.out.print("\nCAD Triangle V3:  X:
"+Triangle.getVertex3().getX()+" Y: "+Triangle.getVertex3().getY()+" Z:
"+Triangle.getVertex3().getZ());

		//System.out.print("\nRebarba:  X: "+BurrPoint.getX()+" Y:
"+BurrPoint.getY()+" Z: "+BurrPoint.getZ());


		double x1, y1, z1, delta;
		double x2, y2, z2; // Projected Point

		x1 = BurrPoint.getX()-Triangle.getVertex1().getX();
		y1 = BurrPoint.getY()-Triangle.getVertex1().getY();
		z1 = BurrPoint.getZ()-Triangle.getVertex1().getZ();

		delta = x1*Triangle.getNormal().getX()+
				y1*Triangle.getNormal().getY()+
				z1*Triangle.getNormal().getZ();

		x2 = BurrPoint.getX()-(delta*Triangle.getNormal().getX());
		y2 = BurrPoint.getY()-(delta*Triangle.getNormal().getY());
		z2 = BurrPoint.getZ()-(delta*Triangle.getNormal().getZ());

		//System.out.println("\nDelta"+delta);
```

```
            //System.out.println("\nPonto Projetado X: "+x2+" Y: "+y2+" Z:
"+z2);

            int test1, test2, test3, test4;

            if(x2==Triangle.getVertex1().getX()&
                        y2==Triangle.getVertex1().getY()&
                        z2==Triangle.getVertex1().getZ()){
                  test1=1;
            }
            else{
                  test1=0;
            }
            if(x2==Triangle.getVertex2().getX()&
                        y2==Triangle.getVertex2().getY()&
                        z2==Triangle.getVertex2().getZ()){
                  test2=1;
            }
            else{
                  test2=0;
            }
            if(x2==Triangle.getVertex3().getX()&
                        y2==Triangle.getVertex3().getY()&
                        z2==Triangle.getVertex3().getZ()){
                  test3=1;
            }
            else{
                  test3=0;
            }

            test4 = test1+test2+test3;

            int c4;

            if(test4>=1){

                  //System.out.println("\nProjected Point Coincident to one
of the Vertex");
                  c4 = 1;

            }
            else{

                  //Burr point according Vertex1
                  //System.out.println("According Vertex1");
                  //Edge P2P1
                  double P2P1x, P2P1y, P2P1z, moduleP2P1;

                  P2P1x = Triangle.getVertex2().getX()-
Triangle.getVertex1().getX();
                  P2P1y = Triangle.getVertex2().getY()-
Triangle.getVertex1().getY();
                  P2P1z = Triangle.getVertex2().getZ()-
Triangle.getVertex1().getZ();

                  //System.out.println("Aresta P2P1:  X: "+P2P1x+" Y:
"+P2P1y+" Z: "+P2P1z);


                  moduleP2P1 = P2P1x*P2P1x+P2P1y*P2P1y+P2P1z*P2P1z;
                  moduleP2P1 = Math.sqrt(moduleP2P1);
```

```
                    //System.out.println("Módulo"+moduleP2P1);

                    //Edge P3P1
                    double P3P1x, P3P1y, P3P1z, moduleP3P1;

                    P3P1x = Triangle.getVertex3().getX()-
Triangle.getVertex1().getX();
                    P3P1y = Triangle.getVertex3().getY()-
Triangle.getVertex1().getY();
                    P3P1z = Triangle.getVertex3().getZ()-
Triangle.getVertex1().getZ();

                    //System.out.println("Aresta P3P1:  X: "+P3P1x+" Y:
"+P3P1y+" Z: "+P3P1z);

                    moduleP3P1 = P3P1x*P3P1x+P3P1y*P3P1y+P3P1z*P3P1z;
                    moduleP3P1 = Math.sqrt(moduleP3P1);

                    //System.out.println("Módulo"+moduleP3P1);

                    //Cosseno P3P1 P2P1
                    double cosseno_P3P1_P2P1, angle_P3P1_P2P1;

                    cosseno_P3P1_P2P1 =
(P2P1x*P3P1x+P2P1y*P3P1y+P2P1z*P3P1z)/(moduleP2P1*moduleP3P1);
                    cosseno_P3P1_P2P1 = Math.abs(cosseno_P3P1_P2P1);

                    angle_P3P1_P2P1=Math.acos(cosseno_P3P1_P2P1);
                    //System.out.println("Angulo P3P1 e P2P1:
"+angle_P3P1_P2P1);



                    //Edge PburrP1
                    double PburrP1x, PburrP1y, PburrP1z, modulePburrP1;

                    PburrP1x = x2-Triangle.getVertex1().getX();
                    PburrP1y = y2-Triangle.getVertex1().getY();
                    PburrP1z = z2-Triangle.getVertex1().getZ();

                    modulePburrP1 =
PburrP1x*PburrP1x+PburrP1y*PburrP1y+PburrP1z*PburrP1z;
                    modulePburrP1 = Math.sqrt(modulePburrP1);

                    //-----------------

                    //Cosseno PburrP1 P2P1
                    double cosseno_PburrP1_P2P1, angle_PburrP1_P2P1;

                    cosseno_PburrP1_P2P1 =
(P2P1x*PburrP1x+P2P1y*PburrP1y+P2P1z*PburrP1z)/(moduleP2P1*modulePburrP1);
                    cosseno_PburrP1_P2P1 = Math.abs(cosseno_PburrP1_P2P1);

                    angle_PburrP1_P2P1=Math.acos(cosseno_PburrP1_P2P1);
                    //System.out.println("Angulo Rebarba e P2P1:
"+angle_PburrP1_P2P1);

                    //Cosseno PburrP1 P3P1
                    double cosseno_PburrP1_P3P1, angle_PburrP1_P3P1;
```

```
                cosseno_PburrP1_P3P1 =
(P3P1x*PburrP1x+P3P1y*PburrP1y+P3P1z*PburrP1z)/(moduleP3P1*modulePburrP1);
                cosseno_PburrP1_P3P1 = Math.abs(cosseno_PburrP1_P3P1);

                angle_PburrP1_P3P1=Math.acos(cosseno_PburrP1_P3P1);
                //System.out.println("Angulo Rebarba e P3P1:
"+angle_PburrP1_P3P1);

                //-----------------

                int a1, b1, c1;
                //Testing if acoording P1 is Burr
                if(angle_PburrP1_P2P1>angle_P3P1_P2P1){
                      a1 = 0;
                }
                else{
                      a1 = 1;
                }
                if(angle_PburrP1_P3P1>angle_P3P1_P2P1){
                      b1 = 0;
                }
                else{
                      b1 = 1;
                }
                c1 = a1*b1;

                //System.out.println("Em P1, se = 1 pertence :::: "+c1);


                //Burr point according Vertex2
                //System.out.println("According Vertex2");
                //Edge P1P2
                double P1P2x, P1P2y, P1P2z, moduleP1P2;

                P1P2x = Triangle.getVertex1().getX()-
Triangle.getVertex2().getX();
                P1P2y = Triangle.getVertex1().getY()-
Triangle.getVertex2().getY();
                P1P2z = Triangle.getVertex1().getZ()-
Triangle.getVertex2().getZ();

                //System.out.println("Aresta P1P2:  X: "+P1P2x+" Y:
"+P1P2y+" Z: "+P1P2z);


                moduleP1P2 = P1P2x*P1P2x+P1P2y*P1P2y+P1P2z*P1P2z;
                moduleP1P2 = Math.sqrt(moduleP1P2);

                //System.out.println("Módulo"+moduleP1P2);

                //Edge P3P2
                double P3P2x, P3P2y, P3P2z, moduleP3P2;

                P3P2x = Triangle.getVertex3().getX()-
Triangle.getVertex2().getX();
                P3P2y = Triangle.getVertex3().getY()-
Triangle.getVertex2().getY();
                P3P2z = Triangle.getVertex3().getZ()-
Triangle.getVertex2().getZ();
```

```
                //System.out.println("Aresta P3P2:  X: "+P3P2x+" Y:
"+P3P2y+" Z: "+P3P2z);

                moduleP3P2 = P3P2x*P3P2x+P3P2y*P3P2y+P3P2z*P3P2z;
                moduleP3P2 = Math.sqrt(moduleP3P2);

                //System.out.println("Módulo"+moduleP3P2);

                //Cosseno P1P2 P3P2
                double cosseno_P1P2_P3P2, angle_P1P2_P3P2;

                cosseno_P1P2_P3P2 =
(P1P2x*P3P2x+P1P2y*P3P2y+P1P2z*P3P2z)/(moduleP1P2*moduleP3P2);
                cosseno_P1P2_P3P2 = Math.abs(cosseno_P1P2_P3P2);

                angle_P1P2_P3P2=Math.acos(cosseno_P1P2_P3P2);
                //System.out.println("Angulo P3P2 e P1P2:
"+angle_P1P2_P3P2);



                //Edge PburrP2
                double PburrP2x, PburrP2y, PburrP2z, modulePburrP2;

                PburrP2x = x2-Triangle.getVertex2().getX();
                PburrP2y = y2-Triangle.getVertex2().getY();
                PburrP2z = z2-Triangle.getVertex2().getZ();

                modulePburrP2 =
PburrP2x*PburrP2x+PburrP2y*PburrP2y+PburrP2z*PburrP2z;
                modulePburrP2 = Math.sqrt(modulePburrP2);

                //Cosseno PburrP2 P1P2
                double cosseno_PburrP2_P1P2, angle_PburrP2_P1P2;

                cosseno_PburrP2_P1P2 =
(P1P2x*PburrP2x+P1P2y*PburrP2y+P1P2z*PburrP2z)/(moduleP1P2*modulePburrP2);
                cosseno_PburrP2_P1P2 = Math.abs(cosseno_PburrP2_P1P2);

                angle_PburrP2_P1P2=Math.acos(cosseno_PburrP2_P1P2);
                //System.out.println("Angulo Rebarba e P2P1:
"+angle_PburrP2_P1P2);

                //Cosseno PburrP2 P3P2
                double cosseno_PburrP2_P3P2, angle_PburrP2_P3P2;

                cosseno_PburrP2_P3P2 =
(P3P2x*PburrP2x+P3P2y*PburrP2y+P3P2z*PburrP2z)/(moduleP3P2*modulePburrP2);
                cosseno_PburrP2_P3P2 = Math.abs(cosseno_PburrP2_P3P2);

                angle_PburrP2_P3P2=Math.acos(cosseno_PburrP2_P3P2);
                //System.out.println("Angulo Rebarba e P3P2:
"+angle_PburrP2_P3P2);

                int a2, b2, c2;
                //Testing if acoording P1 is Burr
                if(angle_PburrP2_P1P2>angle_P1P2_P3P2){
                      a2 = 0;
                }
                else{
                      a2 = 1;
```

```
                }
                if(angle_PburrP2_P3P2>angle_P1P2_P3P2){
                      b2 = 0;
                }
                else{
                      b2 = 1;
                }
                c2 = a2*b2;

                //System.out.println("Em P2, se = 1 pertence :::: "+c2);


                //Burr point according Vertex3
                //System.out.println("According Vertex3");
                //Edge P2P3
                double P2P3x, P2P3y, P2P3z, moduleP2P3;

                P2P3x = Triangle.getVertex2().getX()-
Triangle.getVertex3().getX();
                P2P3y = Triangle.getVertex2().getY()-
Triangle.getVertex3().getY();
                P2P3z = Triangle.getVertex2().getZ()-
Triangle.getVertex3().getZ();

                //System.out.println("Aresta P2P3:  X: "+P2P3x+" Y:
"+P2P3y+" Z: "+P2P3z);


                moduleP2P3 = P2P3x*P2P3x+P2P3y*P2P3y+P2P3z*P2P3z;
                moduleP2P3 = Math.sqrt(moduleP2P3);

                //System.out.println("Módulo"+moduleP2P3);

                //Edge P1P3
                double P1P3x, P1P3y, P1P3z, moduleP1P3;

                P1P3x = Triangle.getVertex1().getX()-
Triangle.getVertex3().getX();
                P1P3y = Triangle.getVertex1().getY()-
Triangle.getVertex3().getY();
                P1P3z = Triangle.getVertex1().getZ()-
Triangle.getVertex3().getZ();

                //System.out.println("Aresta P1P3:  X: "+P1P3x+" Y:
"+P1P3y+" Z: "+P1P3z);

                moduleP1P3 = P1P3x*P1P3x+P1P3y*P1P3y+P1P3z*P1P3z;
                moduleP1P3 = Math.sqrt(moduleP1P3);

                //System.out.println("Módulo"+moduleP3P1);

                //Cosseno P1P3 P2P3
                double cosseno_P1P3_P2P3, angle_P1P3_P2P3;

                cosseno_P1P3_P2P3 =
(P2P3x*P1P3x+P2P3y*P1P3y+P2P3z*P1P3z)/(moduleP2P3*moduleP1P3);
                cosseno_P1P3_P2P3 = Math.abs(cosseno_P1P3_P2P3);

                angle_P1P3_P2P3=Math.acos(cosseno_P1P3_P2P3);
                //System.out.println("Angulo P1P3 e P2P3:
"+angle_P1P3_P2P3);
```

```
                //Edge PburrP3
                double PburrP3x, PburrP3y, PburrP3z, modulePburrP3;

                PburrP3x = x2-Triangle.getVertex3().getX();
                PburrP3y = y2-Triangle.getVertex3().getY();
                PburrP3z = z2-Triangle.getVertex3().getZ();

                modulePburrP3 =
PburrP3x*PburrP3x+PburrP3y*PburrP3y+PburrP3z*PburrP3z;
                modulePburrP3 = Math.sqrt(modulePburrP3);

                //Cosseno PburrP3 P2P3
                double cosseno_PburrP3_P2P3, angle_PburrP3_P2P3;

                cosseno_PburrP3_P2P3 =
(P2P3x*PburrP3x+P2P3y*PburrP3y+P2P3z*PburrP3z)/(moduleP2P3*modulePburrP3);
                cosseno_PburrP3_P2P3 = Math.abs(cosseno_PburrP3_P2P3);

                angle_PburrP3_P2P3=Math.acos(cosseno_PburrP3_P2P3);
                //System.out.println("Angulo Rebarba e P2P3:
"+angle_PburrP3_P2P3);

                //Cosseno PburrP1 P3P1
                double cosseno_PburrP3_P1P3, angle_PburrP3_P1P3;

                cosseno_PburrP3_P1P3 =
(P1P3x*PburrP3x+P1P3y*PburrP3y+P1P3z*PburrP3z)/(moduleP1P3*modulePburrP3);
                cosseno_PburrP3_P1P3 = Math.abs(cosseno_PburrP3_P1P3);

                angle_PburrP3_P1P3=Math.acos(cosseno_PburrP3_P1P3);
                //System.out.println("Angulo Rebarba e P1P3:
"+angle_PburrP3_P1P3);

                int a3, b3, c3;
                //Testing if acoording P3 is Burr
                if(angle_PburrP3_P2P3>angle_P1P3_P2P3){
                      a3 = 0;
                }
                else{
                      a3 = 1;
                }
                if(angle_PburrP3_P1P3>angle_P1P3_P2P3){
                      b3 = 0;
                }
                else{
                      b3 = 1;
                }
                c3 = a3*b3;

                //System.out.println("Em P3, se = 1 pertence :::: "+c3);

                c4 = c1*c2*c3;

          }

          int c5, c6;

          double c7;
```

```
            c7 = Math.abs(delta);

            if(c7>Tolerance){
                  c5 = 0;
            }
            else{
                  c5 = 1;
            }

            c6 = c5 * c4;

            //System.out.println("Pertence se = 1   ::::   "+c4);
            //System.out.println("Nao é rebarba = 1   ::::   "+c6);
            //System.out.println("Está na Tolerancia = 1   ::::   "+c5);

            return c6;
      }
}
```

**Classe Cardano_Method – Classe cuja função é calcular as raízes de um polinômio de grau 3**

```
package MathCalculations;

//import java.util.Vector;

public class Cardano_Method {
	/**
	 * @author Brogio de Oliveira, Bruno
	 */

	/**
	 Cardano_Method Class

	 - Main Objective: Find the three zeros of a three degree polynomial
equation
	 */
	private double a1, b1, c1, d1, a2, b2, c2;
	private double p, q;
	private double a, ai, b, bi, c, ci, d, e, f, fi, g;// di;
	private double u1, u1i, u2, u2i;
	private double t1, t2, t2i, t3, t3i;
	private double x1, x2, x2i, x3, x3i;

	public Cardano_Method(double a, double b, double c, double d){
		this.a1 = a;
		this.b1 = b;
		this.c1 = c;
		this.d1 = d;
		ZerosCalculation();
	}

	public void ZerosCalculation(){

		//A - Erase Later
		//System.out.println("Cúbic Equation:");
		//System.out.println(""+a1+" . x3 + "+b1+" . x2 + "+c1+" . x +
"+d1+" = 0");
		//A - Erase Later

		//Monic Polynomial - 1x3 + ax2 + bx + c = 0
		a2=b1/a1;
		b2=c1/a1;
		c2=d1/a1;

		//B - Erase Later
		//System.out.println("Monic Cúbic Equation:");
		//System.out.println(""+1+" . x3 + "+a2+" . x2 + "+b2+" . x +
"+c2+" = 0");
		//B - Erase Later

		//Generating P and Q Values

		p = b2-((a2*a2)/3);
		q = c2 + ((2*a2*a2*a2-9*a2*b2)/27);

		//C - Erase Later
		//System.out.println("\np = "+p+"\nq = "+q);
		//C - Erase Later
```

```
		//Calculating the Zeros

		e = -(q/2);

		g = ((q*q)/4)+((p*p*p)/27);

		//D - Erase Later
		//System.out.println("\ne = "+e+"\ng = "+g);
		//D - Erase Later

		//f = Square Root of g - If g is positive or negative.

		if(g>0){
			f=Math.sqrt(g);
			fi=0;
		}
		if(g==0){
			f=0;
			fi=0;
		}
		if(g<0){
			f=0;
			fi=Math.sqrt(Math.sqrt(g*g));
		}

		//E - Erase Later
		//System.out.println("\nf = "+f+" + "+fi+" i ");
		//E - Erase Later

		//c and d = e +/- f

		c = e + f;
		ci = 0 + fi;
		d = e - f;
		//di = 0 - fi;

		//F - Erase Later
		//System.out.println("\nc = "+c+" + "+ci+" i ");
		//System.out.println("\nd = "+d+" + "+di+" i ");
		//F - Erase Later

		//G - Erase Later
		//System.out.println("\nU1 and U2 Caculations\n");
		//G - Erase Later

		//U1 = Cubic Root of c - Independently if g is positive or
negative.

		//If c is just Real Value
		if(c!=0&ci==0){
			u1=Math.cbrt(c);
			u1i=0;

			u2=Math.cbrt(d);
			u2i=0;
		}

		//If c is a complex number
		if(c!=0&ci!=0){
```

```
                //module of c
                double modulec;
                modulec = Math.sqrt(c*c+ci*ci);

                //angle
                double angle = 0;

                if(c>0&ci>0){
                      angle=Math.atan(ci/c);
                }
                if(c<0&ci>0){
                      angle=(3.1415926535-Math.atan(Math.abs(ci/c)));
                }
                if(c<0&ci<0){
                      angle=(3.1415926535+Math.atan(Math.abs(ci/c)));
//tudo indica q abs é modulo
                }
                if(c>0&ci<0){
                      angle=(2*3.1415926535-Math.atan(Math.abs(ci/c)));
                }


                u1=Math.cbrt(modulec)*(Math.cos(angle/3));
                u1i=Math.cbrt(modulec)*(Math.sin(angle/3));
                u2=Math.cbrt(modulec)*(Math.cos(angle/3));
                u2i=Math.cbrt(modulec)*(Math.sin(angle/3)*-1);
          }
          //If c just a imaginary number
          if(ci==0&c!=0){
                // Not implemented this comparation
                f=0;
                fi=0;
          }

          //H - Erase Later
          //System.out.println("\nModule of c = "+modulec);
          //System.out.println("\nAngle = "+angle);
          //H - Erase Later


          //I - Erase Later
          //System.out.println("\nU1 = "+u1+" + "+u1i+" i ");
          //System.out.println("\nU2 = "+u2+" + "+u2i+" i ");
          //I - Erase Later

          a = -0.5;
          ai = 0.86602540378443;

          b = -0.5;
          bi = -0.86602540378443;

          //J - Erase Later
          //System.out.println("\nA = "+a+" + "+ai+" i ");
          //System.out.println("\nB = "+b+" + "+bi+" i ");
          //J - Erase Later

          t1 = u1 + u2;

          //A.u1+B.u2

          t2 = (a*u1-ai*u1i)+(b*u2-bi*u2i);
```

```
            t2i = (a*u1i+ai*u1)+(b*u2i+bi*u2);

            t3 = (b*u1-bi*u1i)+(a*u2-ai*u2i);
            t3i = (b*u1i+bi*u1)+(a*u2i+ai*u2);

            //K - Erase Later
            //System.out.println("\nT1 = "+t1);
            //System.out.println("\nT2 = "+t2+" + "+t2i+" i ");
            //System.out.println("\nT3 = "+t3+" + "+t3i+" i ");
            //K - Erase Later


            x1 = t1 - (a2/3);

            x2 = t2 - (a2/3);
            x2i = t2i;

            x3 = t3 - (a2/3);
            x3i = t3i;


      }

      public double GetZero1(){
            return x1;
      }
      public double GetZero2(){
            return x2;
      }
      public double GetZero2i(){
            return x2i;
      }
      public double GetZero3(){
            return x3;
      }
      public double GetZero3i(){
            return x3i;
      }


}
```

**Classe Ferrari_Method– Classe cuja função é calcular as raízes de um polinômio de grau 4**

```
package MathCalculations;

public class Ferrari_Method {
	/**
	 Ferrari_Method Class

	 - Main Objective: Find the four zeros of a four degree polynomial
equation
	 */
	private double a1, b1, c1, d1, e1, a2, b2, c2, d2;
	private double alpha, beta, gama, y;
	private double c, a, b31, b31i, b32, b32i, ai, f, fi, e, d31, d31i,
d32;// d32i;
	private double x1, x1i, x2, x2i, x3, x3i, x4, x4i;

	public Ferrari_Method(double a, double b, double c, double d, double
e){
		this.a1 = a;
		this.b1 = b;
		this.c1 = c;
		this.d1 = d;
		this.e1 = e;
		ZerosCalculation();
	}

	public void ZerosCalculation(){

		//A - Erase Later
		//System.out.println("\nQuartic Equation:");
		//System.out.println(""+a1+" . x4 + "+b1+" . x3 + "+c1+" . x2 +
"+d1+" . x + "+e1+" = 0");
		//A - Erase Later

		alpha = -((3*b1*b1)/(8*a1*a1))+(c1/a1);
		beta = ((b1*b1*b1)/(8*a1*a1*a1))-((b1*c1)/(2*a1*a1))+((d1/a1));
		gama = -
((3*b1*b1*b1*b1)/(256*a1*a1*a1*a1))+((c1*b1*c1)/(16*a1*a1*a1))-
		((b1*d1)/(4*a1*a1))+((e1/a1));

		//C - Erase Later
		//System.out.println("\nAlpha = "+alpha+"\nBeta =
"+beta+"\nGama = "+gama);
		//C - Erase Later

		//D - Erase Later
		//System.out.println("\nFinding the three zeros of a three
degree polynomial equation");
		//D - Erase Later

		a2=1;
		b2=2.5*alpha;
		c2=(2*alpha*alpha-gama);
		d2=(alpha*alpha*alpha/2-alpha*gama/2-beta*beta/8);

		Cardano_Method Polynomial_Cardano = new Cardano_Method((double)
a2, (double) b2,
				(double) c2, (double) d2);
```

```
            //System.out.println("\nX1 = "+Polynomial_Cardano.GetZero1());
            //System.out.println("X2 = "+Polynomial_Cardano.GetZero2()+" +
"+Polynomial_Cardano.GetZero2i());
            //System.out.println("X3 = "+Polynomial_Cardano.GetZero3()+" +
"+Polynomial_Cardano.GetZero3i());

            y = Polynomial_Cardano.GetZero1();
            //System.out.println("\nY = "+y);
            //System.out.println("\nAlpha = "+alpha);
            c = alpha+2*y;

            //E - Erase Later
            //System.out.println("\nC = "+c);
            //E - Erase Later

            if(c>0){
                  a=Math.sqrt(c);
                  ai=0;
            }
            if(c==0){
                  a=0;
                  ai=0;
            }
            if(c<0){
                  a=0;
                  ai=Math.sqrt(Math.sqrt(c*c));
            }

            //F - Erase Later
            //System.out.println("\nA = "+a+" + "+ai+" i ");
            //F - Erase Later

            f = (2*beta*a)/(a*a+ai*ai);
            fi = (2*beta*ai)/(a*a+ai*ai);

            //G - Erase Later
            //System.out.println("\nF = "+f+" + "+fi+" i ");
            //G - Erase Later

            e = 3*alpha+2*y;

            //H - Erase Later
            //System.out.println("\nE = "+e);
            //H - Erase Later

            d31 = -(e + f);
            d31i = -(fi);

            d32 = -(e - f);
            //d32i = -(-fi);

            //I - Erase Later
            //System.out.println("\nD1 = "+d31+" + "+d31i+" i ");
            //System.out.println("\nD2 = "+d32+" + "+d32i+" i ");
            //I - Erase Later


            if(d31>0&d31i==0){
                  b31=Math.sqrt(d31);
                  b31i=0;
```

```
                b32=Math.sqrt(d32);
                b32i=0;
            }
            if(d31<0&d31i==0){
                b31=0;
                b31i=Math.sqrt(Math.abs(d31));

                b32=0;
                b32i=Math.sqrt(Math.abs(d32));
            }

            //If c is a complex number
            if(d31!=0&d31i!=0){

                //module of c
                double moduled;
                moduled = Math.sqrt(d31*d31+d31i*d31i);

                //angle
                double angle = 0;

                if(d31>0&d31i>0){
                    angle=Math.atan(d31/d31i);
                }
                if(d31<0&d31i>0){
                    angle=(3.1415926535-Math.atan(Math.abs(d31/d31i)));
                }
                if(d31<0&d31i<0){
                    angle=(3.1415926535+Math.atan(Math.abs(d31/d31i)));
                }
                if(d31>0&d31i<0){
                    angle=(2*3.1415926535-
Math.atan(Math.abs(d31/d31i)));
                }


                b31=Math.cbrt(moduled)*(Math.cos(angle/2));
                b31i=Math.cbrt(moduled)*(Math.sin(angle/2));
                b32=Math.cbrt(moduled)*(Math.cos(angle/2));
                b32i=Math.cbrt(moduled)*(Math.sin(angle/2)*-1);

            }

            //If c just a imaginary number
            if(d31==0&d31i!=0){
                // Not implemented this comparison
                b31=0;
                b31i=0;
            }

            //J - Erase Later
            //System.out.println("\nModule of c = "+moduled);
            //System.out.println("\nAngle = "+angle);
            //J - Erase Later

            //K - Erase Later
            //System.out.println("\nB1 = "+b31+" + "+b31i+" i ");
            //System.out.println("\nB2 = "+b32+" + "+b32i+" i ");
            //K - Erase Later

            x1 = ( + a + b31)/2;
```

```
            x1i = ( + ai + b31i)/2;

            x2 = ( + a - b31)/2;
            x2i = ( + ai - b31i)/2;

            x3 = ( - a + b32)/2;
            x3i = ( - ai + b32i)/2;

            x4 = ( - a - b32)/2;
            x4i = ( - ai - b32i)/2;

            //L - Erase Later
            //System.out.println("\nX1 = "+x1+" + "+x1i+" i ");
            //System.out.println("X2 = "+x2+" + "+x2i+" i ");
            //System.out.println("X3 = "+x3+" + "+x3i+" i ");
            //System.out.println("X4 = "+x4+" + "+x4i+" i ");
            //L - Erase Later
      }

      public double GetZero1(){
            return x1;
      }
      public double GetZero1i(){
            return x1i;
      }
      public double GetZero2(){
            return x2;
      }
      public double GetZero2i(){
            return x2i;
      }
      public double GetZero3(){
            return x3;
      }
      public double GetZero3i(){
            return x3i;
      }
      public double GetZero4(){
            return x4;
      }
      public double GetZero4i(){
            return x4i;
      }

}
```

**Classe gauss_Method – Classe cuja função é calcular os zeros de um sistema linear homogêneo de 4 equações**

```
package MathCalculations;

import Objects.Quaternion;

public class Gauss_Method {

	double Matrix[][];
	double Matrix2[][];
	double Matrix3[][];
	double Matrix4[][];
	double Factor1;
	double Factor2;
	double Factor3;
	double Q0;
	double Qx;
	double Qy;
	double Qz;
	double Module;
	public Quaternion RotationQuaternion = new Quaternion();

	public Gauss_Method(double Matrix[][]){
		this.Matrix = Matrix;
		ZerosCalculation();
	}

	public void ZerosCalculation(){

		Matrix2 = Matrix;

		Factor1 = (Matrix[1][0])/(Matrix[0][0]);
		Factor2 = Matrix[2][0]/Matrix[0][0];
		Factor3 = Matrix[3][0]/Matrix[0][0];

		//System.out.println("Factor1:"+Factor1);
		//System.out.println("Factor2:"+Factor2);
		//System.out.println("Factor3:"+Factor3);

		Matrix2[1][0] = Matrix[1][0]-Matrix[0][0]*Factor1;
		Matrix2[1][1] = Matrix[1][1]-Matrix[0][1]*Factor1;
		Matrix2[1][2] = Matrix[1][2]-Matrix[0][2]*Factor1;
		Matrix2[1][3] = Matrix[1][3]-Matrix[0][3]*Factor1;

		Matrix2[2][0] = Matrix[2][0]-Matrix[0][0]*Factor2;
		Matrix2[2][1] = Matrix[2][1]-Matrix[0][1]*Factor2;
		Matrix2[2][2] = Matrix[2][2]-Matrix[0][2]*Factor2;
		Matrix2[2][3] = Matrix[2][3]-Matrix[0][3]*Factor2;

		Matrix2[3][0] = Matrix[3][0]-Matrix[0][0]*Factor3;
		Matrix2[3][1] = Matrix[3][1]-Matrix[0][1]*Factor3;
		Matrix2[3][2] = Matrix[3][2]-Matrix[0][2]*Factor3;
		Matrix2[3][3] = Matrix[3][3]-Matrix[0][3]*Factor3;

		/**
		System.out.println("\n   Matrix2:");
		System.out.println("   "+Matrix2[0][0]+"   "+Matrix2[0][1]+"
"+Matrix2[0][2]+"   "+Matrix2[0][3]+"");
```

```
            System.out.println("   "+Matrix2[1][0]+"   "+Matrix2[1][1]+"
"+Matrix2[1][2]+"   "+Matrix2[1][3]+"");
            System.out.println("   "+Matrix2[2][0]+"   "+Matrix2[2][1]+"
"+Matrix2[2][2]+"   "+Matrix2[2][3]+"");
            System.out.println("   "+Matrix2[3][0]+"   "+Matrix2[3][1]+"
"+Matrix2[3][2]+"   "+Matrix2[3][3]+"");
            */

            Matrix3 = Matrix2;

            Factor1 = (Matrix2[2][1])/(Matrix2[1][1]);
            Factor2 = (Matrix2[3][1])/(Matrix2[1][1]);

            //System.out.println("\nFactor1:"+Factor1);
            //System.out.println("Factor2:"+Factor2);

            Matrix3[2][0] = Matrix2[2][0]-Matrix2[1][0]*Factor1;
            Matrix3[2][1] = Matrix2[2][1]-Matrix2[1][1]*Factor1;
            Matrix3[2][2] = Matrix2[2][2]-Matrix2[1][2]*Factor1;
            Matrix3[2][3] = Matrix2[2][3]-Matrix2[1][3]*Factor1;

            Matrix3[3][0] = Matrix2[3][0]-Matrix2[1][0]*Factor2;
            Matrix3[3][1] = Matrix2[3][1]-Matrix2[1][1]*Factor2;
            Matrix3[3][2] = Matrix2[3][2]-Matrix2[1][2]*Factor2;
            Matrix3[3][3] = Matrix2[3][3]-Matrix2[1][3]*Factor2;

            /**
            System.out.println("\n   Matrix3:");
            System.out.println("   "+Matrix3[0][0]+"   "+Matrix3[0][1]+"
"+Matrix3[0][2]+"   "+Matrix3[0][3]+"");
            System.out.println("   "+Matrix3[1][0]+"   "+Matrix3[1][1]+"
"+Matrix3[1][2]+"   "+Matrix3[1][3]+"");
            System.out.println("   "+Matrix3[2][0]+"   "+Matrix3[2][1]+"
"+Matrix3[2][2]+"   "+Matrix3[2][3]+"");
            System.out.println("   "+Matrix3[3][0]+"   "+Matrix3[3][1]+"
"+Matrix3[3][2]+"   "+Matrix3[3][3]+"");
            */

            Matrix4 = Matrix3;

            Factor1 = (Matrix3[3][2])/(Matrix3[2][2]);

            //System.out.println("\nFactor1:"+Factor1);

            Matrix4[3][0] = Matrix3[3][0]-Matrix3[2][0]*Factor1;
            Matrix4[3][1] = Matrix3[3][1]-Matrix3[2][1]*Factor1;
            Matrix4[3][2] = Matrix3[3][2]-Matrix3[2][2]*Factor1;
            Matrix4[3][3] = Matrix3[3][3]-Matrix3[2][3]*Factor1;

            /**
            System.out.println("\n   Matrix4:");
            System.out.println("   "+Matrix4[0][0]+"   "+Matrix4[0][1]+"
"+Matrix4[0][2]+"   "+Matrix4[0][3]+"");
            System.out.println("   "+Matrix4[1][0]+"   "+Matrix4[1][1]+"
"+Matrix4[1][2]+"   "+Matrix4[1][3]+"");
            System.out.println("   "+Matrix4[2][0]+"   "+Matrix4[2][1]+"
"+Matrix4[2][2]+"   "+Matrix4[2][3]+"");
            System.out.println("   "+Matrix4[3][0]+"   "+Matrix4[3][1]+"
"+Matrix4[3][2]+"   "+Matrix4[3][3]+"");
            */
```

```
            Qz = 1;
            Qy = -(Qz*Matrix3[2][3])/Matrix3[2][2];
            Qx = -(Qz*Matrix3[1][3]+Qy*Matrix3[1][2])/Matrix3[1][1];
            Q0 = -
(Qz*Matrix3[0][3]+Qy*Matrix3[0][2]+Qx*Matrix3[0][1])/Matrix3[0][0];

            //Unit Vector

            Module = Math.sqrt(Q0*Q0+Qx*Qx+Qy*Qy+Qz*Qz);

            Q0 = Q0/Module;
            Qx = Qx/Module;
            Qy = Qy/Module;
            Qz = Qz/Module;

            RotationQuaternion.setQ0(Q0);
            RotationQuaternion.setQx(Qx);
            RotationQuaternion.setQy(Qy);
            RotationQuaternion.setQz(Qz);

      }

      public Quaternion GetQuaternion(){
            return RotationQuaternion;
      }

}
```

**Classe Horn_Algorithm – Classe cuja função é implementar o algoritmo de Horn**

```
package MathCalculations;

import Objects.Vertex;
import java.util.Vector;

public class Horn_Algorithm {
	/**
	 Horn_Algorithm Class

	 - Main Objective: Generate the Matrix M and the Matrix N
	   - The Matrix M is used to generate the Matrix N and the Four
Degree Polynomial
	   - The Four Degree Polynomial is used to calculate the four
eigenvalues of the Matrix N
	   - The Matrix N is used to calculate the eigenvector of the most
positive eigenvalue
	   of the Matrix N itself.
	 */

	private Vector<Vertex> Model, Scene;

	public Horn_Algorithm(Vector<Vertex> Model, Vector<Vertex> Scene)
throws Exception {
		this.Scene = Scene;
		this.Model = Model;
		if (Scene.size() != Model.size())
			throw new Exception("Illegal Size - Model and Scene
Vectors does not have the same size");
	}

	public double[][] getM() {
		int size = Model.size();

		double Sxx = 0;
		for (int i = 0; i < size; ++i)
			Sxx += Model.get(i).getX() * Scene.get(i).getX();

		double Sxy = 0;
		for (int i = 0; i < size; ++i)
			Sxy += Model.get(i).getX() * Scene.get(i).getY();

		double Sxz = 0;
		for (int i = 0; i < size; ++i)
			Sxz += Model.get(i).getX() * Scene.get(i).getZ();

		double Syx = 0;
		for (int i = 0; i < size; ++i)
			Syx += Model.get(i).getY() * Scene.get(i).getX();

		double Syy = 0;
		for (int i = 0; i < size; ++i)
			Syy += Model.get(i).getY() * Scene.get(i).getY();

		double Syz = 0;
		for (int i = 0; i < size; ++i)
			Syz += Model.get(i).getY() * Scene.get(i).getZ();
```

```
		double Szx = 0;
		for (int i = 0; i < size; ++i)
			Szx += Model.get(i).getZ() * Scene.get(i).getX();

		double Szy = 0;
		for (int i = 0; i < size; ++i)
			Szy += Model.get(i).getZ() * Scene.get(i).getY();

		double Szz = 0;
		for (int i = 0; i < size; ++i)
			Szz += Model.get(i).getZ() * Scene.get(i).getZ();


		// Matrix M
		double MatrixM[][]=new double [][] {
				{Sxx, Sxy, Sxz},
				{Syx, Syy, Syz},
				{Szx, Szy, Szz}
		};
		return MatrixM;
	}

	public double[][] getN() {
		// Matrix N
		double MatrixM[][]=getM();

		double a = +MatrixM[0][0]+MatrixM[1][1]+MatrixM[2][2];
		double b = +MatrixM[0][0]-MatrixM[1][1]-MatrixM[2][2];
		double c = -MatrixM[0][0]+MatrixM[1][1]-MatrixM[2][2];
		double d = -MatrixM[0][0]-MatrixM[1][1]+MatrixM[2][2];
		double e = +MatrixM[1][2]-MatrixM[2][1];
		double f = +MatrixM[0][1]+MatrixM[1][0];
		double g = +MatrixM[1][2]+MatrixM[2][1];
		double h = +MatrixM[2][0]-MatrixM[0][2];
		double i = +MatrixM[2][0]+MatrixM[0][2];
		double j = +MatrixM[0][1]-MatrixM[1][0];

		double MatrixN[][]=new double [][] {
				{a, e, h, j},
				{e, b, f, i},
				{h, f, c, g},
				{j, i, g, d}
		};
		return MatrixN;
	}

}
```

**Classe Matrix_3x3_Determinant – Classe cuja função é calcular o determinante de uma matriz 3x3**

```
package MathCalculations;

public class Matrix_3x3_Determinant {

	double Matrix[][];
	double Determinant;

	public Matrix_3x3_Determinant(double Matrix[][]){
		this.Matrix = Matrix;
	}

	public double GetDeterminant(){

		Determinant = +Matrix[0][0]*Matrix[1][1]*Matrix[2][2]
			      +Matrix[0][1]*Matrix[1][2]*Matrix[2][0]
			      +Matrix[0][2]*Matrix[1][0]*Matrix[2][1]
			      -Matrix[0][2]*Matrix[1][1]*Matrix[2][0]
			      -Matrix[0][0]*Matrix[1][2]*Matrix[2][1]
			      -Matrix[0][1]*Matrix[1][0]*Matrix[2][2];

		return Determinant;
	}

}
```

**Classe Matrix_4x4_Determinant – Classe cuja função é calcular o determinante de uma matriz 4x4**

```
package MathCalculations;

public class Matrix_4x4_Determinant {

	double Matrix[][];
	double Determinant;

	public Matrix_4x4_Determinant(double Matrix[][]){
		this.Matrix = Matrix;
	}

	public double GetDeterminant(){

		double MatrixA[][]=new double [][] {
				{Matrix[1][1], Matrix[1][2], Matrix[1][3]},
				{Matrix[2][1], Matrix[2][2], Matrix[2][3]},
				{Matrix[3][1], Matrix[3][2], Matrix[3][3]},
				};
		Matrix_3x3_Determinant ADeterminant = new
Matrix_3x3_Determinant(MatrixA);

		double MatrixB[][]=new double [][] {
				{Matrix[0][1], Matrix[0][2], Matrix[0][3]},
				{Matrix[2][1], Matrix[2][2], Matrix[2][3]},
				{Matrix[3][1], Matrix[3][2], Matrix[3][3]},
				};
		Matrix_3x3_Determinant BDeterminant = new
Matrix_3x3_Determinant(MatrixB);

		double MatrixC[][]=new double [][] {
				{Matrix[0][1], Matrix[0][2], Matrix[0][3]},
				{Matrix[1][1], Matrix[1][2], Matrix[1][3]},
				{Matrix[3][1], Matrix[3][2], Matrix[3][3]},
				};
		Matrix_3x3_Determinant CDeterminant = new
Matrix_3x3_Determinant(MatrixC);

		double MatrixD[][]=new double [][] {
				{Matrix[0][1], Matrix[0][2], Matrix[0][3]},
				{Matrix[1][1], Matrix[1][2], Matrix[1][3]},
				{Matrix[2][1], Matrix[2][2], Matrix[2][3]},
				};
		Matrix_3x3_Determinant DDeterminant = new
Matrix_3x3_Determinant(MatrixD);

		//Determinant
		Determinant = +Matrix[0][0]*ADeterminant.GetDeterminant()
				-Matrix[1][0]*BDeterminant.GetDeterminant()
			  +Matrix[2][0]*CDeterminant.GetDeterminant()
			  -Matrix[3][0]*DDeterminant.GetDeterminant();

		return Determinant;
	}

}
```

```
package Objects;

public class Quaternion {

	private double Q0, Qx, Qy, Qz;

	public Quaternion() {
		Q0 = 0;
		Qx = 0;
		Qy = 0;
		Qz = 0;
	}

	public double getQ0() {
		return Q0;
	}

	public void setQ0(double q02) {
		Q0 = q02;
	}

	public double getQx() {
		return Qx;
	}

	public void setQx(double qx) {
		Qx = qx;
	}

	public double getQy() {
		return Qy;
	}

	public void setQy(double qy) {
		Qy = qy;
	}

	public double getQz() {
		return Qz;
	}

	public void setQz(double qz) {
		Qz = qz;
	}


}
```

**Classe Triangle – Classe cuja função é determinar um objeto do tipo triangulo**

```
package Objects;

public class Triangle {

	private Vertex normal;
	private Vertex vertex1, vertex2, vertex3;

	public Triangle() {
		normal = new Vertex();
		vertex1 = new Vertex();
		vertex2 = new Vertex();
		vertex3 = new Vertex();
	}

	public Vertex getNormal() {
		return normal;
	}

	public void setNormal(Vertex normal) {
		this.normal = normal;
	}

	public Vertex getVertex1() {
		return vertex1;
	}

	public void setVertex1(Vertex vertex1) {
		this.vertex1 = vertex1;
	}

	public Vertex getVertex2() {
		return vertex2;
	}

	public void setVertex2(Vertex vertex2) {
		this.vertex2 = vertex2;
	}

	public Vertex getVertex3() {
		return vertex3;
	}

	public void setVertex3(Vertex vertex3) {
		this.vertex3 = vertex3;
	}

}
```

**Classe Vertex – Classe cuja função é determinar um objeto do tipo vértice**

```
package Objects;

public class Vertex {

	private float x, y, z;

	public Vertex() {
		x = 0;
		y = 0;
		z = 0;
	}

	public float getX() {
		return x;
	}

	public void setX(float x) {
		this.x = x;
	}

	public float getY() {
		return y;
	}

	public void setY(float y) {
		this.y = y;
	}

	public float getZ() {
		return z;
	}

	public void setZ(float z) {
		this.z = z;

	}
}
```