UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

DESENVOLVIMENTO DE SOFTWARE E HARDWARE PARA DIGITALIZAÇÃO DE PONTOS 3D

LÉO JONATHAN FAHT

BLUMENAU
2005

2005/1-34

LÉO JONATHAN FAHT

DESENVOLVIMENTO DE SOFTWARE E HARDWARE PARA DIGITALIZAÇÃO DE PONTOS 3D

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Paulo César Rodacki Gomes - Orientador

BLUMENAU
2005

2005/1-34

DESENVOLVIMENTO DE SOFTWARE E HARDWARE PARA DIGITALIZAÇÃO DE PONTOS 3D

Por

LÉO JONATHAN FAHT

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Paulo César Rodacki Gomes, Dr. – Orientador, FURB
	_____
Membro:	Prof. Mauro Marcelo Mattos, Dr. – FURB
	_____
Membro:	Prof. Antônio Carlos Tavares – FURB

Blumenau, 22 de junho de 2005

Dedico este trabalho a todos meus familiares, amigos e conhecidos que acreditaram nesta idéia.

AGRADECIMENTOS

A toda minha família, principalmente meus pais Haro e Léa, e meus irmãos Junior, Luciana e Milene, que me apoiaram e me incentivaram durante toda a jornada, muitas vezes me achando meio louco, mas nunca deixando de acreditar nestas loucuras.

A Vivian Pofahl, pelo companheirismo, carinho e compreensão.

Aos bons amigos, Ariberto, Aurélio, Juliane, Nádia e principalmente ao Fernando pelo apoio e por estarem sempre presentes.

Ao meu orientador, Paulo Rodacki, pelo auxílio e dedicação, e por acreditar na minha capacidade de realização do mesmo.

Ao professor Tavares por toda a sua disposição para troca de idéias e auxílio durante o desenvolvimento do hardware.

Aos demais professores do departamento não citados explicitamente, mas que com certeza contribuíram na construção do meu conhecimento.

A empresa Austriamicrosystems, especialmente ao seu representante aqui no Brasil, Fausto Bertoli pela doação dos encoders magnéticos.

Ao meu tio Raul, pela confecção das peças do hardware.

Ao professor Manuel Menezes de Oliveira Neto da UFRGS, pelo auxílio prestado na revisão bibliográfica sobre reconstrução de superfícies.

Nunca se deve engatinhar quando o impulso é voar.

Hellen Keller

RESUMO

Este trabalho aborda o processo de digitalização de objetos tridimensionais, bem como a construção do *hardware* para coletar coordenadas da face de objetos e do *software* necessário para tratá-los, com a finalidade de gerar o seu modelo geométrico computacional. De maneira geral, este processo divide-se em duas etapas: a captura de coordenadas e a triangulação dos pontos capturados. O *hardware* proposto consiste em um braço articulado, que deve ser manipulado por uma pessoa. Cada uma das articulações do braço é monitorada por sensores (*encoders*) que estão interligados ao computador através da porta paralela. O *software* deve ser capaz de ler os valores dos sensores, determinar qual a coordenada da extremidade do braço e armazenar esta coordenada em uma estrutura na memória. Posteriormente, o *software* deve realizar a triangulação dos pontos capturados, criando uma malha composta de faces triangulares, que representa a superfície do objeto digitalizado.

Palavras-chave: Digitalização 3D. Braço digitalizador. Nuvem de pontos. Triangulação.

ABSTRACT

This work presents the process of digitalizing three-dimensional objects, as well as construction of hardware to collect coordinates of the object face and necessary software for treats them, with the purpose to generate its computational geometric model. In general, way this process is divided in two stages: the capture of coordinates and the triangulation of the captured points. The considered hardware is an articulated arm, which must be manipulated by a person. Each one of the joints of the arm is monitored by sensors (encoders) that they are linked to the computer through the parallel port. Software must capable to read the values of the encoders, to determine which the coordinate of the extremity of the arm and to store this coordinate in a structure in the memory. Later, software must carry through the triangulation of the captured points, creating as mesh composed of triangular faces, which represents the surface of the scanned object.

Key-Words: Scanner 3D. Digitizing arm. Clouds of points. Triangulation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente sistemas de modelagem geométrica tridimensional são muito utilizados em diversas áreas e em diversos tipos de aplicações. No entanto, existem objetos que apresentam muita complexidade em sua forma dificultando bastante a sua modelagem. Para estes objetos, a maneira mais fácil de alcançar seu modelo geométrico com perfeição, é realizar a digitalização do objeto real. Este processo é denominado de “engenharia reversa” (VÁRADY, 1997). Os equipamentos que digitalizam objetos tridimensionais são chamados de *scanners* 3D (tridimensionais). Eles se dividem de acordo com o método utilizado para captura de coordenadas, que podem ser: acústicos, magnéticos, ou através de toque no objeto, como está sendo proposto neste trabalho.

Pode-se dividir o processo de digitalização apresentado em duas etapas: uma para coletar coordenadas 3D na superfície do objeto e a outra que consiste em efetuar um processamento, conhecido como triangulação, para formar uma malha de faces triangulares com base nos pontos capturados.

Na etapa de digitalização, optou-se por um *hardware* que utilize o toque no objeto para a captura de coordenadas, porque pode ser desenvolvido com custo significativamente inferior a um outro, como por exemplo, um scanner a laser.

As técnicas de reconstrução das superfícies a partir da nuvem de pontos normalmente são complexas. Algumas levam em consideração a topologia do objeto que foi digitalizado, outras utilizam parâmetros de orientação nos pontos capturados, ou ainda, impõe restrições quanto à topologia do objeto a ser digitalizado.

Este trabalho apresenta o desenvolvimento de *software* e do *hardware* necessários para realizar a digitalização de coordenadas 3D, possibilitando ao final do processo que o usuário visualize o objeto digitalizado no ângulo desejado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver o *hardware* e *software* para realizar todas as etapas de digitalização de um objeto, que compreendem a captura de coordenadas da face do objeto, a visualização dos pontos capturados e a visualização de uma malha composta por faces triangulares representando o objeto digitalizado.

Para tanto, o *hardware* deve permitir a captura de coordenadas do objeto e o *software* deve ser capaz de se comunicar com ele, para coletar as coordenadas capturadas, armazená-las e mostrá-las na tela, e posteriormente, realizar a triangulação destas para gerar o modelo geométrico.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 deste trabalho consiste em uma revisão bibliográfica. Os itens 2.1 e 2.2, introduzem respectivamente o conceito de *software* CAD, citando suas aplicações, e o conceito de engenharia reversa em projetos de engenharia. O item 2.3 visa detalhar o *hardware* de digitalização proposto, com seus subitens falando a respeito de sensores necessários para o seu funcionamento. O item 2.4 trata do acesso a porta paralela no Windows 2000/XP/NT. O item 2.5 introduz noções de transformações geométricas e suas aplicações. Encerrando este capítulo, o item 2.5.3 fala das técnicas utilizadas para a reconstrução de superfícies a partir de uma nuvem de pontos.

O capítulo 3 trata da implementação, onde são apresentados os requisitos para o *software* e o *hardware*, as respectivas especificações, implementações e seus resultados. Nele é mostrado como foi construído o *hardware* para realizar a captura das coordenadas, e como foi feito o processamento para reconstruir o objeto a partir destes pontos capturados.

Por fim são apresentadas no capítulo 4, as conclusões sobre o desenvolvimento deste trabalho, e sugestões pertinentes a trabalhos que podem dar continuidade ao desenvolvimento deste.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas informações sobre as aplicações da digitalização de objetos, o *hardware* utilizado para digitalização de coordenadas 3D, técnicas de reconstrução de superfícies a partir de uma nuvem de pontos, e noções de computação gráfica que foram necessárias ao cálculo das coordenadas. Estes itens serão abordados para o entendimento do capítulo 3.

2.1 SOFTWARES DE CAD/CAM

A sigla CAD vem do termo "*Computer aided design*" que significa "Projeto auxiliado por computador". Da mesma forma, CAM origina-se do termo "*Computer aided manufacture*", que é a "Fabricação auxiliada por computador" (SILVA, 2001).

Segundo Silva (2001), *software* CAD é um *software* que permite realizar desenhos bi e tridimensionais de peças e alterá-los como se desejar. Estes *softwares* estão sendo utilizados em diversas áreas, tais como na indústria metal-mecânica, construção civil e arquitetura. Aqui estamos nos referindo especificamente a engenharia mecânica, que utiliza o auxílio destas ferramentas durante todo o projeto de construção de peças, podendo mensurar cada etapa do projeto de forma adequada.

Segundo Várady (1997) "A existência de um modelo computacional de um objeto proporciona enorme benefício, aumentando a qualidade e eficiência na criação do desenho, produção, e análise do projeto".

2.2 ENGENHARIA REVERSA

Os processos de engenharia reversa são processos que têm como objetivo, realizar o mapeamento de um resultado final, para uma especificação. Por exemplo, a engenharia reversa de um sistema computacional, resulta nos diagramas, que descrevem e idealizam o sistema.

Várady (1997) utiliza este conceito na área de CAD/CAM, onde o objetivo é encontrar o modelo geométrico computacional de um objeto real. Existem várias técnicas que se propõe a realizar esta tarefa. Estas técnicas podem ser separadas em duas etapas: captura de informações e processamento das informações.

A primeira etapa se refere à aquisição de informações do objeto, uma das informações coordenadas da superfície do objeto, e dependendo da técnica de reconstrução pode necessitar de outros parâmetros, como por exemplo um vetor normal para cada ponto. Tal processo necessita de um *hardware* para digitalização 3D que é detalhado na seção 2.3.

A segunda etapa deve utilizar como entrada os dados coletados pela etapa anterior, e processá-los, a fim de obter um modelo geométrico que seja o mais semelhante possível do objeto real. Para tal processamento existem várias técnicas, que serão comentadas no item 2.6.

2.3 HARDWARE PARA DIGITALIZAÇÃO 3D

Os equipamentos que realizam a captura de informações de objetos 3D também são conhecidos como "*scanners 3D*". Segundo Cioqueta e Neto (2003), estes equipamentos se dividem de acordo com o método de captura: sem contato e com contato.

Este trabalho restringe os estudos à digitalização de objetos utilizando o contato com o objeto para a captura de coordenadas. Um *hardware* que realiza a digitalização de objetos 3D

através de toque é conhecido como braço digitalizador. Ele tem a estrutura semelhante a um manipulador robótico, diferindo apenas na sua movimentação, que ocorre de forma manual ao invés de mecânica.

Segundo Pazos (2002), a estrutura de um manipulador consiste basicamente em uma série de corpos rígidos, denominados de elos. As possibilidades de movimento de um elo em relação ao anterior são determinadas de acordo com o tipo de junta que os une. Este movimento pode ser de rotação, onde o elo pode girar um determinado ângulo em relação ao anterior, caracterizando junta de revolução; ou pode ser um movimento linear, onde o elo afasta-se ou aproxima-se do anterior, caracterizando uma junta prismática.

Normalmente, o braço articulado utilizado para digitalização é um equipamento que possui uma ponta sensora, semelhante a uma caneta, interligada a uma base através de elos e juntas de revolução (articulações) com sensores de deslocamento angular acoplados a elas.

O braço digitalizador deve permitir uma boa mobilidade, permitindo a captura de coordenadas de objetos de diversas formas. Para a definição de mobilidade de um manipulador, é utilizado o termo “grau de liberdade”, sendo este grau um número correspondente à quantidade de articulações encontradas no equipamento em questão. Pode-se observar na Figura 1 um exemplo de braço articulado, com 5 articulações.

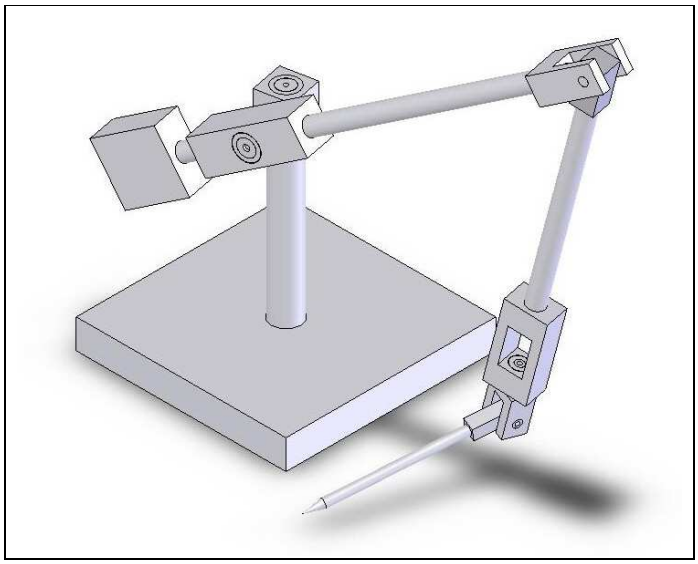


Figura 1 - Braço digitalizador

Um braço digitalizador existente no mercado chama-se *Microscribe* (IMMERSION CORPORATION, 2005). Ele pode ser considerado como uma referência para a precisão na captura das coordenadas. Ele consegue capturar coordenadas com precisão de até 0.1016 mm Para alcançar tal precisão, ele utiliza *encoders* ópticos de alta resolução.

Tendo o conhecimento da estrutura do braço, suas dimensões, e o ângulo das articulações, pode-se calcular a posição 3D da sua extremidade utilizando transformações geométricas 3D. A seção 2.3.1 apresenta sensores utilizados para medir deslocamentos angulares.

2.3.1 Sensores de deslocamento angular

Segundo Matias (2002, p. 36) “O encoder é um transdutor que converte um movimento linear ou angular em pulsos digitais”.

Até pouco tempo atrás, encontrava-se apenas *encoders* ópticos, mas recentemente surgiram no mercado *encoders* magnéticos. Os *encoders* magnéticos se apresentam na forma

de um circuito integrado, e são sensíveis a um campo magnético externo, assim necessitam de cuidados durante sua instalação (mais detalhes serão abordados na seção 2.3.1.1). *Encoders* ópticos são normalmente maiores que os magnéticos e se apresentam em vários formatos e tamanhos que variam de acordo com a precisão oferecida. Este monitora o deslocamento de um eixo próprio, através de discos e sensores de luz internos. Independentemente da tecnologia interna, eles se dividem em duas categorias: incrementais e absolutos (MATIAS, 2002).

Os incrementais possuem duas saídas, A e B, conforme ilustrado na Figura 2, com uma pequena defasagem entre elas. Estas saídas alternam-se entre 1 e 0 durante a movimentação do eixo que está sendo monitorado.

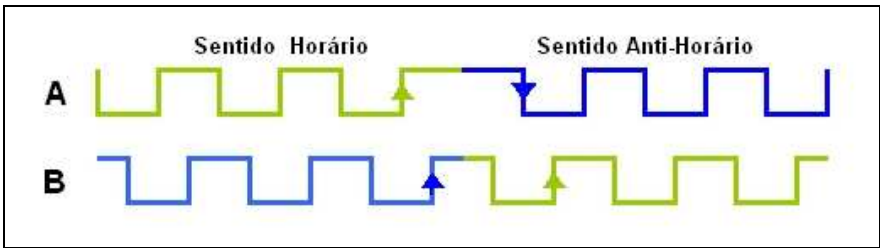


Figura 2 - Exemplo de saída incremental de um *encoder*

Observa-se que a defasagem entre os sinais indica o sentido de rotação do eixo. Este tipo de *encoder* é ideal para aplicações onde não há necessidade de saber em qual posição se encontra o eixo, apenas que ele se movimentou. Porém, se o objetivo for controlar a posição atual do eixo, deve-se ter um outro circuito que monitore estes pulsos. Com esta abordagem surge uma série de problemas, como por exemplo, a calibragem do sistema ao ser ligado.

Para as aplicações onde é necessário ter a posição exata do eixo em dado momento, os *encoders* ideais são os absolutos, eles possuem uma estrutura interna mais complexa, permitindo que se saiba a posição exata do eixo a qualquer momento, mesmo quando acabaram de ser ligados.

Existem vários modelos. Os mais precisos podem ter uma saída com resolução igual ou superior a 15 bits. Isto corresponde a 32768 posições possíveis em 360º, o que torna possível perceber o deslocamento de aproximadamente 0,01º em seu eixo. De acordo com o modelo tem-se um tipo de saída como, por exemplo, serial ou paralela. Vale ressaltar que estes sensores têm um custo normalmente alto, que está diretamente ligado a sua precisão.

2.3.1.1 AS5040

O AS5040 (AUSTRIAMICROSYSTEMS, 2005) é um *encoder* magnético configurável. Sua saída pode ser absoluta de 10 bits, incremental, ou ainda em outros dois modos (PWW e UVW). Ele se apresenta na forma de um CI (Circuito integrado) com dimensões de 5,3mm x 6,2mm. A Figura 3 mostra a foto de um sensor ao lado de uma régua, com esta imagem pode-se ter uma noção do seu tamanho.

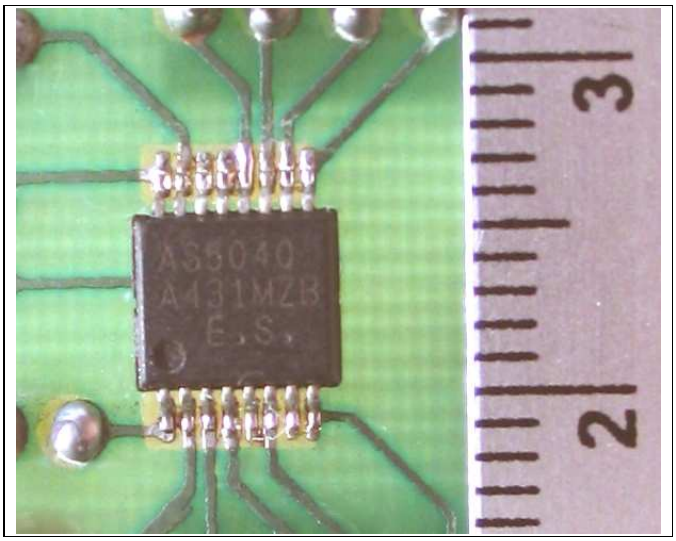


Figura 3 - Foto do AS5040 ao lado de uma régua com escala em centímetros.

Por ser um sensor magnético, ele não possui contato com o eixo que se está monitorando. A captura do movimento ocorre através da variação do campo magnético

gerado por um imã, com uma faixa de magnetismo específica, que deve estar fixo no eixo a ser monitorado e alinhado com o sensor.

Tratando-se de dimensões reduzidas, o processo de fixação e alinhamento do magneto com o sensor é um fator crítico e de grande importância, visto que a tolerância de erro no alinhamento entre eles é relativa ao tamanho do sensor, ou seja, quanto menor o sensor, menor a tolerância.

Para fazer um sensor funcionar em modo absoluto, é necessária a utilização de no mínimo três pinos de comunicação entre ele e o computador. São eles:

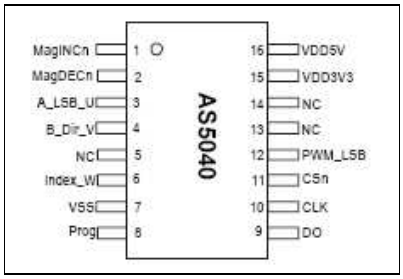
a) DO (Pino 9) – pino de saída, por onde é feita a leitura do sensor;

b) CLK (Pino 10)– pino de *clock* da transmissão, estabelecido pelo computador;

c) CSn (Pino 11)– Pino sinalizador de leitura, é através dele que a leitura é requisitada ao sensor.

E mais os pinos de alimentação VDD5V, VDD3V3 e VSS. Para um melhor detalhamento destes, ou informações sobre outros pinos, pode-se consultar o manual do AS5040 (AUSTRIAMICROSYSTEMS, 2005, p.2).

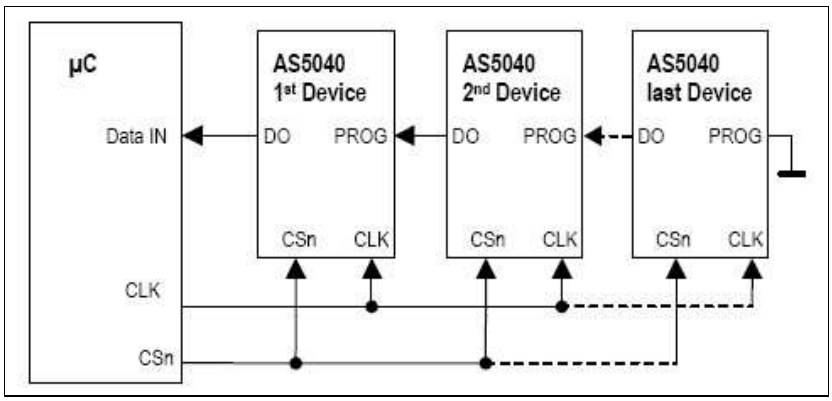
A Figura 4 apresenta a configuração dos pinos do sensor.



Fonte: Austriamicrosystems (2005, p. 2)
Figura 4 - Descrição dos pinos do AS5040

Se a aplicação utilizar mais de um sensor, eles podem ser ligados em cascata conforme ilustra a Figura 5. Para realizar esta ligação, o pino de saída dados (DO), deve ser ligado no

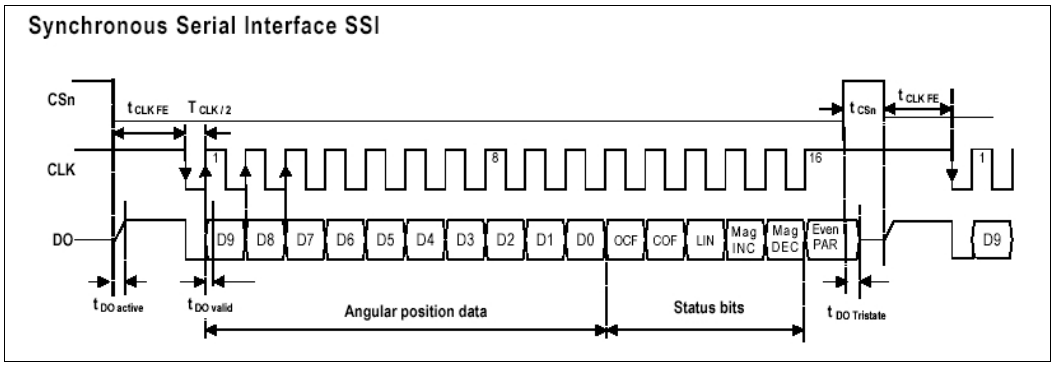
pino de entrada de dados (PROG) do sensor subseqüente, e assim sucessivamente, de acordo com a quantidade de sensores que se deseja interligar.



Fonte: Austriamicrosystems (2005, p.5)

Figura 5 - Sensores ligados em cascata

Desta maneira é possível realizar a leitura de todos os sensores ligados em cascata a partir do pino DO sensor que está conectado diretamente ao computador. A Figura 6 ilustra o estado dos pinos durante a leitura de um sensor. Para iniciar o processo de leitura da posição absoluta do sensor o pino CSn deve ser colocado a baixo. A partir desse momento, deve-se iniciar uma série de pulsos no pino CLK. A cada subida de borda, o sensor disponibiliza um novo bit no pino DO.



Fonte: Austriamicrosystems (2005, p.4)

Figura 6 - Codificação da leitura em modo absoluto

Como se pode observar na Figura 6, os dez primeiros bits se referem a posição absoluta codificada em binário. Os próximos cinco são bits de controle, e o último bit é o cálculo de paridade para garantir a consistência da leitura, totalizando 16 bits.

Tratando-se de sensores conectados em série, existirá um bit de intervalo entre cada conjunto de dados. Portanto o número de bits a ser lido corresponde a n * (16 + 1), onde *n* é o número de sensores interligados.

2.4 ACESSO A PORTA PARALELA NO WINDOWS 2000

Segundo Messias (2005) a porta paralela é uma interface de comunicação entre o computador e um periférico, que existe desde o primeiro computador pessoal fabricado pela IBM, e que foi desenvolvida exclusivamente para permitir a conexão de uma impressora ao computador. Com o avanço da tecnologia esta não é mais a realidade, hoje existe uma infinidade de equipamentos que utilizam esta porta para se comunicar com o PC. Esta porta possui 25 pinos, acessíveis através de três registros, que podem ser observados na Figura 7. Através do registrador "*status"* pode-se ler 5 bits, o registrador "controle" pode enviar 4 bits e o registrador "dados" pode enviar 1 byte paralelamente, deixando a disposição para uso, 12 saídas e cinco entradas.

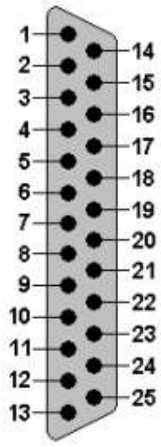
DB25 encontrado no PC (Fêmea)	Nr. do Pino	Descrição	Registro	Sentido
	1	STROBE	Controle	Saída
	14	AUTO FEED		
	16	INIT		
	17	SELECT IN		
	2	D0	Dados	
	3	D1		
	4	D2		
	5	D3		
	6	D4		
	7	D5		
	8	D6		
	9	D7		
	10	ACK	Status	Entrada
	11	BUSY		
	12	PAPER END		
	13	SELECT OUT		
	15	ERROR		
	18 ao 25	GND	-	-

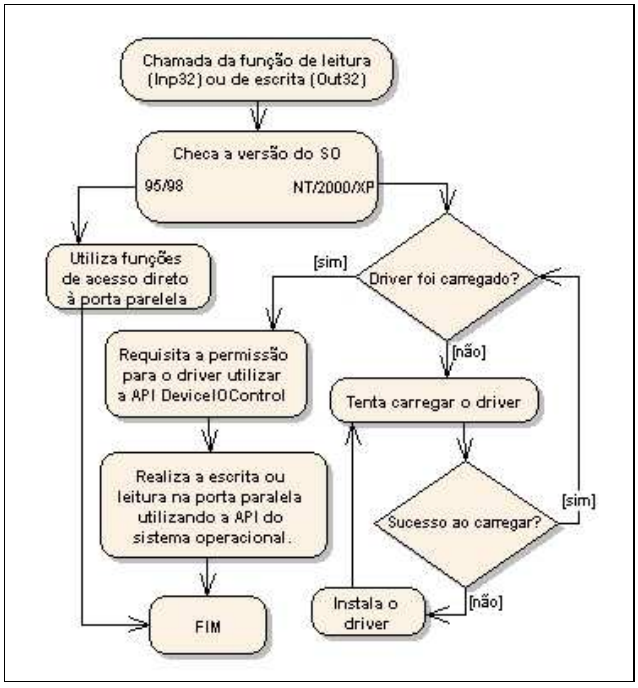
Figura 7 - Conector DB25 e pinagem da porta paralela

O comportamento descrito acima considera a porta paralela em modo SPP (*Standard Parallel Port*). Nos computadores atuais pode-se configurá-la para trabalhar no modo EPP (*Enhanced Parallel Port*), para atingir uma velocidade mais alta na transmissão de dados e também utilizar o registro de dados como entrada e saída.

Segundo LOGIX4U (2005), até a versão do Windows 98, as portas de comunicação do computador podiam ser acessadas diretamente de qualquer programa do usuário, através de comandos particulares de cada linguagem. Assim. Era possível fazer uma aplicação que conseguia ler e escrever dados diretamente na porta paralela sem restrições.

Porém, em versões de Windows como NT, 2000 ou XP este acesso foi bloqueado. Assim, para acessar a porta nestes sistemas operacionais necessitamos de uma DLL (*Dynamic Link Library*) que contenha funções de escrita e leitura na porta paralela. Uma biblioteca pode ser encontrada em LOGIX4U (2005), com o nome de Inpout32.dll.

LOGIX4U (2005) demonstra o funcionamento desta DLL utilizando um fluxograma, ilustrado na Figura 8.



Fonte: LOGIX4U (2005, tradução nossa)

Figura 8 - Funcionamento da Inpout32.dll

Ao ser solicitada a leitura ou escrita de dados na porta paralela através das funções Inp32 ou Out32, a biblioteca verifica a versão do sistema operacional (SO). No caso de ser Windows 2000/XP/NT, ela solicita ao SO a permissão para utilizar a API de comunicação com a porta paralela, caso contrário, realiza o acesso direto ao dispositivo.

2.5 TRANSFORMAÇÕES GEOMÉTRICAS

As técnicas de transformações geométricas são fundamentais na computação gráfica, são técnicas para realizar alterações em um plano cartesiano, tais como alterar o tamanho de um objeto, sua posição, ou rotacioná-lo.

Todas estas operações são aplicadas através da multiplicação dos valores da coordenada, por matrizes de dimensão 4x4. Aqui se faz necessário a utilização de

coordenadas homogêneas, para atender a propriedade de multiplicação de matrizes, no que diz respeito ao tamanho das matrizes envolvidas.

Segundo Adams e Rogers (1990) a coordenada homogênea de um vetor de posição $[x \quad y \quad z]$ é $[x' \quad y' \quad z' \quad 1]$ onde: **x = x'/h, y = y'/h**, **z = z'/h** e **h** é um número real qualquer (com exceção de zero). Para representar uma coordenada qualquer de forma homogênea, sem modificar o seu valor, utiliza-se: $[x' \quad y' \quad z' \quad 1]$.

A operação geral de transformações pode ser expressa como $[P'] = [P][T]$. Esta expressão indica que qualquer transformação pode ser aplicada a um vetor de coordenadas *P*, multiplicando-o por matriz de transformação *T*. O resultado desta operação resulta no vetor *P'* com as modificações aplicadas em relação às coordenadas originais, de acordo com a transformação *T*.

2.5.1 Translação

A translação de um ponto no espaço é a mais simples das transformações, é dada pela soma dos valores da coordenada, aos deslocamentos desejados em cada um dos eixos. A matriz de translação é apresentada no Quadro 1.

$$[T] = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 1 - Matriz de translação

2.5.2 Rotação

A rotação 3D sempre ocorre com base em um dos eixos (*X*, *Y*, ou *Z*). Segundo Adams e Rogers (1990), a rotação em torno de *X* por exemplo, faz com que a coordenada *x* do ponto permaneça inalterada. Isto ocorre porque a rotação ocorre em um plano perpendicular ao eixo da respectiva rotação.

Tratando-se de 3 dimensões, temos 3 eixos através do qual podem ocorrer as rotações, e para cada uma, existe uma matriz específica. As matrizes de rotação sobre os eixos *X*, *Y* e *Z*, são apresentadas respectivamente nos quadros 2, 3 e 4.

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha & 0 \\ 0 & \operatorname{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 2 - Rotação do ângulo α em torno do eixo *X*

Formatados: Marcadores e numeração

$$[T] = \begin{bmatrix} \cos\phi & 0 & \operatorname{sen}\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 3 - Rotação do ângulo ϕ em torno do eixo *Y*

$$[T] = \begin{bmatrix} \cos\psi & -\operatorname{sen}\psi & 0 & 0 \\ \operatorname{sen}\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 4 - Rotação do ângulo ψ em torno do eixo *Z*

Para realizar uma rotação a partir de um eixo arbitrário, deve-se utilizar múltiplas transformações envolvendo rotações e translações, para alinhar este eixo arbitrário com

alguns dos eixos do sistema de coordenadas, e então com os eixos alinhados, aplica-se a rotação com base naquele eixo.

2.5.3 Concatenação de transformações

Em sua obra Adams e Rogers (1990) dedicam um capítulo para demonstrar as transformações 3D. Mostram que múltiplas transformações podem ser concatenadas, através da multiplicação das matrizes. Então podemos compor uma matriz *T* para agrupar *n* transformações desta maneira: $[T] = [T1][T2][T3]\cdots[Tn]$.

Esta propriedade de concatenação de matrizes facilita o processo de múltiplas transformações, e traz um sentido para a utilização de coordenadas homogêneas e matrizes para realizar as operações. Caso contrário seria mais prático deduzir as matrizes em fórmulas para calcular os valores de *x, y,* e *z* de acordo com a operação a ser realizada.

A multiplicação de matrizes é uma propriedade não comutativa $[A][B] \neq [B][A]$, então deve-se atentar para ordem das matrizes envolvidas nas operações.

2.6 RECONSTRUÇÃO DE SUPERFÍCIES

O processo de reconstrução propõe-se a formar o modelo geométrico computacional de um objeto, através de informações sobre sua forma 3D. De posse destas informações deve-se encontrar uma maneira que permita estabelecer uma ligação entre os pontos formando o objeto.

Assim, este processo toma vários rumos, cada qual com sua particularidade, seja pela técnica empregada, ou pelas informações que tem como entrada para realizar a reconstrução dos pontos.

Segundo Góis (2004), nas últimas duas décadas, surgiram vários algoritmos de reconstrução de superfícies, e eles se dividem em duas classes. A primeira é baseada na reconstrução de superfícies através de sessões planares (imagens bidimensionais). A segunda classe é bem mais recente e propõe a reconstrução superfícies através de uma nuvem de pontos, e se encontram vários trabalhos na literatura que buscam por técnicas de reconstrução que tenham eficiência, e garantam a reconstrução da superfície de um objeto através de uma nuvem de pontos.

De acordo com a técnica, esta segunda classe subdivide-se em diversos ramos. As técnicas de reconstrução de superfícies mais populares são:

a) esculpimento: esta técnica realiza inicialmente a triangulação de Delaunay na nuvem de pontos. Esta triangulação, efetuada em pontos dispostos no R3, forma tetraedros denominados de simplexos[1] de Delaunay. Após isto, utiliza-se uma heurística para extrair um conjunto de triângulos que melhor represente o objeto digitalizado (BAJAJ, BERNARDINI e XU, 1995) (EDELSBRUNNER e MÜCKE, 1994);
b) função implícita: um método baseado em função implícita visa definir uma função distância de modo que o conjunto de pontos seja composto pelos zeros desta função. Após a definição da função, pode-se reconstruir a superfície utilizando outros algoritmos para extração de superfícies, que utilizam a função definida e o conjunto de pontos para reconstruir a superfície (HOPPE et al., 1992) (BOISSONNAT e CAZALS, 2000);
c) incrementais: os métodos incrementais partem de uma aresta inicial, ou um conjunto de arestas (denominadas de fronteiras de avanço) e a partir destas arestas realiza uma busca por novos pontos, para formar outras arestas para compor os

[1] Triângulos e suas extensões em outras dimensões, como segmentos de reta, tetraedros, pontos, etc.

triângulos. Normalmente, estes métodos utilizam alguma estrutura de dados especial para que seja realizada a procura por um novo ponto de maneira eficiente (BERNARDINI et al., 1999) (GOPI, KRISHNAN e SILVA, 2000);

d) deformáveis: o objetivo dos modelos deformáveis é partir de uma superfície inicial, onde são aplicadas deformações, para que se enquadre à nuvem de pontos. A agilidade e a eficiência destes algoritmos dependem da distância entre a superfície apresentada em forma de pontos e a superfície inicial (ZHAO et al., 2000) (BARDINETA, COHENB e AYACHE, 1998);

e) família crust: esta classe de algoritmos é a primeira a apresentar garantias de reconstrução próxima a superfície original. Existem várias propostas de algoritmos de reconstrução nesta classe, e vários teoremas para a garantia da reconstrução (AMENTA, CHOI e KOLLURI, 2001).

Para que o resultado apresentado seja fiel à superfície original, todos os algoritmos conhecidos necessitam de uma boa amostragem de pontos da superfície do objeto, com maior densidade em locais que contenham mais detalhes.

3 DESENVOLVIMENTO DO HARDWARE E DO SOFTWARE

Neste capítulo são apresentadas as etapas realizadas para a construção do *hardware* e implementação do *software*: levantamento de requisitos, especificações, e implementação. Também serão detalhadas as técnicas e ferramentas utilizadas na implementação, assim como as dificuldades encontradas e os resultados obtidos.

3.1 REQUISITOS DO PROTÓTIPO

A seguir são detalhados os requisitos de *hardware* e *software*, detalhando-os em seções específicas.

3.1.1 Requisitos do hardware

O *hardware* deve permitir uma mobilidade para tocar em qualquer ponto da superfície do objeto a ser *scaneado*. Ele deve ser composto de uma caneta interligada a uma base, que tenha a mobilidade necessária para se movimentar no espaço ao redor da base. Para permitir esta mobilidade, o meio que interliga a base e a ponta sensora é composto por elos e articulações, conforme o desenho apresentado na Figura 1. Este tipo de *hardware* é conhecido como braço digitalizador.

O braço digitalizador deve permitir que o usuário, em qualquer momento, acione a leitura da posição final da caneta que se encontra em sua extremidade. Neste momento, ele deve enviar um sinal ao computador, informando a necessidade da leitura dos sensores. Depois disso, o *hardware* deve estar preparado para enviar ao computador a valor atual de cada um de seus sensores.

3.1.2 Requisitos de software

O *software* necessário a este protótipo, deve comunicar-se com o *hardware*, permitindo utilizá-lo como uma ferramenta para a digitalização de pontos 3D.

Os requisitos fundamentais ao *software* são:

a) monitorar o *hardware* para saber quando realizar a leitura;
b) realizar a leitura dos sensores;
c) calcular a posição final da caneta com os valores dos sensores;
d) armazenar as coordenadas lidas na memória;
e) possuir uma interface que permita visualizar os pontos capturados;
f) permitir gravar a nuvem de pontos em um arquivo texto que possa ser lido posteriormente;
g) triangular os pontos, gerando uma malha triangular que represente o objeto digitalizado.

3.2 ESPECIFICAÇÃO

Nesta seção são apresentadas as especificações de *hardware* e *software*, atendendo aos requisitos levantados na seção anterior.

3.2.1 Especificação do hardware

A especificação das peças que devem ser construídas para a montagem do hardware é apresentada no apêndice A. Para monitorar as articulações do braço digitalizador, se faz necessária à utilização de cinco sensores. A proposta inicial do trabalho previa a construção

destes sensores de deslocamento angular. Porém, durante a fase de estudos foi conseguida a doação de sensores do tipo AS5040 por parte da própria empresa fabricante.

Para monitorar as articulações do braço digitalizador, se faz necessária a utilização de cinco sensores do modelo AS5040, detalhado na seção 2.3.1.1, interligados em modo cascata. A Figura 9 ilustra as ligações entre o *hardware* e o PC. Os pinos de *clock* (CLK) e de solicitação de leitura (CSN) dos sensores são ligados a um mesmo pino na porta paralela, pois compartilham estes sinais enviados pelo computador.

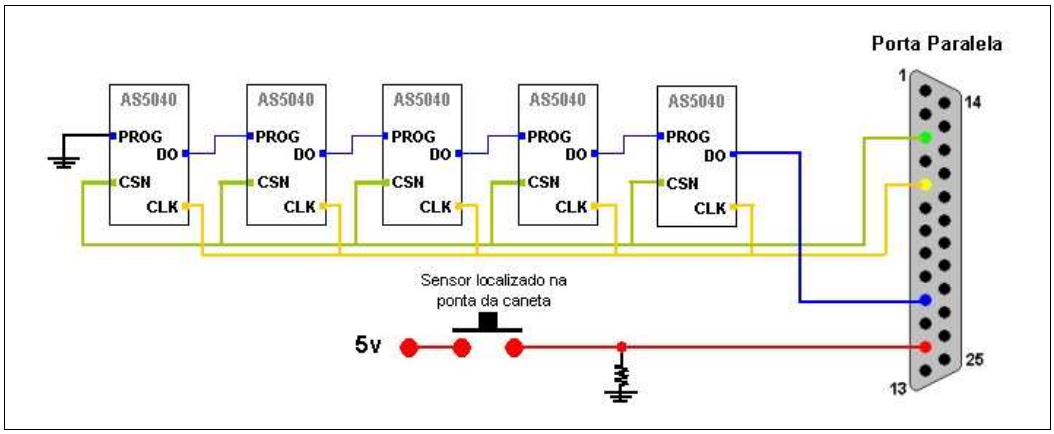


Figura 9 - Esquema de ligação do *hardware* com o computador

É através do pino DO do sensor, ligado diretamente à porta paralela, que são recebidos os valores de todos os sensores que estão ligados em cascata. Este processo de captura de uma coordenada é iniciado sempre que o sensor localizado na ponta da caneta toca em algo, fechando um curto circuito entre a fonte de alimentação e a porta paralela.

Considerando o tamanho do sensor, existe a necessidade de se utilizar placas de adaptação, onde deve ser soldado o sensor e alguns conectores que permitam utilizar seus pinos. O circuito desenhado para a construção destas placas (ver Figura 10), permite que sejam instalados na placa dois conectores com 4 pinos cada, ligados aos pinos necessários

para realizar a leitura dos sensores, e mais um conector de 2 pinos para ligar a fonte de alimentação do sensor.

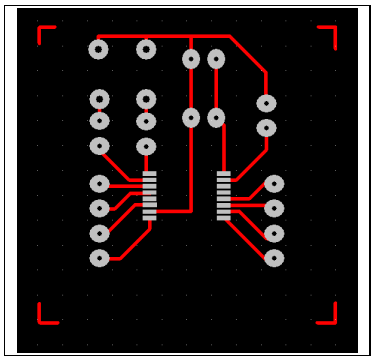


Figura 10 - Esquema da placa de adaptação

A especificação do modo como ocorre a comunicação entre o *hardware* e o *software* é apresentada e comentada na seção 3.2.2, que trata da especificação do *software*.

3.2.2 Especificação do software.

A especificação do *software* foi realizada utilizando diagramas de classe, seqüência e atividades da UML (MATOS, 2002), que é de fato, um padrão na especificação de orientação a objetos. Para construir os diagramas foi utilizado o *software Enterprise Architect* (SPARX SYSTEMS PTY LTDA, 2005).

A seguir pode-se, visualizar o diagrama de classes da aplicação, onde se apresentam as principais classes e os relacionamentos entre elas, desconsiderando os métodos e atributos existentes.

A classe *Interface* mantém o controle da aplicação, é ela que recebe e trata os eventos que ocorrem no programa. Na grande parte destes eventos ela utiliza alguma das outras classes para atendê-los. Para a captura de coordenadas utiliza as classes *ParalelaSinc* e *Calc* para comunicação com o *hardware* e cálculos necessários, e agrega a coordenada capturada

em uma lista. Para realizar a triangulação dos pontos, envia a lista de coordenadas para a classe *Triangulacao* e depois requisita a lista de triângulos gerada. Havendo a necessidade de gravar ou ler um arquivo de pontos é utilizada a classe *ArquivoPontos*, que possui métodos para a manipulação dos arquivos.

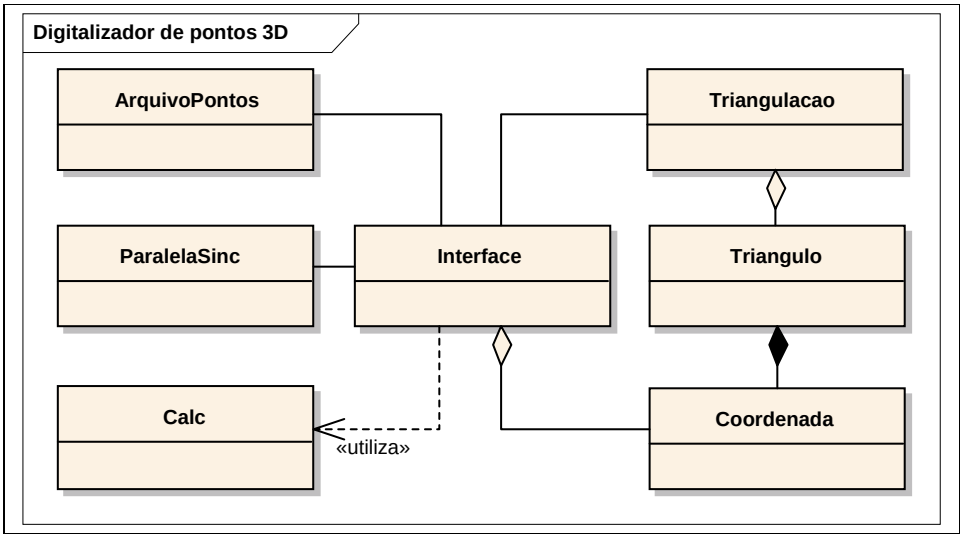


Figura 11 - Diagrama de classes

A seguir, são detalhados os principais métodos de cada uma das classes apresentadas no diagrama.

Figura 12 - Classe ArquivoPontos

A classe *ArquivoPontos* é responsável pela persistência do objeto capturado em um arquivo texto, e disponibiliza os métodos necessários a leitura e escrita destes. O método *gravar* recebe uma lista de coordenadas como parâmetro e grava as coordenadas em um

arquivo texto. O método *ler* realiza a leitura de um arquivo de pontos e tem como retorno uma lista de coordenadas.

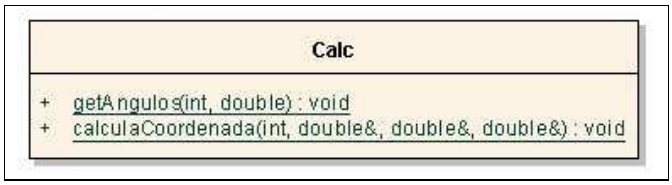


Figura 13 - Classe Calc

A classe *Calc* é uma classe estática utilizada para realizar cálculos. O método *calculaCoordenada* recebe como parâmetro um vetor com os valores lidos dos sensores, e mais três números reais. Ele deve calcular a coordenada correspondente aos valores lidos e retorná-la pelos últimos três parâmetros. O método *getAngulos* recebe como parâmetro dois vetores, o primeiro contém os valores referentes aos sensores, então o método deve, com base nestes valores, calcular o ângulo dos sensores e retorná-los através do segundo parâmetro, que é um vetor de reais.

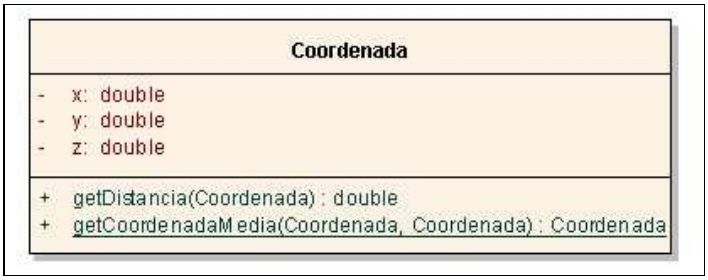


Figura 14 - Classe Coordenada

A classe *Coordenada* é utilizada para representar um ponto capturado. Esta disponibiliza métodos para calcular a distância entre dois pontos e também para o cálculo de um ponto médio entre dois pontos. O método *getDistancia* retorna a distância entre a coordenada passada como parâmetro e aquele ponto. O método *getCoordenadaMedia* é um

método estático que recebe como parâmetro duas coordenadas, cria e retorna uma coordenada que seja intermediara às duas.

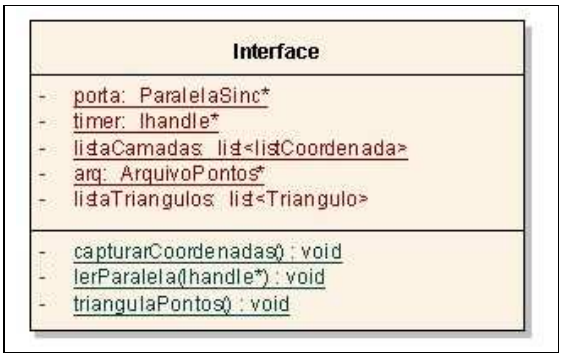


Figura 15 - Classe Interface

A classe *Interface* é responsável pela interface do programa. Todos os eventos de interface são associados a métodos desta classe. A partir dela é realizada a captura de coordenadas, triangulação dos pontos, e a visualização do objeto. Possui uma lista de coordenadas para guardar os pontos digitalizados. O método *capturarCoordenadas* é acionado no momento em que é selecionada a opção de captura de pontos na interface. Ele associa o evento de um temporizador ao método *lerParalela*.

O método *lerParalela* utiliza a classe *ParalelaSinc* para monitorar a porta paralela e se comunicar com o *hardware*. Quando percebe que a ponta sensora toca no objeto, o método solicita à classe *ParalelaSinc* a leitura dos sensores do *hardware*, e depois utiliza classe *Calc* para transformar os valores lidos em uma coordenada. Por fim, ele armazena a coordenada em uma lista e a mostra na tela. Este método é detalhado através do diagrama de seqüência apresentado na Figura 20.

O método *triangulaPontos* é responsável pelo processo de geração da malha triangular. Ele cria um objeto da classe *Triangulação* para o qual passa a lista de coordenadas, e posteriormente requisita a lista dos triângulos formados, para mostrá-los ao usuário.

Figura 16 - Classe ParalelaSinc

A classe *ParalelaSinc* realiza toda a comunicação com o *hardware* através da porta paralela, utilizando a DLL apresentada no item 2.4. Ela interage com o *hardware*, utilizando a configuração de pinos de acordo com o esquema de ligação apresentado na Figura 9.

O método *carregaDLL* é responsável por carregar as funções para realizar escrita e leitura da porta paralela. O método *lerSensores*, recebe como parâmetro o número de sensores a serem lidos, realiza a leitura destes e retorna um vetor de inteiros. Este método é detalhado a seguir no diagrama de atividades apresentado na Figura 19. O método *botaoLeitura*, realiza a leitura da porta paralela e indica se a ponta sensora do braço digitalizador está ou não em contato com o objeto.

Figura 17 - Classe Triangulacao

A classe *Triangulacao* possui métodos que implementam o algoritmo de triangulação apresentado neste trabalho. O método *executa* recebe como parâmetro uma lista de pontos e inicia o processo de triangulação, de acordo com o diagrama de seqüência apresentado na Figura 21. O método *getTriangulos* retorna uma lista de triângulos gerada pelo processo de triangulação.

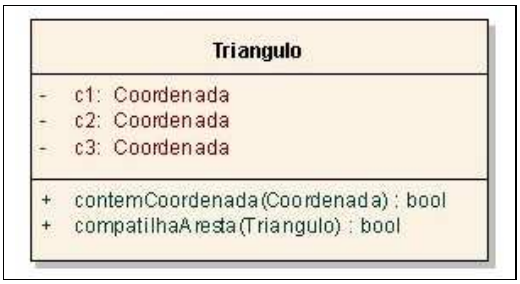


Figura 18 - Classe Triangulo

A classe *Triangulo* é utilizada para representar os triângulos formados pela triangulação. Esta classe é composta por três coordenadas que representam os vértices do triângulo. O método *contemCoordenada*, informa se aquele triângulo é composto ou não, pela coordenada passada como parâmetro.O método *compatilhaAresta*, informa se o triângulo passado como parâmetro compartilha alguma aresta com aquele triângulo.

Feita a apresentação das classes a seguir é apresentada lógica das principais funcionalidades do sistema. A Figura 19 demonstra através de um diagrama de atividades a lógica necessária para realizar a leitura dos cinco sensores.

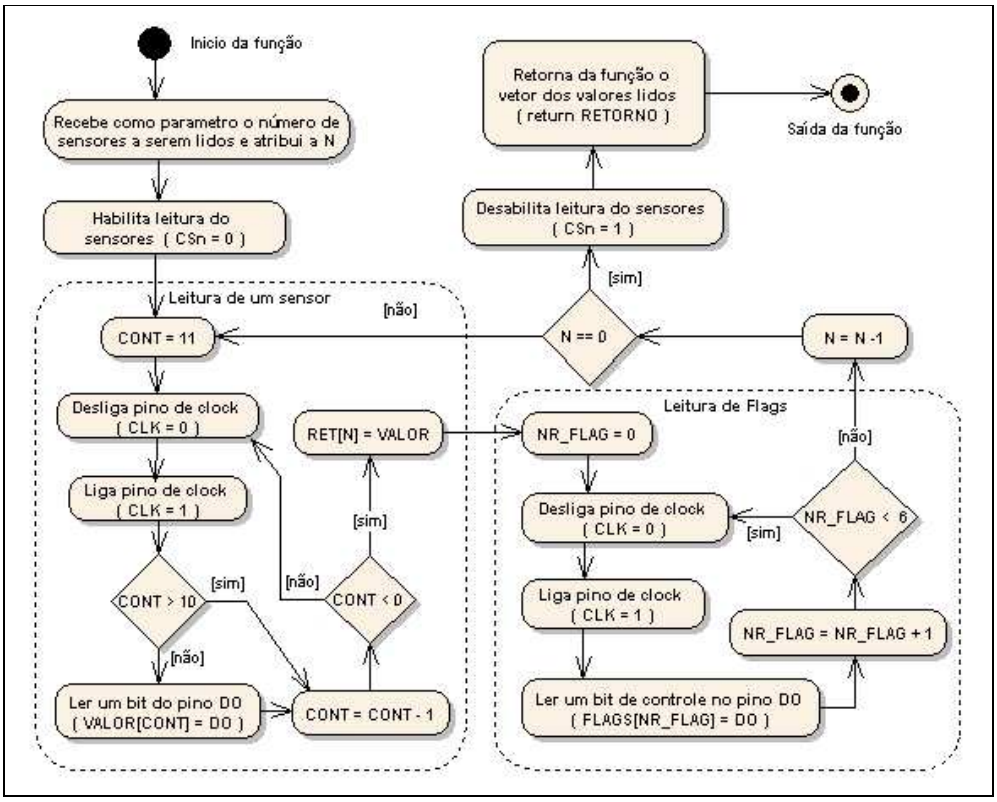


Figura 19 - Diagrama de atividades - Leitura dos sensores

O diagrama ilustra como efetuar a leitura dos sensores, quais bits ler, e quais descartar. Pode-se observar também que uma fonte de *clock* para a leitura é estabelecida pelo programa. De cada sensor são lidos 17 bits, o primeiro bit de cada sensor deve ser desconsiderado, do segundo ao décimo é a posição absoluta, e do décimo primeiro ao último são *flags* de controle.

Os *flags* existentes, com exceção do último, indicam parâmetros de alinhamento do sensor, então, caso o alinhamento não esteja de acordo com as especificações de alinhamento e/ou dimensões, estes bits indicam os problemas. O último bit refere-se à paridade calculada sobre os outros bits transmitidos.

Também foi especificada através de um diagrama de seqüência, a troca de mensagens entre as classes quando é realizada a leitura dos sensores. Veja a Figura 20.

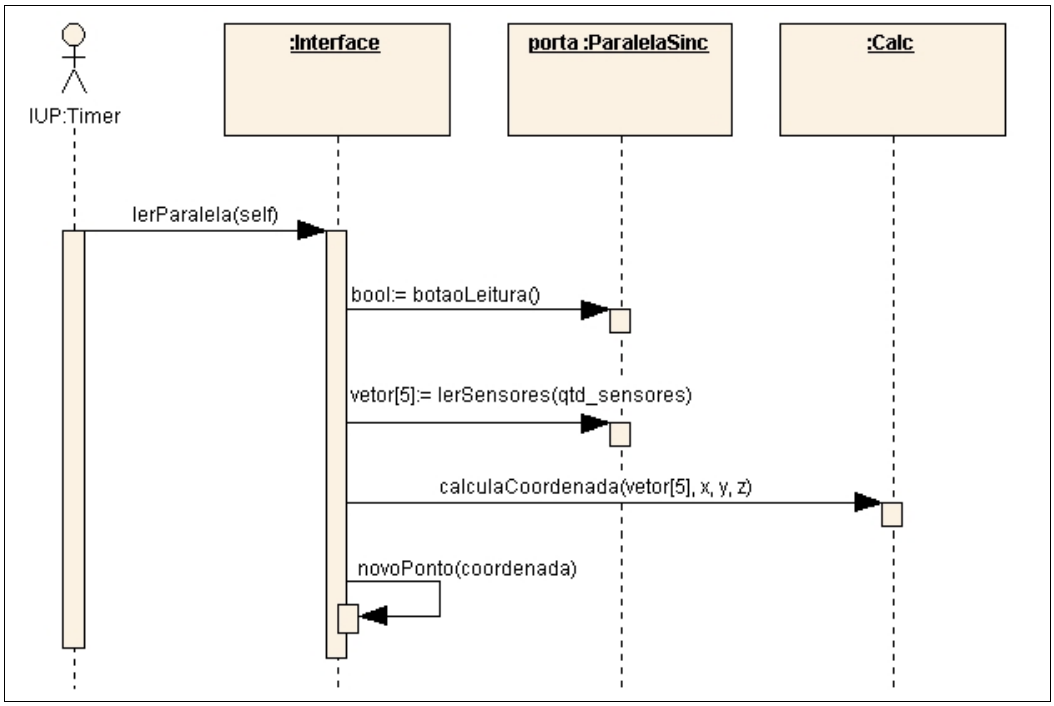


Figura 20 - Diagrama de seqüência - Captura de coordenadas

Este diagrama de seqüência ilustra a troca de mensagens feita entre as classes, durante o monitoramento da porta paralela para captura de coordenadas. O ator representado no diagrama se refere a um temporizador. Este é associado à função *lerParalela* da classe *Interface* assim que o usuário seleciona na interface do programa a opção “Capturar pontos”. A partir deste momento ele chama continuamente a função para verificar se o botão de leitura foi acionado. Caso positivo, o temporizador deve efetuar os passos necessários para capturar uma coordenada e armazená-la em uma lista na memória.

A lógica para realizar a triangulação dos pontos foi especificada utilizando um utilizando um diagrama de seqüência, demonstrado na Figura 21.

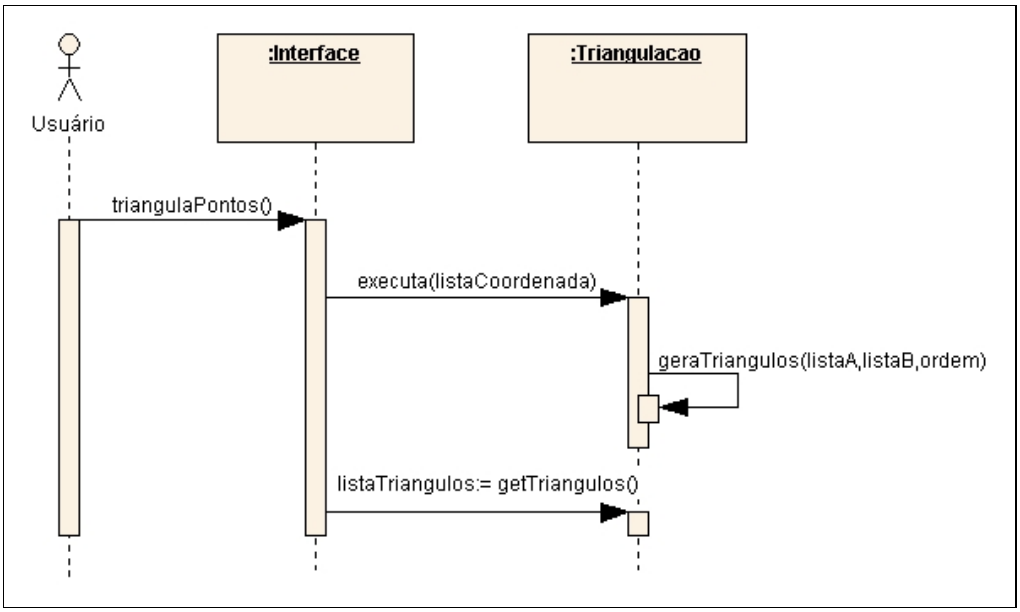


Figura 21 - Diagrama de seqüência 2 - Triangulação dos pontos

A triangulação dos pontos é resultado de uma ação do usuário. O evento é tratado pela classe Interface que por sua vez, chama o método *executa* da classe *Triangulacao*, passando como parâmetro, uma lista de pontos. Este método deve tratar os pontos desta lista de acordo com o algoritmo detalhado na seção 3.3.3. Caso tenha sido possível triangular os pontos, o método *getTriangulos* pode ser utilizado para requisitar à classe *Triangulacao* a lista de triângulos gerada.

3.3 IMPLEMENTAÇÃO

Esta seção contém detalhes de implementação, técnicas e ferramentas utilizadas no desenvolvimento do trabalho.

3.3.1 Hardware

O *hardware* proposto pode ser considerado um conjunto de sensores interligados, que monitoram os movimentos de um braço articulado, o qual é chamado de braço digitalizador. Sua implementação pode ser interpretada como sua construção, já que não foi utilizado nenhum tipo de microcontrolador.

Para a construção de suas peças, a idéia inicial era contratar um serviço de tornearia, para que as peças ficassem de acordo com o desejado, sem folgas e completamente alinhadas. Esta idéia foi descartada após a consulta de preços deste serviço, que tornaria o desenvolvimento do trabalho muito caro.

Uma alternativa encontrada para não abortar o desenvolvimento, foi dividir as peças em partes, conforme a Figura 22, diminuindo a complexidade da construção de maneira que fosse possível construí-las realizando furos perpendiculares a superfície. Com esta alternativa foi possível construir as peças utilizando uma furadeira de bancada e madeira como matéria prima. Apesar da madeira não ser um material ideal para esta aplicação, foi o mais viável a ser utilizado, optando-se pela canela por ser um dos tipos de madeira menos frágeis.

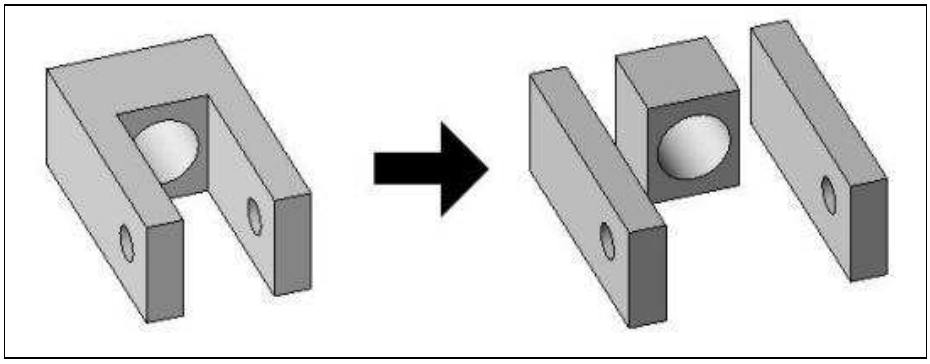


Figura 22 - Peça dividida em partes

A utilização de madeira e o processo manual para confecção das peças diminuem a precisão do *hardware*, devido a fragilidade da madeira, e aos erros de alinhamento dos furos.

Nos eixos das articulações foram utilizados rolamentos para evitar folgas e proporcionar suavidade aos movimentos.

Para realizar o pedido de fabricação das placas de circuito impresso para adaptação do sensor, foi utilizado o esquema apresentado na Figura 10. A Figura 23a apresenta uma imagem da placa fabricada.

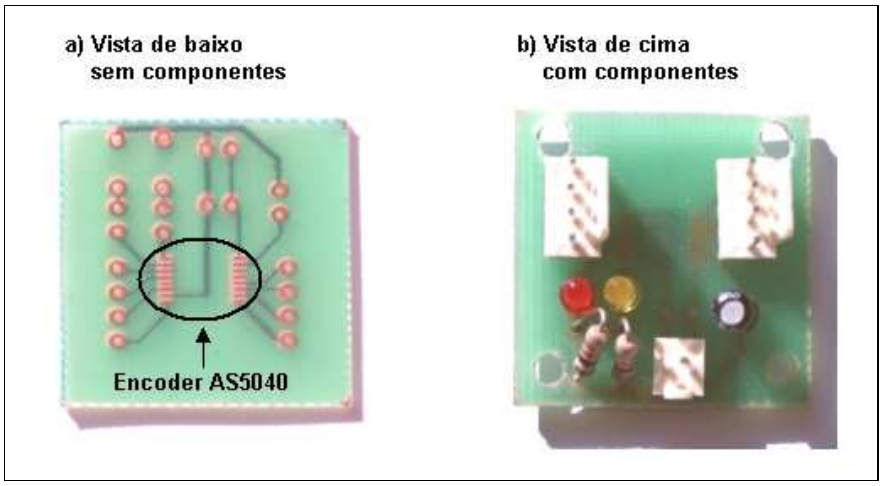


Figura 23 - Placa de adaptação do *encoder*

Com as placas de circuito impresso confeccionadas, foi necessário o auxílio de um serviço especializado para soldar os *encoders* a elas. Depois disso foram soldados os conectores, *leds*, resistores e capacitores necessários ao funcionamento dos sensores e as placas foram furadas em suas bordas, conforme pode ser visto na Figura 23b, para permitir a fixação delas nas articulações, conforme apresentado na Figura 24.

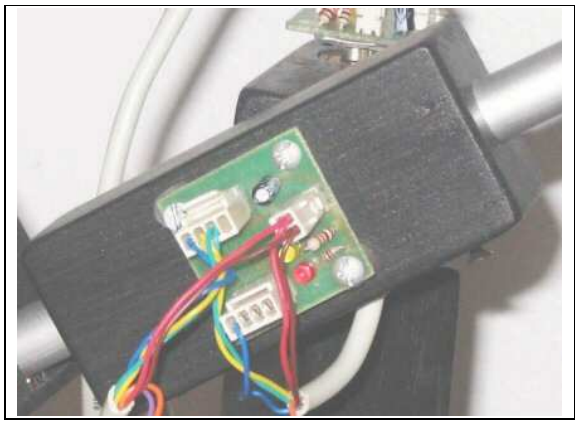


Figura 24 - *Encoder* acoplado na articulação

Os magnetos adquiridos para serem utilizados no projeto, entraram em conflito com as especificações do manual do *encoder*. Eles possuem uma faixa de magnetismo um pouco acima da considerada ideal. Esta diferença fez com que o sensor não responda corretamente à rotação do magneto. Durante testes realizados, percebeu-se que a utilização de dois imãs unidos lateralmente, fazia com que o sensor funcionasse corretamente. Foi adotada esta solução devido à dificuldade de encontrar magnetos na faixa de magnetismo especificada como ideal.

3.3.2 Captura de coordenadas

A captura de coordenadas é resultado de uma ação de usuário, que é percebida pelo computador através da ligação existente entre o sensor encontrado na caneta com a porta paralela. Quando este sensor envia um sinal para o computador, significa que o usuário está capturando uma coordenada e conseqüentemente existe a necessidade de leitura dos sensores.

Devido a uma restrição na fase de triangulação, que é discutida na seção 3.3.3, os pontos devem ser capturados em camadas, como se fossem fatias do objeto.

O processo de captura das coordenadas consiste em ler os sensores, calcular os ângulos com base nos valores lidos, calcular a posição final do braço e armazenar as coordenadas.

Estes procedimentos são discutidos nas seções 3.3.2.1, 3.3.2.2, 3.3.2.3 e 3.3.2.4, respectivamente.

3.3.2.1 Leitura dos sensores

A classe *ParalelaSinc* provê métodos básicos de escrita a leitura da porta paralela e também implementa métodos que tratam de todos os detalhes para a leitura dos sensores. Ela implementa a especificação do diagrama de atividades da Figura 21.

Feita a leitura dos sensores, o novo passo consiste em transformar os cinco valores lidos em ângulos, conforme é apresentado na seção 3.3.2.2.

3.3.2.2 Cálculo do ângulo

Para realizar o cálculo do ângulo com base nos valores lidos, existe a necessidade de um valor de referência. Para obter este valor, houve a necessidade de realização uma calibragem da posição inicial dos sensores. Esta calibragem consistiu em deixar as articulações com ângulo 0º e realizar a leitura dos sensores. Com base nestes valores, e sabendo-se que a saída de um sensor é um valor entre 0 e 1024, pode-se encontrar uma fórmula para transformar o valor lido em um ângulo. A fórmula utilizada pode ser vista no Quadro 5.

$$\text{Ângulo} = (posiçãoZero - valorLido + 1024) \bmod 1024$$

Quadro 5 - Fórmula para cálculo dos ângulos

Dependendo do modo como foi fixado o sensor ao braço, pode haver a necessidade de inverter o resultado multiplicando-o por –1, para inverter o sentido da rotação.

Estes cálculos são implementados na classe *Calc*, através do método com assinatura *void getAngulos(int sensores[5], double sensores[5])* que recebe dois parâmetros. O primeiro é um vetor com os cinco valores lidos e o segundo parâmetro um vetor de números reais, que é passado por referência, para que sejam armazenados nele os ângulos calculados. O código deste método é mostrado a seguir no Quadro 6.

```
void Calc::getAngulos(const int sensores[5], double angulos[5]){
   int sensor;

   // valores adquiridos durante a fase de calibragem, indicam
   // o valor correspondente ao ângulo zero de cada sensor
   int posZero[5] = {390,794,793,955,659};
   for(int i=0; i<5; i++){
      sensor = (posZero[i]-sensores[i]+1024)%1024;// 0 == 0º
                                                  // 1024 == 360º
      angulos[i] = (double)-360*sensor/1024;
   }
   angulos[4]*=-1; // inverte a rotação de sensor posicionado
                   // de forma inversa em relação aos outros
}
```

Quadro 6 - Método para cálculo dos ângulos.

Após esta conversão dos valores lidos do sensor em ângulos, a última etapa da captura de coordenadas consiste em calcular a posição da ponta sensora do braço digitalizador.

3.3.2.3 Cálculo da posição final do braço.

Como visto na seção 2.5, as transformações geométricas podem ser utilizadas para modificar a posição de um ponto no espaço. Então se pode utilizá-las para calcular a posição da ponta sensora, localizada na extremidade do braço digitalizador, tratando-a como um ponto no espaço sobre o qual se aplicam transformações sucessivas para modificar o seu posicionamento, totalizando 10 transformações até chegar a posição final.

Este processo será detalhado e demonstrando passo a passo, compondo uma matriz final, sobre o qual será possível aplicar os valores de ângulos e elos, e obter a coordenada correspondente.

Adota-se como início a origem do sistema de coordenadas 3D (0,0,0), e a partir dela realiza-se transformações de acordo com a estrutura do braço, podendo ser translações, para os elos de ligações entre as articulações, ou então rotações sobre o eixo em que giram as articulações.

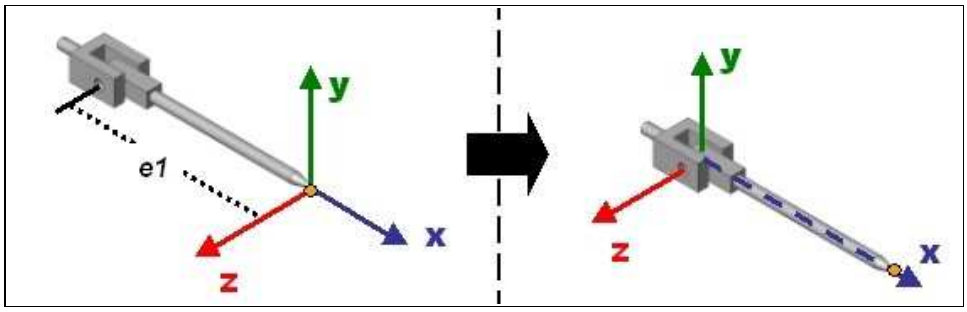


Figura 25 - 1ª operação no cálculo da coordenada

A Figura 25 representa a primeira operação a ser aplicada, que consiste em uma translação de *e1* pelo eixo *X*, onde *e1* corresponde à distância entre a ponta sensora e a articulação mais próxima a ela. Representa-se esta transformação através da matriz de translação, substituindo inserindo o elo *e1* para realizar o deslocamento sobre o eixo *X*, como apresentado no Quadro 7.

$$[T_1] = \begin{bmatrix} 1 & 0 & 0 & e1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 7 - Cálculo da coordenada – 1ª transformação

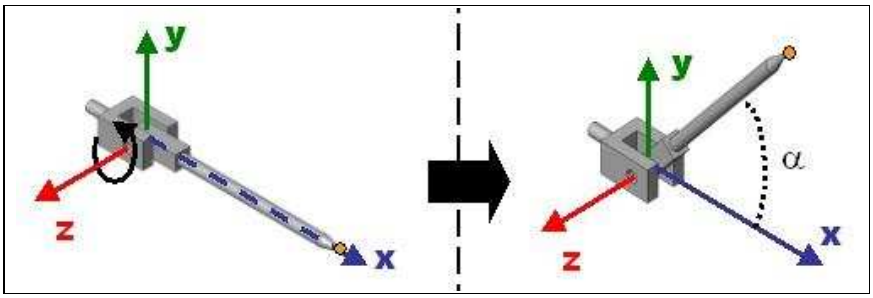


Figura 26 - 2ª operação no cálculo da coordenada.

A Figura 26 ilustra o segundo passo, que consiste em realizar uma transformação de rotação sobre o eixo *Z*, tal operação é representada por uma matriz de rotação. Todas as operações devem ser concatenadas a operação anterior, neste caso, a matriz $[T_1]$. O Quadro 8 apresenta a concatenação destas matrizes.

$$[T_{21}] = [T_2] \cdot [T_1]$$

$$[T_{21}] = \begin{bmatrix} \cos\alpha & -\text{sen}\,\alpha & 0 & 0 \\ \text{sen}\,\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & e1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{21}] = \begin{bmatrix} \cos\alpha & -\text{sen}\,\alpha & 0 & \cos\alpha \cdot e1 \\ \text{sen}\,\alpha & \cos\alpha & 0 & \text{sen}\,\alpha \cdot e1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 8 - Cálculo da coordenada – 2ª transformação

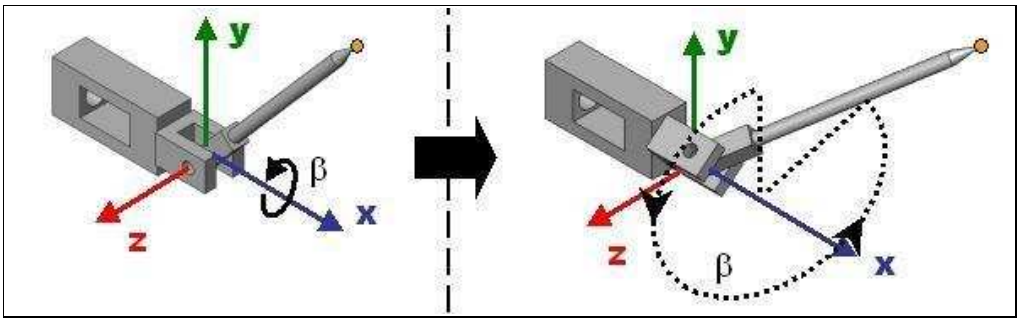


Figura 27 - 3ª operação no cálculo da coordenada

A Figura 27 representa a rotação do segundo sensor sobre o eixo *X*. Observa-se que as rotações ocorrem perpendiculares a um plano, então ao rotacionar uma coordenada em torno de *Y*, o valor *y* desta coordenada não se altera. O Quadro 9 apresenta a transformação de forma matricial.

$$[T_{321}] = [T_3] \cdot [T_{21}]$$

$$[T_{321}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\operatorname{sen}\beta & 0 \\ 0 & \operatorname{sen}\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 & \cos\alpha \cdot e1 \\ \operatorname{sen}\alpha & \cos\alpha & 0 & \operatorname{sen}\alpha \cdot e1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{321}] = \begin{bmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 & \cos\alpha \cdot e1 \\ \operatorname{sen}\alpha \cdot \cos\beta & \cos\alpha \cdot \cos\beta & -\operatorname{sen}\beta & \operatorname{sen}\alpha \cdot e1 \cdot \cos\beta \\ \operatorname{sen}\alpha \cdot \operatorname{sen}\beta & \cos\alpha \cdot \operatorname{sen}\beta & \cos\beta & \operatorname{sen}\alpha \cdot e1 \cdot \operatorname{sen}\beta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 9 - Cálculo da coordenada – 3ª transformação

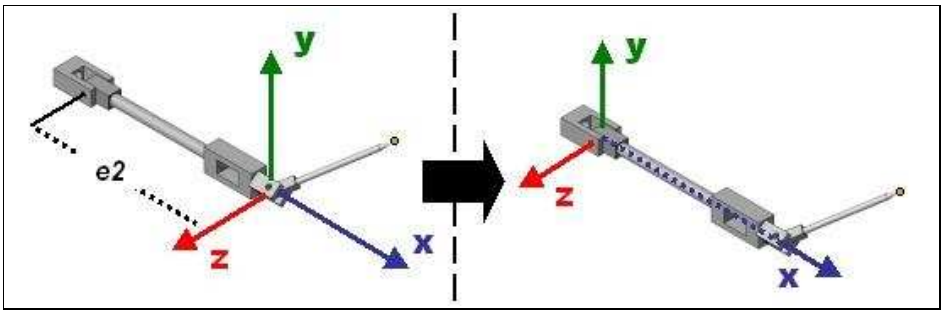


Figura 28 - 4ª operação no cálculo da coordenada

A Figura 28 apresenta outra translação *X* correspondente segundo elo de ligação entre duas articulações, nomeado de *e2*. Esta operação deve ser concatenada as operações anteriores, conforme demonstrado no Quadro 10.

$$[T_{4321}] = [T_4] \cdot [T_{321}]$$

$$[T_{4321}] = \begin{bmatrix} 1 & 0 & 0 & e2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 & \cos\alpha \cdot e1 \\ \operatorname{sen}\alpha \cdot \cos\beta & \cos\alpha \cdot \cos\beta & -\operatorname{sen}\beta & \operatorname{sen}\alpha \cdot e1 \cdot \cos\beta \\ \operatorname{sen}\alpha \cdot \operatorname{sen}\beta & \cos\alpha \cdot \operatorname{sen}\beta & \cos\beta & \operatorname{sen}\alpha \cdot e1 \cdot \operatorname{sen}\beta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{4321}] = \begin{bmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 & \cos\alpha \cdot e1 + e2 \\ \operatorname{sen}\alpha \cdot \cos\beta & \cos\alpha \cdot \cos\beta & -\operatorname{sen}\beta & \operatorname{sen}\alpha \cdot e1 \cdot \cos\beta \\ \operatorname{sen}\alpha \cdot \operatorname{sen}\beta & \cos\alpha \cdot \operatorname{sen}\beta & \cos\beta & \operatorname{sen}\alpha \cdot e1 \cdot \operatorname{sen}\beta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 10 - Cálculo da coordenada – 4ª transformação

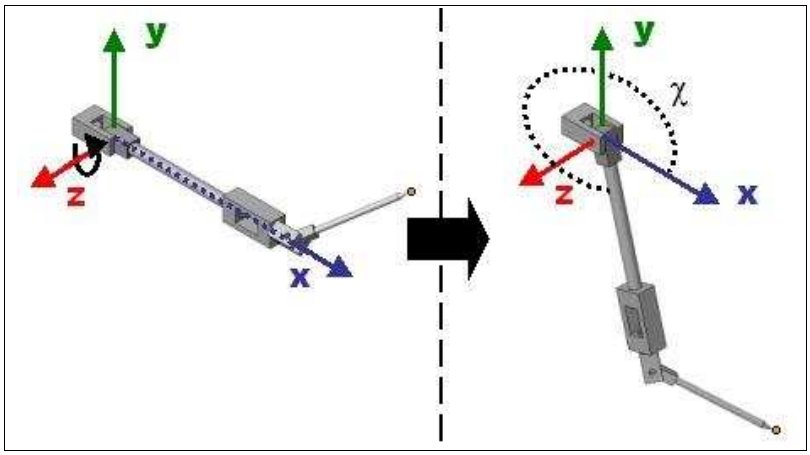


Figura 29 - 5ª operação no cálculo da coordenada

A Figura 29 ilustra a rotação da terceira articulação em torno do eixo *Z*. A partir desta operação serão omitidos os resultados parciais das transformações matriciais, sendo apresentado no fim das transformações, a matriz composta resultante. O Quadro 11 apresenta esta operação sendo concatenada as operações anteriores.

$$[T_{54321}] = [T_5] \cdot [T_{4321}]$$

$$= \begin{bmatrix} \cos\chi & -\text{sen}\,\chi & 0 & 0 \\ \text{sen}\,\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\text{sen}\,\alpha & 0 & \cos\alpha \cdot e1 + e2 \\ \text{sen}\,\alpha \cdot \cos\beta & \cos\alpha \cdot \cos\beta & -\text{sen}\,\beta & \text{sen}\,\alpha \cdot e1 \cdot \cos\beta \\ \text{sen}\,\alpha \cdot \text{sen}\,\beta & \cos\alpha \cdot \text{sen}\,\beta & \cos\beta & \text{sen}\,\alpha \cdot e1 \cdot \text{sen}\,\beta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quadro 11 - Cálculo da coordenada – 5ª transformação

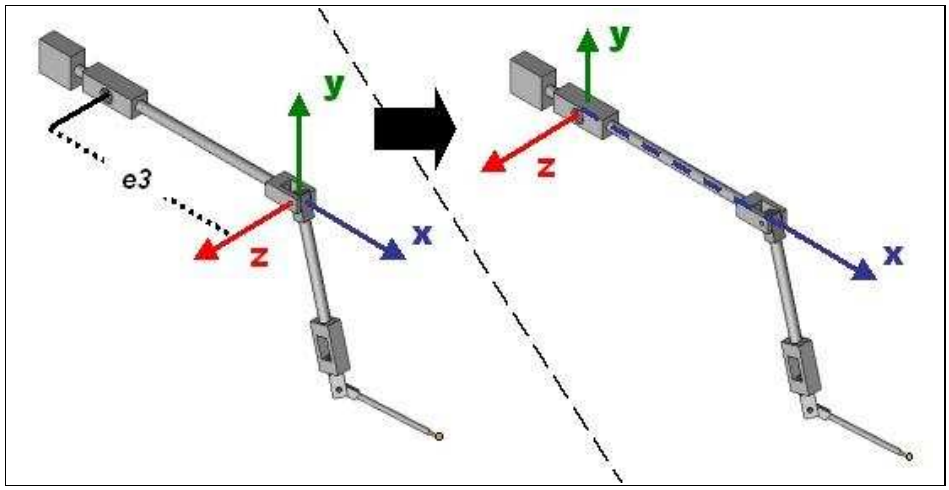


Figura 30 - 6ª operação no cálculo da coordenada

Representa-se na Figura 30 uma translação sobre o eixo *X* correspondente ao elo de ligação *e3*. O Quadro 12 apresenta esta transformação.

$$[T_6] = \begin{bmatrix} 1 & 0 & 0 & e3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{654321}] = [T_6] \cdot [T_{54321}]$$

Quadro 12 - Cálculo da coordenada – 6ª transformação

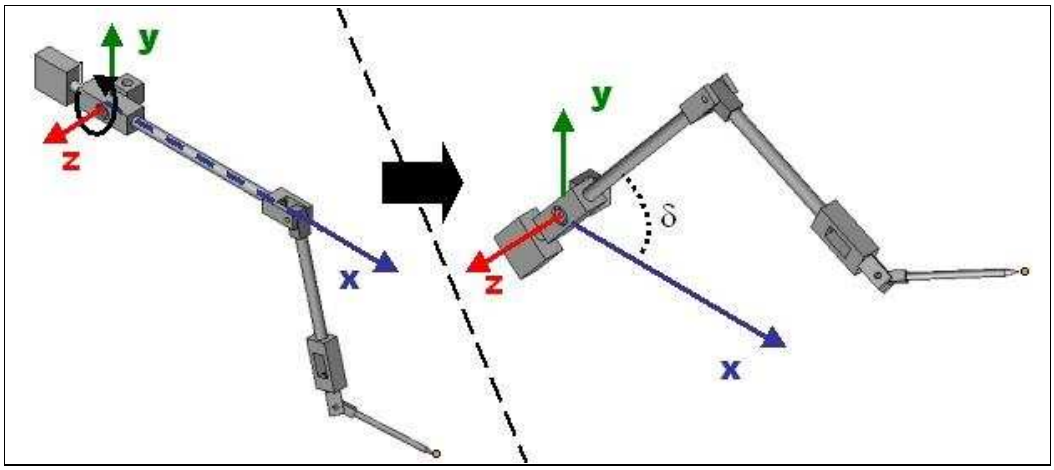


Figura 31 - 7ª operação no cálculo da coordenada

A sétima transformação necessária é apresentada na Figura 31 mostra a rotação referente ao sensor número 4. A representação matricial encontra-se no Quadro 13.

$$[T_7] = \begin{bmatrix} \cos\delta & -\text{sen}\,\delta & 0 & 0 \\ \text{sen}\,\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{7654321}] = [T_7] \cdot [T_{654321}]$$

Quadro 13 - Cálculo da coordenada – 7ª transformação

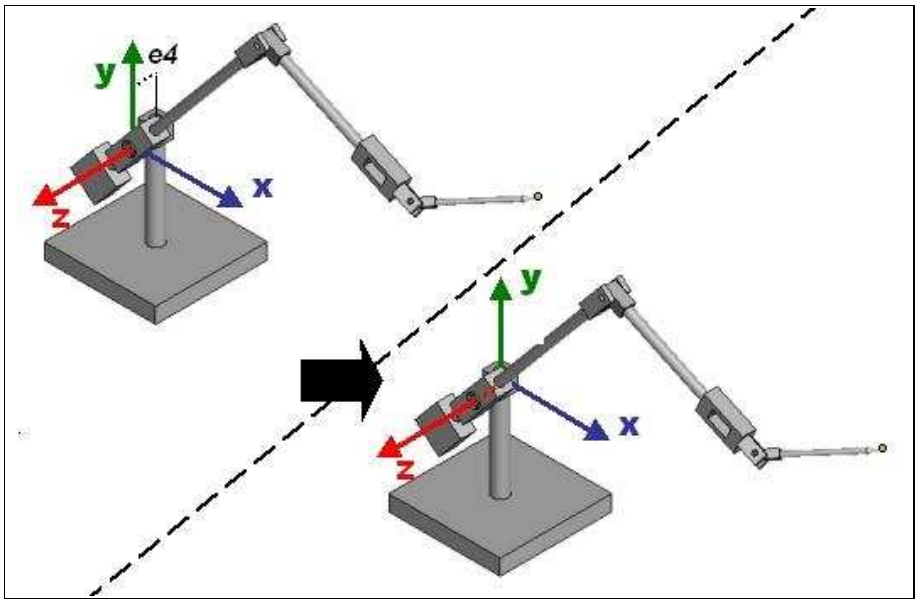


Figura 32 - 8ª operação no cálculo da coordenada

A Figura 32 demonstra um deslocamento pelo eixo Z, devido a um pequeno elo existente entre estas duas articulações. Continua-se concatenando as transformações como representado no Quadro 14.

$$[T_8] = \begin{bmatrix} 1 & 0 & 0 & -e4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{87654321}] = [T_8] \cdot [T_{7654321}]$$

Quadro 14 - Cálculo da coordenada – 8ª transformação

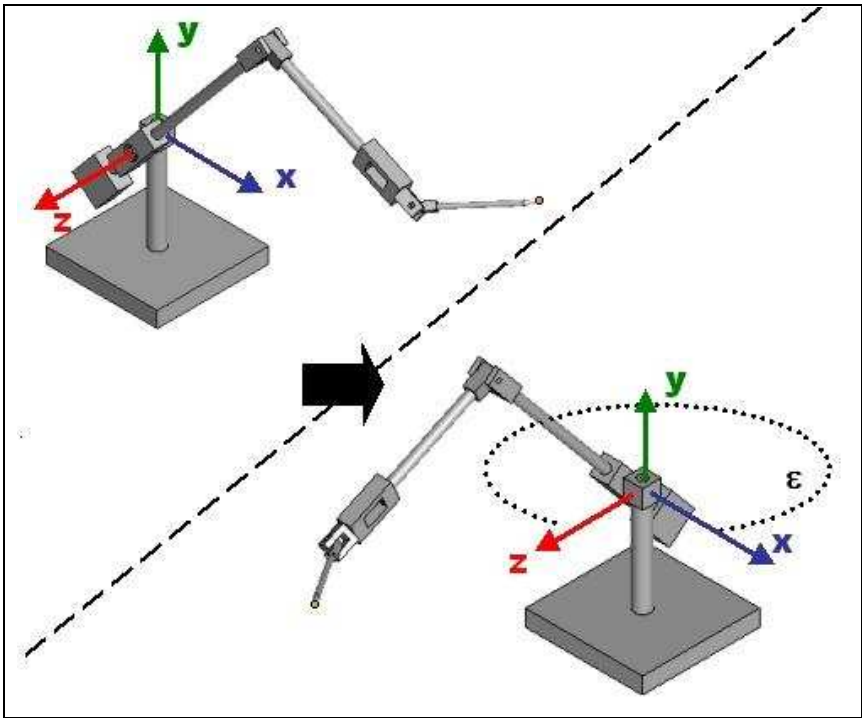


Figura 33 - 9ª operação no cálculo da coordenada

A Figura 33 ilustra a quinta e última rotação realizada, de acordo com o ângulo lido do sensor. No Quadro 15 é apresentada esta transformação.

$$[T_9] = \begin{bmatrix} \cos\varepsilon & 0 & \text{sen}\,\varepsilon & 0 \\ 0 & 1 & 0 & 0 \\ -\text{sen}\,\varepsilon & 0 & \cos\varepsilon & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{987654321}] = [T_9] \cdot [T_{87654321}]$$

Quadro 15 - Cálculo da coordenada – 9ª transformação

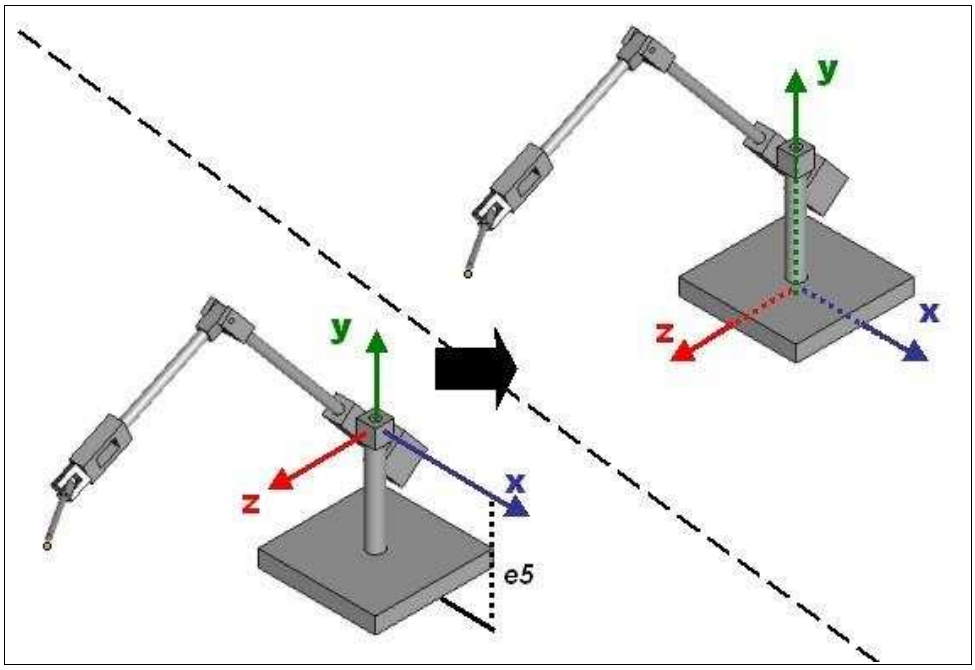


Figura 34 - 10ª operação no cálculo da coordenada

A operação ilustrada na Figura 34, consiste em realizar uma translação em *Y* para compensar o deslocamento realizado pelo elo que interliga a base do braço aos outros membros. Esta é a última operação e poderia ser ignorada sem prejudicar a qualidade do modelo capturado, visto que ela só altera a altura da coordenada no espaço, fazendo com que o valor de *y* seja igual a zero no plano sobre o qual está situada a base do braço digitalizador.

$$[T_{10}] = \begin{bmatrix} 1 & 0 & 0 & e5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_{10987654321}] = [T_{10}] \cdot [T_{987654321}]$$

Quadro 16 - Cálculo da coordenada – 10ª transformação

No Quadro 16 é apresentada a transformação, onde $[T_{10987654321}]$ é a matriz composta pelas dez transformações aplicadas. Para o cálculo da posição final do braço deve-se

multiplicá-la pela coordenada inicial (0,0,0), para obter a coordenada transformada que indica a posição da ponta sensora. Esta operação é apresentada no Quadro 17.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [T_{10987654321}] \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Quadro 17 - Cálculo da coordenada a partir da matriz composta

Os quadros 18, 19 e 20, apresentam os valores de *x, y, z.*

x = cosε • (cosδ • ((cosχ • (cosα • e1 + e2) - senχ • cosβ • senα • e1) + e3) - senδ • (senχ • (cosα • e1 + e2) + cosχ • cosβ • senα • e1)) + senε • (senβ • senα • e1 - e4)

Quadro 18 - Fórmula para cálculo de x

y = (senδ • ((cosχ • (cosα • e1 + e2) - senχ • cosβ • senα • e1) + e3) + cosδ • (senχ • (cosα • e1 + e2) + cosχ • cosβ • senα • e1)) + e5

Quadro 19 - Fórmula para cálculo de y

z = -senε • (cosδ • ((cosχ • (cosα • e1 + e2) - senχ • cosβ • senα • e1) + e3) - senδ • (senχ • (cosα • e1 + e2) + cosχ • cosβ • senα • e1)) + cosε • (senβ • senα • e1 - e4)

Quadro 20 - Fórmula para cálculo de z

A classe *Calc* disponibiliza métodos para realizar as transformações básicas (translações e rotações) e para efetuar o cálculo das coordenadas. Este método recebe quatro parâmetros. O primeiro é um vetor de cinco inteiros, contendo os valores lidos dos sensores. Os outros três parâmetros, passados por referência, retornam os valores da coordenada calculada. Este método transforma os valores do vetor em ângulos, e efetua todas as transformações geométricas ilustradas. O código dele pode ser visto no Quadro 21.

```
void Calc::calculaCoordenada(int sensor[5], double &x, double &y,
double &z){
     // vetor com o tamanho dos elos da base para a ponta sensora
   int elo[6]= {230,75,455,300,105,185};

   double angulos[5];
     // chama função para cálculo dos ângulos com base nos valores
     // lidos dos sensores
   getAngulos(sensor, angulos);
   // inicia valores da coordenada como sendo a origem
     // do sistema de coordenadas 3D
     x=y=z=0;
     // CHAMA SUCESSIVAS TRANSLAÇÕES E ROTAÇÕES PARA
     // CALCULAR A POSIÇÃO FINAL DA PONTA SENSORA
     Calc::transladar(elo[5],0,0,x,y,z);  //elo 6 – caneta
     Calc::rotacionarY(angulos[4],x,y,z); // sensor 5
     Calc::transladar(elo[4],0,0,x,y,z);  // elo 5
     Calc::rotacionarY(5,x,y,z);//Erro do Hardware - punho
     Calc::rotacionarX(angulos[3],x,y,z); //sensor 4
     Calc::transladar(elo[3],0,0,x,y,z);  //elo 4
     Calc::rotacionarZ(-5,x,y,z);//Erro do Hardware - cotovelo
     Calc::rotacionarY(angulos[2],x,y,z); //sensor 3
     Calc::transladar(elo[2],0,0,x,y,z);  //elo 3
     Calc::rotacionarY(angulos[1],x,y,z); //sensor 2
     Calc::transladar(0,elo[1],0,x,y,z);  //elo 2
     Calc::rotacionarZ(angulos[0],x,y,z); //sensor 1
     Calc::transladar(0,0,elo[0],x,y,z);  //elo 1 - base
}
```

Quadro 21 - Método para cálculo de coordenada

Este código poderia ser substituído pelas fórmulas encontradas realizando a concatenação das transformações, porém, existem os comandos citados como “erro de hardware”, que foram inseridos para minimizar os erros causados pela imprecisão existente nas peças que compõe as articulações do braço digitalizador, que não foram inseridos durantes as transformações demonstradas. Poderia-se realizar os cálculos inserindo estas transformações durante composição da matriz de transformação, e então substituir o código apresentado por fórmulas únicas para o cálculo dos valores *x*, *y*, *z*.

3.3.2.4 Armazenando as coordenadas

Como citado anteriormente, os pontos são capturados em camadas, e são internamente armazenados em listas distintas para cada camada. Uma lista principal armazena outras listas que contêm coordenadas. Na implementação do protótipo foram utilizadas listas da classe *list*, disponíveis na biblioteca padrão de gabaritos C++ (MUSSER, SAINI e DERGE 2001, p.345).

As coordenadas permanecem na memória, e só são gravadas em disco caso seja selecionada a opção para gravá-las em um arquivo. O arquivo para armazenar as coordenadas é um arquivo texto, onde é armazenada uma coordenada por linha, separando os valores x, y, z por espaços. Para indicar o início de uma nova camada no arquivo, é utilizada uma coordenada com valores (10000, 10000, 10000). O *layout* deste arquivo texto é detalhado no Quadro 22.

```
10000 10000 10000      // indica início da camada 1
x y z                  // coordenada 1 da camada 1
x y z                  // coordenada 2 da camada 1
...
x y z                  // coordenada n da camada 1
10000 10000 10000      // indica início da camada 2
x y z                  // coordenada 1 da camada 2
x y z                  // coordenada 2 da camada 2
...
x y z                  // coordenada n da camada 2
...
10000 10000 10000      // indica início da camada m
x y z                  // coordenada 1 da camada m
x y z                  // coordenada 2 da camada m
...
x y z                  // coordenada n da camada m
```

Quadro 22 - Layout do arquivo de coordenadas

3.3.3 Triangulação dos pontos

Uma triangulação deve estabelecer uma ligação entre os pontos de maneira que forme uma malha de faces triangulares 3D. Como visto na revisão bibliográfica, existem técnicas que utilizam apenas os pontos sem nenhuma informação, outras utilizam alguns parâmetros adicionais.

No presente trabalho, optou-se por realizar a triangulação através de camadas, como se fossem fatias. Em cada camada os pontos devem seguir a seqüência em que se apresentam na superfície do objeto.

Esta técnica facilita a implementação da triangulação, pois com os pontos estruturados desta forma, pode-se restringir bastante a procura de pontos para formar um triângulo. A seguir são listadas as propriedades definidas para a triangulação.

Sendo uma estrutura de camadas (*Ci*, C_{i+1}, ... C_n), tendo cada uma delas um conjunto de pontos (Pi, P_{i+1}, ... P_n) definiu-se que:

a) é necessária a existência de no mínimo duas camadas para efetuar a triangulação;
b) um triângulo sempre é formado por dois pontos de uma camada C_i e um ponto da camada C_{i+1}, ou dois pontos da camada C_{i+1} e um ponto da camada C_i;
c) todos os pontos de uma camada *C* são interligados na ordem em que foram capturados, e o último ponto é interligado ao primeiro ponto. Ex: P_i e P_{i+1}; P_{i+1} e P_{i+2}; ... P_{n-1} e P_n; P_n e P_i;
d) sendo P_m um ponto médio entre dois pontos P_i e P_{i+1} de uma mesma camada C_i, será escolhido o ponto da camada C_{i+1} mais próximo de P_m, para formar um triângulo com P*i* e P_{i+1};
e) sendo *x* a soma dos pontos da camada C_i e C_{i+1}, serão formados *x*-1 triângulos entre estas camadas;

De acordo com as propriedades citadas acima, foi realizada a implementação tratando individualmente conjuntos compostos por cópias de duas camadas vizinhas, como exemplificado na Figura 35. Os próximos parágrafos detalham esta triangulação.

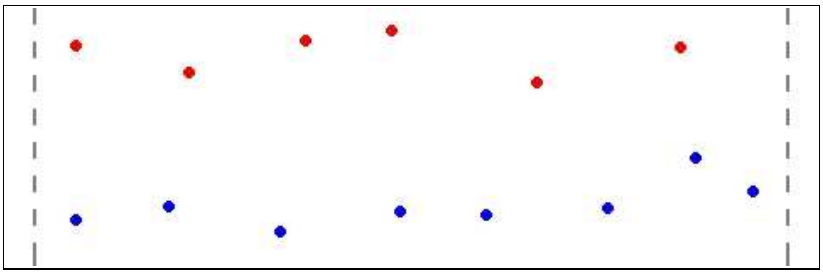


Figura 35 - Triangulação – Camadas vizinhas

A camada com maior quantidade de pontos será nomeada de *ATUAL*, para referenciá-la de maneira mais prática durante as explicações que seguem neste item. Utilizar-se-ão também duas listas para armazenar triângulos, nomeadas de *LT* e *LTAUX*, assumindo que *LTAUX* é uma lista auxiliar que é a cada nova execução é criada e *LT* uma lista que armazena os triângulos criados nas diversas execuções deste pseudocódigo.

Os pontos da camada *ATUAL* devem ser interligados de acordo com a seqüência em que se apresentam, e para cada aresta formada entre eles, procura-se na camada vizinha, o ponto que esteja mais próximo ao centro da aresta. Forma-se um triângulo, com dois pontos subseqüentes da camada *ATUAL* e com o ponto pesquisado na camada vizinha, e adiciona-o a *LTAUX*. Este passo deve ser efetuado para todos os pontos, finalizando ao interligar o último ponto com o primeiro. Após efetuado este passo o resultado deve ficar semelhante ao ilustrado na Figura 36.

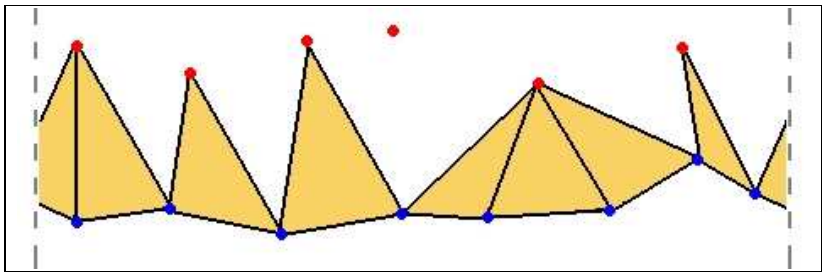


Figura 36 - Triangulação - Passo 1

Neste momento temos vários triângulos, sendo todos formados obrigatoriamente por dois pontos da camada *ATUAL* e um ponto da camada vizinha. Cada ponto da camada *ATUAL* pertence a exatamente dois triângulos contidos em *LTAUX*. Recuperam-se estes dois triângulos e verifica-se se eles possuem alguma aresta em comum (como é o caso dos triângulos formados pelos pontos que foram destacados com um quadrado da cor verde na Figura 37). O caso positivo quer dizer que estes dois triângulos já isolaram o vértice e ele deve ser removido da lista *ATUAL*. Este segundo passo deve ser efetuado com todos os vértices da camada *ATUAL*, removendo todos os vértices que estejam isolados.

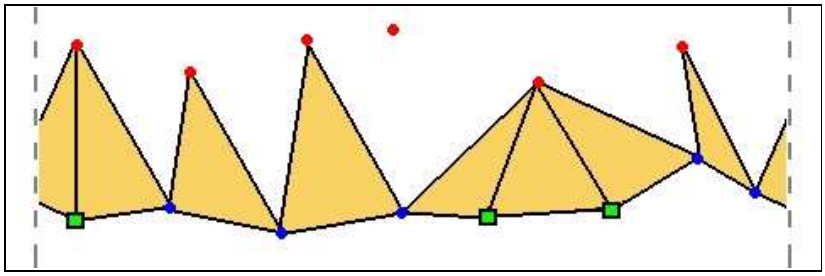


Figura 37 - Triangulação - Passo 2

O próximo passo implementado consiste em varrer novamente a lista *ATUAL* e, para cada vértice existente (neste momento, apenas os vértices ilustrados em cor azul, na Figura 37, ainda estão na lista), recuperar de *LTAUX* os triângulos formados por ele. Com posse dos dois triângulos, seleciona-se de cada um deles, o vértice que pertence à camada vizinha de ATUAL. Feita esta seleção têm-se dois vértices, v_i e v_j. Então se verifica qual é o vértice

inicial para que se possa percorrer a lista seqüencialmente, sem que se passe por algum vértice que já faça parte de algum triângulo existente em *LTAUX*.

Definida esta seqüência, percorre-se o caminho formado entre v_i e v_j, utilizando v_n que é inicializado com v_i e recebe a cada nova iteração o próximo vértice do caminho. A cada iteração é forma-se um triangulo composto por v_n, v_{n+1} e um ponto da camada ATUAL, adicionando-o em *LT*. O resultado após a execução deste passo, seria uma "cinta" formada por triângulos entre as duas camadas de pontos, conforme a Figura 38.

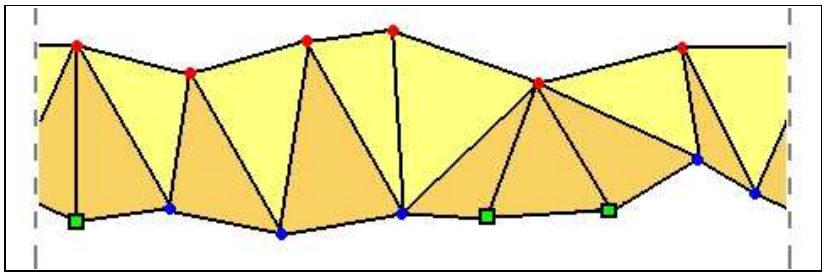


Figura 38 - Triangulação - Passo 3

Para finalizar removem-se os triângulos de *LTAUX*, adicionando-os em *LT*.

Os Quadro 23 apresenta um algoritmo que tem como entrada uma lista de camadas. Ele verifica qual a camada com maior número de pontos e chama o método *FormaTriangulos* para formar os triângulos entres as duas camadas Ca e Cb.

```
PercorreCamadas(LC: Lista de camadas)
    Ca, Cb: Lista de pontos;
    Ca = LC[0];
    LT: lista de triângulos;
    para I = 1 até tamanho(LC)-1 faça
        Cb = LC[I];
        se tamanho(Ca) > tamanho(Cb) então
            FormaTriangulos(Ca, Cb, LT)
        senão
            FormaTriangulos(Cb, Ca, LT)
        fim-se
        Ca = Cb;
    fim-para
Fim.
```

Quadro 23- Algoritmo para percorrer as camadas

O Quadro 24 apresenta o algoritmo do método que se encarrega de formar os triângulos entre as duas camadas passadas como parâmetro. O terceiro parâmetro da função é uma lista de triângulos passada como referência, então ao fim das chamadas ao método *FormaTriangulos*, têm-se em nesta lista todos os triângulos formados.

```
FormaTriangulos(Ca, Cb: Lista de pontos; LT lista de triangulos)
    ltAux: Lista de triangulos;
    T, Ta, Tb: Triangulo;
    Pa, Pb, Pc, Pm: Ponto;
    Pa = último ponto de Ca;
    para I:= 0 até tamanho(Ca)-1 faça
        Pb = Ca[I];
        Pm = PontoMédio(Pa, Pb);
        Pc = ponto mais próximo a Pm pertencente e Cb;
        T = NovoTriangulo(Pa, Pb, Pc);
        ltAux += T;
        Pa = Pb;
    fim-para
    para I:= 0 até tamanho(Ca)-1 faça
        Pa = Ca[I];
        Busca em ltAux triângulos formados por Pa atribuindo-os a Ta e Tb.
        se Ta e Tb não compartilham aresta então
            Pb = ponto de Ta pertencente a Cb;
            Pc = ponto de Tb pertencente a Cb;
            faça
                Pm = ponto seguinte  a Pb;
                T = NovoTriangulo(Pa, Pb, Pm);
                LT += T;
            enquanto Pm <> Pc;
        fim-se
    fim-para
    Adicione em LT o conteúdo de ltAux;
Fim.
```

Quadro 24 - Método para formar triângulos entre duas camadas.

3.3.4 Técnicas e ferramentas utilizadas

Esta seção é destinada a detalhar as principais técnicas e bibliotecas utilizadas na implementação, que não são de conhecimento do grande público. A construção do *software* utilizou como base a plataforma do Visual C++ 6.0 (MICROSOFT CORPORATION, 2005), com o código orientado a objetos.

3.3.4.1 Interface portável ao usuário - IUP

O IUP (*Interface User Portable*) (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, 2005a) é uma biblioteca para construção de interfaces, traz como vantagens a facilidade de uso e portabilidade. Em conjunto com a IUP, utilizou-se o LED que é uma linguagem para especificação de interface, onde todos os elementos de interface são definidos em um arquivo texto.

A criação da interface é feita a partir de elementos especificados no arquivo LED. O IUP possui uma função que se chama *IupLoad,* que é responsável pela criação de elementos gráficos a partir de um arquivo LED.

A Figura 39 apresenta um código simples em LED e seu resultado após ser carregado e exibido pela IUP.

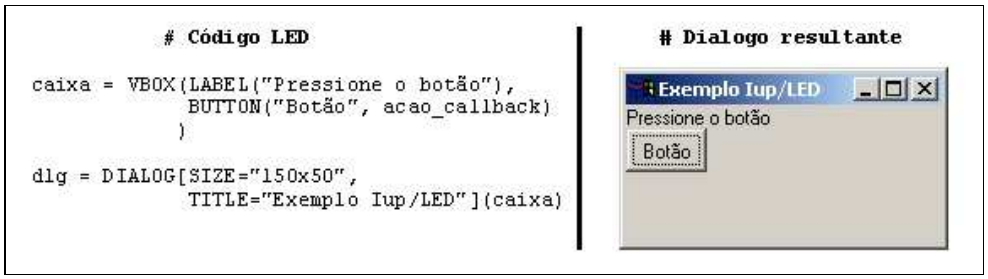


Figura 39 - Exemplo de um diálogo no IUP

Na ocorrência de eventos nos elementos de interface, eles são mapeados para funções. Estas funções são chamadas de *callbacks*. Para associar uma função a um elemento de interface utiliza-se a função *IupSetFunction*, que recebe dois parâmetros, o primeiro é o nome da "*ação_callback*" definida no arquivo LED e o segundo é a função para o qual será mapeado aquele evento. O Quadro 25 mostra um exemplo de código C para fazer a associação de um botão com uma função que dará uma mensagem ao usuário.

```
.
.
void mensagem(){
   IupMessage(“Mensagem”, “Você pressionou o botão”);
}

int main(){
   .
   .
   // registro de função callback
   IupSetFunction(“acao_callback”,(Icallback)mensagem);
   .
   .
   return 0;
}
```

Quadro 25 - Exemplo de associação entre um botão e uma callback

Adotou-se a IUP neste trabalho, pela sua facilidade na construção de interfaces em C/C++, e também por permitir de forma nativa a utilização de OPENGL (SWEET, 2000) para desenho no canvas.

Para controlar a visualização do canvas utilizando o mouse, foi utilizada uma biblioteca chamada VGL (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, 2005b), que trata os eventos de mouse e teclado que ocorrem no canvas com a intenção de modificar a visualização.

3.3.5 Operacionalidade da implementação

O programa conta com uma interface que se apresenta de acordo com a Figura 40 . Ela se divide em um menu situado na parte superior, logo abaixo uma barra de ferramentas com opções de manipulação de arquivos e visualização.E por fim, abaixo da barra de ferramentas encontra-se o canvas, onde são exibidos os resultados da digitalização e da triangulação ao usuário.

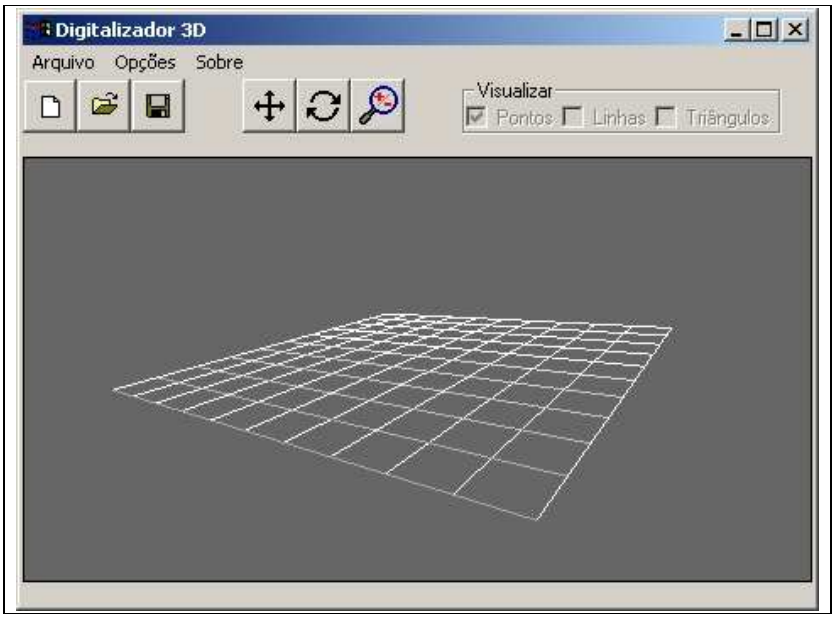


Figura 40 - Interface da aplicação

O menu arquivo apresenta as opções comumente utilizadas no tratamento de arquivos: novo, abrir, fechar, salvar e sair. O segundo menu, intitulado *Opções*, apresenta opções para limpar o canvas, capturar coordenadas ou finalizar a captura, e triangular pontos do canvas. Para a opção de captura de coordenadas, o usuário deve selecionar esta opção e após manipular o braço digitalizador para captura das mesmas.

Para a manipular a visualização do conteúdo do canvas, o usuário pode utilizar os botões localizados na barra de ferramentas, que permitem mover o objeto, rotacioná-lo, ou aplicar zoom. Existe também a possibilidade de animar a visualização, segurando a tecla *shift*, e efetuando uma ação (mover, rotacionar, aplicar *zoom*) deve-se soltar o botão do mouse com ele ainda em movimento a operação de visualização executada é animada.

3.3.5.1 Operacionalidade do hardware

Neste item será exemplificado como realizar a captura de coordenadas de um objeto, utilizando o braço digitalizador que foi construído mostrado na Figura 41.

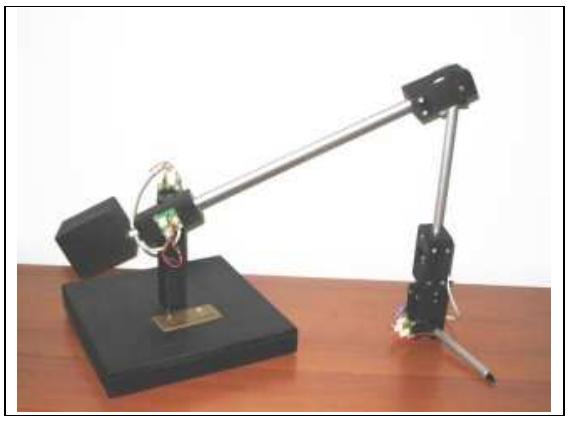


Figura 41 - Foto do braço digitalizador

Para que o *hardware* se torne funcional, ele deve ser ligado a uma fonte de alimentação, e também conectado a porta paralela do computador. Estas entradas para conexão dos respectivos cabos se encontram na lateral da base, de acordo com a Figura 42.

Figura 42 - Entradas de dados e alimentação

Figura 43 - Chave liga/desliga e conector para sensor de leitura

Na parte superior da base, como apresentado na Figura 43, é encontrada uma chave interruptora para ligar e desligar a alimentação dos sensores e também é encontrado um conector que está diretamente ligado aos pinos da ponta sensora do braço. Este pode ser ligado a um botão externo, caso haja a necessidade de capturar coordenadas sem o toque no objeto, por exemplo, uma superfície muito delicada que não permita a pressão necessária para a percepção do toque.

Estando com o cabo de alimentação ligado a tomada e o cabo de dados ligado à porta paralela, deve-se ligar a chave que está localizada na parte superior da base para iniciar o procedimento e captura de coordenadas.

Na interface deve-se selecionar a opção *Capturar coordenadas*, que foi descrita anteriormente. A partir deste momento, para capturar cada uma das coordenadas, basta pressionar a ponta sensora contra objeto e soltar; soará um bip indicando a captura da coordenada, e o ponto é desenhado na tela.

Como a captura de coordenadas deve ser realizada em camadas, sugere-se a demarcação do objeto, com pontos dispostos em linhas horizontais conforme a imagem da apresentada na Figura 44. A captura dos pontos deve ser realizada obedecendo a uma seqüência, de acordo com o exemplo apresentado na Figura 45, do primeiro ponto seguindo até o último, sendo que o ponto posterior ao último, foi o primeiro ponto capturado. Ao capturar o último ponto deve-se deixar a ponta sensora em contato com o objeto por

aproximadamente 3 segundos, um duplo bip soará indicando o fim de captura de coordenadas daquela camada.

Figura 44 - Vaso demarcado com linhas

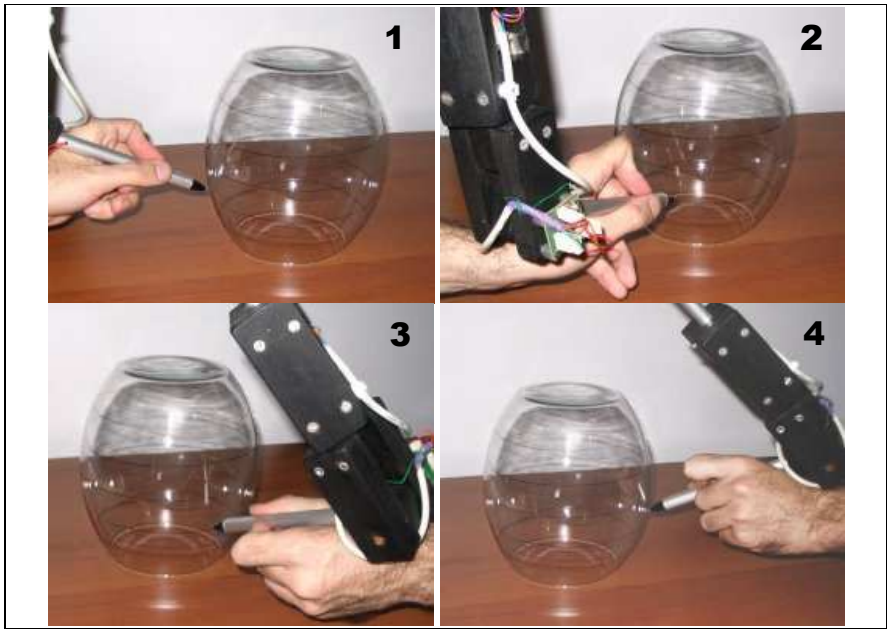


Figura 45 - Captura de pontos seguindo seqüência na camada.

Deve-se tomar cuidado para não deixar a ponta sensora encostada no objeto, para que não seja interpretado erradamente como o início de uma nova camada, o que iria prejudicar a reconstrução da superfície, durante a etapa de triangulação dos pontos.

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos pelo trabalho foram satisfatórios, mesmo contendo algumas restrições de acordo com o que havia sido proposto.

O *hardware* para captura de coordenadas permitiu uma boa mobilidade para a captura das mesmas, de modo que é possível alcançar um ponto que esteja até 1 metro distante da base. O manuseio e captura de coordenada é simples. Mas alguns detalhes como a utilização de madeira e processo artesanal para construção do *hardware*, comentado na seção 3.3.1, e a necessidade de um fino alinhamento entre o sensor e o magneto, citado na seção 2.3.1.1, geraram perda de precisão durante a captura dos pontos.

A Tabela 1 apresenta uma relação entre o valor real de uma coordenada e o valor capturado pelo braço digitalizador.

Tabela 1- Comparativo entre coordenadas reais e lidas

COMPARATIVO ENTRE COORDENADAS REAIS E COORDENADAS LIDAS											
Coordenada real (mm)			Coordenada lida (mm)			Erro (em módulo)			Erro percentual		
(150	150	0)	(157,55	-176,63	0,55)	(7,55	26,63	0,55)	(0,72	2,55	0,05)
(250	150	0)	(252,24	-159,97	4,13)	(2,24	9,97	4,13)	(0,21	0,95	0,4)
(350	150	0)	(357,70	-159,66	-7,54)	(7,7	9,66	7,54)	(0,74	0,92	0,72)
(450	150	0)	(459,62	-158,35	-1,39)	(9,62	8,35	1,39)	(0,92	0,8	0,13)
(150	0	0)	(165,10	-11,54	6,16)	(15,1	11,54	6,16)	(1,44	1,1	0,59)
(250	0	0)	(258,79	-9,81	9,22)	(8,79	9,81	9,22)	(0,84	0,94	0,88)
(350	0	0)	(363,6	-5,45	2,40)	(13,6	5,45	2,4)	(1,3	0,52	0,23)
(450	0	0)	(458,48	2,88	8,12)	(8,48	2,88	8,12)	(0,81	0,28	0,78)
(150	-150	0)	(167,24	143,28	10,64)	(17,24	6,72	10,64)	(1,65	0,64	1,02)
(250	-150	0)	(265,46	141,55	11,10)	(15,46	8,45	11,1)	(1,48	0,81	1,06)
(350	-150	0)	(357,53	140,03	13,13)	(7,53	9,97	13,13)	(0,72	0,95	1,26)
(450	-150	0)	(466,92	147,92	23,94)	(16,92	2,08	23,94)	(1,62	0,2	2,29)
Erro médio						**(10,85**	**9,29**	**8,19)**	**(1,03**	**0,89**	**0,78)**

Observa-se nessa amostragem de pontos que existe erro na captura dos pontos. Este erro traz em média uma variação de 1 centímetro nos eixos *x*, *y*, *z*. Estes resultados poderiam ser mais precisos caso as peças tivessem sido industrializadas, garantindo a exatidão das suas dimensões e articulações ideais. O erro percentual foi calculado considerando o raio de alcance do braço (1 metro), assim cada milímetro corresponde a aproximadamente 0,1% de erro.

A Figura 46 apresenta a foto de um vaso, e também os pontos capturados e a malha triangular correspondentes ao vaso.

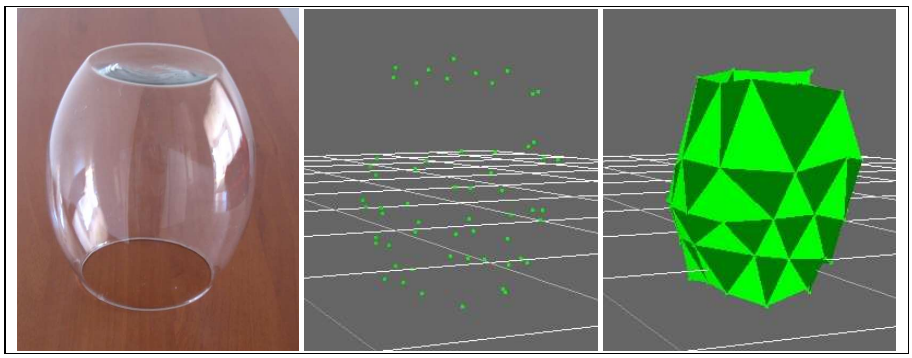


Figura 46 - Vaso reconstruído

A qualidade do objeto gerado depende precisão dos pontos capturados e, como visto na Tabela 1, existe um erro na captura dos pontos que não pode ser desconsiderado, e afeta as demais etapas do trabalho.

As triangulações citadas na revisão bibliográfica não devem ser tomadas como comparação pois normalmente trabalham com pontos capturados por *scanners* precisos que geram uma nuvem de pontos bastante densa. Na presente demonstração, coletaram-se apenas alguns pontos da superfície do objeto a ser reconstruído.

Encontrou-se um *software* comercial chamado 3D Reshaper (TECHNODIGIT, 2005) que conseguiu importar a nuvem de pontos e reconstruir o objeto, sem informações adicionais. A Figura 47 mostra a reconstrução realizada com a mesma nuvem de pontos

utilizada nos testes apresentados acima, sem a necessidade das divisões de camadas. Foi utilizada a opção de “reconstrução sem redução de ruídos”.

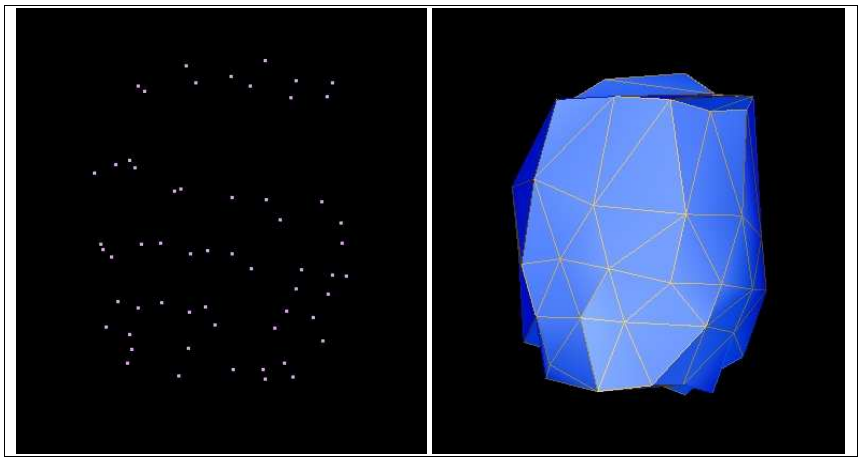


Figura 47 - Vaso reconstruído no 3D Reshaper

4 CONCLUSÕES

A construção de *hardware* e *software* de um digitalizador tridimensional foi concluída. Com o trabalho desenvolvido é possível capturar coordenadas da superfície de um objeto, visualizar os pontos capturados e também realizar a triangulação para visualizar a superfície formada pelos triângulos gerados na triangulação.

A captura de coordenadas apresentou um erro médio de 1%. Este erro ocorreu, devido à baixa resistência da madeira e à utilização de métricas manuais para realizar os furos perpendiculares à sua superfície. Outro fator foi a fixação e o alinhamento dos imãs e *encoders*, que necessitavam de muita precisão, dificilmente alcançada por um trabalho artesanal, onde qualquer milímetro fora do especificado no *datasheet* (AUSTRIAMICROSISTEMS, 2005) deixa margens para pequenos erros de posição que acabam interferindo no resultado final.

O processo de aquisição de dados, que envolve a leitura dos sensores e o cálculo das coordenadas, está funcionando perfeitamente, então a precisão das coordenadas capturadas depende exclusivamente da capacidade do *hardware*.

Todas as técnicas de reconstrução de superfícies a partir da nuvem de pontos citadas na seção 2.6 têm restrições quanto à qualidade dos pontos capturados. Normalmente são capturados com *scanners* a laser e capturam milhares de pontos em uma pequena área. Como no presente trabalho coletam-se poucos pontos, estas técnicas não puderam ser aplicadas, e para realizar a triangulação, foi criada uma técnica de reconstrução que define algumas restrições para a fase de captura dos pontos. Técnica esta, especificada no item 3.3.3.

Os resultados obtidos foram satisfatórios, porém ainda estão longe de resultados ideais. Com uma fonte financiadora para melhorias de *hardware*, e estudos para tornar a aplicação mais robusta, seria possível melhorar o projeto, tornando possível a sua utilização em projetos

que possuem a necessidade de realizar engenharia reversa de objetos, apresentando ao final uma captura de coordenada de melhor qualidade.

4.1 EXTENSÕES

A seguir são apresentadas sugestões para trabalhos futuros, que permitam a continuidade deste trabalho. São elas:

a) melhoria de *hardware* para tornar mais precisa a captura dos pontos;
b) alterar a parte de comunicação do *hardware* com o computador, fazendo com que toda a aplicação fique independente de plataforma. Uma sugestão seria desenvolver um circuito para realizar a comunicação através da porta USB;
c) desenvolver uma triangulação mais robusta, de modo que sejam eliminadas as restrições quanto à topologia do objeto a ser digitalizado e as restrições durante a captura dos pontos. Para desenvolver uma triangulação eliminando as restrições da topologia do objeto (ex: permitir digitalizar uma xícara com sua alça), sugere-se a divisão do objeto em vários componentes, que deveriam ser digitalizados e triangulados individualmente, e na hora da visualização tratá-los como um conjunto único.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, J. Alan; ROGERS, David F. **Mathematical elements for computer graphics**. 2nd ed. New York: McGraw-Hill, 1990. 611 p.

AUSTRIAMICROSYSTEMS. **AS5040 – 10 bit programmable magnetic rotary encoder-data sheet**. Schloss Premstätten/Áustria, fev. 2005. Disponível em: <http://www.austriamicrosystems.com/03products/feature_pdf/AS5040_DataSheet.pdf>. Acesso em: 30 maio 2005.

AMENTA Nina; CHOI Sunghee; KOLLURI Ravi Krishna. The power crust. In: ACM Symposium on Solid Modeling and Applications 01, 6., 2001, Ann Arbor, Michigan, United States. **Proceedings…** New York: ACM Press, 2001. p. 249–266.

BAJAJ Chandrajit L.; BERNARDINI Fausto; XU Guoliang. Automatic reconstruction of surfaces and scalar fields from 3d scans. In: SIGGRAPH 95, 22., 1995, Los Angeles. **Proceedings…** New York: ACM Press, 1995. p. 109-118.

BARDINETA Eric; COHENB Laurent D.; AYACHE Nicholas. A parametric deformable model to fit unstructured 3d data. **Computer Vision and Image Understanding**, New York, v. 71, n. 1, p. 39–54, jul. 1998.

BERNARDINI, Fausto, et al. The ball-pivoting algorithm for surface reconstruction. **IEEE Transactions on Visualization and Computer Graphics,** Piscataway, NJ, USA, v. 5, n. 4, p. 349–359, 1999.

BOISSONNAT, Jean-Daniel; CAZALS, Frédéric. Smooth surface reconstruction via natural neighbour interpolations of distance functions. In: ACM Symposium on Computational Geometry, 16., 2000, Clear Water Bay, Kowloon, Hong Kong. **Proceedings…** New York: ACM Press, 2000. p. 223–232.

CIOQUETA, Higor Rodrigues; NETO, Francisco Paulo Lépore. Idealização, Projeto e Construção de um Braço Digitalizador 3D. **Revista Eletrônica de Iniciação Cientifica – SBC**, Porto Alegre v. 3, n. 3, Setembro 2003. 8 f. Disponível em: <http://www.sbc.org.br/reic/edicoes/2003e3/cientificos/Idealizacao_Projeto_e_Construcao_de_um_Braco_Digitalizador_3D.pdf> Acesso em: 01 jun. 2005.

EDELSBRUNNER, Herbert; MÜCKE, Ernst. P. Three-dimensional alpha shapes. **ACM Transactions on Graphics**, New York, v. 13, n. 1, p. 43–72, 1994.

GOIS, João Paulo. **Reconstrução de Superfícies a partir de Nuvens de Pontos**. 2004. 131 f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

GOPI, M.; KRISHNAN, S.; SILVA, C. T.. Surface reconstruction based on lower dimensional localized delaunay triangulation. **Computer Graphics Forum - Eurographics 2000**, v. 19, n. 3, ago. 2000.

HOPPE, Hugues et al. Surface reconstruction from unorganized points. In: SIGGRAPH 92, 19., 1992, Chicago. **Proceedings...** New York: ACM Press, 1992. p. 71-78.

IMMERSION CORPORATION. **Microscribe**. San Jose, 2005. Disponível em: <http://www.immersion.com/digitizer/>. Acesso em: 01 jun. 2005.

LOGIX4U. **Inpout32.dll for WIN NT/2000/XP**. [S.l.], 2005. Disponível em: <http://www.logix4u.net/inpout32.htm>. Acesso em: 30 maio 2005.

MATIAS, Juliano. Encoders. **Mecatrônica atual**. São Paulo, ano 1, n. 3, p. 36-42, abr. 2002.

MATOS, Alexandre Veloso de. **UML**: prático e descomplicado. São Paulo: Érica, 2002. 187 p.

MESSIAS, Antonio Rogério. **Porta paralela**. [S.l.], 2005. Disponível em <http://www.rogercom.com/pparalela/introducao.htm>. Acesso em: 30 maio 2005.

MICROSOFT CORPORATION. **Visual C++**. [S.l.], 2005. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/visualcpp/Default.mspx>. Acesso em: 16 maio 2005.

MUSSER, David R; SAINI, Atul; DERGE, Gillmer J. **STL tutorial and reference guide:** c++ programming with the standard template library**.** 2nd ed. Reading, Mass: Addison-Wesley, 2001. xxxvi, 509 p, il. (Addison-Wesley professional computing series).

PAZOS, Fernando A. Robôs manipuladores – 1² parte. **Mecatrônica atual**. São Paulo, ano 1, n. 2, p. 20-27, fev. 2002.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO. Departamento de Ciências da Computação. Tecnologia em Computação Gráfica. **IUP – portable user interface**. Versão 2.3. Rio de Janeiro, abr. 2005a. Disponível em: <http://www.tecgraf.puc-rio.br/iup>. Acesso em: 16 maio 2005.

_____. Departamento de Ciências da Computação. Tecnologia em Computação Gráfica. **VGL documentation**. Desenvolvida por Frederico Rodrigues Abraham e Waldemar Celes. Rio de Janeiro, abr. 2005b. Disponível em: <http://www.tecgraf.puc-rio.br/~fabraham/vgl/>. Acesso em: 25 maio 2005.

SILVA, Emilio C. N. CAD/CAE/CAM. **Mecatrônica atual**. São Paulo, ano 1, n. 32, p. 38-47 out/nov 2001.

SPARX SYSTEMS PTY LTDA. **Enterprise Architect.** Versão 5.0. Austrália, 2005. Disponível em: < http://sparxsystems.com.au/products/ea.html>. Acesso em: 01 jun. 2005.

TECHNODIGIT. **3D Reshaper**. Versão 3.1. França, 2005. Disponível em: <http://www.technodigit.com/en1/En_software.htm>. Acesso em: 01 jun. 2005.

SWEET, Michael; WRIGHT, Richard S. **OpenGL superbible**. 2nd ed. Indianopolis: Waite Group Press, 2000. 696 p.

VÁRADY, Tamás; MARTIN, R. Ralph; COX, Jordan. Reverse engineering of geometric models - an introduction. **Computer-Aided Design**, [S.l.], v.29, n.4, 1997, pp 255-269.

ZHAO, Hong-Kai, et al. Implicit nonparametric shape reconstruction from unorganized points using a variational level set method. **Computer Vision and Image Understanding**, Amsterdam, v. 80, n. 3, p. 295–314, dez. 2000.

APÊNDICE A – Especificação das peças do hardware

Neste apêndice é feita a especificação das peças que compõe o braço digitalizador proposto. Na Figura 48 são apresentadas todas as peças montadas, identificando-as. Estas peças são detalhadas individualmente através das figuras 48 até a 66. As peças 2, 4, 8 e 9, possuem uma cavidade para acomodar o rolamento utilizado nas articulações.

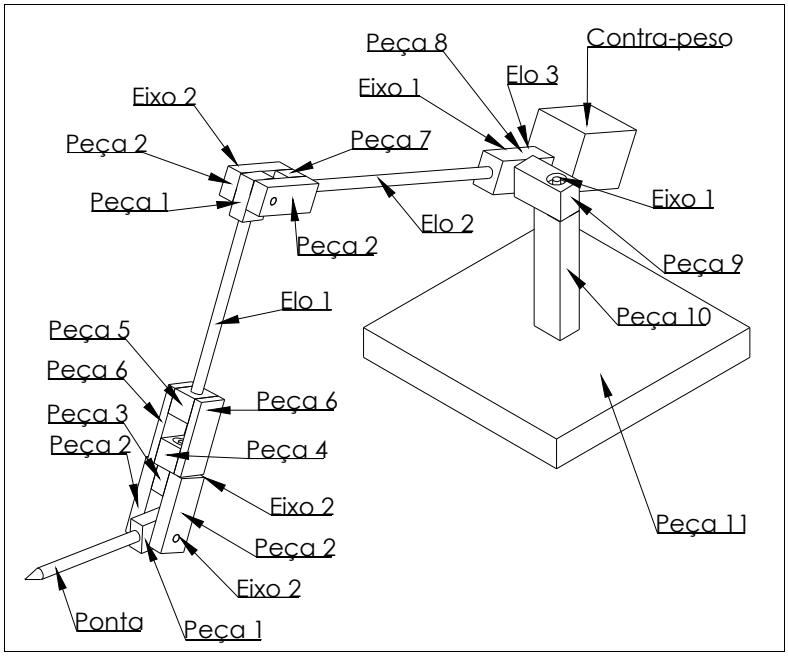


Figura 48 - Peças montadas

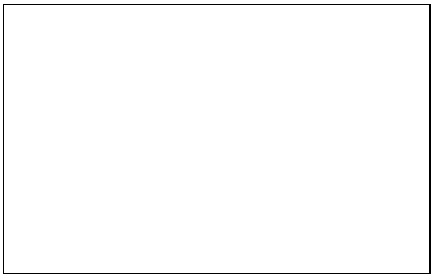


Figura 49 - Peça 1

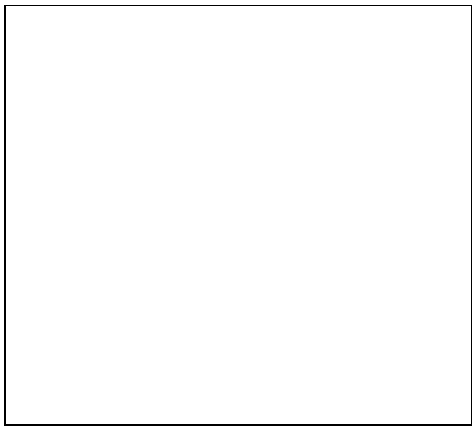


Figura 50 - Peça 2

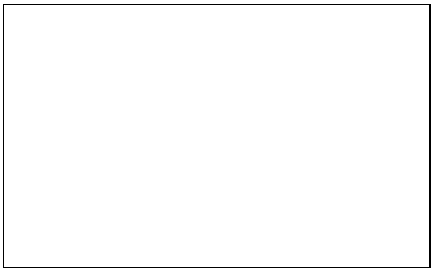


Figura 51 - Peça 3

Figura 52 - Peça 4

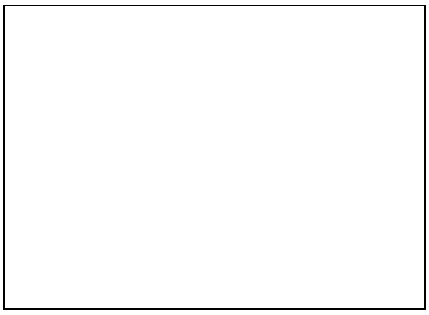


Figura 53 - Peça 5

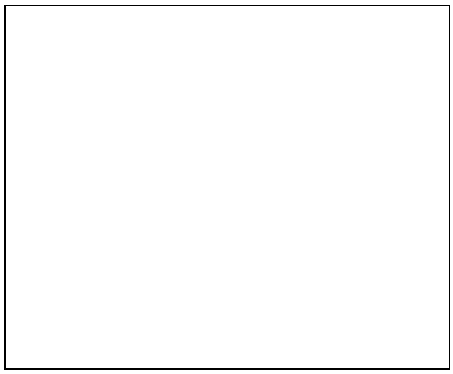


Figura 54 - Peça 6

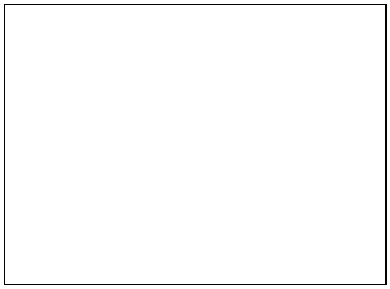


Figura 55 - Peça 7

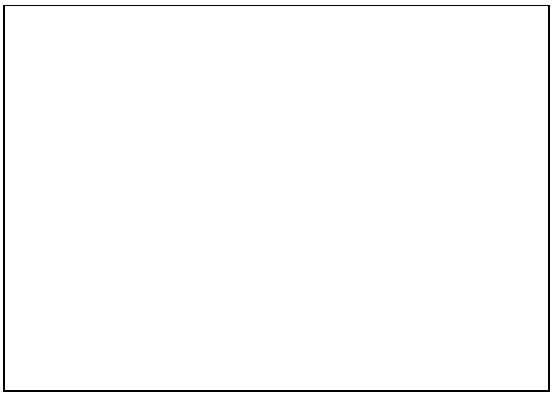


Figura 56 - Peça 8

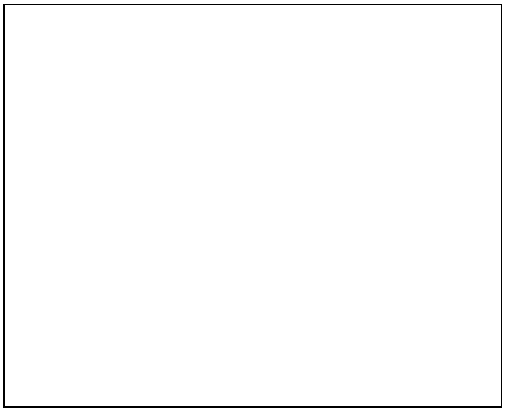


Figura 57 - Peça 9

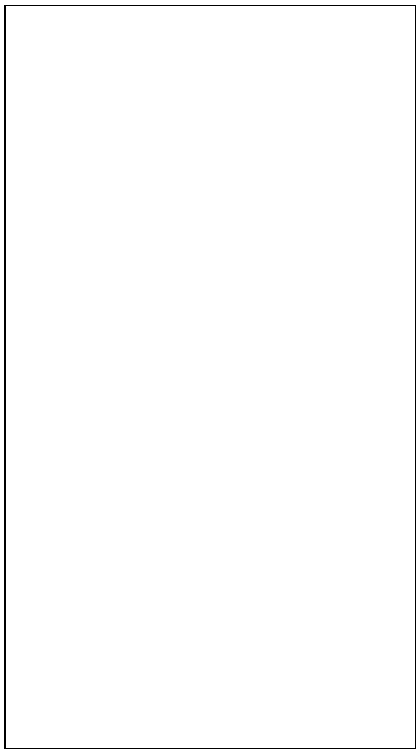


Figura 58 - Peça 10

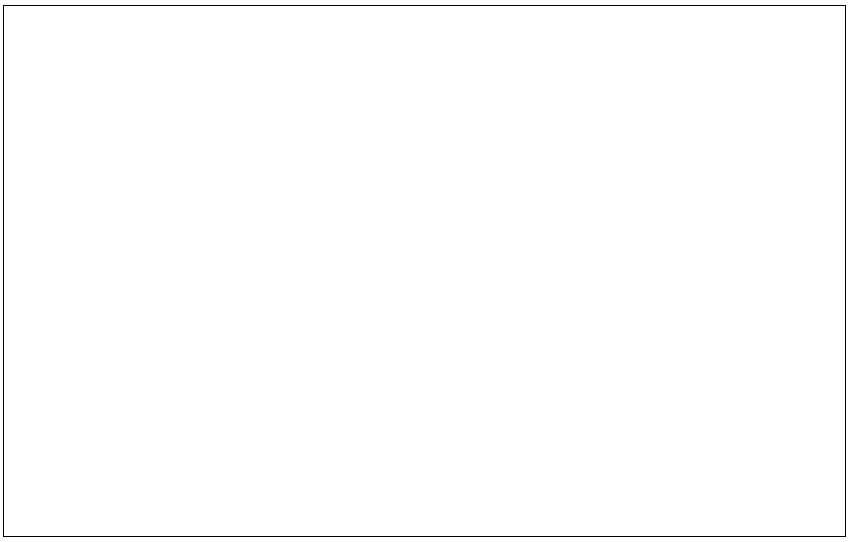


Figura 59 - Peça 11

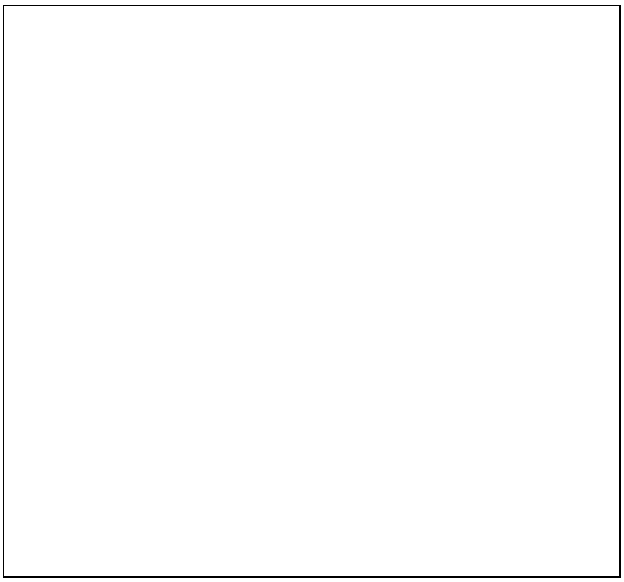


Figura 60 - Contra-peso

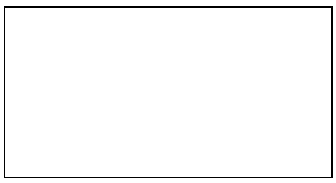


Figura 61 - Eixo 1

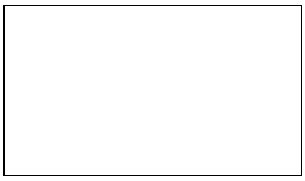


Figura 62 - Eixo 2

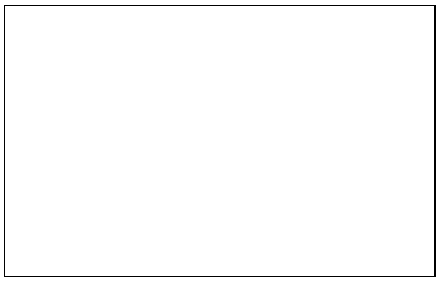


Figura 63 - Ponta

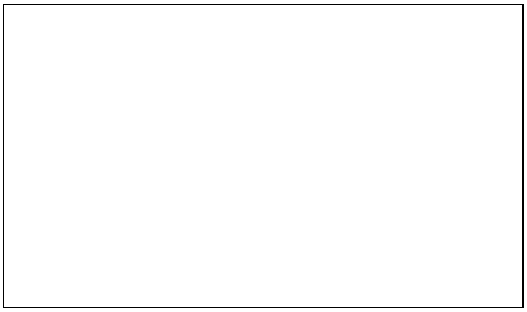


Figura 64 - Elo 1

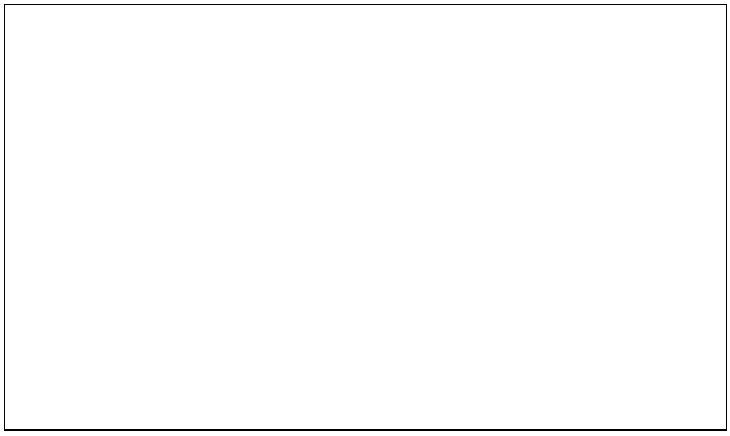


Figura 65 - Elo 2

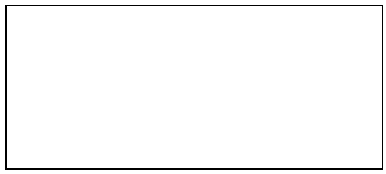


Figura 66 - Elo 3