9,6 (nove e seis)

EDGAR TADAO TAKANO
RENEU LUIZ ANDRIOLI JR.

ESTUDO E IMPLEMENTAÇÃO DO SISTEMA DE CONTROLE DO ACIONAMENTO DE UM ROBÔ CIRÚRGICO

Trabalho de Formatura apresentada à Escola Politécnica da Universidade de São Paulo.

São Paulo
2001

EDGAR TADAO TAKANO
RENEU LUIZ ANDRIOLI JR.

ESTUDO E IMPLEMENTAÇÃO DO SISTEMA DE CONTROLE DO ACIONAMENTO DE UM ROBÔ CIRÚRGICO

Trabalho de Formatura apresentada à Escola Politécnica da Universidade de São Paulo.

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Lucas Moscato

São Paulo
2001

Aos nossos pais.

Agradecemos a todos os amigos que se interessaram
e colaboraram de alguma forma para a realização
deste trabalho.

Sumário

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 Histórico e motivação

A idéia de se construir robôs para aplicações cirúrgicas se difundiu bastante na última década. O interesse pelo desenvolvimento desta tecnologia não é demonstrado somente pelos cientistas, mas também pela população em geral, visto que mesmo essa classe não-cientista é capaz de entender a revolução no tratamento da saúde que esta nova tecnologia pode causar. Para que esta tecnologia se desenvolva é necessário à cooperação entre os profissionais das ciências médicas e de engenharia, o que ainda está no início.

O mundo dos robôs cirúrgicos é ainda pequeno. Atualmente existem apenas seis fornecedores no mundo, sendo que o maior destes não vendeu mais do que 100 unidades. O uso dos robôs segue a demanda médica, e isto limita a velocidade de desenvolvimento. Além disso, não há padrões para o desenvolvimento, instalação, uso e adaptação destes ao OR. A aprovação que também seria necessária pela FDA do EUA e pela MDD da Europa de tais dispositivos é um risco para o investimento das companhias.

Apesar destas dificuldades, existe a certeza que cedo ou tarde, os robôs tornar-se-ão uma ferramenta de suporte no OR. Com exceção de poucos casos, os robôs não serão máquinas completamente autônomas, que realizarão a operação cirúrgica total. Serão instrumentos de suporte a cirurgiões de modo que estes obtenham melhores resultados.

Em 1990, foi realizada uma das primeiras cirurgias - substituição do osso da bacia de um cachorro. Ainda no final deste mesmo ano, foi realizada uma cirurgia cardíaca em seres humanos. E já se prevê robôs que realizem cirurgia nos olhos. Estas cirurgias têm ocorrido na Europa e em breve serão aprovadas no EUA.

Segundo uma previsão do Japan Robot Assn, o mercado de robôs relacionados ao ramo médico e de saúde em 2010 será bastante expressivo, atrás apenas dos robôs de aplicação industrial, com vendas projetadas na ordem de um bilhão de dólares.

Aplicação	Vendas em 2005	Vendas em 2010
Transporte e almoxarifado	450	860(milhões de dólares)
Agricultura, pescaria	390	800
Medica, saúde	250	1050
Utilitários, plantas de potencia	190	320
Coleta de Lixo	130	350
Varejo, Distribuição	125	300
Indústrias	7030	8000

Tabela 1 - Data: Japan Robot Assn.

São muitos os argumentos que justificam o uso de robôs como instrumentos inteligentes, mesmo se hoje a tecnologia ainda não é capaz de satisfazer todos estes:

- Os robôs mostram sempre a mesma potência, eles executam suas tarefas de maneira regular;
- Os comportamentos de um robô durante a execução de uma determinada tarefa podem ser conhecidos exatamente, antes mesmo desta ser realizada, sendo que podem ser gerados uma documentação, avaliação e acompanhamento de tarefas realizadas;
- Os robôs balançam e vibram muito menos do que um ser humano;
- Os robôs podem movimentar-se de modo muito mais preciso em relação ao espaço e tempo do que um ser humano;
- Podem ser controlados remotamente;
- Os Robôs e sistemas de controle podem coordenar processos ou eventos complexos, e estão aptos a cooperar com outras máquinas no OR em milisegundos.

Porém ainda hoje, não há nenhum robô para ser comprado por um neurocirurgião que possa ajudá-lo a realizar *drilling / shaping* na base do crânio, mesmo que houvesse dinheiro e interesse por este equipamento. Os poucos robôs

disponíveis no mercado são limitados a tarefas muito dedicadas, apesar deste grande potencial para pesquisa e projeto de novos robôs, além dos já existentes.

As maiores dificuldades encontradas nas operações cirúrgicas assistidas por robôs são as seguintes:

- Estabilidade mecânica da OR table;
- Fixação, registro e acompanhamento do paciente;
- Aquisição de imagens intraoperativas;
- Posicionamento dos robôs sem a obstrução dos médicos;
- Interfaces de software de fácil uso;
- Segurança;
- Integração do robô com a infraestrutura do OR;
- Adaptação de instrumentos cirúrgicos considerando o work flow de diferentes intervenções.

A acuracidade dos robôs não tem sido um grave problema, comparativamente com a acuracidade que pode ser atingida manualmente. Por outro lado, há alguns problemas como a vibração do OR quando um helicóptero decola de uma região próxima. O robô deve então interromper a cirurgia momentaneamente, assim como um cirurgião faria.

Desta forma o interesse o desenvolvimento de novos robôs capazes de solucionar os problemas citados acima e de atender aos requisitos de mercado mostra-se uma tarefa desafiadora e interessante, visto que há muito a ser desenvolvido ainda e também um potencial muito grande de utilização e exploração dessa nova tecnologia, que pode trazer enormes benefícios para a população em geral, no que se refere aos cuidados da saúde.

1.2 O robô cirúrgico em desenvolvimento no presente projeto

O presente projeto tem por objetivo desenvolver o sistema de controle dos motores que realizam o acionamento de um robô para aplicação em cirurgias cardíacas de invasão mínima. Este robô já está sendo desenvolvido por uma equipe de pesquisadores do departamento, sendo que sua concepção geométrica já está definida. Antes, porém, de descrevermos o modelo do robô cirúrgico em desenvolvimento, vamos apresentar a classificação, de acordo com o ponto de vista da utilização, dos robôs já existentes e mesmo dos que ainda estão por vir:

Robôs Teleoperados para cirurgias de invasão mínima: São controlados remotamente pelo cirurgião no OR através do uso de voz, joysticks com feedback da força, ou mesmo através de dispositivos hápticos mais avançados chamados de Master Manipulator. Estes robôs não funcionam autonomamente. Tipicamente o robô é usado somente para movimentação da câmera no interior de um endoscópio (300 máquinas existentes no mundo). Poucas máquinas são utilizadas para movimentar instrumentos, apenas 10 no mundo todo.

Sistemas robóticos completamente autônomo: São utilizados somente para drilling/shaping os buracos no fêmur durante a substituição da bacia. Após a fixação da perna do paciente, a posição da perna é registrada, sendo que a posição e orientação são realizadas por um sistema de navegação. Então o robô realiza furos por 20 a 40 minutos automaticamente. Se o osso se move ou se o robô é tocado, o robô interrompe a operação. Na verdade, o robô realiza uma seqüência de movimentos programados. Atualmente, foram vendidos menos de 40 robôs para esta aplicação, sendo que um robô com um time de assistentes é capaz de realizar mais que 600 operações em um ano.

Robôs de pilotagem interativa: São sistemas de suporte de ferramentas que carregam, guiam e movem instrumentos cirúrgicos. Tipicamente eles são pilotados e usam um método de registro similar aos robôs automáticos. No entanto, o robô interativo não precisa ser programado, não trabalha automaticamente, mas podem ser ensinados pelos cirurgiões durante a operação. O robô é movido explicitamente pelo cirurgião, mas pode limitar os graus de liberdade dos movimentos. No mundo,

existem cerca de 100 sistemas em uso, sendo que estes trabalham como carregadores de microscópio e em neurocirurgias.

Micro máquinas: podem ser pequenos robôs dedicados para tarefas especiais tal como costura de vasos ou para biopsia automatizada. Eles podem ser utilizados como ferramentas para outros robôs, mas na verdade são mais instrumentos do que robôs propriamente ditos.

Com base nessa classificação, verificamos que o robô em questão está situado na classe dos sistemas robóticos teleoperados para cirurgias de invasão mínima. Ele será composto por três braços mecânicos, sendo um braço principal, que possuirá em sua extremidade um punho (fórceps), com alguns graus de liberdade em relação ao braço, que manipulará os instrumentos cirúrgicos, substituindo a mão do cirurgião. Os outros dois braços serão utilizados para transportar o endoscópio e para inserir gás no paciente (necessário para se obter espaço suficiente para a operação do braço principal).

Neste trabalho será controlado o acionamento do braço principal, desta forma apresentamos abaixo o modelo geométrico do mesmo construído em um software de CAD.

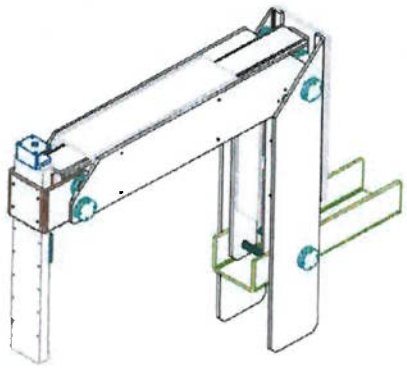


Figura 1 - Modelo geométrico do braço principal.

Através deste modelo foi realizada uma análise de interferência entre as partes móveis do robô em questão. Assim o modelo acima apresentado já se encontra livre de interferências. Através deste modelo foi possível também à obtenção de parâmetros cinemáticos e dinâmicos do braço, tais como: centro de massa, distâncias entre eixos de rotação, momentos e produtos de inércia. Estes parâmetros serão

utilizados em etapa posterior para determinação das especificações de controle, ou seja, dos requisitos necessários de torque e velocidade que os motores deverão proporcionar aos eixos.

O braço mostrado acima será acionado por três motores. Cada um desses motores acionará um dos graus de liberdade do robô, mostrados na figura abaixo.

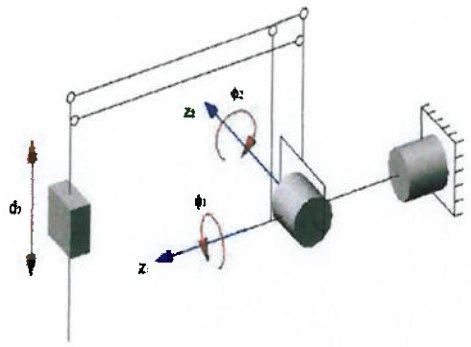


Figura 2 - Eixos de rotação e variáveis de junta do robô.

Além destes três graus de liberdade, existe ainda um fórceps preso ao punho mostrado na figura acima. Este fórceps terá dois graus de liberdade, rotação e abertura e fechamento da garra, constituindo, desta forma, em mais dois graus de liberdade e conseqüentemente em mais dois atuadores.

1.3 Objetivos

Como já citado no item anterior o desenvolvimento completo do robô cirúrgico está sendo realizado por uma equipe de pesquisadores do departamento. Várias etapas da espiral de projeto já foram realizadas e, desta forma, serão utilizadas no presente trabalho. Como exemplo pode-se citar o modelo geométrico do braço do robô apresentado anteriormente.

O projeto completo do robô apresenta várias atividades a serem desenvolvidas, como o controle de acionamento dos motores, o projeto do endoscópio, de um mouse para que o cirurgião seja capaz de movimentar o braço, entre outros. Percebe-se que o projeto completo do robô é muito abrangente, sendo necessário explicitar aqui os objetivos a serem realizados pelo presente trabalho.

Assim, o desenvolvimento do presente projeto tem por objetivo desenvolver e implementar o sistema de controle dos motores que realizarão os acionamentos dos

movimentos do braço do robô cirúrgico descrito no item anterior. Este sistema deverá ser capaz de controlar a posição e a rotação desses motores. O controle será realizado com o auxílio de um microcomputador através de um software de controle que será desenvolvido e implementado como objetivo final deste trabalho.

2 REVISÃO DA LITERATURA

Para a obtenção dos objetivos propostos citados anteriormente, será preciso proceder à escolha adequada dos atuadores e sensores necessários para o acionamento e detecção do controle a ser efetuado, desta forma realizou-se uma pesquisa bibliográfica sobre os diversos tipos de dispositivos de acionamento (motores) e de detecção (sensores). Dentro deste escopo será dada uma breve introdução sobre os seguintes tipos de motores e sensores:

- Motores DC;
- Motor elétrico de passo;
- Motores pneumáticos;
- Motores hidráulicos;
- Encoders;
- Potenciômetros;
- Varicap.

2.1 Motores DC

Para o entendimento de um motor DC, é necessário o entendimento de alguns termos básicos:

- Rotor: é a parte do motor que rotaciona;
- Estator: é a parte fixa do motor;
- Sistema de campo (Field System): é parte do motor a qual fornece um fluxo magnético necessário para a criação do torque. Pode por exemplo ser criado por imãs permanentes e uma estrutura de ferro, formando a parte do estator.
- Armadura: é à parte do motor que carrega a corrente e interage com o fluxo magnético para a criação do torque.

- Escovas: é a parte do circuito através da qual a corrente elétrica é fornecida da fonte de energia até a armadura do motor. As escovas são feitas de grafite ou metais preciosos. Um motor DC tem um ou mais pares de escovas.
- Comutador: é a parte que entra em contato com as escovas.

O princípio de geração de torque no motor consiste no surgimento de uma força em um condutor pelo qual passa uma corrente elétrica, estando este imerso em um campo magnético. Abaixo mostramos um esquema em corte de um motor DC com escovas:

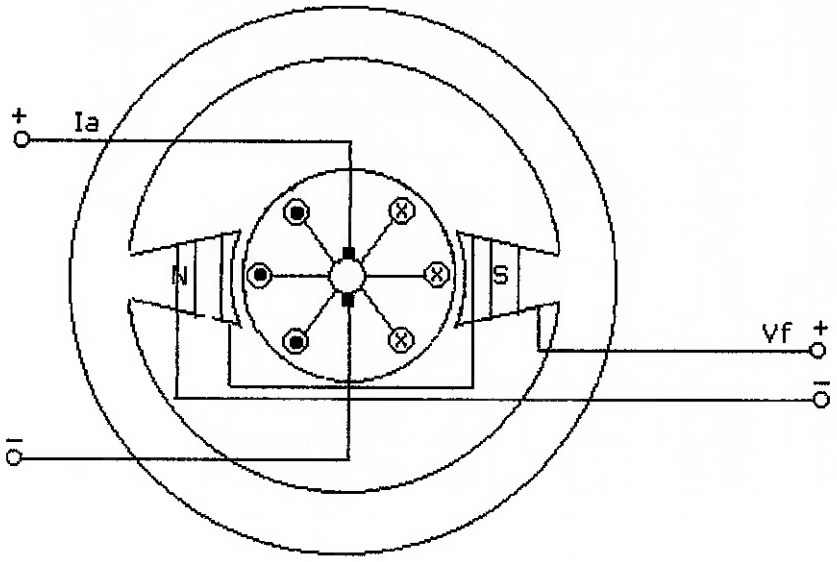


Figura 3 - Esquema em corte de um motor DC com escovas

Motores DC são largamente empregados em plantas onde o controle de velocidade é desejado. Em muitas aplicações onde uma velocidade constante é necessária, a operação em malha aberta pode não ser satisfatória. Em operações em malha aberta, se o torque resistivo diminui, a velocidade de rotação do motor irá mudar também. Porém sabe-se que a velocidade de rotação do rotor é proporcional a voltagem aplicada na armadura do motor, e que o torque desenvolvido pelo mesmo será proporcional a corrente elétrica. Deste modo, em malha fechada, a velocidade pode ser mantida constante simplesmente por ajustar-se a voltagem no terminal do motor.

O diagrama de blocos básico de um sistema de controle de velocidade pode ser observado na figura 4 Se um torque resistivo for aplicado, a velocidade do motor diminui momentaneamente e o erro da velocidade aumenta, o que por sua vez causa um aumento da voltagem do sinal de controle. Este aumento de voltagem transforma-se em torque e restituirá a velocidade do motor.

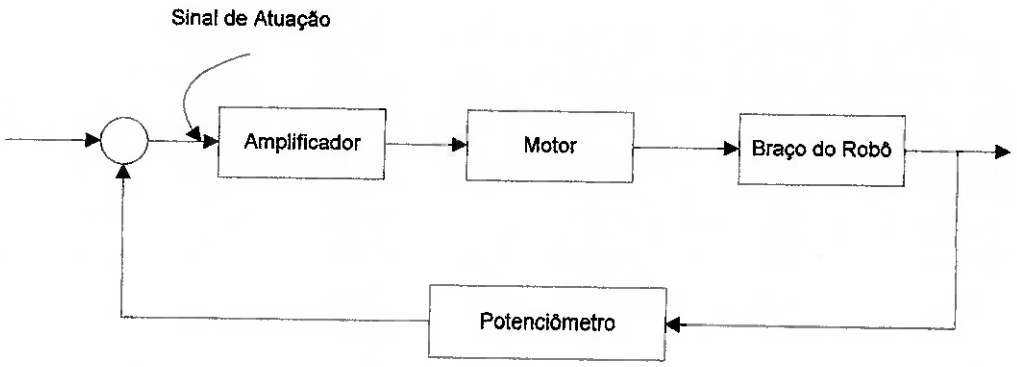


Figura 4 - Diagrama de blocos de um sistema de controle de velocidade.

Como outras vantagens da operação em malha fechada é possível obter maior acuracidade, respostas dinâmicas otimizadas e estabilidade na operação. Também é possível, ao invés de controlar a velocidade, manter o torque ou a potência constante.

2.2 Servomotores

Os servomotores, também chamados de "control motors", são motores elétricos desenvolvidos e construídos especialmente para uso em sistemas de controle em malha fechada, com o papel de atuador. Eles possuem uma potência que pode variar de uma pequena fração de watts até algumas centenas. Eles possuem alta velocidade de resposta, já que o rotor possui baixa inércia. Fisicamente, eles costumam ser de pequeno diâmetro e grande comprimento.

Pode-se encontrar tanto servomotores do tipo DC ou AC. O princípio de funcionamento do servomotor DC é o mesmo de um motor DC comum (citado acima).

Como exemplo de aplicação, motores pequenos são utilizados em relógios, máquinas de escritório, equipamentos de projeção, equipamentos industriais,

ventiladores, brinquedos, etc. Motores de maior precisão são utilizados em sistemas de controle como servomecanismos, e nas demais aplicações onde é requerida uma grande precisão angular.

2.3 Brushless Motor

Em motores DC, como há um contato de deslize entre as escovas e o comutador, ocorre a geração de faíscas e ambos componentes se desgastam. As escovas devem ser substituídas e o comutador deve receber certos cuidados superficiais. Outra desvantagem é que estas faíscas também provocam aquecimento e o torque acaba sendo produzido com algumas irregularidades. Muitas vezes, os motores DC são utilizados de um modo que sua vida seja menor que o desgaste das escovas.

No entanto, em algumas aplicações como em máquinas dispendiosas, o desgaste das escovas é um problema sério que só pode ser resolvido eliminando-as, ou seja, construindo um motor que não possua escovas.

O circuito eletrônico usado substitui as escovas e o comutador. Cada contato mecânico deve ser substituído por um contato eletrônico. Um transistor é o dispositivo eletrônico mais simples que pode ser utilizado como uma chave.

A estrutura de um motor brushless é mais parecida com a de um motor AC síncrono. Em outras palavras, um motor brushless DC é um motor de imã permanente com mecanismos transistorizados que fornecem as características do motor DC e são livres de manutenção.

As bobinas da armadura são partes constituintes do estator, e o rotor é composto por um ou mais imãs. Estas bobinas são muito semelhantes àquelas de motores AC polifásico, e os melhores e mais eficientes motores são aqueles com bobinas trifásicas e operadas com excitação bipolar. Uma grande diferença destes motores para os motores AC é que os Brushless incorporam formas de detectar a posição do rotor (dos pólos magnéticos) para a produção de sinais para o controle das chaves eletrônicas. O sensor mais utilizado para isso é o de efeito Hall, mas também alguns motores utilizam sensores ópticos.

Para a reversão do sentido de rotação em um motor DC convencional, basta que se inverta as tensões nos terminais do motor. Já no caso de um motor Brushless

isto não produziria o efeito desejado, uma vez que este utiliza elementos semicondutores. A inversão do sentido de rotação é realizada através de uma implementação eletrônica, através de uma porta lógica, que muda o controle dos transistores que energizam as bobinas.

Apesar de ser sempre dito que os motores Brushless DC e motores DC convencionais possuem características estáticas similares, eles na verdade apresentam diferenças consideráveis em alguns aspectos. A tabela a seguir apresenta as vantagens e desvantagens de cada um destes motores, de modo a dar um maior entendimento da aplicabilidade de cada um deles.

	Motores DC convencionais	Brushless Motors
Estrutura mecânica	Campo magnético no Estator	Campo magnético no Rotor
Características Distintivas	Resposta rápida e ótima controlabilidade	Durável, sem manutenção
Conexões das Bobinas	Ring Connection Delta Connection	delta, estrela, ou bifásico
Método de Comutação	Contato mecânico entre as escovas e o comutador	Transistores
Método de Detecção da posição do rotor	Detecção automática pelas escovas	Efeito Hall, encoder óptico
Método de Reversão	Reversão do terminal de voltagem	Rearranjando a seqüência lógica

Tabela 2 - Comparação entre Motores DC e Brushless

2.4 Motores de Passo

Os motores de passo são atuadores alimentados com corrente DC e com um comportamento previsível a ponto de permitir posicionamentos precisos, mesmo sem o uso de um sistema de controle em malha fechada.

O princípio de funcionamento de um motor de passo pode ser comparado com a de um motor síncrono; um campo magnético girante puxa um rotor magnético.

O funcionamento do motor de passo pode ser melhor entendido com o auxílio da figura abaixo.

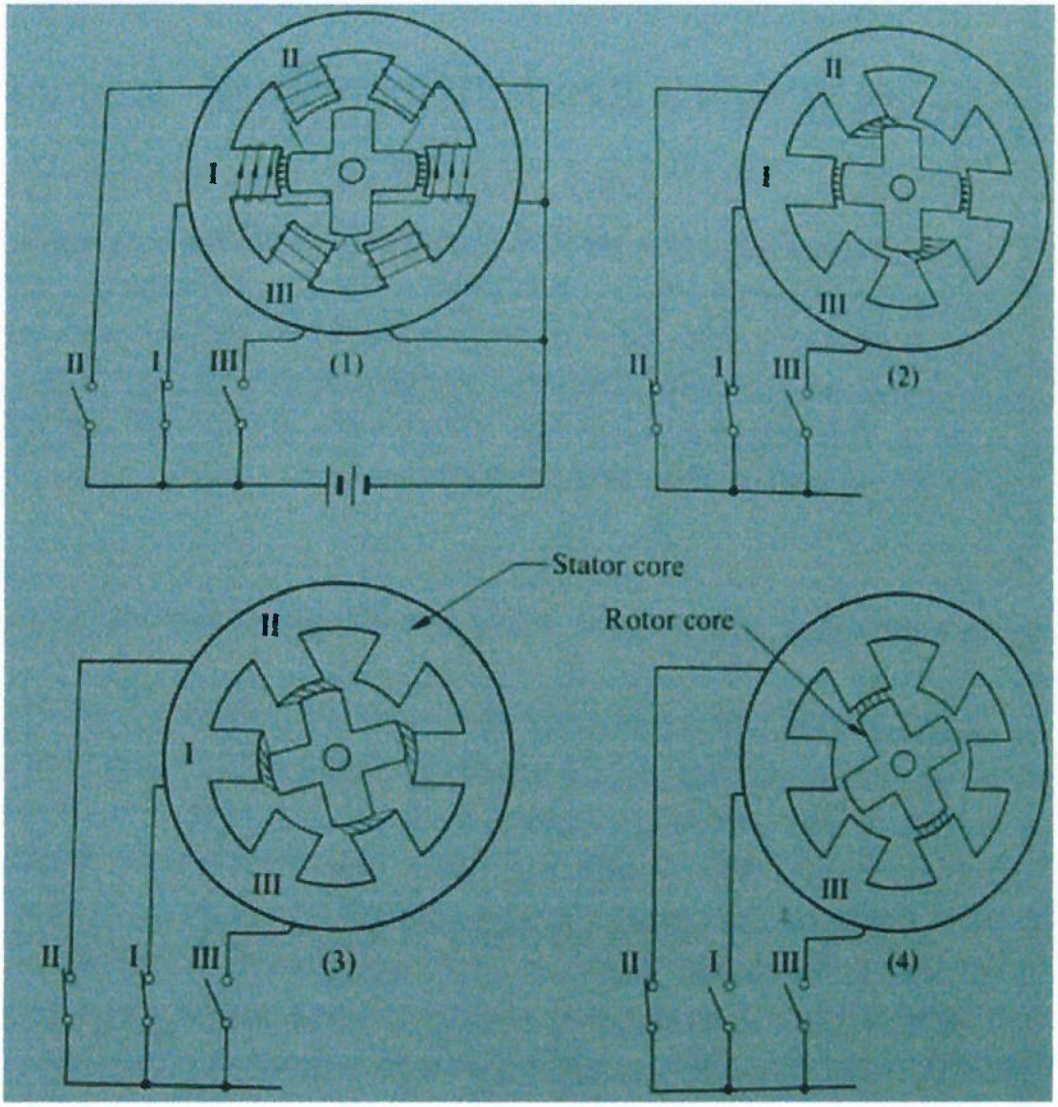


Figura 5 - Esquema de funcionamento do motor de passo.

O estator possui seis pólos ou dentes, enquanto o rotor possui quatro pólos. Três conjuntos de bobinas são dispostos como mostrado na figura. Um par de bobinas é chamada de fase, existindo neste caso, três fases. Uma corrente é fornecida as bobinas através das chaves I, II e III. No estado (1), os dois pólos do estator geram um campo magnético que provoca o alinhamento com dois dos quatro dentes do rotor. Quando a chave II é fechada a fim de excitar a fase II, o novo fluxo magnético

causa um torque no sentido anti-horário como pode ser observado em (2), e o rotor conseqüentemente atingirá o estado (3).

Os motores de passo costumam ser classificados de acordo com o material pelo qual o rotor é construído: material magnético permanente ou ferro de relutância variável.

Motor VR: o motor de passo de relutância variável pode ser considerado o motor de passo mais comum existente. Tanto os dentes do estator como o do rotor devem possuir alta permeabilidade magnética de modo a permitirem a passagem de alto fluxo magnético através dos mesmos. Geralmente são constituídos por *soft-iron.* A medida em que as bobinas são energizadas em um sentido, o rotor gira no sentido oposto, de modo a permitir que o fluxo magnético circule com mínima relutância magnética. O princípio de funcionamento em si é aquele apresentado anteriormente.

Imã permanente: são os motores que utilizam um imã permanente como rotor. O rotor, geralmente na forma cilíndrica, não possui dentes, apenas o estator. Este rotor pode ser imaginado como um simples imã, com um pólo norte e um pólo sul. Conforme as bobinas do estator são energizadas, os pólos do rotor são atraídos pela bobina, rotacionando no mesmo sentido do acionamento dos pólos do estator. Comparativamente, os motores de imã permanente possuem um torque maior em relação a um motor de relutância variável de mesmo tamanho.

Híbrido: O motor híbrido também possui o rotor constituído de imã permanente. O nome híbrido deriva do fato de que o motor é operado a partir de uma combinação dos dois tipos de princípios dos motores já citados, de modo que o resultado obtido é um motor com passo de menor ângulo e um torque maior com um motor menor.

Uma das vantagens de um motor de passo sobre um motor DC convencional é o fato de ser possível realizar o controle do mesmo sem o uso de sistemas de controle dispendiosos. No entanto, este método não é livre de limitações. Por exemplo, se o movimento do rotor tornar-se oscilatório e instável em certas faixas de velocidades, a velocidade e a aceleração de um motor de passo em malha aberta pode não ser tão rápido quanto o controle de um motor DC realimentado. O motor de passo em malha aberta pode também falhar na resposta a um pulso enviado quando a

freqüência dos pulsos enviados for muito alta ou quando o carregamento inercial for muito pesado.

A performance de um motor de passo pode ser muito melhorada ao se empregar realimentação de posição ou mesmo velocidade, de modo a determinar qual fase deve ser acionada e nos instantes corretos. Obtém se um movimento mais rápido e mais suave. A tabela abaixo mostra uma comparação entre motores de passo e servomotores:

Motor de Passo	Servo Motor DC
Controle é relativamente complicado	Controle é simples
Feedback não é necessário	Feedback é essencial
Baixa relação potencia/volume	Boa relação potencia/volume
Robusto, pouco desgaste	Desgaste devido às escovas
Boa característica de parada	Parada requer freio extra

Tabela 3 - Comparação entre Motor de Passo e Servo Motor DC

2.5 Motores Pneumáticos

Os atuadores pneumáticos convertem a energia de pressão em trabalho mecânico (pressão e vazão em força e velocidade ou em torque e rotação). O fluído de trabalho da pneumática é o ar comprimido. Existem várias formas construtivas para os atuadores pneumáticos, que vão desde um simples pistão (atuador linear) a motores rotativos, como por exemplo, motores de palhetas e motores de pistões radiais. A descrição do funcionamento interno destes tipos de motores varia muito para cada forma construtiva e não será descrito aqui por não estar no escopo deste projeto.

Porém, deve-se ressaltar aqui que um dos principais problemas dos motores pneumáticos é a dificuldade em se realizar um controle preciso do mesmo devido aos efeitos da compressibilidade do ar, que torna este controle muito complicado e oneroso.

2.6 Motores Hidráulicos

Assim como os motores pneumáticos convertem a energia de pressão e vazão em energia mecânica, os motores hidráulicos realizam a mesma transformação, mudando-se apenas o fluído de trabalho que ao invés de ar comprimido passa a ser um fluído hidráulico (geralmente óleo).

Existem também diversas formas construtivas e princípios de funcionamento para os motores hidráulicos, dentre os quais podemos destacar: motores de pistões radiais, motores de pistões axiais, motores de engrenagem e motores hidráulicos baseado no principio da roda planetária e eixo central. Da mesma forma que para motores pneumáticos, cada forma construtiva possui um principio de funcionamento e sua descrição foge do escopo do presente trabalho.

2.7 Sensores

O sistema de controle que se pretende desenvolver prevê a utilização de sensores de modo a realimentar a posição angular dos atuadores. Dentre as classificações que podem ser dadas aos sensores, cabe aqui dividi-los quanto à entrada e quanto à saída.Quanto à entrada, os sensores absolutos são aqueles que, dado uma origem fixa, o sinal de saída do sensor representa todos os possíveis valores do sinal físico de entrada sem ambigüidade. Já os sensores incrementais, quando uma origem não pode ser fixada para todos os pontos, ou seja, cada novo ponto adquirido, possui como origem o ponto anterior. E a classificação quanto à saída, estes podem ser analógicos, ou seja, o sinal de saída é contínuo e proporcional ao sinal de entrada, ou digitais, quando a saída pode assumir apenas um número de valores discretos.

2.7.1 Potenciômetros

O potenciômetro é um elemento comumente empregado na eletrônica no ajuste de valores de resistência. Compõem-se basicamente de um resistor, conforme mostra a figura 6 abaixo:

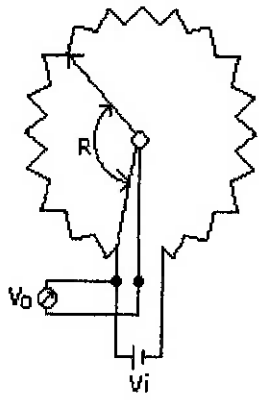


Figura 6 - Sensor angular do tipo potenciômetro.

Conforme a posição do contactor, a resistência entre a extremidade do resistor e o contactor varia. Alguns potenciômetros conhecidos como potenciômetros lineares são tais que a variação de resistência é proporcional ao deslocamento do contactor. Tais potenciômetros permitem dessa forma medir o deslocamento do contactor (no caso a posição angular do contactor) na forma de variação de uma resistência elétrica. Na prática aplica-se uma tensão fixa nas extremidades do resistor e mede-se a tensão no contactor.

Um potenciômetro pode ser representado pelo seguinte diagrama:

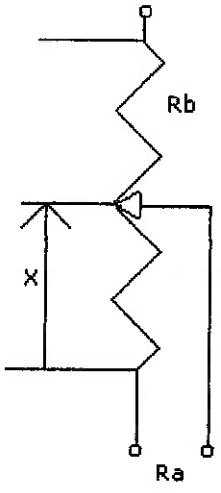


Figura 7 - Símbolo representativo de um potenciômetro.

Os potenciômetros possuem baixo custo, pequeno tamanho e versatilidade de operação, podendo, por exemplo, gerar facilmente funções de saída logarítmicas ou quadráticas.

2.7.2 Encoders ópticos

Os encoders são sensores ópticos, podendo ser incrementais ou absolutos. Ambos transformam uma entrada mecânica, a posição angular da sua haste, numa saída elétrica, por absorção luminosa. A absorção luminosa ocorre de acordo com a posição de um disco que rotaciona, permitindo ou não a passagem de luz para os fotorreceptores.

Os encoders incrementais usam simplesmente uma alternância de linhas opacas e transparentes, que permitem ou não a passagem de luz. Já os encoders absolutos, estes são necessários quando o sistema de controle é sujeito a períodos freqüentes desligado, e, além disso, reposicionamentos constantes precisam ser evitados. Estes possuem o disco de forma que em cada uma das posições que podem ser assumidas, o disco gera uma combinação binária diferente, que determina, portanto, a posição do sistema de forma unívoca.

Os encoders como pontos de vantagem, apresentam uma alta precisão, sem desgaste mecânico. Por outro lado, apresentam uma resolução angular pobre, sendo que o encoder absoluto com uma resolução não muito boa, possui diâmetro grande e uma precisão maior.

2.7.3 Capacitor Variável

O Capacitor variável consiste basicamente de dois eletrodos fixos e uma placa móvel feito de material isolante conectada a um eixo. Fixando-se o eixo, a placa móvel altera a área em oposição dos eletrodos fixos, variando a capacitância do conjunto. A variação de capacitância pode ser medida através do circuito descrito na figura 2.5, que dá uma tensão que é proporcional à capacitância C_x do conjunto. Permitem uma resposta com linearidade melhor que ± 2 % numa faixa de 140°.

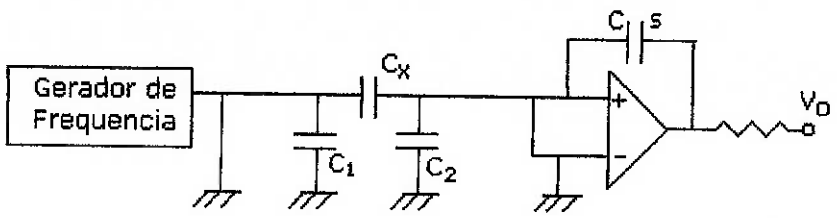


Figura 8 - Circuito do sensor angular do tipo capacitância variável.

2.7.4 Syncro-resolver

É composto por um rotor com uma bobina e dois pares de bobinas estacionárias arranjadas em direções ortogonais. A bobina do rotor é excitada através de anéis contactores (*slip rings*) com uma corrente alternada V_r. Nestas condições, será induzida num dos pares de bobinas estacionárias a corrente $V_s=kV_r sen(\theta)$, e no outro par, a corrente $V_c=kV_r cos(\theta)$.

2.7.5 Outros

Existem ainda outros métodos para a medição e o sensoreamento de posições angulares baseados em interferometria, porém devido ao alto custo destes sensores eles não serão tratados aqui.

3 ANÁLISE E ESPECIFICAÇÕES DO BRAÇO A SER CONTROLADO

Uma foto deste braço pode ser visualizada na foto abaixo:

Figura 9 - Foto do braço a ser acionado.

É importante ressaltar aqui que, no início do desenvolvimento do presente trabalho este braço já se encontrava projetado. Desta forma este se constituiu no ponto de partida para a realização das demais tarefas necessárias para o completo desenvolvimento do presente projeto.

Como já dito anteriormente, o braço principal do robô dispõe de três atuadores para o posicionamento (responsáveis pelo movimento de arrastamento) da ferramenta médica, e um atuador adicional para a atuação propriamente dita desta ferramenta médica (fórceps). Como será justificado mais adiante, todos os atuadores deste projeto são rotativos. Abaixo reapresentamos o esquema de movimentação do braço:

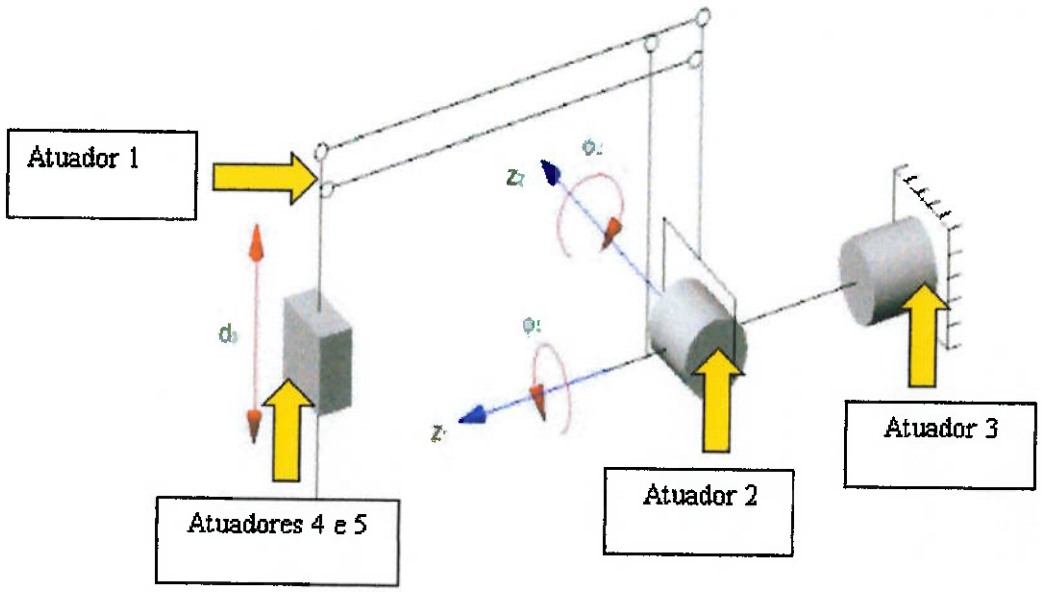


Figura 10 - Esquema de movimentação do braço mecânico.

O atuador 1 é acoplado a um fuso de esferas com passo de 3mm. Ou seja, este atuador será responsável por um movimento linear de subida e descida (uma reta) da extremidade do braço do robô, onde estará localizada a ferramenta médica para a operação, ou seja, o fórceps.

Já o atuador 2 é o responsável pelo acionamento desta estrutura, ainda causando um movimento contido no plano da Fig 3.3. mostrada abaixo.

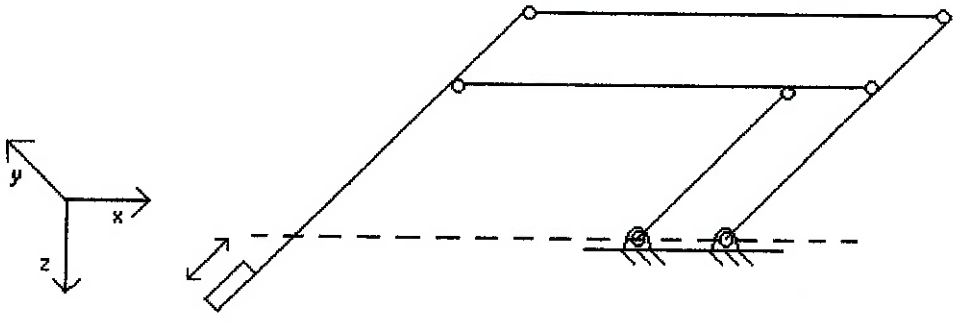


Figura 11 - Esquema em barras do movimento plano do braço.

Ou seja, para uma dada posição do atuador 1, o atuador 2 desenha um arco de circunferência no plano do papel. Combinando estes dois movimentos, obtém-se um setor circular.

Finalmente, o atuador 3 é responsável pela rotação de toda a estrutura em relação ao eixo x, tornando este movimento plano em um movimento no espaço. Combinando estes três movimentos, fica definido um volume de trabalho do robô que é um cone, mas que tem como base, uma calota esférica.

Notar que este volume de operação definido vale para as regiões onde o atuador linear está acionado de modo que a extremidade do robô se localize abaixo da linha que passa pelas articulações do robô (linha tracejada na figura acima 11).

A figura abaixo representa o volume de trabalho descrito acima, considerando neste caso que os motores 2 e 3 podem rotacionar o braço do robô de um ângulo de 180°.

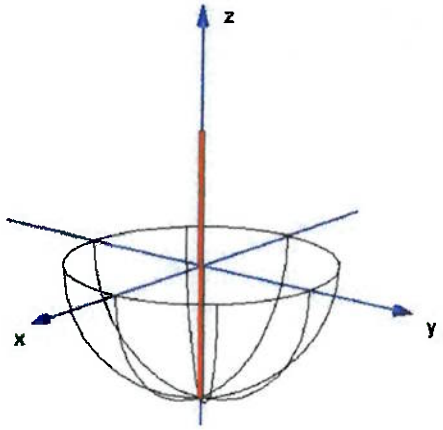


Figura 12 - Representação do Volume de Trabalho do Robô

Realizando-se um estudo sobre o efeito de rotação de um determinado ângulo θ do atuador 2, verifica-se que este movimento causa uma respectiva rotação da extremidade do braço do robô de $l.sen(\theta)$ na direção do eixo x. Agora definindo que a extremidade do braço do robô (l) trabalhará a no máximo 100 mm além da linha de centro, pode-se determinar o deslocamento linear gerado devido ao ângulo teta.

Assim, considerando um motor de passo com 200 passos por volta, ou seja, 1.8° graus por passo, acoplado ao motor com uma relação 1:1, verifica-se que o deslocamento causado na extremidade, com o curso máximo é de 3,1mm ($100*sen(1.8^0)$) em full-step ou de 1.55mm em half-step (com $\theta = 0.9°$).

O mesmo procedimento poderia ser feito para o movimento de rotação em torno do eixo y, obtendo-se os mesmos resultados. Assim, obteríamos uma calota esférica discretizada por estes passos.

Porém isto seria válido para uma relação de transmissão de 1:1, mas na prática haverá uma redução que servirá tanto para aumentar esta precisão, rigidez e também para absorver as possíveis vibrações.

3.1 Cálculo dos torques

O braço do robô desenvolvido pela equipe de pesquisadores pode ser representado esquematicamente pelo seguinte diagrama de barras mostrado na figura 3.3, adicionando-se um grau de liberdade de rotação em torno do eixo y.

Este diagrama é de fundamental importância para a determinação dos esforços que serão exigidos dos atuadores em uma determinada situação de funcionamento do robô. Assim sendo, realizou-se uma estimativa dos torques de atuação em cada um dos 3 atuadores, devido a três carregamentos de 1kgf, em cada uma das três direções x, y, e z. Acredita-se ser este carregamento bem superior aos esforços que serão encontrados pelo robô na prática, tendo assim uma boa margem de segurança.

Deste modo obteve-se o seguinte gráfico do torque necessário em função da posição:

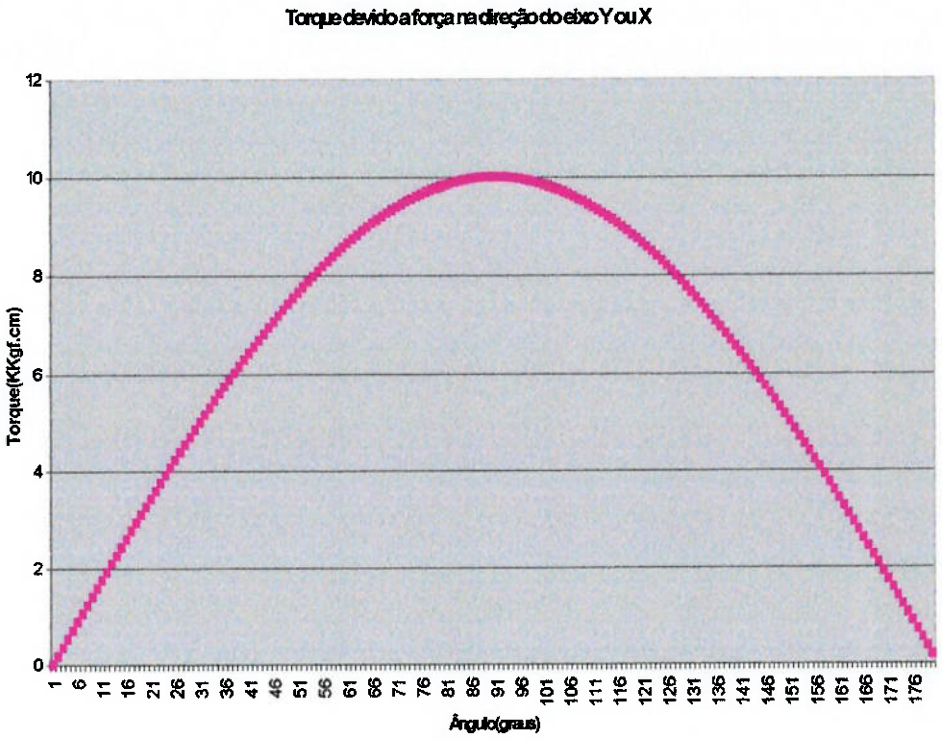


Figura 13 - Gráfico dos torques necessários.

No gráfico acima tem-se que o carregamento de 1kgf no eixo y, causa um torque apenas no motor (3), enquanto o carregamento no eixo x causa um torque apenas no motor(2). O carregamento no eixo z, por sua vez, causa apenas um torque no atuador (1) devido ao fuso de esferas e não provoca torque no motor 2, pois este torque será resistido pelo engastamento de fixação do braço.

Quantitativamente é possível notar que a posição onde o esforço é máximo ocorre quando o braço do robô esta na posição vertical, ou seja, a ferramenta forma um ângulo de 90° com a horizontal. Este valor é de 1 N.m. ou 10Kgf.cm.

4 ANÁLISE DE VIABILIDADE

4.1 Seleção dos componentes

Neste item será abordado o problema de seleção dos componentes necessários para a execução do sistema de controle desejado para os motores. Desta forma apresentamos abaixo um diagrama de blocos de um sistema genérico controlado em malha fechada, onde encontramos alguns componentes necessários para se efetuar a correta implementação e projeto do mesmo.

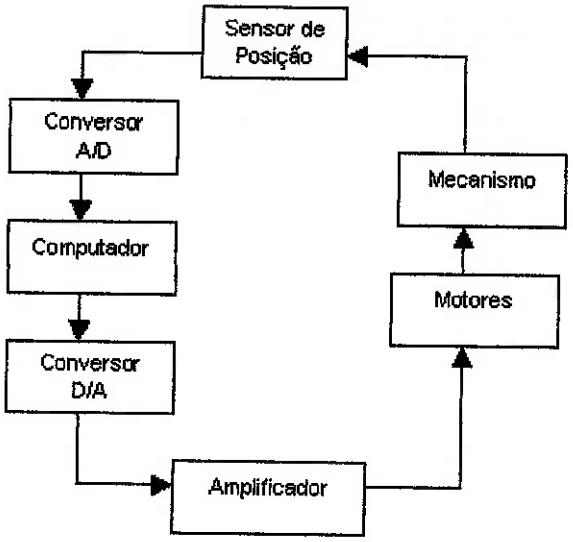


Figura 14 - Diagrama de blocos do sistema.

Assim percebemos a necessidade de selecionar os sensores, atuadores e conversores para este sistema. Além destes equipamentos será necessária a utilização de uma linguagem de programação e do sistema operacional para a implementação do software de controle.

4.1.1 Seleção dos atuadores

Quanto aos motores, como visto no item anterior, existem várias opções de atuadores. Porém verificou-se que para a aplicação em questão os motores

pneumáticos e hidráulicos não são boas opções. Os motores pneumáticos foram descartados pelo fato da dificuldade em se controlar, estes tipos de motores com precisão e baixo custo. Já os motores hidráulicos foram descartados devido à possibilidade de vazamento de fluído de trabalho o que pode contaminar o ambiente e prejudicar a limpeza necessária para um ambiente cirúrgico. Além disso, esses dois tipos de motores são mais utilizados para a movimentação de grandes cargas e por isso possuem um custo mais elevado em reação aos motores elétricos, porém no caso em análise as cargas são extremamente baixas, não justificando o custo adicional para a utilização desses tipos de motores.

Além das desvantagens apresentadas anteriormente pelos atuadores pneumáticos e hidráulicos podemos ainda citar as seguintes vantagens dos servomotores e motores de passo que justificam sua escolha.

Para os motores de passo: os motores de passo são atuadores alimentados com corrente DC, e com um comportamento previsível a ponto de permitir posicionamentos precisos, mesmo sem o uso de um sistema de controle em malha fechada. Estes dispositivos tem se tornado muito popular por se adaptarem ao mundo digital, impulsionado pelo crescente desenvolvimento do mundo dos computadores e dispositivos. Além disso, a queda no preço dos mesmos devido à popularização também é uma característica que deve ser mencionada nos dias de hoje.

Para os servomotores: a relação entre o torque e a velocidade de rotação do motor pode ser obtida matematicamente, e assim é possível provar que o torque diminui linearmente a medida em que a velocidade aumenta. Esta função possui uma inclinação de valor Kt.Ke/Ra e é independente da voltagem no terminal ou da velocidade. Esta característica é o que torna fácil o controle de velocidade e posição em um motor DC e também em DC brushless, características que não podem ser encontradas em motores AC ou em motores de passo.

Deste modo foram selecionados os seguintes motores para as posições indicadas na figura abaixo:

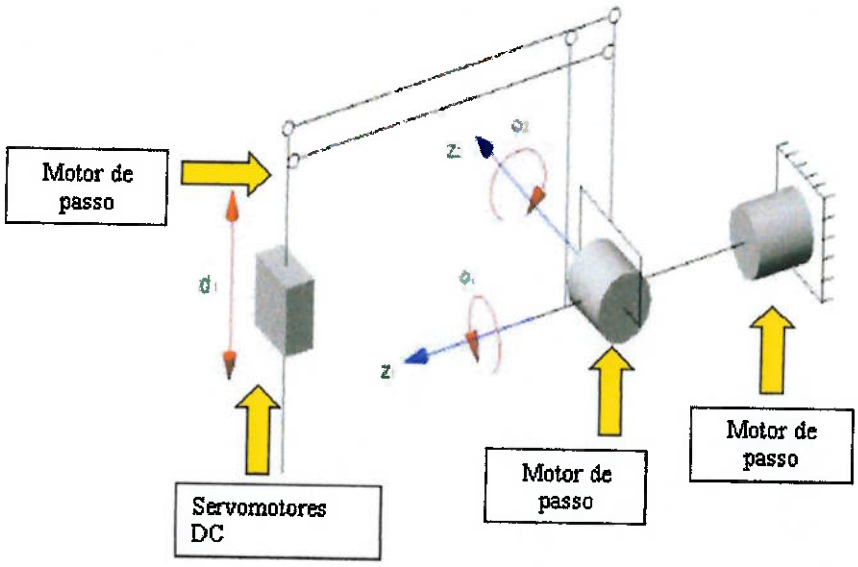


Figura 15 - Esquema de colocação dos motores selecionados.

Na disposição mostrada na figura acima, percebe-se que foram selecionados três motores de passo e dois servomotores. Os três motores de passo serão utilizados para executar o acionamento dos movimentos do braço e do punho, já os servomotores seriam utilizados para o acionamento do fórceps que está acoplado ao punho.

Esta disposição se justifica pelo fato de que como dito anteriormente, os motores de passo são mais fáceis de se controlar, podendo até mesmo ser controlado em malha aberta. Além disso, motores de passo estão cada vez mais baratos. Desta forma procurou-se ao máximo explorar as facilidades de controle dadas pelo motor de passo, selecionando-os para a maioria dos acionamentos necessários, exceto para o acionamento do fórceps propriamente dito, já que este requer maior precisão no seu posicionamento é que se selecionou um servomotor DC, pois devido o movimento dos motores de passo serem discretizados na forma de passos, eles não se mostram eficientes quando se deseja ter alta precisão no controle de posição.

Apesar de todas estas vantagens dos servomotores, serão utilizados motores de passo em todos os acionamentos, pois não foi possível disponibilizar os dois servomotores. Desta forma, estes servomotores foram então substituídos por dois motores de passo que já se encontravam disponíveis para o projeto.

4.1.2 Seleção dos sensores e dos conversores A/D e D/A

Como descrito acima, todos os acionamentos utilizados no presente no projeto são motores de passo. Assim todos serão controlados com metodologias de controle em malha aberta, e desta forma, não utilizarão sensores.

Os conversores A/D e D/A também não serão utilizados, pois o controle do motor de passo é feito através de sinais digitais e assim, este pode ser conectado diretamente a porta paralela do PC.

4.1.3 Seleção da linguagem de programação e do sistema operacional

Uma das características principais no projeto de um robô cirúrgico, e que também o diferencia do projeto de robôs industriais, é o alto grau de segurança e confiabilidade que este deve proporcionar. Neste sentido, torna-se necessário, em primeiro lugar, uma plataforma de desenvolvimento confiável, com regularidade, simplicidade, facilidade de uso e que ofereça o seu completo domínio aos projetistas. Caso contrário, de nada valeriam os complexos sistemas de segurança e controle que virão um dia a ser desenvolvidos para o projeto maior.

Neste contexto, destaca-se a linguagem de programação comercial C, com confiabilidade reconhecida e que proporciona a capacidade da construção de elementos complexos a partir de estruturas simples. Por um lado esta linguagem é de alto nível, como se deseja para a implementação de controladores. Por outro lado, ela oferece recursos para o trabalho em níveis mais baixos, de modo a acessar detalhes da máquina, como por exemplo, o acesso direto à memória RAM, no controle de portas de comunicação, e tudo isso com o máximo de eficiência do computador.

A linguagem C é ainda uma das mais rápidas de alto nível, por estar próximo do hardware. Além disso, essa plataforma de desenvolvimento possui características técnicas compatíveis com a instrumentação utilizada, como por exemplo, a placa de aquisição de sinais AD/DA que possui como recomendação do fabricante utilização da linguagem C.

E finalizando, outro motivo para esta escolha, é o fato do presente trabalho, ser parte de um projeto maior, onde haverá a necessidade de integração e compartilhamento de diversos sistemas e recursos, devendo ser destacado aqui

novamente a performance da linguagem C e também um acordo entre as diversas partes deste grupo maior.

Uma breve consideração ao sistema operacional escolhido também deve ser realizada. Pelos mesmos requisitos já apresentados na escolha da linguagem de programação, o sistema operacional deve ser estável. Assim, já se poderia eliminar a hipótese do emprego do ambiente gráfico Windows. Outro motivo que torna este ambiente não aplicável é o fato do mesmo ser multitarefa, ou seja, ele mesmo gerencia os eventos e interrupções de acordo com seu critério de precedência. E este fato inviabiliza completamente o tratamento de sinais e, portanto impedindo o desenvolvimento dos sistemas de feedback.

Como opções, resta o tradicional DOS e o despontante sistema de domínio público Linux. O primeiro destaca-se pela simplicidade, facilidade de uso e popularidade, com extenso suporte a drivers, enquanto o segundo, é ainda muito recente, não proporcionando ferramentas, suporte a dispositivos, drivers, bibliotecas de funções, o que dificultaria muito o trabalho, sem oferecer grandes vantagens.

4.2 Controle de motor de passo

Para se efetuar o controle de motores de passo deve-se antes proceder a escolha de um controlador. Porém, a escolha deste controlador não depende apenas da finalidade do projeto, mas também do tipo do motor usado (unipolares de imã permanente, de relutância variável e híbrido). Desta forma, vale ressaltar aqui que neste projeto serão utilizados motores de passo do tipo híbrido.

O controle de motores de passo baseia-se em chavear a corrente em cada bobina do motor e no controle de sua direção. Abaixo mostramos um circuito que conecta diretamente as bobinas do motor com a fonte de energia, e é controlado por um sinal digital que determina quando a chave é ligada e desligada. Na figura os blocos representam as chaves.

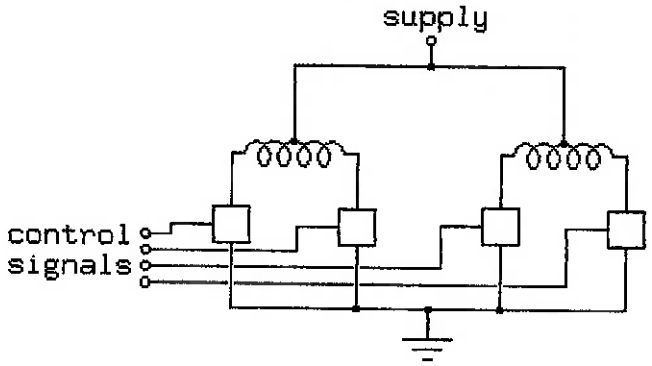


Figura 16 - Motor de passo unipolar – circuito de controle.

Uma unidade de controle não mostrada é responsável por fornecer os sinais de controle para abrir e fechar as chaves no tempo apropriado em ordem a rotacionar o motor. Esta unidade de controle é comumente um computador ou um controlador de interface programável, com o software gerando diretamente as saídas necessárias para controlar as chaves. No caso deste projeto será um microcomputador conectado aos motores via porta paralela.

O controle dos motores pode ser classificado em três tipos de acordo com o tipo do passo dado pelo motor. São eles:

Fullstepping: Neste tipo de controle cada movimento do motor corresponde a um passo inteiro, ou seja, o rotor se move entre as posições determinadas por duas bobinas consecutivas. Para se obter este tipo de movimento deve-se energizar a bobina na qual se deseja alinhar o rotor, desta forma o rotor se alinhará com a mesma. Para que o rotor movimente um passo deve-se desenergizar essa bobina e energizar a próxima bobina, fazendo com que o rotor rotacione o equivalente a um passo inteiro.

Halfstepping: Neste tipo de controle cada movimento do rotor corresponde a meio passo (deslocamento correspondente à metade do ângulo entre duas bobinas consecutivas). Para se obter este modo de deslocamento do rotor, deve-se antes de desenergizar a bobina com a qual o rotor se encontra alinhado, energizar a próxima bobina. Isto faz com que o rotor se desloque o equivalente a meio passo, se posicionando entre as bobinas energizadas. Para proceder ao próximo movimento deve-se desenergizar a primeira bobina, fazendo com que o rotor saia da posição

intermediária e se alinhe com a segunda bobina energizada. Para os próximos meio-passos deve-se repetir a seqüência acima.

Microstepping: Serve para dois propósitos. Primeiro ele permite um motor de passo parar e manter a posição entre uma posição de full step ou uma posição de half step. Segundo, ele elimina a característica JERKY de operação do motor de passo em baixa velocidade e o barulho a velocidades intermediárias, e terceiro, reduz problemas com ressonância. Apesar de alguns controladores de micropasso oferecerem centenas de posições intermediárias entre os passos, deve-se salientar que os micropassos geralmente não oferecem grande precisão, sendo que isso se deve tanto a problemas de linearidade quanto aos efeitos do atrito estático.

4.2.1 Relação entre as curvas de Pull-In e Pull-Out de um motor de passo e seu controle

As curvas de Pull-in e Pull-out apresentam as características do torque do motor de passo em função da freqüência de acionamento do mesmo. Enquanto a curva de Pull-out apresenta a máxima freqüência de acionamento do motor para um dado torque, sem perda de passo, em regime permanente, a curva de pull-in realiza a mesma descrição, porém em regime transitório. Em outras palavras, um motor, quando operado em regime permanente, fornecerá como máximo valor de torque o valor obtido na curva de pull-out. Já a curva de pull-in, também chamada de área de start / stop, apresenta a máxima freqüência de acionamento na qual o motor pode ser ligado ou desligado instantaneamente, com o correspondente carregamento aplicado, sem perda de sincronismo.

Assim, caso verifique-se que o motor deva trabalhar na região externa a curva de pull-out, é necessário que o motor seja levado ao seu ponto de operação através de uma curva de aceleração, ou seja, o motor é partido a uma velocidade inferior, e só então levado ao seu ponto de operação. As curvas de aceleração serão devidamente abordadas num tópico futuro.

4.2.2 Especificação da solução de controle dos motores de passo

Dentre as diversas formas de realizar o acionamento e o controle dos motores de passo, que por sua vez podem ser configurados em diferentes modos de

operação, foi decidido que seria mais conveniente para o presente projeto, realizar o acionamento do mesmo utilizando para isso um driver de controle comercial. Assim, não se pretende neste projeto realizar a construção de simples circuitos de acionamento, e ao invés, disso realizar esforços para maximizar a performance dos atuadores que são o ponto chave do projeto.

Dentre os drivers de potência existentes no mercado, foi selecionado o driver 3535O da Applied Motion Products. Este é um driver para motores de passo do tamanho 14 ao 34, que trabalha com correntes de até 3.5 ampères. Possui opção para acionamento em fullstep ou halfstep, configurada através de um jumper, e algumas outras funcionalidades que não convém citar aqui.

A figura abaixo ilustra suas conexões:

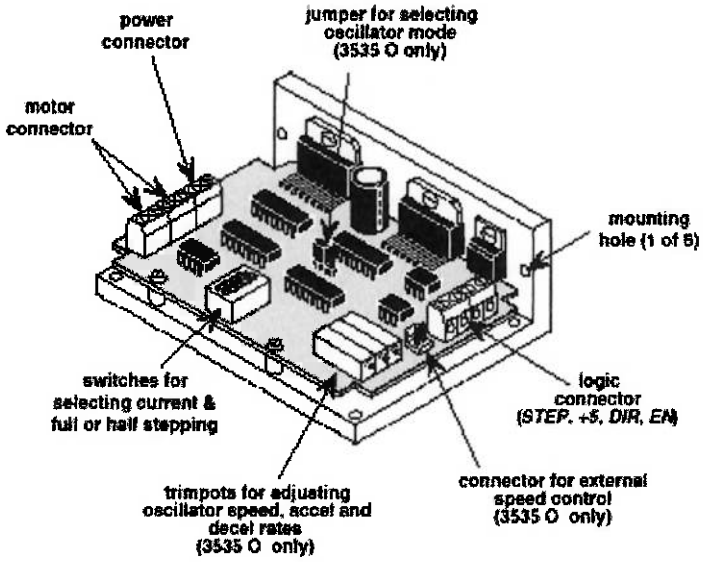


Figura 17 - Driver de controle a ser utilizado para controlar os motores de passo.

Este driver de controle trabalha apenas em modo bipolar, ou seja, o princípio da mudança da direção da corrente que atua no enrolamento é conseguido através da mudança de polaridade da voltagem aplicada nos terminais do enrolamento. Esta mudança é conseguida através de quatro chaves, que formam uma ponte H. O modo unipolar, que requer apenas duas chaves por fase, não pode ser executado com este controlador.

Na prática, a diferença entre acionar um motor em modo bipolar ou em unipolar se faz sentir no torque máximo que se pode obter para um dado motor.

Quando um motor é ligado em configuração center-tap, a cada chaveamento da bobina, está se utilizando apenas metade do volume do cobre total disponível em cada fase. Conseqüentemente, o motor apresentará menores torques e maior perda de potência para uma mesma potência de saída requerida.

Quanto à ligação do motor ao driver, isto depende de quantos fios possui o motor, ou seja, depende dos modos que o motor pode trabalhar. Para um motor com seis fios (o qual é o caso do motor selecionado), este pode ser ligado ao controlador com as bobinas conectadas em série ou em center tap. Como já discutido, a ligação em center-tap fornece um torque máximo menor que a ligação em série, porém nesta configuração é que se obtém maiores velocidades de rotação do motor.

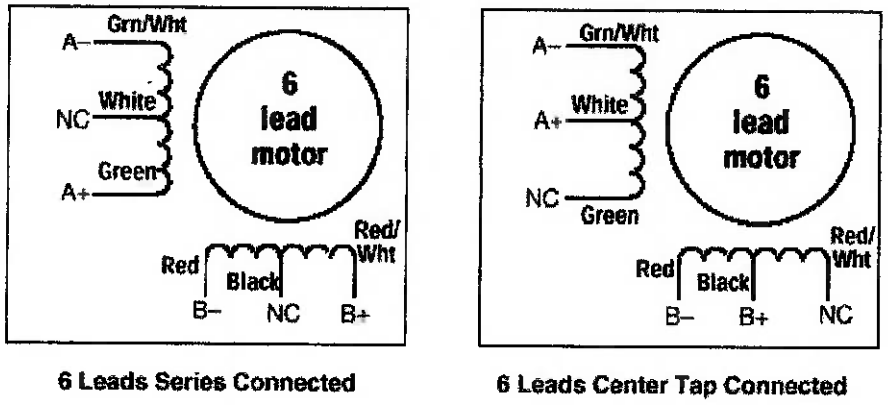


Figura 18 - Esquemas de ligação do motor em série ou com center-tap.

Considerando as características do projeto mecânico e suas especificações de precisão e torque foi decidido conectar o motor em série e operá-lo em modo full-step.

Considerando-se as posições limite do alcance do braço do robô, e ainda o fato de que este será acionado com uma relação de transmissão de redução, percebe-se claramente que o motor em funcionamento será exigido apenas em velocidades extremamente pequenas comparadas a sua capacidade e será solicitado entre as posições de curso mínimo e máximo do robô em um número pequeno de voltas completas. Assim, justifica-se o fato de adotarmos a ligação das bobinas em série, privilegiando o torque.

Utilizando-se o driver de potência acima descrito vê-se que é necessário controlar 3 entradas digitais, além de alimentar o circuito digital do próprio driver.

Este circuito digital possui a exatamente a função de realizar a lógica necessária para transformar estes sinais de controle em uma lógica que chaveia o acionamento das bobinas do motor de passo. A alimentação dos circuitos digitais requer uma corrente de 15 mA, enquanto as demais entradas exigem 5 mA.

O primeiro sinal de controle recebe a informação sobre os passos a serem executados pelo motor. Ou seja, nesta entrada, deve ser fornecido um trem de pulsos, tal que o circuito, que é sensível a borda de descida, transformará cada um destes pulsos em um passo do motor. Assim, controlando a freqüência destes pulsos, controla-se a velocidade do motor e o número de passos desejados.

Já o segundo sinal de controle atua no sentido de rotação do motor, dependendo se o pulso nesta entrada for um sinal alto ou baixo. O último sinal de controle é apenas um Enable / Disable na alimentação do motor, que poderá ser utilizado na implementação de uma função de emergência que desative o motor de passo imediatamente ao recebimento de uma determinada ordem. Desta forma teremos o seguinte esquema para o sistema de controle:

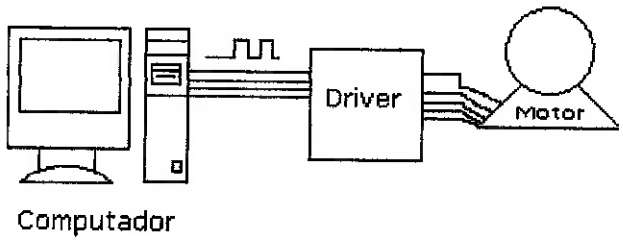


Figura 19 - Esquema do sistema de controle de motores de passo

Como se vê na figura acima a geração destes sinais de controle serão realizadas por um computador (através da porta paralela), que é o responsável pelo controle dos atuadores e também pelo dispositivo de comando (mouse espacial).

4.2.2.1 A porta paralela

A porta de comunicação paralela possui três endereços distintos para a entrada e saída de dados, sendo um endereço de 8 bits para entrada de dados, um endereço de 8 bits para aquisição de dados e um endereço de quatro bits para entrada

/ saída de dados. Aqui nos interessa apenas os bits referentes à saída de dados, portanto, há 12 bits disponíveis.

Conforme visto em itens anteriores, para realizar o controle de um motor de passo, é necessário controlar dois sinais. O primeiro é o sinal que controla a direção do passo do motor (horário ou anti-horário), e o segundo é o sinal de clock, ou seja, quando o nível lógico deste sinal desce, o controlador chaveia as fases do motor de passo de modo que este de um passo na direção escolhida através do sinal anterior.

Assim, foram dedicados 2 bits da porta 378h para o controle de cada um dos 3 motores de posicionamento. Dos dois bits restantes, um deles foi compartilhado entre os 3 drivers como fonte de alimentação para o circuito digital, pois a alimentação deste é independente da alimentação do circuito de potência.

Além dos 3 motores principais, mais 4 bits de sinal são necessários para o controle dos 2 motores de passo referentes à orientação e atuação da garra. Foi utilizado então a porta 37Ah, que dispõem de exatamente 4 bits de saída de dados.

A tabela a seguir apresenta a distribuição dos pinos como construído.

Porta	Bit Lógico	Direção do Sinal	Pino Físico da porta paralela	Função
378h	7	Saída	9	Disponível
	6	Saída	8	Alimentação +5V
	5	Saída	7	Clock Motor 1
	4	Saída	6	Direção do Motor 1
	3	Saída	5	Clock Motor 2
	2	Saída	4	Direção do Motor 2
	1	Saída	3	Clock Motor 3
	0	Saída	2	Direção do Motor 3
37Ah	3	Saída / Entrada	17(L)	Clock Motor 5
	2	Saída / Entrada	16	Direção do Motor 5
	1	Saída / Entrada	14(L)	Clock Motor 4
	0	Saída / Entrada	1(L)	Direção do Motor 4

(L) Indica que o sinal é invertido

Tabela 4 - Distribuição dos bits da Porta Paralela.

4.3 A integração dos movimentos dos atuadores

Depois de completada a implementação dos controladores para cada motor individualmente, deverá ser abordado o problema de controle simultâneo dos vários atuadores. Desta forma será realizada uma interpolação linear entre os pontos iniciais

e finais dos deslocamentos do robô e deverá ser efetuada uma trajetória retilínea entre esses pontos, através da movimentação simultânea dos vários atuadores.

Deste modo o funcionamento do software em desenvolvimento deverá apresentar o seguinte fluxograma:

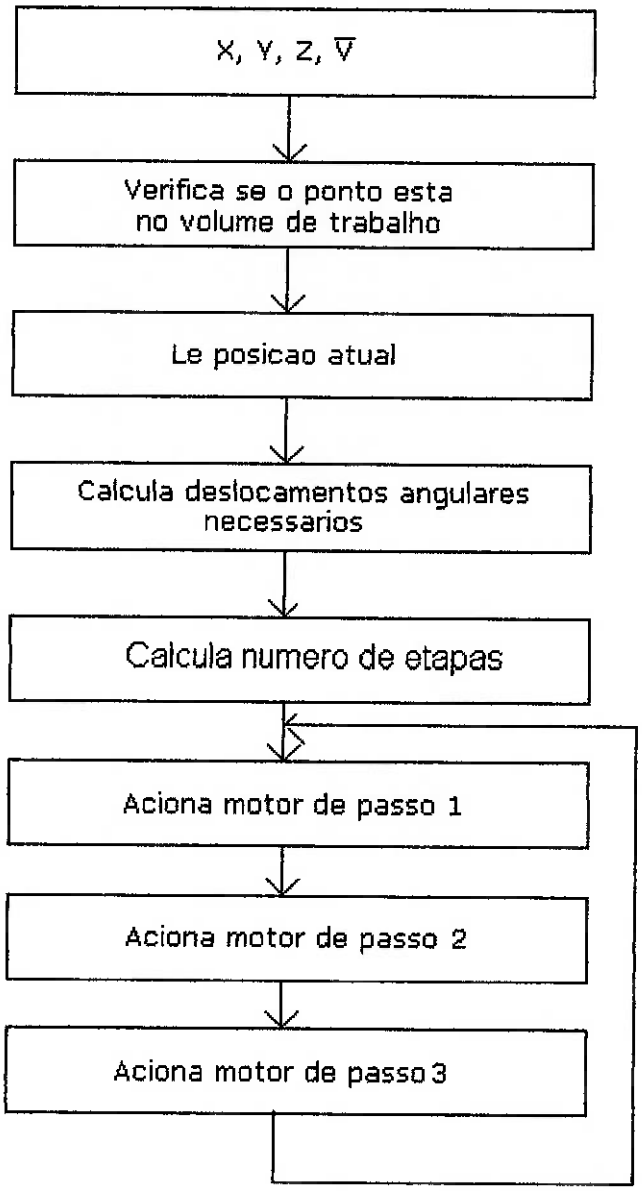


Figura 20 - Fluxograma de operação do software a ser implementado

No fluxograma acima percebemos uma etapa relacionada à verificação de um intertravamento de posição, ou seja, o software deverá ser capaz de reduzir o volume máximo de alcance do robô, para um volume de trabalho a ser pré-

programado antes de cada cirurgia, de acordo com as características de cada paciente.

5 CINEMÁTICA DO ROBÔ CIRÚRGICO

Deseja-se neste instante, realizar o estudo da Cinemática Direta e Inversa do Robô Cirúrgico. Em outras palavras, o objetivo agora é realizar o estudo da posição e da velocidade do efetuador e dos demais ligamentos constituintes do mesmo. Enquanto na Cinemática Direta procurar-se-á obter a posição e velocidade do efetuador para uma dada posição das articulações, na Cinemática Inversa estaremos preocupados em, fornecidas a posição e a velocidade do efetuador desejadas, obter as posições e velocidades correspondentes das articulações.

5.1 Cinemática Direta

Para o desenvolvimento do cálculo da cinemática direta, decidiu-se por utilizar a notação de Denavit-Hartenberg, que é um modo sistemático de posicionar sistemas de coordenadas nos ligamentos do manipulador, e assim descrever a posição e orientação relativa entre ligamentos consecutivos, baseando em transformações homogêneas. Para mais informações sobre o assunto, recomenda-se [3].

Utilizando-se então a notação de Denavit-Hartenberg chegamos ao seguinte modelo:

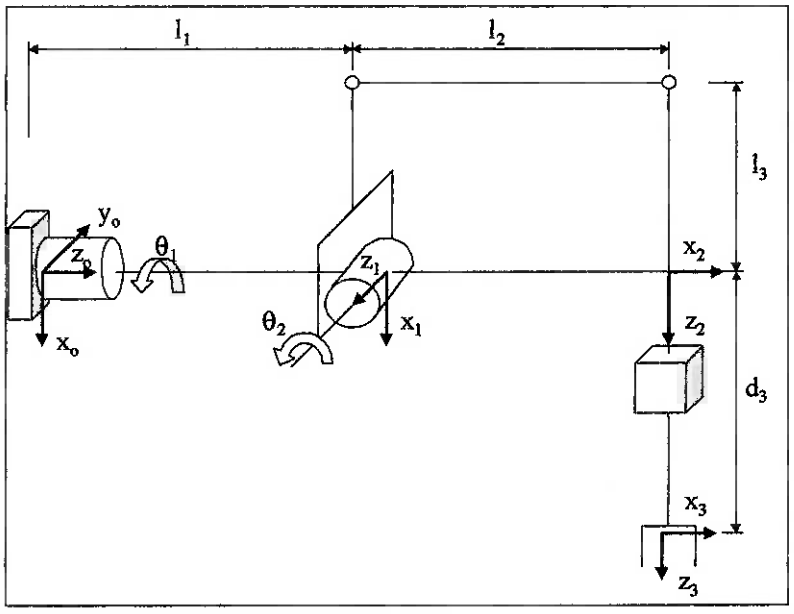


Figura 21 - Esquema do robô indicando os eixos de Denavit-hartenberg

A partir da análise do modelo geométrico do braço apresentado acima e da mecânica do mecanismo, o qual envolve um mecanismo de barras articuladas, podemos simplificá-lo ligeiramente, obtendo o modelo abaixo.

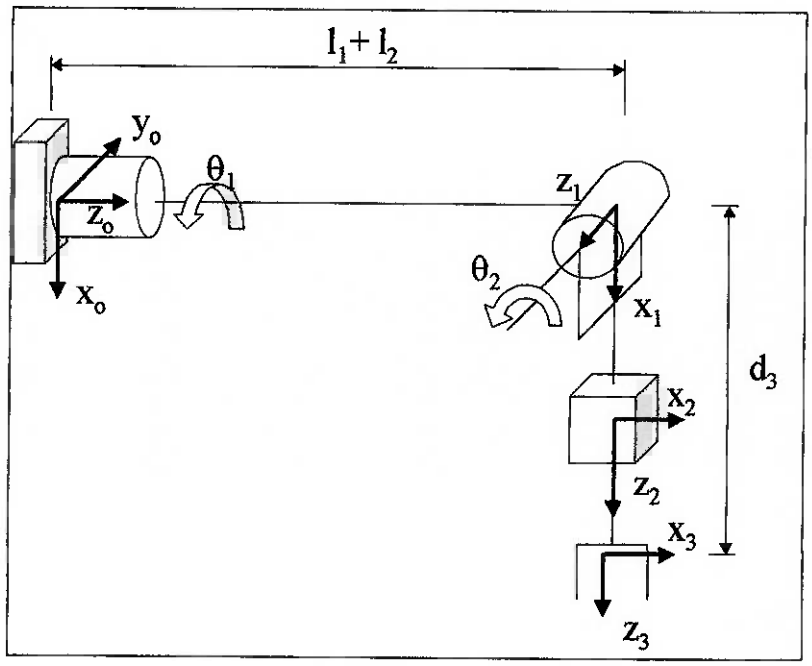


Figura 22 - Modelo simplificado indicando os eixos de Denavit-Hartenberg

Esta é uma mudança de modo a obter um sistema funcionalmente equivalente, mas que, no entanto, possuirá um modelo cinemático cuja notação de Denavit-Hartenberg é mais direta.

Deve-se notar que no modelo equivalente, está sendo incluído no problema, apenas 3 graus de liberdade. Chamaremos estes 3 graus de liberdade como sendo o resultado do acionamento e controle dos 3 atuadores de Posicionamento. Os demais 2 graus de liberdade, correspondentes a uma rotação do efetuador do Robô e ao movimento de abertura e fechamento da garra, estarão sendo aqui tratados como sendo o Problema de Orientação do Efetuador, e sua solução está sendo separada do Problema do Posicionamento. Trabalharemos inicialmente apenas com o Problema de Posicionamento do Efetuador, deixando a Orientação para mais adiante. O motivo desta decisão foi devido a dois fatores. O primeiro diz respeito ao modo como a orientação será controlada. Acredita-se que um cirurgião inicialmente posicionará o efetuador do robô, e só então controlará a orientação. O segundo motivo diz respeito

à indefinição e à inexistência de solução do projeto mecânico até o instante deste estudo.

Dadas as orientações dos sistemas de coordenadas, as dimensões principais do modelo e as articulações, foram obtidos os parâmetros de Denavit-Hartenberg, apresentados na tabela a seguir.

Ligamento	a_i	α_i	d_i	θi
1	0	90°	$l_1 + l_2$	θ_1*=90°
2	0	90°	0	θ_2*=90°
3	0	0°	d_3*	0°

Tabela 5 - Parâmetros de Denavit-Hartenberg

Apresenta-se também como resultado deste processo, as matrizes $A_i^{i-1}(q_i)$ que são matrizes homogêneas, e também a matriz $A_o^3(q_i)$, representando a posição e orientação do efetuador em relação ao sistema da base, em função das posições de todas as articulações.

Para a transcrição das matrizes abaixo, definimos:

- $c_1 = \cos(\theta_1)$;
- $c_2 = \cos(\theta_2)$;
- $s_1 = \text{sen}(\theta_1)$;
- $s_2 = \text{sen}(\theta_2)$;

$$A_o^1 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & l_1 + l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_1^2 = \begin{bmatrix} c_2 & 0 & s_2 & 0 \\ s_2 & 0 & -c_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_2^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ e}$$

finalmente a matriz :

$$A_0^3 = \begin{bmatrix} c_1 \cdot c_2 & s_1 & c_1 \cdot s_2 & c_1 \cdot s_2 \cdot d_3 \\ s_1 \cdot c_2 & -c_1 & s_1 \cdot s_2 & s_1 \cdot s_2 \cdot d_3 \\ s_2 & 0 & -c_2 & l_1 + l_2 - c_2 \cdot d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Desta forma, podemos relacionar as coordenadas de um ponto descrito no sistema do efetuador com as coordenadas equivalentes da base. Porém, como veremos mais adiante, é de nosso interesse apenas obter a posição do centro do efetuador em função das coordenadas de junta do robô. Para isto, basta multiplicarmos a matriz homogênea A_0^3 pelo seguinte vetor de posição descrito nas coordenadas do efetuador $p = [0 \quad 0 \quad 0 \quad 1]$ (este ponto descreve a posição do centro do efetuador, ou seja, do fórceps). Desta forma, obtemos as seguintes equações:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{base} = \begin{bmatrix} c_1 \cdot s_2 \cdot d_3 \\ s_1 \cdot s_2 \cdot d_3 \\ l_1 + l_2 - c_2 \cdot d_3 \end{bmatrix}$$

Ainda de posse da matriz A_0^3, há mais algumas informações relevantes e importantíssimas, que também são objetos deste trabalho. Pode-se por exemplo obter a velocidade linear do efetuador em relação ao sistema de coordenadas da base por derivação da expressão anterior,

$$\begin{bmatrix} v_x^3 \\ v_y^3 \\ v_z^3 \end{bmatrix}_{base} = \begin{bmatrix} -s_1 s_2 d_3 & c_1 c_2 d_3 & c_1 s_2 \\ -c_1 s_2 d_3 & -s_1 c_2 d_3 & s_1 s_2 \\ 0 & s_2 d_3 & -c_2 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{d}_3 \end{bmatrix}$$

e também o Jacobiano:

$$J = \begin{bmatrix} -s_1 s_2 d_3 & c_1 c_2 d_3 & c_1 s_2 \\ -c_1 s_2 d_3 & -s_1 c_2 d_3 & s_1 s_2 \\ 0 & s_2 d_3 & -c_2 \\ 0 & s_1 & 0 \\ 0 & -c_1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

5.2 Cinemática Inversa

Diferentemente de equações lineares, não há algoritmos genéricos para serem aplicados a fim de resolver qualquer conjunto de equações não lineares como as obtidas acima.

Em geral, não são consideradas soluções numéricas, pois procedimentos iterativos não garantem que sejam encontradas todas as soluções, além de ser um processo lento. Fica aqui então explícito nosso desejo de trabalhar com soluções na forma fechada, ou seja, resolvendo expressões analiticamente.

Para o cálculo da cinemática inversa utilizamos o sistema de coordenadas fixo na base do robô, ou seja, o sistema $0_0x_oy_oz_o$.

Analisando o mecanismo posicionador do robô, verifica-se que sua estrutura é a de um robô do tipo esférico, donde tira-se que:

$$\begin{cases} \theta_1 = a\tan 2(p_y, p_x) \\ \theta_2^{*} = a\tan 2(p_z - l_1 - l_2, \sqrt{p_x^{\,2} + p_y^{\,2}}) \\ d_3 = \sqrt{p_x^{\,2} + p_y^{\,2} + (p_z - l_1 - l_2)^2} \end{cases}$$

onde: $\theta_2 = \theta_2^{*} + \frac{\pi}{2}$ e $p = [px, py, pz]^t$ é a posição do efetuador em coordenadas da base.

5.3 Simplificações da cinemática

Podemos simplificar ainda mais estas equações se adotarmos um sistema de coordenas fixo no ponto em que a linha média encontra-se com a ferramenta. Assim é possível relacionar a posição dos acionadores com a posição do efetuador. Analisando a figura abaixo, verificamos que este ponto permanece constante para quaisquer valores das coordenadas de junta e desta forma podemos adotá-lo como nosso novo centro de coordenadas.

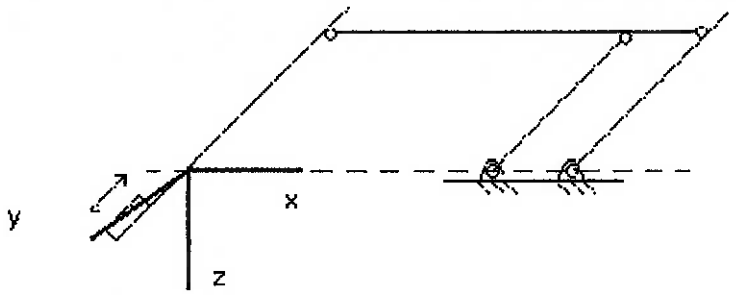


Figura 23 - Sistema de Coordenadas adotado

Para relacionarmos este novo centro de coordenadas, com o obtido através da formulação de Denavit-Hartenberg basta fazermos as seguintes equivalências:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{novo} = \begin{bmatrix} l_1 + l_2 - z_0 \\ y_0 \\ x_0 \end{bmatrix}_{DH}$$

A partir desta orientação, são variáveis de interesse:

θ: o ângulo formado entre o eixo z e a ponta do braço, no plano yz;

α: o ângulo formado entre o eixo z e a projeção da ponta do braço no plano xz;

r: o comprimento da ponta do braço do robô.

Percebe-se que estes ângulos possuem uma relação direta e independente com cada um dos graus de liberdade dos dois motores de passo utilizados nas rotações dos braços. Estas relações são as seguintes:

$$\begin{cases} \theta = \theta_1 \\ \alpha = \theta_2^* \end{cases}$$

Por sua vez, o comprimento da ponta do braço possui uma relação direta com o motor que aciona a guia de esferas, ou seja:

$$\{r = d_3$$

Uma justificativa para esta simplificação tem como um primeiro argumento o fato de que no Robô Cirúrgico, prioriza-se o controle de posição do efetuador, em

detrimento ao controle das demais barras constituintes do braço. Em outras palavras, sabe-se que no funcionamento deste robô, não é necessário ter controle sobre o movimento ou trajetória de qualquer outro ponto constituinte do braço do robô e o que importa é o controle da ferramenta dentro do Volume de Trabalho, que está de fato inserida no Paciente.

Do ponto de vista da utilização final do robô, acredita-se que a adoção deste novo sistema de coordenadas seja mais conveniente, pois a origem desta base é exatamente o ponto de introdução da ferramenta cirúrgica no paciente. Este é o ponto de referência a todos os movimentos e orientações, contribuindo também de forma lógica na integração do robô com outros dispositivos. Como exemplo, um mouse espacial o qual o médico possivelmente utilizará na operação poderá ter como centro de coordenadas o mesmo ponto de inserção da ferramenta.

5.4 Análise das transmissões

Além da cinemática inversa e da cinemática direta é necessário conhecermos as transmissões entre os atuadores e as coordenadas de junta por ele guiadas, para que possamos determinar o deslocamento angular que é necessário dar ao motor para que ele leve a junta ao estado (posição) desejada.

5.4.1 Motor R

O motor que controla o movimento linear (movimento na direção de r), está acoplado a um fuso de esferas. Como o passo do fuso de esferas é conhecido, pode-se facilmente relacionar o deslocamento angular do motor com o deslocamento linear da mesa (onde está conectado o efetuador) no fuso de esferas, através da seguinte relação:

$$DeslocAngular_R = \frac{2\pi \bullet DeslocLinear}{passo}$$

5.4.2 Motor θ

O motor θ está acoplado a um conjunto de correia dentada e polias, sendo que a relação de transmissão é dada pela relação entre os diâmetros da polia movida e da polia motora. Uma vez conhecida à relação de transmissão, basta aplicarmos a seguinte relação:

$DeslocAngular_{motor\theta} = \theta \bullet i$, onde i = relação de transmissão.

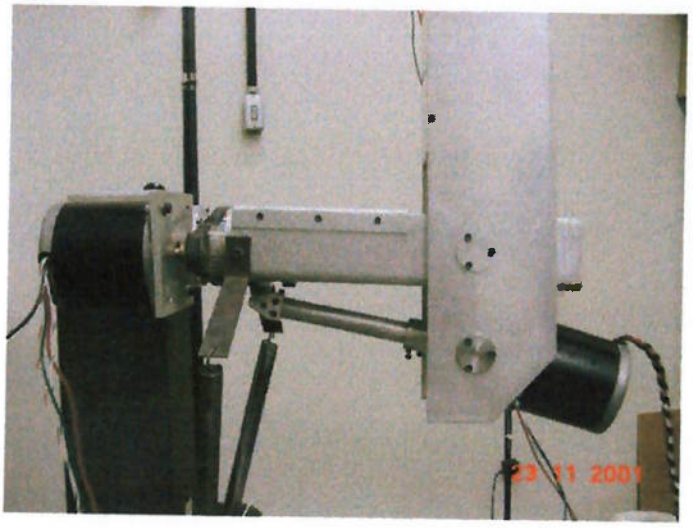


Figura 24 - Foto do motor θ

5.4.3 Motor α

Diferentemente dos casos anteriores, o motor α não apresenta uma relação linear entre o seu deslocamento angular e o deslocamento angular da junta por ele controlada. O motor α está acoplado na estrutura da seguinte forma:

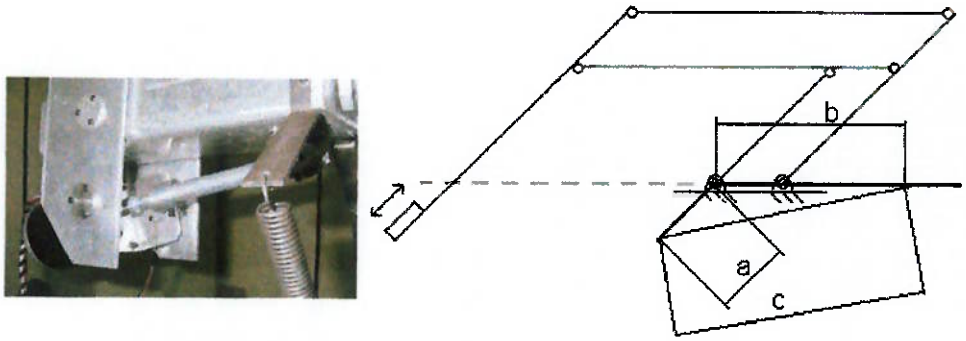


Figura 25 - Esquema de conexão do motor α.

O motor α altera através de um fuso o comprimento da barra de comprimento c (e que, portanto é variável), indicado na figura acima.

Para podermos calcular o ângulo de junta α que é dado pelo ângulo formado entre as barras de comprimento fixo a e b indicadas na figura acima é necessário que conheçamos o valor atual do comprimento c. Conhecendo-se o comprimento c, podemos aplicar a regra dos co-senos e obter:

$$\alpha = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

Se adotarmos a posição em que $\alpha = \frac{\pi}{2}$ como o comprimento original da barra, basta calcularmos a variação do comprimento (a partir do original) que deveremos dar a esta barra. Assim teremos: $\Delta c = c - c_0$.

Uma vez conhecido o valor de Δc, podemos calcular o deslocamento angular do motor α, através da mesma expressão utilizada para o motor r, ou seja:

$$DeslocAngular_{motor\alpha} = \frac{2\pi \bullet \Delta c}{passo}$$

Em outras palavras, enquanto o motor θ possuía uma equivalência direta entre número de passos dados e deslocamento angular do robô, a equivalência de passos do motor α e do ângulo do robô não é direta, pois o efeito de um passo no deslocamento angular depende também da posição atual do efetuador. Isso ocorre devido à transmissão em forma de fuso introduzir mais uma equação, que no caso é uma função trigonométrica.

5.5 Volume de Trabalho e existência de Solução Cinemática

Uma vez de posse das equações da cinemática inversa determinada nos itens anteriores, surge agora a próxima etapa do desenvolvimento. Refere-se à solução quantitativa propriamente dita das equações, ou seja, deseja-se saber se a solução existe ou não e encontrar os valores numéricos. Esta tarefa depende também do Volume de Trabalho do manipulador. Para que uma solução exista, o ponto desejado deve estar contido dentro do Volume de Trabalho do manipulador.

Geralmente é útil considerar duas definições para Volume de Trabalho: Volume de Trabalho Dextrous é a região do Volume de Trabalho na qual o efetuador do robô consegue alcançar com qualquer orientação que for arbitrariamente escolhida, e o Volume de Trabalho Alcançável é o volume de Trabalho na qual o robô pode alcançar com pelo menos uma orientação.

Como características do robô em desenvolvimento, este não possuirá Volume de Trabalho Dextrous, ou seja, ele não possui a capacidade de posicionar o efetuador em uma determinada posição, com uma orientação arbitrária. Pelo contrário, este robô permite o posicionamento com uma única orientação (considerando apenas os três motores). Isto deve se basicamente ao fato de que quando um manipulador possui menos que seis graus de liberdade, ele não consegue atingir posições e orientações de destino gerais em um espaço tridimensional.

Uma descrição sobre a movimentação do braço do robô e seus respectivos pontos de alcance foi realizada no início deste documento. No entanto, para a aplicação cirúrgica para o qual o robô está sendo desenvolvido, é de grande conveniência que o robô possua os movimentos limitados a um determinado volume de trabalho, definido de modo que não seja permitido algum movimento de maior amplitude, por motivos de segurança. Assim sendo, o volume de trabalho será

definido em função de algumas constantes que podem ser facilmente modificadas pelo operador, de modo a atender as necessidades reais do usuário final.

Estas constantes serão dadas em função dos próprios parâmetros de acionamento do robô:

$$\theta_{min} \leq \theta \leq \theta_{max}$$

$$\alpha_{min} \leq \alpha \leq \alpha_{max}$$

$$r_{min} \leq r \leq r_{max}$$

Uma vez definidos estes valores, o robô não atingirá posições além destas, mesmo que fisicamente permitidas. Estas são definições que portanto serão incorporadas na solução da cinemática inversa.

5.6 Soluções Múltiplas

Considerando as equações da cinemática inversa e a definição de volume de trabalho nos itens anteriores, já temos a capacidade de verificar se uma determinada posição desejada pode ser alcançada pelo efetuador, e obter ao menos uma relação entre a posição angular final do motor em função de uma posição fornecida em coordenadas cartesianas.

Como também é de conhecimento, a solução da Cinemática Inversa recai em equações simultâneas, não lineares, com funções trigonométricas. Isto significa que existe a possibilidade de que estas equações possuam soluções múltiplas. Este fato pode causar problemas porque o sistema deve estar apto a escolher uma das soluções. O critério no qual baseia-se a decisão pode variar. Por exemplo, uma boa opção é escolher como a melhor solução aquela que é a mais próxima da configuração atual, minimizando o deslocamento que cada junta será requisitada a se mover. Assim, este procedimento teria também como parâmetro de entrada a posição atual do manipulador.

A presença de obstáculos no caminho de qualquer uma das articulações também pode ser um problema a ser considerado na escolha de uma solução entre várias soluções possíveis.

De fato, no caso do robô cirúrgico em questão, sabe-se que este problema matemático ocorre. No entanto, verificou-se que apenas uma solução física é possível, podendo-se descartar as demais soluções matemáticas não convenientes.

Como um breve exemplo deste problema, para realizar o posicionamento do efetuador em um ponto qualquer contido no Volume de Trabalho, surge além da solução trivial que pode ser imaginada, a solução em que o Motor 3 deveria girar o ângulo θ de 180°, invertendo completamente o robô, seguido então pela retração do Fuso de Esferas de um comprimento negativo, comandada pelo Motor 1.

5.7 Análises decorrentes do Jacobiano

5.7.1 Pontos de Singularidade do Mecanismo

Uma das formas de interpretar-se o Jacobiano obtido anteriormente é como este sendo uma transformação linear, para um determinado instante, que relaciona a velocidade das juntas com a Velocidade Cartesiana da ponta do braço.

$$V = J(\theta).\dot{\theta}$$

No entanto, a maioria dos manipuladores possui posições de singularidade do mecanismo, ou seja, posições angulares onde o Jacobiano torna-se nulo. Todos os manipuladores possuem singularidades nos limites do Volume de Trabalho, e em algumas vezes, em alguns outros pontos internos bem localizados. As singularidades costumam ser classificadas em duas categorias:

- Singularidade no Contorno do Volume de Controle, na qual ocorre quando um mecanismo é completamente alongado ou dobrado sobre si próprio, atingindo uma posição próxima ou exatamente sobre num ponto extremo do Volume de Trabalho;
- Singularidade no interior do Volume de Controle, na qual duas ou mais juntas alinham-se, não necessariamente nas bordas do Volume de Trabalho.

Quando um manipulador está em uma posição singular, isto significa que do ponto de vista do espaço Cartesiano, o mecanismo perdeu um ou mais graus de liberdade, ou seja, há uma direção na qual o mecanismo não consegue se mover, independentemente do modo como se acionam as juntas.

De posse do Jacobiano, procurou-se então calcular as posições nas quais o determinante tornava-se nulo. O determinante do Jacobiano é dado pela seguinte expressão:

$$\det(J) = -2s_2 d_3^{\,2} c_2^{\,2} + 2c_1^{\,2} s_2 d_3^{\,2} c_2^{\,2} + s_2 d_3^{\,2} - 2d_3^{\,2} c_1^{\,2} s_2$$

Percebe-se a partir desta expressão que quando d_3 é nulo, o determinante também vai a zero, pois todos os termos dependem de d_3. Este caso acontece fisicamente quando o efetuador está totalmente recolhido, ou seja, este encontra-se posicionado sobre o centro de coordenadas simplificado. Nesta posição, para quaisquer valores de atuação dos motores 2 e 3, o ponto permanece sem deslocamento algum. Pode-se afirmar ainda que há infinitos modos de orientar o efetuador nesta posição.

Outro modo de anular esta expressão é quando o ângulo θ_2 vale 0°, ficando nulo o seno deste ângulo, que também aparece em todos os termos da expressão. Neste caso, fisicamente, o motor 2 estaria posicionando o efetuador sobre o eixo z_0, e neste caso, variando o ângulo θ_1, novamente o atuador permaneceria na mesma posição.

Concluindo este assunto, vale lembrar que devido às limitações geométricas do projeto mecânico, o caso 2, quando θ_2 vale 0°, não ocorre. Já o primeiro caso, este sim causa algumas limitações na operação do robô. Este problema será tratado junto à implementação dos algoritmos desenvolvidos.

5.7.2 Cinemática Inversa da Velocidade

Tomando-se a equação

$$V = J(\theta).\dot{\theta}$$

percebe-se que a partir desta, poderíamos desejar impor uma determinada velocidade em coordenadas cartesianas ao efetuador, e então determinar com quais velocidades angulares deveriam ser acionados os respectivos motores. Lembrando-se que uma matriz é inversível quando seu determinante é diferente de nulo, e garantindo esta condição, bastaria fazer:

$$\dot{\theta} = J^{-1}(\theta).V$$

Esta expressão foi obtida e simplificada com o auxílio de ferramentas de manipulação simbólica, e é apresentada aqui apenas a título ilustrativo.

$$\dot{\theta} = \begin{bmatrix} 2s_1 v_x c_2^{\,2} - s_1 v_x + c_1 v_y + \dfrac{2c_1 c_2 s_1 s_2 v_z}{-s_2 d_3\left(-2c_2^{\,2} + 2c_1^{\,2} c_2^{\,2} + 1 - 2c_1^{\,2}\right)} \\ -c_1 c_2 v_x + s_1 c_2 v_y - 2s_2 v_z c_1^{\,2} + \dfrac{s_2 v_z}{d_3\left(-2c_2^{\,2} + 2c_1^{\,2} c_2^{\,2} + 1 - 2c_1^{\,2}\right)} \\ c_1 s_2 v_x + s_1 s_2 v_y + 2s_2 v_z^{\,2} + \dfrac{c_2 v_z}{\left(-2c_2^{\,2} + 2c_1^{\,2} c_2^{\,2} + 1 - 2c_1^{\,2}\right)} \end{bmatrix}$$

5.7.3 Cálculo de esforços nas articulações

Um estudo que surge naturalmente no projeto de um robô é determinar como as forças e momentos se propagam de uma junta a outra. Tipicamente o robô estará empurrando ou segurando alguma carga. Considerando estes esforços externos como conhecidos, desejamos então determinar os torques necessários em cada articulação do manipulador, de modo a manter o sistema em equilíbrio estático.

Ao considerar forças estáticas em um manipulador, primeiramente travam-se as juntas de modo que o manipulador torne-se uma estrutura. Então se considera cada link e escrevem-se as relações entre força e momento.

A partir do princípio dos trabalhos virtuais, é possível provar que o procedimento acima é equivalente a fazer:

$$\tau = {}^0J^T . {}^0F$$

onde o Jacobiano possui agora a função de transformar forças agindo no sistema cartesiano em torques nas juntas correspondentes. Deve-se apenas lembrar que ${}^0F$ é o vetor de forças expresso em coordenadas relativas a base O_0.

A expressão acima foi desenvolvida tomando-se um vetor força F decomposto em suas componentes F_x, F_y, F_z, e o resultado são os apresentados abaixo.

$$\tau = \begin{bmatrix} -s_1 s_2 d_3 F_x - c_1 s_2 d_3 F_y \\ c_1 c_2 d_3 F_x - s_1 c_2 d_3 F_y + s_2 d_3 F_z \\ c_1 s_2 F_x - s_1 s_2 F_y + c_2 F_z \end{bmatrix}$$

A interpretação deste resultado leva exatamente ao que é esperado pela intuição. Tomando por exemplo o torque que surge no motor 1, este não é influenciado pela componente na direção z, e sim pelas demais componentes, tendo

como braço da força as projeções de d_3. O terceiro termo da matriz é o efeito no atuador 1, que é a guia, e neste caso, o resultado tem dimensão de força.

Deve-se apenas fazer uma comparação com o Torque calculado no início deste projeto. A diferença entre este e o anterior é que o primeiro havia sido calculado com forças expressas em relação ao sistema de coordenadas no efetuador, e assim, era um esforço diferente do utilizado agora.

6 PLANEJAMENTO E GERAÇÃO DE TRAJETÓRIA

O controle das articulações de um robô manipulador de forma coordenada, com o objetivo de realizar uma tarefa pré-definida, exige um planejamento do movimento (trajetória) do robô. O problema imediato é movimentar o manipulador de uma posição inicial para uma posição final. Estaríamos assim realizando um controle de posição.

No entanto, é geralmente necessário especificar e executar o movimento com mais detalhes do que simplesmente fornecer as posições inicial e final, pois existem infinitas possibilidades de trajetórias que levam o efetuador entre dois pontos dados.

Uma forma de incluir mais detalhes é fornecer uma seqüência de pontos intermediários, ou pontos de passagem, entre as posições inicial e final. Cada ponto de passagem especifica uma posição obrigatória a ser obtida pelo efetuador.

Porém, poderia ser definindo também um outro modo não tão simples. Este deveria ser capaz de realizar o posicionamento do efetuador, partindo de uma posição inicial qualquer para uma posição final desejada, porém agora com conhecimento prévio da trajetória que será descrita pelo efetuador ao longo de todo o caminho.

Em geral, no planejamento de trajetória, podem ser identificados dois métodos: o planejamento em coordenadas de articulação e o planejamento em coordenadas do efetuador. No planejamento de trajetória em coordenadas de articulação fornece-se algumas informações, como a posição e velocidade de alguns pontos de passagem, e assim as articulações devem ser acionadas de modo a cumprir tais condições, não importando a trajetória entre estes pontos de passagem. A vantagem deste método é a facilidade da implementação, podendo ser inclusive realizada em tempo real.

No caso de planejamento utilizando as coordenadas do efetuador, especifica-se explicitamente a trajetória que o efetuador deverá percorrer. Por exemplo, pode-se especificar a trajetória através de uma função analítica. Assim, torna-se necessário realizar o cálculo da cinemática inversa a todo instante, de modo a garantir a trajetória desejada.

Na aplicação prática deste robô, acredita-se que ambos os métodos poderiam ser aplicados, garantindo se algumas condições. Uma vantagem do planejamento em coordenadas do efetuador é sem dúvida a necessidade de movimentar os instrumentos cirúrgicos no interior do paciente com o máximo de segurança. Isto se traduz no desejo de movimentar tais instrumentos entre dois pontos com um perfeito conhecimento da trajetória, de modo a evitar surpresas. Neste contexto, a reta seria uma forma bem efetiva de solucionar o problema acima mencionado. A trajetória retilínea é a mais simples conhecida, apesar de não ser necessariamente fácil a sua implementação. A reta é ainda uma função que daria um bom resultado na tarefa de realizar a interpolação da trajetória entre uma seqüência de pontos sucessivos.

Por outro lado, o planejamento em coordenadas da articulação não deixaria a desejar a medida em que os deslocamentos entre dois pontos sucessivos tornarem-se menores. Isso possivelmente acontecerá, dependendo da forma de como será construída a interface de captura de dados para o acionamento deste robô.

Não é desejo neste trabalho de decidir a respeito do melhor modo de acionamento, mas sim desenvolvê-los para a utilização futura, conforme as necessidades específicas de cada tarefa que o robô deverá cumprir em seu trabalho. Assim sendo, decidiu-se por realizar o estudo das duas formas de planejamento de trajetórias.

6.1 Planejamento de Trajetória em Coordenadas da Articulação

6.1.1 Acionamento Sucessivo dos Três Motores

Uma primeira forma de realizar o acionamento utiliza, os resultados da Cinemática Inversa. Este método recebe um ponto em coordenadas cartesianas e um valor percentual para a velocidade. Após verificar a possibilidade de alcançar este ponto (deve estar dentro do volume de trabalho), o robô posiciona o efetuador no local desejado, partindo do ponto em que ele se encontrava no instante anterior.

Este posicionamento ocorre acionando-se cada um dos motores na seqüência: θ, α, e r. Em relação aos efeitos no atuador, o valor da velocidade angular dos motores é constante durante todo o percurso. Como exigência, acaba sendo

necessário que esta velocidade seja tal que o motor possua um torque de partida suficiente para mover o manipulador. Caso contrário, ocorreria a perda de passos.

6.1.2 Acionamento Sucessivo dos Três Motores com Curva de Aceleração

Este método é uma modificação do método anterior. Após o cálculo da cinemática inversa e da determinação do número de passos que cada motor deve executar, estes são acionados seqüencialmente, porém segundo uma curva de aceleração.

O motor parte do repouso com uma aceleração pequena, até atingir a velocidade de trabalho, que no caso foi feita como sendo a máxima velocidade possível para cada motor. Assim, este método possui a vantagem sobre o método anterior de ser praticamente impossível a perda de passos devido a movimentos bruscos na partida e na parada.

6.1.3 Acionamento Simultâneo dos Três Motores

Uma terceira forma de realizar o acionamento dos motores é semelhante ao primeiro modo. Porém a diferença é que os três motores são acionados simultaneamente, de modo a iniciarem e terminarem seus movimentos nos mesmos instantes. Neste caso, a trajetória obtida não é conhecida, mas a medida em que a distância entre os pontos inicial e final diminuem, a trajetória tende a uma reta, com erro conhecido.

6.2 Planejamento de Trajetória em Coordenadas do Efetuador

6.2.1 Acionamento de forma simultânea através de uma trajetória retilínea

Uma última forma de realizarmos o acionamento dos motores é acioná-los de forma simultânea, assim como no método anterior, porém, controlando sua trajetória através de uma reta, ou seja, impondo uma trajetória retilínea entre as posições inicial e final do efetuador.

Neste caso, foi feito como sendo mais um parâmetro de entrada, o tempo total do movimento. Como a trajetória é uma reta, o acionamento dos motores não é contínuo. Realiza-se o cálculo da cinemática inversa em vários instantes da trajetória, e a cada instante, é necessário para o correto funcionamento deste modo que os motores de passo tenham a capacidade de serem acionados de um modo a trabalhar em condições de parada e partida, ou seja, surgem mudanças de direção no sentido de rotação do motor. Na verdade, a mudança de direção ocorrerá com freqüência no motor 1, que aciona o fuso de esferas. Em geral, as rotações independentes dos motores 2 e 3 geram um arco de circunferência, e a transformação deste arco em uma reta acaba solicitando o motor 1 de forma a ser retraída no início do movimento, e ser alongada na segunda metade do movimento, passando portanto por uma inversão de sentido.

Deve-se alertar também sobre o comportamento neste método em pontos próximos ao ponto de singularidade determinado anteriormente. Quando o efetuador é solicitado a movimentar-se em trajetória reta entre dois pontos, sendo um destes o ponto de origem do sistema de coordenas adotado, numa região bem próxima a este, o algoritmo desenvolvido acaba exigindo acelerações muito grandes do motor. Este efeito já era esperado por ser comum na prática.

A explicação deste módulo de aceleração elevado é que quando se deseja mover o atuador, mesmo que de uma pequena distância e a baixas velocidades em uma região próxima ao ponto de singularidade, os atuadores 2 e 3 são solicitados a dar uma grande quantidade de passos, sendo que este grande número de passos acaba por não provocar deslocamentos.

7 IMPLEMENTAÇÃO COMPUTACIONAL

As implementações computacionais dos métodos descritos no capítulo anterior necessitam da utilização dos cálculos da cinemática direta, da cinemática inversa e das relações de transmissão descritas nos itens anteriores (Capítulo 5).

A implementação computacional desses cálculos pode ser feita de forma direta, utilizando-se as funções matemáticas disponíveis na linguagem C, ou seja, as funções da biblioteca "math.h". Além desses cálculos, como todos os atuadores são motores de passo devemos calcular o número de passos a serem dados por cada motor para se realizar o movimento desejado.

Além disso, existem aspectos particulares sobre cada método implementado. Os aspectos relevantes da implementação computacional serão descritos a seguir. Para se ter uma visão geral da implementação veja o código fonte apresentado no Anexo A.

7.1 Visão geral da implementação

Todos os métodos de acionamento descritos no Capítulo 6, com exceção do método de acionamento independentes via teclado, utilizam a cinemática inversa para obterem as coordenadas de junta para a qual o robô deve ser levado para se atingir a posição desejada do efetuador.

Além disso, no término da movimentação é realizado o cálculo da cinemática direta para se obter a posição na qual o robô terminou seu movimento. Isto é realizado, devido ao fato de trabalharmos com motores de passo, os quais apresentam movimentação quantizada e, portanto, podem não assumir exatamente a posição desejada. Se assumíssemos que o robô tivesse alcançado a posição desejada, poderíamos estar acumulando erro de posicionamento que com o passar do tempo de tornariam muito grandes. Para exemplificar esta situação imaginemos o caso em que se deseja movimenta o efetuador da posição $p_0 = [0 \quad 0 \quad 60]$ para a posição $p_{final} = [20 \quad 20 \quad 60]$. Agora imaginemos que devido a resolução imposta pelos motores de passo o efetuador só consiga atingir o ponto $p'_{final} = [19.98 \quad 19.99 \quad 59.99]$. Neste caso, se a posição tivesse sido atualizada de

acordo com 1 com os valores de p_{final}, estaria sendo acumulado um erro de da ordem de: $erro = [0.02 \quad 0.01 \quad 0.01]$ somente neste posicionamento. Com o passar do tempo estes erros iriam se acumulando.

Cada um dos métodos de acionamento será descrito nos itens seguintes.

7.2 Acionamento independente pelo Teclado

A primeira forma e a mais simples de atuação dos motores foi simplesmente o acionamento dos mesmos de forma independente, de modo a controlar cada um dos graus de liberdade do robô separadamente. Esta forma de acionamento será útil mais adiante em situações fora de trabalho, em calibrações, etc. Os atuadores neste caso são acionados através do teclado, quando se pressiona as teclas 'a', 's', 'z', 'x', 'q' e 'w'.

Vale a pena ressaltar aqui que este modo de movimentação pode ser utilizado para fazer a calibração do intertravamento de posição que controla o volume de trabalho do robô. Assim, somente através deste modo de acionamento é possível ultrapassar os limites do volume de trabalho que estiverem em uso no *software* de controle.

7.3 Implementações comuns a todos os métodos

Todos os métodos em análise recebem como um dos parâmetros a posição do efetuador em coordenadas cartesianas da base para a qual ele deve se mover. Deste modo é necessário transformar o valor desta posição em número de passos necessários em cada motor para se executar o deslocamento desejado.

Este cálculo se inicia com o cálculo da cinemática inversa para a posição final do efetuador. A seguir, como a posição atual do robô é conhecida, é possível calcular os deslocamentos em coordenadas de junta necessários para a realização do movimento.

Neste ponto, falta apenas utilizarmos as informações sobre as relações de transmissão (apresentada na seção 5.4), o número de passos por volta e o tipo de acionamento, fullstep ou halfstep de cada motor e desta forma, calcular o número de

passos necessário em cada motor. Os motores em questão possuem 200 passos por volta e estão sendo acionados em fullstep.

7.4 Implementação do método de acionamento seqüencial

O método de acionamento seqüencial (sem curva de aceleração) recebe como parâmetros à posição em coordenadas cartesianas da base para qual o efetuador deve se mover e a velocidade em percentual da velocidade máxima possível de se alcançar com cada um dos três motores, sem que nenhum delas perca passo.

O número de passos é calculado através do método da seção 7.3. A velocidade de acionamento é calculada como sendo uma fração da velocidade máxima possível de atingir com cada um dos atuadores. Esta velocidade é então convertida em freqüência de passos que são gerados pelo PC e fornecido aos motores através da porta paralela.

Neste método, o acionamento dos motores é realizado de forma independente e seqüencialmente, possuindo a seguinte ordem de acionamento: motor θ, motor α e finalmente motor r.

7.5 Implementação do método de acionamento seqüencial com curva de aceleração

O acionamento do motor seqüencialmente com curva de aceleração recebe como parâmetros apenas a posição desejada para o efetuador. Como velocidade de trabalho, ela utiliza a velocidade máxima possível de acionamento de cada motor. Este parâmetro foi ajustado a partir de testes de desempenho. Outro parâmetro também ajustado experimentalmente é a taxa de aceleração, que é dada em passos para atingir a velocidade máxima.

Uma vez de posse do ponto inicial e dos parâmetros de velocidade, esta rotina utiliza as funções básicas para calcular o número de passos que deve ser dado, e então aciona cada um dos três motores seqüencialmente. Durante o acionamento de um motor, este possui três fases distintas, que são o período de aceleração, o período de trabalho em velocidade máxima e o período de desaceleração.

Terminado o processo, atualiza-se a posição final do atuador.

7.6 Acionamento simultâneo

O método de acionamento simultâneo dos motores recebe como parâmetros à posição em coordenadas cartesianas da base para qual o efetuador deve se mover e o tempo total para a realização da movimentação.

O software calcula o número de passos necessários para a execução dos deslocamentos pelo método descrito na seção 7.3 .

A diferença aqui consiste na forma de entrada do parâmetro de velocidade e na forma de acionamento dos motores, que ao invés de independente e seqüencial, é realizado de forma simultânea, ou seja, todos os motores iniciam seus movimentos conjuntamente e terminam seus movimentos no mesmo instante.

A realização deste sincronismo é realizada da seguinte forma: com o tempo total para o deslocamento e o número de passos necessários a cada motor pode-se calcular o período do trem de pulsos a ser enviado a cada motor. A seguir é iniciada uma contagem através de uma função que lê o timer do PC e a cada loop do algoritmo verifica-se o tempo decorrido a partir do início da contagem e a necessidade ou não de se alterar o estado de uma das saídas, ou seja, a necessidade de se chavear o pulso para um dos motores. Desta forma é possível controlar os três motores simultaneamente, mesmo quando eles possuem velocidades de rotação distintas entre si.

7.7 Acionamento simultâneo com trajetória retilínea

O método de acionamento simultâneo com trajetória retilínea é idêntico ao método de acionamento anterior, exceto no fato que ele também controla a trajetória durante o deslocamento. Para efetuar este controle de trajetória, o deslocamento entre o ponto inicial e o ponto final é dividido em várias etapas através de uma interpolação linear. Uma vez obtidos estes pontos, pode-se chamar o método de acionamento simultâneo para pequenos deslocamentos, forçando o efetuador a passar por cada um destes pontos e, desta forma, a descrever uma trajetória retilínea.

7.8 Outras funções

Para que fosse possível implementar um sistema de controle em malha aberta através do controle de motores de passo, que são sistemas incrementais foi necessário incorporar ao sistema variáveis que armazenam a posição atual do efetuar. Para isto foi declarada uma variável global para o armazenamento da posição atual. Esta posição inicial é ajustada como sendo o ponto $p=[0\ \ 0\ \ 0]^t$, independentemente da posição física do efetuador. Assim devem estar disponibilizados métodos para a calibração do zero e para o ajuste do volume de trabalho.

7.8.1 Função de calibração da origem do sistema de coordenadas

Para a execução da calibração da origem do sistema de coordenadas existe o método de acionamento independente via teclado. Assim pode-se movimentar o efetuador até a posição $[0\ 0\ 0]^t$ e então através de uma função disponibilizada pelo software ajustar a origem do sistema de coordenadas, como sendo a posição ocupada pelo efetuador naquele instante.

7.8.2 Função de ajuste dos limites do volume de trabalho

Para que fosse possível o ajuste do volume de trabalho diferente do ajuste *default* do software, foi disponibilizada uma função que permite movimentar independentemente os atuadores via teclado, levando o efetuador até uma posição limite desejada, onde então, através de outra função pode-se ajustar a posição do efetuador naquele instante como sendo sua posição limite. Para que este limite seja válido em todas as direções, pega-se as coordenadas de junta do robô naquele instante e constrói-se um ângulo sólido que possui seu limite passando por aquele ponto e tem vértice na origem.

7.9 Relação entre pulsos no Controlador e Direção de Movimento

O comando em linguagem C utilizado para enviar um pulso para a porta paralela do computador é o "outp (arg1, arg2)". Arg1 representa o endereço da porta

paralela em hexadecimal, e arg2 é um número inteiro de 0 a 255, representando oito bits utilizados para montar um byte.

A tabela que contém explicações sobre estes bits já foi apresentada anteriormente no tópico específico sobre a porta paralela. Convém agora relacionar o sentido que o motor da um passo com o deslocamento do efetuador no espaço. Estas informações são fornecidas na tabela a seguir.

Motor	Sinal enviado ao Driver	Sentido do deslocamento do efetuador	NPassos calculados no programa
θ	1	+ y	+
θ	0	- y	-
α	1	+ x	+
α	0	- y	-
R	1	- z	+
R	0	+ z	-

Tabela 6 - Relação entre o sentido do passo e o deslocamento do Efetuador

8 SUGESTÕES PARA TRABALHOS FUTUROS

Durante o desenvolvimento do presente projeto foram percebidos vários fatores que aperfeiçoariam e melhorariam o sistema aqui implementado. Estes fatores são aqui apresentados sob a forma de sugestões para desenvolvimentos futuros. São eles:

- Integração do braço com o mouse espacial: devem ser estudadas formas de se realizar a captura de dados pelo mouse espacial e então a passagem destes dados criteriosamente ao robô cirúrgico. Criteriosamente, pois se deve cuidar dos ruídos que este apresenta, evitando um acidente bastante provável. Há muitas análises a serem feitas em relação a esta implementação. Por exemplo, se deve ser realizado em tempo real ou não. Pelas análises observadas até presente momento é difícil de se implementar um sistema em tempo real, pois o robô talvez não consiga se mover na mesma velocidade do mouse. Ainda em relação à conexão com o mouse espacial, uma sugestão seria a de implementar um botão na ponta do mouse, de modo que somente quando o médico o apertasse, a leitura da posição fosse realizada e conseqüentemente o braço acionado. Ainda pensando em tempo real, vêm muitos pontos dependendo da taxa de aquisição, como por exemplo, devido ao critério de Nyquist a taxa de aquisição possui um limite inferior, e desta forma o mouse deve gerar uma seqüência de pontos para movimentação mais rápida do que o sistema pode suportar. Desta forma, deve ser pesquisada a melhor solução para contornar este problema. Algumas soluções pensadas até o momento seriam implementar uma fila dinâmica armazenando os pontos ou então utilizar um número menor de pontos dentre os fornecidos pelo mouse;
- Ainda considerando a conexão com o mouse escolher qual dos métodos de acionamento implementados no presente trabalho que melhor se adeque para realizar o acionamento dos motores;
- Uma vez disponibilizado um mouse com capacidade de realimentação de força, o projeto pode fazer uso deste fato e então utilizar sensores para a o sensoriamento das forças que agem na ponta do efetuador;
- Como dito na seção 4.1.1, a melhor solução para o acionamento do fórceps seria a utilização de servomotores, porém, como já explicado, devido à falta de

disponibilidade destes motores, o acionamento do fórceps foi implementado com motores de passo. Desta forma, sugere-se a substituição destes motores de passo por servomotores.

9 Bibliografia

[1] KENJO, Takahashi e SUGAWARA, Akira - Stepping Motors and their micoprocessor controls. Ed. Oxford, 1994.

[2] ASADA, H. e SLOTINE, J. J. E. - Robot Analysis and Control. Ed. Jonh Wiley and Sons, Nova York, 1986.

[3] CRAIG, J.J – Introduction to Robotics Mechanics and Control. Ed. Addison-Wesley, 1989.

[4] SZAFIR, Silvio – Sistema baseado em processador digital de sinais para controle em tempo real de motor elétrico – Dissertação de mestrado, EPUSP, São Paulo, 2000.

Anexo A

Código fonte do software implementado

```
////////////////////////////////////////////////////////
//                    INCLUDES                        //
////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "Transf~1.h"
#include "Motores.h"
#include "timer.h"
#include "mouse.h"

////////////////////////////////////////////////////////
//          VARIAVEIS GLOBAIS                         //
////////////////////////////////////////////////////////

CART PosicaoAtual;              //Armazena Posicao Atual em Coord Cartesianas
RCOORD Posicao;                 //Armazena Posicao Atual em Coord de Junta
RCOORD DeslocporPasso;          //Desloc por Passo do motor em Coord de junta
NPASSOS nPassosDados;                   //Armazena nro de passos dados apos o inicio do
programa
double PosicaoFusoAtual;        //Armazena a posicao atual do fuso alfa

double MAX_POS_TETA;
double MAX_POS_ALFA;            // Armazena definicoes do Volume de Trabalho das articulacoes
double MAX_POS_R;


////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                 //
// Funcao:    main                                                                                 //
//                                                                                                 //
// Prososito:  Pega os parametros, inicializa o sistema e inicia                                   //
//                                  o processamento do deslocament                                 //
// Parametros:     nenhum                                                                          //
//                                                                                                 //
// Retornos:       nenhum                                                                          //
//                                                                                                 //
////////////////////////////////////////////////////////////////////////////////////////////////////
void main()
{
        CART Destino;       // Recebe os pontos de destino em coord Cart
        char aux;           // Variavel auxiliar para repeticao
        double TempoMovimento;          // Recebe o tempo em ms para movimentacao
        int funcao;                             // Int que recebe linha de comando
        double Angulo;                  // Recebe o angulo p/ controle de orientacao
        int i;
        RCOORD VelMaxima;       // Velocidade maxima de rotacao dos motores em
                                                        // passos por segundo
```

```
        inicializaTimer();     //inicializa o timer

//////////////////////////
//          INICIALIZACOES                  //
//////////////////////////

        aux = 'S';                                  //inicializa aux para sim (para efetuar 1o
movimento)

        Posicao.teta = 0;
        Posicao.alfa = 0;           //posicao atual em coord de junta
        Posicao.r = 0;

        nPassosDados.npTeta = 0;
        nPassosDados.npAlfa = 0;//nro de passos dados
        nPassosDados.npR = 0;

        PosicaoAtual.x = 0;
        PosicaoAtual.y = 0;                 //posicao atual em coord cartesianas
        PosicaoAtual.z = 0;

        PosicaoFusoAtual = 0;  // posicao atual do fuso alfa

        AnguloAtual = 0;            //Angulo atual do efetuador


// Inicializa Volume de Trabalho das articulacoes

        MAX_POS_TETA = 1;
        MAX_POS_ALFA = 1;
        MAX_POS_R = 120;

// Inicializa os parametros relativos aos motores

        SetaParamMotores();

// Inicializa Velocidade Maxima dos Motores em passos por segundo

        VelMaxima.alfa = 100;  //  passos por segundo
        VelMaxima.teta = 10;
        VelMaxima.r = 300;

        do
        {
                // Mostrar tela padrão
                TelaPrincipal();
                aux = getch();

                switch (aux){

                case '0':  // Encerra Programa

                        break;


                case '1' :    // Movimento atraves do teclado
```

```
                    TelaTeclado();

                    AcionamTeclado(); // Funcao que aciona motor a partir do teclado
                                      // Atualiza a posicao do Robo
                    break ;


                case '2' :  // Definicao do centro de coordenadas
                        // Basta inicializar as variaveis na posicao desejada


                            Posicao.teta = 0;
                            Posicao.alfa = 0;          //posicao atual em coord de junta
                            Posicao.r = 0;

                            nPassosDados.npTeta = 0;
                            nPassosDados.npAlfa = 0;//nro de passos dados
                            nPassosDados.npR = 0;

                            PosicaoAtual.x = 0;
                            PosicaoAtual.y = 0;              //posicao atual em coord
cartesianas
                            PosicaoAtual.z = 0;

                            PosicaoFusoAtual = 0;  // posicao atual do fuso alfa
                            AnguloAtual = 0;


                        break;

                case '3' :  // Acionamento Sequencial


                TelaEntrada(3);

                // Pega pontos de destino

                _displaycursor( _GCURSORON );
                _settextposition(11,33);
                scanf("%lf", &Destino.x);
                _settextposition(12,33);
                scanf("%lf", &Destino.y);
                _settextposition(13,33);
                scanf("%lf", &Destino.z);

                // Pega Velocidade em Valor Percentual
                _settextposition(14,44);
                scanf("%lf", &TempoMovimento);

                MovimentaRobo2(Destino, TempoMovimento, VelMaxima); // funcao de
acionamento dos 3 motores sequencialmente

                        break;

                case '4' :

        TelaEntrada(4);
```

```
                _displaycursor( _GCURSORON );
                _settextposition(11,33);
                scanf("%lf", &Destino.x);
                _settextposition(12,33);
                scanf("%lf", &Destino.y);
                _settextposition(13,33);
                scanf("%lf", &Destino.z);

                _settextposition(14,44);
                scanf("%lf", &TempoMovimento);

                MovimentaRobo3(Destino, TempoMovimento); //funcao que movimenta o robo


                        break;

                case '5' :

                TelaEntrada(5);

                // Pega pontos de destino

                _displaycursor( _GCURSORON );
                _settextposition(11,33);
                scanf("%lf", &Destino.x);
                _settextposition(12,33);
                scanf("%lf", &Destino.y);
                _settextposition(13,33);
                scanf("%lf", &Destino.z);


                MovimentaRobo4(Destino, VelMaxima); // funcao de acionamento dos 3 motores
sequencialmente

                break;

                case '6' :

                TelaEntrada(6);

                _displaycursor( _GCURSORON );
                _settextposition(11,33);
                scanf("%lf", &Destino.x);
                _settextposition(12,33);
                scanf("%lf", &Destino.y);
                _settextposition(13,33);
                scanf("%lf", &Destino.z);

                _settextposition(14,44);
                scanf("%lf", &TempoMovimento);

                MovimentaRobo(Destino, TempoMovimento); //funcao que movimenta o robo


                break;

                case '7' :
```

```
                                for(i=0; i<40 ; i=i+1 )
                                        {
                                        Destino.x = 12*sin((2*3.14/20)*i  );
                                        // Destino.y = i;
                                        Destino.z = 80 + 12*cos((2*3.14/20)*i  );
                                        TempoMovimento = 600;
                                        MovimentaRobo(Destino, TempoMovimento);
                                        }

                        break;


                case '9' :
                 TelaEntrada(9);
                _displaycursor( _GCURSORON );
                _settextposition(11,43);
                scanf("%lf", &Angulo);
                MoveEfetuador(Angulo);

                break;

                case 'B' :  // Pega posicao atual como Limite do Volume de trabalho
                case 'b' :

                        MAX_POS_TETA = fabs(Posicao.teta);              // Inicializa padrao do
 Volume de Trabalho das articulacoes
                        MAX_POS_ALFA = fabs(Posicao.alfa);
                        MAX_POS_R = fabs(Posicao.r);

                        TelaVolume();
                        aux = getch();
                        break;

                }

        }
        while (aux != '0');

        nPassosDados.npTeta = -nPassosDados.npTeta;
        nPassosDados.npAlfa = -nPassosDados.npAlfa; //calcula passos necessarios para voltar a
posicao inicial
        nPassosDados.npR = -nPassosDados.npR;

        TelaFinal();
        VoltaZero(VelMaxima);  // volta para a posicao atual
        _clearscreen(_GCLEARSCREEN);
        recuperaTimer();  //recupera o Timer

}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:              Calcula Deslocamentos                                                                     //
//                                                                                                                //
// Prosposito:      Calcular os deslocamento em coord de junta
        //
//                                          necessario para movimentar o robo da posicao                          //
//                                          atual para o destino                                                  //
//                                                                                                                //
// Parametros:  (entrada)       Destino em coord de junta                                                         //
//                              (saida)         Ponteiro para variavel que sera                                   //
//                      preenchida com os deslocamentos necessarios em coord                                      //
//                      de junta                                                                                  //
//                                                                                                                //
// Retornos:        nenhum                                                                                        //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void CalculaDeslocamento(RCOORD Destino, LPRCOORD DeslocRobo)
{
        DeslocRobo->alfa = Destino.alfa - Posicao.alfa;
        DeslocRobo->teta = Destino.teta - Posicao.teta;       // calcula os deslocamentos
        DeslocRobo->r = Destino.r - Posicao.r;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:              TransformaCoordenada                                                                      //
//                                                                                                                //
// Prosposito:      Transforma um ponto descrito em coordenadas                                                   //
//                                      cartesianas para coordenadas do robo (de junta)                           //
//                                                                                                                //
// Parametros:  (entrada) Ponto em Coordenada Cartesiana                                                          //
//                              (saida)    Ponteiro para variavel que sera prenchida                              //
//                              com a descricao do ponto em coordenadas do robo                                   //
//                                                                                                                //
// Retornos:        0       Sucesso                                                                               //
//                  -1      Ponto fora do volume de controle                                                      //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int TransformaCoordenada(CART Cart, LPRCOORD Robo)
{
        if (Cart.z < 0)             // se z negativo pto fora do volume
        return -1;                  // de controle

//////////////////////////////////////////////////////
// TRANSFORMA AS COORDENADAS //
//////////////////////////////////////////////////////

        if(Cart.y == 0 && Cart.z == 0)              // verifica deslocamentos para determinar
                Robo->teta = 0;                     // se desloc = 0    ou se é obtido com a
        else                                        // funcao atan2, pois atan2(0,0) gera erro
                Robo ->teta = atan2(Cart.y , Cart.z);

        if(Cart.x == 0 && Cart.z == 0)              // verifica deslocamentos para determinar
                Robo->alfa = 0;                     // se desloc = 0    ou se é obtido com a
        else                                        // funcao atan2, pois atan2(0,0) gera erro
                Robo ->alfa = atan2(Cart.x, sqrt(pow(Cart.y, 2) + pow(Cart.z, 2)));
```

```
// calcula a coord do fuso r
        Robo ->r = sqrt(pow(Cart.x, 2) + pow(Cart.y, 2) + pow(Cart.z, 2 ));

        //verifica se esta dentro do volume de controle
        if (Robo->teta > MAX_POS_TETA || Robo->alfa > MAX_POS_ALFA ||
                Robo->r > MAX_POS_R || -Robo->teta > MAX_POS_TETA ||
                -Robo->alfa > MAX_POS_ALFA ||-Robo->r > MAX_POS_R)

                return -1;

        return 0;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                        //
// Funcao:              CalculaNumeroPassos                                                               //
//                                                                                                        //
// Prosposito:      Calcula o numero de passos necessarios para atingir                                   //
//                          o deslocamento desejado                                                       //
//                                                                                                        //
// Parametros:      (entrada) Deslocamento em Coord do ROBO                                               //
//                          (saida)     Ponteiro para variavel que receberá                           //
//                          o numero de       passos                                                    //
//                                                                                                        //
// Retornos:        sempre 1                                                                              //
//                                                                                                        //
////////////////////////////////////////////////////////////////////////////////////////////////////////////
int CalculaNumeroPassos(RCOORD DeslocRobo, LPNPASSOS nPassos)
{
        double Comp_Mais_Desloc;
0
        nPassos->npTeta = (long) (DeslocRobo.teta / DeslocporPasso.teta);
        Comp_Mais_Desloc = sqrt(pow(COMP_A, 2) + pow (COMP_B, 2) - *COMP_A
*COMP_B*
                                                                cos (PI/2  - DeslocRobo.alfa));
        nPassos->npAlfa = (long)((Comp_Mais_Desloc - COMP_C)
        //DeslocporPasso.alfa);
        nPassos->npR =   (long) (-DeslocRobo.r / DeslocporPasso.r);

        return 1;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                        //
// Funcao:              SetaParamMotores                                                                  //
//                                                                                                        //
// Proposito:       Seta os parametros de entrada dos motores                                             //
//                                                                                                        //
// Parametros:      nenhum                                                                                //
//                                                                                                        //
// Retornos:        nenhum                                                                                //
//                                                                                                        //
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void SetaParamMotores()
{
        double halfStep;

        if(HALFSTEP)
```

```
                halfStep = 2;
        else
                halfStep = 1;

// Calcula a resolução do motor de passo

        DeslocporPasso.teta  = 2*PI/(NUMEROPASSOS*halfStep*TRANSMISSAO_TETA);
        DeslocporPasso.alfa = PASSO_FUSO_ALFA/(NUMEROPASSOS*halfStep);
        DeslocporPasso.r = PASSO_FUSO_R/(NUMEROPASSOS*halfStep);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:              AtualizaPosicaoAtual                                                                      //
//                                                                                                                //
// Proposito:   Atualiza as variaveis globais que armazenam as                                                    //
//                              posicoes atuais tanto em coord cartesinas quanto em                               //
//                              coord das juntas                                                                  //
//                                                                                                                //
// Parametros:  (entrada) Numero de passos dados no ultimo
        //
//                                                      deslocamento                                              //
//                                                                                                                //
// Retornos:    nenhum                                                                                            //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void AtualizaPosicaoAtual (NPASSOS nPassos)
{
        double Arg;
        char Buffer[50];

        PosicaoFusoAtual = PosicaoFusoAtual + nPassos.npAlfa *DeslocporPasso.alfa;

        Posicao.teta = Posicao.teta + (nPassos.npTeta*DeslocporPasso.teta);

        Arg = (pow(COMP_A, 2) + pow(COMP_B, 2) - pow(PosicaoFusoAtual + COMP_C, 2) ) /
(2*COMP_A*COMP_B);

        Posicao.alfa  = PI/2 - acos( Arg );

        Posicao.r = Posicao.r - (nPassos.npR*DeslocporPasso.r);

        PosicaoAtual.x = -cos(Posicao.alfa + PI/2)*Posicao.r;
        PosicaoAtual.y = sin(Posicao.teta)*sin(Posicao.alfa + PI/2)*Posicao.r;
        PosicaoAtual.z = cos(Posicao.teta)*sin(Posicao.alfa + PI/2)*Posicao.r;

        _settextposition(26, 33);
        sprintf(Buffer, "x = %04lf mm", PosicaoAtual.x);
        _outtext(Buffer);
        _settextposition(27, 33);
        sprintf(Buffer, "y = %04lf mm", PosicaoAtual.y);
        _outtext(Buffer);
        _settextposition(28, 33);
        sprintf(Buffer, "z = %04lf mm", PosicaoAtual.z);
        _outtext(Buffer);

}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:                  CalculaEtapas                                                                         //
//                                                                                                                //
// Prosposito:      Calcula o numero de etapas necessario para deslocar
        //
//                                  o robo do numero de passos dado por nPassos                                   //
//                                                                                                                //
// Parametros:      (entrada) Numero de passos a ser dado                                                         //
//                  (saida)   Ponteiro para variavel que receberá o nro                                           //
//                                  de etapas                                                                     //
//                                                                                                                //
// Retornos:        nenhum                                                                                        //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void CalculaEtapas(NPASSOS nPassos, long *nSteps)
{
        int Identificador;
        NPASSOS nEtapas;

        // calcula o nuumero de etaps necessário para cada motor
        nEtapas.npTeta = fabs(nPassos.npTeta)/MAX_PASSO_TETA + 1;
        nEtapas.npAlfa = fabs(nPassos.npAlfa)/MAX_PASSO_ALFA + 1;
        nEtapas.npR = fabs(nPassos.npR)/MAX_PASSO_R + 1;

        //seleciona o numero de etapas critico
        Identificador = PegaMaior(nEtapas);

        //atribui o nro de etaps critico a saida
        switch (Identificador)
        {
                case 1:
                        *nSteps = fabs(nEtapas.npTeta);
                   break;
                case 2:
                        *nSteps = fabs(nEtapas.npAlfa);
                   break;
                case 3:
                        *nSteps = fabs(nEtapas.npR);
                   break;
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:                  PegaMaior                                                                             //
//                                                                                                                //
// Prosposito:      Acha o maior valor contido no parametro de entrada                                            //
//                          nEtapas                                                                               //
//                                                                                                                //
// Parametros:      (entrada) nEtapas - struct que contem as etapas                                               //
//                          necessarias para cada motor                                                           //
//                                                                                                                //
// Retornos:        1 - etapas para o motor teta é a maior                                                        //
//                  2 - etapas para o motor alfa é a maior                                                        //
//                  3 - etapas para o motor r é a maior                                                           //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
int PegaMaior (NPASSOS nEtapas)
{
        int aux = 1;
        int maior;

        // assume que nro de etapas de teta e maior
        maior = fabs(nEtapas.npTeta);

        // verifica se nro de etaps de alfa é maior
        if (fabs(nEtapas.npAlfa) > maior)
        {
                maior = fabs(nEtapas.npAlfa);
                aux = 2;
        }

        //verifica se nro de etapas de r e maior
        if (fabs(nEtapas.npR) > maior)
                aux = 3;

        return aux;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                    //
// Funcao:              MovimentaRobô                                                                                 //
//                                                                                                                    //
// Prososito:     Efetua a movimentação do robô em trajetoria                                                         //
//                              retilinea                                                                             //
//                                                                                                                    //
// Parametros:    (entrada)Ponto para o qual o robo deve mover-se em                                                  //
//                                      coordenadas cartesianas                                                       //
//                              (entrada)Tempo no qual o robo deve mover-se em ms
        //
//                                                                                                                    //
// Retornos:      nenhum                                                                                              //
//                                                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MovimentaRobo (CART Destino, double TempoTotal)
{
        int ReturnCode, i;
        RCOORD DeslocRobo, DestinoRobo;
        NPASSOS nPassos;
        CART DestinoAux, PosicaoInicial;
        long nEtapas;
        double TempoPorEtapa;

        PosicaoInicial.x = PosicaoAtual.x;
        PosicaoInicial.y = PosicaoAtual.y;          //guarda a posição de partida em var locais
        PosicaoInicial.z = PosicaoAtual.z;

        // calcula o nro de etapas e verifica se esta no volume de controle
        ReturnCode = PegaEtapas(Destino, &nEtapas);
        if (ReturnCode)
                printf(" Ponto fora do volume de controle!!! \n");
        else
        {
                // calcula o tempo de movimentação por etapa
                TempoPorEtapa = TempoTotal/(double)nEtapas;
```

```
                for(i = 1; i <= nEtapas; i++)
                {
                        // calcula a posição final para esta etapa
                        DestinoAux.x = (Destino.x - PosicaoInicial.x) * ((double)i / (double)
nEtapas) + PosicaoInicial.x;
                        DestinoAux.y = (Destino.y - PosicaoInicial.y) * ((double)i / (double)
nEtapas) + PosicaoInicial.y;
                        DestinoAux.z = (Destino.z - PosicaoInicial.z) * ((double)i / (double)
nEtapas) + PosicaoInicial.z;

                        // calcula a cinemática inversa e verifica volume de controle
                        ReturnCode = TransformaCoordenada(DestinoAux, &DestinoRobo);

                        if (ReturnCode)
                                printf(" Ponto fora do volume de controle!!! \n");
                        else
                        {
                                // calcula o deslocamento em coord de junta
                                CalculaDeslocamento(DestinoRobo, &DeslocRobo);

                                //calcula nro de passos
                                CalculaNumeroPassos(DeslocRobo, &nPassos);

                                // atualiza contador global de passos
                                nPassosDados.npTeta = nPassosDados.npTeta + nPassos.npTeta;
                                nPassosDados.npAlfa = nPassosDados.npAlfa + nPassos.npAlfa;
                                nPassosDados.npR = nPassosDados.npR + nPassos.npR;

                                //aciona os motores
                                AcionaMotores(nPassos, TempoPorEtapa);
                                // realiza a cinemática direta e atualiza a posição atual
                                AtualizaPosicaoAtual(nPassos);
                        }
                }
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:              MovimentaRobô2                                                                            //
//                                                                                                                //
// Prososito:       Efetua a movimentação do robô, acionando os motores                                           //
//                              em sequencia                                                                      //
//                                                                                                                //
// Parametros:      (entrada)Ponto para o qual o robo deve mover-se em                                            //
//                                      coordenadas cartesianas                                                   //
//                  (entrada)Tempo no qual o robo deve mover-se em ms
        //
//                                                                                                                //
// Retornos:        nenhum                                                                                        //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MovimentaRobo2 (CART Destino, double VelPercentual, RCOORD VelMaxima)
{
        int ReturnCode, i;
        RCOORD DeslocRobo, DestinoRobo;
```

```
        NPASSOS nPassos;
        RCOORD Periodo;

        ReturnCode = TransformaCoordenada(Destino, &DestinoRobo); // Entrada cart e saida em
CoordRobo

        if (ReturnCode)
                printf(" Ponto fora do volume de controle!!! \n");
                else
        {
                CalculaDeslocamento(DestinoRobo, &DeslocRobo);
                CalculaNumeroPassos(DeslocRobo, &nPassos);

// Calculo do Periodo da Onda

                Periodo.alfa = (100000 / (2*VelMaxima.alfa * VelPercentual));
                Periodo.teta = (100000 / (2*VelMaxima.teta * VelPercentual));
                Periodo.r = (100000 / (2*VelMaxima.r * VelPercentual));

// Acionamento dos Motores

        if ( nPassos.npTeta > 0 ) {

                outp (0x378 , 127 );        // setar a direcao previamente 01111111
                espera((long)(1000*Periodo.teta));
                for (i=0; i < nPassos.npTeta; i++){

                                outp (0x378 , 125  );      // envia sinal baixo  01111101
                                espera((long)(1000*Periodo.teta));
                                outp (0x378 , 127 );       // envia sinal alto   01111111
                                espera((long)(1000*Periodo.teta));
                }
        }

        if (nPassos.npTeta < 0) {


                outp (0x378 , 126 );       // setar a direcao previamente 01111110
                espera((long)(1000*Periodo.teta));

                for (i=0; i > nPassos.npTeta; i--){
                                outp (0x378 , 124  );      // envia sinal baixo  01111100
                                espera((long)(1000*Periodo.teta));
                                outp (0x378 , 126);        // envia sinal alto   01111110
                                espera((long)(1000*Periodo.teta));


                }
        }


        // Motor Alfa
        if ( nPassos.npAlfa > 0 ) {

                outp (0x378 , 127 );       // setar a direcao previamente 01111111
                espera((long)(1000*Periodo.alfa));

                for (i=0; i < nPassos.npAlfa; i++){
```

```
                outp (0x378 , 119 );      // envia sinal baixo  01110111
                espera((long)(1000*Periodo.alfa));        // espera periodopulso
                outp (0x378 , 127 );      // envia sinal alto   01111111
                espera((long)(1000*Periodo.alfa));        // espera periodopulso
        }
}

if (nPassos.npAlfa < 0) {

        outp (0x378 , 123 );      // setar a direcao previamente 01111011
        espera((long)(1000*Periodo.alfa));        // espera periodopulso

        for (i=0; i > nPassos.npAlfa; i--){

                outp (0x378 , 115 );      // envia sinal baixo  01110011
                espera((long)(1000*Periodo.alfa));        // espera periodopulso
                outp (0x378 , 123 );      // envia sinal alto   01111011
                espera((long)(1000*Periodo.alfa));        // espera periodopulso
        }
}


// Motor Linear

if ( nPassos.npR > 0 ) {

        outp (0x378 , 127 );      // setar a direcao previamente 01111111
        espera((long)(1000*Periodo.r));

        for (i=0; i < nPassos.npR; i++){

                outp (0x378 , 95 );       // envia sinal baixo 01011111
                espera((long)(1000*Periodo.r));
                outp (0x378 , 127 );      // envia sinal alto  01111111
                espera((long)(1000*Periodo.r));

        }
}

if (nPassos.npR < 0) {

        outp (0x378 , 111 );      // setar a direcao previamente 01101111
        espera((long)(1000*Periodo.r));


        for (i=0; i > nPassos.npR; i--){

                outp (0x378 , 79 );       // envia sinal baixo 01001111
                espera((long)(1000*Periodo.r));
                outp (0x378 , 111 );      // envia sinal alto  01101111
                espera((long)(1000*Periodo.r));

        }
}

// Controle do numero de passo dados e posicao alcancada apos movimentacao
```

```
                nPassosDados.npTeta = nPassosDados.npTeta + nPassos.npTeta;  // Atualiza o
controle de passos global
                nPassosDados.npAlfa = nPassosDados.npAlfa + nPassos.npAlfa;
                nPassosDados.npR = nPassosDados.npR + nPassos.npR;

                AtualizaPosicaoAtual(nPassos); // Atualiza a variavel da posicao


        } // fim do else
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:              MovimentaRobô3                                                                            //
//                                                                                                                //
// Prososito:     Efetua a movimentação do robô em trajetoria                                                     //
//                              desconhecida porem 3 motores simultaneamente                                      //
//                                                                                                                //
// Parametros:    (entrada)Ponto para o qual o robo deve mover-se em                                              //
//                                      coordenadas cartesianas                                                   //
//                (entrada)Tempo no qual o robo deve mover-se em ms
        //
//                                                                                                                //
// Retornos:      nenhum                                                                                          //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MovimentaRobo3 (CART Destino, double TempoTotal)
{
        int ReturnCode;
        RCOORD DeslocRobo, DestinoRobo;
        NPASSOS nPassos;

        ReturnCode = TransformaCoordenada(Destino, &DestinoRobo);
        if (ReturnCode)
                printf(" Ponto fora do volume de controle!!! \n");
        else
        {
                CalculaDeslocamento(DestinoRobo, &DeslocRobo);
                CalculaNumeroPassos(DeslocRobo, &nPassos);

                nPassosDados.npTeta = nPassosDados.npTeta + nPassos.npTeta;
                nPassosDados.npAlfa = nPassosDados.npAlfa + nPassos.npAlfa;
                nPassosDados.npR = nPassosDados.npR + nPassos.npR;

                AcionaMotores(nPassos, TempoTotal);
                AtualizaPosicaoAtual(nPassos);
        }


}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:                  PegaEtapas                                                                            //
//                                                                                                                //
// Prosposito:      Calcula o numero de etapas necessarios para o                                                 //
//                                  deslocamento até o pto descrito no parametro                                  //
//                                   de entrada Destino                                                           //
//                                                                                                                //
// Parametros:      (entrada)Ponto para o qual o robo deve mover-se em                                            //
//                                          coordenadas cartesianas                                               //
//                  (saida)ponteiro para variavel que sera preenchida                                             //
//                                          com o numero de etapas necessarias                                    //
//                                                                                                                //
// Retornos:        nenhum                                                                                        //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int PegaEtapas (CART Destino, long *pnEtapas)
{
        int ReturnCode;
        RCOORD DeslocRobo, DestinoRobo;
        NPASSOS nPassos;

        ReturnCode = TransformaCoordenada(Destino, &DestinoRobo);

        if (ReturnCode)
                return -1;
        else
        {
                CalculaDeslocamento(DestinoRobo, &DeslocRobo);
                CalculaNumeroPassos(DeslocRobo, &nPassos);
                CalculaEtapas(nPassos, pnEtapas);
        }
        return 0;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:                  AcionaMotores                                                                         //
//                                                                                                                //
// Prosposito:      Aciona os tres motores em paralelo                                                            //
//                                                                                                                //
// Parametros:      (entrada) Npassos - nro de passos para cada motor                                             //
//                                  (entrada) TempoTotal -    Tempo para movimentacao                             //
//                                                                                                                //
// Retornos:        nenhum                                                                                        //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void AcionaMotores(NPASSOS nPassos, double TempoTotal)
{
        RCOORD Tempo, TempoLimite;
        long TempoInicial, PulsoTeta, PulsoAlfa, PulsoR;

        if(nPassos.npTeta < 0)
        {
                // coloco direcao negativa
                PulsoTeta = 0;
                nPassos.npTeta = - nPassos.npTeta;
        }
```

```
        else
                PulsoTeta = 1;

        if(nPassos.npAlfa < 0)
        {
                // coloco direcao negativa
                PulsoAlfa = 0;
                nPassos.npAlfa = -nPassos.npAlfa ;
        }
        else
                PulsoAlfa = 1;

        if(nPassos.npR < 0)
        {
                // coloco direcao negativa
                PulsoR = 0;
                nPassos.npR = -nPassos.npR ;
        }
        else
                PulsoR = 1;

        if(nPassos.npTeta)
        {
                Tempo.teta = 0.5*TempoTotal / (double)nPassos.npTeta;
                TempoLimite.teta = Tempo.teta;
        }

        if(nPassos.npAlfa)
        {
                Tempo.alfa  = 0.5*TempoTotal / (double)nPassos.npAlfa;
                TempoLimite.alfa = Tempo.alfa;
        }

        if(nPassos.npR)
        {
                Tempo.r = 0.5*TempoTotal / (double)nPassos.npR;
                TempoLimite.r = Tempo.r;
        }

        TempoInicial = leTimer();

        while (nPassos.npTeta || nPassos.npAlfa || nPassos.npR )
        {
                // verifica se e necessário chavear
                 if((tempoDecorrido(TempoInicial, leTimer())>=TempoLimite.teta ) &&
(nPassos.npTeta))
                {
                        nPassos.npTeta = nPassos.npTeta - ChaveiaSaida(&PulsoTeta);
                        TempoLimite.teta = TempoLimite.teta + Tempo.teta;

                }

                // verifica se e necessário chavear
                if((tempoDecorrido(TempoInicial, leTimer() ) >= TempoLimite.alfa )&&
nPassos.npAlfa)
                {
                        nPassos.npAlfa = nPassos.npAlfa - ChaveiaSaida(&PulsoAlfa);
                        TempoLimite.alfa = TempoLimite.alfa + Tempo.alfa;
```

```
                }

                // verifica se e necessário chavear
                if((tempoDecorrido(TempoInicial, leTimer() ) >= TempoLimite.r ) &&
nPassos.npR)
                {
                        nPassos.npR = nPassos.npR - ChaveiaSaida(&PulsoR);
                        TempoLimite.r = TempoLimite.r + Tempo.r;
                }
                // monta o byte de saida
                outp(0x378, (int) (PulsoTeta + (PulsoAlfa << 2)  + (PulsoR << 4) + (1 << 6)));
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                            //
// Funcao:              ChaveiaSaida                                                                          //
//                                                                                                            //
// Prososito:   Chaveia a saida do trem de pulso                                                              //
//                                                                                                            //
// Parametros:  (entrada e saida) ponteiro para variavel que                                                  //
//                              descreve o trem de      pulso                                                 //
//                                                                                                            //
// Retornos:    nenhum                                                                                        //
//                                                                                                            //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int ChaveiaSaida (long *Saida)
{
        // verifica se o pulso esta alto se estiver abaixa
        if(*Saida & 2)
        {
                *Saida = *Saida - 2;
                return 0;
        }
        else
                //      senao ergue
                *Saida = *Saida + 2;

        return 1;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                            //
// Funcao:              AcionamTeclado                                                                        //
//                                                                                                            //
// Prososito:   Aciona motores de acordo com entradas do teclado                                              //
//                                                                                                            //
// Parametros:  Entrada: Variavel auxiliar para identificacao da                                              //
//                              da tecla digitada                                                             //
//                                                                                                            //
// Retornos:    nenhum                                                                                        //
//                                                                                                            //
// Variavel Global: Atualiza a posicao atual do robo                                                          //
//                                                                                                            //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void AcionamTeclado ()
{
        NPASSOS nPassos;
```

```
char aux = 0;
int i, imax;
CART BufferJuntas[20];
double Orientacao[20];
int nponto, uponto;
double Tempo[20];

nPassos.npAlfa = 0;    // Inicializa a contagem de passos dados
nPassos.npR = 0;
nPassos.npTeta = 0;

nponto=0;
uponto=0;


while (aux != '0' )
{
        if (kbhit !=0)
        {
                aux = getch();
                switch (aux)
                {
                // Acionamento do motor Teta

                        case 'a':
                        case 'A':
                                if (aux == 'a')
                                        imax=5;
                                else
                                        imax=1;


                                outp (0x378 , 127 );      // setar a direcao previamente
01111111
                                espera(100000);
                                for(i = 0; i < imax; i++)
                                {
                                        outp (0x378 , 125 );      // envia sinal baixo
01111101
                                        espera(100000);            // espera periodopulso
                                        outp (0x378 , 127 );      // envia sinal alto
01111111
                                        espera(100000);            // espera periodopulso
                                        nPassos.npTeta--;
                                }
                                nPassosDados.npTeta = nPassosDados.npTeta +
nPassos.npTeta;  // Atualiza o controle de passos global
                                nPassosDados.npAlfa = nPassosDados.npAlfa +
nPassos.npAlfa;
                                nPassosDados.npR = nPassosDados.npR + nPassos.npR;
                                AtualizaPosicaoAtual(nPassos); // Atualiza a posicao
atual apos os passos dados
                                nPassos.npTeta = 0;
                                nPassos.npAlfa = 0;
                                nPassos.npR = 0;

                        break;
```

```
                        case 's':
                        case 'S':
                                if (aux == 's')
                                        imax=5;
                                else
                                        imax=1;

                                outp (0x378 ,126  );      // setar a direcao previamente
01111110
                                espera(100000);
                                for(i = 0; i < imax; i++)
                                {
                                        outp (0x378 , 124 );      // envia sinal baixo
01111100
                                        espera(100000);           // espera periodopulso
                                        outp (0x378 , 126);       // envia sinal alto
01111110
                                        espera(100000);           // espera periodopulso
                                        nPassos.npTeta++;
                                }
                                nPassosDados.npTeta = nPassosDados.npTeta +
nPassos.npTeta;  // Atualiza o controle de passos global
                                nPassosDados.npAlfa = nPassosDados.npAlfa +
nPassos.npAlfa;
                                nPassosDados.npR = nPassosDados.npR + nPassos.npR;
                                AtualizaPosicaoAtual(nPassos); // Atualiza a posicao
atual apos os passos dados
                                nPassos.npTeta = 0;
                                nPassos.npAlfa = 0;
                                nPassos.npR = 0;

                        break;


                // Acionamento do motor Alfa


                        case 'q':
                        case 'Q':
                                if (aux == 'q')
                                        imax=200;
                                else
                                        imax=1;

                                outp (0x378 , 127 );      // setar a direcao previamente
01111111
                                espera(10000);
                                for(i = 0; i < imax; i++)
                                {
                                        outp (0x378 , 119  );      // envia sinal baixo
01110111
                                        espera(10000);           // espera periodopulso
                                        outp (0x378 , 127 );      // envia sinal alto
01111111
                                        espera(10000);           // espera periodopulso
                                        nPassos.npAlfa++;
                                }
```

```
                                        nPassosDados.npTeta = nPassosDados.npTeta +
nPassos.npTeta;  // Atualiza o controle de passos global
                                        nPassosDados.npAlfa = nPassosDados.npAlfa +
nPassos.npAlfa;
                                        nPassosDados.npR = nPassosDados.npR + nPassos.npR;
                                        AtualizaPosicaoAtual(nPassos); // Atualiza a posicao
atual apos os passos dados
                                        nPassos.npTeta = 0;
                                        nPassos.npAlfa = 0;
                                        nPassos.npR = 0;

                                break;

                                case 'w':
                                case 'W':
                                        if (aux == 'w')
                                                imax=200;
                                        else
                                                imax=1;

                                outp (0x378 , 123 );      // setar a direcao previamente 01111011
                                espera(10000);
                                for(i = 0; i < imax; i++)
                                {
                                        outp (0x378 , 115  );      // envia sinal baixo  01110011
                                        espera(10000);           // espera periodopulso
                                        outp (0x378 , 123 );      // envia sinal alto   01111011
                                        espera(10000);           // espera periodopulso
                                        nPassos.npAlfa--;
                                }
                                nPassosDados.npTeta = nPassosDados.npTeta + nPassos.npTeta;
// Atualiza o controle de passos global
                                nPassosDados.npAlfa = nPassosDados.npAlfa + nPassos.npAlfa;
                                nPassosDados.npR = nPassosDados.npR + nPassos.npR;
                                AtualizaPosicaoAtual(nPassos); // Atualiza a posicao atual apos os
passos dados
                                nPassos.npTeta = 0;
                                nPassos.npAlfa = 0;
                                nPassos.npR = 0;

                                break;


                        // Acionamento do motor da Guia Linear

                                case 'z':
                                case 'Z':
                                        if (aux == 'z')
                                                imax=200;
                                        else
                                                imax=1;

                                        outp (0x378 , 127 );      // setar a direcao previamente
01111111
                                        espera(1000);
                                        for(i = 0; i < imax; i++)
                                        {
```

```
                                        outp (0x378 , 95 );      // envia sinal baixo
01011111
                                        espera(1000);           // espera periodopulso
                                        outp (0x378 , 127 );      // envia sinal alto
01111111
                                        espera(1000);           // espera periodopulso
                                        nPassos.npR++;
                                }
                                nPassosDados.npTeta = nPassosDados.npTeta +
nPassos.npTeta;  // Atualiza o controle de passos global
                                nPassosDados.npAlfa = nPassosDados.npAlfa +
nPassos.npAlfa;
                                nPassosDados.npR = nPassosDados.npR + nPassos.npR;
                                AtualizaPosicaoAtual(nPassos); // Atualiza a posicao
atual apos os passos dados
                                nPassos.npTeta = 0;
                                nPassos.npAlfa = 0;
                                nPassos.npR = 0;

                        break;

                        case 'x':
                        case 'X':
                                if (aux == 'x')
                                        imax=200;
                                else
                                        imax=1;


                                outp (0x378 , 111 );      // setar a direcao previamente
01101111
                                espera(1000);

                                for (i = 0; i < imax; i++)
                                {
                                        outp (0x378 , 79 );      // envia sinal baixo
01001111
                                        espera(1000);           // espera periodopulso
                                        outp (0x378 , 111 );      // envia sinal alto
01101111
                                        espera(1000);           // espera periodopulso
                                        nPassos.npR--;
                                }
                                nPassosDados.npTeta = nPassosDados.npTeta +
nPassos.npTeta;  // Atualiza o controle de passos global
                                nPassosDados.npAlfa = nPassosDados.npAlfa +
nPassos.npAlfa;
                                nPassosDados.npR = nPassosDados.npR + nPassos.npR;
                                AtualizaPosicaoAtual(nPassos); // Atualiza a posicao
atual apos os passos dados
                                nPassos.npTeta = 0;
                                nPassos.npAlfa = 0;
                                nPassos.npR = 0;


                        break;
```

```
// Orientacao da Garra

        case 'd':
        case 'D':
                if (aux == 'd')
                        imax=100;
                else
                        imax=1;

                outp(0x37a, 15);     // setar a direcao previamente 1111
                espera(1000);

                for(i = 0; i < imax; i++)
                {

                        outp (0x37a , 13  );     // envia sinal baixo  1101
                        espera(1000);            // espera periodopulso
                        outp (0x37a , 15 );      // envia sinal alto 1111
                        espera(1000);            // espera periodopulso
                        AtualizaAngulo(1);
                }

        break;


        case 'f':
        case 'F':
                if (aux == 'f')
                        imax=100;
                else
                        imax=1;

                outp(0x37a, 14);     // setar a direcao previamente 1110
                espera(1000);

        for(i = 0; i < imax; i++)
        {
                outp (0x37a , 12  );     // envia sinal baixo 1100
                espera(1000);            // espera periodopulso
                outp (0x37a ,  14);      // envia sinal alto   1110
                espera(1000);            // espera periodopulso
                AtualizaAngulo(-1);
        }
        break;


// Abertura e fechamento da Garra


        case 'e':
        case 'E':
                if (aux == 'e')
                        imax=100;
                else
                        imax=1;

                outp(0x37a, 15);     // setar a direcao previamente 1111
                espera(1000);
```

```
                    for(i = 0; i < imax; i++)
                    {
                            outp (0x37a , 7 );      // envia sinal baixo 0111
                            espera(1000);           // espera periodopulso
                            outp (0x37a , 15 );     // envia sinal alto  1111
                            espera(1000);           // espera periodopulso
                    }

            break;


            case 'r':
            case 'R':
                    if (aux == 'r')
                            imax=100;
                    else
                            imax=1;


                    outp(0x37a, 11);    // setar a direcao previamente 1011
                    espera(1000);

                    for(i = 0; i < imax; i++)
                    {
                            outp (0x37a , 3 );      // envia sinal baixo 0011
                            espera(1000);           // espera periodopulso
                            outp (0x37a , 11 );     // envia sinal alto  1011
                            espera(1000);           // espera periodopulso
                    }

            break;

// Aquisicao de pontos

                    case 'p': // aquisicao de um ponto para reproduzir


            // Armazenagem dos pontos no Buffer

                            BufferJuntas[nponto].x = PosicaoAtual.x;
                            BufferJuntas[nponto].y = PosicaoAtual.y;
                            BufferJuntas[nponto].z = PosicaoAtual.z;
                            Orientacao[nponto] = AnguloAtual;

            // Armazena o tempo do movimento
                            TelaTempo();
                            //printf("\nDigite o tempo total do movimento: ");
                            scanf("%lf",&Tempo[nponto]);
                            TelaTeclado();

            // Prepara para armazenar o proximo ponto
                            nponto++;

                    break;

                    case 'l': // Aciona ponto a ponto em linha reta
```

```
                                        uponto = nponto; // ponto de parada
                                        for(nponto = 0; nponto<uponto; nponto++)
                                        {
                                                // Aciona robo em linha reta e atualiza posicao
automa.
                                                MovimentaRobo(BufferJuntas[nponto],
Tempo[nponto]);
                                                MoveEfetuador(Orientacao[nponto]);
                                        }

                                        break;


                        }               // fim do switch


                } // fim do if

        } // fim do while

        aux = 100; // Para nao sair do menu anterior

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                //
// Funcao:              MovimentaRobô4                                                                            //
//                                                                                                                //
// Prososito:   Efetua a movimentação do robô com motores                                                         //
//                              em sequencia e com curava de aceleracao                                           //
//                                                                                                                //
// Parametros:  (entrada)Ponto para o qual o robo deve mover-se em                                                //
//                                      coordenadas cartesianas                                                   //
//              (entrada)Velocidade maxima de acionamento de cada
        //
//                      motor                                                                                     //
//                                                                                                                //
// Retornos:    nenhum                                                                                            //
//                                                                                                                //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MovimentaRobo4 (CART Destino, RCOORD VelMaxima)
{
        int ReturnCode, i;
        RCOORD DeslocRobo, DestinoRobo;
        NPASSOS nPassos;
        RCOORD VelStart;
        RCOORD aceleracao;
        RCOORD incremento;
        double deltat;
        double contapassos;

        contapassos = 0;

        VelStart.teta = 10; // Velocidade de partida ou parada em passos por segundo
        VelStart.alfa = 50; // Velocidade de partida ou parada em passos por segundo
        VelStart.r = 100;   // Velocidade de partida ou parada em passos por segundo

        aceleracao.teta = 5 ;   // Em passos para atingir a velocidade maxima
```

```
        aceleracao.alfa = 150 ;  // Em passos para atingir a velocidade maxima
        aceleracao.r = 500 ;     // Em passos para atingir a velocidade maxima


        // Calcula a variação de tempo linear que causa a aceleração do motor

        incremento.teta = (1000/VelStart.teta - 1000/VelMaxima.teta ) /( 2*aceleracao.teta);
        incremento.alfa = (1000/VelStart.alfa - 1000/VelMaxima.alfa ) /( 2*aceleracao.alfa);
        incremento.r = (1000/VelStart.r - 1000/VelMaxima.r ) /( 2*aceleracao.r);

        ReturnCode = TransformaCoordenada(Destino, &DestinoRobo); // Entrada cart e saida em
CoordRobo

        if (ReturnCode)
                printf(" Ponto fora do volume de controle!!! \n");

        else
        {

                CalculaDeslocamento(DestinoRobo, &DeslocRobo);
                CalculaNumeroPassos(DeslocRobo, &nPassos);


// Acionamento dos Motores

// Acionamento do Teta

                deltat = 1000/(2*VelStart.teta);  // Calcula o meio período para a geração da onda do
motor

                if (nPassos.npTeta > 0 ){       // caso deseje teta positivo

                        contapassos = 0;
                        outp (0x378 , 127 );        // setar a direcao previamente 01111111
                        espera((long)(1000*deltat/2));


                        // Período de Aceleracao

                        for (i=0; (i<aceleracao.teta) && (contapassos<nPassos.npTeta)  ; i++){
                                outp (0x378 , 125  );      // envia sinal baixo  01111101
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 127 );       // envia sinal alto   01111111
                                espera((long)(1000*deltat/2));
                                deltat= deltat - incremento.teta;
                                contapassos++;
                        }

                        // Periodo de Velocidade Maxima

                        for (i = (int)aceleracao.teta; (i < nPassos.npTeta - aceleracao.teta) &&
(contapassos<nPassos.npTeta) ; i++){
                                outp (0x378 , 125  );      // envia sinal baixo  01111101
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 127 );       // envia sinal alto   01111111
                                espera((long)(1000*deltat/2));
                                contapassos++;
                        }
```

```
                    // Periodo de Desaceleracao

                    for (i=0; (i<aceleracao.teta) && (contapassos<nPassos.npTeta) ; i++){
                            outp (0x378 , 125 );      // envia sinal baixo  01111101
                            espera((long)(1000*deltat/2));
                            outp (0x378 , 127 );      // envia sinal alto   01111111
                            espera((long)(1000*deltat/2));
                            deltat= deltat + incremento.teta;
                            contapassos++;
                            }
            }

            if (nPassos.npTeta < 0 ){ // Caso deseje-se um teta negativo

                    outp (0x378 , 126 );      // setar a direcao previamente 01111110
                    espera((long)(1000*deltat/2));
                    contapassos = 0;


                    // Periodo de Aceleracao

                    for (i=0; (i<aceleracao.teta) && (contapassos>nPassos.npTeta) ; i++){
                            outp (0x378 , 124 );      // envia sinal baixo  01111100
                            espera((long)(1000*deltat/2));
                            outp (0x378 , 126);       // envia sinal alto   01111110
                            espera((long)(1000*deltat/2));
                            deltat= deltat - incremento.teta;
                            contapassos--;
                    }

                    // Velocidade Máxima

                    for (i = (int)aceleracao.teta; (i < -nPassos.npTeta - aceleracao.teta) &&
(contapassos>nPassos.npTeta); i++){
                            outp (0x378 , 124 );      // envia sinal baixo  01111100
                            espera((long)(1000*deltat/2));
                            outp (0x378 , 126);       // envia sinal alto   01111110
                            espera((long)(1000*deltat/2));
                            contapassos--;
                    }

                                            // Periodo de Desaceleracao

                    for (i=0; (i<aceleracao.teta) && (contapassos>nPassos.npTeta) ; i++){
                            outp (0x378 , 124 );      // envia sinal baixo  01111100
                            espera((long)(1000*deltat/2));
                            outp (0x378 , 126);       // envia sinal alto   01111110
                            espera((long)(1000*deltat/2));
                            deltat= deltat + incremento.teta;
                            contapassos--;
                    }
            }

// Acionamento do motor Alfa - Segue a mesma lógica do motor Teta


            deltat = 1000/(2*VelStart.alfa);
```

```
        if (nPassos.npAlfa > 0 ){  // Alfa positivo

                outp (0x378 , 127 );      // setar a direcao previamente 01111111
                espera((long)(1000*deltat/2));
                contapassos = 0;

                for (i=0; (i<aceleracao.alfa) && (contapassos<nPassos.npAlfa)  ; i++){
                        outp (0x378 , 119  );      // envia sinal baixo  01110111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );      // envia sinal alto   01111111
                        espera((long)(1000*deltat/2));
                        deltat= deltat - incremento.alfa;
                        contapassos++;
                }

                for (i = (int)aceleracao.alfa; (i < nPassos.npAlfa - aceleracao.alfa) &&
(contapassos<nPassos.npAlfa); i++){
                        outp (0x378 , 119  );      // envia sinal baixo  01110111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );      // envia sinal alto   01111111
                        espera((long)(1000*deltat/2));
                        contapassos++;
                }

                for (i=0; (i<aceleracao.alfa) && (contapassos<nPassos.npAlfa) ; i++){
                        outp (0x378 , 119  );      // envia sinal baixo  01110111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );      // envia sinal alto   01111111
                        espera((long)(1000*deltat/2));
                        deltat= deltat + incremento.alfa;
                        contapassos++;
                }
        }

        if (nPassos.npAlfa < 0 ){ // Alfa negativo

                outp (0x378 , 123 );      // setar a direcao previamente 01111011
                espera((long)(1000*deltat/2));
                contapassos = 0;

                for (i=0; (i<aceleracao.alfa) && (contapassos>nPassos.npAlfa) ; i++){
                        outp (0x378 , 115  );      // envia sinal baixo  01110011
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 123 );      // envia sinal alto   01111011

                        espera((long)(1000*deltat/2));
                        deltat= deltat - incremento.alfa;
                        contapassos--;
                }

                for (i = (int)aceleracao.alfa; (i < -nPassos.npAlfa - aceleracao.alfa) &&
(contapassos>nPassos.npAlfa); i++){
                        outp (0x378 , 115  );      // envia sinal baixo  01110011
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 123 );      // envia sinal alto   01111011
```

```
                        espera((long)(1000*deltat/2));
                        contapassos--;
                }

                for (i=0; (i<aceleracao.alfa) && (contapassos>nPassos.npAlfa) ; i++){
                        outp (0x378 , 115 );      // envia sinal baixo  01110011
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 123 );      // envia sinal alto   01111011

                        espera((long)(1000*deltat/2));
                        deltat= deltat + incremento.alfa;
                        contapassos--;
                }
        }

// Acionamento do Linear - Segue a mesma lógica do motor Teta

        deltat = 1000/(2*VelStart.r);

        if (nPassos.npR > 0 ){  // R positivo

                outp (0x378 , 127 );      // setar a direcao previamente 01111111
                espera((long)(1000*deltat/2));
                contapassos = 0;

                for (i=0; (i<aceleracao.r) && (contapassos<nPassos.npR) ; i++){
                        outp (0x378 , 95 );       // envia sinal baixo 01011111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );      // envia sinal alto  01111111
                        espera((long)(1000*deltat/2));
                        deltat= deltat - incremento.r;
                        contapassos++;
                }

                for (i = (int)aceleracao.r; (i < nPassos.npR - aceleracao.r) &&
(contapassos<nPassos.npR); i++){
                        outp (0x378 , 95 );       // envia sinal baixo 01011111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );      // envia sinal alto  01111111
                        espera((long)(1000*deltat/2));
                        contapassos++;
                }

                for (i=0; (i<aceleracao.r) && (contapassos<nPassos.npR) ; i++){
                        outp (0x378 , 95 );       // envia sinal baixo 01011111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );      // envia sinal alto  01111111
                        espera((long)(1000*deltat/2));
                        deltat= deltat + incremento.r;
                        contapassos++;
                }
        }

        if (nPassos.npR < 0 ){ // R negativo

                outp (0x378 , 111 );      // setar a direcao previamente 01101111
                espera((long)(1000*deltat/2));
                contapassos = 0;
```

```
                        for (i=0; (i<aceleracao.r) && (contapassos>nPassos.npR) ; i++){
                                outp (0x378 , 79 );      // envia sinal baixo 01001111
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 111 );     // envia sinal alto  01101111
                                espera((long)(1000*deltat/2));
                                deltat= deltat - incremento.r;
                                contapassos--;
                        }

                        for (i = (int)aceleracao.r; (i < -nPassos.npR - aceleracao.r) &&
(contapassos>nPassos.npR); i++){
                                outp (0x378 , 79 );      // envia sinal baixo 01001111
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 111 );     // envia sinal alto  01101111
                                espera((long)(1000*deltat/2));
                                contapassos--;
                        }

                        for (i=0; (i<aceleracao.r) && (contapassos>nPassos.npR) ; i++){
                                outp (0x378 , 79 );      // envia sinal baixo 01001111
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 111 );     // envia sinal alto  01101111
                                espera((long)(1000*deltat/2));
                                deltat= deltat + incremento.r;
                                contapassos--;
                        }
                }


// Controle do numero de passo dados e posicao alcancada apos movimentacao

                nPassosDados.npTeta = nPassosDados.npTeta + nPassos.npTeta;  // Atualiza o
controle de passos global
                nPassosDados.npAlfa = nPassosDados.npAlfa + nPassos.npAlfa;
                nPassosDados.npR = nPassosDados.npR + nPassos.npR;

                AtualizaPosicaoAtual(nPassos); // Atualiza a variavel da posicao


        } // fim do else
}


void MoveEfetuador (double Angulo)
{
        double DeslocAngular;
        short Passos;
        short Saida;
        double aux;

        DeslocAngular = Angulo - AnguloAtual;

        aux = (DeslocAngular*TRANSEFET / DESLOCPASSO);

        Passos = aux;
```

```
        AtualizaAngulo (Passos);

        if (Passos < 0)
        {
                Saida = 14;
                Passos = -Passos;
        }
        else
                Saida = 15;

        outp(0x37a, Saida);

        while (Passos)
        {
                outp(0x37a, Saida - 2);
                espera(1000);
                outp(0x37a, Saida);
                Passos--;
        }
}

void AtualizaAngulo(long Passos)
{
        AnguloAtual = AnguloAtual + Passos*DESLOCPASSO/TRANSEFET;
}

////////////////////////////////////////////////////////////////
//                                                            //
// Funcao:    VoltaZero                                       //
//                                                            //
// Prososito:  Traz o robo de volta pra origem                //
//                                                            //
// Parametros: (entrada)Velocidade maxima dos motores         //
//                                                            //
// Retornos:   nenhum                                         //
//                                                            //
////////////////////////////////////////////////////////////////
void VoltaZero (RCOORD VelMaxima)
{
        int ReturnCode, i;

        NPASSOS nPassos;
        RCOORD VelStart;
        RCOORD aceleracao;
        RCOORD incremento;
        double deltat;
        double contapassos;

        contapassos = 0;

        VelStart.teta = 10; // Velocidade de partida ou parada em passos por segundo
        VelStart.alfa = 50; // Velocidade de partida ou parada em passos por segundo
        VelStart.r = 100; // Velocidade de partida ou parada em passos por segundo

        aceleracao.teta = 5 ; // Em passos para atingir a velocidade maxima
        aceleracao.alfa = 150 ; // Em passos para atingir a velocidade maxima
        aceleracao.r = 500 ; // Em passos para atingir a velocidade maxima
```

```
        incremento.teta = (1000/VelStart.teta - 1000/VelMaxima.teta ) /( 2*aceleracao.teta);
        incremento.alfa = (1000/VelStart.alfa - 1000/VelMaxima.alfa ) /( 2*aceleracao.alfa);
        incremento.r = (1000/VelStart.r - 1000/VelMaxima.r ) /( 2*aceleracao.r);

        nPassos = nPassosDados;

// Acionamento dos Motores

// Acionamento do Teta

        deltat = 1000/(2*VelStart.teta);

        if (nPassos.npTeta > 0 ){  // teta positivo

                        contapassos = 0;
                        outp (0x378 , 127 );      // setar a direcao previamente 01111111
                        espera((long)(1000*deltat/2));

                        for (i=0; (i<aceleracao.teta) && (contapassos<nPassos.npTeta)  ; i++){
                                outp (0x378 , 125  );      // envia sinal baixo  01111101
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 127 );      // envia sinal alto   01111111
                                espera((long)(1000*deltat/2));
                                deltat= deltat - incremento.teta;
                                contapassos++;
                        }

                        for (i = (int)aceleracao.teta; (i < nPassos.npTeta - aceleracao.teta) &&
(contapassos<nPassos.npTeta) ; i++){
                                outp (0x378 , 125  );      // envia sinal baixo  01111101
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 127 );      // envia sinal alto   01111111
                                espera((long)(1000*deltat/2));
                                contapassos++;
                        }

                        for (i=0; (i<aceleracao.teta) && (contapassos<nPassos.npTeta) ; i++){
                                outp (0x378 , 125  );      // envia sinal baixo  01111101
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 127 );      // envia sinal alto   01111111
                                espera((long)(1000*deltat/2));
                                deltat= deltat + incremento.teta;
                                contapassos++;
                                }
                }

                if (nPassos.npTeta < 0 ){ // teta negativo

                        outp (0x378 , 126 );      // setar a direcao previamente 01111110
                        espera((long)(1000*deltat/2));
                        contapassos = 0;

                        for (i=0; (i<aceleracao.teta) && (contapassos>nPassos.npTeta) ; i++){
                                outp (0x378 , 124  );      // envia sinal baixo  01111100
                                espera((long)(1000*deltat/2));
                                outp (0x378 , 126);      // envia sinal alto   01111110
                                espera((long)(1000*deltat/2));
```

```
                        deltat= deltat - incremento.teta;
                        contapassos--;
                }

                for (i = (int)aceleracao.teta; (i < -nPassos.npTeta - aceleracao.teta) &&
(contapassos>nPassos.npTeta); i++){
                        outp (0x378 , 124 );    // envia sinal baixo  01111100
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 126);     // envia sinal alto   01111110
                        espera((long)(1000*deltat/2));
                        contapassos--;
                }

                for (i=0; (i<aceleracao.teta) && (contapassos>nPassos.npTeta) ; i++){
                        outp (0x378 , 124 );    // envia sinal baixo  01111100
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 126);     // envia sinal alto   01111110
                        espera((long)(1000*deltat/2));
                        deltat= deltat + incremento.teta;
                        contapassos--;
                }
        }

// Acionamento do Alfa

        deltat = 1000/(2*VelStart.alfa);

        if (nPassos.npAlfa > 0 ){  // Alfa positivo

                outp (0x378 , 127 );    // setar a direcao previamente 01111111
                espera((long)(1000*deltat/2));
                contapassos = 0;

                for (i=0; (i<aceleracao.alfa) && (contapassos<nPassos.npAlfa)  ; i++){
                        outp (0x378 , 119 );    // envia sinal baixo  01110111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );    // envia sinal alto   01111111
                        espera((long)(1000*deltat/2));
                        deltat= deltat - incremento.alfa;
                        contapassos++;
                }

                for (i = (int)aceleracao.alfa; (i < nPassos.npAlfa - aceleracao.alfa) &&
(contapassos<nPassos.npAlfa); i++){
                        outp (0x378 , 119 );    // envia sinal baixo  01110111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );    // envia sinal alto   01111111
                        espera((long)(1000*deltat/2));
                        contapassos++;
                }

                for (i=0; (i<aceleracao.alfa) && (contapassos<nPassos.npAlfa) ; i++){
                        outp (0x378 , 119 );    // envia sinal baixo  01110111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );    // envia sinal alto   01111111
                        espera((long)(1000*deltat/2));
```

```
                    deltat= deltat + incremento.alfa;
                    contapassos++;
                }
            }

            if (nPassos.npAlfa < 0 ){ // Alfa negativo

                outp (0x378 , 123 );      // setar a direcao previamente 01111011
                espera((long)(1000*deltat/2));
                contapassos = 0;

                for (i=0; (i<aceleracao.alfa) && (contapassos>nPassos.npAlfa) ; i++){
                    outp (0x378 , 115 );      // envia sinal baixo  01110011
                    espera((long)(1000*deltat/2));
                    outp (0x378 , 123 );      // envia sinal alto   01111011
                    espera((long)(1000*deltat/2));
                    deltat= deltat - incremento.alfa;
                    contapassos--;
                }

                for (i = (int)aceleracao.alfa; (i < -nPassos.npAlfa - aceleracao.alfa) &&
(contapassos>nPassos.npAlfa); i++){
                    outp (0x378 , 115 );      // envia sinal baixo  01110011
                    espera((long)(1000*deltat/2));
                    outp (0x378 , 123 );      // envia sinal alto   01111011
                    espera((long)(1000*deltat/2));
                    contapassos--;
                }

                for (i=0; (i<aceleracao.alfa) && (contapassos>nPassos.npAlfa) ; i++){
                    outp (0x378 , 115 );      // envia sinal baixo  01110011
                    espera((long)(1000*deltat/2));
                    outp (0x378 , 123 );      // envia sinal alto   01111011
                    espera((long)(1000*deltat/2));
                    deltat= deltat + incremento.alfa;
                    contapassos--;
                }
            }

// Acionamento do Linear

            deltat = 1000/(2*VelStart.r);

            if (nPassos.npR > 0 ){  // R positivo

                outp (0x378 , 127 );      // setar a direcao previamente 01111111
                espera((long)(1000*deltat/2));
                contapassos = 0;

                for (i=0; (i<aceleracao.r) && (contapassos<nPassos.npR) ; i++){
                    outp (0x378 , 95 );      // envia sinal baixo 01011111
                    espera((long)(1000*deltat/2));
                    outp (0x378 , 127 );      // envia sinal alto  01111111
                    espera((long)(1000*deltat/2));
                    deltat= deltat - incremento.r;
                    contapassos++;
                }
```

```
                    for (i = (int)aceleracao.r; (i < nPassos.npR - aceleracao.r) &&
(contapassos<nPassos.npR); i++){
                        outp (0x378 , 95 );      // envia sinal baixo 01011111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );     // envia sinal alto  01111111
                        espera((long)(1000*deltat/2));
                        contapassos++;
                    }

                    for (i=0; (i<aceleracao.r) && (contapassos<nPassos.npR) ; i++){
                        outp (0x378 , 95 );      // envia sinal baixo 01011111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 127 );     // envia sinal alto  01111111
                        espera((long)(1000*deltat/2));
                        deltat= deltat + incremento.r;
                        contapassos++;
                    }
                }

                if (nPassos.npR < 0 ){ // R negativo

                    outp (0x378 , 111 );     // setar a direcao previamente 01101111
                    espera((long)(1000*deltat/2));
                    contapassos = 0;

                    for (i=0; (i<aceleracao.r) && (contapassos>nPassos.npR) ; i++){
                        outp (0x378 , 79 );      // envia sinal baixo 01001111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 111 );     // envia sinal alto  01101111
                        espera((long)(1000*deltat/2));
                        deltat= deltat - incremento.r;
                        contapassos--;
                    }

                    for (i = (int)aceleracao.r; (i < -nPassos.npR - aceleracao.r) &&
(contapassos>nPassos.npR); i++){
                        outp (0x378 , 79 );      // envia sinal baixo 01001111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 111 );     // envia sinal alto  01101111
                        espera((long)(1000*deltat/2));
                        contapassos--;
                    }

                    for (i=0; (i<aceleracao.r) && (contapassos>nPassos.npR) ; i++){
                        outp (0x378 , 79 );      // envia sinal baixo 01001111
                        espera((long)(1000*deltat/2));
                        outp (0x378 , 111 );     // envia sinal alto  01101111
                        espera((long)(1000*deltat/2));
                        deltat= deltat + incremento.r;
                        contapassos--;
                    }
                }
// Controle do numero de passo dados e posicao alcancada apos movimentacao

                nPassosDados.npTeta = nPassosDados.npTeta + nPassos.npTeta;  // Atualiza o
controle de passos global
                nPassosDados.npAlfa = nPassosDados.npAlfa + nPassos.npAlfa;
                nPassosDados.npR = nPassosDados.npR + nPassos.npR;
```

```
                AtualizaPosicaoAtual(nPassos); // Atualiza a variavel da posicao
}

void TelaPrincipal ()
{
        char Buffer[50];

        _displaycursor( _GCURSOROFF );


        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 150, 150, 490, 330);
        _rectangle(_GFILLINTERIOR, 200, 110, 440, 140);

        _rectangle(_GFILLINTERIOR, 250, 380, 390, 460);
        _settextposition(25, 33);
        _settextcolor(14);
        _outtext("PosicaoAtual:" );
        _settextposition(26, 33);
        sprintf(Buffer, "x = %04lf mm", PosicaoAtual.x);
        _outtext(Buffer);
        _settextposition(27, 33);
        sprintf(Buffer, "y = %04lf mm", PosicaoAtual.y);
        _outtext(Buffer);
        _settextposition(28, 33);
        sprintf(Buffer, "z = %04lf mm", PosicaoAtual.z);
        _outtext(Buffer);



        _settextposition(5,22);

        _outtext("INTERFACE DE SOFTWARE DO ROBO CIRURGICO");
        _setcolor(14);
        _rectangle(_GBORDER,150,150,490,330);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,200,110,440,140);
        _rectangle(_GBORDER, 250, 380, 390, 460);
        _settextposition(8,28);
        _outtext("Selecione a opcao desejada:");

        _settextposition(11,20);
        _outtext(" 0 - Encerrar");
        _settextposition(12,20);
        _outtext(" 1 - Acionamento via teclado");
        _settextposition(13,20);
        _outtext(" 2 - Definir posicao atual como origem");
        _settextposition(14,20);
        _outtext(" 3 - Acionamento sequencial");
        _settextposition(15,20);
```

```
        _outtext(" 4 - Acionamento simultaneo");
        _settextposition(16,20);
        _outtext(" 5 - Acionamento com curva de aceleracao");
        _settextposition(17,20);
        _outtext(" 6 - Acionamento simultaneo retilineo");
        _settextposition(18,20);
        _outtext(" 9 - Acionamento da orientacao do forceps");
        _settextposition(19,20);
        _outtext(" B - Definir volume de trabalho");

}

void TelaTeclado()
{
        char Buffer[50];
        _displaycursor( _GCURSOROFF );

        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 150, 150, 490, 330);
        _rectangle(_GFILLINTERIOR, 200, 110, 440, 140);

        _rectangle(_GFILLINTERIOR, 250, 380, 390, 460);
        _settextposition(25, 33);
        _settextcolor(14);
        _outtext("PosicaoAtual:" );
        _settextposition(26, 33);
        sprintf(Buffer, "x = %04lf mm", PosicaoAtual.x);
        _outtext(Buffer);
        _settextposition(27, 33);
        sprintf(Buffer, "y = %04lf mm", PosicaoAtual.y);
        _outtext(Buffer);
        _settextposition(28, 33);
        sprintf(Buffer, "z = %04lf mm", PosicaoAtual.z);
        _outtext(Buffer);

        _settextposition(5,30);

        _outtext("ACIONAMENTO VIA TECLADO");
        _setcolor(14);
        _rectangle(_GBORDER,150,150,490,330);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,200,110,440,140);
        _rectangle(_GBORDER, 250, 380, 390, 460);
        _settextposition(8,28);
        _outtext("Selecione a opcao desejada:");
        _settextposition(11,20);
        _outtext(" 'a' ou 's' - Motor Teta");
        _settextposition(12,20);
        _outtext(" 'q' ou 'w' - Motor Alfa");
        _settextposition(13,20);
        _outtext(" 'z' ou 'x' - Motor Linear");
        _settextposition(14,20);
        _outtext(" 'd' ou 'f' - Orientacao da garra ");
```

```
        _settextposition(15,20);
        _outtext(" 'e' ou 'r' - Abertura/fechamento da garra");
        _settextposition(16,20);
        _outtext(" 'p' - Aquisicao de ponto" );
        _settextposition(17,20);
        _outtext(" 'l' - Disparar pontos aquisitados");
        _settextposition(18,20);
        _outtext(" 0 - Voltar ao menu inicial");
}

void TelaEntrada(int flag)
{
        char Buffer[50];
        _displaycursor( _GCURSOROFF );

        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 150, 150, 490, 330);
        _rectangle(_GFILLINTERIOR, 200, 110, 440, 140);

        _rectangle(_GFILLINTERIOR, 250, 380, 390, 460);
        _settextposition(25, 33);
        _settextcolor(14);
        _outtext("PosicaoAtual:" );
        _settextposition(26, 33);
        sprintf(Buffer, "x = %04lf mm", PosicaoAtual.x);
        _outtext(Buffer);
        _settextposition(27, 33);
        sprintf(Buffer, "y = %04lf mm", PosicaoAtual.y);
        _outtext(Buffer);
        _settextposition(28, 33);
        sprintf(Buffer, "z = %04lf mm", PosicaoAtual.z);
        _outtext(Buffer);

        _setcolor(14);
        _rectangle(_GBORDER,150,150,490,330);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,200,110,440,140);
        _rectangle(_GBORDER, 250, 380, 390, 460);

        if (flag != 9)
        {
                _settextcolor(14);
                _settextposition(8,28);
                _outtext(" Entre com os parametros.");
                _settextposition(11,20);
                _outtext(" Destino x:");
                _settextposition(12,20);
                _outtext(" Destino y:");
                _settextposition(13,20);
                _outtext(" Destino z:");
                switch (flag)
                {
                        case 3:
```

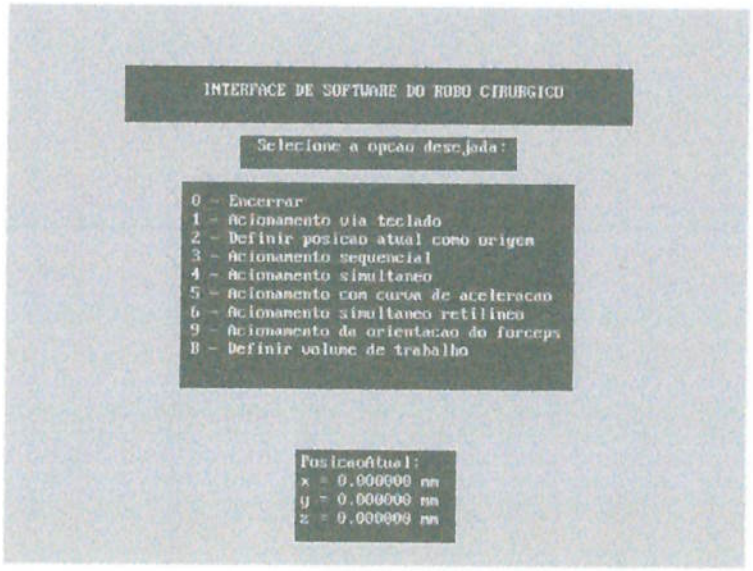
```
                                _settextposition(5,31);
                                _outtext("ACIONAMENTO SEQUENCIAL");
                                _settextposition(14,20);
                                _outtext(" Velocidade percentual:");
                                break;
                        case 4:
                                _settextposition(5,31);
                                _outtext("ACIONAMENTO SIMULTANEO");
                                _settextposition(14,20);
                                _outtext(" Tempo da movimentacao:");
                                break;
                        case 5:
                                _settextposition(5,25);
                                _outtext("ACIONAMENTO COM CURVA DE
ACELERACAO");
                                break;
                        case 6:
                                _settextposition(5,31);
                                _outtext("ACIONAMENTO RETILINEO");
                                _settextposition(14,20);
                                _outtext(" Tempo da movimentacao:");
                                break;
                }
        }
        else
        {
                _settextcolor(14);
                _settextposition(8,28);
                _outtext(" Entre com os parametros.");
                _settextposition(11,20);
                _outtext(" Orientacao da garra:");
                _settextposition(5,31);
                _outtext("ORIENTACAO DO FORCEPS");
        }
}

void TelaVolume()
{
        char Buffer[50];

        _displaycursor( _GCURSOROFF );

        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 150, 150, 490, 330);
        _rectangle(_GFILLINTERIOR, 200, 110, 440, 140);

        _rectangle(_GFILLINTERIOR, 250, 380, 390, 460);
        _settextposition(25, 33);
        _settextcolor(14);
        _outtext("PosicaoAtual:" );
        _settextposition(26, 33);
        sprintf(Buffer, "x = %04lf", PosicaoAtual.x);
        _outtext(Buffer);
```

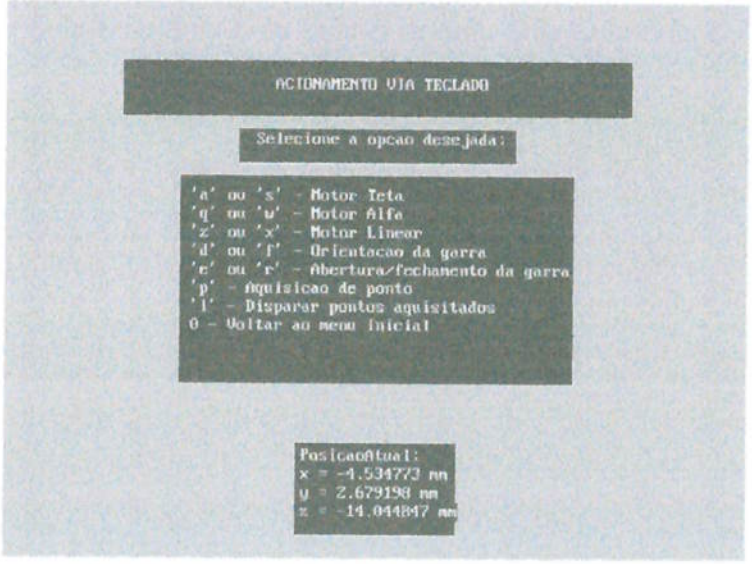
```
        _settextposition(27, 33);
        sprintf(Buffer, "y = %04lf", PosicaoAtual.y);
        _outtext(Buffer);
        _settextposition(28, 33);
        sprintf(Buffer, "z = %04lf", PosicaoAtual.z);
        _outtext(Buffer);

        _settextposition(5,26);

        _outtext("DEFINICAO DE NOVO VOLUME DE TRABALHO");
        _setcolor(14);
        _rectangle(_GBORDER,150,150,490,330);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,200,110,440,140);
        _rectangle(_GBORDER, 250, 380, 390, 460);
        _settextposition(8,28);
        _outtext("Volume de trabalho:");
        _settextposition(11,20);
        sprintf(Buffer, "Teta = %04lf rad", Posicao.teta);
        _outtext(Buffer);

        _settextposition(12,20);
        sprintf(Buffer, "Alfa = %04lf rad", Posicao.alfa);
        _outtext(Buffer);

        _settextposition(13,20);
        sprintf(Buffer, "R = %04lf mm", Posicao.r);
        _outtext(Buffer);

        _settextposition(17,20);
        _outtext("PRESSIONE QUALQUER TECLA PARA CONTINUAR!!!");
}

void TelaFinal()
{
        _displaycursor( _GCURSOROFF );

        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 100, 150, 540, 330);

        _settextcolor(14);
        _settextposition(12,20);
        _outtext("Aguarde!!! Voltando para posicao inicial..." );

        _settextposition(5,30);

        _outtext("TERMINANDO EXECUCAO");
        _setcolor(14);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,100,150,540,330);
```

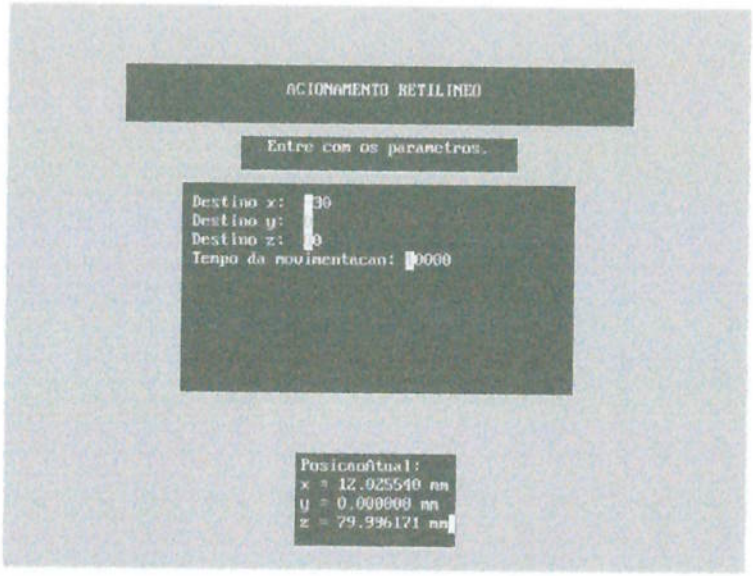
```
}
void TelaForaVolume()
{
        _displaycursor( _GCURSOROFF );

        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 100, 150, 540, 330);

        _settextcolor(14);

        _settextposition(5,30);
        _outtext("ALERTA !!! ALERTA!!!");
        _settextposition(12,20);
        _outtext("Ponto fora do volume de controle!!!");
        _settextposition(14,20);
        _outtext("Pressione qualquer tecla para continuar!!!");
        _setcolor(14);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,100,150,540,330);
        getch();
}
void TelaTempo()
{
        _displaycursor( _GCURSOROFF );

        _clearscreen(_GCLEARSCREEN);
        _setvideomode(_VRES16COLOR);
        _setcolor(9);
        _rectangle(_GFILLINTERIOR, 0,0,640,480);
        _setcolor(0);
        _rectangle(_GFILLINTERIOR, 100,50,540,100);
        _rectangle(_GFILLINTERIOR, 100, 150, 540, 330);

        _settextcolor(14);

        _settextposition(5,30);
        _outtext("PROGRAMACAO ONLINE");
        _settextposition(12,20);

        _outtext("Digite o tempo para movimentacao: ");
        _displaycursor( _GCURSORON );
        _setcolor(14);
        _rectangle(_GBORDER,100,50,540,100);
        _rectangle(_GBORDER,100,150,540,330);
}
```

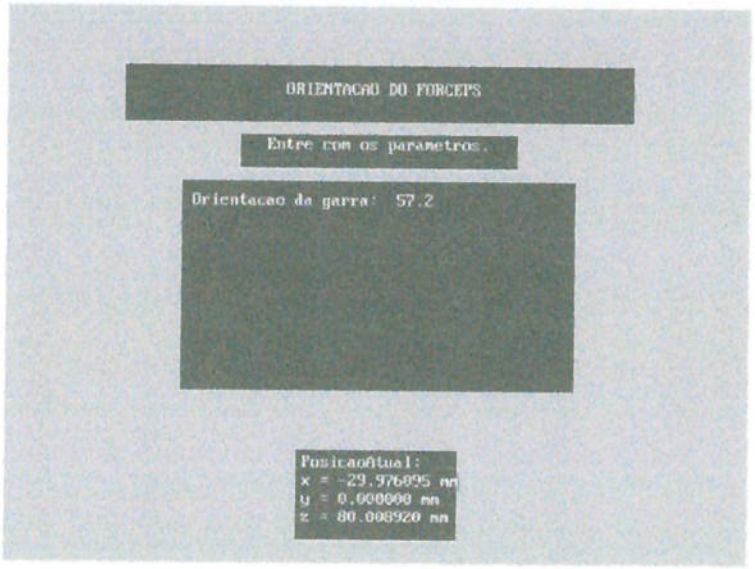
Anexo B

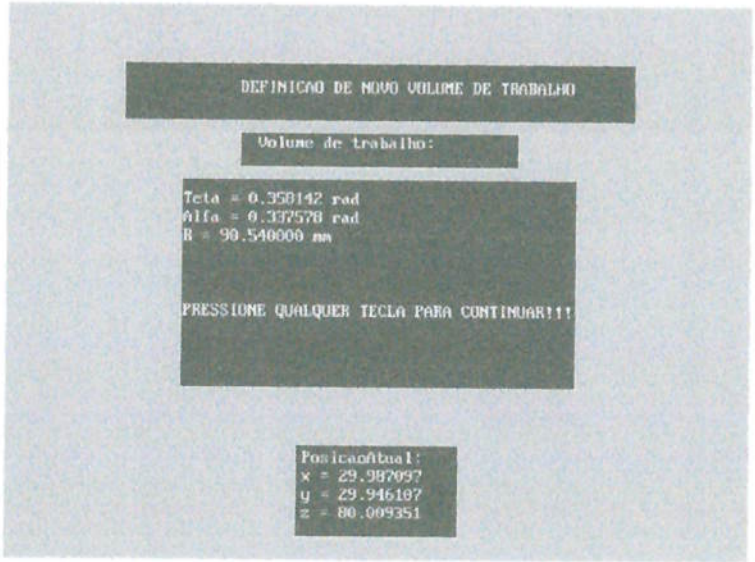
Telas do Programa desenvolvido






ACIONAMENTO RETILINEO
Entre com os parametros.
Destino x: 30
Destino y:
Destino z: 0
Tempo da movimentacao: 0000
PosicaoAtual:
x = 12.025540 mm
y = 0.000000 mm
z = 79.996171 mm


ORIENTACAO DO FORCEPS
Entre com os parametros.
Orientacao da garra: 57.2
PosicaoAtual:
x = -29.976095 mm
y = 0.000000 mm
z = 80.008920 mm


DEFINICAO DE NOVO VOLUME DE TRABALHO
Volume de trabalho:
Teta = 0.350142 rad
Alfa = 0.337578 rad
R = 90.540000 mm
PRESSIONE QUALQUER TECLA PARA CONTINUAR!!!
PosicaoAtual:
x = 29.987097
y = 29.946107
z = 80.009351

Apêndice A

Dados relativos ao Motor de Passo

MOTOR DATABASE				
MOTOR PART NO.		4034-329		
WINDING CONN.		SERIES		
DRIVE		3535		
BUS VOLTAGE		12 V. MIN		
		24 V. MID		
		35 V. MAX		
CURRENT		2.8 AMP		
RESOLUTION		FULL		
		HALF		
	SPEED		TORQUE N.m	
RESOLUTION	RPS	V MIN	V MID	V MAX
FULL STEP	1	1,4948591927	2,2238934554	2,3469056295
	2	0,7761347908	1,6203429088	2,0467898201
	3	0,5114679547	1,1417111544	1,6338304663
	4	0,3634578990	0,8485862780	1,2614750162
	5	0,2844391718	0,6831341977	1,0345874643
	10	0,1068412281	0,3090133604	0,4910600797
	15	0,0483716069	0,1833177979	0,3078128972
	20	0,0189249499	0,1206112476	0,2138236873
	25	0,0014829252	0,0829025789	0,1573312996
	30	-	0,0573397735	0,1196932463
	35	-	0,0393328249	0,0927181312
	40	-	0,0256334209	0,0722396407
	45	-	0,0152529447	0,0565630031
	50	-	0,0067084710	0,0440640624
	60	-	-	0,0251391125
	70	-	-	0,0114397085
	80	-	-	0,0014123097
	90	-	-	-
	100	-	-	-

HALF STEP	1	1,4225489365	2,0787786346	2,1632347541
	2	0,6187328757	1,6199192159	1,9957348247
	3	0,3700957546	1,0982826314	1,6228850662
	4	0,2580289806	0,7558681468	1,2562494704
	5	0,1989944355	0,5726915798	0,9843092393
	10	0,0683557891	0,2386803378	0,4078750388
	15	0,0247154196	0,1364291162	0,2434821908
	20	0,0014123097	0,0851622744	0,1636160777
	25	-	0,0550094625	0,1177160128
	30	-	0,0343191255	0,0862921221
	35	-	0,0194898737	0,0638363980
	40	-	0,0087563201	0,0475948366
	45	-	0,0001412310	0,0349546649
	50	-	-	0,0245035731
	60	-	-	0,0087563201
	70	-	-	-
	80	-	-	-
	90	-	-	-
	100	-	-	-