Centro Federal de Educação Tecnológica de Minas Gerais
Campus V – Divinópolis
Graduação em Engenharia Mecatrônica

Lucas Silva de Oliveira.

AUTOMAÇÃO DA TÉCNICA DE BRAQUITERAPIA DE ALTA DOSE POR AGULHAMENTO UTILIZANDO UMA MESA XY ORIENTADA POR IMAGEM RADIOGRÁFICA

Divinópolis
2013

Lucas Silva de Oliveira

AUTOMAÇÃO DA TÉCNICA DE BRAQUITERAPIA DE ALTA DOSE POR AGULHAMENTO UTILIZANDO UMA MESA XY ORIENTADA POR IMAGEM RADIOGRÁFICA

Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.
Eixo de Formação: Mecânica, Automação e Programação.

Orientador: Renato de Sousa Dâmaso
Co-orientador: Tarcísio Passos Ribeiro de Campos

Divinópolis
2013

Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG / *Campus V* - Divinópolis
Curso de Engenharia Mecatrônica

Monografia intitulada "Automação da Técnica de Braquiterapia de Alta Dose por Agulhamento Utilizando uma Mesa XY Orientada por Imagem Radiográfica" de autoria do graduando Lucas Silva de Oliveira, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Renato de Sousa Dâmaso - CEFET-MG / *Campus* Divinópolis - Orientador

---

Prof. M.Sc Gustavo Campos Menezes - CEFET-MG / *Campus* Divinópolis

---

Prof. Dr. Luiz Claúdio Oliveira - CEFET-MG / *Campus* Divinópolis

---

Prof. Dr. Sandro Trindade Mordente Gonçalves - CEFET-MG / *Campus* Divinópolis

---

Prof. Dr. Valter Júnior de Souza Leite
Coordenador do Curso de Engenharia Mecatrônica
CEFET-MG / *Campus* Divinópolis

Divinópolis - Abril de 2013

Dedico esta monografia aos meus pais que me deram muito apoio nos momentos mais difíceis da minha vida, a minha irmã e minha namorada que estiveram ao meu lado, e nunca mediram esforços para me ajudar, aos meus professores que me ensinaram que por mais que achamos que o nosso conhecimento já está bem profundo, estamos enganados, pois o conhecimento é algo que está sempre se renovando. Obrigado por tudo!

Agradecimentos

Agradeço,

em primeiro lugar a Deus que iluminou o meu caminho durante esta longa caminhada. Que me deu força de vontade, fé e coragem para superar todas as adversidades. A todos os professores do curso, que foram tão importantes na minha vida acadêmica, que atuaram efetivamente em meu processo de formação como engenheiro e que me auxiliaram no desenvolvimento desta monografia. Em especial, gostaria de agradecer ao Prof. Dr. Renato de Sousa Dâmaso e ao Prof. Dr.Tarcísio Passos Ribeiro de Campos, responsáveis pela realização deste trabalho.
Aos amigos e colegas, pelo incentivo, paciência e pelo apoio constantes.
Dedico esta, bem como todas as minhas demais conquistas, aos meus amados pais - José Francisco de Oliveira Neto e Maria da Conceição Silva de Oliveira, minha irmã - Cynthia Silva de Oliveira e a minha namorada - Marília Fernandes Miranda. Obrigado a todos vocês pela paciência, pelo incentivo, pela força e principalmente pelo carinho. Valeu a pena toda essa espera, todo sofrimento, todas as renúncias... Hoje estamos colhendo, juntos, os frutos do nosso empenho!
Muito Obrigado!

"Tenho a impressão de ter sido uma criança brincando a beira-mar, divertindo-me em descobrir uma pedrinha mais lisa ou uma concha mais bonita que as outras, enquanto o imenso oceano da verdade continua misterioso diante de meus olhos."

Isaac Newton

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos e Notação

LMI	Linear Matrix Inequality (desigualdade matricial linear)
LFT	Linear Fractional Transformation (transformação linear fracionária)
LPV	Linear Parameter-Varying (linear com parâmetros variantes)
IQC	Integral Quadratic Constraint (restrição de integral quadrática)

$\star$	indica bloco simétrico nas LMIs
$L > 0$	indica que a matriz L é simétrica definida positiva
$L \geq 0$	indica que a matriz L é simétrica semi-definida positiva
A	notação para matrizes (letras maiúsculas do alfabeto latino)
A'	($'$), pós-posto a um vetor ou matriz, indica a operação de transposição
$\mathbb{R}$	conjunto dos números reais
$\mathbb{Z}$	conjunto dos números inteiros
$\mathbb{Z}_+$	conjunto dos números inteiros não negativos
$\mathbb{N}$	conjunto dos números naturais (incluindo o zero)
$\mathbf{I}$	matriz identidade de dimensão apropriada
$\mathbf{0}$	matriz de zeros de dimensão apropriada
$g!$	símbolo (!), denota fatorial, isto é, $g! = g(g-1)\cdots(2)(1)$ para $g \in \mathbb{N}$
N	especialmente utilizada para denotar o número de vértices de um politopo
n	especialmente utilizada para representar a ordem uma matriz quadrada
Δ_N	simplex unitário de N variáveis
α	especialmente utilizada para representar as incertezas de um sistema

Capítulo 1

Introdução Geral

1.1 Introdução

O câncer é uma doença conhecida e comum nos dias atuais. Segundo dados do INCA - Instituto Nacional de Câncer, há uma estimativa de ocorrência de 257.870 novos casos em homens e 260.640 novos casos em mulheres para o ano de 2012 (Saúde [2013]). Inicialmente esses tumores poderão se localizar em diferentes partes do corpo, tais como: mama, próstata, colo do útero, traquéia, brônquios e pulmão, entre outros. Ainda de acordo com esse estudo, a maior incidência da doença em homens ocorrerá na próstata, com 60.180 novos casos. Enquanto nas mulheres o orgão mais atingido será a mama, com 52.680 novos casos.

Sabe-se também que o aumento de casos dessa doença não é uma exclusividade brasileira, mas sim um problema que afeta toda a população mundial. Segundo um estudo publicado pela Organização Mundial de Saúde, realizado pela Agência Internacional de Pesquisa do Câncer em Lyon na França, os casos de câncer aumentarão em 75% até o ano de 2030 (Heavens [2012]) . Esse significativo aumento deve-se a vários fatores: alimentação, sedentarismo e outros hábitos nocivos. Ainda segundo esse estudo, os casos que apresentarão as maiores altas nos índices são: o câncer de mama, próstata e colorretal. Em um outro estudo semelhante realizado pela Agência Internacional de Pesquisa do Câncer (IARC and CRUK [2012]), é apresentado no mapa mundial os tipos de câncer mais comuns em cada região, com esse estudo é possível relacionar o estilo de vida da sociedade de certa região com tipo de câncer.

Segundo o Centro de Combate ao Câncer, " tratamento oncológico é sempre muito individualizado. É importante observar as necessidades e possibilidades terapêuticas de cada paciente com câncer. O tratamento pode ter intenção curativa ou paliativa (alívio dos sintomas, com vistas a uma melhora da sobrevida e da qualidade de vida). Pode ser feito por cirurgia, radioterapia ou quimioterapia, de forma isolada ou combinada, dependendo do tipo celular do órgão de origem e do grau de invasão do tumor. Ou seja, as variações são muitas" (Câncer [2012]).

A escolha do tratamento mais adequado, consiste de uma série de análises realizadas pelo corpo clínico. Algumas das situações analisadas encontram-se apresentadas a seguir:

- Cirurgia - a definição do tratamento por cirurgia consiste da análise de alguns critérios, tais como: estágio da doença, suas características biológicas, região atingida, bem como as

condições do paciente (Grabarz and Hattori [2009]) é importante ressaltar que as cirurgias se modernizaram, assim elas se tornaram muito menos mutilantes e mais individualistas, buscando sempre tratar a menor região. As cirurgias podem ser realizadas em tratamento único ou em associação com a quimioterapia ou radioterapia.

- Quimioterapia - "consiste em nada mais nada menos do que a aplicação de medicamentos combinados ou não por quatro vias: intravenosa - o medicamento é ministrado diretamente na veia, geralmente no antebraço; oral - a droga pode ser ministrada em cápsulas, pílulas ou mesmo líquida, sendo ingerida pela boca; intramuscular - por meio de injeção; e intratecal - aplicada no líquor, líquido da medula raquidiana - localizada próximo à região lombar , para destruir células cancerosas no local. A quimio não somente busca eliminar o tumor e a impedir a sua multiplicação, mas também evita que as células doentes entrem na corrente sangüínea e atinja outros órgãos, o que provocaria a metástase" (INCA [2010]). Dentro da quimioterapia o corpo clínico pode optar por diferentes técnicas, sendo elas: uso de medicamentos inteligentes, hormonioterapia, terapia biológica entre outras.

- Radioterapia - esse método utiliza a aplicação direcionada de feixes de radiações ionizantes (Grabarz and Hattori [2009]). A aplicação de doses de radiação sobre o local a ser tratado, busca eliminar todas as células tumorais, causando os menores danos possíveis aos tecidos adjacentes a região infectada. A radioterapia se divide em duas grandes áreas:

  - Radioterapia Externa - quando o corpo clínico decide realizar a radioterapia externa, é possível aplicá-la segundo os métodos: Radioterapia Tridimensional Conformada (3D-CRT), Radioterapia de Intensidade Modulada (IMRT), Feixe de Prótons ou Radioterapia Estereotáxica, para maiores informações sobre esses métodos consultar (Grabarz and Hattori [2009]).
  - Radioterapia Interna - com o uso desse método, também denominado de braquiterapia. O corpo clínico poderá optar em realizar o implante permanente ou temporário de sementes radioativas no tecido tumoral. Sendo necessário em algumas situações a internação do paciente (Salvajoli and Salvajoli [2012]). Uma das vantagens dessa técnica quando comparada á técnica anterior, consiste no fato desse tratamento não agredir as células saudáveis adjacentes (INCA [2002]). Isso representa para o paciente, uma recuperação em um tempo menor. Além, da redução ou eliminação dos efeitos colaterais, tais como: náuseas, dores e necessidade da realização de cirurgia.

Dentre as técnicas de radioterapia interna, podemos destacar a Técnica de Braquiterapia de Alta Taxa de Dose. Esse é um antigo método utilizado no tratamento de câncer, que foi introduzido no Brasil em 1991 (Esteves et al. [2004]). Essa técnica é uma modalidade terapêutica, que faz uso de fontes radioativas em contato direto com a região a ser tratada. O tratamento tem como objetivo administrar altas doses de radioatividade apenas nos tecidos doentes, de modo a causar um dano mínimo aos tecidos saudáveis adjacentes (Anthony et al. [1998]). Devido ao tamanho reduzido das sementes radioativas, cerca de 1,6 mm de comprimento com diâmetro de 0,3 mm (Costa and Campos [2007]) é possível realizar vários implantes intersticiais dentro do volume ocupado pelo tecido doente. Esses implantes podem ser aplicados, utilizando cateteres

plásticos ou por agulhamentos (Esteves et al. [2004]). Quando a técnica adotada for a de agulhamento, o processo exige o uso da placa crivada. Um exemplo de placa crivada Ã© mostrada na Figura 1.1.

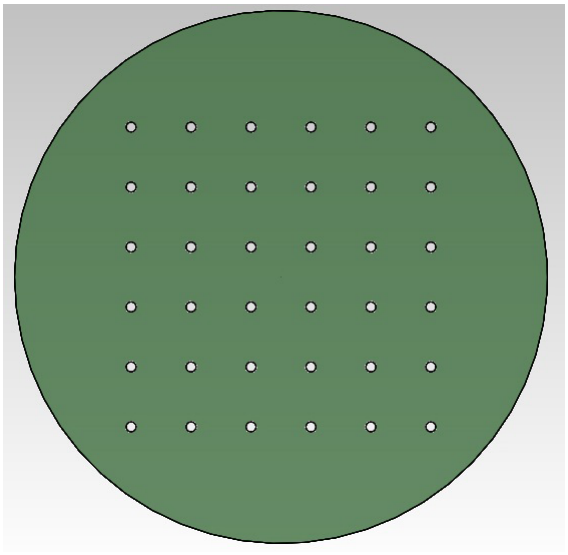


Figura 1.1: Placa crivada utilizada na técnica de braquiterapia por agulhamento

A placa crivada consiste de uma placa com furos equidistantes. Por esses furos será possível que o corpo clínico realize a inserção da agulha e tenha a garantia do posicionamento do material dentro do tecido. Uma importante característica dessas placas, consiste no fato dos furos apresentarem uma distância fixa entre os furos: 10, 8 ou 5 mm. Essa característica faz com que a técnica apresente certa restrição prática. Por exemplo: dada uma situação em que determinada região do tumor devá receber uma dose mais elevada de radioatividade. Para tal é necessário que naquela região, o tecido receba um número maior de sementes. Um possível método para solucionar o problema, consiste na introdução das sementes utilizando um passo menor entre os furos. O que é impossível, visto que a placa crivada apresenta um passo constante para a inserção das sementes.

Em todo o mundo vários pesquisadores vêm estudando maneiras e técnicas que possibilitem melhorar os índices dos resultados nos procedimentos aplicados à saúde. Alguns desses estudos estão voltados para automatização do procedimento empregado na técnica de braquiterapia por agulhamento, conforme pode ser verificado em (Trejos et al. [2008]) e (Meltsner et al. [2007]). Nos estudos publicados, pode-se observar principalmente uma preocupação em eliminar o contato do clínico com o material radioativo. Um desses grupos de estudo encontra-se no Departamento de Engenharia Nuclear da Universidade Federal de Minas Gerais (UFMG). Seguindo essa linha de pesquisa, o grupo coordenado pelo Professor Doutor Tarcísio Passos Ribeiro Campos estuda, desde 1993, vários processos e técnicas para desenvolvimento do tratamento de câncer. O estudo desenvolvido pelo grupo aborda, desde o efeito da radioatividade nos tecidos até a automatização dos procedimentos. Nessa linha de estudos de automação do procedimento para a técnica de braquiterapia e em parceria com o grupo de estudos do Departamento de Engenharia Nuclear da UFMG é proposto nesse trabalho o desenvolvimento de um sistema eletro-mecânico que: possibilite a eliminação da placa crivada e também a alteração da

dose aplicada ao longo do procedimento de modo a otimizar o tratamento na região central do tumor.

O projeto será composto:

1. pela elaboração do projeto mecânico do dispositivo - uma mesa de coordenadas cartesianas XY.

2. desenvolvimento do software de controle e operação dos motores.

3. desenvolvimento de um software para tratar a imagem obtida através do raio X e, a partir dela, gerar o sistema de coordenadas para realização do implante das sementes radioativas.

A precisão do posicionamento mecânico será garantido pela estrutura mecânica do dispositivo. Essa, por sua vez será desenvolvida utilizando guias lineares de precisão e *drive* de acionamento dos motores que possuem configuração de micro passo. Assim, será possível posicionar o suporte da seringa em qualquer coordenada dentro da área de trabalho da mesa de coordenadas. Além de incluir essas características, o sistema proposto apresentará também a flexibilidade de se posicionar em qualquer local do espaço em torno do paciente, uma vez que o dispositivo será fixado em um braço articulado. A fixação do dispositivo no braço articulado, permitirá que a técnica seja aplicada em qualquer plano de referência, eliminando a restrição encontrada na atual aplicação da técnica (Roberto et al. [2005]) Nos próximos capítulos, serão descritos cada um dos itens: projeto mecânico, software para tratamento da imagem e desenvolvimento do sistema de coordenadas e por fim o software para acionamento e posicionamento dos motores.

Capítulo 2

Estrutura Mecânica

2.1 Estrutura Mecânica

O sistema mecânico foi projetado visando atender as necessidades operacionais, ou seja, um dispositivo com acionamento remoto, que proporcione a redução do tempo de exposição do clínico à radioatividade emitida pelas sementes, que ofereça a liberdade de posicionamento da agulha dentro da área que se encontra em tratamento, que garanta repetibilidade e fidelidade ao tratamento planejado. Assim, foi proposto o desenvolvimento e construção de uma mesa de coordenadas cartesianas XY, conforme pode ser observado na Figura 2.1.

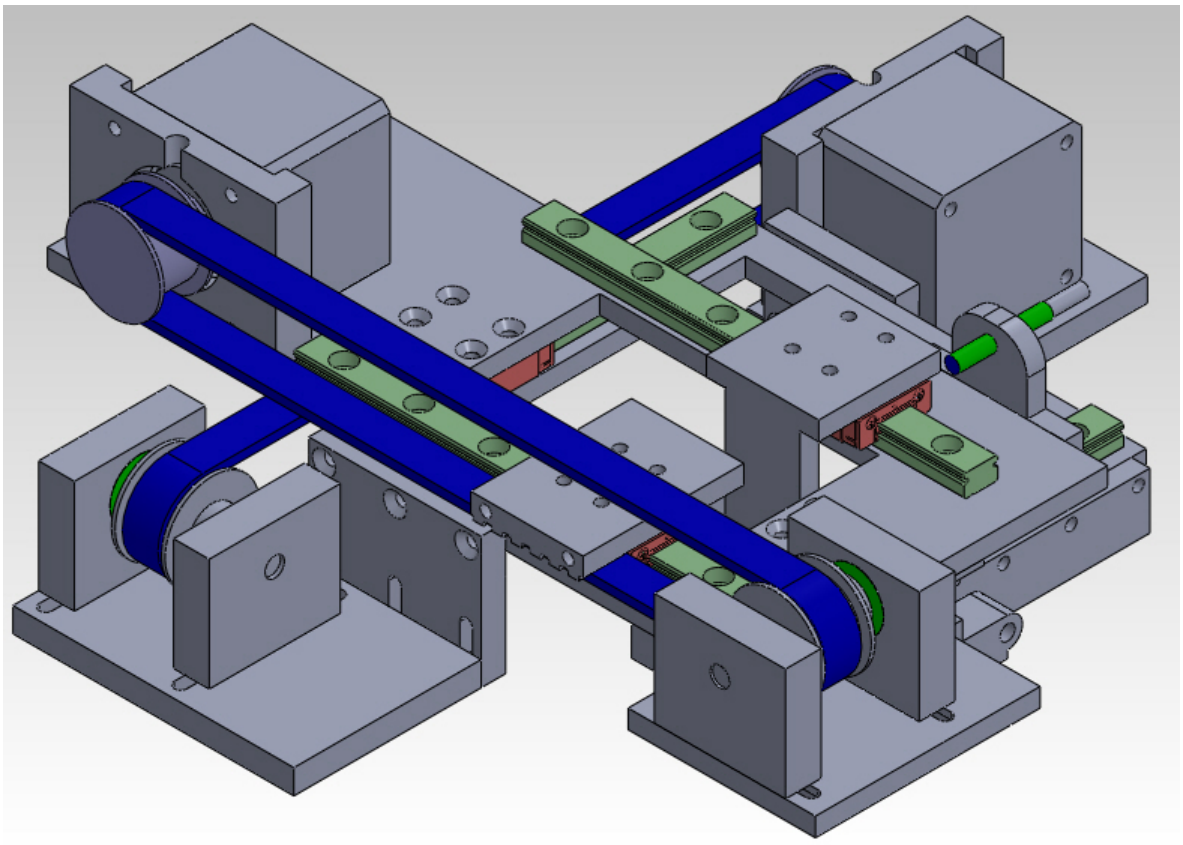


Figura 2.1: Mesa de coordenadas cartesianas XY proposta no trabalho.

Conforme pode ser observado na Figura 2.1, a mesa de coordenadas proposta consiste basicamente de um conjunto de dois eixos ortogonais movidos por dois motores de passo. Com esse sistema poderemos garantir a precisão do posicionamento devido ao sistema de acionamento dos motores ser composto por um *driver* que possui configuração de micro passos e pelo sistema de movimentação linear utilizar guias lineares de precisão. Poderemos garantir também a confiabilidade exigida pelo procedimento, através do processo de execução do *software* de comando dos motores. Este permitirá ao usuário avançar para o próximo ponto onde ocorrerá o implante das sementes ou retornar a um ponto em que já tenha ocorrido o procedimento.

Um dos requisitos exigido do projeto mecânico está relacionado com o tamanho da estrutura, a qual deve apresentar as menores dimensões possíveis. Assim, todo o projeto foi desenvolvido visando atender a esse requisito.

2.2 Mesa de Coordenadas Eixo X

O protótipo virtual da mesa de coordenadas do eixo X é apresentado na Figura 2.2. A mesa apresenta as seguintes dimensões externas: 228×157 mm. Essa mesa é composta por um motor de passo de tamanho padrão *NEMA 17*, guias lineares de precisão e sistema de acionamento por correia sincronizada.

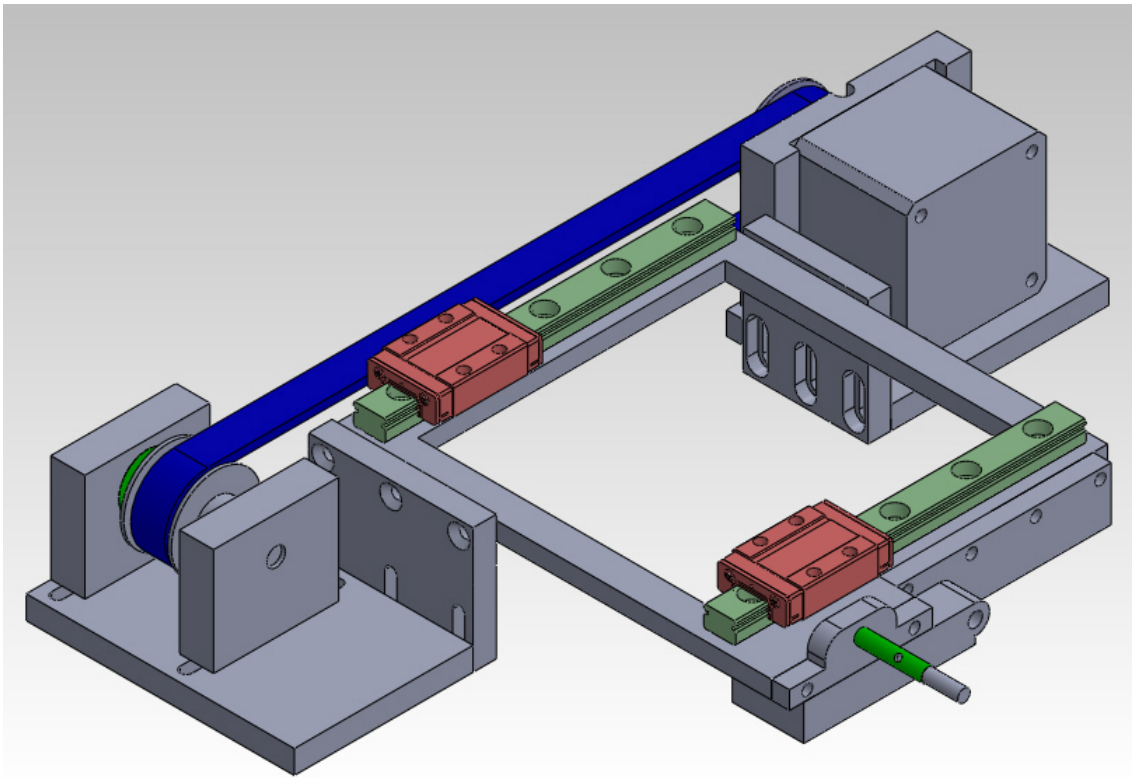


Figura 2.2: Mesa de coordenadas do eixo X

O motor *NEMA 17* foi escolhido por ser o menor motor de passo presente no mercado com torque suficiente para atender com segurança o dispositivo. As correias foram definidas através das características mecânicas e funcionais necessárias ao projeto, tais como: precisão na

movimentação e durabilidade. Após verificar e analisar os diferentes tipos de correias presentes no mercado, optou-se pela correia sincronizada. As correias sincronizadas apresentam boa eficiência na transmissão da potência do motor, grande precisão de posicionamento, por serem abundantes e de baixo custo no mercado (Shigley et al. [2005]).

A escolha da guia linear a ser usada baseou-se nos mesmos princípios utilizados na determinação da correia sincronizada. Além desses critérios, ainda foi observado que a limpeza do dispositivo após o uso poderia resultar na oxidação das guias, o que resultaria em uma redução na vida útil desse componente. Para evitar tal problema, foram definidas guias lineares construídas em aço inox. Essas guias foram desenvolvidas para operarem em ambientes e atmosferas agressivas ao metal, desse modo as guias em aço inox são dispositivos adequados para o ambiente no qual esse sistema de posicionamento deverá operar, além de tolerar bem os produtos e processos de limpeza e de esterilização.

Outra opção para o acionamento seria a utilização de fusos. Utilizando esse sistema de acionamento, o tamanho do dispositivo podería ser reduzir significativamente. Estima-se que possa reduzir o tamanho da mesa em aproximadamente 50 mm em cada eixo. Porém, o fuso apresentou um valor de mercado superior a quatro vezes ao custo para aquisição das correias, polias e dos conjuntos de guias lineares. Assim, foi decidido aumentar o tamanho do dispositivo para reduzir os custos.

A estrutura mecânica da mesa deveria ser construída utilizando aço inox, pelo mesmos motivos apresentados na escolha da guia linear. Porém, devido ao alto valor comercial do aço inox e dificuldade na execução do processo de usinagem, foi decidido desenvolver a estrutura em alumínio. O alumínio é um metal que não oxida tão facilmente como o aço carbono, apresenta boa usinabilidade e resistência mecânica. Além disso possui baixo custo quando comparado com o aço inox. Todos os parafusos utilizados para fixação das peças são usinados em aço inox e também são produtos de catálogo disponíveis em diferentes fornecedores.

A utilização do alumínio na estrutura da mesa proporcionou um ganho importante no projeto. O alumínio reduziu significantemente a massa da estrutura. Essa propriedade é importante, uma vez que a mesa será fixada em um braço móvel articulado. O aumento da massa resulta em um torque maior e consequentemente em um braço articulado mais rígido.

2.3 Mesa de Coordenadas Eixo Y

Seguindo os mesmos critérios adotados para desenvolver a mesa de coordenadas do eixo X, desenvolveu-se também o projeto da mesa de coordenadas do eixo Y. O protótipo virtual dessa mesa se encontra apresentado na Figura 2.3.

A mesa de coordenadas Y apresenta as seguintes dimensões externas: 234×125 mm. Foi possível reduzir uma das dimensões, devido ao fato da mesa Y ter que movimentar a placa de suporte para a seringa, a qual apresenta dimensões reduzidas.

A escolha dos componentes para construção foram os mesmos utilizados para determinação dos componentes da mesa de coordenadas do eixo X. Assim, a mesa do eixo Y, será composta por um motor *NEMA 17*, correia sincronizada, guias lineares em aço inox e estrutura em alumínio e aço inox.

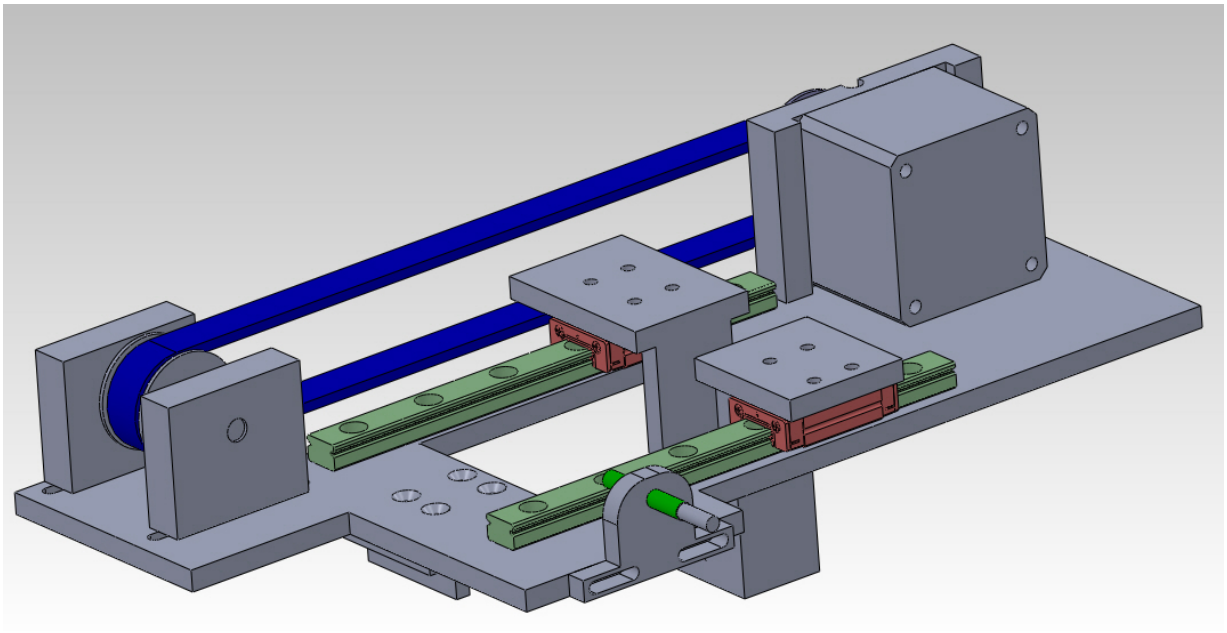


Figura 2.3: Mesa de coordenadas do eixo Y

2.4 Mesa Completa

A união das mesas de coordenadas X e Y, resulta em um conjunto com as seguintes dimensões externas: $228 \times 234 \times 87.5$mm. Essa mesa cartesiana de coordenadas será fixada em um braço articulado, que possibilitará ao clínico escolher o melhor posicionamento do dispositivo em relação ao paciente. A Figura 2.4 apresenta o dispositivo em sua montagem completa.

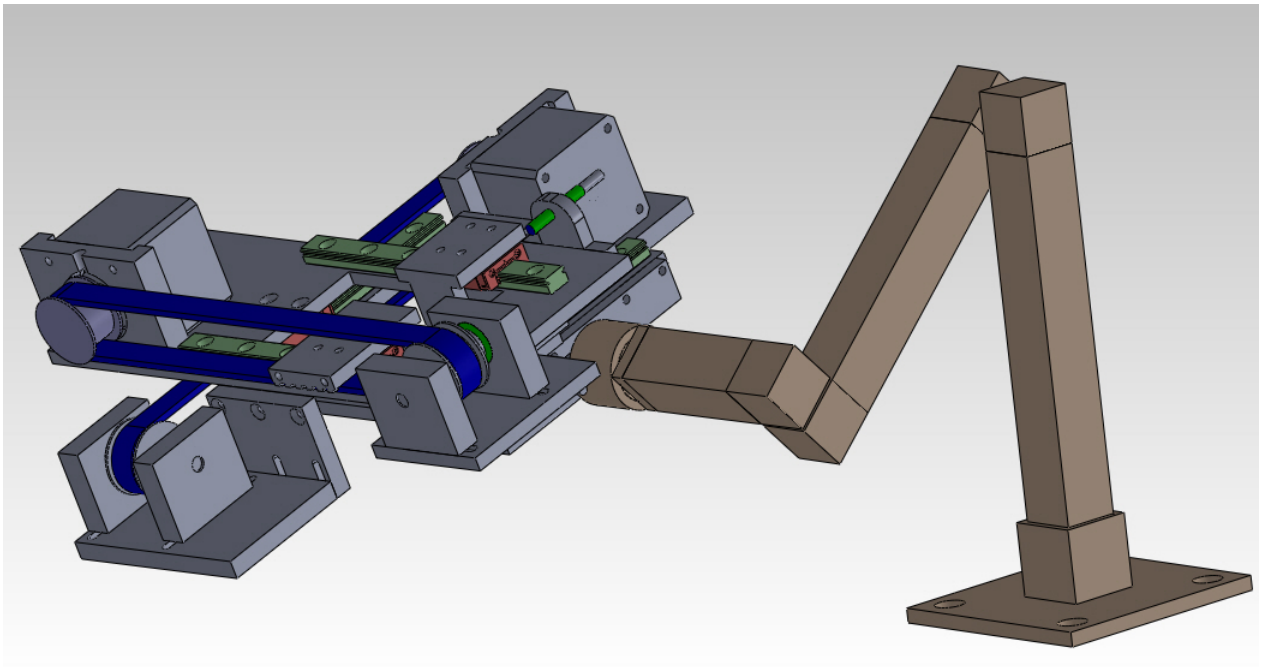


Figura 2.4: Mesa de Coordenadas Cartesianas XY fixada ao braço articulado.

Nesse conjunto a agulha poderá operar em qualquer ponto dentro de uma área de um retângulo de dimensões: $50 \times 49,6$ mm. A liberdade de manipulação da agulha dentro dessa área é um ponto interessante quando se compara o método atual de aplicação com o método proposto no projeto em questão. O método atual possui uma área disponível para aplicação da técnica de braquiterapia com as dimensões de 50×50 mm, porém com pontos fixos e já marcados para inserção da agulha. Normalmente a distância entre um ponto e outro é de 10 mm. Com a mesa de coordenadas XY a área de atuação da agulha permaneceu praticamente inalterada, porém, houve um ganho quanto à liberdade para realização das inserções, pois a distância entre os pontos será definida pelo usuário, através do *software* de processamento da imagem. Como pode ser observado na Figura 2.4, o braço articulado acrescentará ao sistema mecânico projetado a completa liberdade para movimentação. Com ele será possível operar em qualquer plano em torno do paciente.

2.5 Referenciamento do Sistema Mecânico em Relação à Imagem

Para que o software possa processar a imagem e criar um sistema de coordenadas, era preciso desenvolver uma relação mecânica com a imagem - Raio X. A solução encontrada, consiste em substituir a placa de suporte da seringa por uma placa de acrílico com uma marcação de metal das direções dos eixos X e Y. Assim, antes do clínico obter a imagem do Raio X da região, o mesmo deve posicionar o sistema mecânico, e na sequência realizar a operação com o Raio X. A Figura 2.5 apresenta a mesa de coordenadas Y com a placa de acrílico.

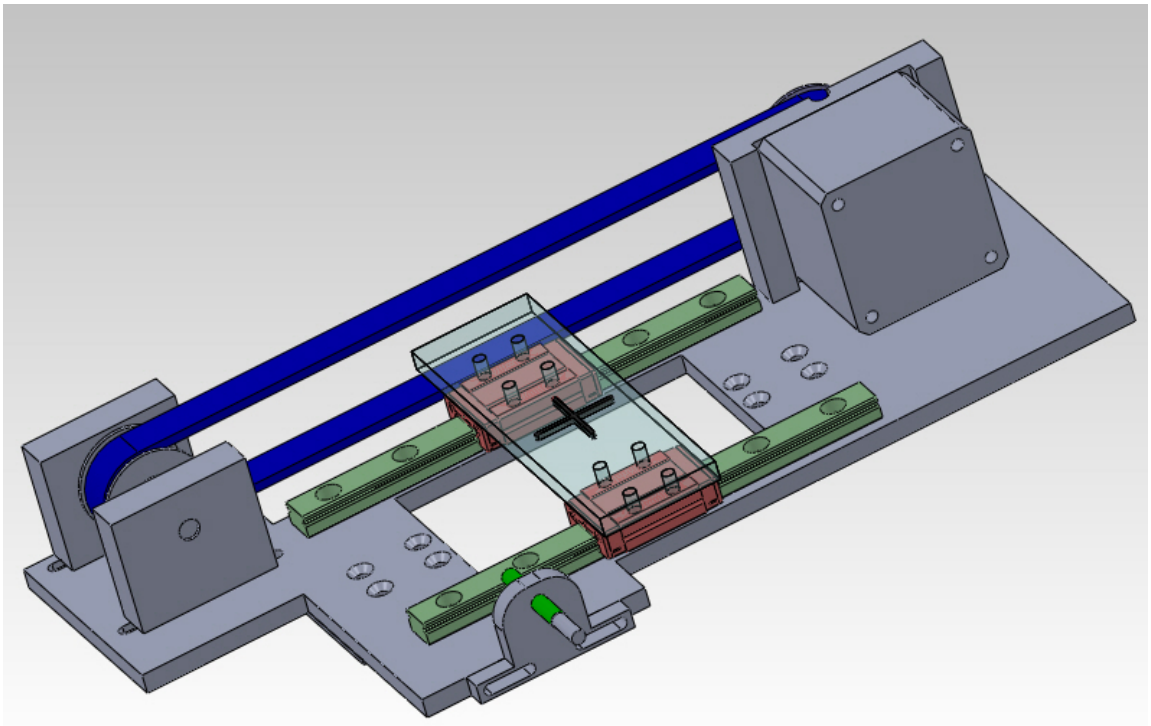


Figura 2.5: Mesa de coordenadas do eixo Y com a placa de acrílico.

Conforme pode ser observado na Figura 2.5, a placa de acrílico marca a direção dos movimentos X e Y. E será por essa marcação poderá relacionar a imagem com o sistema mecânico no desenvolvimento do sistema de coordenadas.

2.6 Sistema Eletrônico

O acionamento dos motores acontecerá utilizando o programa de interface com o usuário para controle dos motores. Esse programa desenvolvido utilizando o *software MatLab*, enviará os sinais para o drive de controle do motores utilizando as placas: *PCI6221* e *ASSY183030C-01* da *National Instruments*. As placas foram definidas partindo do critério que atenda as necessidades dos comandos, sendo essas: velocidade e tipo de sinal a ser transmitido analógico e digital. E também pelo fato de estarem disponíveis no laboratório.

O drive especificado para controlar os motores, foi o modelo *AKDMP5-1.7A*, fornecido pela *Akiyama Automação*. Esse drive é o modelo indicado para trabalhar com o motor de passo *NEMA 17*. Algumas características interessantes desse drive, encontram-se disponíveis na Tabela 2.1.

Descrição	Min.	Max	Unidade
Tensão de alimentação	12	24	V
Corrente de entrada	0	1,7	A
Corrente de saída	0	1,7	A
Frequência de operação	0	50	KHz

Tabela 2.1: Parâmetros das características do drive *AKDMP5-1.7A*

Além das características apresentadas na Tabela 2.1, o drive possui divisão de micro passo com as seguintes configurações: 1/2, 1/5, 1/10, 1/20 e 1/40. Assim, para a configuração de 1/40 seria possível dividir a revolução em 8000 micro passos. O que nos permite realizar pequenos deslocamentos. Para mensurar esse efeito no projeto, vamos adotar que a polia do motor apresente um diâmetro de 25 mm, então, em uma revolução completa o deslocamento linear será dado pela Equação (2.1).

$$d_{linear} = 2\pi r \tag{2.1}$$

O que resulta num deslocamento linear de $78,5398$ mm. Dividindo o deslocamento linear pelo número de micro passos, obtemos a relação apresentada na Tabela 2.2.

Divisão	Nº de Micro Passos	Deslocamento por pulso (mm)
1/2	400	0,1963
1/5	1000	0,0785
1/10	2000	0,0393
1/20	4000	0,0196
1/40	8000	0,0098

Tabela 2.2: Parâmetros das características do drive *AKDMP5-1.7A*

Ou seja, a cada pulso para a configuração de 1/40 é obtido um deslocamento linear de $0,0098$ mm. Que resulta em um deslocamento de $0,0393\%$ em relação ao deslocamento por revolução.

2.7 Relação dos Materiais Necessários Para Construção Mecânica

A Tabela 2.3 apresenta os materiais, insumos e serviços necessários para a construção da estrutura mecânica.

Descrição	Fornecedor	Quantidade	Valor Unid. (R\$)	Custo (R\$)
Trilho guia linear	Rac Mov. linear	4	150,00	600,00
Patim	Rac Mov. linear	4	170,00	680,00
Motor $AK17/1.10F6LN1.8$	Neoyama	2	40,00	80,00
Drive AKDMP5 − 1.7A	Neoyama	2	170,00	240,00
Alumínio Liga 6351 3/8”	Admetal	500 × 500 mm	60,00	60,00
Alumínio Liga 6351 1/4”	Admetal	500 × 500 mm	40,00	40,00
Alumínio Liga 6351	Admetal	110 × 40 × 70 mm	80,00	80,00
Acrílico Transparente	Bestplugs	100 × 30 × 10 mm	50,00	50,00
Correias 160 XL 037	Correias Schneider	1	35,00	35,00
Correias 162 XL 037	Correias Schneider	1	35,00	35,00
Polia 12 XL 037	Correias Schneider	4	50,00	200,00
Sensor $SIEN-4B-PS-K-L$	Festo Automação	2	120	240,00
Parafusos Aço Inox	Diversos	-	150	150,00
Usinagem	Diversos	-	1000	1000,00
Total				3490,00

Tabela 2.3: Tabela de materiais, insumos e serviços.

Esse levantamento de custo foi realizado na época de proposição do projeto, sendo que o Prof. Tarcísio, co-orientador do projeto se empenhou na obtenção da verba necessária para aquisição dos componentes. No entanto, como a liberação desses recursos não foi efetivada posteriormente junto aos órgãos de fomento, não foi possível a construção física do dispositivo projetado no prazo de realização desse TCC. Essa construção será deixada como perspectiva

para trabalhos futuros, onde acredita-se que algumas das informações consolidadas ao longo desse trabalho possam ser úteis para uma nova submissão do projeto aos órgãos de fomento.

Capítulo 3

Software de Processamento de Imagem

3.1 Conceito do *Software*

Esse software é fundamental para o projeto proposto, pois ele será responsável por gerar um sistema de coordenadas na imagem, mas que sejam coincidentes com as coordenadas reais do dispositivo, permitindo, que o tumor receba os implantes nos nos pontos determinados pela simulação computacional na etapa de planejamento do tratamento.

Para que o programa opere corretamente é necessário que, ao ser realizado o exame com o equipamento de Raio X, seja obtida uma imagem similar à imagem apresentada na Figura 3.1.

Figura 3.1: Exemplo de resultado de imagem do Raio X esperado.

Conforme pode ser observado na Figura 3.1, o que permitirá ao usuário definir corretamente esse sistema de coordenadas é a marca deixada pela cruz em metal presente na placa de acrílico, conforme apresentado na Figura 2.5. Esta por sua vez marcará o centro do sistema mecânico no resultado obtido com o exame de raio X. Assim, será possível referenciarmos o centro do sistema mecânico com a imagem, e a partir desse, por desenvolver o sistema de coordenadas que possa ser seguindo e executado pelos motores.

3.2 O Software

O *software* apresentado na Figura 3.2 realizará o processamento da imagem. Esse *software* é responsável em desenvolver os cálculos e transformações a partir da imagem do Raio X, gerando desse modo as coordenadas onde os implantes das sementes serão realizados. Esse programa foi desenvolvido utilizando as linguagens de programação do *MatLab*. O código do programa e das suas funções encontram-se apresentado no Anexo A.

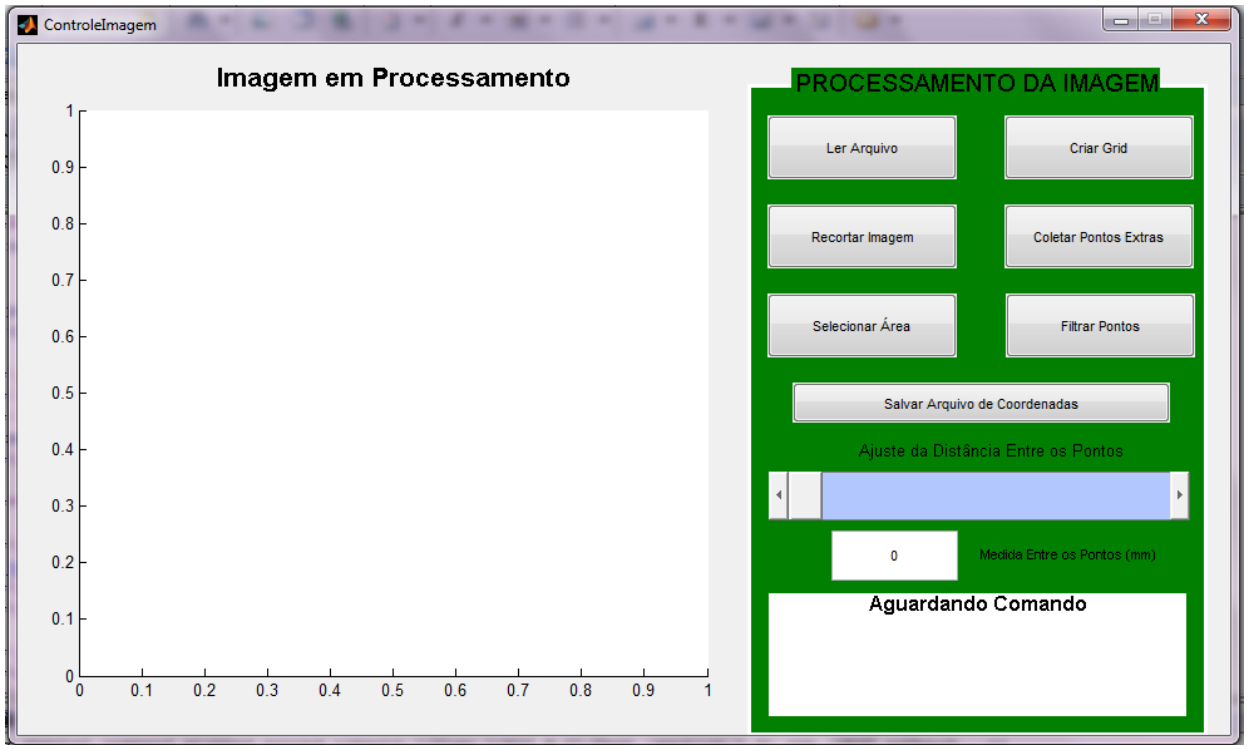


Figura 3.2: Janela de interface com o usuário do *software* de processamento de imagem.

Conforme pode ser observado na Figura 3.2, o programa apresenta uma janela de interface com o usuário com poucos botões, mas que possuem as funções necessárias para gerar as coordenadas de implante das sementes. O *software* foi desenvolvido de modo a instruir o usuário, a fim de facilitar seu uso. O detalhamento do funcionamento do *software* será apresentado e discutido na próxima Seção 3.3.

3.3 Etapas e Operações no Processamento da Imagem

Para abrir o programa de processamento de imagem, é necessário executar inicialmente o MatLab e compilar o código do programa apresentado no Anexo A. Assim, quando o MatLab terminar a compilação do código, a janela de interface do usuário apresentado na Figura 3.2 será aberta, nesse momento o usuário poderá iniciar o processamento da imagem.

As etapas e as ações a serem executadas pelo usuário, se encontram apresentadas no fluxograma da Figura 3.3.

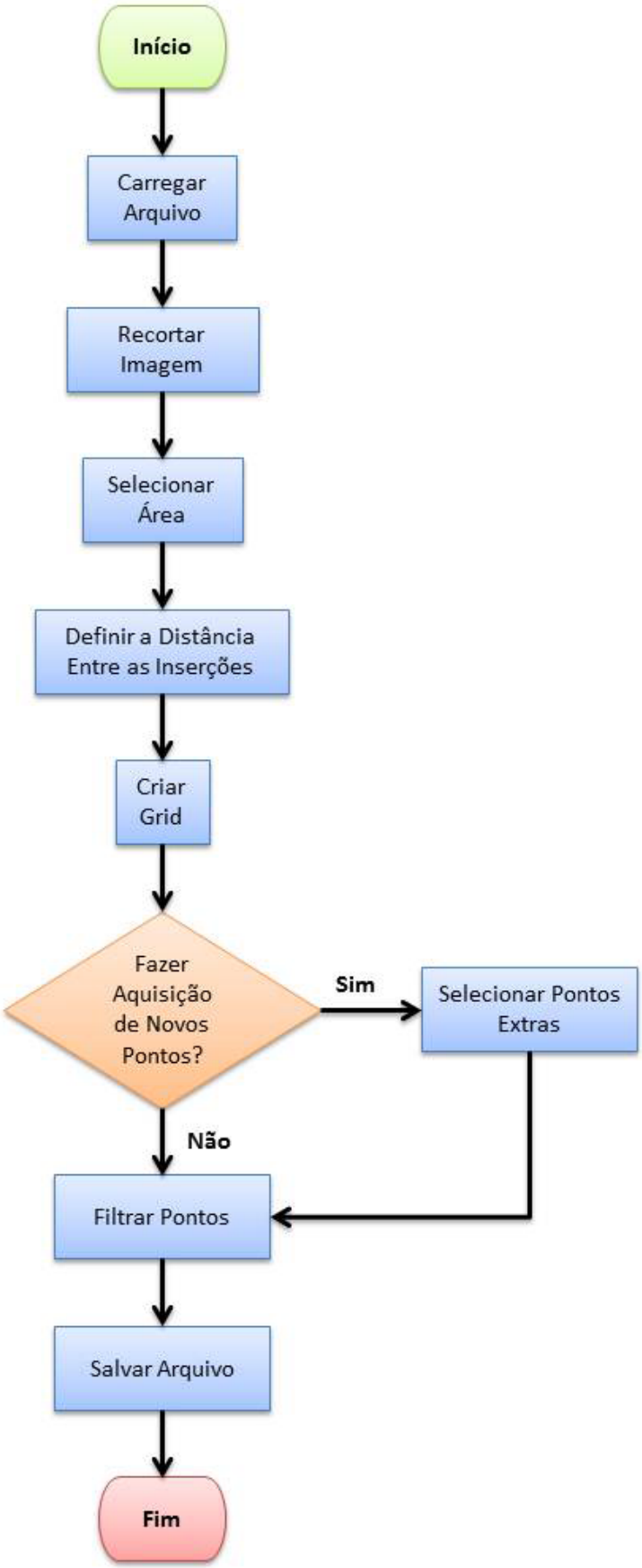


Figura 3.3: Fluxograma de operação do software de processamento de imagem.

Conforme pode ser observado, cada etapa corresponde a um dos botões apresentados na janela de interface da Figura 3.2. Assim, para processamento correto da imagem, basta que o usuário siga os passos na sequência apresentada no fluxograma da Figura 3.3.

O primeiro passo para iniciarmos o processamento da imagem, consiste em carregar o arquivo que contém a imagem a ser analisada. Para tal, é necessário que o usuário pressione o botão *"Ler Arquivo"*. Quando o botão for pressionado, será aberta uma janela para busca do arquivo. É importante observar que nesta operação o usuário deverá buscar uma imagem no formato *.jpg*. O código está preparado para analisar somente este formato de arquivo. Assim, quando a imagem do Raio X for salva em arquivo, a mesma deverá ser salva em *.jpg*. Se porventura for carregado um arquivo em outro formato o programa exibirá uma mensagem de erro, no *workspace do MatLab*. Ao final do procedimento a janela de interface apresentará a imagem carregada, conforme pode ser observado na Figura 3.4.

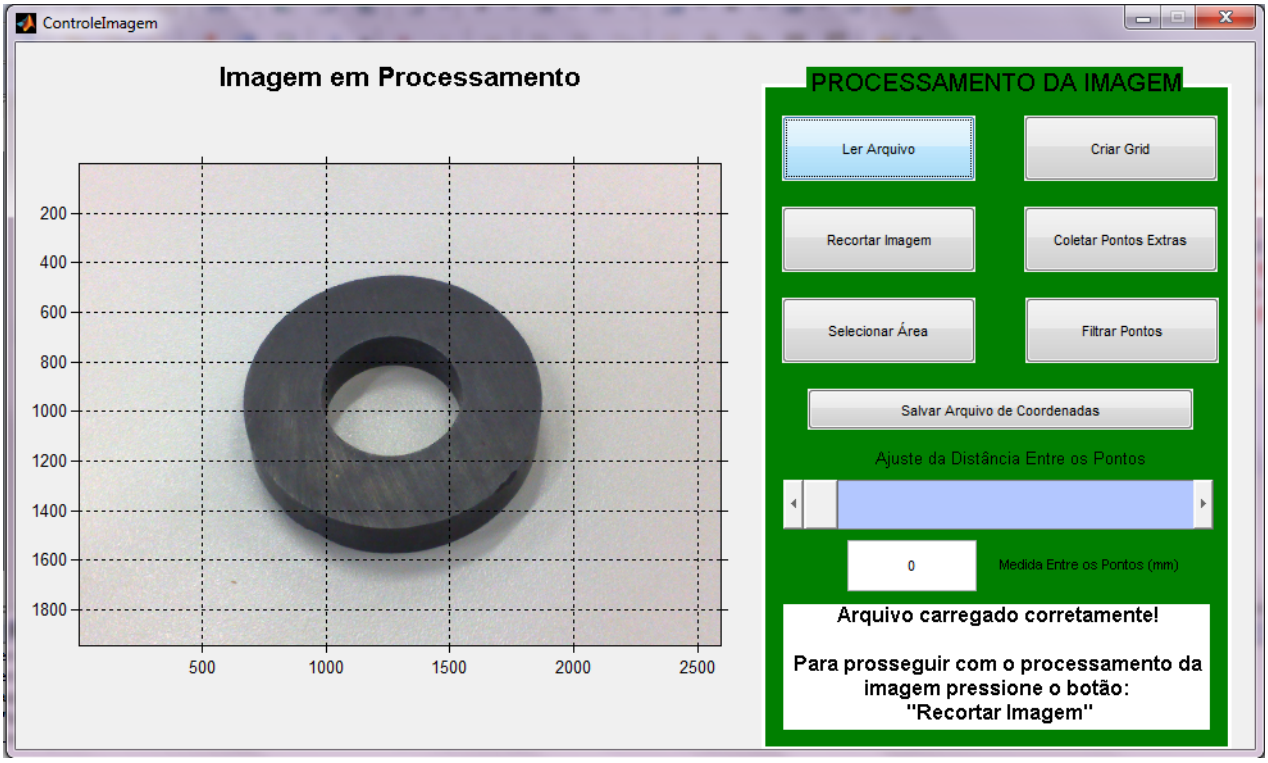


Figura 3.4: Janela de interface com o arquivo carregado

Analisando a imagem da Figura 3.4, observa-se que o arquivo de imagem carregado se encontra submetido a um par de eixos cartesianos. Esses eixos apresentam a imagem disposta em suas coordenadas, utilizando como unidade de medida o *pixel*. Isso acontecerá, uma vez que o arquivo se encontrar carregado. O código portanto realizará a análise da imagem e obterá todas as informações necessárias a realização das operações, tais como: resolução da imagem, número de pixels no eixo X e número de pixels no eixo Y, entre outras informações. Assim, antes de exibir a imagem ao usuário o programa, acrescentará algumas das informações processadas da imagem, como por exemplo sua distribuição em pixels e guardará outras informações necessárias para o processamento dos cálculos e ações.

O próximo passo a ser realizado consiste em recortar a imagem de modo a selecionar a região

de interesse. Para tal, o usuário deverá pressionar o botão *"Recortar Imagem"*. Assim, surgirá sobre a imagem um cursor, onde o usuário deverá indicar a posição central. Esse ponto pode ser observado na Figura 3.5.

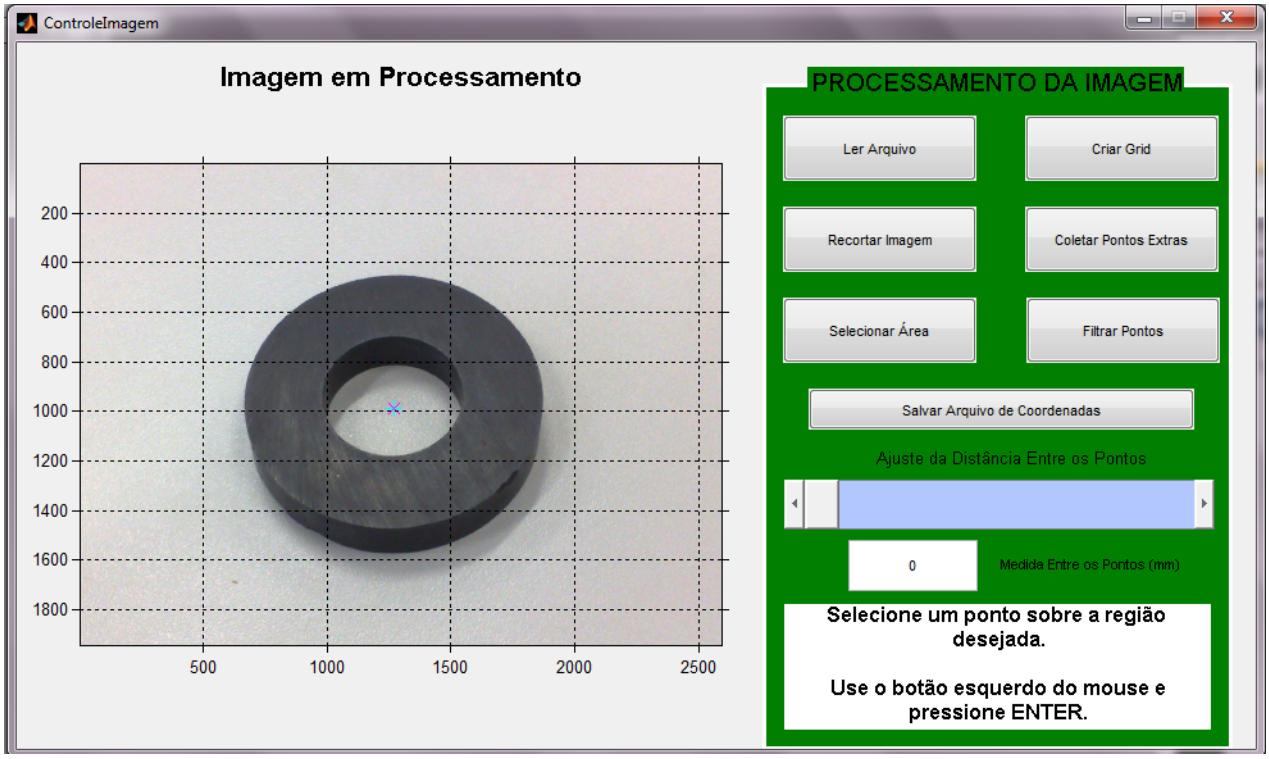


Figura 3.5: Imagem com o curso para seleção do ponto central.

É importante observar que esse ponto deverá ser aplicado sobre a marca em cruz, deixada pela cruz em metal durante o Raio X. Mas, como a estrutura mecânica não foi construída, não foi possível apresentar melhor esse procedimento. Porém, deve-se dedicar muita atenção a esse procedimento, pois ele é de suma importância para o desenvolvimento do sistema de coordenadas relacionando o sistema mecânico com a imagem. Esse ponto que o usuário definirá será interpretado pelo código, como sendo o ponto central de todo o sistema mecânico, o qual será representado na imagem pela marca em cruz de metal. Uma vez que o ponto for definido, o usuário pressionará a tecla *"enter"* e a imagem será recortada nas dimensões de $80 \times 69,6$ mm. Assim, a imagem estará no tamanho real dos eixos X e Y dentro dos limites de movimentação de cada eixo. E o canto inferior esquerdo da imagem será considerado a origem de coordenadas do sistema mecânico. Após realização completa da operação, a janela de interface com o usuário apresentará a uma imagem conforme pode ser observado na Figura 3.6. Como pode-se observar na Figura 3.6, o sistema de coordenadas sobre a imagem não apresenta a origem do sistema mecânico conforme esperado. Isso acontece porque o sistema mecânico utilizará as coordenadas medidas em milímetros, enquanto todo processamento da imagem será realizado usando o *pixel* como unidade de medida. Desse modo, o sistema será convertido em milímetro somente, após todos os pontos de inserção serem definidos. Até esse momento continuaremos utilizando o sistema de referência em *pixel*.

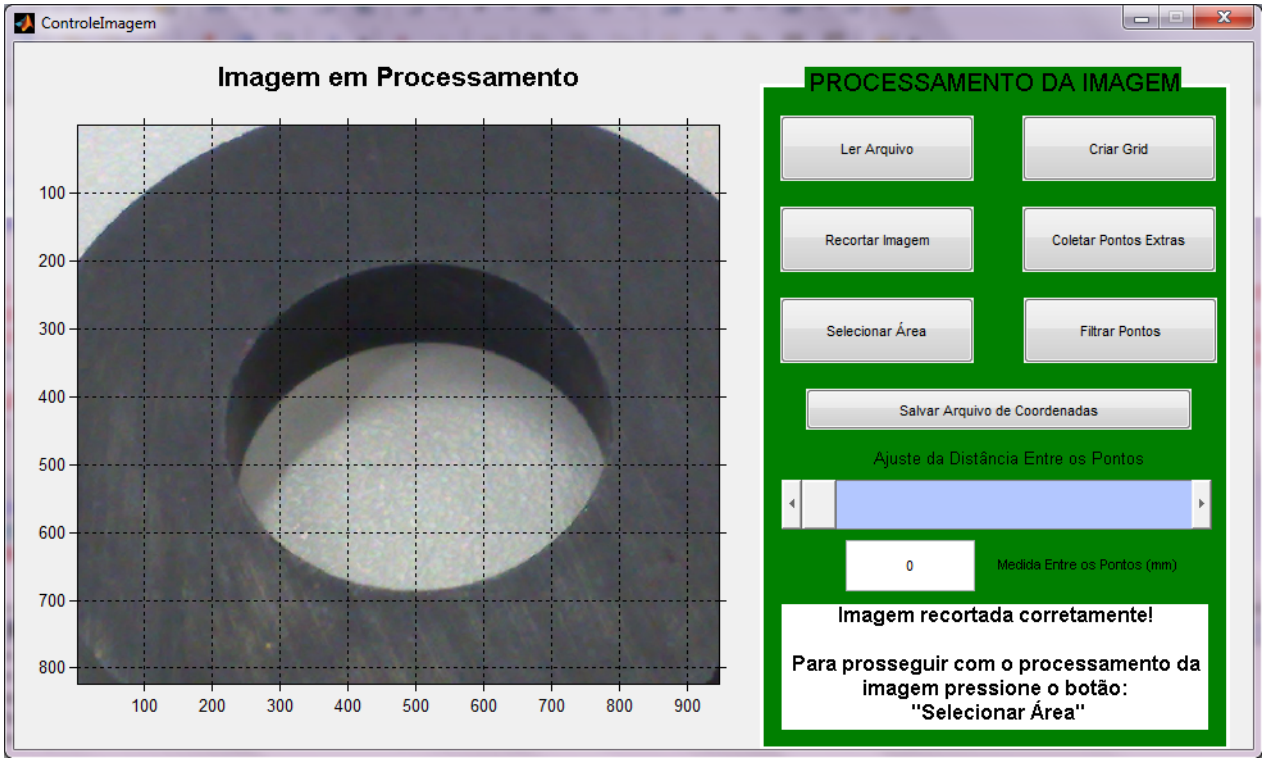


Figura 3.6: Imagem após completar-se o processo de recorte.

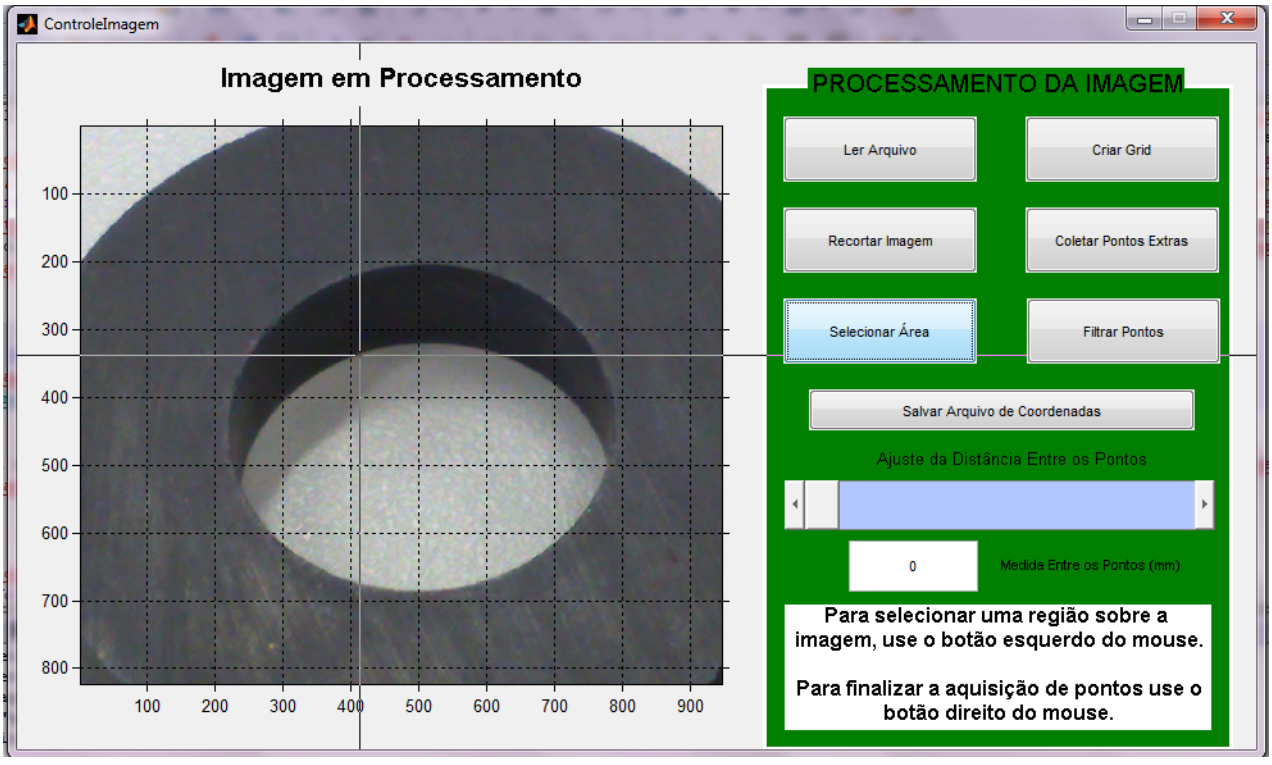


Figura 3.7: Procedimento de captura dos pontos de limite da área a ser tratada.

Uma vez que a imagem se encontrar na dimensões reais do dispositivo, a próxima etapa a ser realizada pelo usuário consiste em definir a área onde será aplicado o procedimento de implante das sementes. Para tal o usuário deverá pressionar o botão " Selecionar Área". Quando o botão for pressionado, surgirá sobre a imagem um longo cursor, conforme pode ser observado na Figura 3.7.

Assim, para que o usuário selecione os pontos desejados, o mesmo deverá pressionar o botão esquerdo do *mouse*, e para encerrar a coleta de pontos deverá pressionar o botão direito do *mouse*. Ao final, esse processo resultará numa região selecionada e demarcada conforme pode ser observado na Figura 3.8.

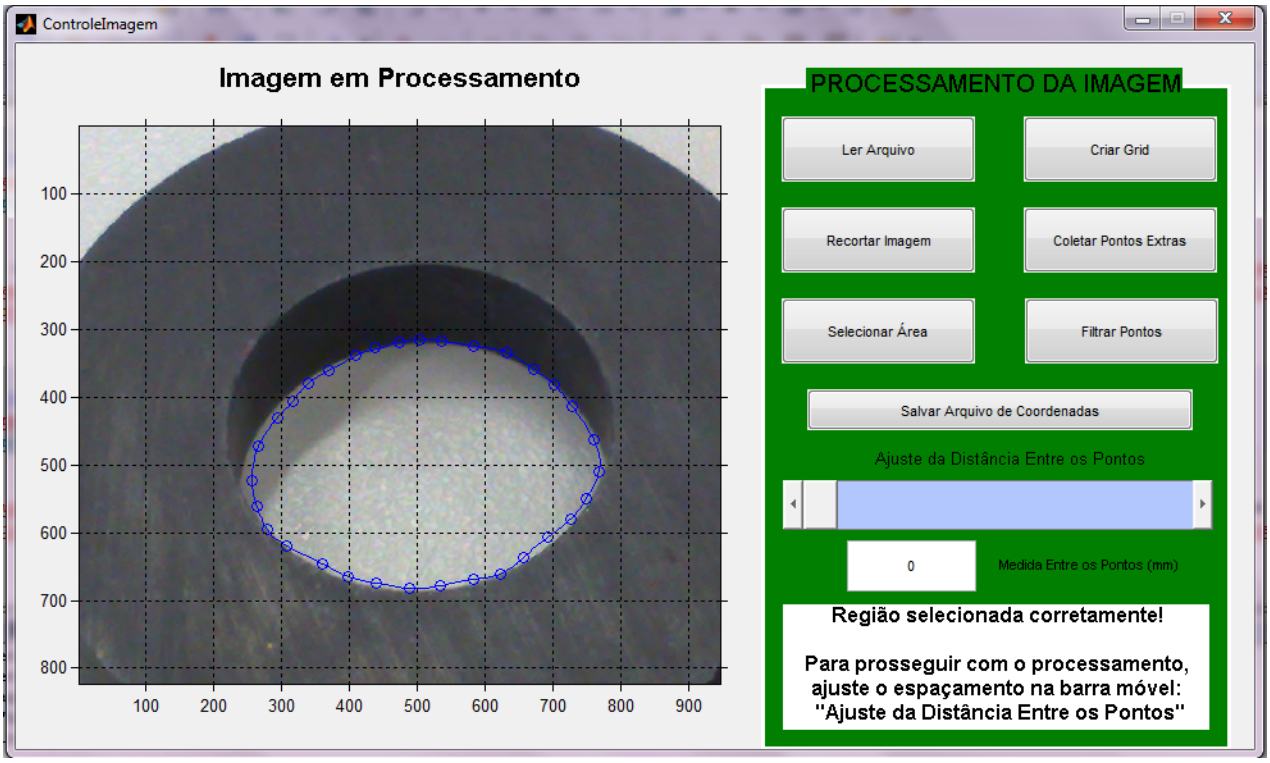


Figura 3.8: Área delimitada e demarcada após seleção dos pontos pelo usuário.

Definida a região em que ocorrerá o implante das sementes, o usuário deverá especificar a distância entre as inserções. Para tal deverá movimentar o *slider* "Ajuste da Distância Entre os Pontos". Com essa operação o usuário poderá especificar a distância entre as inserções, em um intervalo de 0 a 20 mm. Em nosso exemplo, definimos a distância entre os pontos de 8 mm. Conforme pode ser verificado na imagem da Figura 3.9.

Figura 3.9: Definição da distância entre as inserções.

O usuário deverá pressionar o botão “Criar Grid”, após especificar a distância entre as aplicações. Ao pressionar esse botão o programa criará um *grid* de coordenadas na distância especificada pelo usuário dentro e nas proximidades da área selecionada para aplicação do tratamento. Ao final do procedimento, o programa exibirá a imagem com o *grid* criado, conforme pode ser observado na Figura 3.10.

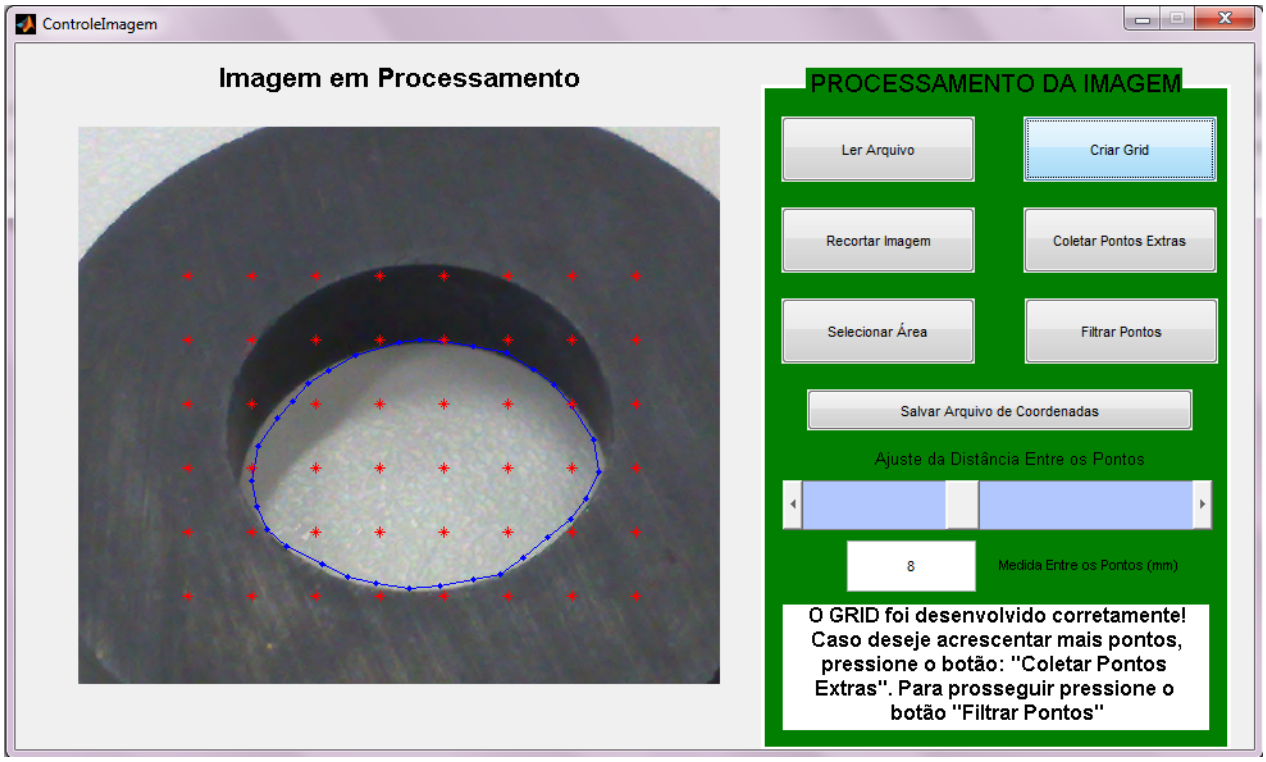


Figura 3.10: Imagem com o grid, conforme especificação do usuário.

Ao analisar a imagem apresentada, o clínico poderá avaliar os pontos em que ocorrerão as inserções dentro da área a ser tratada. Caso julgue que seja necessário a aplicação em algum outro ponto, o mesmo poderá optar em fazer a marcação desses pontos extras. Para isso deverá pressionar o botão “Coletar Pontos Extras”. Caso julgue desnecessário a aquisição de mais alguma coordenada, o clínico deverá pressionar o botão “Filtrar Pontos”. Para desenvolvimento do exemplo ilustrativo, é realizada a aquisição de novos pontos. Assim, após pressionar o botão “Coletar Pontos Extras”, é usado o botão esquerdo do mouse para capturar essas novas coordenadas e, para encerrar a coleta usar o botão direito do mouse.

Ao final do procedimento tem-se todos os pontos alocados, conforme pode ser observado na Figura 3.11, foram adicionados 7 pontos extras, sendo 6 pontos internos e 1 ponto externo à área a ser tratada. Para facilitar a visualização, os pontos foram plotados na cor verde. Após a aquisição desses novos pontos, o usuário deverá pressionar o botão “Filtrar Pontos”. Dessa forma, ao realizar o procedimento todos os pontos externos à área em tratamento serão eliminados, restando sobre a imagem somente os pontos de interesse. Com isso qualquer ponto extra coletado fora da região selecionada será eliminado, não causando nenhum efeito no tratamento planejado. Ao finalizar a operação, teremos o resultado que pode ser observado na Figura 3.12.

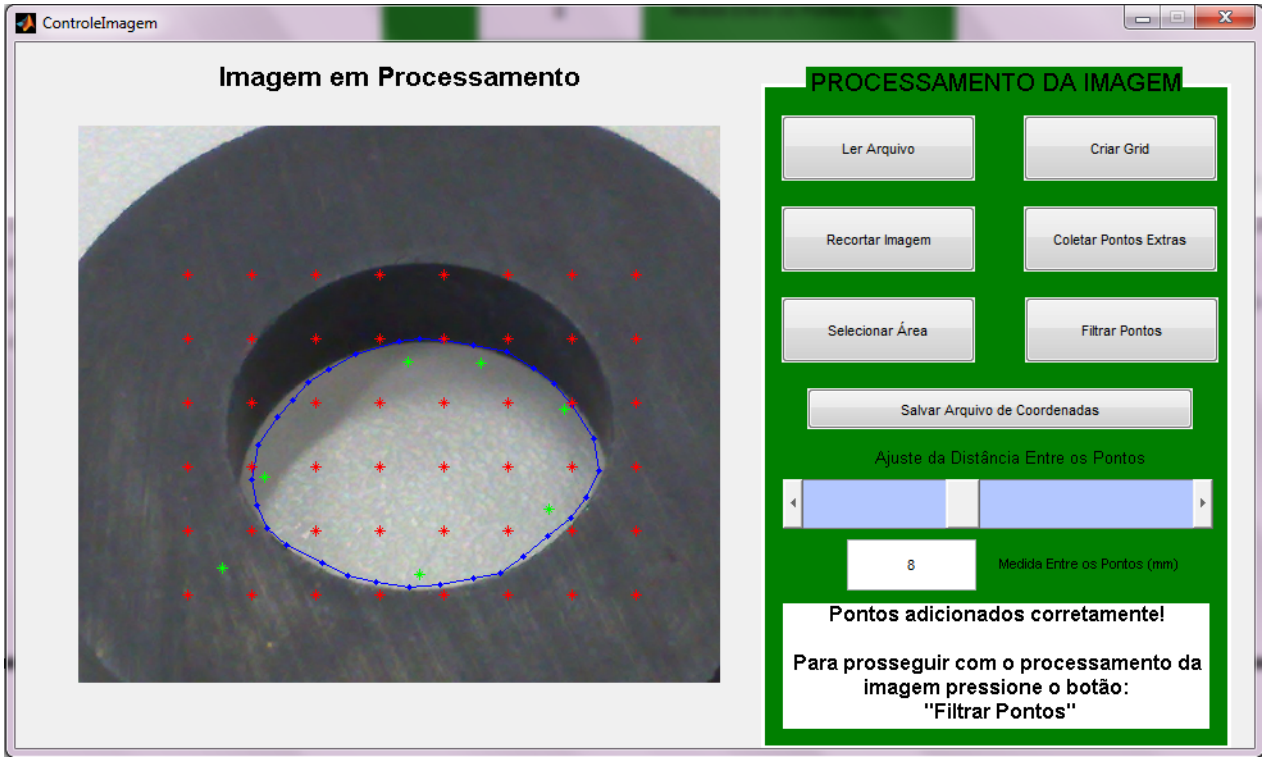


Figura 3.11: Imagem com os pontos extras adicionados.

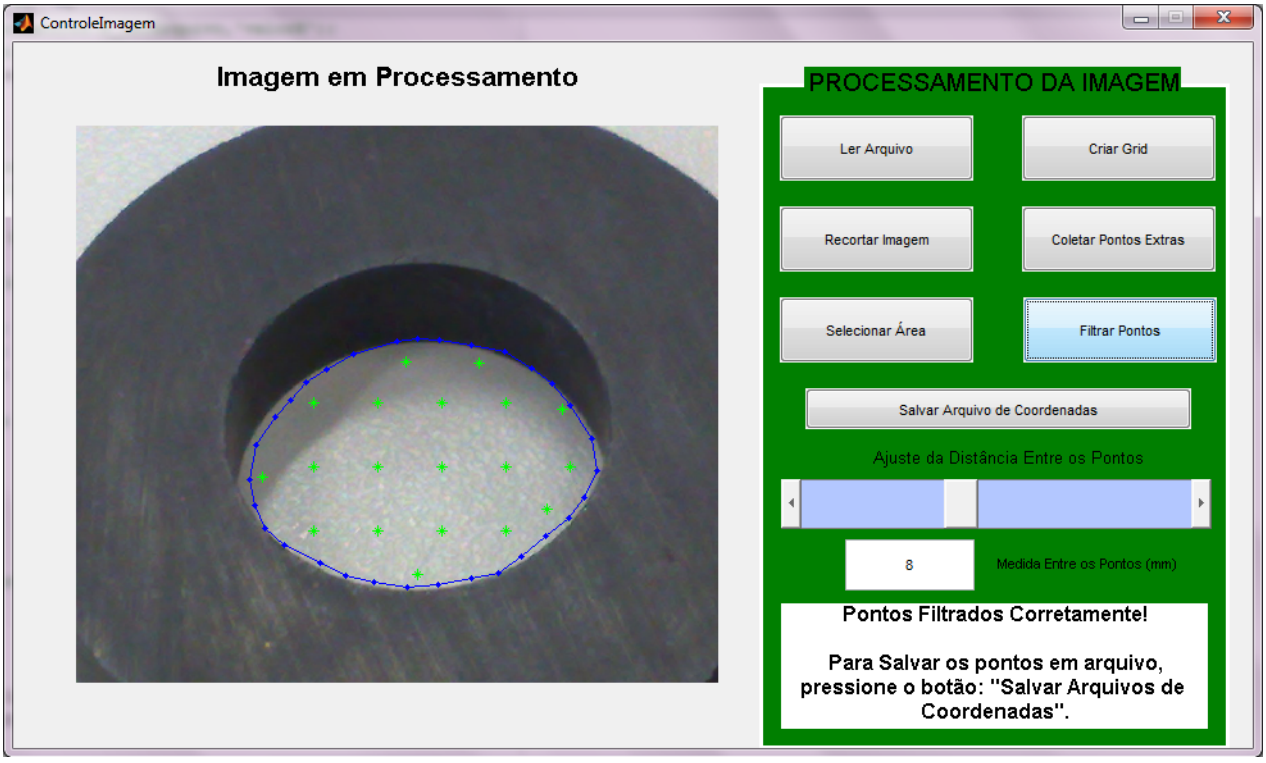


Figura 3.12: Imagem dos pontos onde realmente ocorrerá a inserção das sementes.

Para finalizar o tratamento da imagem, resta ao usuário simplesmente salvar o arquivo contendo as coordenadas dos pontos apresentados na Figura 3.12. Para realizar essa operação o usuário deverá pressionar o botão "Salvar Arquivo de Coordenadas". Quando o botão for pressionado as coordenadas serão convertidas de *pixel* para milímetros e, por fim, será aberta uma caixa de diálogo que permite ao usuário definir em qual diretório o arquivo será salvo.

É importante lembrar que o arquivo deverá ser salvo com o formato ".mat", pois esse é o formato reconhecido pelo MatLab para armazenamento de variáveis em arquivo. O arquivo salvo nesta operação será usado pelo software de controle dos motores que será apresentado no Capítulo 4.

Se em algum momento o usuário não seguir a sequência apresentada no fluxograma da Figura 3.3, o programa apresentará uma mensagem de erro obrigando ao usuário a reiniciar todo o procedimento. A mensagem exibida pelo programa encontra-se apresentada na Figura 3.13.

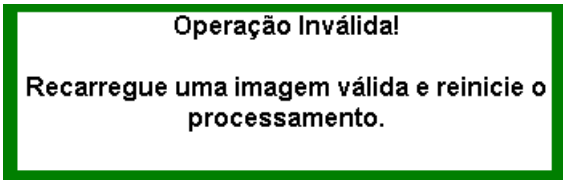


Figura 3.13: Mensagem de erro, caso alguma operação ocorra fora da sequência.

3.4 Uso do Software e a Dosagem da Radioatividade

O *software* desenvolvido para processamento da imagem, atende ao que foi proposto no projeto desse trabalho de conclusão de curso. Porém, para definir a dosagem de radioatividade a ser empregada no tratamento de cada tumor, será necessário o uso de outros softwares para realização do cálculo da dosagem na área do tratamento em função da distância entre as inserções. Uma vez que o número de inserções deverá atender com segurança a dosagem especificada para o tratamento, vale observar que a distância entre os pontos não poderá ser definida aleatoriamente, mas de modo a atender a especificação do procedimento. Essa complementação, conforme ficou definido na época da definição do projeto, ficou para um desenvolvimento futuro, possivelmente utilizando algoritmos baseados em simulações da propagação das radiações nos tecidos baseados no Método de Monte Carlo (Roberto et al. [2005]).

Capítulo 4

Software de Controle dos Motores

4.1 O Código

O código apresentado no Anexo B é responsável por tratar a tabela de coordenadas que será gerada no *software* de tratamento das imagens apresentado no Capítulo 3. Esse código foi escrito utilizando as linguagens de programação do *MatLab*. Quando o código é executado, o mesmo gera a janela de interface com o usuário, a qual se encontra apresentada na Figura 4.1.

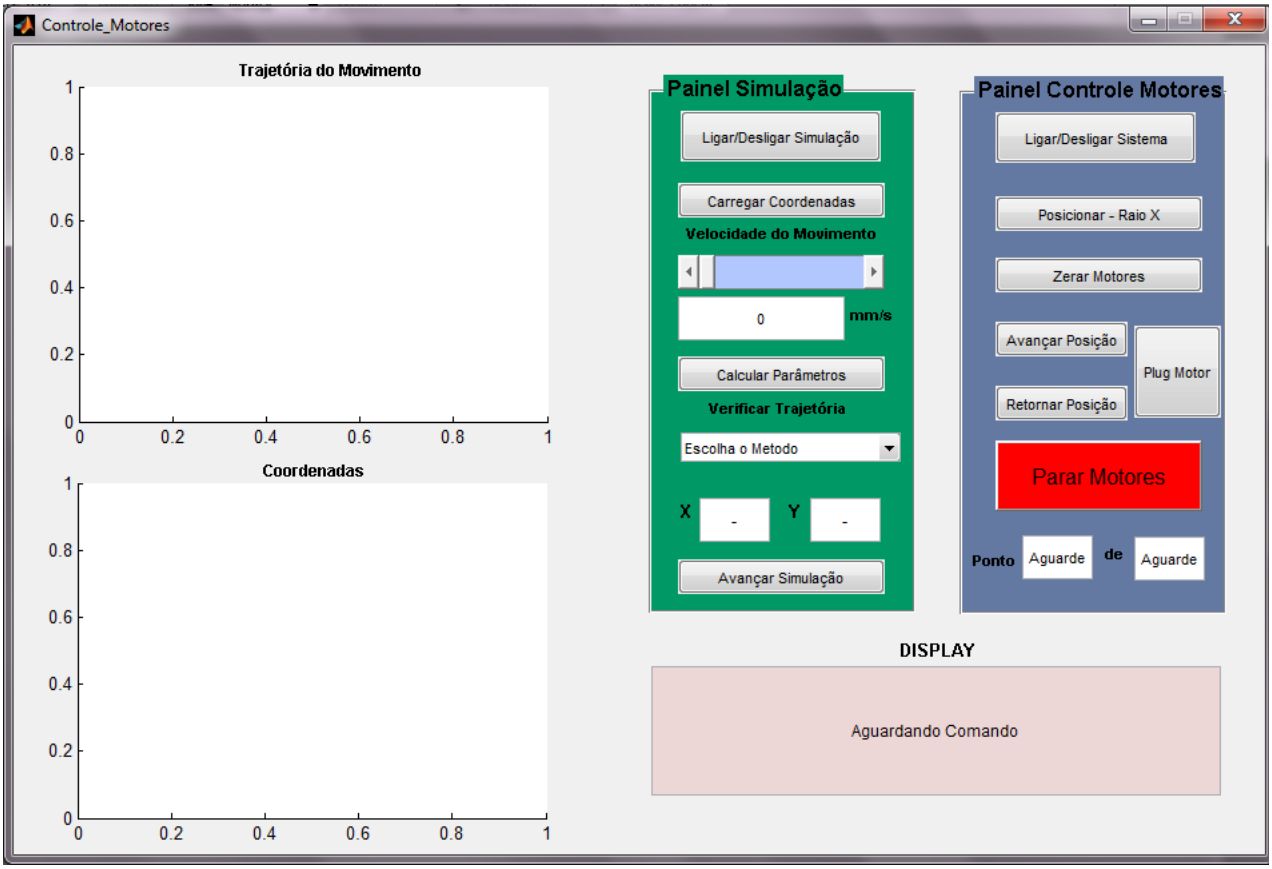


Figura 4.1: Janela de interface com o usuário do *software* de controle dos motores.

Conforme pode ser observado na Figura 4.1, o programa apresenta dois modos de operação: virtual, sendo operado pelo *Painel de Simulação* ou o modo real, sendo este modo controlado pelo *Painel Controle Motores*. No modo virtual é possível que o usuário realize as configurações da operação, como, por exemplo: carregar o arquivo de coordenadas, definir a velocidade de deslocamento da agulha até o próximo ponto de inserção, definir o modo de plotagem da simulação, entre plotagem passo a passo ou plotagem automática. Assim como realizar todos os cálculos necessários para o acionamento dos motores. É importante ressaltar que essa é uma etapa ***obrigatória*** para o funcionamento correto do dispositivo, ou seja, é necessário que o usuário execute a simulação do movimento antes de iniciar o procedimento real. Já no modo real, o programa enviará os trens de pulso e o sentido de rotação dos motores, os quais foram calculados e definidos durante a simulação. Assim, serão obtidos os deslocamentos reais das duas guias lineares na planta.

4.2 Operação do Painel Simulação

Como a técnica de braquiterapia consiste em uma aplicação direta de material radioativo no paciente, é de extrema importância que não ocorram falhas durante o procedimento. Por isso, deve-se realizar a simulação do procedimento e verificar se as coordenadas e trajetórias a serem executadas pela mesa de coordenadas cartesianas XY estão corretas. Por esse motivo o *software* desenvolvido nesse trabalho impõem a execução da simulação antes de se dar início à aplicação direta das sementes radioativas. Para realização da simulação do procedimento, deve-se realizar a sequência de operações apresentadas na Figura 4.2.

O primeiro passo para se dar início à simulação, consiste no acionamento do botão *"Ligar Simulação"*. Quando essa etapa é executada, o programa é ativado para iniciar o processo de simulação. Uma vez pressionado o botão *"Ligar Simulação"*, o usuário pode dar continuidade ao procedimento. Para tal, é necessário que o mesmo carregue o programa com as coordenadas onde ocorrerá o agulhamento. Essa tabela de coordenadas que foi gerada pelo *software* de tratamento de imagens, apresentado no Capítulo 3. Ao pressionar o botão *"Carregar Coordenadas"*, será aberto uma janela de busca, que possibilitará ao usuário carregar no programa o arquivo contendo os dados do procedimento.

Uma vez finalizado o processo de busca do arquivo que contém as coordenadas, o usuário deverá definir através da barra de deslizamento, a velocidade com que a agulha se deslocará entre os pontos de inserção das sementes. Assim, o usuário poderá definir a velocidade num intervalo de 0 a 10 mm/s com passos de $0,25$. É interessante observar que o limite imposto pelo driver de micro passos e pela estrutura mecânica, para a velocidade de movimentação é de 490 mm/s. Porém, como a distância a ser percorrida entre as aplicações é pequena, normalmente cerca de 10 mm, não foi encontrado justificativas para aumentar significativamente a velocidade do movimento. Assim, operando em velocidades menores, são reduzidas as chances de desenvolver elevados graus de vibração na estrutura mecânica, durante as acelerações positiva e negativa do movimento.

Quando a velocidade do movimento se encontrar definida, o usuário deverá pressionar o botão *"Calcular Parâmetros"*. Uma vez que esse botão for pressionado, o programa executará

Figura 4.2: Fluxograma da sequência de operações do *Painel de Simulação*.

um código interno que definirá todos os parâmetros necessários para que os motores sejam acionados. Uma vez finalizada essa operação, o usuário receberá uma mensagem informando que os resultado se encontram disponíveis no *workspace do MatLab*. Com isso, pode-se verificar todos os dados que serão enviados ao motores.

Para concluir a operação de simulação, o usuário deve escolher uma das maneiras como a trajetória a ser executada pela mesa deve ser plotada nos gráficos. São duas as opções: método passo a passo ou utilizando o método automático. Quando a opção escolhida for o método passo a passo, as coordenada e trajetória serão plotadas uma por uma, seguindo a lógica sequencial da tabela de coordenadas. Para tal, é necessário que o usuário pressione a todo momento o botão *"Avançar Simulação"*. Vale observar que esse método é a simulação do procedimento que realmente será executado. A outra opção: método automático, irá plotar todas as coordenadas e trajetórias de uma única vez.

4.3 Resultados Obtidos com a Simulação

Para verificar o funcionamento do programa descrito na Seção 4.2, criou-se uma tabela de coordenadas.

Nº Coordenada	Eixo X	Eixo Y
1	0	0
2	2	5
3	4	8
4	5	10
5	3	16
6	2	5
7	0	0

Tabela 4.1: Tabela de coordenadas para simulação do *software*.

E executando as etapas apresentadas no fluxograma da Figura 4.2, carregando o arquivo que continha os valores apresentados na Tabela 4.1, configurando a velocidade de movimentação para $10mm/s$ e definindo o modo automático para plotagem da trajetória. Ao final da simulação, obteve-se o resultado apresentado pela Figura 4.3 e pela Tabela 4.2.

Coord. X	Nº Pulsos X	F. X (Hz)	Coord. Y	Nº Pulsos Y	F. Y (Hz)	t (s)
0	0	0	0	0	0	0
2	204	378,8	5	509	945,2	0,5385
4	204	565,8	8	306	848,7	0,3606
5	102	456,2	10	204	912,3	0,2236
3	204	322,6	16	611	966,1	0,6325
2	102	92,3	5	1120	1014,0	1,1045
0	204	378,8	0	509	945,2	0,5385

Tabela 4.2: Resultado apresentado no *workspace do MatLab*.

Para exemplificar como obteve-se os valores apresentados na Tabela 4.2, pode-se analisar o deslocamento da coordenada $(0, 0)$ para a coordenada $(2, 5)$. A primeira operação realizada pelo *software*, consiste em calcular o deslocamento no eixo X, conforme Equação (4.1).

$$d_X = X_2 - X_1 = 2 - 0 = 2 \tag{4.1}$$

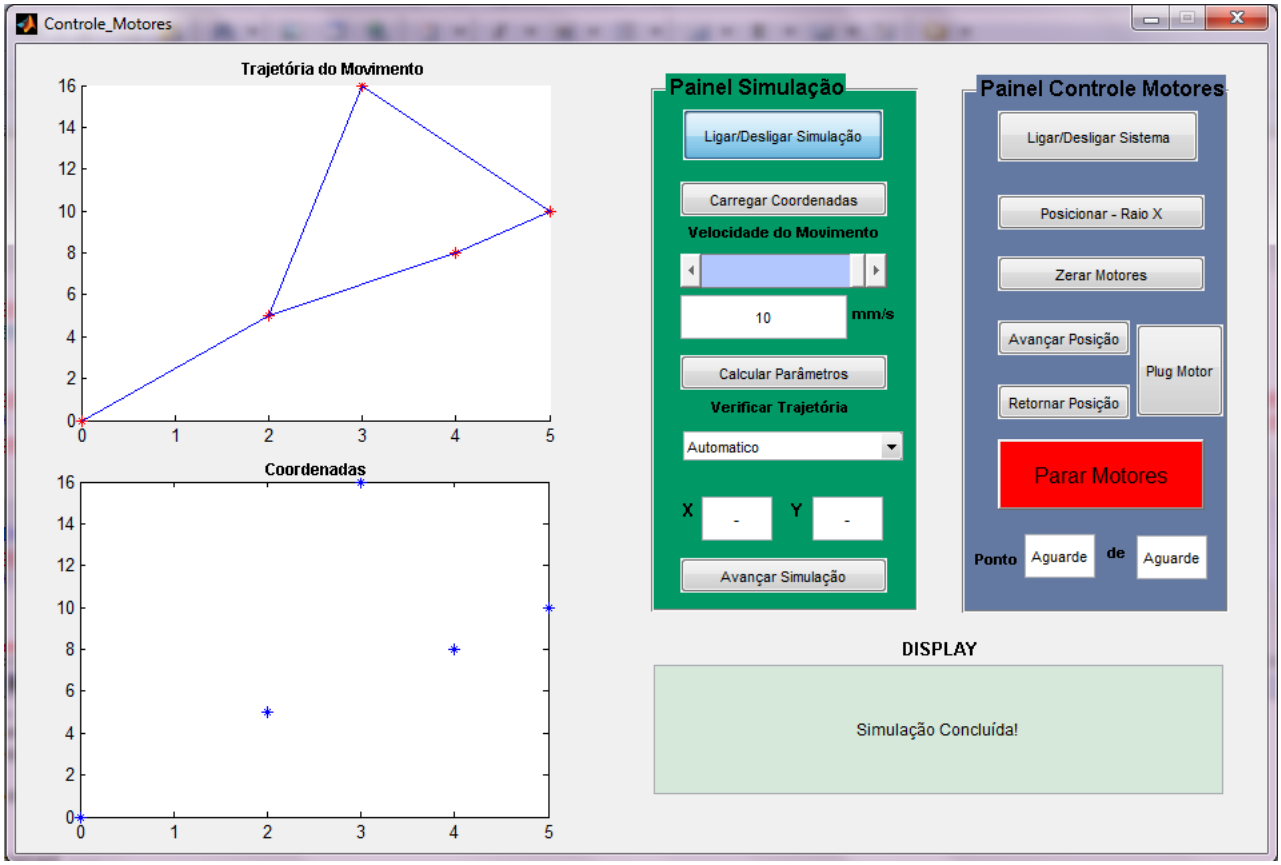


Figura 4.3: Resultado apresentado pelo programa ao fim da simulação.

no qual d_X é o deslocamento na direção do eixo X. Sabendo que o diâmetro primitivo da polia acoplada ao motor é de 25 mm, e fazendo uso do deslocamento por pulso apresentado na Tabela 2.2, para a configuração de 1/40, pode-se então determinar o número de pulsos necessários para realizar o movimento, conforme Equação (4.2)

$$Pulsos_X = \frac{d_X}{0.0098} \tag{4.2}$$

o que nos resulta em 204 pulsos. Para determinar o tempo gasto para realização do movimento, basta aplicarmos a Equação (4.3)

$$t = \frac{\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}}{V}, \tag{4.3}$$

estando a velocidade (V) configurada para 10 mm/s, obtem-se o tempo de movimentação de $0,5385$ s. Para determinar a frequência com que com que os pulsos devem ser gerados, deve-se aplicar a Equação (4.4)

$$F_X = \frac{Pulsos_X}{t} \tag{4.4}$$

em que F_X é a frequência do trem de pulsos do motor ligado ao eixo X. Para determinar os valores para o eixo de coordenadas Y, basta repetir o procedimento executado anteriormente.

Ao simular as coordenadas geradas pelo exemplo desenvolvido na Seção 3.3 do Capítulo 3, obtêm-se o resultado apresentado pela Figura 4.4.

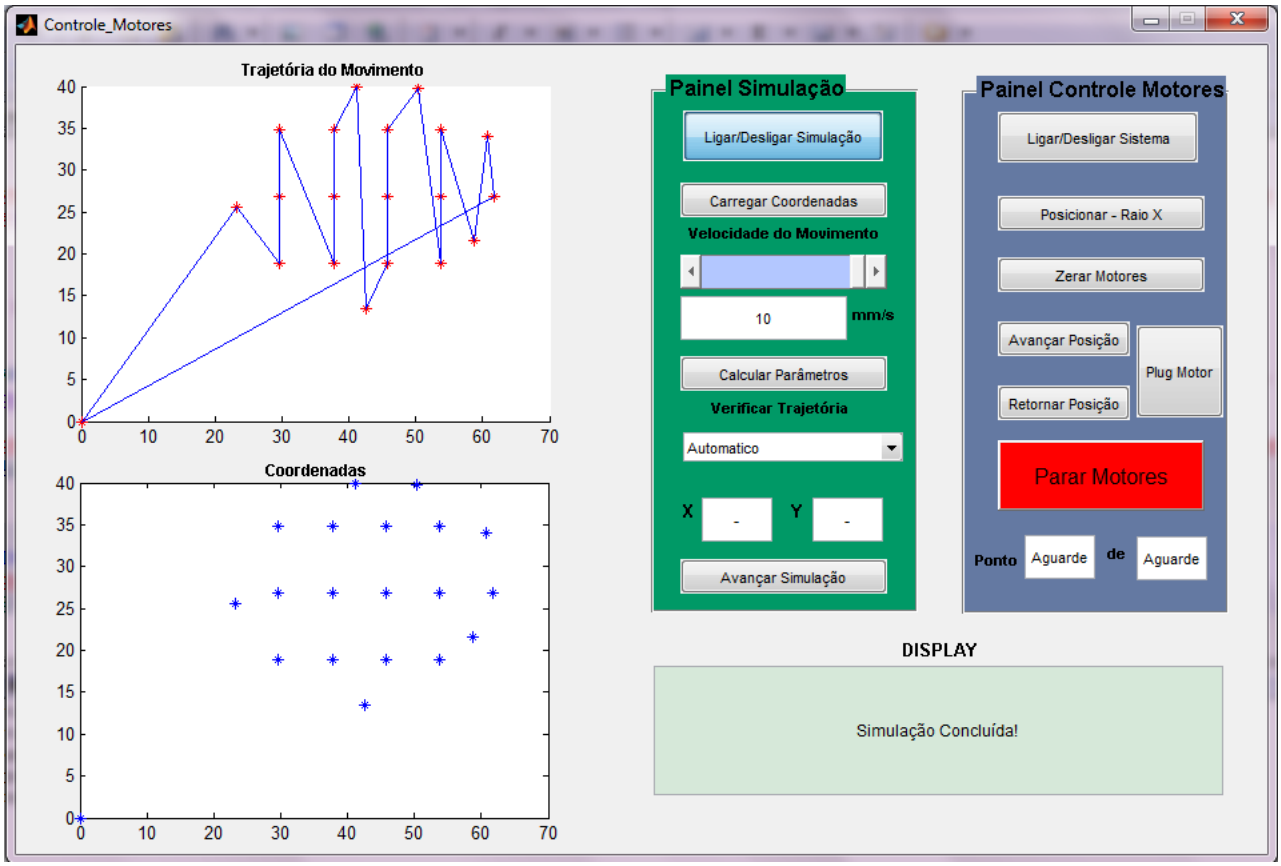


Figura 4.4: Resultado da simulação dos pontos de agulhamento.

A agulha parte da coordenada $(0, 0)$, percorrendo todas as coordenadas planejadas. Por fim, quando se encerra as coordenadas planejadas, o sistema retorna à posição inicial, ou seja, à origem do sistema mecânico.

4.4 Acionamento dos Motores

O *Painel Controle de Motores* tem a finalidade de realizar as operações de conexão e acionamento dos motores. A Figura 4.5 apresenta a sequência para acionamento correto dos motores.

A primeira etapa para acionamento dos motores consiste em ativar o próprio painel. Para tal é necessário que o usuário pressione o botão "Liga/Desligar Sistema". Uma vez que o painel se encontrar ativado, o usuário poderá dar continuidade ao processo. Assim, ele deverá realizar o procedimento para posicionar o sistema mecânico na origem, ou seja, posicionar o sistema

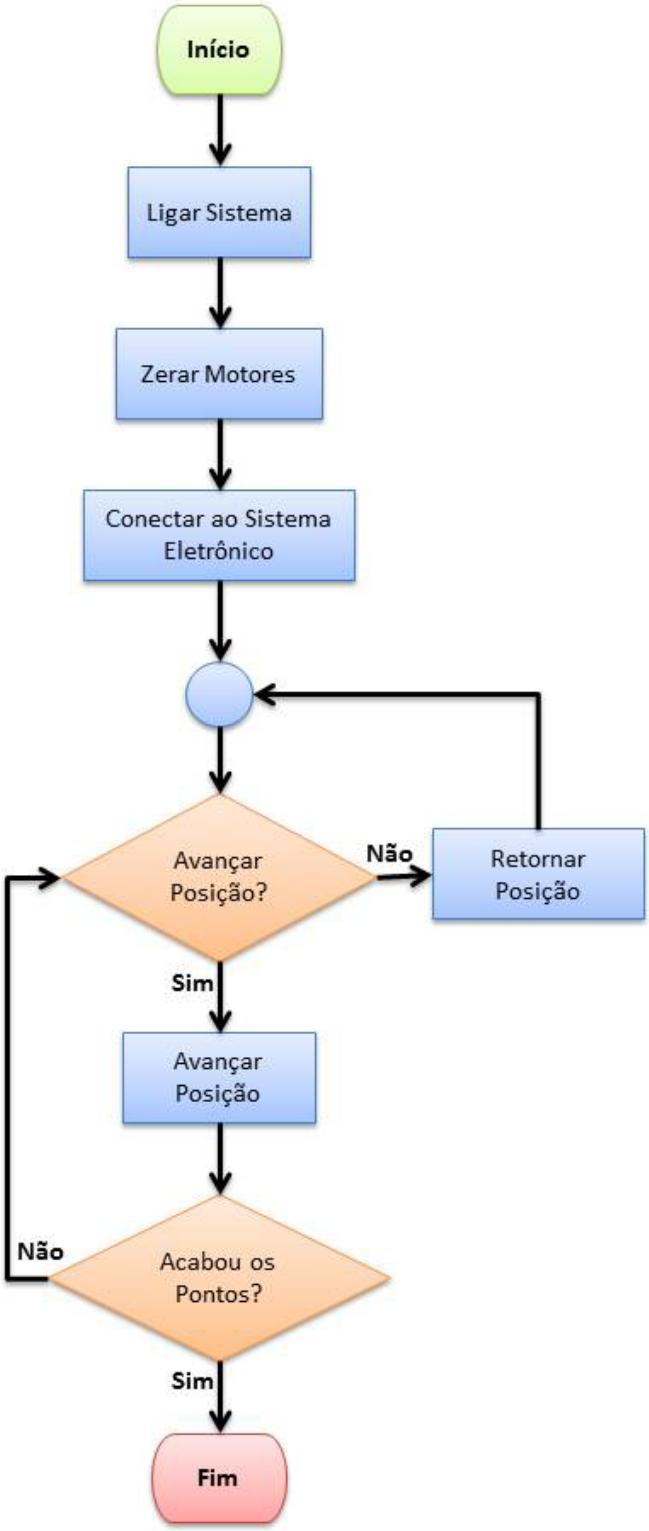


Figura 4.5: Sequência de operações para acionamento dos motores.

mecânico na coordenada $(0, 0)$. Para iniciar essa operação o usuário deverá pressionar o botão "Zerar Motores". Quando o botão for precionado ele abrirá o diagrama de blocos do *Simulink* e executará o arquivo: *Controle Zero*. O diagrama de blocos: *Controle Zero* segue apresentado na Figura 4.6. Esse será responsável em analizar o sinal emitido pelos sensores indutivos e, a partir daí, posicionar o sistema mecânico na origem.

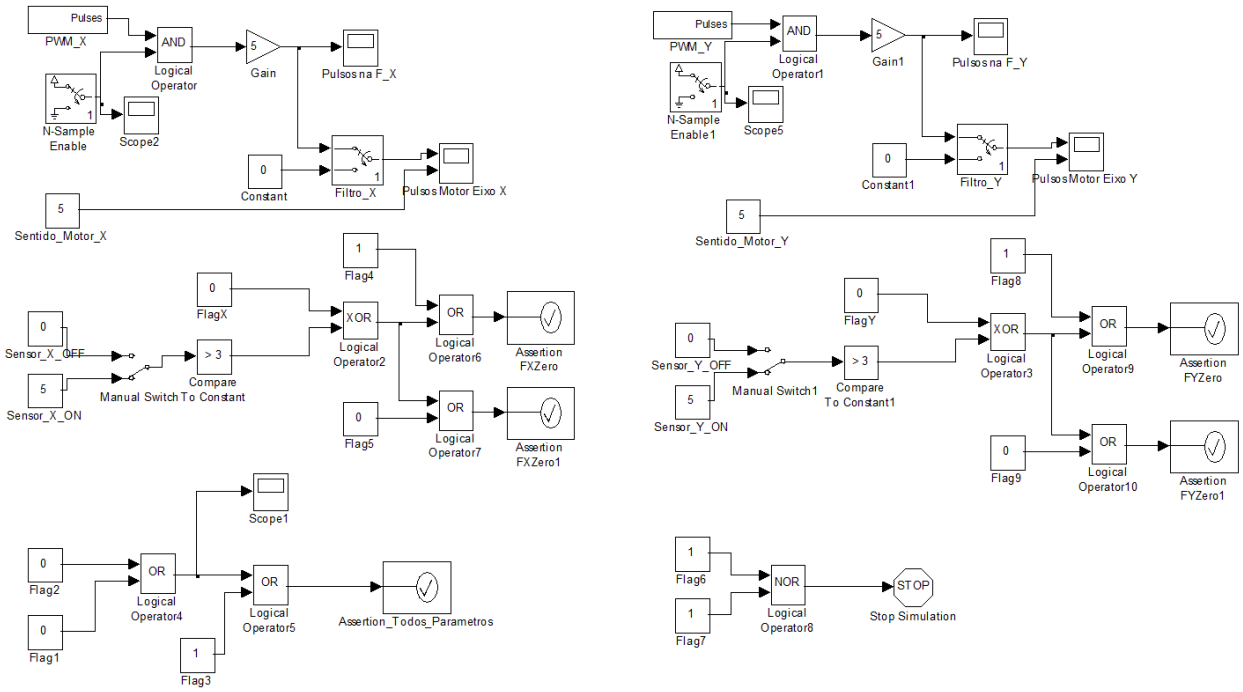


Figura 4.6: Diagrama de blocos para posicionar o sistema na origem.

A sequência de operações e análises realizadas pelo diagrama apresentado na Figura 4.6, encontra-se disponível no fluxograma da Figura 4.7, sendo auxiliado pela Tabela 4.3

Sinal do sensor 0	O sistema mecânico não está fechando o contato do sensor.
Sinal do sensor 1	O sistema mecânico está fechando o contato do sensor.
Sentido do Motor 0	Os motores devem levar o sistema móvel na direção da origem do sistema mecânico, ou seja, aproximando-se da origem.
Sentido do Motor 1	Os motores devem levar o sistema móvel na direção oposta à origem do sistema mecânico, ou seja, afastando-se da origem.

Tabela 4.3: Tabela significados dos termos usados no fluxograma da Figura 4.7.

Após o usuário realizar o referenciamento dos eixos, o próximo passo consiste em conectar os motores para receber os sinais de movimentação definidos e calculados durante a simulação. Para realizar essa conexão o usuário deverá pressionar o botão "Plug Motores". Quando o botão for pressionado, o *Simulink* será novamente acionado, porém, nesse momento, ele abrirá o programa *Controle Motor PWM*. Esse diagrama de blocos apresentado na Figura 4.8 é responsável por aplicar aos motores os trens de pulsos calculados e definidos durante a simulação na Seção 4.2.

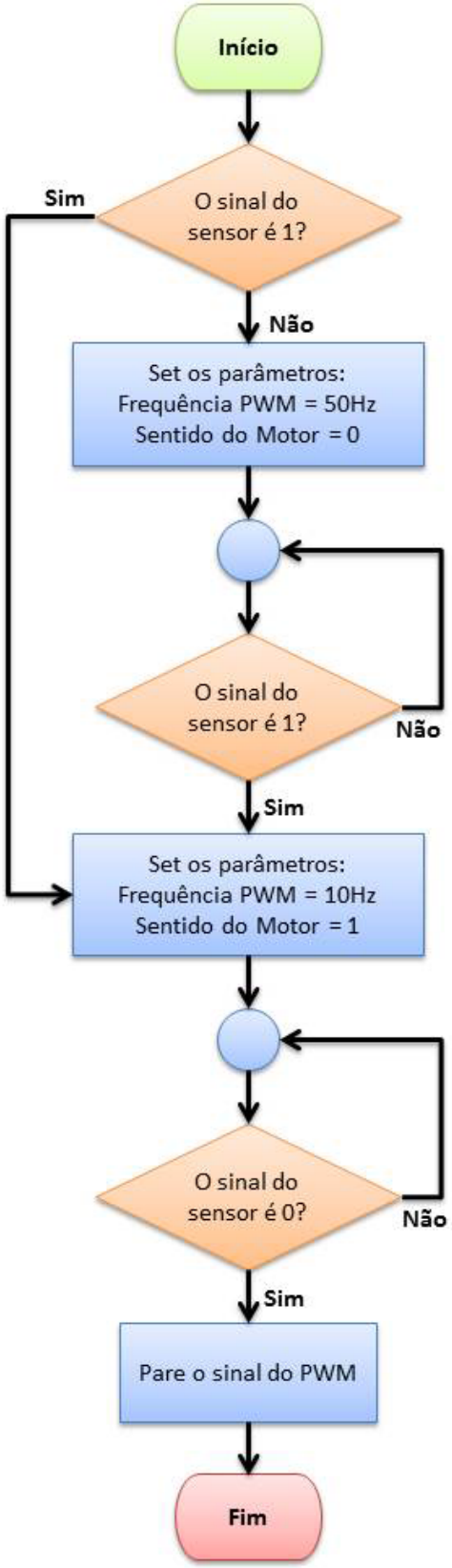


Figura 4.7: Fluxograma de análise para posicionamento do sistema na origem.

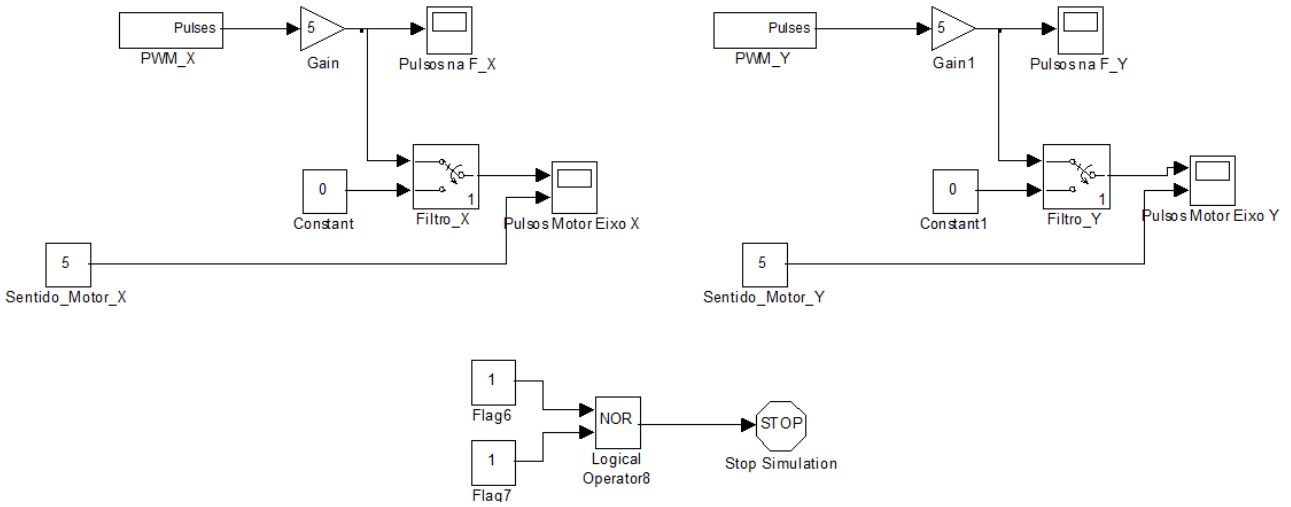


Figura 4.8: Diagrama de blocos usados para acionar os motores.

Conforme pode ser observado no diagrama de blocos apresentado na Figura 4.8, esse programa é menos complexo que aquele usado para referenciar os eixos. A cada vez que o usário pressionar o botão "Avançar Posição", os parâmetros: frequência do PWM, sentido motor e tempo de movimentação serão setados e o dispositivo mecânico iniciará seu deslocamento até o ponto desejado. O deslocamento será encerrado quando for alcançado o tempo necessário para realização da movimentação.

O botão "Retornar Posição" estará disponível para o usuário, caso ocorra algum erro durante a aplicação e seja necessário retornar ao ponto anterior. Porém, espera-se que esse botão não seja acionado durante o procedimento. Caso seja necessário parar emergencialmente os motores durante o procedimento, basta que o usuário use o botão "Parar Motores". Esse botão seta o comando no diagrama *Controle Motores PWM* e interrompe os trens de pulsos que são enviados para os motores, apresentando um funcionamento idêntico à de um botão de emergência.

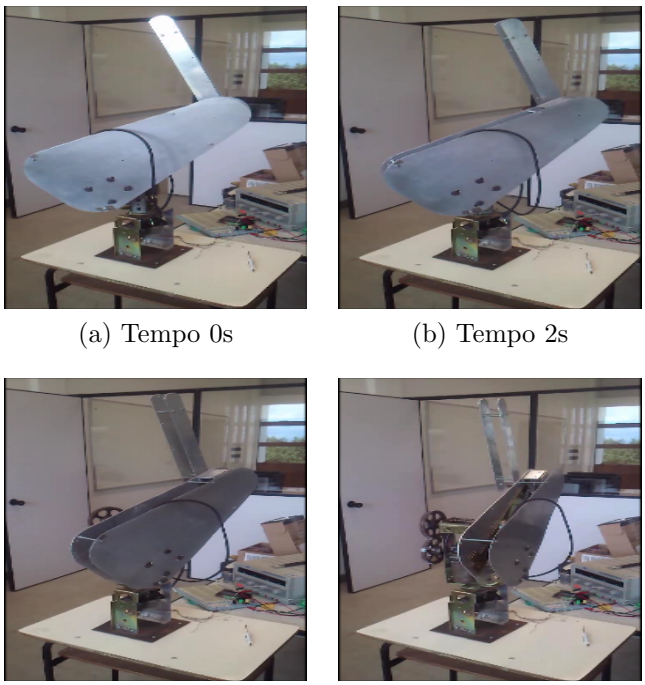
Ainda resta observar o uso e função do botão Posicionar Raio X. Esse botão deverá ser acionado no início do procedimento para realização do Raio X, pois sua função é garantir que os motores posicionem todo o conjunto mecânico no centro referência. Assim, obter a posição correta para todo o conjunto apresentado na Figura 2.5. E, por fim, a garantia do referenciamento correto do sistema mecânico na imagem.

4.5 Resultados Experimentais

Apesar de não se encontrar disponível a receita financeira prevista inicialmente para a realização da montagem do arranjo mecânico, foi possível realizar a aquisição de um driver e de um motor. Assim, mesmo sem as devidas condições técnicas para verificação da precisão dos movimentos, foi possível verificar e validar a funcionalidade do *software* de acionamento de um dos motores, o qual respondeu aos comandos enviados conforme esperado. Com isso, observou-se a movimentação do motor tanto no sentido horário, quanto no sentido anti-horário. E que o mesmo atuava tanto em baixas quanto em altas velocidades, sem produzir ruídos providos pelo

movimento de rotação, que pudessem prejudicar o posicionamento do dispositivo.

O resultado experimental foi obtido utilizando um protótipo de braço mecânico, que se encontra disponível no Laboratório de Robótica. Esse dispositivo, contém em sua base um motor de passo com a mesma divisão angular e precisão do motor proposto nesse projeto. Assim como, características elétricas semelhantes ao motor especificado no projeto mecânico. A Figura 4.9 apresenta uma sequência de imagens, em que é possível observar a movimentação rotacional do braço robótico.



(a) Tempo 0s (b) Tempo 2s

(c) Tempo 4s (d) Tempo 6s

Figura 4.9: Movimentação do braço articulado com o uso do *software*.

Capítulo 5

Considerações Finais

Pelo trabalho realizado até o presente momento, pode-se verificar que o projeto encontra-se bem encaminhado, apesar de não se encontrar finalizado. O projeto mecânico virtual encontra-se finalizado. A próxima etapa referente ao braço mecânico a ser realizada consiste em usinar e montar toda a estrutura mecânica. Quando a construção mecânica for concluída, poderão ser realizados testes de validação mais rigorosos. E, por fim, utilizar o dispositivo em uma aplicação real. Os dois softwares desenvolvidos, tanto para acionamento dos motores quanto para processamento de imagem, corresponderam às expectativas do projeto proposto, de modo a não deixar dúvidas sobre sua funcionalidade para o projeto.

Perspectivas

Em um próximo trabalho, pode-se dar continuidade no desenvolvimento desse projeto. O que incluiria providenciar a conclusão da construção mecânica e realização de testes de validação mais rigorosos. Seria interessante também, implementar no código de controle da aceleração dos motores, de forma a configurar rampas de velocidade nas fases de aceleração e desaceleração de cada movimento. Com isso, serão obtidos movimentos de partida e parada mais suaves. Outra importante e interessante implementação, seria completar a automação do procedimento, o incluíria: a automação da seringa e a substituição do braço articulado por um posicionador automatizado. . Havendo essa complementação, seria possível realizar o procedimento sem que o corpo clínico se encontrasse em contato direto com as sementes radioativas, configurando um procedimento completamente remoto.

Apêndice A

Código do Software de Processamento de Imagem

A seguir será apresentado algumas das funções empregadas no código de processamento de imagem.

```
function varargout = ControleImagem(varargin)
% CONTROLEIMAGEM M-file for ControleImagem.fig
%      CONTROLEIMAGEM, by itself, creates a new CONTROLEIMAGEM or raises the existing
%      singleton*.
%
%      H = CONTROLEIMAGEM returns the handle to a new CONTROLEIMAGEM or the handle to
%      the existing singleton*.
%
%      CONTROLEIMAGEM('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CONTROLEIMAGEM.M with the given input arguments.
%
%      CONTROLEIMAGEM('Property','Value',...) creates a new CONTROLEIMAGEM or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before ControleImagem_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to ControleImagem_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ControleImagem

% Last Modified by GUIDE v2.5 04-Mar-2013 10:47:02

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @ControleImagem_OpeningFcn, ...
    'gui_OutputFcn',  @ControleImagem_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ControleImagem is made visible.
function ControleImagem_OpeningFcn(hObject, eventdata, handles,
   varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ControleImagem (see VARARGIN)

% Choose default command line output for ControleImagem
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);



% --- Outputs from this function are returned to the command line.
function varargout = ControleImagem_OutputFcn(hObject, eventdata,
```

```
    handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in B_LerArquivo.
function B_LerArquivo_Callback(hObject, eventdata, handles)
% hObject    handle to B_LerArquivo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

cla(handles.axes1,'reset')% reseta a area do gr\'{a}fico
set(handles.AjusteDistancia,'value',0) % reseta o espa\c{c}amento
set(handles.ShowDistancia,'String',0)% reseta o display slider
setappdata(handles.PontoExtra,'controle',0) %seta o flag de controle
    do ponto extra

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

setappdata(handles.B_LerArquivo,'flag',[1 0 0 0 0 0 0 0])
setappdata(handles.B_LerArquivo,'flag2',0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


[imag] = uigetfile(); %Ler arquivo
I = imread(imag); %L\^{e} o arquivo de imagem
imshow(I); %apresenta/exibi a imagem
axis on, grid on;

%retira as informa\c{c}\~{o}es da imagem
a = imfinfo(imag,'jpg');

%retira e aramzena a resolu\c{c}\~{a}o de escala de cada eixo
tamX=a.XResolution;
tamY =a.YResolution;

%retira largura m\'{a}xima em pixels de cada eixo
valorX = a.Width;
valorY = a.Height;

%disponibiliza os valores para outras fun\c{c}\~{o}es
setappdata(handles.B_LerArquivo,'tamX',tamX)
```

```
setappdata(handles.B_LerArquivo,'tamY',tamY)
setappdata(handles.B_LerArquivo,'valorX',valorX)
setappdata(handles.B_LerArquivo,'valorY',valorY)
setappdata(handles.B_LerArquivo,'I',I)

set(handles.Display,'String',sprintf('Arquivo carregado corretamente
   !\n \n Para prosseguir com o processamento da imagem pressione o
   bot\~{a}o:\n "Recortar Imagem"'))


% --- Executes on button press in RecortarImagem.
function RecortarImagem_Callback(hObject, eventdata, handles)
% hObject    handle to RecortarImagem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
flag = getappdata(handles.B_LerArquivo,'flag');

if flag == [1 0 0 0 0 0 0 0];
    set(handles.Display,'String',sprintf('Selecione um ponto sobre a
        regi\~{a}o desejada.\n \n Use o bot\~{a}o esquerdo do mouse e
        pressione ENTER.'))
    tamX = getappdata(handles.B_LerArquivo,'tamX');
    tamY = getappdata(handles.B_LerArquivo,'tamY');
    I = getappdata(handles.B_LerArquivo,'I');

    %coverte pixel em milimitros.
    pixelX = 25.4/tamX;

    %faz aquisi\c{c}\~{a}o de uma coordenada da imagem
    [x, y] = getpts;
    aux = length(x);
    if aux > 1
        x = x(1,1);
        y = y(1,1);
    end

    %procedimento que converte dist\^{a}ncia em pixel - dimens\~{o}
       es do retangulo

    distx = 80;
    disty = 69.6;

    largura = distx*tamX/25.4;
    altura = disty*tamY/25.4;
    x = x - 0.4*x;
    y = y - y/2;
    C = imcrop(I,[x y largura altura]);
```

```
    T = maketform('affine',[1 0 0; 0 1 0; 0 0 1]);
    tformfwd([10 20],T);
    [D]  = imtransform(C,T);
    imshow(D); %apresenta/exibi a imagem
    axis on, grid on;
    setappdata(handles.RecortarImagem,'altura',altura)
    setappdata(handles.RecortarImagem,'pixelX',pixelX)
    setappdata(handles.RecortarImagem,'imagemD',D)
    setappdata(handles.B_LerArquivo,'flag',[1 1 0 0 0 0 0 0])
    set(handles.Display,'String',sprintf('Imagem recortada
       corretamente!\n \n Para prosseguir com o processamento da
       imagem pressione o bot\~{a}o:\n "Selecionar \'{A}rea"'))
else
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end


% --- Executes on button press in SelecionarArea.
function SelecionarArea_Callback(hObject, eventdata, handles)
% hObject    handle to SelecionarArea (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
flag = getappdata(handles.B_LerArquivo,'flag');

if flag == [1 1 0 0 0 0 0 0];
    set(handles.Display,'String',sprintf('Para selecionar uma regi
       \~{a}o sobre a imagem, use o bot\~{a}o esquerdo do mouse.\n \n
        Para finalizar a aquisi\c{c}\~{a}o de pontos use o bot\~{a}o
       direito do mouse.'));

    %selecionando a \'{a}rea de aplica\c{c}\~{a}o das sementes
       spline.
    hold on
    % Initially, the list of points is empty.
    xy = [];
    n = 0;
    but = 1;
    while but == 1
        [xi,yi,but] = ginput(1);
        plot(xi,yi,'bo')
        n = n+1;
        xy(:,n) = [xi;yi];
    end
```

```
    i = length(xy);
    xy(1,i)= xy(1,1);
    xy(2,i)= xy(2,1);

    % Interpolate with a spline curve and finer spacing.
    t = 1:n;
    ts = 1: 0.1: n;
    xys = spline(t,xy,ts);

    % Plot the interpolated curve.
    plot(xys(1,:),xys(2,:),'b-');
    xy=xy';
    setappdata(handles.SelecionarArea,'xy',xy)
    setappdata(handles.B_LerArquivo,'flag',[1 1 1 0 0 0 0 0])
    set(handles.Display,'String',sprintf('Regi\~{a}o selecionada
       corretamente!\n \n Para prosseguir com o processamento, ajuste
        o espa\c{c}amento na barra m\'{o}vel:\n "Ajuste da Dist\^{a}
       ncia Entre os Pontos"'))
else
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end

% --- Executes on slider movement.
function AjusteDistancia_Callback(hObject, eventdata, handles)
% hObject    handle to AjusteDistancia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
   range of slider
flag = getappdata(handles.B_LerArquivo,'flag');

if flag == [1 1 1 0 0 0 0 0];
    slider_value = get(hObject,'Value');
    teste01 = floor(slider_value);
    teste02 = mod(slider_value,1);
    if teste02>=0 && teste02<0.5
        slider_value = teste01;
    else
        slider_value = teste01+1;
    end
    set(handles.AjusteDistancia,'value',slider_value)
```

```
    setappdata(handles.AjusteDistancia,'distancia',slider_value)
    set(handles.ShowDistancia,'String',slider_value)

    setappdata(handles.B_LerArquivo,'flag2',1)
    set(handles.Display,'String',sprintf('Mova a barra at\'{e} o
       valor desejado!\n \n Para prosseguir com o processamento da
       imagem pressione o bot\~{a}o:\n "Criar Grid"'))
else
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end



% --- Executes during object creation, after setting all properties.
function AjusteDistancia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to AjusteDistancia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function ShowDistancia_Callback(hObject, eventdata, handles)
% hObject    handle to ShowDistancia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ShowDistancia as
   text
%        str2double(get(hObject,'String')) returns contents of
   ShowDistancia as a double


% --- Executes during object creation, after setting all properties.
function ShowDistancia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ShowDistancia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in CriarGrid.
function CriarGrid_Callback(hObject, eventdata, handles)
% hObject    handle to CriarGrid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
flag = getappdata(handles.B_LerArquivo,'flag');
flag2 = getappdata(handles.B_LerArquivo,'flag2');

if flag == [1 1 1 0 0 0 0 0]
    if flag2 ==1
        distancia = getappdata(handles.AjusteDistancia,'distancia');
        pixelX = getappdata(handles.RecortarImagem,'pixelX');
        xy = getappdata(handles.SelecionarArea,'xy');
        D = getappdata(handles.RecortarImagem,'imagemD');

        k = distancia/pixelX;

        [m,m2,x,y,gridx]=getfuncoes(xy,k);

        kx = length(x);

        cla(handles.axes1,'reset')% reseta a area do gr\'{a}fico

        imshow(D);
        hold on
        plot(xy(:,1),xy(:,2),'.-b')

        for i=1:kx
            plot(gridx(i,1:end),y,'*r');
        end

        setappdata(handles.CriarGrid,'m',m)
        setappdata(handles.CriarGrid,'m2',m2)
        setappdata(handles.CriarGrid,'x',x)
        setappdata(handles.CriarGrid,'y',y)
        setappdata(handles.B_LerArquivo,'flag',[1 1 1 1 1 0 0 0])
```

```
        set(handles.Display,'String',sprintf('O GRID foi
           desenvolvido corretamente!\n Caso deseje acrescentar mais
           pontos, pressione o bot\~{a}o: "Coletar Pontos Extras".
           Para prosseguir pressione o bot\~{a}o "Filtrar Pontos"'))
    else
        set(handles.Display,'String',sprintf('Ajuste primeiro a dist
           \^{a}ncia entre os pontos.'))
    end
else
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end


% --- Executes on button press in PontoExtra.
function PontoExtra_Callback(hObject, eventdata, handles)
% hObject    handle to PontoExtra (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
flag = getappdata(handles.B_LerArquivo,'flag');
if flag == [1 1 1 1 1 0 0 0]
    setappdata(handles.PontoExtra,'controle',1)
    set(handles.Display,'String',sprintf('Para selecionar pontos
       sobre a imagem, use o bot\~{a}o esquerdo do mouse.\n \n Para
       finalizar a aquisi\c{c}\~{a}o de pontos use o bot\~{a}o
       direito do mouse.'))
    hold on

    % Initially, the list of points is empty.
    PE = [];
    n = 0;
    but = 1;
    while but == 1
        [xi,yi,but] = ginput(1);
        plot(xi,yi,'*g')
        n = n+1;
        PE(:,n) = [xi;yi];
    end
    PE = PE';
    setappdata(handles.PontoExtra,'PE',PE)
    set(handles.Display,'String',sprintf('Pontos adicionados
       corretamente!\n \n Para prosseguir com o processamento da
       imagem pressione o bot\~{a}o:\n "Filtrar Pontos"'))
else
```

```
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end

% --- Executes on button press in FiltrarPontos.
function FiltrarPontos_Callback(hObject, eventdata, handles)
% hObject    handle to FiltrarPontos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
flag = getappdata(handles.B_LerArquivo,'flag');

if flag == [1 1 1 1 1 0 0 0]
    m = getappdata(handles.CriarGrid,'m');
    m2 = getappdata(handles.CriarGrid,'m2');
    x = getappdata(handles.CriarGrid,'x');
    y = getappdata(handles.CriarGrid,'y');
    controle = getappdata(handles.PontoExtra,'controle');
    xy = getappdata(handles.SelecionarArea,'xy');
    D = getappdata(handles.RecortarImagem,'imagemD');
    x = x';
    y = y';
    if controle == 1
        PE = getappdata(handles.PontoExtra,'PE');
        v = PE(:,1);
        t = PE(:,2);
        [gridf1]=getpontos(m,m2,x,y);
        [gridf2]=getpontosextra(m,m2,v,t);
        gridf = vertcat(gridf1,gridf2);
    else
        [gridf]=getpontos(m,m2,x,y);
    end

    cla(handles.axes1,'reset')% reseta a area do gr\'{a}fico

    imshow(D);
    hold on
    plot(xy(:,1),xy(:,2),'.-b',gridf(:,1),gridf(:,2),'*g')

    setappdata(handles.FiltrarPontos,'gridf',gridf)
    setappdata(handles.B_LerArquivo,'flag',[1 1 1 1 1 1 0 0])
    set(handles.Display,'String',sprintf('Pontos Filtrados
       Corretamente!\n \n Para Salvar os pontos em arquivo, pressione
        o bot\~{a}o: "Salvar Arquivos de Coordenadas".'))
```

```
else
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end


% --- Executes on button press in B_Salvar.
function B_Salvar_Callback(hObject, eventdata, handles)
% hObject    handle to B_Salvar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
flag = getappdata(handles.B_LerArquivo,'flag');
tamX = getappdata(handles.B_LerArquivo,'tamX');
altura = getappdata(handles.RecortarImagem,'altura')
if flag == [1 1 1 1 1 1 0 0]
    gridf = getappdata(handles.FiltrarPontos,'gridf');
    gridf = sortrows(gridf);
    aux2 = altura -gridf(:,2);
    gridf(:,2) = aux2;
    k = length(gridf);
    aux = zeros(k+2,2);
    for i=2:1:k+1
        aux(i,1)=gridf(i-1,1);
        aux(i,2)=gridf(i-1,2);
    end
    gridf = aux*25.4/tamX;
    [file,path] = uiputfile('*.mat','Save file name');
    dados = gridf;
    save(file,'dados')
    set(handles.Display,'String',sprintf('Arquivo Salvo Corretamente
       !\n \n Processo finalizado. Recarregue uma nova imagem e
       reinicie o processo.'))

else
    set(handles.Display,'String',sprintf('Opera\c{c}\~{a}o Inv\'{a}
       lida!\n \n Recarregue uma imagem v\'{a}lida e reinicie o
       processamento.'))
    setappdata(handles.B_LerArquivo,'flag',[0 0 0 0 0 0 0 0])
    setappdata(handles.B_LerArquivo,'flag2',0)
end

function Display_Callback(hObject, eventdata, handles)
% hObject    handle to Display (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Display as text
%        str2double(get(hObject,'String')) returns contents of
   Display as a double


% --- Executes during object creation, after setting all properties.
function Display_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Display (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Apêndice B

Código do Software de Controle dos Motores

A seguir será apresentado algumas das funções empregadas no código de controle dos motores.

```
function varargout = Controle_Motores(varargin)
% CONTROLE_MOTORES M-file for Controle_Motores.fig
%      CONTROLE_MOTORES, by itself, creates a new CONTROLE_MOTORES
   or raises the existing
%      singleton*.
%
%      H = CONTROLE_MOTORES returns the handle to a new
   CONTROLE_MOTORES or the handle to
%      the existing singleton*.
%
%      CONTROLE_MOTORES('CALLBACK',hObject,eventData,handles,...)
   calls the local
%      function named CALLBACK in CONTROLE_MOTORES.M with the given
   input arguments.
%
%      CONTROLE_MOTORES('Property','Value',...) creates a new
   CONTROLE_MOTORES or raises the
%      existing singleton*.  Starting from the left, property value
   pairs are
%      applied to the GUI before Controle_Motores_OpeningFcn gets
   called.  An
%      unrecognized property name or invalid value makes property
   application
%      stop.  All inputs are passed to Controle_Motores_OpeningFcn
   via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
   only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help
   Controle_Motores

% Last Modified by GUIDE v2.5 22-Mar-2013 09:05:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @Controle_Motores_OpeningFcn, ...
    'gui_OutputFcn',  @Controle_Motores_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Controle_Motores is made visible.
function Controle_Motores_OpeningFcn(hObject, eventdata, handles,
   varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Controle_Motores (see
   VARARGIN)

% Choose default command line output for Controle_Motores
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes Controle_Motores wait for user response (see UIRESUME
   )
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Controle_Motores_OutputFcn(hObject, eventdata,
   handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in LerCoordenadas.
function LerCoordenadas_Callback(hObject, eventdata, handles)
% hObject    handle to LerCoordenadas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if (ligado==1) && (ligadosis ==0)
    cla(handles.axes1,'reset')
    cla(handles.axes2,'reset')
    set(handles.axes1,'NextPlot','add')
    clc
    [filename1,filepath1]=uigetfile({'*.*','All Files'},'Select Data
        File 1');
    cd(filepath1);

    data=load([filepath1 filename1]);
    dad=data.dados;
    setappdata(handles.LerCoordenadas,'teste',dad)
    set(handles.Display,'String','Arquivo Carregado')
    set(handles.SelecaoPlotter,'value',1)
    set(handles.Display,'BackgroundColor',[0.75,0.75,0])
    set(handles.AjusteVelocidade,'value',0)
    set(handles.ShowVelocidade,'String',0)
    set(handles.ShowCoodX,'String','-')
    set(handles.ShowCoodY,'String','-')

elseif (ligado==1) && (ligadosis ==1)
    set(handles.Display,'String','Simula\c{c}\~{a}o e Sis.Motores
       ativados simultaneamente. Opera\c{c}\~{a}o inv\'{a}lida!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Simula\c{c}\~{a}o Desligada!')
```

```
    set(handles.Display,'BackgroundColor','red')
end



%--- Executes during object creation, after setting all properties.
function LerCoordenadas_CreateFcn(hObject, eventdata, handles)
% hObject    handle to LerCoordenadas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called



% --- Executes on slider movement.
function AjusteVelocidade_Callback(hObject, eventdata, handles)
% hObject    handle to AjusteVelocidade (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
   range of
%        slider
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==1 && ligadosis ==0
    slider_value = get(hObject,'Value');
    teste01 = floor(slider_value);
    teste02 = mod(slider_value,1);
    if teste02>=0 && teste02<0.125
        slider_value = teste01;
    elseif teste02>=0.125 && teste02 <0.375
        slider_value = teste01+0.25;
    elseif teste02>=0.375 && teste02 <0.625
        slider_value = teste01+0.50;
    elseif teste02>=0.625 && teste02 <0.875
        slider_value = teste01+0.75;
    else
        slider_value = teste01+1;
    end
    set(handles.AjusteVelocidade,'value',slider_value)
    setappdata(handles.AjusteVelocidade,'veloz',slider_value)
    set(handles.ShowVelocidade,'String',slider_value)
    set(handles.Display,'String','Velocidade Ajustada')
    set(handles.Display,'BackgroundColor','white')
elseif ligado==1 && ligadosis ==1
```

```
    set(handles.Display,'String','Simula\c{c}\~{a}o e Sis.Motores
       ativados simultaneamente. Opera\c{c}\~{a}o inv\'{a}lida!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Simula\c{c}\~{a}o Desligada!')
    set(handles.Display,'BackgroundColor','red')
end


% --- Executes during object creation, after setting all properties.
function AjusteVelocidade_CreateFcn(hObject, eventdata, handles)
% hObject    handle to AjusteVelocidade (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function ShowVelocidade_Callback(hObject, eventdata, handles)
% hObject    handle to ShowVelocidade (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ShowVelocidade as
    text
%        str2double(get(hObject,'String')) returns contents of
%        ShowVelocidade as a double
% Ensure model is open
% Get the new value for the Kf Gain
NewStrVal = get(hObject,'String');
NewVal = str2double(NewStrVal);
if  isempty(NewVal) || (NewVal< 0) || (NewVal>10),
    OldVal = get(hObject.AjusteVelocidade,'Value');
    set(hObject,'String',OldVal)

else
    set(handles.AjusteVelocidade,'Value',NewVal)
end
% --- Executes during object creation, after setting all properties.
```

```
function ShowVelocidade_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ShowVelocidade (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in RotinaCalculos.
function RotinaCalculos_Callback(hObject, eventdata, handles)
% hObject    handle to RotinaCalculos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==1 && ligadosis ==0
    data=getappdata(handles.LerCoordenadas, 'teste');
    v_usuario = getappdata(handles.AjusteVelocidade, 'veloz');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Dados do Projeto Mec\^{a}nico Que Afetam a Programa\c{c}\~{a}o
    rp = 25/2; %raio primitivo da polia da correia
    N_mic_passos = 8000; %N\'{u}mero de divis\~{o}es, micro passos
       utilizados pelo drive
    %do motor, esse valor vem da configura\c{c}\~{a}o do drive.
    dist = 2*pi*rp; %dist\^{a}ncia percorrida por revolu\c{c}\~{a}o
       completa (360°) do motor
    dpp = dist/N_mic_passos; %distancia percorrida por pulso
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %IN\'{I}CIO PROGRAMA

    x_orig = data(:,1);
    y_orig = data(:,2);
    %verifica\c{c}\~{a}o do comando
    assignin('base','x_orig',x_orig) %-Verificado ok
    assignin('base','y_orig',y_orig) %-Verificado ok
    %----------------------------------------------------------------------------
```

```
%rotina de normaliza\c{c}\~{a}o das coordenadas, para determina\
   c{c}\~{a}o futura das
%velocidades vetoriais (mm/s)

%a rotina verifica o tamanho do vetor de coordenadas x, faz a
   pre aloca\c{c}\~{a}o
%da matriz v_norm(k,2). Essa matriz ser\'{a} preenchida
   inicialmente com zeros.
%A seguir \'{e} calculado com a fun\c{c}\~{a}o norm, a
   hipotenusa formada pelas
%coordenadas x e y de cada linha do vetor. Antes de normalizar
   os vetores
%de coordenadas, \'{e} realizado um teste para verificar se n\~{
   a}o h\'{a} alguma
%coordenada em zero, e assim eliminar o resultado NAN do Matlab,
    quando
%zero \'{e} divido por alguma coisa.

k = length(x_orig);
v_norm = zeros(k,2);
Data_ALL=zeros(k,7);

for i=1:1:k
    aux1 = norm([x_orig(i) y_orig(i)]);
    if i>1
        if x_orig(i)==0 && y_orig(i)==0
            v_norm(i,1) = 0;
            v_norm(i,2) = 0;
        elseif (x_orig(i)==x_orig(i-1) && y_orig(i)==y_orig(i-1)
           )
            v_norm(i,1)=0;
            v_norm(i,2)=0;
        elseif (x_orig(i)==x_orig(i-1))
            v_norm(i,1)=0;
            v_norm(i,2)=1;
        elseif (y_orig(i)==y_orig(i-1))
            v_norm(i,1)=1;
            v_norm(i,2)=0;
        else
            v_norm(i,1)= x_orig(i)/aux1;
            v_norm(i,2)= y_orig(i)/aux1;
        end
    else
        if x_orig(i)==0 && y_orig(i)==0
            v_norm(i,1) = 0;
            v_norm(i,2) = 0;
```

```
        else
            v_norm(i,1)= x_orig(i)/aux1;
            v_norm(i,2)= y_orig(i)/aux1;
        end
    end
end
%-------------------------------------------------------------------------

%rotina para calculo do n\'{u}mero de pulsos de cada motor

% aqui \'{e} calculado o n\'{u}mero de pulsos para que ocorra
   cada um dos
% deslocamentos, coordenadas por coordenadas
x_pulsos = zeros(k,1);
y_pulsos = zeros(k,1);
for i=1:1:k
    if i>1
        if x_orig(i)==x_orig(i-1)
            x_pulsos(i)=0;
        else
            x_pulsos(i) = round(abs(x_orig(i)-x_orig(i-1))/dpp);
        end
    else
        x_pulsos(i) = round(x_orig(i)/dpp);
    end

    if i>1
        if y_orig(i)==y_orig(i-1)
            y_pulsos(i)=0;
        else
            y_pulsos(i) = round(abs(y_orig(i)-y_orig(i-1))/dpp);
        end
    else
        y_pulsos(i) = round(y_orig(i)/dpp);
    end
end
%-------------------------------------------------------------------------

%rotina para determinar o sentido do motor
% 5-sentido hor\'{a}rio
% 0-sentido anti-hor\'{a}rio
% os valores ser\~{a}o armazenados em uma matriz (k,2), em que a
    primeira
% coluna sentido motor x e segunda coluna sentido motor y.
sentido=zeros(k,2);
for i=1:1:k
    if i==1
```

```
        sentido(1,1)=5;
        sentido(1,2)=5;
    else
        if x_orig(i)>x_orig(i-1)
            sentido(i,1)=5;
        elseif x_orig(i)== x_orig(i-1)
            sentido(i,1)=sentido(i-1,1);
        else
            sentido(i,1)=0;
        end

        if y_orig(i)>y_orig(i-1)
            sentido(i,2)=5;
        elseif y_orig(i)==y_orig(i-1)
            sentido(i,2)=sentido(i-1,2);
        else
            sentido(i,2)=0;
        end
    end
end
%-------------------------------------------------------------------------

%rotina determinando a velocidade de cada eixo.

dist = zeros(k,1);
for i=2:1:k
    dist(i)=sqrt((x_orig(i)-x_orig(i-1))^2+(y_orig(i)-y_orig(i
       -1))^2);
end
tempo_mov = dist/v_usuario;
tempo_mov_icr = tempo_mov + 0.5;
v_eixos = v_usuario*v_norm;
fx=zeros(k,1);
fy=zeros(k,1);
for i=1:1:k
    if (x_pulsos(i)==0) || (tempo_mov(i)==0)
        fx(i)=0;
    else
        fx(i)=x_pulsos(i)/tempo_mov(i);
    end

    if (y_pulsos(i)==0) || (tempo_mov(i)==0)
        fy(i)=0;
    else
        fy(i)=y_pulsos(i)/tempo_mov(i);
    end
end
```

```
    Data_ALL(:,1)=x_orig;
    Data_ALL(:,2)=x_pulsos;
    Data_ALL(:,3)=fx;
    Data_ALL(:,4)=y_orig;
    Data_ALL(:,5)=y_pulsos;
    Data_ALL(:,6)=fy;
    Data_ALL(:,7)=tempo_mov_icr;
    assignin('base','Data_ALL',Data_ALL)
    display('DESCRI\c{C}\~{A}O DA VARI\'{A}VEL "Data_ALL"')
    display('"Data_ALL(:,1) => Coordenadas no Eixo X"')
    display('"Data_ALL(:,2) => N\'{u}mero de Pulsos Eixo X"')
    display('"Data_ALL(:,3) => Frequ\^{e}ncia do Motor Eixo X"')
    display('"Data_ALL(:,4) => Coordenadas no Eixo Y"')
    display('"Data_ALL(:,5) => N\'{u}mero de Pulsos Eixo Y"')
    display('"Data_ALL(:,6) => Frequ\^{e}ncia do Motor Eixo Y"')
    display('"Data_ALL(:,7) => Tempo Gasto em Cada Movimenta\c{c}\~{
       a}o"')
    display(Data_ALL)
    set(handles.Display,'String','Ver Resultado no WorkSpace')
    set(handles.Display,'BackgroundColor',[0.75,0.75,0])
    setappdata(handles.RotinaCalculos,'X',Data_ALL(:,1))
    setappdata(handles.RotinaCalculos,'Y',Data_ALL(:,4))
    setappdata(handles.RotinaCalculos,'NPX',Data_ALL(:,2))
    setappdata(handles.RotinaCalculos,'NPY',Data_ALL(:,5))
    setappdata(handles.RotinaCalculos,'FX',Data_ALL(:,3))
    setappdata(handles.RotinaCalculos,'FY',Data_ALL(:,6))
    setappdata(handles.RotinaCalculos,'T',Data_ALL(:,7))
    setappdata(handles.RotinaCalculos,'DPP',dpp)
    setappdata(handles.RotinaCalculos,'SX',sentido(:,1))
    setappdata(handles.RotinaCalculos,'SY',sentido(:,2))
elseif ligado==1 && ligadosis ==1
    set(handles.Display,'String','Simula\c{c}\~{a}o e Sis.Motores
       ativados simultaneamente. Opera\c{c}\~{a}o inv\'{a}lida!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Simula\c{c}\~{a}o Desligada!')
    set(handles.Display,'BackgroundColor','red')
end


% --- Executes on button press in ZerarMotores.
function ZerarMotores_Callback(hObject, eventdata, handles)
% hObject    handle to ZerarMotores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado = get(hObject,'value');
```

```
setappdata(handles.LigarSistema,'estado',estado)

ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==0 && ligadosis ==1
    z=20;
    w=10;
    set(handles.Display,'String','Motores em Movimento!')
    set(handles.Display,'BackgroundColor',[0.75,0.75,0])

    open_system('Controle_Zero');
    set_param('Controle_Zero','StopTime',num2str(z))

    set_param('Controle_Zero/PWMLX','Fc','50')
    set_param('Controle_Zero/Filtro_X','Ts',num2str(w))
    set_param('Controle_Zero/Sentido_Motor_X','Value','0')

    set_param('Controle_Zero/PWMLY','Fc','50')
    set_param('Controle_Zero/Filtro_Y','Ts',num2str(w))
    set_param('Controle_Zero/Sentido_Motor_Y','Value','0')
    set_param('Controle_Zero/FlagX','Value','1')
    set_param('Controle_Zero/FlagY','Value','1')
    set_param('Controle_Zero/Flag1','Value','1')
    set_param('Controle_Zero/Flag2','Value','1')
    set_param('Controle_Zero/Flag3','Value','0')
    set_param('Controle_Zero/Flag4','Value','0')
    set_param('Controle_Zero/Flag5','Value','1')
    set_param('Controle_Zero/Flag6','Value','1')
    set_param('Controle_Zero/Flag7','Value','1')
    set_param('Controle_Zero/Flag8','Value','0')
    set_param('Controle_Zero/Flag9','Value','1')
    set_param('Controle_Zero', 'SimulationCommand', 'Start')
    set(handles.Display,'String','Motores Posicionados!')
    set(handles.Display,'BackgroundColor',[0.75,0.75,0])
else
    set(handles.Display,'String','Painel de Controle dos Motores est
       \'{a} Desligado!')
    set(handles.Display,'BackgroundColor','red')
end


% --- Executes on button press in AvancarPosicao.
function AvancarPosicao_Callback(hObject, eventdata, handles)
% hObject    handle to AvancarPosicao (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==0 && ligadosis ==1
    grid on
    b = getappdata(handles.LigarSistema,'j');
    assignin('base','j',b)
    x_orig=getappdata(handles.RotinaCalculos, 'X');
    y_orig=getappdata(handles.RotinaCalculos, 'Y');
    x_pulsos=getappdata(handles.RotinaCalculos, 'NPX');
    y_pulsos=getappdata(handles.RotinaCalculos, 'NPY');
    fx=getappdata(handles.RotinaCalculos, 'FX');
    fy=getappdata(handles.RotinaCalculos, 'FY');
    dpp=getappdata(handles.RotinaCalculos, 'DPP');
    T=getappdata(handles.RotinaCalculos, 'T');
    SX=getappdata(handles.RotinaCalculos, 'SX');
    SY=getappdata(handles.RotinaCalculos, 'SY');
    k = length(x_orig);
    hold on
    b = b+1;
    if b<k && b>1
        z=T(b);
        sx=SX(b);
        sy=SY(b);
        if fx(b)==0
            w=1;
            d = 0.1;
        else
            w = (1/fx(b))*(fx(b)+x_pulsos(b));
            d=fx(b);
        end
        if fy(b)==0
            w2 = 1;
            f = 0.1;
        else
            w2 = (1/fy(b))*(y_pulsos(b)+fy(b));
            f=fy(b);
        end
        plot(handles.axes1,x_orig(b),y_orig(b),'r*')
        plot(handles.axes2,x_orig(b),y_orig(b),'*')
        setappdata(handles.LigarSistema, 'j',b);
        set(handles.Display,'String','Pressione "Avan\c{c}ar Posi\c{
            c}\~{a}o"')
        set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
        set(handles.ShowCoodX,'String',x_orig(b))
        set(handles.ShowCoodY,'String',y_orig(b))
        set_param('Controle_Motores_PWM2','StopTime',num2str(z))
        set_param('Controle_Motores_PWM2/PWM_X','Fc',num2str(d))
```

```
        set_param('Controle_Motores_PWM2/PWMLY','Fc',num2str(f))
        set_param('Controle_Motores_PWM2/Filtro_X','Ts',num2str(w))
        set_param('Controle_Motores_PWM2/Filtro_Y','Ts',num2str(w2))
        set_param('Controle_Motores_PWM2/Sentido_Motor_X','value',
           num2str(sx))
        set_param('Controle_Motores_PWM2/Sentido_Motor_Y','value',
           num2str(sy))
        set(handles.Display,'String','Aguarde motores em movimento')
        set(handles.Display,'BackgroundColor',[0.99,0.99,0])
        set_param('Controle_Motores_PWM2', 'SimulationCommand', '
           Start')
    elseif b==k
        if fx(b)==0
            w=1;
            d = 0.1;
        else
            w = (1/fx(b))*(fx(b)+x_pulsos(b));
            d=fx(b);
        end
        if fy(b)==0
            w2 = 1;
            f = 0.1;
        else
            w2 = (1/fy(b))*(y_pulsos(b)+fy(b));
            f=fy(b);
        end
        plot(handles.axes1,x_orig(b),y_orig(b),'r*')
        plot(handles.axes2,x_orig(b),y_orig(b),'*')
        setappdata(handles.LigarSistema, 'j',b);
        set(handles.Display,'String','Fim da Opera\c{c}\~{a}o')
        set(handles.Display,'BackgroundColor',[0.85,0.7,1])
        set(handles.ShowCoodX,'String',x_orig(k))
        set(handles.ShowCoodY,'String',y_orig(k))
    elseif b==1
        plot(handles.axes1,x_orig(b),y_orig(b),'r*')
        plot(handles.axes2,x_orig(b),y_orig(b),'*')
        setappdata(handles.LigarSistema, 'j',b);
    else
        set(handles.Display,'String','Comando Inv\'{a}lido')
        set(handles.Display,'BackgroundColor','red')
    end

    if b<=k
        if(y_pulsos(b)>x_pulsos(b))
            n1 = y_pulsos(b);
            yplot=zeros(n1,1);
            xplot=zeros(n1,1);
```

```
        Ty = (1/fy(b));
        Tx = (1/fx(b));
        for j=2:1:n1
            if(y_orig(b)>y_orig(b-1))
                yplot(1,1)=y_orig(b-1);
                yplot(j) = yplot(j-1)+dpp;
            else
                yplot(1,1)=y_orig(b-1);
                yplot(j) = yplot(j-1)-dpp;
            end
        end

        for j=2:1:n1
            if(x_orig(b)>x_orig(b-1))
                xplot(1,1)=x_orig(b-1);
                xplot(j) = xplot(1)+(round(j*Ty/Tx))*dpp;
            else
                xplot(1,1)=x_orig(b-1);
                xplot(j) = xplot(1)-(round(j*Ty/Tx))*dpp;
            end
        end
        hold on
        plot(handles.axes1,xplot,yplot)
        set(handles.Display,'String','Pressione "Avan\c{c}ar
           Posi\c{c}\~{a}o"')
        set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
    elseif(x_pulsos(b)>y_pulsos(b))
        n1 = x_pulsos(b);
        yplot=zeros(n1,1);
        xplot=zeros(n1,1);
        Ty = (1/fy(b));
        Tx = (1/fx(b));
        for j=2:1:n1
            if(x_orig(b)>x_orig(b-1))
                xplot(1,1)=x_orig(b-1);
                xplot(j) = xplot(j-1)+dpp;
            else
                xplot(1,1)=x_orig(b-1);
                xplot(j) = xplot(j-1)-dpp;
            end
        end
        for j=2:1:n1
            if(y_orig(b)>y_orig(b-1))
                yplot(1,1)=y_orig(b-1);
                yplot(j) = yplot(1)+(round(j*Tx/Ty))*dpp;
            else
                yplot(1,1)=y_orig(b-1);
```

```
                    yplot(j) = yplot(1)-(round(j*Tx/Ty))*dpp;
                end
            end
            hold on
            plot(handles.axes1,xplot,yplot)
            set(handles.Display,'String','Pressione "Avan\c{c}ar
               Posi\c{c}\~{a}o"')
            set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
        else
            n1 = x_pulsos(b);
            yplot=zeros(n1,1);
            xplot=zeros(n1,1);
            for j=2:1:n1
                if(x_orig(b)>x_orig(b-1))
                    xplot(1,1)=x_orig(b-1);
                    xplot(j) = xplot(j-1)+dpp;
                else
                    xplot(1,1)=x_orig(b-1);
                    xplot(j) = xplot(j-1)-dpp;
                end
            end
            for j=2:1:n1
                if(y_orig(b)>y_orig(b-1))
                    yplot(1,1)=y_orig(b-1);
                    yplot(j) = yplot(j-1)+dpp;
                else
                    yplot(1,1)=y_orig(b-1);
                    yplot(j) = yplot(j-1)-dpp;
                end
            end
            hold on
            plot(handles.axes1,xplot,yplot)
            set(handles.Display,'String','Pressione "Avan\c{c}ar
               Posi\c{c}\~{a}o"')
            set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
        end
    else
        set(handles.Display,'String','Todos os Pontos j\'{a} Foram
           Plotados')
        set(handles.Display,'BackgroundColor','red')
    end
elseif ligado==1 && ligadosis ==1
    set(handles.Display,'String','Simula\c{c}\~{a}o e Sis.Motores
       ativados simultaneamente. Opera\c{c}\~{a}o inv\'{a}lida!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Simula\c{c}\~{a}o Desligada!')
```

```
    set(handles.Display,'BackgroundColor','red')
end

% --- Executes on button press in LigarSistema.
function LigarSistema_Callback(hObject, eventdata, handles)
% hObject    handle to LigarSistema (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of LigarSistema
ligadosis = get(hObject,'value');
setappdata(handles.LigarSistema,'ligadosis',ligadosis)
if ligadosis==1
    open_system('Controle_Motores_PWM2');
end
j=0;
setappdata(handles.LigarSistema,'j',j)
cla(handles.axes1,'reset')
cla(handles.axes2,'reset')
set(handles.axes1,'NextPlot','add')
set(handles.ShowCoodX,'String','Aguarde')
set(handles.ShowCoodY,'String','Aguarde')
a=0;
setappdata(handles.AvancarPosicao,'J',a)
set(handles.Display,'String','Pressione "Avan\c{c}ar Posi\c{c}\~{a}o
    "')
set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])


% --- Executes on button press in AvancarSimulacao.
function AvancarSimulacao_Callback(hObject, eventdata, handles)
% hObject    handle to AvancarSimulacao (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==1 && ligadosis ==0
    grid on
    a=getappdata(handles.SelecaoPlotter, 'a');
    opcao=getappdata(handles.SelecaoPlotter,'opcao');
    if opcao == 2
        x_orig=getappdata(handles.RotinaCalculos, 'X');
        y_orig=getappdata(handles.RotinaCalculos, 'Y');
        x_pulsos=getappdata(handles.RotinaCalculos, 'NPX');
        y_pulsos=getappdata(handles.RotinaCalculos, 'NPY');
        fx=getappdata(handles.RotinaCalculos, 'FX');
        fy=getappdata(handles.RotinaCalculos, 'FY');
```

```
        dpp=getappdata(handles.RotinaCalculos, 'DPP');
        k = length(x_orig);
        hold on
        a = a+1;
        if a<k
            plot(handles.axes1,x_orig(a),y_orig(a),'r*')
            plot(handles.axes1,x_orig(a),y_orig(a),'b-')
            plot(handles.axes2,x_orig(a),y_orig(a),'*')
            setappdata(handles.SelecaoPlotter, 'a',a);
            set(handles.Display,'String','Pressione "Avan\c{c}ar
               Posi\c{c}\~{a}o"')
            set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
            set(handles.ShowCoodX,'String',x_orig(a))
            set(handles.ShowCoodY,'String',y_orig(a))
        elseif a==k
            plot(handles.axes1,x_orig(a),y_orig(a),'r*')
            plot(handles.axes2,x_orig(a),y_orig(a),'*')
            setappdata(handles.SelecaoPlotter, 'a',a);
            set(handles.Display,'String','Fim da Simula\c{c}\~{a}o')
            set(handles.Display,'BackgroundColor',[0.85,0.7,1])
            set(handles.ShowCoodX,'String',x_orig(k))
            set(handles.ShowCoodY,'String',y_orig(k))
        else
            set(handles.Display,'String','Comando Inv\'{a}lido')
            set(handles.Display,'BackgroundColor','red')
        end
        if a>1 && a<=k
            plot(handles.axes1,[x_orig(a-1) x_orig(a)],[y_orig(a-1)
               y_orig(a)],'b-')
        end
    else
        set(handles.Display,'String','Comando Inv\'{a}lido')
        set(handles.Display,'BackgroundColor','red')
    end
elseif ligado==1 && ligadosis ==1
    set(handles.Display,'String','Simula\c{c}\~{a}o e Sis.Motores
       ativados simultaneamente. Opera\c{c}\~{a}o inv\'{a}lida!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Simula\c{c}\~{a}o Desligada!')
    set(handles.Display,'BackgroundColor','red')
end



function ShowCoodX_Callback(hObject, eventdata, handles)
% hObject    handle to ShowCoodX (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ShowCoodX as text
%        str2double(get(hObject,'String')) returns contents of
   ShowCoodX as a double



% --- Executes during object creation, after setting all properties.
function ShowCoodX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ShowCoodX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Display_Callback(hObject, eventdata, handles)
% hObject    handle to Display (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Display as text
%        str2double(get(hObject,'String')) returns contents of
   Display as a double



% --- Executes during object creation, after setting all properties.
function Display_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Display (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function ShowLigarSistema_Callback(hObject, eventdata, handles)
% hObject    handle to ShowLigarSistema (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ShowLigarSistema
   as text
%        str2double(get(hObject,'String')) returns contents of
   ShowLigarSistema as a double


% --- Executes during object creation, after setting all properties.
function ShowLigarSistema_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ShowLigarSistema (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in SelecaoPlotter.
function SelecaoPlotter_Callback(hObject, eventdata, handles)
% hObject    handle to SelecaoPlotter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
   SelecaoPlotter contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
   SelecaoPlotter
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==1 && ligadosis ==0
    opcao = get(hObject,'value');
    switch opcao
```

```
case 1
    set(handles.Display,'String','Escolha o M\'{e}todo de
       Simula\c{c}\~{a}o!')
    set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])

case 2
    cla(handles.axes1,'reset')
    cla(handles.axes2,'reset')
    set(handles.axes1,'NextPlot','add')
    set(handles.ShowCoodX,'String','Aguarde')
    set(handles.ShowCoodY,'String','Aguarde')
    a=0;
    setappdata(handles.SelecaoPlotter,'a',a)
    setappdata(handles.SelecaoPlotter,'opcao',opcao)

    set(handles.Display,'String','Pressione "Avan\c{c}ar
       Posi\c{c}\~{a}o"')
    set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
case 3
    %-------------------------------------------------------------------

    %%rotina de plotagem, este ser\'{a} o movimento
       realizado pelo manipulador.
    % Essa rotina foi tra\c{c}ada increamento, passo a passo
        o deslocamento de cada
    % um dos motores, conforme interpola\c{c}\~{a}o linear
       necess\'{a}ria ao movimento.
    cla(handles.axes1,'reset')
    cla(handles.axes2,'reset')
    set(handles.axes1,'NextPlot','add')
    set(handles.ShowCoodX,'String','-')
    set(handles.ShowCoodY,'String','-')
    x_orig=getappdata(handles.RotinaCalculos, 'X');
    y_orig=getappdata(handles.RotinaCalculos, 'Y');
    x_pulsos=getappdata(handles.RotinaCalculos, 'NPX');
    y_pulsos=getappdata(handles.RotinaCalculos, 'NPY');
    fx=getappdata(handles.RotinaCalculos, 'FX');
    fy=getappdata(handles.RotinaCalculos, 'FY');
    dpp=getappdata(handles.RotinaCalculos, 'DPP');
    k = length(x_orig);
    grid on
    plot(handles.axes1,x_orig,y_orig,'r*')
    plot(handles.axes1,x_orig,y_orig,'b-')
    plot(handles.axes2,x_orig,y_orig,'*');
    set(handles.Display,'String','Simula\c{c}\~{a}o Conclu
       \'{\i}da!')
    set(handles.Display,'BackgroundColor',[0.84,0.91,0.85])
```

```
    end
elseif ligado==1 && ligadosis ==1
    set(handles.Display,'String','Simula\c{c}\~{a}o e Sis.Motores
       ativados simultaneamente. Opera\c{c}\~{a}o inv\'{a}lida!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Simula\c{c}\~{a}o Desligada!')
    set(handles.Display,'BackgroundColor','red')
end



% --- Executes during object creation, after setting all properties.
function SelecaoPlotter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SelecaoPlotter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: popupmenu controls usually have a white background on
   Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function ShowCoodY_Callback(hObject, eventdata, handles)
% hObject    handle to ShowCoodY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ShowCoodY as text
%        str2double(get(hObject,'String')) returns contents of
   ShowCoodY as a double


% --- Executes during object creation, after setting all properties.
function ShowCoodY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ShowCoodY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in LigarSimulacao.
function LigarSimulacao_Callback(hObject, eventdata, handles)
% hObject    handle to LigarSimulacao (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ligado = get(hObject,'value');
setappdata(handles.LigarSimulacao,'ligado',ligado)
setappdata(handles.LigarSistema,'ligadosis',0)
%set(handles.ShowLigarSistema,'String','Aguardando Simula\c{c}\~{a}o
    ')
%assignin('base','opcao02',ligado)

% Hint: get(hObject,'Value') returns toggle state of LigarSimulacao


% --- Executes on button press in RetornarPosicao.
function RetornarPosicao_Callback(hObject, eventdata, handles)
% hObject    handle to RetornarPosicao (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of
    edit10 as a double


% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of
   edit11 as a double


% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of
   edit12 as a double


% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of
   edit13 as a double


% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
   called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in BotaoEmergencia.
function BotaoEmergencia_Callback(hObject, eventdata, handles)
% hObject    handle to BotaoEmergencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ligado = getappdata(handles.LigarSimulacao,'ligado');
ligadosis = getappdata(handles.LigarSistema,'ligadosis');
if ligado==0 && ligadosis ==1
```

```
    estado = getappdata(handles.ZerarMotores,'estado');
    if estado==1
        set_param('Controle_Zero/Flag6','Value','0')
        set_param('Controle_Zero/Flag7','Value','0')
    else
        display('funcionou')
    end
    set(handles.Display,'String','Parada de Emerg\^{e}ncia Acionada
       !')
    set(handles.Display,'BackgroundColor',[0.75,0.75,0])
elseif ligado==1 && ligadosis==1
    set(handles.Display,'String','Opera\c{c}\~{a}o Inv\'{a}lida
       Simula\c{c}\~{a}o Ativada!')
    set(handles.Display,'BackgroundColor','red')
else
    set(handles.Display,'String','Painel de Controle dos Motores est
       \'{a} Desligado!')
    set(handles.Display,'BackgroundColor','red')
end


% --- Executes on button press in PlugMotor.
function PlugMotor_Callback(hObject, eventdata, handles)
% hObject    handle to PlugMotor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in PosicionarRaioX.
function PosicionarRaioX_Callback(hObject, eventdata, handles)
% hObject    handle to PosicionarRaioX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Bibliografia

V. D. M. D. Anthony, C. Robert, and M. T. Clare. Real-time magnetic resonance image-guided interstitial brachytherapy in the treatment of select patients with clinically localized prostate cancer. *Int. J. Radiation Oncology Biol. Phys*, 1998.

Centro de Combate ao Câncer. Atualmente, a forma mais adequada de definir um tratamento parte do princípio da "medicina baseada em evidências". a partir daí, existem vários caminhos possíveis., 2012. URL `http://www.cccancer.net/site/index.php/tipos-de-tratamento/`. [Online; Acessado em 26 de março de 2013].

I. T. Costa and T. P. R. Campos. Resposta radiodosimétrica de implantes de sementes de biovidros radioativos no cérebro de coelhos. *Revista Matéria*, 2007.

S. C. B. Esteves, A. C. Z. Oliveira, and L. F. A. Feijó. Braquiterapia no brasil. *Radiol Bras*, 2004.

D. Grabarz and R. R. Hattori. Indicações de radiocirurgia e estereotaxia fracionada nos tumores de sistema nervoso central. *Revista Oncologia*, 66, 2009.

Andrew Heavens. Global cancer cases seen surging 75 percent by 2030, 2012. URL `http://www.reuters.com/article/2012/05/31/`. [Online; Acessado em 25 de março de 2013].

International Agency for Research Cancer IARC and Cancer Research UK CRUK. World cancer factsheet, 2012.

Instituto Nacional de Câncer INCA. *Programa Nacional de Controle do Câncer da Próstata*. INCA, 2002.

Instituto Nacional de Câncer INCA. *Quimioterapia - Orientações aos Pacientes*. INCA, 2010.

M. A. Meltsner, N. J. Ferrier, and B. R. Thomadsen. Observations on rotating needle insertions using a brachytherapy robot. *Physics in Medicine and Biology*, 2007.

W. S. Roberto, T. P. R. Campos, and M. M. Pereira. Desenvolvimento e análises de sementes sintetizadas através da rota sol-gel para implantes em tumores de próstata. Master's thesis, Universidade Federal de Minas Gerais. Escola de Engenharia da UFMG, 2005.

J. V. Salvajoli and B. P. Salvajoli. O papel da radioterapia no tratamento câncer - avanços e desafios. *Revista Onco*, 2012.

Brasil Ministério da Saúde. Instituto nacional de câncer, 2013. URL `http://www.inca.gov.br`. [Online; Acessado em 25 de março de 2013].

J. E. Shigley, C. R. Mischke, and R. G. Budynas. *Projeto de Engenharia Mecânica*. Bookman, 2005.

A. L. Trejos, A. W. Lin, and S. Mohan. Mira v: An integrated system for minimally invasive robot-assisted lung brachytherapy. *International Conference on Robotics and Automation*, 2008.