CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Unidade Araxá

MÁRLON AFONSO BORGES DAMASCENO

ROBÔ SEGUIDOR DE ALVOS MÓVEIS COM MIRA A *LASER* AUTOMÁTICA

Araxá-MG

2017

MÁRLON AFONSO BORGES DAMASCENO

ROBÔ SEGUIDOR DE OBJETOS MÓVEIS COM MIRA A *LASER* AUTOMÁTICA

Trabalho de Conclusão de Curso apresentado ao Centro Federal de Educação Tecnológica de Minas Gerais - Unidade Araxá, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Automação Industrial.

Orientador: Prof. Me. Sérgio Luiz da Silva Pithan

Araxá-MG

2017

MÁRLON AFONSO BORGES DAMASCENO

ROBÔ SEGUIDOR DE OBJETOS MÓVEIS COM MIRA A LASER AUTOMÁTICA

Trabalho de Conclusão de Curso apresentado ao Centro Federal de Educação Tecnológica de Minas Gerais - Unidade Araxá, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Automação Industrial.

Defesa: Araxá, 05 de dezembro de 2017.

BANCA EXAMINADORA

Me. SÉRGIO LUIZ DA SILVA PITHAN - Orientador
Centro Federal de Educação Tecnológica de Minas Gerais - Unidade Araxá

Me. ALEXANDRE DIAS LINHARES - Avaliador
Centro Federal de Educação Tecnológica de Minas Gerais - Unidade Araxá

Dra. ALINE FERNANDA BIANCO - Avaliadora
Centro Federal de Educação Tecnológica de Minas Gerais - Unidade Araxá

“E conhecereis a verdade, e a verdade vos libertará.”

JOÃO, 8:32.

RESUMO

O processamento digital de imagens permite aos sistemas identificarem e processarem características contidas em uma cena ou imagem capturada por diversos aparelhos, como por exemplo, câmeras, aparelhos de raios x, ultrassom. Este trabalho apresenta um protótipo em forma de robô que foi desenvolvido para identificar os contornos de objetos e apontar, através de uma mira laser, o alvo em questão. O sistema é constituído por uma câmera webcam que fornece a imagem de entrada, em seguida o processamento digital da imagem é realizado pelo software computacional Matlab e os comandos físicos executados por microcontrolador Arduino, servo motores e articulação mecânica. O protótipo tentou identificar diferentes formas de objetos com diferentes variações, sendo, mudança de posição e rotação, através de rotinas de programação com técnicas diferentes de processamento digital de imagens implementadas, apresentadas e testadas ao longo deste trabalho.

Palavras-chave: Processamento digital de imagens. Mira laser. Microcontrolador Arduino. Software computacional Matlab. Servo motores.

ABSTRACT

Digital image processing allows systems to identify and process features contained in a scene or image captured by various devices, such as cameras, x-ray machines, ultrasound. This work presents a prototype in the form of a robot that was developed to identify the contours of objects and to point, through a laser sight, the target in question. The system consists of a webcam camera that provides the input image, then the digital image processing is performed by Matlab computational software and the physical commands performed by Arduino microcontroller, self-motor and mechanical articulation. The prototype tried to identify different forms of objects with different variations, as change of position and rotation, through programming routines with different techniques of digital image processing implemented, as was presented and tested throughout this work.

Keywords: Digital processing of images. Laser target. Arduino microcontroller. Computacional Matlab software. Servo motors.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ATS - Automated Targeting System
Sistema automatizado de alvos

GND - Ground
Terra, ponto comum de referência de tensão elétrica

IDE - Plataforma de programação do Arduino

PWM - Pulse Width Modulation;

UART - Universal Asynchronous Receiver/Transmiter
Receptor/Transmissor Assíncrono Universal

USB - Universal Serial Bus
Porta de comunicação serial

SUMÁRIO

1 INTRODUÇÃO

Na opinião de Gonzalez e Woods (2010) a visão é o mais avançado dos sentidos que os seres humanos possuem e que lhes proporciona uma melhor percepção do mundo ao seu redor. Assim sendo, *não é de surpreender que as imagens exerçam o papel mais importante na percepção humana* (p. 1).

Essa percepção de mundo em forma visual é limitada à faixa de luz visível dentre um espectro de radiação eletromagnética presente no planeta, onde essa luz é fornecida por uma fonte, atinge o objeto e o objeto reflete parte dessa luz em direção aos olhos e, no interior destes, se encontram os cones e bastonetes responsáveis por identificar as diferentes intensidades presentes na luz que é recebida. Em seguida, esses sinais são transmitidos através do nervo óptico até o cérebro, que processa e ordena os sinais recebidos, formando a imagem.

Nos equipamentos dotados de processamento de imagens, o processo de identificação de imagens se dá em forma similar à da visão humana, porém esses equipamentos podem trabalhar com quase todo o espectro eletromagnético, não estando apenas condicionado à banda de luz visível, o que torna o processamento digital de imagens um campo bem amplo de aplicações em variadas áreas do conhecimento (GONZALEZ, WOODS, 2010). Assim sendo,

> [...] os aparelhos de processamento de imagem cobrem quase todo o espectro EM, variando de ondas gama a ondas de rádio. Eles podem trabalhar com imagens geradas por fontes que os humanos não estão acostumados a associar com imagens. Essas fontes incluem ultrassom, microscopia eletrônica e imagens geradas por computador. Dessa forma, o processamento digital de imagens inclui um amplo e variado campo de aplicações (GONZALEZ, WOODS, 2010, p. 1).

A classificação de imagens está presente em diversos momentos da vida sendo necessária a classificação de objetos e padrões durante a execução de um trabalho, de um processo, de um projeto *[...] e nem é preciso ressaltar o papel básico que desempenha como um necessário passo inicial antes de tarefas de seleção ou de tomada de decisão* (SOLOMON, BRECKON, 2013, p. 261).

Segundo Solomon e Breckon, considerando a conjuntura de processamento de imagens, *o objetivo da classificação é identificar propriedades, padrões ou estruturas características em uma imagem para acomodá-las (ou a própria imagem) em uma classe particular* (2013, p. 261).

Este trabalho tem como propósito a elaboração de um protótipo robotizado para classificação e identificação automatizada de objetos através de processamento de imagens que, em conjunto com uma articulação mecânica e microcontrolada, posiciona um marcador *laser* sobre um alvo e o acompanha durante uma trajetória. A classificação e identificação possibilitam a substituição de identificadores físicos e elementos sensores por câmeras, que juntamente com o software para processamento de imagens, formam o conjunto de entrada e processador de dados/sinais do sistema.

O usuário (operador do sistema) poderá, através de um computador, selecionar na imagem o objeto que mais lhe interessar, fazendo com que o robô posicione o identificador *laser* sobre esse objeto determinado e o acompanhe durante sua trajetória, até que lhe seja dado um comando para parar.

O processamento da imagem adquirida pela câmera se resume em tratar os dados da mesma como parâmetros de controle para ações de operação do protótipo em questão, ou seja, através do processamento da imagem com software computacional torna-se possível interpretar e extrair características necessárias para classificação e tomada de decisões pelo robô.

Existem artigos que tratam de assuntos variados sobre *Automated Targeting System* (ATS), que consiste num sistema automatizado de mira, onde uma câmera persegue movimentos genéricos ou, como no caso deste projeto, objetos específicos, com predeterminações diversas, como tamanho, formato, cor, textura, composição física, etc.

O desenvolvimento deste trabalho teve início em função do interesse do autor do mesmo por sistemas automatizados por processamento de imagem em substituição dos tradicionais sistemas automatizados. Em sistemas tradicionais de controle, sensores são construídos com características únicas de aplicação, ou seja, sensores que não são versáteis, pois atuam com limitação de aplicação. Com a evolução das câmeras e a acessibilidade financeira facilitada, houve um aumento no interesse pelos sistemas de identificação por imagem a baixo custo, versáteis e intercambiáveis, que funcionem tão bem quanto os tradicionalmente sensoriados.

O desenvolvimento deste trabalho visa validar uma maneira de sintetizar conhecimentos do curso de Engenharia de Automação Industrial nas áreas de tecnologia - Processamento de Imagens e Automação.

A apresentação de uma solução acessível e versátil, como uma câmera e um computador operando em conjunto e interagindo com um sistema físico, torna esse trabalho uma opção e uma motivação para o desenvolvimento de ideias inovadoras de aplicação na

área de Engenharia, despertando, inclusive, interesse pelo desenvolvimento de pesquisas voltadas para a indústria nacional, visto que trabalhos na área de sensores e aplicações com imagem são quase que exclusivamente importados.

A aplicação descrita por este trabalho sugere soluções automatizadas nas áreas de: segurança - com monitoramento automático de imagens definido sobre objeto com características pré-determinadas; industrial - com identificação de objetos em processo contínuo e; em meio acadêmico - com utilização do robô para aulas práticas em disciplinas do curso de Engenharia de Automação Industrial.

O objetivo geral deste trabalho é desenvolver um protótipo identificador de objetos móveis com *laser* onde o operador escolha um objeto dentre os existentes na cena, através de comando computacional, e o protótipo deve manter o *laser* sobre o objeto até que seja dado comando para que cesse a ação.

Para tanto, tem-se como objetivos específicos:

- Montar uma plataforma robótica física, que compreende estrutura física, hardware e lógica operacional para o microcontrolador;
- Montar interfaces físicas entre software e hardware;
- Promover programação adequada para execução de ações pelo robô;
- Extrair atributos de controle da imagem capturada pela câmera;
- Analisar o comportamento do sistema;
- Verificar se o controle aplicado atende aos requisitos estipulados;
- Desenvolver operacionalidade para o usuário do sistema.

2 REFERENCIAL TEÓRICO

2.1 Processamento de Imagens

De acordo com Gonzalez e Woods, uma imagem pode ser definida como um conjunto de pontos em um plano cartesiano de coordenadas *x* e *y*, onde, cada ponto possui uma intensidade particular. O sistema de coordenadas define a localização espacial do ponto e a intensidade quantifica o nível de um determinado pigmento presente em uma escala de 0 a 255. Quando essas definições são compreendidas de forma finita e discreta, ou seja, com valores mínimos e máximos definidos e de maneira a não haver valores espaciais intermediários (*x* e *y* sendo sempre números inteiros), a imagem é considerada como uma imagem digital. As imagens naturalmente geradas dependem de uma fonte de iluminação e da reflexão da luz por objetos contidos no cenário de interesse, sendo esse formato de imagem analógico em termos de coordenadas espaciais, havendo infinitos pontos espaciais na imagem, não permitindo a representação da imagem natural por meio matricial (discreto) (2010).

Para Solomon e Breckon, a representação discreta é interessante em processamento digital de imagens, pois é possível em um único ponto coexistir as coordenadas espaciais em conjunto com o coeficiente de intensidade, criando uma ideia matricial do sistema, podendo ser considerada uma representação contendo informação espaciais (*layout*) e de intensidade (cor). Uma imagem discreta bidimensional é o conjunto que representa amostragens em uma série de posições espaciais cartesianas, obtidas através do sinal captado por algum elemento sensor que execute um processo de amostragem de acordo com sua capacidade de resolução. Este processo de amostragem é denominado discretização (2013).

De acordo com Gonzalez e Woods, a digitalização dos valores de amplitude existente em cada coordenada cartesiana é chamada quantização. Cada par de coordenadas (x,y) cartesianas possui um valor quantizado de intensidade (2010).

"[...] há três formas básicas de representar $f(x,y)$" (GONZALEZ, WOODS, 2010, p. 35), podendo ser representada graficamente como uma superfície, onde é possível visualização em três eixos, (x, y, z), sendo, x e y coordenadas espaciais, resultado da amostragem e z a intensidade de $f(x,y)$ na coordenada, resultado da quantização.

A imagem pode ser representada como uma matriz de intensidade visual em escala de cinza, onde o nível de cinza é proporcional ao valor da intensidade de $f(x,y)$ e também pode ser representada como uma matriz numérica, com cada elemento de (linha, coluna) em sua respectiva coordenada espacial e seu valor numérico representando a intensidade de $f(x,y)$.

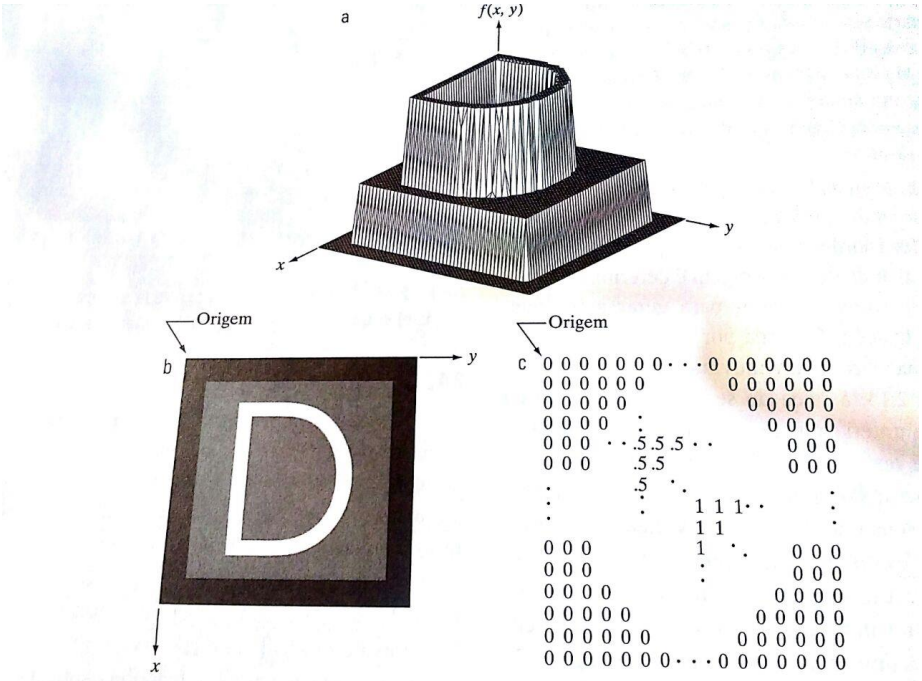


Figura 1 - (a) Imagem representada graficamente como uma superfície.
(b) Imagem representada como uma matriz de intensidade visual.
(c) Imagem representada como uma matriz numérica 2-D
(0, .5 e 1 correspondem ao preto, cinza e branco, respectivamente).
Fonte: GONZALEZ, WOODS, 2010, p. 36.

"Cada elemento dessa matriz é chamado de *elemento de imagem, elemento pictórico, pixel* ou *pel*" (GONZALEZ, WOODS, 2010, p. 37).

Conforme Gonzalez e Woods, por convenção, a representação cartesiana de uma imagem digital se dá considerando-se que a origem da imagem está localizada na parte superior esquerda, com o eixo x crescente para baixo e o eixo y crescente para a direita. Essa maneira de se apresentar espacialmente a imagem digital é relacionada ao modo de varredura de diversos equipamentos de visualização de imagens, os quais iniciam sua varredura do canto superior esquerdo e movendo-se para a direita (2010, p. 37).

Para se trabalhar com processamento de imagens são utilizadas técnicas diversas de transformação na imagem, sendo possível se atuar diretamente nos *pixels* ou realizar operações na transformada de uma imagem. As técnicas que atuam diretamente nos *pixels* são chamadas *técnicas de domínio espacial* e técnicas que atuam na transformada da imagem são chamadas *técnicas de domínio da frequência* (GONZALEZ, WOODS, 2010).

A *filtragem espacial* é um procedimento baseado em utilizar-se da vizinhança do *pixel* para, através de uma operação matemática (ou operador) transformar a intensidade de um *pixel*, onde o "[...] tipo de operação realizada na vizinhança determina a natureza do processo de filtragem" (GONZALEZ, WOODS, 2010, p. 69). O realce é um exemplo de transformação de intensidade.

Segundo Solomon e Breckon, o realce consiste em "[...] processar a imagem de modo que possamos ver e avaliar a informação visual nela contida com maior clareza". São exemplos de realce: agudização de bordas da imagem e efeito foco suave (ou *blurring*), podendo ser implementadas através de filtragem no domínio espacial (2013 p. 78).

"Um dos principais usos de filtragem linear e não linear em processamento de imagens é na remoção de ruído [...] como ruído aditivo [...] e ruído gaussiano [...]" (SOLOMON, BRECKON, 2013 p. 83).

Os filtros de média, mediana e por ordem realizam com um desempenho razoável a tarefa de filtragem do ruído através de *filtragem espacial*, sendo o filtro de média o filtro linear mais simples, podendo ser utilizado como passo preliminar do processamento para suavizar uma imagem. Por sua vez, o filtro de mediana, superior ao filtro de média, é mais eficaz na preservação de detalhes durante a filtragem do ruído. Já o filtro por ordem, que é um filtro que utiliza os valores mínimos e máximos da vizinhança do *pixel* para a eliminação do ruído, é de utilização restrita para determinadas naturezas de ruídos, pois pode amplificar o efeito do ruído em certas ocasiões. O filtro por ordem realiza a remoção do ruído, mas há uma perda na qualidade de detalhes da imagem (SOLOMON, BRECKON, 2013).

A filtragem gaussiana opera como *filtragem de domínio da frequência* e é um filtro muito importante, pois é capaz de remover o ruído com pouca perda de qualidade de detalhes na imagem (SOLOMON, BRECKON, 2013).

De acordo com Solomon e Breckon, a detecção de bordas é uma parte importante na aplicação de filtragem de imagens. Através da identificação de descontinuidades de intensidade nos *pixels* da imagem é possível representar o contorno presente em determinada região da imagem, sendo divididas em detecção de bordas de primeira e segunda ordem. Três núcleos de filtros de primeira ordem mais comumente utilizados para detecção de bordas são os detectores de bordas de Roberts, Prewitt e Sobel. Uma abordagem comumente utilizada para detecção de bordas de segunda ordem é o operador laplaciano. O operador de Roberts é de cálculo rápido, porém, sensível ao ruído e, os detectores de Prewitt e Sobel são mais robustos do que o de Roberts, porém, são ligeiramente mais complexos. O operador laplaciano permite uma detecção de bordas mais finas e aguçadas em comparação aos detectores de primeira ordem, porém, muito sensível ao ruído. Essa sensibilidade pode ser contornada utilizando a combinação de um filtro gaussiano, onde, primeiro se suaviza a imagem através de um núcleo gaussiano e logo após aplica-se o operador laplaciano. Como modo de economizar processamento, podem-se associar as duas operações em um único núcleo, o filtro laplaciano de gaussiano, onde é aplicado à imagem uma única vez (2013).

A segmentação de imagens é parte componente do trabalho, sendo importante para se subdividir a imagem em regiões ou objetos que a compõem, podendo ser o primeiro passo vital para tarefas de extração de características, classificação, descrição, associando rótulos que façam sentido (SOLOMON, BRECKON, 2013).

2.2 Arduino

Arduino é uma plataforma eletrônica de software e hardware, conceito surgido na Itália em 2005, que possibilita a prototipagem com custo razoável.

A plataforma é utilizada como interface para controle e comando de motores de passo, visto que possui saídas compatíveis para comando de elementos finais de controle, bem como pinos de entrada para monitoramento de variáveis diversas, sendo parte versátil do trabalho no quesito sensoriamento e com um grande diferencial em modo de programação, por ser em código aberto.

Existe um portfólio de aplicações e informações sobre as variedades da plataforma Arduino, modos e linguagens de programação na IDE específica, modelos e aplicações possíveis descritas com trocas de experiências disponíveis no site www.arduino.cc.

O microcontrolador utilizado neste trabalho é o modelo Arduino® MEGA 2560. É um microcontrolador que possui 54 entradas/saídas digitais (15 destas podem ser utilizadas como saídas PWM), 16 saídas analógicas, 4 UART’s (portas seriais assíncronas), um cristal oscilador de 16 MHz, um conector USB e um conector P8 macho, para alimentação externa de 7 - 12 Vcc. Sua função neste protótipo é controlar e comandar os servo motores e o módulo laser, recebendo sinais de comando do software MATLAB, através da porta serial USB, interpretando e repassando as ações requeridas aos servo motores e ao módulo laser, em formato de sinal PWM, ou seja, funcionando como interface entre os comandos do MATLAB e os servo motores da articulação mecânica.

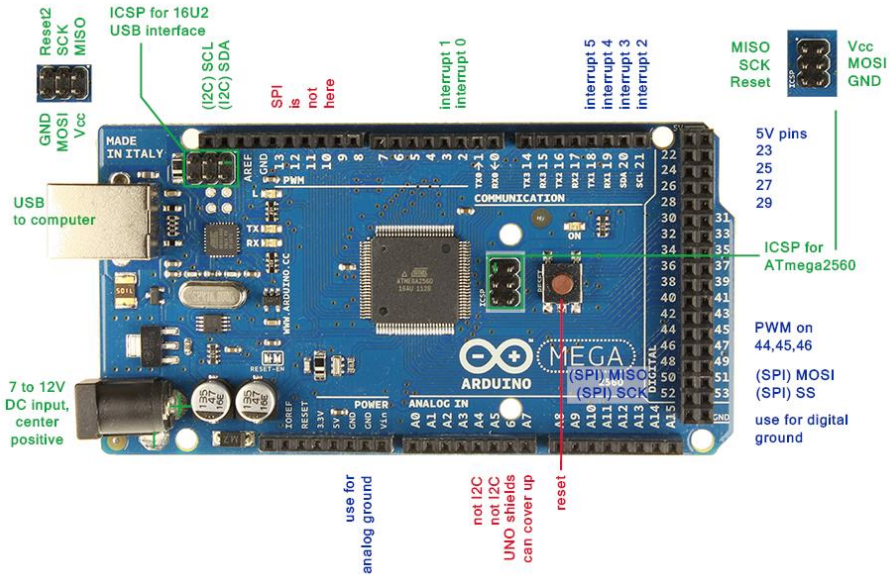


Figura 1 - Microprocessador Arduino MEGA 2560.
Fonte: https://arduino-info.wikispaces.com/MegaQuickRef (12/06/2017).

2.3 MATLAB

O software para programação, simulação e processamento é o MATLAB versão 2015. Sua função é servir de interface entre o meio externo e o protótipo, processando elementos de imagem e enviando comandos através de porta USB ao microcontrolador Arduino® MEGA 2560.

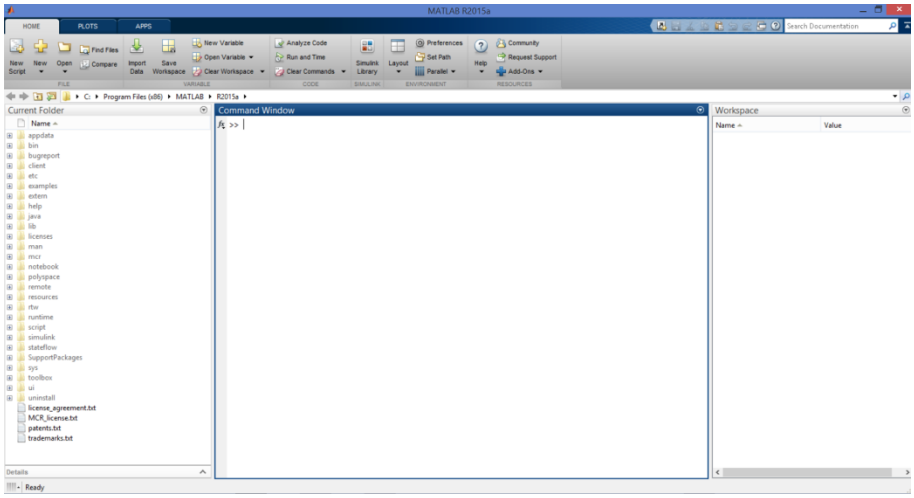


Figura 2 - Janela inicial do Matlab® versão 2015.

O MATLAB® é uma linguagem de programação poderosa em termos de computação técnica e vem da elisão das palavras MATrix LABoratory, oriundo de sua base operacional, que são matrizes (GILAT, 2006).

O MATLAB® é bastante versátil em cálculos matemáticos, modelagens, simulações, análises, sendo largamente utilizado nas universidades e faculdades, especialmente nas engenharias (GILAT, 2006).

O pacote-padrão do MATLAB® disponibiliza ferramentas e funções comuns a diversas áreas do conhecimento, além de possuir ferramentas adicionais, as toolboxes, que complementam o software, capacitando-o a resolver problemas específicos (GILAT, 2006). Dentre as *toolboxes* disponíveis está o processamento digital de imagens.

Há alguns anos, o MATLAB era formado por um público que possuía bastante conhecimento no campo de linguagem de programação, consequentemente a maior parte da literatura pressupunha um conhecimento prévio em programação pelo leitor. Porém, o MATLAB® foi-se desvinculando desses pré-requisitos, sendo inclusive adotado em cursos introdutórios nas universidades (GILAT, 2006).

2.4 Servo Motor

Servo motores são utilizados em aplicações onde existe a necessidade de uma movimentação precisa e controlada e que se mantenha a posição determinada, mesmo quando ocorrer atuação de forças em outra direção.

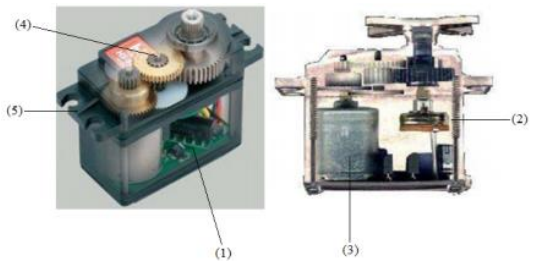


Figura 3 - Partes do servo motor.
Fonte: http://www.feis.unesp.br/Home/departamentos/engenhariaelétrica /aula-4---servo-motor-13-03-2013-final.pdf (02/06/2017).

O servo motor é composto por: circuito de controle (1), potenciômetro (2), motor (3), engrenagens (4), caixa do servo (5).

- O *circuito de controle* é responsável por monitorar a posição do eixo do motor e seu acionamento.
- O *potenciômetro* é utilizado para o monitoramento e manutenção da posição do eixo do servo motor.

- O *motor* é o elemento responsável por converter a energia elétrica de entrada em energia mecânica, que resulta em movimento de eixo e torque.
- As *engrenagens* que compõem o servo motor são reduções responsáveis por diminuir a rotação do eixo, transferindo maior torque ao eixo principal de saída.
- A *caixa do servo* é a carcaça que acondiciona todos os elementos do servo motor.

O servo motor é alimentado com uma tensão de 5Vcc. O comando do servo motor é realizado através de sinal em PWM - modulação por largura de pulso. O sinal em PWM é digital, ou seja, assume valores discretos, no caso do servo motor, valores de 0V e 5V. O que define o comando para posicionamento não é o valor da tensão, mas o período de tempo em que permanece o pulso de sinal.

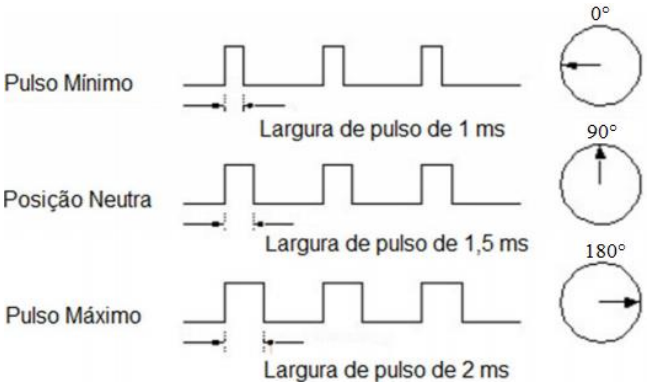


Figura 4 - Exemplo de comando no servo motor.
Fonte: http://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/aula-4---servo-motor-13-03-2013-final.pdf (02/06/2017).

O monitoramento realizado pelo circuito de controle do servo é de um período total de 20ms, onde um sinal de 5V durante 1ms corresponde à posição do eixo à 0 grau, um sinal de 5V durante 1,5ms corresponde à posição do eixo à 90 graus e um sinal de 5V durante 2ms corresponde à posição do eixo à 180 graus.

Ao se tentar movimentar forçadamente o servo motor alimentado e posicionado, surge um torque contrário para que se mantenha a posição do eixo.

O padrão de alimentação para servo motores geralmente encontrados no mercado é o que se segue:

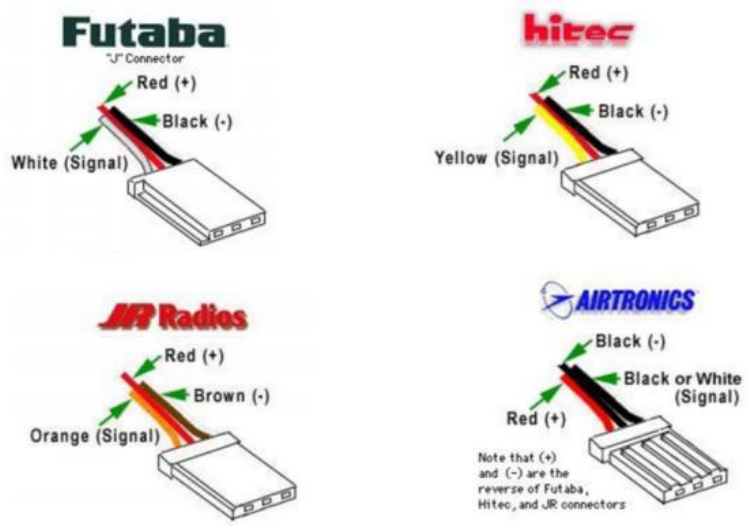


Figura 5 - Padrões comerciais de cores para conectores de servo motores.
Fonte: http://www.feis.unesp.br/Home/departamentos/engenhariaeletrica
/aula-4---servo-motor-13-03-2013-final.pdf (02/06/2017).

Os servo motores utilizados no trabalho são micro servos de 9 gramas, modelo SG90 marca Tower Pro, e seguem o padrão de plugues "JR Radios".

Figura 6 - Servo motor SG90 Tower Pro.
Fonte: http://blog.filipeflop.com/motores-e-servos/micro-servo
-motor-9g-sg90-com-arduino-uno.html (12/06/2017).

Segundo o manual do fabricante, as especificações técnicas dos servos motores utilizados no trabalho são:

- Tensão de operação: 3 - 7,2 Vcc
- Velocidade: 0,12 seg./60 Graus (a 4,8 Vcc e sem carga no eixo)
- Torque: 1,2 kg/cm (a 4,8 Vcc) e 1,6 kg/cm (a 6,0 Vcc)
- Peso: 9 gramas
- Temperatura de operação: -30°C a +60°C
- Dimensões: 32x30x12mm
- Tipo de engrenagens: Nylon
- Tamanho do cabo: 245 mm

2.5 Módulo *Laser*

Como maneira de se destacar uma identificação ou apresentar algo visualmente, utiliza-se a ferramenta de ponto *laser*, que consiste em um feixe de luz *laser* que é emitido controladamente na direção do objeto que se deseja destacar.

O módulo utilizado no trabalho para tal função é o modelo KY-008, do fabricante KEYES, e possui as seguintes características:

- Tensão de operação: 5 Vdc.
- Potência: 5 mW.
- Corrente: 30 mA a 5 Vdc.
- Cor: vermelha.
- Comprimento de onda: 650 nm.
- Peso: 2 gramas.
- Dimensões: 18 x 19 x 10 mm.

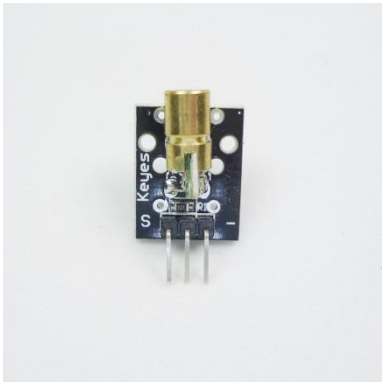


Figura 7 - Módulo laser KEYES KY-008.
Fonte: http://produto.mercadolivre.com.br/MLB-692218468-modulo-laser-keyes-ky-008-arduino-pic-_JM (02/08/2017).

O módulo KY-008 possui três pinos em sua placa, com sua respectiva função:

- Pino S: alimentação 5 Vdc.
- Pino central: não conectado.
- Pino "-": 0 Vdc.

2.6 Fonte de Alimentação

Todo circuito eletrônico para operar necessita de um elemento fornecedor de energia. Para tal finalidade, neste trabalho, é utilizada uma fonte comum de computador, modelo EP-280N / ATX312NS001L5A0, com os seguintes níveis de tensão de saída:

- + 12 Vdc (17 A) - fio amarelo;
- + 5,0 Vdc (21 A) - fio vermelho;
- + 3,3 Vdc (20 A) - fio laranja;
- - 5,0 Vdc (0,3 A) - fio branco;
- - 12 Vdc (0,8 A) - fio azul;
- GND (0 Vdc) - fio preto.

A tensão de alimentação da fonte selecionável entre 115 - 220 Vac, e consome uma corrente de 8 A.

3 MATERIAIS E MÉTODOS

A metodologia desse trabalho principia pela pesquisa bibliográfica sobre os conteúdos teóricos e técnicos relacionados a cada etapa envolvida na execução da parte experimental, de elaboração do protótipo, abaixo especificadas:

- Execução da montagem física dos componentes do projeto, criando um arranjo que possibilitasse a interação do software computacional, do microprocessador e dos periféricos envolvidos no projeto.
- Criação de interfaces entre software e hardware, integrando os sistemas como um todo, tornando possível se concretizar a abstração do software no mundo real, agregando dispositivos físicos (hardwares) que compreendam e executem as ações propostas pelo software e sua interface, realizando simulação e testes.
- Busca de conceitos do processamento de imagens, visando implementar o software para executar o objetivo que é identificar e acompanhar o objeto, executando simulações e comparando resultados.
- Operacionalizar o sistema, possibilitando interatividade entre um operador e o software.

O material a ser utilizado no trabalho está relacionado aos itens de necessidade operacional do robô, como itens mecânicos (articulação, plataforma, etc.), itens eletromecânicos (servo motores), itens eletrônicos (câmera, módulo *laser*, fonte de alimentação, protoboard para interligações elétricas, microprocessador, cabos, etc.), e itens computacionais (softwares de programação). A integração entre os itens, os dados, os resultados e as características do modelo do protótipo são os principais norteadores das necessidades para concepção do protótipo em si.

A obtenção de dados e resultados, assim como as simulações, somente foi possível após a montagem do aparato, o qual conferiu maior domínio e maior assertividade no campo real.

Como etapa importante do trabalho, a montagem do protótipo teve início pela articulação mecânica. A articulação utilizada é do modelo “Tilt-Pan”, que confere dois graus de liberdade de movimento. O termo “Tilt” faz referência ao movimento na vertical e o termo “Pan” ao movimento horizontal. A articulação utilizada tem dimensões de 11 cm (altura) x 7 cm (profundidade) x 3 cm (largura). Está apresentada na FIG. 7.

Figura 8 - Articulação "Tilt-Pan".
Fonte: https://pt.aliexpress.com/item/1set-Nylon-FPV-Pan-tilt-Camera-Mount-2pcs-Tower-Pro-SG90-9g-Servo-Retail-Promotion-Dropship/1623404058.html?spm=2114.42010308.4.2.JCnfyk (02/06/2017).

Cada grau de liberdade representa um eixo de movimento possível sendo no sistema cartesiano, um movimento no eixo x ("*Pan*") e outro movimento no eixo y ("*Tilt*"). Cada eixo recebe o seu respectivo servo motor, que é responsável pela execução do movimento e o posicionamento das partes.

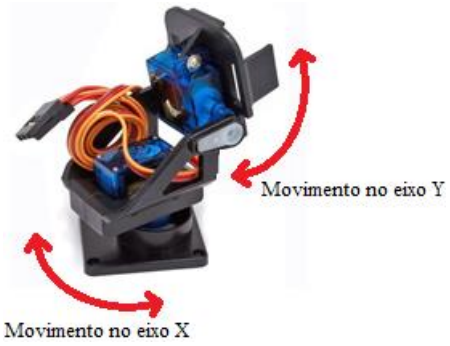


Figura 9 - Articulação com servo motores e sua respectiva movimentação.
Fonte: https://www.embarcg.com.br/suporte-pan-tilt-para-servo-motor-e-camera (16/08/2017).

Na parte superior da articulação está fixado o módulo laser KEYES KY-008, responsável por emitir a luz que identifica o objeto desejado, comandado pelo microprocessador.

O chassi utilizado no projeto foi a carcaça da própria fonte de alimentação, que abriga a articulação, o *laser*, o microprocessador, a câmera e o *protoboard*, além de cabos, espaçadores para placas de circuito impresso e bornes para conexão elétrica. A fonte de alimentação utilizada é uma fonte de alimentação comum para computadores, a qual necessitou de modificação em seu circuito interno. A mesma é de aplicação típica em gabinetes de computadores *desktop* que possuem circuitos internos que comandam sua operação, habilitando e desabilitando as saídas de tensão contínua da fonte conforme a necessidade operacional. Quando o computador está em pleno funcionamento, os circuitos de

controle habilitam as saídas de tensão contínua. Quando o computador está ocioso, as saídas de tensão são desabilitadas, permanecendo em modo *stand-by*. O pino que recebe o sinal para habilitar/desabilitar as saídas de tensão é o pino denominado “PS_ON” ou simplesmente “ON”. Quando habilitado, a placa mãe mantém esse pino em nível baixo (0 Vdc), e quando desabilitado, o pino permanece em nível alto de tensão. Pode ser identificado na fonte pelo fio verde claro. Para possibilitar que a fonte funcione externamente a um gabinete de computador, deve-se ligar o fio “PS_ON” em qualquer referência GND (0 Vdc).

As diferentes tensões contínuas fornecidas pela fonte de alimentação estão representadas na FIG. 9:

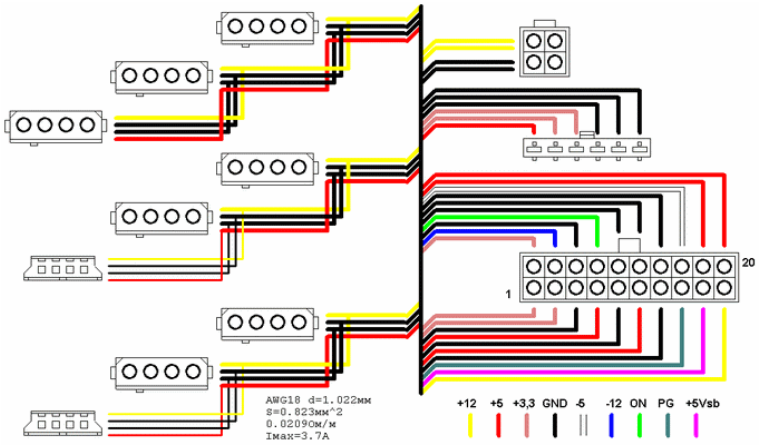


Figura 10 - Esquema de ligação da fonte de tensão contínua.
Fonte: https://www.electronica-pt.com/reparacao-fonte-atx (12/06/2017).

Foram utilizados os seguintes níveis de tensão da fonte de alimentação para o protótipo:

- +5Vcc (vermelho): Alimentação dos servo motores.
- +12Vcc (amarelo): Alimentação do microcontrolador Arduino® MEGA 2560.
- GND (preto): Referência 0 Vcc para o microcontrolador Arduino®, servo motores, módulo *laser* e habilita pino “PS_ON” (ou “ON”) da fonte.

A fonte alimenta o microcontrolador Arduino através de um plugue conector fêmea de 2,1 mm, com o fio amarelo (+12 Vcc) conectado ao pino do centro e o fio preto (GND) conectado ao pino externo.

Figura 11 - Plugue fêmea 2,1 mm para alimentação do Arduino.
Fonte: https://www.circuitar.com.br/tutoriais/esquema-eletrico-do-arduino-o-guia-definitivo/ (17/08/2017).

Os servo motores são alimentados pela fonte de tensão com +5 Vcc (fio vermelho) e GND (fio preto), conectados, respectivamente, ao fio vermelho (+Vcc) e ao fio marrom (-Vcc) dos servo motores.

Além da alimentação de tensão do sistema, são utilizados pinos do microcontrolador Arduino para enviar comandos aos servo motores e ao módulo *laser*. Dentre as entradas/saídas disponíveis no microcontrolador, são definidos 02 pinos de saída PWM para controle da movimentação dos servo motores e 01 pino de saída PWM para controle do acionamento do módulo *laser*, sendo:

- Pino 12: Servo motor para movimentos verticais (eixo y).
- Pino 08: Servo motor para movimentos horizontais (eixo x).
- Pino 07: Comando aciona módulo *laser*.

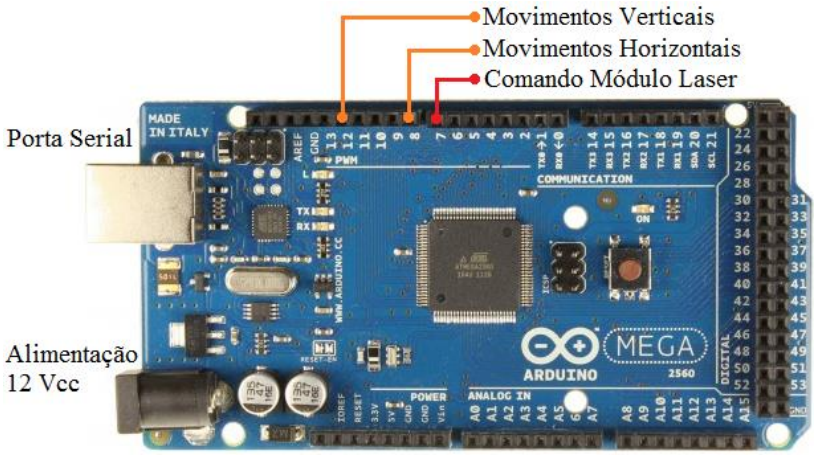


Figura 12 - Conexão ao microprocessador Arduino.
Fonte: http://www.electroschematics.com/7963/arduino-mega-2560-pinout/ (12/06/2017).

A montagem do protótipo é demonstrada pelo seguinte *layout*:

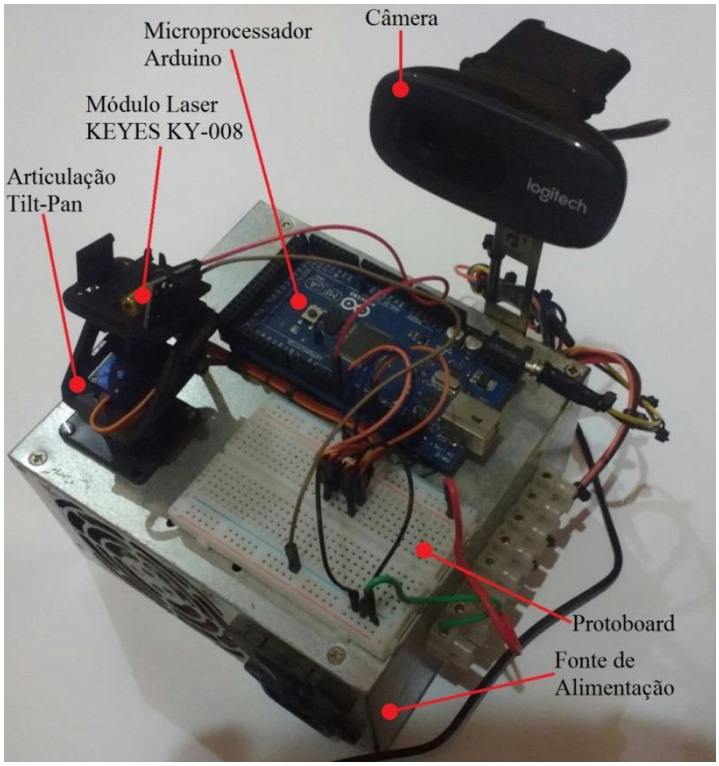


Figura 13 - Layout da montagem do robô.
Fonte: Autor.

A porta serial do Arduino é essencial para que seja realizada a conexão do microcontrolador ao computador externo, possibilitando a configuração e o envio de comandos diretamente ao microprocessador através de rotinas desenvolvidas no software (computador). Esta conexão é realizada através de um cabo USB comum que possua conector USB macho A em uma extremidade (ligada ao computador) e conector USB macho B na outra extremidade (ligada ao Arduino).

Figura 14 - Plugue USB Macho A.
Fonte: https://pt.aliexpress.com/item/IMC-hot-New-10pcs-Type-A-Male-USB-4-Pin-Plug-Socket-Connector-With-Black-Plastic/32576119112.html?spm=2114.42010208.4.50.uZHkHf (12/06/2017).

Figura 15 - Plugue USB Macho B.
Fonte:
http://www.cetronic.es/sqlcommerce/disenos/plantilla1/seccion/producto/DetalleProducto.jsp?idIdioma=&idTienda=93&codProducto=999158003&cPath=1064 (12/06/2017).

Para se executar a programação do Arduino, é utilizado software específico, denominado como compilador, responsável por verificar, validar e carregar a programação da rotina na memória do microprocessador. A rigor, compilar significa converter a linguagem de programação em linguagem ou código que possa ser lido ou corrido (processado) por um computador, que no caso do protótipo, será compilada uma rotina de programação para ser lida pelo microprocessador Arduino® MEGA 2560. Para a compilação de programas no Arduino, é utilizado o software “Arduino IDE”:

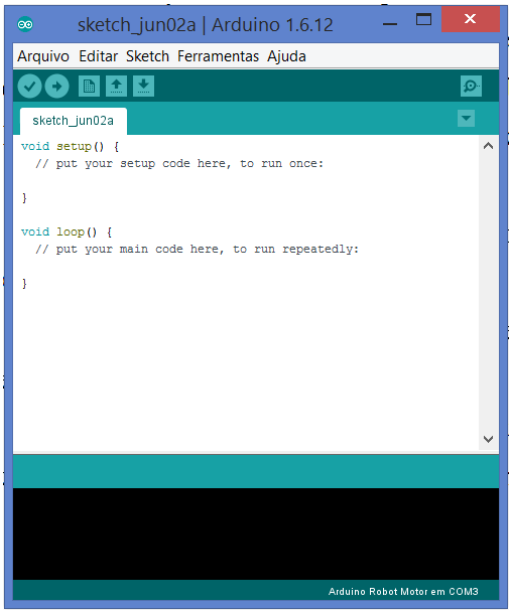


Figura 16 - Exemplo de janela de compilador para Arduino versão 1.6.12.
Fonte: Autor.

O compilador é disponibilizado no endereço https://www.arduino.cc/em /Main/Software, com diversas versões.

Como elemento de entrada de dados ao sistema, uma câmera USB é utilizada no projeto. A câmera é um acessório periférico que possibilita a aquisição de imagens para o processamento via software computacional. É uma *webcam* modelo C270, fabricante Logitech, USB 2.0, 60 fps (quadros por segundo), HD720P.

Figura 17 – Webcam Logitech C270.
Fonte: http://www.logitech.com/pt-br/product/hd-webcam-c270?crid=34 (17/08/2017).

É necessária a configuração de interfaces para a comunicação e correta interação entre componentes de hardware (Câmera, servo motores, *laser*, Arduino® e Computador) e software (MATLAB e compilador Arduino IDE). Uma interface é um dispositivo (material ou lógico) ao qual se efetuam trocas de informações entre dois sistemas diferentes, como a interface entre o computador e o Arduino® MEGA 2560, que prepara o microcontrolador para o correto entendimento de informações geradas pelas rotinas de programa do computador. As interfaces de software utilizadas no protótipo são as rotinas "motor_v1" e "arduino.m".

A rotina "motor_v1" configura o microcontrolador para que interprete informações do computador recebidas via porta serial, condicione e envie essas informações aos pinos de saída do Arduino como sinais elétricos de comando para os servo motores e para o módulo *laser.*

A rotina "arduino.m" adiciona funcionalidades ao Matlab® que permitem interação em tempo real com o microcontrolador Arduino® MEGA 2560 via porta serial do computador. A rotina "arduino.m" cria o objeto "arduino" no Workspace do Matlab®, possibilitando que se trabalhe com operações ligadas ao microcontrolador Arduino® MEGA 2560 em programas desenvolvidos no Matlab®, estabelece o protocolo de comunicação entre MATLAB e Arduino, configura as funções dos pinos do microcontrolador (saída PWM) e inicializa funções específicas de vetores e matrizes utilizadas no comando dos servo motores.

Para o bom curso do trabalho, a recomendação é compilar primeiro a rotina "motor_v1" no Arduino e depois executar a rotina "arduino.m" no MATLAB, nessa ordem.

A compilação da rotina "motor_v1" é realizada através do compilador "Arduino IDE" conforme os passos que seguem:

- Obter o compilador "Arduino IDE", disponível em https://www.arduino.cc/en/Guide/HomePage, selecionando o sistema operacional equivalente ao do computador que, neste trabalho é o Windows, selecionar a opção *Download the Arduino Software (IDE)* e executar a instalação do *software*.
- Conectar o Arduino ao computador com o cabo USB que atenda aos requisitos.

- Verificar se o modelo de microcontrolador Arduino configurado no compilador é compatível ao que se está utilizando. No caso deste trabalho, o modelo é o Arduino Mega 2560.

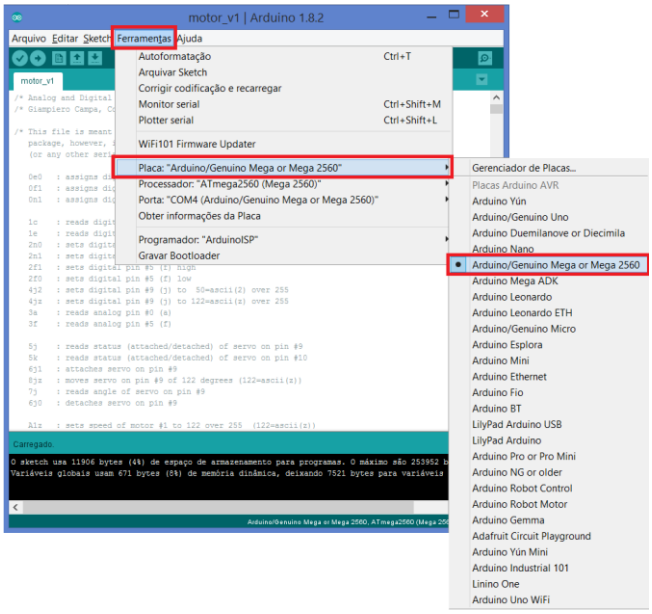


Figura 18 - Identificação do modelo de Arduino utilizado.
Fonte: Autor.

- Verificar qual a porta serial de comunicação está sendo utilizada pelo Arduino. No exemplo da FIG. 17, está selecionada a porta COM4.

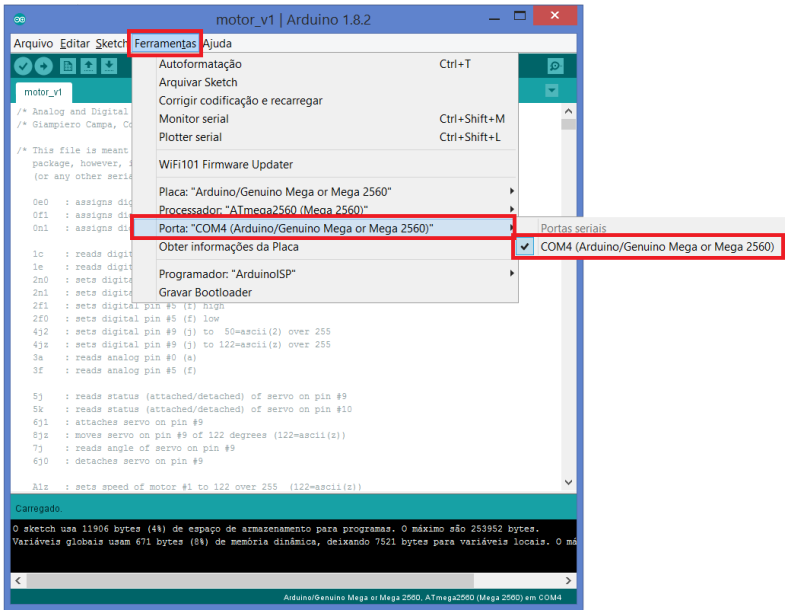


Figura 19 - Identificação da porta serial de comunicação utilizada pelo Arduino.
Fonte: Autor.

- Obter a rotina “motor_v1”, que está disponível na página da MathWorks, no endereço (www.mathworks.com/hardware-support/arduino-Matlab.html), na *toolbox “Arduino Support from MATLAB”*, onde é possível realizar o *download* do arquivo “arduinoio.mlpkginstall”. Após o *download*, instalar o arquivo e extrair a pasta de nome “ArduinoIO”, que possui o seguinte conteúdo:

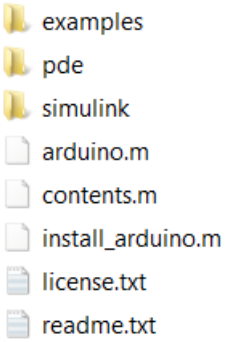


Figura 20 - Conteúdo da pasta "ArduinoIO".
Fonte: Autor.

Acessar a pasta “pde”, conforme segue:

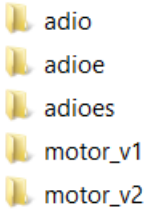


Figura 21 - Conteúdo da pasta "pde".
Fonte: Autor.

Acessar a pasta “motor_v1” e abrir o seguinte arquivo:

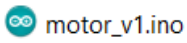


Figura 22 - Conteúdo da pasta "motor_v1".
Fonte: Autor.

O arquivo é a rotina “motor_v1”, na interface do *software* Arduino.

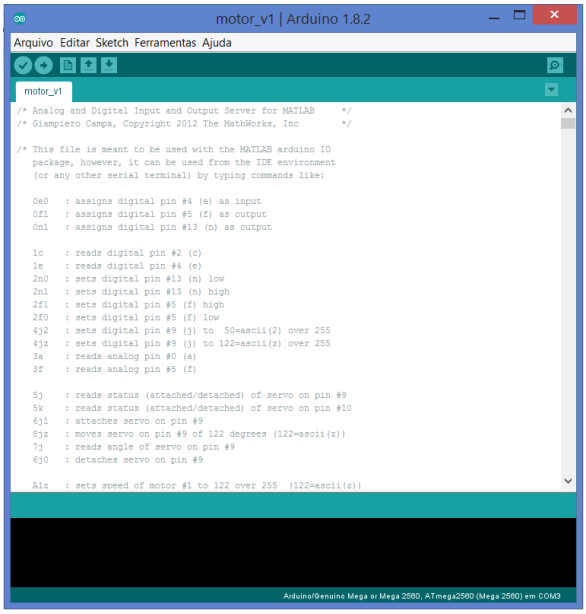


Figura 23 - Rotina "motor_v1" no compilador Arduino IDE.
Fonte: Autor.

- Verificar/Compilar a rotina “motor_v1” através do botão em destaque na FIG. 22, observando o *status* correto “Compilação terminada”.

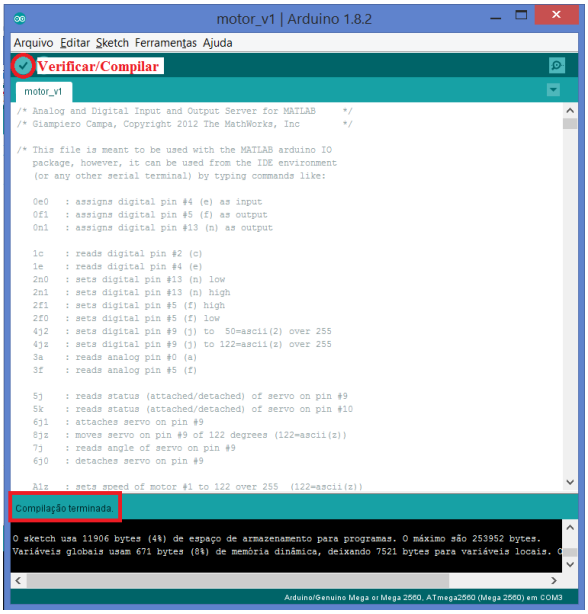


Figura 24 - Janela do Compilador com o botão "Verificar/Compilar" e status de "Compilação terminada".
Fonte: Autor.

- Enviar o programa compilado ao Arduino através do botão “Enviar”.

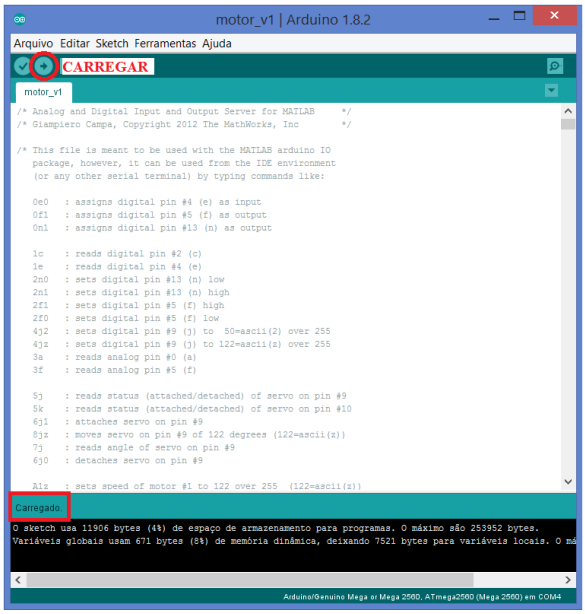


Figura 25 - Janela do Compilador com o botão "Carregar" e status de "Carregamento concluído".
Fonte: Autor.

Para os diversos sistemas operacionais existentes no mercado, é possível ocorrerem erros de compilação devido a versão de *software*. Neste trabalho, o *software* apresentou erro no processo de carregamento devido a incompatibilidade entre a versão do sistema operacional (*Windows 8*) e a versão do "Arduino IDE" (1.6.12). Foi solucionada através do *download* (https://www.arduino.cc/en/Main/Software) e instalação da versão 1.8.2 do *software* "Arduino IDE". A versão é apresentada na aba superior da janela.

Figura 26 - Identificação da versão do software "Arduino IDE".
Fonte: Autor.

Após as etapas de verificação/compilação e carregamento do programa, o microcontrolador Arduino está preparado para receber e interpretar comandos enviados pelo computador através da porta serial UART.

A próxima etapa é preparar a interface do MATLAB para comunicação eficiente com o microcontrolador Arduino. Para isso é utilizada a rotina "arduino.m", conforme as seguintes etapas:

- Obter a rotina “arduino.m”, que está disponível na pasta de nome “ArduinoIO” (mesma pasta que contém a rotina “motor_v1”):

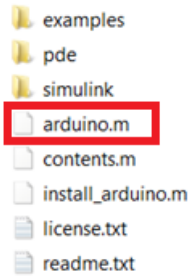


Figura 27 - Rotina "arduino.m" dentro da pasta "ArduinoIO".
Fonte: Autor.

- Abrir e executar a rotina “arduino.m” no MATLAB. Essa ação irá criar o objeto “arduino” no Workspace do MATLAB.

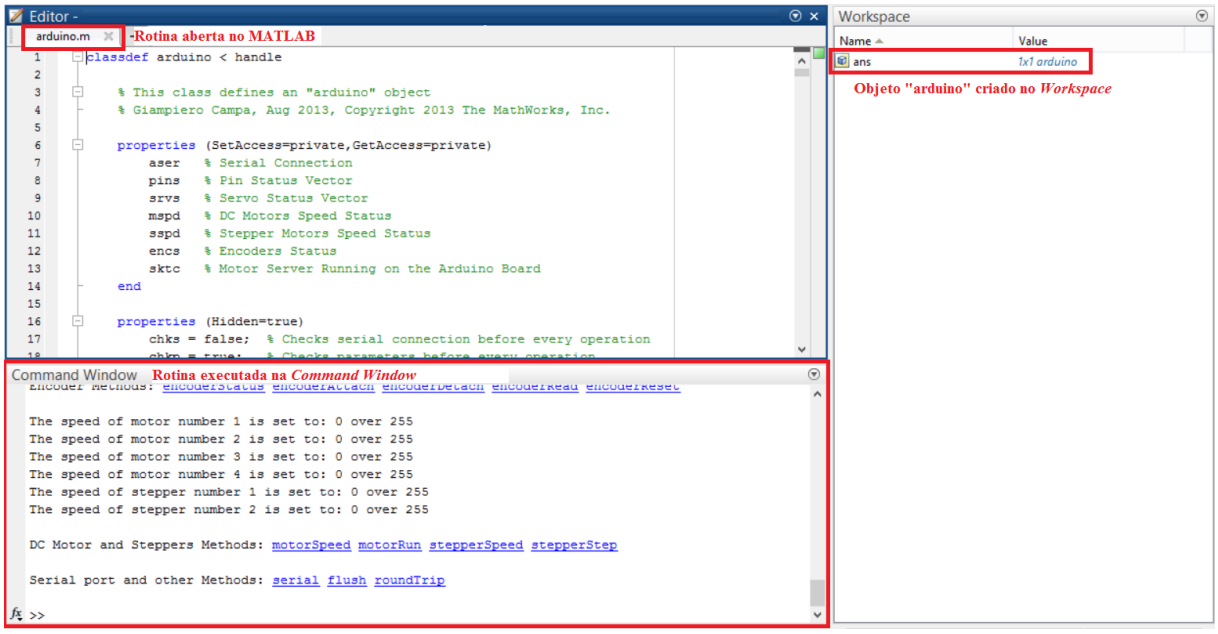


Figura 28 - Execução da rotina "arduino.m".
Fonte: Autor.

- Definir o objeto “arduino” em uma variável e indicar a porta serial do computador onde está conectado o microcontrolador Arduino. No trabalho, a variável “a” representa o objeto “arduino” para programação do MATLAB. Para identificar qual a porta serial está conectado o microcontrolador, basta abrir o compilador “Arduino IDE”, acessar o menu “Ferramentas → Porta:” e observar qual porta de comunicação está sendo utilizada:

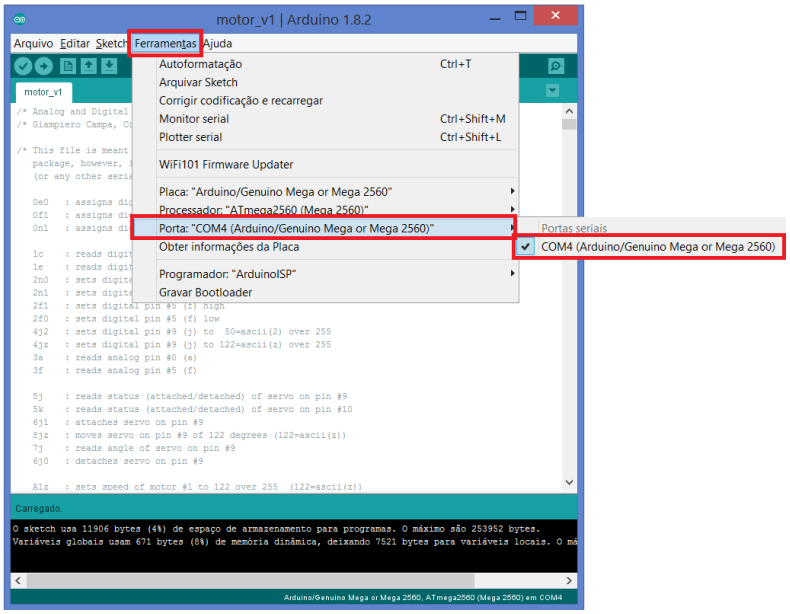


Figura 29 - Identificação da porta serial utilizada pelo Arduino.
Fonte: Autor.

A porta de comunicação em que o Arduino está conectado, conforme a FIG. 27 é a porta COM4. Executa-se o comando “a=arduino(‘COM4’)”, definindo a variável “a” como sendo o objeto “arduino” e estabelecendo comunicação entre MATLAB e microcontrolador através da porta serial COM4.

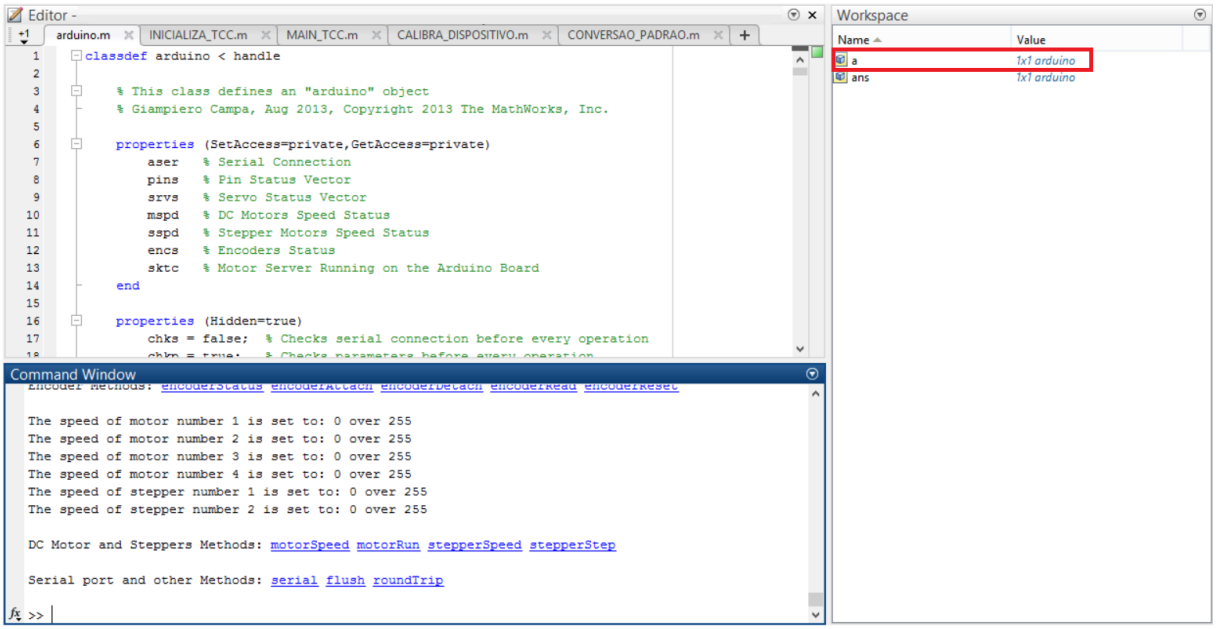


Figura 30 - Execução do comando “a=arduino(‘COM4’)” para definir variável “a” no *workspace* e comunicação.
Fonte: Autor.

Com as definições dos parâmetros de comunicação, o MATLAB e o Arduino estão preparados para interagir entre si.

A rotina “arduino.m” possui funções de sintaxe particular desenvolvidas com o objetivo de interação com o microcontrolador Arduino. A sintaxe das principais funções utilizadas neste trabalho está descrita a seguir, bem como os argumentos internos de entrada da função:

- servoAttach(a,pino) ou a.servoAttach(pino): define qual o pino do Arduino é utilizado como saída PWM para controle do servo motor. O argumento “a” da função representa o objeto “arduino” e o argumento “pino” é o número do pino do microcontrolador onde é conectado o servo motor. Exemplo: servoAttach(a,10) – define o pino 10 como saída PWM para servo motor.
- servoWrite(a,pino,valor) ou a.servoWrite(pino,valor): envia um valor inteiro de 0º a 180º ao respectivo pino de saída do Arduino onde o servo motor está conectado, fazendo-o movimentar até este valor . O argumento “a” da função representa o objeto “arduino”, o argumento “pino” é o número do pino do microcontrolador onde está conectado o servo motor e o argumento “valor” é o valor inteiro numérico do ângulo de movimento do servo motor. Exemplo: servoWrite(a,10,45) – posiciona o servo motor conectado ao pino 10 em 45º.
- analogWrite(a,pino,valor) ou a.analogWrite(pino,valor): escreve um valor PWM de intensidade de 0 a 255 no respectivo pino de saída. O argumento “a” da função representa o objeto “arduino”, o argumento “pino” é o número do pino do microcontrolador onde está conectado o servo motor e o argumento “valor” é o valor inteiro numérico de modulação PWM. Exemplo: analogWrite (a,5,78) – escreve o valor 78 de modulação PWM no pino 5 do Arduino.

Os pinos de saída analógica PWM disponíveis no microcontrolador Arduino MEGA 2560 utilizado no protótipo são os pinos 2 ao 13 e 44 ao 46, sendo estes pinos compatíveis com as funções “servoAttach”, “servoWrite” e “analogWrite”. Os demais pinos do microcontrolador não conseguem executar estas funções, pois, não são saídas analógicas PWM.

A próxima etapa do trabalho é definir os pinos de saída utilizados para o comando dos servo motores e comando do módulo *laser*. Conforme indicado anteriormente neste trabalho, os pinos do microcontrolador utilizados são:

- Pino 12: Servo motor para movimentos verticais (eixo y).
- Pino 08: Servo motor para movimentos horizontais (eixo x).
- Pino 07: Comando do módulo *laser*.

Para configuração das saídas de controle dos servo motores, as funções são estruturadas da seguinte maneira:

- “servoAttach(a,8)”- Pino 8 definido como saída PWM para servo motor.
- “servoAttach(a,12)” - Pino 12 definido como saída PWM para servo motor.

Para a configuração do pino de comando do módulo *laser*, temos:

- “analogWrite (a,7,100)” - Valor percentual de modulação PWM equivalente a 100 no módulo *laser* (pino 7).

É interessante o controle da intensidade da potência do ponto laser, pois em casos onde o objeto alvo seja de dimensões menores, a intensidade alta do laser pode causar instabilidade do sistema, prejudicando a operação do protótipo.

A preparação das interfaces software-hardware é uma etapa importante, visto que, sem uma interação consistente, não é possível haver operacionalidade satisfatória de sistema. Basicamente, o passo-a-passo da preparação das interfaces do sistema para utilização dos servo motores e do módulo *laser* segue o seguinte fluxo:

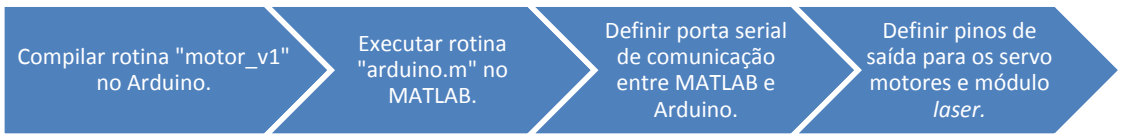


Figura 31 - Fluxo para configuração do sistema.
Fonte: Autor.

O MATLAB é um software versátil que permite diversas aplicações com interações internas e externas ao *software*, exigindo a manipulação de variáveis de entrada para gerar variáveis de saída de acordo com o objetivo proposto pelo sistema implementado.

Neste trabalho, a variável de entrada do sistema é a imagem capturada pela câmera e seus elementos de cena. A variável de saída é o par de coordenadas cartesianas que posicionam o marcador *laser* no centro do objeto de interesse na cena.

Para que seja possível inserir a imagem no sistema são necessários alguns procedimentos de inicialização e configuração da câmera no MATLAB. Para configuração e inicialização, são utilizadas as seguintes funções:

- webcamlist: reconhece quais as *webcams* conectadas e disponíveis ao MATLAB. Este comando gera na c*ommand window* do MATLAB uma lista com o nome das *webcams* conectadas e disponíveis ao *software*.

Figura 32 – Exemplo da função *webcamlist* do MATLAB.
Fonte: Autor.

- cam=webcam(‘Nome da webcam’): cria no workspace do MATLAB o objeto “cam” como sendo a webcam do argumento ‘Nome da webcam’ definido e mostra todas as propriedades de hardware da webcam.

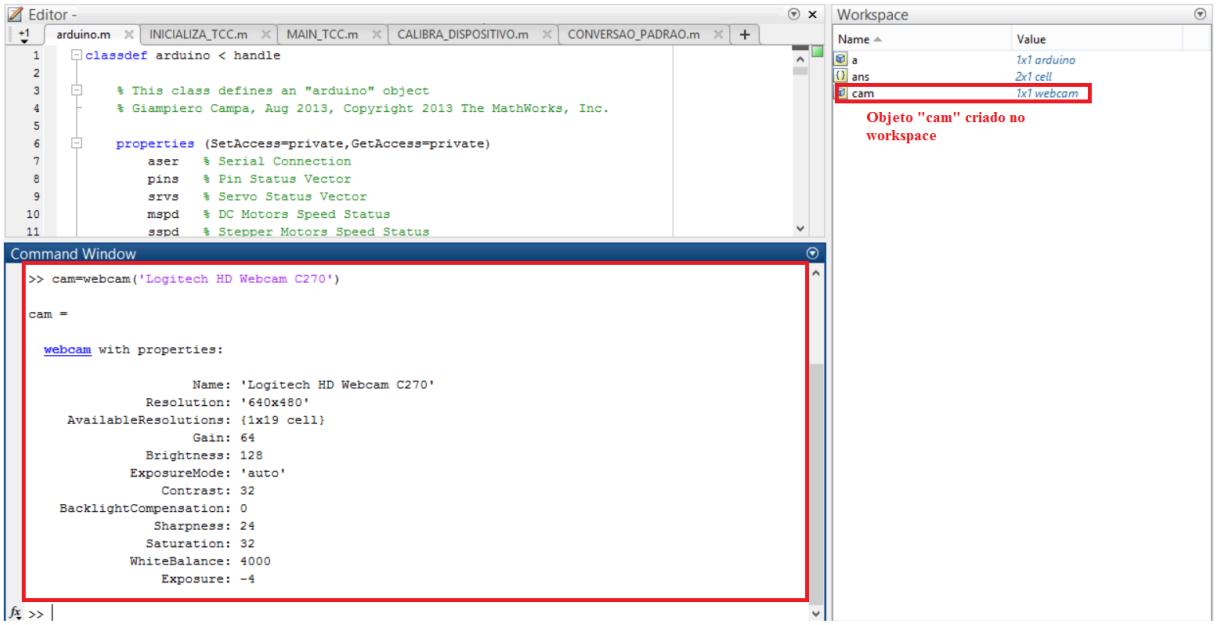


Figura 33 - Exemplo da função cam=webcam (‘Nome da *webcam*’) do MATLAB.
Fonte: Autor.

Conforme apresentado anteriormente, a *webcam* utilizada é modelo C270, fabricante *Logitech*, USB 2.0, 60 fps (quadros por segundo), HD720P. Com a *webcam* conectada a uma das portas USB disponíveis do computador, executar o comando

"webcamlist" no MATLAB para verificar se o *software* reconhece a *webcam* do protótipo. Como resultado esperado, a listagem deverá apresentar a *webcam* do robô com a descrição "*Logitech* HD *Webcam* C270" na *command window* do MATLAB. Após a certificação de que o MATLAB reconhece a *webcam*, o próximo passo é criar um objeto no *workspace* através da função definida com a sintaxe "cam=webcam('Logitech HD Webcam C270')".

Com a função "cam", tem-se as seguintes propriedades da *webcam* do protótipo:

Name: 'Logitech HD Webcam C270'
Resolution: '640x480'
AvailableResolutions: {1x19 cell}
Brightness: 128
WhiteBalance: 4000
Sharpness: 24
ExposureMode: 'auto'
Exposure: -4
Saturation: 32
Gain: 64
BacklightCompensation: 0
Contrast: 32

Figura 34 - Características da webcam do protótipo.
Fonte: Autor.

Existem comandos no MATLAB que permitem manipular inicialmente a imagem obtida pela câmera. São eles:

- "**Im = snapshot(cam);**" – este comando captura um quadro de imagem através da câmera selecionada e armazena na variável "Im". A variável "Im" é uma variável pré-determinada pelo usuário, sua nomenclatura ou identificação é aleatória, mas não deve coincidir com nenhuma outra em uso.
- "**imshow(Im)"** ou **"imshow(Im);**" – este comando apresenta o quadro de imagem capturado e armazenado na variável "Im".

Basicamente, a configuração e captura da imagem por uma câmera no software Matlab® segue o seguinte fluxo:

Figura 35 - Fluxo para entrada da imagem no MATLAB.
Fonte: Autor.

O *hardware* instalado no protótipo e que possui a função de marcar o objeto alvo dentro da imagem é o módulo *laser*. O módulo *laser* está instalado na parte superior da articulação *tilt-pan*, permitindo o movimento do laser sobre todo o quadro da imagem, cobrindo a área espacial de coordenadas (*x*, *y*) da cena em questão. A montagem do módulo é apresentada na figura a seguir:

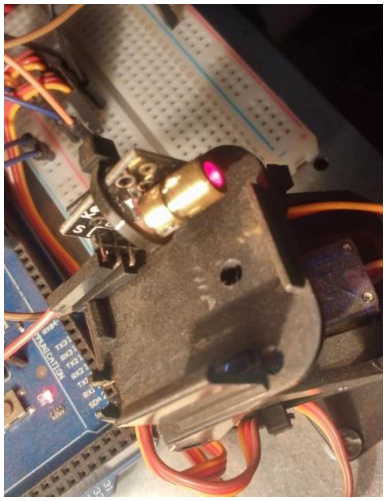


Figura 36 - Módulo *laser* instalado no dispositivo *tilt-pan*.
Fonte: Autor.

Conforme descrito anteriormente, o pino 07 é o pino do Arduino responsável pelo acionamento do módulo *laser* conforme a necessidade. O controle do acionamento do módulo é utilizado para evitar o superaquecimento do diodo *laser* quando em operação por grandes períodos de tempo, o que prejudica a luminosidade e a integridade física do componente. Esta estratégia evita a instalação de um sistema de resfriamento dedicado ao módulo *laser* no protótipo.

A montagem do *hardware* do protótipo está concebida satisfatoriamente neste ponto, onde a ideia é obter uma construção compacta e bem feita, que permita a observação e a realização de todos os testes necessários neste trabalho. A montagem consistente da parte de *hardware* é fundamental para que facilite e simplifique a detecção das causas raízes de possíveis limitações emergentes ao decorrer do trabalho.

Após a realização da parte de *hardware* e interfaces, inicia-se o trabalho com foco no software MATLAB. O ponto inicial no MATLAB é a criação de uma rotina para inicialização completa do sistema, partindo-se do ponto zero até o processamento da imagem na tela do computador.

Na inicialização são cumpridos os seguintes objetivos:

- Configuração da comunicação entre Arduino® MEGA 2560 e MATLAB;
- Definição da *webcam* utilizada;
- Configuração dos pinos do Arduino® MEGA 2560 que serão utilizados para comandar os servo motores e o módulo *laser*.

A rotina que agrega os objetivos da inicialização é denominada "INICIALIZA_TCC", e pode ser encontrada no Apêndice deste trabalho.

É necessário executar a calibração do aparato. O dispositivo possui servo motores que movimentam em um *range* de 0º a 180º na vertical e na horizontal, porém, as características construtivas do protótipo não permitem a utilização total deste *range* de movimentos, podendo o ponto *laser* ultrapassar os limites da imagem, não cumprindo a função deste trabalho. Como forma de contornar esse problema, é realizada a calibração dos pontos limites extremos de movimento dos servo motores de acordo com o *range* espacial da imagem, a fim de definir quais são as coordenadas correspondentes a determinados pontos da imagem. Os pontos calibrados no processo são:

- HC: Centro horizontal da imagem;
- VC: Centro vertical da imagem;
- LDH: Limite direito horizontal da imagem;
- LEH: Limite esquerdo horizontal da imagem;
- LSV: Limite superior vertical da imagem;
- LIV: Limite inferior vertical da imagem.

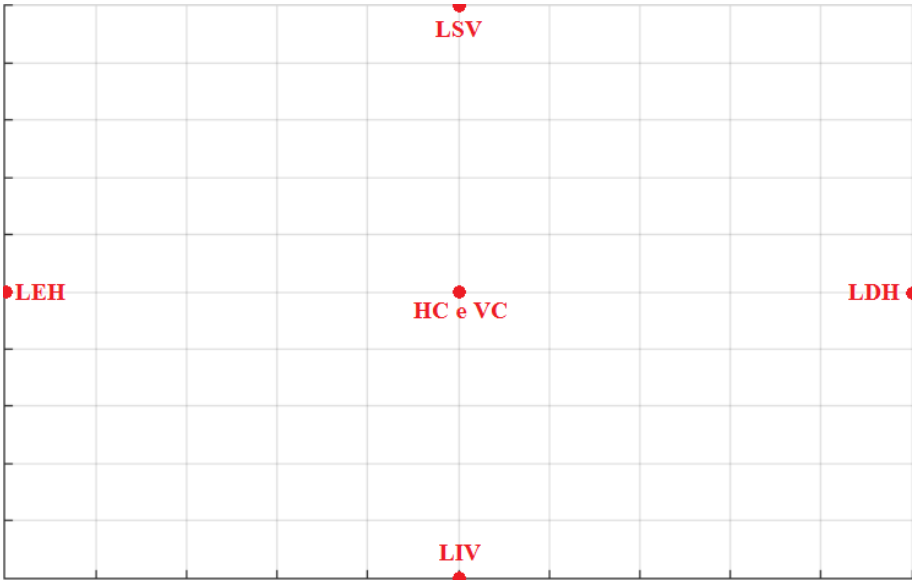


Figura 37 - Disposição dos pontos na imagem.
Fonte: Autor.

A rotina de programação denominada “CALIBRA_DISPOSITIVO” é a rotina que executa a função de calibração, que varre e mapeia os cinco pontos principais da imagem através de coordenadas cartesianas. A calibração dos pontos é feita manualmente, com o usuário inserindo valores numéricos no MATLAB, observando o posicionamento do *laser* em relação aos pontos de referência da imagem. A rotina de programação é feita para interagir com o usuário, a fim de criar um ambiente amigável para se cumprir o objetivo definido de calibração. A rotina “CALIBRA_DISPOSITIVO” pode ser encontrada no Apêndice deste trabalho.

Após a execução da rotina “CALIBRA_DISPOSITIVO”, os valores numéricos dos 05 pontos principais listados estarão armazenados nas respectivas variáveis HC, VC, LDH, LEH, LSV e LIV, que podem ser observadas no *workspace* do MATLAB.

Os valores sofrem limitações de acordo com a montagem e posicionamento da câmera e do robô em relação ao alvo, devido ao *range* de imagem. O *range* de imagem é o valor de comprimento dos pontos extremos da imagem, sendo o *range* horizontal obtido pela operação LDH – LEH, e o *range* vertical obtido pela operação LSV – LIV.

A calibração garante que o servo motor permaneça sempre dentro da imagem /(cena) capturada pela câmera.

Além da calibração, é necessário executar a conversão entre as escalas de *range* real (movimentação dos servo motores) e de *range* da imagem capturada pela câmera. A imagem gerada pela câmera neste trabalho tem largura 640 e altura 480 pixels. Os servo motores estão limitados a movimentação na largura LDH – LEH e na altura LSV – LIV.

O fluxo do programa para execução de uma ação do protótipo, segue:

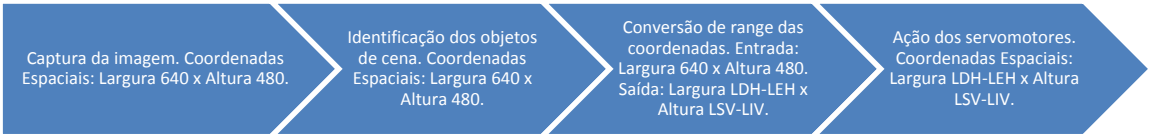


Figura 38 - Fluxo do programa para execução de uma ação.
Fonte: Autor.

Conforme indicado pelo fluxo do programa para execução da ação, o início do cálculo e todo o processamento da imagem é realizado utilizando-se as coordenadas espaciais da imagem (largura 640 x altura 480), mas, para execução de comando dos servo motores, é necessário converter para a escala calibrada inicialmente, representada pelas variáveis LDH, LEH, LSV e LIV. Para a conversão, é utilizado o seguinte cálculo:

$$\frac{(V_{E1}) - (Zero_{E1})}{(Span_{E1}) - (Zero_{E1})} = \frac{(V_{E2}) - (Zero_{E2})}{(Span_{E2}) - (Zero_{E2})} \quad (1)$$

Onde:

- V_{E1} Valor numérico na escala 1;
- V_{E2} Valor numérico na escala 2;
- $Zero_{E1}$ Valor mínimo do *range* da escala 1;
- $Zero_{E2}$ Valor mínimo do *range* da escala 2;
- $Span_{E1}$ Valor máximo do *range* da escala 1;
- $Span_{E2}$ Valor máximo do *range* da escala 2.

Contextualizando a equação à aplicação sobre as variáveis utilizadas pelo protótipo, será obtida a equação para a conversão entre as escalas teórica e real. É válido observar que a representação espacial de uma imagem possui característica específica, conforme Gonzalez e Woods apresentam, a origem da imagem está localizada na parte superior esquerda, com o eixo x crescente para baixo e o eixo y crescente para a direita (2010, p. 37). Seguindo estes parâmetros, temos:

Equação para conversão de escalas no eixo vertical:

$$S_Y = \left\{\frac{(\mathrm{LIV} - \mathrm{LSV}).(I_Y - 480)}{480}\right\} + LIV \quad (2)$$

Onde:

- I_Y Valor numérico da escala da imagem horizontal;
- S_Y Valor numérico da escala calibrada horizontal;
- LIV Limite inferior vertical da imagem;
- LSV Limite superior vertical da imagem.

Equação para conversão de escalas no eixo horizontal:

$$S_X = \{\frac{(\mathrm{LDH} - \mathrm{LEH}).[\mathrm{I_X}]}{640}\} + LEH \quad (3)$$

Onde:

- Y_1 Valor numérico da escala da imagem vertical;
- Y_R Valor numérico da escala calibrada vertical;
- LDH Limite direito horizontal da imagem;
- LEH Limite esquerdo horizontal da imagem.

A função desenvolvida para executar a conversão entre escala da imagem e escala calibrada é a função “CONVERSAO_PADRAO” e pode ser encontrada no Apêndice deste trabalho.

Com a conversão entre escalas da imagem e escala calibrada, a resolução do movimento do protótipo é reduzida, uma vez que se diminui o range do ângulo de movimentação. A resolução de movimento vertical é reduzida de 180/(LSV-LIV) vezes e a resolução de movimento horizontal é reduzida de 180/(LEH-LDH). A redução da resolução cria “vazios”, ou seja, limitações físicas no interior da imagem devido aos servo motores utilizados no trabalho se movimentarem apenas com valores inteiros de ângulo, impedindo o posicionamento do marcador laser em coordenadas de valores intermediários, o que impossibilita, em algumas situações, o protótipo de reduzir o erro de regime permanente a zero.

A FIG. 40 exemplifica as coordenadas fisicamente possíveis para posicionamento do marcador *laser* de acordo com a resolução obtida através da calibração dos *ranges* de movimento:

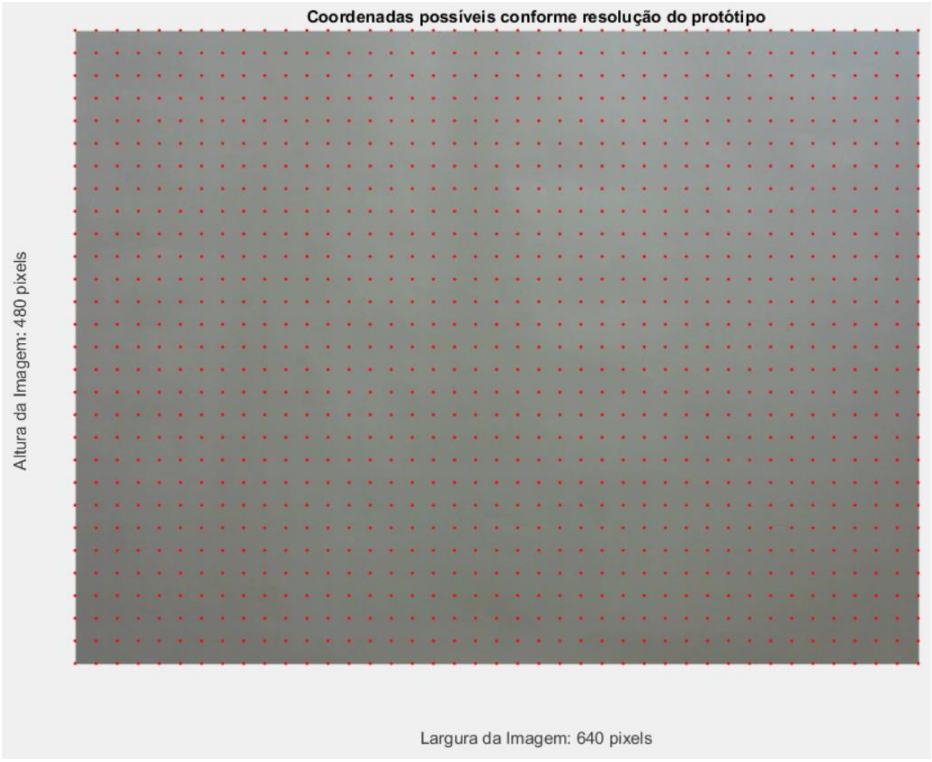


Figura 39 - Coordenadas possíveis ao robô.
Fonte: Autor.

Valores calibrados no dispositivo para resolução obtida na FIG. 40 acima:

- LEH: 102
- LDH: 62
- LSV: 28
- LIV: 0

É notável que, apesar de muitos pontos possíveis, existem lacunas onde não é possível se posicionar o marcador *laser*. São exatamente essas lacunas que podem contribuir para a não correção de um erro de regime permanente quando ocorrer de o centro do alvo estar situado na lacuna. Utilizando os valores calibrados, o erro de regime permanente máximo gerado decorrente da resolução do robô será de +-12 pixels (+-1,5%).

Para o processamento de imagens implementado no trabalho, optou-se por utilizar a função denominada "[B,L,Nb,A] = bwboundaries(f,'noholes')" para identificar o contorno do alvo. Essa função extrai da imagem binária as coordenadas do contorno externo dos objetos da cena, sendo aplicados em seguida algoritmos para encontrar o valor do centro do contorno. A função "[B,L,Nb,A] = bwboundaries(f,'noholes')" retorna as seguintes variáveis:

- B - representa o conjunto de arranjos matriciais para cada contorno identificado na imagem.
- L - é um arranjo bidimensional (matriz) dos inteiros não negativos das regiões contíguas. Inclui todos os elementos em uma única matriz L.

- Nb - é o valor de contornos, ou objetos, detectados na imagem processada.
- A - é uma matriz esparsa, onde suas linhas e colunas correspondem às posições das fronteiras dos contornos identificados pelo processamento e armazenados em B.

Através da função "[B,L,Nb,A] = bwboundaries(f,'noholes')" é possível identificar as posições do alvo e do marcador *laser*, adaptando uma rotina específica para cada objetivo.

A rotina para processamento do alvo é denominada "[Y1,X1]=ALVO(a,cam)" e a rotina para processamento do marcador *laser* é chamada de "MIRA" e podem ser encontradas no Apêndice deste trabalho.

Em conjunto com as rotinas "MIRA" e "ALVO" estão as funções "[YM,XM,Ro,theta] = Polar2(b)" e "[Y1,X1,Ro,theta] = Polar2M(b)". Essas funções são responsáveis por determinar as coordenadas polares do centro do contorno do objeto identificado, seja o objeto alvo ou o marcardor *laser*. Coordenada polar é uma forma de se exprimir matematicamente um valor com seu módulo e ângulo. As rotinas estão no Apêndice deste trabalho.

As funções "[YM,XM,Ro,theta] = Polar2(b)" e "[Y1,X1,Ro,theta] = Polar2M(b)" são bastante parecidas, porém, existem diferenças. O ângulo *theta* calculado pela função "[YM,XM,Ro,theta] = Polar2(b)" é incrementado de modo discreto, ou seja, é incrementado de 1 em 1 grau. Já a função "[Y1,X1,Ro,theta] = Polar2M(b)" tem o ângulo *theta* incrementado de modo analógico, onde o incremento é a menor resolução possível da imagem, que no caso deste trabalho é o pixel.

Agregando o ponto de vista de sistemas de controle ao processamento das informações extraídas da imagem, as coordenadas do centro do alvo representam o *set point*, as coordenadas da mira *laser* representam a variável de processo, o valor angular gerado pelo Arduino para movimentar os servo motores representa a variável manipulada e a diferença de distância entre o alvo e a mira *laser* representa o erro.

Com o objetivo de melhorar o funcionamento do protótipo, foram realizadas alterações no robô. Foi instalada outra articulação *tilt-pan* na *webcam*, visando comandar o posicionamento da câmera, o que possibilita expandir o *range* de movimentação do marcador *laser* na imagem. A articulação permite movimentos horizontais e verticais da *webcam*, variando de 0º a 180º. Os servo motores que operam os movimentos estão conectados ao pino 9 e 10 do microcontrolador. Assim sendo, os pinos de saída do Arduino são:

- Pino 07: Comando do módulo *laser*;
- Pino 08: Servo motor para movimentos horizontais do *laser* (eixo x);

- Pino 09: Servo motor para movimentos horizontais da *webcam* (eixo x);
- Pino 10: Servo motor para movimentos verticais da *webcam* (eixo y);
- Pino 12: Servo motor para movimentos verticais do *laser* (eixo y).

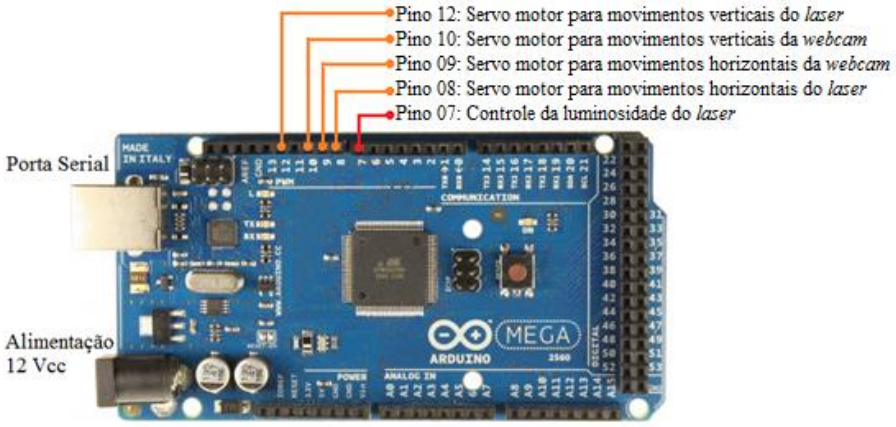


Figura 40 - Interligação modificada do Arduino.
Fonte: Autor.

Foi realizado ajuste mecânico nos servo motores da articulação de movimento do *laser* para que os pontos centrais de movimento horizontal e vertical da articulação coincidam com as coordenadas centrais dos *ranges* de movimento horizontal e vertical dos servo motores, simplificando a operação do robô e aumentando as possibilidades de movimentação vertical e horizontal.

A calibração realizada de forma manual foi substituída por uma rotina desenvolvida para calibrar o protótipo automaticamente, o que simplifica a operação do sistema. A operação da rotina automática primeiro posiciona os servo motores na coordenada de ângulo (90,90), que são os pontos centrais de movimento do laser. Em seguida, centraliza o *laser* horizontalmente e verticalmente na cena movimentando os servo motores da articulação da *webcam*, calcula os pontos extremos da imagem através da relação entre ângulo de movimentação do servo motor e pixels, chamado de passo ou resolução. O passo é dado em pixels/passo, ou seja, é o valor da variação de pixels a cada ângulo do servo motor movimentado, sendo que esse valor é diferente para movimentação horizontal e vertical. Após o cálculo da resolução do movimento dos servo motores, a rotina calcula os *ranges* horizontal e vertical, encerrando a calibração automática. A rotina de calibração automática é denominada “CALIBRA_DISPOSITIVO_AUTOMATICO” e pode ser encontrada no Apêndice deste trabalho.

A calibração automática permitiu executar a melhoria na forma de se converter as coordenadas espaciais da imagem para as coordenadas de movimento dos servo motores, de forma a tornar automatizado o processo de conversão, onde são substituídas constantes predeterminadas por variáveis obtidas através do processamento da imagem. Segue o trecho de programação referente à conversão de escalas:

```
%CONVERTE A POSIÇÃO DO ALVO PARA A ESCALA DO ROBÔ
MOVX=round(-(X1/RH)+LEH);
MOVY=round(-(Y1/RV)+LSV);
```

Figura 41 - Algoritmos para conversão das escalas da imagem e dos servo motores.
Fonte: Autor.

Onde:

- MOVX e MOVY: Coordenadas do alvo em ângulo (escala dos servo motores);
- X1 e Y1: Coordenadas do alvo em pixels (escala da imagem);
- LEH: Valor do limite esquerdo horizontal, em graus;
- LSV: Valor do limite superior vertical, em graus;
- RH: Valor da resolução (passo) horizontal, em pixels/passo;
- RV: Valor da resolução (passo) vertical, em pixels/passo.

Após o desenvolvimento das rotinas, o programa apresenta os resultados. Foi desenvolvida a rotina principal denominada “MAIN_TCC_2”, com o objetivo de agregar todas as rotinas e funções necessárias para o objetivo do robô. Pode ser encontrada no Apêndice deste trabalho.

Foram realizados testes para observar o comportamento do protótipo, além de permear adequações para melhores resultados. Os testes foram realizados sob as seguintes condições:

- Distância entre o robô e a parede onde o alvo se movimentava: 43 cm;
- Quantidade de alvos para processamento: 01 alvo;
- Formato do alvo: circular;
- Quantidade de amostragens: 100 amostras;
- Movimentação do alvo: sazonal.

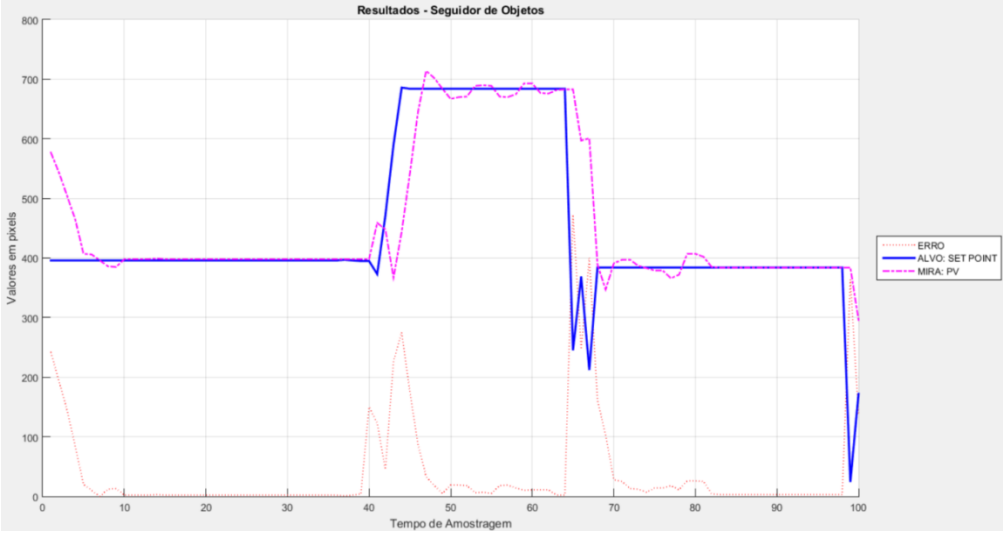


Figura 42 - Gráfico do resultado de 100 amostragens para o primeiro teste.
Fonte: Autor.

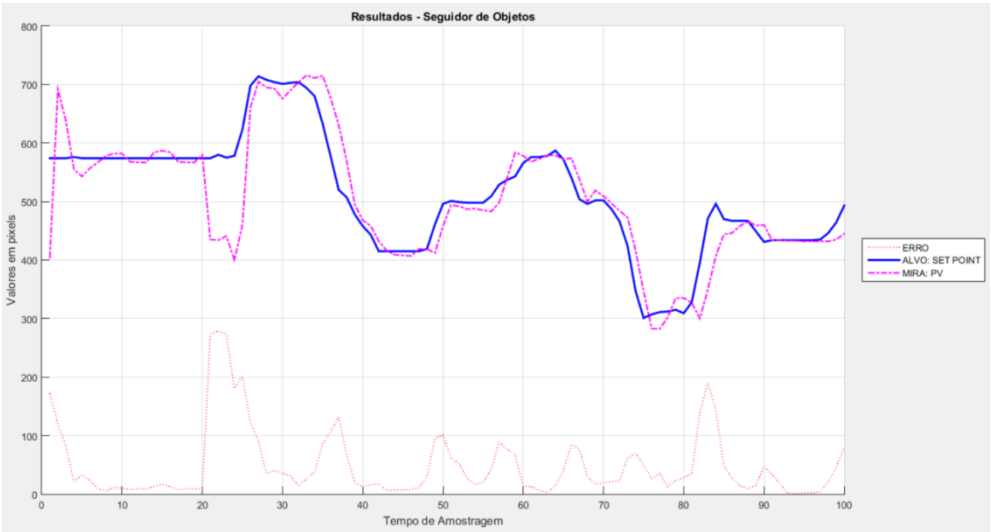


Figura 43 - Gráfico do resultado de 100 amostragens para o segundo teste.
Fonte: Autor.

Com olhar analítico sob o comportamento do protótipo no teste, foram observados pontos que podem melhorar a performance do robô. São eles:

- Estabilidade do protótipo: o protótipo detecta o alvo, o segue, porém existem instabilidades que causam variações como pode ser observado entre os instantes 50 e 60 no "Tempo de Amostragem" da FIG. 49.
- Rotina para o robô operar com alvos a distâncias maiores.

Após o teste realizado, foi observado que não há a necessidade de existir uma rotina de calibração, visto que a estratégia de controle incrementa ou decrementa o valor de ângulo dos servo motores baseado no valor do erro, independente de sua escala.

Prosseguindo em conformidade com os objetivos específicos, foi desenvolvida a rotina “[Y1,X1,OBJETO]=MULTI_ALVO(a,cam)” para identificar objetos múltiplos na tela, enumerando e apresentando para o usuário escolher qual objeto se transformará em um alvo a ser marcado e acompanhado pelo *laser*. A base para esta rotina é a função “[B,L,Nb,A] = bwboundaries(f,'noholes')”, onde é explorada a variável “Nb” extraída da imagem. A rotina pode ser encontrada no Apêndice deste trabalho.

Com o desenvolvimento dessa rotina, foi realizada alteração na rotina para detecção de alvos denominada “ALVO”. Foi necessário incluir a variável “OBJETO”, que contém o índice numérico do contorno determinado pelo usuário. A sintaxe passa a ser “[Y1,X1]=ALVO(OBJETO,a,cam)” e, o trecho alterado na função é o seguinte:

```
%IDENTIFICA O CONTORNO DA MIRA
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                          %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                         %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
MIRA)
b = B{1};                           %SELECIONA DO CONTORNO DA MIRA PARA
                                    %PROCESSAMENTO. DEVERÁ SER O ÚNICO
CONTORNO
```

Figura 44 - Trecho antigo da rotina "ALVO".
Fonte: Autor.

```
%IDENTIFICA O CONTORNO DOS OBJETOS
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                          %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                         %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
ALVO)
b = B{OBJETO};                      %SELECIONA QUAL CONTORNO PARA
PROCESSAMENTO
```

Figura 45 - Trecho alterado da rotina “[Y1,X1]=ALVO(OBJETO,a,cam)”.
Fonte: Autor.

A rotina “[Y1,X1,OBJETO]=MULTI_ALVO(a,cam)” retorna a imagem com os alvos enumerados, e aguarda a escolha pelo usuário, conforme mostra a FIG. 54:

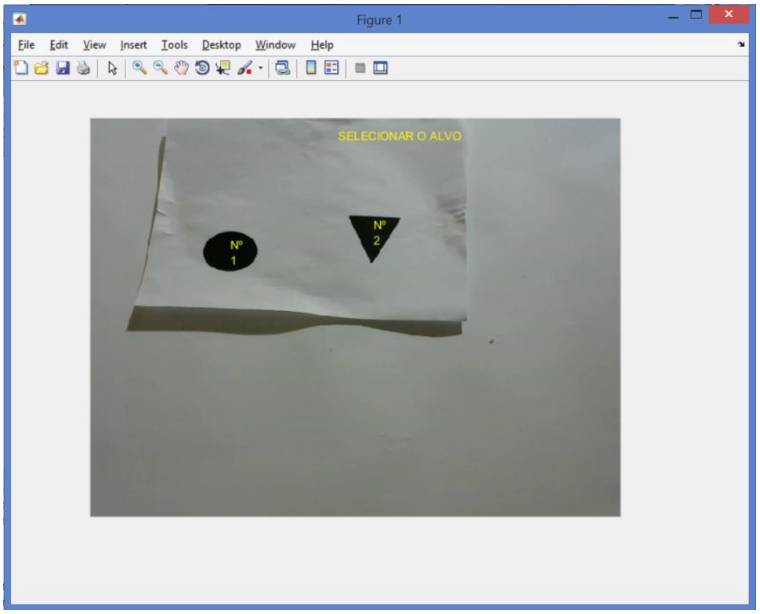


Figura 46 - Apresentação de objetos identificados pela função “[Y1,X1,OBJETO]=MULTI_ALVO(a,cam)” aguardando escolha do alvo pelo usuário.
Fonte: Autor.

Após a escolha realizada pelo usuário, é definido o alvo e são identificadas as coordenadas do centro, demarcando-o com um “X”. A FIG. 55 apresenta o alvo selecionado com o centro demarcado.

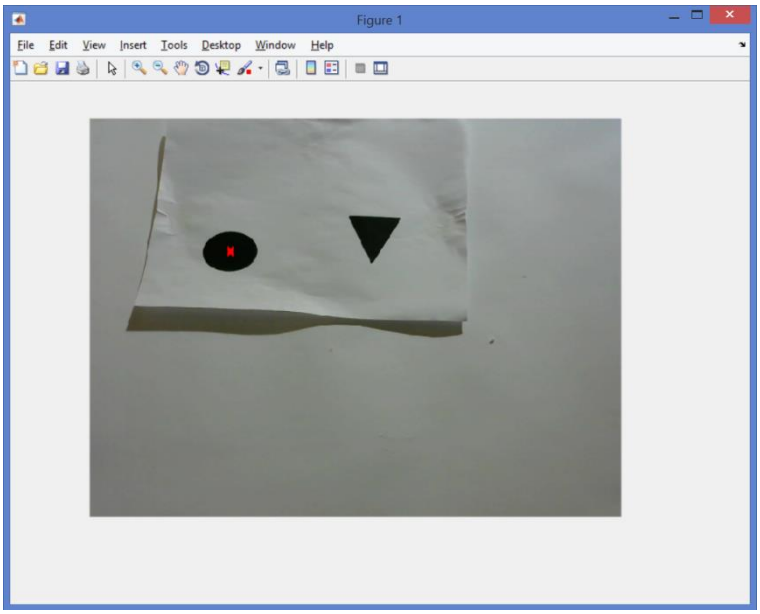


Figura 47 - Objeto selecionado como alvo e identificado com marcador em “X” no centro.
Fonte: Autor.

Após a rotina para identificação de múltiplos alvos, o programa principal prossegue mantendo sempre identificando automaticamente como alvo o objeto selecionado. Esta identificação é eficiente para qualquer tipo de contorno e apenas para movimentos de translação do alvo. Para movimentos que provocam transformações de escala, rotação, cisalhamento vertical e cisalhamento horizontal no alvo, o protótipo não consegue identificar qual objeto de cena é o alvo, perdendo a referência para posicionar o marcador laser. A FIG. 56 demonstra as transformações geométricas possíveis para os objetos neste trabalho:

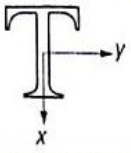
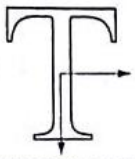
Nome da transformação	Matriz afim, T	Equações coordenadas	Exemplo
Identidade	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
Escala	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotação	$\begin{bmatrix} \cos\theta & \text{sen}\,\theta & 0 \\ -\text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\,\text{sen}\,\theta$ $y = v\,\text{sen}\,\theta + w\cos\theta$	
Translação	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Cisalhamento (vertical)	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v + s_v w$ $y = w$	
Cisalhamento (horizontal)	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

Figura 48 - Transformações geométricas.
Fonte: GONZALEZ, WOODS, 2010, p. 57.

Para solução deste problema, foram estudadas e testadas algumas alternativas. A primeira alternativa é a técnica de assinatura de imagens.

De acordo com Solomon e Breckon, a assinatura de uma imagem é a representação de um contorno/fronteira 2D como uma função 1D onde tipicamente é representada pelas coordenadas do centroide do contorno, da distância “Ro” entre o centroide e o contorno em função do ângulo “theta”. Para um contorno fechado, a assinatura é uma função periódica que se repete em uma escala angular de 2∏ (360°) (2013).

Neste trabalho, a assinatura do contorno do objeto pode ser obtida através da função “[Y1,X1,Ro,theta] = Polar2(b)”, onde os argumentos de saída “Ro” e “theta” são o conjunto de valores da distância e do ângulo respectivamente, em relação ao par de coordenadas “X1” e “Y1”, representando a assinatura do contorno de entrada “b”.

Os testes foram realizados aplicando-se o cálculo de assinaturas, porém não houve êxito na aplicação da técnica, pois, apesar da assinatura de um determinado objeto possuir valores com características individuais, os valores de “Ro” e “theta” podem variar em função da transformação geométrica sofrida pelo alvo. Cada transformação geométrica diferente da translação altera a fase da assinatura, dificultando a identificação do objeto por este parâmetro. A FIG. 57 apresenta a defasagem provocada na assinatura quando o objeto sofre a transformação geométrica de rotação.

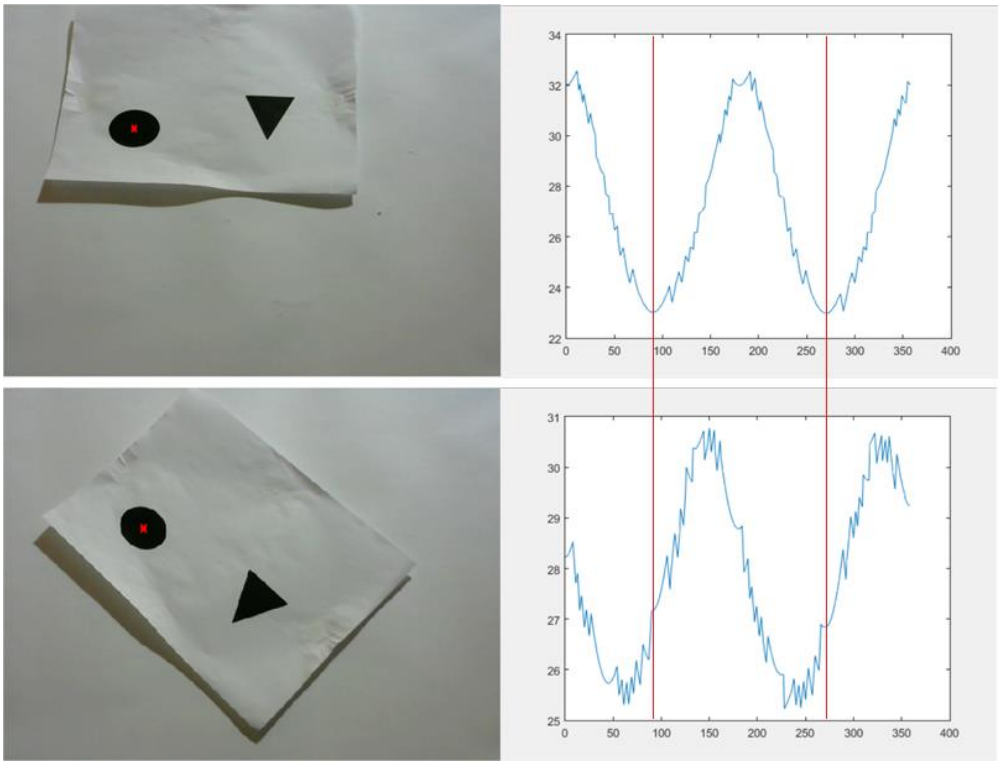


Figura 49 - Diferença entre fases da função assinatura de um objeto causada pela rotação.
Fonte: Autor.

Com tal defasagem, a técnica de assinatura se tornou uma alternativa inviável para este momento, pois é um trabalho extenso para desenvolver a aplicação e adequá-la ao robô.

Outra possibilidade para aplicação é a técnica de momentos invariantes, ou momentos de Hu.

Segundo Solomon e Breckon, momentos invariantes são momentos estatísticos que são utilizados como descritores de regiões, baseado na função densidade de probabilidade (2013, p. 217). A função densidade de probabilidade é uma função que descreve a probabilidade relativa de uma variável assumir um valor em uma faixa particular (WIKIPÉDIA, 2017).

De acordo com Solomon e Breckon ((2013), para a obtenção dos sete valores referidos como momentos invariantes em relação à translação, mudança de escala e rotação, é necessário se conhecer o momento de ordem zero e os momentos centrais, onde, através do conhecimento de todos os momentos é possível determinar a função densidade sem ambiguidade, assim, os momentos codificam a informação sobre a forma da função densidade. O *n*-ésimo momento pode ser calculado através da equação:

$$m_n = \int_{-\infty}^{\infty} x^n \, p(x) dx \tag{4}$$

Onde:

- m_n Momento da função densidade de probabilidade para a variável aleatória;
- x^n Variável aleatória;
- $p(x)$ Função densidade de probabilidade.

A variação em torno da média é definida como os momentos centrais da função densidade, e podem ser calculador pela equação:

$$M_n = \int_{-\infty}^{\infty} (x - \bar{x})^n \, p(x) dx \tag{5}$$

Onde:

- M_n Momento central da função densidade de probabilidade para a variável aleatória;
- x Variável aleatória;
- $p(x)$ Função densidade de probabilidade;
- $\bar{x}$ Valor médio para a variável aleatória.

Para calcular momentos em imagens, a função densidade é substituída pela imagem de intensidade e coordenadas I(x,y), o que transforma a variável aleatória em uma

intensidade provável e, a imagem em uma faixa particular. Como a imagem é um conjunto de coordenadas em um formato de duas dimensões, ou seja, 2-D, é possível, analogamente reescrever a equação de cálculo não normalizado do momento central para a imagem:

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q \, I(x, y) dx dy \tag{6}$$

Onde:

- M_{pq} Momento central da imagem I(x,y) para as coordenadas da imagem;
- $\bar{x}$ Valor médio para a coordenada x da imagem I(x,y);
- $\bar{y}$ Valor médio para a coordenada y da imagem I(x,y);
- $I(x, y)$ Função intensidade da imagem.

É necessário normalizar o momento central para que o mesmo seja invariante quanto à rotação e mudança de escala. Para tal normalização, é calculada a razão entre o momento central e o momento de ordem zero (onde p=0 e q=0), obtendo a relação de distância entre o momento central e o momento de ordem zero. Segue a equação:

$$\eta_{pq} = \frac{M_{pq}}{M_{00}^{\beta}} \; em \; que \; \beta = \frac{p + q}{2} + 1 \; e \; p + q \geq 2 \tag{7}$$

Onde:

- M_{pq} Momento central da imagem I(x,y) para as coordenadas da imagem;
- M_{00}^{β} Momento de ordem zero para a forma 2-D;
- β Valor médio para a coordenada y da imagem I(x,y);
- η_{pq} Função intensidade da imagem.

A partir da norma do momento central, é possível obter os sete momentos invariantes:

$$\emptyset_1 = \eta_{20} + \eta_{02} \tag{8}$$

$$\emptyset_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{9}$$

$$\emptyset_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{10}$$

$$\emptyset_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{30})^2 \tag{11}$$

$$\emptyset_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} - \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{03} + \eta_{21}) \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] \quad (12)$$

$$\emptyset_6 = (\eta_{20} - \eta_{02})[(\eta_{12} + \eta_{30})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{21} + \eta_{03})(\eta_{12} + \eta_{30}) \quad (13)$$

$$\emptyset_7 = 3(\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2] + (3\eta_{21} - \eta_{30})(\eta_{21} + \eta_{03}) \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] \quad (14)$$

Os momentos $\emptyset_1$, $\emptyset_2$, $\emptyset_3$, $\emptyset_4$, $\emptyset_5$, $\emptyset_6$ e $\emptyset_7$ são invariantes em relação a mudança de escala, translação e rotação (SOLOMON e BRECKON, 2013).

Para o cálculo dos momentos invariantes no Matlab, são utilizadas as funções "MomentosInvariantes" e "invmoments". A rotina "invmoments" calcula os sete momentos invariantes conforme as respectivas equações. A rotina "invmoments" pode ser encontrada no Apêndice deste trabalho.

A rotina "MomentosInvariantes" retorna os valores absolutos de cada elemento calculado pela rotina "invmoments", para se trabalhar apenas com a parte real dos momentos invariantes. A rotina "MomentosInvariantes" pode ser encontrada no Apêndice deste trabalho.

Foram realizadas experimentações para testar a função e buscar o resultado que melhor se aplique ao trabalho. Os testes foram realizados utilizando o triângulo e o círculo apresentados na FIG. 54. Os testes e experimentos realizados são apresentados a seguir:

Teste 01 – Foi aplicada a rotina de momentos invariantes aos vetores de contorno de cada objeto da cena. Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo os resultados de momentos invariantes para cada posição conforme TAB. 1.

Tabela 1 - Valores de momentos invariantes obtidos através do teste 01.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	1,4253	2,8508	6,6306	6,6309	13,2617	8,0564	17,5438
	Triângulo	1,5539	3,1079	7,1263	7,1265	14,2529	8,6805	18,6753
Posição 02	Círculo	1,4104	2,8209	6,4540	6,4545	12,9087	7,8649	17,2270
	Triângulo	1,5527	3,1056	7,3166	7,3168	14,6334	8,8696	19,3207
Posição 03	Círculo	1,4299	2,8598	6,5636	6,5643	13,1282	7,9942	17,5359
	Triângulo	1,4699	2,9399	8,4824	8,4828	16,9654	9,9528	20,9404
Posição 04	Círculo	1,4866	2,9732	6,9463	6,9469	13,8936	8,4335	18,1602
	Triângulo	1,3305	2,6612	5,9333	5,9334	11,8668	7,2640	16,6301

Fonte: Autor.

Analiticamente, os resultados do teste não são satisfatórios ao trabalho, pois não possibilitam aplicação de algoritmos para comparação de valores, visto que, tanto para o objeto círculo quanto para o triângulo, os valores de momento invariante são muito próximos e não obedecem padrões que permitam distinguir as formas, como por exemplo, os valores obtidos na posição 04 difere, em grandeza, das posições 01, 02 e 03, possuindo valores de momento invariante do círculo maiores do que os valores do triângulo, enquanto que nas outras posições acontece o contrário. Outro ponto observado é a grande variação dos momentos conforme muda a posição dos objetos na cena, o que não satisfaz o objetivo da técnica. Os resultados observados não tornaram consistente esta forma de aplicação dos momentos invariantes no protótipo.

Teste 02 – Foi aplicada a rotina de momentos invariantes aos valores de “Ro” do contorno de cada objeto da cena. Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo os resultados de momentos invariantes para cada posição conforme TAB. 2.

Tabela 2 - Valores de momentos invariantes obtidos através do teste 02.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	0,3085	0,6171	4,7391	4,7391	9,4783	5,0477	-
	Triângulo	0,3008	0,6016	3,7611	3,7611	7,5222	4,0619	-
Posição 02	Círculo	0,3200	0,6399	4,9498	4,9498	9,8997	5,2698	-27,7754
	Triângulo	0,2608	0,5215	3,9781	3,9781	7,9562	4,2389	-
Posição 03	Círculo	0,2680	0,6581	7,1812	7,1812	14,3625	7,5103	-
	Triângulo	0,2680	0,5360	3,9106	3,9106	7,8211	4,1785	26,3122
Posição 04	Círculo	0,3288	0,6575	4,4278	4,4278	8,8556	4,7566	27,0324
	Triângulo	0,2597	0,5195	3,9217	3,9217	7,8434	4,1814	-26,2505

Fonte: Autor.

Os valores obtidos no teste mostram que houve valores aleatórios entre os momentos invariantes, ou seja, em algumas situações as diferenças são pequenas entre círculo e triângulo, como por exemplo $\emptyset_1$ na posição 01. Em outras não há diferença entre valores de momento invariante para círculo e triângulo, como por exemplo $\emptyset_1$ na posição 03, e em algumas situações há grandes diferenças, como por exemplo $\emptyset_5$ na posição 03, o que não permite a identificação de valores padrão de momento invariante para círculo e para triângulo, tornando os resultados não satisfatórios para o trabalho. Além da não identificação de padrões, houve situações onde não foi possível o cálculo do momento invariante devido aos valores de entrada serem insuficientes. Os resultados deste teste são inconsistentes para o trabalho.

Teste 03 – Foi aplicada a rotina de momentos invariantes aos valores de “theta” e “Ro” concatenados horizontalmente. A função utilizada foi “cat(2,A,B)”, que organiza as matrizes da seguinte forma:

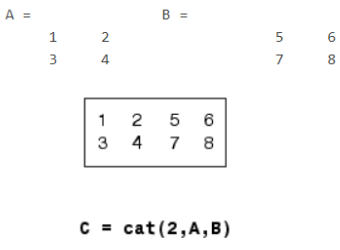


Figura 50 - Representação da função "cat(2,A,B,)".
Fonte: <https://www.mathworks.com/help/Matlab/ref/cat.html>.

Os argumentos de entrada A e B foram substituídos por “theta” e “Ro”, respectivamente. Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo resultados de momentos invariantes para cada posição conforme TAB. 3.

Tabela 3 - Valores de momentos invariantes obtidos através do teste 03.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	1,2864	2,5729	4,4076	4,4075	8,8150	5,6939	-14,0567
	Triângulo	1,2352	2,4706	4,1764	4,1763	8,3527	5,4116	-13,7251
Posição 02	Círculo	1,2830	2,5661	4,3624	4,3622	8,7245	5,6453	-13,9754
	Triângulo	1,2442	2,4885	4,1776	4,1774	8,3549	5,4217	-13,7022
Posição 03	Círculo	1,2963	2,5927	4,4291	4,4289	8,8579	5,7253	-14,0632
	Triângulo	1,2622	2,5244	4,2741	4,2740	8,5480	5,5362	-13,9015
Posição 04	Círculo	1,2749	2,5500	4,3358	4,3357	8,6714	5,6106	-13,9566
	Triângulo	1,2818	2,5637	4,4506	4,4504	8,9009	5,7323	-14,0961
Posição 05	Círculo	1,2748	2,5498	4,3613	4,3611	8,7223	5,6360	-14,0054
	Triângulo	1,2331	2,4663	4,1363	4,1361	8,2723	5,3693	-13,6217

Fonte: Autor.

Os resultados para o teste não são satisfatórios para aplicação neste trabalho, pois os valores de momento invariante para cada forma, círculo ou triângulo, são muito próximos, e a diferença entre valores de uma posição para outra do mesmo objeto se confunde com a diferença de valores entre objetos, ou seja, não há como se definir com exatidão se houve uma mudança de posição ou se são dois objetos diferentes. Os resultados deste teste são inconsistentes para este trabalho.

Teste 04 – Foi aplicada a rotina de momentos invariantes aos valores de “theta” e “Ro”, concatenados horizontalmente e acrescentando moldura nas laterais da matriz resultante da concatenação dos vetores “theta” e “Ro”. A função utilizada para concatenação foi “cat(2,theta,Ro)”, e a função “padarray(MI,[200 200],'both')” para inserir moldura de 200 x 200 elementos nas laterais. O exemplo do resultado da função “padarray” é o seguinte:

Figura 51 - Moldura (em preto) inserida na figura através da função "padarray".
Fonte: <https://www.mathworks.com/help/images/ref/padarray.html?s_tid=doc_ta>.

Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo resultados de momentos invariantes para cada posição conforme TAB. 4.

Tabela 4 - Valores de momentos invariantes obtidos através do teste 04.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	1,2736	2,5474	4,3625	4,3624	8,7248	5,6360	-14,0135
	Triângulo	1,2431	2,4862	4,1820	4,1819	8,3638	5,4250	-13,6937
Posição 02	Círculo	1,2983	2,5967	4,4314	4,4313	8,8626	5,7296	-14,0615
	Triângulo	1,2600	2,5201	4,2780	4,2779	8,5559	5,5379	-13,9160
Posição 03	Círculo	1,2868	2,5736	4,3817	4,3815	8,7631	5,6683	-14,0097
	Triângulo	1,2575	2,5152	4,2216	4,2215	8,4430	5,4791	-13,7507
Posição 04	Círculo	1,3061	2,6123	4,4658	4,4656	8,9314	5,7718	-14,1130
	Triângulo	1,2548	2,5096	4,2987	4,2985	8,5972	5,5534	-13,8913
Posição 05	Círculo	1,2744	2,5489	4,3524	4,3523	8,7046	5,6267	-13,9871
	Triângulo	1,2398	2,4797	4,1836	4,1835	8,3670	5,4233	-13,7049

Fonte: Autor.

Os resultados para o teste não são satisfatórios para aplicação neste trabalho, pois os valores de momento invariante para cada forma, círculo ou triângulo, são muito próximos, e a diferença entre valores de uma posição para outra do mesmo objeto se confunde com a diferença de valores entre objetos, ou seja, não há como se definir com exatidão se houve uma mudança de posição ou se são dois objetos diferentes. Os resultados deste teste são inconsistentes para este trabalho.

Teste 05 – Foi aplicada a rotina de momentos invariantes ao vetor de cada contorno, completando os vetores até 200 elementos verticais por método de concatenação vertical. A função utilizada para concatenação foi “cat(1,b,zeros(200 - size(b,1),2))”, onde, o

argumento de entrada “b” representa o contorno do objeto identificado em cena, e o argumento “zeros(200 - size(b,1),2)” representa um vetor de 200 linhas e 2 colunas, contendo os valores do contorno do objeto “b” e, caso o contorno “b” não possua 200 linhas, o vetor é complementado com zeros até a linha 200. O resultado da função “cat(1,A,B)” aplicada é:

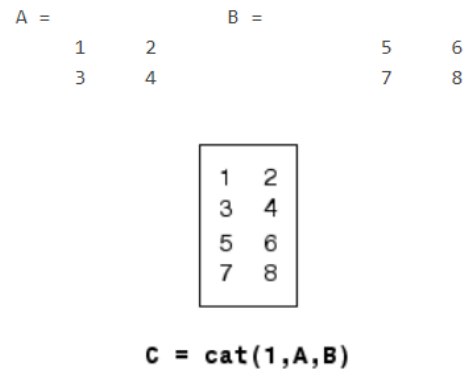


Figura 52 - Representação da função "cat(1,A,B,)".
Fonte: < https://www.mathworks.com/help/Matlab/ref/cat.html?s_tid=doc_ta>.

Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo resultados de momentos invariantes para cada posição conforme TAB. 5.

Tabela 5 - Valores de momentos invariantes obtidos através do teste 05.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	1,4673	2,9348	6,8873	6,8875	13,7749	8,3549	18,4347
	Triângulo	1,5526	3,1053	7,7315	7,7320	15,4637	9,2846	19,5883
Posição 02	Círculo	1,5983	3,1968	7,3927	7,3928	14,7856	8,9912	19,7482
	Triângulo	1,4206	2,8413	7,8225	7,8238	15,6469	9,2445	19,5546
Posição 03	Círculo	1,5808	3,1618	7,5412	7,5414	15,0827	9,1223	19,4055
	Triângulo	1,3403	2,6808	6,2502	6,2504	12,5008	7,5909	17,0346
Posição 04	Círculo	1,4965	2,9930	6,9378	6,9382	13,8762	8,4348	18,1048
	Triângulo	1,4174	2,8350	6,3626	6,3627	12,7254	7,7802	-18,4779
Posição 05	Círculo	1,4293	2,8588	6,5406	6,5410	13,0818	7,9704	17,3820
	Triângulo	1,5489	3,0979	7,3261	7,3263	14,6525	8,8753	19,4967

Fonte: Autor.

Os resultados para o teste não são satisfatórios para aplicação neste trabalho, pois os valores de momento invariante para cada forma, círculo ou triângulo, são muito próximos,

e a diferença entre valores de uma posição para outra do mesmo objeto se confunde com a diferença de valores entre objetos, ou seja, não há como se definir com exatidão se houve uma mudança de posição ou se são dois objetos diferentes. Além disso, os valores resultantes no teste não obedecem padrões que permitam distinguir as formas existentes na cena, tornando a alternativa experimentada inconcludente, assim sendo, os resultados deste teste são inconsistentes para este trabalho.

Teste 06 – Foi aplicada a rotina de momentos invariantes ao vetor de cada contorno, completando os vetores até 480 elementos verticais por método de concatenação vertical. A função utilizada para concatenação foi "cat(1,b,zeros(480 - size(b,1),2))", onde, o argumento de entrada "b" representa o contorno do objeto identificado em cena, e o argumento "zeros(480 - size(b,1),2)" representa um vetor de 480 linhas e 2 colunas, contendo os valores do contorno do objeto "b" e, caso o contorno "b" não possua 480 linhas, o vetor é complementado com zeros até a linha 480. Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo os resultados de momentos invariantes para cada posição conforme TAB. 6.

Tabela 6 - Valores de momentos invariantes obtidos através do teste 06.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	1,4697	2,9397	6,8784	6,8787	13,7572	8,3485	18,1933
	Triângulo	1,5643	3,1286	8,1578	8,1586	16,3168	9,7230	20,2993
Posição 02	Círculo	1,4198	2,8397	6,5151	6,5156	13,0310	7,9355	17,3415
	Triângulo	1,5442	3,0886	7,2884	7,2886	14,5771	8,8329	19,3279
Posição 03	Círculo	1,4967	2,9936	6,9007	6,9012	13,8021	8,3980	18,0330
	Triângulo	1,5222	3,0445	8,3000	8,3002	16,6002	9,8224	-21,9558

Fonte: Autor.

Apesar dos valores numéricos do teste obedecerem em sua maioria ao padrão de menores valores de momento invariante para os momentos do círculo em relação ao triângulo, os valores dos sete momentos invariantes do círculo e do triângulo são próximos e em em alguns casos não é possível definir com exatidão se houve uma mudança na posição ou se é o contorno desejado, assim sendo, os resultados deste teste são inconsistentes para este trabalho.

Teste 07 – Foi aplicada a rotina de momentos invariantes ao vetor de cada contorno, completando os vetores com 480 linhas e 640 colunas, por método de concatenação. A função utilizada para concatenação foi "cat(1,b,zeros(480 - size(b,1)),zeros(640 - size(b,2)))", onde, o argumento de entrada "b" representa o contorno do objeto identificado em cena, e o argumento "zeros(480 - size(b,1)),zeros(640 - size(b,2)))" completa o vetor de contorno com zeros até formar uma matriz 480 linhas e 640 colunas, contendo os valores do contorno do objeto "b". Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo os resultados de momentos invariantes para cada posição conforme TAB. 7.

Tabela 7 - Valores de momentos invariantes obtidos através do teste 07.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	1,5939	3,1879	7,5419	7,5419	15,0838	9,1359	-20,2754
	Triângulo	1,4012	2,8026	7,3621	7,3621	14,7241	8,7635	-18,8339
Posição 02	Círculo	1,6334	3,2670	7,6140	7,6141	15,2282	9,2476	19,6945
	Triângulo	1,4149	2,8299	6,6296	6,6298	13,2596	8,0448	17,9178
Posição 03	Círculo	1,5891	3,1783	7,5531	7,5534	15,1067	9,1426	19,2780
	Triângulo	1,4213	2,8427	6,4119	6,4120	12,8240	7,8334	18,6209
Posição 04	Círculo	1,4723	2,9447	6,7074	6,7078	13,4154	8,1802	17,6899
	Triângulo	1,5251	3,0503	7,5430	7,5431	15,0861	9,0683	20,5618
Posição 05	Círculo	1,4038	2,8077	6,5066	6,5070	13,0138	7,9108	17,3291
	Triângulo	1,5669	3,1340	7,3271	7,3272	14,6544	8,8943	19,3531

Fonte: Autor.

Os resultados para o teste não são satisfatórios para aplicação neste trabalho, pois os valores de momento invariante para cada forma, círculo ou triângulo, são muito próximos, e a diferença entre valores de uma posição para outra do mesmo objeto se confunde com a diferença de valores entre objetos, ou seja, não há como se definir com exatidão se houve uma mudança de posição ou se são dois objetos diferentes. Os resultados deste teste são inconsistentes para este trabalho.

Teste 08 – Foi aplicada a rotina de momentos invariantes considerando cada objeto como uma imagem independente. Foi utilizada a função "imcrop(f,[(X1-(length(b)/2)) (Y1-(length(b)/2)) length(b) length(b)])". A função "imcrop" seleciona e corta partes da

imagem original conforme as coordenadas inseridas em seus argumentos de entrada. O efeito da função “imcrop” pode ser observado na FIG. 61:

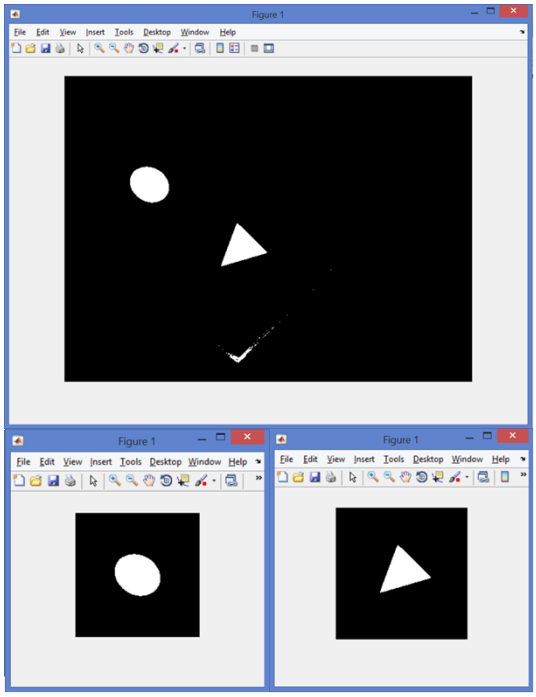


Figura 53 - Objetos cortados da imagem original através da função "imcrop".
Fonte: Autor.

Após a aplicação da função “imcrop”, foi realizado o cálculo dos momentos invariantes para cada objeto. Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo os resultados de momentos invariantes para cada posição conforme TAB. 8.

Tabela 8 - Valores de momentos invariantes obtidos através do teste 08.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	0,7928	3,1976	6,6749	8,9663	-17,2388	-11,0468	-16,8158
	Triângulo	0,7119	3,0561	2,3428	4,5234	7,9797	6,0758	8,4534
Posição 02	Círculo	0,7839	2,7645	7,4073	9,2293	-17,8730	-10,9292	-17,6026
	Triângulo	0,7148	3,4681	2,3466	5,0075	8,7502	6,8057	-8,9762
Posição 03	Círculo	0,7824	2,7210	7,2468	8,9360	17,0912	10,3666	-17,3244
	Triângulo	0,7115	3,0823	2,3380	4,5966	-9,3148	-7,0976	-8,0647

Fonte: Autor.

Os resultados do teste são satisfatórios, pois há forte distinção entre os valores de momento invariante obtidos para o círculo e para o triângulo, possuindo padrões que possibilitam a identificação exata da forma do objeto. Além disso, é possível observar pequenas variações, em módulo, para os valores de momento invariante de um objeto quando se altera a posição. Os resultados são consistentes e podem ser implementados no robô.

Teste 09 – Foi aplicada a rotina de momentos invariantes considerando cada objeto como uma imagem independente através da função "imcrop" e inserindo uma moldura através da função "padarray", com o objetivo de se padronizar o tamanho das matrizes geradas. Os objetos foram rotacionados de modo que as posições na cena fossem diferentes, obtendo os resultados de momentos invariantes para cada posição conforme TAB. 9.

Tabela 9 - Valores de momentos invariantes obtidos através do teste 09.

Posição	Forma	Momento Invariante						
		$\emptyset_1$	$\emptyset_2$	$\emptyset_3$	$\emptyset_4$	$\emptyset_5$	$\emptyset_6$	$\emptyset_7$
Posição 01	Círculo	0,7829	2,7335	6,8884	8,5589	16,3365	10,0018	-16,6110
	Triângulo	0,7122	3,0794	2,3443	4,6121	-9,5562	-7,3356	-8,0906
Posição 02	Círculo	0,7872	2,8805	9,0243	10,9380	20,9492	12,4368	21,3634
	Triângulo	0,7063	2,7985	2,3251	4,2755	7,7981	5,9532	-7,6725
Posição 03	Círculo	0,7851	2,8033	6,8840	8,7633	-16,6788	-10,2476	-16,8184
	Triângulo	0,7143	3,3770	2,3450	4,8687	8,4862	6,5704	-9,1367

Fonte: Autor.

Os resultados do teste são satisfatórios, pois há forte distinção entre os valores de momento invariante obtidos para o círculo e para o triângulo, com padrões que possibilitam a identificação exata da forma do objeto. Além disso, é possível observar pequenas variações, em módulo, para os valores de momento invariante de um objeto quando se altera a posição. Os resultados são consistentes e podem ser implementados no robô.

Após os testes de cálculo dos momentos invariantes com o intuito de resolver o problema de rotação da imagem, foi escolhida a aplicação no formato do teste 09, onde cada objeto é considerado como uma imagem independente acrescido de moldura como forma de padronização do tamanho dos elementos.

Algumas rotinas foram adaptadas com o intuito de incorporar ao robô essa melhoria. Uma das rotinas adaptadas foi a "MULTI_ALVO", sendo renomeada para "MULTI_ALVO2". Esta rotina tem função de definir qual objeto será utilizado como alvo, de

calcular os valores de coordenadas espaciais do centro do alvo e também os momentos invariantes. Pode ser encontrada no Apêndice deste trabalho.

Outra rotina que foi modificada é a rotina “ALVO”, renomeada para “ALVO3”. Essa rotina tem o objetivo de identificar o alvo correto dentre os objetos da cena e calcular suas respectivas coordenadas espaciais de centro. A modificação ocorrida na rotina foi o acréscimo do cálculo e comparação dos momentos invariantes dos objetos de cena. O cálculo do valor padrão de momentos invariantes para o alvo é realizado pela rotina “MULTI_ALVO2”, armazenado no argumento de saída “MI”. A rotina “ALVO3” lê a variável “MI”, identifica todos os contornos de objetos existentes na cena, calculando o momento invariante para cada um. Logo após, os valores de momentos invariantes identificados na cena são comparados com os quatro primeiros valores de momentos invariantes do alvo armazenado na variável “MI”. O valor mais próximo de “MI” é definido como sendo o contorno do alvo. Para efetuar a comparação do valor padrão de momentos invariantes com os valores obtidos na rotina “ALVO3” é utilizado o cálculo da variância.

A variância é uma medida de dispersão que mostra o quão distante cada valor desse conjunto está do valor central (BRASIL ESCOLA, 2017). É calculada através da fórmula:

$$VARIÂNCIA = \frac{(x_1 - \bar{x})^2}{n - 1} \tag{15}$$

Onde:

- x_1 Elemento da amostra;
- $\bar{x}$ Média aritmética dos elementos;
- n Número de elementos da amostra.

Na rotina “ALVO3”, o valor padrão de momentos invariantes “MI” do alvo é considerado como sendo a média aritmética para o cálculo da variância.

Após o cálculo da variância, é feita a comparação dos valores obtidos para os quatro primeiros coeficientes numéricos de momentos invariantes, possibilitando, assim, se identificar qual o objeto é o alvo. A rotina “ALVO3” pode ser encontrada no Apêndice deste trabalho.

Conforme observado na página 65 deste trabalho, foi realizada modificação na rotina “MOVIMENTA_SERVOS”, que controla o movimento dos servos motores. Tal alteração foi realizada com o objetivo de melhorar a estabilidade do protótipo.

A rotina “MOVIMENTA_SERVOS” foi renomeada para “MOVIMENTA_SERVOS_3” e é apresentada no Apêndice deste trabalho.

A rotina “MOVIMENTA_SERVOS_3” operacionalmente lê o valor do erro, o erro na coordenada x e o erro na coordenada y. Analisa os valores dos erros em x e y conforme as seguintes situações:

- Erro maior que 50 pixels: incrementa 5 graus no servo motor;
- Erro menor que -50 pixels: decrementa 5 graus no servo motor;
- Erro maior que 5 e menor ou igual a 50 pixels: incrementa 1 grau no servo motor;
- Erro menor que -5 e maior ou igual a -50 pixels: decrementa 1 grau no servo motor;
- Erro menor que 5 pixels e maior que 0: não movimenta o servo motor;
- Erro maior que -5 pixels e menor que 0: não movimenta o servo motor.

No final da rotina “MOVIMENTA_SERVOS_3” os servos são movimentados de maneira conjunta, diferentemente da rotina “MOVIMENTA_SERVOS”, onde os servo motores são movimentados em cada fluxo de decisão e separadamente.

Para a operação em conjunto de todas as rotinas de programação, foi criada a rotina principal denominada “MAIN_TCC_3”. Essa rotina engloba todas as rotinas necessárias para que o protótipo opere de forma satisfatória, cumprindo os objetivos do trabalho. A rotina “MAIN_TCC_3” é apresentada no Apêndice deste trabalho.

A rotina de programação “MAIN_TCC_3” apresentada na FIG. 67 executa as operações na seguinte ordem:

- Inicializa o protótipo, configurando Arduino e *webcam;*
- Identifica e seleciona o alvo dentre os objetos presentes na cena;
- Posiciona o *laser* dentro da cena;
- Executa um ciclo de 200 amostragens onde: identifica as coordenadas de centro do alvo, calcula o módulo das coordenadas de centro do alvo; identifica as coordenadas da posição do *laser*; calcula o módulo das coordenadas da posição do *laser*; calcula o módulo do erro, o erro em x e o erro em x; apresenta o valor do erro na tela; movimenta os servo motores executando o controle de posição e armazena os valores de posição do alvo, do *laser* e do erro;
- Caso o *laser* exceda os limites da cena, reposiciona-o novamente;
- Apresenta o gráfico de resultados.

A forma de operação e os resultados da rotina “MAIN_TCC_3” são dependentes de outras rotinas de programação que executam tarefas separadamente. São elas:

- INICIALIZA_TCC;
- MULTI_ALVO2;

- ALVO3;
- MIRA;
- MOVIMENTA_SERVOS_3.

O conjunto de rotinas que fazem parte da rotina principal produzem resultados satisfatórios às proposições do trabalho. Os resultados dos testes realizados são apresentados a seguir.

4 RESULTADOS E DISCUSSÕES

Teste 01 – Alvo circular.

Figura 54 - Alvo circular utilizado no teste 01.
Fonte: Autor.

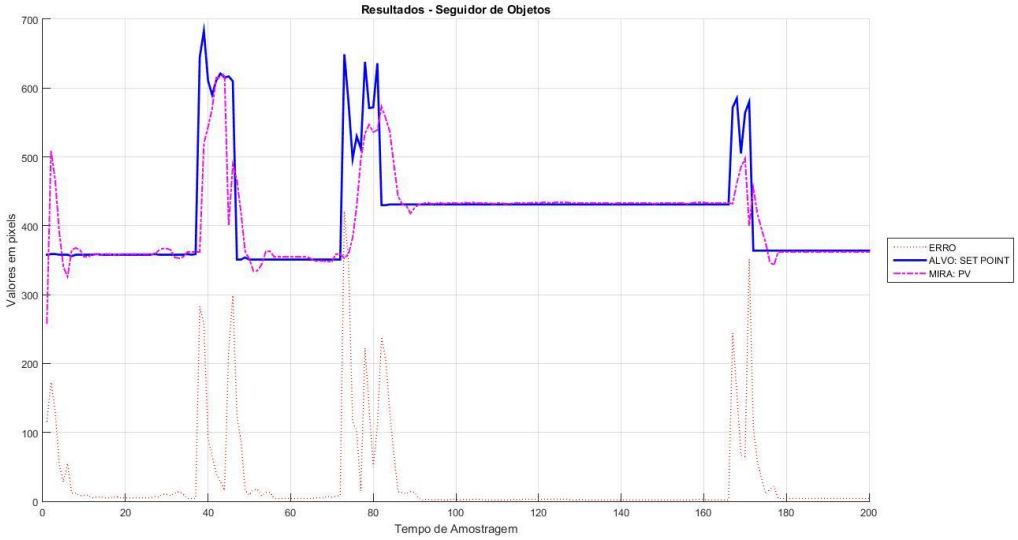


Figura 55 - Gráfico do resultado do teste 01.
Fonte: Autor.

O resultado do teste com um alvo circular único foi eficaz, demonstrando variações apenas ao final do momento transitório e em seguida estabilizando a mira sobre o centroide do alvo. O momento transitório é representado nos trechos de instabilidade, ou seja, onde podem ser observados picos angulosos da linha azul que representa o alvo na FIG. 69.

Teste 02 – Alvo retangular.

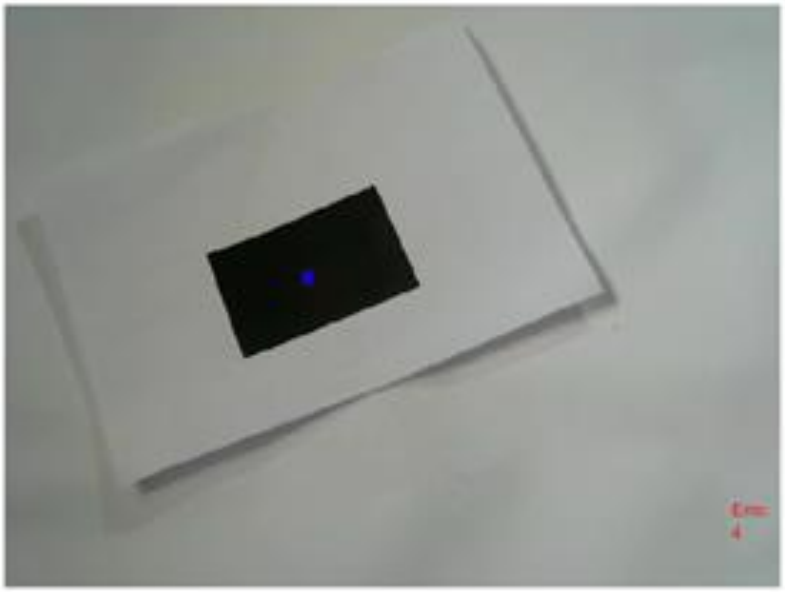


Figura 56 - Alvo retangular utilizado no teste 02.
Fonte: Autor.

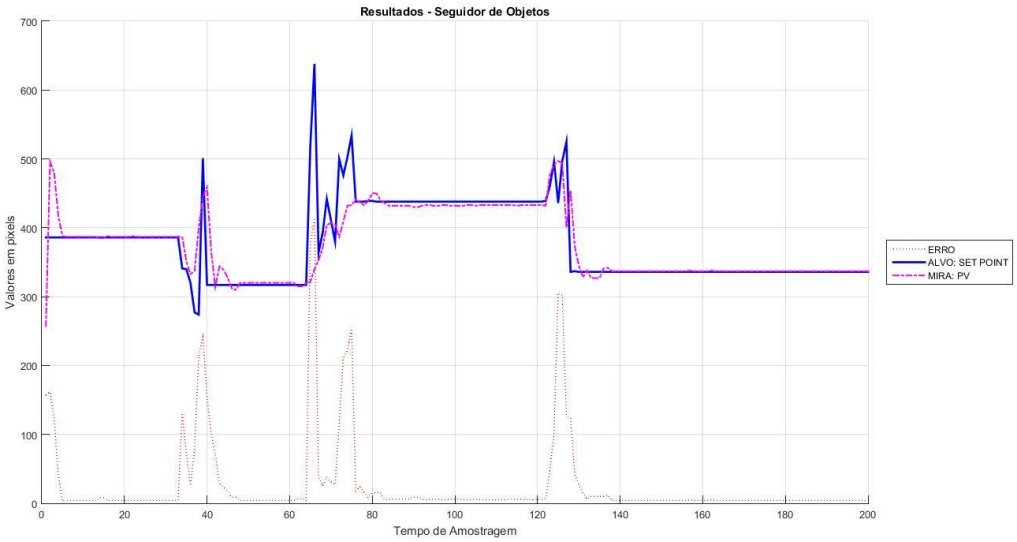


Figura 57 - Gráfico do resultado do teste 02.
Fonte: Autor.

O resultado do teste com um alvo retangular único foi eficaz, demonstrando variações apenas ao final do momento transitório e em seguida estabilizando a mira sobre o centroide do alvo. O momento transitório é representado nos trechos de instabilidade, ou seja, onde podem ser observados picos angulosos da linha azul que representa o alvo na FIG. 71.

Teste 03 – Alvo em formato de seta.

Figura 58 - Alvo em formato de seta utilizado no teste 03.
Fonte: Autor.

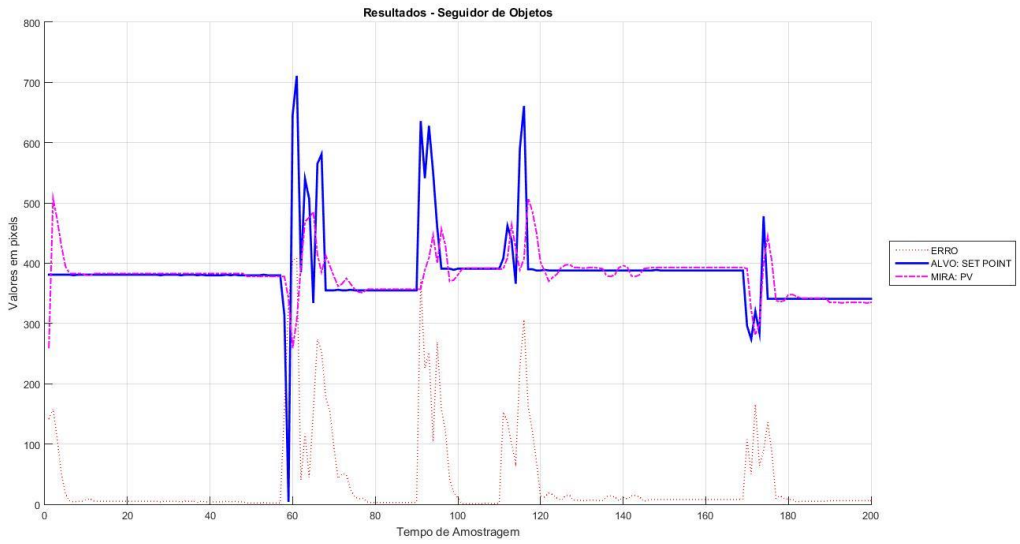


Figura 59 - Gráfico do resultado do teste 03.
Fonte: Autor.

O resultado do teste com um alvo único na cena e em formato de seta foi eficaz, demonstrando variações ao final do momento transitório e em seguida estabilizando a mira sobre o centroide do alvo, porém houveram trechos com pequenas variações menores do que 10 pixels. O momento transitório é representado nos trechos de instabilidade, ou seja, onde podem ser observados picos angulosos da linha azul que representa o alvo na FIG. 73.

Teste 04 – Alvo com formato irregular.

Figura 60 - Alvo com formato irregular utilizado no teste 04.
Fonte: Autor.

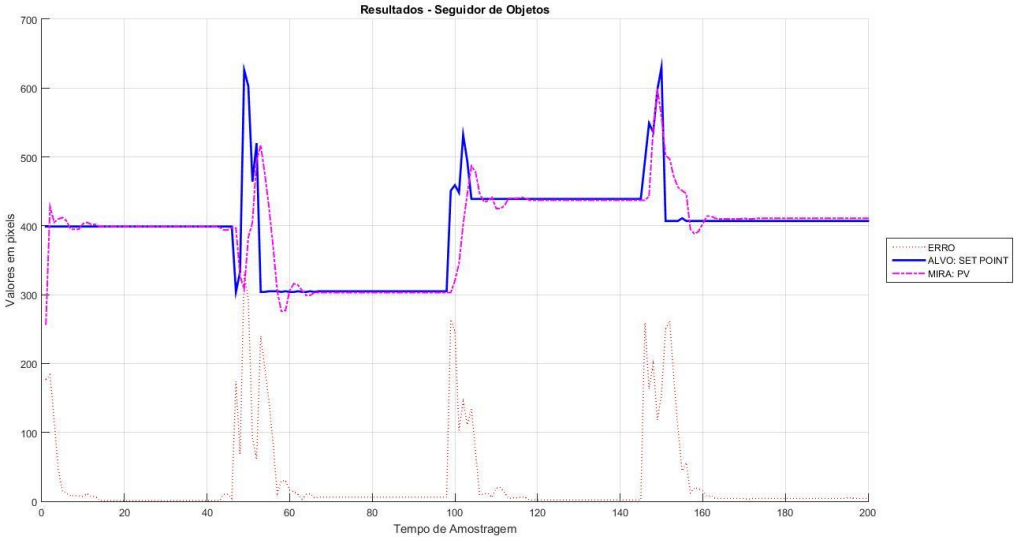


Figura 61 - Gráfico de resultados para o teste 04.
Fonte: Autor.

O resultado do teste com um alvo em formato irregular e único na cena foi eficaz, demonstrando variações ao final do momento transitório e em seguida estabilizando a mira sobre o centroide do alvo. O momento transitório é representado nos trechos de instabilidade, ou seja, onde podem ser observados picos angulosos da linha azul que representa o alvo na FIG. 75.

Teste 05 – Objetos múltiplos em cena, com alvo circular selecionado.

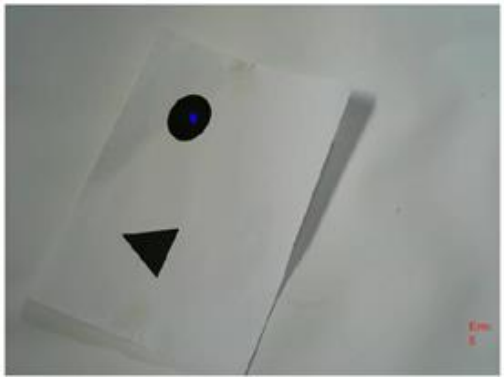


Figura 62 - Objeto circular selecionado como alvo no teste 05.
Fonte: Autor.

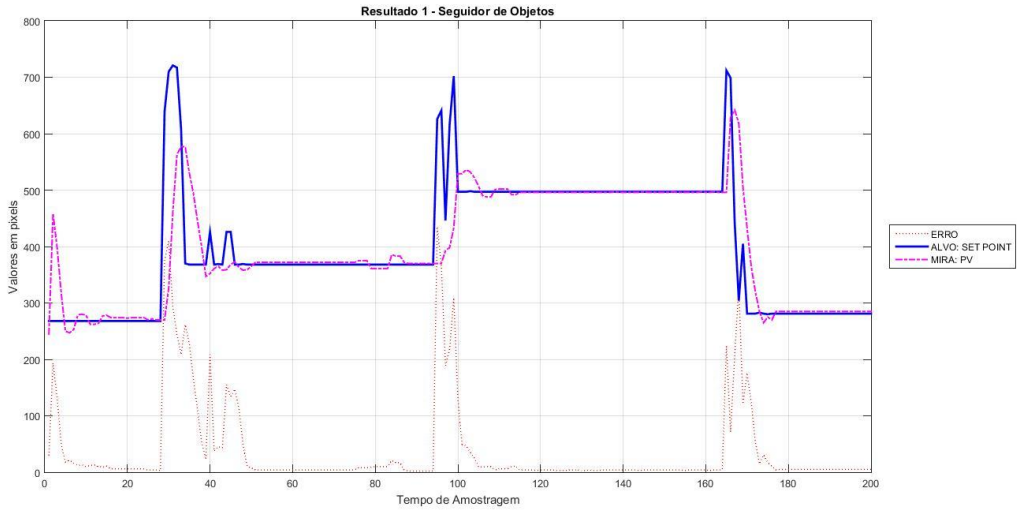


Figura 63 - Gráfico de resultados para o teste 05.
Fonte: Autor.

O resultado do teste com dois objetos em cena, sendo selecionado como alvo o objeto circular demonstrou variações ao final do momento transitório e em seguida estabilizando a mira sobre o centroide do alvo. O momento transitório é representado nos trechos de instabilidade, ou seja, onde podem ser observados picos angulosos da linha azul que representa o alvo na FIG. 77.

Teste 06 – Objetos múltiplos em cena, com alvo triangular selecionado.

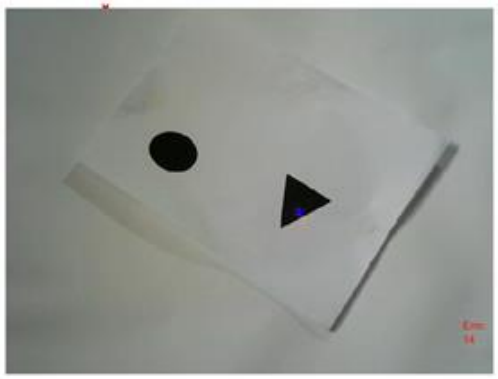


Figura 64 - Objeto triangular selecionado como alvo no teste 06.
Fonte: Autor.

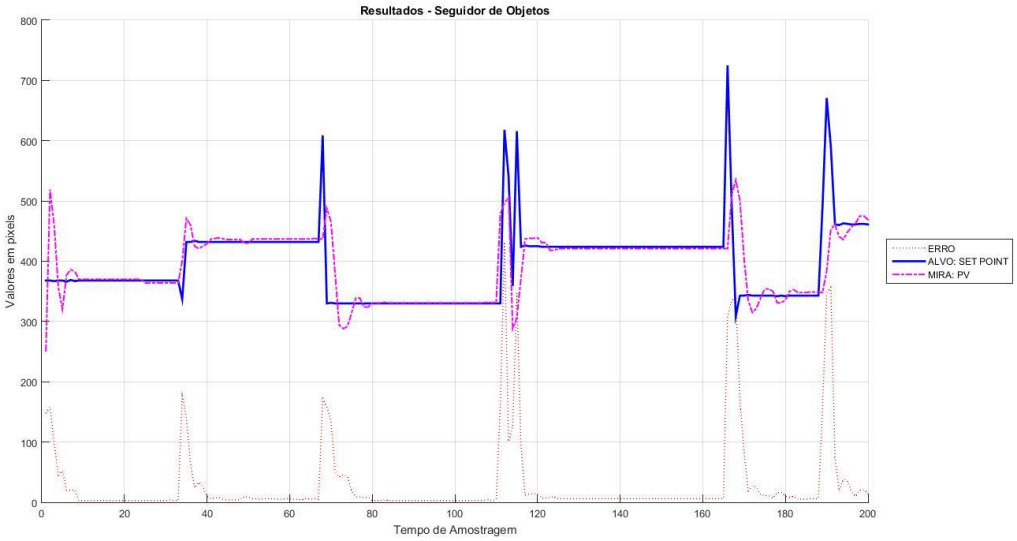


Figura 65 - Gráfico de resultado para o teste 06.
Fonte: Autor.

O resultado do teste com dois objetos em cena, sendo selecionado como alvo o objeto triangular demonstrou variações ao final do momento transitório e em seguida estabilizando a mira sobre o centroide do alvo. O momento transitório é representado nos trechos de instabilidade, ou seja, onde podem ser observados picos angulosos da linha azul que representa o alvo na FIG. 79.

CONSIDERAÇÕES FINAIS

No âmbito dos objetivos gerais, os resultados dos testes realizados neste trabalho demonstram ter alcançado com sucesso, onde é possível identificar e selecionar alvos móveis através de mira laser. No decorrer do desenvolvimento foram encontrados desafios que foram superados, o que contribuiu positivamente para consistência nos resultados do trabalho.

Apesar de ter atendido ao objetivo inicial, o trabalho encontrou pontos a serem considerados. O primeiro deles é a interferência da iluminação no funcionamento do sistema. Em algumas situações a iluminação causa forte sombreamento em pontos da imagem capturada pela câmera, ocorrendo a falha de identificação do alvo devido à isso. É possível a instalação de sensores que compensem a variação de iluminação, porém, no trabalho esta compensação é feita através de ajuste manual do valor do limiar da máscara de transformação da imagem colorida para escala de cinza. O segundo ponto são os reflexos luminosos que podem aparecer em cena, provocando identificação errada da localização da mira, levando à instabilidade do sistema. O terceiro ponto é o ciclo de scan do programa limitar, em algumas situações, o acompanhamento em tempo real da mira laser sobre o alvo em movimento. O ciclo de scan programa gera atrasos consideráveis na correção do erro entre mira e centro do alvo. O quarto ponto é a dificuldade de diferenciação entre objetos circulares e quadrados na mesma cena, sendo os recursos de programação utilizados insuficientes para alcançar a eficiência esperada nesta situação.

Como sugestão para aplicações em automação industrial, este trabalho pode ser utilizado em reconhecimento, inspeção e detecção de dimensões de objetos em processos.

Para reconhecimento de objetos, o protótipo pode ser utilizado na detecção de posição de peças e comparação de contornos no controle de qualidade e tarefas de separação de objetos defeituosos da linha de produção. Pode ser utilizado também para avaliação de presença de objetos, identificando os locais de peças faltantes.

Para inspeção, pode ser utilizado na detecção de falhas e furos em peças e componentes fabricados e identificação de materiais aplicados de forma irregular em uma linha de montagem.

Como sugestão para aplicação m ambiente acadêmico, o robô configura plataforma didática para estudos e experimentos na área de programação e processamento digital de imagens, complementando os recursos didáticos disponíveis na instituição de ensino.

REFERÊNCIAS

BRASIL ESCOLA. Disponível em: http://brasilescola.uol.com.br/matematica/medidas-dispersao-variancia-desvio-padrao.htm. Acesso em: 22/11/2017.

GILAT, Amos. **MATLAB com aplicações em engenharia.** Tradução Glayson Eduardo de Figueiredo. 2. ed. Porto Alegre: Bookman, 2006.

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento Digital de Imagens.** São Paulo: Pearson, 2010.

PITHAN, Sérgio Luiz da Silva. **Notas de aula.** CEFET-MG /Unidade Araxá. Araxá/MG, 2017.

SOLOMON, Chris; BRECKON, Toby. **Fundamentos de Processamento Digital de Imagens.** Rio de Janeiro: LTC, 2013.

WIKIPÉDIA, a enciclopédia livre. 25 jul. 2017. Disponível em: https://pt.wikipedia.org/wiki/Wikip%C3%A9dia:P%C3%A1gina_principal. Acesso em: 22/11/2017.

APÊNDICE

ROTINA: INICIALIZA_TCC

```
function [a, cam]=INICIALIZA_TCC
%OBJETIVO DA FUNÇÃO:    INICIALIZAR E DEIXAR OS DISPOSITIVOS PRONTOS
PARA
%                       RECEBER E EXECUTAR COMANDOS E INSTRUÇÕES
%
clc
arduino
clc
fprintf('MICROCONTROLADOR ARDUINO MEGA 2560 RECONHECIDO.\n');
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
clc
a = arduino('COM3')     %CRIA A VARIÁVEL "a" NO WORKSPACE E DEFINE A
PORTA
                        %SERIAL COM3 COMO PORTA DE COMUNICAÇÃO ENTRE O
                        %MICROPROCESSADOR ARDUINO MEGA 2560 E O
COMPUTADOR
clc
fprintf('PORTA SERIAL CONFIGURADA.\n');
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
clc
webcamlist;             %LISTA AS WEBCAMS DISPONÍVEIS
cam = webcam(2)         %DEFINE A WEBCAM QUE SERÁ UTILIZADA
clc
fprintf('WEBCAM CONFIGURADA.\n');
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
clc
servoAttach(a,8);       %DEFINE O PINO 8 DO ARDUINO COMO SENDO PINO DE
                        %SINAL PARA COMANDO DO SERVO MOTOR HORIZONTAL
servoAttach(a,12);      %DEFINE O PINO 12 DO ARDUINO COMO SENDO PINO DE
                        %SINAL PARA COMANDO DO SERVO MOTOR VERTICAL


servoWrite(a,8,78);
servoWrite(a,12,18);


fprintf('SERVO MOTORES CONFIGURADOS.\n');
fprintf('SISTEMA CONECTADO COM SUCESSO.\n');
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
clc
```

Figura 66 - Rotina INICIALIZA_TCC.
Fonte: Autor.

ROTINA: CALIBRA_DISPOSITIVO

```
function [HC, VC, LEH, LDH, LSV, LIV]=CALIBRA_DISPOSITIVO(a,cam)
%OBJETIVO DA FUNÇÃO:    CALIBRAR O DISPOSITIVO PARA DESCOBRIR AS
%                       COORDENADAS DO CENTRO DA IMAGEM
clc;
close all;
clear HC;
clear VC;
servoWrite(a,8,78);
servoWrite(a,12,5);
Im = snapshot(cam);
figure(1);
hold on;
RI = imref2d(size(Im));
RI.XWorldLimits = [0 180];
RI.YWorldLimits = [0 180];
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',3,'color','r');
f=plot(90,0:180,'.','LineWidth',2,'color','r');
OK=0;
while(OK~=1)
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',3,'color','r');
f=plot(90,0:180,'.','LineWidth',2,'color','r');
HC = input('DIGITE O VALOR DO CENTRO HORIZONTAL:\n');
servoWrite(a,8,HC);
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',3,'color','r');
f=plot(90,0:180,'.','LineWidth',2,'color','r');
OK = input('ESTÁ CORRETO?\n(NÃO = 0/SIM = 1):\n');
end
OK=0;
while(OK~=1)
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',3,'color','r');
f=plot(0:180,90,'.','LineWidth',2,'color','r');
VC = input('DIGITE O VALOR DO CENTRO VERTICAL:\n');
servoWrite(a,12,VC);
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',3,'color','r');
f=plot(0:180,90,'.','LineWidth',2,'color','r');
OK = input('ESTÁ CORRETO?\n(NÃO = 0/SIM = 1):\n');
end
fprintf('AS COORDENADAS DE CENTRO DA IMAGEM SÃO:\n(Xc,Yc) =
(%.f,%.f);\n', HC,VC);
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
clc;
OK=0;
```

Continuação...

```
while(OK~=1)
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(179,90,'x','LineWidth',3,'color','r');
f=plot(179,0:180,'.','LineWidth',2,'color','r');
LDH = input('DIGITE O VALOR DO LIMITE DIREITO HORIZONTAL:\n');
servoWrite(a,8,LDH);
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(179,90,'x','LineWidth',3,'color','r');
f=plot(179,0:180,'.','LineWidth',2,'color','r');
OK = input('ESTÁ CORRETO?\n(NÃO = 0/SIM = 1):\n');
end
servoWrite(a,8,HC);
servoWrite(a,12,VC);
Im = snapshot(cam);
imshow(Im,RI);
hold on
OK=0;
clc;
while(OK~=1)
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(1,90,'x','LineWidth',3,'color','r');
f=plot(1,0:180,'.','LineWidth',2,'color','r');
LEH = input('DIGITE O VALOR DO LIMITE ESQUERDO HORIZONTAL:\n');
servoWrite(a,8,LEH);
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(0,90,'x','LineWidth',3,'color','r');
f=plot(0,0:180,'.','LineWidth',2,'color','r');
OK = input('ESTÁ CORRETO?\n(NÃO = 0/SIM = 1):\n');
end
fprintf('AS COORDENADAS DE RANGE HORIZONTAL SÃO:\n(%.f,%.f);\n',
LDH,LEH);
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
servoWrite(a,8,HC);
servoWrite(a,12,VC);
Im = snapshot(cam);
imshow(Im,RI);
hold on
OK=0;
clc;
while(OK~=1)
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,0,'x','LineWidth',3,'color','r');
f=plot(0:180,1,'.','LineWidth',2,'color','r');
LSV = input('DIGITE O VALOR DO LIMITE SUPERIOR VERTICAL:\n');
servoWrite(a,12,LSV);
Im = snapshot(cam);
```

Continuação...

```
imshow(Im,RI);
hold on
f=plot(90,0,'x','LineWidth',3,'color','r');
f=plot(0:180,1,'.','LineWidth',2,'color','r');
OK = input('ESTÁ CORRETO?\n(NÃO = 0/SIM = 1):\n');
end
servoWrite(a,8,HC);
servoWrite(a,12,VC);
Im = snapshot(cam);
imshow(Im,RI);
hold on
OK=0;
clc;
while(OK~=1)
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,180,'x','LineWidth',3,'color','r');
f=plot(0:180,179,'.','LineWidth',2,'color','r');
LIV = input('DIGITE O VALOR DO LIMITE INFERIOR VERTICAL:\n');
servoWrite(a,12,LIV);
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,180,'x','LineWidth',3,'color','r');
f=plot(0:180,179,'.','LineWidth',2,'color','r');
OK = input('ESTÁ CORRETO?\n(NÃO = 0/SIM = 1):\n');
end
fprintf('AS COORDENADAS DE RANGE VERTICAL SÃO:\n(%.f,%.f);\n', LSV,LIV);
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...');
pause;
clc;
passo=1;
fprintf('CENTRO\n');
for (k=5:-passo:0)
servoWrite(a,8,(HC));
servoWrite(a,12,(VC));
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',3,'color','r');
end

fprintf('CANTO INFERIOR ESQUERDO\n');
for (k=5:-passo:0)
servoWrite(a,8,(LEH));
servoWrite(a,12,(LIV));
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(1,179,'x','LineWidth',3,'color','r');
end
printf('CANTO INFERIOR DIREITO\n');
for (k=5:-passo:0)
servoWrite(a,8,(LDH));
servoWrite(a,12,(LIV));
```

Continuação...

```
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(179,179,'x','LineWidth',3,'color','r');
end
pause(1);
fprintf('CANTO SUPERIOR ESQUERDO\n');
for (k=5:-passo:0)
servoWrite(a,8,(LEH));
servoWrite(a,12,(LSV));
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(1,1,'x','LineWidth',3,'color','r');
end
pause(1);
fprintf('CANTO SUPERIOR DIREITO\n');
for (k=5:-passo:0)
servoWrite(a,8,(LDH));
servoWrite(a,12,(LSV));
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(179,1,'x','LineWidth',3,'color','r');
end
pause(1);
fprintf('CENTRO\n');
for (k=5:-passo:0)
servoWrite(a,8,(HC));
servoWrite(a,12,(VC));
Im = snapshot(cam);
imshow(Im,RI);
hold on
f=plot(90,90,'x','LineWidth',5,'color','g');
end
fprintf('FIM DA CALIBRAÇÃO.\n');
fprintf('PRESSIONE "ENTER" PARA CONTINUAR...\n');
pause;
```

Figura 67 - Rotina "CALIBRA_DISPOSITIVO".
Fonte: Autor.

ROTINA: CONVERSAO_PADRAO

```
function [SX, SY] = CONVERSAO_PADRAO(IX,IY,LDH,LEH,LIV,LSV)
SY=((-28*(IY-480))/480)+2;   %CONVERSÃO CORRETA DE ACORDO COM A
REPRESENTAÇÃO DA IMAGEM
SX=((-44*(IX))/640)+103;     %CONVERSÃO CORRETA DE ACORDO COM A
REPRESENTAÇÃO DA IMAGEM
SX=round(SX)
SY=round(SY)
```

Figura 68 - Rotina "CONVERSAO_PADRAO".
Fonte: Autor.

ROTINA: ALVO

```
%FUNÇÃO PARA DETECTAR O CENTRO DO ALVO
%RETORNA AS COORDENADAS X1 E Y1 DO CENTRO DO ALVO
 function [Y1,X1]=ALVO(a,cam)
 figure(1)                          %ABRE A JANELA DA FIGURA 1
 hold on;                           %MANTÉM A JANELA EM UTILIZAÇÃO
warning off
Im = snapshot(cam);                 %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                    %PASSA A IMAGEM DE RGB (03 MATRIZES)
   f = rgb2gray(Im);                %PARA ESCALA DE CINZA(01 MATRIZ)
end
f= imcomplement(f);                 %ENCONTRA O NEGATIVO DA IMAGEM
f= im2bw(f,.82);                    %BINARIZA A IMAGEM PARA UM LIMIAR
%IDENTIFICA O CONTORNO DOS OBJETOS
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                          %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                         %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
ALVO)
b = B{1};                           %SELECIONA QUAL CONTORNO PARA
PROCESSAMENTO
[Y1,X1,Ro,theta] = Polar2(b);       %RETORNA O CENTRO DO CONTORNO ATRAVÉS DE
                                    %COORDENADAS POLARES
%[Y1,X1,Ro,theta] = Polar2M(b); RETORNA O CENTRO DO CONTORNO ATRAVÉS DE
                                    %COORDENADAS POLARES COMPENSADAS DE
                                    %DISTORÇÃO
X1=round(X1);                       %ARREDONDAMENTO PARA NÚMERO INTEIRO
Y1=round(Y1);                       %ARREDONDAMENTO PARA NÚMERO INTEIRO
figure(1);                          %ABRE A JANELA DA FIGURA 1
hold on;                            %MANTÉM A JANELA EM UTILIZAÇÃO
imshow(Im)                          %MOSTRA A IMAGEM INICIAL
hold on;                            %MANTÉM A JANELA EM UTILIZAÇÃO
Im = plot(X1,Y1,'X');               %PLOTA UM X NO CENTRO DO ALVO
%CONFIGURAÇÕES DO MARCADOR DO CENTRO DO ALVO
set(Im,'LineWidth',3,'color','r');
hold on;                            %MANTÉM A JANELA EM UTILIZAÇÃO
%CASO O TAMANHO DO ARRANJO B=0, SIGNIFICA QUE NÃO HOUVE IDENTIFICAÇÃO
%DE NENHUM CONTORNO. CASO ACONTEÇA, ESCREVE MENSAGEM NA TELA E
%DÁ PROSSEGUIMENTO À ROTINA DE PROGRAMAÇÃO
else
    imshow(Im);                     %MOSTRA A IMAGEM INICIAL
%ESCREVE MENSAGEM NA IMAGEM
    text(20,20,'ALVO NÃO DETECTADO','Color','red','FontSize',10);
    X1=320;                         %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    Y1=240;                         %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    servoWrite(a,8,90);             %REPOSICIONA O SERVO MOTOR HORIZONTAL
    servoWrite(a,12,70);            %REPOSICIONA O SERVO MOTOR VERTICAL
    pause(2);                       %PAUSA PARA ESTABILIZAÇÃO DOS SERVO
MOTORES
end
```

Figura 69 - Rotina "ALVO" para identificar o objeto alvo e suas coordenadas de centro.
Fonte: Autor.

ROTINA: MIRA

```
%FUNÇÃO PARA DETECTAR O CENTRO DA MIRA
%RETORNA AS COORDENADAS XM E YM DO CENTRO DA MIRAO
function [YM,XM,RESET]=MIRA(a,cam)
Im = snapshot(cam);              %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                 %PASSA A IMAGEM DE RGB (03 MATRIZES)
   f = rgb2gray(Im);             %PARA ESCALA DE CINZA(01 MATRIZ)
end
f= im2bw(f,.87);                 %BINARIZA A IMAGEM PARA UM LIMIAR
%IDENTIFICA O CONTORNO DA MIRA
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                       %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                      %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
MIRA)
b = B{1};                        %SELECIONA DO CONTORNO DA MIRA PARA
                                 %PROCESSAMENTO. DEVERÁ SER O ÚNICO
CONTORNO
                                 %DETECTADO, LOGO "L" DEVERÁ SER L=1
[YM,XM,Ro,theta] = Polar2(b);    %RETORNA O CENTRO DA MIRA EM COORDENADA
                                 %POLARES
%[Y1,X1,Ro,theta] = Polar2M(b); %RETORNA O CENTRO DA MIRA EM COORDENADA
                                 %POLARES
XM=round(XM);                    %ARREDONDAMENTO PARA NÚMERO INTEIRO
YM=round(YM);                    %ARREDONDAMENTO PARA NÚMERO INTEIRO
figure(1);                       %ABRE A JANELA DA FIGURA 1
hold on;                         %MANTÉM A JANELA EM UTILIZAÇÃO
imshow(Im)                       %MOSTRA A IMAGEM INICIAL
hold on;                         %MANTÉM A JANELA EM UTILIZAÇÃO
Im = plot(XM,YM,'x');            %PLOTA UM X NO CENTRO DA MIRA
%CONFIGURA O MARCADOR PLOTADO
set(Im,'LineWidth',3,'color','b');
hold on;                         %MANTÉM A JANELA EM UTILIZAÇÃO
RESET=0;                         %ZERA O ÍNDICE RESET
%CASO O TAMANHO DO ARRANJO B=0, SIGNIFICA QUE NÃO HOUVE IDENTIFICAÇÃO
%DE NENHUM CONTORNO. CASO ACONTEÇA, ESCREVE MENSAGEM NA TELA E
%DÁ PROSSEGUIMENTO À ROTINA DE PROGRAMAÇÃO
else
    imshow(Im);                  %MOSTRA A IMAGEM INICIAL
%ESCREVE MENSAGEM NA IMAGEM
text(20,55,'MIRA NÃO DETECTADA','Color','blue','FontSize',10);
    XM=320;                      %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    YM=240;                      %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    servoWrite(a,8,90);          %REPOSICIONA O SERVO MOTOR HORIZONTAL
    servoWrite(a,12,70);         %REPOSICIONA O SERVO MOTOR VERTICAL
    RESET=1;                     %ARMAZENA O VALOR 1 NO ÍNDICE RESET
    pause(1);                    %PAUSA PARA ESTABILIZAÇÃO DOS SERVO
MOTORES
end
```

Figura 70 - Rotina "MIRA" para identificar o marcador *laser* e suas coordenadas de centro.
Fonte: Autor.

ROTINA: Polar2

```
function [x0,y0,Ro,angle] = Polar2(b,x0,y0)
[np,nc] = size(b);
if (np < nc || nc ~= 2)                 %CONTROLE DE FLUXO PARA
                                        %VERIFICAR O FORMATO CORRETO
                                        %DO ARRANJO DE ENTRADA.
   error('b DEVE SER N-LINHAS-POR-2.'); %MENSAGEM DE ERRO CASO ARRANJO
                                        %NÃO CONFORME
end
if isequal(b(1,:),b(np,:))              %CONTROLE DE FLUXO PARAVERIFICAR
  b = b(1:np - 1,:);                    %O NUMERO DE LINHAS DO ARRANJO
   np = np - 1;                         %DE %ENTRADA.
end
if nargin == 1                          %CONTROLE DE FLUXO PARA
                                        %VERIFICAR OS ARGUMENTOS DE
                                        %ENTRADA.
   x0 = (sum(b(:,1))/np);               %COORDENADAS DO CENTRO
   y0 = (sum(b(:,2))/np);               %COORDENADAS DO CENTRO
end
b(:,1) = b(:,1) - x0;                   %DETERMINA O CENTRO DO ARRANJO
b(:,2) = b(:,2) - y0;                   %DETERMINA O CENTRO DO ARRANJO
xcart = b(:,2);                         %COORDENADA X CARTESIANA
ycart = -b(:,1);                        %COORDENADA Y CARTESIANA
[theta,rho] = cart2pol(xcart,ycart);    %CONVERTE DE CARTESIANO PARA
POLAR
theta = theta.*(180/pi);                %CONVERTE DE RADIANOS PARA GRAUS
j = theta == 0;                         %ARMAZENA OS ÍNDICES DE THETA=0
                                        %PARA USO POSTERIOR.
%CONVERTE OS ÂNGULOS NEGATIVOS PARA SEUS RESPECTIVOS VALORES POSITIVOS
theta = theta.*(0.5*abs(1 + sign(theta)))...
        - 0.5*(-1 + sign(theta)).*(360 + theta);
theta(j) = 0;                           %PRESERVA OS VALORES ZERO
%APROXIMA THETA PARA INCREMENTOS DE 1 GRAU
theta = round(theta);
tr = [theta,rho];
%APAGA ÂNGULOS DUPLICADOS
[w,u] = unique(tr(:,1));
tr = tr(u,:);
%SE O ÂNGULO 360° FOR ATINGIDO E HOUVER 0° O ÚLTIMO ÂNGULO É APAGADO.
if tr(end,1) == tr(1) + 360
   tr = tr(1:end - 1, :);
end
angle = tr(:,1);
Ro = tr(:,2);
```

Figura 71 - Função “[YM,XM,Ro,theta] = Polar2(b)”.
Fonte: Autor.

ROTINA: Polar2M

```
function [x0,y0,Ro,angle] = Polar2M(b,x0,y0)
[np,nc] = size(b);
if (np < nc || nc ~= 2)                  %CONTROLE DE FLUXO PARA
VERIFICAR
                                         %O FORMATO CORRETO DO ARRANJO DE
                                         %ENTRADA.
   error('b DEVE SER N-LINHAS-POR-2.'); %MENSAGEM DE ERRO CASO ARRANJO
                                         %NÃO CONFORME
end
if isequal(b(1,:),b(np,:))               %CONTROLE DE FLUXO PARA
VERIFICAR
                                         %O NUMERO DE LINHAS DO ARRANJO
DE
                                         %ENTRADA.
   b = b(1:np - 1,:);
   np = np - 1;
end
if nargin == 1                           %CONTROLE DE FLUXO PARA
VERIFICAR
                                         %OS ARGUMENTOS DE ENTRADA.
   x0 = (sum(b(:,1))/np);                %COORDENADAS DO CENTRO
   y0 = (sum(b(:,2))/np);                %COORDENADAS DO CENTRO
end
b(:,1) = b(:,1) - x0;                    %DETERMINA O CENTRO DO ARRANJO
b(:,2) = b(:,2) - y0;                    %DETERMINA O CENTRO DO ARRANJO
xcart = b(:,2);                          %COORDENADA X CARTESIANA
ycart = -b(:,1);                         %COORDENADA Y CARTESIANA
[theta,rho] = cart2pol(xcart,ycart);     %CONVERTE DE CARTESIANO PARA
POLAR
theta = theta.*(180/pi);                 %CONVERTE DE RADIANOS PARA GRAUS
j = theta == 0;                          %ARMAZENA OS ÍNDICES DE THETA=0
                                         %PARA USO POSTERIOR.
%CONVERTE OS ÂNGULOS NEGATIVOS PARA SEUS RESPECTIVOS VALORES POSITIVOS
theta = theta.*(0.5*abs(1 + sign(theta)))...
        - 0.5*(-1 + sign(theta)).*(360 + theta);
theta(j) = 0;                            %PRESERVA OS VALORES ZERO
tr = [theta,rho];
%APAGA ÂNGULOS DUPLICADOS
[w,u] = unique(tr(:,1));
tr = tr(u,:);
%SE O ÂNGULO 360° FOR ATINGIDO E HOUVER 0° O ÚLTIMO ÂNGULO É APAGADO.
if tr(end,1) == tr(1) + 360
tr = tr(1:end - 1, :);
end
angle = tr(:,1);
Ro = tr(:,2);
```

Figura 72 - Função “[Y1,X1,Ro,theta] = Polar2M(b)”.
Fonte: Autor.

ROTINA: CALIBRA_DISPOSITIVO_AUTOMATICO

```
function [RANGE_HORIZONTAL, RANGE_VERTICAL, RV,
RH]=CALIBRA_DISPOSITIVO_AUTOMATICO(a,cam,INICIO_HCAM,INICIO_VCAM,INICIO_
HMOT,INICIO_VMOT)
%OBJETIVO DA FUNÇÃO:    CALIBRAR O DISPOSITIVO AUTOMATICAMENTE

%PARTE 1: AJUSTE DO LASER N0 PONTO CENTRAL DA IMAGEM E DOS SERVO MOTORES
%ORGANIZAÇÃO INICIAL:

clc;                          %LIMPA A COMMAND WINDOW
close all;                    %FECHA TODAS JANELAS ABERTAS
OK1=0;                        %ÍNDICE CONDICIONADOR PARA O FIM DO AJUSTE
HORIZONTAL
OK2=0;                        %ÍNDICE CONDICIONADOR PARA O FIM DO AJUSTE
VERTICAL
CVC=INICIO_VCAM;              %POSIÇÃO 10° VERTICAL DA WEBCAM
CHC=INICIO_HCAM;              %CENTRO HORIZONTAL DA WEBCAM
CVM=INICIO_VMOT;              %DEFINE O ANGULO INICIAL DO SERVO MOTOR
VERTICAL
CHM=INICIO_HMOT;              %DEFINE O ANGULO INICIAL DO SERVO MOTOR
HORIZONTAL

%FASE 1.1: POSICIONAMENTO INICIAL DOS SERVO MOTORES:

analogWrite(a,7,250);         %CONFIGURA A LUMINOSIDADE DO LASER PARA
50/255
servoWrite(a,8,CHM);          %CONFIGURA O SERVO MOTOR HORIZONTAL DO LASER
PARA
                              %O PONTO CENTRAL 90
servoWrite(a,12,CVM);         %CONFIGURA O SERVO MOTOR VERTICAL DO LASER
PARA
                              %O PONTO CENTRAL 90
servoWrite(a,9,CHC);          %CONFIGURA O SERVO MOTOR HORIZONTAL DA
CÂMERA PARA
                              %O PONTO CENTRAL 90
servoWrite(a,10,CVC);         %CONFIGURA O SERVO MOTOR VERTICAL DA CÂMERA
PARA
                              %O PONTO 10°
pause(1);                     %PAUSA DE 1 SEGUNDO PARA POSICIONAMENTO DOS
SERVOS

while OK1==0

%FASE 1.2: AQUISIÇÃO DA IMAGEM, TRANSFORMAÇÃO PARA ESCALA DE CINZA E
%BINARIZAÇÃO DA IMAGEM:

Im = snapshot(cam);                          %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                             %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f = rgb2gray(Im);                         %PARA ESCALA DE CINZA(01
MATRIZ)
end
f= im2bw(f,.87);                             %BINARIZA A IMAGEM PARA UM
LIMIAR
%FASE 1.3: IDENTIFICAÇÃO DO CONTORNO DO LASER E COORDENADAS DO CENTRO DO
CONTORNO:
```

Continuação...

```
 [B,L,Nb,A] = bwboundaries(f,'noholes');      %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b = B{1};
[YM,XM,Ro,theta] = Polar2(b);                %IDENTIFICA O CENTRO DO
CONTORNO E
                                             %AS COORDENADAS
XM=round(XM);                                %ARREDONDAMENTO DAS
COORDENADAS
YM=round(YM);                                %PARA NÚMERO INTEIRO

%FASE 1.4: APRESENTAÇÃO DA IMAGEM:

figure(1);                                   %ABRE UMA JANELA DE FIGURA
hold on;                                     %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                      %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                   %DIMENSIONA A ESCALA DA
IMAGEM EM X
RI.YWorldLimits = [0 180];                   %DIMENSIONA A ESCALA DA
IMAGEM EM Y
imshow(Im)                                   %MOSTRA A IMAGEM
hold on;                                     %MANTÉM A JANELA EM
UTILIZAÇÃO
MIRA = plot(XM,YM,'x');                      %PLOTA UM X NO CENTRO DO
LASER
set(MIRA,'LineWidth',3,'color','b');         %CONFIGURA O X PLOTADO
hold on;                                     %MANTÉM A JANELA EM
UTILIZAÇÃO

%PLOTA UMA LINHA VERTICAL CENTRAL PARA REFERÊNCIA VISUAL DA POSIÇÃO DO
LASER
%EM RELAÇÃO AO CENTRO DA IMAGEM
CENTRO=plot(size(Im,2)/2,2:(size(Im,1)-
2),'.','LineWidth',2,'color','r');

%FASE 1.5: CALCULO DO ERRO E AJUSTE DA CÂMERA HORIZONTALMENTE:

e=0;                                         %ÍNDICE CONDICIONADOR
ERRO=XM-(size(Im,2)/2);                      %CALCULA O ERRO DO LASER EM
                                             %RELAÇÃO AO CENTRO DA IMAGEM
%APRESENTA O VALOR DO ERRO NA IMAGEM
str={'Erro:',ERRO};
text(600,460,str,'Color','red','FontSize',10);
pause(0.5);

if (ERRO>3)&&(e==0)
    CHC=CHC-1;                               %CORREÇÃO NEGATIVA DO ERRO
    servoWrite(a,9,CHC);                     %MOVIMENTA O LASER
    e=1;                                     %ALTERA O VALOR DO ÍNDICE
end
if (ERRO<-3)&&(e==0)
    CHC=CHC+1;                               %CORREÇÃO POSITIVA DO ERRO
    servoWrite(a,9,CHC);                     %MOVIMENTA O LASER
```

Continuação...

```
    e=1;                                        %ALTERA O VALOR DO ÍNDICE
end

if (ERRO>=-3)&&(e==0)
    if(ERRO<=3)&&(e==0)
        text(325,230,'OK!','Color','green','FontSize',20);



       OK1=1;                                      %ALTERA O VALOR DO ÍNDICE

       e=1;                                        %ALTERA O VALOR DO ÍNDICE

       pause(1);
    end
end
end

%REORGANIZAÇÃO PREPARATÓRIA PARA AJUSTE VERTICAL DA CÂMERA

clc;                          %LIMPA A COMMAND WINDOW

while OK2==0

%REORGANIZAÇÃO

Im = snapshot(cam);                             %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                                %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f = rgb2gray(Im);                            %PARA ESCALA DE CINZA(01
MATRIZ)
end
f= im2bw(f,.87);                                %BINARIZA A IMAGEM PARA UM
LIMIAR

%FASE 1.6: IDENTIFICAÇÃO DO CONTORNO DO LASER E COORDENADAS DO CENTRO DO
CONTORNO:

[B,L,Nb,A] = bwboundaries(f,'noholes');         %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b = B{1};
[YM,XM,Ro,theta] = Polar2(b);                   %IDENTIFICA O CENTRO DO
CONTORNO E
                                                %AS COORDENADAS
XM=round(XM);                                   %ARREDONDAMENTO DAS
COORDENADAS
YM=round(YM);                                   %PARA NÚMERO INTEIRO
%FASE 1.7: APRESENTAÇÃO DA IMAGEM:
figure(1);                                      %ABRE UMA JANELA DE FIGURA
hold on;                                        %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                         %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                      %DIMENSIONA A ESCALA DA
IMAGEM EM X
```

Continuação...

```
RI.YWorldLimits = [0 180];                    %DIMENSIONA A ESCALA DA
IMAGEM EM Y
imshow(Im)                                    %MOSTRA A IMAGEM
hold on;                                      %MANTÉM A JANELA EM
UTILIZAÇÃO
MIRA = plot(XM,YM,'x');                       %PLOTA UM X NO CENTRO DO
LASER
set(MIRA,'LineWidth',3,'color','b');          %CONFIGURA O X PLOTADO
hold on;                                      %MANTÉM A JANELA EM
UTILIZAÇÃO

%PLOTA UMA LINHA HORIZONTAL CENTRAL PARA REFERÊNCIA VISUAL DA POSIÇÃO DO
LASER
%EM RELAÇÃO AO CENTRO DA IMAGEM
CENTRO=plot(2:(size(Im,2)-
2),size(Im,1)/2,'.','LineWidth',2,'color','r');

%FASE 1.8: CALCULO DO ERRO E AJUSTE DA CÂMERA VERTICALMENTE:

e=0;                                          %ÍNDICE CONDICIONADOR
ERRO=YM-(size(Im,1)/2);                       %CALCULA O ERRO DO LASER EM
                                              %RELAÇÃO AO CENTRO DA IMAGEM
str={'Erro:',ERRO};
text(600,460,str,'Color','red','FontSize',10);
pause(0.5);

if (ERRO>5)&&(e==0)
    CVC=CVC-1;                                %CORREÇÃO NEGATIVA DO ERRO
    servoWrite(a,10,CVC);                     %MOVIMENTA O LASER
    e=1;                                      %ALTERA O VALOR DO ÍNDICE
end

if (ERRO<-5)&&(e==0)
    CVC=CVC+1;                                %CORREÇÃO POSITIVA DO ERRO
    servoWrite(a,10,CVC);                     %MOVIMENTA O LASER
    e=1;                                      %ALTERA O VALOR DO ÍNDICE
end

if (ERRO>=-5)&&(e==0)
    if (ERRO<=5)&&(e==0)
        text(325,210,'OK!','Color','green','FontSize',20);
        OK2=1;                                    %ALTERA O VALOR DO
ÍNDICE
        e=1;                                      %ALTERA O VALOR DO
ÍNDICE
        pause(1)
    end
end
end

%AJUSTE FINO DO CENTRO DA IMAGEM, POIS PODEM OCORRER VARIAÇÕES DEVIDO A
%DISTORÇÃO

OK1=0;          %ZERA OS ÍNDICES DE CONTROLE DE FLUXO
OK2=0;
```

Continuação...

```
while OK1==0

%FASE 1.9: AQUISIÇÃO DA IMAGEM, TRANSFORMAÇÃO PARA ESCALA DE CINZA E
%BINARIZAÇÃO DA IMAGEM:

Im = snapshot(cam);                          %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                             %PASSA A IMAGEM DE RGB (03
MATRIZES)

   f = rgb2gray(Im);                         %PARA ESCALA DE CINZA(01
MATRIZ)
end
f= im2bw(f,.87);                             %BINARIZA A IMAGEM PARA UM
LIMIAR

%FASE 1.10: IDENTIFICAÇÃO DO CONTORNO DO LASER E COORDENADAS DO CENTRO
DO CONTORNO:

[B,L,Nb,A] = bwboundaries(f,'noholes');      %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b = B{1};
[YM,XM,Ro,theta] = Polar2(b);                %IDENTIFICA O CENTRO DO
CONTORNO E
                                             %AS COORDENADAS
XM=round(XM);                                %ARREDONDAMENTO DAS
COORDENADAS
YM=round(YM);                                %PARA NÚMERO INTEIRO

%FASE 1.11: APRESENTAÇÃO DA IMAGEM:

figure(1);                                   %ABRE UMA JANELA DE FIGURA
hold on;                                     %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                      %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                   %DIMENSIONA A ESCALA DA
IMAGEM EM X
RI.YWorldLimits = [0 180];                   %DIMENSIONA A ESCALA DA
IMAGEM EM Y
imshow(Im)                                   %MOSTRA A IMAGEM
hold on;                                     %MANTÉM A JANELA EM
UTILIZAÇÃO
MIRA = plot(XM,YM,'x');                      %PLOTA UM X NO CENTRO DO
LASER
set(MIRA,'LineWidth',3,'color','b');         %CONFIGURA O X PLOTADO
hold on;                                     %MANTÉM A JANELA EM
UTILIZAÇÃO
%FASE 1.12: CALCULO DO ERRO E AJUSTE DA CÂMERA HORIZONTALMENTE:

e=0;                                         %ÍNDICE CONDICIONADOR
ERRO=XM-(size(Im,2)/2);                      %CALCULA O ERRO DO LASER EM
                                             %RELAÇÃO AO CENTRO DA IMAGEM
str={'Erro:',ERRO};
text(600,460,str,'Color','red','FontSize',10);
pause(0.5);
```

Continuação...

```
if (ERRO>3)&&(e==0)
    CHC=CHC-1;                                  %CORREÇÃO NEGATIVA DO ERRO
    servoWrite(a,9,CHC);                        %MOVIMENTA O LASER
    e=1;                                        %ALTERA O VALOR DO ÍNDICE
end

if (ERRO<-3)&&(e==0)
    CHC=CHC+1;                                  %CORREÇÃO POSITIVA DO ERRO
    servoWrite(a,9,CHC);                        %MOVIMENTA O LASER
    e=1;                                        %ALTERA O VALOR DO ÍNDICE
end

if (-3<=ERRO<=3)&&(e==0)
    text(325,230,'OK!','Color','green','FontSize',20);
    OK1=1;                                      %ALTERA O VALOR DO ÍNDICE
    e=1;                                        %ALTERA O VALOR DO ÍNDICE
    pause(2);
end
end

%REORGANIZAÇÃO PREPARATÓRIA PARA AJUSTE FINO VERTICAL DA CÂMERA

clc;                          %LIMPA A COMMAND WINDOW

while OK2==0

%REORGANIZAÇÃO

Im = snapshot(cam);                             %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                                %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f = rgb2gray(Im);                            %PARA ESCALA DE CINZA(01
MATRIZ)
end
f= im2bw(f,.87);                                %BINARIZA A IMAGEM PARA UM
LIMIAR

%FASE 1.13: IDENTIFICAÇÃO DO CONTORNO DO LASER E COORDENADAS DO CENTRO
DO CONTORNO:

[B,L,Nb,A] = bwboundaries(f,'noholes');     %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b = B{1};
[YM,XM,Ro,theta] = Polar2(b);               %IDENTIFICA O CENTRO DO
CONTORNO E
                                            %AS COORDENADAS
XM=round(XM);                               %ARREDONDAMENTO DAS
COORDENADAS
YM=round(YM);                               %PARA NÚMERO INTEIRO
```

Continuação...

```
%FASE 1.14: APRESENTAÇÃO DA IMAGEM:

figure(1);                                          %ABRE UMA JANELA DE FIGURA
hold on;                                            %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                             %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                          %DIMENSIONA A ESCALA DA
IMAGEM EM X
RI.YWorldLimits = [0 180];                          %DIMENSIONA A ESCALA DA
IMAGEM EM Y
imshow(Im)                                          %MOSTRA A IMAGEM
hold on;                                            %MANTÉM A JANELA EM
UTILIZAÇÃO
MIRA = plot(XM,YM,'x');                             %PLOTA UM X NO CENTRO DO
LASER
set(MIRA,'LineWidth',3,'color','b');                %CONFIGURA O X PLOTADO
hold on;                                            %MANTÉM A JANELA EM
UTILIZAÇÃO

%PLOTA UMA LINHA HORIZONTAL CENTRAL PARA REFERÊNCIA VISUAL DA POSIÇÃO DO
LASER
%EM RELAÇÃO AO CENTRO DA IMAGEM
CENTRO=plot(2:(size(Im,2)-
2),size(Im,1)/2,'.','LineWidth',2,'color','r');

%FASE 1.15: CALCULO DO ERRO E AJUSTE DA CÂMERA VERTICALMENTE:

e=0;                                                %ÍNDICE CONDICIONADOR
ERRO=YM-(size(Im,1)/2);                             %CALCULA O ERRO DO LASER EM
                                                    %RELAÇÃO AO CENTRO DA IMAGEM
str={'Erro:',ERRO};
text(600,460,str,'Color','red','FontSize',10);
pause(0.5);

if (ERRO>4)&&(e==0)
    CVC=CVC-1;                                      %CORREÇÃO NEGATIVA DO ERRO
    servoWrite(a,10,CVC);                           %MOVIMENTA O LASER
    e=1;                                            %ALTERA O VALOR DO ÍNDICE
end

if (ERRO<-4)&&(e==0)
    CVC=CVC+1;                                      %CORREÇÃO POSITIVA DO ERRO
    servoWrite(a,10,CVC);                           %MOVIMENTA O LASER
    e=1;                                            %ALTERA O VALOR DO ÍNDICE
end
if (-4<=ERRO<=4)&&(e==0)
    text(325,210,'OK!','Color','green','FontSize',20);
    OK2=1;                                          %ALTERA O VALOR DO ÍNDICE
    e=1;                                            %ALTERA O VALOR DO ÍNDICE
    pause(2)
end
end
%APRESENTAÇÃO AO USUÁRIO:

analogWrite(a,7,0);                                 %DESLIGA O LASER
pause(0.2);                                         %PAUSA PARA
```

Continuação...

```
ESTABILIZAÇÃO DA IMAGEM
Im = snapshot(cam);                                    %AQUISIÇÃO DA IMAGEM
analogWrite(a,7,250);                                  %LIGA O LASER
figure(1);                                             %ABRE UMA JANELA DE
FIGURA
hold on;                                               %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                                %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                             %DIMENSIONA A ESCALA DA
IMAGEM EM X
RI.YWorldLimits = [0 180];                             %DIMENSIONA A ESCALA DA
IMAGEM EM Y
imshow(Im)                                             %MOSTRA A IMAGEM
hold on;                                               %MANTÉM A JANELA EM

UTILIZAÇÃO
str={'Câmera ajustada e Laser centrado'};              %ESCREVE TEXTO NA IMAGEM
text(10,20,str,'Color','green','FontSize',10);         %CONFIGURA COORDENADAS
DO TEXTO NA IMAGEM
pause(2);                                              %TEMPO DE LEITURA DO
USUÁRIO

%-------------------------------------------------------------------------
--%
%PARTE 2: CALIBRAÇÃO DOS PONTOS EXTREMOS DO LASER NA IMAGEM

%APRESENTAÇÃO AO USUÁRIO:

analogWrite(a,7,0);                                    %DESLIGA O LASER
pause(0.2);                                            %PAUSA PARA
ESTABILIZAÇÃO DA IMAGEM
Im = snapshot(cam);                                    %AQUISIÇÃO DA IMAGEM
analogWrite(a,7,50);                                   %LIGA O LASER
Im = snapshot(cam);                                    %AQUISIÇÃO DA IMAGEM
figure(1);                                             %ABRE UMA JANELA DE
FIGURA
hold on;                                               %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                                %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                             %DIMENSIONA A ESCALA DA
IMAGEM EM X
RI.YWorldLimits = [0 180];                             %DIMENSIONA A ESCALA DA
IMAGEM EM Y
imshow(Im)                                             %MOSTRA A IMAGEM
hold on;                                               %MANTÉM A JANELA EM
UTILIZAÇÃO
str={'ADQUIRINDO DADOS DE MOVIMENTAÇÃO...'};           %ESCREVE TEXTO NA IMAGEM
text(10,20,str,'Color','red','FontSize',10);           %CONFIGURA COORDENADAS
DO TEXTO NA IMAGEM
pause(2);                                              %TEMPO PARA LEITURA DO
USUÁRIO
%ORGANIZAÇÃO INICIAL:
clc;                          %LIMPA A COMMAND WINDOW
OK2=0;                        %ÍNDICE CONDICIONADOR PARA O FIM DO AJUSTE
```

Continuação...

```
%FASE 2.1: POSICIONAMENTO INICIAL DOS SERVO MOTORES:

analogWrite(a,7,250);        %CONFIGURA A LUMINOSIDADE DO LASER PARA
50/255
servoWrite(a,8,CHM);         %CONFIGURA O SERVO MOTOR HORIZONTAL DO LASER
PARA
                             %O PONTO CENTRAL 90
servoWrite(a,12,CVM);        %CONFIGURA O SERVO MOTOR VERTICAL DO LASER
PARA
                             %O PONTO CENTRAL 90
servoWrite(a,9,CHC);         %CONFIGURA O SERVO MOTOR HORIZONTAL DA
CÂMERA PARA
                             %O PONTO CENTRAL 90
servoWrite(a,10,CVC);        %CONFIGURA O SERVO MOTOR VERTICAL DA CÂMERA
PARA
                             %O PONTO 10°
pause(0.5);                  %PAUSA PARA POSICIONAMENTO DOS SERVOS

%FASE 2.2: AQUISIÇÃO DA IMAGEM, TRANSFORMAÇÃO PARA ESCALA DE CINZA,
%BINARIZAÇÃO DA IMAGEM E OBTENÇÃO DAS DIMENSÕES DA IMAGEM:

Im = snapshot(cam);                          %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                             %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f = rgb2gray(Im);                         %PARA ESCALA DE CINZA(01
MATRIZ)
end
f= im2bw(f,.87);                             %BINARIZA A IMAGEM PARA UM
LIMIAR
[Imy Imx]=size(f);                           %CARREGA OS VALORES DA
DIMENSÃO
                                             %ESPACIAL DA IMAGEM NAS
                                             %VARIÁVEIS Imx(LARGURA) E
                                             %Imy(ALTURA)
%FASE 2.3: IDENTIFICAÇÃO DO CONTORNO DO LASER E COORDENADAS DO CENTRO DO
CONTORNO:

[B,L,Nb,A] = bwboundaries(f,'noholes');      %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b = B{1};
[YM,XM,Ro,theta] = Polar2(b);                %IDENTIFICA O CENTRO DO
CONTORNO E
                                             %AS COORDENADAS
XM=round(XM);                                %ARREDONDAMENTO DAS
COORDENADAS
YM=round(YM);                                %PARA NÚMERO INTEIRO
%FASE 2.4: CÁLCULO ESTIMADO DA RESOLUÇÃO DO PASSO HORIZONTAL E VERTICAL
DO LASER:
%CÁLCULO ESTIMADO DA RESOLUÇÃO HORIZONTAL
servoWrite(a,8,100);                         %MOVIMENTA O SERVO
HORIZONTAL
                                             %10° EM RELAÇÃO AO CENTRO
pause(1);                                    %PAUSA DE 1 SEGUNDO PARA
POSICIONAMENTO
```

Continuação...

```
                                                %DOS SERVOS
Im2=snapshot(cam);                              %CAPTURA A IMAGEM REFERENTE
AO MOVIMENTO
if size(Im2,3)==3                               %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f2 = rgb2gray(Im2);                          %PARA ESCALA DE CINZA(01
MATRIZ)
end
f2= im2bw(f2,.87);                              %BINARIZA A IMAGEM PARA UM
LIMIAR
[B,L,Nb,A] = bwboundaries(f2,'noholes');        %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b=B{1};
[YM2,XM2,Ro2,theta2] = Polar2(b);               %IDENTIFICA O CENTRO DO
CONTORNO E
                                                %AS COORDENADAS

XM2=round(XM2);                                 %ARREDONDAMENTO DAS
COORDENADAS
YM2=round(YM2);                                 %PARA NÚMERO INTEIRO

servoWrite(a,8,80);                             %MOVIMENTA O SERVO
HORIZONTAL
                                                %-10° EM RELAÇÃO AO CENTRO
pause(1);                                       %PAUSA DE 1 SEGUNDO PARA
POSICIONAMENTO
                                                %DOS SERVOS


Im3=snapshot(cam);                              %CAPTURA A IMAGEM REFERENTE
AO MOVIMENTO
if size(Im3,3)==3                               %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f3 = rgb2gray(Im3);                          %PARA ESCALA DE CINZA(01
MATRIZ)
end
f3= im2bw(f3,.87);                              %BINARIZA A IMAGEM PARA UM
LIMIAR
[B,L,Nb,A] = bwboundaries(f3,'noholes');        %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b=B{1};
[YM3,XM3,Ro3,theta3] = Polar2(b);               %IDENTIFICA O CENTRO DO
CONTORNO E
                                                %AS COORDENADAS


XM3=round(XM3);                                 %ARREDONDAMENTO DAS
COORDENADAS
YM3=round(YM3);                                 %PARA NÚMERO INTEIRO
RH=(XM3-XM2)/20;                                %RH = RESOLUÇÃO HORIZONTAL =
NÚMERO DE
                                                %PIXELS/PASSO
servoWrite(a,8,90);                             %RETORNA O SERVO HORIZONTAL
PARA
                                                %A POSIÇÃO CENTRAL
pause(1);                                       %PAUSA DE 1 SEGUNDO PARA
POSICIONAMENTO
```

Continuação...

```
%DOS SERVOS

%APRESENTAÇÃO AO USUÁRIO

str={'RESOLUÇÃO HORIZONTAL...OK'};              %ESCREVE TEXTO NA IMAGEM
text(10,40,str,'Color','green','FontSize',10);  %CONFIGURA COORDENADAS
DO TEXTO NA IMAGEM

%CÁLCULO ESTIMADO DA RESOLUÇÃO VERTICAL
servoWrite(a,12,100);                           %MOVIMENTA O SERVO VERTICAL
                                                %10° EM RELAÇÃO AO CENTRO
pause(1);                                       %PAUSA DE 1 SEGUNDO PARA
POSICIONAMENTO
                                                %DOS SERVOS
Im2=snapshot(cam);                              %CAPTURA A IMAGEM REFERENTE
AO MOVIMENTO
if size(Im2,3)==3                               %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f2 = rgb2gray(Im2);                          %PARA ESCALA DE CINZA(01
MATRIZ)
end
f2= im2bw(f2,.87);                              %BINARIZA A IMAGEM PARA UM
LIMIAR
[B,L,Nb,A] = bwboundaries(f2,'noholes');        %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b=B{1};
[YM2,XM2,Ro2,theta2] = Polar2(b);               %IDENTIFICA O CENTRO DO
CONTORNO E
                                                %AS COORDENADAS
XM2=round(XM2);                                 %ARREDONDAMENTO DAS
COORDENADAS
YM2=round(YM2);                                 %PARA NÚMERO INTEIRO


servoWrite(a,12,80);                            %MOVIMENTA O SERVO
HORIZONTAL
                                                %-10° EM RELAÇÃO AO CENTRO
pause(1);                                       %PAUSA DE 1 SEGUNDO PARA
POSICIONAMENTO
                                                %DOS SERVOS
Im3=snapshot(cam);                              %CAPTURA A IMAGEM REFERENTE
AO MOVIMENTO
if size(Im3,3)==3                               %PASSA A IMAGEM DE RGB (03
MATRIZES)
   f3 = rgb2gray(Im3);                          %PARA ESCALA DE CINZA(01
MATRIZ)
end
f3= im2bw(f3,.87);                              %BINARIZA A IMAGEM PARA UM
LIMIAR
[B,L,Nb,A] = bwboundaries(f3,'noholes');        %IDENTIFICA O CONTORNO DO
LASER
h=size(B);
b=B{1};
```

Continuação...

```
 [YM3,XM3,Ro3,theta3] = Polar2(b);            %IDENTIFICA O CENTRO DO
CONTORNO E
                                             %AS COORDENADAS
XM3=round(XM3);                              %ARREDONDAMENTO DAS
COORDENADAS
YM3=round(YM3);                              %PARA NÚMERO INTEIRO

RV=(YM3-YM2)/20;                             %RV = RESOLUÇÃO VERTICAL =
NÚMERO DE
                                             %PIXELS/PASSO NA VERTICAL
servoWrite(a,12,90);                         %RETORNA O SERVO VERTICAL
PARA
                                             %A POSIÇÃO CENTRAL
pause(1);                                    %PAUSA DE 1 SEGUNDO PARA
                                              %POSICIONAMENTO DOS SERVOS


%APRESENTAÇÃO AO USUÁRIO

str={'RESOLUÇÃO VERTICAL...OK'};                %ESCREVE TEXTO NA IMAGEM
text(10,60,str,'Color','green','FontSize',10);  %CONFIGURA COORDENADAS
DO TEXTO NA IMAGEM
%FASE 2.5: CÁLCULO ESTIMADO DO RANGE DE PASSOS HORIZONTAIS E VERTICAIS
DO LASER:

RANGE_HORIZONTAL=round(Imx/RH);
RANGE_VERTICAL=round(Imy/RV);

%APRESENTAÇÃO AO USUÁRIO

str={'RANGE HORIZONTAL...OK'};                  %ESCREVE TEXTO NA IMAGEM
text(10,80,str,'Color','green','FontSize',10);  %CONFIGURA COORDENADAS
DO TEXTO NA IMAGEM
str={'RANGE VERTICAL...OK'};                    %ESCREVE TEXTO NA IMAGEM
text(10,100,str,'Color','green','FontSize',10); %CONFIGURA COORDENADAS
DO TEXTO NA IMAGEM
pause(2);                                       %TEMPO DE LEITURA DO
USUÁRIO

%FASE 2.6: APRESENTA NA TELA OS VALORES EXPRESSIVOS:

analogWrite(a,7,0);                             %DESLIGA O LASER
pause(0.2);                                     %PAUSA PARA
ESTABILIZAÇÃO DA IMAGEM
Im = snapshot(cam);                             %AQUISIÇÃO DA IMAGEM
analogWrite(a,7,250);                           %LIGA O LASER
Im = snapshot(cam);                             %AQUISIÇÃO DA IMAGEM
figure(1);                                      %ABRE UMA JANELA DE
FIGURA
hold on;                                        %MANTÉM A JANELA EM
UTILIZAÇÃO
RI = imref2d(size(Im));                         %DIMENSIONA A ESCALA DA
IMAGEM
RI.XWorldLimits = [0 180];                      %DIMENSIONA A ESCALA DA
IMAGEM EM X
RI.YWorldLimits = [0 180];                      %DIMENSIONA A ESCALA DA
IMAGEM EM Y
```

Continuação...

```
imshow(Im)                                          %MOSTRA A IMAGEM
hold on;                                            %MANTÉM A JANELA EM
UTILIZAÇÃO
%APRESENTA OS VALORES NA TELA
str={'Range Horizontal:',RANGE_HORIZONTAL};
text(10,20,str,'Color','red','FontSize',10);
str={'Range Vertical:',RANGE_VERTICAL};
text(10,55,str,'Color','red','FontSize',10);
str={'Resolução Horizontal: (pixels/passo)',RH};
text(10,90,str,'Color','red','FontSize',10);
str={'Resolução Vertical: (pixels/passo)',RV};
text(10,125,str,'Color','red','FontSize',10);
pause(5);                                           %TEMPO PARA LEITURA DO
USUÁRIO
close all;
clc;
```

Figura 73 - Rotina "CALIBRA_DISPOSITIVO_AUTOMATICO".
Fonte: Autor.

ROTINA: MAIN_TCC_2

```
%TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO
%
%ROBÔ SEGUIDOR DE ALVOS MÓVEIS COM MIRA A LASER AUTOMÁTICA
%
%ALUNO: MÁRLON AFONSO BORGES DAMASCENO
%PROFESSOR ORIENTADOR: SÉRGIO LUIZ DA SILVA PITHAN
%CURSO: ENGENHARIA DE AUTOMAÇÃO INDUSTRIAL
%
%ROTINA DE PROGRAMA PRINCIPAL
clear all;      %LIMPA A WORKSPACE
close all;      %FECHA TODAS AS JANELAS ABERTAS
clc;            %LIMPA A COMMAND WINDOW
%CONFIGURAÇÕES INICIAIS
[a, cam]=INICIALIZA_TCC;

%CALIBRA AUTOMATICAMENTE O DISPOSITIVO
[RANGE_HORIZONTAL, RANGE_VERTICAL, RV,
RH]=CALIBRA_DISPOSITIVO_AUTOMATICO(a,cam,90,20,90,90);

%CALCULA OS LIMITES DA IMAGEM CONFORME A CALIBRAÇÃO EXECUTADA
[LEH, LDH, LSV, LIV]=CALCULA_LIMITES(RANGE_HORIZONTAL, RANGE_VERTICAL,
RH, RV);

%IDENTIFICA O ALVO
[Y1,X1]=ALVO(a,cam);

%CONVERTE A POSIÇÃO DO ALVO PARA A ESCALA DO ROBÔ
MOVX=round(-(X1/RH)+LEH);
MOVY=round(-(Y1/RV)+LSV);

ex=0;
ey=0;
```

Continuação...

```
for TEMPO=1:50;
%IDENTIFICA O ALVO
[Y1,X1]=ALVO(a,cam);

%CALCULA O MÓDULO DA POSIÇÃO DO ALVO
MODULO_ALVO=round(sqrt((X1^2)+(Y1^2)));

%IDENTIFICA A MIRA
[YM,XM,RESET]=MIRA(a,cam);

%CALCULA O MÓDULO DA POSIÇÃO DA MIRA
MODULO_MIRA=round(sqrt((XM^2)+(YM^2)));

%CALCULA OS ERROS
ERRO_X=XM-X1;
ERRO_Y=YM-Y1;
ERRO=round(sqrt((ERRO_X^2)+(ERRO_Y^2)));

%APRESENTA O VALOR DO ERRO NA TELA
hold on
str={'Erro:',ERRO};
text(600,425,str,'Color','red','FontSize',10);
%MOVIMENTA OS SERVO MOTORES
[MOVX, MOVY] = MOVIMENTA_SERVOS_2(a,MOVX,MOVY,ERRO_X,ERRO_Y);
    if RESET==1
        %IDENTIFICA O ALVO
        [Y1,X1]=ALVO(a,cam);
        %CONVERTE A POSIÇÃO DO ALVO PARA A ESCALA DO ROBÔ
        MOVX=round(-(X1/RH)+LEH);
        MOVY=round(-(Y1/RV)+LSV);
        RESET=0;
        %IDENTIFICA A MIRA
        [YM,XM,RESET]=MIRA(a,cam);
    end

%CONFIGURA TABELAS DE RESULTADOS
TAB_ERRO(1,TEMPO)=ERRO;                     %TABELA DE ERRO
TAB_MODULO_ALVO(1,TEMPO)=MODULO_ALVO;       %TABELA DE SET-POINT
TAB_MODULO_MIRA(1,TEMPO)=MODULO_MIRA;       %TABELA DE VARIÁVEL DE
PROCESSO
end
%CONFIGURA E APRESENTA UMA JANELA COM O GRÁFICO
figure(3)
grid on
hold on
plot(TAB_ERRO,':','LineWidth',1,'color','red');
plot(TAB_MODULO_ALVO,'LineWidth',2,'color','blue');
plot(TAB_MODULO_MIRA,'-.','LineWidth',1.5,'color','m');
legend('ERRO','ALVO: SET POINT','MIRA: PV');
legend('Location','eastoutside');
title('Resultados - Seguidor de Objetos');
xlabel('Tempo de Amostragem');
ylabel('Valores em pixels');
```

Figura 74 - Rotina principal "MAIN_TCC_2".
Fonte: Autor.

ROTINA: MULTI_ALVO

```
%FUNÇÃO PARA DEFINIR O OBJETO COMO ALVO E
%DETECTAR O CENTRO DO ALVO SELECIONADO
%RETORNA AS COORDENADAS X1 E Y1 DO CENTRO DO ALVO


function [Y1,X1,OBJETO]=MULTI_ALVO(a,cam)
figure(1)                          %ABRE A JANELA DA FIGURA 1
hold on;                           %MANTÉM A JANELA EM UTILIZAÇÃO
Im = snapshot(cam);                %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                   %PASSA A IMAGEM DE RGB (03 MATRIZES)
f = rgb2gray(Im);                %PARA ESCALA DE CINZA(01 MATRIZ)
end
f= imcomplement(f);                %ENCONTRA O NEGATIVO DA IMAGEM
f= im2bw(f,.82);                   %BINARIZA A IMAGEM PARA UM LIMIAR

%IDENTIFICA O CONTORNO DOS OBJETOS
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                         %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                        %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
ALVO)
    imshow(Im);                    %MOSTRA A IMAGEM INICIAL
    hold on;                       %MANTÉM A JANELA EM UTILIZAÇÃO
    if Nb==1                       %CONTROLE DE FLUXO PARA VERIFICAR
                                   %SE HÁ APENAS 01 OBJETO IDENTIFICADO
        OBJETO=1;                  %MARCA O ÚNICO OBJETO IDENTIFICADO
          b = B{OBJETO};             %SELECIONA O ARRANJO DO CONTORNO DO
                                   %OBJETO ALVO
        [Y1,X1,Ro,theta] = Polar2(b);    %RETORNA O CENTRO DO CONTORNO
                                         %ATRAVÉS DE COORDENADAS POLARES
    end

if Nb>1                          %CONTROLE DE FLUXO PARA QUANDO EXISTIR MAIS
                                   %DE 01 OBJETO IDENTIFICADO
        str={'SELECIONAR O ALVO'};  %ESCREVE NA TELA A FRASE
        text(300,20,str,'Color','y','FontSize',10); %POSICIONA A ESCRITA
 %LOOP PARA MARCAR TODOS OS OBJETOS IDENTIFICADOS CONFORME DUA POSIÇÃO
 %NA MATRIZ B
        for OBJETO=1:Nb
            b = B{OBJETO};
            [Y1,X1,Ro,theta] = Polar2(b);
            str={'N°',OBJETO};
            text(X1,Y1,str,'Color','y','FontSize',10);
            hold on
        end
        str='DIGITE O VALOR DO OBJETO PARA TRANSFORMÁ-LO EM ALVO: ';
        hold on;
        OBJETO=input(str);
        hold off
    end
    b = B{OBJETO};                   %SELECIONA QUAL CONTORNO PARA
PROCESSAMENTO
    [Y1,X1,Ro,theta] = Polar2(b);    %RETORNA O CENTRO DO CONTORNO
ATRAVÉS DE
                                     %COORDENADAS POLARES
    %[Y1,X1,Ro,theta] = Polar2M(b); %RETORNA O CENTRO DO CONTORNO
ATRAVÉS DE
```

Continuação...

```
                                    %COORDENADAS POLARES COMPENSADAS DE
                                    %DISTORÇÃO
    X1=round(X1);                   %ARREDONDAMENTO PARA NÚMERO INTEIRO
    Y1=round(Y1);                   %ARREDONDAMENTO PARA NÚMERO INTEIRO
    figure(1);                      %ABRE A JANELA DA FIGURA 1
    hold on;                        %MANTÉM A JANELA EM UTILIZAÇÃO
    imshow(Im)                      %MOSTRA A IMAGEM INICIAL
    hold on;                        %MANTÉM A JANELA EM UTILIZAÇÃO
    Im = plot(X1,Y1,'X');           %PLOTA UM X NO CENTRO DO ALVO
%CONFIGURA O MARCADOR DO CENTRO DO ALVO
    set(Im,'LineWidth',3,'color','r');
    hold on;                        %MANTÉM A JANELA EM UTILIZAÇÃO
%CASO O TAMANHO DO ARRANJO B=0, SIGNIFICA QUE NÃO HOUVE IDENTIFICAÇÃO
%DE NENHUM CONTORNO. CASO ACONTEÇA, ESCREVE MENSAGEM NA TELA E
%DÁ PROSSEGUIMENTO À ROTINA DE PROGRAMAÇÃO
else
    imshow(Im);                 %MOSTRA A IMAGEM INICIAL
%ESCREVE MENSAGEM NA IMAGEM
    text(20,20,'ALVO NÃO DETECTADO','Color','red','FontSize',10);
    X1=320;                     %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    Y1=240;                     %DEFINE OS PONTOS CENTRAIS DA IMAGEM
     servoWrite(a,8,90);         %REPOSICIONA O SERVO MOTOR HORIZONTAL
     servoWrite(a,12,90);        %REPOSICIONA O SERVO MOTOR VERTICAL
     pause(1);                   %PAUSA PARA ESTABILIZAÇÃO DOS SERVO
MOTORES
end
```

Figura 75 - Rotina "[Y1,X1,OBJETO]=MULTI_ALVO(a,cam)".
Fonte: Autor.

ROTINA: invmoments

```
function phi = invmoments(F)
%INVMOMENTS CALCULA OS MOMENTOS INVARIANTES DA IMAGEM.
%   PHI = INVMOMENTS(F) CALCULA OS MOMENTOS INVARIANTES DA IMAGEM
%   F. PHI É UM VETOR COLUNA DE SETE ELEMENTOS CONTENDO OS MOMENTOS
%   INVARIANTES.
%   A MATRIZ DEVE SER BIDIMENSIONAL, NÃO ESPARSA, LÓGICA OU NUMÉRICA.

%CONTROLE DE FLUXO PARA ANÁLISE DOS REQUISITOS DA MATRIZ
%PARA O CÁLCULO DOS MOMENTOS INVARIANTES
if (ndims(F) ~= 2) || issparse(F) || ~isreal(F) || ...
        ~(isnumeric(F) || islogical(F))
   error(['F must be a 2-D, real, nonsparse, numeric or logical' ...
          'matrix.']);
end

F = double(F);

phi = compute_phi(compute_eta(compute_m(F)));

%--------------------------------------------------------------------%
```

Continuação...

```
function m = compute_m(F)

[M, N] = size(F);
[x, y] = meshgrid(1:N, 1:M);

%TRANSFORMA X, Y E F EM VETORES COLUNA PARA FACILITAR OS CALCULOS
x = x(:);
y = y(:);
F = F(:);

%MOMENTO 2D (BIDIMENSIONAL)
m.m00 = sum(F);
%CONTROLE PARA EVITAR DIVISÃO POR ZERO
if (m.m00 == 0)
   m.m00 = eps;
end
%OS OUTROS MOMENTOS CENTRAIS
m.m10 = sum(x .* F);
m.m01 = sum(y .* F);
m.m11 = sum(x .* y .* F);
m.m20 = sum(x.^2 .* F);
m.m02 = sum(y.^2 .* F);
m.m30 = sum(x.^3 .* F);
m.m03 = sum(y.^3 .* F);
m.m12 = sum(x .* y.^2 .* F);
m.m21 = sum(x.^2 .* y .* F);

%--------------------------------------------------------------------%
function e = compute_eta(m)

%MOMENTOS CENTRAIS NORMALIZADOS

xbar = m.m10 / m.m00;
ybar = m.m01 / m.m00;

e.eta11 = (m.m11 - ybar*m.m10) / m.m00^2;
e.eta20 = (m.m20 - xbar*m.m10) / m.m00^2;
e.eta02 = (m.m02 - ybar*m.m01) / m.m00^2;
e.eta30 = (m.m30 - 3 * xbar * m.m20 + 2 * xbar^2 * m.m10) / ...
           m.m00^2.5;
e.eta03 = (m.m03 - 3 * ybar * m.m02 + 2 * ybar^2 * m.m01) / ...
           m.m00^2.5;
e.eta21 = (m.m21 - 2 * xbar * m.m11 - ybar * m.m20 + ...
           2 * xbar^2 * m.m01) / m.m00^2.5;
e.eta12 = (m.m12 - 2 * ybar * m.m11 - xbar * m.m02 + ...
           2 * ybar^2 * m.m10) / m.m00^2.5;

%--------------------------------------------------------------------%


function phi = compute_phi(e)
```

Continuação...

```
%CÁLCULO DOS 7 MOMENTOS INVARIANTES

phi(1) = e.eta20 + e.eta02;
phi(2) = (e.eta20 - e.eta02)^2 + 4*e.eta11^2;
phi(3) = (e.eta30 - 3*e.eta12)^2 + (3*e.eta21 - e.eta03)^2;
phi(4) = (e.eta30 + e.eta12)^2 + (e.eta21 + e.eta03)^2;
phi(5) = (e.eta30 - 3*e.eta12)*(e.eta30 + e.eta12)* ...
         ((e.eta30 + e.eta12)^2 - 3*(e.eta21 + e.eta03)^2 ) + ...
         (3*e.eta21 - e.eta03)*(e.eta21 + e.eta03) * ...
         (3*(e.eta30 + e.eta12)^2 - (e.eta21 + e.eta03)^2 );
phi(6) = (e.eta20 - e.eta02) * ((e.eta30 + e.eta12)^2 - ...
         (e.eta21 + e.eta03)^2 )+ ...
        4 * e.eta11*(e.eta30 + e.eta12)*(e.eta21 + e.eta03);
phi(7) = (3*e.eta21 - e.eta03)*(e.eta30 + e.eta12)*...
         ((e.eta30 + e.eta12)^2 - 3*(e.eta21 + e.eta03)^2 ) + ...
         (3*e.eta12 - e.eta30)*(e.eta21 + e.eta03)* ...
         (3*(e.eta30 + e.eta12)^2 - (e.eta21 + e.eta03)^2 );
```

Figura 76 - Rotina "invmoments".
Fonte: Baseado em GONZALEZ, WOODS, 2010.

ROTINA: MomentosInvariantes

```
function [Mi]=MomentosInvariantes(f)
I = invmoments(f);
Mi = -sign(I).*(log10(abs(I)));
%figure,imshow(f);
%title({{['Imagem ( ',(name),') ']}});
%xlabel({{[' Momentos Invariantes = (',num2str(Mi),') ']}})
```

Figura 77 - Rotina "MomentosInvariantes".
Fonte: PITHAN.

ROTINA: MULTI_ALVO2

```
%FUNÇÃO PARA DEFINIR O OBJETO COMO ALVO E
%DETECTAR O CENTRO DO ALVO SELECIONADO
%RETORNA AS COORDENADAS X1 E Y1 DO CENTRO DO ALVO
%RETORNA O VALOR MI DOS MOMENTOS INVARIANTES DO OBJETO SELECIONADO
function [Y1,X1,MI]=MULTI_ALVO2(a,cam)
close all
figure(1)                          %ABRE A JANELA DA FIGURA 1
hold on;                           %MANTÉM A JANELA EM UTILIZAÇÃO
Im = snapshot(cam);                %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                   %PASSA A IMAGEM DE RGB (03 MATRIZES)
   f = rgb2gray(Im);               %PARA ESCALA DE CINZA(01 MATRIZ)
end
f= imcomplement(f);                %ENCONTRA O NEGATIVO DA IMAGEM
f= im2bw(f,.82);                   %BINARIZA A IMAGEM PARA UM LIMIAR
%IDENTIFICA O CONTORNO DOS OBJETOS
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                         %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                        %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
                                   %ALVO)
    imshow(Im);                    %MOSTRA A IMAGEM INICIAL
    hold on;                       %MANTÉM A JANELA EM UTILIZAÇÃO
    if Nb==1                       %CONTROLE DE FLUXO PARA VERIFICAR
                                   %SE HÁ APENAS 01 OBJETO IDENTIFICADO
        OBJETO=1;                  %MARCA O ÚNICO OBJETO IDENTIFICADO
        b = B{OBJETO};             %SELECIONA O ARRANJO DO CONTORNO DO
                                   %OBJETO ALVO
        [Y1,X1,Ro,theta] = Polar2(b);   %RETORNA O CENTRO DO CONTORNO
                                        %ATRAVÉS DE COORDENADAS
                                        %POLARES
 %RECORTA O CONTORNO DO OBJETO DA IMAGEM ORIGINAL
        SLC{OBJETO}=imcrop(f,[(X1-(length(b)/2))...
            (Y1-(length(b)/2)) length(b) length(b)]);
 %COLOCA UMA MOLDURA PADRÃO NO OBJETO
        SLC{OBJETO}=padarray(SLC{OBJETO},[50 50],'both');
 %EXECUTA O CÁLCULO DOS MOMENTOS INVARIANTES PARA O OBJETO
        MI{OBJETO}=MomentosInvariantes(SLC{OBJETO});
    end
if Nb>1                        %CONTROLE DE FLUXO PARA QUANDO EXISTIR
                               %MAIS DE 01 OBJETO IDENTIFICADO
        str={'SELECIONAR O ALVO'};          %ESCREVE NA IMAGEM
        text(300,20,str,'Color','y','FontSize',10); %POSICIONA ESCRITA
 %LOOP PARA MARCAR TODOS OS OBJETOS IDENTIFICADOS CONFORME DUA POSIÇÃO
 %NA MATRIZ B
for OBJETO=1:Nb
            b = B{OBJETO};                  %SELECIONA CONTORNO
            [Y1,X1,Ro,theta] = Polar2(b);   %IDENTIFICA AS
                                            %COORDENADAS
%CORTA DE ACORDO COM O COMPRIMENTO E SEPARA DO NEGATIVO DA IMAGEM
 %CADA CONTORNO IDENTIFICADO E O ARMAZENA NO ARRANJO SLC
            SLC{OBJETO}=imcrop(f,[(X1-(length(b)/2))...
                (Y1-(length(b)/2)) length(b) length(b)]);
%COLOCA UMA MOLDURA PADRÃO NA PERIFERIA DE CADA OBJETO RECORTADO DA
 %IMAGEM
SLC{OBJETO}=padarray(SLC{OBJETO},[50 50],'both');
%EXECUTA O CÁLCULO DOS MOMENTOS INVARIANTES PARA CADA OBJETO
            MI{OBJETO}=MomentosInvariantes(SLC{OBJETO});
```

Continuação...

```
            str={'N°',OBJETO};
            text(X1,Y1,str,'Color','y','FontSize',10);
            hold on
        end
        str='DIGITE O VALOR DO OBJETO PARA TRANSFORMÁ-LO EM ALVO: ';
        hold on;
        OBJETO=input(str);          %AGUARDA A SELEÇÃO DO OBJETO
                                    %DESEJADO
        hold off
    end
    b = B{OBJETO};                  %SELECIONA QUAL CONTORNO PARA
                                    %PROCESSAMENTO
    [Y1,X1,Ro,theta] = Polar2(b);   %RETORNA O CENTRO DO CONTORNO
                                    %ATRAVÉS DE COORDENADAS POLARES
%CORTA E SEPARA DO NEGATIVO DA IMAGEM O ALVO SELECIONADO
    SLC{OBJETO}=imcrop(f,[(X1-(length(b)/2))...
        (Y1-(length(b)/2)) length(b) length(b)]);
 %COLOCA UMA MOLDURA PADRÃO NA PERIFERIA DO ALVO RECORTADO DA
 %IMAGEM
    SLC{OBJETO}=padarray(SLC{OBJETO},[50 50],'both');
%EXECUTA O CÁLCULO DOS MOMENTOS INVARIANTES PARA O ALVO
    MI{OBJETO}=MomentosInvariantes(SLC{OBJETO});
%ARMAZENA O VALOR NO ARGUMENTO DE SAÍDA
    MI=MI{OBJETO};
    X1=round(X1);           %ARREDONDAMENTO PARA NÚMERO INTEIRO
    Y1=round(Y1);           %ARREDONDAMENTO PARA NÚMERO INTEIRO
    figure(1);              %ABRE A JANELA DA FIGURA 1
    hold on;                %MANTÉM A JANELA EM UTILIZAÇÃO
    imshow(Im)              %MOSTRA A IMAGEM INICIAL
    hold on;                %MANTÉM A JANELA EM UTILIZAÇÃO
    Im = plot(X1,Y1,'X');   %PLOTA UM X NO CENTRO DO ALVO
%CONFIGURA O MARCADOR DO CENTRO DO ALVO
    set(Im,'LineWidth',3,'color','r');
    hold on;                        %MANTÉM A JANELA EM UTILIZAÇÃO
%CASO O TAMANHO DO ARRANJO B=0, SIGNIFICA QUE NÃO HOUVE
%IDENTIFICAÇÃO DE NENHUM CONTORNO. CASO ACONTEÇA, ESCREVE MENSAGEM
%NA TELA E DÁ PROSSEGUIMENTO À ROTINA DE PROGRAMAÇÃO
else
    imshow(Im);                 %MOSTRA A IMAGEM INICIAL
%ESCREVE MENSAGEM NA IMAGEM
    text(20,20,'ALVO NÃO DETECTADO','Color','red','FontSize',10);
    X1=320;                     %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    Y1=240;                     %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    servoWrite(a,8,90);         %REPOSICIONA O SERVO MOTOR HORIZONTAL
    servoWrite(a,12,90);        %REPOSICIONA O SERVO MOTOR VERTICAL
    pause(1);                   %PAUSA PARA ESTABILIZAÇÃO
                                %DOS SERVO MOTORES
end
```

Figura 78 - Rotina “MULTI_ALVO2”.
Fonte: Autor.

ROTINA: ALVO3

```
%FUNÇÃO PARA DETECTAR O CENTRO DO ALVO
%SELECIONA O OBJETO CORRETO E O IDENTIFICA COMO ALVO ATRAVÉS
%DOS MOMENTOS INVARIANTES
%RETORNA AS COORDENADAS X1 E Y1 DO CENTRO DO ALVO
 function [Y1,X1]=ALVO3(MI,a,cam)
 figure(1)                      %ABRE A JANELA DA FIGURA 1
 hold on;                       %MANTÉM A JANELA EM UTILIZAÇÃO
Im = snapshot(cam);             %AQUISIÇÃO DA IMAGEM
if size(Im,3)==3                %PASSA A IMAGEM DE RGB (03 MATRIZES)
   f = rgb2gray(Im);            %PARA ESCALA DE CINZA(01 MATRIZ)
end
f= imcomplement(f);             %ENCONTRA O NEGATIVO DA IMAGEM
f= im2bw(f,.82);                %BINARIZA A IMAGEM PARA UM LIMIAR
%IDENTIFICA O CONTORNO DOS OBJETOS
[B,L,Nb,A] = bwboundaries(f,'noholes');
h=size(B);                      %ARMAZENA O TAMANHO DO ARRANJO B
if h~=[0 0]                     %CONTROLE DE FLUXO CASO B=0 (NÃO HAJA
                                %ALVO)
    for OBJETO=1:Nb
        b = B{OBJETO};                  %SELECIONA QUAL CONTORNO PARA
                                        %PROCESSAMENTO
        [Y1,X1,Ro,theta] = Polar2(b);   %RETORNA O CENTRO DO CONTORNO
                                        %ATRAVÉS DE
                                        %COORDENADAS POLARES

%RECORTA O CONTORNO DO OBJETO DA IMAGEM ORIGINAL
        SLC{OBJETO}=imcrop(f,[(X1-(length(b)/2))...
            (Y1-(length(b)/2)) length(b) length(b)]);
%COLOCA UMA MOLDURA PADRÃO NO OBJETO
        SLC{OBJETO}=padarray(SLC{OBJETO},[50 50],'both');
%EXECUTA O CÁLCULO DOS MOMENTOS INVARIANTES PARA O OBJETO
        MI1{OBJETO}=MomentosInvariantes(SLC{OBJETO});
%EXECUTA O CÁLCULO DA VARIÂNCIA PARA CADA OBJETO IDENTIFICADO
        VAR{OBJETO}=((sqrt((MI1{OBJETO}).^2))-(sqrt((MI).^2))).^2;
    end
    for OBJETO=1:Nb
%FLUXO DE CONTROLE PARA COMPARAR OS VALORES DE MOMENTOS INVARIANTES
%ATRAVÉS DA VARIÂNCIA E IDENTIFICAR QUAL OBJETO É O ALVO
        if (((VAR{OBJETO}(1)<=3)&&((VAR{OBJETO}(2)<=3))...
        &&((VAR{OBJETO}(3))<=3)))

            b = B{OBJETO};                  %SELECIONA QUAL CONTORNO
                                            %PARA PROCESSAMENTO
            [Y1,X1,Ro,theta] = Polar2(b);   %RETORNA O CENTRO DO
                                            %CONTORNO ATRAVÉS DE
                                            %COORDENADAS POLARES
            X1=round(X1);       %ARREDONDAMENTO PARA NÚMERO INTEIRO
            Y1=round(Y1);       %ARREDONDAMENTO PARA NÚMERO INTEIRO
            figure(1);          %ABRE A JANELA DA FIGURA 1
            hold on;            %MANTÉM A JANELA EM UTILIZAÇÃO
            imshow(Im);         %MOSTRA A IMAGEM INICIAL
            hold on;            %MANTÉM A JANELA EM UTILIZAÇÃO
            S = plot(X1,Y1,'X');%PLOTA UM X NO CENTRO DO ALVO
            %CONFIGURA O MARCADOR DO CENTRO DO ALVO
            set(S,'LineWidth',3,'color','r');
            hold on;            %MANTÉM A JANELA EM UTILIZAÇÃO
```

Continuação...

```
        end
    end

%CASO O TAMANHO DO ARRANJO B=0, SIGNIFICA QUE NÃO HOUVE IDENTIFICAÇÃO
%DE NENHUM CONTORNO. CASO ACONTEÇA, ESCREVE MENSAGEM NA TELA E
%DÁ PROSSEGUIMENTO À ROTINA DE PROGRAMAÇÃO
else
    imshow(Im);                 %MOSTRA A IMAGEM INICIAL
%ESCREVE MENSAGEM NA IMAGEM
    text(20,20,'ALVO NÃO DETECTADO','Color','red','FontSize',10);
    X1=320;                     %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    Y1=240;                     %DEFINE OS PONTOS CENTRAIS DA IMAGEM
    pause(1);                   %PAUSA PARA ESTABILIZAÇÃO DOS SERVO
                                %MOTORES
end
```

Figura 79 - Rotina “ALVO3”.
Fonte: Autor.

ROTINA: MOVIMENTA_SERVOS_3

```
function [MOVX, MOVY] =
MOVIMENTA_SERVOS_3(a,MOVX,MOVY,ERRO_X,ERRO_Y,ERRO)
%ROTINA DE PROGRAMA PARA CORREÇÃO DO ERRO ENTRE ALVO E MIRA E
%MOVIMENTAÇÃO DOS SERVO MOTORES, COM AJUSTE FINO E GROSSO

ex=0;       %ZERA OS ÍNDICE DE CONTROLE DE FLUXO

%CORREÇÃO DO ERRO

if (ERRO_X>0)&&(ERRO_Y>0)           %ERRO NO PRIMEIRO QUADRANTE

    if (ERRO_X<=5)

        if(ERRO_Y<=5)&&(ex==0)
            MOVX=MOVX;
            MOVY=MOVY;
            ex=1;
        end

        if(ERRO_Y>5)
            if(ERRO_Y<=50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY+1;
                ex=1;
            end

            if(ERRO_Y>50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY+5;
                ex=1;
            end
```

Continuação...

```
        end
    end
    if (ERRO_X>5)
        if(ERRO_Y<=5)
            if (ERRO_X<=50)&&(ex==0)
                MOVX=MOVX+1;
                MOVY=MOVY;
                ex=1;
            end

            if (ERRO_X>50)&&(ex==0)
                MOVX=MOVX+5;
                MOVY=MOVY;
                ex=1;
            end

        end
        if(ERRO_Y>5)

            if(ERRO_Y<=50)

                if (ERRO_X<=50)&&(ex==0)
                    MOVX=MOVX+1;
                    MOVY=MOVY+1;
                    ex=1;
                end

                if (ERRO_X>50)&&(ex==0)
                    MOVX=MOVX+5;
                    MOVY=MOVY+1;
                    ex=1;
                end

            end

            if(ERRO_Y>50)

                if (ERRO_X<=50)&&(ex==0)
                    MOVX=MOVX+1;
                    MOVY=MOVY+5;
                    ex=1;
                end

                if (ERRO_X>50)&&(ex==0)
                    MOVX=MOVX+5;
                    MOVY=MOVY+5;
                    ex=1;
                end

            end

        end

    end

end
```

Continuação...

```
%-------------------------------------------------------------------------
---

if (ERRO_X<0)&&(ERRO_Y>0)           %ERRO NO SEGUNDO QUADRANTE

    if (ERRO_X>=-5)

        if(ERRO_Y<=5)&&(ex==0)
            MOVX=MOVX;
            MOVY=MOVY;
            ex=1;
        end

if(ERRO_Y>5)

            if(ERRO_Y<=50)&&(ex==0)


                MOVX=MOVX;
                MOVY=MOVY+1;
                ex=1;
            end

            if(ERRO_Y>50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY+5;
                ex=1;
            end

        end

    end

    if (ERRO_X<-5)

        if(ERRO_Y<=5)

            if (ERRO_X>=-50)&&(ex==0)
                MOVX=MOVX-1;
                MOVY=MOVY;
                ex=1;
            end

            if (ERRO_X<-50)&&(ex==0)
                MOVX=MOVX+5;
                MOVY=MOVY;
                ex=1;
            end

        end

        if(ERRO_Y>5)

            if(ERRO_Y<=50)
```

Continuação...

```
                if (ERRO_X>=-50)&&(ex==0)
                    MOVX=MOVX-1;
                    MOVY=MOVY+1;
                    ex=1;
                end

                if (ERRO_X<-50)&&(ex==0)
                    MOVX=MOVX-5;
                    MOVY=MOVY+1;
                    ex=1;
                end

            end
            if(ERRO_Y>50)
                if (ERRO_X>=-50)&&(ex==0)
                    MOVX=MOVX-1;
                    MOVY=MOVY+5;
                    ex=1;
                end

                if (ERRO_X<-50)&&(ex==0)
                    MOVX=MOVX-5;
                    MOVY=MOVY+5;
                    ex=1;
                end

            end

        end
    end
end
%-------------------------------------------------------------------------
---
if (ERRO_X<0)&&(ERRO_Y<0)                           %ERRO NO TERCEIRO
QUADRANTE
    if (ERRO_X>=-5)
        if(ERRO_Y>=-5)&&(ex==0)
            MOVX=MOVX;
            MOVY=MOVY;
            ex=1;
        end
        if(ERRO_Y<-5)
            if(ERRO_Y>=-50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY-1;
                ex=1;
            end
            if(ERRO_Y<50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY-5;
                ex=1;
            end
        end
```

Continuação...

```
    end
    if (ERRO_X<-5)
        if(ERRO_Y>=-5)
            if (ERRO_X>=-50)&&(ex==0)
                MOVX=MOVX-1;
                MOVY=MOVY;
                ex=1;
            end
            if (ERRO_X<-50)&&(ex==0)
                MOVX=MOVX-5;
                MOVY=MOVY;
                ex=1;
            end
        end
        if(ERRO_Y<-5)
            if(ERRO_Y>=-50)
                if (ERRO_X>=-50)&&(ex==0)
                    MOVX=MOVX-1;
                    MOVY=MOVY-1;
                    ex=1;
                end
                if (ERRO_X<-50)&&(ex==0)
                    MOVX=MOVX-5;
                    MOVY=MOVY-1;
                    ex=1;
                end
            end
            if(ERRO_Y<-50)
                if (ERRO_X>=-50)&&(ex==0)
                    MOVX=MOVX-1;
                    MOVY=MOVY-5;
                    ex=1;
                end
                if (ERRO_X<-50)&&(ex==0)
                    MOVX=MOVX-5;
                    MOVY=MOVY-5;
                    ex=1;
                end
            end
        end
    end
end
%-------------------------------------------------------------------------
---
if (ERRO_X>0)&&(ERRO_Y<0)           %ERRO NO QUARTO QUADRANTE
    if (ERRO_X<=5)
        if(ERRO_Y>=-5)&&(ex==0)
            MOVX=MOVX;
            MOVY=MOVY;
            ex=1;
        end
```

Continuação...

```
        if(ERRO_Y<-5)
            if(ERRO_Y>=-50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY-1;
            end
            if(ERRO_Y<-50)&&(ex==0)
                MOVX=MOVX;
                MOVY=MOVY-5;
                ex=1;
            end
        end
    end
    if (ERRO_X>5)
        if(ERRO_Y>=-5)
            if (ERRO_X<=50)&&(ex==0)
                MOVX=MOVX+1;
                MOVY=MOVY;
                ex=1;
            end
            if (ERRO_X>50)&&(ex==0)
                MOVX=MOVX+5;
                MOVY=MOVY;
                ex=1;
            end
        end
        if(ERRO_Y<-5)
            if(ERRO_Y>=-50)
                if (ERRO_X<=50)&&(ex==0)
                    MOVX=MOVX+1;
                    MOVY=MOVY-1;
                    ex=1;
                end
                if (ERRO_X>50)&&(ex==0)
                    MOVX=MOVX+5;
                    MOVY=MOVY-1;
                    ex=1;
                end
            end
            if(ERRO_Y<-50)
                if (ERRO_X<=50)&&(ex==0)
                    MOVX=MOVX+1;
                    MOVY=MOVY-5;
                    ex=1;
                end
                if (ERRO_X>50)&&(ex==0)
                    MOVX=MOVX+5;
                    MOVY=MOVY-5;
                    ex=1;
                end
            end
        end
    end
end

%-------------------------------------------------------------------------
---
```

Continuação...

```
servoWrite(a,8,MOVX);                          %MOVIMENTA O LASER
servoWrite(a,12,MOVY);
pause(0.1);                                    %PAUSA BREVE PARA
POSICIONAMENTO
                                               %DO SERVO MOTOR

if (ERRO==0)&&(ex==0)                  %ETAPA FINAL QUANDO ERRO ESTÁ ENTRE
                                       %+5 E -5 ESCREVE MENSAGEM DE OK
    text(600,460,'OK!','Color',...     %NA TELA
        'green','FontSize',10);
    ex=1;                              %ALTERA O VALOR DO ÍNDICE
end
```

Figura 80 - Rotina "MOVIMENTA_SERVOS_3".
Fonte: Autor.

ROTINA: MAINT_TCC_3

```
%TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO
%
%ROBÔ SEGUIDOR DE ALVOS MÓVEIS COM MIRA A LASER AUTOMÁTICA
%
%ALUNO: MÁRLON AFONSO BORGES DAMASCENO
%PROFESSOR ORIENTADOR: SÉRGIO LUIZ DA SILVA PITHAN
%CURSO: ENGENHARIA DE AUTOMAÇÃO INDUSTRIAL
%
%
%ROTINA DE PROGRAMA PRINCIPAL
%
clear all;      %LIMPA A WORKSPACE
close all;      %FECHA TODAS AS JANELAS ABERTAS
clc;            %LIMPA A COMMAND WINDOW

%CONFIGURAÇÕES INICIAIS
[a, cam]=INICIALIZA_TCC;
%IDENTIFICA O OBJETO COMO ALVO
[Y1,X1,MI]=MULTI_ALVO2(a,cam);

%DEFINE AS POSIÇÕES INICIAIS DA MIRA
MOVX=80;
MOVY=80;
servoWrite(a,8,MOVX);
servoWrite(a,12,MOVY);

ex=0;
ey=0;

for TEMPO=1:200;

%IDENTIFICA O ALVO
[Y1,X1]=ALVO3(MI,a,cam);

%CALCULA O MÓDULO DA POSIÇÃO DO ALVO
MODULO_ALVO=round(sqrt((X1^2)+(Y1^2)));
```

Continuação...

```
%IDENTIFICA A MIRA
[YM,XM,RESET]=MIRA(a,cam);

%CALCULA O MÓDULO DA POSIÇÃO DA MIRA
MODULO_MIRA=round(sqrt((XM^2)+(YM^2)));

%CALCULA OS ERROS
ERRO_X=XM-X1;
ERRO_Y=YM-Y1;
ERRO=round(sqrt((ERRO_X^2)+(ERRO_Y^2)));

%APRESENTA O VALOR DO ERRO NA TELA
hold on
str={'Erro:',ERRO};
text(600,425,str,'Color','red','FontSize',10);

%MOVIMENTA OS SERVO MOTORES
[MOVX, MOVY] = MOVIMENTA_SERVOS_3(a,MOVX,MOVY,ERRO_X,ERRO_Y,ERRO);

    if RESET==1

        %DEFINE AS POSIÇÕES INICIAIS DA MIRA
        MOVX=80;
        MOVY=80;
        servoWrite(a,8,MOVX);
        servoWrite(a,12,MOVY);
        pause(1);

        %IDENTIFICA A MIRA
        [YM,XM,RESET]=MIRA(a,cam);
    end

%CONFIGURA TABELAS DE RESULTADOS
TAB_ERRO(1,TEMPO)=ERRO;                         %TABELA DE ERRO
TAB_MODULO_ALVO(1,TEMPO)=MODULO_ALVO;           %TABELA DE SET-POINT
TAB_MODULO_MIRA(1,TEMPO)=MODULO_MIRA;           %TABELA DE VARIÁVEL DE
PROCESSO
end
%CONFIGURA E APRESENTA UMA JANELA COM O GRÁFICO
figure(3)
grid on
hold on
plot(TAB_ERRO,':','LineWidth',1,'color','red');
plot(TAB_MODULO_ALVO,'LineWidth',2,'color','blue');
plot(TAB_MODULO_MIRA,'-.','LineWidth',1.5,'color','m');
legend('ERRO','ALVO: SET POINT','MIRA: PV');
legend('Location','eastoutside');
title('Resultados - Seguidor de Objetos');
xlabel('Tempo de Amostragem');
ylabel('Valores em pixels');
```

Figura 81 - Rotina “MAIN_TCC_3”.
Fonte: Autor.