



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

ARTHUR RIBEIRO REZENDE

APRIMORAMENTO DE UM SOFTWARE PARA TREINAMENTO DE
TIRO ESPORTIVO PARAOLÍMPICO

UBERLÂNDIA – MG

2017

|

Arthur Ribeiro Rezende

**APRIMORAMENTO DE UM SOFTWARE DE TREINAMENTO DE TIRO
ESPORTIVO PARAOLÍMPICO**

Trabalho de Fim de Curso a ser apresentado
à Faculdade de Engenharia Mecânica da
Universidade Federal de Uberlândia, como
parte dos requisitos para a obtenção do título
de BACHAREL EM ENGENHARIA
MECÂNICA.

Orientador: Prof. Dr. José Antônio Ferreira
Borges

UBERLÂNDIA – MG
2017

AGRADECIMENTOS

À Deus por ter me dado saúde, força e as energias necessárias para superar todas as dificuldades que se apresentaram na minha vida até esse momento.

À Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, seu corpo docente, direção e coordenação por propiciar a estrutura e o ambiente próprios para o meu desenvolvimento intelectual e profissional.

Aos meus pais, Alvaro e Lenina, por terem me dado o suporte financeiro e emocional para eu conseguir vencer essa importante fase da minha vida.

À minha querida Gelcimary, por ter me lembrado, sempre com muito carinho, da minha capacidade.

Finalmente, a todos que direta ou indiretamente fizeram parte da minha formação.

RESUMO

Com o intuito de auxiliar os praticantes de tiro esportivo paraolímpico, o estudo aqui realizado se propôs a aprimorar o software elaborado previamente no trabalho de MORAES (2015), visando facilitar a sua implementação prática. Esse software funcionava como um jogo, permitindo a prática de tiro esportivo paraolímpico em ambientes fechados com uso de captura de imagens, processadas através da plataforma do Matlab. Ao programa original foi acrescentado uma função de calibração para que a câmera pudesse ser acoplada na região externa da arma, sem prejudicar a precisão da simulação. Para a calibração é usado de um colimador laser no formato de munição, que é capaz de apontar com precisão o local de impacto dos projéteis disparados pela arma. A função de calibração analisa os pixels da região da tela, comparando-os com parâmetros que identificam o laser. Esses parâmetros foram criados com base em uma análise de oito testes satisfatórios, obtidos de doze realizados. Os critérios obtidos com essa análise se provaram eficazes quando testados, pois não ocorreu a identificação de falsos positivos quando as recomendações para o bom funcionamento foram seguidas e, mesmo assim, permitiram a identificação de uma média de 28,8% dos pixels que caracterizam a projeção laser. Por tanto, o resultado final foi um software com mais aplicabilidade prática que o original, e que manteve a sua capacidade de operar com equipamentos de baixo custo.

Palavras-chave: Análise de imagem. Projeção laser. Webcam. Matlab. Tiro esportivo. Paraolímpico. Pixel.

SUMÁRIO

RESUMO.....	4
SUMÁRIO.....	5
Lista De Figuras.....	7
Lista De Tabelas:	9
1 Introdução:	9
1.1 Motivação e Objetivos do Trabalho:	11
2 Revisão Bibliográfica:	13
2.1 Ponto de Partida deste Trabalho:	13
2.1.1 Processamento de Imagens	14
2.1.2 Arquivos e Pastas.....	16
2.1.3 Interfaces do Software:	18
2.1.4 Funções Usadas no Software	23
2.1.5 Teste de Precisão e Conclusão.....	28
2.2 Dispositivos de Tiro com Câmeras:	29
2.2.1 Seeburg Ray-O-Lite.....	30
2.2.2 Magnavox Odyssey Light Gun:	31
2.2.3 Nintendo Zapper (Duck Hunt).....	33
2.2.4 Super Scope / Nintendo Scope:	35
3 Desenvolvimento do Hardware:	38
3.1 Comparação de sistemas de mira de jogos de tiro:	38
3.2 Projeto do Hardware:	38
3.2.1 Hipóteses assumidas:	42
3.2.2 Caracterização do Dispositivo:	43
4 Desenvolvimento do Software:	44
4.1 Melhoramentos e Correções no Software	45
4.1.1 Alteração da interface regras:	45

4.1.2	Ajuste da distância do usuário à tela:	46
4.1.3	Cálculo do tamanho do alvo na tela:.....	48
4.2	Elaboração da Função de Calibração	51
4.2.1	Interface calibr.1:	51
4.2.2	Interface calibr.2:	53
4.2.3	Identificando o laser nas imagens capturadas:.....	55
4.3	Teste da Função de Calibração e Resultados	66
5	Considerações Finais:.....	70
6	Trabalhos Futuros:.....	71
	Bibliografia	72
	APÊNDICES	75
	APÊNDICE A – Função calibracao.m.....	75
	APÊNDICE B – Função callback_botao.m	79
	APÊNDICE C – Função gatilho.m	87
	APÊNDICE D – Seção da Função InterfaceResize.m Referente à Calibração	92
	APÊNDICE E – Função telaalvo.m.....	94

Lista De Figuras

Figura 1: Esquema do jogo elaborado para a prática de tiro esportivo. (MORAES, 2015).....	13
Figura 2: Maneira como uma imagem é interpretada pelo Matlab usando o sistema RGB. Fonte: (THE MATHWORKS, [2016?]).....	14
Figura 3: Sequência de imagens utilizadas para o processamento de dados do programa. (MORAES, 2015).....	15
Figura 4:Arquivos e pastas utilizados no programa original.	16
Figura 5: Estrutura de armazenamento de dados utilizada no programa. (MORAES, 2015).....	17
Figura 6: Interface inicial.....	18
Figura 7: Interface regras	19
Figura 8:Interface desempenho.....	19
Figura 9: Interface plot modalidades.	20
Figura 10: Interface plot datas.	21
Figura 11: Interface de configurações.	21
Figura 12: Interface modalidades.	22
Figura 13: Interface alvo. (MORAES, 2015)	23
Figura 14: Blocos de comandos da função “InterfaceResize.m”. (MORAES, 2015) ..	25
Figura 15: Sequência de funções associadas à interface inicial. (MORAES, 2015)	27
Figura 16: Sequência de funções associadas à interface alvo. (MORAES, 2015)	28
Figura 17: Bancada de teste do programa original. (MORAES, 2015).....	28
Figura 18: Gabinete e Interface do Seeburg Ray-O-Lite, de 1936. (Bueschel, 1995)..	30
Figura 19: Odyssey rodando o seu famoso jogo de tênis de mesa. (Gegan & Winter, 2012).....	31
Figura 20: Películas plásticas do Odyssey usadas nos jogos de tiro. Jogos correspondentes: (A) Dogfight; (B) Prehistoric Safari; (C) Shootout; (D) Shooting Gallery. (Gegan & Winter, 2012, p. Shooting Gallery)	32
Figura 21: Conjunto Shooting Gallery do Magnavox Odyssey. (Gegan & Winter, 2012, p. Rifle).....	32
Figura 22: Nintendo Zapper. (Sic!, 2007)	33
Figura 23:Pato sendo acertado no jogo Duck Hunt. (Toad_Speedgaming, 2007 [arquivo de vídeo])	34

Figura 24: Sequência de frames seguintes a um tiro no jogo Duck Hunt. (a) Logo antes do tiro, um dos patos está atrás da árvore; (b) Tela escura; (c) 1º pato sendo destacado (a árvore não interfere no acerto); (d) 2º pato sendo destacado; (e) Delay antes de os patos serem mostrados novamente; (f) Representação de qual pato foi acertado. (Ramy32.exe, 2013 [arquivo de vídeo]).....	35
Figura 25: SNES Super Scope e o seu receptor. (The Strong, 2016).....	36
Figura 26: Esquema de um televisor de tubo de raio catódico. (Nintendo®, 1993, pp. 4-2-1)	37
Figura 27: Exemplo de trilho para acoplamento de dispositivos em carabinas. (Johnson, 2014)	39
Figura 28: Munição laser para armas de calibre 0,45. (Volk, 2014)	40
Figura 29: Ilustração da necessidade de se realizar a calibração usando da superfície da tela que exibe o software.	43
Figura 30: Interferência do cano da arma na imagem. (A) Acoplamento da câmera na arma; (B) Imagem capturada pela câmera com esse acoplamento.	44
Figura 31: Interface instruções, que substitui a antiga interface regras.....	45
Figura 32: Interface inicial reestruturada.....	46
Figura 33: Nova interface alvo, sem o slider para ajuste do tamanho no lado direito..	48
Figura 34: Semelhança de triângulos usada para determinação da distância à tela.....	49
Figura 35: Relação entre o diâmetro na tela e o diâmetro na imagem "bordas".....	51
Figura 36: Interface calibr.1, que contém as instruções para a calibração e um espaço para inserção da distância.	52
Figura 37: Interface Calibr.2, onde a calibração é realizada de fato.	54
Figura 38:Diferença do ponto laser projetado na imagem1 (esquerda) e na imagem2 (direita).	57
Figura 39: Exemplificação do reflexo do ponteiro laser na tela.	58
Figura 40: Detalhe ampliado do ponto laser e do reflexo do emissor laser na tela.	59
Figura 41: Exemplificação de como evitar o reflexo do emissor laser na tela com a inclinação da mesma.....	60
Figura 42: Testes usados para identificar as características do ponto laser na tela.	61
Figura 43: Pixels pertencentes ao ponto laser nos testes T2, T4, T5 e T6 utilizados na caracterização do mesmo.....	63
Figura 44: Pixels pertencentes ao ponto laser nos testes T8, T9, T11 e T12 utilizados na caracterização do mesmo.	63

Figura 45: Exemplificação da presença de pixels azuis na região identificada pelo programa como pertencente à tela	65
Figura 46: Teste dos parâmetros de identificação do ponto laser.....	67
Figura 47: Destaque dos pixels referentes ao ponto laser de cada imagem que atenderam aos critérios estabelecidos.....	68

Lista De Tabelas:

Tabela 1: Resultados do teste de calibração. (MORAES, 2015).....	29
Tabela 2: Comparação entre os métodos de fixação da câmera na arma.	41
Tabela 3: Valores máximos, mínimos, e médios de R, G e B dos pixels destacados como sendo pertencentes ao ponto laser.....	65
Tabela 4: Comparação da quantidade de pixels selecionados pelo critério com o total de pixels pertencentes à projeção laser.....	68

1 **Introdução:**

O tiro esportivo é uma prática na qual os participantes utilizam armas de fogo para acertar alvos móveis ou estáticos, podendo ser praticado nas posições em pé, ajoelhado ou deitado. Ao atleta, é atribuída uma pontuação que é proporcional ao número de acertos e à precisão dos impactos no alvo. (Comitê Olímpico Brasileiro, 2016)

O surgimento do tiro como prática esportiva é um tanto quanto incerto devido à falta de registros históricos precisos. Apesar de as armas de fogo datarem de 1300 D.C, a primeira competição de tiro de que se tem registro ocorreu apenas em 1477, em Eichstätt, na região da Bavária, atualmente uma província da Alemanha. Nesta ocasião os atiradores muito provavelmente utilizavam mosquetes, atirando em alvos a duzentos metros de distância. O esporte se popularizou no século 16, sendo praticado principalmente no interior da Alemanha. É possível encontrar em vários museus alemães, alvos de madeira que datam da década de 1540; esses alvos eram utilizados em casamentos, para os convidados atirarem e depois eram guardados pelos anfitriões como uma recordação daquela data. (OS EDITORES DE ENCYCLOPÆDIA BRITANNICA, [2015])

A prática de tiro esportivo chegou ao Brasil trazida pelos primeiros imigrantes alemães que chegaram ao país, na região sul, no ano de 1824. Nessa época, as armas de fogo já eram usadas no país, mas a prática de tiro ao alvo não era difundida. A cidade de Santa Cruz do Sul (RS) teve uma das primeiras sociedades de tiro do país, a Schützengilde, fundada em 1863 (ASSMANN & MAZO, 2012). No ano de 1885 foi fundado o “Von Musterreiter Club Porto Alegre” ou Clube dos Cavaleiros de Amstras, na capital do estado. Esse clube promovia, além de práticas esportivas, diferentes festividades, as Schützenfest ou Festa dos Atiradores (MAZO & GAYA, 2006). Os eventos culturais promovidos em toda a região sul e o próprio tiro ao alvo possuíam uma grande importância para a preservação da cultura e da identidade dos alemães recém-chegados ao Brasil (ASSMANN & MAZO, 2012).

Além da criação de clubes e associações de tiro e a disseminação do esporte pelo país, no dia 5 de setembro de 1906 foi criada a Confederação do Tiro Brasileiro através do decreto de número 1503. No ano seguinte, 8 países se juntaram para formar a União Internacional de Tiro (UTI), que durou até o início da primeira Guerra Mundial. O primeiro Campeonato Brasileiro de Tiro ao Alvo foi realizado em 1910, quatro anos após a criação da Confederação. (OS EDITORES DO SITE A HISTÓRIA, 2016) (ASSMANN, 2010).

No dia 26 de outubro de 1917, houve um enfraquecimento do tiro como esporte no Brasil. O motivo disso foi a incorporação das sociedades de tiro existentes em território nacional pelo exército brasileiro através do decreto legislativo de número 3.361 (OS EDITORES DO SITE A HISTÓRIA, 2016). Mesmo assim, em 1920 o tiro esportivo garantiu ao Brasil a sua primeira medalha de ouro olímpica na primeira participação do país nos jogos. O feito foi realizado por Guilherme Paraense, tenente do exército Brasileiro, que fez 274 pontos, apenas 2 acima do medalhista de prata norte americano (AUTORES DO PORTAL BRASIL 2016, 2016). Na atualidade é possível encontrar locais para a prática de tiro esportivo em quase todos os estados da federação. (CBTE, [201?])

O tiro esportivo já estava presente na primeira olimpíada da era moderna, realizada em 1896 na cidade de Atenas. Ele era uma das 10 modalidades esportivas que seriam disputadas, sendo as outras o atletismo, o ciclismo, a esgrima, a ginástica, o halterofilismo, a luta, a natação, o tênis e o remo, sendo que esta última modalidade não foi de fato disputada devido às condições climáticas. Participaram do evento 241 atletas de 14 países, sendo todos eles homens (EDITORES DO PORTAL EDUCAÇÃO FÍSICA, 2012) (EDITORA ABRIL S.A, [entre 2010 e 2016]). Na primeira competição olímpica de tiro esportivo a disputa ficou restrita a apenas 5 categorias: rifle a 200m e a 300m, pistola a 50m, pistola de tiro rápido a 25m e pistola militar a 25m. (INTERNATIONAL OLYMPIC COMMITTEE, 2016). Com o passar do tempo o esporte se aperfeiçoou e atualmente é disputado em 15 categorias, sendo 9 masculinas e 6 femininas. (INTERNATIONAL OLYMPIC COMMITTEE, 2016)

Antes de 1945 os deficientes físicos não tinham muito espaço no mundo dos esportes. Porém, naquele ano o médico Ludwig Guttmann organizou uma competição intitulada por ele mesmo de Jogos de Stoke Mandeville, destinada a pessoas que dependiam de cadeiras de rodas. Somente em 1960 que os jogos ocorreram com o nome de Paraolimpíadas, nesta ocasião o número de inscritos já havia saltado de 16 para 400. Apesar disso, o tiro esportivo só ganhou espaço na competição em 1976, nos Jogos Paraolímpicos de Toronto, onde atletas de 14 países disputaram em 3 modalidades diferentes. A partir daí o esporte foi ganhando espaço e desde 2000 as provas se estabilizaram em 12 modalidades, masculinas, femininas e mistas e desde de 2008 atletas de 44 países participam deste evento. (IPC, [2016?]) (MINISTÉRIO DA CULTURA, [2016?])

1.1 **Motivação e Objetivos do Trabalho:**

O tiro esportivo tem crescido e ganhado espaço, tanto no Brasil como no mundo. Porém, a sua prática está associada a algumas dificuldades, sendo uma delas o custo associado aos equipamentos e insumos necessários à prática do esporte. Além disso, no caso específico do Brasil, os atletas têm de enfrentar um longo e tortuoso caminho burocrático para obter o certificado de registro e demais documentos que o habilitam legalmente a praticar esta atividade que é controlada pelo exército brasileiro. Cabe ressaltar também que esportes praticados a partir do uso de equipamentos onde a tecnologia, qualidade de fabricação e uso de materiais tecnológicos avançados afetam diretamente o desempenho, implicam em custos significativos para se manter competitivo. Como o tiro esportivo demanda o uso constante de insumos nacionais e importados (munição, material de recarga, peças de reposição, etc), o enfrentamento da burocracia e os prazos longos para a obtenção dos materiais são uma constante na vida do atleta.

Vencida a burocracia para a obtenção dos documentos necessários e supridos os custos com equipamentos e insumos, o atleta precisa então se deslocar para o local de treinamento, o qual muitas vezes não fica próximo de sua residência. No Brasil existem hoje 23 locais de treino registrados na Confederação Brasileira de Tiro Esportivo (CBTE), espalhados em 21 estados da federação (CBTE, [201?]). Além disso, durante o treino existe a necessidade de trocar os alvos, recarregar as armas e suportar o impacto produzido na arma pelos disparos. Há que se ressaltar que em seu conjunto, essas tarefas não são fáceis para alguns atletas paraolímpicos.

Os atletas paraolímpicos de tiro esportivo são separados em 2 categorias: SH1, que são aqueles que não precisam de suporte para a arma; e SH2, que são aqueles que precisam de suporte para arma, por não conseguirem sustentar o seu peso. Alguns atletas da classe SH2 podem inclusive contar com auxílio durante as competições para recarregar a arma (IPC, [2016?]). É esperado que esses atletas tenham maiores dificuldades para se locomover até o campo de treinamento, trocar os alvos usados e recarregar as armas durante seus treinamentos, além de estarem mais sujeitos a desgaste físico em função do impacto causado pelos disparos.

Jonas Jacobsson, atleta com lesão na medula e que ganhou 17 medalhas de ouro em um total de 9 participações em jogos paraolímpicos escreveu o artigo “A Week in The Life of a Shooter”. Em seu texto ele diz: “Na Suécia, tende a nevar de vez em quando e atirar ao ar livre não é tão bom quando a temperatura está abaixo de zero” (Jacobsson, 2012). Esta afirmação,

vinda de um praticante ativo do esporte, estimula um esforço para permitir a prática do tiro esportivo em ambientes mais confortáveis.

A motivação deste trabalho consiste em facilitar o treinamento de todos os praticantes de tiro esportivo, especialmente aqueles atletas paraolímpicos da classe SH2 com maiores limitações física. Para isso, será dada continuidade ao desenvolvimento de um software, no formato de um jogo eletrônico, visando implementar a possibilidade da prática do tiro em ambientes internos por meio de uma simulação realística. Como objetivo final, pretende-se conceber um software comercial que funcione com um hardware simples, barato e que possa, de maneira precisa, ser acoplado a armas de fogo esportivas, armas de pressão e simulacros tipo AirSoft.

2 Revisão Bibliográfica:

2.1 Ponto de Partida deste Trabalho:

Este trabalho se baseia em um outro intitulado “Estudo e Desenvolvimento de Jogos Aplicados ao Treinamento de Atletas Paralímpicos”, de MORAES (2015). Foi desenvolvido um software em formato de jogo na plataforma Matlab, visando permitir a prática de tiro esportivo paraolímpico em ambientes fechados, através de uma simulação, eliminando com isso a necessidade de o atleta se locomover até um campo de treinamento. Os hardwares utilizados são um dispositivo de captura de imagens, como por exemplo uma webcam ou uma câmera digital e uma chave do tipo botão de pressão, que desempenhará a função do botão esquerdo do mouse. A ideia é que esta câmera seja fixada no cano da arma e que o botão de pressão seja acoplado ao gatilho. Dessa forma, quando o gatilho é pressionado, ele ativa o botão de pressão que manda um comando ao programa, fazendo com que a câmera capture imagens do ambiente e, em seguida, as envie para o computador. Essas imagens são utilizadas para determinar em que ponto da tela o usuário estava mirando. Um esquema simplificado ilustrando o funcionamento do jogo pode ser visto na Figura 1.

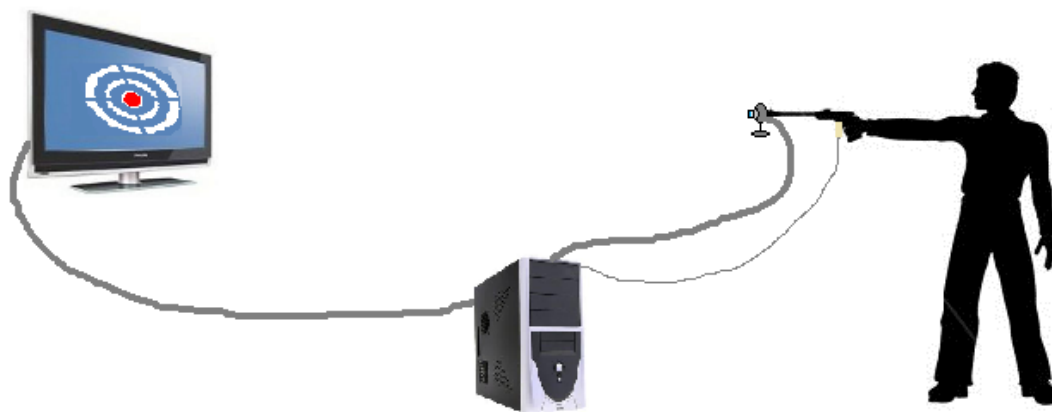


Figura 1: Esquema do jogo elaborado para a prática de tiro esportivo. (MORAES, 2015)

O motivo pelo qual o Matlab foi a plataforma escolhida para fazer o processamento das imagens capturadas foi o fato de ele ter bibliotecas de processamento de imagens já embutidas no seu código fonte e possuir um ambiente de trabalho amigável e várias funções úteis já prontas. Esse conjunto de características reduz o esforço requerido na programação e permite que maior ênfase seja dada às estratégias relacionadas ao problema em si.

2.1.1 Processamento de Imagens

As imagens capturadas pela webcam são interpretadas pelo Matlab como matrizes, onde cada elemento da matriz corresponde a um determinado pixel da imagem e os seus valores estão associados às cores do mesmo. O modelo de matrizes e cores utilizado nesse projeto é o RGB, do inglês Red, Green e Blue, que significam respectivamente Vermelho, Verde e Azul. Nesse modelo cada pixel da imagem é representado por 3 números, um para cada uma das cores citadas, que são interpretados pelo Matlab como variáveis do tipo *uint8*, o que quer dizer que eles são sempre positivos, inteiros e ocupam 8 bits na memória, ou seja, seu valor pode variar de 0 a 255. Um pixel completamente branco seria representado pelos valores [255,255,255] e um pixel completamente preto pelos valores [0,0,0].

Como o valor de cada uma das cores do sistema RGB tem de ser armazenado separadamente, as imagens capturadas são representadas por uma matriz de 3 dimensões, composta por 3 planos, cada um com quantidade de elementos coincidente com a quantidade de pixels da imagem. Cada plano contém as informações de uma das cores do modelo RGB. Esse sistema permite a representação de mais de 16 milhões de cores. (THE MATHWORKS, [2016?]). A figura 2 mostra um esquema detalhando a maneira como o Matlab interpreta alguns dos pixels de uma imagem, porém, no caso específico desta imagem ilustrativa, ao invés dos elementos serem do tipo *uint8* eles são do tipo *double*, variando assim entre 0 e 1.

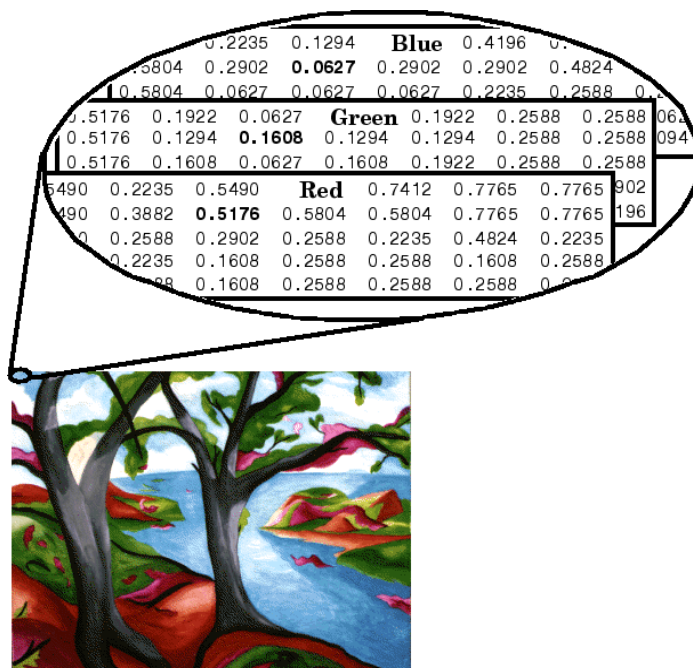


Figura 2: Maneira como uma imagem é interpretada pelo Matlab usando o sistema RGB. Fonte: (THE MATHWORKS, [2016?])

Para que o software identifique a tela nas imagens capturadas pela webcam, são tiradas duas fotos em sequência que devem abranger a tela onde o jogo está rodando. Para a primeira foto a interface do jogo está iluminada e exibe o alvo a ser acertado, conforme mostra a Figura 3A. Logo antes da segunda foto a tela é escurecida, resultando em uma imagem coincidente com a primeira, porém com a tela toda escura como mostra a Figura 3B. Em seguida, os valores RGB das matrizes referentes às duas imagens são subtraídos, resultando em uma matriz onde todos os elementos são zero ou próximos de zero, com exceção daqueles que compõem a tela, conforme mostra a Figura 3C. Essa imagem é então tratada, resultando em uma figura toda preta, com exceção da tela, que fica toda branca e por tanto destacada como mostra a Figura 3D.

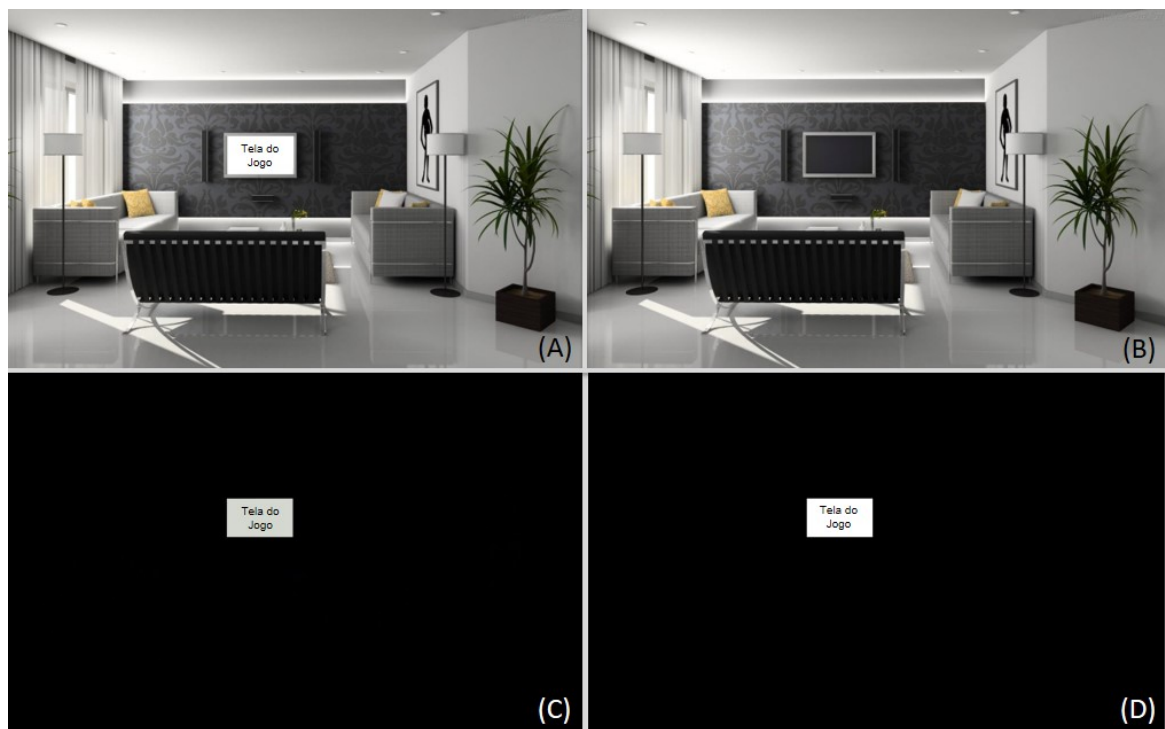


Figura 3: Sequência de imagens utilizadas para o processamento de dados do programa. (MORAES, 2015)

Desta forma o software consegue identificar a localização da tela com a interface nas imagens capturadas pela câmera. Para dar prosseguimento é necessário saber em qual ponto destas imagens um suposto projétil teria atingido caso a arma estivesse carregada e houvesse de fato realizado um disparo. O trabalho desenvolvido por MORAES (2015) supõe que o ponto de impacto do projétil coincidiria com o ponto central da imagem capturada pela câmera, partindo do pressuposto que a câmera estaria fixada na saída do cano da arma e perfeitamente alinhada com o mesmo. Assim, estando o centro da câmera dentro da região correspondente à tela nas imagens capturadas, o software consegue identificar em que ponto

específico ocorreria o impacto de um projétil. Neste caso, a interface do jogo mostra na tela o ponto em que o disparo teria atingido, a pontuação obtida no último disparo e os pontos acumulados até o momento.

No final do treino os pontos computados são armazenados em uma estrutura de dados similar a uma matriz. Isso permite o acesso a esses dados posteriormente e a comparação dos mesmos através de gráficos gerados pelo próprio software. A estrutura separa os dados por modalidade e por cada sessão de treino realizada, permitindo que cada participante registrado possa visualizar o histórico dos seus treinos, avaliar a evolução do seu desempenho em uma determinada modalidade e inclusive ver o desempenho em um treino específico que já tenha realizado.

2.1.2 Arquivos e Pastas

Uma vez que esse trabalho usa como base os códigos desenvolvidos no trabalho de MORAES (2015), cabe neste momento descrever de forma detalhada a sua estrutura de pastas e arquivos, visando proporcionar um melhor entendimento que permitirá, em uma etapa posterior, compreender as diversas alterações implementadas com o objetivo de ampliar os recursos do software.

Para o seu funcionamento, o programa utiliza 16 arquivos e algumas imagens, armazenadas em separado na pasta “Imagens”, conforme apresentado na Figura 4.

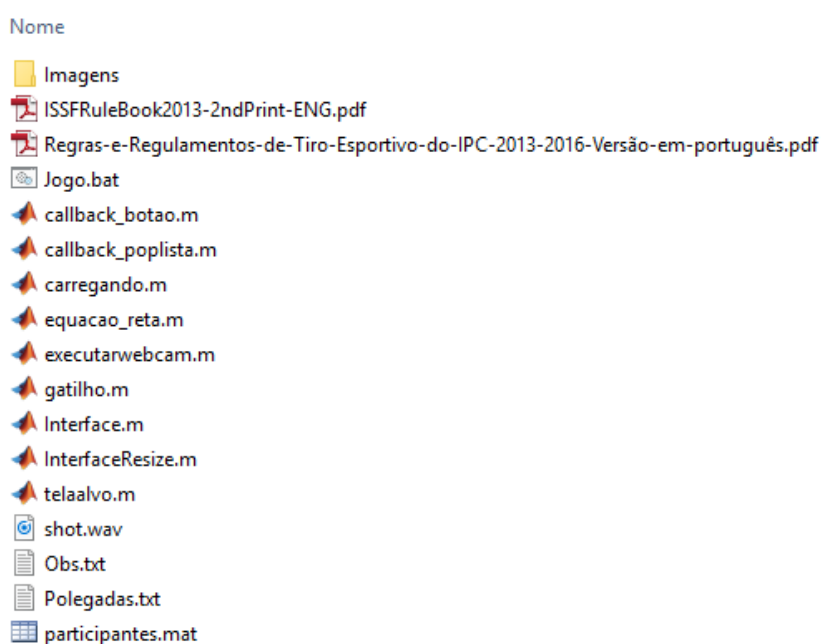


Figura 4: Arquivos e pastas utilizados no programa original.

A pasta “Imagens” contém todas as imagens utilizadas no software, o que inclui os alvos de todas as modalidades de tiro esportivo paraolímpico, as imagens de fundo das interfaces do jogo e uma imagem usada para indicar a marca do tiro na tela. Os documentos com terminação “.pdf” contém os regulamentos da Federação Internacional de Tiro Esportivo e do Comitê Paraolímpico Internacional, sendo a primeira em inglês e a segunda em português, de forma que ambos podem ser acessados pela interface do jogo ou diretamente pelo diretório.

Os arquivos terminados em “.m” são as funções executáveis pela plataforma Matlab e que, em suma, são responsáveis pela execução do jogo em si. O arquivo “jogo.bat” é um executável que abre o Matlab e roda a função “Interface.m”, que executa o software. Essa estratégia permite que o usuário possa realizar os treinos sem a necessidade de interagir com a interface do Matlab diretamente. O arquivo “shot.wav” é um arquivo de som que reproduz o barulho de um disparo toda vez que o gatilho é puxado pelo usuário. O arquivo “Obs.txt” contém as instruções do funcionamento do software, que podem ser lidas diretamente pelo arquivo na pasta raiz, ou em uma das interfaces do jogo. O arquivo “Polegadas.txt” armazena o tamanho da tela atual, dado que é utilizado no software para o cálculo da distância que o usuário deve ficar da tela.

Por último, o arquivo “participantes.mat” contém os dados de todos os usuários do jogo. Ele é modificado através da própria interface do software que permite adicionar, excluir e renomear usuários. Além disso, as pontuações dos treinos são registradas automaticamente ao final dos mesmos. Os dados são armazenados como mostra a Figura 5.

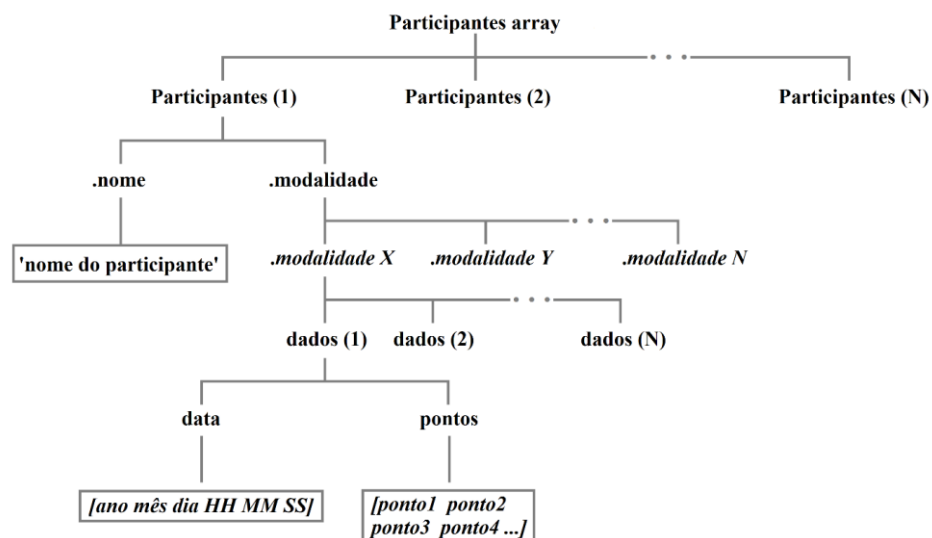


Figura 5: Estrutura de armazenamento de dados utilizada no programa. (MORAES, 2015)

2.1.3 Interfaces do Software:

Quando o software é executado, a primeira interface exibida é a *interface inicial*, que pode ser vista na Figura 6.

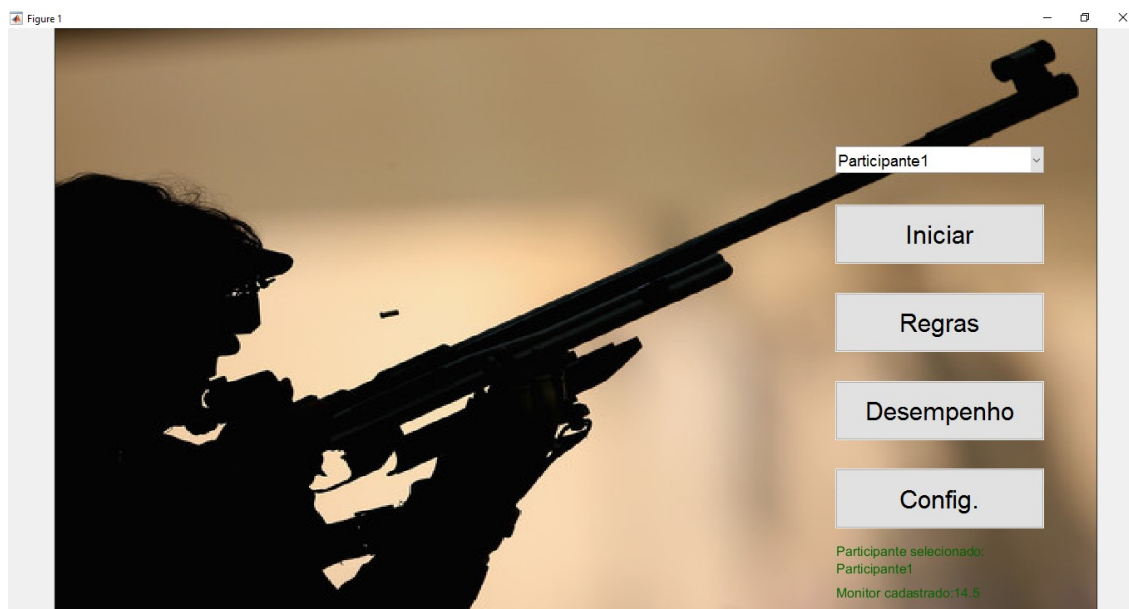


Figura 6: Interface inicial

Essa interface é composta por um menu pop-up que permite selecionar qual dos usuários realizará o treino, além de 4 botões que abrem outras interfaces do jogo e um texto em verde, indicando qual usuário está selecionado no momento, além do tamanho cadastrado do monitor.

Ao clicar no botão “Regras”, é carregada a *interface regras*, que contém um texto explicativo sobre o funcionamento do software. Esse texto está contido no documento de texto “Obs.txt”, mostrado na Figura 4. Além disso, esta tela exibe três botões, sendo que dois deles abrem os regulamentos do International Shooting Sport Federation (ISSF) e do International Paralympic Committee (IPC) usando o leitor de PDF padrão. O último botão permite voltar à *interface inicial*. A *interface regras* pode ser visualizada na Figura 7.

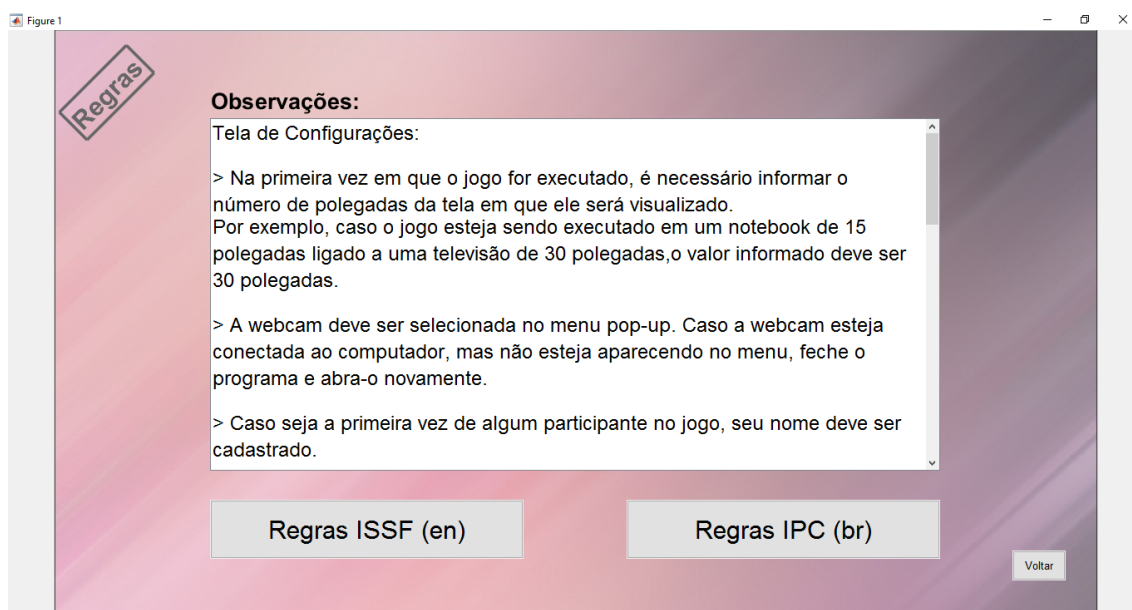


Figura 7: Interface regras

Para um melhor entendimento do usuário, os botões que abrem os regulamentos do ISSF e do IPC serão transferidos para a *interface inicial*. Isso permitirá que a *interface regras* exiba apenas as instruções necessárias para o uso do software. Além disso, o texto do botão “Regras” na *interface inicial* será substituído por “Instruções”, que é mais intuitivo para os usuários.

Na *interface inicial*, o botão “Desempenho” abre a *interface desempenho*, que pode ser vista na Figura 8.

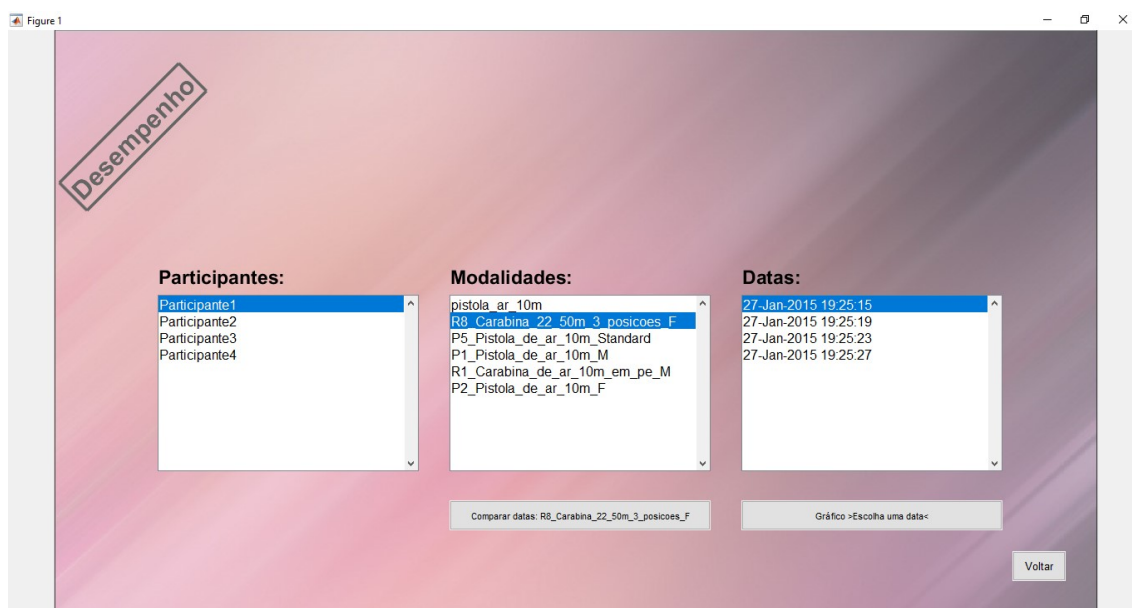


Figura 8: Interface desempenho

Essa interface é composta por 3 listas, a primeira contendo o nome de todos os participantes registrados, a segunda contendo as modalidades em que o participante selecionado já realizou alguma prova, e a terceira contendo as datas dos testes realizados na modalidade selecionada na segunda lista. A segunda lista só é preenchida perante a seleção de um participante na primeira e a terceira lista só é preenchida perante a seleção de uma modalidade na segunda. O botão voltar abre novamente a *interface inicial*. O botão embaixo da segunda lista abre a *interface plot modalidades*, que exibe um gráfico de barras mostrando a pontuação de todos os treinos realizados na modalidade selecionada, permitindo ao usuário avaliar a evolução do seu desempenho ao longo do tempo. A *interface plot modalidades* pode ser vista na Figura 9. Já o botão embaixo da terceira lista abre a *interface plot datas*, que permite a visualização da pontuação de um treino específico através de dois gráficos. O primeiro mostra a pontuação obtida em cada disparo, na ordem em que os mesmos foram realizados, já o segundo exibe a frequência de tiros (eixo das ordenadas) por pontuação (eixo das abscissas). A *interface plot datas* pode ser vista na Figura 10.

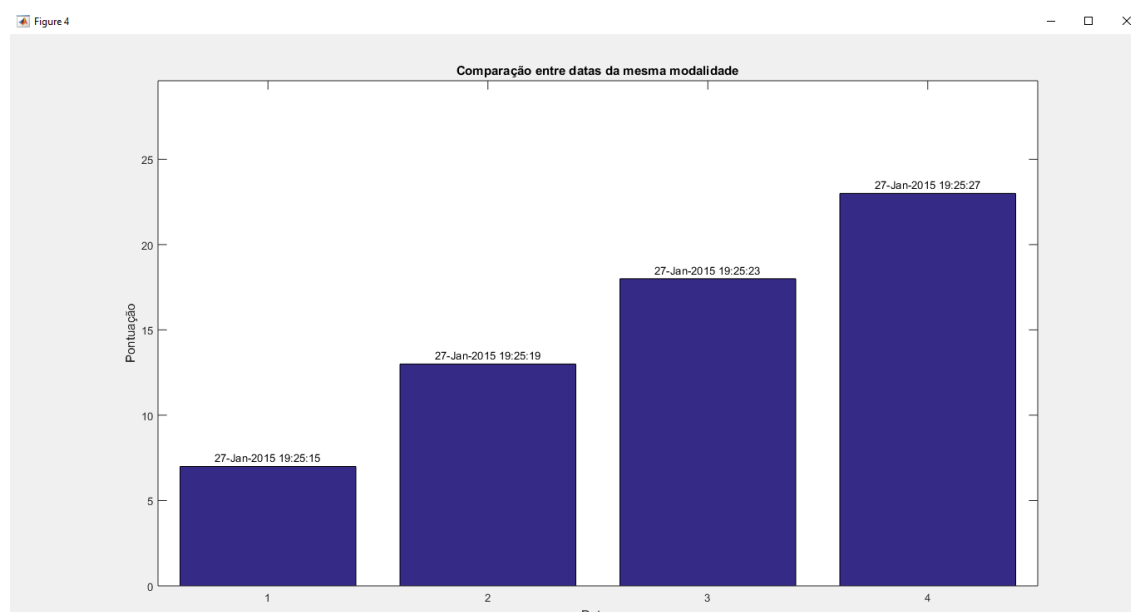


Figura 9: Interface plot modalidades.

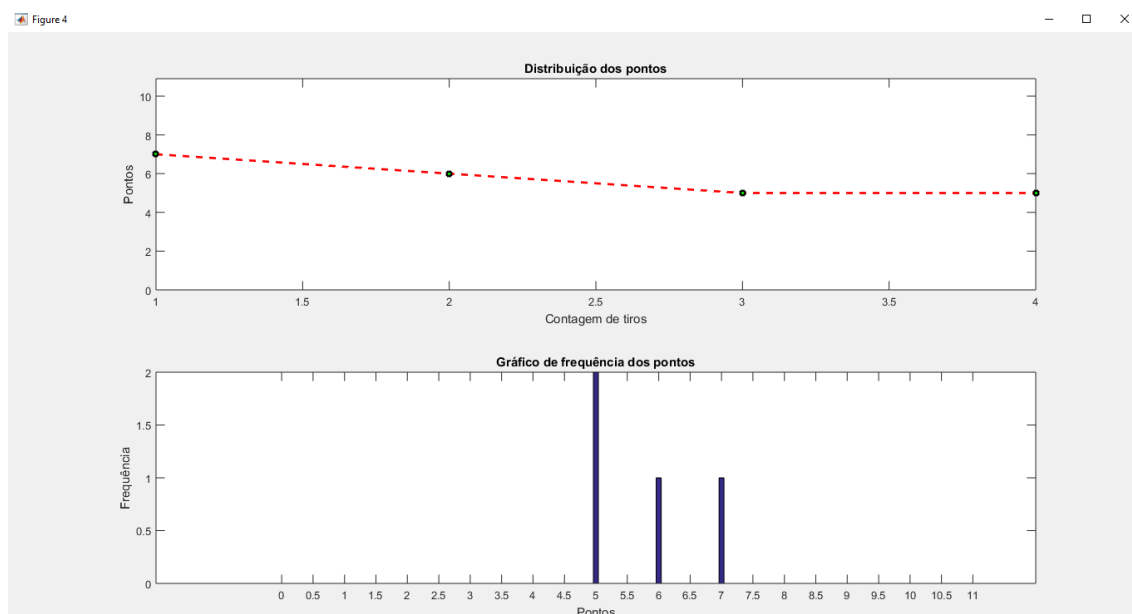


Figura 10: Interface plot datas.

Novamente a partir da *interface inicial*, ao clicar no botão “Config.” é carregada a interface de configurações, que pode ser vista na Figura 11.

Figura 11: Interface de configurações.

Essa Interface é dividida em três seções, monitor, webcam e cadastro. A primeira mostra o tamanho do monitor cadastrado e permite que o usuário altere tal dado, qualquer alteração edita diretamente o arquivo “Polegadas.txt”, que foi apresentado na Figura 4. A segunda seção permite ao usuário visualizar as webcams identificadas pelo Matlab, através de uma menu pop-up, e selecionar qual delas será usada no treino. Por último, a seção cadastro permite ao usuário

ver todos os jogadores cadastrados, renomeá-los, excluir os seus dados e cadastrar novos jogadores.

Para iniciar um treino o usuário deve clicar no botão “Iniciar”, na *interface inicial*. Esse botão abre a *interface modalidades*, que pode ser vista na Figura 12.



Figura 12: Interface modalidades.

Essa interface possui todas as modalidades presentes nos torneios oficiais de tiro esportivo paraolímpico, divididas de acordo com as categorias SH1 e SH2. Os botões de cor azul são modalidades exclusivamente masculinas, os de cor rosa são modalidades exclusivamente femininas e os de cor cinza são modalidades mistas. Ao selecionar uma das opções, o programa seleciona os dados pertinentes daquela modalidade, como alvo, distância, quantidade de disparos e tempo permitido no treino. Em seguida, com o uso destes dados, o software carrega a *interface alvo*, que pode ser vista na Figura 13.

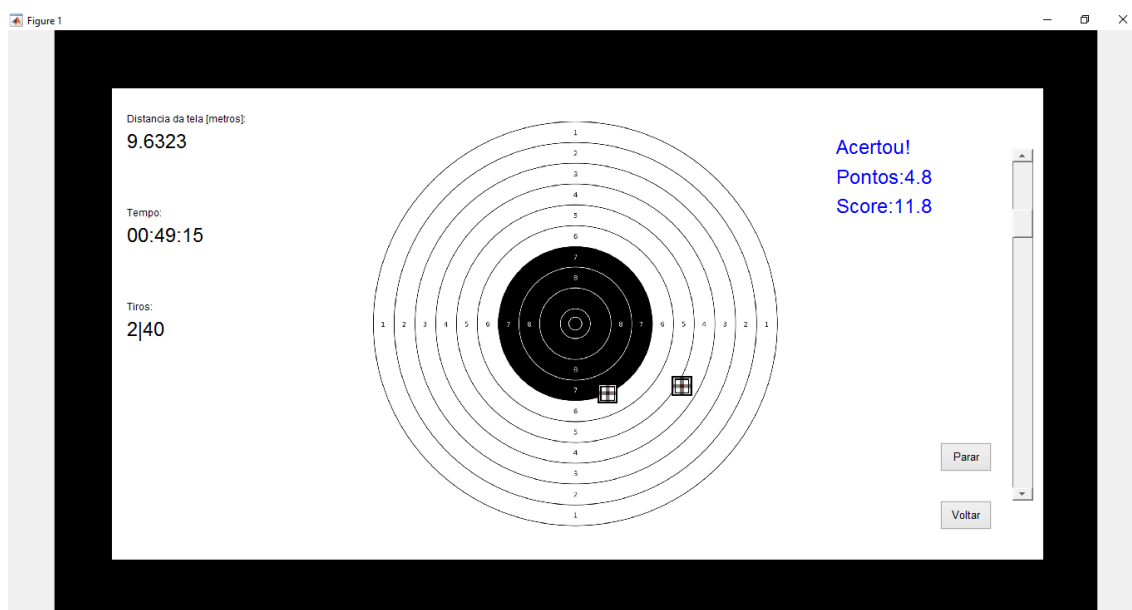


Figura 13: Interface alvo. (MORAES, 2015)

Na interface alvo estão plotados o alvo referente à modalidade escolhida, um slider, alguns objetos de texto e dois botões. O botão “Parar” finaliza o treino, independentemente da quantidade de disparos realizada e do timer, e salva o resultado no arquivo “participantes.mat”. O botão “Voltar” finaliza o treino sem salvar a pontuação e retorna o programa para a *interface modalidades*. O slider é responsável por alterar o tamanho do alvo na interface, alterando assim, consequentemente, a distância que o usuário deve ficar da tela.

São 4 os objetos de texto presentes nesta interface. O objeto de texto com nome “Distância da tela” indica a quantos metros o jogador deve se posicionar da tela, e é atualizado toda vez que slider é arrastado. O objeto com nome “Tempo” é um contador regressivo que indica quanto tempo o usuário ainda tem para concluir o treino. O objeto “Tiros” indica quantos disparos já foram realizados do total disponível para aquela modalidade. Por último, no canto direito superior é apresentada a informação sobre o acerto ou não do último disparo, a sua correspondente pontuação e a pontuação total obtida até o momento.

2.1.4 Funções Usadas no Software

Para a execução do programa são utilizadas nove funções do Matlab, conforme exibido na Figura 4. Para que o programa comece a funcionar a função “Interface.m” tem de ser executada. As demais funções são chamadas à medida que o usuário interage com a interface, sendo que elas se comunicam umas com as outras através de variáveis globais. O papel e atuação de cada uma dessas funções se dá como mostrado a seguir.

2.1.4.1 Função *callback_botao*:

É a função chamada quando qualquer botão (elemento gráfico gerado pelo comando *uicontrol* do tipo *pushbutton*) é pressionado em qualquer interface. Estes elementos possuem várias propriedades que têm de ser fornecidas no momento de sua criação, entre elas está a propriedade *Tag*, que identifica o elemento gráfico e é definida por uma *string*. A função “*callback_botao.m*” recebe essa *string* da propriedade *Tag* como informação de entrada, que é então analisada com o uso do comando *switch case* para determinar qual botão foi pressionado. Aqui estão contidas todas as rotinas a serem executadas para cada botão pressionado nas interfaces do software.

2.1.4.2 Função *callback_poplista*:

Essa função é semelhante à função “*callback_botao.m*”, porém ela é chamada quando o usuário interage com as listas presentes nas interfaces, ao invés dos botões. Essas listas são elementos gráficos do tipo *popupmenu*, também gerados pelo comando *uicontrol*. A função também usa do comando *switch case* para determinar com qual lista o usuário está interagindo e executar as rotinas pertinentes.

2.1.4.3 Função *carregando*:

A função “*carregando.m*” é executada toda vez que o software carrega uma nova interface. Ela fecha a janela com a interface aberta e exibe uma imagem presente na pasta “Imagens” (Figura 4) com a frase carregando, enquanto faz a transição de interfaces. Além disso, ela também limpa os dados da Command Window, do inglês janela de comandos.

2.1.4.4 Função *equacao_reta*:

Essa função tem como entrada as coordenadas de 4 pontos, na forma de dois vetores, um com os valores das abscissas e o outro com os valores das ordenadas. Esses pontos pertencem ao quadrilátero que representa a tela nas imagens capturadas pela câmera. A função processa esses dados e retorna as equações das quatro retas que ligam esses pontos, retas essas que nada mais são que os lados do quadrilátero.

2.1.4.5 Função *executarwebcam*:

Essa função, antes de todo treino, abre uma janela *Video Preview*, que exibe o vídeo sendo capturado pela câmera, de forma a mantê-la aberta durante todo o treino. Isso é feito para que o hardware de captura de imagens possa ajustar seus parâmetros internos para se adaptar à situação do ambiente, tanto de claridade quanto de distância da tela. Sem a execução prévia da câmera, a mesma só seria iniciada no momento que algum disparo fosse realizado, não havendo assim tempo para ela se adaptar à luminosidade ambiente. Com isso, as imagens capturadas

seriam de baixa qualidade, dificultando o seu processamento e aumentando a possibilidade de falhas.

2.1.4.6 Função gatilho:

Essa função é executada toda vez que um disparo é realizado pelo usuário na *interface alvo*. Ela realiza a captura e tratamento das imagens da webcam, além de realizar os devidos cálculos, determinando se o usuário acertou ou não e, caso positivo, em qual ponto do alvo o impacto teria atingido. A função também verifica o tempo restante no timer e, caso este haja findado, a mesma é interrompida. Ela também é interrompida quando é verificado que a quantidade de disparos realizados atingiu a quantidade limite de disparos previstos para aquela modalidade.

2.1.4.7 Função Interface:

É a função que deve ser executada para dar início ao programa. Ela carrega a imagem completamente escura usada na *interface alvo* para a identificação da tela através da webcam, além de carregar a *interface inicial*, com a sua imagem de fundo.

2.1.4.8 Função InterfaceResize:

É a função responsável por criar todos os elementos gráficos do software, com os quais o usuário interagirá. Também é responsável por dimensionar a interface, garantindo a proporcionalidade da imagem de fundo da interface e dos seus elementos gráficos. Isso garante que o software vai se apresentar de maneira adequada em qualquer tamanho de tela em que ele seja executado.

Como a função tem de atuar na criação de todas as interfaces do jogo, ela foi dividida em blocos, um para cada uma dessas interfaces. Esses blocos podem ser vistos na Figura 14.

```
function InterfaceResize(objectHandle , eventData)

%% Inicio Declaração Global de Variáveis: %%...%%
%% Fim Declaração Global de Variáveis

%% Inicio Código em Comum: %%...%%
%% Fim Código em Comum

%% Inicio Interface: %%...%%
%% Fim Interface

%% Inicio Tela de Modalidades (botao iniciar): %%...%%
%% Fim Tela de Modalidades (botao iniciar)

%% Inicio Regras: %%...%%
%% Fim Regras

%% Inicio Configurações: %%...%%
%% Fim Configurações

%% Inicio Desempenho: %%...%%
%% Fim Desempenho

%% Inicio Tela Alvo: %%...%%
%% Fim Tela Alvo
```

Figura 14: Blocos de comandos da função “InterfaceResize.m”. (MORAES, 2015)

Cada um desses blocos contém os comandos necessários para criar os elementos gráficos da interface especificada no título do bloco, com exceção dos dois primeiros, “Declaração Global de Variáveis” e “Código em Comum”, que realizam a declaração e processamento das variáveis utilizadas nos demais blocos.

Para determinar qual bloco será chamado, o software utiliza uma variável global chamada de *tag*, que é do tipo *char*. Essa variável contém um nome indicando qual interface está sendo chamada pelo usuário. Com o uso do comando *if* verifica-se a qual interface a variável se refere e são executados os comandos pertinentes.

2.1.4.9 Função *telaalvo*:

Essa função é executada quando a *interface alvo* é aberta ou quando ela precisa ser alterada, ou seja, ela é executada quando uma modalidade é selecionada na *interface modalidades* e quando o usuário move o slider na própria *interface alvo*. Na sua primeira execução ela plota a imagem branca de bordas pretas, que pode ser vista na Figura 13, e determina as coordenadas dos 4 cantos da parte branca da imagem, além de, juntamente com a função “InterfaceResize.m”, criar os demais elementos gráficos.

Quando o usuário interage com o slide, o software altera o tamanho do alvo na tela. Este tamanho pode variar entre o tamanho máximo que cabe na tela, que é 80% da altura da mesma, devido à presença das bordas pretas, até 10% da altura total. Quando o tamanho do alvo na tela é alterado, a função “telaalvo.m” têm de determinar a nova distância que o usuário deve ficar da tela, para isso ela calcula o tamanho real do alvo na tela, em milímetros. Isso é feito aferindo-se o tamanho do alvo em pixels e convertendo esse valor de pixels para milímetros. Para essa conversão, é aferido as dimensões totais da tela em pixels, tanto em X quanto em Y, os quais são usados para calcular o tamanho da diagonal da tela, também em pixels. De posse desse valor e do tamanho real da diagonal em polegadas da tela, que foi informado pelo usuário na *interface configurações*, é possível converter os valores de pixels da tela em polegadas e, em seguida, em milímetros.

O software já possui as informações do tamanho real que caracteriza o alvo e da distância real que o usuário deve se posicionar do mesmo, informações essas tiradas do livro de regras da ISSF. Agora, já de posse do tamanho do alvo virtual na tela, em milímetros, o software consegue calcular a distância que o usuário deve estar da tela com o uso de semelhança de triângulos. Para esse cálculo, considera-se que a distância real está para o diâmetro real assim

como a distância da tela está para o tamanho virtual do alvo na tela, resultando em uma equação simples, representada a seguir:

$$Distância_{tela} = \frac{Distância_{real} * Diâmetro_{tela}}{Diâmetro_{real}}$$

Além do já citado, a função “telaalvo.m” também é responsável por controlar o contador regressivo usado na *interface alvo*.

2.1.5 Exemplos das funções nas interfaces:

Todas as funções citadas acima se comunicam entre si, criando as interfaces, processando os dados e fazendo com que o software funcione de maneira adequada. Um exemplo do funcionamento está ilustrado na Figura 15 a seguir.

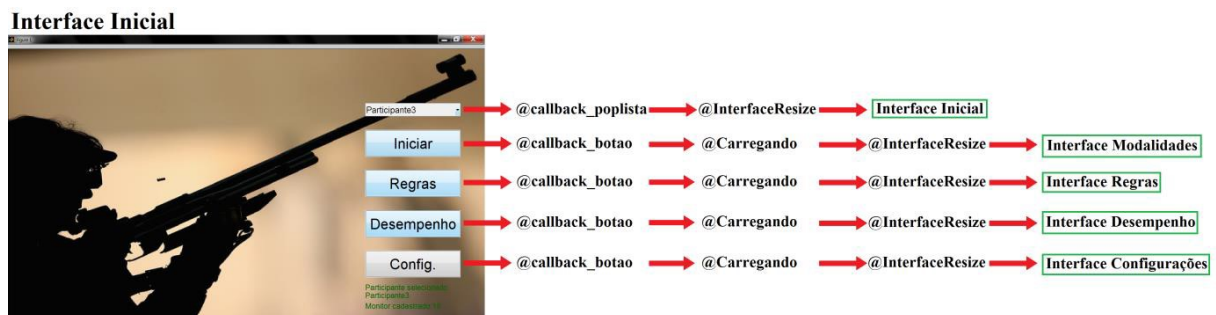


Figura 15: Sequência de funções associadas à interface inicial. (MORAES, 2015)

Como pode ser visto, todos os elementos de controle do tipo *pushbutton* chamam a função “callback_botao.m”. Todos os comandos que levam à alteração de interface têm de chamar a função “carregando.m”. Para estruturar uma nova interface, é utilizado a função “InterfaceResize.m”. A lista de participantes, que é um elemento do tipo *popupmenu*, chama a função “callbackpoplista.m” e na sequência não tem de chamar a função “carregando.m”, já que as alterações feitas pela interação com esse objeto são apresentadas na própria *interface inicial*.

Outro exemplo da interação entre as funções pode ser observado na *interface alvo*, na Figura 16.

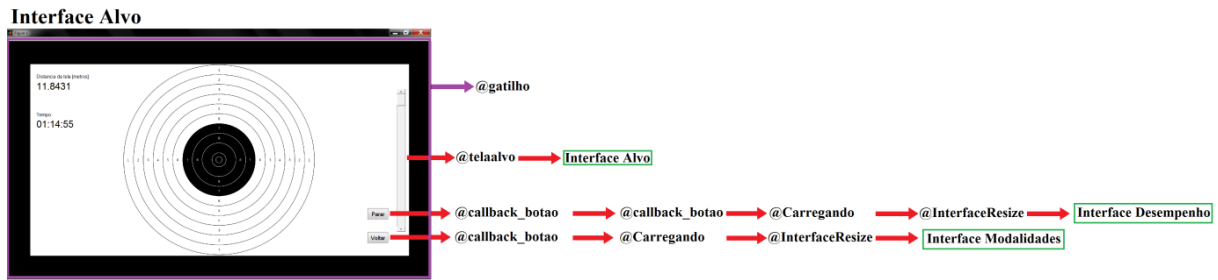


Figura 16: Sequência de funções associadas à interface alvo. (MORAES, 2015)

Assim como na *interface inicial*, os dois botões chamam a função “callback_botao.m”, seguido da função “carregando.m”, já que uma nova interface terá de ser carregada após a interação do usuário com esses elementos. As novas interfaces serão estruturadas com o uso da função “InterfaceResize.m”, que vem em seguida na lista de funções executadas. Interações com a barra lateral serão processadas pela função “telaalvo.m”, que é capaz de alterar o tamanho do alvo e a informação sobre a distância do usuário à tela sem o auxílio da função “InterfaceResize.m”. Nessa interface, qualquer clique que não seja nos elementos de controle já citados chama a função “gatilho.m”, que interpreta esse clique como um disparo da arma, capturando as duas imagens necessárias e processando elas para, em seguida, mostrar as informações pertinentes.

2.1.6 Teste de Precisão e Conclusão

Em sua versão original, o software foi testado por MORAES (2015) usando o esquema mostrado na Figura 17.



Figura 17: Bancada de teste do programa original. (MORAES, 2015)

Com a bancada acima foram realizadas quinze simulações de disparos em três posições diferentes, sendo cinco disparos por posição. Em todas as três posições a câmera estava sobre a marcação mostrada na mesa, que foi posicionada a um metro de distância da tela. Com isso, foram obtidas as coordenadas dos disparos na tela executando o software na plataforma Matlab. Os resultados obtidos estão apresentados na Tabela 1:

Tabela 1: Resultados do teste de calibração. (MORAES, 2015)

Análise Estatística	Tiro	X [pixel]	Y [pixel]
Teste 1	1	2136,500	624,233
	2	2136,500	624,233
	3	2131,800	619,892
	4	2136,500	624,233
	5	2136,500	624,233
Média		2135,560	623,365
Desvio Padrão		1,880	1,736

Análise Estatística	Tiro	X [pixel]	Y [pixel]
Teste 2	6	2020,900	618,345
	7	2020,900	618,345
	8	2020,900	618,345
	9	2020,900	618,345
	10	2020,900	618,345
Média		2020,900	618,345
Desvio Padrão		0,000	0,000

Análise Estatística	Tiro	X [pixel]	Y [pixel]
Teste 3	11	1755,100	622,587
	12	1755,000	619,695
	13	1755,100	622,587
	14	1755,100	622,587
	15	1755,000	619,695
Média		1755,060	621,430
Desvio Padrão		0,055	1,584

A análise dos resultados apresentados na tabela mostra que o desvio padrão obtido em cada teste é mínimo quando comparado com a média. Em termos percentuais, os valores obtidos para o desvio padrão em relação à média foram 0,088% em X e 0,279% em Y para o primeiro teste; 0% tanto em X quanto em Y para o segundo teste; e 0,003% em X e 0,255% em Y para o terceiro teste. Assim sendo, considera-se que os resultados obtidos foram satisfatórios, mesmo utilizando um equipamento de captura de imagens de baixo custo.

O trabalho de MORAES (2015) ainda conclui falando de possíveis trabalhos futuros, onde ele cita a necessidade de criar um acessório que permita acoplar a webcam na arma sem atrapalhar o manuseio da mesma. Ele sugere ainda que a webcam seja sem fio, para garantir ao usuário maior liberdade de movimento.

2.2 Dispositivos de Tiro com Câmeras:

Já houveram diversos jogos de tiro que utilizaram da captura de imagens ou de sinais luminosos para o seu funcionamento. Essa modalidade de jogos apareceu primeiramente como jogos eletromecânicos, cuja interface era composta por circuitos elétricos e componentes mecânicos. Posteriormente essa tecnologia evoluiu em videogames completamente eletrônicos. Seguem alguns exemplos desses jogos que marcaram a sua época:

2.2.1 Seeburg Ray-O-Lite

Lançado em 1936, o Ray-O-Lite foi o primeiro jogo de tiro que funcionava com a tecnologia de um emissor e de um captador de luz. Ele era um jogo eletromecânico onde um alvo no formato de um pato se movia em um trilho, com um sensor de luz acoplado a ele. O objetivo do jogador era atirar no pato com uma arma que, quando o gatilho era pressionado, emitia um raio de luz. Se o tubo sensor de luminosidade acoplado ao pato captasse tal luz, o tiro era contabilizado e o pato caía. O gabinete de madeira que continha a interface eletromecânica do Seeburg Ray-O-Lite pode ser visto na Figura 18. (Bueschel, 1995)



Figura 18: Gabinete e Interface do Seeburg Ray-O-Lite, de 1936. (Bueschel, 1995)

A Seeburg era uma companhia que, já na década de 30, fabricava fonógrafos e juke-boxes. Eles já possuíam a tecnologia e a competência para fabricar os gabinetes de madeira, os jogos de engrenagens e os amplificadores de tubo a vácuo utilizados pelo Ray-O-Lite. Isso possibilitou que a companhia fosse a pioneira a atuar neste ramo de negócios. (Escritores do site Jukeboxhistory.info, 2010)

2.2.2 Magnavox Odyssey Light Gun:

O Odyssey nasceu da ideia de se jogar videogames em casa, utilizando para isso televisores. Ele foi o primeiro console feito para ambientes domésticos e sua criação só foi possível com a associação dos seus criadores, Ralph H. Baer, Bill Rudh e Bill Harrison com a Magnavox, companhia que fabricava televisores. Apesar de os protótipos do Odyssey terem sido testados já em 1967, ele só foi oficialmente lançado em 1972, cinco anos depois (Gegan & Winter, 2012). A Figura 19 apresenta duas pessoas jogando o jogo de tênis de mesa do Magnavox Odyssey.

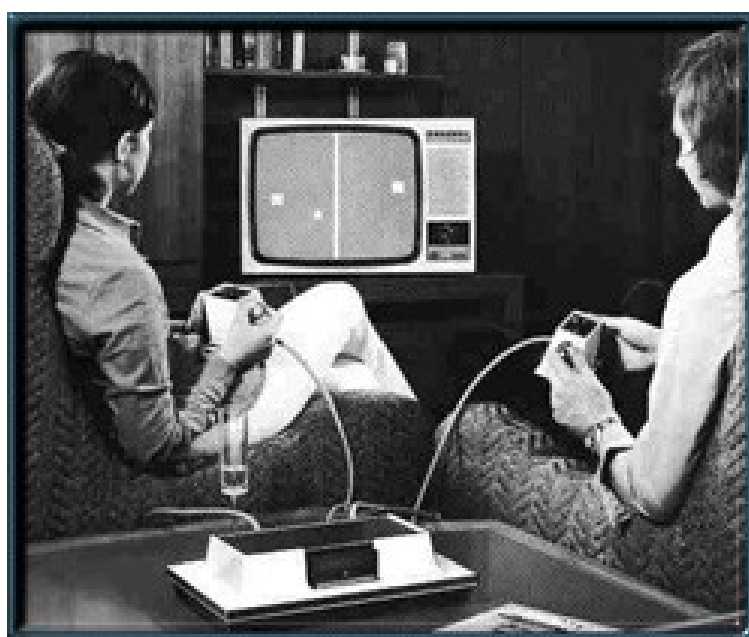


Figura 19: Odyssey rodando o seu famoso jogo de tênis de mesa. (Gegan & Winter, 2012)

Os jogos lançados eram bem simples e sem som, se tratavam apenas de pixels brancos em um fundo de tela escuro que se comportavam de maneiras diferentes dependendo do jogo que estivesse sendo executado. Para complementar essa interface, o Magnavox Odyssey era vendido com diferentes películas de plástico, que eram colocadas na frente da tela, para trazer um tom mais real para os jogadores. As películas não eram utilizadas em todos os jogos, apenas em alguns. Os jogos de tiro eram uma exceção, já que todos eles utilizavam películas.

Para os jogos de tiro, o usuário deveria comprar a Shooting Gallery, que era um pacote extra ao jogo que continha um rifle bem realista feito com materiais de qualidade, dois cartuchos para o console, quatro películas, um manual de instruções e um cartão chave de inspeção. O conjunto completo pode ser visto nas Figuras 20 e 21.



Figura 20: Películas plásticas do Odyssey usadas nos jogos de tiro. Jogos correspondentes: (A) Dogfight; (B) Prehistoric Safari; (C) Shootout; (D) Shooting Gallery. (Gegan & Winter, 2012, p. Shooting Gallery)



Figura 21: Conjunto Shooting Gallery do Magnavox Odyssey. (Gegan & Winter, 2012, p. Rifle)

O rifle possuía um receptor de luz que reagia aos pixels brancos da tela. Portanto, se o gatilho fosse puxado enquanto ele apontava para uma luz branca, o videogame computava um acerto. A principal falha desse sistema é que se o usuário mirasse em uma lâmpada e disparasse, o acerto também seria computado, abrindo espaço para trapaças do jogador. Além disso, o rifle era conectado ao console por um fio, que acaba restringindo um pouco o seu movimento.

2.2.3 Nintendo Zapper (Duck Hunt)

O Nintendo Zapper foi criado em 1984 como um acessório para o Nintendo Entertainment System (NES), tratava-se de um hardware no formato de um revólver com design futurístico, que pode ser visto na Figura 22.



Figura 22: Nintendo Zapper. (Sic!, 2007)

Semelhantemente ao rifle do Odyssey, o Nintendo Zapper é capaz de detectar a presença ou ausência de luminosidade em um ponto específico, ponto esse para o qual ele estiver mirado. Para explicar melhor o seu funcionamento, usa-se de exemplo o jogo mais famoso que o utilizava, o Duck Hunt. O jogo Duck Hunt foi lançado em 1984 pela NintendoTM para o NES e fez muito sucesso, sendo até hoje referência em jogos de tiro. Com os avanços tecnológicos ocorridos desde o Magnavox Odyssey, os televisores já conseguiam reproduzir cores, permitindo que o software apresentasse uma interface colorida e mais complexa, onde patos selvagens levantam voo em meio a um cenário pantanoso e o usuário tinha que acertá-los antes que os mesmos saíssem da tela. A Figura 23 mostra a interface do jogo em funcionamento.



Figura 23: Pato sendo acertado no jogo Duck Hunt. (Toad_Speedgaming, 2007 [arquivo de vídeo])

Apesar dos gráficos coloridos já suportados pelo NES e pelos televisores da época, o Nintendo Zapper era incapaz de determinar a cor exata do ponto para o qual ele estava apontado, sua capacidade se limitava a determinar se naquele ponto havia luz ou não. Por tanto, quando o gatilho era pressionado o jogo projetava um frame completamente preto e, no frame seguinte, apareciam pixels brancos no local onde estava o pato na tela. Dessa forma, se o Zapper Gun detectasse um frame preto seguido de um branco, o jogo computava um acerto. Esse sistema de determinação de acerto impedia que o usuário trapaceasse apontando a arma para uma lâmpada, pois nesse caso o frame preto não seria detectado.

No caso de terem 2 patos na tela, o jogo poderia utilizar um frame a mais para verificar se algum pato havia sido acertado e, se sim, qual deles. O funcionamento do jogo se dava seguinte forma nessa situação: quando o gatilho era puxado, o frame preto era exibido normalmente, seguido dos pixels brancos que representavam apenas o primeiro pato; se o software computasse um acerto, o jogo continuava normalmente; apenas se um acerto não fosse computado que os pixels brancos representativos do segundo pato eram exibidos, se o zapper

detectasse o frame preto seguido de um branco nesse momento, então o segundo pato que havia sido acertado. A Figura 24 mostra os frames utilizados pelo jogo no caso de haverem dois patos na tela e o segundo deles ser acertado.

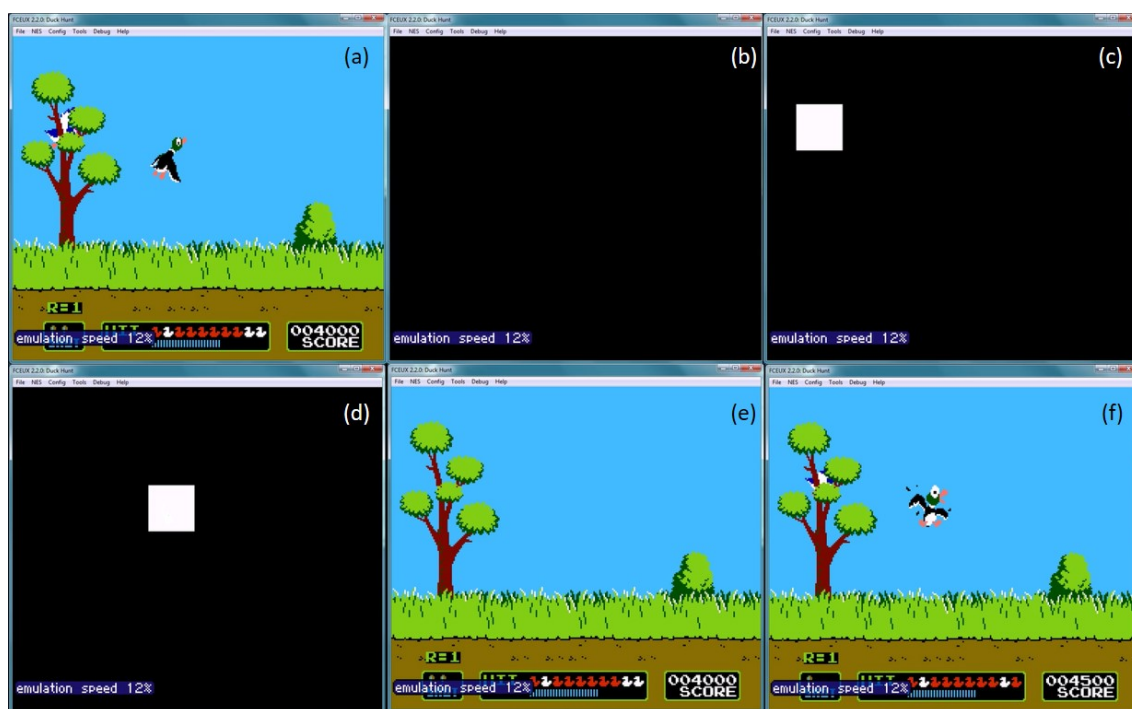


Figura 24: Sequência de frames seguintes a um tiro no jogo Duck Hunt. (a) Logo antes do tiro, um dos patos está atrás da árvore; (b) Tela escura; (c) 1º pato sendo destacado (a árvore não interfere no acerto); (d) 2º pato sendo destacado; (e) Delay antes de os patos serem mostrados novamente; (f) Representação de qual pato foi acertado. (Ramy32.exe, 2013 [arquivo de vídeo])

O Nintendo Zapper funcionava, portanto de uma forma muito parecida com o do já citado Magnavox Odyssey, com algumas melhorias que foram implementadas usando contagem dos frames e projeções de telas monocromáticas nos instantes dos tiros. Com isso as trapças dos usuários foram inibidas; o jogo funcionava normalmente, mesmo com uma interface colorida; e os frames monocromáticos eram mostrados tão rapidamente que eram quase imperceptíveis aos jogadores.

2.2.4 Super Scope / Nintendo Scope:

O Super Scope, também conhecido como Nintendo Scope, foi lançado em 1992 como um acessório do console Super Nintendo Entertainment System (SNES). Assim como o SNES é o sucessor do NES, o Super Scope é o sucessor do Nintendo Zapper. Ele é composto

basicamente por duas partes, a arma em si, também designada de emissor de sinal, e um receptor de sinal, que permite que a arma se comunique com o console sem a necessidade de fios. O dispositivo que atua como receptor é ligado diretamente no console e deve ser posicionado em cima da televisão. Os dois dispositivos comunicam entre si através de raios infravermelhos emitidos pela arma e captados pelo dispositivo receptor de sinal, que passa as informações recebidas para o console (Nintendo®, 1993, pp. 4-1-1). Eles podem ser vistos na Figura 25.



Figura 25: SNES Super Scope e o seu receptor. (The Strong, 2016)

Diferentemente do seu antecessor, o Nintendo Zapper, que é apenas capaz de determinar se existe ou não de sinal luminoso em uma pequena área para a qual estiver apontado, o SNES Super Scope é capaz de determinar com precisão o ponto da tela para o qual ele está apontado. Porém, ele só funciona em televisores que usam tubo de raio catódico, devido à maneira como esses televisores projetam imagens na tela. (Nintendo®, 1993, pp. 4-1-1)

Para entender como o sistema de mira do SNES Super Scope funciona, precisa-se primeiro entender o funcionamento dos televisores que operam com tubos de raios catódicos, ou televisores CRT (do inglês, Cathode Ray Tube). A tela de vidro desses televisores é revestida internamente com uma camada fluorescente composta por fósforo, camada essa que emite luz ao ser bombardeada por elétrons. Acoplado nessa tela há um tubo, que tem um formato afunilado e termina em um canhão de elétrons acoplado a bobinas defletoras. Esse sistema pode ser visto na Figura 26.

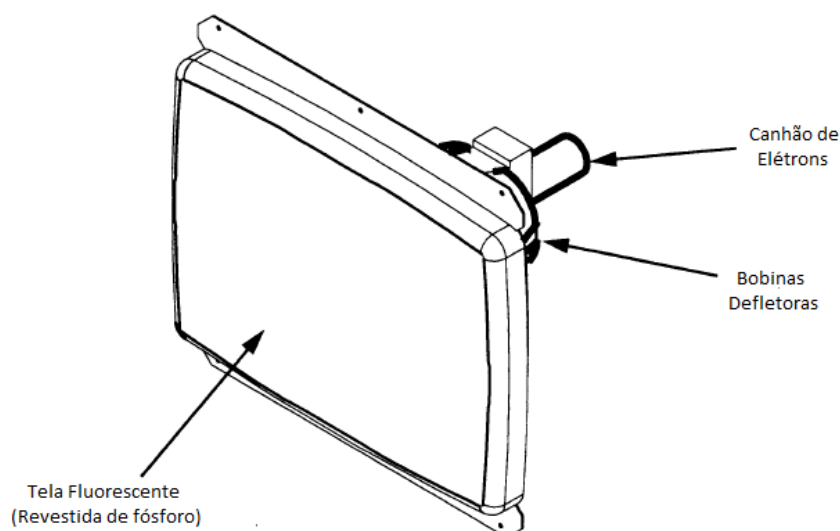


Figura 26: Esquema de um televisor de tubo de raio catódico. (Nintendo®, 1993, pp. 4-2-1)

O revestimento interno de fósforo da tela emite luz quando é bombardeado por elétrons. Assim, para iluminar a tela, o canhão de elétrons emite um raio de elétrons que atinge um ponto nesta tela fluorescente, iluminando o mesmo em uma cor específica. Este raio é então defletido pelas bobinas defletoras, que são capazes de alterar a sua trajetória e, conseqüentemente, o ponto na tela que será iluminado. O raio percorre então a tela linha por linha, da esquerda para a direita, começando no canto esquerdo superior e acabando no canto direito inferior, criando cada imagem que compõe a animação, pixel por pixel. Apesar de cada pixel se apagar praticamente instantaneamente, a criação das imagens é tão rápida que esse processo é imperceptível para o olho humano. (Nintendo®, 1993, pp. 4-2-2)

O videogame SNES projeta 60 imagens por segundo na tela, o que significa que a cada 16,67 milissegundo (1/60 de segundo) o feixe de elétrons percorre toda a região da tela, iluminando-a pixel por pixel. O sensor de luminosidade do SNES Super Scope capta o sinal luminoso em apenas uma pequena área da tela, área essa para qual ele está apontado. Dessa forma, no momento exato em que essa pequena área é iluminada pelo feixe de elétrons, o SNES Super Scope emite um sinal para o captador de sinal que está ligado ao console. O console registra o valor da coordenada do raio de elétrons daquele instante e assim determina o ponto exato para a qual o SNES Super Scope está apontado. (Nintendo®, 1993, pp. 4-2-1 a 4-2-3)

Como essa nova tecnologia permitia determinar o ponto exato da tela para a qual a arma estaria apontada, foi possível o desenvolvimento de jogos mais elaborados que aqueles do seu antecessor, o Nintendo Zapper do NES.

3 Desenvolvimento do Hardware:

O desenvolvimento deste trabalho parte do que já foi realizado no trabalho anterior de MORAES (2015) no seu estudo intitulado “Estudo e Desenvolvimento de Jogos Aplicados ao Treinamento de Atletas Paralímpicos”, que foi detalhado na revisão bibliográfica. O objetivo aqui é analisar o que foi desenvolvido em comparação com as tecnologias já existentes, determinando as diretrizes para o desenvolvimento de qualquer hardware que seja necessário, visando realizar as alterações necessárias para o seu bom funcionamento.

3.1 Comparação de sistemas de mira de jogos de tiro:

As provas de tiro esportivo oficiais são divididas em diferentes modalidades, sendo que na maioria delas o objetivo é acertar um alvo, de modo que quanto mais central for o impacto neste alvo, maior será a pontuação obtida. Das tecnologias vistas e citadas na revisão bibliográfica a única que poderia ser usada para o desenvolvimento de uma plataforma que suportasse esse tipo de prova seria a do SNES Super Scope, por ser capaz de determinar o ponto exato da tela para o qual o usuário está apontado no momento do disparo. Porém, isso implicaria na necessidade de usar um televisor CRT, que é ultrapassado e cada vez mais difícil de ser encontrado, o que inviabilizaria o projeto. Sendo assim, concluiu-se que para atingir os objetivos propostos neste trabalho seria necessário adotar uma tecnologia diferente e que funcione adequadamente nos modernos televisores digitais de tela plana, que formam imagens na tela de maneira contínua.

Desta forma, e considerando o exposto acima, a tecnologia de captação e tratamento de imagem sugerida no trabalho de MORAES (2015) se apresenta como um método alternativo e eficaz para abordar esse problema. Ressalta-se que esta estratégia carece de maior esforço computacional, decorrente da necessidade do processamento das imagens captadas, que são bem mais complexas que os sinais luminosos usados pelos hardwares dos jogos de tiro do passado. Entretanto, considerando a evolução da capacidade de processamento dos computadores pessoais atuais, isso não representa um problema.

3.2 Projeto do Hardware:

Seguindo a linha de raciocínio sugerida no tópico de “Trabalhos Futuros” do projeto de MORAES (2015), a câmera deveria ser fixada na ponta do cano da arma, de modo a ficar

perfeitamente alinhada com a linha da trajetória do projétil. Porém, devido à complexidade para fazer esse acoplamento com precisão e à possibilidade de se danificar o cano, o que poderia prejudicar permanentemente a precisão da arma, foi pensada uma solução alternativa: acoplar a câmera nas partes externas da arma, de preferência nas áreas destinadas ao acoplamento de dispositivos como trilhos, etc. As carabinas esportivas geralmente possuem locais onde alguns dispositivos podem ser acoplados, como por exemplo o trilho modelo picatinny que pode ser visto na Figura 27. Tanto a alternativa sugerida por MORAES (2015) quanto esta última serão exploradas nas próximas etapas deste trabalho.



Figura 27: Exemplo de trilho para acoplamento de dispositivos em carabinas. (Johnson, 2014)

No caso da câmera não ser fixada no cano da arma, é inviável garantir o alinhamento da mesma com a trajetória do projétil. Portanto, esse tipo de fixação dependeria de uma calibração, para que o programa saiba em que ponto da imagem capturada o projétil teria acertado. Como em toda calibração, faz-se necessário um padrão, que nada mais é do que algo que mostre para o software onde está este ponto de impacto.

O padrão não poderia ser baseado na capacidade de mira do próprio usuário, pois este é o fator a ser treinado pelo software e não há como garantir com certeza a sua exatidão. Qualquer desvio cometido, por menor que fosse, geraria um erro sistemático que seria imperceptível ao usuário, mas afetaria negativamente seu desempenho em treinamento. Para contornar esta dificuldade, sugere-se fazer uso de um acessório capaz de apontar com precisão o alinhamento

do cano e o local de impacto do projétil, sem necessidade de um disparo real, são os chamados “boresighter” ou colimadores a laser. Trata-se de um dispositivo com o formato de uma munição que é carregado na câmara da arma e que emite um feixe de laser que indica o direcionamento do cano. Utilizando processamento de imagens, é possível fazer o software procurar na região da tela todos os pixels com a cor do laser, determinando assim onde o laser estaria na imagem capturada e, conseqüentemente, calibrando o sistema. Um exemplo de colimador a laser pode ser visto na Figura 28.



Figura 28: Munição laser para armas de calibre 0,45. (Volk, 2014)

A partir desta análise, pode concluir-se que existem dois métodos viáveis de fixação da câmera na arma, o que leva à necessidade de comparação entre os dois. Esta comparação pode ser vista na Tabela 2 a seguir:

Tabela 2: Comparação entre os métodos de fixação da câmera na arma.

Item	Câmera no cano	Câmera nos trilhos
Fixação	Carece de um estudo detalhado, para que não haja risco de danificar a parte interna do cano e afetar a precisão da arma, que é crucial ao esporte.	Utiliza de um suporte já existente para encaixe de acessórios. Caso esse suporte não exista, a câmera pode ser acoplada em qualquer área externa da arma.
Alinhamento da câmera	O alinhamento deve ser garantido na fabricação do equipamento, para que sempre que acoplada, a câmera esteja alinhada com a trajetória do projétil.	A arma não precisa estar perfeitamente alinhada com a linha de centro do cano, basta que a tela do jogo seja capturada pela câmera no momento do disparo.
Calibração	É inviável, dado que a câmera estaria obstruindo o cano, tornando impossível o uso do colimador laser para calibrar o sistema. Essa montagem não pode requerer que o usuário calibre a arma.	É necessária, dado que a câmera não estará alinhada com a trajetória do projétil. Requer uma etapa de calibração antes de qualquer treino
Manutenção	Carece de um estudo para analisar a possibilidade do acoplamento entre a câmera e o cano desalinharem com o uso.	Não carece de nenhuma manutenção, fora aquela necessária a cada parte individual do conjunto.
Custo físico extra	O suporte vai requerer uma usinagem com tolerância bem restrita, dada a necessidade de um alinhamento preciso entre a câmera e o cano da arma, o que pode encarecê-lo.	O colimador a laser usado na calibração do sistema tem de ser de alta qualidade, para que o alinhamento com a linha de centro do cano seja preciso e sem desvios de paralelismo. O colimador a laser contribui para aumentar o custo do equipamento.

A partir da análise consolidada pelos dados da Tabela 2, foi selecionada a opção de fixação da câmera nos trilhos. Apesar dessa montagem requerer um maior esforço computacional para sua execução, esse esforço não há de deixar o treino em si mais lento, pois

ele só será realizado durante a calibração. Uma vez realizada a calibração, o resultado é armazenado em uma variável, passado para a função “gatilho.m” (Fig. 4) e a arma permanecerá calibrada durante toda a seção de treino.

3.2.1 Hipóteses assumidas:

Para o desenvolvimento deste projeto, algumas hipóteses tiveram de ser assumidas, tanto físicas quanto em relação à rotina de treino. São elas:

3.2.1.1 *Linearidade da trajetória do projétil:*

Nas provas do tiro esportivo paraolímpico, o atleta fica a uma distância de no máximo 50 metros do alvo. Considerando-se a alta velocidade do projétil, supõe-se que essa distância seja muito pequena para que ocorra qualquer interferência significativa da gravidade em sua trajetória. Considera-se também que não haja correntes de ar ambiente com intensidade suficientemente alta para afetar a trajetória do projétil. Com essas considerações, a trajetória do projétil pode ser considerada linear, o que torna viável o uso de um emissor de laser para a calibração do sistema composto pela arma, o suporte e a câmera.

As hipóteses formuladas acima, apesar de razoáveis, carecem de um estudo experimental que as sustentem. Porém, devido a fatores como prazo e equipamentos, esse estudo não pôde ser aprofundado neste trabalho.

3.2.1.2 *Imobilidade do usuário apenas durante o treino:*

Nas provas de tiro esportivo, tanto olímpico quanto paraolímpico, o atleta não se desloca entre os disparos, permanecendo no mesmo local durante toda a prova. Apesar disso, não se pode garantir que o usuário do software desenvolvido neste trabalho vá sempre utilizá-lo na mesma posição e com o a câmera acoplada à arma sempre da mesma maneira. Com isso em mente, o software há de solicitar nova calibração da arma antes de todo treino, mas essa calibração valerá durante toda a duração do mesmo.

3.2.1.3 *Equidistância de todos os pontos da tela à câmera:*

Para explicar melhor a importância dessa hipótese, é necessário entender primeiramente como funcionará a calibração. Para calibrar o sistema composto por câmera, suporte, arma e tela contendo a interface é necessário projetar o laser na superfície da própria tela que exibe a interface. Caso a calibração ocorra com o laser projetado em outra superfície, a mesma não será eficaz, como pode ser visualizado na Figura 29.

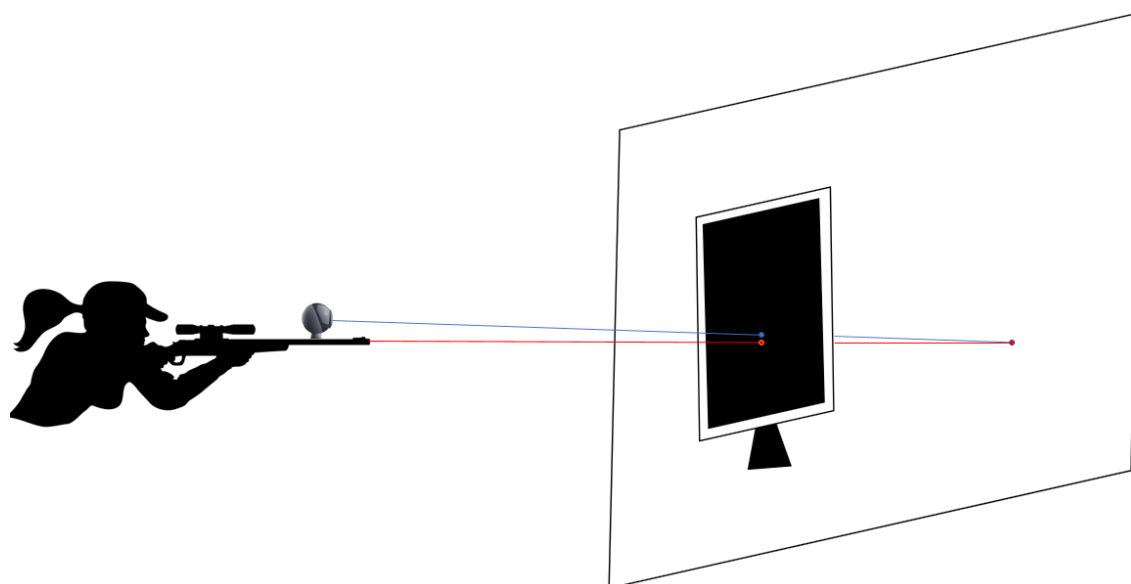


Figura 29: Ilustração da necessidade de se realizar a calibração usando da superfície da tela que exibe o software.

O erro exibido na imagem acima ocorre devido à distância entre a webcam e a tela ser diferente da distância entre a webcam e a superfície utilizada para a calibração. Teoricamente, a calibração só é válida para superfícies que tenham a mesma distância da câmera que o ponto no qual ela foi feita. Como o usuário, assim como a câmera, permanecem imóveis durante o treino, a calibração só seria válida para pontos em uma superfície côncava de centro na câmera e que passasse pelo ponto usado para a calibração. Porém a grande maioria das televisões possuem telas planas, não satisfazendo esse critério.

Para que a calibração seja efetiva para toda a tela com a interface do jogo, é necessário considerar que as diferenças de distância entre a câmera e um ponto qualquer da tela e a câmera e o ponto usado para a calibração sejam desprezíveis. Em outras palavras, considera-se que todos os pontos da tela são equidistantes da câmera.

3.2.2 Caracterização do Dispositivo:

Definido e justificado o método de fixação a ser utilizado, cabe caracterizá-lo, tanto o suporte quanto a câmera.

3.2.2.1 Imobilidade da câmera em relação à arma:

Para que o software simule de maneira adequada a prática esportiva, o usuário não deve ser interrompido durante o treino para uma nova calibração da arma. Portanto, a câmera tem de

estar fixa à arma e imóvel em relação à mesma, para que uma vez que o sistema seja calibrado, essa calibração não se perca.

3.2.2.2 Posicionamento da câmera na arma:

O software precisa ser capaz de identificar a tela da interface nas imagens capturadas pela webcam e para isso essa tela tem de estar completamente visível nas imagens. Algo que pode obstruir a imagem é o próprio cano da arma, dependendo da posição da lente da câmera em relação ao mesmo, como pode ser visto na Figura 30.

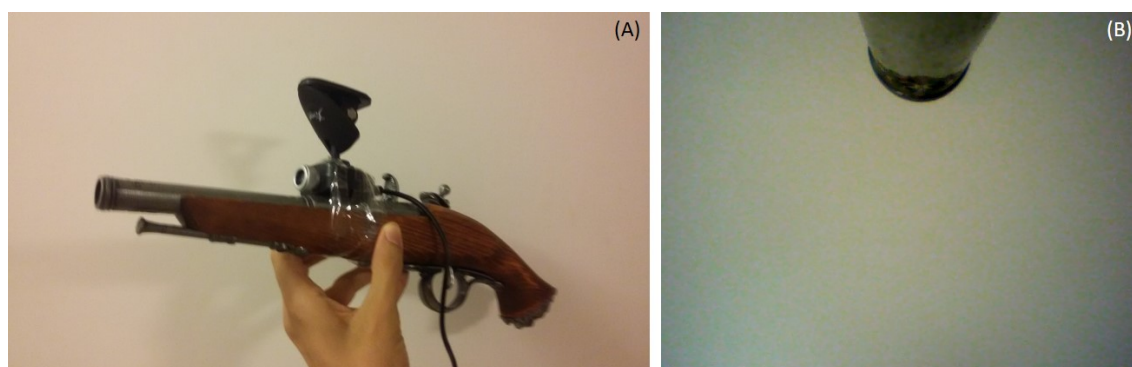


Figura 30: Interferência do cano da arma na imagem. (A) Acoplamento da câmera na arma; (B) Imagem capturada pela câmera com esse acoplamento.

Para essa imagem foi usada um modelo de arma antiga não funcional, mas que é capaz de demonstrar o efeito desejado. Em armas com cano mais longo, como as carabinas utilizadas em algumas modalidades do tiro esportivo, essa interferência é ainda maior. Sendo assim, o suporte tem de posicionar a lente da câmera próxima à saída do cano, ou levemente afastada do corpo da arma para que nenhuma de suas partes apareça na imagem, de forma a obstruí-la parcialmente.

3.2.2.3 Equipamentos sem fio:

Tanto a webcam a ser utilizada quanto o botão de pressão a ser fixado no gatilho não devem necessitar de fios, pois os mesmos representariam restrições ao movimento do usuário. Para proporcionar maior fidelidade entre as simulações do software e os treinos reais, os itens do hardware devem operar sua comunicação com o computador sem a utilização de fios.

4 Desenvolvimento do Software:

O desenvolvimento do software foi dividido em duas etapas, o melhoramento das funções já existentes, e a criação de novas funções, o que abrange modificação dos atuais

arquivos de código, criação de novos arquivos e criação de novas tarefas nos arquivos de código já existentes.

4.1 Melhoramentos e Correções no Software

4.1.1 Alteração da interface regras:

A *interface regras* original continha as instruções para utilização do software, as regras do ISSF e do IPC, como pode ser visto na Figura 7. Essa interface foi alterada para que contenha apenas as instruções de operação do software, com o objetivo de tornar mais intuitivo ao usuário a localização de tais instruções. O nome da interface também foi alterado de *interface regras* para *interface instruções* e o identificador utilizado nas linhas de código para chamar essa interface foi alterado de “Regras” para “Instrucoes”. O resultado das alterações pode ser visto na Figura 31.

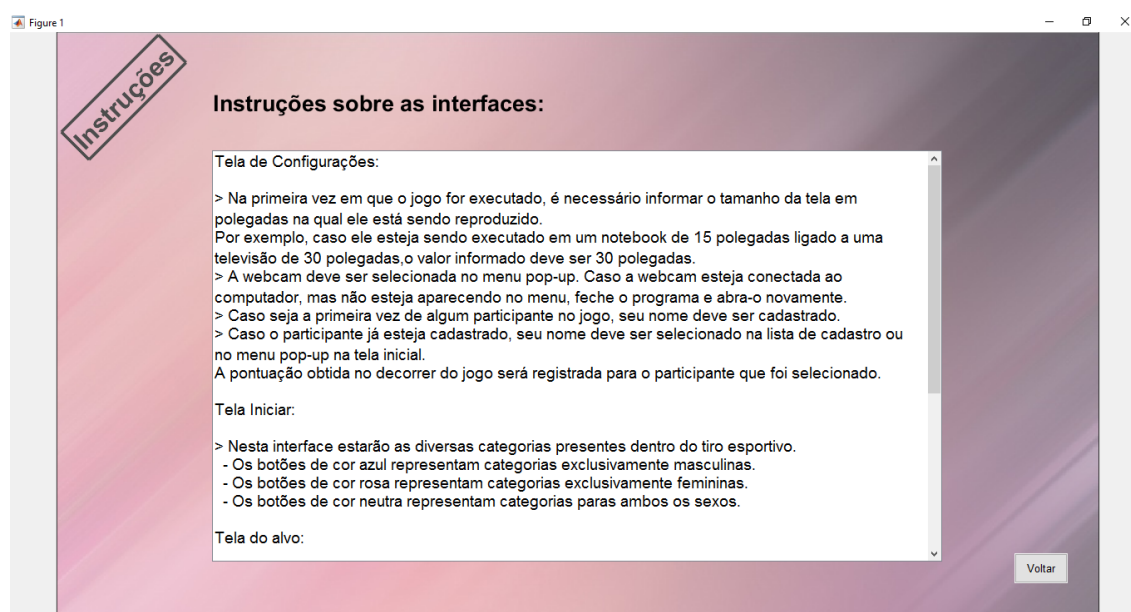


Figura 31: Interface instruções, que substitui a antiga interface regras.

Uma vez que a remoção dos botões que abriam as regras do ISSF e do IPC liberou espaço na parte inferior da tela da interface, o objeto do tipo *edit* que contém as regras foi estendido para maior conforto visual do usuário. O texto exibido, que é lido do arquivo Obs.txt (Fig. 4), também sofreu pequenas alterações para facilitar o seu entendimento. Por fim, o elemento de texto que fica sobre as instruções foi alterado de “Observações” para “Instruções sobre as Interfaces: ”.

As regras do ISSF e do IPC não foram removidas do programa, apenas realocadas para a *interface inicial*, foram acrescentados também dois elementos de texto informando o significado das siglas nos botões.. O botão que leva à *interface instruções* também teve o seu texto alterado de “Regras” para “Instruções”. A nova tela da *interface inicial* pode ser vista na Figura 32.

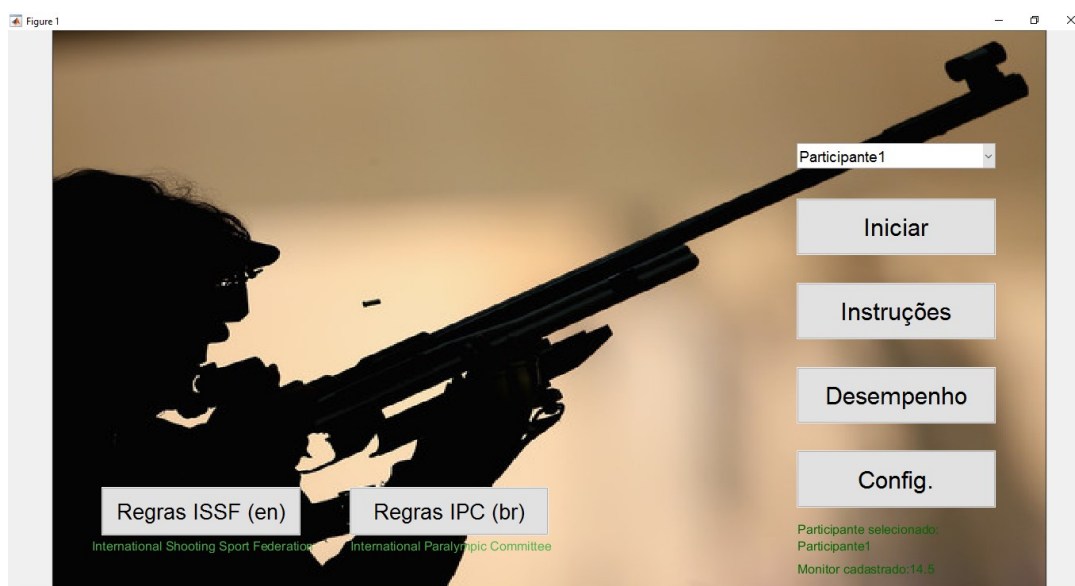


Figura 32: Interface inicial reestruturada.

4.1.2 Ajuste da distância do usuário à tela:

Na maioria das provas de tiro olímpico e paraolímpico a distância entre o atleta e o alvo é constante, o que implica que o tamanho do alvo na tela que está executando o programa também deve ser constante. No software original o tamanho do alvo é ajustado com o uso do objeto *slide*, que pode ser visto na *interface alvo* (Fig. 13). O software utiliza o valor do slide, o tamanho da tela e o tamanho real do alvo para determinar o tamanho que o mesmo deve ser mostrado na tela e calcular a distância que o jogador deve estar da mesma. A distância é mostrada em metros, com precisão de décimos de milímetros.

Toda vez que o usuário altera a posição do slide, e assim o seu valor, o alvo é reajustado para um novo tamanho e o treinamento reinicia. Qualquer ponto marcado antes desta operação é descartado e o contador regressivo é reiniciado. Esse sistema de determinação da distância entre o usuário e a tela, bem como do tamanho do alvo na interface é de difícil uso quando colocado em prática. O usuário tem de ajustar o slide, que passará a indicar a nova distância que ele tem que assumir em relação à tela e, rapidamente, assumir uma nova posição com essa distância. O tempo que ele tem para fazer isso é apenas aquele que o jogo leva para carregar a

nova interface e os novos parâmetros. Neste curto intervalo de tempo não é possível aferir a nova distância sem comprometer o tempo daquela seção de treinamento. Operando desta forma, o usuário teria que realizar o seguinte procedimento:

1. Determinar a posição na qual ele irá realizar o treino e demarca-la;
2. Aferir a distância entre essa posição e a tela;
3. Ajustar o slide até que a distância indicada na interface seja equivalente à aferida no item 2, lembrando que a distância é indicada com precisão de décimos de milímetros, o que dificulta bastante o posicionamento do slide de maneira exata;
4. Retornar rapidamente para a posição marcada no item 2;
5. Começar o treino;

Pensando neste procedimento foi decidido que, para o melhor funcionamento do programa, o slide será descartado e o usuário terá de informar a sua distância em relação à tela antes dos treinos. Devido à necessidade de calibração do conjunto “câmera + arma”, que deve ser feita na mesma posição onde o usuário realizará o treino, o parâmetro “distância” deverá ser inserido antes mesmo da calibração. Dessa forma, o novo procedimento, incluindo a calibração, será o seguinte:

1. Determinar a posição na qual irá se realizar o treino e demarca-la;
2. Aferir a distância entre essa posição e a tela e informar o software no campo apropriado;
3. Dar o comando de início da calibração e retornar à posição demarcada no item 1, lembrando que a calibração não tem um tempo limite para ser realizada, de forma que o usuário não precisa de pressa;
4. Inserir o colimador a laser na arma e realizar um disparo na interface;
5. Com o sistema já calibrado, remover o colimador;
6. ar início ao treino;

Com essa mudança implementada, a *interface alvo* será um pouco diferente daquela apresentada na Figura 13. A nova *interface alvo* pode ser vista na Figura 33, onde foi selecionada a modalidade P1 pistola de ar 10m masculino.

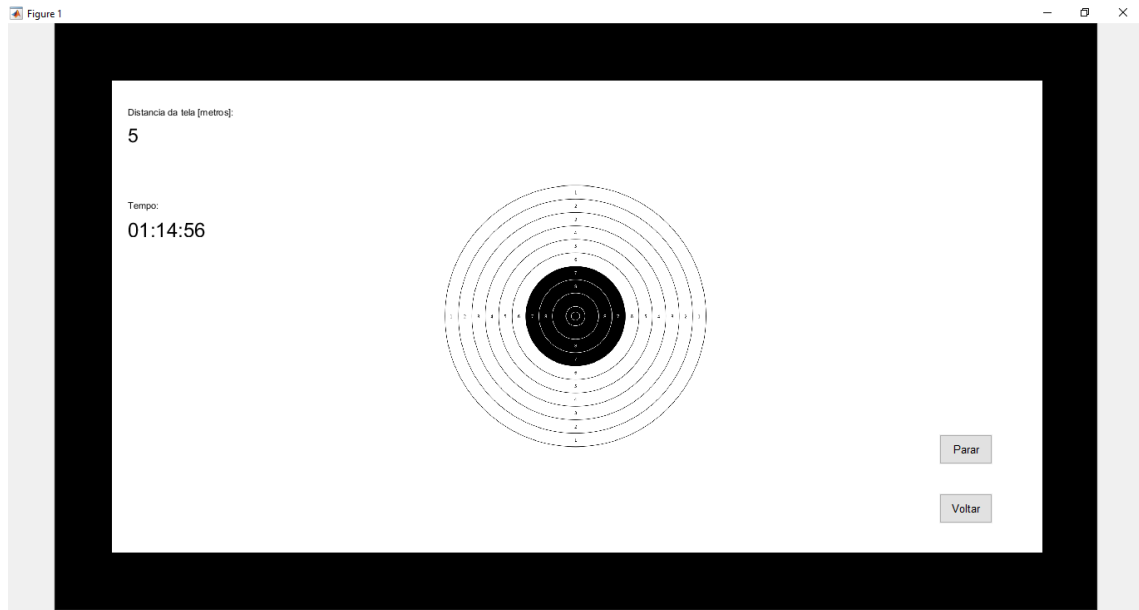


Figura 33: Nova interface alvo, sem o slider para ajuste do tamanho no lado direito.

4.1.3 Cálculo do tamanho do alvo na tela:

Com a retirada do slider, o cálculo da distância entre o usuário e a tela deixou de ser necessário, porém o ajuste do alvo teve de ser refeito, de forma que ele seja criado com o tamanho devido na tela. O slider estava associado a uma variável denominada *valor*, que variava entre 0,8, para o slider na posição máxima (superior) e 0,1 para o slider na posição mínima (inferior). Essa variável era usada para determinar qual o tamanho que o alvo deveria assumir na tela e, em seguida, para calcular a distância que o usuário deveria ficar da mesma. Para a primeira dessas tarefas, a imagem do alvo era redimensionada usando o comando *imresize*, que altera o tamanho de uma imagem com base em um parâmetro. A imagem modificada pelo comando é aquela do próprio alvo, enquanto o parâmetro, denominado *escala*, é calculado com a seguinte equação:

$$escala = \frac{H_{borda} * valor}{H_{alvo}}$$

O numerador da equação define o tamanho que o alvo deve assumir na tela, que é dado pela dimensão vertical da imagem de fundo “borda.png”, multiplicada pela variável *valor*. O tamanho que o alvo deve assumir na tela tem de ser determinado em relação à imagem de fundo, já que ele será criado sobre ela, o que explica a multiplicação pela altura da imagem “borda.png”. Esse valor tem de ser dividido pelo próprio tamanho do alvo para que, quando o alvo for redimensionado por esse parâmetro, o tamanho da imagem do alvo não influencie o tamanho final que ele assumirá na tela, por isso a necessidade do denominador na equação.

O software original ainda precisava calcular a distância que o usuário deve tomar da tela, para isso o tamanho do alvo em pixels é determinado com o uso da variável *valor* e do tamanho vertical da figura de fundo. Em seguida, o software afere as dimensões da tela em pixels, com o uso do comando *get(0, 'MonitorPositions')*, e as utiliza para calcular o tamanho da diagonal da tela, também em pixels. Esse valor é comparado com o tamanho da diagonal da tela, informado pelo usuário na *interface configurações* em polegadas e convertido em milímetros, e assim determina-se quantos pixels existem por milímetro linear da tela. A dimensão do alvo é então convertida de pixels para milímetros e o valor é utilizado em uma semelhança de triângulos para determinar a distância a ser assumida pelo usuário da tela. Essa semelhança está ilustrada na Figura 34:

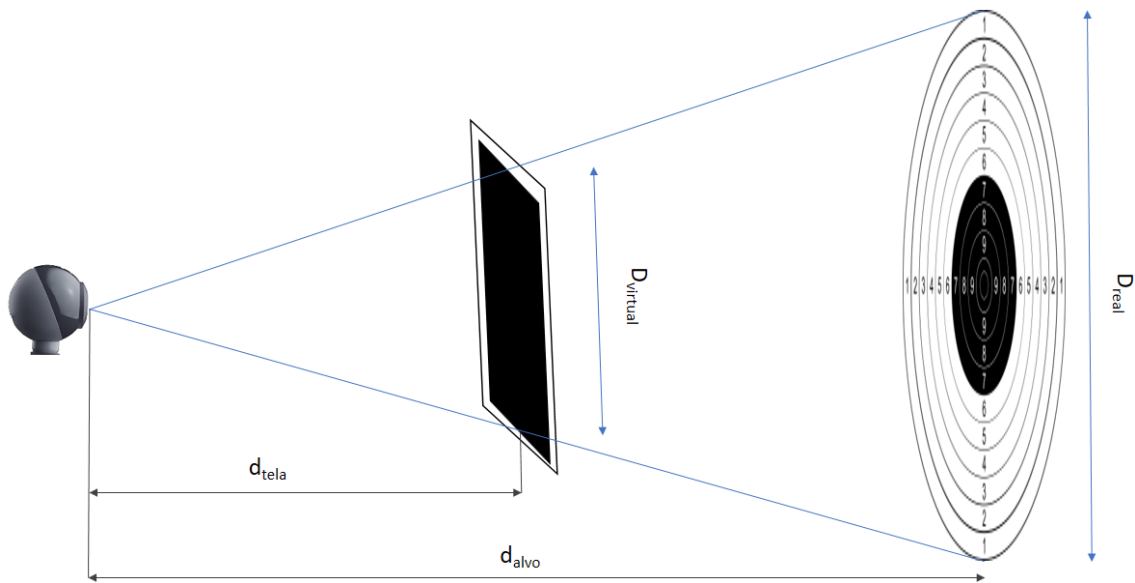


Figura 34: Semelhança de triângulos usada para determinação da distância à tela.

Nessa imagem, as letras “d”, em minúsculo, significam distância, enquanto as letras “D”, em maiúsculo, significam diâmetro. $D_{virtual}$ refere-se ao diâmetro do alvo na tela em milímetros. Com isso, a equação para o cálculo da distância à tela será:

$$d_{tela} = \frac{d_{alvo} * D_{virtual}}{D_{real}}$$

Com a remoção do slider, a variável *valor* deixa de existir e a distância do alvo à tela não precisa mais ser calculada, já que ela é informada pelo próprio usuário. Porém, a variável *escala*, usada como parâmetro para o comando *imresize*, tem de ser calculada de outra maneira, devido à ausência da variável *valor*. De acordo com o funcionamento do comando *imresize*, o parâmetro usado para redimensionar o alvo tem de ser igual ao tamanho desejado do mesmo na

tela dividido pela altura do próprio alvo, em pixels. Para determinar este tamanho, o diâmetro do alvo em pixels tem de ser calculado.

Sendo a distância do atleta à tela informada pelo próprio usuário, a semelhança de triângulo ilustrada na Figura 34 é usada de base para calcular o diâmetro virtual, ou seja, o diâmetro que o alvo deve ter na tela, em milímetros. A equação resultante da semelhança de triângulos é a seguinte:

$$D_{virtual} = \frac{D_{real} * d_{tela}}{d_{alvo}}$$

O resultado desta equação é então convertido em pixels, de maneira similar à como o trabalho anterior convertia pixels em milímetros. As dimensões da tela em pixels são obtidas com o uso do comando `get(0, 'MonitorPositions')` e utilizadas para calcular o tamanho da diagonal da tela, também em pixels. Esse valor é comparado com o tamanho da diagonal da tela, informado pelo usuário na *interface configurações* em polegadas e convertido em milímetros, e assim determina-se quantos pixels existem por milímetro linear da tela. Multiplicando-se o $D_{virtual}$ pelo resultado, obtém-se o diâmetro que o alvo deve ter na tela em pixels, D_{pixel} .

A variável D_{pixel} indica quantos pixels o alvo deverá ocupar na janela, porém o uso do comando `imresize` altera o tamanho do alvo em relação à imagem de fundo já plotada e não em relação à janela. Portanto, precisa-se da dimensão que o alvo ocuparia na imagem “bordas.png” em pixels. Para isso os valores da altura da janela e da imagem “bordas.png” foram colocados em um gráfico, que relaciona o tamanho do alvo em cada uma delas. Esse gráfico pode ser visto na Figura 35. H_{bordas} representa a altura da imagem bordas, y_{tam} representa o tamanho da janela, D_{pixel} representa o tamanho que o alvo deve ocupar na janela e D_{final} representa o tamanho do alvo em relação à imagem “bordas.png”. Desse gráfico retira-se a relação matemática para o cálculo de D_{final} , como mostrado a seguir.

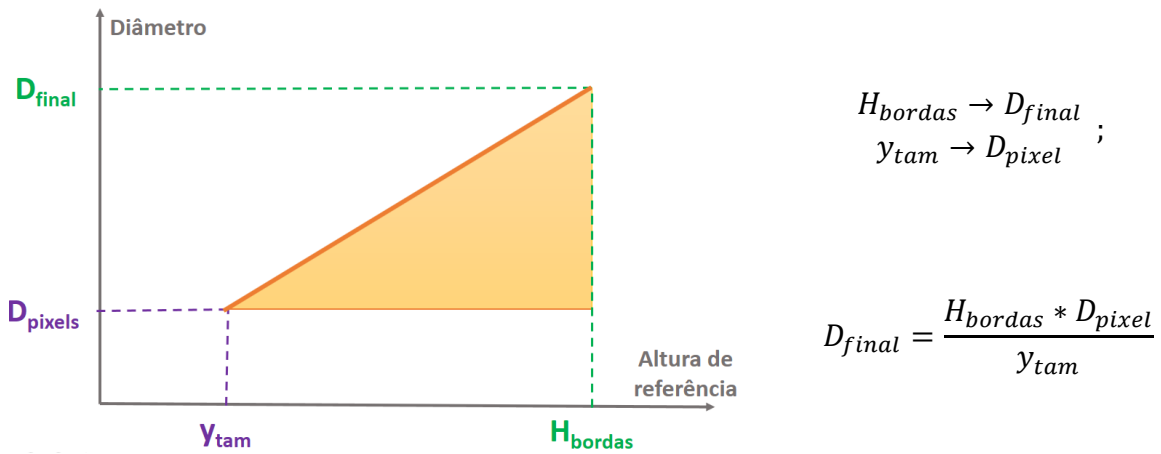


Figura 35: Relação entre o diâmetro na tela e o diâmetro na imagem "bordas".

Por último, o parâmetro *escala* deve ser igual ao tamanho que o alvo deve ter, D_{final} , dividido pelo próprio tamanho do alvo, D_{alvo} :

$$escala = \frac{D_{final}}{D_{alvo}} = \frac{H_{bordas} * D_{pixel}}{y_{tam} * D_{alvo}}$$

O comando final usado para redimensionar o alvo será $alvo = imresize(alvo, escala)$.

4.2 Elaboração da Função de Calibração

O método de calibração do sistema foi idealizado com o uso de duas novas interfaces, uma para instruir o usuário do funcionamento e importância da calibração e a outra para efetivamente realizar a calibração. À primeira delas foi dado o nome de *calibr.1* e à segunda de *calibr.2*. Para a criação de ambas foi criada uma nova função com o nome de “*calibracao.m*”, que foi integrada ao software com o uso de novas linhas de comandos inseridas nas já existentes funções “*InterfaceResize.m*” e “*callback_botao.m*”.

4.2.1 Interface *calibr.1*:

Essa interface tem três objetivos, instruir o usuário para que ele entenda a importância da calibração, como realizar a mesma corretamente e permitir que seja informada a distância que o atleta estará da tela durante a seção de treino. Com isso em mente, esta interface é composta por um quadro contendo as instruções pertinente e um espaço para a inserção da distância. Esta interface pode ser vista na Figura 36:

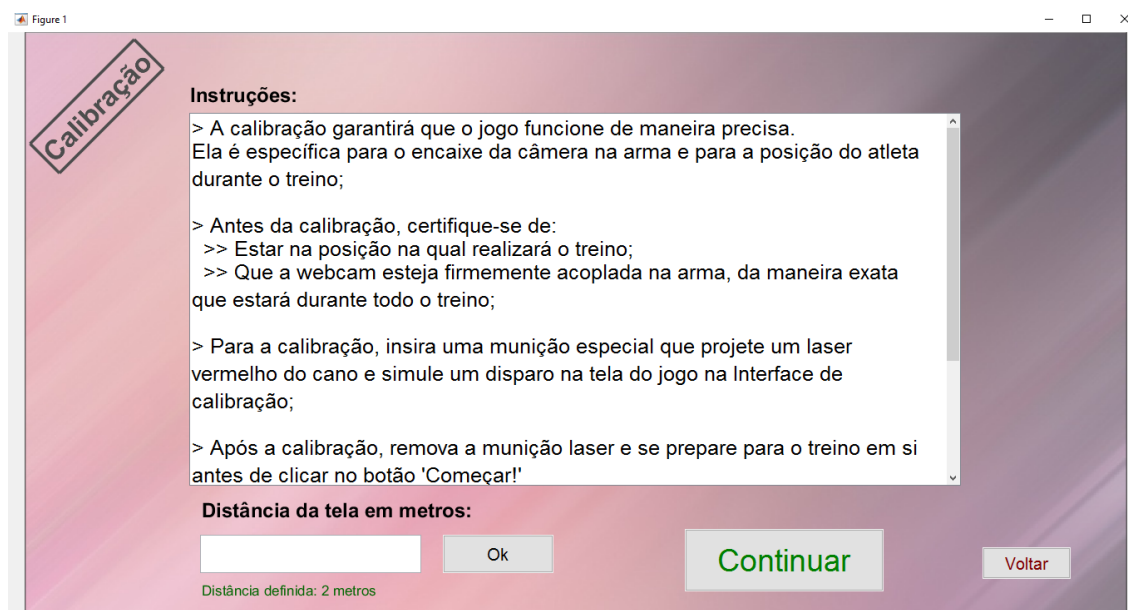


Figura 36: Interface *calibr.1*, que contém as instruções para a calibração e um espaço para inserção da distância.

Para o fundo da interface foi utilizada a mesma imagem utilizada no fundo das interfaces *instruções*, *configurações*, *desempenho* e *modalidades*. Foram usados um total de quatro elementos de texto nesta interface, todos criados com o uso do comando *text*, inclusive o título da interface, no canto superior esquerdo. Um destaque para o objeto de texto que indica a distância do usuário à tela, que é alterado automaticamente toda vez que uma nova distância é inserida e confirmada, com o uso do botão “Ok”.

As duas caixas brancas foram criadas usando o comando *uicontrol*, sendo que elas são objetos do tipo *edit*. As instruções apresentadas na caixa maior estão contidas em um documento de texto de nome “Obs2.txt”. O botão “Voltar” abre a *interface modalidades*, que é a única a partir da qual a *interface calibr.1* pode ser aberta. O botão “Continuar” abre a *interface calibr.2*, dando início de fato à calibração.

A distância a ser inserida na segunda caixa é salva em um objeto de texto ao se clicar no botão “Ok”, procedimento idêntico ao usado na *interface configurações* para armazenar o tamanho da tela em polegadas. O nome do objeto de texto criado para essa finalidade é “Distancia.txt”. Esse procedimento permite que o valor inserido de distância em um treino seja memorizado para os treinos posteriores.

4.2.2 Interface *calibr.2*:

Após informar a distância na *interface calibr.1* e clicar no botão “Continuar”, a *interface calibr.2* é carregada. É nesta interface que a calibração será realizada de fato. Para que ela seja feita, o usuário tem de mirar o laser na região da tela e efetuar um disparo, que fará com que a webcam capture uma imagem. O software então buscará, na imagem capturada, um padrão de pixel que corresponde ao laser, salvando em seguida, as coordenadas de todos os pixels identificados como atendendo a esse padrão. O resultado de onde o laser estava na imagem será uma média dessas coordenadas. Porém, o uso deste procedimento de calibração gera um problema: podem haver, na imagem capturada, pixels que se assemelhem àqueles do laser, mas que não pertencem a ele, e que teriam suas coordenadas salvas e usadas na determinação das coordenadas do laser, gerando erro na calibração.

Para solucionar esses problemas, o software só buscará pelo laser no conjunto de pixels que correspondem à região da tela. Além de evitar o erro, isso irá reduzir o esforço computacional requerido para a busca, já que ela será realizada em uma área menor. Mas, mesmo com o uso dessa técnica ainda podem haver, na região da tela, pixels que se confundam com aqueles do laser, gerando o mesmo erro. Por conta desta possibilidade, o software vai fazer a correspondência entre a tela na imagem capturada pela webcam e a tela que contém a interface do jogo, mostrando nesta o local onde o laser foi identificado. Dessa forma, o usuário poderá verificar visualmente se o local onde o laser foi identificado pelo software coincide com o local onde ele estava de fato projetado.

Como o software precisa identificar a tela nas imagens capturadas pela webcam, a *interface calibr.2* usa como imagem de fundo a imagem “borda.png”, contida na pasta “Imagens” (Figura 4), a mesma usada na *interface alvo*. A interface pode ser vista na Figura 37.

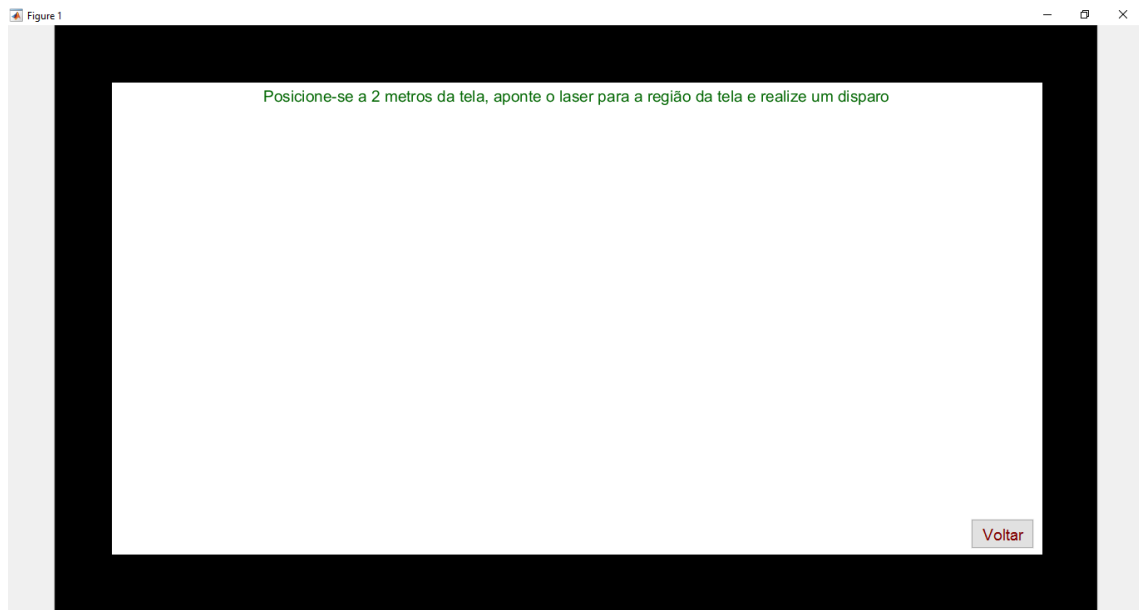


Figura 37: Interface Calibr.2, onde a calibração é realizada de fato.

O único objeto de controle presente na interface é o botão “Voltar”, gerado com o uso do comando *uicontrol*, sendo do tipo *pushbutton*. Pressionar esse botão abre novamente a *interface calibr.1*, que é a única a partir da qual a *interface calibr.2* pode ser acessada. O elemento de texto presente na interface apresenta ao usuário a distância a qual ele deve ficar da tela para realizar a calibração, distância essa informada pelo próprio usuário na interface anterior.

Para a *interface alvo*, a função “*telaalvo.m*” elabora a interface, enquanto a função “*gatilho.m*”, é responsável por realizar os comandos pertinentes quando o gatilho é pressionado (clique do botão esquerdo do mouse em qualquer parte da interface que não seja um elemento de controle). Na *interface calibr.2*, a função “*calibracao.m*” realiza apenas um dos papéis citados acima, o equivalente àquele da função “*gatilho.m*”. Para ser capaz de fazer isso e ainda criar a *interface calibr.1*, como citado no item 5.2.1, essa função usa o comando *if elseif* para analisar a variável global *tag* (não ser confundida com a variável *Tag*, explicada no item 3.1.4.1 deste trabalho). Caso essa variável seja igual à string “Calibr.”, são executados os comandos para a criação da *interface calibr.1*, caso ela seja igual à string “Calibr.2”, são executados os comandos para captura e processamento de dados necessários quando o usuário puxa o gatilho na *interface calibr.2*.

A elaboração da *interface calibr.2*, por sua vez, é feita na função “*callback_botao.m*”. Para isso, foi acrescentado um comando *case* à estrutura de comandos *switch case* que compõe

essa função, descrita no item 3.1.4.1 deste trabalho. Caso a variável *Tag*, analisada pela estrutura de comandos *switch case*, seja equivalente à string “Calibr.2”, os comandos necessários à elaboração da *interface calibr.2* serão executados. A variável *Tag* é definida como “Calibr.2” quando o usuário pressiona o botão “Continuar”, na *interface calibr.1* (Fig. 36), botão esse que chama a função “callback_botao.m”.

Ressalta-se que a função de calibração foi elaborada de uma maneira que, mesmo se ela for bem-sucedida e o software for capaz de identificar o laser na tela, o usuário poderá realizar a calibração novamente de forma imediata, caso não fique satisfeito com o resultado obtido. Para refazer a calibração bastará um simples clique do botão de pressão acoplado ao gatilho, que corresponde a um clique com o botão esquerdo do mouse. Portanto, caso o software não identifique o laser em uma primeira tentativa, novas tentativas podem ser realizadas imediatamente pelo usuário. Com isso em mente, os critérios para identificação do laser podem ser elaborados de maneira mais detalhada, para evitar qualquer erro gerado por elementos estranhos na tela. Assim, a função de calibração pode até ter dificuldades na identificação do laser, mas isso não será um problema devido à facilidade com que a sua execução pode ser repetida.

4.2.3 Identificando o laser nas imagens capturadas:

4.2.3.1 Procedimentos de tratamento das imagens nas interfaces alvo e calibr.2:

Para dar prosseguimento, é interessante detalhar o procedimento de tratamento das imagens capturadas usado para que o software identifique a tela. O procedimento a ser detalhado é aquele realizado pela função “gatilho.m”, acionada na *interface alvo* quando um clique é realizado na tela. Este procedimento é baseado na mesma função do trabalho de MORAES (2015), mas com algumas pequenas mudanças. A elaboração da função “calibracao.m” requer algumas alterações extras, que serão detalhadas e explicadas nas seções seguintes. Abaixo segue um passo a passo do procedimento de tratamento de imagens, com as alterações sofridas em relação ao original, usado na função “gatilho.m”. O ponto de partida é a *interface alvo*, já elaborada e plotada:

1. Captura da *imagem1* pela câmera com uso da função *getsnapshot*.
2. Execução do som de um disparo, criado com o comando *wavread*, que lê o arquivo “shot.wav” (Fig. 4).
3. Projeção de uma figura escura na tela e ocultação dos elementos de controle.
4. Captura da *imagem2*, com uso da mesma função *getsnapshot*.

5. Deleta-se a imagem escura e retorna-se a visibilidade dos elementos de controle, resultando na interface alvo sendo exibida novamente na tela.
6. Subtração de ambas as imagens capturadas, resultando na imagem3.
7. Binarização da imagem3 com o uso da função *rgb2gray*, seguida da função *im2bw*, com um fator de 0,1. Esta última função converte todos os pixels com luminosidade maior que esse fator de 0,1 em 1 (branco) e os demais pixels em 0 (preto). O resultando é uma imagem preta com grupos de elementos brancos, dos quais um deles é a tela.
8. Eliminação na imagem3 de objetos menores que um determinado parâmetro com uso da função *bwareaopen*. No trabalho original esse parâmetro era igual a 0,02% da quantidade de pixels existentes na imagem capturada pela câmera, ele foi aumentado para 0,2% desta quantidade.
9. Binarização da *Imagem1* e eliminação de elementos pequenos nela, de maneira idêntica à dos passos 7 e 8.
10. Comparação das imagens 1 e 3, binarizadas e tratadas, com uso do comando *and*. O resultado é a variável *imagem_inter*, na qual um pixel será branco se o mesmo o fosse nas imagens 1 e 3. Caso ambos fossem pretos ou apenas um deles brancos, o pixel resultante na *imagem_inter* será preto.
11. Eliminação na *Imagem_inter* dos objetos maiores que um parâmetro, de maneira idêntica à do passo 8.
12. No trabalho original, o resultado era salvo em uma variável no Workspace global, com o uso da função *assignin('base', 'imagem_inter', imagem_inter)*. Como não foi encontrado uso nenhum desta variável em nenhuma função além da “gatilho.m”, essa etapa foi removida.
13. Salva-se as coordenadas dos pixels brancos da variável *img_inter* em uma variável do tipo estrutura, chamada *reg_inter*, com uso do comando *regionprops(imagem_inter, 'PixelList')*. Cada elemento dessa estrutura é uma matriz que contém as coordenadas dos pixels de uma das regiões brancas de *imagem_inter*. A dimensão dessas matrizes é de Nx2, onde N é a quantidade de elementos da região branca à qual aquela matriz corresponde. A primeira coluna contém as coordenadas em X de cada pixel, e a segunda coluna as coordenadas em Y.
14. Calcula-se quantidade de elementos de cada matriz da estrutura *reg_inter*, com uso do comando *numel*.
15. Encontra-se qual matriz tem mais elementos, e supõe-se que ela corresponde à região da tela. Essa matriz, que estava contida na variável *reg_inter*, do tipo estrutura, é agora salva em uma variável denominada *pixels11*, para facilitar seu processamento.
16. No trabalho original, a matriz com mais elementos era salva em uma variável no Workspace global, de maneira similar à do passo 12. Novamente essa etapa foi removida, pelo mesmo motivo.
17. A região cujas coordenadas estão salvas na matriz *pixels11* não será um quadrilátero perfeito, que é necessário aos procedimentos de cálculo seguintes. Para definir tal quadrilátero, identificar os seus vértices é suficiente. Por tanto, nesse passo é calculado a distância de todos os pontos da matriz *pixels11* ao canto superior direito da *imagem1*. O ponto com a menor distância é considerado um dos vértices do quadrilátero.
18. O passo 17 é repetido para os três demais cantos da *imagem1*, resultando nos demais vértices da tela presente nas imagens capturadas pela câmera.
19. O resultado dessas coordenadas é salvo em vetores, um para as coordenadas em X e o outro para as coordenadas em Y, que são usados nos demais procedimentos de cálculo.

Após identificar a tela, a função responsável pela calibração tem de localizar o ponto vermelho projetado pelo colimador laser na própria tela. Para isso, é preciso que ela saiba como distinguir os pixels que correspondem ao laser de todos os demais, sendo ideal que o laser esteja o mais destacado possível. Por conta disso, a busca não será realizada na *imagem1*, pois a claridade da tela ofusca o ponto laser, de modo que uma câmera com menor resolução não identifica nenhum sinal de sua projeção, como mostra a Figura 38. A imagem na qual a busca pelo laser será realizada será a *imagem2*, na qual a tela é escurecida e o laser se destaca mais, como pode ser visto na mesma figura:

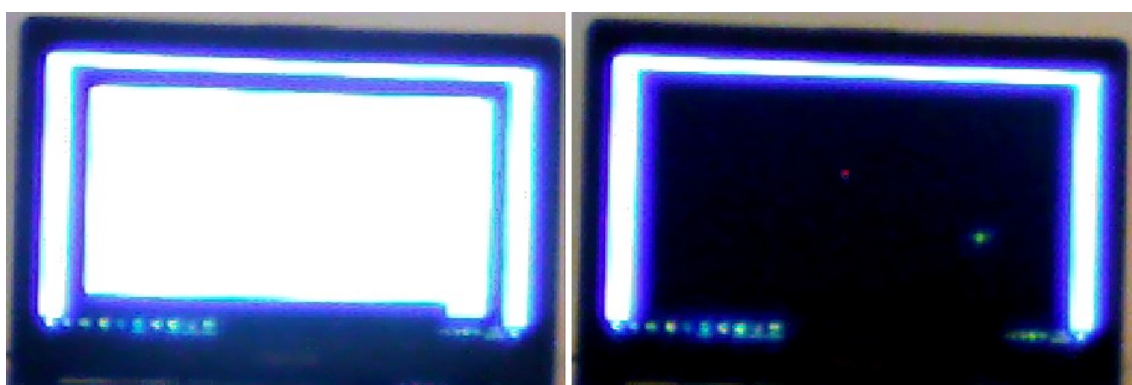


Figura 38: Diferença do ponto laser projetado na *imagem1* (esquerda) e na *imagem2* (direita).

A Figura 38 acima ilustra as variáveis *imagem1* e *imagem2* obtidas a partir de uma simulação da função de calibração. A região da tela de ambas as imagens foi destacada para melhor visualização. Para a captura dessas imagens, e de todas as demais capturas apresentadas neste trabalho, foi utilizada uma webcam da marca Knup, modelo GZE-173, que captura imagens com resolução de 640x480 pixels.

4.2.3.2 Problema do cursor na tela:

Na *imagem2* da Figura 38 observa-se, além do ponto vermelho correspondente ao laser, uma marca branca na região mais à direita. Essa marca corresponde ao ponteiro do mouse, que apesar de não apresentar coloração vermelha, é interessante ser ocultado na *imagem2* por dois motivos:

1. O mouse poderia se sobrepor ao ponteiro laser, alterando a sua coloração nas imagens capturadas e dificultando a sua identificação;
2. Em alguns dos ensaios, o ponto laser é caracterizado por alguns pixels brancos ou de cor muito clara. Se o cursor do mouse aparecer nas imagens, pixels esbranquiçados não poderão ser considerados como pertencentes ao laser, pois

os pixels do cursor poderiam eventualmente ser confundidos com os pertencentes ao laser, gerando erro na calibração.

A ocultação do mouse é realizada simultaneamente com o escurecimento da tela e ocultação dos demais objetos, no passo 3 descrito acima. Para ocultar o mouse foi utilizado o comando `set(gcf, 'Pointer', 'custom', 'PointerShapeCData', NaN(16,16))`. A propriedade *Pointer*, definida como *custom* neste caso, transforma o cursor do mouse em uma figura de 16x16 pixels, a ser definida. A propriedade *PointerShapeCData* cria essa figura, e deve ler uma matriz de dimensão 16 por 16, onde um elemento com valor 1 corresponderá a um pixel preto, um elemento com valor 2 corresponderá a um pixel branco e um elemento com valor *NaN* (sigla para Not a Number, do inglês, Não é um Número) corresponderá a um pixel transparente. Portanto, o componente *Nan(16,16)*, que serve de objeto para a propriedade *PointerShapeCData*, cria uma matriz 16x16, onde todos os elementos são *NaN*. Com isso, o cursor poderá estar posicionado em qualquer local no momento do disparo, que ele ficará invisível.

A visibilidade do cursor é restituída com o comando `set(gcf, 'Pointer', 'arrow')`, que altera a imagem do cursor para aquela da seta tradicional, retornando ele assim ao seu padrão. Isso é feito no passo 5, onde a imagem escura na tela é deletada e os elementos de controle voltam a ser visíveis.

4.2.3.3 *Problema do reflexo do emissor laser:*

Ainda existe um outro elemento que pode aparecer na tela eventualmente, o reflexo do emissor do laser, como pode ser visto na Figura 39:

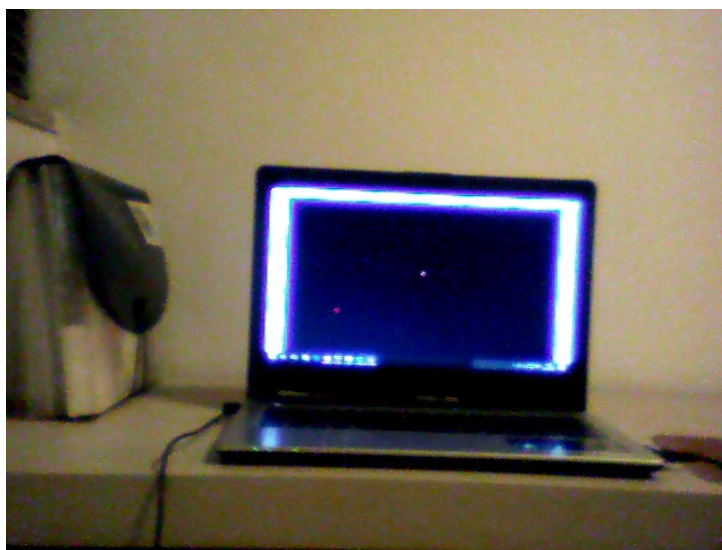


Figura 39: Exemplificação do reflexo do ponteiro laser na tela.

O reflexo do emissor laser na tela pode ser muito semelhante ao próprio ponto laser. Para enxergar isso melhor, quatro ensaios nos quais ocorreram o efeito do reflexo foram selecionados. Para melhor analisar os efeitos deste evento, as imagens que exibem o ponto laser foram cortadas e ampliadas, de modo a destacar os pixels que representam a projeção do laser e o reflexo do seu emissor. Essas imagens ampliadas foram agrupadas e podem ser vistas na Figura 40.

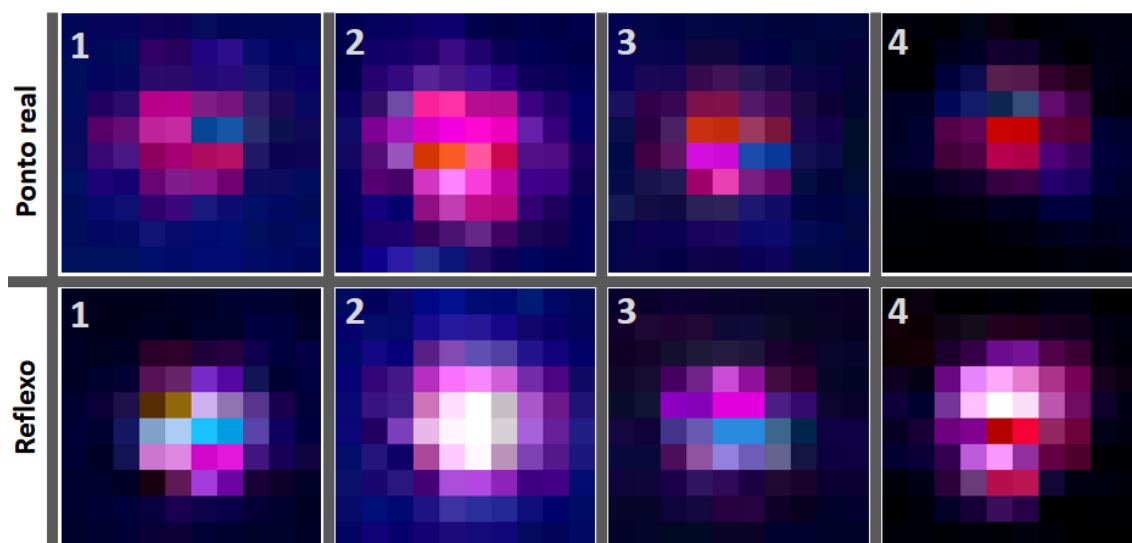


Figura 40: Detalhe ampliado do ponto laser e do reflexo do emissor laser na tela.

Por mais que de maneira geral não haja tanta semelhança, como no exemplo de número 1, a presença de pixels de cores próximas relacionados tanto à projeção do laser quanto ao reflexo pode confundir o programa, que considerará que alguns pixels deste reflexo pertencem ao ponto vermelho buscado. Isso fará com que as coordenadas desses pixels confundidos sejam usadas no cálculo das coordenadas do ponto laser, gerando um erro, que será maior ou menor dependendo da distância entre a projeção real e o reflexo e da quantidade de pixels considerados como sendo do laser. Se este erro for significativo, o usuário irá percebê-lo e poderá refazer a calibração. Se o erro for pequeno, ele pode não ser percebido pelo usuário, de forma que todo o treino seguinte será realizado com um erro sistemático, podendo inclusive afetar o desempenho do atleta nas competições ou nos treinos reais.

A única forma de evitar a aparição deste reflexo nas imagens é evitando a perpendicularidade entre a tela e o feixe laser. Se a tela estiver ligeiramente inclinada em relação ao usuário, o reflexo do laser projetado na tela não estará no campo de visão da câmera. A Figura 41 exemplifica esta situação.

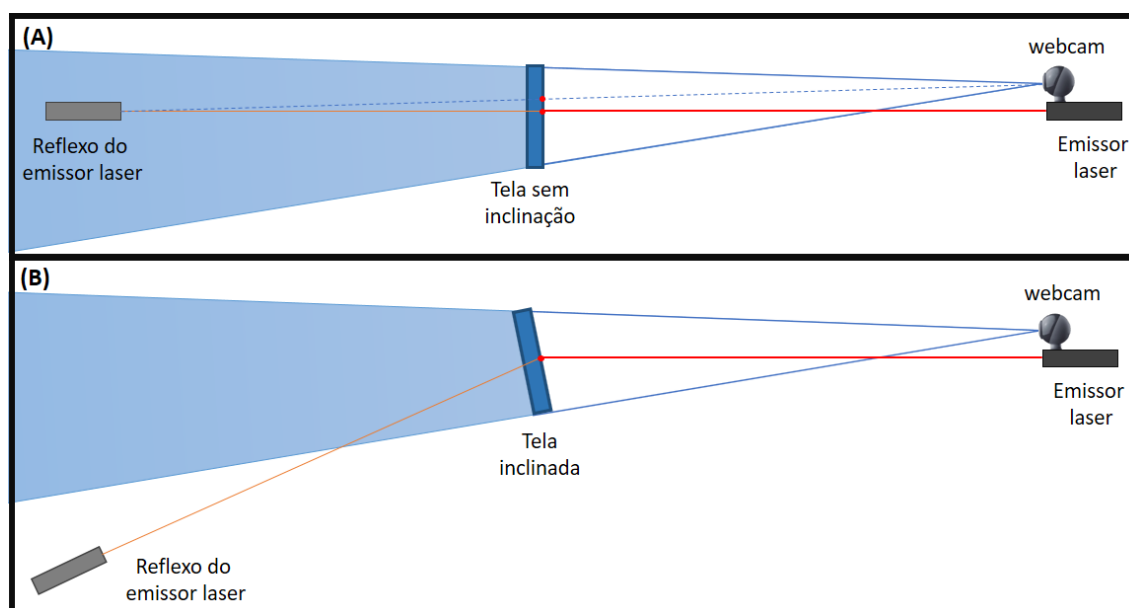


Figura 41: Exemplificação de como evitar o reflexo do emissor laser na tela com a inclinação da mesma.

Nas imagens, o usuário é representado pelo emissor laser acoplado à arma que ele segura. A região azul claro representa a área em que a webcam é capaz de enxergar os reflexos de objetos na tela, os objetos cujos reflexos estiverem dentro desta área aparecerão refletidos na tela nas imagens capturadas pela webcam, e bem destacados se eles forem emissores de luz.

A Figura 41A representa o resultado para um teste com a inclinação da tela próxima à perpendicularidade em relação ao feixe laser. Como o reflexo do emissor está dentro da região azul, aparecerão dois pontos vermelhos na tela, conforme está representado. O de baixo é o ponto laser projetado na tela, que a webcam precisa de fato enxergar para realizar a calibração. O de cima é o reflexo do emissor laser que aparece nas imagens capturadas, ele é indesejado e gera um erro na calibração. Além disso, nessa situação o feixe laser será refletido na direção do próprio usuário, podendo eventualmente ser projetado em um de seus olhos, o que pode representar um risco à saúde.

A Figura 41B representa o resultado obtido com a tela sensivelmente inclinada em relação ao usuário. Devido à inclinação da tela, o reflexo do emissor laser não estará no campo de visão da webcam e as imagens capturadas por ela enxergarão apenas um ponto, referente à projeção do laser na tela. Além disso, o feixe laser será refletido para uma direção mais distante do usuário, assim será muito menos provável que o laser seja direcionado aos seus olhos.

Supõe-se que a maneira mais intuitiva do usuário se posicionar durante um treino será o mais alinhado o possível com a tela, ou seja, a tela estaria praticamente perpendicular à sua

linha de visão. Como isso acarretará o mesmo problema descrito acima, faz-se necessário uma instrução clara ao usuário de como ele deve se posicionar em relação à tela. Essa instrução foi inserida no objeto de texto “Obs2.txt”, que é usado para exibir as instruções na *interface calibr.1* (Fig. 36).

4.2.3.4 Padrão dos pixels pertencentes ao laser:

Para que o programa seja capaz de identificar o ponto laser na tela, é necessário que ele saiba quais as características dos pixels pertencentes a este ponto. Além disso, é interessante saber o que caracteriza os demais pontos da tela, na qual o laser será buscado, para ter certeza que as características do laser não se repitam em nenhum outro pixel da tela, o que geraria um erro. Para isso, foram realizados alguns ensaios, dos quais as imagens com a tela escurecida foram selecionadas e, sobre essas imagens, foi plotado um quadrilátero que representa a área que será varrida na busca pelo laser. Para garantir uma melhor visualização da parte de interesse, a tela com a interface foi destacada do resto da imagem. Os resultados estão apresentados na Figura 42.



Figura 42: Testes usados para identificar as características do ponto laser na tela.

As imagens foram capturadas sempre utilizando o brilho máximo da tela, condição recomendada para o bom funcionamento do software, no trabalho de MORAES (2015).

Tanto a tonalidade da tela quanto a visibilidade do ponto laser variam de imagem para imagem. Destaca-se ainda que no teste T08 o efeito do reflexo do emissor laser ocorreu, indicando que a inclinação da tela não foi suficiente, a projeção do laser está mais embaixo, enquanto o outro elemento mais centrado na tela é o reflexo do emissor. Além disso, vê-se que em alguns dos testes a claridade na lateral da tela aparece parcialmente dentro do retângulo de análise, claridade essa decorrente do não preenchimento total da janela pela imagem de fundo. Na Figura 37 pode-se ver que a imagem de fundo não preenche completamente a janela e, por conta disso, há a presença de seções cinzas nas laterais da interface, que geram essa claridade na lateral dos testes apresentados.

As imagens com melhor visibilidade do ponto laser foram selecionadas para análise, sendo elas a T2, T4, T5, T6, T8, T9, T11 e T12. O fato de algumas imagens não apresentarem boa visibilidade não é um problema, pois é muito simples para o usuário repetir o processo de calibração caso o laser não seja identificado. No caso do teste T8, o efeito do reflexo também não seria um problema, pois o mesmo está distante do ponto real e o erro gerado pelo reflexo seria percebido pelo usuário conforme explicação apresentada na seção 5.2.3.2 deste trabalho.

Nos testes selecionados, os pixels pertencentes ao ponto de projeção do laser foram isolados em uma imagem de dimensões 10x10 pixels. As imagens resultantes foram lidas pelo Matlab, que as interpreta como imagens RGB. Esta interpretação significa caracterizar a imagem por meio de três matrizes, uma para a cor vermelha, uma para a cor verde e uma para a cor azul, todas com as mesmas dimensões de 10x10 elementos. Os pixels e as suas respectivas matrizes foram agrupados nas figuras 43 e 44.

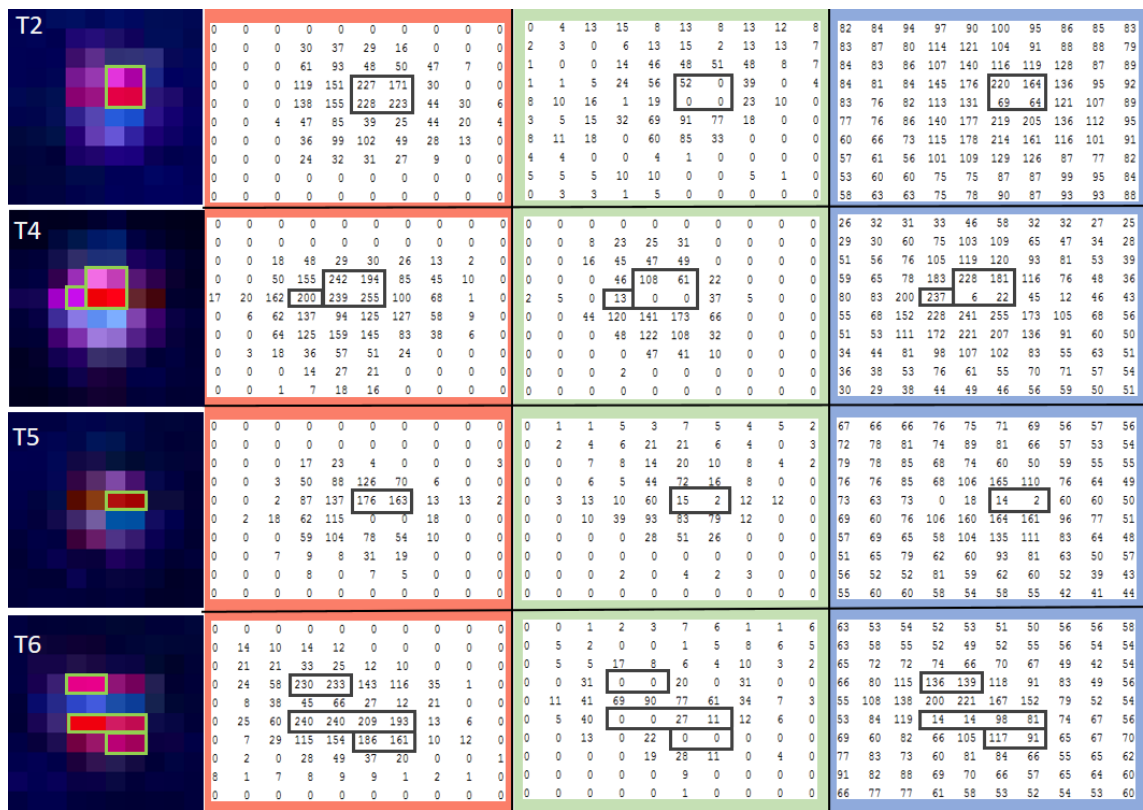


Figura 43: Pixels pertencentes ao ponto laser nos testes T2, T4, T5 e T6 utilizados na caracterização do mesmo.

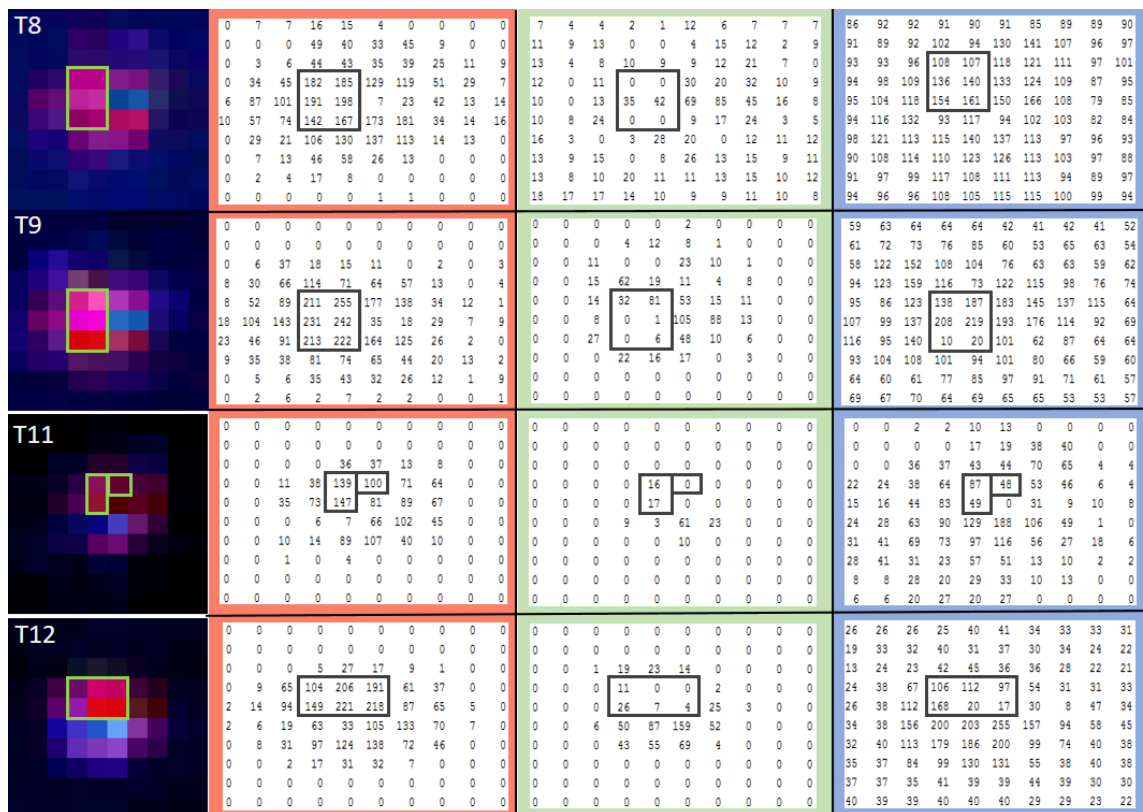


Figura 44: Pixels pertencentes ao ponto laser nos testes T8, T9, T11 e T12 utilizados na caracterização do mesmo.

A primeira coluna, em ambas as imagens, ilustra o ponto referente àquele ensaio, pixel por pixel, e o seu arredor imediato. As demais colunas, compostas de matrizes numéricas, correspondem aos valores do sistema RGB associados a cada um dos elementos do ponto e do seu arredor, representados na primeira coluna. A primeira matriz corresponde aos valores associados à coloração vermelha, a segunda à coloração verde e a terceira à coloração azul. Nessas matrizes, os valores representando cada uma das cores são inteiros e variam entre 0 e 255, sendo que quanto menor eles forem, menor será a intensidade daquela cor. Fica definido que a partir deste ponto, os valores associados a cada uma das cores serão referidos da seguinte forma:

- Os valores das matrizes da esquerda, correspondentes ao vermelho, como valores de R, do inglês Red;
- Os valores das matrizes do meio, correspondentes ao verde, como valores de G, do inglês Green;
- E os valores das matrizes da direita, correspondentes ao azul, com valores de B, do inglês Blue;

Como pode ser observado, a projeção do laser não é composta apenas por pixels vermelhos, na realidade a sua maioria possui uma tonalidade mais próxima de marrom, rosa ou roxo. Além disso, alguns dos pixels referentes à projeção do laser são azuis, possuindo valores de R menores que 40 e de B maiores que 200. Levando isso em consideração, foram identificados de maneira visual e destacados em cada uma das imagens, os pixels que o programa deveria ter identificado como pertencentes à projeção do laser. Os valores correspondentes a esses pixels também foram destacados nas três matrizes.

Os pixels azuis não podem ser incluídos na caracterização do laser, devido ao fato de poder haver outros pixels azuis em outras regiões da imagem identificada pelo programa como pertencente à tela. Um exemplo destes pixels azuis pode ser visto no teste T2, representado na Figura 45 com um detalhe ampliado.

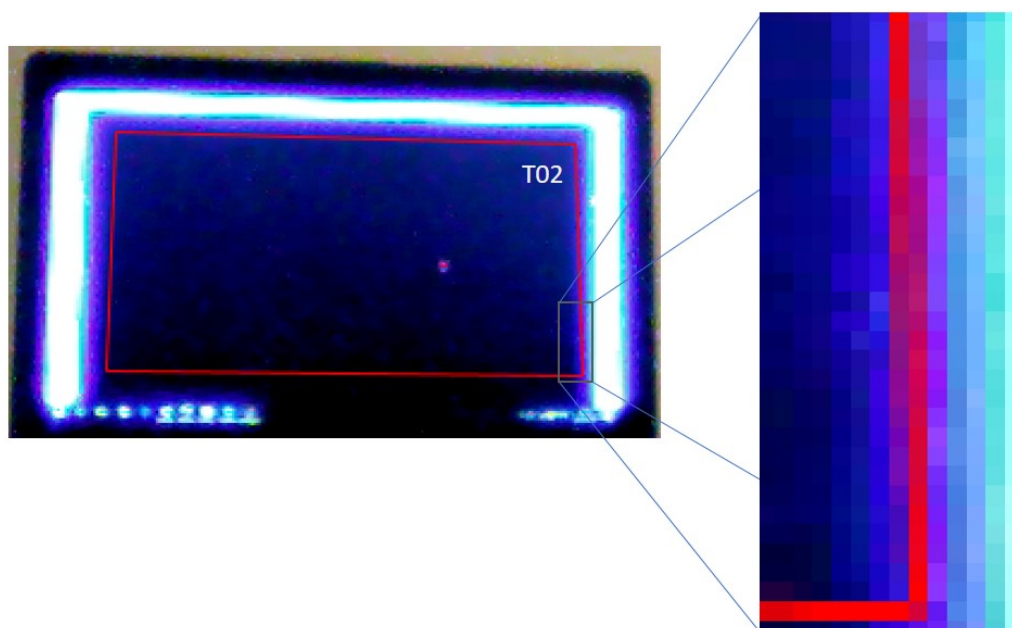


Figura 45: Exemplificação da presença de pixels azuis na região identificada pelo programa como pertencente à tela

Parte da região clara presente na tela nas imagens capturadas pela webcam será varrida na busca pelos pixels pertencentes à projeção do laser. Como essa região é caracterizada por pixels azuis de várias tonalidades, pixels azuis não podem ser identificados pelo programa como pertencentes à projeção do laser.

Retornando às figuras 43 e 44, têm-se 40 pixels destacados. Os valores que representam esses pixels foram isolados em uma planilha e três fatores foram usados para a sua análise, o máximo valor que representa cada cor, o mínimo, e a média de todos os valores. Esses fatores estão apresentados na Tabela 3, a seguir:

Tabela 3: Valores máximos, mínimos, e médios de R, G e B dos pixels destacados como sendo pertencentes ao ponto laser.

Vermelho (R)	Máx	255	Verde (G)	Máx	108	Azul (B)	Máx	237
	Min	100		Min	0		Min	2
	Média	198,10		Média	14,18		Média	104,73

Os valores de R são sempre altos, não ocorrendo nenhum caso de valores menores que 100 nos exemplos analisados. Portanto, foi definido que os valores de R que caracterizarão os pixels do ponto de projeção do laser terão de ser sempre maiores que 90.

Os valores de G em geral são bem baixos, como a sua média indica, mas apresenta alguns picos, chegando, nos exemplos estudados, até 108. Nos seus valores de pico, foi

observado também valores bem altos para R. Os quatro maiores valores de G vistos foram 108, 81, 61 e 52, para os quais os valores de R foram 242, 255, 194 e 227, respectivamente. Levando isso em conta, os valores de G que caracterizarão os pixels do ponto laser terão de ser menores que 40, ou 100 unidades menores que os valores de R.

Já os valores de B variam bastante, sendo nos casos estudados entre 2 e 237. Percebe-se que vários dos pixels destacados e analisados tem cor rosada ou roxa, nesses pixels os valores de B se aproximam bastante daqueles de R, sendo quase sempre menores que eles, com exceção de quatro casos, um em T4, um em T8 e dois em T12. Esses pixels rosados e roxos não podem ser desprezados na busca do programa pelo laser na tela. Com isso em mente, os valores de B que caracterizarão os pixels do ponto laser terão de ser simplesmente menores do que os de R.

Os parâmetros definidos irão descartar alguns dos pixels selecionados anteriormente, mas é preferível que alguns dos pixels pertencentes ao laser não sejam identificados, do que pixels que não pertencem ao laser sejam identificados como pertencendo a ele. Em outras palavras, é melhor ter parâmetros conservadores, ainda mais levando em conta que a calibração pode ser facilmente repetida pelo usuário.

4.3 Teste da Função de Calibração e Resultados

As imagens apresentadas na Figura 42 foram utilizadas para testar os parâmetros definidos na última seção. A região da tela dessas imagens foi recalculada usando o mesmo passo a passo citado no item 5.2.3.1, e foi destacada de verde ($R=0$, $G=255$, $B=0$). A imagem com a região da tela destacada foi salva em uma nova variável, denominada *img_teste*, deixando assim a variável *imagem2* inalterada. Esta última foi mantida inalterada para que os pixels da sua região da tela possam ser comparados com os parâmetros definidos no último passo, sem que aqueles próximos às extremidades da tela tenham sofrido alterações pelo realce da mesma.

Após a identificação e destaque da tela, os pixels da *imagem2* na região identificada foram comparados com os parâmetros citados no final do item 5.5.3.4. Todos aqueles que atendiam aos parâmetros, ou seja, que foram identificados como pertencentes à projeção do laser, foram realçados em vermelho ($R=255$, $G=0$, $B=0$) na variável *img_teste*. O resultado final é uma imagem com as bordas da região da tela destacadas em verde e todos os pixels circunscritos nessa região que foram identificados como sendo do ponto laser, destacados em

vermelho. Isso foi feito para todas as imagens que foram apresentadas na Figura 42, e o resultado é exibido a seguir, na Figura 46.



Figura 46: Teste dos parâmetros de identificação do ponto laser.

Em nenhum dos testes houve ocorrência de pixels selecionados pelos critérios em locais diferentes daqueles efetivamente relacionados à projeção do laser, o que é uma demonstração que os critérios escolhidos para definir a identificação do laser são satisfatórios. A única exceção a isso ocorreu no teste T8, no qual aparecem pixels vermelhos referentes ao reflexo do emissor laser na tela, ressaltando a necessidade de evitar a aparição deste reflexo por meio de uma leve inclinação da tela.

Apenas nos testes T1 e T10 não houve detecção do laser na tela. Inclusive nos testes T3 e T7, que foram descartados da análise feita no item 5.2.3.4, foram detectados pixels que satisfazem os critérios estabelecidos.

Para melhor visualizar os pontos, os mesmos foram destacados na variável *imagem2*, na qual estavam inalterados. Eles foram ampliados da mesma forma que nas imagens 42 e 43,

destacando uma imagem de 10x10 pixels. Para permitir a visualização de quais destes pixels satisfizeram os critérios, eles foram agrupados em uma única imagem, apresentada na Figura 47 a seguir.

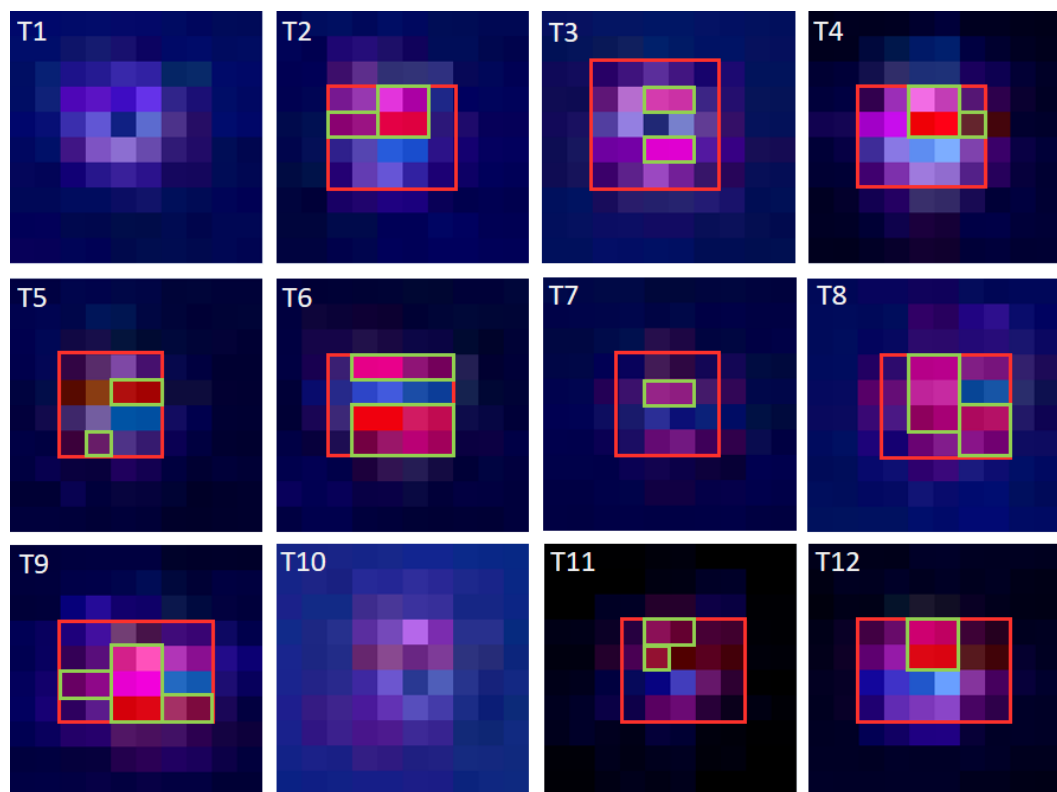


Figura 47: Destaque dos pixels referentes ao ponto laser de cada imagem que atenderam aos critérios estabelecidos.

Os pixels destacados nos quadriláteros verde claro foram os identificados como satisfazendo o critério. Já os quadriláteros avermelhados destacam os pixels que caracterizam o laser, demarcados visualmente com intuito de permitir uma análise mais detalhada. As quantidades de pixels destacados estão apresentadas na Tabela 4 a seguir:

Tabela 4: Comparação da quantidade de pixels selecionados pelo critério com o total de pixels pertencentes à projeção laser.

Teste	Pixels selecionados	Total de pixels	% de pixels identificados
T2	6	20	30,0%
T3	4	25	16,0%
T4	5	20	25,0%
T5	3	16	18,8%
T6	12	20	60,0%

T7	2	16	12,5%
T8	10	20	50,0%
T9	10	24	41,7%
T11	3	20	15,0%
T12	4	24	16,7%
Total	59	205	28,8%

Usando um espaço amostral de 10 amostras, constatou-se que os critérios estabelecidos para a identificação da projeção laser identificam em média 28,8% dos pixels que de fato caracterizam tal projeção.

Originalmente foram selecionados 40 pixels de maneira visual entre aqueles das Figuras 43 e 44. Com a aplicação dos parâmetros numéricos, foram identificados nos testes T2, T4, T5, T6, T8, T9, T11 e T12, 53 pixels, que somados aos dos testes T3 e T7, totalizam 59 pixels. Para que ocorresse esse aumento em relação aos 40 pixels selecionados visualmente, houve identificação de novos pixels, mas também alguns descartes daqueles previamente selecionados (de forma visual) por não atenderem aos requisitos estabelecidos. Mas, a quantidade de descartes foi menor que a de novos pixels que atenderam aos parâmetros.

Um maior número de pixels identificados resulta em mais elementos sendo usados para se determinar as coordenadas do ponto laser na tela, gerando uma calibração mais precisa e mais satisfatória.

5 Considerações Finais:

O estudo realizado neste trabalho foi capaz de aprimorar o software desenvolvido no trabalho de MORAES (2015), aumentando a sua aplicabilidade através de algumas alterações estéticas de layouts e da adição de uma função de calibração. O uso desta função de calibração permite que a câmera possa ser fixada à arma nas suas partes externas, como nos trilhos. Com isso, foi descartada a necessidade do projeto de um dispositivo para fixação da câmera na saída do cano da arma sem danificá-la, como proposto no trabalho de MORAES (2015).

Para tornar o uso do software mais intuitivo, a *interface regras*, que continha tanto as instruções para o uso do programa quanto as regras do ISSF e do IPC, foi dedicada exclusivamente às instruções de uso do programa. O texto original das instruções também foi alterado para ficar mais fácil de se interpretar e se adequar às alterações realizadas no software. Com essas duas alterações espera-se que os usuários tenham mais facilidade para realizar os treinos.

O estudo realizado sobre o software desenvolvido por MORAES (2015) atestou o seu caráter inovador quanto ao método de tratamento de imagem usado para identificar qual ponto da tela a arma estaria alvejando. Esse método foi complementado pela implementação do sistema de calibração laser, que utiliza do dispositivo de captura de imagens já usado pelo software original, para identificar o laser projetado na região da tela.

A identificação do ponto laser projetado pelo colimador laser é feita pelo tratamento das imagens capturadas pela câmera, seguida de uma comparação de todos os pixels da região da tela com três parâmetros que definem os pixels pertencentes ao laser. Esses parâmetros foram criados de maneira conservadora, de modo que não ocorreu nenhum falso positivo nos testes realizados para a elaboração deste trabalho, quando as recomendações de operação foram seguidas. Mesmo assim, foi-se identificado uma média de 28,8% dos pixels pertencentes ao laser nas imagens capturadas pela câmera.

As alterações realizadas no software desenvolvido por MORAES (2015) resultaram em um programa com mais aplicabilidade prática. O aumento de custo dos hardwares utilizados se limita àquele referente ao colimador laser. Assim como no trabalho que serviu de ponto de partida para este, os esforços computacionais permitiram o uso de hardwares de baixo custo sem prejudicar a qualidade do resultado final.

Dentre as funções desenvolvidas no trabalho de MORAES (2015), 4 delas sofreram alterações, a *callback_botao.m*, a *gatilho.m*, a *InterfaceResize.m* e a *telaalvo.m*. Além disso uma nova função foi elaborada, a *calibracao.m*. Todas elas podem ser vistas em detalhes nos apêndices de A, B, C, D e E, localizados no final deste trabalho.

6 **Trabalhos Futuros:**

Para aprimorar os trabalhos já realizados, sugere-se um estudo sobre o efeito da gravidade e de ventos sobre os projéteis disparados nos treinos de tiro esportivo, usando para isso as categorias com distância maiores e portanto mais críticas.

Além disso, recomenda-se uma análise da resolução mínima requerida pela câmera para que se tenha um resultado satisfatório, levando em consideração a quantidade de pixels que vai compor a tela da interface nas imagens capturadas. Essa quantidade de pixels será função não apenas da resolução da câmera, mas também da distância entre o usuário e a tela.

Bibliografia

Antunes, F. (22 de 02 de 2011). *Plug and Blast: Super Scope (SNES) - Nintendo Blast*.

Acesso em 19 de 11 de 2016, disponível em Nintendo Blast:

<http://www.nintendoblast.com.br/2011/02/plug-and-blast-super-scope-snes.html>

Arencibia, R., & Sousa, C. (2011). Teoria para aulas práticas de metrologia. Uberlândia, MG, Brasil.

ASSMANN, A. (2010). *As Associações de Tiro ao Alvo em Santa Cruz de sul/ Rio Grande do Sul: da fundação a nacionalização*. Porto Alegre.

ASSMANN, A., & MAZO, J. (2012). AS SCHÜTZENVEREINE – Sociedades de Atiradores – de Santa Cruz do Sul: um tiro certo na história do esporte no Rio Grande do Sul. *Esporte e Sociedade*, 122 e 123.

AUTORES DO PORTAL BRASIL 2016. (2016). *Portal Brasil 2016*. Fonte: Guilherme Paraense: <http://www.brasil2016.gov.br/pt-br/olimpiadas/medalhistas/guilherme-paraense>

Bueschel, R. M. (1995). *Collector's Guide to Vintage Coin Machines*. Atglen - PA: Schiffer Publishing.

CBTE. ([201?]). *Onde posso praticar o Tiro Esportivo em meu estado?* Fonte: Confederação Brasileira da Tiro Esportivo: <http://www.cbte.org.br/>

Comitê Olímpico Brasileiro. (2016). *Tiro Esportivo Olímpico - Olimpíadas Rio 2016*. Acesso em 15 de abril de 2016, disponível em <http://www.rio2016.com/tiro-esportivo>

EDITORA ABRIL S.A. ([entre 2010 e 2016]). *Abril de 1896: Primeira Olimpíada | Especial: A Conquista da Grécia*. Fonte: VEJA na história: <http://veja.abril.com.br/historia/olimpiada-1896/especial-jogos-olimpicos-era-moderna-atenas.shtml>

EDITORES DO PORTAL EDUCAÇÃO FÍSICA. (13 de Abril de 2012). *I Jogos Olímpicos da Era Moderna: Atenas 1896*. Fonte: Portal Educação Física: <http://www.educacaofisica.com.br/esportes/jogos-olimpicos2/i-jogos-olimpicos-da-era-moderna-atenas-1896/>

Escritores do site Jukeboxhistory.info. (2010). *Seeburg History*. Fonte: American Jukebox History: <http://www.jukeboxhistory.info/Seeburg/SEE-history.html>

Gegan, S., & Winter, D. (16 de 10 de 2012). *Magnavox Odyssey*. Acesso em 07 de 11 de 2016, disponível em Magnavox Odyssey: <http://www.magnavox-odyssey.com/>

INTERNATIONAL OLYMPIC COMMITTEE. (2016). *London 2012 Shooting - Results & Videos*. Fonte: Olympic Results - Official Records: <https://www.olympic.org/london-2012/shooting>

INTERNATIONAL OLYMPIC COMMITTEE. (2016). *Shooting Results - Athens 1896 | Olympic Medals*. Acesso em 10 de março de 2016, disponível em Olympic Results - Official Records: <http://www.olympic.org/content/results-and-medalists/gamesandsportsummary/?sport=31611&games=1896%2f1&event=>

IPC. ([2016?]). *Shooting - About the Sport*. Fonte: International Paralympic Committee | IPC: <https://www.paralympic.org/shooting/about>

Jacobsson, J. (13 de Abril de 2012). *Jonas Jacobsson: A Week in the Life of a Shooter*. Fonte: IPC Shooting - Paralympic Shooting Athletes, Events, Categories: <https://www.paralympic.org/blog/jonas-jacobsson-week-life-shooter>

Johnson, D. (22 de 11 de 2014). *Ruger Yukon Air Rifle Test Review*. Acesso em 18 de 02 de 2017, disponível em Hard Air Magazine: <http://hardairmagazine.com/reviews/airgun-test-reviews/ruger-yukon-air-rifle-test-review-177-cal/>

MAZO, J., & GAYA, A. (2006). As associações desportivas em Porto Alegre, Brasil: espaço de representação da identidade cultural teuto-brasileira. *Revista Portuguesa de Ciências do Desporto*, 205-207.

MINISTÉRIO DA CULTURA. ([2016?]). *Tiro Esportivo*. Fonte: Portal Brasil 2016: <http://www.brasil2016.gov.br/pt-br/paraolimpiadas/modalidades/tiro-esportivo>

MORAES, R. A. (2015). *Estudo E Desenvolvimento De Jogos Aplicados Ao Treinamento De Atletas Paralímpicos*. Uberlândia - MG: UFU.

Nintendo®. (1993). *Super Nintendo Entertainment System Development Manual - Book II*.

OS EDITORES DE ENCYCLOPÆDIA BRITANNICA. ([2015]). *shooting | sport | Britanica.com*. Acesso em 15 de abril de 2016, disponível em Encyclopædia Britannica: <http://global.britannica.com/sports/shooting>

OS EDITORES DO SITE A HISTÓRIA. (2016). *História do Tiro*. Fonte: A História: <http://www.ahistoria.com.br/do-tiro/>

Ramy32.exe. (05 de 05 de 2013 [aquivo de vídeo]). *Funzionamento delle Light Gun - Duck Hunt - slowmotion*. Fonte: Youtube: <https://www.youtube.com/watch?v=NBzBkuA1Kz0>

Sic! (09 de 03 de 2007). *File:Nes zapper.jpg*. Fonte: Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Nes_zapper.jpg

THE MATHWORKS. ([2016?]). *Image Types - MATLAB & Simulink*. Fonte: MathWorks - Makers of MATLAB and Simulink: http://www.mathworks.com/help/matlab/creating_plots/image-types.html

The Strong. (2016). *113.6123: Super Nintendo Super Scope and Receiver*. Acesso em 24 de 11 de 2016, disponível em The Strong | National Museum of Play: <http://www.museumofplay.org/online-collections/22/43/113.6123>

Toad_Speedgaming. (12 de 04 de 2007 [arquivo de vídeo]). *Duck Hunt - NES*. Fonte: Youtube: <https://www.youtube.com/watch?v=x-daxzVxrQI>

Volk, O. (03 de 02 de 2014). *Use Lasers to Stay Trained and Sighted In Without Range Time*. Acesso em 19 de 02 de 2017, disponível em AllOutdoor.com - All Outdoor, All The Time: <http://www.alloutdoor.com/2014/02/03/laserlyte-stay-trained-sighted-range-time/>

APÊNDICES

APÊNDICE A – Função *calibracao.m*

```
% Função responsável por calibrar a câmera
function calibracao(objectHandle , eventData)
%% VARIÁVEIS GLOBAIS + Interface
global preto
global botao1 botao2 texto1
global tag
global vid
global x_rd y_rd
global hand
global borda
global xfig yfig
global deltay deltax
global x_ini y_ini
global y_tam

if strcmp(tag,'Calibr.') %1ª execução - Exibir instruções
    carregando();
    Padr = imread(strcat(pwd, '\Imagens\imagem_padrao.png'));
    assignin('base', 'figuradefundo', Padr);
    hand=imagesc(Padr); %Plota a imagem de fundo

    %Ajuste da imagem e chamada da função @InterfaceResize
    axis image;
    set(gca, 'position', [0,0,1,1], 'xtick', [], 'ytick', []);

    set(gcf, 'Position', get(0, 'Screensize'), 'menubar', 'none', 'ResizeFcn', @InterfaceResize)

    %% Calibração da mira
elseif strcmp(tag, 'Calibr.2') %Demais execuções - Calibração em si
    delete(botao2); %atua cajo a calibração seja repetida depois de dar certo
    imagem1=getsnapshot(vid); %imagem normal
    % #####Imagem Escura
    hold on %##
    set(gcf, 'Pointer', 'custom', ... %##
        'PointerShapeCData', NaN(16,16)); %##
    hh=imagesc(preto); %##
    set(botao1, 'Visible', 'off'); %##
    drawnow %##
    %#####Imagem Escura

    try
        [y,Fs, nbits]=wavread('shot');
        sound(0.3*y,Fs,nbits)
    catch ME
    end

    pause(0.3)
```

```

imagem2=getsnapshot(vid); %imagem com a tela escura
##### Fechamento imagem escura
delete(hh) %##
set(gcf, 'Pointer', 'arrow'); %##
set(botao1, 'Visible', 'on'); %##
##### Fechamento imagem escura

imagem3 = imagem1-imagem2; %imagem diferenca com a tela em branco em
destaque
imagem3 = im2bw(rgb2gray(imagem3),0.5);%imagem diferenca binarizada
imagem3 = bwareaopen(imagem3,ceil(0.0002*numel(imagem3)));
imagem1 = im2bw(rgb2gray(imagem1),0.5);
imagem1 = bwareaopen(imagem1,ceil(0.0002*numel(imagem1)));

imagem_inter=and(imagem1,imagem3);
imagem_inter=
bwareaopen(imagem_inter,ceil(0.0002*numel(imagem_inter)));
assignin('base', 'imagem_inter', imagem_inter);
reg_inter = regionprops(imagem_inter, 'PixelList') ;

if ~isempty(reg_inter)
    num_elementos(size(reg_inter,1)) = 0;
    for i=1:size(reg_inter,1)
        num_elementos(i)=numel(reg_inter(i).PixelList);
        %vetor com o numero de pixel de cada parte branca
    end
    regioao=find(num_elementos==max(num_elementos));
    pixels11=reg_inter(regiao).PixelList;
    %Matriz Nx2 onde cada linha contém as coordenadas de um dos pixels
    %brancos da maior região branca da imagem.
    #####

    y = size(imagem1,1);
    x = size(imagem1,2);

    assignin('base', 'pixels11', pixels11);
    dist= sqrt((pixels11(:,1)-0).^2 + (pixels11(:,2)-0).^2);
    vertaux=pixels11(dist==min(dist),:);
    %vertaux contém coordenadas do pixel branco + próximo de (0,0)
    X(1)=vertaux(1,1); Y(1)=vertaux(1,2);

    dist= sqrt( (pixels11(:,1)-x).^2 + (pixels11(:,2)-0).^2);
    vertaux=pixels11(dist==min(dist),:);
    %vertaux contém coordenadas do pixel branco + próximo de (x,0)
    X(2)=vertaux(1,1); Y(2)=vertaux(1,2);

    dist= sqrt( (pixels11(:,1)-x).^2 + (pixels11(:,2)-y).^2);
    vertaux=pixels11(dist==min(dist),:);
    %vertaux contém coordenadas do pixel branco + próximo de (x,y)
    X(3)=vertaux(1,1); Y(3)=vertaux(1,2);

    dist= sqrt( (pixels11(:,1)-0).^2 + (pixels11(:,2)-y).^2);
    vertaux=pixels11(dist==min(dist),:);
    %vertaux contém coordenadas do pixel branco + próximo de (0,y)
    X(4)=vertaux(1,1); Y(4)=vertaux(1,2);

    X=X'; %Transposta de X
    Y=Y'; %Transposta de Y
    c = 0;

```

```

        pixels = fix(y*x*0.0001); %Qtdd de pixels da imagem que poderão
compor o ponto laser;
        rdot = zeros(2,pixels); %Alocando espaço;
        for j=min(X):max(X)
            for i=min(Y):max(Y)
                if imagem2(i,j,1)>=90 && (imagem2(i,j,2)<=40 || ...
                    imagem2(i,j,2)<=imagem2(i,j,1)-100) &&
                    imagem2(i,j,3)<=imagem2(i,j,1)
                        c = c+1;
                        rdot(1,c)=j;
                        rdot(2,c)=i;
                    end
                end
            end
        x_rd = round(mean(rdot(1,rdot(1,:)~=0))); %Média das coordenadas
onde
        y_rd = round(mean(rdot(2,rdot(1,:)~=0))); %foram encontrados pontos
vermelhos;
        xi=x_rd;
        yi=y_rd;

        if ~isnan(xi) %Caso exista 'xi'
            extr = regionprops(borda,'Extrema');
            extr = extr(1).Extrema;

            Xo=extr(:,1);
            Yo=extr(:,2);
            Xo=vertcat(Xo(1:2),Xo(5:6));
            Yo=vertcat(Yo(1:2),Yo(5:6)); %coordenadas das quinas da parte
branca da imagem

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            [Eq_reta]=equacao_reta(X,Y);
            %      A1=Eq_reta(1,1); A2=Eq_reta(2,1); %Dados não usados nos cálculos
            A3=Eq_reta(3,1); A4=Eq_reta(4,1);
            %      B1=Eq_reta(1,2); B2=Eq_reta(2,2); %Dados não usados nos cálculos
            B3=Eq_reta(3,2); B4=Eq_reta(4,2);
            %      X1=Eq_reta(1,3); X2=Eq_reta(2,3); X3=Eq_reta(3,3); %Dados não
usados
            X4=Eq_reta(4,3);

            if X4==0 %se X4=0, a reta 4 não se encontra na perpendicular, e
                %possui um coeficiente angular válido
                xfoto_lado=(B4-(yi-A3*xi))/(A3-A4);
                xfoto_baixo=(B3-(yi-A4*xi))/(A4-A3);
            else
                xfoto_lado=X4;
                xfoto_baixo=xi;
            end
            yfoto_lado=A3*xfoto_lado+yi-A3*xi;
            %deve-se utilizar esta equação em vez da equação da reta4 para
evitar coeficientes angulares inválidos

            [ConversaoY]=equacao_reta([Y(1) Y(4)],[Yo(1) Yo(4)]);
            A=ConversaoY(1,1); B=ConversaoY(1,2);
            yorig_lado=A*yfoto_lado + B;

            [ConversaoX]=equacao_reta([X(3) X(4)],[Xo(3) Xo(4)]);
            A=ConversaoX(1,1); B=ConversaoX(1,2);

```

```

xorig_baixo=A*xfoto_baixo + B;

%COORDENADAS EQUIVALENTES NA TELA:
xfinal=xorig_baixo;
yfinal=yorig_lado;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Projeção de uma marca onde o laser foi identificado:

I = imread(strcat(pwd, '\Imagens\marca.png'));
escala = y_tam/(20*size(I,1));
I = imresize(I,escala);
xf = ceil(xfinal-size(I,1)/2);
yf = ceil(yfinal-size(I,2)/2);

imagesc(xf,yf,I);

mensagem = strcat(sprintf('Calibração efetuada com sucesso!
\nS'),...
    sprintf('e o sinal mostrado divergir do laser no momento do
tiro, calibre novamente. \nU'),...
    'ma possível causa da divergência são reflexos de luzes
vermelhas na tela!');
cor = [0.1 0.4 0.6]; %Esverdeado
botao2 = uicontrol('Style', 'pushbutton', 'String', 'Começar!',...
    'FontSize',deltay/50,'ForegroundColor',[0 0 0.6],'Position',...
    [0.77*(deltax)+x_ini 0.04*(deltay)+y_ini 0.08*(deltax)
0.05*(deltay)],...
    'Callback',@callback_botao,'tag','TelaAlvo');
else
    mensagem = 'O laser não foi identificado na região da tela.';
    cor = [0.7 0 0]; %Avermelhada
end
else
    mensagem = 'Tela não identificada pela câmera. Tente novamente!';
    cor = [0.7 0 0]; %Avermelhada
end
delete(textol);
textol=text(0.5*xfig,0.20*yfig,mensagem,'Color',cor,...
    'fontsize',deltay/50,'HorizontalAlignment', 'center');
%Mensagem de erro (avermelhada) ou acerto (esverdeada)
end

```

APÊNDICE B – Função *callback_botao.m*

```
function callback_botao( objectHandle , eventData)
global pol
global edit1 edit2
global selec_lista1 selec_lista2
global pontos
global modalidades
global modalidadesselecionada
global datas
global alvo
global D
global distancia
global secs mins horas
global maxtiros
global mod
global timerobj
global tag
global figuradefundo
global vid dist_tela borda

try
    load('participantes.mat');
catch ME
end

switch get(objectHandle, 'Tag');

    case {'ErrorDist'}
        msgbox('Informe a sua distância da tela.', 'Error','error');
    case {'polegada'} %okpolegada
        valor = get(edit1, 'String');
        pol=str2num(valor);
        fid3 = fopen('Polegadas.txt','w+');
        fprintf(fid3, '%d', pol);
        fclose(fid3);
        InterfaceResize(gcf, []); %para atualizar que o monitor foi
cadastrado

    case {'distancia'}
        valor = get(edit2, 'String');
        dist=str2double(valor);
        fid3 = fopen('Distancia.txt','w+');
        fprintf(fid3, '%d', dist);
        fclose(fid3);
        InterfaceResize(gcf, []); %para atualizar que a distância foi
cadastrada

    case {'cadastrar'} %okcadastro
        val = get(edit2, 'String');
        if ~isempty(participantes(1).nome)
            participantes(size(participantes,2)+1).nome=val;
            save('participantes.mat','participantes');
        else
            participantes(1).nome=val;
            save('participantes.mat','participantes');
        end
        InterfaceResize(gcf, []);
```

```

case {'deletar'} %deleta
load('participantes.mat');
choice = questdlg(strcat('Deseja excluir: ',...
    participantes(selec_listal).nome,'?'), ...
    'Deletar', ...
    'Sim','Nao','Sim');
switch choice
case 'Sim'
    participantes(selec_listal)=[];
    save('participantes.mat','participantes');
    selec_listal=1;
    InterfaceResize(gcf,[]);
case 'Nao'
end

case {'renomear'} %renome
load('participantes.mat');
x = inputdlg(strcat('Novo nome
para:',participantes(selec_listal).nome),...
    'Renomear', [1 50]);
if ~isempty(x)
    participantes(selec_listal).nome=x;
    save('participantes.mat','participantes');
    InterfaceResize(gcf,[]);
end

case {'grafico_data'} %grafico_data
if ~isempty(pontos)
    figure (4)
    subplot(2,1,1), plot(1:size(pontos,2),pontos,'--rs',...
        'LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor',...
        'g','MarkerSize',5);
    set(gca,'YLim',[0,10.9]);
    title('Distribuição dos pontos','FontWeight','bold');
    xlabel('Contagem de tiros');
    ylabel('Pontos');

    x=0:0.1:10.9;
    aux=0;
    for i=0:0.1:10.9
        aux=aux+1;
        y(aux)=numel(find((pontos==i)~=0));
    end
    subplot(2,1,2),bar(x,y);
    set(gca,'XTick',linspace(0,11,23))
    title('Gráfico de frequência dos pontos','FontWeight','bold');
    xlabel('Pontos');
    ylabel('Frequência');

    set(gcf,'Position',get(0,'Screensize'),'menubar','none')
else
    msgbox('Pontos para grafico nao encontrados','Grafico');
end

case {'comparar_dadas'}
if ~strcmp(modalidadesselecionada,'>Escolha uma modalidade<')
    for i=1:size(participantes(selec_listal).modalidade.(char(...
        modalidades(selec_lista2))).dados,2)

pontuacao{i}=sum(participantes(selec_listal).modalidade.(char(...

```

```

        modalidades(selec_lista2))).dados(1,i).pontos);
    end
    figure (4)
    hbar=bar(cell2mat(pontuacao));
    x = get(hbar,'XData');
    y = get(hbar,'YData');

    ygap = 0.1; % Specify vertical gap between the bar and label
    ylims = get(gca,'YLim');
    set(gca,'YLim',[ylims(1),ylims(2)+0.2*max(y)]);
    for i = 1:length(x) % Loop over each bar
        xpos = x(i); % Set x position for the text label
        ypos = y(i) + ygap; % Set y position, including gap
        htext = text(xpos,ypos,datas{i}); % Add text label
        set(htext,'VerticalAlignment','bottom',...
            'HorizontalAlignment','center')
    end
    title('Comparação entre datas da mesma modalidade',...
        'FontWeight','bold');
    xlabel('Datas');
    ylabel('Pontuação');

    set(gcf,'Position',get(0,'ScreenSize'),'menubar','none')
else
    msgbox('Escolha uma modalidade','Comparar datas');
end
case {'IPC'}
    winopen(...
'Regras-e-Regulamentos-de-Tiro-Esportivo-do-IPC-2013-2016-Versão-em-
português.pdf');
case {'ISSF'}
    winopen('ISSFRuleBook2013-2ndPrint-ENG.pdf');
case {'TelaAlvo'}
    telaalvo()

case{'R1'} %Carabina de ar 10m em pé
    alvo=imread(strcat(pwd,'\Imagens\10m_Air_Rifle.png'));
    D1=45.5; D10=0.5;
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=10;
    secs = 0;
    mins = 15;
    horas= 1;
    maxtiros= 60;
    mod= 'R1_Carabina_de_ar_10m_em_pe_M';
    tag = 'Calibr.';
    calibracao()
case{'R2'} %Carabina de ar 10m em pé
    alvo=imread(strcat(pwd,'\Imagens\10m_Air_Rifle.png'));
    D1=45.5; D10=0.5;
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=10;
    secs = 0;
    mins = 50;
    horas= 0;
    maxtiros= 40;
    mod= 'R2_Carabina_de_ar_10m_em_pe_F';
    tag = 'Calibr.';
    calibracao()
%
    telaalvo()

```

```

case{'R3'} %Carabina de ar 10m deitado
    alvo=imread(strcat(pwd, '\Imagens\10m_Air_Rifle.png'));
    D1=45.5; D10=0.5;
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=10;
    secs = 0;
    mins = 50;
    horas= 0;
    maxtiros= 60;
    mod= 'R3_Carabina_de_ar_10m_deitado';
    tag = 'Calibr.';
    calibracao()
%     telaalvo()
case{'R6'} %Carabina .22 50m deitado
    alvo=imread(strcat(pwd, '\Imagens\50m_22_Rifle.png'));
    D1=154.4; D10=10.4;
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=50;
    secs = 0;
    mins = 50;
    horas= 0;
    maxtiros= 60;
    mod= 'R6_Carabina_22_50m_deitado';
    tag = 'Calibr.';
    calibracao()
%     telaalvo()
case{'R7'} %Carabina .22 50m 3 posições
    alvo=imread(strcat(pwd, '\Imagens\50m_22_Rifle.png'));
    D1=154.4; D10=10.4;
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=50;
    secs = 0;
    mins = 45;
    horas= 2;
    maxtiros= 120;
    mod= 'R7_Carabina_22_50m_3_posicoes_M';
    tag = 'Calibr.';
    calibracao()
%     telaalvo()
case{'R8'} %Carabina .22 50m 3 posições
    alvo=imread(strcat(pwd, '\Imagens\50m_22_Rifle.png'));
    D1=154.4; D10=10.4; %Diâmetro do maior e menor círculo do alvo
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=50;
    secs = 0;
    mins = 45;
    horas= 1;
    maxtiros= 60;
    mod= 'R8_Carabina_22_50m_3_posicoes_f';
    tag = 'Calibr.';
    calibracao()
%     telaalvo()
case{'P1'} %Pistola de ar 10m
    alvo=imread(strcat(pwd, '\Imagens\10m_Air_Pistol.png'));
    D1=155.5; D10=11.5;
    D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
    distancia=10;
    secs = 0;
    mins = 15;
    horas= 1;
    maxtiros= 60;

```

```

mod= 'P1_Pistola_de_ar_10m_M';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'P2'} %Pistola de ar 10m
alvo=imread(strcat(pwd, '\Imagens\10m_Air_Pistol.png'));
D1=155.5; D10=11.5;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=10;
secs = 0;
mins = 50;
horas= 0;
maxtiros= 40;
mod= 'P2_Pistola_de_ar_10m_F';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'P3'} %Pistola Sport 25m
alvo=imread(strcat(pwd, '\Imagens\25m_50m_Pistol.png'));
D1=500; D10=50;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=25;
secs = 0;
mins = 0;
horas= 10;
maxtiros= 60;
mod= 'P3_Pistola_Sport_25m';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'P4'} %Pistola Livre 50m
alvo=imread(strcat(pwd, '\Imagens\25m_50m_Pistol.png'));
D1=500; D10=50;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=50;
secs = 0;
mins = 30;
horas= 1;
maxtiros= 60;
mod= 'P4_Pistola_Livre_50m';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'P5'} %Pistola de ar 10m Standard
alvo=imread(strcat(pwd, '\Imagens\10m_Air_Pistol.png'));
D1=155.5; D10=11.5;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=10;
secs = 0;
mins = 0;
horas= 10;
maxtiros= 40;
mod= 'P5_Pistola_de_ar_10m_Standard';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'R4'} %Carabina de ar 10m em pé
alvo=imread(strcat(pwd, '\Imagens\10m_Air_Rifle.png'));
D1=45.5; D10=0.5;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=10;

```

```

secs = 0;
mins = 15;
horas= 1;
maxtiros= 60;
mod= 'R4_Carabina_de_ar_10m_em_pe';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'R5'} %Carabina de ar 10m deitado
alvo=imread(strcat(pwd, '\Imagens\10m_Air_Rifle.png'));
D1=45.5; D10=0.5;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=10;
secs = 0;
mins = 50;
horas= 0;
maxtiros= 60;
mod= 'R5_Carabina_de_ar_10m_deitado';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'R9'} %Carabina .22 50m deitado
alvo=imread(strcat(pwd, '\Imagens\50m_22_Rifle.png'));
D1=154.4; D10=10.4;
D=[linspace(D1,D10,91) linspace(0.9*D10,0.1*D10,9)];
distancia=50;
secs = 0;
mins = 0;
horas= 1;
maxtiros= 60;
mod= 'R9_Carabina_22_50m_deitado';
tag = 'Calibr.';
calibracao()
%      telaalvo()
case{'parar'}
try
    participantes(selec_listal).modalidade.(mod...
        ).dados(1,end+1).data=clock;
    participantes(selec_listal).modalidade.(mod...
        ).dados(1,end).pontos=pontos;
catch ME
    participantes(selec_listal).modalidade.(mod) ...
        .dados.data=clock;
    participantes(selec_listal).modalidade.(mod) ...
        .dados.pontos=pontos;
end
save('participantes.mat','participantes');
h = waitbar(0,'Capacidade máxima de tiros realizada');
steps = 3000;
for step = 1:steps
    % computations take place here
    waitbar(step / steps)
end
callback_botao(uicontrol('tag','Desempenho'),[]);
close(h)

case{'Calibr.2'}
tag='Calibr.2';
carregando();
borda = imread(strcat(pwd, '\Imagens\borda.png'));

```

```

borda=im2bw(borda);
assignin('base','figuradefundo',borda);
colormap(gray);
hand=imagesc(borda); %Plota a imagem 'borda'
%Ajuste da imagem e chamada da função @InterfaceResize:
axis image;
set(gca,'position',[0,0,1,1],'xtick',[],'ytick',[]);

set(gcf,'Position',get(0,'Screensize'),'menubar','none','ResizeFcn',@Interf
aceResize)
try
    warning off MATLAB:HandleGraphics:ObsoletedProperty:JavaFrame
    pause(0.4)
    set(get(gcf,'JavaFrame'),'Maximized',1)
catch ME
end
set(hand,'ButtonDownFcn',@calibracao); %Chama a função de
calibração quando o mouse é clicado
if isempty(vid)
    pause(1)
    msgbox('Selecione uma webcam no menu de configurações',
'Error','error');
end

case{'Iniciar','Instrucoes','Desempenho','Config','Calibr.'}
%oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
figuradefundo=imread(strcat(pwd,'\Imagens\imagem_padrao.png'));
try
    stop(timerobj);
catch ME
end

try
    delete(timerobj);
catch ME
end

try
    pol=importdata('Polegadas.txt');
    if isempty(pol)
        pol=19;
    end
catch ME
    pol=19;
end

if strcmp(get(objectHandle,'Tag'),'Iniciar')
    tag='Iniciar';
end

if strcmp(get(objectHandle,'Tag'),'Instrucoes')
    tag='Instrucoes';
end

if strcmp(get(objectHandle,'Tag'),'Desempenho')
    tag='Desempenho';
end

if strcmp(get(objectHandle,'Tag'),'Config')

```

[illegible]

PÊNDICE C – Função *gatilho.m*

```
function gatilho( objectHandle , eventData)
global vid
global Xo Yo xc yc
global diametro
global selec_listal
global tiro_txt1 tiro_txt2 tiro_txt3 tiro_txt4
global pontos
global numerotiros
global D
global maxtiros
global mod
global H
global deltax
global xfig yfig
global sl_slider sl_p1 sl_p2
global preto
global x_rd y_rd

try
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if numerotiros==maxtiros %Utilizado caso o tempo de treino acabe
    h = waitbar(0, 'Capacidade máxima de tiros realizada');
    steps = 3000;
    for step = 1:steps
        waitbar(step / steps)
    end
    callback_botao(uicontrol('tag', 'Desempenho'), []);
    close(h)
end

if rem(numerotiros,2)==0
    try
        delete(H);
        clear global H;
        global H;
    catch ME
    end
end

numerotiros=numerotiros+1;

load('participantes.mat');

if isempty(selec_listal)
    selec_listal=1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
imagem1=getsnapshot(vid); %imagem normal

#####Imagem Escura
hh=imagesc(preto); %##
set(sl_slider, 'Visible', 'off'); %##
set(sl_p1, 'Visible', 'off'); %##
set(sl_p2, 'Visible', 'off'); %##
drawnow %##
#####Imagem Escura
```

```

yi= y_rd;
xi= x_rd;
%Valores importados da função 'Calibracao'

try
    [y, Fs, nbits]=wavread('shot');
    sound(0.3*y,Fs,nbits)
catch ME
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try

    pause(0.3)
    imagem2=getsnapshot(vid); %imagem com a tela escura
    %##### Fechamento imagem escura
    delete(hh) %##
    set(sl_slider,'Visible','on'); %##
    set(sl_p1,'Visible','on'); %##
    set(sl_p2,'Visible','on'); %##
    %##### Fechamento imagem escura

    imagem3= imagem1-imagem2; %imagem diferenca com a tela em branco em
destaque
    imagem3=im2bw(rgb2gray(imagem3),0.1);%imagem diferenca binarizada
    imagem3 = bwareaopen(imagem3,ceil(0.002*numel(imagem3)));
    %bwareaopen: Elimina em Imagem3, objetos com - que 'ceil...' pixels
    %numel:Nº of Elements | ceil: Arredonda pra cima
    imagem1=im2bw(rgb2gray(imagem1),0.5);
    imagem1 = bwareaopen(imagem1,ceil(0.002*numel(imagem1)));

    imagem_inter=and(imagem1,imagem3);
    imagem_inter=
bwareaopen(imagem_inter,ceil(0.002*numel(imagem_inter)));
    reg_inter = regionprops(imagem_inter,'PixelList');
    %reg_inter: estruct onde cada estrutura contém as coordenadas dos
    %pixels de uma das regiões brancas

    for i=1:size(reg_inter,1)
        num_elementos(i)=numel(reg_inter(i).PixelList);
        %Vetor com o numero de pixel de cada parte branca
    end
    regiao=find(num_elementos==max(num_elementos));

    pixels11=reg_inter(regiao).PixelList;
    %Matriz Nx2 onde cada linha contém as coordenadas de um dos pixels
    %brancos da maior região branca da imagem.
    %#####

    y = size(imagem1,1);
    x = size(imagem1,2);

    dist= sqrt((pixels11(:,1)-0).^2 + (pixels11(:,2)-0).^2);
    vertaux=pixels11(dist==min(dist),:);
    %vertaux: coordenadas do pixel branco + próximo de (0,0)
    X(1)=vertaux(1,1); Y(1)=vertaux(1,2);

    dist= sqrt( (pixels11(:,1)-x).^2 + (pixels11(:,2)-0).^2);
    vertaux=pixels11(dist==min(dist),:);
    %vertaux: coordenadas do pixel branco + próximo de (x,0)

```

```

X(2)=vertaux(1,1); Y(2)=vertaux(1,2);

dist= sqrt( (pixels11(:,1)-x).^2 + (pixels11(:,2)-y).^2);
vertaux=pixels11(dist==min(dist),:);
%vertaux: coordenadas do pixel branco + próximo de (x,y)
X(3)=vertaux(1,1); Y(3)=vertaux(1,2);

dist= sqrt( (pixels11(:,1)-0).^2 + (pixels11(:,2)-y).^2);
vertaux=pixels11(dist==min(dist),:);
%vertaux: coordenadas do pixel branco + próximo de (0,y)
X(4)=vertaux(1,1); Y(4)=vertaux(1,2);
X=X'; %Transposta de X
Y=Y'; %Transposta de Y

IN = inpolygon(xi,yi,X,Y);
%Determina se o ponto xi,yi está dentro da região da tela.

catch ME
    Resultado='Errou';
    IN=0;
end

if IN
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [Eq_reta]=equacao_reta(X,Y);
    A1=Eq_reta(1,1); A2=Eq_reta(2,1); A3=Eq_reta(3,1); A4=Eq_reta(4,1);
    B1=Eq_reta(1,2); B2=Eq_reta(2,2); B3=Eq_reta(3,2); B4=Eq_reta(4,2);
    X1=Eq_reta(1,3); X2=Eq_reta(2,3); X3=Eq_reta(3,3); X4=Eq_reta(4,3);

    if X4==0 %se X4=0, a reta 4 não se encontra na perpendicular, e
possui um coeficiente angular válido
        % if X2==0
        xfoto_lado=(B4-(yi-A3*xi))/(A3-A4);
        xfoto_baixo=(B3-(yi-A4*xi))/(A4-A3);
    else
        xfoto_lado=X4;
        xfoto_baixo=xi;
    end
    yfoto_lado=A3*xfoto_lado+yi-A3*xi;
    %deve-se utilizar esta equação em vez da equação da reta4 para
evitar coeficientes angulares inválidos
    % yfoto_baixo=A3*xfoto_baixo+B3; %Variável não utilizada
    [ConversaoY]=equacao_reta([Y(1) Y(4)],[Yo(1) Yo(4)]);
    A=ConversaoY(1,1); B=ConversaoY(1,2);
    yorig_lado=A*yfoto_lado + B;

    [ConversaoX]=equacao_reta([X(3) X(4)],[Xo(3) Xo(4)]);
    A=ConversaoX(1,1); B=ConversaoX(1,2);
    xorig_baixo=A*xfoto_baixo + B;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    xfinal=xorig_baixo;
    yfinal=yorig_lado;
    assignin('base', 'xfinal',
xfinal); %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    assignin('base', 'yfinal',
yfinal); %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    raio(101) = 0; %Locação de espaço

```

```

for i=1:100
    raio(i)=(D(i)/D(1))*diametro/2;
end

I=imread(strcat(pwd, '\Imagens\marca.png'));
I=imresize(I, (raio(1)/10)/size(I,1));

xf=ceil(xfinal-size(I,1)/2);
yf=ceil(yfinal-size(I,2)/2);
aux=0;
for i=1:0.1:10.9
    aux=aux+1;
    if (xfinal-xc)^2 + (yfinal-yc)^2 <=raio(aux)^2 &&...
        (xfinal-xc)^2 + (yfinal-yc)^2>raio(aux+1)^2
        Resultado='Acertou';
        break
    end
end

if (xfinal-xc)^2 + (yfinal-yc)^2 >raio(1)^2
    Resultado='Quase';
end
else
    Resultado='Errou';
end

#####
try
    delete(tiro_txt4);
    delete(tiro_txt5);
catch ME
end

tiro_txt4=text(0.07*xfig,0.47*yfig,'Tiros:', 'fontsize',deltay/80, 'Background
dColor', [1 1 1]);
tiro_txt5=text(0.07*xfig,0.51*yfig, strcat(num2str(numerotiros), '|', ...
num2str(maxtiros)), 'fontsize',deltay/40, 'BackgroundColor', [1 1 1]);
#####

try
    delete(tiro_txt1);delete(tiro_txt2);delete(tiro_txt3);
catch ME
end
switch Resultado
    case {'Acertou'}
        pontos(numerotiros)=i;
        tiro_txt1=text(0.75*xfig,0.2*yfig, '\color{blue}Acertou!', ...
            'fontsize',deltay/40, 'BackgroundColor', [1 1 1]);
        tiro_txt2=text(0.75*xfig,0.25*yfig, strcat('\color{blue}Pontos:
', ...
            num2str(i)), 'fontsize',deltay/40, 'BackgroundColor', [1 1
1]);
        tiro_txt3=text(0.75*xfig,0.30*yfig, strcat('\color{blue}Score:
', ...
            num2str(sum(pontos))), 'fontsize',deltay/40, 'BackgroundColor', [1 1 1]);
        H(end+1)=imagesc(xf, yf, I);

    case {'Quase'}

```

```

        pontos(numerotiros)=0;
        tiro_txt1=text(0.75*xfig,0.2*yfig,'\color{green}Quase! ',...
            'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
        tiro_txt2=text(0.75*xfig,0.25*yfig,'\color{green}Pontos:0
    ',...
            'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
        tiro_txt3=text(0.75*xfig,0.30*yfig,strcat('\color{green}Score:
    ',...
            'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
num2str(sum(pontos)), 'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
H(end+1)=imagesc(xf,yf,I);

        case {'Errou'}
            pontos(numerotiros)=0;
            tiro_txt1=text(0.75*xfig,0.2*yfig,'\color{red}Errou! ',...
                'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
            tiro_txt2=text(0.75*xfig,0.25*yfig,'\color{red}Pontos:0 ',...
                'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
            tiro_txt3=text(0.75*xfig,0.30*yfig,strcat('\color{red}Score:
    ',...
                'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
num2str(sum(pontos)), 'fontsize',deltay/40,'BackgroundColor',[1 1 1]);
        end

        if numerotiros==maxtiros
            try
                participantes(selec_listal).modalidade.(mod).dados(1,end+1).data=clock;

                participantes(selec_listal).modalidade.(mod).dados(1,end).pontos=pontos;
            catch ME
                participantes(selec_listal).modalidade.(mod).dados.data=clock;
                participantes(selec_listal).modalidade.(mod).dados.pontos=pontos;
            end
            save('participantes.mat','participantes');
            h = waitbar(0,'Capacidade máxima de tiros realizada');
            steps = 3000;
            for step = 1:steps
                waitbar(step / steps)
            end
            callback_botao(uicontrol('tag','Desempenho'),[]);
            close(h)
        end
        set(H,'ButtonDownFcn',@gatilho);
    catch ME
        msgbox('Selecione uma webcam no menu de configurações',
            'Error','error');
    end

    screenSizes = get(0, 'MonitorPositions');
    posicao=get(gcf, 'position');
    robot = java.awt.Robot;
    robot.mouseMove(posicao(1)+posicao(3)/2,screenSizes(4)-posicao(2)-
        posicao(4)/2);

```

APÊNDICE D – Seção da Função InterfaceResize.m Referente à Calibração

```

%% Início Calibração:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmp(tag, 'Calibr.')
    delete(texto1); delete(texto2); delete(texto3); delete(texto4);
    delete(botao1); delete(botao2); delete(botao3);
    delete(edit1); delete(edit2);

    %@@INSTRUÇÕES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    texto1 = text(0.02*xfig,0.2*yfig,'Calibração','Color',[0.3 0.3 0.3],...
        'FontWeight','bold','Rotation',45,'EdgeColor',[0.3 0.3 0.3],...
        'LineWidth',3,'fontsize',deltay/30);
    texto2=text(0.15*xfig,0.10*yfig,'\color{black}Instruções:',...
        'FontWeight','bold','fontsize',deltay/40);

    observacoes=importdata('Obs2.txt');
    edit1 = uicontrol(gcf, 'Style','edit', 'units','pixels','FontSize',...
        deltax/40,'Min',0, 'Max',2, 'HorizontalAlignment','left',...
        'Position',[0.15*(deltax)+x_ini 0.27*(deltay)+y_ini ...
            0.7*(deltax) 0.6*(deltay)], 'String',observacoes);

    %@@DISTÂNCIA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    texto3 = text(0.16*xfig,0.77*yfig,'\color{black}Distância da tela em
    metros:',...
        'FontWeight','bold','fontsize',deltay/40);
    edit2 = uicontrol('Style','edit','FontSize',deltay/30,'Position',...
        [0.16*deltax+x_ini 0.13*deltay+y_ini 0.2*deltax 0.06*deltay]);
    botao1 = uicontrol('Style', 'pushbutton', 'String', 'Ok',...
        'FontSize',deltay/50,'tag','distancia','Position', ...
        [0.38*deltax+x_ini 0.13*deltay+y_ini 0.1*deltax 0.06*deltay],...
        'Callback',@callback_botao);
    distancia=importdata('Distancia.txt');

    if isempty(distancia)
        texto4=text(0.16*xfig,0.90*yfig,'Distância não cadastrada',...
            'Color',[0.4 0 0],'fontsize',deltay/60);
        botao2 = uicontrol('Style', 'pushbutton', 'String', 'Continuar',...
            'FontSize',deltay/27,'ForegroundColor',[0.5 0.5 0.5],'Position',...
            [0.60*(deltax)+x_ini 0.1*(deltay)+y_ini 0.18*(deltax)
            0.10*(deltay)],...
            'Callback',@callback_botao,'tag','ErrorDist');
    else
        texto4=text(0.16*xfig,0.90*yfig,['Distância definida: ',...
            num2str(distancia), ' metros'],'Color',[0 0.4
            0],'fontsize',deltay/60);
        botao2 = uicontrol('Style', 'pushbutton', 'String', 'Continuar',...
            'FontSize',deltay/27,'ForegroundColor',[0 0.5 0],'Position',...
            [0.60*(deltax)+x_ini 0.1*(deltay)+y_ini 0.18*(deltax)
            0.10*(deltay)],...
            'Callback',@callback_botao,'tag','Calibr.2');
    end

    botao3 = uicontrol('Style', 'pushbutton', 'String', 'Voltar',...
        'FontSize',deltay/50,'ForegroundColor',[0.5 0 0],'Position',...
        [0.87*(deltax)+x_ini 0.12*(deltay)+y_ini 0.08*(deltax)
        0.05*(deltay)],...

```

```

        'Callback',@callback_botao,'tag','Iniciar');
end

%%%INTERFACE DE CALIBRAÇÃO%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if strcmp(tag,'Calibr.2')
    delete(texto2); delete(botao1);

    dist_tela=importdata('Distancia.txt');
    pol=importdata('Polegadas.txt');

    Instruc = ['Posicione-se a ' num2str(distancia)...
        ' metros da tela, aponte o laser para a região da tela e realize um
disparo'];
    botao1 = uicontrol('Style','pushbutton','String','Voltar',...
        'FontSize',deltay/50,'ForegroundColor',[0.5 0 0],'Position',...
        [0.88*(deltax)+x_ini 0.11*(deltay)+y_ini 0.06*(deltax)
0.05*(deltay)],...
        'Callback',@callback_botao,'tag','Calibr.2');

    if ~isempty(pol)
        texto2=text(0.20*xfig,0.12*yfig,Instruc,...
            'Color',[0 0.4 0],'fontsize',deltay/50);
    else
        texto2=text(0.25*xfig,0.10*yfig,'ERRO: Monitor não cadastrado!',...
            'Color',[0 0.4 0],'fontsize',deltay/50);
    end

    executarwebcam()
    figure(1)
end
%% Fim    Calibração
end

```

APÊNDICE E – Função *telaalvo.m*

```
function telaalvo( objectHandle , eventData)

global borda
global pol
global xc yc xp yp
global diametro
global Xo Yo
global alvo
global sl_txt3
global hand hand2
global xfig yfig
global deltay
global y_tam
global pontos
global numerotiros
global secsElapsed tempotexto
global timerobj
global D
global distancia
global secs mins horas
global horainicial_seg
global maxtiros
global H
global tag

global dist_tela

#####
tag='TelaAlvo';

pontos=[];
H=[];
numerotiros=0;

#####
#####
    carregando();
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    borda=imread(strcat(pwd,'\Imagens\borda.png'));
    assignin('base', 'figuradefundo', borda); %'base':workspace base do
Matlab
    borda=im2bw(borda); %binariza a imagem 'borda'

    extr = regionprops(borda,'Extrema');
    extr = extr(1).Extrema;

    Xo=extr(:,1);
    Yo=extr(:,2);
    Xo=vertcat(Xo(1:2),Xo(5:6));
    Yo=vertcat(Yo(1:2),Yo(5:6)); %coordenadas das quinas da parte branca da
imagem
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    borda=imread(strcat(pwd,'\Imagens\borda.png')); %carrega novamente,
pois havia sido binarizada
    hand=imagesc(borda);
    axis image;
    set(gca,'position',[0,0,1,1],'xtick',[],'ytick',[]);
```

```

set(gcf, 'Position', get(0, 'Screensize'), 'menubar', 'none', 'ResizeFcn', @InterfaceResize)
try
    warning off MATLAB:HandleGraphics:ObsoletedProperty:JavaFrame
    pause(0.2)
    set(get(gcf, 'JavaFrame'), 'Maximized', 1)
catch ME
end

#####
try
    cla(hand) %Close axes
catch ME
end

%size(borda,1)-> altura da imagem de fundo medida com os pixels desta
imagem
%y_tam -> altura da imagem medida com os pixels da resolução de tela
%%%Determinação distância%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dist_tela = importdata('Distancia.txt');
D_mm = dist_tela*D(1)/distancia; %[mm]
pol_emMM = pol * 25.40; %Diagonal da tela [mm]
screenSizes = get(0, 'MonitorPositions');
diagonal=sqrt(screenSizes(3)^2 + screenSizes(4)^2); %[pixels]
pixel_tela_por_mm=diagonal/pol_emMM; %[pixels/mm]
D_pixel=D_mm*pixel_tela_por_mm; %[pixels]
%%%Redimensionamento alvo%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
D_borda = size(borda,1);
diametro = D_borda*D_pixel/y_tam;
escala = diametro/size(alvo,1);
alvo=imresize(alvo,escala);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hold on

xc=size(borda,2)/2;yc=size(borda,1)/2;
xp=xc-size(alvo,2)/2;yp=yc-size(alvo,1)/2;
hand2=imagesc(xp,yp,alvo);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sl_txt3=text(0.07*xfig,0.30*yfig,strcat('\color{red}Aguarde...'),'fontsize'
,...
    deltat/40,'BackgroundColor',[1 1 1],'Tag','Task String');
pause(5);
delete(findobj(gca,'Tag','Task String'));

posicao=get(gcf, 'position');
robot = java.awt.Robot;
robot.mouseMove(posicao(1)+posicao(3)/2,screenSizes(4)-posicao(2)-
posicao(4)/2);
%Move o ponteiro do mouse para a posição desejada após cada disparo
set(hand, 'ButtonDownFcn', @gatilho); %Chama a função @gatilho quando o mouse
é clicado
set(hand2, 'ButtonDownFcn', @gatilho);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

secs = secs + rem(mins,1)*60 + rem(horas,1)*60*60;
mins = floor(mins);
horas = floor(horas);

```

%