ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos

Edson Yuji Tsuzuke

Pedro Paulos Cabral Marques Rosa

Sensoriamento Visual de Movimentos Mandibulares

São Paulo

19 de novembro de 2015

Edson Yuji Tsuzuke
Pedro Paulos Cabral Marques Rosa

Sensoriamento Visual de Movimentos Mandibulares

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Mecatrônica e de Sistemas Mecânicos, da Universidade de São Paulo.

Orientador: Prof. Newton Maruyama

São Paulo
19 de novembro de 2015

DECLARAÇÃO DE ORIGINALIDADE

Este projeto foi realizado com dados secundários, coletados e utilizados somente para o que se referia aos objetivos do mesmo, sendo as informações apresentadas de forma coletiva e com referências, sem qualquer prejuízo para as pessoas envolvidas.

_____ _____

Edson Yuji Tsuzuke Pedro Paulos Cabral Marques Rosa

São Paulo

19 de novembro de 2015

CONSIDERAÇÕES PROFISSIONAIS

Para o desenvolvimento do projeto serão realizados testes com os dois integrantes do grupo: Edson Yuji Tsuzuke e Pedro Paulos Cabral Marques Rosa, com total consciência das consequências geradas na fase de testes do projeto. Os testes serão realizados visando máxima segurança e controle para que nenhum dos integrantes sejam expostos à situações ou atividades prejudiciais à integridade moral e física.

_____ _____

Edson Yuji Tsuzuke Pedro Paulos Cabral Marques Rosa

São Paulo

19 de novembro de 2015

AVALIAÇÃO DO ORIENTADOR

_____

Professor Doutor: Newton Maruyama

São Paulo

19 de novembro de 2015

RESUMO

Neste projeto será realizado o sensoriamento de movimentos mandibulares humanos através da análise e processamento de imagens capturadas por câmeras. Através deste processo, a movimentação mandibular será rastreada e representada graficamente em modelos 2D para estudo e verificação de disfunções da articulação temporomandibular (ATM) em pacientes.

Palavras Chave: ATM, disfunção, sensoriamento, mandibula, câmera, visão

SUMÁRIO

1. INTRODUÇÃO

O sistema estomatognático consiste em um conjunto de estruturas bucais que desenvolvem funções comuns, tendo como característica semelhante o envolvimento da mandíbula, justificando o nome *gnatos* (mandíbula em grego). O sistema estomatognático passa por diversas adaptações e transformações dinâmicas, podendo apresentar modificações morfológicas durante toda a vida de uma pessoa.

Este sistema é controlado pelo sistema nervoso central e é formado por estruturas passivas ou estáticas e por estruturas ativas ou dinâmicas, estruturas estas que são responsáveis por realizar as suas funções que por sua vez são divididas entre funções adaptadas, ligadas ao sorriso, beijo, bocejos, e funções clássicas relacionadas a mastigação, deglutição, fala e respiração.

As estruturas ativas ou dinâmicas são representadas pela unidade neuromuscular, sendo responsáveis por mobilizar as estruturas passivas ou estáticas que são formadas pelos arcos osteodentários, maxila e a mandíbula, sendo estes elementos ligados entre si pela articulação temporomandibular (ATM). A seguir uma ilustração da ATM em um sistema estomatognático:

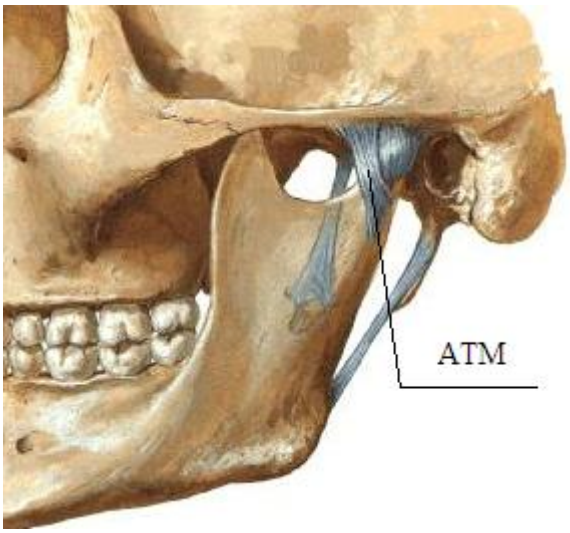


Figura 1: Articulação Temporomandibular [1]

Esta ligação ocorre de maneira tão direta que um mau funcionamento ou desenvolvimento de qualquer uma das estruturas relacionadas, reflete no mau funcionamento das demais. Desta forma, para que haja integridade da saúde física do indivíduo é necessário um equilíbrio saudável entre todas as estruturas.

De acordo com o Instituto Nacional de Saúde dos EUA, mais de dez milhões de americanos apresentam disfunções da ATM, este problema está relacionado a hábitos comuns como morder objetos, roer unhas, mastigar chicletes, prender o telefone com o queixo, ou ainda aperto dentário e bruxismo (ranger os dentes), além de fatores ligados a depressão, ansiedade e estresse.

Disfunções na ATM originam diversas complicações à saúde como principalmente:

- Dores de cabeça (frequentemente parecidas com enxaquecas), dores de ouvido, dor e pressão atrás dos olhos.
- Um "clique" ou sensação de desencaixe ao abrir ou fechar a boca.
- Dor ao bocejar, ao abrir muito a boca ou ao mastigar.
- Mandíbulas que "ficam presas", travam ou saem do lugar.
- Flacidez dos músculos da mandíbula.
- Uma brusca mudança no modo em que os dentes superiores e inferiores se encaixam.
- Perfurações nos tecidos da mandíbula.

A análise dos dados obtidos com o projeto visa explorar a movimentação do sistema estomatognático, auxiliando no tratamento de pessoas que apresentam disfunções temporomandibulares. Este tema traz grande interesse ao grupo, uma vez que aborda o objetivo de auxiliar pessoas, motivação e metas pessoais dos integrantes do grupo, além da utilização prática de métodos e teorias estudadas e pesquisadas durante o curso da Universidade.

2. REVISÃO E ESTADO DA ARTE

Este tema já foi estudado e analisado anteriormente, alguns dos métodos de estudo mais antigos foram dispositivos gráficos de medição, como o criado por W. E. Walker **[2],** em 1896, com o uso de uma agulha marcadora presa por um arco facial nos dentes inferiores, deslizando sobre um disco de marcação fixo em relação à arcada dentária superior, que seria aprimorado pelo uso de ampliadores (pantógrafos) a partir da década de 1950, como se observa na figura abaixo:

Figura 2: Pantógrafo utilizado para marcação mandibular [3]

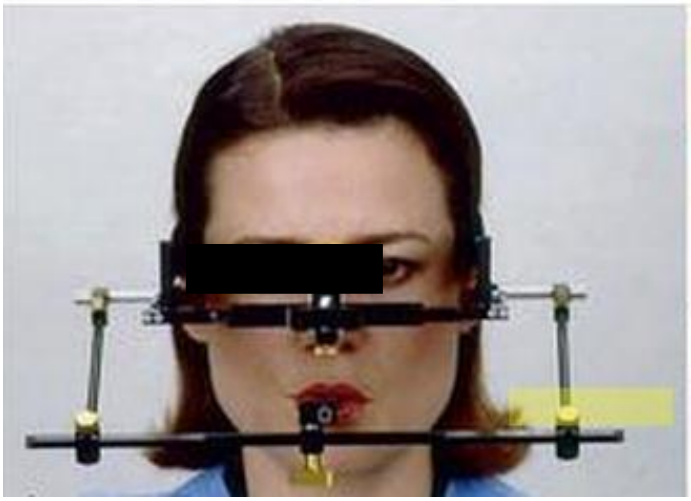


Figura 3: Dispositivo eletrônico de rastreamento da mandíbula[3]

O primeiro método não intrusivo para a captura de movimentos mandibulares surgiu em 1939 com cinerradiografia, usando raios-X frente a uma tela fluorescente (fluoroscopia) que expõe o paciente a contínuas doses de radiação.

Atualmente os instrumentos de diagnóstico mais utilizados são ligados ao uso de radiação, os mais comuns são:

- Radiografias Convencionais: rápidas e relativamente baratas, no entanto mostram apenas a estrutura óssea da articulação.
- 

Figura 4: Radiografia Convencional da vista lateral de uma cabeça humana [4]

- Tomografia Linear: permite melhor visualização em relação à utilização de radiografias convencionais, demonstrando "fatias" da articulação, porém a utilização deste método exige elevado custo e tempo de processo, além disso mostram somente a estrutura óssea do indivíduo.

Figura 5: Tomografia Linear de uma cabeça humana [5]

- Tomografia Computadorizada: este método permite o detalhamento da estrutura óssea utilizando uma dose relativamente pequena de radiação minimizando os efeitos à saúde relacionados com a exposição à radiação, no entanto, os custos são relativamente altos e apenas uma visão limitada do disco articular é disponibilizada.
- 

Figura 6: Tomografia Computadorizada da ATM [6]

- Ressonância Magnética: atualmente é o melhor método para estudar a ATM, nenhuma radiação é utilizada e é capaz de produzir imagens precisas e detalhadas da articulação, entretanto, por ser realizado a partir de equipamentos sofisticados, o custo é extremamente alto quando comparado aos demais métodos.

Figura 7: Ressonância Magnética de mandíbula humana[7]

3. METODOLOGIA

Para o estudo deste projeto, o conhecimento sobre a literatura da área de odontologia é necessário, frente a isso, o contato com membros da Faculdade de Odontologia da USP também é de grande utilidade para orientações gerais e críticas, uma vez que o projeto tem como objetivo ser útil para o auxílio de tratamentos e estudos odontológicos.

O projeto em si consiste no sensoriamento de movimentos mandibulares humanos através da utilização de câmeras, capturando, processando imagens e sinais que serão analisados e comparados.

Para que se possa realizar a captura de imagens de maneira útil, um primeiro estudo envolvendo o posicionamento da câmera tem grande importância. Com base nas características dos equipamentos disponíveis e as dimensões do ambiente de trabalho, espera-se adquirir valores de distância da câmera e ângulo de incidência para que a imagem seja melhor aproveitada. Outra característica importante dos equipamentos a ser considerada é a taxa de captura das câmeras utilizadas, a fim de obter uma captura cinemática em vídeo satisfatória para a observação do movimento. Limitações na capacidade de processamento das imagens podem decorrer de hardware e software utilizado, com cada captura a ser realizada em frações de segundo por cada câmera.

Para que se possa realizar a análise de imagens capturadas da maneira mais satisfatória ao projeto, estudos e testes envolvendo o modo de captura das imagens são de grande importância. Processos como modificação da cor, resolução, quantidade de pontos amostrados e presença de ruídos são de grande consideração no projeto.

A análise das imagens capturadas (e toda a comparação a ser realizada) envolvendo algoritmos SIFT/SURF (mais detalhados na seção 6.1), requer também o estudo de diferentes abordagens em software destes algoritmos para a construção do programa de manipulação de modelos mandibulares.

O desenvolvimento de uma interface homem-máquina simples e intuitiva é de grande importância para a utilização do software e método de análise de imagens e vídeos capturados por dentistas, pesquisadores, estudantes, entre outros, que não tiveram contato com o desenvolvimento e estudos relacionados ao projeto.

A construção de um modelo mecânico para verificação da precisão e acurácia do software e método de captura se faz necessária também. O grupo acha importante o desenvolvimento de um modelo mecânico com dimensões e movimentos definidos que simule o movimento de

abertura e fechamento da boca de uma pessoa, para aplicação do método de captura e do software sobre este mecanismo e posterior comparação e análise entre estes.

Todos estes pontos concatenam para a construção de modelos de movimento mandibular virtual bidimensional e tridimensional manipulável.

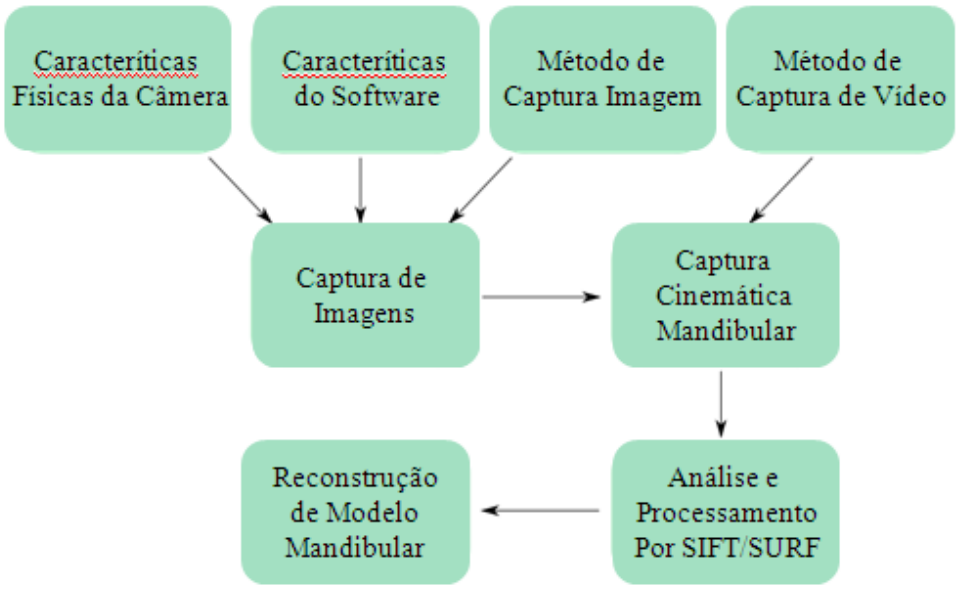


Figura 8: Fluxo de Metodologia do Projeto

4. REQUISITOS DO PROJETO

A elaboração de um projeto compreensivo para o sistema de sensoriamento visual requer consideração das ferramentas utilizadas e dimensões físicas do projeto a fim de garantir sua viabilidade técnica, assim como as considerações de funcionalidade de interface para o usuário final.

Características físicas de quantidade de câmeras e posicionamento devem ser consideradas para a realização do projeto, estas análises foram realizadas nos subtópicos 6.2.2, 6.2.3 e 6.2.4.

A captura do movimento em vídeo acabará por enviar diversas capturas em frações de segundo para serem processadas. Para que o sistema de visão possa calcular os algoritmos de todas as capturas satisfatoriamente em um computador pessoal, a utilização de uma taxa baixa de quadros por segundo é necessária. Uma taxa abaixo de 30 quadros por segundo permite o estudo através de câmeras comerciais capazes de gravar a essa velocidade, no entanto taxas mais altas permitem mais pontos de representação na captura, dando melhor continuidade ao movimento mandibular. Uma faixa de operação englobando valores baixos e altos de taxa de captura será testada no tópico 6.2.5.

As imagens e vídeos devem ser capturados sob métodos que simplifiquem o processamento destes com o uso do software. Controles de cor, ruídos do ambiente e amostragem de pontos foram analisados nos tópicos 6.2.6, 6.2.9 e 6.2.8 respectivamente.

Tendo em vista a importância do acompanhamento visual para a captura de imagens, as câmeras devem proporcionar abertura de imagem suficiente para o enquadramento completo das dimensões da mandíbula, obtendo a melhor resolução possível em pixels (análise no subtópico 6.2.7), além de deixar espaço para possíveis deslocamentos da própria mandíbula e da cabeça. Isso é necessário para garantir o reconhecimento de características faciais e evitar que partes reconhecidas fiquem fora da área de captura, analisada no subtópico 6.2.10.

Por fim, o software a ser programado deverá atender a funcionalidade e intuitividade de funcionamento, permitindo ideal visualização da captura realizada pelo programa.

5. AVALIAÇÃO DAS ALTERNATIVAS DE PROJETO CONSIDERADAS

Para a execução do projeto foram avaliadas as seguintes alternativas de projeto:

- Dispositivo de captura sem controle de posição angular e de distância: as fotos seriam capturadas de posições aleatórias e posteriormente seriam analisas e processadas pelo algoritmo SIFT. A desvantagem observada deste método foi o pior desempenho na localização de pontos de interesse e descritores com a utilização do software online do algoritmo SIFT [8], em relação a posições angulares e de distância calculadas e definidas no tópico 6.2.3 e 6.2.4.
- Utilização de duas câmeras: duas imagens seriam capturadas e processadas simultaneamente utilizando-se o algoritmo SIFT. O grupo analisou que a captura simultânea de imagens não pôde ser realizada no momento atual, uma vez que foi observada a ineficiência do sistema para o processamento simultâneo, a placa com interface FireWire utilizada apresentou limitações nos testes feitos. Foi realizada uma análise comparando-se a utilização de uma ou duas câmeras no projeto, representada no tópico 6.2.2.
- Captura a baixas taxas de captura: a captura de vídeo para análise do SIFT em tempo real seria realizada a taxas de captura de *frames* (quadros) baixa – cerca de 5 FPS. Este método teria a vantagem de prover maior tempo para o processamento do algoritmo SIFT em cada *frame* capturado, no entanto a captura de imagens seria realizada de maneira mais discretizada do que a pretendida pelo grupo, como pode ser observado no tópico 6.2.5.
- Processamento do algoritmo SIFT em *Real-Time*. O processamento se daria simultaneamente à captura de vídeo. Este método teria como vantagem a flexibilidade e maior agilidade de análises realizadas pelo usuário, no entanto, como verificado no subtópico 6.2.11, o desenvolvimento deste método encontra barreiras quanto a utilização de memória dos computadores utilizados pelo grupo, causando lentidão e até interrupção de processamento.

6. DETALHAMENTO DO PROJETO

6.1 BASE TEÓRICA

Pretende-se utilizar um método de captura de imagens para conseguir monitorar o movimento do sistema estomatognático através do uso de câmeras. Sendo desenvolvidos adesivos que são colados ou presos na face da pessoa que contêm os “alvos” para a câmera que captura sinais para o posterior processamento destes. Estes adesivos foram desenvolvidos, visando a não intrusão do movimento mandibular, evitando desta forma, imprecisão e inexatidão dos resultados obtidos.

Foram realizados estudos para escolha de características “ótimas” para obtenção das imagens. Esta é uma consideração importante, uma vez que os movimentos mandibulares são complexos e relacionados a diversos fatores [9]:

- Posição fisiológica inicial que é a relação cêntrica (RC).
- O tipo de movimento: rotação e translação.
- A direção do movimento e o plano em que ocorre.
- O grau do movimento e sua relação com as superfícies oclusais.
- Os significados clínicos do movimento que expressa as diferenças entre pacientes.

Missaka (2007) realizou uma medição semelhante à proposta pelo grupo utilizando um suporte metálico de encaixe oral semelhante a um aparelho odontológico (fig. 9).

Figura 9: Suporte com "alvos" fixados na cavidade oral da pessoa [10]

Para a aquisição de imagens, Missaka utilizou uma câmera digital em um suporte fixo. Para cada frame gravado, a posição dos alvos foi determinada por processamento de imagem. Combinando os movimentos gravados pela câmera com o movimento dos alvos, uma representação espacial do movimento mandibular pode ser analisada.

Depois de determinar os centros geométricos dos alvos, a partir de processamento de imagem, as trajetórias dos movimentos puderam ser estabelecidas, estes movimentos tiveram sua posição relativa subtraídos dos movimentos da cabeça da pessoa, desta forma foi possível a captura de dados mais acurados e precisos. O projeto do grupo visa utilizar mais de uma câmera e discutir a vantagem disto sobre a utilização de apenas uma, além da criação de suportes mecânicos móveis para garantir maior flexibilidade e dados comparativos na obtenção de imagens.

Para a captação de imagens serão utilizados algoritmos de processamento de imagens como o *Scale Invariant Feature Transform* (SIFT), proposto por Lowe, e a variação deste, o *Speed Up Robust Features* (SURF), proposto por Bay et al. Tanto o SIFT quanto o SURF são algoritmos completos que envolvem diversas metodologias relacionadas aos processos de identificação de pontos de interesse, construção de descritores e correspondência entre pontos de interesse.

Estes algoritmos foram desenvolvidos com características que os tornam invariantes a transformações escalares, translação, perspectiva e rotação, alterações nas condições de iluminação do ambiente onde as imagens são obtidas e a existência de ruídos existentes devido ao processo de gravação ou aquisição de imagens. No entanto, a eficiência no processo de localização de pontos de interesse, geração de descritores e correspondência de pontos pode ser comprometida em razão do grau de alteração de algum fator de invariância mencionado acima.

O SIFT tem como objetivo principal o reconhecimento de objetos em cenas reais onde a ocorrência de oclusões é frequente. O SIFT transforma as cenas reais capturadas em fotos em um conjunto de vetores de feições locais (descritores), este conjunto é chamado de espaço-escala. Cada um destes vetores é invariante a translação, rotação e escala e parcialmente invariante a projeções 3D e iluminação. Os descritores são identificados utilizando-se um processo de filtragem em estágios.O primeiro estágio identifica as posições chave no espaço-escala por meio da busca por posições de máximo ou mínimo de uma função diferença de gaussianas (DoG) que é uma aproximação do laplaciano da gaussiana (LoG). Cada ponto encontrado é utilizado para gerar um vetor de feições que descreve uma região da imagem em relação aos eixos da coordenada espaço-escala, ou seja, os pontos detectados têm uma

localização no eixo das escalas (σ) e também nas coordenadas espaciais (x, y) e o vetor de características descreve a vizinha ao redor destes pontos em relação a cada escala utilizada do espaço-escala. Por fim, com esta descrição em várias escalas, é obtida a invariância a escala.

O SURF é uma variação mais robusta e ágil do SIFT. No trabalho de Bauer et al, os algoritmos foram submetidos a diversos testes com o objetivo de avaliação da invariância em relação a rotação, variação escalar, de ruído, de iluminação e de perspectiva, por fim foi realizada uma análise com base na repetibilidade dos pontos em relação às mudanças apresentadas.

A seguir será apresentado um exemplo de processamento de imagem utilizando o algoritmo SIFT, o exemplo é disponibilizado pela publicação online de Rey-Otero e Delbracio [8].

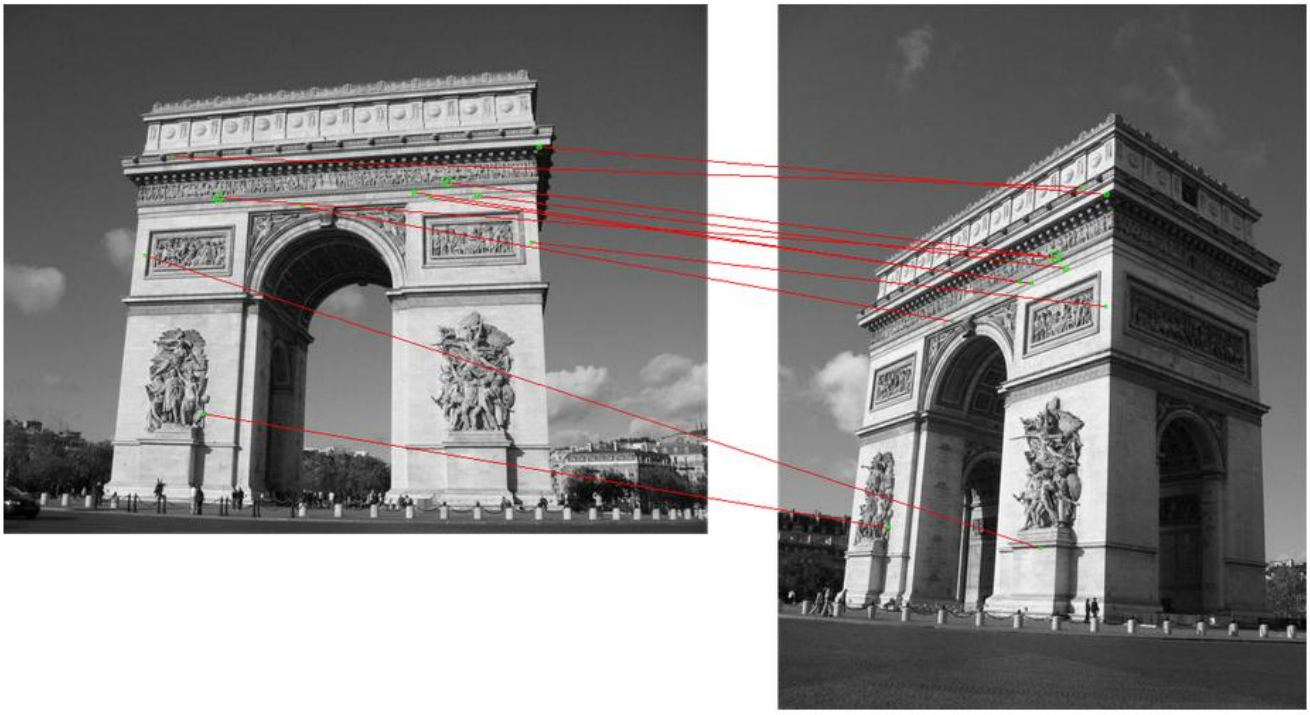


Figura 10: Exemplo de aplicação do SIFT [8]

As linhas vermelhas representam as correspondências de pontos de interesse entre as duas imagens de um mesmo objeto capturadas de posições diferentes. Observa-se grande correspondência de pontos entre as imagens, fato que chama a atenção para a eficiência do algoritmo e colabora na escolha do uso deste pelo grupo, no entanto observa-se também que alguns pontos foram correspondidos de forma incongruente, o que evidencia a dificuldade do projeto na questão de processamento de imagens. O grupo levará em consideração tais dificuldades e irá analisar e elaborar métodos para mitigar erros ou impactos causados pelos

erros encontrados no desenvolvimento do projeto. A seguir será apresentado a imagem com seus descritores:

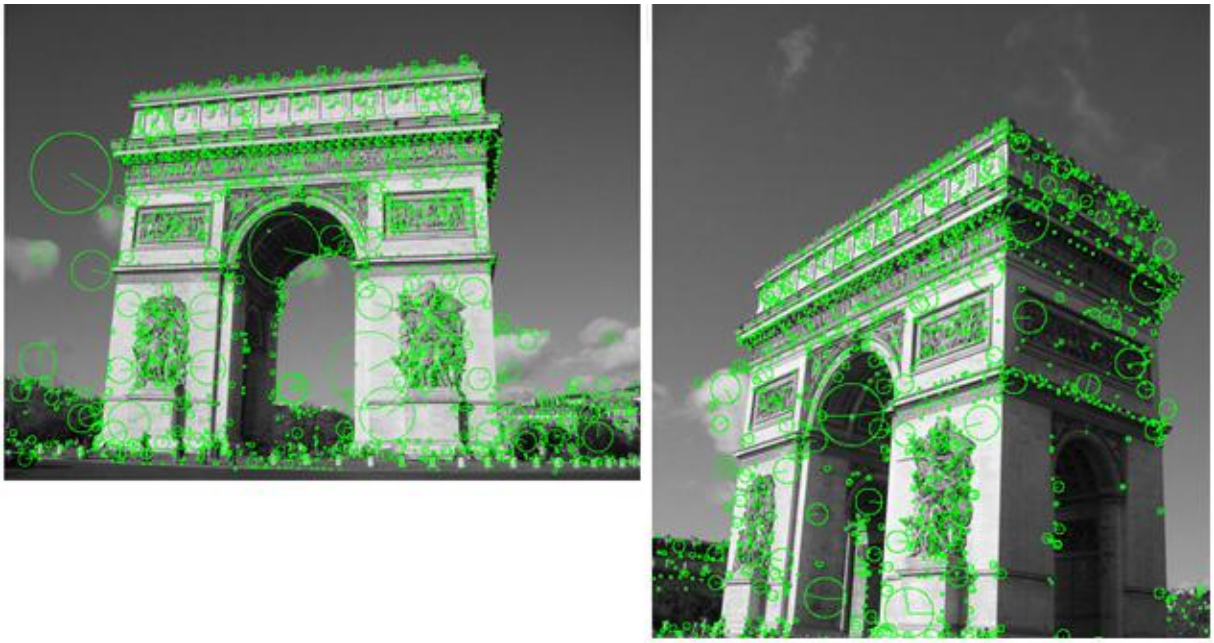


Figura 11: Descritores das imagens[8]

A seguir os pontos de correspondência após a aplicação do algoritmo SIFT nas imagens:

Figura 12: Pontos de correspondência[8]

6.2.RESULTADOS

A seguir os resultados obtidos e analisados pelo grupo.
Os códigos utilizados para as análises dos subtópicos anteriores à 6.2.13 serviram de base para o código completo que se encontra no tópico 10 [Apêndice A], o grupo decidiu expor apenas o código final, uma vez que este engloba todos os códigos de todas as análises realizadas neste tópico.

6.2.1. CÂMERAS UTILIZADAS

Para a obtenção dos resultados e análises foram utilizadas três câmeras: o dispositivo DFM72BUC02, o dispositivo DFK31BF03 e o dispositivo Microsoft LifeCam HD-3000.

- DFM72BUC02:

Figura 13: Dispositivo DFM72BUC02 [11]

- Especificações:
  - Interface: USB
  - Resolução máxima: 2592x1944 pixel
  - Velocidade de captura máxima: 6 imagens/segundo
  - Formato de cor: Y800, RGB32
  - Tamanho do pixel: H: 2.2 µm, V: 2.2 µm
  - Voltagem de alimentação: 4.5 a 5.5 VDC
  - Corrente consumida: 250 mA à 5 VDC
  - Dimensões mecânicas: H: 30 mm, W: 30 mm, L: 15 mm

- Massa: 7g
- Máxima temperatura de operação: -5 ºC a 45 ºC
- Máxima temperatura de repouso: -20 ºC a 60 ºC

- DFK31BF03:

Figura 14: Dispositivo DFK31BF03 [11]

- Especificações:
  - Interface: Firewire
  - Resolução máxima: 1024x768 pixel
  - Velocidade de captura máxima: 30 imagens/segundo
  - Formato de cor: UYVY, BY8
  - Tamanho do pixel: H: 4.65 µm, V: 4.65 µm
  - Voltagem de alimentação: 8 a 30 VDC
  - Corrente consumida: 200 mA à 12 VDC
  - Dimensões mecânicas: H: 50.6 mm, W: 50.6 mm, L: 56 mm
  - Massa: 265g
  - Máxima temperatura de operação: -5 ºC a 45 ºC
  - Máxima temperatura de repouso: -20 ºC a 60 ºC

- Microsoft LifeCam HD-3000:

Figura 15: Dispositivo Microsoft LifeCam HD-3000

- Especificações:
  - Interface: USB 2.0
  - Resolução máxima: 1280x720 pixel
  - Velocidade de captura máxima: 30 imagens/segundo
  - Tamanho do pixel: H: 4.65 µm, V: 4.65 µm
  - Foco fixo em 0,3m a 1,5m
  - Frequência entre 200Hz a 20kHz
  - Dimensões mecânicas: H: 109 mm, W: 44.5 mm, L: 20 mm
  - Massa: 150g
  - Máxima temperatura de operação: -40 ºC a 60 ºC
  - Máxima temperatura de repouso: 0 ºC a 40 ºC

Do subtópico 6.2.2 ao 6.2.6 foram utilizadas as câmeras DFM72BUC02 e DFK31BF03, a partir do subtópico 6.2.7 foi utilizada a câmera Microsoft LifeCam HD-3000. Esta mudança foi realizada devido à maior capacidade de mobilidade e flexibilidade que este equipamento apresentou em relação à anteriores, os dados não foram impactados negativamente, uma vez que as mesmas especificações que impactariam nas capturas foram utilizadas, como resolução e taxa de amostragem.

6.2.2. ANÁLISE DO NÚMERO DE CÂMERAS

A primeira análise necessária para a escolha da quantidade de câmeras que serão utilizadas no projeto foi realizada.

A discussão abordou a utilização de uma única câmera e capturas de fotos sequenciais com a mesma, ou a utilização de duas câmeras e capturas simultâneas realizadas por elas.

A captura de foto única foi realizada à distância e ângulos iniciais conhecidos (distância de 595mm e ângulo entre eixo central da lente e nariz da pessoa de 0º - dimensões ótimas calculadas e apresentadas nos próximos subtópicos do relatório), posteriormente esta câmera foi movida de sua posição inicial e outra foto foi capturada desta nova posição, desta forma estas duas fotos foram analisadas e comparadas pelo algoritmo SIFT. Para o teste, foi utilizado o dispositivo DFM72BUC02 com saída USB, a seguir seguem fotos do procedimento para esta análise:

Figura 16: A e B – Primeira e segunda fotos capturadas com DFM72BUC02, respectivamente.

As duas imagens foram processadas pelo algoritmo SIFT de recurso online [10].

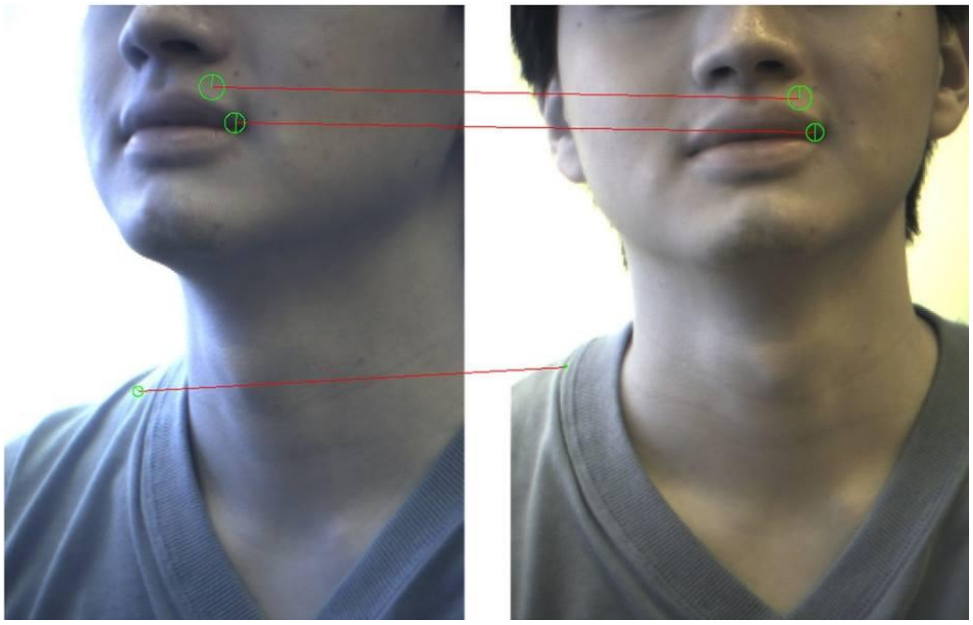


Figura 17: Algoritmo SIFT [8]

Como observado, foram encontrados pontos de interesse condizentes nas duas imagens.

O grupo também utilizou o software Visual Studio para processamento do algoritmo SURF [Apêndice A].

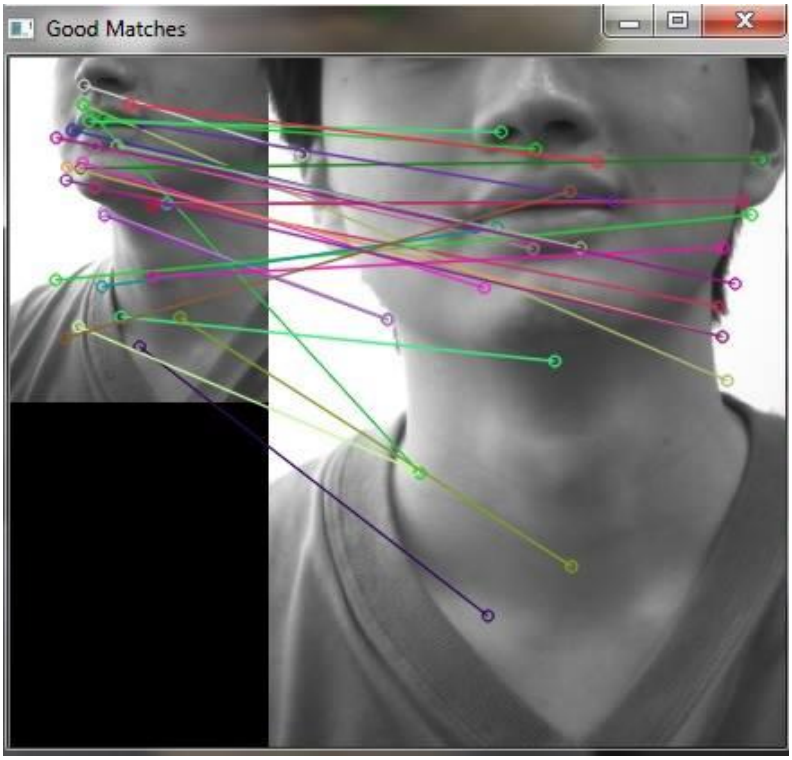


Figura 18: Algoritmo SURF [código fonte - Apêndice A]

O processamento das duas imagens gerou uma grande quantidade de pontos de interesse encontrados, além disso, grande parte destes pontos é condizente nas duas imagens.

Já na captura simultânea, as duas câmeras teriam posições definidas, e estas duas imagens capturadas seriam analisadas e comparadas novamente pelo SIFT. Para o teste, foram utilizados o dispositivo DFM72BUC02 com saída USB e o dispositivo DFK31BF03 com saída Firewire. Foram escolhidos dispositivos diferentes entre si, devido a limitações de hardware que o grupo apresentou no atual momento, no entanto as imagens das duas câmeras foram realizadas visando máxima correlação de regulagens, conseguindo um resultado satisfatório para análise proposta neste momento atual.

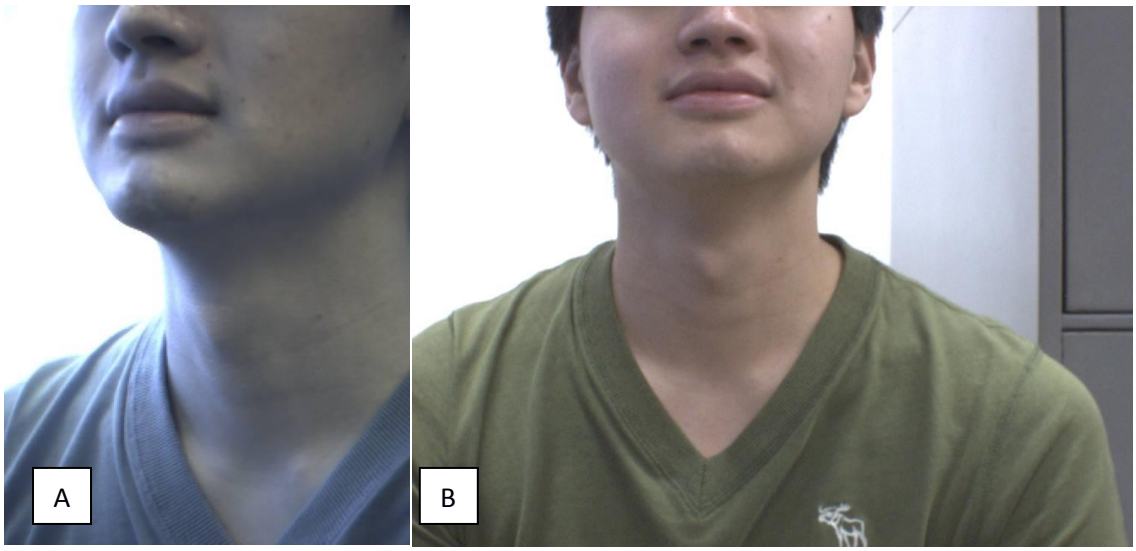


Figura 19: A e B - Primeira e segunda fotos capturadas com DFM72BUC02 e DFK31BF03, respectivamente

Figura 20: Algoritmo SIFT utilizado de [8]

Observa-se que foram encontrados pontos de interesse incondizentes no procedimento.

Processando as duas imagens com o algoritmo SURF [Apêndice A].

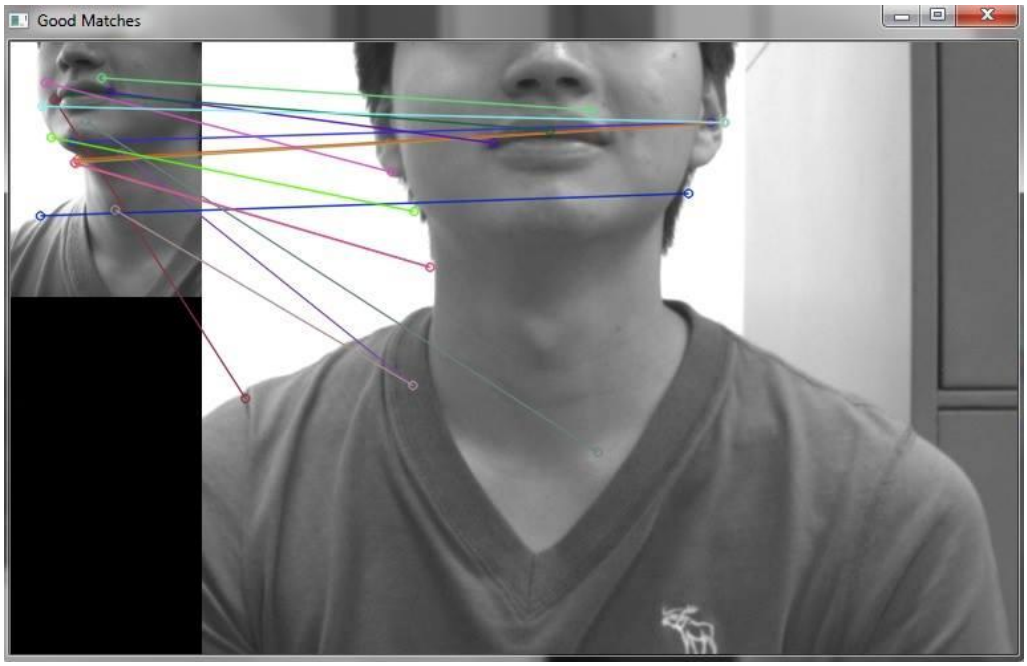


Figura 21: Algoritmo SURF utilizado de [11] - código fonte [Apêndice A]

O processamento do algoritmo SURF resultou em localização de poucos pontos de interesse, além disso, os pontos encontrados não são condizentes entre as imagens capturadas.

6.2.3. ANÁLISE DE POSICIONAMENTO – DISTÂNCIA

A obtenção de um enquadramento ideal para o posicionamento das câmeras necessita de análise em diferentes distâncias. Para o procedimento foram realizadas algumas capturas em distâncias conhecidas e a partir das dimensões do rosto foi possível determinar a distância ideal do mesmo para a câmera.

Área de trabalho do rosto:	Abertura máx. mandibular:	138 mm (nariz ao queixo)
	Distância entre lados:	125 mm

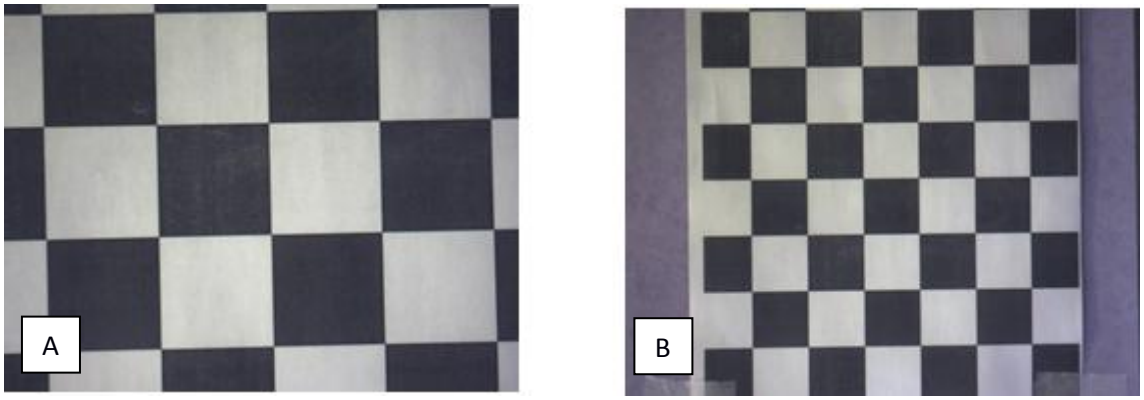


Figura 22: A/B: Capturas de padrão quadriculado realizadas a distância de 50 cm e 1 m:

Para as análises com a abertura de mandíbula total, fez-se uma medida de proporção linear da dimensão dos quadrados em relação à linha ocupada na figura.

As medições realizadas em A e B resultaram em uma dimensão mínima de trabalho (na resolução de 768 pixels de altura) de 105 milímetros em A (50 cm de distância) e de 207 milímetros em B.

Para que a distância entre os lados do rosto (125 mm) seja suficiente para a captura de toda a superfície necessária, um posicionamento da câmera a uma distância aproximada de 595 mm.

6.2.4. ANÁLISE DE POSICIONAMENTO – ÂNGULO

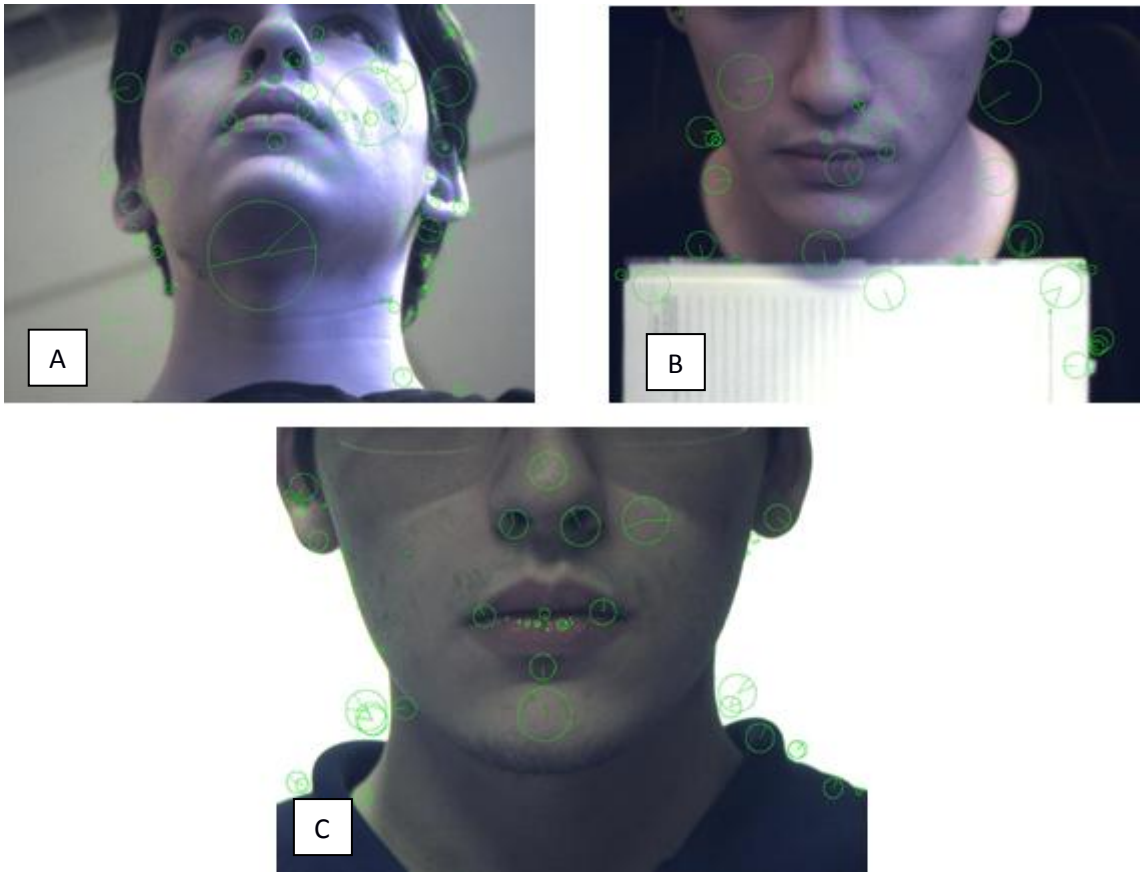


Figura 23: A/B/C: Análises SIFT tiradas de [8] após tomadas em ângulo inferior (A), superior (B) e frontal (C) ao rosto

Para a análise entre possíveis variações angulares, foram feitas diferentes tomadas com posicionamentos da câmera em visão superior, inferior e frontal à face. Os resultados são mostrados nas figuras 22 A/B/C.

É possível notar que a figura de visão frontal foi a que melhor apresentou as características faciais através do modelo de [8] recorrentes da literatura odontológica, enquanto que a localização das mesmas não obteve o mesmo sucesso nas demais posições de ângulo.

6.2.5. ANÁLISE DE VELOCIDADE DE CAPTURA

Para a análise de captura cinemática, um movimento teste de abertura da mandíbula foi gravado a 25 FPS, marcando dois dos pontos relevantes encontrados no tópico 6.2.3 para a medição manual entre cada tomada. Para o teste entre taxas de captura foi retirada uma amostra de 1 em cada 5 para a reconstrução da captura em 5 FPS. Cada quadro da animação (foi separado e o posicionamento das marcações foi plotado em um gráfico (figura 24).

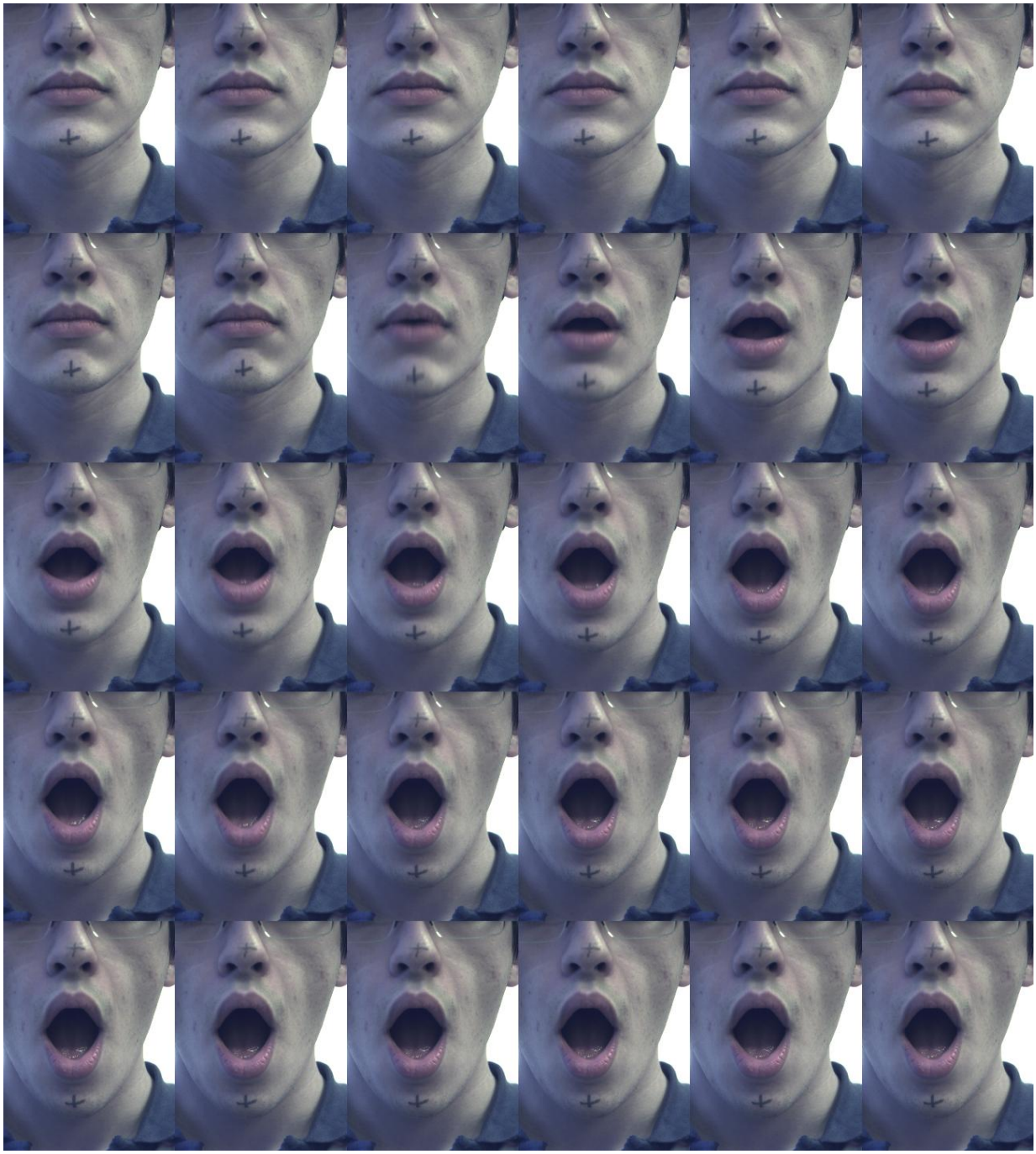


Figura 24: quadros capturados do movimento de abertura mandibular

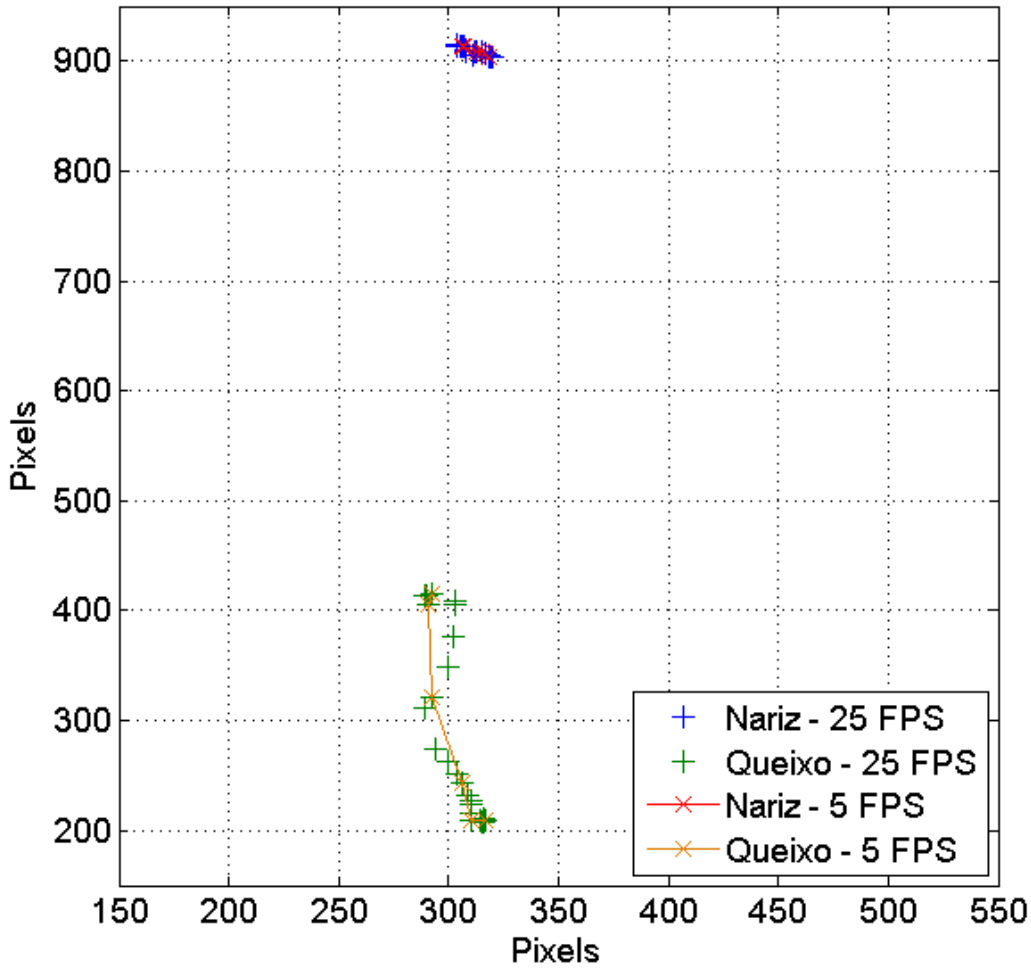


Figura 25: Trajetória de pontos capturados da abertura da mandíbula

Para a construção do gráfico, o grupo coletou as posições dos pontos do nariz e do queixo manualmente através do software GIMP, utilizado frequentemente para manipulação de imagem, a seguir foi construído o gráfico através do software Microsoft Excel 2010.

A análise do gráfico permite constatar a necessidade de maiores taxas de captura, considerando que o movimento do elemento queixo durante a gravação apresentou mais variações importantes no percurso capturado a 40 milissegundos (25 FPS) do que as capturas a cada 200 milissegundos (5 FPS).

6.2.6 ANÁLISE DE VARIAÇÃO NA COR DAS IMAGENS DE CAPTURA

Foi realizado o processamento de imagens pelo código [Apêndice A], diferenciadas unicamente pela cor (uma delas apresentando apenas coloração em tons de cinza [Figura 25] e outra com coloração da base RGB [Figura 26]), capturadas através do código [Apêndice A].

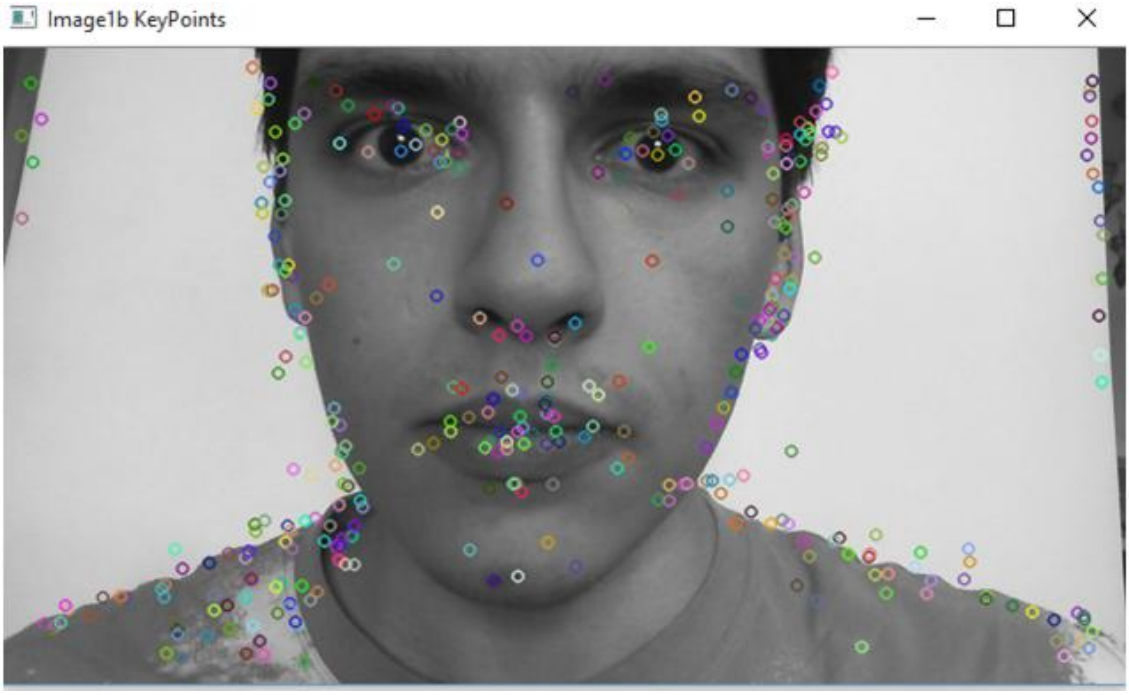


Figura 26: Imagem em escala de cinza com os pontos de interesse

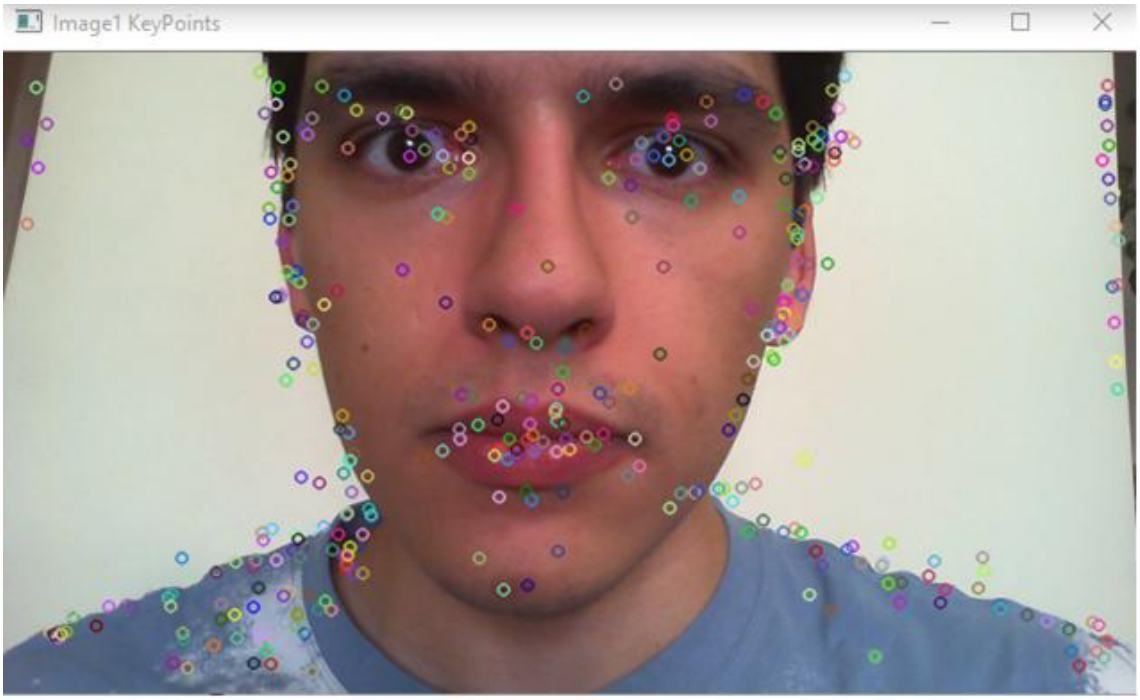


Figura 27: Imagem em escala RGB com pontos de interesse

Observando as duas imagens, os pontos de interesse se mantiveram. Desta forma, em relação a pontos de interesse obtidos, as duas formas de captura não se diferenciam, no entanto o SURF faz uso da matriz hessiana que se baseia em valores de luminosidade para detecção de bordas e cantos, assim a imagem colorida que grava informações em três canais (R – vermelho, G – Verde, B – Azul) ocupa mais memória do que a imagem em escala de cinza que só tem variância de luminosidade (sem variância de matiz e saturação).

6.2.7 ANÁLISE DE VARIAÇÃO DE RESOLUÇÃO DAS IMAGENS DE CAPTURA

Foi realizado o processamento de duas imagens pelo código [Apêndice A] diferenciadas apenas pela resolução.
A primeira análise se deu entre a imagem de resolução original, ou seja, 100% de resolução e outra imagem com 30% da resolução normal.

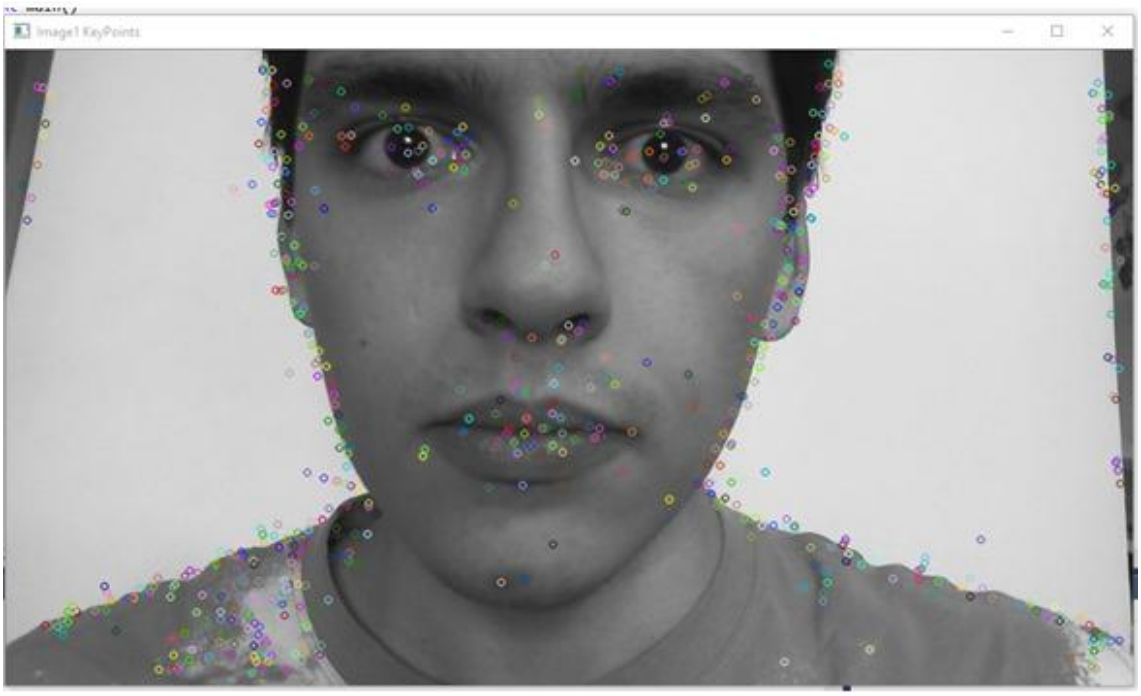


Figura 28:Imagem com resolução original (100%)

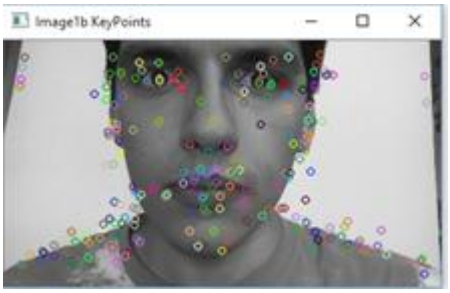


Figura 29:Imagem com resolução de 30% da original

A partir das duas imagens processadas, é possível notar a queda da quantidade de pontos de interesse com a diminuição da resolução da imagem. Desta forma, percebe-se a perda de informação na imagem de menor resolução.

Uma segunda análise foi realizada com imagem maior do que a original (130%).

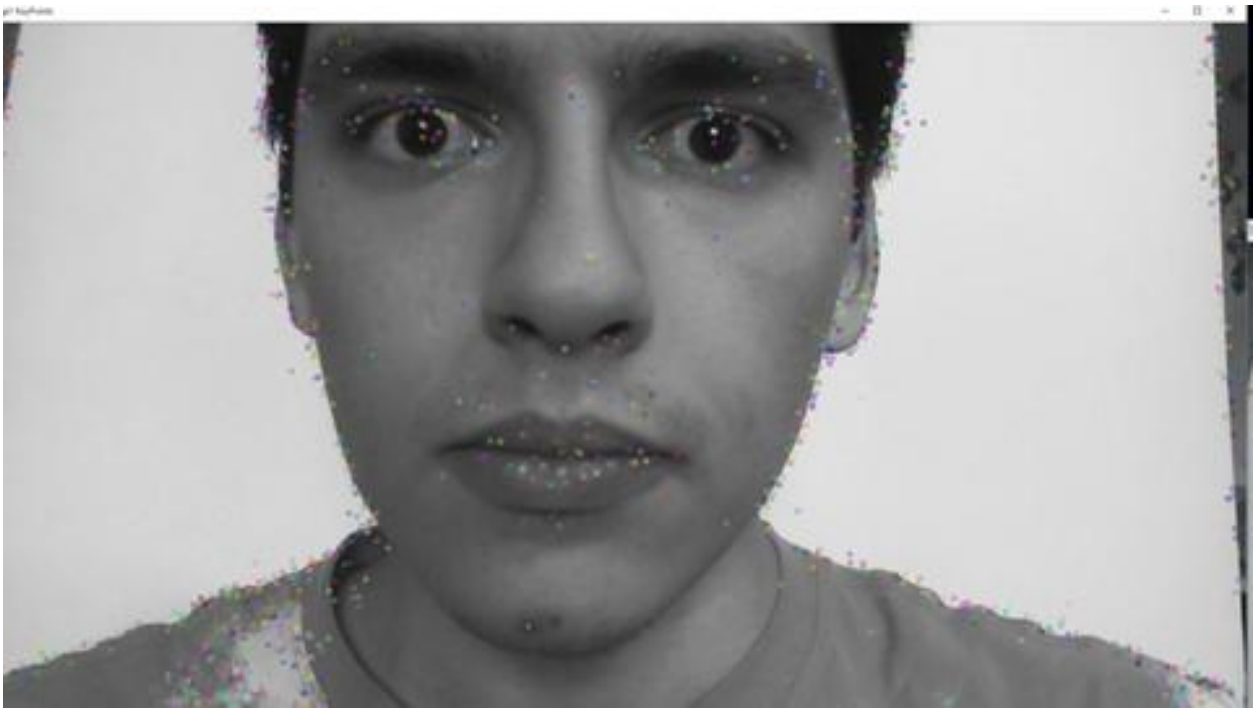


Figura 30:Imagem com resolução de 130% da original

Em relação à quantidade de pontos de interesse, a imagem com resolução de 130% da original e a imagem original apresentaram variações pouco significativas, no entanto é possível observar certa distorção na imagem de resolução maior.

6.2.8 ANÁLISE DE AMOSTRAGEM DE PONTOS DETECTADOS

Como citado anteriormente, o método SURF permite a extração de pontos de interesse em uma imagem. Este método inclui uma comparação entre pixels em determinada região da imagem, através do cálculo da matriz hessiana de cada ponto da imagem. Os pontos detectados podem ser limitados por valores no cálculo da matriz, fazendo com que escolhas de valores baixos ocasionem na maior amostragem de pontos.
Para o aprimoramento de captura de pontos de interesse, a escolha de altos valores de hessiana permite filtragem de pontos de alto contraste.

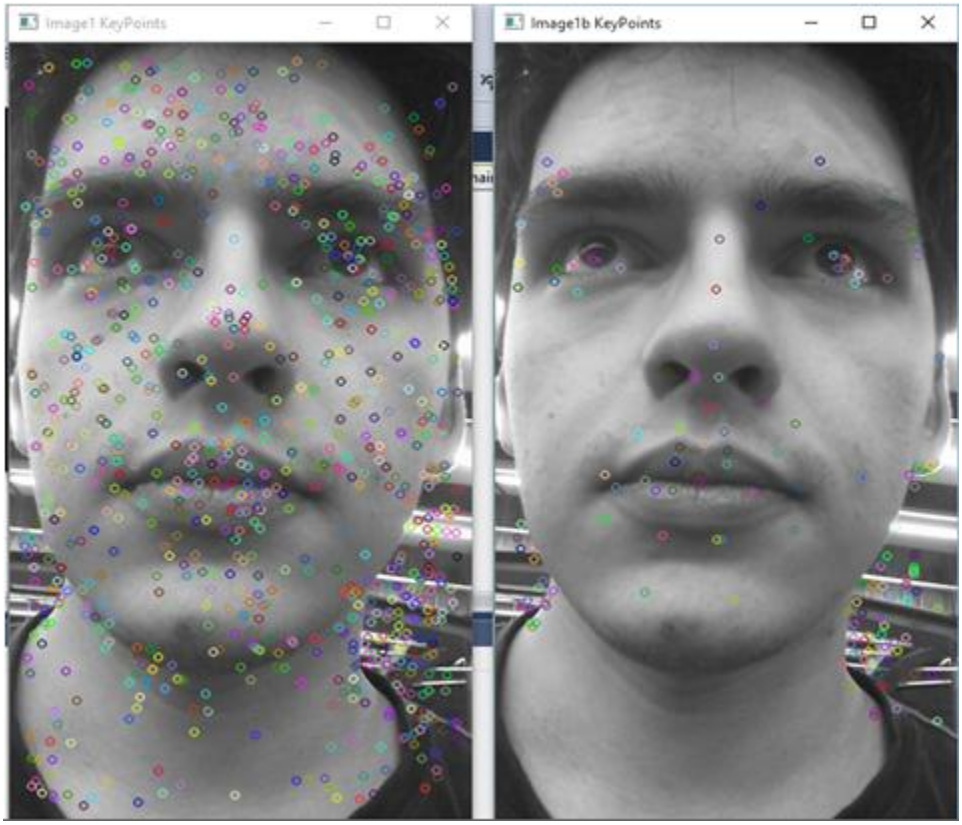


Figura 31:Pontos de interesse obtidos com valor de hessiana de 50 (imagem à esquerda) e 200 (imagem à direita)

6.2.9 ANÁLISE DE CAPTURA DE IMAGENS COM RUÍDOS

Ruídos do *background* da imagem capturada podem influenciar resultados de processamento de imagens pelo código [Apêndice A], desta forma foram realizadas análises para decisão do background escolhido na captura das imagens.

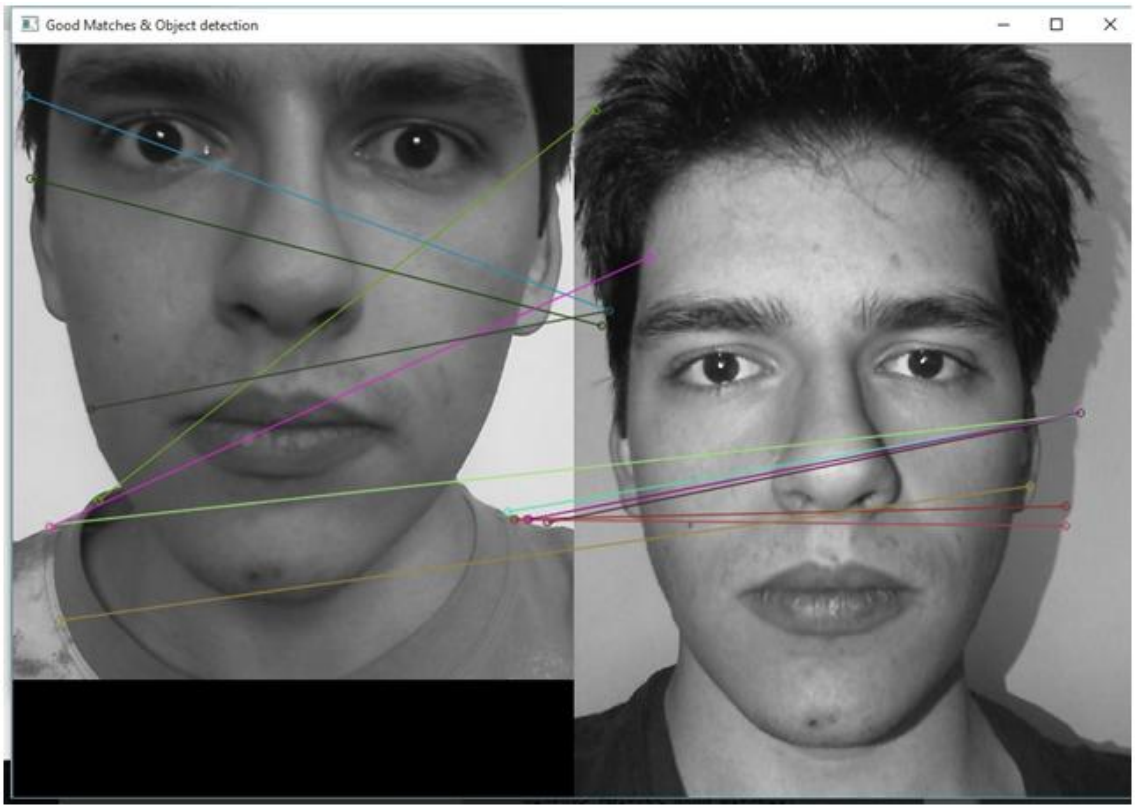


Figura 32:Imagens com background de cor clara

A escolha de um plano de fundo de coloração clara criou grandes contrastes na região de contorno da imagem, gerando desta forma ruídos que dificultaram a obtenção de pontos de interesse consistentes no ‘‘interior’’ da imagem, ou seja, na região do rosto.

Figura 33:Imagens com background de cor escura

Com um plano de fundo de coloração escura, os pontos de maior contraste encontram-se no rosto da pessoa, desta forma os pontos de interesses são obtidos do rosto.

6.2.10 ANÁLISE DE CAPTURA DE IMAGENS RECORTADAS

A seleção parcial de uma imagem para comparação com outra imagem original (não parcial) foi realizada com o objetivo de diminuir ruídos.

Para esta análise, foi recortada parte da imagem que o grupo definiu como importante ao projeto, ou seja, a região composta pelo sistema mandibular e nariz, esta região foi comparada com a imagem original a partir da detecção de pontos de interesse pelo código [Apêndice A]

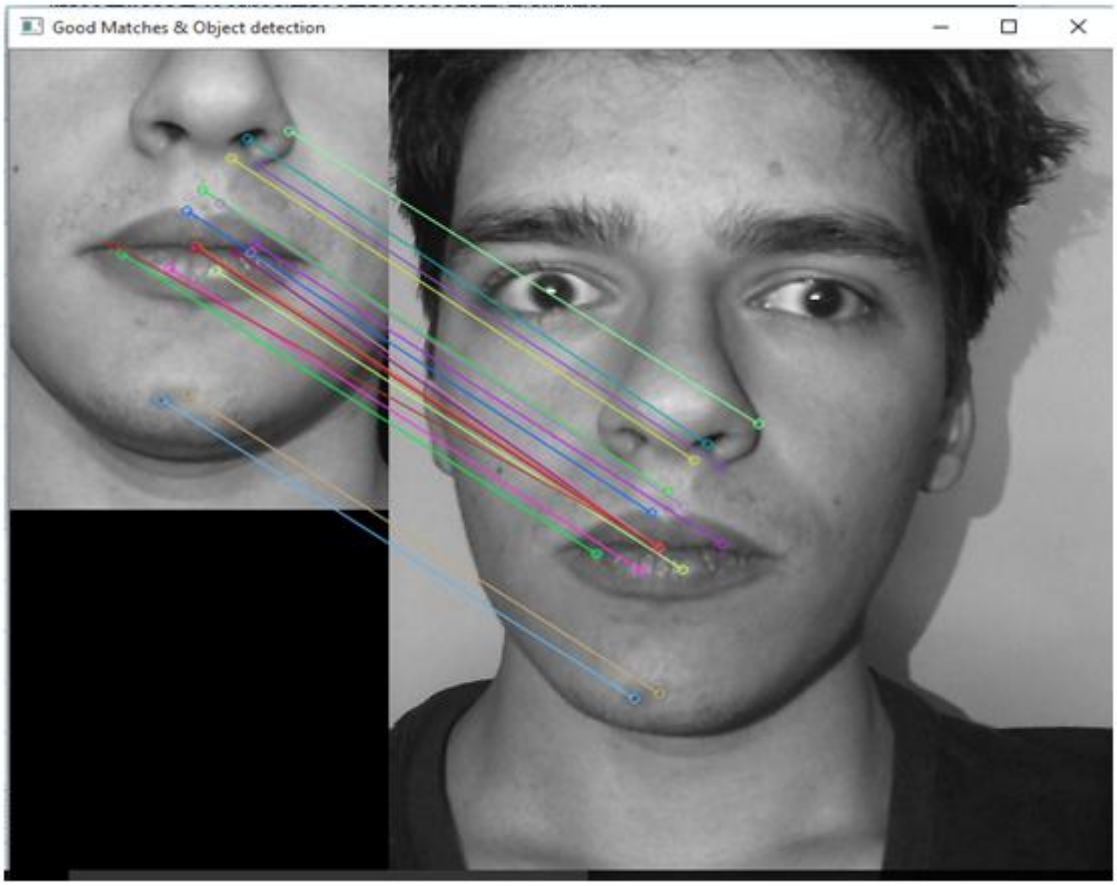


Figura 34:Match de pontos de interesse de imagem recortada e original

6.2.11 ANÁLISE DE PROCESSAMENTO *REAL TIME* OU DE VÍDEO

Processando o algoritmo SURF com captura *real time*, ou seja, através de captura de vídeo ao vivo, foi observada uma demora muito grande para a obtenção de pontos de interesse e cálculo dos seus descritores. Além da demora, após um curto espaço de tempo (aproximadamente 30 segundos) o código parava de funcionar, acusando erro por falta de memória de processamento.

Um segundo processo foi realizado, um vídeo capturando a movimentação mandibular foi gravado e salvo na rede do computador, posteriormente o algoritmo SURF foi processado sobre este vídeo, sendo possível observar o processamento de vídeos capturados a taxas de amostragem de 30 FPS sem estouro ou insuficiência de memória.

6.2.12 ANÁLISE DE PONTOS PELO SURF

Para verificar a consistência de obtenção de pontos de interesse em um vídeo, foi realizada a comparação do primeiro frame do vídeo com o vídeo original. Esta comparação se baseou na detecção de pontos de interesse semelhantes na imagem estática (primeiro frame) com os demais frames do vídeo. A detecção de pontos de interesse foi delimitada à regiões interessantes ao estudo do grupo (ponta do nariz e centro do queixo) através de adesivos de coloração preta e branca que garantiam grande contraste e assim pontos de interesse.

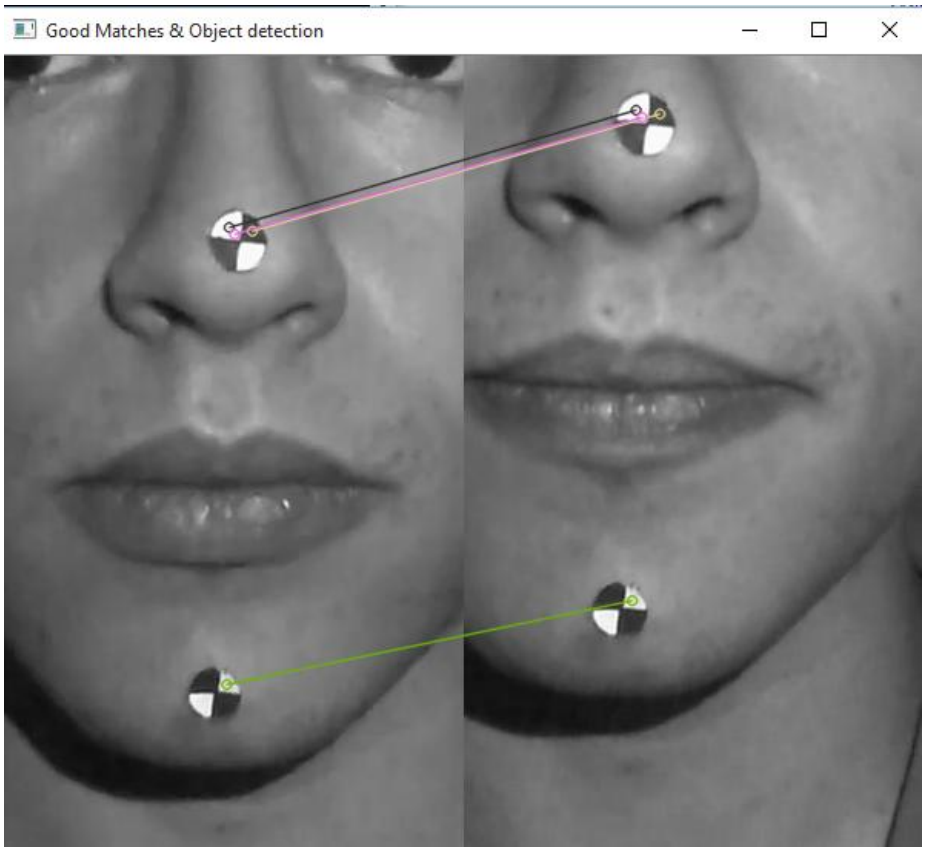


Figura 35: Relação entre pontos de interesse do primeiro frame do vídeo (imagem à esquerda) e frame N do vídeo (imagem à direita)

6.2.13 RASTREAMENTO DE PONTOS PELO SURF

A partir das coordenadas dos pontos de interesse encontrados na imagem estática (primeiro frame) e no vídeo, foi possível representar graficamente a movimentação da ponta do nariz e do centro do queixo em coordenada universal, e também em coordenadas relativas, com a ponta do nariz como ponto de origem.

Foi utilizada uma função para desenho da curva no gráfico, para poder localizar o movimento de subida e descida, foi-se variando a tonalidade da cor da curva.

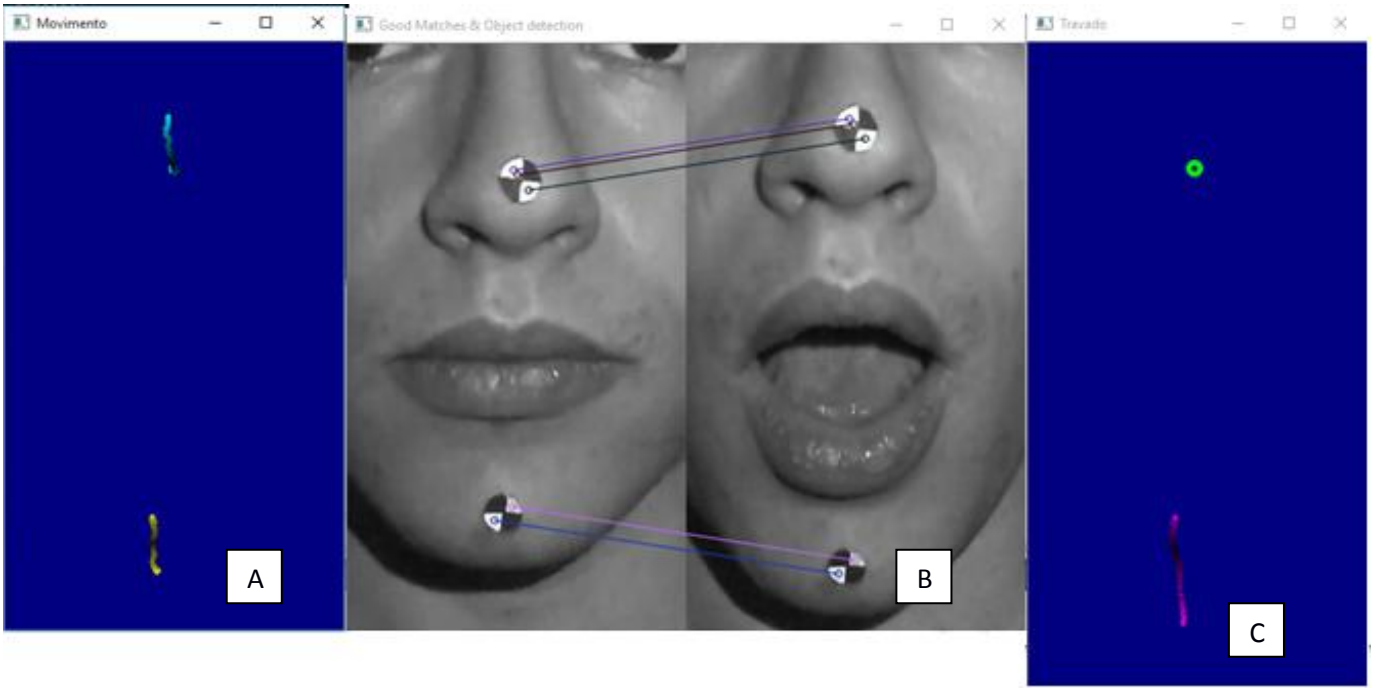


Figura 36: (A) Trajetória em coordenada universal, (B) Relação de pontos de interesse de imagem e vídeo, (C) Trajetória em coordenada relativa com origem na ponta do nariz

6.2.14 DESENVOLVIMENTO DO PROCESSO DE FORMATAÇÃO DE IMAGEM

Foi criado um processo para escolher a área da imagem a ser processada, facilitando desta forma, a localização de pontos de interesse que serão obtidos em todos os frames do vídeo.
O processo consiste na escolha de uma área que será considerada para análise e a exclusão de outros pontos que não pertencem a essa área, como pode se observar a seguir:
A função desenvolvida reconhece a posição de lados do retângulo desenhado, após isso apaga todos os pontos representados pelo lado externo do retângulo.

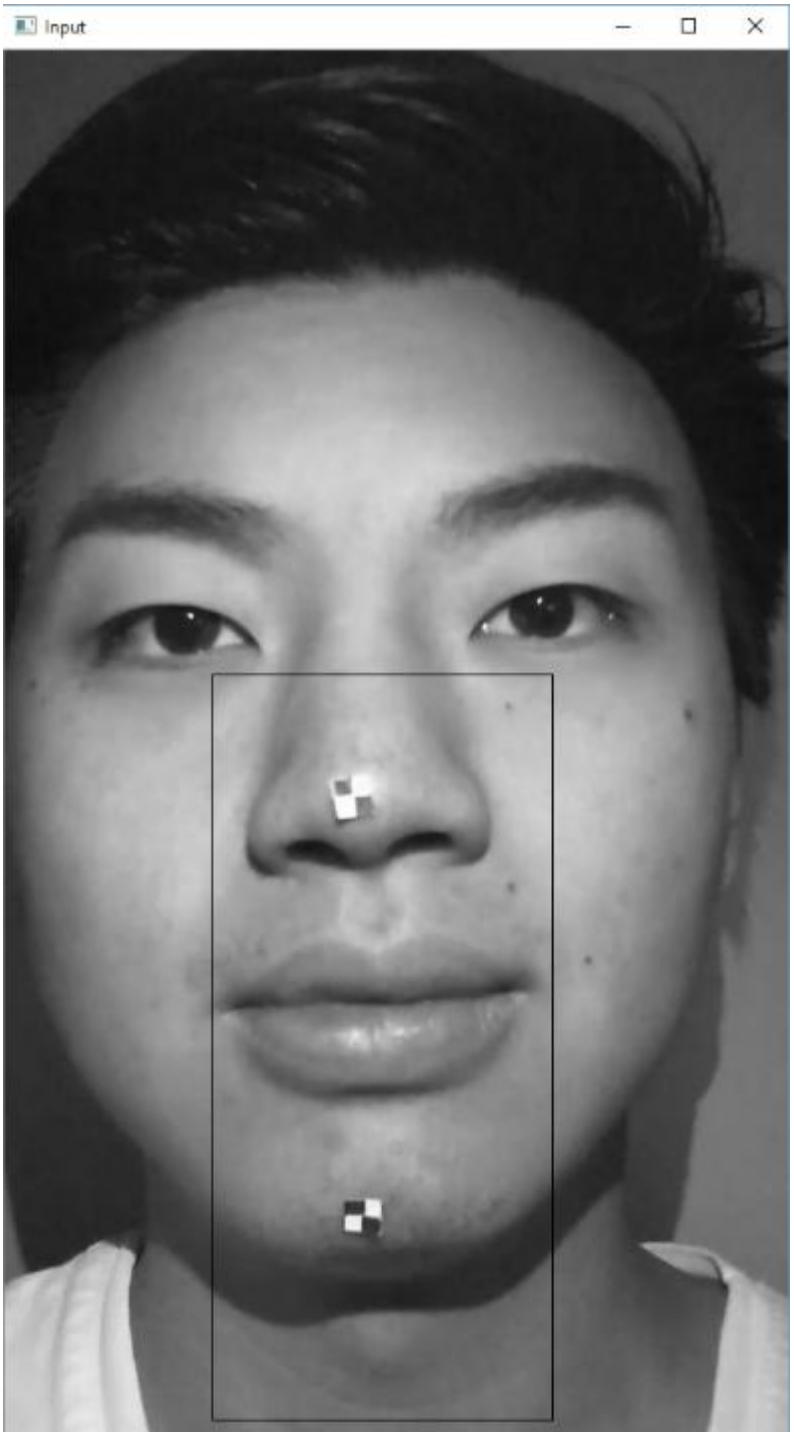


Figura 37: Foto original e área selecionada para análise

Após a seleção, apenas a área escolhida é considerada:

Figura 38: Área da imagem selecionada e posteriormente processada

6.2.15 DESENVOLVIMENTO DE ESCALA GRÁFICA

Para obter uma noção de posicionamento e dimensão no rastreamento dos pontos do subtópico 6.2.13, foi realizado o desenvolvimento da escala gráfica. O procedimento foi utilizar adesivos de coloração preta e branca de dimensões definidas (quadrados com quadrados de 0,6 cm), e a obtenção da dimensão destes adesivos no vídeo, através do uso de análise de geometria analítica para cálculo de distância entre pontos, foi realizada uma comparação da dimensão real dos adesivos com a dimensão apresentada no vídeo capturado.

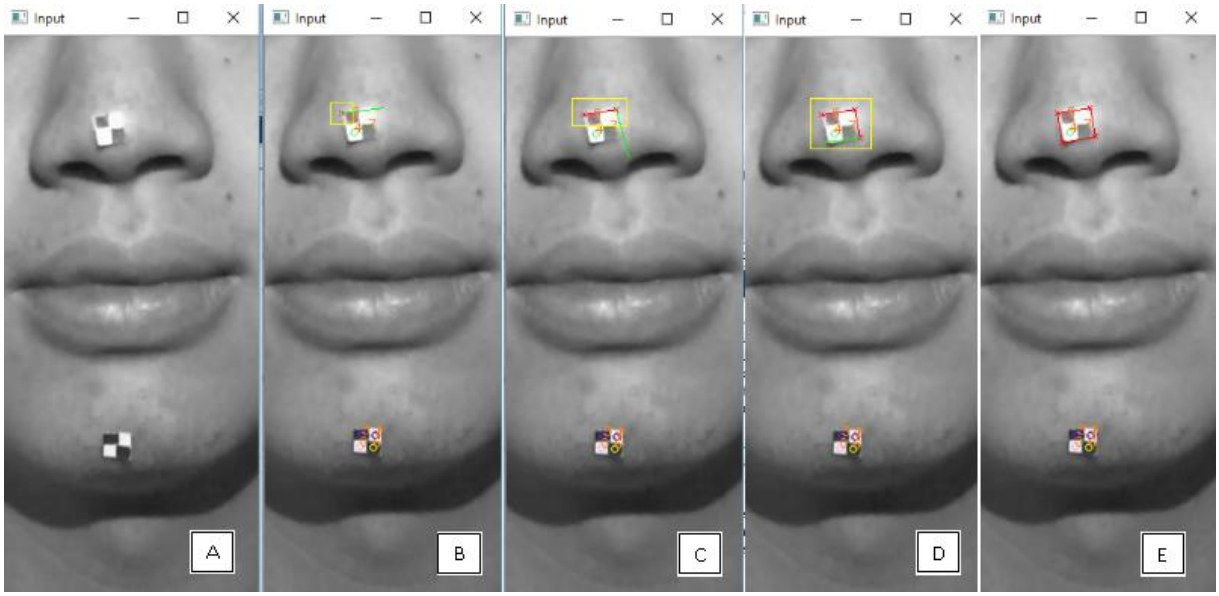


Figura 39: Processo para seleção da dimensão do adesivo na imagem

Com a comparação, foi obtida a dimensão da escala, cada quadrado do gráfico representa 1 cm da vida real [Figura 37].

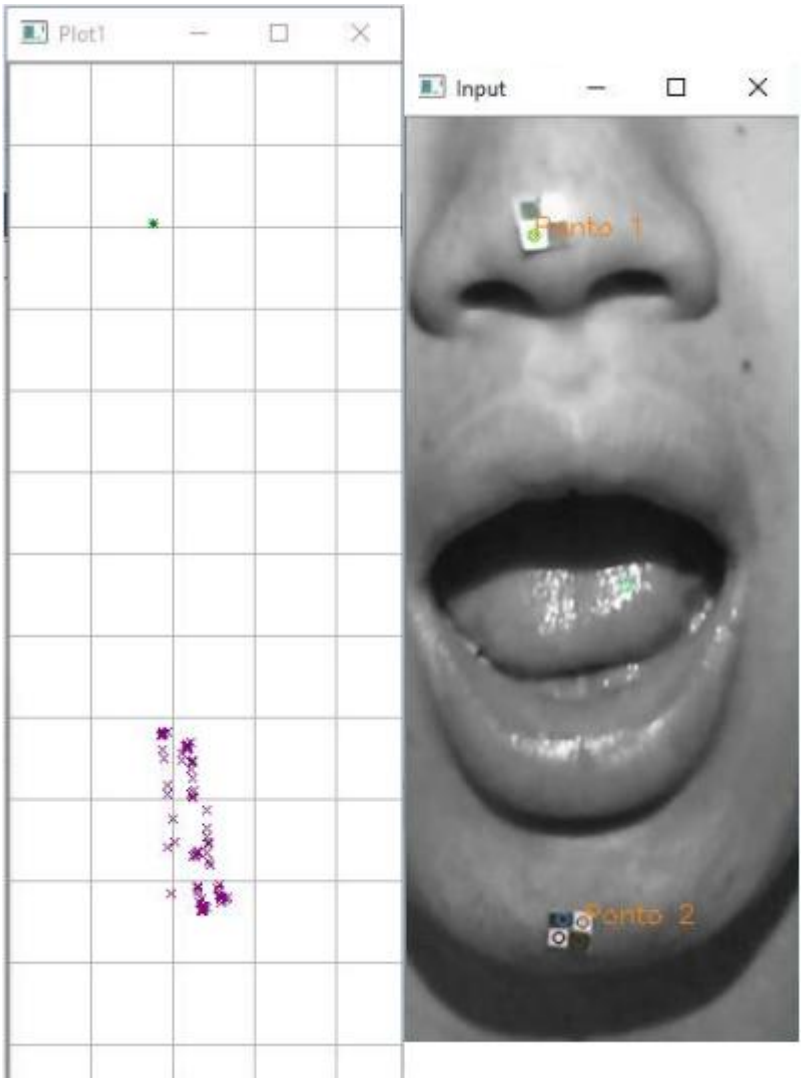


Figura 40: Rastreamento de pontos com escala real de 1 cm

6.2.16 DESENVOLVIMENTO DE GRÁFICOS DOS EIXOS X E Y PELO TEMPO

A partir da coordenada dos pontos no interior da área delimitada no processo do subtópico 6.2.15, foram desenvolvidos gráficos de posição em relação ao tempo, a medida de tempo na escala do gráfico foi realizada dessa forma: sabendo que o vídeo apresenta 24 frames, a cada 24 frames do vídeo que se passam, 24 pontos são desenhados no gráfico (1 ponto por frame), quando o 24º ponto é desenhado é traçada uma reta, esta reta delimita a escala do gráfico, desta forma, a distância entre duas retas representa 24 frames ou 1 segundo.

A seguir o gráfico da posição sob o eixo X em relação ao tempo:

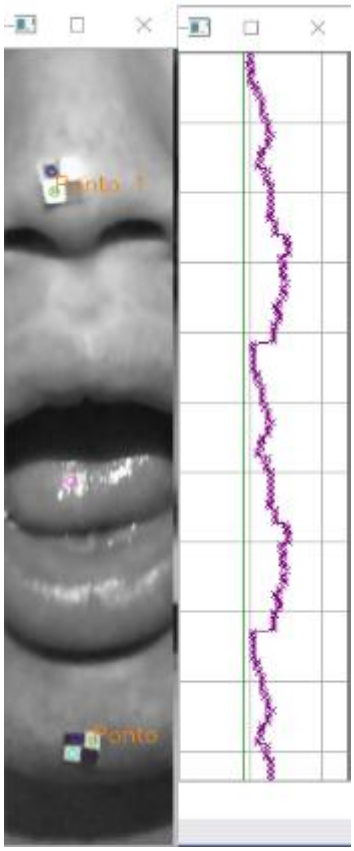


Figura 41: Gráfico de posição X em relação ao tempo

A linha vertical verde representa aposição do ponto de interesse do nariz em relação ao tempo, já as linhas de cor roxas representam as posições do ponto de interesse do queixo em função do tempo.

A seguir o gráfico da posição Y em função do tempo:

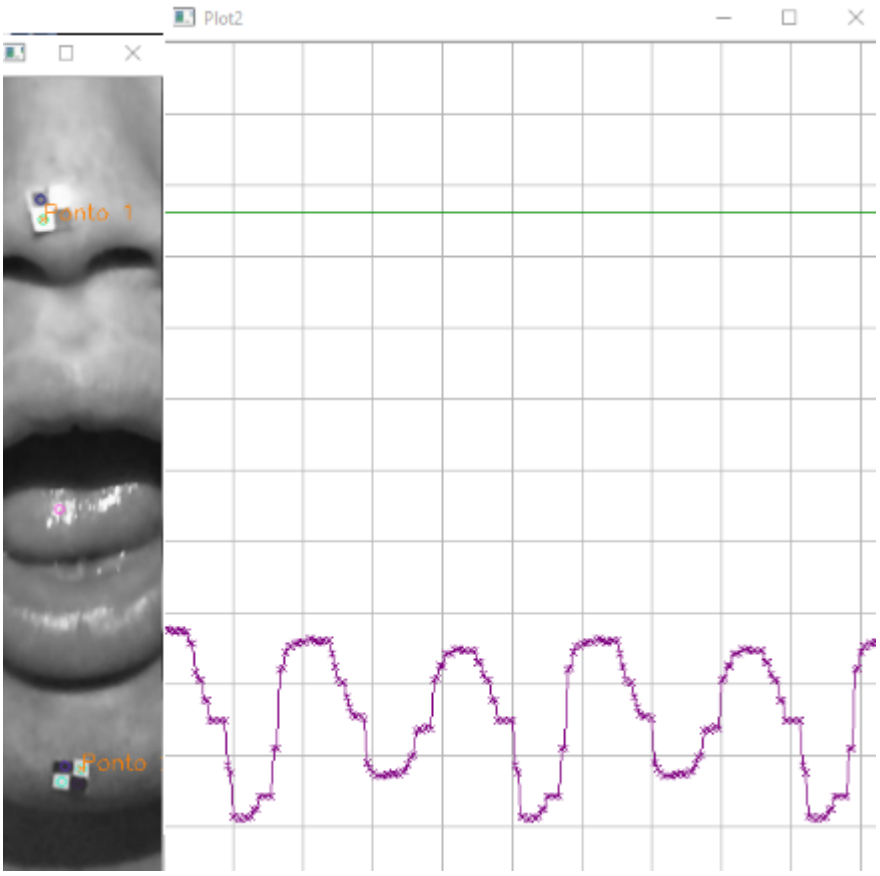


Figura 42: Gráfico da posição Y em relação ao tempo

A linha verde representa a posição do nariz em relação ao tempo e a roxa, por sua vez, representa o movimento em eixo Y do queixo em relação ao tempo.

6.2.17 HOMOGRAFIA

Realizando testes com um mecanismo de dimensão definida [Apêndice B], o grupo observou que em determinados frames, pontos de interesse não eram localizados no interior da área delimitada no subtópico 6.2.15, e, além disso, como esta área era fixa, ou seja, baseada apenas na escolha desta área no primeiro frame, nos frames seguintes o adesivo tinha se deslocado na imagem e o processo perdia referência.

Para resolver este problema, o grupo desenvolveu um processo na qual pontos que formam um polígono são escolhidos pelo usuário antes do processamento das imagens. O polígono foi formado com a ligação entre os pontos “clicados” na imagem.

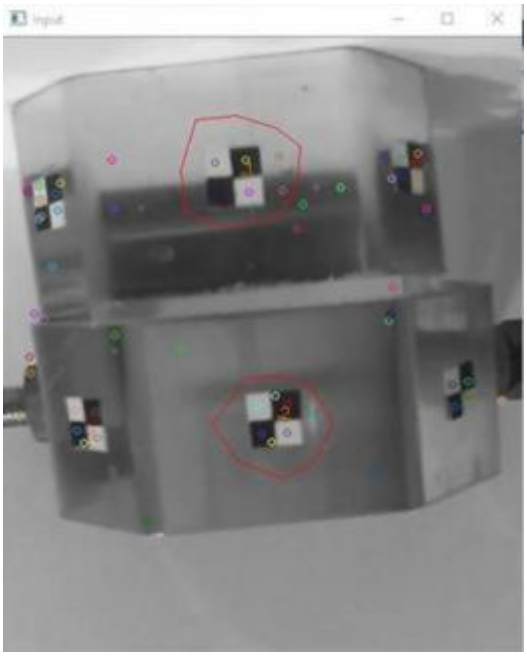


Figura 43: Polígonos (vermelho) formados por pontos escolhidos pelo usuário

A escolha destas regiões formadas pelos polígonos delimitam regiões nas quais serão analisados os pontos de interesse. Estes pontos de interesse são utilizados em uma função de homografia, ou seja, a localização dos pontos de interesse no interior de cada área dos polígonos é computada e processada para que o polígono seja desenhado em cada frame de acordo com a posição dos pontos de interesse atuais. Desta forma, o problema de perda de referência citado anteriormente foi resolvido, uma vez que a área de processamento dos pontos de interesse varia de acordo com a movimentação destes.

6.2.18 GRÁFICOS COM HOMOGRAFIA

Os resultados a seguir foram utilizados com o uso da função de homografia, as curvas dos gráficos foram obtidas a partir da posição da média dos vértices do polígono escolhido.

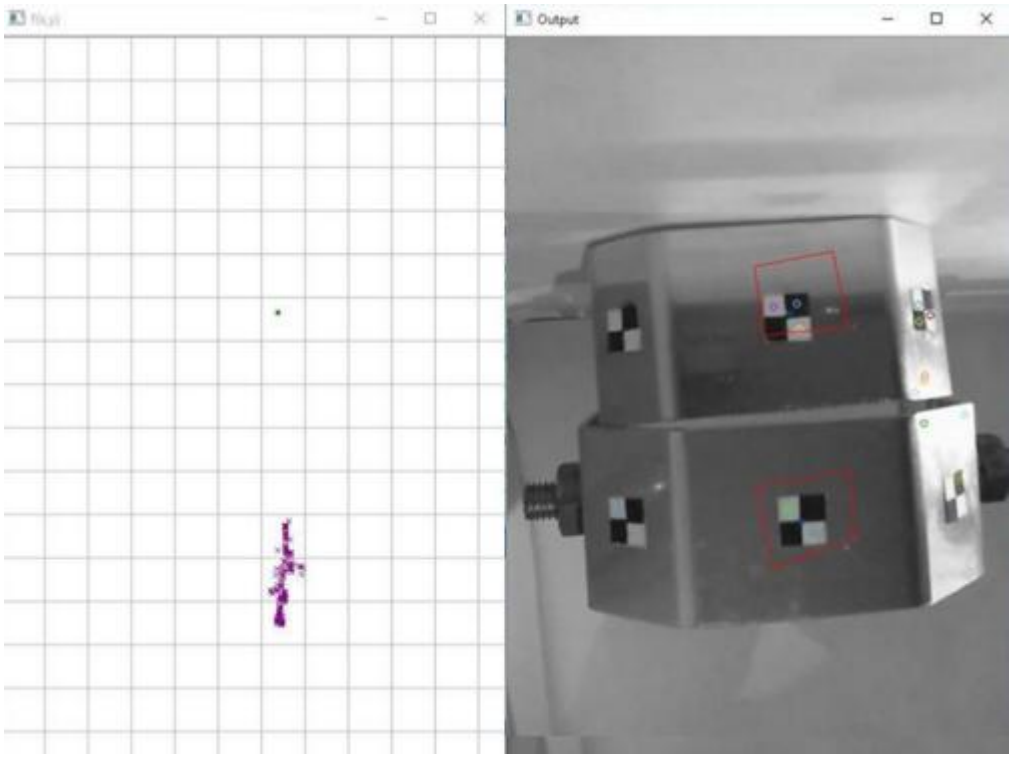


Figura 44:Rastreamento dos pontos com homografia

A seguir, o gráfico do eixo Y em relação ao tempo:

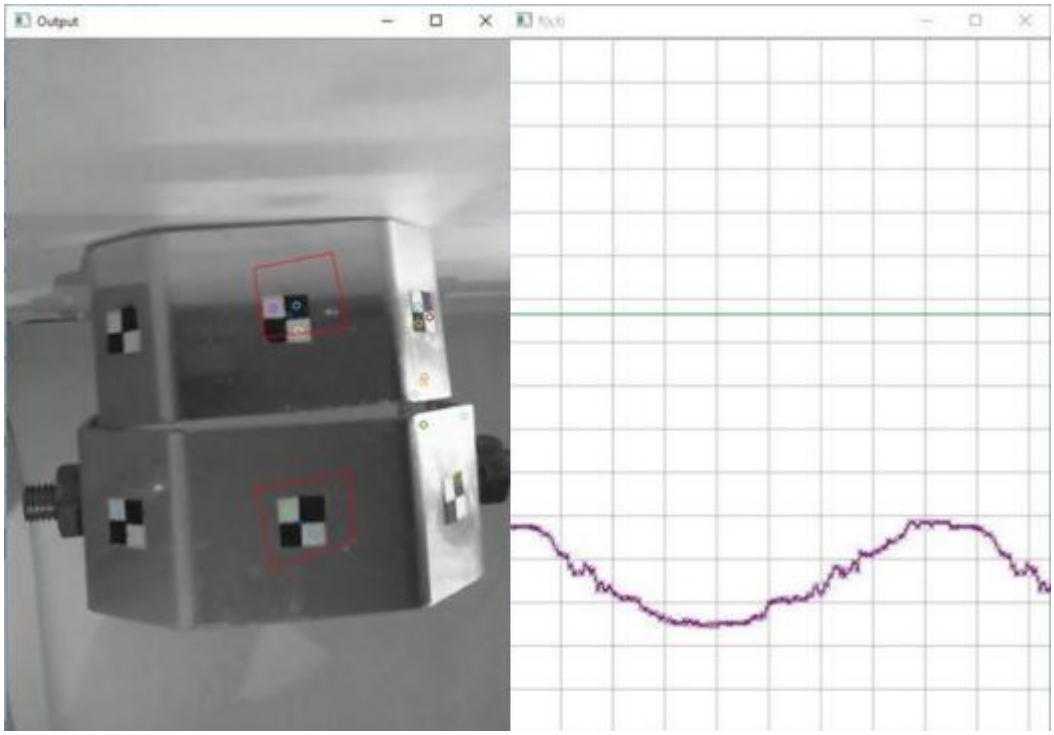


Figura 45:Gráfico da posição em Y em relação ao tempo com homografia

Abaixo, o gráfico da posição em X em relação ao tempo:

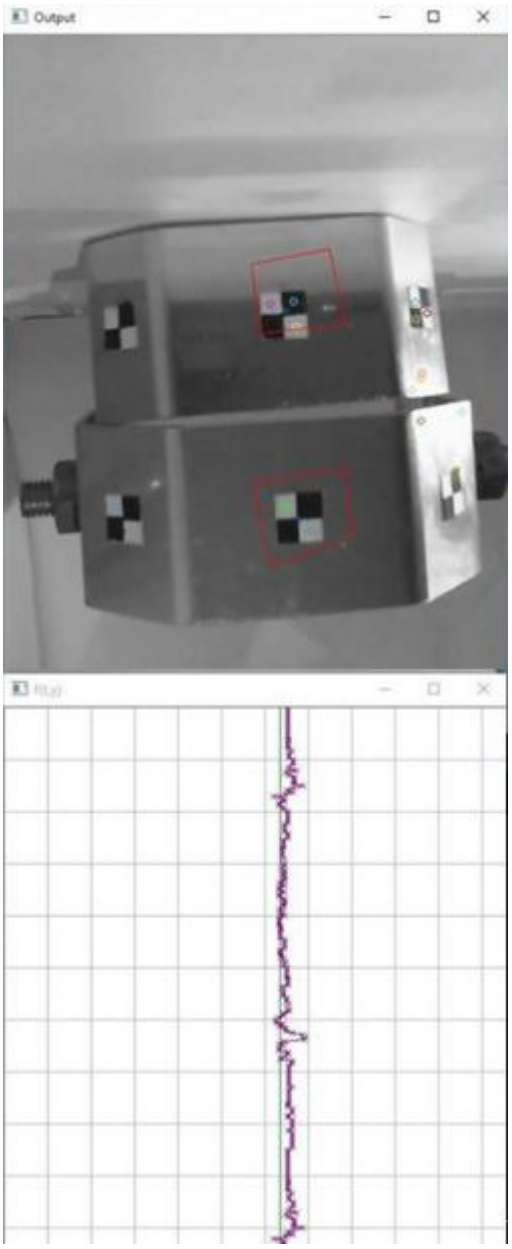


Figura 46:Gráfico da posição em X em relação ao tempo com homografia

6.2.19 VERIFICAÇÃO DE PRECISÃO

O grupo realizou um procedimento para verificar a precisão do rastreamento de pontos, para isso foi realizada a medição da trajetória real do mecanismo de testes e após isso, uma comparação com a distância da trajetória capturada pelo experimento.

Com a ajuda de transferidor e paquímetro, o grupo realizou a medição do movimento vertical do adesivo acoplado à parte móvel do mecanismo de teste, o movimento de rotação foi de 45°

no sentido de abertura, como resultado, o grupo mediu uma variação na posição vertical de 2,62 cm.

Analisando a trajetória do experimento plotada no seguinte gráfico:

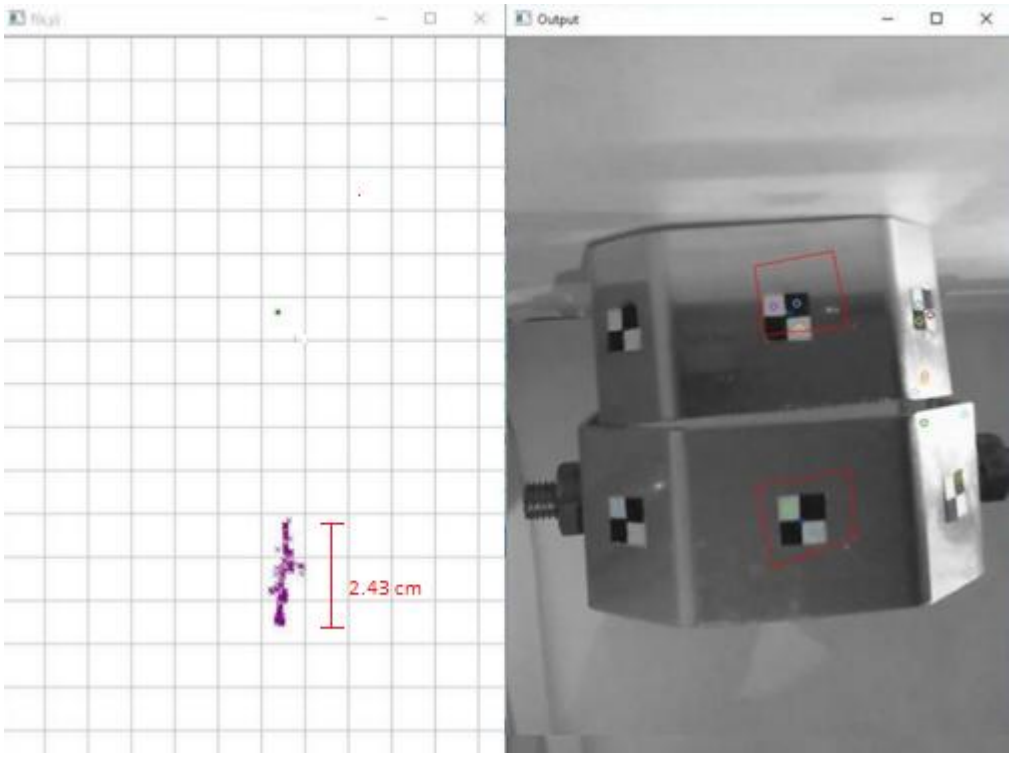


Figura 47: Rastreamento dos pontos com homografia

Utilizando o software de processamento de imagens GIMP, o grupo visualizou que um quadrado do gráfico (1 cm reais) representava 23 pixels, já a trajetória representava 56 pixels, desta forma, realizando uma relação de semelhança, foi obtido que a trajetória representou 2,43 cm na vida real.

6.2.20 RECONSTRUÇÃO DE IMAGENS EM PERSPECTIVA

A fim de possibilitar o trabalho e análise das imagens capturadas com o modelo real, foi proposto um modelo de reconstrução tridimensional dos movimentos.

O método empregado para os testes seguiu a marcação de limites dos adesivos quadriculados. As áreas delimitadas são processadas, uma a uma, utilizando funções presentes no OpenCV para determinação de pose tridimensional a partir de uma captura de imagem 2D. Obtém-se dessa forma os vetores de translação $\boldsymbol{t}$ e matrizes de rotação de Rodrigues $\boldsymbol{R}$ respectivos a cada região de adesivo.

O distanciamento dos adesivos em relação à câmera (*Z_est*) pode ser estimado por semelhança de triângulos a partir do tamanho aparente de cada adesivo. Portanto cada canto de adesivo com coordenadas homogêneas (u, v, 1) terá coordenadas tridimensionais

(X, Y, Z) = R^-1 * (C^-1 * s * (u, v, 1) – t)

onde C é a matriz da câmera contendo a distância focal e o centro da imagem em coordenadas homogêneas. A constante s pode ser calculada para a terceira linha da equação matricial com Z igual a Z_est.

Para a visualização da projeção em perspectiva dos pontos, fez-se uso de uma matriz de rotação no eixo Y em 45 graus:

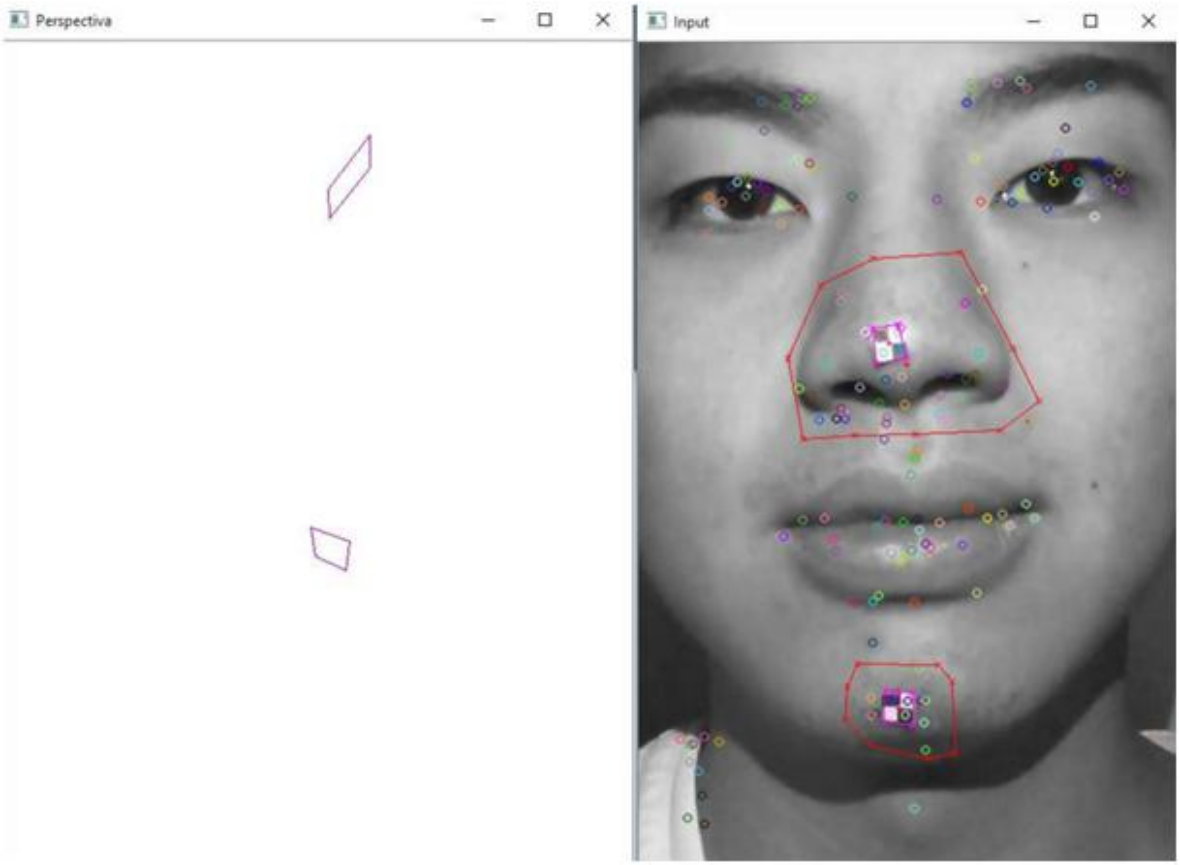


Figura 48: Perspectiva 3D

6.3.1 RECURSOS UTILIZADOS PARA ANÁLISES

O grupo utilizou diversos recursos de software e disponibilizados durante o projeto, a seguir a explicação e uso destes.

6.3.2 SOFTWARE INSTALADO

- MATLAB

Fez-se utilização da Camera Calibration Toolbox [12] com o uso do software Matlab, código disponibilizado pelo Instituo de Tecnologia da Califórnia – Caltech para apoio na execução e desenvolvimento de análises e código.

O processamento do código gera informações sobre a calibração de propriedades de imagens como distorção e distância focal da câmera, além disso, realiza a extração de pontos de interesse e seus descritores para em fim, realizar uma projeção 3D da relação entre imagens e a câmera de captura.

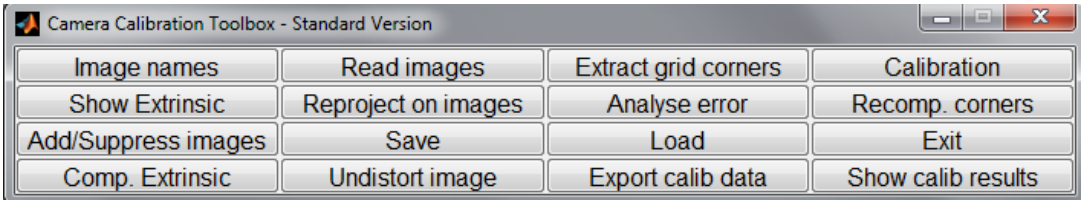


Figura 49: Menu de opções de processos do Camera Calibration Toolbox [12]

Foi feita a captura de dez fotos de uma imagem de um tabuleiro de xadrez de dimensões definidas – 3 cm por 3 cm cada quadrado preto e branco, à diferentes posições angulares e de distância. Estas fotos foram lidas pelo código a partir da opção *Read Images* do menu de opções.

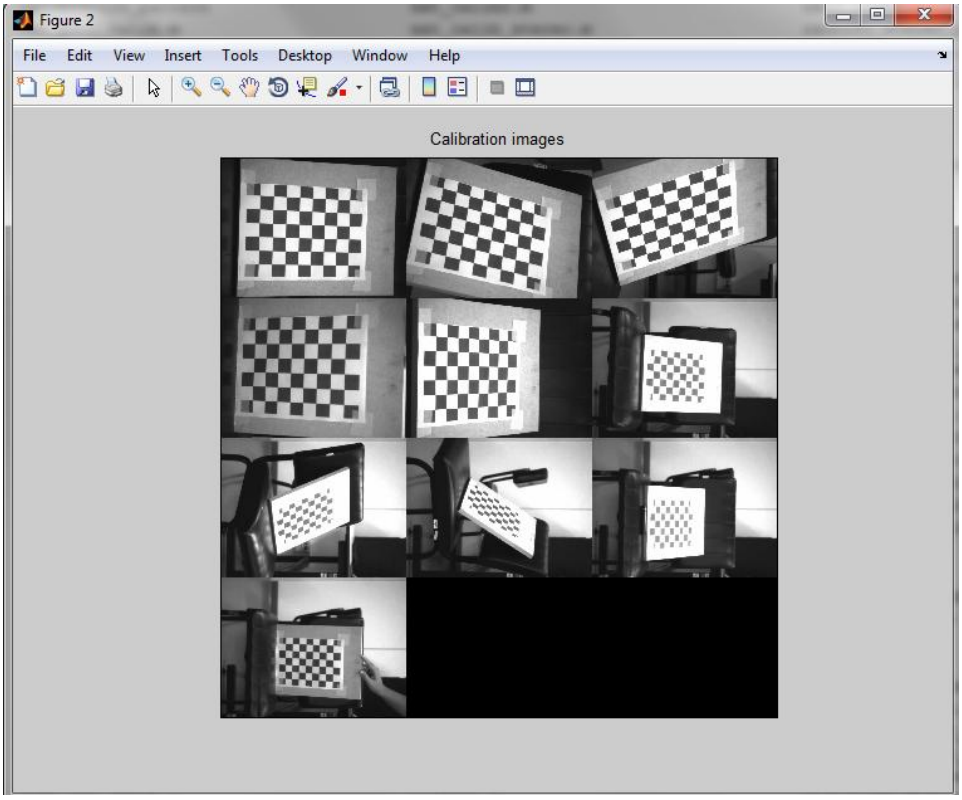


Figura 50: Dez fotos capturadas em posições angulares e de distâncias diferentes[12]

Em seguida com a opção *Extract the Grid Corners* ocorreu uma contagem da quantidade de quadrados na região especificada. Em seguida os pontos de contato entre cada quadrado (*corners*) foram localizados e os descritores dessas regiões foram estabelecidos, com isso os pontos de interesse da imagem foram localizados.

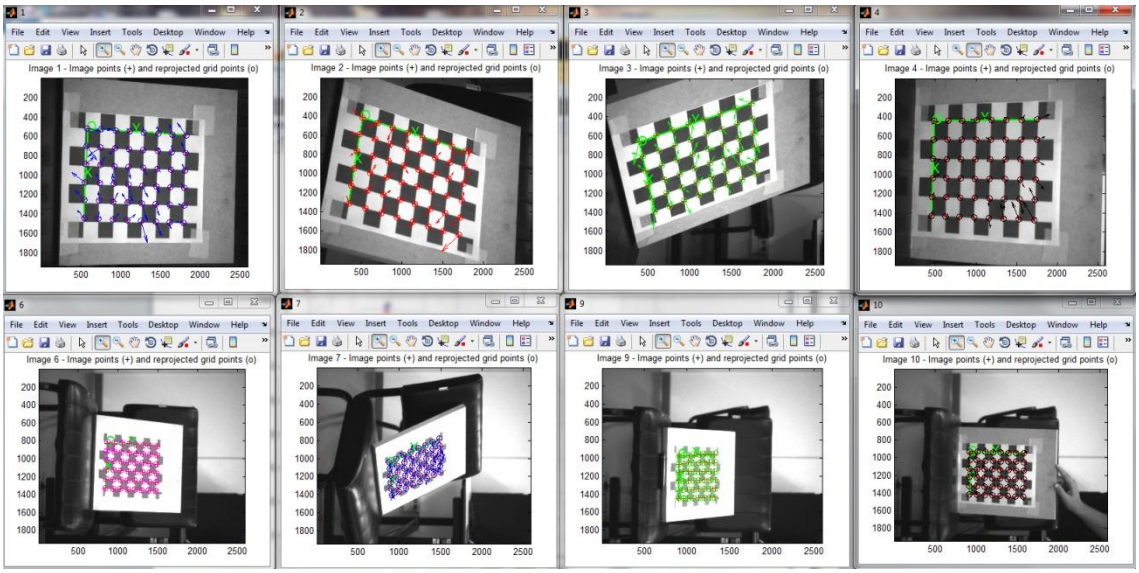


Figura 51: Corners localizados em cada imagem [12]

Após a localização dos pontos de interesse, foi realizada a calibração das imagens para otimização destas realizando a minimização de distorções nas imagens capturadas. A otimização é feita por iterações realizadas computacionalmente da matriz Jacobiana [12] o resultado da calibração das imagens é demonstrado pela figura 25.

```
Calibration parameters after initialization:

Focal Length:          fc = [ 787.30222   787.30222 ]
Principal point:       cc = [ 1295.50000   971.50000 ]
Skew:             alpha_c = [ 0.00000 ]   => angle of pixel = 90.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000   0.00000 ]

Main calibration optimization procedure - Number of images: 10
Gradient descent iterations: 1...2...
Warning: View #8 ill-conditioned. This image is now set inactive. (note: to disactivate this option, set check_cond=0)
3...Intrinsic parameters at frame 8 do not exist

New optimization including the images that have been deactivated during the previous optimization.

Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
    Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
```

Figura 52: Resultados da calibração das imagens capturadas[12]

Por fim, a opção *Show Extrinsic* realiza uma demonstração das imagens em relação à câmera de captura e às coordenadas absolutas em coordenadas 3D.

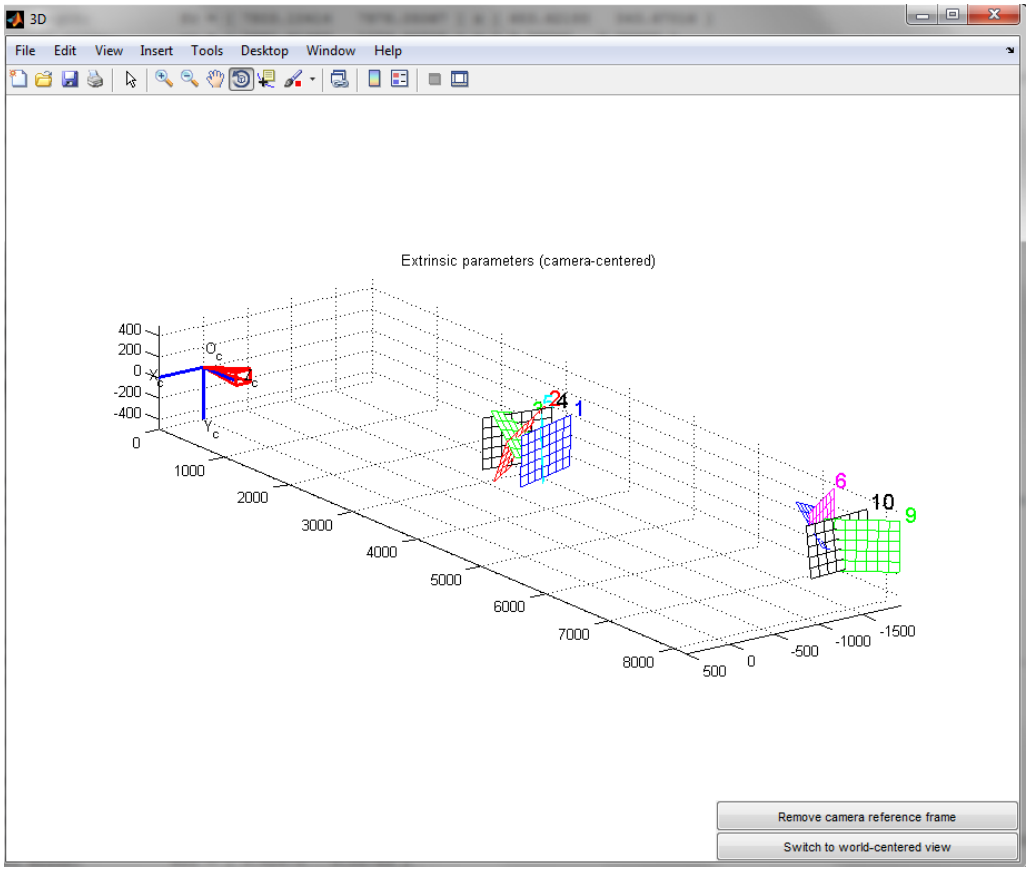


Figura 53: Imagens em relação à câmera [12]

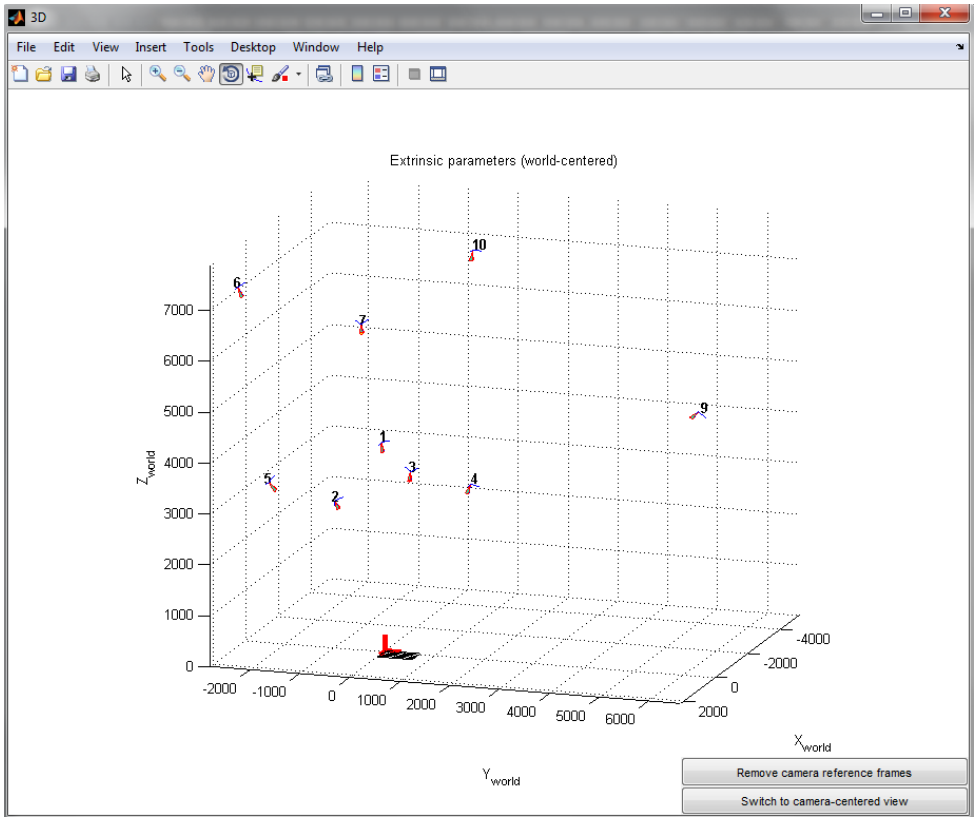


Figura 54: Imagens em relação às coordenadas absolutas [12]

- Visual Studio

Foi feita a utilização do Software *Visual Studio* para processamento de código do algoritmo SURF – disponibilizado em [13] – e o código desenvolvido para utilização do grupo se encontra no Apêndice A.

O processamento do SURF no código começa pela leitura das imagens escolhidas pelo usuário, em seguida percorre os seguintes passos:

- Detecção de pontos de interesse utilizando a função *Surf Detector*
- Cálculo dos descritores
- Localização de pontos condizentes de descritores das imagens
- Representação de pontos condizentes satisfatórios das imagens
- Representação de linhas entre pontos condizentes satisfatórios das duas imagens

6.3.3 RECURSOS ONLINE

O grupo utilizou o simulador online [8] para o processamento do SIFT em imagens. A seguir, os passos que o grupo realizou para a utilização deste simulador:

- Captura de imagens utilizando os dispositivos DFM72BUC02 e DFK31BF03

- *Upload* das fotos na base de dados do simulador (podem ser carregadas apenas duas imagens por vez)
- Processamento das duas imagens pelo algoritmo SIFT
- Análise dos resultados obtidos do escopo do projeto

7 CONCLUSÕES

A captura de imagens será realizada a partir da utilização de uma câmera. Utilizando apenas uma câmera, foi observada, em relação ao uso de duas câmeras, maior detecção de pontos de interesse, além de pontos de interesse condizentes entre as duas imagens comparadas, resultado encontrado tanto no simulador online - tópico 6.3.2, quanto no uso do software Visual Studio – tópico 6.3.1.

A distância utilizada entre a câmera e o rosto da pessoa será de 595 mm, distância esta calculada tomando como base o tamanho máximo de abertura mandibular nos testes realizados pelo grupo. Esta distância foi encontrando utilizando-se os requisitos do dispositivo DFM72BUC02.

A captura das imagens será realizada com posição angular de 0º entre o eixo axial da câmera e o plano normal ao rosto da pessoa, ou seja, as capturas serão frontais. Na captura frontal realizada pelo grupo, foram obtidos como pontos de interesse, os pontos relacionados à ponta do nariz e do queixo. Estas regiões têm a vantagem de apresentarem movimentos mais estáveis no rosto, além disso, ao se considerar o ponto do queixo (que representa a mandíbula), o objetivo é relacioná-lo ao ponto do nariz (que representa o crânio) que servirá de coordenada fixa para a mandíbula, uma vez que o nariz apresenta estabilidade quando em movimento da mandíbula como pode ser observado na figura 26.

A captura de imagens *será* realizada à taxa de 25 FPS. O grupo observou que a câmera DFM72BUC02 apresentou processamento de imagens e capturas efetivas para 25 FPS, abaixo deste valor, foi concluído que a discretização das imagens tiradas em sequência é tão grande, que informações para análise dos dados obtidos são perdidas, para esta análise foi considerada uma captura sequencial a 10 FPS. O grupo concordou que as capturas não precisariam ser efetuadas a valores superiores a 25 FPS, uma vez que os resultados obtidos a esta taxa já foram conclusivos e devido à dificuldade de processamento pelo computador utilizado pelo grupo a taxas superiores.

O processamento do algoritmo SURF será realizado sobre imagens em escalas de cinza, uma vez que a quantidade e consistência na detecção de pontos de interesse não variam de forma significativa entre uma imagem colorida e uma em escalas de cinza, e também em razão da imagem colorida necessitar maior memória para processamento, uma vez que a imagem colorida grava informação de três canais (RGB) e a imagem em escalas de cinza apenas a variação de luminosidade.

A resolução da imagem utilizada será de 100% da câmera utilizada, uma vez que resoluções acima deste valor distorcem a imagem e valores menores, por sua vez, eliminam informações da imagem, excluindo pontos de interesse.

O valor de limite para matriz hessiana será de 200, o processamento do algoritmo SURF com este valor traz pontos de interesse satisfatórios ao grupo, tanto em questões de quantidade quanto de consistência. Valores baixos como 50 trazem muitos pontos de interesse desnecessários, sendo relacionados com ruídos da imagem, valores altos como 1000 trazem poucos pontos de interesse.

As capturas serão realizadas em ambientes com *background* de coloração escura, fazendo com que ruídos sejam minimizados, uma vez que imagens capturadas sobre planos de fundo claros geram contrastes grandes no contorno do rosto da pessoa, pontos de interesse não satisfatórios ao grupo.

O recorte e seleção de áreas interessantes ao projeto (região mandibular e do nariz) aumentou consideravelmente a consistência e precisão de detecção e *matching* de pontos de interesse entre imagens.

O algoritmo SURF será processado em um vídeo gravado e não em uma captura *real time*, uma vez que a última causa problemas relacionados a estouro e insuficiência de memória.

Para comparação entre pontos de interesse de uma imagem com o vídeo gravado, será utilizado o primeiro frame do vídeo como a imagem de comparação. Este procedimento visa maximizar a compatibilidade entre o vídeo e a imagem para que sejam escolhidos pontos de interesse condizentes entre as duas estruturas, ou seja, que não sejam tão afetados por elementos externos, tal como variação de luminosidade e contraste.

O rastreamento será demonstrado de duas maneiras: em coordenadas universais considerando a movimentação do nariz e do queixo, desta forma a representação gráfica é intuitiva e comparável ao *matching* do vídeo com a imagem, e uma segunda maneira em coordenadas relativas, considerando a movimentação do queixo em relação ao nariz (centro estático), esta representação permite a visualização do movimento relativo do queixo, fazendo com que possa ser verificada uma disfunção do movimento mandibular do paciente.

Um procedimento de homografia foi utilizado para que a área na qual os pontos de interesse fossem processados acompanhassem os movimentos do grupo destes pontos, desta forma, fazendo com que não ocorresse uma perda de referência dos pontos. As curvas obtidas pelo procedimento utilizando homografia apresentaram movimentos menos discretizados em relação ao procedimento sem o uso da função de homografia, mais semelhantes a um movimento humano.

Foi realizada uma verificação de precisão utilizando o mecanismo de testes, o grupo obteve divergência de 0,19 cm entre a trajetória obtida no experimento e a trajetória da face real.

8 REFERÊNCIAS

[1] Aula de Anatomia – http://www.auladeanatomia.com/site/pagina.php?idp=34 (último acesso em 10/06/2015)

[2] Daniele Manfrim **Influência da inclinação do garfo de mordida do arco facial na montagem do modelo superior em articulador semi-ajustável do tipo arcon e não-arcon**, USP – Faculdade de Odontologia de Ribeirão Preto, 2008

[3] Okeson, Jeffrey P. **Tratamento das Desordens Temporomandibulares e Oclusão** 7ª Ed. 2013

[4] Odonto Image – http://www.odontoimagem.odo.br/index.php?pg=tomografia-computadorizada (último acesso em 10/06/2015)

[5] Radio North – http://www.radionorth.com.br/radiografias-extra-orais (último acesso em 10/06/2015)

[6] Vomicae – http://vomicae.net/tecnologia/novo-equipamento-de-tomografia-computadorizada-faz-imagens-em-3d-de-grande-precisao-e-reduz-o-nivel-de-radiacao/ (último acesso em 10/06/2015)

[7] Cedav – http://www.cedav.com.br/exames.php?exa=0 (último acesso em 10/06/2015)

[8] Rey-Otero I. e Delbracio M., **Anatomy of the SIFT Method,** IPOL Journal 4 (2014), pp. 370–396. Site da demonstração: http://demo.ipol.im/demo/82/ (último acesso em 15/06/2015)

[9] Fernandes Neto A. F., et al. Movimentos Mandibulares. Univ. Fed. Uberlândia,2006

[10] Missaka R., Adachi LK., Tamaki R., Shinkai R. S. A., Campos T. N.,Horikawa O. **Development of an experimental optoelectronic device to study the amplitude of mandibular movements.** Brazilian Oral Research 2008; 22(2):151-7

[11] The Imaging Source – http://www.theimagingsource.com/en_US/products (último acesso em 15/06/2015)

[12] Camera Calibration Toolbox for Matlab http://www.vision.caltech.edu/bouguetj/calib_doc/ (último acesso em 16/06/2015)

[13] GitHub – https://gist.github.com/Jinqiang (último acesso em 10/06/2015)

9 BIBLIOGRAFIA

Fernandes Neto A. F., et al. **Movimentos Mandibulares.** Univ. Fed. Uberlândia, 2006

Soboļeva U., Lauriņa L., Slaidiņa A. **Jaw tracking devices - historical review of methodsdevelopment. Parts I & II**. Stomatologija, Baltic Dental and Maxillofacial Journal, 7:67-76, 2005

Missaka R., Adachi LK., Tamaki R., Shinkai R. S. A., Campos T. N., Horikawa O. **Development of an experimental optoelectronic device to study the amplitude of mandibular movements.** Brazilian Oral Research 2008; 22(2):151-7

Lowe D. G. **Distinctive Image Features from Scale-Invariant Keypoints.** International Journal of Computer Vision 60(2), 91-110, 2004

Ives R. Otero, Mauricio Delbravio. **The Anatomy of the SIFT Method.** Preprint, 2013
Jan J. Koenderink. **The Structure of Images.** Department of Medical and Physiological Physics, State University Ultrecht, 1984

Sung Kim, Riley Casper. **Applications of Convolution in Image Processing with Matlab.** University of Washington, 2013

Adrian Kaehler, Gary Bradski. **Learning OpenCV**. O'Reilly Media, 2013 pg 140 - 230

Peter Corke. **Robotics, Vision and Control.** Springer, 2011 pg 98 - 132

Herbert Bay, Tynne Tuytelaars. **SURF: Speeded Up Robust Features.** Katholieke Universiteit Leuven

Reinaldo Missaa, **Centros Instantâneos de rotação mandibular por meio de processamento de imagem obtida por metodologia optoeletrônica**, Faculdade de Odontologia da Universidade de São Paulo, 2010

10. APÊNDICE

Apêndice A

Código Visual Studio – SURF

```
#include <vector>
#include "opencv2\opencv.hpp"
#include <opencv2\nonfree\features2d.hpp>

using namespace std;
using namespace cv;

Mat img_1, img_2;

Mat plot1, plot2, plot3, plot4;
Mat img_prev, img_keypoints;

vector<Point3f> origpat;

vector<vector<Point2f>> firstscene, oldscene, newscene, pat_scene;

vector<Point2f> polygon;

int rectFlag = 0;
Point2i ptRect[2];

int drawFlag = 0;
int loopFlag = 0;
int clckflag = 0;

String input = "Input";
```

```
String polyput = "Output";

String whatput = "Prevput";

String put3dee = "Perspectiva";

String winname = "Frame";

String output1 = "f(x,y)";

String output2 = "f(x,t)";

String output3 = "f(t,y)";


void xpoint(Mat img, Point pt, Scalar color, int style = 1, int width = 2,
int thickness = 1) {

                if (style == 1 || style == 3) {

                  line(img, Point(pt.x - width, pt.y - width), Point(pt.x +
width, pt.y + width), color, thickness);

                  line(img, Point(pt.x + width, pt.y - width), Point(pt.x -
width, pt.y + width), color, thickness);

                }

                if (style == 2 || style == 3) {

                  line(img, Point(pt.x - width, pt.y), Point(pt.x + width,
pt.y), color, thickness);

                  line(img, Point(pt.x, pt.y + width), Point(pt.x, pt.y -
width), color, thickness);

                }

}


Point2f findCenter(vector<Point2f> setpoints) {

                Point2f center = Point2f(0, 0);

                for (Point2f& pt : setpoints)   center += pt;

                return center*((float) 1 / setpoints.size());

}


void CutRect(int event, int x, int y, int, void*) {

                if (event == EVENT_LBUTTONDOWN) {

                  if (rectFlag == 0) {
```

```
                ptRect[0] = ptRect[1] = Point2f(x, y);
                rectFlag++;
            }
            if (rectFlag == 2)
                rectFlag++;
        }
        if (event == EVENT_LBUTTONUP)   {
            if (rectFlag == 1)
                rectFlag++;
        }
        if (event == EVENT_RBUTTONDOWN) {
            if (rectFlag < 3) {
                ptRect[0] = ptRect[1] = Point2i(0, 0);
                rectFlag = 0;
            }
        }
        if (event == EVENT_MOUSEMOVE)   {
            if (rectFlag == 1)
                ptRect[1] = Point2f(x, y);
        }
}

// rectDraw: desenha área retangular
Rect rectDraw(Mat img) {
        String input = "Cutting rectangle";

        // Criando janela para recortar frame
        imshow(input, img);

        //Loop de recorte
```

```
                while (rectFlag < 4) {
                  Mat img_clone = img.clone();  // clona
                  setMouseCallback(input, CutRect, NULL);
                  line(img_clone, ptRect[0], Point2i(ptRect[0].x,
ptRect[1].y), 0);
                  line(img_clone, ptRect[0], Point2i(ptRect[1].x,
ptRect[0].y), 0);
                  line(img_clone, Point2i(ptRect[0].x, ptRect[1].y),
ptRect[1], 0);
                  line(img_clone, Point2i(ptRect[1].x, ptRect[0].y),
ptRect[1], 0);
                  imshow(input, img_clone);
                  char key = waitKey(1);
                  if (key == '\r')  {     // Pressionado Enter
                        if (rectFlag == 0) {    // Caso não esteja
desenhando, não recortar retângulo
                              ptRect[0] = Point2i(0, 0);
                              ptRect[1] = Point2i(img.cols, img.rows);
                              rectFlag = 4;
                        }
                        rectFlag++;
                  }
                  if (rectFlag == 3)
                        rectFlag++;
                }
                destroyWindow(input);
                return Rect(ptRect[0], ptRect[1]);
}

// DrawPolygons: desenha polígonos para acompanhar áreas próximas
void DrawPolygons(int event, int x, int y, int, void*)
{
                Point2i ptMouse = Point2i(x, y);
```

```
                // Ao clicar com o botão esquerdo, desenha um ponto e
linhas
                if (event == EVENT_LBUTTONDOWN) {
                  if (drawFlag == 1 && polygon.size() > 0) {
                        xpoint(img_prev, ptMouse, CV_RGB(255, 0, 0));
                        line(img_prev, ptMouse, polygon[polygon.size() -
1], CV_RGB(255, 0, 0));
                        polygon.push_back(ptMouse);
                  }
                  else if (drawFlag == 0 && polygon.size() == 0) {
                        xpoint(img_prev, ptMouse, CV_RGB(255, 0, 0));
                        drawFlag = 1;
                        polygon.push_back(ptMouse);
                  }
                  imshow(input, img_prev);
                }
                if (event == EVENT_MBUTTONDOWN) {
                  if (drawFlag == 2 && polygon.size() > 0) {
                        xpoint(img_prev, ptMouse, CV_RGB(255, 0, 255));
                        line(img_prev, ptMouse, polygon[polygon.size() -
1], CV_RGB(255, 0, 255));
                        polygon.push_back(ptMouse);
                  }
                  else if (drawFlag == 0 && polygon.size() == 0) {
                        xpoint(img_prev, ptMouse, CV_RGB(255, 0, 255));
                        drawFlag = 2;
                        polygon.push_back(ptMouse);
                  }
                  imshow(input, img_prev);
                }
                if (event == EVENT_MOUSEMOVE) {
                  Mat img_clone;
```

```
                if (drawFlag == 1) {
                      img_clone = img_prev.clone();
                      line(img_clone, polygon[polygon.size() - 1],
ptMouse, CV_RGB(0, 255, 255));
                      imshow(input, img_clone);
                }
                if (drawFlag == 2) {
                      img_clone = img_prev.clone();
                      line(img_clone, polygon[polygon.size() - 1],
ptMouse, CV_RGB(0, 255, 0));
                      imshow(input, img_clone);
                }
              }
              if (event == EVENT_RBUTTONDOWN) {
                if (drawFlag == 1) {
                      line(img_prev, polygon[polygon.size() - 1],
polygon[0], CV_RGB(255, 0, 0));
                      xpoint(img_prev, findCenter(polygon), CV_RGB(255,
0, 0), 2);
                      firstscene.push_back(polygon);
                      polygon.clear();
                      drawFlag = 0;
                }
                else if (drawFlag == 2) {
                      line(img_prev, polygon[polygon.size() - 1],
polygon[0], CV_RGB(255, 0, 255));
                      xpoint(img_prev, findCenter(polygon), CV_RGB(255,
0, 255), 2);
                      pat_scene.push_back(polygon);
                      polygon.clear();
                      drawFlag = 0;
                }
                else {
```

```
                        firstscene.clear();

                        pat_scene.clear();

                        img_prev = img_keypoints.clone();

                    }

                    imshow(input, img_prev);

                }

}


// void DrawGuidelines: linhas auxiliares para plotar gráficos

void DrawGuidelines(Mat plot_image, int scale, int pat_size, int
is_timeplot = 0) {

                int cm = int(scale / pat_size * 10); // Número de pixels
igual a 1 centímetro

                int del_t = 48;                                          //
Constante de tempo



                int tx = cm, ty = cm;                                    //
Escolhendo se plotar f(x,t), f(t,y) ou f(x,y)

                if (is_timeplot > 0) tx = del_t;

                if (is_timeplot < 0) ty = del_t;



                // Plotar linhas verticais com espaçamento de 1 cm ou
constante de tempo

                for (int i = 0; i < plot_image.cols; i += tx)

                  line(plot_image, Point2i(i, 0), Point2i(i,
plot_image.rows), CV_RGB(180, 180, 180));



                // Plotar linhas horizont. com espaçamento de 1 cm ou
constante de tempo

                for (int i = 0; i < plot_image.rows; i += ty)

                  line(plot_image, Point2i(0, i), Point2i(plot_image.cols,
i), CV_RGB(180, 180, 180));

}
```

```
// Função de configuração de uma imagem para o padrão a ser usado
Mat arrumarImagem(Mat original){
		Mat clone;
		cvtColor(original, clone, CV_RGB2GRAY);
		transpose(clone, clone);
		flip(clone, clone, 1);
		resize(clone, clone, Size(0, 0), 0.75, 0.75, INTER_LINEAR);
		return clone;
}

int main()
{
		// Obtendo arquivo de vídeo
		VideoCapture capture;
		String file = "F:\\pedro\\Desktop\\TCC\\video9.mp4";
		capture.open(file);

		// Estabelecendo tamanho original do padrão xadrez em
milímetros
		float p_sz = 10.0;

		origpat.push_back(Point3f(0.0f, 0.0f, 0.0f));
		origpat.push_back(Point3f(0.0f, p_sz, 0.0f));
		origpat.push_back(Point3f(0.1f, p_sz, p_sz));
		origpat.push_back(Point3f(0.1f, 0.0f, p_sz));

		Mat isometric = Mat::zeros(3, 1, DataType<double>::type);
		double isoth = acos((1 / sqrt(2) + 2 / sqrt(6) + 1 /
sqrt(3) - 1) * .5);
		isometric.at<double>(0, 0) = -(1 / sqrt(3) + 1 / sqrt(6)) /
(2 * sin(isoth));
```

```
                isometric.at<double>(1, 0) = -(1 / sqrt(2) + 1 / sqrt(3)) /
(2 * sin(isoth));
                isometric.at<double>(2, 0) = (1 / sqrt(6)) / (2 *
sin(isoth));

                Mat fortyfive = Mat::zeros(3, 1, DataType<double>::type);
                fortyfive.at<double>(0, 0) = 0;
                fortyfive.at<double>(1, 0) = -1 /sqrt(2);
                fortyfive.at<double>(2, 0) = -1 /sqrt(2);

                Mat tr_isom = Mat::zeros(3, 1, DataType<double>::type);
                tr_isom.at<double>(0, 0) = 0;
                tr_isom.at<double>(1, 0) = 0;
                tr_isom.at<double>(2, 0) = 0;

                // Capturando e transformando o primeiro frame do video
                capture.read(img_1);
                img_1 = arrumarImagem(img_1);

                Rect rectdim = rectDraw(img_1);
                img_1 = img_1(rectdim);

                // Determinando valor mínimo de contraste para captura de
pontos de interesse
                int minHessian = 300;
                SurfFeatureDetector detector(minHessian);

                vector<KeyPoint> keypoints_1;
                detector.detect(img_1, keypoints_1);

                // Desenhando pontos de interesse em uma nova imagem
```

```
                drawKeypoints(img_1, keypoints_1, img_keypoints,
Scalar::all(-1), DrawMatchesFlags::DEFAULT);

                // Extraindo descritores
                SurfDescriptorExtractor extractor;
                Mat descriptor_1;
                extractor.compute(img_1, keypoints_1, descriptor_1);

                // Criando janela para desenhar padrões na imagem
                img_prev = img_keypoints.clone();
                imshow(input, img_prev);

                while (drawFlag >= 0) {
                  setMouseCallback(input, DrawPolygons, NULL);
                  char key = waitKey(30);
                  if (key == '\r' || key == ' ')
                        drawFlag = -1;
                }
                destroyWindow(input);

                for (vector<Point2f>& vop: firstscene)
                  xpoint(img_keypoints, findCenter(vop), CV_RGB(255, 0, 0),
2);

                vector<vector<Point3f>> objpts;
                objpts.push_back(origpat);
                vector<vector<vector<Point2f>>> imgpts;
                vector<Mat> allMat;

                vector<vector<Point3f>> pat_scene3d;
                vector<double> scale (2);
```

```
                double pat_size = norm(origpat[0] - origpat[2]);
                Mat distCoeffs = Mat::zeros(5, 1, DataType<double>::type);

                Mat plot4(500, 500, CV_8UC3, Scalar(255, 255, 255));

                for (int i = 0; i < pat_scene.size(); i++) {
                  scale[i] = 0.5*norm(pat_scene[i][0] - pat_scene[i][2]) +
0.5*norm(pat_scene[i][1] - pat_scene[i][3]);

                  vector<vector<Point2f>> ararr;
                  ararr.push_back(pat_scene[i]);
                  imgpts.push_back(ararr);

                  Mat camMat = initCameraMatrix2D(objpts, imgpts[0],
img_1.size());
                  //Mat camMat = initCameraMatrix2D(objpts, imgpts[i],
img_1.size());
                  allMat.push_back(camMat);

                  Mat rvec, tvec, rotMat;
                  solvePnPRansac(origpat, pat_scene[i], camMat, distCoeffs,
rvec, tvec);
                  Rodrigues(rvec, rotMat);

                  vector<Point3f> pat3d;
                  for (Point2f& pat_pt : pat_scene[i]) {
                        Mat projPt = Mat::ones(3, 1,
DataType<double>::type);
                        projPt.at<double>(0, 0) = pat_pt.x;
                        projPt.at<double>(1, 0) = pat_pt.y;
```

```
                        // Matrizes que fazem parte da equação s *
projection_Point = camera_Matrix * ( rotation_Matrix * 3D_Point +
translation)

                        Mat sMat, sMat2;

                        float s;

                        sMat = rotMat.inv() * camMat.inv() * projPt;    // *
3D_Point

                        sMat2 = rotMat.inv() * tvec;  // Account for
translation



                        double zest = tvec.at<double>(2, 0) * scale[i] /
pat_size;



                        cout << zest << endl;



                        s = zest + sMat2.at<double>(2, 0);

                        s /= sMat.at<double>(2, 0);



                        Mat mat3D = Mat::zeros(3, 1,
DataType<double>::type);

                        mat3D = rotMat.inv() * (camMat.inv() * s * projPt -
tvec);



                        pat3d.push_back(Point3f(mat3D.at<double>(0, 0),
mat3D.at<double>(1, 0), mat3D.at<double>(2, 0)));

                }



                vector<Point2f> iso_pat;

                projectPoints(pat3d, fortyfive, tr_isom, camMat,
distCoeffs, iso_pat);




                for (int j = 0; j < iso_pat.size(); j++) {

                        cout << "Isopat[" << i <<"][" << j <<"]= " <<
iso_pat[j] << endl;
```

```
                        xpoint(plot4, Point2f(10, 10), CV_RGB(180, 0,
180));
                        line(plot4, iso_pat[j], iso_pat[(j + 1) %
iso_pat.size()], CV_RGB(180, 0, 180));
                  }
                  imshow(put3dee, plot4);
                  waitKey();

                  pat_scene3d.push_back(pat3d);
                }

                vector<vector<Point2f>> inside (firstscene.size());
                vector<Point2f> firstponto;
                vector<Point2f> velocity (firstscene.size());

                for (int i = 0; i < firstscene.size(); i++) {
                  for (KeyPoint& kp : keypoints_1)
                        if (pointPolygonTest(firstscene[i], kp.pt, false)
== 1)
                              inside[i].push_back(kp.pt);

                  firstponto.push_back(findCenter(inside[i]));
                  xpoint(img_keypoints, firstponto[i], CV_RGB(255, 128, 0),
3);

                  velocity[i] = Point2f(0, 0);
                  for (int j = 0; j < firstscene[i].size() - 1; j++)
                        line(img_keypoints, firstscene[i][j],
firstscene[i][j + 1], CV_RGB(255, 0, 0));
                  line(img_keypoints, firstscene[i][firstscene[i].size() -
1], firstscene[i][0], CV_RGB(255, 0, 0));
                }
```

```
                imshow(polyput, img_keypoints);


                // Criando imagens para plotar gráficos

                Mat plot1(rectdim.height + 50,  rectdim.width,    CV_8UC3,
Scalar(255, 255, 255));

                DrawGuidelines(plot1, scale[0], pat_size);

                xpoint(plot1, firstponto[0], CV_RGB(0, 140, 0), 3);

                xpoint(plot1, firstponto[1], CV_RGB(140, 0, 140), 2);


                Mat plot2(rectdim.height + 50, 500, CV_8UC3, Scalar(255,
255, 255));

                DrawGuidelines(plot2, scale[0], pat_size, 1);

                line(plot2, Point2i(0, firstponto[0].y),
Point2i(plot2.cols, firstponto[0].y), CV_RGB(0, 140, 0));


                Mat plot3(500, rectdim.width, CV_8UC3, Scalar(255, 255,
255));

                DrawGuidelines(plot3, scale[0], pat_size, -1);

                line(plot3, Point2i(firstponto[0].x, 0),
Point2i(firstponto[0].x, plot3.rows), CV_RGB(0, 140, 0));


                vector<Point2f> oldpdiff(firstscene.size() - 1);

                for (int i = 0; i < firstponto.size() - 1; i++) {

                  oldpdiff[i] = firstponto[i+1] - firstponto[i];

                  xpoint(img_keypoints, firstponto[i], CV_RGB(255, 128,
0));

                }


                oldscene = firstscene;

                vector<Point2f> oldponto = firstponto;


                waitKey();

                int frame = 0;
```

```
                while (true) {


                  if (loopFlag == 1) {
                        loopFlag = 0;
                        capture.open(file);
                        oldscene = firstscene;
                        oldponto = firstponto;
                        for (int i = 0; i < firstponto.size(); i++) {
                              velocity[i] = Point2f(0, 0);
                              if (i = firstponto.size() - 1)
                                    break;
                              oldpdiff[i] = firstponto[i + 1] -
firstponto[i];
                        }
                  }
                  //Verificando se o vídeo chegou ao último frame
                  //'-1' é adicionado porque dois frames são lidos por vez,
caso contrário ocorre erro de memória
                  while (capture.get(CV_CAP_PROP_POS_FRAMES) <
capture.get(CV_CAP_PROP_FRAME_COUNT) - 1) {
                        img_prev = img_keypoints;
                        imshow(whatput, img_prev);
                        newscene.clear();


                        capture.read(img_2);
                        img_2 = arrumarImagem(img_2);
                        img_2 = img_2(rectdim);


                        vector<KeyPoint> keypoints_2;
                        detector.detect(img_2, keypoints_2);
```

```
                        Mat descriptor_2;
                        extractor.compute(img_2, keypoints_2,
descriptor_2);

                        // Matches with Flann Based Matching
                        FlannBasedMatcher matcher;
                        vector<DMatch> match;
                        matcher.match(descriptor_1, descriptor_2, match);

                        // 1a forma de Calcular distância de máxima para
escolha de matches - média de todas as distâncias de matches
                        float maxdist = 0.0; float mindist = 100.0; float
meandist = 0.0; float sd = 0.0;
                        for (int i = 0; i < descriptor_1.rows; i++)     {
                              float dist = match[i].distance;
                              meandist += (float)dist / descriptor_1.rows;
                              if (mindist > dist)
                                    mindist = dist;
                              if (maxdist < dist)
                                    maxdist = dist;
                        }

                        // Calculando distância de máxima para escolha de
matches - média de todas as distâncias de matches
                        vector<vector<DMatch>> radius_match;
                        matcher.radiusMatch(descriptor_1, descriptor_2,
radius_match, maxdist * .1);

                        vector< DMatch> good_matches;
                        for (int i = 0; i < descriptor_1.rows; i++) {

                              if (radius_match[i].size() > 0)
```

```
                good_matches.push_back(radius_match[i][0]);
                        }

                        /* Draw Good Matches */
                        Mat img_goodmatches;
                        drawMatches(img_1, keypoints_1, img_2, keypoints_2,
good_matches, img_goodmatches,
                                Scalar::all(-1), Scalar::all(-1),
vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

                        Mat img_matches = img_goodmatches;
                        imshow(winname, img_matches);
                        vector<Point2f> newponto(oldscene.size());

                        for (int j = 0; j < oldscene.size(); j++) {
                              inside[j].clear();

                              vector<Point2f> obj, scn;
                              for (int i = 0; i < good_matches.size(); i++)
{
                                    if (pointPolygonTest(oldscene.at(j),
keypoints_1[good_matches[i].queryIdx].pt, false) == 1) {
                obj.push_back(keypoints_1[good_matches[i].queryIdx].pt);
                scn.push_back(keypoints_2[good_matches[i].trainIdx].pt);
                                        xpoint(img_keypoints,
scn[scn.size() - 1], CV_RGB(140, 0, 0), 1, 3);
                                    }
                              }

                              Point2f center_obj = findCenter(obj);
                              Point2f center_scn = findCenter(scn);
```

```
                              cout << "scene.size(): " << scn.size() <<
endl;

                              bool too_warped = false;

                              if (scn.size() >= 4) {
                                    vector<Point2f>
nscn(oldscene[j].size());
                                    Mat H = findHomography(obj, scn,
CV_RANSAC);
                                    cout << H << endl;
                                    perspectiveTransform(oldscene[j], nscn,
H);
                                    for (int i = 0; i < nscn.size(); i++)
                                          if (too_warped) {
                                          too_warped = true;
                                          nscn.clear();
                                          }
                                    newscene.push_back(nscn);
                              }

                              if (scn.size() < 4 || too_warped) {
                                    vector<Point2f> nscn;
                                    if (scn.size() == 0) {
                                          center_obj = Point2f(0, 0);
                                          center_scn = velocity[1];
                                    }
                                    for (int i = 0; i < oldscene[j].size();
i++) {
                                          Point2f n_pt = oldscene[j][i] +
(center_scn - center_obj);
                                          nscn.push_back(n_pt);
```

```
                                        //cout << "newscene[" << j << "]["
<< nscn.size() - 1 << "]: " << nscn[nscn.size() - 1] << endl;

                                  }

                                  newscene.push_back(nscn);

                            }


                            for (int i = 0; i < keypoints_2.size(); i++)

                                  if (pointPolygonTest(newscene[j],
keypoints_2[i].pt, false) == 1)

                inside[j].push_back(keypoints_2[i].pt);


                                  newponto[j] = findCenter(inside[j]);


                      }


                      vector<Point2f> newpdiff(newscene.size() - 1);

                      for (int i = 0; i < newponto.size() - 1; i++) {

                            newpdiff[i] = newponto[i + 1] - newponto[i];

                            if (norm(newpdiff[i]) != 0)

                                  xpoint(plot1, newpdiff[i] +
firstponto[0], CV_RGB(140, 0, 140));

                      }



                      frame++;

                      xpoint(plot2, Point2i(10 * frame, (newpdiff[0] +
firstponto[0]).y), CV_RGB(128, 0, 128));

                      line(plot2, Point2i(10 * frame, (newpdiff[0] +
firstponto[0]).y),

                            Point2i(10 * (frame - 1), (oldpdiff[0] +
firstponto[0]).y), CV_RGB(128, 0, 128));
```

```
                        xpoint(plot3, Point2i((newpdiff[0] +
firstponto[0]).x, 10 * frame), CV_RGB(128, 0, 128));

                        line(plot3, Point2i((newpdiff[0] +
firstponto[0]).x, 10 * frame),

                              Point2i((oldpdiff[0] + firstponto[0]).x, 10 *
(frame - 1)), CV_RGB(128, 0, 128));


                        imshow(output1, plot1);

                        imshow(output2, plot2);

                        imshow(output3, plot3);


                        drawKeypoints(img_2, keypoints_2, img_keypoints,
Scalar::all(-1), DrawMatchesFlags::DEFAULT);



                        for (int i = 0; i < newscene.size(); i++) {

                              cout << newponto[i] << endl;

                              velocity[i] = newponto[i] - oldponto[i];

                              xpoint(img_keypoints, newponto[i], CV_RGB(255,
128, 0));

                              xpoint(img_keypoints, findCenter(inside[i]),
CV_RGB(0, 128, 255));

                              for (int j = 0; j < newscene[i].size() - 1;
j++)

                                    line(img_keypoints, newscene[i][j],
newscene[i][j + 1], CV_RGB(255, 0, 0));

                              line(img_keypoints,
newscene[i][newscene[i].size() - 1], newscene[i][0], CV_RGB(255, 0, 0));

                        }


                        imshow(polyput, img_keypoints);



                        oldponto = newponto;
```

```
                oldpdiff = newpdiff;

                keypoints_1 = keypoints_2;
                descriptor_1 = descriptor_2;
                img_1 = img_2;
                oldscene = newscene;

                char key = waitKey(clckflag);

                if (key == 27)          //ESC to exit
                      return 0;
                if (key == 'r')         //restart
                      break;
                if (key == 'f')         //frame by frame
                      clckflag = 0;
                if (key == 'p')         //pause-unpause
                      clckflag = 30 - clckflag;
            }
            loopFlag = 1;
            capture.release();
        }
        return 0;
}
```

Apêndice B

Mecanismo de teste

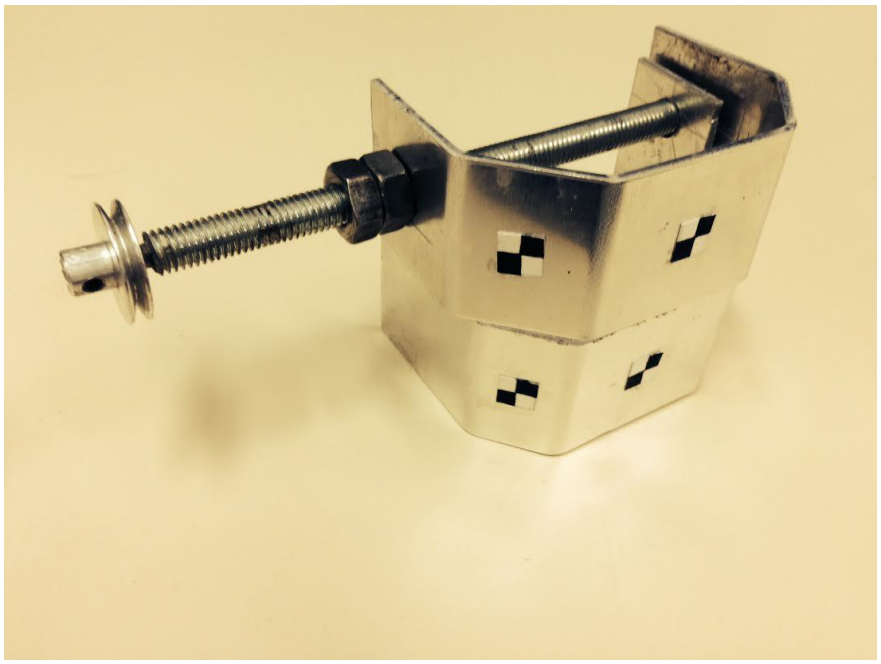
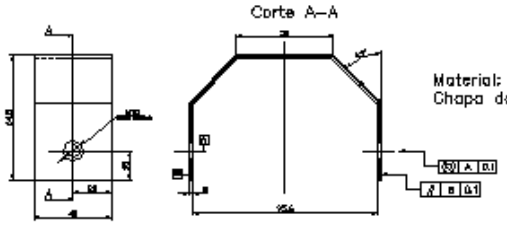
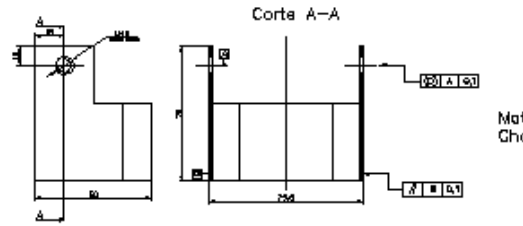


Figura 55: Mecanismo de teste

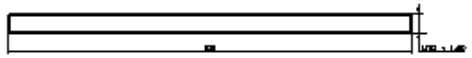
Desenho de fabricação do mecanismo de teste



Material: SAE 1020
Chapa dobrada



Material: SAE 1020
Chapa dobrada



Material: SAE 1020
Barra laminada rosacada

Material:SAE1020

ITEM	QTDE.	DESCRIÇÃO	MATERIAL	OBS.
USP - Escola Politécnica de São Paulo			RAIOS NÃO ESP.: R	DIMENSÕES EM MM
TOL. GERAL TOLERÂNCIA	Chapa Dobrada		CHANFROS NÃO ESP.: 1x45	FOLHA A3
ESCALA 1:2	PROJETO NOME Grupo 23	19/11/15	1º DIEDRO	DESENHO Nº 01
	DESENHO NOME Grupo 23	19/11/15		
	DIA/SEMESTRE/PERÍODO	19-11-15/10º/Integral		