INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE – CÂMPUS PELOTAS - VISCONDE DA GRAÇA
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

Controle Sombra:

Interface de Controle Robótico por Gestos baseada em

Processamento de Imagens

Matheus de Moraes Gadea

Pelotas, 2025.

Matheus de Moraes Gadea

Controle Sombra:
Interface de Controle Robótico por Gestos baseada em

Processamento de Imagens

Projeto de Desenvolvimento de Sistema apresentado como requisito na disciplina de Metodologia de Pesquisa II do curso Técnico em Desenvolvimento de Sistemas, do Instituto Federal Sul-Rio-Grandense – *Campus* Pelotas - Visconde da Graça.

Orientador: Profa Verlani Timm Hinz.
Coorientador: Jardel da Silva

Pelotas, 2025

Resumo

Este trabalho apresenta o desenvolvimento de um sistema de controle robótico denominado Controle Sombra. O trabalho tem como tema a área de Robótica Aplicada, explorando o uso de técnicas de Inteligência Artificial e Visão Computacional para criar um mecanismo intuitivo de interação entre usuários e dispositivos robóticos. A motivação pessoal para o desenvolvimento do trabalho vem do interesse do autor em ficção científica, robótica e inteligência artificial, junto à convicção de que essas tecnologias podem gerar avanços significativos em diferentes áreas. O objetivo central é possibilitar a operação de um braço robótico por meio da imitação de gestos humanos capturados em tempo real. Entre os objetivos específicos do projeto estão a simplificação da operação de braços robóticos, a adaptação do sistema a múltiplos contextos de uso e a demonstração de um protótipo viável para aplicações futuras, como o manuseio de materiais químicos ou biológicos em situações de risco. A metodologia utilizada foi pesquisa, dedução e análise pessoal, com a modelagem sendo realizada por meio de casos de uso e concepção do banco de dados. Conclui-se que o Controle Sombra representa uma aplicação promissora da robótica no cotidiano, demonstrando como gestos simples podem ser traduzidos em comandos precisos a dispositivos mecânicos. O sistema alia inovação, praticidade e segurança, além de apontar caminhos para aplicações mais avançadas, reforçando a relevância do uso da inteligência artificial na interface homem-máquina.

Palavras-chave: robótica; usabilidade; controle por gestos.

SUMÁRIO

Resumo **3**
1. INTRODUÇÃO **5**
2. OBJETIVOS **6**
2.1. OBJETIVO GERAL 6
2.2. OBJETIVOS ESPECÍFICOS 6
3. ESPECIFICAÇÃO DE REQUISITOS **7**
3.1. MÉTODOS DE ESPECIFICAÇÃO DE REQUISITOS **7**
3.2. REQUISITOS FUNCIONAIS 7
3.3. REQUISITOS NÃO FUNCIONAIS **8**
4. MODELAGEM **9**
4.1. MODELO DE CASOS DE USO 9
4.2. MODELAGEM CONCEITUAL DO BANCO DE DADOS 9
4.3. MODELAGEM LÓGICA DO BANCO DE DADOS 10
5. TECNOLOGIAS UTILIZADAS **12**
6. DESCRIÇÃO DO SISTEMA **13**
6.1 PRINCIPAIS TELAS 13
6.2 - Back-end de movimento robótico 15
8. REFERÊNCIAS **26**
APENDICE I - INSTRUÇÔES PARA CRIAÇÂO DA BASE DE DADOS **27**

1. INTRODUÇÃO

No mundo atual, o avanço da robótica tem sido promissor, consolidando-se como uma área em constante e acelerado crescimento. Isso se dá pelo fato de que a robótica tem diversos meios de uso em nossa realidade, sendo um destes a produção, que atrai o interesse de grandes empresários pois o uso de corpos mecânicos autônomos gera cortes de custo consideráveis e ao mesmo tempo aumentam significativamente a produção.

Entretanto, a robótica não se limita aos meios de produção. Diversas empresas ao redor do mundo como Nvidia, SpaceX e a Open Bionics pesquisam amplamente a aplicabilidade de corpos robóticos em áreas como exploração espacial, extensores de membros, e até mesmo em áreas de contato direto com pessoas em áreas de serviços domésticos.

O objetivo deste trabalho é ressaltar o poder que a robótica pode exercer no nosso cotidiano e mostrar que é possível fazer coisas verdadeiramente surpreendentes com o uso dela. Além disso, busca-se mostrar um pouco do que é possível criar com nosso conhecimento atual e as ferramentas sobre robótica disponíveis no ambiente do CAVG, assim como inspirar o desenvolvimento futuro de novas aplicações e inovações nessa área.

2. OBJETIVOS

2.1. OBJETIVO GERAL

O objetivo deste trabalho é desenvolver uma interface que seja capaz de ler gestos de um braço humano por meio de uma câmera e transpasse a informação a um braço robótico, de maneira que ele imite da forma que puder o movimento em tempo real.

2.2. OBJETIVOS ESPECÍFICOS

- Facilitar a operação de uma ferramenta como um braço robótico e tornar essa operação intuitiva.
- Permitir que a aplicabilidade fique a critério do usuário, permitindo que o sistema alcance as expectativas da maioria de possíveis usuários.

- Integrar bibliotecas Python com um braço robótico Dobot e proporcionar um movimento fluido e um controle intuitivo da ferramenta

- Minimizar o *delay* de processamento utilizando um cabo USB conectado diretamente ao computador

3. ESPECIFICAÇÃO DE REQUISITOS

3.1. MÉTODOS DE ESPECIFICAÇÃO DE REQUISITOS

A especificação de requisitos foi feita com base em observação pessoal de ambientes com potencial para uso do sistema. Filmes e séries de ficção serviram como base para a conceitualização do projeto como um todo. Os requisitos de segurança do sistema foram pensados com base em conhecimento pessoal de controle de ferramentas em ambientes controlados.

3.2. REQUISITOS FUNCIONAIS

Tabela 1: requisitos funcionais

REF 1	Gerenciar Usuários	O administrador deverá cadastrar os usuários no sistema e também ser capaz de gerenciar os dados de login dos mesmos.
REF 3	Fazer login/logout	O usuário deverá fazer login para liberar seu acesso ao sistema.
REF 4	Capturar imagens	O sistema deverá capturar imagens em tempo real do usuário.
REF 5	Comandar o braço	O usuário deverá mover-se para que o sistema interprete e copie o movimento.

Fonte: autoria própria

3.3. REQUISITOS NÃO FUNCIONAIS

Tabela: 2 requisitos não funcionais

RNF1	Usabilidade	O sistema deve ser de fácil utilização e intuitivo facilitando a interação com o usuário.
RNF2	Segurança	O sistema só deverá ser acessado por um usuário por vez.
RNF3	Desenvolvimento	O sistema deverá funcionar em Windows.
RNF4	Segurança	O braço robótico deve funcionar sempre dentro de seu ambiente controlado.
RNF 5	Segurança	Os usuário só poderão ter a acesso ao sistema caso sejam cadastrados no mesmo por um administrador.
RNF 6	desempenho	O sistema deve ter o mínimo delay possível na interpretação do gesto e no movimento do braço.

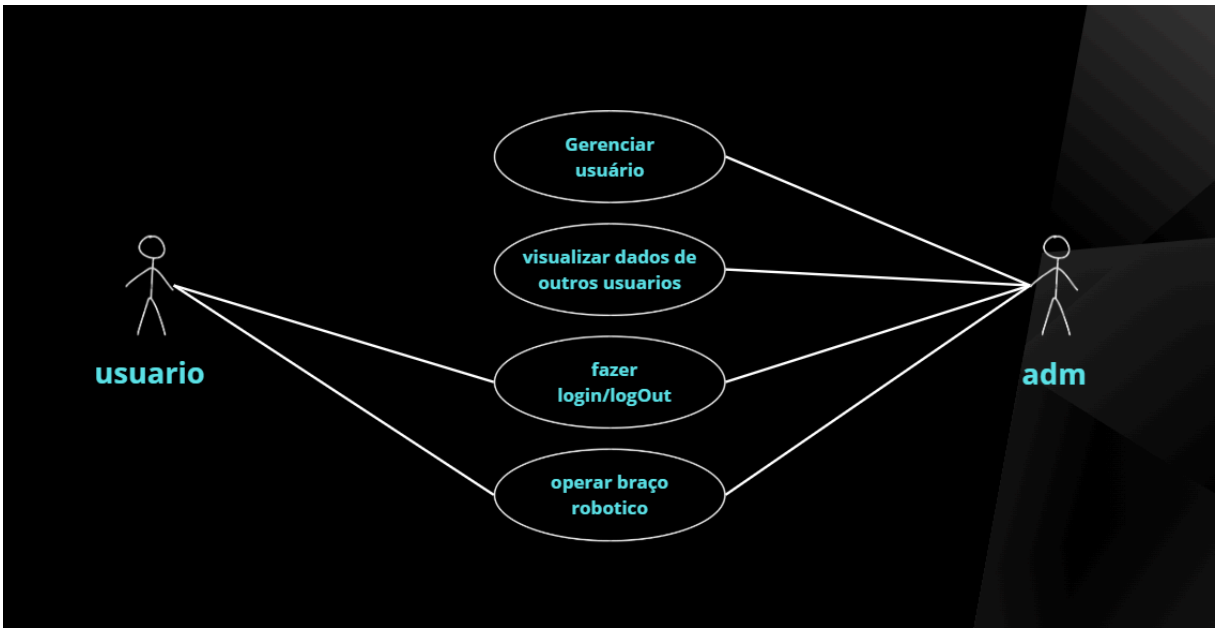
Fonte: autoria própria

4. MODELAGEM

4.1. MODELO DE CASOS DE USO

O diagrama da Figura 1 ilustra os principais autores do sistema, sendo eles o usuário comum e o administrador. O usuário comum pode autenticar-se no sistema e utilizar a ferramenta robótica. O administrador pode autenticar-se, e deverá cadastrar novos usuários no sistema e gerenciar suas informações, assim como também pode utilizar da ferramenta robótica.

Figura 1: casos de uso



Fonte: autoria própria

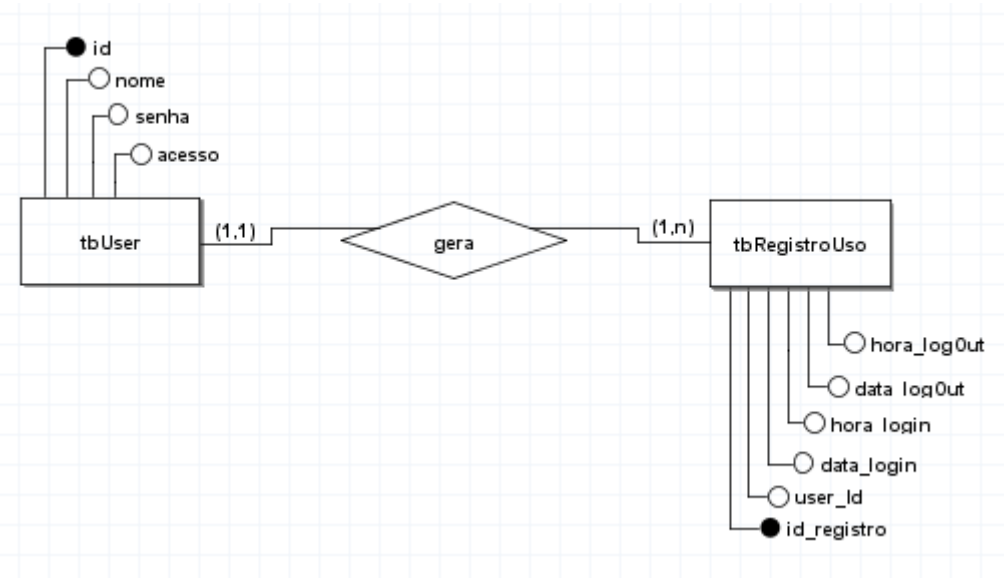
4.2. MODELAGEM CONCEITUAL DO BANCO DE DADOS

A Figura 2 ilustra o modelo conceitual do banco de dados do projeto. Nela, é possível identificar duas tabelas, sendo elas:

- tbUser guarda as principais informações dos usuários, como o id de identificação, o nome utilizado para login, a senha e seu nível de acesso (sendo 1 para administrador e o para usuário comum);
- tbRegistroUso:visa recolher informações do usuário o qual está usufruindo do sistema no momento presente como meio de controle de

ações com a ferramenta. Seus atributos são o ID de identificação do registro, o ID que identifica o usuário e as datas e horários de login e logout por parte do usuário no sistema. Suas cardinalidades descrevem que um usuário poderá ter vários registros de uso, enquanto cada registro de uso pertencerá a um único usuário.

Figura 2: modelo conceitual do banco de dados

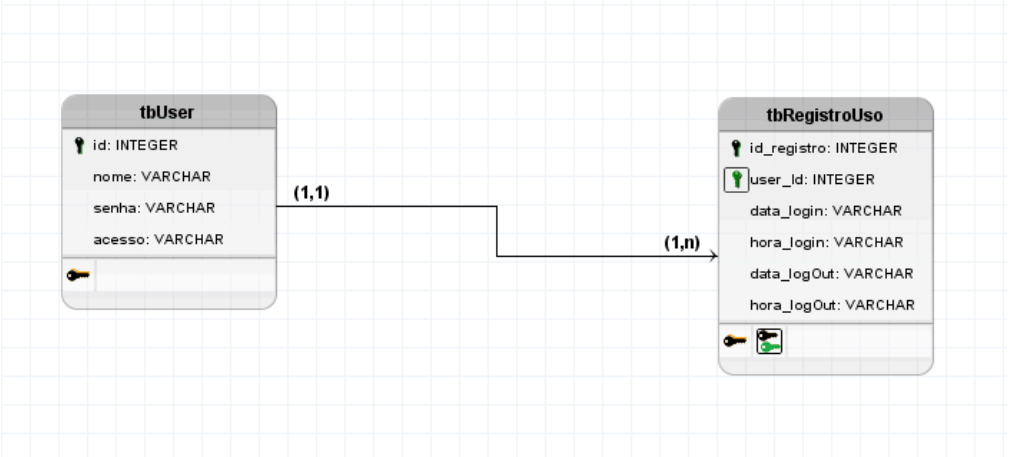


Fonte: autoria própria

4.3. MODELAGEM LÓGICA DO BANCO DE DADOS

A Figura 3 ilustra o modelo lógico do banco de dados, este modelo é utilizado para definir os tipos de dados presentes nas colunas das tabelas.

Figura 3: modelo lógico do banco de dados



Fonte: autoria própria

5. TECNOLOGIAS UTILIZADAS

- MediaPipe-hands: uma biblioteca do python que utiliza um modelo de inteligência artificial não generativa treinada especificamente para mapear partes do corpo humano e interpretar seus movimentos.
- OpenCV: uma biblioteca do python utilizada para rodar vídeos gravados ou em tempo real e oferecer dados para manipulação.
- DobotDLLType: uma biblioteca criada pela própria empresa Dobot, essa biblioteca possui as funções necessárias para os movimentos do braço robótico e a conexão dele como o hardware do computador.
- Numpy: biblioteca do python essencial para expressões matemáticas necessárias para o mapeamento e cálculo das landmarks do mediapipe.
- PyQt5: biblioteca do python necessária para construção de aplicativos windows, todas as telas que o sistema possui foram desenvolvidas com o aplicativo QT designer e sua biblioteca PyQt5.

Além das bibliotecas, há também o próprio braço robótico denominado Dobot Magician, desenvolvido pela empresa chinesa DOBOT ROBOTICS e o SGBD MySql escolhido para gerenciar o banco de dados do sistema. A escolha destas tecnologias se dá pela boa integração entre elas e também por suas funções únicas.

6. DESCRIÇÃO DO SISTEMA

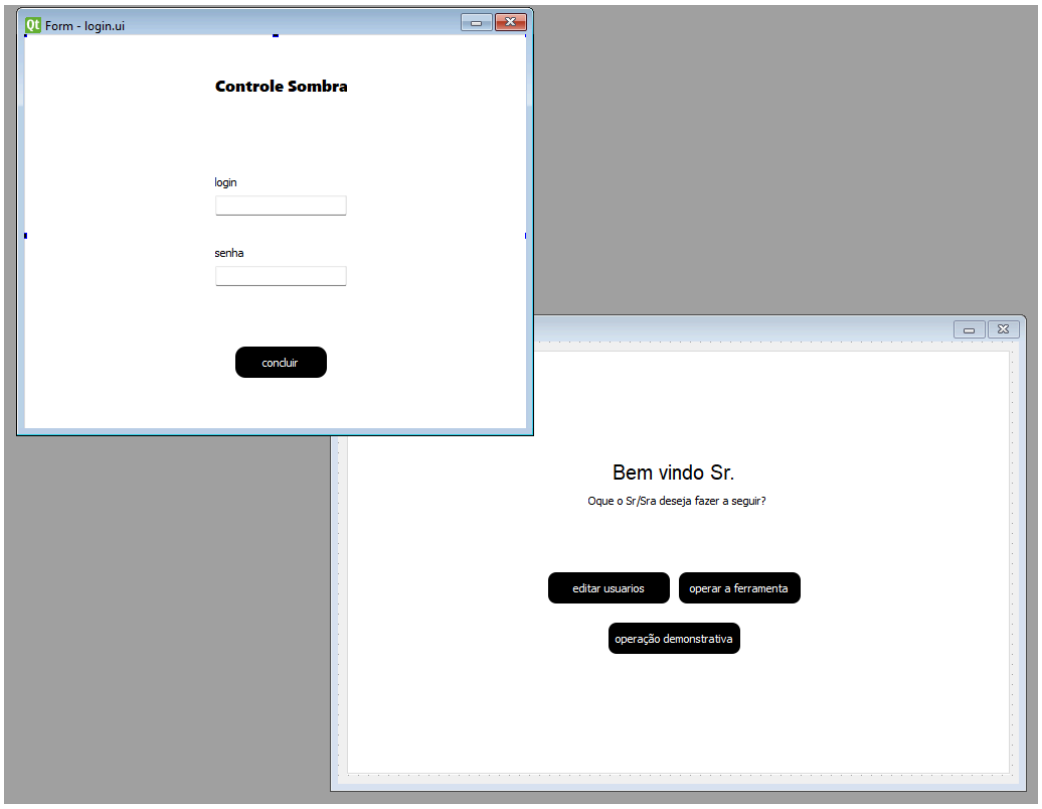
6.1 PRINCIPAIS TELAS

Após o login, o usuário é direcionado para a tela principal onde existem 3 escolhas: editar usuários, operar ferramentas e operação demonstrativa. Somente usuários com acesso administrativo podem editar usuários, restando apenas operar a ferramenta ou vídeo demonstrativo.

O Botão “operar ferramenta” inicializa o braço robótico e dá início ao controle por gestos.

Botão “operação demonstrativa” inicializa o braço robótico mas quem opera é um vídeo gravado, visando demonstrar como deve ser feito o uso do braço robótico, conforme a Figura 4.

Figura 4: tela de login e tela principal

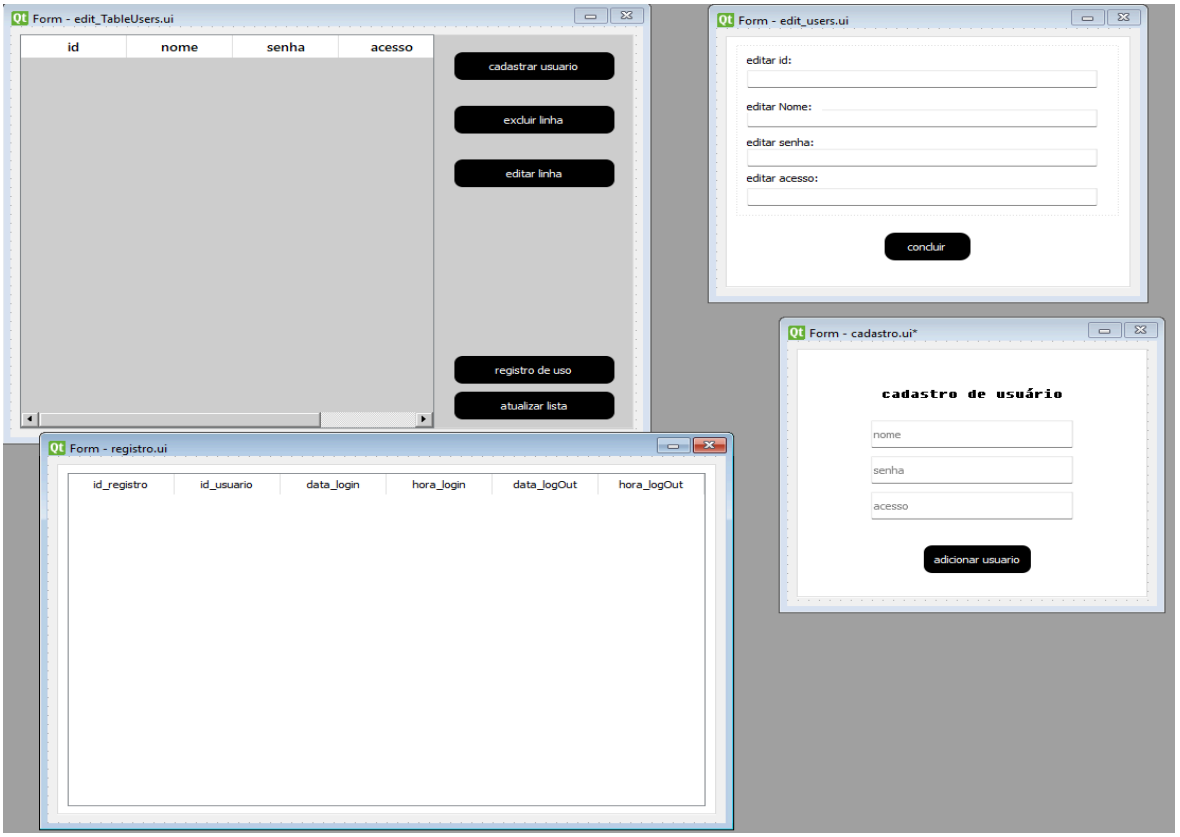


Fonte: autoria própria

A Figura 5 demonstra a tela de edição que possui uma tabela no lado esquerdo que exibe todos os usuários cadastrados no banco de dados e suas informações, à direita existem 5 botões com funções únicas.

1. **Botão cadastrar usuário:** abre uma tela contendo 3 campos a serem preenchidos por um administrador, o cadastro deve contar com o usuário a ser cadastrado estando presente.
2. **Botão excluir linha:** exclui a linha que estiver selecionada na tabela, naturalmente excluindo o perfil do usuário no sistema.
3. **Botão editar linha:** abrirá uma tela contendo as informações do usuário selecionado na tabela, informações essas que podem ser alteradas e salvas.
4. **Botão registro uso:** abrirá uma tela contendo uma tabela que descreve uma log com registros de login e logout de usuários no sistema, essa funcionalidade visa monitorar o uso do sistema registrando quem estava utilizando a ferramenta em determinada data e horário.
5. **Botão atualizar lista:** deve ser usado sempre que uma mudança for feita.

Figura 5: tela de edição geral de usuários e suas sub telas



Fonte: autoria própria

A Figura 6 ilustra o modelo de ferramenta robótica denominada Dobot Magician, desenvolvido pela empresa chinesa Dobot Robotics. Possui 3 juntas

principais: base, junta traseira e junta dianteira. Há também a última junta que rotaciona o módulo da ponta.

Figura 6: Dobot Magician

Fonte: https://www.dobot.nu/wp-content/uploads/2017/11/dobot-magician.jpg

6.2 - Back-end de movimento robótico

A Figura 7 mostra a inicialização das principais variáveis utilizadas pelas bibliotecas Open CV, Media Pipe-hands e DobotDLLtype. As primeiras variáveis são utilizadas para armazenar o desenho e posição das mãos no vídeo contido na variável “cap”, juntamente com uma função usada para testar a qualidade de frames

por segundo (FPS) da câmera usada. No bloco “variáveis de posição” estão as variáveis que armazenam as posições contidas no penúltimo frame carregado, assim como as variáveis que inicializam a sensibilidade de tolerância de movimento. No bloco “flags de controle” estão as variáveis de valor booleano que controlam se um movimento de junta está sendo feito ou não.

Figura 7: inicialização de variáveis principais

ControleSombra_robot.py > robot

```python
def robot(cam):

    import DobotDllType as dType
    import keyboard as kb
    import cv2
    import mediapipe as mp
    import numpy as np

    #desenhando as mãos
    mp_hands = mp.solutions.hands
    mp_drawing = mp.solutions.drawing_utils
    hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.7)

    cap = cv2.VideoCapture(cam)  #seleção de tipo de video

    fps = cap.get(cv2.CAP_PROP_FPS)
    if fps == 0: # Se for webcam (ou erro), define um padrão
        fps = 30
    # Calcula o tempo de espera (em milissegundos) por frame
    delay_ms = int(1000 / fps)
    print(f"Video rodando a {fps:.0f} FPS (delay de {delay_ms}ms por frame).")


    #\\\\\\\\\\\\\\\\\variaveis de posição\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    x_media_anterior = None
    z_media_anterior = None
    prev_size = None
    estado_garra_atual = None

    #sensibilidade do eixo x base
    sensi_x = 0.0053

    #sensibilidade do eixo z base
    sensi_z = 0.0068

    #sensibilidade do eixo y base
    sensi_y = 0.003



    #\\\\\\\\\\\\\\\\\flags de controle\\\\\\\\\\\\\\\\\\\\\\
    moving_y = False
    moving_x = False
    moving_z = False
    #\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
```

Fonte: autoria própria

A Figura 8 ilustra a definição de funções muito importantes para o funcionamento do código. A função “calcular_tamanho_mao()” é responsável por calcular a proximidade da mão referente a tela, isso é usado para determinar a posição do eixo Z da mão em relação a câmera. A função funciona executando a função matemática $Z = \sqrt{((x\square - x_a)^2 + (y\square - y_a)^2)} + \sqrt{((x_i - x\square)^2 + (y_i - y\square)^2)}$, que funciona traçando uma cruz em ângulos de X e Y na base da mão e comparando a distância entre as extremidades, caso Z aumente significa que a mão está mais próxima pois suas extremidades estão mais distantes, caso as extremidades estejam mais próximas significa que a mão está se afastando. A função “sensi_ajust()” é responsável por calcular e redefinir a sensibilidade de tolerância dos movimentos com base na distância da mão em relação à câmera. Isso é necessário pois quando a mão está perto da câmera qualquer movimento é muito significativo, e isso exige que a tolerância de movimento se adapte.

Figura 8: função de proximidade e variáveis de controle

```python
#\\\\\\\\\\função de calculo de tamanho da mao\\\\\\\\\\
def calcular_tamanho_mao(landmarks):
    pulso = landmarks[0]
    palma = landmarks[9]
    indicador_base = landmarks[5]
    mindinho_base = landmarks[17]

    # distância pulso-palma (profundidade)
    dist_pulso_palma = np.sqrt((pulso.x - palma.x)**2 + (pulso.y - palma.y)**2)

    # largura da mão (altura perpendicular)
    largura_mao = np.sqrt((indicador_base.x - mindinho_base.x)**2 + (indicador_base.y - mindinho_base.y)**2)

    # métrica final
    return dist_pulso_palma + largura_mao

#\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

#\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\def para ajuste de sensibilidade adaptativa por tamanho\\\\\\\\\\\\\\\\\\\\\\\\
def sensi_ajust(tamanho_atual, sensi_base):

    tamanho_referencia=0.17

    # Evita divisão por zero
    if tamanho_referencia == 0:
        return sensi_base

    # Calcula a proporção
    proporcao = tamanho_atual / tamanho_referencia

    # O novo threshold escala junto com o tamanho da mão
    nova_sensi = sensi_base * proporcao

    return nova_sensi
```

fonte: autoria própria

A Figura 9 mostra a execução das funções da biblioteca DobotDll que conectam e inicializam o braço robótico ao computador. Uma vez obtendo sucesso no bloco “if” que testa se a conexão foi feita, o resto do código inteiro prossegue, caso contrário uma mensagem de erro é acionada e uma reinicialização é necessária.

Figura 9: Variáveis DobotDlltype

```
84
85      CON_STR = {
86          dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
87          dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
88          dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
89
90      #Carrega dll e o obj CDLL
91      api = dType.load()
92
93      #Conecta Dobot
94      state = dType.ConnectDobot(api, 'COM4', 115200)[0]
95      print("Connect status:",CON_STR[state])
96
97      if (state == dType.DobotConnect.DobotConnect_NoError):
98
99          #limpa linha de comando
100         dType.SetQueuedCmdClear(api)
101
102         #parametros de velocidade e aceleração das juntas
103
104         dType.SetJOGJointParams(api,
105                     j1Velocity=65,
106                     j1Acceleration=35,
107                     j2Velocity=100,
108                     j2Acceleration=100,
109                     j3Velocity=50,
110                     j3Acceleration=50,
111                     j4Velocity=35,
112                     j4Acceleration=35,
113                     isQueued=1
114                 )
115
116         #remove alarmes que impessam movimento
117         dType.ClearAllAlarmsState(api)
118
119         #Home
120         dType.SetHOMECmdEx(api, temp = 0, isQueued = 1)
121
122         # Inicia a execução da fila
123         dType.SetQueuedCmdStartExec(api)
124
125         print("Homing concluído!")
126
127         dType.ClearAllAlarmsState(api)
128
129
```

Fonte: autoria própria

A Figura 10 mostra o começo do bloco principal do código, este bloco é executado em looping até que seja cessado pelo usuário. O bloco inicia com a execução da função “isOpened” do OpenCV que testa se há algum vídeo atribuído para processamento, caso não haja vídeo após a execução da função “read()”, um break é acionado. Caso esteja tudo certo, o código prossegue para um bloco de tratamento de imagem, necessário para que a biblioteca MediaPipe leia os dados corretamente, neste bloco 1 frame de cada vez é tratado, inicialmente passa pela função “flip” que espelha o video na tela, logo após ele passa por “cvtColor()” responsável por definir a dinâmica de coloração do frame para RGB. Após isso, a função “process()” processa o frame atual e desenha pontos chave na mão detectada, isso é denominado “landmarks” ao longo do código. Após isso, caso o bloco “if” verifique que há landmarks desenhadas, ele permite que o codigo prossiga, onde no bloco denominado “calculo de posições” estão as variáveis que armazenam a posição atual da mão referente a câmera, e logo após esta a execução da função que atualiza a sensibilidade das juntas de acordo com o tamanho atual da mão.

Figura 10: Começo do bloco principal

```
131  #\\\
```

A Figura 11 mostra o primeiro bloco de movimento, o de abrir e fechar a garra do robô. No princípio, o contador de dedos estendidos começa sempre em zero para impedir empilhamento indesejado no contador. O bloco “for” pega o identificador das landmarks correspondentes às pontas dos 4 dedos principais, excluindo o polegar e roda uma vez para cada, pegando as posições das landmarks das pontas dos dedos e as landmarks das bases dos dedos. Caso os valores de posição da ponta de um dedo seja menor que sua base, ele identifica um dedo fechado. Após isso, o código prossegue para os testes de estado, que vão verificar se a mão está fechada ou aberta. A mão só será considerada fechada caso os 4 dedos estejam fechados.

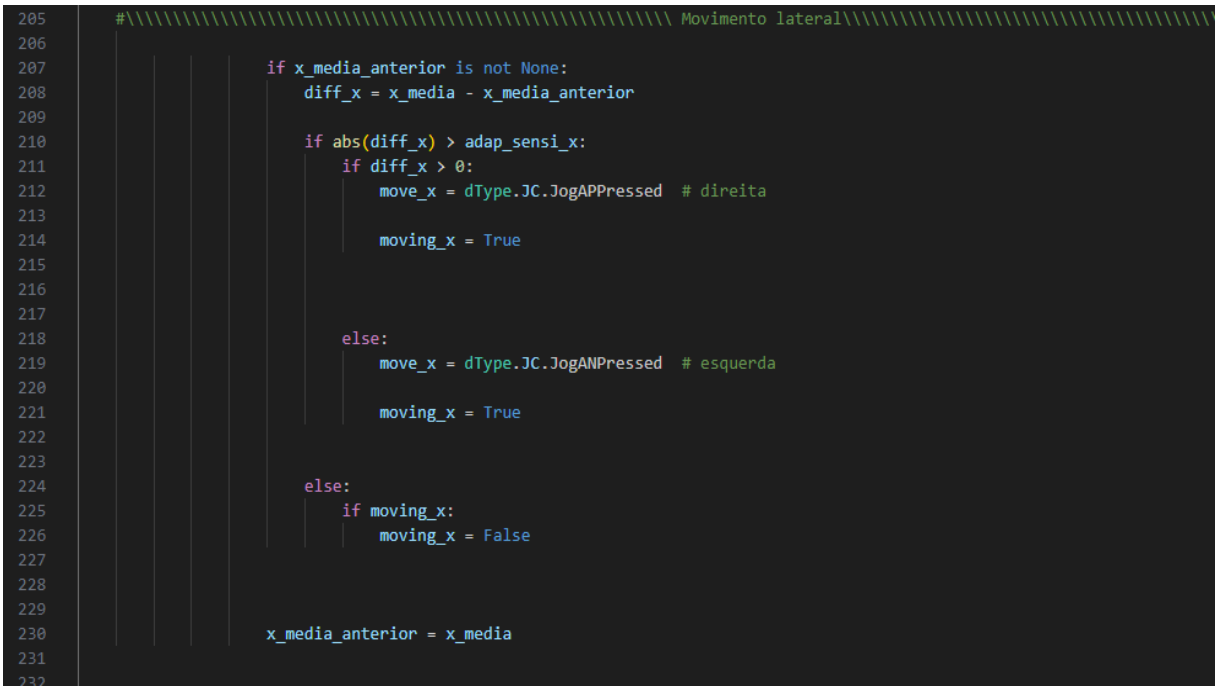
Figura 11: bloco de movimento abrir e fechar garra

```
                                #\\\\\\\\\\\\\\\\    movimentos    \\\\\\\\\\\\\\\\\\\

#\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ Fechar e abrir garra\\\\\\\\\\\\\\\\\\\
```

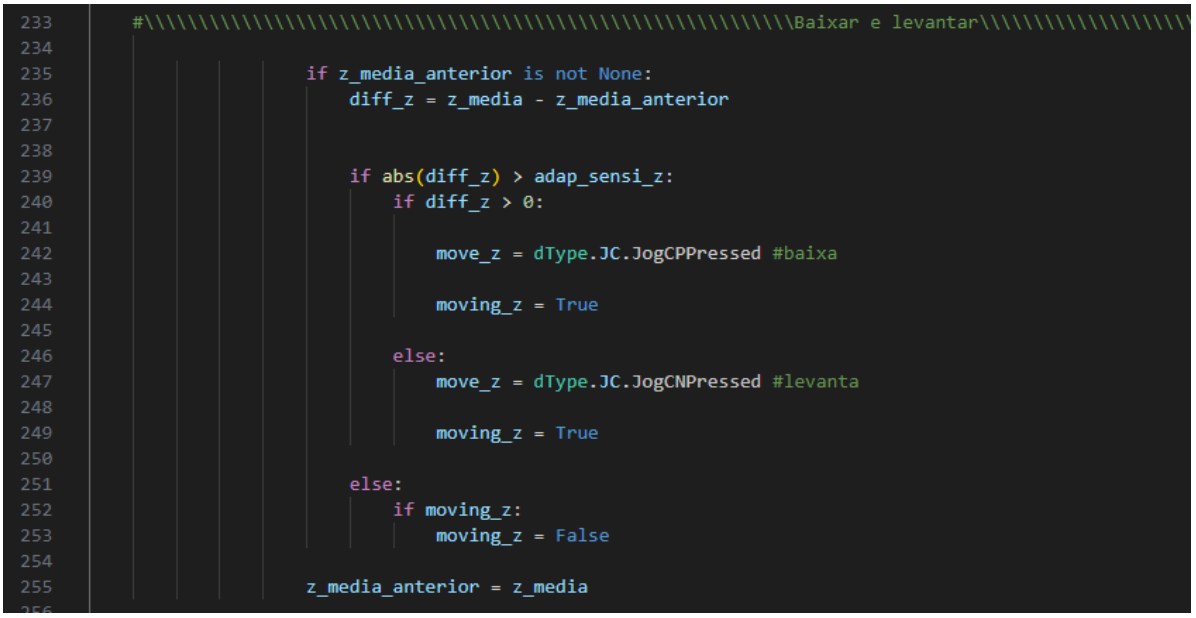
fazendo um comparativo da posição do frame anterior com o frame atual e subtrai os mesmos para obter um valor de diferença. Caso essa diferença seja maior que a sensibilidade de tolerância, o bloco verifica se o resultado é positivo ou negativo, movendo respectivamente a direita ou à esquerda e acionando a flag de controle lateral.

Figura 12: movimento lateral



```
205    #\\\
```

Figura 13: movimento vertical



```
#\\\
```

A figura 15 mostra o bloco do código que diz ao robô o que fazer. Caso uma flag de movimento esteja ativa, o eixo correspondente sera ativado, respeitando uma hierarquia de prioridade, sendo o movimento lateral no topo, movimento vertical no meio, e movimento de frente e trás com menos prioridade, é importante perceber que o movimento de frente e trás só será acionado enquanto houver um único dedo levantado na mão do usuário, isso é necessário pois a percepção de micro-movimentos da mão acabava gerando conflitos entre as juntas. Isso se dá pela grande diferença nas sensibilidades de cada junta, sendo necessário um movimento de junta por vez dado a uma limitação do firmware do robô. Os nomes de variáveis podem ser confusos, pois enquanto no mapa euclidiano comumente usado o movimento Z é referente a profundidade, para o hardware do robô o eixo Z é a junta de movimento vertical, por isso a flag “moving_z” controla o movimento vertical, e não o de trás e frente.

Figura 15: controlador de movimento

```
            else:
                # Se não há mão, desliga tudo
                moving_x = False
                moving_y = False
                moving_z = False
                dType.SetJOGCmd(api, 1, dType.JC.JogIdle, 0)

            # ENVIO DE COMANDOS (COM PRIORIDADE)
            # Essa seção decide o que fazer com base nas flags de controle

            if moving_x and dedos_estendidos != 1:
                # Prioridade 1: Movimento lateral
                dType.SetJOGCmd(api, 1, move_x, 0)

            elif moving_z and dedos_estendidos != 1:
                # Prioridade 2: Movimento vertical
                dType.SetJOGCmd(api, 1, move_z, 0)

            elif moving_y and dedos_estendidos == 1:
                # Prioridade 3: Movimento frontal e traseiro
                dType.SetJOGCmd(api, 1, move_y, 0)

            else:
                # Se nenhuma flag estiver ativa, para todas as juntas
                dType.SetJOGCmd(api, 1, dType.JC.JogIdle, 0)

            cv2.imshow('Detecção', frame)
            if cv2.waitKey(delay_ms) & 0xFF == 27:
                break
```

Fonte: autoria própria

A figura 16 mostra a parte final do código, onde ao finalizar o looping que mantém a leitura de vídeo e processamento de movimentos, o código cai no bloco “finally” responsável por forçar as funções de finalização do robo, “SetJOGCmd” força qualquer junta a parar, “SetQueueCmdStopExec” finaliza qualquer execução interna que o robô deixou na fila de execução, “SetQueueCmdClear” limpa qualquer resquicio na fila de execução do roubo, “cap.release” limpa a variavel que contém o vídeo, “destroyAllWindows” fecha a janela aberta pelo OpenCV que exibe o vídeo que está em vigor, e por fim “DisconnectDobot” desconecta o robô do computador

Figura 16: fim do codigo

```
    finally:

        #Para a queue de comando
        dType.SetJOGCmd(api, 1, dType.JC.JogIdle, 0)
        dType.SetQueuedCmdStopExec(api)
        dType.SetQueuedCmdClear(api)

        cap.release()
        cv2.destroyAllWindows()

        #desconecta o robo
        dType.DisconnectDobot(api)
        print("fim de sessão")

else:
    print('algo deu errado com a conexão dobot...')
    dType.DisconnectDobot(api)
    print("fim de sessão")
```

Fonte: autoria própria

7. CONSIDERAÇÕES FINAIS

Este trabalho trouxe muitos desafios no decorrer de seu desenvolvimento, sendo um dos principais desafios a lógica complexa para interação das bibliotecas no movimento do braço Dobot, pois o fato do braço ser de origem chinesa e seu conteúdo na internet também ser desta mesma origem, foi uma grande dificuldade encontrada. Mas apesar das dificuldades, foi um trabalho satisfatório de desenvolver, visto o fascínio em tecnologia por parte do autor.

Embora o conteúdo do curso não focar em robótica, as aulas de lógica no decorrer do curso e o módulo de Qt designer foram extremamente cruciais no desenvolvimento do trabalho, possibilitando um resultado agradável do sistema. Um dos principais objetivos com a escolha desse tema era justamente fugir do padrão de sites web dos outros TCC 's, e desenvolver um sistema focado em robótica que fosse satisfatório.

Como ideias para desenvolvimentos futuros poderiam ser confeccionar o próprio braço mecânico utilizando Arduino e motores, deixando assim de depender das limitações do software Dobot e tornando o sistema mais adaptável a diferentes tipos de hardware.

Embora o presente trabalho tenha sido desenvolvido para validação inicial do Controle Sombra, o ambiente esperado para este sistema é o funcionamento em ambientes de risco, como um laboratório por exemplo, possibilitando o manuseio de químicos e materiais de risco biológico a uma distância segura para os usuários do Controle Sombra.

Concluindo, verifica-se que as metas para este trabalho foram alcançadas e espera-se que este trabalho possa inspirar outros alunos a utilizarem os materiais de robótica disponíveis no Campus para desenvolver trabalhos incríveis e ainda mais impressionantes no futuro.

8. REFERÊNCIAS

BRADSKI, Gary; KAHLER, Adrian. *OpenCV Documentation*. OpenCV.org, 2025. Disponível em: https://opencv.org. Acesso em: 21 nov . 2025.

DOBOT. *Dobot SDK and API Documentation*. Dobot Robotics, 2025. Disponível em: https://portal.dobot-robots.com/home. Acesso em: 21 nov . 2025.

GOOGLE. *MediaPipe Hands: High-fidelity hand tracking solution*. MediaPipe Documentation, 2025. Disponível em: https://developers.google.com/mediapipe/. Acesso em: 21 nov . 2025.

HINZ, V. T. Materiais da disciplina Linguagem de Programação III. Ambiente virtual: Moodle, Instituto Federal Sul-rio-grandense, Pelotas, 2025. Disponível em: https://apnp.ifsul.edu.br. Acesso em: 21 nov. 2025.

LOPES, J. L. B Materiais da disciplina Banco de Dados II. Ambiente virtual: Moodle, instituto Federal Sul-rio-grandense, Pelotas 2025. Disponível em: https://apnp.ifsul.edu.br. Acesso em: 21 nov. 2025

NUMPY DEVELOPMENT TEAM. *NumPy User Guide*. NumPy.org, 2025. Disponível em: https://numpy.org/doc/. Acesso em: 21 nov . 2025.

QT GROUP. *Qt Designer Manual*. Qt Documentation, 2025. Disponível em: https://doc.qt.io/. Acesso em: 21 nov . 2025.

APENDICE I - INSTRUÇÔES PARA CRIAÇÂO DA BASE DE DADOS

```
CREATE TABLE tbUser (
    id INTEGER PRIMARY KEY,
    nome CHARACTER,
    senha CHARACTER,
    acesso CHARACTER
);

CREATE TABLE tbRegistroUso (
    id_registro INTEGER PRIMARY KEY,
    user_Id INTEGER,
    data_login CHARACTER,
    hora_login CHARACTER,
    data_logOut CHARACTER,
    hora_logOut CHARACTER
);
```