Matheus dos Anjos Inoue

Controle remoto de uma câmera pan-tilt a partir do reconhecimento de poses faciais

Santo André - SP

2018

Matheus dos Anjos Inoue

Controle remoto de uma câmera pan-tilt a partir do reconhecimento de poses faciais

Monografia de conclusão de curso apresentada à Universidade Federal do ABC para obtenção do título de Engenheiro de Informação.

Universidade Federal do ABC – UFABC

Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas

Programa de Graduação

Orientador: Prof° Dr Celso Setsuo Kurashima

Santo André - SP

2018

Resumo

Apesar da grande evolução nas últimas décadas, sistemas de comunicação ainda são limitados em relação à imersividade, e sistemas de telepresença ainda estão em desenvolvimento, e os sistemas existentes são controlados por controles manuais, que limitam a imersividade, comparado a sistemas de controle mais naturais, como gestos.

Este trabalho propõe um sistema de controle para dispositivos de telepresença, uma câmera pan-tilt no caso, que busca ser simples, mas com possibilidade dos mais variados usos, como videoconferências imersivas, videoaulas e sistemas de telepresença, permitindo um maior grau de imersividade do usuário. O sistema é baseado na captura de imagens do usuário local a partir de uma câmera, que irá reconhecer uma determinada pose facial. A detecção de uma determinada pose facial especifica irá mover remotamente uma segunda câmera, que é do tipo pan-tilt.

Palavras-chave: telepresença. interação humano-computador. visão computacional. aprendizado de máquina. processamento de imagem.

Abstract

Despite the great evolution in the last decades, communication systems are still limited in relation to immersiveness, and telepresence is a tecnology still in development, and existing systems are mainly controlled by manual controls, which limits the immersion, compared to more natural control systems such as gestures.

This work proposes a control system for telepresence devices, a pan-tilt camera in this project, that seeks to be simple, but with possibility of the most varied uses, such as immersive videoconferences, videoconferences and telepresence systems, allowing a greater degree of immersiveness of the user. It is based on capturing user images, which are used as input in programs that use image processing and machine learning techniques to detect the user current facial pose, the detection of a pose makes the program to send a control signal to move the remote pan-tilt camera to a position that is specific to the detected facial pose.

Keywords: telepresence. human–computer interaction (HCI). computer vision. machine learning. image processing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CLAHE	*Contrast Limited Adaptive Histogram Equalization*
CSV	*Comma-Separated Values*
GPIO	*General Purpose Input Output*
HCI	*Human–Computer Interaction*
IDE	*Integrated Development Environment*
I/O	*Input/Output*
JSON	*JavaScript Object Notation*
MATLAB	*MATrix LABoratory*
OpenCV	*Open Source Computer Vision Library*
Pan-Tilt	Descreve um dispositivo capaz de girar direcionalmente, em dois eixos, Pan (sentido horizontal) e Tilt (sentido vertical)
PWM	*Pulse-Width Modulation*
ReLU	*Rectified Linear Unit*
RGB	*Red, Green and Blue*
SGD	*Stochastic Gradient Descent*
TCP	*Transmission Control Protocol*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
Wi-Fi	*Wireless Fidelity*

Sumário

1 Introdução

No último século, se observou um salto de evolução nos sistemas de telecomunicação, com a criação e consolidação de tecnologias como o rádio, televisão, sistemas celulares, internet etc. Hoje é possível se comunicar com praticamente qualquer pessoa no mundo, seja mandando um *email*, ou através de uma chamada de voz, por exemplo. Isso significa que a interação entre humanos não depende mais da distância hoje em dia, já existem posições de trabalho remoto, onde uma empresa pode contratar empregados de localidades que não estão próximas de seus escritórios. Apesar disso, as tecnologias atuais de comunicação ainda são perceptivelmente mais pobres que a comunicação cara a cara, e isso é o objetivo principal de sistemas de comunicação imersiva, que utilizando tecnologias como processamento multimídia, visão computacional, computação gráfica, sensores e *displays* busca reduzir a distância entre pessoas, mas mantendo um nível de imersão próximo à comunicação cara a cara.

Uma característica de sistemas de comunicação imersivos, isto é, que dão a sensação de cercar o usuário de modo que ele se sinta totalmente envolvido, é a capacidade dos usuários de interagirem com o ambiente remoto, e poder verificar essas interação através dos seus sentidos. Por exemplo, conforme nós movemos no espaço, visualmente podemos ver diferentes perspectivas de um ambiente, ou a reação de outras pessoas, fazendo com que sejamos parte da cena, ou quando o cenário da comunicação é transmitido, como quando a intensidade dos sons são transmitidos dependendo da localidade do usuário, quando ele se mexe, ele tem a sensação que o ambiente reagiu a esse movimento. Essa interação bidirecional estimula a imersão, a sensação de estar em outro lugar, e sistemas de comunicação imersivos podem ser de diversos tipos, como os destinados a comunicação natural entre pessoas, sistemas destinados a troca de informação, até sistemas de telepresença [1].

Exemplos de sistemas imersivos de telepresença incluem os sistemas Cisco TelePresence, destinados para videoconferências, como visto na Figura 1, que utilizando equipamentos de alta qualidade, dão a sensação dos usuários compartilharem a mesma sala de teleconferência. Essa aplicação permite um grande nível de imersão, mas ele é destinado para uso de instituições, como corporações, não para um usuário. Atualmente o máximo disponível para usuários comuns são dispositivos como *laptops* e *smartphones*, que contém uma câmera embutida que pode ser usada em videoconferências, mas nada muito além disso. Para aumentar o nível de imersão, existem trabalhos utilizando sistemas de videoconferência 3D [2], com elevado custo operacional, e trabalhos que utilizam câmeras móveis com o seu controle feito através de gestos, ou movimentos da face do usuário remoto [3].

Figura 1 – Sistema Cisco TelePresence[1].

Outra tecnologia que está em desenvolvimento são os robôs de telepresença, que permitem a usuários participarem de atividades, como eventos, sem a necessidade de longas viagens. Esses sistemas usam técnicas avançadas de interação humano-computador (ou HCI, *Human–Computer Interaction*) para permitir que o usuário se sinta envolvido pelo ambiente que ele interage com o robô, através de câmeras, microfones, controles etc. A principal vantagem desses sistemas é a capacidade de usuários participarem remotamente em eventos, conferencias [4] e aulas em escolas e universidades. Atualmente existem diferentes tipos de robôs de telepresença, dos mais avançados, destinados a usos específicos, até os destinados a uso geral, como o robô PadBot P1 da TelepresenceRobots.com[2], que é controlado via aplicativo de um *smartphone*, visto na Figura 2.

Um dos usos mais interessantes de sistemas imersivos e de telepresença são para o caso de usuários que tem limitações de mobilidade devido a problemas de saúde, principalmente crianças em idade escolar que são diagnosticadas com doenças que impedem com que saiam das suas casas. Isso pode causar impactos no desenvolvimento dessas crianças, por estarem isoladas do ambiente escolar, que além do seu valor educacional, tem todo um valor social, de interação entre colegas. Estudos já foram realizados sobre o uso de sistemas imersivos de telepresença para crianças em idade escolar afetadas por doenças que os impedem assistir a aulas e o impacto social é notável [5].

[1] Ver: <https://www.cisco.com/c/en/us/products/collaboration-endpoints/ix5000-series/index.html>
[2] Ver: <https://telepresencerobots.com/>
[3] Ver: <https://telepresencerobots.com/robots/padbot-p1-inbot-technology>

Figura 2 – Robô de telepresença PadBot P1[3].

O robô de telepresença AV1 da NoIsolation[4], visto na Figura 3, foi feito com esse caso em mente, com suas funções baseadas em possíveis necessidades de crianças e adolescentes em idade escolar que não podem sair de suas casas por problemas de saúde, como a capacidade de assistir aulas na escola, mas também poder participar de atividades não escolares, como ir com os seus pais ao supermercado para fazer compras, ou participar de atividades sociais. Ele é controlado por aplicativos de dispositivos móveis, como *smartphones* e *tablets*.

Os sistemas de robôs de telepresença apresentados são eficientes em dar a capacidade de participar de eventos sociais remotos, mas o seu controle que depende do uso de aparelhos como *smartphones* não dá o maior grau de imersão, e pessoas com problemas em manipular esses dispositivos, como pessoas com deficiências físicas, não conseguem utilizar esses robôs. Existem estudos para utilizar sistemas de telepresença controlados por outros meios, seja por processamento de imagens do usuário, que faz algum movimento para controlar o robô, como movimentação do usuário no espaço [3], ou capturando imagens do olhar do usuário, para detectar aonde ele está olhando, até estudos que utilizam sinais vindo diretamente do cérebro, através de sensores para o controle dos robôs [6].

Nesse trabalho é proposto criar um protótipo para um sistema de telepresença simples e de baixo custo, que utiliza uma câmera pan-tilt que é controlada remotamente utilizando a estimação de poses faciais como sinal de controle, proveniente de uma câmera

[4] Ver: <https://www.noisolation.com/us/av1/>
[5] Ver: <https://www.noisolation.com/us/av1/features/>

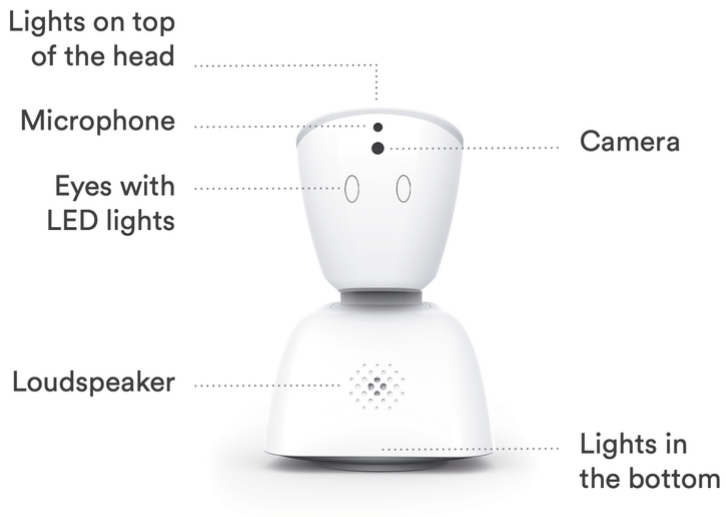


Figura 3 – Robô de telepresença AV1, destinado a crianças em idade escolar[5].

fixa que utiliza técnicas de processamento de imagem e aprendizado de máquina para a detecção e classificação das poses faciais. Esse protótipo poderá ser utilizado para exibição e ou gravação de videoconferências, vídeo-aulas, permitindo um maior grau de imersão do usuário remoto. O sistema é composto pela câmera pan-tilt, programas auxiliares para comunicação entre dispositivos, como os motores utilizados para mover a câmera e computadores do sistema, e o programa que faz o controle por estimação de pose facial do usuário.

2 Análise teórica

Neste capítulo, são discutidos o Estado da Arte das tecnologias relacionadas ao trabalho, bem como as suas especificações e técnicas alternativas que poderiam ser utilizadas. Também é feito uma análise da viabilidade do trabalho, de um ponto e vista técnico e econômico.

2.1 Estado da Arte

Nesta seção serão discutidas as principais técnicas e os avanços mais recentes dos principais componentes do sistema, bem como uma breve introdução aos tópicos mais importantes:

- Técnicas de processamento de imagens e visão computacional;
- Aprendizado de máquina, ou *Machine Learning*;
- Técnicas para detecção de faces em imagens;
- *Hardware* do sistema.

2.1.1 O que é Visão Computacional e quais são algumas de suas técnicas?

Uma imagem digital pode ser considerada como uma matriz bidimensional composta por *pixels*. *Pixels* são uma representação numérica da cor da imagem em um ponto particular. Para uma imagem em escala de cinza, basta uma matriz bidimensional com um valor por *pixel* com os valores para representar a imagem, com os valores de cada *pixel* em um intervalo de 0 até o valor máximo possível, representando a escala de preto (brilho mínimo) até branco (brilho máximo). Mas para imagens coloridas, uma matriz com apenas um valor por *pixel* não possui informação suficiente. Assim, para estes casos, a imagem digital é representada por três matrizes, cada matriz com os valores de algum componente de cor da imagem, em um modelos de cor [7], onde os componentes de cada matriz são chamados de canais. O modelo de cor mais utilizado em imagens digitais é o modelo RGB, que utiliza 3 canais, cada uma representando o valor para o componente de cada cor do modelo: vermelho, verde e azul. Outros modelos incluem: HSV (*Hue, Saturation, Value*), YCbCr entre outros. A Figura 4 mostra como uma imagem é representada por *pixels*.

A área de Processamento de Imagem é o estudo de extração e melhoria de informação de imagens. Técnicas de processamento de imagem incluem redimensionamento de imagem,

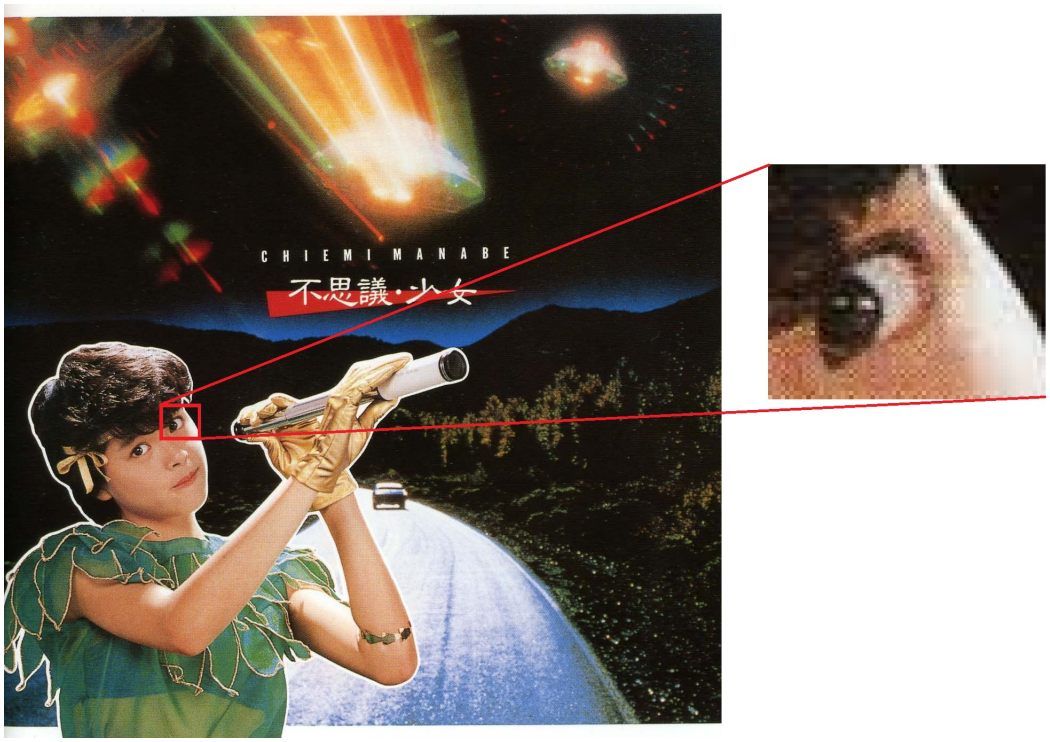


Figura 4 – Representação de imagem.

Fonte: O autor.

filtragem, detecção de borda, limiarização, entre outros. Essas técnicas são de nível baixo, sendo usadas mais para um pré-processamento para melhorar uma imagem [8].

Técnicas de nível mais alto estão no campo da Visão Computacional, onde o objetivo é extrair informações de imagens, em ordem para um sistema fazer decisões baseadas em o que o sistema entende da imagem detectada. Na maioria dos casos, a imagem é pré-processada por técnicas de Processamento de Imagem para então serem utilizadas métodos de Visão Computacional. Técnicas dessa área incluem: Fotografia Computacional, aplicações estereoscópicas, reconhecimento de objetos entre outros, conforme mostrado na Figura 5.

2.1.2 Técnicas de *Machine Learning*

Machine Learning, ou aprendizado de máquina, é uma área da ciência da computação que estuda algoritmos que conseguem fazer reconhecimento de padrões. Algoritmos de *Machine Learning* são algoritmos capazes de aprender com dados. Esse aprendizado pode ser explicado como a capacidade de aprender com uma experiência E, com respeito a uma classe de tarefas T e uma medida de performance P, se a performance em T, medida por P melhora com E. É importante notar que existem uma grande variedade de tarefas, medidas de performance de um algoritmo e tipos diferentes de experiência (como o algoritmo pode

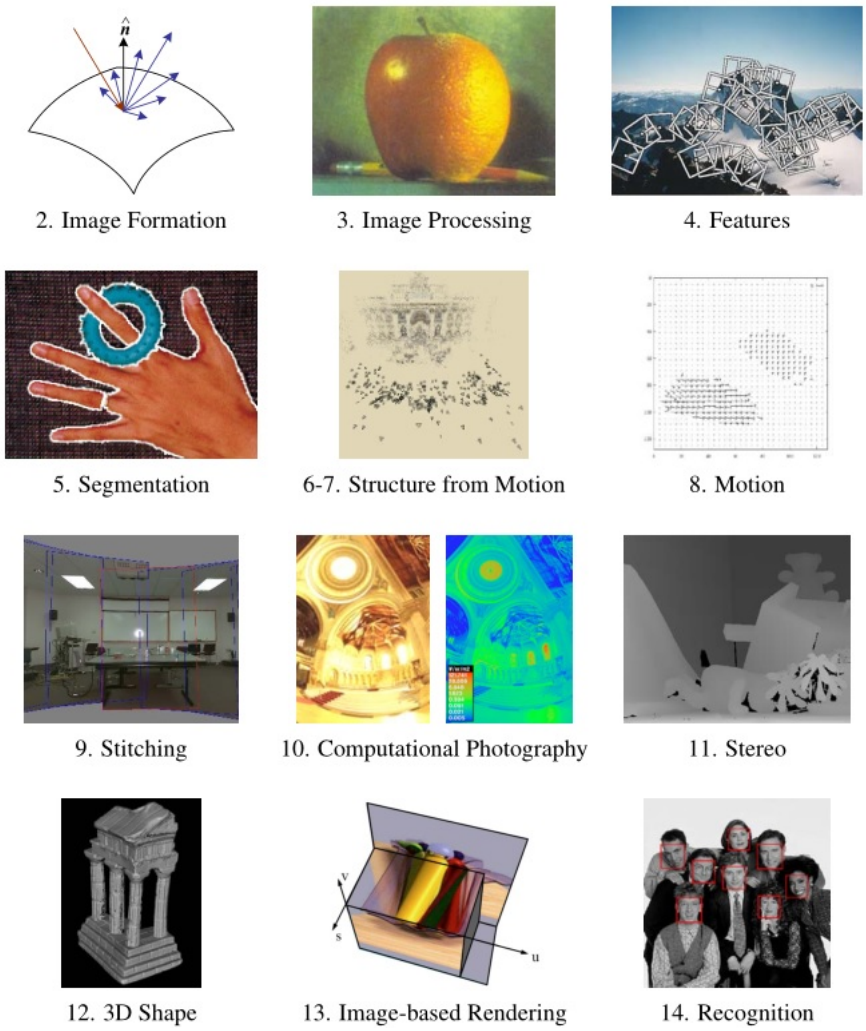


Figura 5 – Exemplos de técnicas de processamento de imagem e visão computacional. Fonte:[9].

aprender) [10].

Técnicas dessa área são atualmente usadas pelas mais diversas áreas e campos [8]. Tarefas que podem ser resolvidas por *Machine Learning* incluem:

- **Classificação**: Nessa tarefa, o programa de computador deve conseguir classificar em qual categoria k os dados de entrada pertencem. Para isso o algoritmo de aprendizado deve produzir uma função f tal que;

$$f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$$

  Onde o modelo deve atribuir um vetor representando os dados de entrada x a uma categoria identificada por um valor numérico y. Existem variantes de tarefas de classificação, como, por exemplo, a função f tem como resultado a distribuição de probabilidades entre todas as classes. Um exemplo de sistemas de classificação são

algoritmos de reconhecimento de objetos, onde a entrada é uma imagem de algum objeto e a saída é uma identificação de que categoria o objeto da imagem pertence.

- **Regressão**: Neste tipo de tarefa, o programa deve prever um valor numérico dado alguma entrada. Para resolver esse problema, o algoritmo de aprendizado deve construir uma função f tal que:

$$f : \mathbb{R}^n \to \mathbb{R}$$

  Esse tipo de tarefa é similar a classificação, exceto que o resultado final é diferente. Um exemplo de regressão são sistemas que fazem previsões de preços futuros de títulos financeiros.

- **Detecção de anomalias**: Neste tipo de tarefa, o objetivo do programa de computador faz uma varredura através de um intervalo de eventos ou objetos e identifica casos em que os dados aparecem ser estranhos ou atípicos. Um exemplo de um sistema de detecção de anomalias são programas para identificação de fraude de crédito, onde uma empresa de cartões de crédito analisa os costumes de compra do seu cliente e detecta casos de uso indevido do cartão, o que pode significar algo como roubo de cartão e, assim, para prevenir maiores problemas, a empresa pode bloquear preventivamente o cartão, evitando prejuízos.

- **Tradução de Máquina**: Em um sistema de tradução de máquina, os dados de entrada consistem em uma sequência de caracteres em uma língua, e o algoritmo deve converter em uma sequência de caracteres em outra língua. O principal uso desses sistemas são a tradução automática entre línguas naturais, como traduzir automaticamente um texto em Francês para Português.

O aprendizado dos algoritmos de *Machine Learning* depende da presença de dados de treinamento, de modo que algoritmos de *Machine Learning* podem ser divididos baseados na forma dos dados de treinamento:

- **Aprendizado não supervisionado**: possuem um *dataset* que contém várias características, mas sem nenhum valor de rótulo, onde o algoritmo aprende propriedades da do *dataset*, como a distribuição das probabilidades dos valores, e realiza tarefas como clusterização, que consiste em dividir um *dataset* em *clusters* de dados, com cada *cluster* tendo suas próprias características.

- **Aprendizado supervisionado**: Os dados do *dataset* possuem além de características, rótulos, ou valores alvo associados a cada exemplo do *dataset*. Por exemplo, um *dataset* de dígitos manuscritos pode ser composto por imagens dos dígitos manuscritos com um rótulo, como o nome do arquivo de imagem, explicando a qual categoria a imagem pertence, como uma imagem de um *"3"* em um arquivo *"3.jpg"*.

2.1.2.1 Redes Neurais

Uma das técnicas mais famosas de *Machine Learning* são as Redes Neurais Artificiais [11], um grupo de algoritmos que modela dados em um modelo que imita como os neurônios de um cérebro funcionam. Eles, como todo algoritmo de *Machine Learning*, utilizam grandes volumes de dados para executar o treinamento e aprendizado de tarefas.

No começo do desenvolvimento das redes neurais, foi considerado o uso de modelos de neurônios artificiais, como os *Perceptrons*, neurônios que recebem vários valores de entrada e produzem um valor binário de saída. No exemplo da Figura 6, temos como entrada de dados as entradas *x1*, *x2* e *x3*. Associada a cada entrada, temos pesos *p1*, *p2* e *p3*, que expressam a importância de cada entrada em respeito ao resultado final. A saída do neurônio, 0 ou 1 é determinada se o resultado da soma $\sum_j p_j x_j + b$, onde b é um valor de *bias*, é maior ou menor que um certo limite, L. Os pesos e o valor limite são números reais. Assim, o perceptron pode ser modelado como:

$$resultado = \begin{cases} 0, & \text{se } \sum_j p_j x_j + b <= L \\ 1, & \text{se } \sum_j p_j x_j + b > L \end{cases}$$

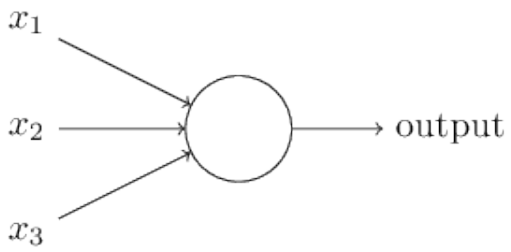


Figura 6 – Modelo de perceptron.

Fonte:[11].

O fato do resultado do perceptron ser binário, limita a sua capacidade de aprendizado, pois ele acaba tendo uma fronteira de decisão linear. Para remediar isso, além do perceptron, existem outros neurônios que utilizam outras funções de ativação, como a sigmoide, uma função contínua e derivável, elementos que são desejáveis em algoritmos de aprendizado, pois são bons em capturar pequenas mudanças de valores. Assim, um neurônio que utiliza a função de ativação sigmoide, tem a saída na forma:

$$resultado = \frac{1}{1 + exp(-\sum_j p_j x_j + L)}$$

Obviamente, dada a sua simples estrutura, um neurônio artificial como um perceptron não é um modelo para algo baseado em como um cérebro toma decisões, mas sim uma rede de neurônios, que pode desempenhar tarefas mais complexas , como exemplificado na

Figura 7. Nesse exemplo, temos a camada de entrada de dados seguida por duas camadas de neurônios, com 4 e 3 neurônios, respectivamente, as chamadas camadas ocultas, onde são executadas os cálculos para a camada de saída, onde temos o resultado final. Nas camadas ocultas, na primeira camada de neurônios consegue realizar decisões simples, adicionando pesos para as entradas, e a segunda camada recebe os valores da primeira camadas, adicionando novos pesos. Com isso, as decisões podem ser mais complexas que os resultados da primeira camada, aumentando o poder de decisão da rede neural.

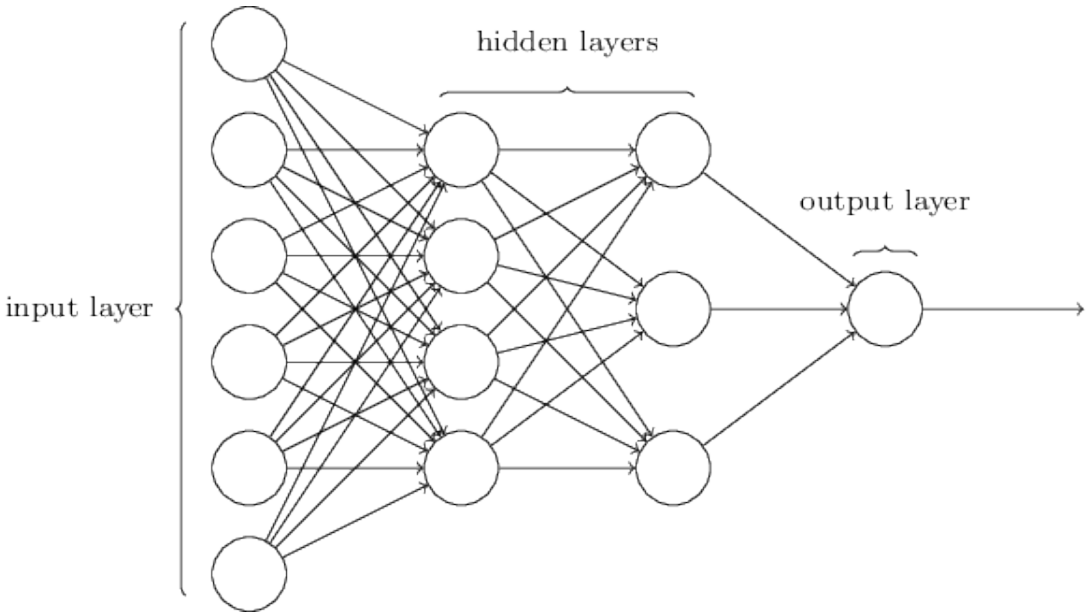


Figura 7 – Um exemplo de rede neural MLP (*Multilayer Perceptron*).

Fonte:[11].

Obviamente o cérebro de um ser vivo e uma máquina são bastante diferentes, mas os trabalhos feitos na área se preocupam mais em propor novas técnicas, ou usos para algoritmos de aprendizado do que emular um cérebro. Outras algoritmos de redes neurais incluem Redes Convolucionais, redes *Neuro-fuzzy* etc [12].

2.1.3 Técnicas para detecção de faces e sua posição no espaço

Considerando o objetivo do projeto, uma das principais partes para o funcionamento do sistema é a capacidade de detectar faces em uma imagem, ou seja, detectar o usuário da câmera fixa, para então fazer a detecção de pose para controlar a câmera pan-tilt. Atualmente existem várias técnicas de detecção de faces a partir de imagens, mas dois se destacam: o método utilizando classificadores de Haar [13] e o método utilizando árvores de regressão [14].

2.1.3.1 Uso de Cascatas de Classificadores de Haar

A detecção de objetos utilizando as cascatas de características de Haar, proposto por Viola e Jones [13] utiliza técnicas de *Machine Learning* onde uma cascata de características de um objeto é treinada, utilizando variadas amostras, positivas e negativas. O classificador treinado então consegue detectar o objeto em outras imagens. Para treinar esta classificador, características de Haar são utilizadas, onde cada característica é um único valor obtido subtraindo a soma dos *pixels* sob o retângulo branco da soma dos *pixels* sob o retângulo preto, conforme mostra a Figura 8.

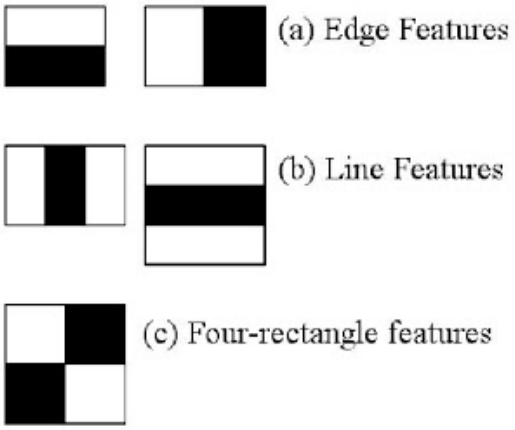


Figura 8 – *Knernels* de Haar.

Fonte:[15].

Como são consideradas inúmeras características para uma dada imagem (por exemplo, para uma face, temos a Figura 9), o tempo de processamento seria muito alto, mesmo para imagens de tamanho reduzido. Assim, foi proposto a técnica *AdaBoost*, onde aplicamos todos as características em todas as imagens de treinamento. Para cada característica, ele encontra o melhor limiar que irá classificar os objetos positivos e negativos. Assim, poderemos ter erros de classificação. Para corrigir estes erros, selecionamos as caraterísticas com mínima taxa de erro, o que significa que eles são as caraterísticas que melhor classificam as imagens entre o objeto e não-objeto. O classificador final é uma soma ponderada destes classificadores fracos. É chamado de fraco porque sozinho ele não consegue classificar uma imagem, mas, junto com outros, formam um classificador forte. Para um funcionamento ainda mais eficiente, utilizamos o conceito de Cascata de Classificadores, onde se agrupa as características em diferentes estágios de classificadores e estes são aplicados um a um, se uma amostra da imagem falha em um dos estágios, ela é descartada (pois o objeto não está nessa amostra da imagem), economizando tempo de processamento.

O uso de Cascatas de Haar já vem implementado em ferramentas para aplicações

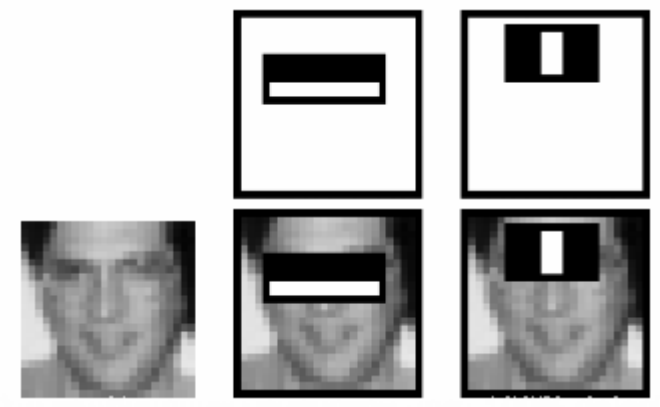


Figura 9 – Características de uma face.

Fonte:[15].

de Visão Computacional, com muitos classificadores já treinados, para detecção de faces, olhos, nariz etc. Existem trabalhos que propõem a comparação de diferentes classificadores para detecção de face, ou de detalhes, como os olhos [16].

Esse método foi muito influente quando foi originalmente proposto, e ainda é um dos métodos mais utilizados, mas ele apresenta certos problemas, como a alta taxa de detecção de falsos positivos, bem como a incapacidade de detectar faces que estão em posições oblíquas em relação a câmera. Além disso, ele é pesado computacionalmente, principalmente quando é utilizado mais de um classificador ao mesmo tempo, para amenizar o problema de detectar falsos positivos, que pode reduzir o desempenho consideravelmente. Assim, atualmente, esse método não é o mais recomendado para sistemas de detecção facial em tempo real.

2.1.3.2 Uso de árvores de regressão

A detecção de faces utilizando árvores de regressão, proposto por Kazemi-Sullivan [14], estimam posições de referência (*landmarks*) de faces a partir de um subconjunto de *pixels*, que alcança alto desempenho e qualidade de detecção. Um exemplo de *landmarks* faciais pode ser visto na Figura 10

Métodos baseados em árvores de regressão utilizam não a forma ou aparência como um todo, mas sim estudam as correlações entre características de uma imagem para inferir alguma forma facial. Esse métodos aprendem diretamente de uma função de regressão, como:

$$M : \phi(imagem) \rightarrow s \in \mathbb{R}^{2N}$$

Onde M é o modelo, $\phi(imagem)$ é a função que extrai as características de uma imagem e s é a forma facial resultante. Assim, essa técnica utiliza árvores de regressão, onde um conjunto de N regressores trabalham estágio a estágio para inferir a forma e

Figura 10 – Exemplo de imagens com anotações de *landmarks* faciais.

Fonte: Adaptado de [17].

posição dos *landmarks* faciais. Considerando que $x_i \in \mathbb{R}^2$ sejam as coordenadas x, y de i-ésimo *landmark* facial de uma imagem I. Assim, o vetor $S = (x_1^T, x_2^T, \ldots, x_p^T) \in \mathbb{R}^{2p}$ denota as coordenadas de todos os *landmarks* faciais p em I e cada regressor $r_t(.,.)$ na árvore de regressão prevê um vetor de atualização de uma imagem, e $\widehat{S}^{(t)}$, que denota a estimativa atual pode ser atualizado como:

$$\widehat{S}^{(t+1)} = \widehat{S}^{(t)} + r_t(I, \widehat{S}^{(t)})$$

É importante notar que r_t faz as suas predições a partir dos valores de *pixels* computados a partir da imagem I e indexados em relação à forma da estimativa $\widehat{S}^{(t)}$ atual. A Figura 11 demonstra como o processo de atualização de previsão ocorre.

Este processo de detecção de faces apresenta desempenho superior à detecção usando classificadores de Haar, pois além das detecções não serem dependentes da face estar alinhada com a câmera, devido a regressão aos *landmarks* permitir uma certa flexibilidade para detecção, as detecções apresentam uma taxa menor de detectar falsos positivos. Por fim, esse método tem tempo de execução menor que o método de Viola-Jones, alcançando tempos de até $1\mu s$, de modo que esse método é o mais desejável para sistemas de detecção de faces atualmente. Outro ponto interessante é que além do método retornar a detecção

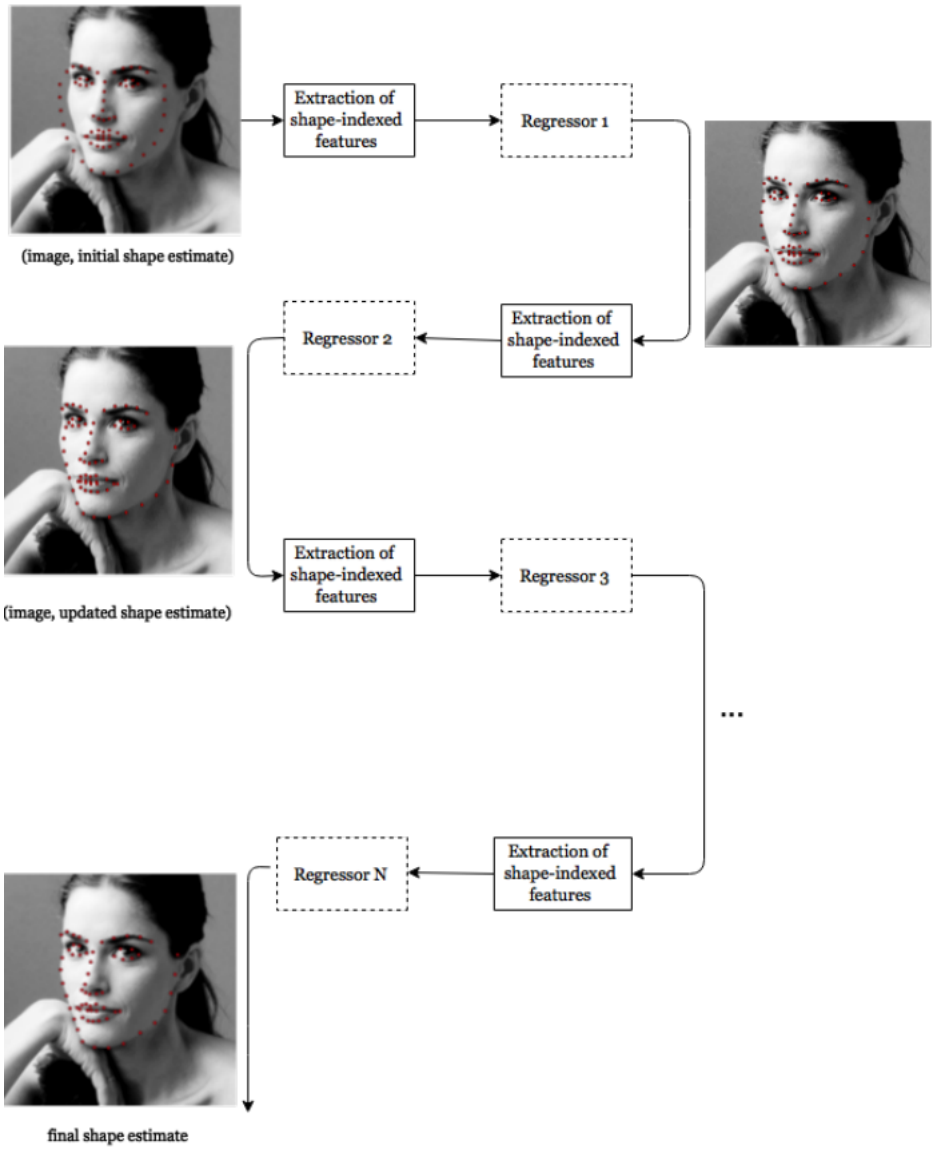


Figura 11 – Diagrama de como uma regressão em árvore funciona.

Fonte:[8].

de faces, ele retorna também as coordenadas de cada *landmark* facial detectado, o que pode ser interessante para outros usos. Um exemplo de detecção de face pode ser visto na Figura 12.

2.1.4 Hardware existentes

Uma câmera pan-tilt é composta pela câmera e um suporte movido por motores nos eixos de pan e tilt, que são comuns em projetos de robótica que utilizam componentes eletrônicos. Outro componente é o microcontrolador, que recebe remotamente os comandos da parte de detecção de pose facial e os transmite para os motores moverem a câmera. Um microcontrolador muito utilizado para prototipagem de sistemas é a plataforma Arduino

Figura 12 – Exemplo de detecção de face.

Fonte:[14].

[18], que permite o controle de motores e a comunicação entre diferentes sistemas, através da porta USB.

3 Metodologia e concepção do projeto

Neste capítulo é discutido a metodologia e desenvolvimento do sistema, partindo das especificações e viabilidade até em que partes o projeto foi dividido, e como estas partes trabalham juntas para que o sistema funcione como um todo. Primeiramente descrevemos o projeto de forma ampla, e então partimos para uma melhor descrição de cada componente deste sistema.

3.1 Especificação dos requisitos do projeto

O projeto final consistirá de uma câmera pan-tilt, que é controlada remotamente através de um usuário remoto. O controle se baseia em detectar a pose facial do usuário, que corresponde a alguma posição desejada para a câmera pan-tilt. Assim, o sistema tem duas partes principais: A câmera pan-tilt e um microcontrolador para fazer o movimento da câmera, e o programa que é utilizado pelo usuário remoto, que extraí o comando a ser mandando para o controlador da câmera através da detecção de poses faciais.

Esse comando por pose facial controla dois servomotores, cada um responsável pelo movimento da câmera pan-tilt em um eixo: O eixo *pan*, horizontal, e o *tilt*, vertical, ou seja, o controle da câmera se dá em dois eixos do espaço. Para fazer esse controle por pose facial, é necessário também ter uma câmera fixa para capturar a sequência de frames a serem processados, bem como um computador capaz de rodar o algoritmo de processamento de imagem. Assim, o sistema é composto dos seguintes itens:

- Computador com capacidade para processamento de vídeo.
- Computador com capacidade para se comunicar com um microcontrolador.
- Microcontrolador que possa receber comandos de um computador.
- 2 servomotores.
- Câmera a ser colocada no suporte móvel.
- Câmera fixa.

3.2 Alternativas de Soluções

Existem várias alternativas, tanto quanto a diferentes métodos para o controle da câmera, como detectar outros gestos, ou utilizar diferentes tipos de câmeras, como usar

uma câmera *fisheye*, de largo ângulo visual, com os movimentos de pan-tilt sendo feitos via software [3].

A câmera pan-tilt pode ser controlada de várias maneiras, como com o controle sendo a posição da face no espaço [3], por gestos com as mãos ou utilizando gestos corporais [19]. Um sistema similar ao proposto, mas que utiliza técnicas de controle e tipo de câmera diferentes, pode ser visto na Figura 13, onde a câmera utilizada não é pan-tilt, mas sim uma câmera *fisheye* que muda o seu foco conforme a posição da face detectada no espaço, ou seja, para mover o foco da câmera é necessário o usuário se mover pelo espaço, o que pode não ser o mais confortável.

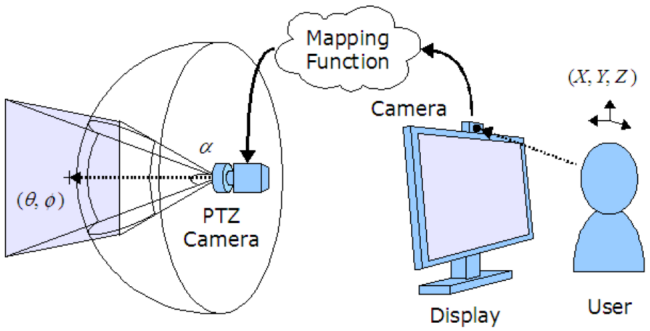


Figura 13 – Sistema similar ao proposto.

Fonte: [3].

Também pode-se considerar o uso de um arranjo de câmeras para auxiliar na aquisição de informações sobre a posição do usuário no espaço, como um arranjo de câmeras em estéreo, onde se obtêm informações da profundidade. Em ambos os casos existe um aumento de custo, econômico e computacional.

Para o controle de todo o conjunto, também pode ser utilizado diferentes micro-controladores como o LaunchPad[1], o Teensy[2], o brasileiro Blackboard[3] e inúmeros outros, pois como o sistema requer apenas o controle de dois motores, a complexidade nesta parte não é alta.

Considerando produtos já existentes e disponíveis no mercado, temos produtos para uso pessoal ou de instituições com ideias similares ao projeto proposto:

- Suportes para câmeras que se movem automaticamente estão disponíveis no mercado para o usuário comum. Algumas baseiam o movimento em informações provenientes de sensores de localização, chamados de "tags". O suporte usa a informação obtida dos sensores e então estima onde o usuário está e aciona a câmera montada para a

[1] Ver:<http://www.ti.com/lsds/ti/tools-software/launchpads/overview/overview.page>
[2] Ver :<https://www.pjrc.com/teensy/>
[3] Ver:<https://www.robocore.net/loja/produtos/arduino-blackboard.html>

(a) Câmera de segurança pan-tilt, controlada de modo convencional, por meio de controle.

(b) Suporte que movimenta a câmera automaticamente

Figura 14 – Exemplos de produtos similares[7].

captura de imagens. Exemplos desses suportes incluem: STOPSHOT[4] e a Move 'n See[5].

- Câmeras controladas automaticamente para uso em vídeo-aulas, como a câmera *Auto Tracker*[6] da 1beyond. Ela faz gravações de aulas rastreando a movimentação do professor, com o uso de uma câmera extra de ângulo largo mais uma câmera que é utilizada para a captura do vídeo. A câmera de ângulo largo é responsável pela detecção dos objetos de interesse que movimentará a câmera que grava o vídeo. Um dos pontos fracos de usar essa técnica (câmera com ângulo largo) é o encarecimento do produto final, sendo que esses produtos são destinados para uso de instituições.

- Câmeras de segurança pan-tilt, que se movem nos eixos de pan e tilt, mas são controladas de forma convencional pelo usuário, utilizando um controle, como um aplicativo de celular.

3.3 Análise de Viabilidade do Projeto

A análise de viabilidade do sistema foi dividida em duas partes:

- A **viabilidade técnica**, que consiste na real possibilidade do protótipo funcionar de maneira satisfatória, pois o sistema pode ser muito complexo para funcionamento correto, ou é pesado computacionalmente, o que gera um sistema impróprio para tempo-real, como vídeo-conferências.

[4] Ver <https://shop.soloshot.com/>
[5] Ver <http://www.movensee.com/>
[6] Ver <http://1beyond.com/autotracker>
[7] Ver: <https://www.swann.com/us/swwhd-ptcam> e <https://shop.movensee.com/en/pixio-robot-cameraman-bundles/18-pixio-robot-cameraman.html>

- A **viabilidade econômica**, onde é verificado se o projeto tem um custo que é interessante considerando os potenciais usuários do sistema.

3.3.1 Viabilidade Técnica

Analisando o estado da arte existente para a parte de processamento de imagem, é visto que para as técnicas propostas já existem artigos que mostram o seu uso, com desempenho avaliado como suficiente para sistemas em tempo-real. O programa feito para esse projeto considerou técnicas que fossem mais econômicas computacionalmente, e alcançou números muito bons para sistemas em tempo-real, conseguindo alcançar cerca de 30 *frames* por segundo, superior ao idealizado inicialmente, 10 *frames* por segundo.

Além disso, para a parte do sistema que recebe os comandos para controlar o suporte pan-tilt, a viabilidade existe, principalmente por usarmos uma plataforma estabelecida, com suporte a diversos componentes, além dos que serão utilizados no sistema, a plataforma Arduino, que já tem compatibilidade nativa para trabalhar com os servo-motores.

3.3.2 Viabilidade Econômica

Para a análise da viabilidade econômica do projeto, foi feito uma estimativa do preço de cada componente que o sistema utiliza, e considerando o que foi adquirido para a montagem do sistema, foi montada a Tabela 1 com todos os itens considerados no projeto:

Tabela 1 – Cotação e custo real das peças.

Componente	**Cotação (R$)**	**Preço real (R$)**
Arduino UNO	120,00	119,00
2 Servos MG995	50,00	30,00
Suporte Pan/Tilt	25,00	24,50
Webcam para suporte	100,00[8]	25,00[9]
Câmera fixa	100,00	-
TOTAL	**395,00**	**198,50**

Fonte: o autor.

Assim, é estimado um custo total de aproximadamente 400 reais, um custo razoavelmente alto, mas como este projeto é um protótipo, existem componentes em que é possível reduzir custos para diminuir o custo total, isso sem nenhuma queda de qualidade, pois a ideia é utilizar componentes que cumpram suas funcionalidades especificamente, ao invés de componentes mais versáteis, que acabam sendo mais caros. O exemplo mais óbvio é substituir o micro-controlador Arduino, que foi escolhido devido a sua particular versatilidade e praticidade para prototipagem de produtos, por modelos mais baratos que

[8] Webcam cotada: Logitech C270 HD 720p 3MP
[9] Webcam comprada: Microsoft LifeCam VX-800 480p 1,3MP usada

compartilham a mesma arquitetura, ou utilizar um micro-controlador que tenha apenas as funcionalidades necessárias para o projeto. Também é possível economizar na câmera a ser escolhida para a câmera móvel, pois em tese o único requisito é a câmera ser leve e compacta, para ser montada no suporte pan-tilt.

Já o custo real no projeto foi de aproximadamente 200 reais, cerca de 50% menor que o custo estimado. Essa diferença foi devido principalmente ao fato de a câmera utilizada ser uma câmera usada, comprada especificamente para este projeto, que além do custo baixo, tinha qualidades como ser compacta e ter um formato exterior que permitiu a montagem no suporte pan-tilt sem maiores modificações externas e também porque foi considerado para a câmera fixa a *webcam* do *laptop* utilizado no desenvolvimento do projeto, e não algo que foi adquirido especialmente para o trabalho.

3.4 Descrição Sistêmica do Projeto

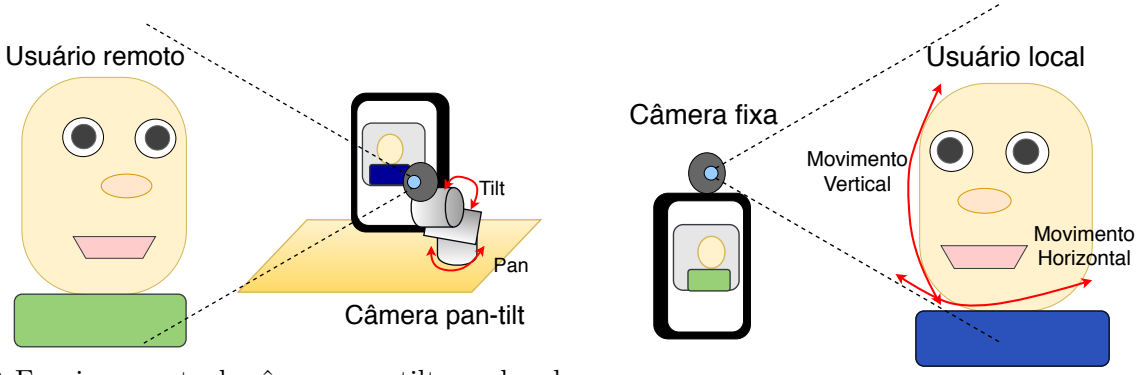
Considerando o objetivo do projeto, a criação de um protótipo de câmera pan-tilt, controlada por reconhecimento de poses faciais, ele pode ser dividido pelas seguintes partes, que serão explicadas com maior detalhe:

- Processamento de Imagens para detecção e reconhecimento de pose facial, totalmente em *software*.
- Câmera Pan-Tilt, o protótipo em si.
- Comunicação entre componentes, para o funcionamento do sistema como um todo.

O *hardware* do sistema, que consiste da câmera pan-tilt, controlada com o uso de um microcontrolador que recebe comandos do programa que está conectado diretamente à câmera pan-tilt, que por sua vez recebe os comandos do programa que faz o reconhecimento de pose facial, este remoto. Para maior clareza, a Figura 15 demonstra o funcionamento básico do sistema final, em um exemplo de uso em uma teleconferência, onde um usuário local, em azul, faz o controle da câmera pan-tilt remota por meio de movimentação da face:

A Figura 16 mostra como funciona resumidamente cada sistema separadamente, com a comunicação serial sendo feita pontualmente pela parte de software para controle da posição da câmera. É importante notar que existem três componentes principais na comunicação do sistema, com as suas principais atividades em destaque na Figura 16:

- Computador que roda o programa de controle, por detecção de pose facial, em roxo.
- Computador que está conectado diretamente à câmera pan-tilt, em verde, que recebe os comandos do outro computador.



(a) Funcionamento da câmera pan-tilt, recebendo os comandos remotamente.

(b) Funcionamento do controle da câmera remota via poses faciais.

Figura 15 – Utilização típica do sistema de controle da câmera pan-tilt controlado por reconhecimento de poses faciais.

Fonte: o autor.

- Microcontrolador que recebe os comandos do computador ligado à câmera pan-tilt e movimenta a câmera, em ciano.

3.5 Processamento de Imagens para detecção e reconhecimento de pose facial

A parte de processamento de imagens será a responsável por fazer o processamento dos *frames* da câmera fixa, identificando primeiramente a face do usuário, estimar a sua pose, e então converter em comandos a serem transmitidos para o controlador do suporte pan-tilt para mover a câmera móvel.

Para este projeto, a linguagem de programação escolhida foi a linguagem *Python*[10], uma uma linguagem moderna, interpretada e de alto nível, muito popular para aplicações em áreas como: *Machine Learning*, Ciência dos Dados [20] e computação científica [21]. Características gerais incluem: Ser simples e limpa, sendo fácil de ler e intuitiva, tipada dinamicamente, alocação automática de memória. Para esse projeto, a IDE (*Integrated Development Environment*, Ambiente de Desenvolvimento Integrado) escolhida foi a ferramenta Spyder[11], uma IDE com a edição do código, execução e depuração podendo ser feitos no mesmo ambiente, conforme é visto na Figura 17. Um ponto interessante a ser notado é a similaridade com ferramentas como *MATLAB*, que pode facilitar a transição de alguém que utiliza essas ferramentas, como um aluno de Engenharia, para o desenvolvimento de programas em *Python*.

[10] Ver <https://www.python.org/>
[11] Ver <https://github.com/spyder-ide/spyder>

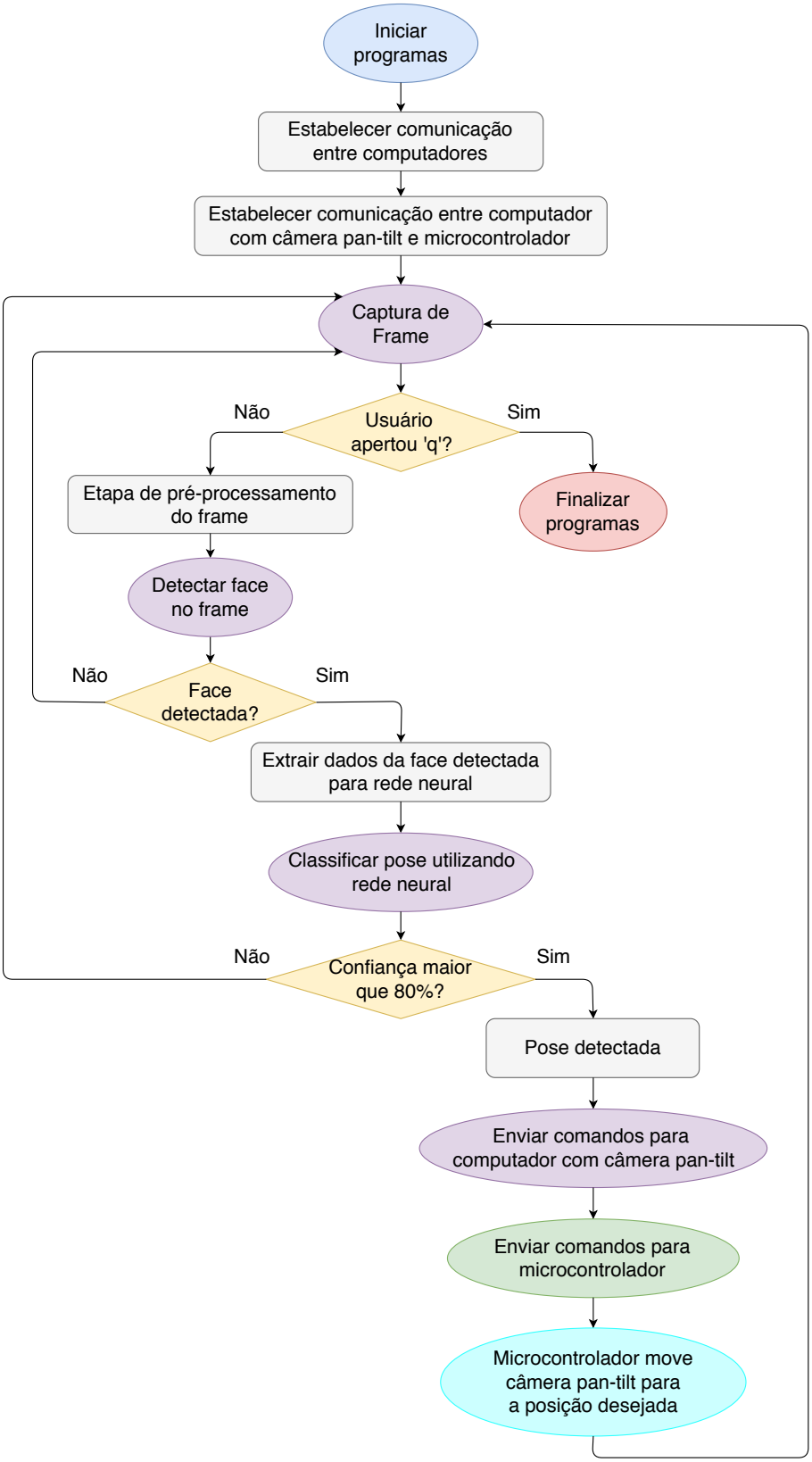


Figura 16 – Fluxograma do funcionamento do sistema, com a parte de detecção de pose facial em roxo, o computador com câmera pan-tilt em verde e em ciano o microcontrolador que move a câmera pan-tilt.

Fonte: O autor.

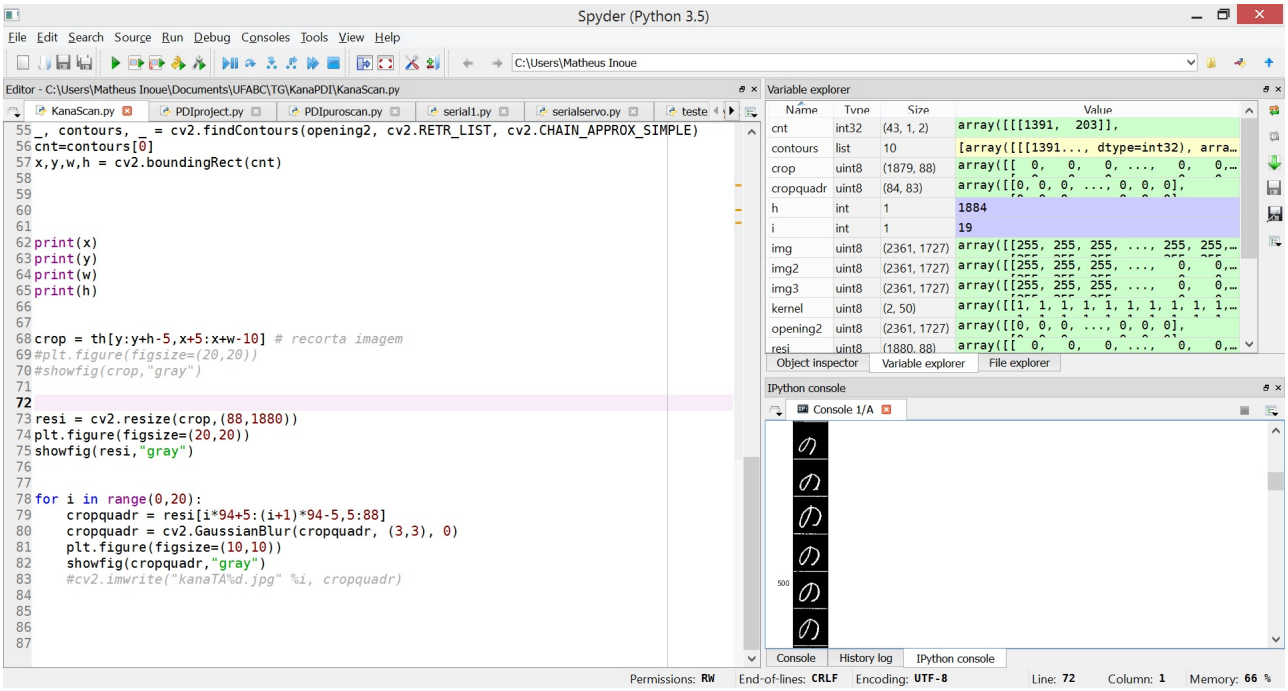


Figura 17 – IDE Spyder.

Fonte: o autor.

Uma das principais características da linguagem *Python* é a fácil instalação e uso de pacotes, ou *packages*. A principal ideia desses pacotes é fornecer códigos prontos para uso, que realizam tarefas especificas, não precisando o desenvolvedor *Python* recriar partes de códigos que já foram implementadas por algum outro desenvolvedor que contribui desenvolvendo pacotes *Python*. Esses pacotes podem realizar tarefas das mais variadas áreas como *Machine Learning*, Processamento em Linguagem Natural ou criação de gráficos, por exemplo. Esses pacotes são normalmente disponíveis gratuitamente[12],e são desenvolvidos por pessoas de diferentes áreas e segmentos, que contribuem em projetos *Open Source*.

Para processar as imagens, utilizamos as bibliotecas OpenCV[13] [22] (Open Source Computer Vision Library) que são de código aberto, escritas em C e C++, compatíveis com os principais sistemas operacionais (Windows, Linux e MacOS), e com interfaces para outras linguagens de programação, como *Python*. As bibliotecas OpenCV foram feitas pensando em eficiência computacional, com foco em sistemas de tempo-real. As bibliotecas facilitam na criação de programas de Visão Computacional, incluindo funções em diversas áreas, como: Inspeção de produtos, imagens médicas, segurança, visão estéreo, robótica etc [23]. Em *Python*, as bibliotecas OpenCV foram implementadas como um pacote a ser instalado, mas como outros pacotes, é basicamente um encapsulamento em *Python* dos programas originais em C e C++.

[12] Ver <https://pypi.python.org/pypi>
[13] Ver <http://opencv.org/>

Devido ao fato de ser uma linguagem interpretada e dinamicamente tipada, a execução de códigos em *Python* tende a ser mais lenta comparada à linguagens como C, o que poderia ser um problema para aplicações de processamento de imagens em tempo-real, que costumam ser pesadas computacionalmente. Mas em testes feitos para aplicações do tipo, a linguagem *Python* com as bibliotecas OpenCV mostrou performance muito similar ao ser comparada com o uso da linguagem C [24]. Um dos principais fatores para isso é que as bibliotecas OpenCV utilizadas são escritas em C e C++, e são encapsuladas em *Python* para execução.

3.5.1 Pré-processamento para detecção de face

Para melhorar a capacidade de detecção de faces do sistema e diminuir o seu peso computacional foram implementados duas técnicas de pré-processamento para os *frames* capturados:

Primeiramente, o *frame* lido da câmera é reamostrado à metade das dimensões originais. É um passo simples, mas que reduz o número de *pixels* por quatro, o que diminui o número de operações realizadas pelo programa para qualquer manipulação com a imagem, salvando tempo de processamento. Um ponto interessante nesse redimensionamento é a técnica de interpolação utilizada, como por área, por vizinhos mais próximos, bicúbica, linear etc. No caso deste projeto, como a imagem tem suas dimensões reduzidas, foi escolhido utilizar a interpolação por área, pois ela reduz o efeito de *aliasing*, sendo a técnica mais indicada para *downsampling* de imagens[25], como mostrado na Figura 18, onde o efeito de *aliasing* foi mínimo comparado as outras técnicas de interpolação.

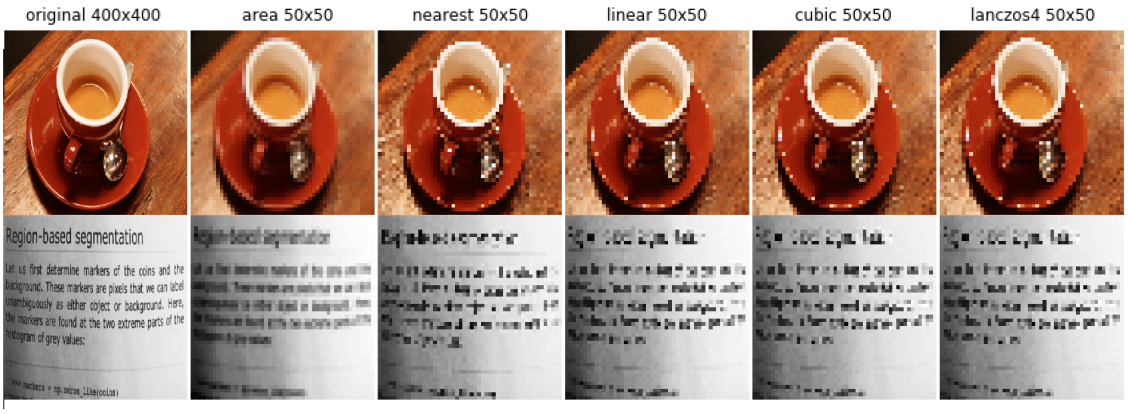


Figura 18 – Comparação de técnicas de interpolação para *downsampling* de imagens. De uma imagem de tamanho 400x400 *pixels* para uma imagem 50x50.

Fonte: Adaptado de [25].

Com o *frame* redimensionado, é realizado um pré-processamento para aumentar a capacidade de detecção de faces: A equalização adaptativa de histograma com limitação

de contraste da imagem [26] (*Contrast Limited Adaptive Histogram Equalization*, *CLAHE*). Ele é umas das técnicas utilizadas para melhorar imagens, principalmente em condições em que a iluminação não é estável, ou suficiente para a detecção de face [27].

A equalização de histograma é uma técnica para ajustar o contraste de imagens, ajustando o histograma de uma imagem para cobrir um intervalo maior de valores, alargando a forma do histograma, o que significa em um aumento do contraste da imagem, como ilustrado na Figura 19, onde uma imagem que tinha histograma delimitado em uma zona intermediária, com um contraste baixo, foi transformada em uma imagem com um histograma que abrange todo o intervalo de valores, o que gerou uma imagem com contraste maior.

Essa equalização de histograma global (ou seja, o mesmo processo para toda a imagem), pode ser útil, mas para casos onde o histograma não é limitado a um certo intervalo, a equalização global pode levar a perda de informação devido ao alargamento do histograma acabar diminuindo diferenças sutis de brilho em certas áreas da imagem. Para esse problema, pode ser usado técnicas de equalização adaptativas, onde a imagem é dividida em pequenos blocos (neste projeto, foi utilizado blocos de tamanho 8x8 *pixels*), onde a equalização ocorre para cada bloco separadamente. Um dos problemas dessa técnica é que ruídos tendem a serem amplificados. Para contornar isso é feita uma limitação por contraste, onde pontos de alto contraste são limitados e distribuídos uniformemente no histograma para outros valores do histograma[14].

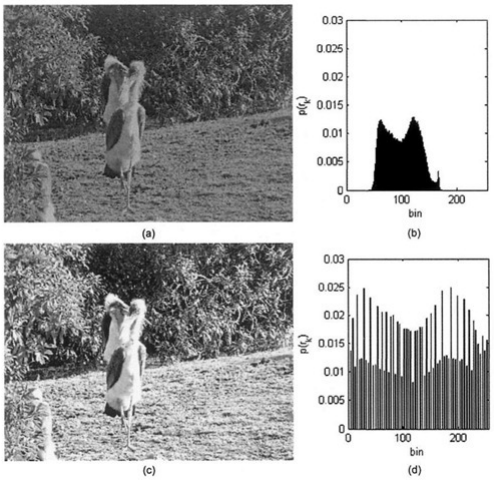


Figura 19 – Exemplo de equalização de histograma, (a) Imagem original, (b) Histograma da imagem original, (c) Imagem equalizada, (d) Histograma da imagem equalizada.

Fonte:[28].

Utilizando essas técnicas, o projeto captura *frames* de uma câmera com resolução

[14] Ver <https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html>

de 640x480 *pixels* e faz primeiramente o *downsampling*, reduzindo as dimensões da imagem para 320x240 *pixels*, utilizando a interpolação por área, converte a imagem para escala de cinza e finalmente aplica a equalização adaptativa de histograma com limitação de contraste. A imagem resultante é utilizada para os processos seguintes do sistema, conforme explicado na Figura 20. Já a Figura 21 mostra a comparação de um *frame* capturado e do *frame* processado.

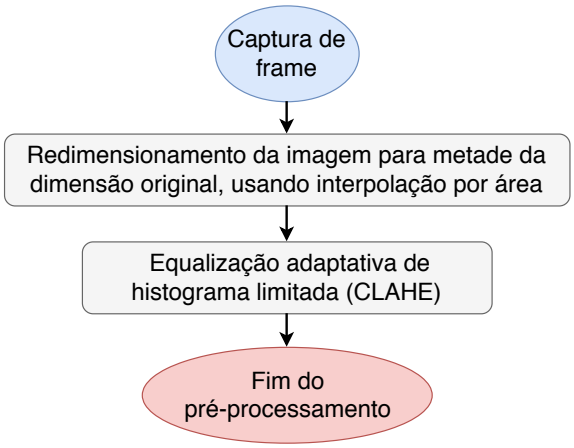
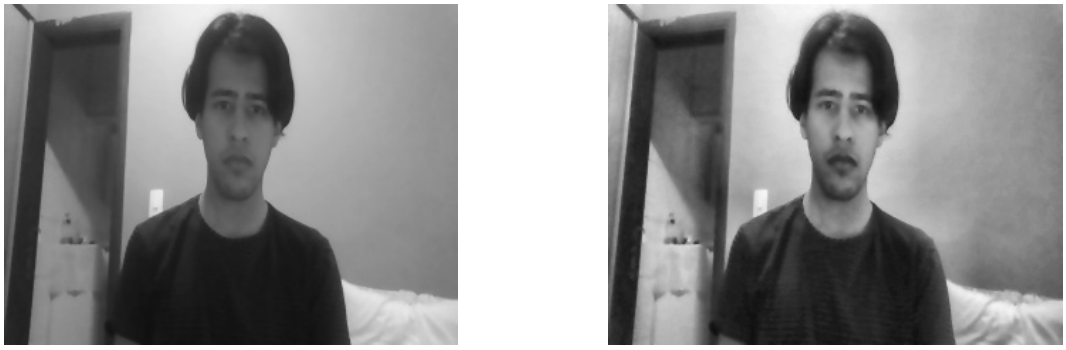


Figura 20 – Fluxograma de como o pré-processamento foi aplicado.

Fonte: O autor.

3.5.2 Detecção de face

Com os *frames* da câmera fixa processados, é feita a detecção de faces no *frame*. A técnica utilizada no sistema, é a de Kazemi-Sullivan [14], utilizando árvores de classificadores para detectar a face, bem como os seus *landmarks*. O método foi escolhido devido ao seu bom desempenho em detecção de faces, e especialmente devido a sua boa performance,



(a) *Frame* capturado (b) CLAHE

Figura 21 – Comparação do *frame* original com o pré-processado.

Fonte: o autor.

alcançando taxas de *frames* por segundo superiores a 30 fps, o que é o ideal para um sistema em tempo-real.

Assim, utilizamos a implementação do método de Kazemi-Sullivan disponível no *toolkit* Dlib [29], um *toolkit* em C++ que contém algoritmos de *Machine Learning* e ferramentas para criação algoritmos complexos em C++. É uma ferramenta utilizada em trabalhos tanto acadêmicos quanto na indústria, em campos como robótica, sistemas embarcados, telefones celulares, entre outros, pois além das suas ferramentas, possui licença *open source* que permite o seu uso gratuitamente. Apesar de ser uma biblioteca em C++, o *toolkit* também possui uma implementação em *Python*. O modelo implementado no Dlib[15] utiliza como modelo de *landmarks* faciais, o modelo de 68 *landmarks* faciais[16] [17], como visto na Figura 22. A Figura 36 demostra a detecção de face após a etapa de pré-processamento.

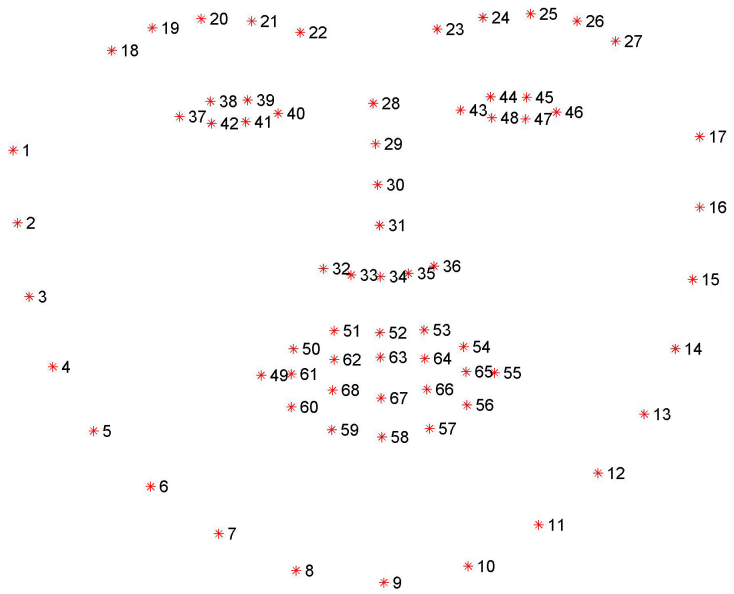


Figura 22 – Modelo de 68 *landmarks* faciais.

Fonte: [17].

3.5.3 Controle por estimação de pose facial

O modo de controle proposto neste projeto é um controle baseado no reconhecimento de poses faciais, onde o ângulo dos servomotores de Pan e Tilt são determinados pela pose reconhecida pelo sistema. Como o trabalho proposto é um protótipo, foi considerado vital a capacidade do sistema conseguir operar em tempo-real, de forma que o trabalho faz o reconhecimento de 9 poses faciais diferentes, como exemplificado na Figura 24.

[15] Ver: <https://github.com/davisking/dlib-models>
[16] Ver: <https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/>

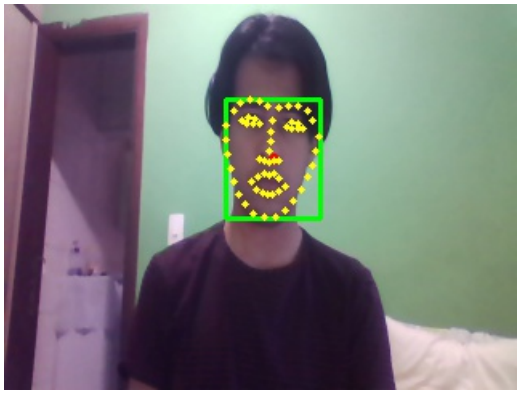


Figura 23 – Detecção de face.

Fonte: o autor.

Utilizando técnicas mais complexas, é perfeitamente possível criar um sistema que faz o reconhecimento de mais poses, que pode ter um desempenho ainda aceitável para tempo-real.

Para conseguirmos fazer a detecção de pose facial a partir da face encontrada, foi criada uma rede neural do tipo MLP (*Multilayer Perceptron*), densamente conectada, com camadas ocultas com 256, 128 e 256 neurônios, respectivamente, que foi treinada a partir de um *dataset* criado especialmente para este projeto. Esse *dataset* considerou as coordenadas retornadas de 60 dos 68 *landmarks* faciais (pontos 1 a 60 da Figura 22)da detecção de face para construir um *dataset* que apesar de possuir um números grande de amostras, tem tamanho reduzido. A rede neural então utilizou esse *dataset* como dados de treinamento.

Nesse projeto, foi proposto o seguinte método de controle:

- Inicialização do programa que detecta faces e início de captura de *frames*.
- Checar se o programa detectou uma face.
- Caso positivo, extrair dados da face detectada que são usados como entrada na rede neural para classificação de pose.
- Fazer a classificação da pose com a rede neural, e verificar se a classificação tem índice de confiança superior a 80
- Caso positivo, mandar comandos relacionados a pose detectada
- Capturar próximo *frame*.

É interessante notar que mesmo que a rede neural classifique uma pose, ela pode retornar um valor baixo de certeza, significando que a rede neural ficou dividida entre

(a) Acima e direita (b) Acima (c) Acima e esquerda

(d) Direita (e) Centro (f) Esquerda

(g) Abaixo e direita (h) Abaixo (i) Abaixo e esquerda

Figura 24 – Poses faciais consideradas.

mais de uma posição para classificação, o que pode ocorrer nos limites entre duas poses distintas. Isso poderia implicar que a classificação fique entre duas posições, e a câmera fique variando entre duas posições quando o usuário não realizou esses movimentos. Para prevenir isso, foi decidido que o programa só enviará comandos para a câmera pan-tilt quando o índice de confiança for superior a 80%. A Figura 25 exemplica o processo na forma de um fluxograma.

3.5.3.1 Criação de *dataset* para estimação de pose facial

Conforme explicado no item anterior, para o sistema de detecção de pose facial, foi decidido a criação de uma rede neural que detecta a pose facial automaticamente a partir da sua detecção na imagem capturada. Para a criação e treinamento da rede neural, é necessário criar um *dataset* para o treinamento. Na maioria dos *datasets* relacionados a algoritmos de *Machine Learning* que trabalham com imagens, são usadas imagens de

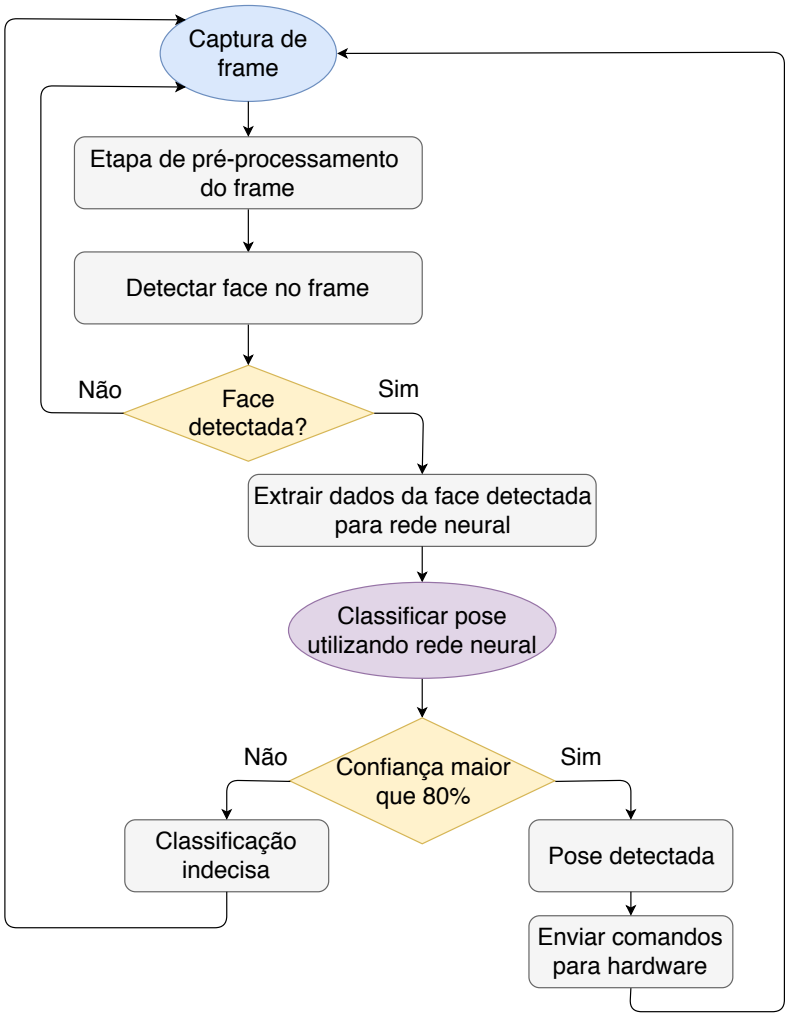


Figura 25 – Fluxograma de o controle foi proposto.

Fonte: O autor.

treinamento, isto é, um *dataset* composto por imagens. Considerando a complexidade para a sua criação, devido a necessidade de ter um ambiente amplo e preparado para capturar imagens de forma controlada[17], além de considerarmos inicialmente que imagens podem ser pesadas para o processamento em tempo-real, foi decidido a criação do *dataset* utilizando as coordenadas dos *landmarks* faciais retornados pela etapa de detecção facial.

Com isso, foi considerada 60 dos 68 *landmarks* faciais retornados pelo modelo de detecção facial. Originalmente, são retornadas as coordenadas x_j, y_j do j-ésimo *landmark*,que é um valor que depende mais de onde o usuário se encontra em relação a câmera do que a sua pose. Por isso, foi decidido utilizar a distância entre a coordenada de cada *landmark* e o centro do retângulo que cobre todas os *landmarks*, isto é, o retângulo de menor área que contém todos os pontos faciais. Considerando que p_j^x é a coordenada em x do j-ésimo *landmark*, que p_j^y é a coordenada em y, que C^x e C^y sejam as coordenadas do centro de

[17] Ver:<http://www-prima.inrialpes.fr/perso/Gourier/Faces/HPDatabase.html>

retângulo em x e y, respectivamente, a distância euclidiana do i-ésimo *frame* e do j-ésimo *landmark* D_{ij} pode ser representada como:

$$D_{ij} = \sqrt{(C^x - p_j^x)^2 + (C^y - p_j^y)^2}$$

Um valor para cada *landmark* de um *frame* especifico já possibilitaria a sua utilização para uma rede neural, mas os valores de distância ainda são suscetíveis a problemas de escala, as distâncias são dependentes do quão longe o usuário está da câmera, quanto mais perto, maiores serão as distâncias, pois a face cobrirá um número maior de *pixels*. Para contornar esse problema, foi realizada a normalização das distâncias, em relação a cada *frame*, onde todos os j-ésimos valores de distância de um frame i, D_{ij}, são divididos pelo valor máximo da distância entre os *landmarks*, de forma que os valores são normalizados entre 0 e 1:

$$D_{ij} = D_{ij}/max(D_i)$$

Com o método proposto, é necessário escolher quais ferramentas utilizar para a criação do *dataset*. A principal ferramenta utilizada foi o pacote *pandas*[18], um pacote em *Python*, que permite a criação e manipulação de estruturas de dados, projetada para permitir processos de análise de dados simples e intuitiva. Para armazenar os dados das distâncias, foi utilizado *Dataframes* [30], um objeto que representa dados em um forma tabular, formado por linhas e colunas, que cabe perfeitamente no caso do projeto. O pacote *pandas* ainda tem funções que facilitam o armazenamento e leitura de arquivos em diferentes formatos, como *CSV, XLS, JSON* etc, o que é interessante para armazenamento dos dados. Outro pacote em *Python* utilizado foi o pacote NumPy[19], um pacote que facilita o cálculos em vetores, sendo fundamental para qualquer aplicação em computação cientifica feita em *Python*, pois permite que cálculos vetoriais sejam aplicados de maneira simples, além de possuir ferramentas para áreas como Álgebra linear, geração de variáveis aleatórias, ou até mesmo Transformada de Fourier, de maneira similar ao *MATLAB*. Utilizando o método proposto e esses dois pacotes, a construção do *dataset* é simples.

Com o método proposto, e as ferramentas para criação e armazenamento do *dataset* definidas, podemos definir o processo de como o *dataset* é construído:

- Inicialização do programa que detecta faces, inicializando um *dataframe* vazio.
- Começar a capturar *frames*, e analisar se o programa detectou uma face.
- Caso positivo, obter as coordenadas dos 60 *landmarks* faciais, calcular as distâncias, e adicionar ao *dataframe* criado.

[18] Ver: <https://pandas.pydata.org/>
[19] Ver: <http://www.numpy.org/>

- Checar quando o usuário aperta a tecla "q", indicando o fim do programa, quando isso ocorre, salvar o *dataframe* criado como um arquivo *CSV*.

Para melhor visualização, este processo pode ser explicado pela Figura 26. Foi decidido salvar o *dataset* como arquivos *CSV*, pois é um formato frequentemente usado em trabalhos de análise de dados, pois é um formato livre, sem a necessidade de um *software* proprietário, como o *Microsoft Excel* para fazer a leitura do arquivo. Assim, este processo de criação do *dataset* foi rodado individualmente para cada pose considerada, e no fim foram criados vários arquivos *CSV* para cada pose, totalizando 7314 amostras distintas. Um exemplo do *dataset* gerado pode ser visto na Tabela 2, onde temos as primeiras cinco amostras de um *dataframe* para a pose central.

Tabela 2 – Exemplo da forma do *dataset* criado.

	***Landmarks* faciais**						
frame	**0**	**1**	**2**	**...**	**57**	**58**	**59**
0	0.9463	0.8403	0.7562	...	0.4943	0.4754	0.4544
1	0.8567	0.7585	0.6425	...	0.5132	0.5074	0.4884
2	0.8789	0.7602	0.6573	...	0.4642	0.4864	0.4726
3	0.8964	0.7699	0.6798	...	0.4869	0.5013	0.4928
4	0.8895	0.7694	0.6705	...	0.4901	0.4851	0.4884

Fonte: o autor [31].

3.5.3.2 Criação de Rede Neural para estimar pose facial

Com os dados de treinamento armazenados, podemos criar e treinar uma rede neural para fazer a detecção de poses faciais para o controle do sistema. Conforme discutido no item anterior, os dados a serem extraídos da detecção de faces para a entrada na rede neural foi a distância normalizada de 60 *landmarks* faciais até o centro do retângulo que contém todos os *landmarks*, em um vetor com 60 elementos. Como o *dataset* possui dados para 9 poses faciais diferentes, essa rede neural é um algoritmo de classificação, onde a saída da rede classifica uma detecção facial em alguma das poses consideradas pelo *dataset*.

Para a criação da rede neural, foi utilizada o pacote *Keras*[20] [32] uma API de alto nível para redes neurais, que roda por cima de interfaces de nível mais baixo, como o TensorFlow [33], uma interface que permite a criação e execução de algoritmos de *Machine Learning*. A principal vantagem de usar a API *Keras* é a facilidade em construir e configurar parâmetros de redes neurais. Uma rede neural pode ser de diferentes tipos e tamanhos, variando parâmetros como quantidade de camadas ocultas, ou o número de neurônios em cada camada, então vários treinamentos foram executados até ser escolhida a arquitetura final. A Figura 27 mostra como foi o processo para treinamento da rede neural, onde o modelo final idealmente possui a melhor performance entre todas as arquiteturas testadas.

[20] Ver: <https://keras.io/>

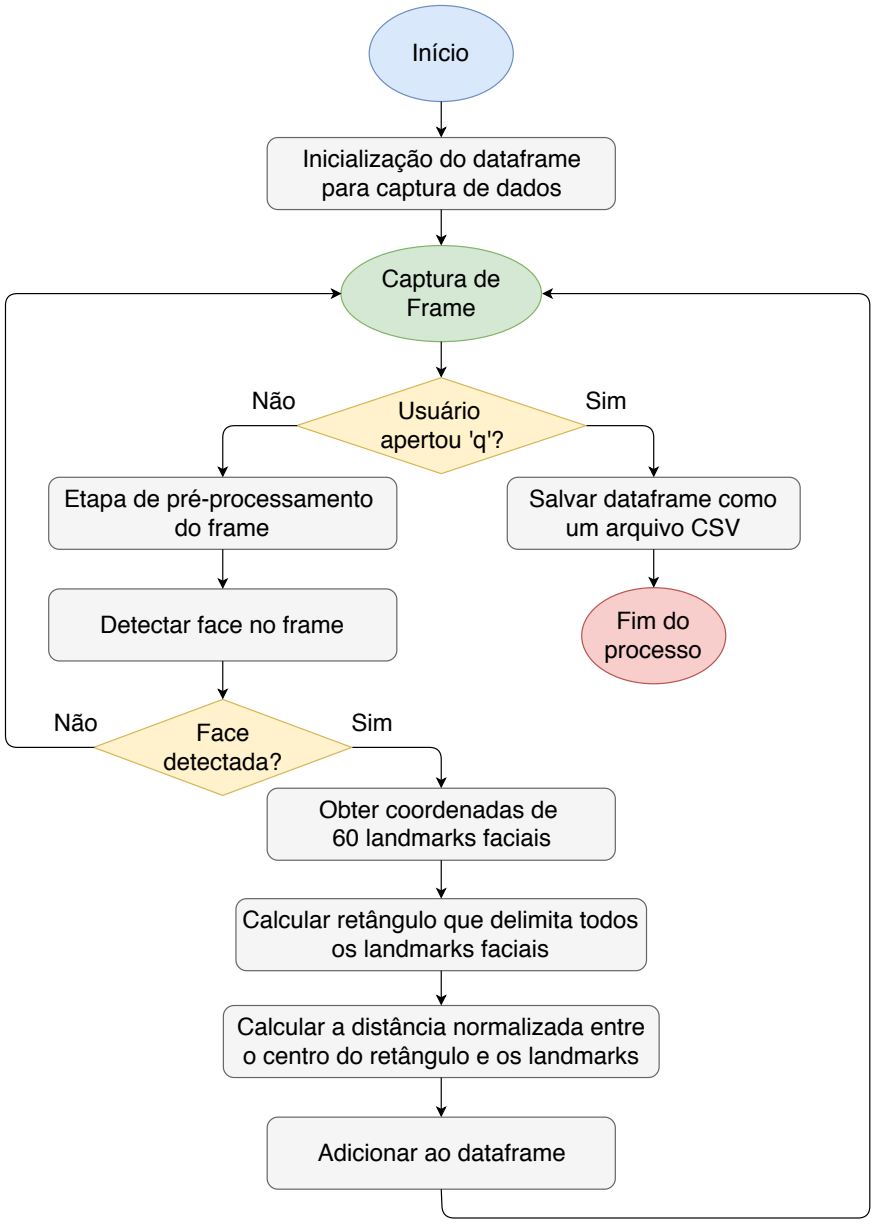


Figura 26 – Fluxograma de como o *dataset* foi criado.

Fonte: O autor.

Com os treinamentos e testes realizados, foi definido como modelo final desse projeto uma rede neural composta pela camada de entrada que recebe todos os 60 valores de distância normalizados dos *landmarks* faciais, seguida por três camadas ocultas densamente conectadas (cada neurônio de uma camada conecta com todos os neurônios da camada subsequente), composta por 256, 128 e 256 neurônios respectivamente, com funções de ativação[21] *ReLU* (*Rectified Linear Unit*) para as camadas ocultas intermediárias e *softmax* na última camada, para fazer a classificação da pose facial.

Além disso, foram adicionadas camadas de normalização em *batches* do treinamento

[21] Ver: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

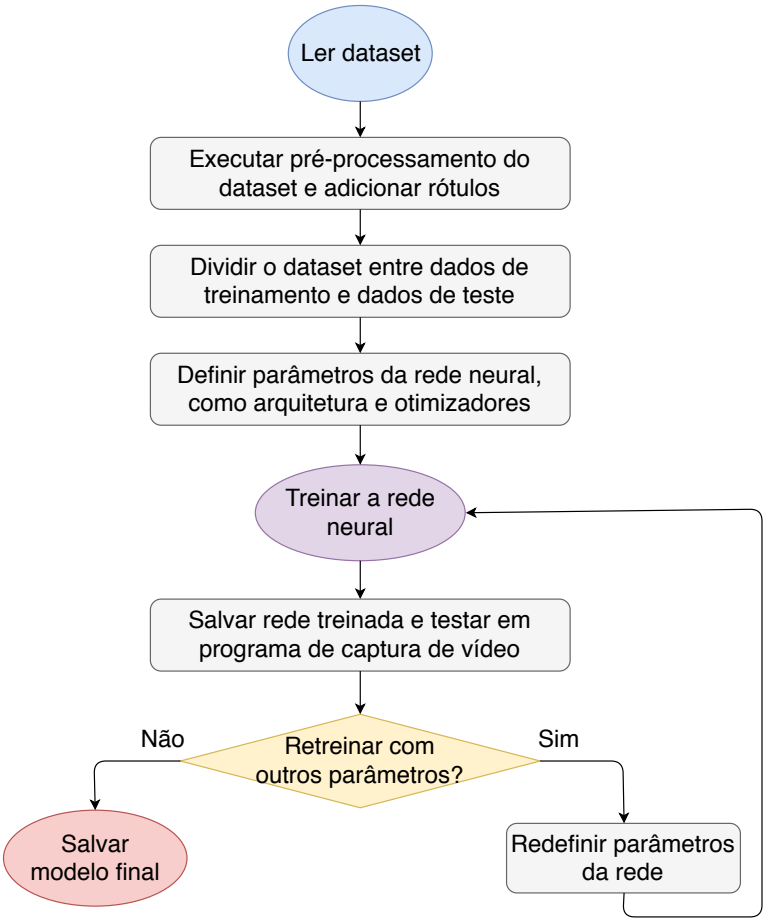


Figura 27 – Fluxograma de como o treinamento da rede neural foi feito.

Fonte: O autor.

[34], para aumentar a taxa de aprendizado, reduzindo a mudança de covariância interna e camadas de *dropout* [35] para reduzir *overfitting* zerando aleatoriamente alguns neurônios da rede neural durante cada época de treinamento.

Definida a estrutura da rede neural, é necessário definir como será executado o treinamento, definindo parâmetros como a função de perda a ser otimizada (ou seja, minimizada no caso) e a taxa de aprendizado, que define o quão rápido o treinamento é feito. Como a rede criada faz a classificação entre diferentes poses, a função de perda utilizada foi a entropia cruzada entre categorias, pois essa função interpreta os dados de treinamento e os valores de saída da rede neural como probabilidades que devem ser minimizadas, ou seja, valoriza quanto mais certeza a rede em classificar uma pose. Para o algoritmo de aprendizagem, foi utilizada a SGD[22], (*Stochastic Gradient Descent*, ou Otimização por Descida de Gradiente), um otimizador muito utilizado, que combina bom desempenho de treinamento com rápida convergência.

[22] Ver: <http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/>

3.6 Câmera Pan-Tilt

A parte do *hardware* do sistema recebe os comandos processados pela parte de detecção e classificação de pose facial e os executa, movendo a câmera pan-tilt para a posição desejada. Ele é composto pelas seguintes partes:

- Câmera compacta.
- Suporte pan-tilt, movida por servomotores MG995.
- Microcontrolador capaz de receber comandos pela porta USB.

3.6.1 Suporte Pan-tilt e câmera

A câmera pan-tilt recebe os comandos de um microcontrolador e move para a posição correspondente ao comando. Ele é composto pela câmera, no caso uma câmera compacta, como uma *webcam*, e um suporte pan-tilt, que é movida por servomotores MG995, um para cada eixo, pan e tilt, respectivamente, e que é capaz de suportar a câmera para fazer a movimentação entre as diferentes posições consideradas. Para esse projeto, foi utilizado um suporte já existente, como na Figura 28,que está disponível para compra em diversas lojas de equipamentos eletrônicos, sem a necessidade de fabricar um suporte.

Figura 28 – Suporte Pan-Tilt com servomotores[23].

3.6.2 Microcontrolador

Um microcontrolador é um dispositivo integrado que contém um núcleo processador, memória e periféricos programáveis para entrada e saída de dados (pinos utilizados para controle, chamados de GPIO - *General Purpose Input Output* , Pinos de uso geral para entrada e saída) [36].

[23] Ver: <https://www.aliexpress.com/item/2-DOF-Servo-Bracket-Mount-Kit-With-2pcs-180-Degree-High-T
32854995998.html>

O microcontrolador escolhido neste projeto foi a plataforma Arduino, mais precisamente a placa Arduino Uno, pois é um microcontrolador muito utilizado para prototipagem de sistemas eletrônicos, oferecendo um ambiente para programação simples e diversas entradas para sensores e motores. Assim, a plataforma Arduino é composta de duas partes principais: A placa, que seria o *hardware* onde conectamos motores, sensores etc, e a parte de programação, a IDE, onde é programado o que a placa deve fazer [18]. Outro ponto interessante da placa é a capacidade de ser conectada e alimentada através da porta USB, facilitando a programação os testes do sistema.

A placa Arduino Uno utiliza o microcontrolador ATmega328, feito pela Atmel, junto de todos os componentes necessários para este microcontrolador funcionar de maneira adequada, incluindo resistores, capacitores, oscilador de cristal etc. Junto com isso, a placa tem vários pinos GPIO para conectar os outros componentes, sendo:

- 14 pinos digitais I/O.
- 6 pinos analógicos de entrada.
- 2 servomotores.
- 6 pinos analógicos de saída.

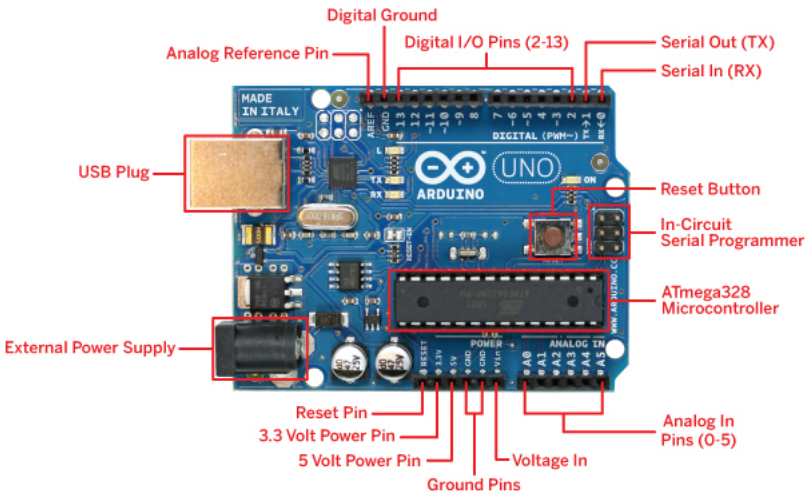


Figura 29 – Placa Arduino Uno.

Fonte:[37].

A IDE do Arduino é um ambiente simples e compatível com os principais sistemas operacionais (Windows, Linux e MacOS). A programação é feita em um *subset* da linguagem de programação C e C++[24]. Após ser escrito algum código, é possível fazer o *upload*

[24] Ver <https://www.arduino.cc/en/Main/FAQ#toc13>

do programa para a placa através da IDE, que faz a "tradução" do programa feito para a linguagem do microcontrolador [18]. Uma tela com um exemplo desse ambiente de programação é visto na Figura 30.

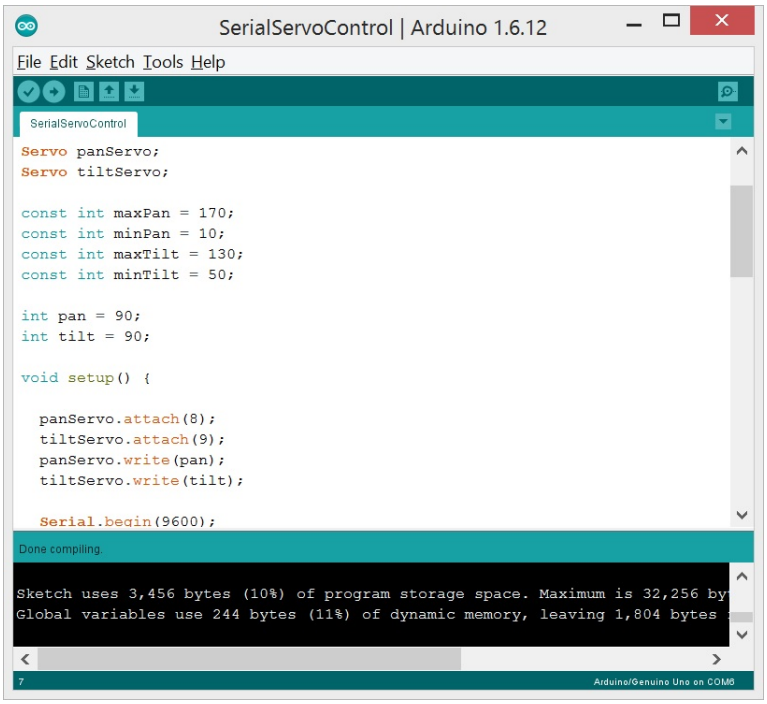


Figura 30 – IDE do Arduino.

Fonte: o autor.

O microcontrolador Arduino Uno permite controlar vários componentes como motores, com as devidas bibliotecas, que são bastante populares, e componentes comuns em projetos, como servomotores, já disponíveis com a IDE. Para o sistema, é necessário usar dois servomotores, cada um movimentando cada eixo, Pan e Tilt respectivamente. Um esquemático simples do arranjo utilizado está na Figura 31. Os servomotores são controlados com o uso dos pinos PWM (*Pulse-Width Modulation*, modulação por largura de pulso), que com o uso das bibliotecas disponíveis é necessário apenas apontar qual o ângulo para posicionar o servomotor, entre 0 e 180 graus.

Com as outras partes do *hardware* montadas, é necessário alguma interface para comunicação com a parte que classifica as poses faciais e envia comandos. A plataforma Arduino já oferece essa compatibilidade, permitindo comunicação serial através da porta USB.

3.7 Comunicação entre componentes

Com as principais partes do sistema definidas e suas arquiteturas trabalhadas, é necessário criar meios de interação entre os seus componentes:

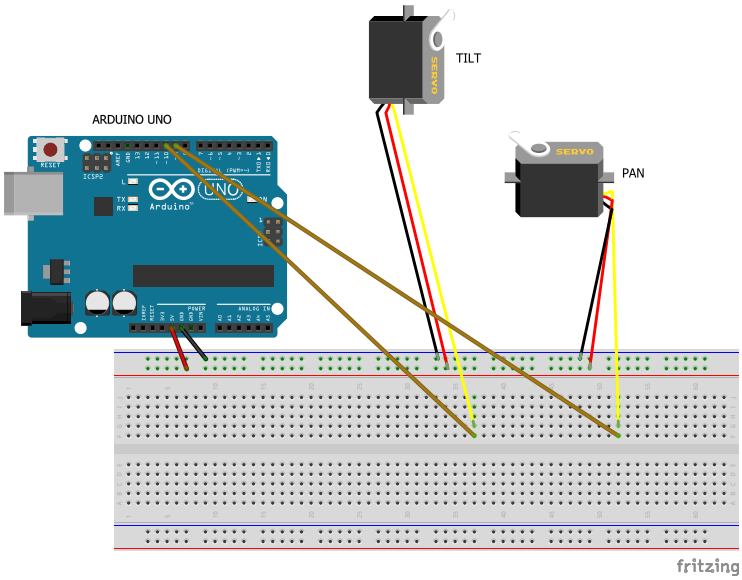


Figura 31 – Esquema simplificado do hardware, com o microcontrolador Arduino Uno e os dois servomotores que serão utilizados.

Fonte: o autor.

- Computador que roda o programa que detecta comandos por reconhecimento de pose facial.
- Computador que está conectado com a câmera pan-tilt.
- Microcontrolador que controla a câmera pan-tilt.

A Figura 32 explica como a comunicação ocorre, onde a partir do estabelecimento da comunicação entre as partes do sistema, o programa que faz a detecção de pose facial tenta detectar algum comando, e quando detecta, envia para o outro computador que está conectado diretamente com a câmera pan-tilt, e esse apenas repassa o comando recebido para o microcontrolador que aciona os servomotores para mover a câmera pan-tilt para a posição relacionada ao comando recebido.

3.7.1 Comunicação entre computadores

Para que seja feito uma simulação de um sistema de teleconferência por vídeo, é necessário algum tipo de comunicação entre o computador com a câmera fixa (que executa faz a parte de classificação de poses faciais) e o computador que utiliza a câmera pan-tilt. Uma das formas mais comuns de fazer essa comunicação é utilizando *sockets*[25]. Os *sockets* permitem a comunicação entre dois processos em máquinas diferentes, sendo uma forma de conversar com outros computadores usando descritores padrões. No UNIX, cada ação

[25] Ver:<https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm>

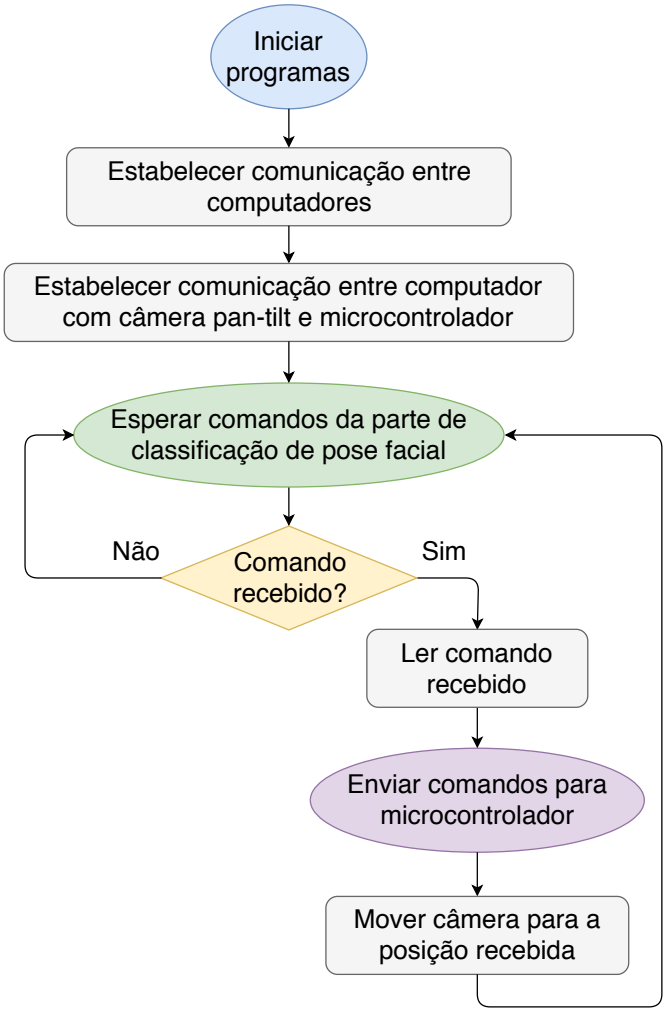


Figura 32 – Fluxograma da comunicação do sistema.

Fonte: O autor.

de *I/O* é feita escrevendo ou lendo um descritor de arquivo, e um *Socket* se comporta como um descritor de arquivo de baixo nível.

Existem 4 tipos de *sockets*: *Stream Sockets*, *Datagram Sockets*, *Raw Sockets* e *Sequenced Packet Sockets*, cada tipo com seu uso mais adequado. Considerando a natureza do projeto, é necessário garantir a transmissão dos comandos para que todas as interfaces estejam sincronizadas. Assim, o tipo a ser utilizado são os *Stream Sockets*, que garantem a entrega de cada pacote, com a ordem dos mesmo sendo também preservada. *Stream Sockets* usam o protocolo TCP (*Transmission Control Protocol*) para transmissão de dados. Esse tipo de *socket* pode ser facilmente implementado para programas em Python [38].

A Figura 33 mostra como a comunicação é feita. O programa que recebe os comandos apenas transmite eles para o Arduino. A transmissão de *sockets* acontece utilizando uma rede Wi-Fi comum entre os computadores, apenas para provar o conceito da câmera pan-tilt, onde uma comunicação mais ampla, entre diferentes redes Wi-Fi é perfeitamente

possível, mas necessitando configurações extras, sendo um possível trabalho futuro.

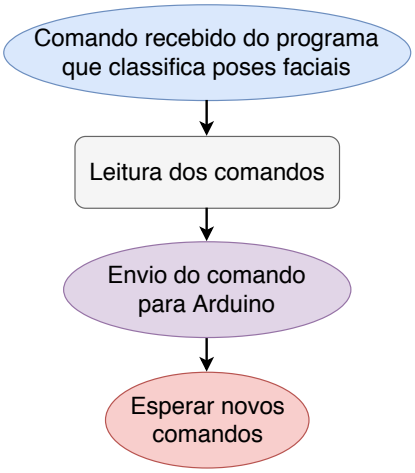


Figura 33 – Fluxograma da comunicação entre computadores.

Fonte: O autor.

3.7.2 Comunicação entre Microcontrolador e Software

A comunicação entre o Arduino e o programa em *Python* é necessária para transmitir os comandos gerados pela detecção de pose facial até os servomotores, para mover a câmera pan-tilt para a posição desejada. Essa transmissão é feita via a porta USB, pela porta serial UART[26] (*Universal Asynchronous Receiver/Transmitter*) do Arduino[27], que permite a comunicação do Arduino com outros dispositivos como outros computadores, sendo necessária apenas configurar a taxa de bits por segundo, a *baud rate*, da comunicação. Para mandar os comandos para o Arduino, foi escolhido utilizar o pacote pySerial[28], que permite fazer essa interface serial .

Para a comunicação em si, o programa em *Python* envia uma série de caracteres, que tem formato específico (ou seja, uma espécie de protocolo) onde o programa Arduino recebe essa série e consegue identificar qual o ângulo recebido de pan e tilt, e move os servomotores para a posição desejada. A Figura 34 mostra como essa comunicação acontece.

[26] Ver <http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C11_SerialInterface.htm>
[27] Ver <https://www.arduino.cc/reference/en/language/functions/communication/serial/>
[28] Ver <https://pythonhosted.org/pyserial/>

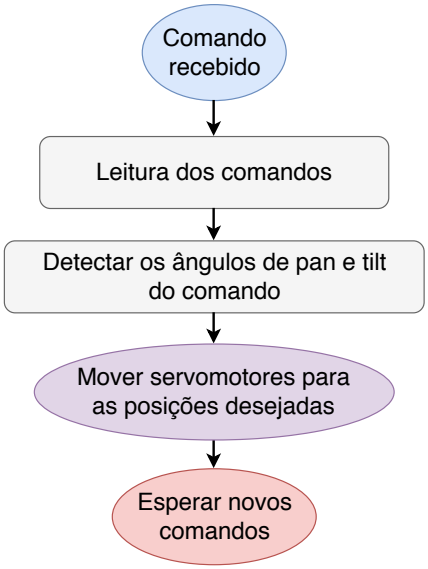


Figura 34 – Fluxograma da comunicação entre computador e microcontrolador.

Fonte: O autor.

4 Implementação e resultados

Neste capítulo, é mostrado a prova de conceito, ou seja, o que foi feito baseado na parte da concepção do projeto, com o desenvolvimento de todas as partes discutidas, bem como uma breve análise dos resultados obtidos.

4.1 Processamento de Imagens para detecção e reconhecimento de pose facial

Nessa seção são discutidos os resultados obtidos pelos algoritmos que processam *frames* da câmera fixa e fazem a detecção e reconhecimento de poses faciais, incluindo:

- **Criação do *dataset***: Programa auxiliar para criação e armazenamento das amostras de dados utilizadas no treinamento da rede neural que classifica as poses faciais.
- **Rede Neural para estimar pose facial**: Programas criados para a criação e uso da rede neural utilizada para a classificação de pose facial, que é o sinal de controle proposto da câmera pan-tilt.
- **Desempenho do sistema**: Análise do desempenho dos programas criados.

4.1.1 Criação do *dataset*

O programa para a criação do *dataset* faz a detecção e captura das distâncias utilizadas para treinamento da rede neural, conforme discutido no capítulo de concepção do projeto. O programa criado faz essa captura de pontos para cada *frame* com detecção de face, e armazena na memória o *dataframe* com dados de todas as amostras, e assim que o programa é finalizado (apertando a letra "q") o *dataframe* armazenado na memória é salvo como um arquivo *CSV*. A Figura 35 mostra a saída para o usuário, mostrando o número de amostras armazenadas na atual execução do programa.

Esse programa foi rodado diversas vezes para a criação de vários arquivos *CSV*, que constroem o *dataset* utilizado para o treinamento da rede neural construída no projeto, com um total de 7314 amostras, divididas entre as nove poses consideradas, conforme mostradas na Tabela 3. Uma observação sobre o *dataset* é que é usual uma distribuição uniforme entre as classes, o que não ocorreu no *dataset* utilizado.

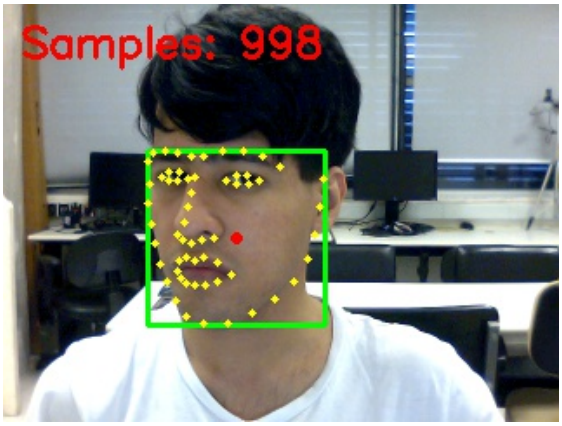


Figura 35 – Exemplo do programa de criação do *dataset*.

Fonte: O autor.

Tabela 3 – Amostras do *dataset*

Posição	**Amostras**
Centro	1189
Direita	1298
Esquerda	2061
Acima	309
Abaixo	347
Acima e direita	257
Acima e esquerda	423
Abaixo e direita	356
Abaixo e esquerda	1074
TOTAL	**7314**

Fonte: o autor.

4.1.2 Rede Neural para estimar pose facial

Com os dados de treinamento salvos, foi feita a criação e treinamento da rede neural para classificação de pose facial. Para isso, foi executado os passos discutidos no capítulo de concepção do projeto, como definição da arquitetura da rede, e os parâmetros do treinamento. Um dos parâmetros para a análise da capacidade de generalização da rede neural é comparar o desempenho da rede entre os dados do *dataset* utilizados no treinamento com os dados que foram separados para teste da rede neural. A rede criada foi treinada com 70% de dados de treinamento e 30% de dados de teste, para validação do desempenho da rede neural.

Os resultados do treinamento na Tabela 4 mostram que além da rede treinada conseguir classificar corretamente cerca de 96% das amostras do *dataset*, o desempenho entre treinamento e teste foi similar, mostrando que a rede conseguiu generalizar os dados, não aprendendo apenas as nuances dos dados de treinamento, mas sim a forma geral do

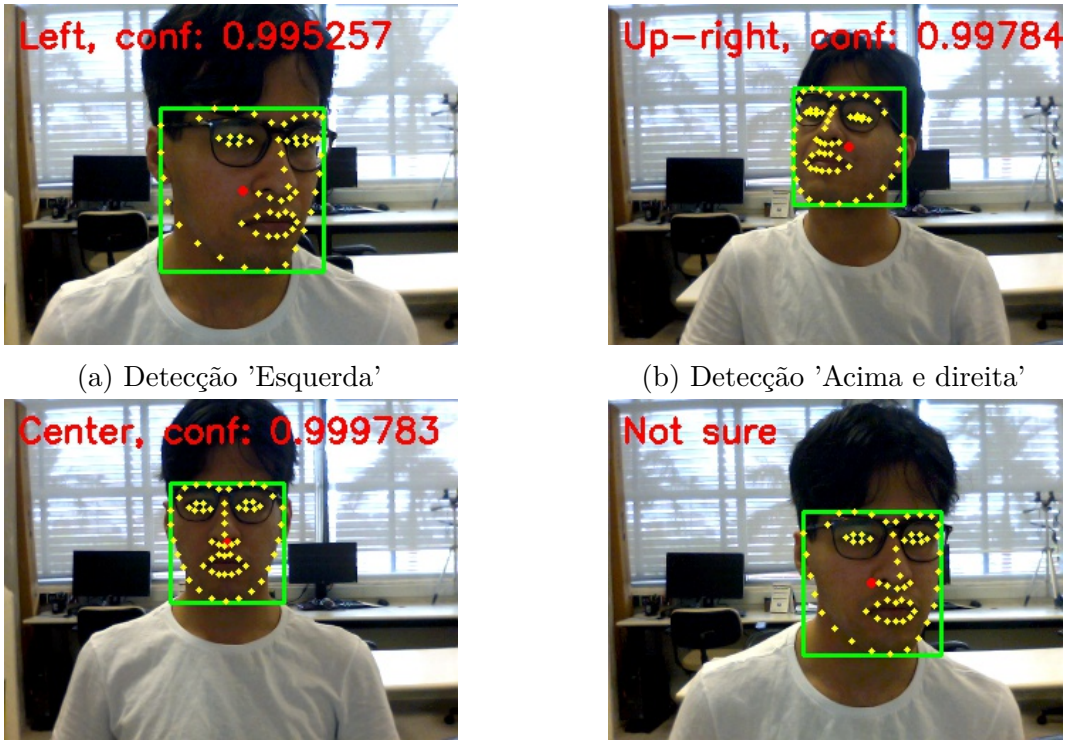
dataset.

Tabela 4 – Resultados de desempenho da rede treinada

Dados	Precisão
Treinamento	96,2%
Teste	96,9%

Fonte: o autor.

Com o modelo da rede neural salvo, é possível utilizá-lo para fazer o controle da câmera pan-tilt. A classificação ocorre na última camada da rede neural, que utiliza a função de ativação *softmax* que tem como saída um vetor com os valores de certeza para cada pose, como se fosse a probabilidade de que determinado conjunto de dados corresponde a determinada pose, limitado de 0 à 1. Para a classificação da pose para controlar a câmera, é lido o valor máximo dessa função, e caso o valor for maior que 0,8, ou seja 80%, a pose é classificada e pode ser enviado um comando para mover a câmera pan-tilt para a posição correspondente, Caso contrário, a saída da rede ficou dividida entre duas ou mais poses, então a classificação é indecisa e nenhum comando é enviado. Exemplos do programa de classificação de poses faciais podem ser vistos na Figura 36, onde as poses são classificadas juntos ao valor de certeza de classificação.



(a) Detecção 'Esquerda'

(b) Detecção 'Acima e direita'

(c) Detecção 'Centro'

(d) Detecção 'Indecisa'

Figura 36 – Amostras do resultado da rede neural treinada em uso.

Fonte: o autor.

Enquanto os itens a,b e c da Figura 36 mostram detecções com índice de certeza superior à 99%, a Figura 36d mostra quando o nível de certeza não é superior à 80%, uma classificação indecisa. Nesse exemplo o programa ficou indeciso pela face detectada estar entre as posições 'Esquerda' e 'Centro'.

4.1.3 Desempenho do sistema

Considerando os possíveis usos do protótipo, como teleconferências, é importante a análise do desempenho dos programas criados, principalmente os programas que fazem o processamento de imagem, que além de serem pesados computacionalmente, são os responsáveis para o controle da câmera. Um tempo de processamento alto adicionaria latência em todo o sistema. Essa foi uma das razões da rede neural proposta para o controle considerar apenas dados dos *landmarks* faciais, ao invés da imagem da face detectada. Assim, com os programas construídos, foram feitos testes para medir o tempo médio do processamento dos algoritmos propostos, em dois computadores diferentes, um de alta capacidade para os padrões atuais e um de capacidade regular:

- **Computador A**: Computador pessoal, utilizado durante a criação do projeto, com processador Intel i7-5500U e 16 GB de RAM.
- **Computador B**: Computador disponível no laboratório NAVI3, utilizado para testes, com processador AMD A8-5500B APU e 4 GB de RAM.

Foram feitos testes usando apenas o algoritmo de detecção de faces utilizado e com o algoritmo de detecção de face mais a rede neural para classificação de pose. Os resultados, vistos na Tabela 5 mostram números satisfatórios, com tempo médio de processamento inferior à 33ms, ou seja, seria possível processar até $\frac{1}{33ms} = 30,3$ *frames* por segundo. Outra observação foi que o impacto da rede neural não chegou a 4ms, comprovando que algoritmos mais complexos podem ser considerados para um sistema de controle mais amplo sem adicionar muita latência.

Tabela 5 – Tempo de processamento médio dos algoritmos

	Tempo de processamento (ms)	
Algoritmo	**Computador A**	**Computador B**
Apenas detecção de face	23,5	28,9
Com rede neural	+1,6	+3,6
TOTAL	**25,1**	**32,5**

Fonte: o autor.

4.2 Câmera Pan-Tilt

Para a câmera pan-tilt, seguindo o desenvolvimento da parte de concepção do projeto, foi primeiramente desenvolvido e montado o suporte pan-tilt, movido por dois servomotores MG995, um para cada eixo de rotação, pan e tilt respectivamente e controlada por um microcontrolador Arduino Uno, que recebe comandos através da porta USB de outro programa que envia comandos de forma serial.

Com o suporte montado, foi escolhido utilizar a *webcam* Microsoft LifeCam VX-800 como a câmera pan-tilt. A sua escolha foi devido ao fato de ser compacta, o que permitiu a montagem junto ao suporte montado sem maiores modificações, e ao baixo custo necessário, pois era uma câmera usada. O protótipo da câmera pan-tilt montada nesse trabalho pode ser vista na Figura 37.

Figura 37 – Câmera pan-tilt.

Fonte: O autor.

4.3 Comunicação entre componentes

Complementando os programas criados, foi estabelecido os programas e o protocolo de comunicação entre os sistemas do projeto. A comunicação ocorre quando o computador que faz a detecção e classificação de poses faciais, detecta uma pose facial, e assim manda

o comando da pose correspondente para o computador que tem a câmera pan-tilt. Esse computador então repassa o comando recebido para o microcontrolador que controla a câmera pan-tilt e move a mesma para a posição correspondente ao comando.

4.3.1 Comunicação entre Microcontrolador e Software

O Arduino deve receber comandos para a posição dos dois servomotores para mover a câmera pan-tilt, e para mover a câmera pan-tilt, o Arduino deve definir os ângulos dos servomotores em um valor inteiro entre 0 e 180 graus. Com a comunicação estabelecida entre o computador e o Arduino, foi proposto um protocolo de comunicação em que cada comando que é enviado tem o valor do ângulo de pan e tilt. o Arduino recebe o comando, lê os valores numéricos inteiros de cada servomotor e move a câmera pan-tilt para a posição desejada. Para um ângulo θ de pan e ϕ de tilt, a forma do comando foi definido como:

$$comando = p\theta t\phi$$

Ou seja, o Arduino recebe uma sequência, vai lendo caractere a caractere até encontrar um "p", a seguir ele lê o valor inteiro que corresponde ao ângulo de pan, e a seguir ele procura pelo caractere "t", com o ângulo de tilt a seguir. Como foram definidos nove diferentes poses faciais, são nove comandos diferentes que podem ser enviados no total, conforme visto na Tabela 6, mas a forma do comunicação definida permite que mais posições de pan e tilt sejam definidas em trabalhos futuros ou derivados.

Tabela 6 – Comandos enviados

Posição	**Comando**
Centro	p90t90
Direita	p60t90
Esquerda	p120t90
Acima	p90t120
Abaixo	p90t60
Acima e direita	p60t120
Acima e esquerda	p120t120
Abaixo e direita	p60t60
Abaixo e esquerda	p120t60

Fonte: o autor.

4.3.2 Comunicação entre computadores

A comunicação entre computadores foi implementada utilizando um *sockets* do tipo *stream socket*, que utiliza o protocolo TCP/IP, tornando a troca de pacotes ordenada e com garantia de transmissão, mesmo com o custo de algum atraso mínimo. Como a troca de informação é apenas comandos curtos, esse atraso é menor do que a transmissão de vídeo, por exemplo.

O sistema atual utiliza uma mesma internet *Wi-Fi* para ambos os computadores, sendo necessária apenas o compartilhamento do endereço IP local do programa que atua como servidor. O computador escolhido para atuar como servidor foi o programa que interage com a câmera pan-tilt, ou seja, o computador que recebe comandos da parte de processamento de imagens, enquanto que o cliente é o computador que faz o processamento de imagem para detectar comandos a serem enviados.

Nos *sockets* são enviados os comandos que serão retransmitidos para o microcontrolador de forma direta, sem nenhuma modificação, a não ser quando o programa de detecção de poses é finalizado, que então é enviado o comando que finaliza todas as comunicações existentes entre os componentes.

5 Considerações Finais

Nesse capítulo são discutidas as conclusões finais do trabalho, assim como as contribuições geradas e os possíveis trabalhos futuros.

5.1 Conclusões

O trabalho executado propôs um sistema de controle para dispositivos de telepresença que é simples e que não exige dispositivos de alto desempenho. O sistema de controle proposto apesar de baseado em um número limitado de dados, apenas 60 pontos, conseguiu ter um desempenho bom em detecção e principalmente em alcançar um tempo baixo de processamento, conseguindo processar mais de 30 *frames* por segundo em computadores de capacidade regular para os padrões atuais.

O trabalho executado nesse projeto foi dividido em duas partes principais, o programa que faz o processamento de imagem para classificação de poses e a parte de *hardware*, incluindo a câmera pan-tilt e o microcontrolador. Os trabalhos foram feitos separadamente, com a junção ocorrida durante o desenvolvimento da comunicação do sistema. Isso gerou um desafio de trabalhar em pequenos subprojetos dentro de um projeto maior.

5.2 Contribuições

Contribuições principais no desenvolvimento do projeto incluem os códigos dos programas criados, que estão disponíveis na plataforma GitHub:

- O programa criado para a criação do *dataset* [31].
- O programa criado para treinamento da rede neural [39].
- Os programas do sistema funcionando como um todo [40].

5.3 Trabalhos futuros

Considerando o desempenho do programa que faz a classificação de poses faciais, e limitações atuais do sistema, são propostos as seguintes melhorias, que podem ocorrer em trabalhos futuros ou derivados:

- Adição de mais posições de controle, com a criação de um sistema de classificação utilizando técnicas mais complexas, como a criação de um *dataset* de imagens, para uso em uma rede neural convolucional. O uso de imagens permitirá um maior número de dados para aferição da pose facial, muito superior ao método utilizado de apenas 60 pontos.

- Criação dos programas do sistema para usuários, não apenas programas para prova de conceito.

- Melhorias na parte de *hardware*, como adição de funções na câmera pan-tilt, como a capacidade de locomoção, como um robô de teleconferência, ou fazer um conjunto mais compacto, próximo de um produto a ser comercializado.

Referências

[1] J. G. Apostolopoulos, P. A. Chou, B. Culbertson, T. Kalker, M. D. Trott, and S. Wee, "The road to immersive communication," *Proceedings of the IEEE*, vol. 100, no. 4, pp. 974–990, 2012. Citado na página 10.

[2] D. G. Ribeiro, J. P. Silva, and C. S. Kurashima, "A System for 3D Teleconference over IP Network," *2015 XVII Symposium on Virtual and Augmented Reality (SVR)*, pp. 71–74, 2015. Citado na página 10.

[3] K. Yamaguchi, T. Komuro, and M. Ishikawa, "Ptz control with head tracking for video chat," in *CHI'09 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2009, pp. 3919–3924. Citado 3 vezes nas páginas 10, 12 e 26.

[4] S. Boll, "Multimedia at chi: Telepresence at work for remote conference participation," *IEEE MultiMedia*, vol. 24, no. 3, pp. 5–9, 2017. Citado na página 11.

[5] V. A. Newhart, M. Warschauer, and L. Sender, "Virtual inclusion via telepresence robots in the classroom: An exploratory case study," 2016. Citado na página 11.

[6] R. Leeb, L. Tonin, M. Rohm, L. Desideri, T. Carlson, and J. d. R. Millan, "Towards independence: a bci telepresence robot for people with severe motor disabilities," *Proceedings of the IEEE*, vol. 103, no. 6, pp. 969–982, 2015. Citado na página 12.

[7] H. K. Kelda and P. Kaur, "A review: color models in image processing," *International Journal Computer Technology and Applications*, vol. 5, pp. 319–322, 2014. Citado na página 14.

[8] A. Maris, S. Zafeirou, and B. Glocker, "Implementation and study of cascaded-regression methods for facial feature points detection," 2015. Citado 3 vezes nas páginas 15, 16 e 23.

[9] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010. Citado na página 16.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. Citado na página 16.

[11] M. A. Nielsen, *Neural networks and deep learning*. Determination Press, 2015. Citado 2 vezes nas páginas 18 e 19.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice-Hall, 1999. Citado na página 19.

[13] P. Viola and M. J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," *Proc. of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. I–511– I–518, 2001. Citado 2 vezes nas páginas 19 e 20.

[14] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *CVPR*, 2014. Citado 4 vezes nas páginas 19, 21, 24 e 35.

[15] Face detection using haar cascades. [Online]. Available: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html Citado 2 vezes nas páginas 20 e 21.

[16] M. Castrillón-Santana, O. Déniz-Suárez, L. Antón-Canalís, and J. Lorenzo-Navarro, "FACE AND FACIAL FEATURE DETECTION EVALUATION: Performance Evaluation of Public Domain Haar Detectors for Face and Facial Feature Detection," *VISAPP*, 2008. Citado na página 21.

[17] C. Sagonas, E. Antonakos, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 faces in-the-wild challenge: Database and results," *Image and Vision Computing*, vol. 47, pp. 3–18, 2016. Citado 2 vezes nas páginas 22 e 36.

[18] M. Banzi, *Getting Started with Arduino*, 2nd ed. MakeBooks, 2010. Citado 3 vezes nas páginas 24, 45 e 46.

[19] S. Chu and J. Tanaka, "Hand Gesture for Taking Self Portrait," *Human-Computer Interaction. Interaction Techniques and Environments*, 2010. Citado na página 26.

[20] J. VanderPlas, *Python data science handbook: Essential tools for working with data.* "O'Reilly Media, Inc.", 2016. Citado na página 30.

[21] R. Johansson, *Introduction to Scientific Computing in Python*, 2016. [Online]. Available: https://github.com/jrjohansson/scientific-python-lectures Citado na página 30.

[22] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. Citado na página 32.

[23] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 2nd ed. O'Reilly Media, 2008. Citado na página 32.

[24] B. Thorne and R. Grasset, "Python for Prototyping Computer Vision Applications," *Proc. of New Zealand Computer Science Research Student Conference (NZCSRSC 2010)*, Apr. 2010. Citado na página 33.

[25] Comparison of opencv interpolation algorithms. [Online]. Available: http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/ Citado na página 33.

[26] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer vision, graphics, and image processing*, vol. 39, no. 3, pp. 355–368, 1987. Citado na página 34.

[27] M. A. Onabid and D. T. Charly, "Enhancing gray scale images for face detection under unstable lighting condition," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 10, pp. 12–20, 2017. Citado na página 34.

[28] Histogram equalization (image processing) part 1. [Online]. Available: http://what-when-how.com/embedded-image-processing-on-the-tms320c6000-dsp/histogram-equalization-image-processing-part-1/ Citado na página 34.

[29] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009. Citado na página 36.

[30] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.", 2012. Citado na página 40.

[31] M. Inoue, "extract-facepoints-dataset," https://github.com/navi3-research-group/extract-facepoints-dataset, 2018. Citado 2 vezes nas páginas 41 e 58.

[32] F. Chollet *et al.*, "Keras," https://keras.io, 2015. Citado na página 41.

[33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/ Citado na página 41.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. Citado na página 43.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. Citado na página 43.

[36] A. Smith, *Introduction to Arduino*, 2nd ed., 2011. [Online]. Available: http://www.introtoArduino.com Citado na página 44.

[37] M. Marschalko. Arduino cheat sheet. [Online]. Available: http://www.webondevices.com/resources/arduino-cheat-sheet/ Citado na página 45.

[38] G. McMillan, *Socket Programming HOWTO*, Python Software Foundation, 2017. [Online]. Available: https://docs.python.org/3/howto/sockets.html Citado na página 48.

[39] M. Inoue, "head-pose-control-model," https://github.com/navi3-research-group/head-pose-control-model, 2018. Citado na página 58.

[40] M. Inoue, "pan-tilt-pose-control," https://github.com/navi3-research-group/pan-tilt-pose-control, 2018. Citado na página 58.