UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Faculdade de Engenharia e Ciências de Guaratinguetá

DANIEL SIMÃO IANNI

Integrando Visão Computacional e Sistemas Embarcados: Rastreio de objetos com CSRT e controle automático de câmera Pan-Tilt usando ESP32

Guaratinguetá
2023

Daniel Simão Ianni

Integrando Visão Computacional e Sistemas Embarcados : Rastreio de objetos com CSRT e controle automático de câmera Pan-Tilt usando ESP32

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica .

Orientador: Profº Dr. Daniel Julien Barros da Silva Sampaio

Guaratinguetá
2023

Ianni, Daniel Simão
I11i Integrando visão computacional e sistemas embarcados: rastreio de objetos com CSRT e controle automático de câmera Pan-Tilt usando ESP32. / Daniel Simão Ianni. – Guaratinguetá, 2023.
76 f : il.
Bibliografia: f. 68-70

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia e Ciências de Guaratinguetá, 2023.
Orientador: Prof. Dr. Daniel Julien Barros da Silva Sampaio

1. Rastreamento (Engenharia). 2. Sistemas embarcados (Computadores). 3. Processamento de imagem assistida por computador. 4. Microcontroladores. I. Título.

CDU 621.381

Luciana Máximo
Bibliotecária/CRB-8 3595

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

DANIEL SIMÃO IANNI

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUANDO EM ENGENHARIA ELÉTRICA "**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Profº Dr. DANIEL JULIEN BARROS DA SILVA SAMPAIO
Coordenador

BANCA EXAMINADORA:

Prof. Dr. Daniel Julien Barros da Silva Sampaio
Orientador/UNESP-FEG

Prof. Dr. Samuel Euzédice de Lucena
UNESP-FEG

Prof. Dr. T/eófilo Miguel de Souza
UNESP-FEG

Fevereiro , 2023

“Fracassei em tudo o que tentei na vida.
Tentei alfabetizar as crianças brasileiras, não consegui.
Tentei salvar os índios, não consegui.
Tentei fazer uma universidade séria e fracassei.
Tentei fazer o Brasil desenvolver-se autonomamente e fracassei.
Mas os fracassos são minhas vitórias.
Eu detestaria estar no lugar de quem me venceu.
(Darcy Ribeiro)“

RESUMO

Este projeto tem o propósito de demonstrar a integração entre a visão computacional e sistemas embarcados ao construir um sistema de rastreamento de objetos em tempo real. Para isso, é utilizado um microcontrolador ESP32 e um módulo de câmera OV2640. As imagens capturadas são enviadas para um computador, onde são processadas com a ajuda da biblioteca OpenCV e do algoritmo de rastreio CSRT (Filtro de Correlação Discriminativo com Confiabilidade Espacial e de Canal). O computador calcula os ângulos de panorâmica e inclinação da câmera, e então envia sinais de controle para o ESP32 para ajustar a posição dos servo motores, mantendo o objeto no centro da imagem. Os resultados comprovam que o sistema é eficiente e preciso na tarefa de rastreamento, mesmo em situações desafiadoras. Além disso, o baixo custo e consumo de energia tornam o sistema adequado para aplicações diversas, como vigilância, robótica e veículos autônomos. Em suma, este projeto evidencia a potencialidade de combinar sistemas embarcados e visão computacional para criar soluções de baixo custo e alto desempenho para aplicações práticas.

PALAVRAS-CHAVE: Visão Computacional; Sistemas Embarcados; Câmera Pan-tilt; Rastreamento de Objetos; CSRT; ESP32.

ABSTRACT

This project aims to demonstrate the integration of computer vision and embedded systems by building a real-time object tracking system using an ESP32 micro-controller and an OV2640 camera module. The system captures images and sends them to a PC for processing using the OpenCV library and the CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) algorithm. The PC computes the pan and tilt angles of the camera and sends control signals to the ESP32 to adjust the position of two servo motors, keeping the object centered in the frame. The results show that the system is able to track the object with high accuracy and robustness in real-time, even in challenging scenarios. The low cost and power consumption of the system make it suitable for a variety of applications, including surveillance, robotics, and autonomous vehicles. The project highlights the potential of combining embedded systems and computer vision for building low-cost and high-performance systems for real-world applications.

KEYWORDS: Computer Vision; Embedded Systems; Pan-Tilt Camera; Object Tracking; CSRT; ESP32.

LISTA DE ABREVIATURAS E SIGLAS

DoG	Diferença de Gaussianas
HoG	Histograma de Gradiente
CSRT	Filtro de Correlação Discriminativo com Confiabilidade Espacial e de Canal
DCT	Filtro de Correlação Discriminativo
VP	Verdadeiro Positivo
FP	Falso Positivo
VN	Verdadeiro Negativo
FN	Falso Negativo
BG	*Background*
KP	*Keypoints*
IP	*Internet Protocol*
PC	*Personal Computer*
HTML	*HyperText Markup Language*
BMS	*Battery Manage System*
ULP	*Ultra Low Power*
BLE	*Bluetooth Low Energy*
GPIO	*General Purpose Input Output*
USB	*Universal Serial Bus*
OTA	*Over the Air Update*
RF	Radiofrequência
ROM	*Read Only Memory*
SRAM	*Static Random-Access Memory*
PSRAM	*Pseudostatic RAM*
CPU	*Central Processing Unity*
IOT	*Internet of Things*

MCU	*Microcontroller Unity.*
PCB	*Printed Circuit Board*
DSP	*Digital Signal Processor*
ADC	Conversor Analógico Digital
CI	Circuito Integrado
SoC	*System-on-a-Chip*
FPS	*Frames* por Segundo
TCC	Trabalho de Conclusão de Curso
UNESP	Universidade Estadual Paulista

SUMÁRIO

1 INTRODUÇÃO

A visão computacional é uma área importante da Engenharia Elétrica, que tem sido amplamente aplicada em diversos setores da sociedade, incluindo segurança, transporte, automação industrial, medicina, entre outras. Com a evolução da tecnologia e aumento da demanda por soluções mais eficientes, o processamento de imagens tem sido cada vez mais integrado a sistemas embarcados, possibilitando a utilização de dispositivos de pequeno porte para realizar tarefas antes possíveis apenas em computadores de grande porte.

A utilização de processamento de imagem em conjunto com sistemas micro-controlados apresenta muitos desafios, mas também oferece muitas vantagens. O desafio principal é equilibrar a necessidade de processamento intensivo com a limitação de recursos de hardware. Em compensação, o uso de visão computacional em processamento de borda de imagem permite aos profissionais de diferentes setores resolverem desafios complexos e tomar decisões baseadas em informações precisas e confiáveis.

No entanto, a implementação desse tipo de tecnologia em sistemas embarcados apresenta barreiras significativas. Esses dispositivos geralmente são pequenos e possuem recursos limitados, como memória e capacidade de processamento. Além disso, eles geralmente são alimentados por baterias e precisam ser projetados para consumir energia de maneira eficiente.

Tendo em vista as dificuldades apresentadas para realizar o processamento de imagens em micro-controladores, este projeto tem como objetivo investigar a possibilidade de se utilizar um ESP32 para capturar imagens e enviá-las para um computador realizar o processamento de imagens. A ideia é capturar os *frames* por meio de uma câmera conectada ao ESP32, e transmitir esses dados para o PC através de uma conexão WiFi.

Dessa forma, a combinação de ESP32 e PC permite aproveitar as vantagens de ambos os dispositivos, capturando imagens com praticidade e mobilidade pelo SoC e processando-as com eficiência pelo computador. Além disso, a utilização de um sistema embarcado como o ESP32 possibilita a implementação do projeto em aplicações práticas que envolvem controle em tempo real e IOT.

1.1 OBJETIVOS

Este trabalho de conclusão de curso tem como propósito a utilização de uma câmera embarcada para realizar o controle de um suporte Pan-Tilt composto por dois servo motores. O problema de pesquisa a ser abordado é o desafio de controlar a câmera de forma que ela sempre capture a imagem de um objeto de interesse.

A metodologia utilizada para alcançar este objetivo é a combinação de algoritmos modernos de rastreio e de controle do suporte Pan-Tilt. O algoritmo de localização é responsável por identificar e acompanhar o objeto de interesse, enquanto o controle de movimento do suporte é responsável por ajustar a posição da câmera para que o objeto fique sempre no centro da imagem.

A primeira parte do projeto incluirá o desenvolvimento de um código para capturar imagens com o ESP32 e o módulo OV2640 e enviá-las para o computador. A segunda parte incluirá o desenvolvimento de código para o processamento de imagens no computador, utilizando a biblioteca OpenCV em

Python. O projeto também incluirá a utilização de servo-motores para controlar o módulo pan-tilt. O ESP32 será programado para controlar com PWM os servo-motores de acordo com as informações de rastreamento de objetos obtidas pelo computador.

Em resumo, os objetivos deste trabalho de conclusão de curso incluem:

- Desenvolver um sistema de captura de imagens com o ESP32 e o módulo OV2640.
- Programar o ESP32 para enviar imagens para o PC de forma rápida e confiável.
- Desenvolver código para processamento de imagens no computador utilizando OpenCV.
- Controlar servo-motores para centralizar o objeto na foto.
- Realizar testes para avaliar a eficiência e precisão do sistema.

Este trabalho de graduação é uma excelente maneira de aprender sobre processamento de imagem, reconhecimento de objetos e controle de servo-motores, além de possibilitar a criação de projetos IoT interessantes e úteis. Além disso, o baixo custo e consumo de energia do ESP32 o torna acessível e fácil de ser utilizado em projetos de pequena escala.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece uma visão geral dos conceitos teóricos que servem como base para o projeto. Dessa forma, são apresentados as matrizes de transformadas geométricas em coordenadas homogêneas, bem como as matrizes intrínsecas e extrínsecas da câmera. O estudo dessas matrizes se faz necessário para mapear pontos em sistemas de visão formados por múltiplas projeções.

O capítulo apresenta também uma visão sucinta da teoria por trás de uma câmera *rolling shutter* CMOS, abordando os conceitos teóricos relacionados aos sensores digitais de duas dimensões. Por fim, são discutidos os algoritmos de processamento de imagens, dando especial atenção às técnicas de rastreio.

2.1 ÁLGEBRA E GEOMETRIA DO SISTEMA DE VISÃO

Esta seção é destinada a desenvolver as principais equações geométricas que compõem um sistema de visão.

2.1.1 Matrizes das transformadas geométricas

2.1.1.1 Translação

Sendo $P = (x, y, z)$ um ponto no espaço euclidiano, para realizar uma translação de P basta soma-lo pelo vetor $\vec{u} = (a, b, c)$. Tendo assim:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} a \\ b \\ c \end{pmatrix} \leftrightarrow \begin{cases} x' = x + a \\ y' = y + b \\ z' = z + c \end{cases} \tag{1}$$

Além disso, pode-se representar a translação por uma transformação linear $L : R^3 \mapsto R^3$ associada a multiplicação da matriz T pelo ponto P:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = T \times P = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2}$$

2.1.1.2 Mudança de escala

A mudança de escala do ponto $P = (x, y, z)$ pode ser representada pela multiplicação matricial a seguir:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = S \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leftrightarrow \begin{cases} x' = x \times S_x \\ y' = y \times S_y \\ z' = z \times S_z \end{cases} \tag{3}$$

2.1.1.3 Rotação

Ao introduzir a matriz de rotação, é interessante analisar o problema primeiramente em duas dimensões. Para rotacionar com ângulo α o ponto $P = (x, y)$ no sentido anti-horário basta multiplicar pela matriz R:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{4}$$

2.1.1.4 Demonstração da matriz de rotação

Relembrando as identidades trigonométricas:

$$\sin(A \pm B) = \sin(A)\cos(B) \pm \sin(B)\cos(A) \tag{5}$$

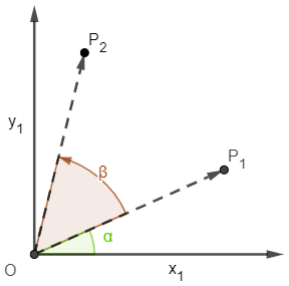
$$\cos(A \pm B) = \cos(A)\cos(B) \mp \sin(A)\sin(B) \tag{6}$$

Sendo o ponto $P_1 = e^{i\alpha}$, também $P_1 = \hat{x}(\cos(\alpha)) + \hat{y}(\sin(\alpha))$.

$$\begin{cases} x_1 = \cos(\alpha) \\ y_1 = \sin(\alpha) \end{cases} \tag{7}$$

Rotacionando P_1 em β no sentido anti-horário como mostra na Figura 1, tem se o ponto P_2:

Figura 1 – Rotação do ponto P_1 com ângulo de β



fonte: Produção do próprio autor. Feito no GeoGebra.

$$P_2 = \hat{x}(\cos(\alpha + \beta)) + \hat{y}(\sin(\alpha + \beta)) \tag{8}$$

Usando as equações 5 e 6:

$$\begin{cases} x_2 = \cos(\beta)\underbrace{\cos(\alpha)}_{x_1} - \sin(\beta)\underbrace{\sin(\alpha)}_{y_1} \\ y_2 = \sin(\beta)\underbrace{\cos(\alpha)}_{x_1} + \sin(\beta)\underbrace{\sin(\alpha)}_{y_1} \end{cases} \tag{9}$$

Tendo assim o P_2 com relação P_1 e o ângulo de rotação β:

$$\begin{cases} x_2 = \cos(\beta)x_1 - \sin(\beta)y_1 \\ y_2 = \sin(\beta)x_1 + \sin(\beta)y_1 \end{cases} \tag{10}$$

Finalmente, tem-se a matriz de rotação R:

$$R = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{11}$$

2.1.2 Transformações lineares

Sendo M e N dois espaços vetoriais, $T : M \mapsto N$ é uma transformação linear se e somente se:

$$T(a + b) = T(a) + T(b) \tag{12}$$

$$T(\lambda a) = \lambda T(a) \tag{13}$$

Com $a, b \in M$ e $\lambda \in \mathbf{R}$.

Na prática, as transformações geométricas de rotação e escala apresentadas nas seções anteriores podem ser consideradas como operadores lineares. O caso da translação impõe uma condição de não linearidade e a solução para resolver esse problema sera abordada no próximo capítulo.

2.1.3 Coordenadas homogêneas

Um exemplo das limitações das matrizes apresentadas até este momento é o caso da transformação composta por uma translação e por uma mudança de escala.

Sendo um ponto cartesiano de duas dimensões $P = (x, y)$, é simples de ver que as equações abaixo representam uma translação e uma mudança de escala:

$$\begin{cases} x_2 = S_x \times x_1 + T_x \\ y_2 = S_y \times y_1 + T_y \end{cases} \tag{14}$$

Além disso, escrevendo na forma matricial essa composição de transformadas:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = S \begin{pmatrix} x \\ y \end{pmatrix} + T = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \end{pmatrix} \tag{15}$$

A transformação $(x_1, y_1) \mapsto (x_2, y_2)$, apresentada em 15, não é uma aplicação linear e esse tipo de formulação impõe dificuldades computacionais quando se concatenam diversas transformadas. Dessa forma, faz-se necessário introduzir as coordenadas homogêneas.

Um ponto definido em $\mathbf{R}^N$, com $N \in \mathbf{N}$, quando escrito em coordenada homogênea é acrescido de uma dimensão. O ponto $P = (x, y)$ equivale em coordenada homogênea como mostrado a seguir:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix}, \forall \lambda \in \mathbf{R}^* \tag{16}$$

Na prática utiliza-se $\lambda = 1$, tendo o vetor $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$.

Por fim a utilidade dessa ferramenta pode ser vista nas equações a seguir, onde as duas transformadas de rotação e escala formam uma única matriz M, que é uma transformação linear.

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = M \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & T_x \\ 0 & S_x & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{17}$$

Quando trabalha-se com pontos em N dimensões, para simplificar utiliza-se coordenadas homogêneas adicionando uma coluna com o valor 1 e formando assim um vetor de $N + 1$ dimensões.

2.1.4 Transformadas geométricas homogêneas

Tendo em vista as vantagens de se utilizar coordenadas homogêneas para resolver os problemas de geometria computacional, nesta seção serão reapresentadas as matrizes de transformações.

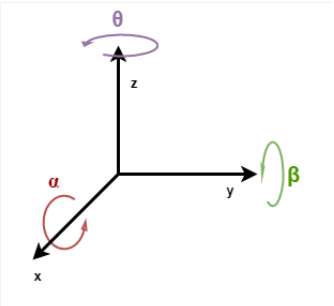
Para a mudança de escala:

$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{18}$$

Para a translação:

$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{19}$$

Figura 2 – Eixos e ângulos de rotação.



fonte: Produção do próprio autor. Feito no draw.io.

Para melhor visualizar as rotações, utiliza-se como referência a Figura 2.

A rotação no eixo x:

$$R_\alpha = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{20}$$

A rotação β no eixo y:

$$R_\beta = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{21}$$

A rotação no eixo z:

$$R_\theta = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{22}$$

2.1.5 Concatenando transformadas geométricas

Como apresentado anteriormente, uma das principais vantagens de se utilizar matrizes e coordenadas homogêneas em computação gráfica é poder concatenar uma sequência de transformações em uma matriz única. Por exemplo, uma translação T, seguida por uma rotação R_α e de uma rotação R_β pode ser descrito assim:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = TR_\alpha R_\beta \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} \tag{23}$$

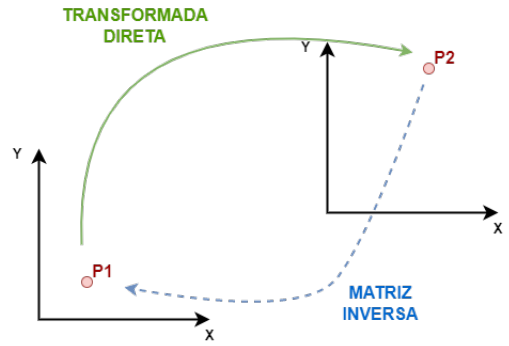
Visto que uma composição de transformadas é o equivalente a uma sequência de multiplicações matriciais e que essas matrizes não comutam, a ordem das operações geométricas importa.

Além disso, a matriz $A = TR_\alpha R_\beta$ tem o formato 4×4. Pode-se ver novamente a vantagem de utilizar coordenadas homogêneas, pois essa nova matriz representa por sua vez uma operação linear quando aplicamos a um ponto da forma $(x, y, z, 1)^T$.

Concluindo, uma sequência de operações geométricas pode ser representada por uma única matriz $M_{4\times4}$:

$$M_{4\times4} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \tag{24}$$

Figura 3 – Mapeamento direto e inverso de pontos usando matriz.



fonte: Produção do próprio autor. Feito no draw.io.

2.1.6 A inversa da matriz de transformação

Sendo M uma matriz representando uma transformada geométrica que mapeia o ponto P_1 para a posição P_2. A matriz inversa M^{-1} mapeia o ponto P_2 até a posição inicial.

Para melhor visualizar essa propriedade, considere o ponto $P_1 = (1, 2)$ e a matriz de translação:

$$T = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix} \tag{25}$$

Analiticamente tem-se $P_2 = (4, 7)$. Calculando a matriz inversa:

$$T^{-1} = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{pmatrix} \tag{26}$$

Pode-se, finalmente, observar a propriedade da matriz inversa, com $P_1 = T^{-1}P_2$:

$$P_1 = T^{-1}P_2 = T^{-1}\begin{pmatrix} 4 \\ 7 \end{pmatrix} = \begin{pmatrix} 4-3 \\ 7-5 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{27}$$

A Figura 4 mostra um exemplo com várias transformadas geométricas. Para retornar ao ponto inicial basta aplicar a transformada inversa da matriz de concatenação. Esse processo é muito útil para a mudança de base em sistemas de coordenadas de três dimensões. Além disso, no contexto deste trabalho essa técnica é utilizada para transformar entre as coordenadas da câmera para as coordenadas do mundo real.

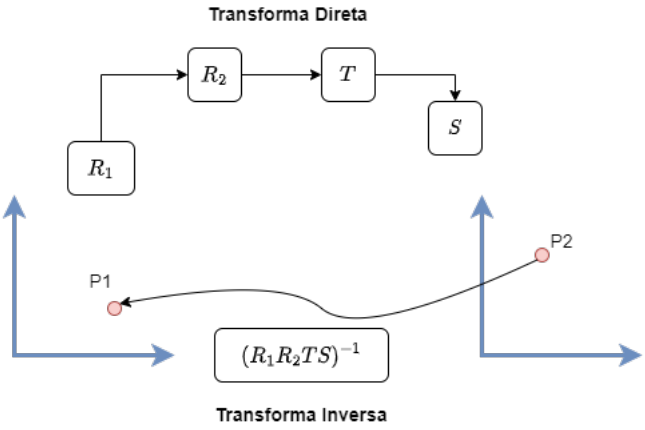
Por fim, para matrizes mais complexas, utiliza-se métodos computacionais de cálculo da matriz inversa.

2.1.7 Transformando um conjunto de pontos

As matrizes geométricas apresentadas até este momento foram expostas em exemplos transformando apenas um ponto. Porém o mesmo raciocínio pode ser aplicado para um conjunto de pontos.

Sendo P uma matriz $4 \times n$ formada por colunas com n pontos:

Figura 4 – Concatenação de várias transformações geométricas e suas matriz equivalente inversa.



fonte: Produção do próprio autor. Feito no draw.io.

$$P_{4\times n} = \begin{pmatrix} x_1 & x_2 & x_3 & \cdots \\ y_1 & y_2 & y_3 & \cdots \\ y_1 & x_2 & x_3 & \cdots \\ 1 & 1 & 1 & \cdots \end{pmatrix} \tag{28}$$

Ao multiplicar a matriz geométrica $M_{4\times4}$ por $P_{4\times n}$, tem-se uma matriz $P'_{4\times n}$ representando a transformada de cada ponto individual. Ou seja, a coluna *i-ésima* coluna $\vec{p_i} = (x_i, y_i, z_i, 1)^T$ de P' é o resultado da operação: $\vec{p'_i} = M_{4\times4} \times \vec{p_i}$.

2.1.8 Modelo da câmera *pinhole*

A Figura 5 traz uma representação didática da formação da imagem e dos elementos que a compõem: fonte de luz, reflexões na superfície do objeto, sistema ótico e plano de formação da imagem.

Pode-se modelar geometricamente a formação da imagem na câmera representando na figura Figura 6. O centro do sistema ótico é representado pelo ponto C, nele todos os raios vindo do objeto convergem, ou seja é o ponto de homotetia. Considerando os eixos da figura Figura 6 nota-se que na realidade a imagem é formada num plano π localizado em $Z < 0$. Porém para simplificar o raciocínio vamos considerar a formação da imagem em $Z > 0$.

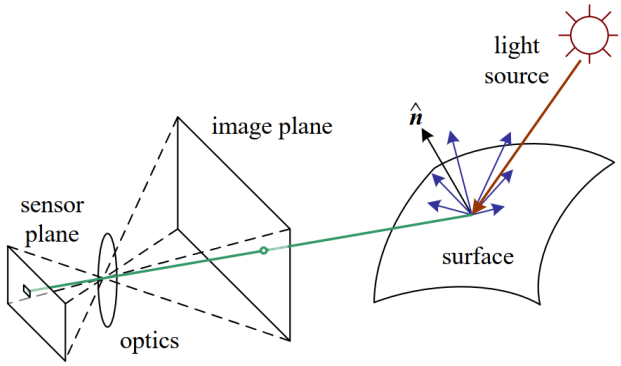
2.1.8.1 Matriz de projeção

O interesse da apresentar o modelo *pinhole* da câmera é definir uma matriz que mapeie os pontos do mundo $(X, Y, Z)^T$ para o sistema de coordenadas do sensor. Para simplificar a análise, na Figura 7 a imagem está sendo formada em $Z > 0$.

Assim, com f sendo a distância focal, tem-se a seguinte aplicação $\mathbf{R^3} \to \mathbf{R^2}$:

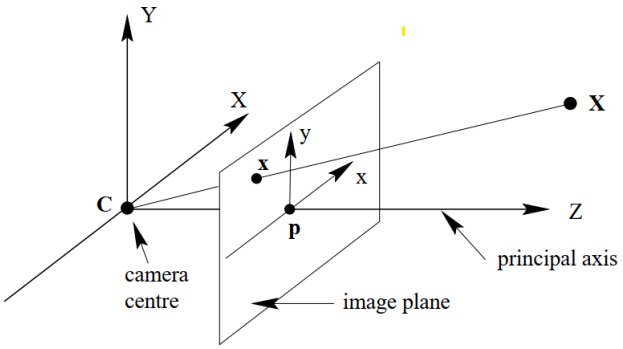
$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \end{pmatrix} \tag{29}$$

Figura 5 – Representação da formação da imagem no plano do sensor desde a origem da fonte luminosa.



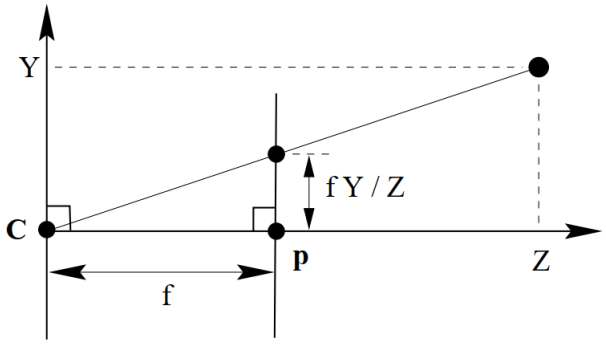
fonte: Szeliski (2022).

Figura 6 – Modelo em três dimensões da formação da imagem na câmera.



fonte: Hartley e Zisserman (2003).

Figura 7 – Plano YZ do modelo *pinhole* distância focal f.



fonte: Hartley e Zisserman (2003).

Considerando o plano de formação da imagem paralelo ao plano XY, que o sensor esteja centralizado e descartando a presença de cisalhamento, tem-se de projeção como uma matriz diagonal:

$$P = diag(f, f, 1)[I|\mathbf{0}] \tag{30}$$

A equação anterior está com notação usada por Szeliski (2022): a matriz identidade I diz que o plano de projeção não sofre rotação em relação às coordenadas globais e a matriz nula $\mathbf{0}$ diz que não há translação.

Pode-se traduzir a notação anterior no exemplo abaixo, onde observa-se a matriz P projetando os pontos do mundo $(X, Y, Z)^T$ no plano do sensor.

$$\begin{pmatrix} f_x X \\ f_y Y \\ Z \end{pmatrix} = P \times \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{31}$$

2.1.9 Matriz intrínseca da câmera

Quando o plano do sensor não é paralelo com o plano XY da Figura 6 e tampouco está centralizado no eixo Z, outros elementos podem ser inseridos na matriz de projeção P.

Para modelar a posição central do sensor, utiliza-se o vetor $(C_x, C_y, 1)^T$ como última coluna de P. Adiciona-se também um fator s de cisalhamento. Assim, obtém-se a matriz K intrínsica da câmera:

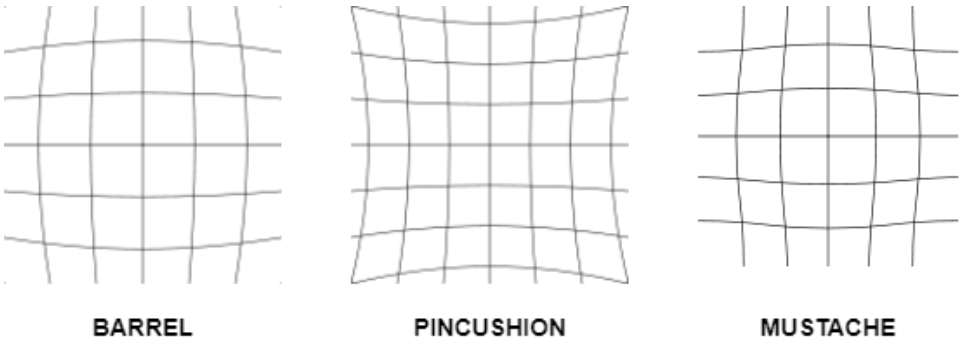
$$K = \begin{pmatrix} f_x & s & 0 & C_x \\ 0 & f_y & 0 & C_y \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{32}$$

Essa matriz pode ser chamada também de matriz de calibração.

2.1.10 Distorções óticas

Até este momento a formação da imagem foi modelada geometricamente considerando apenas o plano do sensor e a distância focal. De mesma forma, o jogo de lentes produz uma distorção radial e tangencial na imagem. Neste trabalho será abordada apenas a distorção radial.

Figura 8 – Três tipos de distorções óticas.



fonte: Wikipedia contributors (2022).

Existem três tipos de distorções óticas radiais:

- *Barrel*: também chamada de distorção de barril. As extremidades tendem a ficar mais próximas do centro.
- *Barrel*: também chamada de distorção de almofada. Nesse caso as extremidades longe do centro são acentuadas.

• *Mustache distortion*: pouco presente na maioria das áreas de aplicação.

As distorções radiais neste documento seguirão as notações de Bradski (2000). Assim, partindo do centro da imagem, um ponto distorcido segue a seguinte lei:

$$\begin{cases} x_{dist} = x \times L(r) \\ y_{dist} = y \times L(r) \end{cases} \tag{33}$$

Sendo r a distância radial do centro até o ponto (x, y):

$$r = \sqrt{(x - x_x)^2 + (y - y_c)^2} \tag{34}$$

E $L(r)$ é o modelo de distorção radial, que pode ser aproximado usando series de Taylor, como elaborado por Alvarez, Gomez e Sendra (2010):

$$L(r) = 1 + \sum_{i=1} k_i r^{2i} = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots \tag{35}$$

Por fim tem-se os pontos (x_{dist}, y_{dist}), representando a distorção radial com referencia a distância do centro da formação da imagem:

$$\begin{cases} x_{dist} = x \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{dist} = y \times (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{cases} \tag{36}$$

2.1.10.1 Calibrar a câmera

Calibrar uma câmera significa encontrar os parâmetros que foram modelados nas seções 2.1.10 e 2.1.9. Inicialmente, busca-se o conjunto de valores $\{f_x, f_y, s, C_x, C_y\}$ para a matriz intrínseca e o conjunto $\{k_1, k_2, k_3\}$ para a distorção ótica.

O conjunto $S_{calib} = \{f_x, f_y, C_x, C_y, k_1, k_2, k_3\}$ é considerado constante para a câmera, ou seja, só há a necessidade de encontra-los apenas uma vez. Nota-se que o parâmetro s de cisalhamento foi descartado, pois não será considerado no desenvolvimento deste relatório.

Dessa forma, após encontrar o conjunto S_{calib}, deve-se corrigir a imagem capturada usando as equações apresentadas anteriormente. Faz-se necessário salientar que a correção só é necessária em aplicações que requerem alta precisão do sistema de visão. Por exemplo, uma linha de inspeção por imagem que computa a distância física entre dois elementos. Em outras aplicações mais simples, como câmeras de segurança, não há necessidade de calibração.

2.1.11 Matriz extrínseca da câmera

Até este momento a câmera foi considerada estando centralizada nas coordenadas do mundo. A posição da câmera com relação as coordenadas globais pode ser modelada por uma translação e/ou uma rotação em três dimensões.

Essa movimentação ao longo das coordenada do mundo pode ser representada pela matriz E chamada de Matriz Extrínseca da Câmera.

Na figura Figura 9 tem-se um exemplo de uma câmera que parte do centro das coordenadas globais e se movimenta realizando duas transformada geométricas afins: uma translação no plano XY e uma rotação no eixo Z. Seguindo o principio da concatenação de transformações essa matriz extrínseca pode ser escrita da seguinte forma:

$$E = T_{XY} \times R_Z = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{37}$$

Desenvolvendo a matriz:

$$E = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & T_x \\ \sin(\alpha) & \cos(\alpha) & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{38}$$

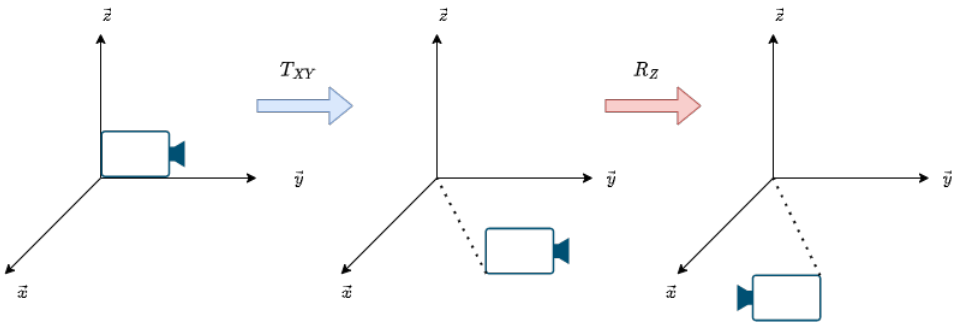
Muitos autores escrevem a matriz extrínseca como $E = (R|T)$, representando a rotação e a translação. Sendo R e T respectivamente a matriz 3×3 de rotação e translação não-homogêneas. Neste relatório poderá ser utilizada essa notação, porém na forma homogênea:

$$E = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \tag{39}$$

Nota-se também que quando a câmera está alinhada com o sistema de coordenadas globais, pode-se descrever a matriz extrínseca como uma matriz identidade:

$$E = \begin{pmatrix} I & 0 \\ 0 & 1 \end{pmatrix} \tag{40}$$

Figura 9 – Exemplo de deslocamento da câmera em relação às coordenadas globais.



fonte: Produção do próprio autor. Feito no draw.io.

Como exemplo, na Figura 9 a câmera translada primeiramente no plano XY. Na sequência, a câmera rotaciona $180°$ no eixo $\vec{z}$. Para esse caso a matriz extrínseca é $E = T_{XY} R_Z$.

2.1.12 Coordenadas do mundo para coordenadas da câmera

Sendo $P = (X, Y, Z)^T$ um ponto definido com relação às coordenadas globais $(\vec{x}, \vec{y}, \vec{z})$. Tendo uma câmera com matriz intrínseca de projeção K e com posição definida pela matriz extrínseca E, pode-se calcular a projeção dos pontos P no plano de imagem $(\vec{i}, \vec{j})$ apenas utilizando as matrizes K e E.

Mais precisamente, pontos no espaço são projetados no plano do sensor multiplicando-se $(K \times E)P$. Além disso, sabendo que os pontos são descritos por coordenadas homogêneas $(X, Y, Z, 1)$ faz-se necessário introduzir o operador $\sim$, que representa a projeção de coordenadas homogêneas no plano da imagem. Por exemplo:

$$(X', Y')^T \sim (X', Y', Z')^T \tag{41}$$

Este operador é extremamente útil para desenvolver o problema da projeção de pontos em três dimensões no plano da imagem. Dessa forma, o ponto P' formado no sensor da câmera pode ser calculado pela seguinte projeção:

$$P' \sim (\underbrace{K}_{\text{Intrínseca}} \times \underbrace{E}_{\text{Extrínseca}})P \tag{42}$$

Utilizando as matrizes dos exemplos 38 e 32, tem-se a projeção:

$$P' = \begin{pmatrix} X' \\ Y' \end{pmatrix} \sim \begin{pmatrix} f_x & 0 & 0 & C_x \\ 0 & f_y & 0 & C_y \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & T_x \\ \sin(\alpha) & \cos(\alpha) & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{43}$$

Finalmente:

$$\begin{cases} X' = [f_x(\cos\alpha - \sin\alpha + T_x) + C_x]X \\ Y' = [f_y(\sin\alpha + \cos\alpha + T_y) + C_y]Y \end{cases} \tag{44}$$

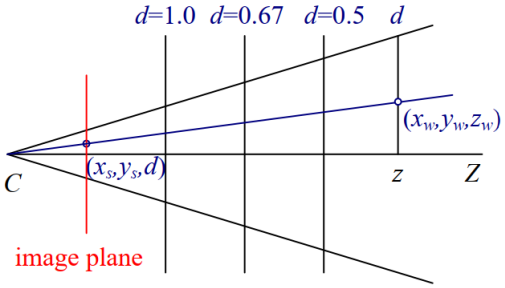
Na Figura 10 pode-se observar que diversos pontos em profundidades diferentes são formados no mesmo ponto do plano da imagem. Desta forma, a noção de profundidade do ponto de vista do plano do sensor é irrelevante, pois imageando com uma câmera mono, o valor de Z é perdido.

Concluindo, o operador $\sim$ faz-se útil pois diferentes pontos sobre a mesma linha que passa pelo ponto central C, são vistos pelo sensor como pontos iguais. Dessa forma, reduzindo a projeção para apenas duas dimensões simplifica-se a modelagem e o equacionamento de problemas mais complexos.

2.1.13 Projeção em duas câmeras

Esta seção destina-se a estudar apenas as situações onde duas câmeras distintas possuem os mesmo centro de formação de imagem. Geometricamente falando, quando um plano de formação é rotacionado em relação ao outro. Ou seja, quando a reta perpendicular ao centro do plano pi_1 da câmera 1 intersecta

Figura 10 – Problema da paralaxe na formação da imagem.

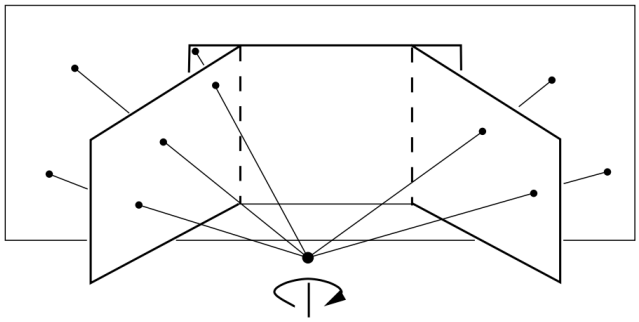


fonte: Hartley e Zisserman (2003).

a reta normal ao centro do plano pi_2 da câmera 2 e esse "cruzamento"ocorre atrás dos planos de formação da imagem.

A Figura 11 exemplifica a situação descrita acima. Nela está representado um exemplo de vários plano rotacionando em torno do mesmo eixo. Para fins práticos, neste relatório será estudado apenas a situação em que os pontos projetam a suas respectivas imagens no ponto de convergência. Ou seja, o ponto no de rotação é o "pinhole"de todas as câmeras.

Figura 11 – Rotação em torno de um eixo e projeção de diferentes pontos.



fonte: Hartley e Zisserman (2003).

Faz-se necessário discriminar o exemplo anterior, pois a câmera poderia se deslocar no espaço de acordo com a Figura 12, que descreve uma geometria epipolar.

A geometria epipolar está representada na Figura 12. Nesse caso, conhecendo o plano formado pelos "pinholes"das câmera e o ponto X pode-se mapear um plano π. O ponto x na câmera da esquerda encontrará o seu equivalente x' sobre a reta I na câmera da direita. A relação entre esses dois pontos é dada por $x'Fx = 0$, sendo F a matriz fundamental da geometria epipolar

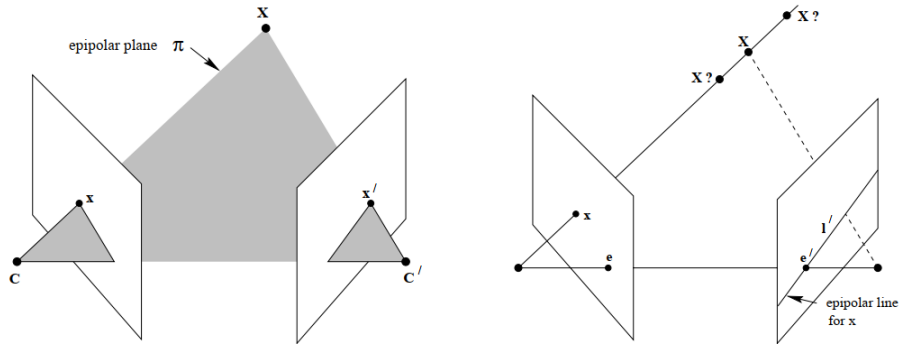
O caso da geometria epipolar, mesmo sendo de grande interesse quando se estuda múltiplas projeções, não será abordado neste trabalho.

2.1.13.1 Projeção em duas câmeras diferentes

Dizer que duas câmeras são diferentes significa dizer que possuem matrizes intrínsecas diferentes: K_1 e K_2.

Para a situação da Figura 11, considera-se que a câmera inicial está orientada e centralizada nas coordenadas globais. Ou seja, possui matriz extrínseca $E_1 = I$.

Figura 12 – Exemplo de geometria epipolar entre as câmeras



fonte: Hartley e Zisserman (2003).

A segunda câmera é rotacionada ao longo do eixo Z no espaço. Sendo assim a matriz extrínseca E_2 pode ser escrita:

$$E_2 = \begin{pmatrix} R_Z & 0 \\ 0 & 1 \end{pmatrix} \tag{45}$$

Sendo $P = (X, Y, Z)^T$ um ponto qualquer no exterior do cilindro formado pela rotação das câmeras.

A projeção de P no plano π_1 da câmera 1 é o ponto x_1:

$$x_1 \sim (K_1 E_1)P = (K_1 I)P = (K_1)P \tag{46}$$

A projeção de P no plano π_2 da câmera 2 é o ponto x_2:

$$x_2 \sim (K_2 E_2)P \tag{47}$$

Equacionando 46 e 47:

$$x_2 \sim K_2 E_2 (K_1)^{-1} x_1 \tag{48}$$

Em termos práticos, tem-se a posição de um ponto na foto da câmera 2 conhecendo as coordenadas do ponto equivalente na câmera 1.

Ademais, sabendo-se que E_2 é uma simples rotação e que ao projetar em π_2 não é necessário utilizar a notação homogênea, pode-se representar a matriz extrínseca da seguinte forma: $E_2 = R_Z$, ou seja, apenas a rotação em torno de Z. Finalmente obtendo a relação entre x_1 e x_2:

$$x_2 \sim K_2 R_Z (K_1)^{-1} x_1 \tag{49}$$

2.1.13.2 Projeção em duas câmeras idênticas

Para duas câmeras idênticas, tem-se $K_1 = K_2 = K$.

Assim a equação 51 fica:

$$x_2 \sim K E_2 (K)^{-1} x_1 \tag{50}$$

Considerando E_2 como uma rotação simples:

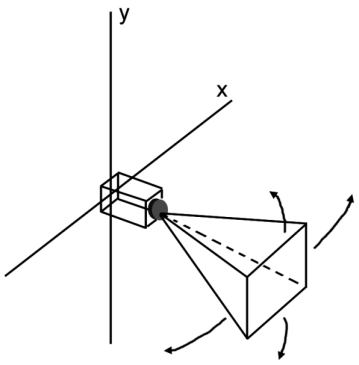
$$x_2 \sim K R_Z(K)^{-1} x_1 \tag{51}$$

A matriz $H = K R_Z(K)^{-1}$, pode ser chamada também de homografia que mapeia o ponto x_1 no plano π_2.

2.1.14 Câmera *pan-tilt*: rotação em torno de um ponto

Uma câmera do tipo *pan-tilt*,também chamda de PTU(do inglês *pan-tilt unity*, pode ser caracterizada por um sistema de visão centralizado nas coordenadas globais, que rotaciona nos eixos x e y. A Figura 13 representa esse tipo de câmera.

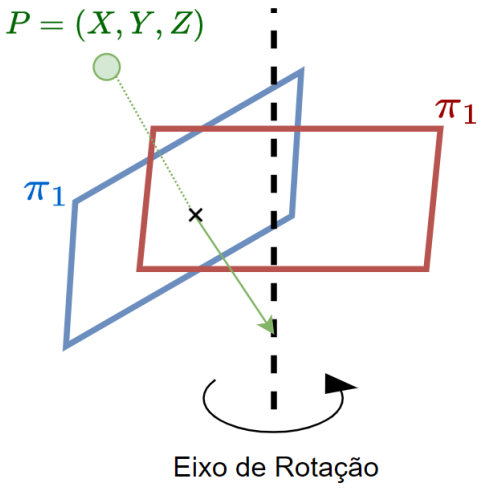
Figura 13 – Exemplo de câmera *pan-tilt* que realiza rotação em torno do eixo x e em torno do eixo y.



fonte: Davis e Chen (2003).

Além disso pode-se ter uma compreensão visual do problema na Figura 14 que representa a formação da imagem na câmera e um ponto P sendo projetado em dois *frame* que rotacionam. Nota-se que no *frame* π_1 a projeção de P está centralizada, enquanto que em π_2 está mais afastado.

Figura 14 – Ponto nas coordenadas globais sendo projetado em dois plano de rotação.



fonte: Produção do próprio autor. Feito no draw.io.

Com relação a matriz de transformada, sendo α uma rotação em torno do eixo x e β uma rotação em torno do eixo y, tem-se a matriz composta $R_{\alpha\beta} = R_\alpha R_\beta$:

$$R_{\alpha\beta} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ \sin(\alpha)\sin(\beta) & \cos(\alpha) & -\sin(\alpha)\cos(\beta) & 0 \\ -\cos(\alpha)\sin(\beta) & \sin(\alpha) & \cos(\alpha)\cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{52}$$

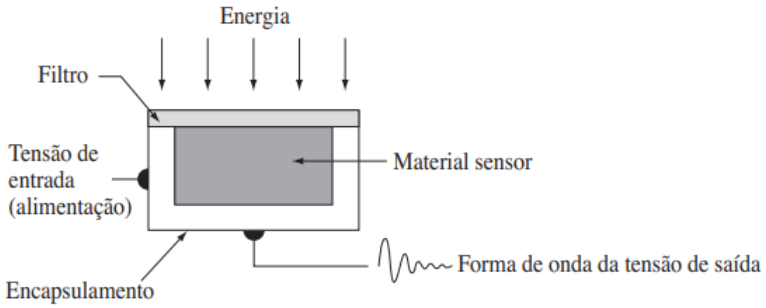
Concluindo, para uma câmera PTU, pode-se considerar o mapeamento dos pontos para diferentes *frames* como a projeção a seguir:

$$x_2 \sim K_1 R_{\alpha\beta} (K_1)^{-1} x_1 \tag{53}$$

2.2 SENSORES E A FORMAÇÃO DA IMAGEM DIGITAL

Os diferentes tipos de sistema de imageamento podem ser resumidos por uma por sensores que modificam as suas características elétricas quando recebem um tipo de energia. A figura Figura 15 é uma representação geral para os diferentes tipos de sensores, portanto poderia-se utiliza-la para descrever o funcionamento de um sistema de ultrassom ou de raio-X. Entretanto o intuito desta seção é estudar a formação da imagem em sensores sensores óticos correspondentes à faixa espectral do olho humano (380 à 780 nm) .

Figura 15 – Modelo geral de unidade foto-sensível representando os diversos tipos de sensores de imagem.



fonte: Gonzalez e Woods (2000).

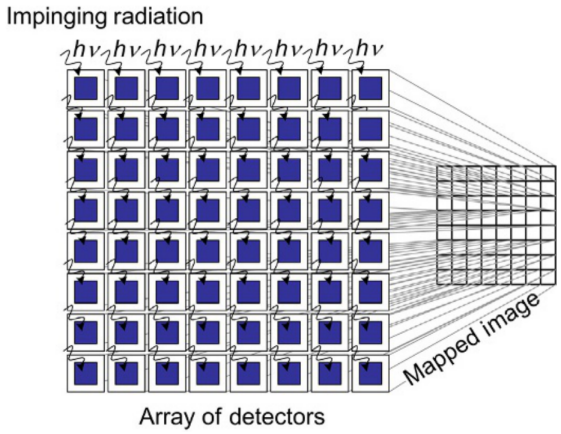
Ademais o modelo da Figura 15 caracteriza uma unidade de sensor que retorna um sinal analógico. Sendo um fotosensor, tem-se na saída uma tensão proporcional à entrada de luz na área do sensor. Esse modelo pode ser considerado como um pixel e para gerar uma imagem com resolução suficiente, deve-se construir uma matriz enfileirando fotosensores como mostrado na imagem Figura 16.

Dessa forma, as imagens digitais são geradas pela incidência de uma energia luminosa sobre uma matriz de sensores foto-sensíveis. Como apresentado anteriormente, os raios vindo da cena externa devem ser condicionados por meio de um sistema óptico que possui a função de centralizar os feixes de luz sobre a área do sensor.

A Figura 17 mostra todas as etapas da formação da imagem digital: o objeto refletindo uma fonte luminosa, os raios de luz passando pelo sistema óptico e a imagem digital sendo formada no sensor.

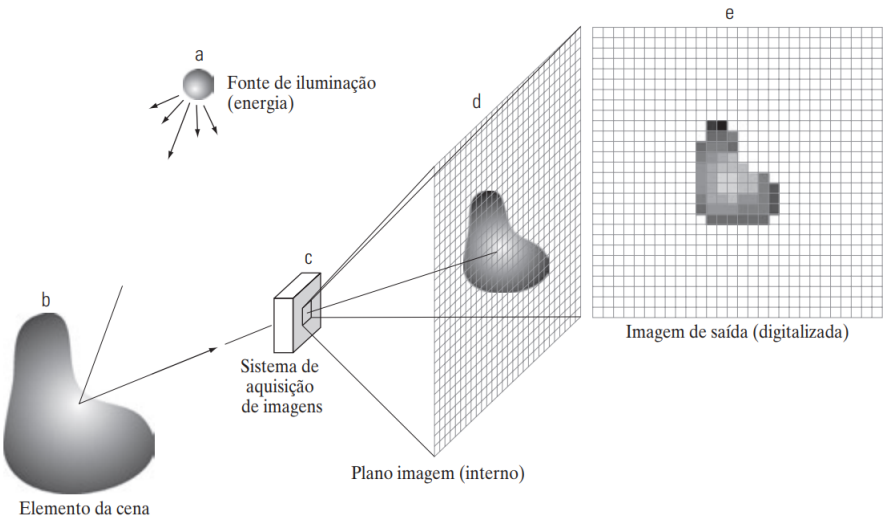
As câmeras digitais modernas possuem sensores semicondutores (Si) de dois tipos: CCD e CMOS.

Figura 16 – Representação da matriz de fotosensores organizada em colunas.



fonte: Durini (2019).

Figura 17 – Processo completo de aquisição da imagem no sensor.



fonte: Gonzalez e Woods (2000).

O sensor CCD (*charge-coupled device*) foi até os anos 90 o padrão da industrial de imageamento digital, de acordo com Durini (2019). O CCD é um *chip* de silício com fileiras de fotodiodos. Nesse dispositivo a imagem é formada transferindo a carga de uma célula até a o final da fileira, como um registrador de deslocamento. Por fim, o sensor do tipo CCD é atualmente muito utilizado na astronomia e em câmeras NIR (*near infraed*).

Para as outras áreas que demandam um sistema de visão, utiliza-se mais o sensor do tipo CMOS como relatado por Rashidian (2011). Esse sensor será abordado neste relatório pois é o tipo da câmera utilizada no projeto.

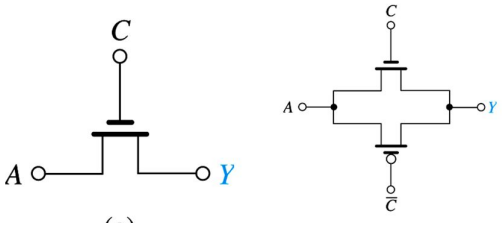
2.2.0.1 Tecnologia CMOS

A tecnologia CMOS (*complementary metal oxide semiconductor*) é amplamente utilizada como *switch* ou inversor em circuitos integrados devido suas características elétricas e tamanho reduzido. O CMOS é uma combinação de dois transistores MOSFET como mostra a Figura 18.

Na Figura 18 tem-se à esquerda o símbolo do transistor NMOS controlando a tensão AY e à direita

um exemplo de CMOS (formado por dois FET) fazendo o mesmo controle de tensão.

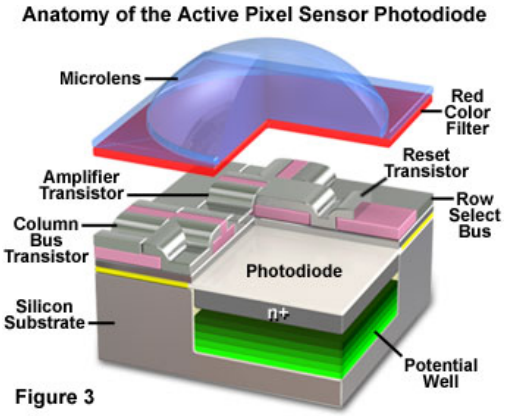
Figura 18 – Símbolo do NMOS e CMOS



fonte: Sedra et al. (2004).

Nas câmeras CMOS o raio de luz entra por uma microlente e ao atingir o fotodiodo, a unidade de sensor armazena uma carga proporcional à incidência luminosa. Este sensor está representado na Figura 19

Figura 19 – Representação de todos os elementos da unidade de sensor de uma câmera CMOS.



fonte: Turchetta e Spring (2023).

Uma das desvantagens do CMOS em relação à tecnologia CCD é a quantidade de raios incidentes no fotodiodo. Visto que há uma camada de substrato entre a microlente e o sensor, o feixe de luz é reduzido. Uma das soluções para esse problema é inverter a posição da camada de metal como mostra a Figura 62 do Anexo.

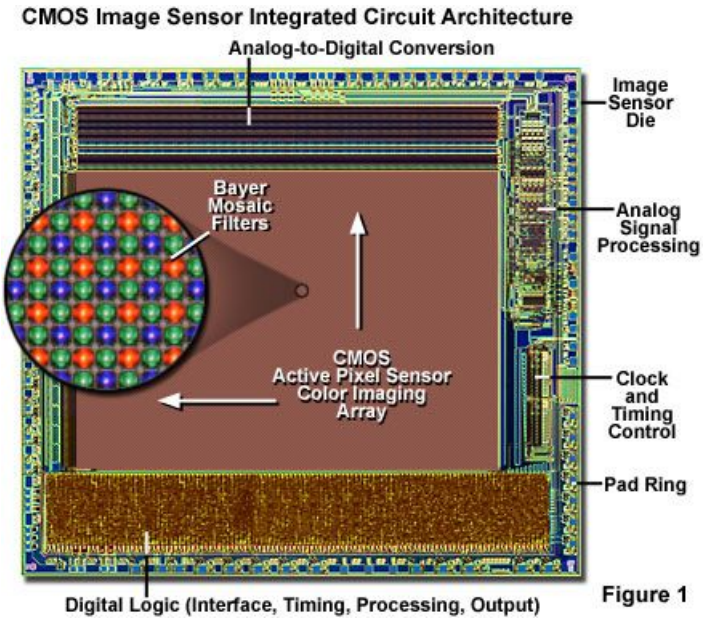
Para se obter uma imagem colorida, a técnica utilizada pelos sensores mais baratos é de filtrar o espectro de cor após que o raio passou pela microlente. A disposição dos filtros geralmente é de dois píxeis verdes para um vermelho e um azul. Forma-se então uma grade de cores chamada de mosaico de Bayer. Para se obter a imagem final basta interpolar os quatro sinais coloridos; essa operação é chamada de *demosaicing*.

Na Figura 20, ampliando a região fotossensível, observa-se a disposição dos filtros de cor na forma de mosaico de Bayer.

2.2.0.2 *Rolling shutter* e *global shutter*

A Figura 63 do Anexo mostra o processo de aquisição da imagem. Na operação chamada de *rolling shutter*,a primeira fileira de píxeis é exposta a luz e seus sinais são enviando para o ADC. Na sequência, a segunda fileira de píxeis é submetida ao mesmo processo e assim por diante. Dessa forma as fileiras

Figura 20 – Foto amplificada de um circuito integrado de sensor CMOS indicando seus principais componentes.



fonte: Turchetta e Spring (2023).

da imagem formada não são expostas ao mesmo tempo. Quando a imagem é considerada estática, não há problema. Porém quando faz-se o imageamento de um objeto móvel, o sensor *rolling shutter* não é suficiente, pois gera imagens com efeito de cisalhamento.

Para contornar o problema das fileiras não estarem expostas ao mesmo tempo, existe a tecnologia chamada *global shutter*, que inclui um capacitor junto ao fotodiodo para armazenar por mais tempo o sinal luminoso recebido, como apresentado por Musa (2018). Dessa forma, todas as fileiras são exposta ao mesmo tempo, porém o envio dos seus sinais é enviado um de cada vez ao ADC. Dessa forma, pode-se obter fotos sem distorções de objetos em deslocamento.

2.3 PROCESSAMENTO DE IMAGENS

Esta seção introduz a teoria das duas etapas principais do algoritmo de processamento de imagens usado neste projeto. Primeiramente é apresentada a técnica de definir se uma imagem está desfocado usando o operador Laplaciano; por fim, é estudado como fazer o rastreio de objetos numa sequência de imagens.

2.3.1 Identificar imagem borrada

Quando a cena, o objeto ou a câmera estão em constante deslocamento a imagem digital é formada com um aspecto de borrão. Esse fenômeno é conhecido também como *motion blur*.

Pode-se detectar uma imagem borrada utilizando-se técnicas convencionais de processamento de imagens e estatística. De acordo com Pech-Pacheco et al. (2000), uma métrica simples é calcular a variância do Laplaciano e assim definir empiricamente um limiar para as fotos borradas.

O Laplaciano é uma medida da variação de intensidade de cor em uma imagem e é calculado aplicando uma máscara de filtro a uma imagem. A variância do Laplaciano é uma medida da uniformidade

da informação: quanto menor for, mais borrada é a imagem.

Sendo $I(x, y)$ a intensidade de píxel numa imagem de canal único, o operador Laplaciano em $\mathbf{R^2}$ aplicado a I pode ser escrito como:

$$\text{Laplaciano}(I) = \nabla^2(I) = \Delta(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{54}$$

Escolhendo a janela de convolução k:

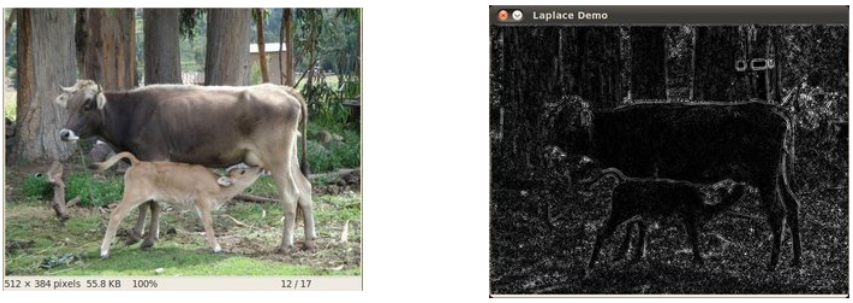
$$k = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{55}$$

O Laplaciano, ilustrado na Figura 21, é encontrado convoluindo a imagem discreta $I[x, y]$ pelo *kernel* k:

$$L[x, y] = I[x, y] * k \tag{56}$$

.

Figura 21 – Imagem original e seu respectivo Laplaciano.



fonte: Bradski (2000).

Deve-se definir empiricamente para cada situação um valor limite da variância que representa uma imagem desfocada.

Por fim, o método da variância do Laplaciano permite que os *frames* borrados sejam detectados e removidos antes de processar a imagem, garantindo melhores resultados em aplicações que requerem imagens nítidas, como o rastreamento de objetos ou a análise de imagens médicas.

2.3.2 Rastrear objetos numa sequência de imagens

Esta seção introduz teoricamente os diferentes tipos de algoritmos de rastreio. Inicialmente, os métodos tradicionais de detecção de objetos são apresentados, em sequência a importância dos descritores locais para o rastreio é discutida e finalmente o algoritmo CSRT é estudado.

2.3.2.1 Diferença entre detectar e rastrear

Detectar e Rastrear são duas áreas do Processamento de Imagens muito amplas, com algoritmos e soluções provenientes de diversos campos de estudo. Mesmo sendo muito parecido, esses dois temas

se distinguem em relação as suas aplicações. Enquanto que "detectar"implica apenas em encontrar um *template* (também chamado na literatura de 'classe') em uma imagem desconhecida, "rastrear"significa buscar esse objeto numa sequência de *frames*.

Um algoritmo de rastreio pode englobar diferentes tipos de estratégias para seguir a posição de um objeto. Pode-se calcular o fluxo ótico (FLEET; WEISS, 2006) e estimar o vetor deslocamento; pode-se buscar a posição de máximas entre *frames* (EXNER et al., 2010); ou mesmo usar um filtro preditivo de Kalman (DELLAERT; THORPE, 1997).

Na prática o *tracking* apresenta muito mais desafios, pois a localização deve ser contínua ao longo de um vídeo. Os principais problemas que podem aparecer são a oclusão momentânea do objeto e o deslize indesejado da região de rastreio.

Entretanto as soluções de rastreio podem ser mais rápidas, pois ao longo do vídeo tem-se conhecimento de informações prévias do objeto, por exemplo a sua posição e o seu vetor deslocamento. Dessa forma, pode-se prever no próximo quadro a posição de interesse mesmo quando o objeto sofre uma oclusão na cena.

Outras métodos mais complexos foram desenvolvidas para resolver o problema de detecção. Como, por exemplo, usar técnicas de estatística e processamento de sinais, utilizar descritores locais da imagem e até mesmo aprendizagem profunda. Mesmo sendo bem distintas, para todas essas estratégias o problema da detecção pode ser definido formalmente como a busca de uma classe em uma imagem desconhecida, chamada de *frame* de busca.

Dessa forma, Murty e Devi (2015) caracteriza um algoritmo de detecção por três etapas: primeiro define-se o *template*, em seguida faz-se o *matching* na imagem de busca e finalmente localiza-se geometricamente o objeto.

2.3.2.2 Técnicas tradicionais de rastreio

O método mais tradicional e conservador, computacionalmente falando, de detecção e rastreio é segmentar[1] por canal de cor. Essa técnica consiste em encontrar um espaço de cor onde o objeto possa ser facilmente localizado, ou seja, um canal composto apenas pelo *template*. Dessa forma, fazendo uma binarização simples com *threshold* bem definido, o objeto pode ser rastreado.

Caso o resultado dessa binarização retorne uma imagem com componentes não conectados, pode-se selecionar apenas a maior área ou utilizar-se de filtros morfológicos.

No ramo da visão industrial, que é a aplicação do visão computacional em processos fabris, essa técnica tradicional de detecção por espaço de cor é muito utilizada. Como os processos industriais requerem maior confiabilidade e repetibilidade, as inspeções [2] [3] devem ser mais robustas possíveis. Dessa maneira, em ambiente controlado pode-se escolher a iluminação, lente, filtro e cor do BG que realce o objeto. Simplificando, assim, a complexidade da computação.

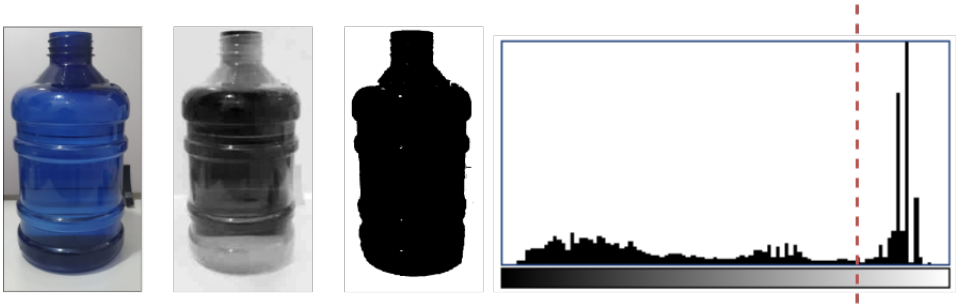
---

[1] Assim como Detecção e Rastreio, a Segmentação é uma área de estudo muito vasta da ciência do Processamento de Imagens

[2] Nas câmeras processadas da Cognex, existe a funcionalidade de segmentar por espaço de cor. Material da empresa disponível em <https://www.cognex.com/products/machine-vision/vision-tools/rule-based-tools/color-tools>

[3] A empresa ESMaquinas possui um sistema classificador de cenouras que passam em esteiras azuis. A escolha da cor do fundo e da iluminação correta simplificam o processo de segmentação. Material institucional disponível em <https://esmaquinas.com/>

Figura 22 – Exemplo de segmentação de imagem usando o espaço de cor Lab.



fonte: Produção do próprio autor. Feito com ImageJ (SCHINDELIN et al., 2012).

Na figura Figura 22, a primeira foto é a imagem original, a segunda é o canal 'b' da imagem no espaço Lab e a terceira é a imagem binarizada. O gráfico de histograma representa a distribuição de píxeis do canal 'b' e a linha tracejada é o *thrshold* escolhido manualmente. Nota-se que a informação está a direita desse limiar.

Além da técnica da segmentação por espaço de cor, um dos métodos mais tradicionais de detecção é fazer uma varredura do *template* na imagem de busca, como apresentado por Fisher e Oliver (1995). O sinal 2D resultante apresentará máximas e mínimas que indicam a posição do objeto na imagem onde houveram maiores correspondências. Esse método, evidentemente, é robusto apenas para variações de translação. Quando há rotação, mudança de escala ou de projeção essa técnica é pouco efetiva.

Essa estratégia da convolução é comumente referida por *cross-correlation*, pois é inspirada na técnica homônima de processamento de sinais. Além da correlação cruzada outras métricas podem ser usadas, como por exemplo o quadrado da diferença ou a correlação de Pearson. Por fim, o OpenCV disponibiliza uma função [4] que realiza esse tipo de detecção de objeto encapsulando também várias métricas.

Na Figura 23, tem-se à esquerda a imagem de busca e o *template* é cabeça do animal. À direita o resultado da operação com o local de máxima indicado. Nesse caso foi aplicado o método da correlação-cruzada normalizada.

Figura 23 – Exemplo de *template matching* usando filtro de correlação.

fonte: Adaptado de Bradski (2000).

[4] Função *matchTemplate()*. Disponível em <https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html>

2.3.2.3 Bons pontos para se rastrear

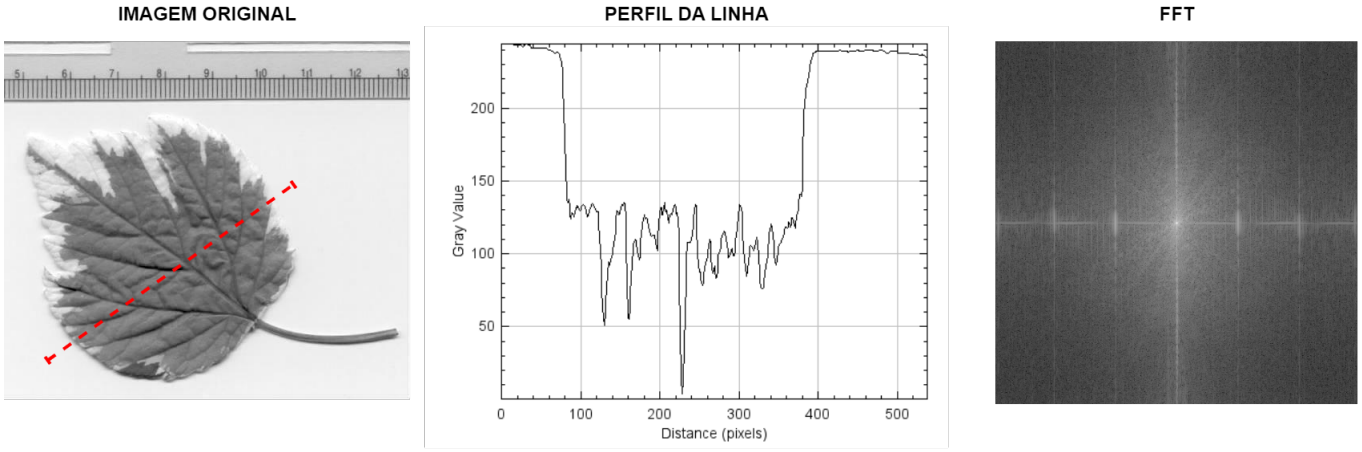
Quando procura-se pontos marcantes em sinais de uma dimensão, as técnicas convencionais são encontrar pontos de máxima ou mínima, inflexões ou até mesmo estudar as suas harmônicas. Em se tratando sinais de duas dimensões, pode-se fazer um estudo análogo para encontrar pontos de interesse em imagens.

Empiricamente , o que chama atenção aos olhos humanos numa cena pode ser uma reta, uma quina ou até mesmo um ponto. Matematicamente, esses exemplos podem ser descritos como locais na imagem com alta frequência ou como. Esse conceito é fácil de compreender quando se visualiza o perfil de um traço que cruza duas regiões da foto com intensidades muito distintas.

A Figura 24 mostra a foto de uma folha onde foi traçado um perfil cruzando áreas da imagem com diferentes intensidades. O perfil do traço está representado no gráfico central e nota-se que nas transições do BG para a folha há uma variação brusca do sinal, caracterizado por ser de alta frequência. Também sobre a folha, a frequência tem um papel importante pois há a presença de textura.

O último quadro da Figura 24 mostra a folha sendo representada frequencialmente no espaço de Fourier [5]. Com essa representação, pode-se fazer um estudo análogo ao das harmônicas em uma dimensão. Porém no caso das imagens, a informação está contida no centro do *k-space* e as frequências da imagem original aparecem ao se afastar radialmente.

Figura 24 – Foto de folha com perfil traçado e representação da imagem no espaço de Fourier.



fonte: Produzido pelo autor.

Posto isso, assim como para sinais de uma dimensão, os operadores equivalentes da primeira e da segunda derivada são muito úteis para detectar pontos chaves. Como exemplo os operadores Laplaciano, Gradiente, HoG, DoG estão estão presentes na maioria do s algoritmos analíticos de *local features*.

Do ponto de vista do *tracking*, de acordo com Shi et al. (1994), [6] uma estratégia é detectar pontos de interesse na imagem e rastreá-los. É importante destacar que para uma melhor acurácia esses pontos devem ser invariantes à todos os tipos de transformações geométricas, variações de iluminação e cor. Por exemplo, um descritor local robusto deve ser capaz de ser identificado em diferente quadros quando o módulo PTU for rotacionado em torno de um eixo.

[5] Também chamado de *k-space*.

[6] O título desta seção foi inspirado no trabalho homônimo de Shi et al. (1994) sobre rastreio usando *local features* .

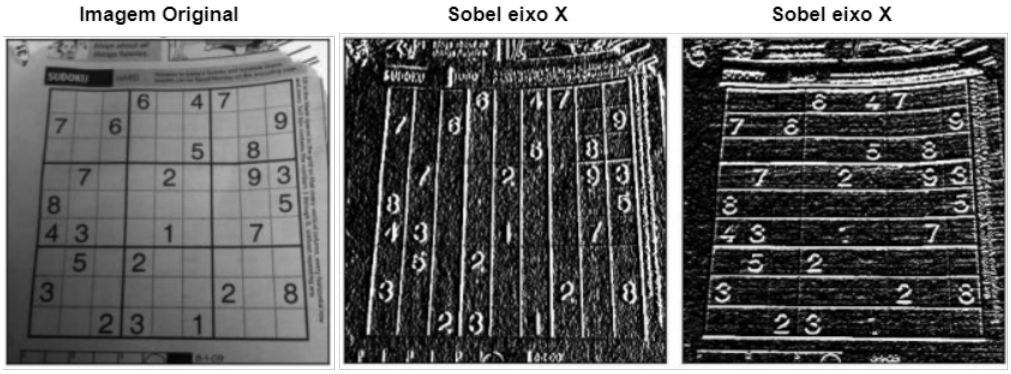
Detectores de borda são um dos métodos mais antigos e que ainda possuem grande utilidade para evidenciar características em imagens. Os exemplos mais conhecidos são o filtro de Sobel e de Prewit. Ambos os métodos são baseados em matrizes de convolução bem escolhidos para realçar linhas na horizontal ou na vertical.

As matrizes de convolução do filtro de Sobel estão apresentadas a seguir:

$$S_x = \frac{1}{2}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \; ; S_y = \frac{1}{2}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{57}$$

A Figura 25 mostra o processamento de uma imagem contendo uma grelha do jogo Sudoku usando o filtro de Sobel. Nesse exemplo foi realizado dois processamentos sobre a foto original para realçar separadamente as linhas horizontais e verticais. Além disso, na prática a magnitude do Gradiente discreto é computado como a soma dessas duas imagens.

Figura 25 – Aplicação do filtro de Sobel na horizontal e vertical.



fonte: Adaptado de (BRADSKI, 2000).

Com relação aos descritores locais, há dois grupos principais: os algoritmos baseados em esquinas e aqueles baseados em *blobs*.

Esquinas em imagens podem ser encontradas usando o algoritmo de Shi et al. (1994) ou Derpanis (2004).

As técnicas de *blob* buscam detectar pontos na imagem com propriedades específicas que se destacam de sua região, por exemplo variações na frequência, cor e iluminação. Essas características podem ser visualmente pouco distinguíveis, porém quando modeladas matematicamente são facilmente detectadas. As filtragens de *blob* são baseadas na correlação espacial de um *kernel* específico que resulta num espaço chamado de *features-space*, representando os descritores locais. Como exemplo tem-se os algoritmos de Diferença de Gaussiano (DoG), Diferença de Laplaciano (LoG) e Diferença de Hessiano (LoG).

Além desses algoritmos, é importante ressaltar a existência de descritores locais mais complexos como o SIFT desenvolvido por Lowe (2004). Esse método encapsula técnicas diversas para tornar os pontos-chave robustos à maioria dos tipos de variações. Por exemplo, para representar a mudança de escala foi utilizada uma abordagem piramidal para filtragem com o DoG; para descrever rotações e

iluminação, foi utilizada uma simplificação do histograma dos gradientes (HoG). Por fim, tem-se um vetor com 128 colunas representando as propriedades do ponto-chave.

Além do SIFT os algoritmos SURF (BAY; TUYTELAARS; GOOL, 2006) e ORB (RUBLEE et al., 2011) são alternativas para se extrair *local features*. Usando *Deep-Learning* tem-se a rede *Super Point* (DETONE; MALISIEWICZ; RABINOVICH, 2018).

2.3.2.4 Rastrear usando descritores locais

A principal ideal de detectar um objeto usando *local features* é de extrair pontos-chave que possam ser descritos matematicamente por um vetor de escalares. Esse vetor pode ser chamado de descritor local e representa um ponto não mais pela intensidade do pixel, mas por uma classe de características.

A vantagem evidente de usar descritores locais é que se reduz o custo e complexidade computacional para se comparar duas imagens, pois com uma lista de vetores pode-se representar pontos chaves em uma imagem.

Na prática, compara-se duas imagens escolhendo seus respectivos pontos-chave e extraindo seus descritores locais. Assim, faz-se o casamento desses vetores como métrica distâncias entre as classes de cada imagem. Por exemplo para o SIFT utiliza-se a norma L^2 para o ORB, a distância de Hamming.

Esse método de casamento de pontos retorna uma lista com pares de coordenadas que deram *match*. Assim, a localização do *template* é geralmente realizada calculando a matriz de homografia inversa.

Sendo $\vec{a}$ coordenadas da imagem, $\vec{b}$ pontos correspondentes no *template* e H a matriz de homografia, o problema problema inverso é posto a seguir:

$$\vec{a} = H \times \vec{b} \tag{58}$$

Esse problema inverso pode ser interpretado também como encontrar um matriz H, conhecendo apenas os pontos a e b. Para a matriz homogênea, esse problema é geralmente resolvido utilizando-se o RANSAC (CANTZLER, 1981). Esse algoritmo de regressão foi desenvolvido pela comunidade de Processamento de Imagens justamento para resolver esse tipo de desafio. A técnica consiste em iterar uma função de perda L escolhendo aleatoriamente alguns pontos a e b. A função que retornar a melhor métrica é selecionada como resposta do problema. Esse método é muito rápido e robusto com relação a *outliers*.

Além das técnicas clássicas de *Machine Learning*, existem algorimos de *Deep Learn* para fazer o casamento de pontos e computar a homografia, como exemplo a rede de nome *Super Glue* (SARLIN et al., 2020).

2.3.2.5 Rastrear com o algoritmo CSRT

A biblioteca Opencv possue oito tipos de algoritmos diferentes de rastreio[7]. Dentre os métodos disponíveis, foi escolhido para este projeto o CSRT pois possui a melhor acurácia, de acordo com Rosebrock (2018). e o seu FPS, mesmo sendo mais lento, é suficiente para este projeto.

[7] Algoritmos de rastreio disponíveis em <https://docs.opencv.org/3.4/d9/df8/group__tracking.html>

O Algoritmo de rastreio CSRT [8] utiliza um filtro de correlação discriminativo (DCT[9]) com confiabilidade espacial e colorimétrica (LUKEZIC et al., 2017). Os autores utilizam de técnicas de filtragem por correlação semelhantes às apresentadas nas seções 2.3.2.2 e 2.3.2.4. Porém o espaço de características selecionado é formado pelo Histograma de Orientação de Gradiente (HoG) e pelo vetor de probabilidade de cor (espaço *Colornames*). Além disso, foi utilizado um *kernel* gaussiano de tamanho adaptável que se ajusta ao *frame* que está sendo rastreado, podendo assim localizar objetos que não são retangulares.

2.3.2.5.1 Filtros de correlação discriminativos

O uso de filtros de correlação discriminativos é uma estratégia antiga de rastrear objetos (HESTER; CASASENT, 1980). Entretanto apenas em 2010, com a introdução do algoritmo MOSSE [10] (BOLME et al., 2010) que o uso do DCT se tornou mais popular. Atualmente as técnicas de rastreio que possuem melhor acurácia são baseadas no DCT e desde 2014 esse tipo de algoritmo estão no *top-5* do desafio VoT (Visual Object Tracking)[11] para rastreio de objetos.

Para compreender melhor a estratégia do DCT é interessante estudar inicialmente o algoritmo MOSSE. Nessa técnica o filtro f é treinado conhecendo uma lista l de pares de imagens de mesma dimensão (x_i, y_i). Onde x_i é uma imagem e y_i, um sinal gaussiano 2D centralizado no objeto a ser detectado. Na prática, encontrando-se f tem-se uma função que mapeia o *template* para uma imagem desconhecida x.

O treinamento do MOSSE é simples: minimiza-se o quadrado da diferença entre $(f * x_i)$ e y_i. Sendo $(f * x_i)$ a correlação do filtro f e da imagem x_i contendo o objeto:

$$\arg\min_{f} = \sum_{i} |f * x_i - y_i|^2 \tag{59}$$

Aplicando f para uma nova imagem x, tem-se o espaço de características s:

$$s = f * x \tag{60}$$

O espaço de característica s retorna uma gaussiana de duas dimensões com valores de máxima centralizados na posição do objeto na imagem nova (ROBINSON, 2021).

A vantagem do método DCT é que ele pode ser formulado no espaço de Fourier simplificando, assim, as complexidade computacional. Pois é conhecido que a correlação de sinais 2D equivale à multiplicação matricial ponto-a-ponto no espaço de Fourier. Por exemplo, sendo g uma imagem e k um *kernel* tem-se a seguinte equivalência:

$$(g * k)(x) = \mathbf{F}^{-1}\{(G^* \cdot K)(x)\} \tag{61}$$

[8] Do inglês *Discriminative Correlation Filter with Channel and Spatial Reliability*.

[9] Do inglês *Discriminative Correlation Filter*

[10] O algoritmo MOSSE está implementado na biblioteca OpenCV. Disponível em <https://docs.opencv.org/3.4/d0/d02/classcv_1_1TrackerMOSSE.html>.

[11] Página institucional do desafio VoT disponível em <https://www.votchallenge.net/>

Com G^* sendo o complemento complexo de G e "$\cdot$", a multiplicação ponto-a-ponto.

Dessa forma, tem-se a otimização 59 no espaço de Fourier:

$$\arg\min_{F^*} = \sum_i |F^* \cdot *X_i - Y_i|^2 \tag{62}$$

Com a solução analítica:

$$F^* = \frac{\sum_i |Y_i \cdot X_i^*|}{\sum_i |X_i \cdot X_i^*|} \tag{63}$$

Finalmente a métrica s calculada pelo espaço de Fourier:

$$s = \mathbf{F^{-1}}\{F^* \cdot X\} \tag{64}$$

2.3.2.5.2 Filtros de correlação discriminativos multicanais

O DCT, devido as suas simplificações e resultados, é considerado o pilar dos algoritmos modernos de rastreio. Dessa forma, o CSRT utiliza-se também dessa estratégia de correlação porém com vários canais descrevendo o mapa de características.

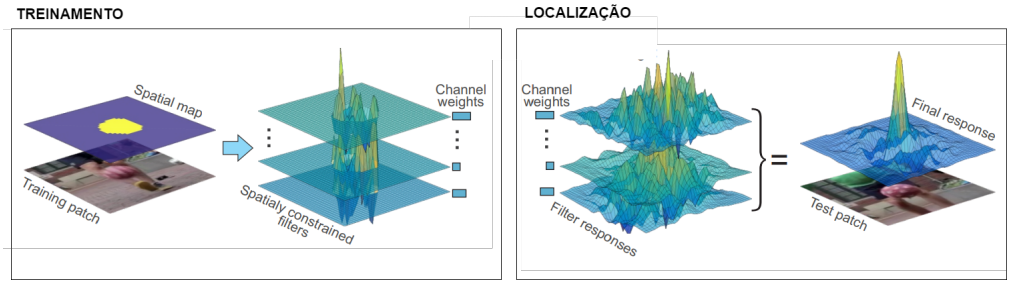
O uso de vários filtros de correlação f podem ser usados para rastrear baseando-se em características específicas. Dado um conjunto C_n com n canais de *features* , tem-se $\vec{f} = \{f_d\}_{d=1,2,3...,n}$ o vetor de filtros para cada canal e $\vec{y} = \{y_d\}_{d=1,2,3...,n}$ o vetor de posições do objeto em cada canal.

A posição do objeto é estimada como a localização do valor máximo da seguinte correlação:

$$\hat{p}(\vec{y}) = \sum_{d=1}^{C_n} f_d * y_d \tag{65}$$

A equação 65 pode ser interpretada da seguinte forma: cada canal de *feature* representa um peso na formação do mapa de características final.

Figura 26 – Treinamento e localização multicanal do algoritmo CSRT .



fonte: Adaptado de Bolme et al. (2010).

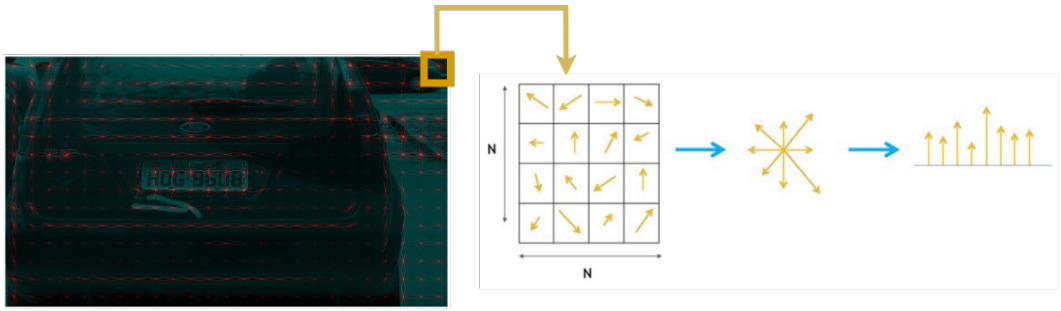
A Figura 26 mostra tanto o processo de treinamento quanto de localização do objeto usando a estratégia de filtros descritivos multicanais. Nota-se que os sinais gaussianos são empilhados para formar o filtro único de correlação.

2.3.2.5.3 Espaço de características do CSRT

O histograma de orientação de gradiente é uma ferramenta muito usada no processamento de imagens para extrair características locais. Na prática, essa característica é calculada dividindo uma imagem imagem em vários blocos de tamanhos idênticos.

Em um bloco são calculadas o ângulo e a magnitude do gradiente para cada pixel. Então é gerado um histograma com valores discretos de ângulos no eixo X e as respectivas somas de magnitude para cada unidade.

Figura 27 – Filtro Hog aplicado em foto de carro com placa "HOG".



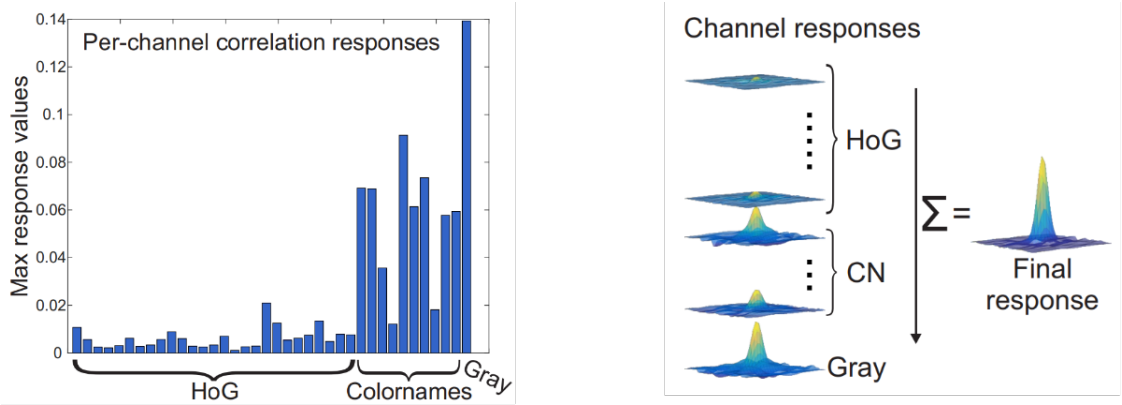
fonte: Imagem gerada pelo autor usando a figura de Carcagnì et al. (2015).

Na Figura 27 tem-se uma foto de um carro com placa "HOG"e diversos Histogramas de Orientação de Gradiente plotados. A seta indica como o histograma é formado.

Além do HoG, o CSRT se diferencia por utilizar as características de cores para descrever a imagem localmente. Esse descritor é chamado de *Colornames* e é definido por um vetor k contendo as probabilidades de uma cor estar presente no pixel, como apresentado por Weijer e Schmid (2007).

No caso do CSRT implementado pelo OpenCV, esse vetor $\vec{k}$ é composto por 11 elementos: 10 cores discretas e 1 canal de cinza.

Figura 28 – Vetor de características do algoritmo CSRT.



fonte: Lukezic et al. (2017).

A Figura 28 mostra o vetor de características do CSRT plotados em um histograma. Além disso o processo de filtragem multicanal é representado pelo empilhamento de gaussianas.

Concluindo, o algoritmo CSRT utiliza-se do filtro DCT para rastrear objetos e garante estabilidade de cor usando o espaço *Colornames* e estabilidade espacial com o HoG. Com alta acurácia e

performance em torno de 20 FPS, esse algoritmo é ideal para o protótipo deste trabalho. O seu ponto negativo é a resposta quando o objeto sai das extremidades da cena.

3 DESENVOLVIMENTO

Neste capítulo serão apresentados os elementos práticos que foram realizados para desenvolver o projeto. Inicialmente é desenvolvida a simplificação geométrica do sistema de controle. Na sequência, o SoC ESP32, a programação do *firmware* e a montagem do módulo PTU são abordados. Por fim, os algoritmos de processamento de imagens são detalhados por meio de fluxogramas e figuras.

3.1 DESCRIÇÃO DO DESAFIO

O objetivo deste projeto é conceber um sistema de visão capaz de rastrear um objeto em tempo real e enviar sinais para um suporte motorizado para manter o alvo sempre centralizado na imagem.

Primeiramente, com relação ao *Hardware* um suporte capaz de rotacionar nos dois eixos (*pan* e *tilt*) foi fabricado por uma impressora. Para realizar o movimento dois servo motores modelo SG90 foram utilizados.

Com relação à câmera, utiliza-se o modelo OV2640 da Omnivision. Essa escolha foi feita pois existe no mercado uma placa ESP32 integrada com esse sensor.

Para simplificar a realização deste trabalho, o processamento das imagens será realizado em um computador convencional (PC) usando a biblioteca OpenCv.

Por fim, o SoC ESP32-cam é responsável por fazer as capturas de imagens, enviar o *buffer* para o PC, receber o *setpoint* e controlar os servo-motores.

As especificidades de cada etapa serão apresentadas à seguir.

3.2 MODELAGEM GEOMÉTRICA

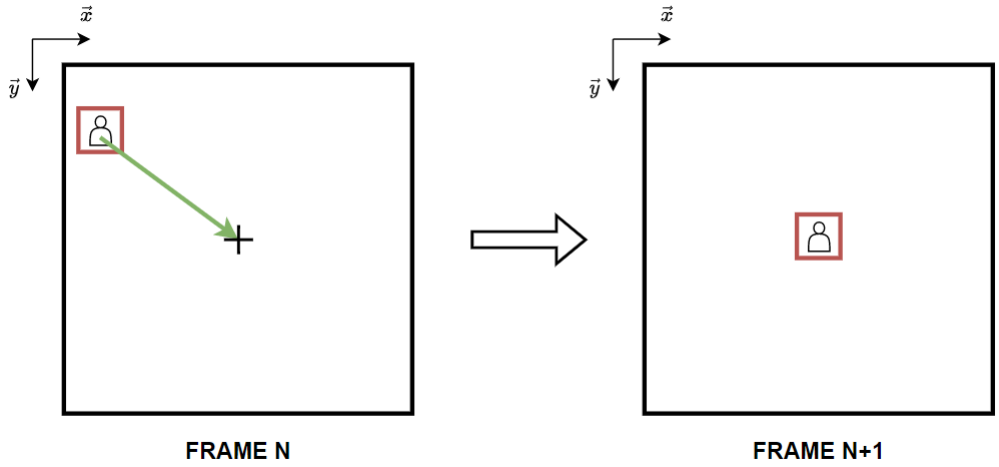
A modelagem geométrica é a principal etapa deste projeto, pois definindo os limites de operação pode-se simplificar as leis de controle. O maior desafio é traduzir o deslocamento necessário para mover um ponto no plano da imagem para os ângulos *pan* e *tilt* do suporte.

Assim, a primeira etapa é centralizar o objeto detectado subtraindo a posição das coordenadas de sua centroide com as coordenadas do centro da imagem tem-se o vetor $\vec{v}$ de deslocamento. A Figura 29 mostra, do ponto de vista do plano do sensor, o vetor de deslocamento. Dessa forma, os servo-motores deverão ser pilotados para que no próximo *frame* o objeto esteja centralizado.

Na Figura 29 está representado o processo de centralização de objeto em imagens adquiridas com câmera PTU. Os dois *frames* são para ângulos α e/ou β distintos. No quadro N o objeto é detectado é marcado com um retângulo vermelho. No plano XY, para centraliza-lo basta adicionar à sua centroide o vetor v, em verde. Esse processo é realizado na etapa $N + 1$, onde pode-se ver que o "boneco"está no centro. O desafio deste projeto é encontrar uma equação que traduza o vetor $\vec{v}$ para ângulos *pan* e *tilt* dos servo motores.

Uma solução para centralizar os pontos em diferentes fotos é tratar a câmera como um *soft sensor* e controlar a posição dos servo motores com algum algoritmo de malha fechada, usando como *feedback*

Figura 29 – Representação do problema de centralizar o objeto detectado.



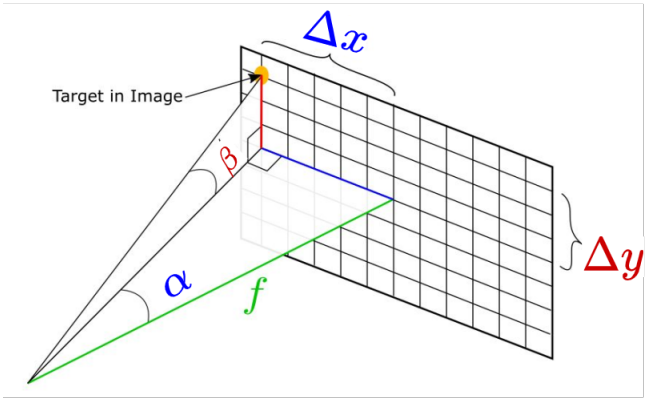
fonte: Produção do próprio autor. Feito no draw.io.

a posição da centroide na imagem formada (KIKUCHI, 2007; PAPANIKOLOPOULOS; KHOSLA; KANADE, 1993). Esse tipo de resolução está fora do escopo deste trabalho.

Além disso, uma solução envolvendo a resolução da matriz de projeção 53 poderia ser utilizada. Entretanto realizar esse cálculo matricial em cada iteração pode ser computacionalmente custoso e gerar imprecisões (DAVIS; CHEN, 2003) pois considera-se os elementos da matrizes extrínsecas (α, β) e intrínseca (distância focal, cisalhamento e centro do sensor).

São feitas algumas considerações para simplificar o controle do módulo PTU. Primeiro, o cálculo do rastreio é iterativo e considera apenas dois *frames* não borrados consecutivos, $I[N]$ e $I[N+1]$. Para $I[N]$ a câmera é considerada centralizada e ordenada com as coordenadas globais. Na imagem seguinte, o PTU é rotacionado com os ângulos *pan* e *tilt* (α e β).

Figura 30 – Representação gráfica da geometria da translação no plano da imagem interpretada como ângulos *pan* e *tilt* do módulo PTU.



fonte: adaptado de Ogorzalek (2019).

A Figura 30 mostra essa simplificação geométrica. Sendo f a distância focal idêntica para x e y, a translação na abscissa (Δx) da imagem representa uma rotação *pan*, como mostra a equação a seguir:

$$\alpha = \arctan\left(\frac{\Delta x}{f}\right) \tag{1}$$

E o ângulo β é dado por:

$$\beta = \arctan\left(\frac{\Delta y}{\sqrt{(\Delta x)^2 + (f)^2}}\right) \tag{2}$$

Dessa forma, é simplificada a aplicação $f : (x, y) \mapsto (\alpha, \beta)$, que mapeia o vetor translação no plano da imagem para os ângulos extrínsecos da câmera. Além disso, a distância focal f deve estar em valores de pixel.

3.3 MATERIAIS UTILIZADOS

3.3.1 Câmera OV2640

O sensor de câmera OV2640 começou a ser fabricado em 2005 pela empresa sino-americana OmniVision. Para sua época o produto foi considerado revolucionário pois não existia no mercado *chips* integrados para soquetes 8x8mm com 2M píxeis de resolução (OMNIVISION, 2005). Além disso, esse módulo é um sensor "semi-processado", pois é capaz de controlar diversos parâmetros da câmera e possui um DSP integrado. Por esses motivos, o produto revolucionou o mercado de celulares, *webcams* e brinquedos em sua época.

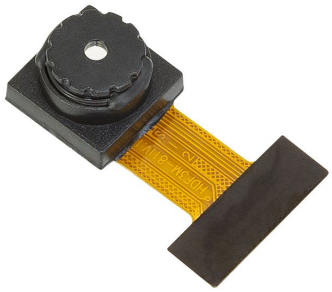
O produto foi descontinuado em 2009, porém continua a ser produzido por fabricantes não oficiais, pois esse módulo ainda é muito utilizado por hobistas devido as suas qualidades e preço reduzido.

As principais características do módulo OV2640 estão listadas abaixo:

- Sensor tipo CMOS do tipo *rolling shutter*;
- Imagem colorida com filtragem óptica e mosaico de Bayer;
- C.I. com microcontrolador e DSP para controle e pré-processamento dos quadros (*In chip programming*) ;
- Resolução do quadro programável com *sub-sampling* (há ganho de FPS);
- Interface de comunicação SCCB (compatível com I2C);
- Resolução máxima UXGA (1600×1200 píxeis) com 15 FPS;
- Resolução mínima CIF (400×300) com 60 FPS;
- ADC de resolução 10 bits;
- Compressão da imagem diretamente no circuito integrado;
- Profundidade de cor: 8 bits para imagem comprimida (JPEG) e 10 bits para RAW;
- Tamanho do pixel: $2,2\mu m \times 2,2\mu m$;

- Lente 1/4";
- Controle manual e automático de ganho e exposição;

Figura 31 – Foto do módulo OV2640 contendo o circuito integrado, jogo de lentes e conector.



fonte: Arducam (2022).

3.3.2 ESP32

O ESP32 é um SoC de baixo custo para aplicações IOT desenvolvido pela empresa chinesa Espressiff [1]. Esse produto integra num PCB blindado memória interna (SRAM e ROM), MCU com dois núcleos de CPU 32 bits e circuito RF para comunicação Wi-fi e *Bluetooth*.

O ESP32 é distribuido pela Espressif como um PCB com as furos *castelled* para conectar com periféricos externos. Como mostra a Figura 32, o ESP32 é apenas o circuito indicado pela seta "ESP32-S". Entretanto, para simplificar o uso do produto, algumas empresas desenvolveram diversas placas de desenvolvimento com esse SoC.

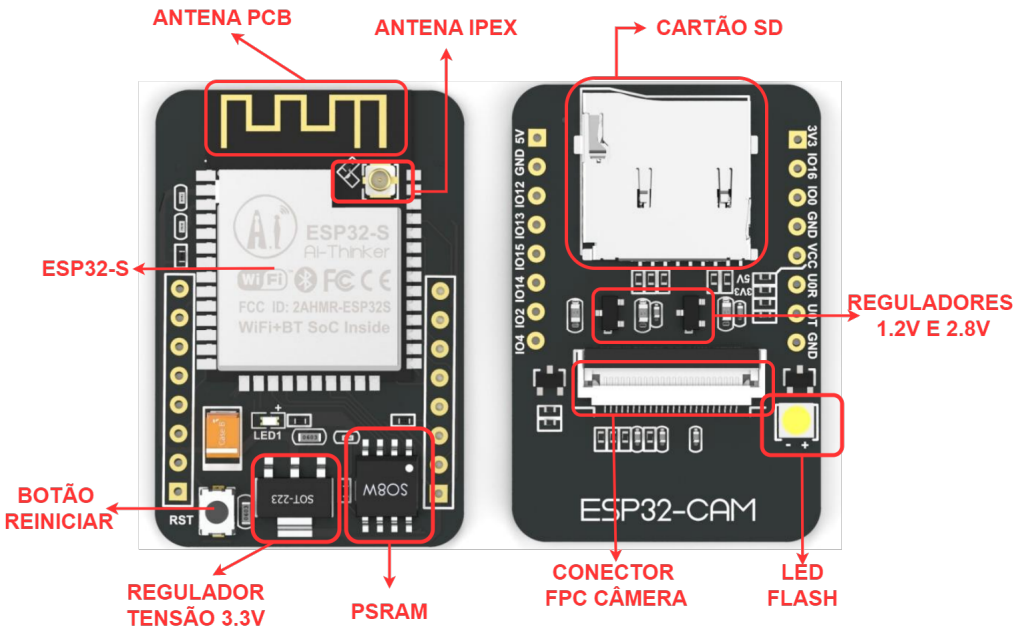
Neste projeto utilizou-se a placa ESP32-CAM distribuída pela empresa chinesa AI-THINKER [2], pois esse produto integra um módulo de câmera OV2640 e memória externa PSRAM de 4Mb. Os principais elementos desta placa de desenvolvimento estão indicados na figura Figura 32.

Para programar o SoC ESP32, pode-se utilizar uma placa FTDI (UART), um depurador JTAG ou enviar o binário por OTA. Para facilitar a programação, em geral as placas de desenvolvimento já possuem um circuito FTDI integrado, entretanto o ESP32-CAM não foi projetado com tal elemento. Dessa forma para enviar os *scripts* para a placa foi utilizado o módulo "ESP32-CAM-MB"vendido no AliExpress [3].

Outra solução para simplificar o envio do programa seria utilizar o mecanismo inovador do ESP32 de atualizar o código por Wi-fi. Essa funcionalidade, chamada de *Over the Air Update*, deve ser incluída no código de programa e assim a placa pode atualizada sem conexão física. Porém o tamanho

[1] <https://www.espressif.com/>
[2] <https://http://www.ai-thinker.com/>
[3] <https://pt.aliexpress.com>

Figura 32 – Placa de desenvolvimento ESP32-CAM com principais componentes indicados.



fonte: feito pelo autor.

do binário não pode ultrapassar 70% da memória *Flash* disponível. Visto que o *firmware* deste tabalho passou desse limiar e que está fora de escopo otimizar o programa, todas as atualizações foram feitas por USB.

As principais características da placa de desenvolvimento ESP32-CAM:

- SoC ESP32-S com dois núcleos de CPU 32 bits Xtensa LX6;
- 520kb de SRAM interna e 4Mb de PSRAM externa (SPI);
- Memória Flash de 4Mb;
- Memória de Programa pode ser expandida com cartão SD;
- Máximo consumo energético: 160 a 240mA usando Wi-fi;
- ULP (*ultra low power processor*): em modo *sleep* consome entre 2 e 150 μA, dependendo dos recursos utilizados;
- Wi-fi (802.11 b/g/n);
- Bluetooth 4.2 BR/EDR e padrão BLE;
- Antena impressa em PCB e conector IPEX;
- Placa de desenvolvimento possui 9 pinos de GPIO disponíveis;
- Tensão de 3,3V para comunicar-se com GPIO;
- Conector FPC e circuito para câmera OV2640;
- *Slot* para carão SD;

- Suporte a SPI, I2C, UART, PWM e outros protocolos;

3.3.3 Servo motor SG90

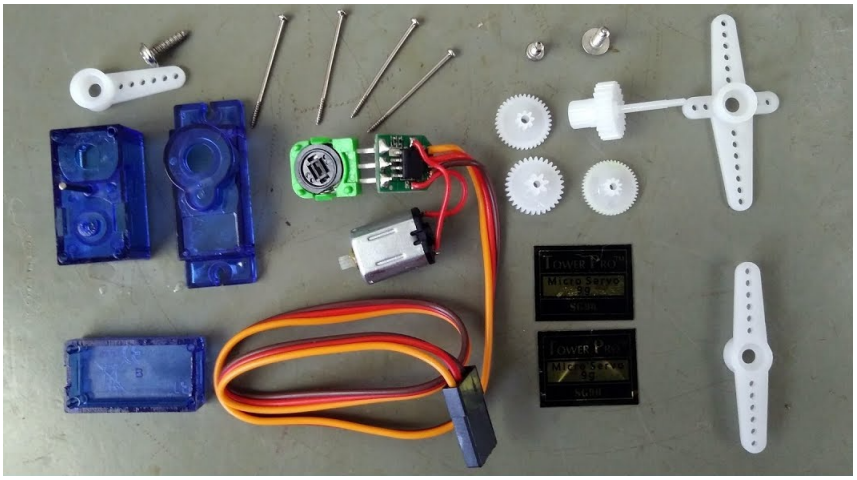
Para realizar a rotação *pan* e *tilt* dois servo-motores SG90 são utilizados. Esses dispositivos são controlados por modulação de pulso (PWM) e recomenda-se um sinal de 5V, porém o produto pode ser pilotado com 3,3V também. Esse tipo de servo-motor é muito utilizado para prototipagem e é composto por um motor DC controlado por PWM indicando a posição angular (variação de $180°$ apenas).

O controle da posição angular é feito por um sistema em malha fechada que compara o valor do *encoder* (potenciometro) com o *setpoint* de entrada. O CI KC9102 é o responsável por esse controle. Ademais, há um jogo de engrenagens para aumentar o torque do servo-motor.

As principais características do servo-motor SG90:

- Tensão de alimentação: 5V;
- Tensão do sinal PWM: 3,3 a 5V;
- Torque: 2,5 kgf·cm ou 0,25Nm
- *Duty cicle*: 1 a 2ms;
- Período: 2ms (50Hz);

Figura 33 – Todas as peças que compõem o servo-motor SG90.



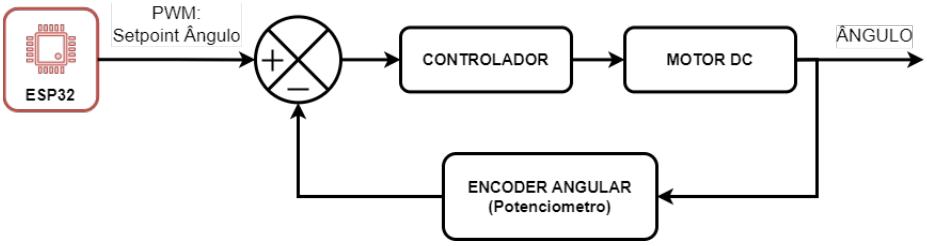
fonte: Feliciano (2019)

3.3.4 Suporte *pan-tilt*

Os suportes utilizados neste projeto foram feitos em Impressora 3D (manufatura aditiva) e o modelo CAD foi disponibilizado pela comunidade MakerBot's Thingiverse [4].

[4] <https://www.thingiverse.com/thing:708819/files>

Figura 34 – Diagrama do controle em malha fechada do servo motor SG90. Simplificação do controle feito pelo CI KC9102.

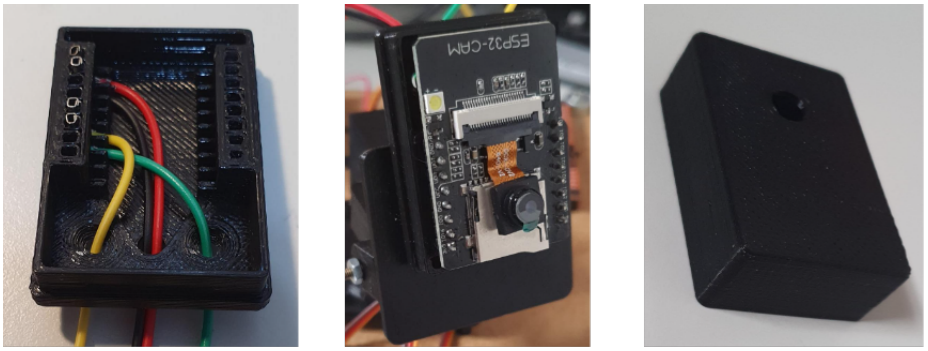


fonte: Produção do próprio autor. Feito no draw.io.

O módulo PTU [5] foi projetado para encaixar dois servo-motores SG90 para realizar a rotação *pan-tilt*.

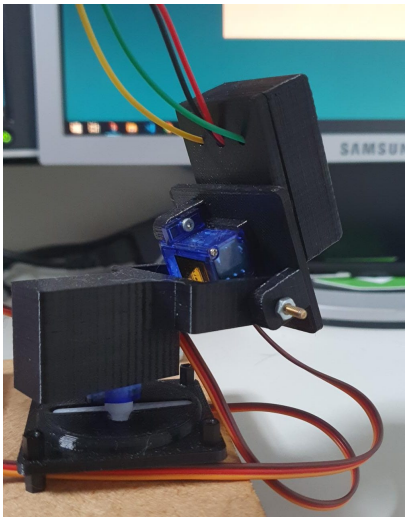
Para acomodar a placa ESP32, foi impresso um *case* [6] com espaço para encaixar conectores do tipo Dupont.

Figura 35 – *Case* para placa de desenvolvimento ESP32-CAM.



fonte: feito pelo autor.

Figura 36 – Suporte *pan-tilt* montado com servo-motores SG90 e *case* do ESP32-CAM.



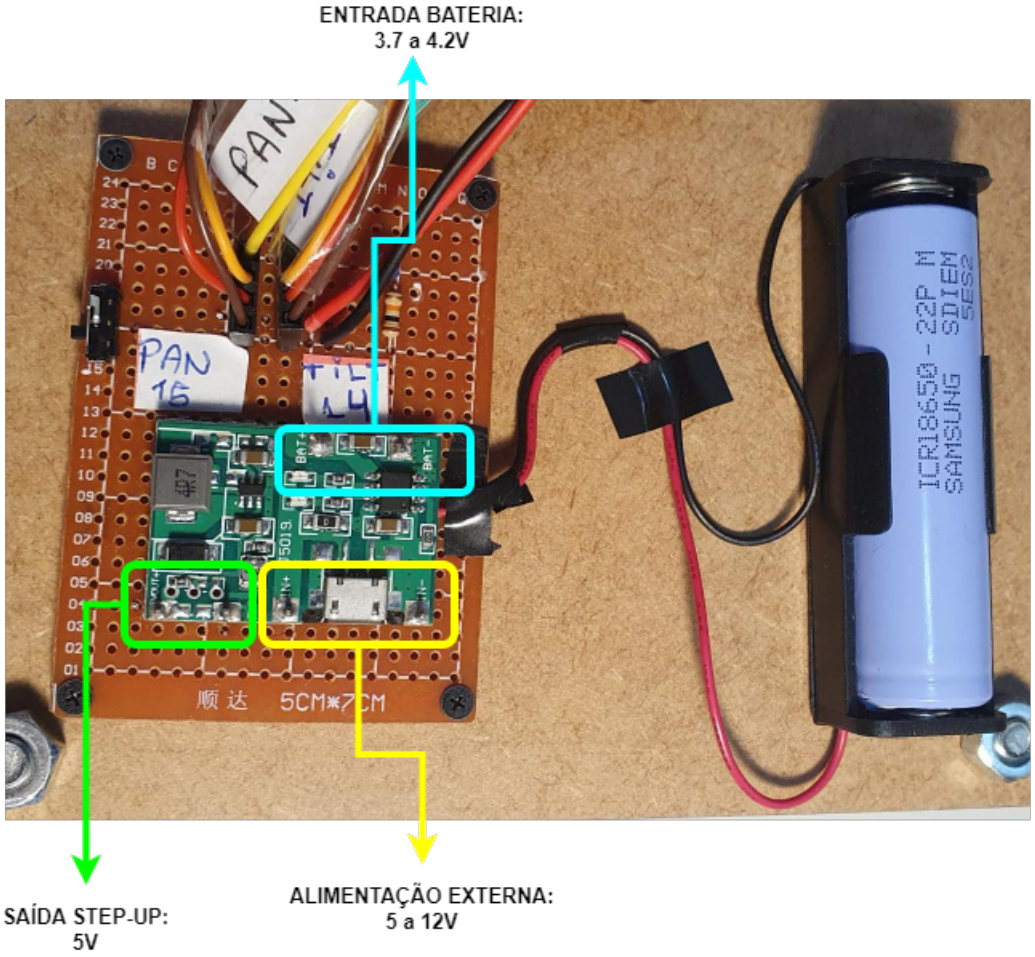
fonte: feito pelo autor.

[5] Modelo disponível em <https://www.thingiverse.com/thing:708819/files>
[6] Modelo disponível em <https://www.thingiverse.com/thing:4192843>

3.3.5 Circuito de energia

Para alimentar tanto o módulo ESP32-CAM quanto os servo-motores utiliza-se uma bateria 18650 (1800mAh) e um circuito BMS (1s) com regulação chaveada *Step-up*. A Figura 37 mostra o PCB em verde que encapsula todos esses elementos.

Figura 37 – Foto do circuito regulador de tensão e bateria 18650.



fonte: feito pelo autor.

A tensão da bateria é de 3,7 a 4,2V, dependendo da carga. O circuito *Step-up* regula a tensão para 5 a 9V na saída. Para 5V na saída, tem-se corrente máxima de 1,4A. O circuito BMS provem uma proteção contra sobretensão, subtensão e curto circuito. Além disso, pode-se carregar a bateria pelo conector micro USB, com alimentação externa de 5 a 12V.

A escolha desse circuito foi feita pensando na alta demanda de corrente tanto dos servo motores quanto do ESP32. Esse último, quando utiliza recursos de RF, tem picos de corrente que ultrapassam a corrente máxima nominal do *datasheet* (mais ou menos 200mA). Na prática, quando o SoC ESP32 não recebe energia suficiente o *chip* envia um sinal de *Brown out Detector*, que pode ser lido no teminal.

Além disso, foi adicionado uma micro-chave interruptora para ligar e desligar o ESP32 e um LED indicativo de energia.

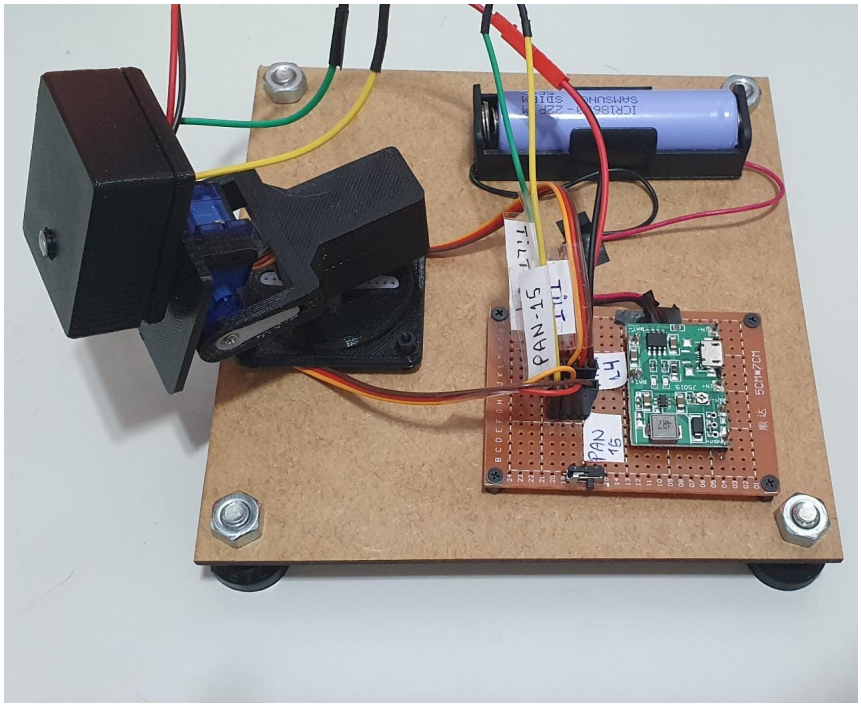
Por fim, o consumo energético do sistema não será estudado. A utilização desse circuito de alimentação é apenas para facilitar a prototipação do módulo PTU, podendo testa-lo sem dificuldades

em qualquer lugar conectado ou não na tomada.

3.3.6 Montagem do equipamento

Todos os elementos descritos nesta seção foram montados numa placa de MDF e para garantir a estabilidade fora, adicionados quatro pés niveladores.

Figura 38 – Montagem do módulo PTU, circuito de energia e bateria 18650.



fonte: feito pelo autor.

3.3.7 Programas, bibliotecas e softwares utilizados

3.3.7.1 PlatformIO

O SoC ESP32 pode ser programado com C, C++, Lua ou Python (MicroPython). Além da diversidade de linguagem, a maior vantagem dessa placa é poder utilizar o *framework* do Arduino, o que resulta numa prototipação mais rápida. A Espressif disponibiliza um *framework* mais robusto com características industriais chamado *esp-idf*, porém não foi utilizado neste projeto.

Para desenvolver um projeto que envolva diversas bibliotecas, a interface de programação do Arduino não é muito prática. Dessa forma, foram utilizados o VSCode[7] e a plataforma PlatformIO[8] para programar o *firmware*.

3.3.7.2 OpenCV

A programação do *Software* é feita em Python e para o processamento de imagens foi utilizada a biblioteca OpenCV. Essa é uma biblioteca que foi desenvolvido inicialmente em 2000 por um

[7] Disponível em <https://code.visualstudio.com/>
[8] Disponível em <https://platformio.org/>

laboratório da Intel e atualmente é um projeto *open-source*. O código-fonte da biblioteca é escrito em C++ e existe um *wrapper* oficial para Python. Dessa forma, não há perda relevante de desempenho computacional ao utilizar a versão em Python [9].

O OpenCV possui diversos algoritmos optimizados de processamento de imagens, incluindo os algoritmos de *tracking* que foram utilizados neste trabalho para fazer o controle em tempo real.

3.4 PROGRAMAÇÃO DO FIRMWARE PARA ESP32

A programação do *Firmware* é feita em C++ e as bibliotecas dos periféricos foram integradas com o PlatformIO.

3.4.1 O ESP32 como *access point*

O ESP32 é responsável por capturar as imagens do módulo OV2640 e envia-las para o PC. Para isso, o SoC foi programado para ser um roteador Wi-fi (*Access Point*) e gernciar um servidor. Ou seja, o computador se conecta na rede do ESP32 e faz uma requisição como cliente de *frames* por HTTP.

Uma solação para enviar os quadros seria disponibilizar um serviço na porta 554 com protocolo RTSP[10] disponibiliazndo um *stream* de imagem. Entretanto durante os testes deste projeto não foi possível pilotar os servo-motores e manter o fluxo de imagens com boa performance.

3.4.2 Capturar imagem da câmera OV2640

O *driver* para capturar os frames do módulo OV2640 pelo ESP32 foi desenvolvido pela Espressif [11] e a empresa disponibiliza também um exemplo [12] de rotina que conecta o SoC a uma rede Wi-fi e fornece uma interface HTML para configurar os parâmetros da câmera, alterando diretamente os seus registradores.

Além disso, nesse *script* há um exemplo de como codificar uma imagem capturada pelo módulo OV2640 e envia-la ao PC por meio de requisição no serviço "/capture".

3.4.3 Controle dos servo motores

Para simplificar o controle dos servo-motores a biblioteca ESP32Servo[13] foi utilizada. O seu funcionamento é muito prático: basta configurar um pino do ESP32, indicar os limites máximo e mínimo do *duty cycle* e definir o período.

Para pilotar o servo-moto, a biblioteca gera automaticamente um sinal PWM proporcional ao ângulo entre $0°$ e $180°$.

---

[9] Uma diferença evidente e conhecida entre o OpenCV para Python e C++ é o tempo e memória gasta para carregar um quadro do *buffer* de vídeo ou *stream*. Entretanto há soluções em Python usando *threads* que possuem boa performance. Exemplo disponível em <https://pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/>

[10] Biblioteca do protocolo RTSP para ESP32-CAM disponível em <https://github.com/rzeldent/esp32cam-rtsp>

[11] Disponível em <https://github.com/espressif/esp32-camera>

[12] Disponível em <https://github.com/espressif/arduino-esp32/tree/master/libraries/ESP32/examples/Camera/CameraWebServer>

[13] Disponível em <https://github.com/jkb-git/ESP32Servo>

Entretanto, para que os *timers* do ESP32 utilizados na captura da imagem não atrapalhem o controle dos servo-motores, foi necessário fazer a alocação de *timers* específicos para gerar o PWM. Com a biblioteca ESP32Servo, basta chamar a função "ESP32PWM::allocateTimer(3)"com o número do temporizador desejado[14].

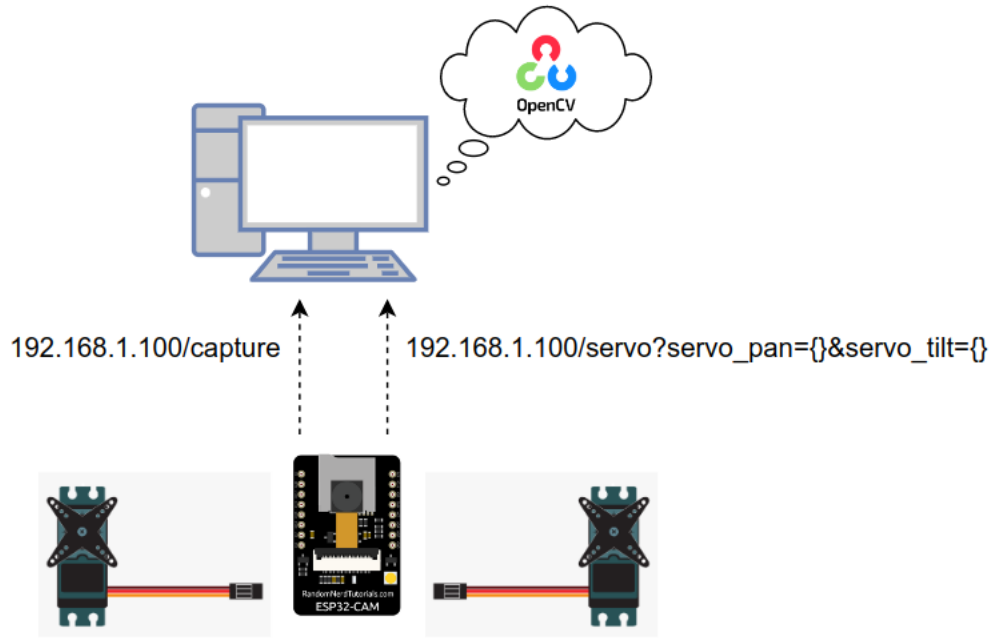
Por fim, foi programado um serviço no endereço "</servo?servo_pan={}&servo_tilt={}>". Quando, o servidor recebe uma requisição nesse formato o ESP32 controla os servo-motores nos ângulos indicados.

3.4.4 O ESP32 como servidor

Reiterando o que foi apresentado, o ESP32 foi programado para ser um roteador Wi-fi e ao mesmo tempo um servidor com IP fixo "192.168.1.100". Para esse endereço, estão disponibilizados três serviços http:

- <192.168.1.100/>: interface para configurar parâmetros da câmera;
- <192.168.1.100/capture>: ESP32 retorna ao cliente uma imagem capturada do módulo OV2640;
- <192.168.1.100/servo?servo_pan={}&servo_tilt={}>: cliente envia sinal para o ESP32 pilotar os dois servo-motores;

Figura 39 – Conexões de todos os elementos do projeto do ponto de vista do ESP32.



fonte: Produção do próprio autor.

[14] O SoC ESP32 possui 4 temporizadores de uso geral, além de possuir um módulo PWM dedicado a controlar motores (MCPWM), com seus próprios *timers*. A programação mais aprofundada desse módulo está fora do escopo do projeto, por isso foi utilizada a biblioteca ESP32Servo

3.5 PROGRAMAÇÃO DO SOFTWARE

Para o processamento das imagens utilizou-se a linguagem Python. Para obter os *frames* do módulo OV2640, deve-se estar conectado na rede do ESP32 e fazer uma requisição <192.168.1.100/capture> usando a biblioteca Requests[15].

3.5.1 Rotina de detecção de *blur*

Para classificar se uma imagem está desfocada utiliza-se o método da Variância do Laplaciano da imagem apresentado na seção 2.3.1.

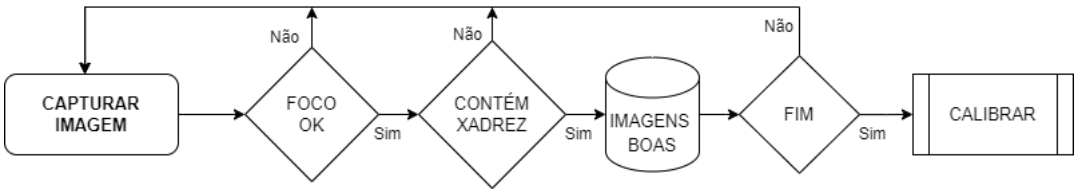
Para definir um valor limite da Variância de uma imagem boa e uma imagem ruim, foram capturadas e classificadas manualmente 30 imagens. Assim, utilizando a técnica de regressão logística[16] a curva da Figura 47 foi desenhada. Visualmente o limiar foi traçado em 250. Ou seja, se a Variância do Laplaciano for menor que esse valor, o quadro é considerado borrado.

3.5.2 Rotina de calibração da câmera

Para se encontrar os parâmetros intrínsecos da câmera, utilizou-se o exemplo[17] do OpenCV. Esse código foi adaptado para fazer uma calibração semi-automática da câmera enquanto o PC está conectado.

Para efetuar a calibração, um quadro quadro xadrez [18] foi impresso. Assim, com a câmera conectada deve-se variar a posição do quadriculado em relação ao PTU e o *script* vai fazer a captura apenas dos *frames* bons. No final do processo tem-se no terminal a matriz intrínseca como mostra na figura **??**.

Figura 40 – Algoritmo iterativo de calibração online do módulo PTU.



fonte: feito pelo autor.

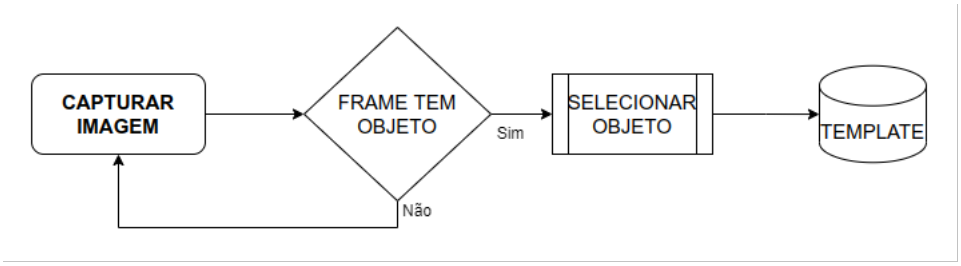
3.5.3 Algoritmo de detecção e rastreio

Estando conectado no módulo PTU, o processo de *tracking* inicia com o usuário definindo manualmente um *frame* bom do *stream* e selecionando o retângulo do objeto na imagem a ser rastreado. O *template* é, assim, definido.

Em seguida, se dá inicio à rotina principal do programa: rastrear o objeto e mantê-lo centralizado na imagem. O critério para mover o PTU é se a centroide do objeto estiver fora de um retângulo no centro da imagem. As dimensões desse quadrilátero podem variar: quanto menor for, mais vezes os servo-motores serão acionados.

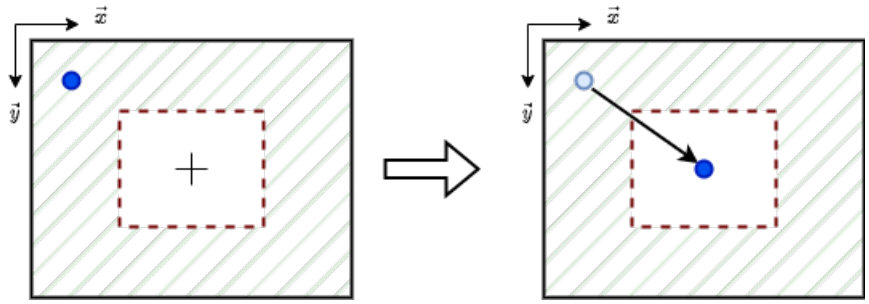
[15] Documentação disponível em <https://requests.readthedocs.io/en/latest/>.
[16] Foi utilizada a biblioteca Seaborn. Disponível em <https://seaborn.pydata.org/>
[17] Disponível em <https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html>.
[18] Disponível em <https://docs.opencv.org/4.x/da/d0d/tutorial_camera_calibration_pattern.html>

Figura 41 – Algoritmo iterativo de seleção do *template* a ser rastreado.



fonte: feito pelo autor.

Figura 42 – Representação gráfica da região definida como fora do centro da imagem.



fonte: Produção do próprio autor. Feito no draw.io

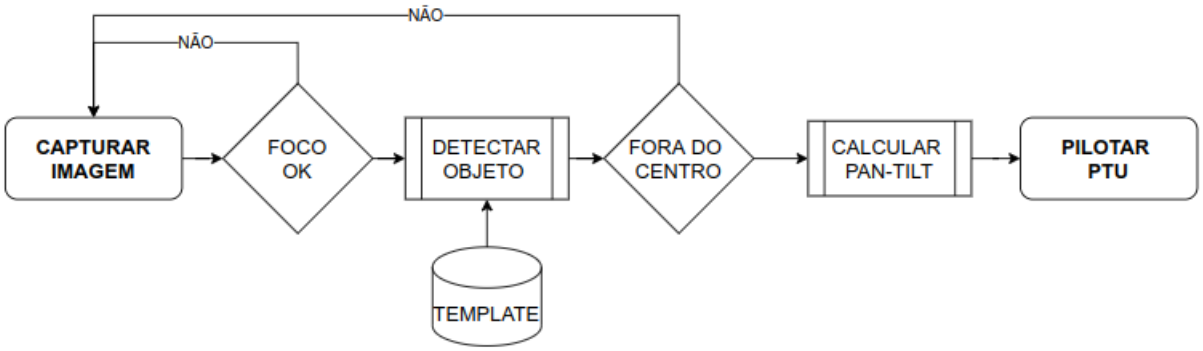
Por fim, quando a centroide estiver na região hachurada da Figura 42 o PC envia o sinal para pilotar o módulo PTU com as ângulos *pan-tilt* que deixam o objeto no centro da imagem formada.

3.5.4 *Pipeline* completo de processamento de imagem

Enquanto o algoritmo de rastreio encontrar o objeto, o PC executa a rotina da figura Figura 43, tentando sempre manter a centroide do objeto na região central.

Caso o *tracking* não encontre o objeto, o PC reinicia o processo, retomando a rotina de seleção de *template* (Figura 41).

Figura 43 – Diagrama completo da rotina de centralização realizada pelo PC.



fonte: Produção do próprio autor. Feito no draw.io

4 RESULTADOS

Este capítulo apresenta os resultados obtidos da implementação do projeto de controle de módulo PTU formado por dois servo motores e um ESP32-cam. Dessa forma, são apresentadas fotos capturadas com o módulo OV2640, bem como o estudo quantitativo do tempo de latência para o PC receber as imagens. Além disso, são mostrados exemplos de como operar o algoritmo de rastreio e selecionar o *template*.

Os resultados obtidos a partir do processo de rastreio são apresentados através de imagens e tabelas, que mostram os ângulos calculados pelo algoritmo. Este capítulo também apresenta as dificuldades e problemas encontrados durante a implementação e dos testes, fornecendo uma visão geral da eficiência do sistema desenvolvido.

4.1 ESP32 E MÓDULO OV2640

Nesta seção são apresentados os resultados quantitativos relativos à captura de imagem do módulo OV2640 pelo ESP32.

4.1.1 Resolução dos quadros

Todos os resultados foram obtidos com a câmera OV2640 capturando imagens no formato CIF com resolução 320×240. Além disso, a câmera está configurada para controle automático de ganho e exposição.

Figura 44 – Foto proveniente do módulo OV2640 no formato CIF.

fonte: feito pelo autor.

4.1.2 Tempo de captura do quadro

Para se conhecer o tempo de captura da imagem do módulo OV2640 pelo ESP32 deve-se conectar o SOC no computador e acessar o terminal UART enquanto houver fluxo de quadros.

A Figura 45 mostra que o tempo para capturar uma imagem CIF 320×240 pelo microcontrolador é de 12 FPS. Aproximando: 1 quadro a cada $83ms$.

Figura 45 – Tempo de envio do .

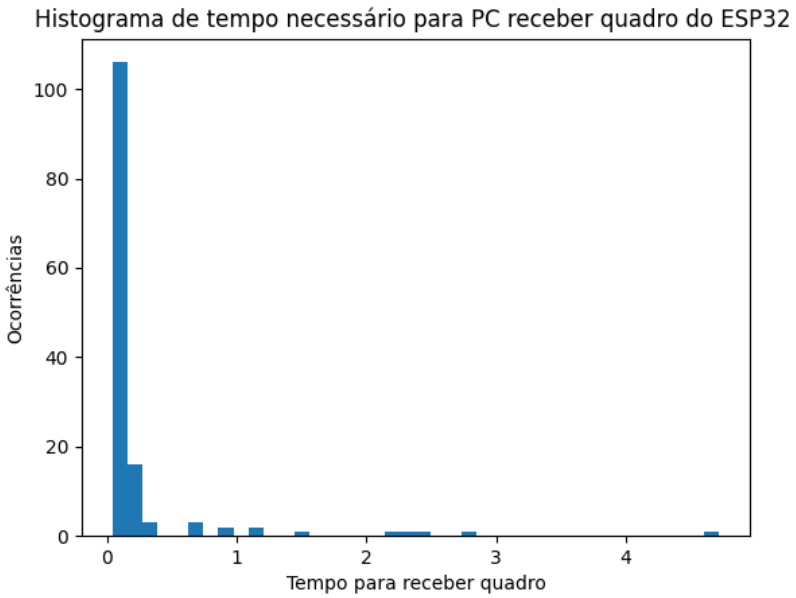
```
16:03:34.656 -> MJPG: 3931B 82ms (12.2fps), AVG: 83ms (12.0fps), 0+0+0+0=0 0
16:03:34.755 -> MJPG: 3940B 73ms (13.7fps), AVG: 83ms (12.0fps), 0+0+0+0=0 0
16:03:34.822 -> MJPG: 3976B 87ms (11.5fps), AVG: 83ms (12.0fps), 0+0+0+0=0 0
16:03:34.922 -> MJPG: 3944B 74ms (13.5fps), AVG: 83ms (12.0fps), 0+0+0+0=0 0
```

fonte: feito pelo autor.

4.1.3 Tempo para receber quadro

Para identificar os tempos de latência o seguinte teste foi realizado: com o módulo PTU num raio de $15cm$ o tempo para o PC receber um quadro foi computado via *Software*. Dessa forma, foi gerado uma lista de tempos que estão plotados no histograma da Figura 46. Pode-se considerar que o tempo para o computador receber um quadro é $82ms$, ou seja a comunicação WIFI não aumenta significativamente a latência para essa distância.

Figura 46 – Histograma do tempo, em segundo, necessário para o PC receber um *frame*.



fonte: feito pelo autor.

4.2 DETECÇÃO DE *BLUR*

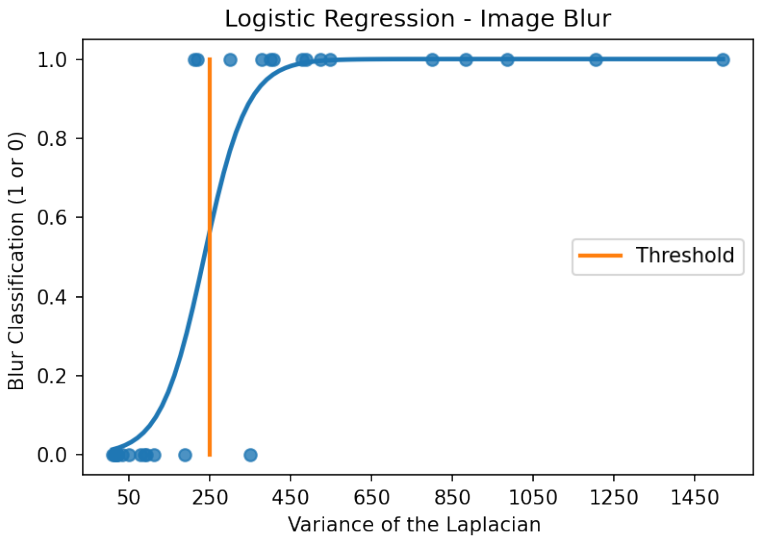
A regressão logística é uma técnica de aprendizado de máquina amplamente utilizada para classificação binária. Neste projeto, a regressão logística foi treinada para definir um valor limite de variância

do Laplaciano de imagem e assim identificar se um *frame* está borrado.

Para treinar a regressão logística, foram coletadas 30 amostras de imagens borradas e não borradas e a variância do Laplaciano foi calculada para cada amostra. A regressão logística foi então treinada usando esses dados para aprender a definir um valor limite de variância do Laplaciano que melhor separa as imagens borradas das não borradas.

A escolha do valor limite foi feita visualmente no gráfico da Figura 47. Com uma variância inferior à 250 a imagem é considerada desfocada.

Figura 47 – Curva da regressão logística treinada para definir se imagem está borrada.



fonte: feito pelo autor.

A Figura 48 mostra quatro exemplo de fotos com seus respectivos valores de variância de Laplaciano. Pode-se perceber que a regressão logística treinada gerou uma função que está de acordo com o nível do foco percebido visualmente.

4.3 CALIBRAÇÃO

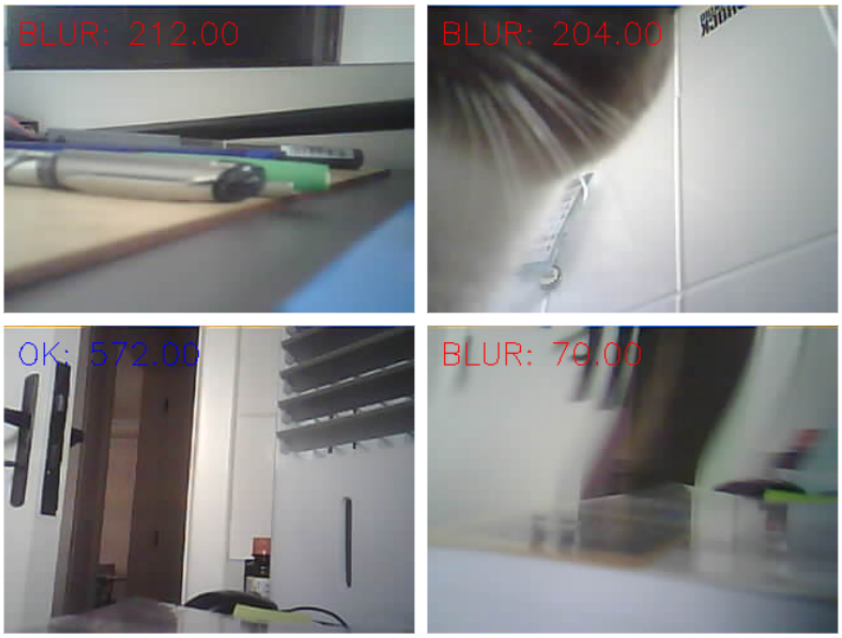
A calibração de câmera é um passo importante na utilização de imagens capturadas por dispositivos digitais, pois permite a correção de distorções e a obtenção de informações precisas sobre as características intrínsecas da câmera. Neste trabalho, utilizamos a biblioteca OpenCV para calibrar a câmera OV2640, utilizando imagens de um tabuleiro xadrez como referência.

A técnica de calibração utilizada consiste em detectar marcas específicas em um tabuleiro xadrez, permitindo a obtenção de informações precisas sobre a geometria da câmera. Com o uso dessa técnica, conseguimos encontrar os parâmetros intrínsecos da câmera, como matriz de focalização e distorção radial e tangencial.

A Figura 49 foi gerada durante o processo de calibração da Câmera OV2640. O algoritmo detectou o tabuleiro xadrez, portanto o quadro foi utilizado na calibração.

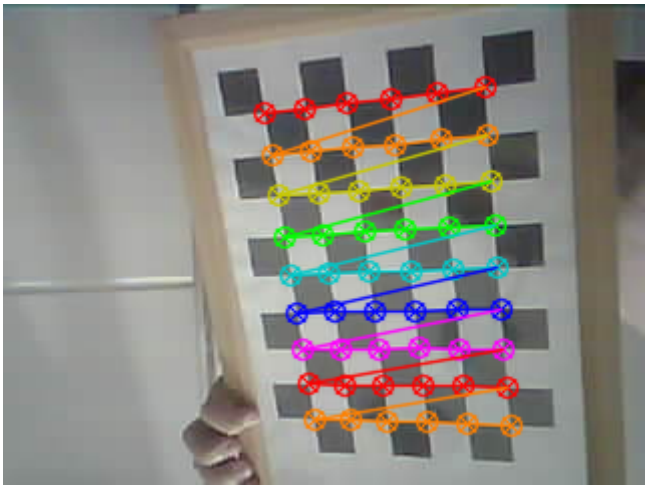
A calibração utilizando *frames* recebidos em tempo real foi realizada com sucesso, e os parâmetros intrínsecos encontrados foram utilizados para corrigir distorções nas imagens capturadas pela câmera OV2640.

Figura 48 – Quatro imagens capturadas com o módulo OV2640 com diferentes graus de *blur*.



fonte: feito pelo autor.

Figura 49 – Algoritmo de calibração do OpenCV encontrando o tubuleiro de xadrez.



fonte: feito pelo autor.

A Figura 50 mostra a matriz intrínseca resultante em valores de pixel. As distorções radiais não serão corrigidas nesse projeto.

Para fins de simplificação, a distância focal para x e y foi considerada como 400 pixeis.

Figura 50 – Matriz intrínseca da câmera após o processo de calibração.

```
[[400.7452474    0.         149.85681176]
 [  0.         410.72172762 122.14782421]
 [  0.           0.           1.        ]]
```

fonte: feito pelo autor.

Em resumo, a calibração de câmera é uma etapa fundamental para garantir a qualidade e precisão das imagens capturadas pelo dispositivo. Isso permite o uso da câmera em aplicações que requerem

precisão espacial.

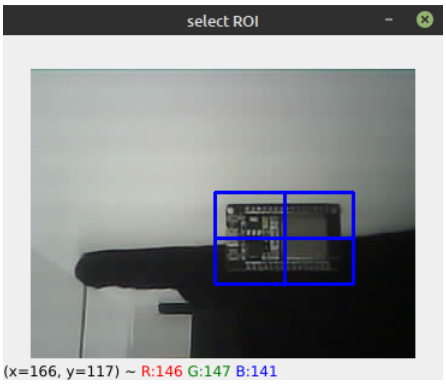
4.4 RASTREIO DE OBJETOS

Nesta seção é apresentado o funcionamento dos algoritmos de detectar *blur* e de rastrear objetos. Além disso, são discutidos os resultados qualitativos do processamento de imagens por meio de tabelas e figuras.

4.4.1 Definir *template*

Após o PC estar conectado na rede e o algoritmo iniciar, o usuário pode selecionar qual o objeto na imagem para será rastreado. A Figura 51 mostra a janela de seleção manual do *template*.

Figura 51 – Janela para seleção do *template*.



fonte: feito pelo autor.

4.4.2 Rastrear objeto com câmera estática

Os resultados a seguir foram realizados com a câmera estática, ou seja, sem controlar o módulo PTU para centralizar o objeto.

Na Figura 52 tem-se quatro exemplos de detecção bem sucedida do algoritmo CSRT. Nota-se que o tempo gasto para o rastreio é maior do que 10 FPS e para a aplicação deste projeto processar uma imagem a cada $0,1ms$ é suficiente.

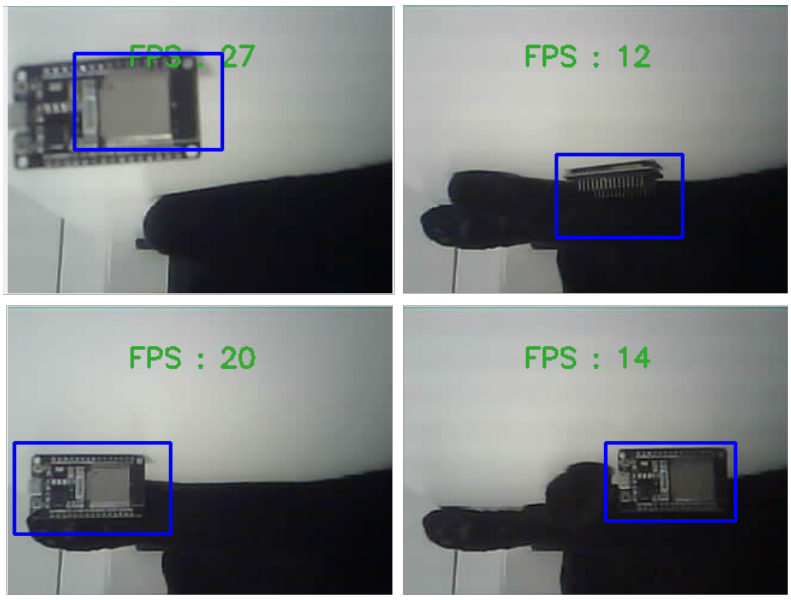
Nota-se que a técnica de rastreio é robusta com um BG de cor semelhante ao objeto e à mudança de escalas. Além disso o CSRT foi capaz de continuar localizando mesmo quando o objeto muda a face visível.

4.4.3 Centralizar objetos

Sabendo que a imagem recebida pelo PC tem resolução 320×240, foi desenhado um quadrado centralizado no *frame* com lado de 40 píxeis. O objeto detectado deve estar no centro desse quadrado.

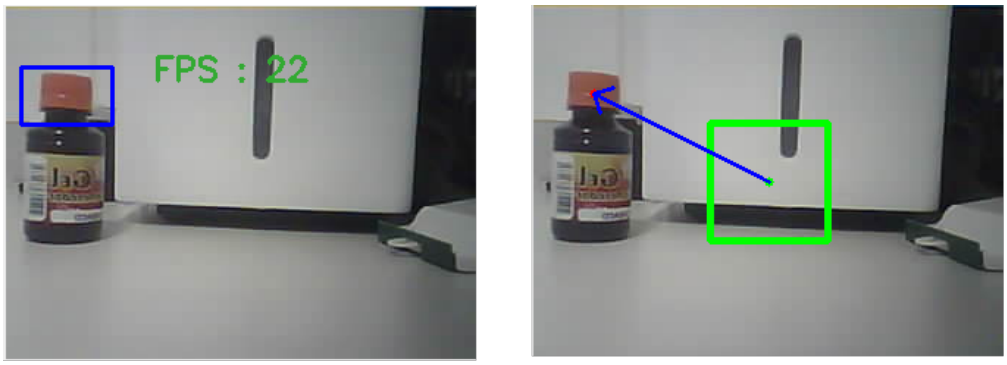
A Figura 53 mostra o algoritmo localizando a tampa e indicando o vetor no plano da imagem para centralizar o objeto.

Figura 52 – Detecção de objeto em quatro posições diferentes.



fonte: feito pelo autor.

Figura 53 – Detecção de objeto e vetor para centralizá-lo.



fonte: feito pelo autor.

4.4.4 Rastreando objeto em duas cenas

Para mostrar o funcionamento do algoritmo de rastreio e a possibilidade de controlar o módulo PTU, foram gerados quatro sequências distintas de imagens com o objeto sendo centralizado .

O módulo PTU inicia sempre com os ângulos *pan* e *tilt* iguais a $90°$. Após o algoritmo localizar o objeto é enviado um sinal para o ESP32 com a variação de ângulo necessária para orientar a câmera.

A Figura 54 mostra esses quatro exemplos. As duas primeiras colunas contém a imagem do objeto antes da centralização. Nas duas colunas finais o objeto está centralizado. Estão representadas também o vetor de centralização, o ROI do objeto rastreado e o quadrado central.

A tabela 1 mostra para cada sequência da Figura 54 os deslocamentos X e Y no plano da imagem e as respectivas variações nos ângulos do módulo PTU.

4.4.5 Rastreio do objeto em tempo real

Com a finalidade de testar o rastreio em tempo real, o objeto da cena foi deslocado em sequência e as imagens processadas estão apresentadas na Figura 55.

Tabela 1 – Sequência de Centralização Discreta

Sequência	X	Y	ΔPan	ΔTilt
(A)	+6	+94	$0°$	$-25°$
(B)	+125	+70	$40°$	$-25°$
(C)	+53	+70	$10°$	$-20°$
(D)	+88	+108	$25°$	$-30°$

fonte: Produção do próprio autor.

A tabela 2 mostra o vetor centralização e os ângulos usados para controlar os servo-motores para cada posição.

Tabela 2 – Sequência de Centralização Continua

Posição	X	Y	Pan	Tilt
0	0	0	$90°$	$90°$
1	-44	+75	$80°$	$65°$
2	-93	+44	$55°$	$50°$
2	+40	+5	$85°$	$50°$

fonte: Produção do próprio autor.

4.4.6 Problemas encontrados

Esta seção é dedicada para discutir os desafios encontrados durante o desenvolvimento do projeto e quais foram as soluções identificadas.

4.4.6.1 Problema de oclusão

O algoritmo CSRT para rastreamento de objetos é altamente preciso, mas ainda enfrenta o desafio da oclusão. A oclusão ocorre quando o objeto sendo rastreado é parcial ou totalmente ocultado por outro objeto ou superfície na cena. Isso pode causar uma perda temporária ou permanente da visão do objeto, resultando em uma falha no rastreamento.

Embora existam técnicas para lidar com a oclusão, como a utilização de modelos de fundo e de objeto, neste projeto essas técnicas não serão necessárias.

Na Figura 56 foi realizada uma avaliação do desempenho do CSRT na presença de oclusão, mostrando que o algoritmo é robusto e mantém o objeto rastreado mesmo se estiver ausente na cena.

4.4.6.2 Problema de deslize

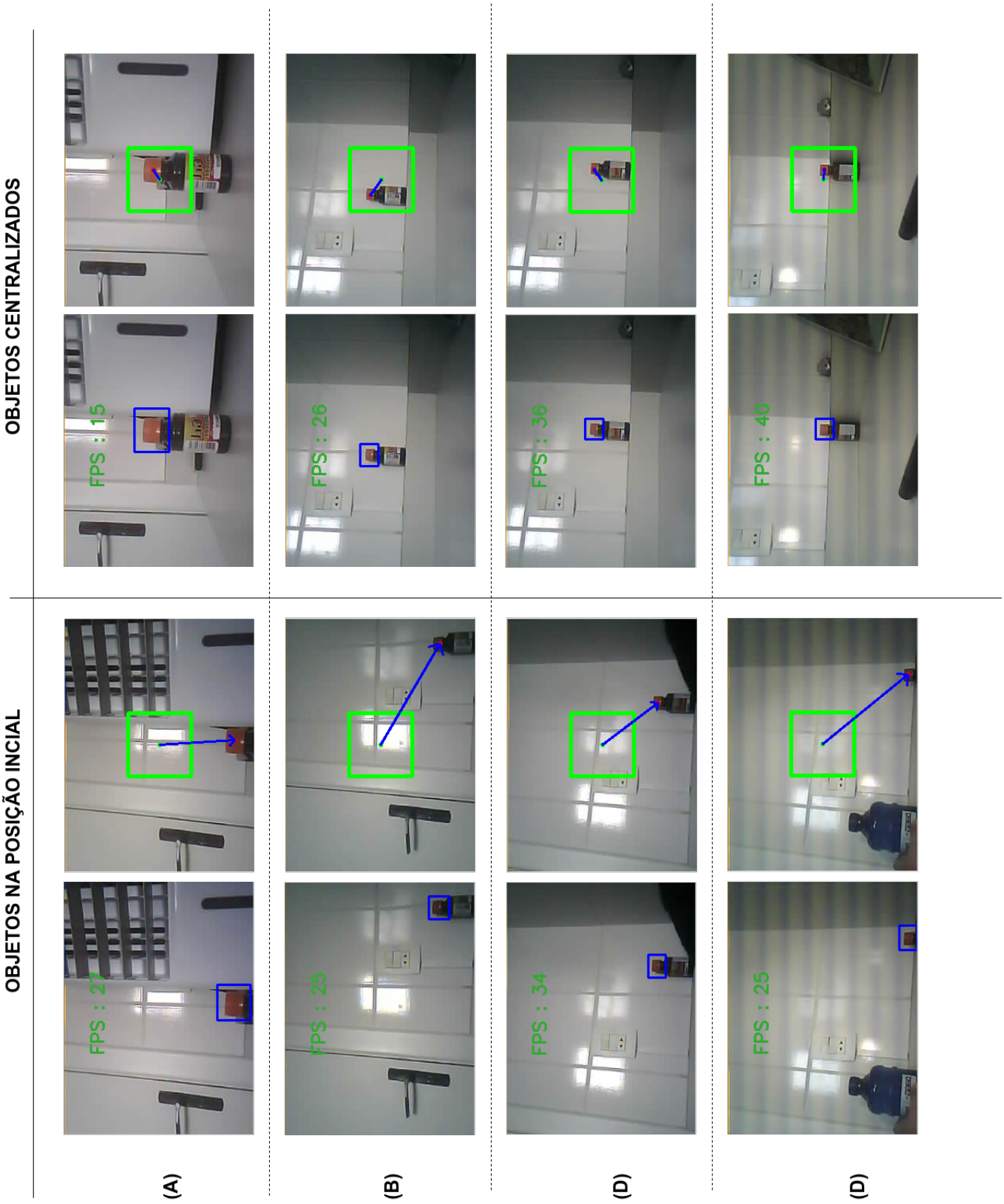
Além da oclusão em algoritmos de rastreio outro problema crítico é a modificação indesejada do objeto localizado. Esse deslize do ROI está representado na Figura 57.

4.4.6.3 Pouca informação na imagem

Em alguns casos foi notado que o detector de foco estava acusando falsamente uma imagem borrada. Esse falso negativo é gerado quando há pouca informação na imagem, por exemplo uma parede branca.

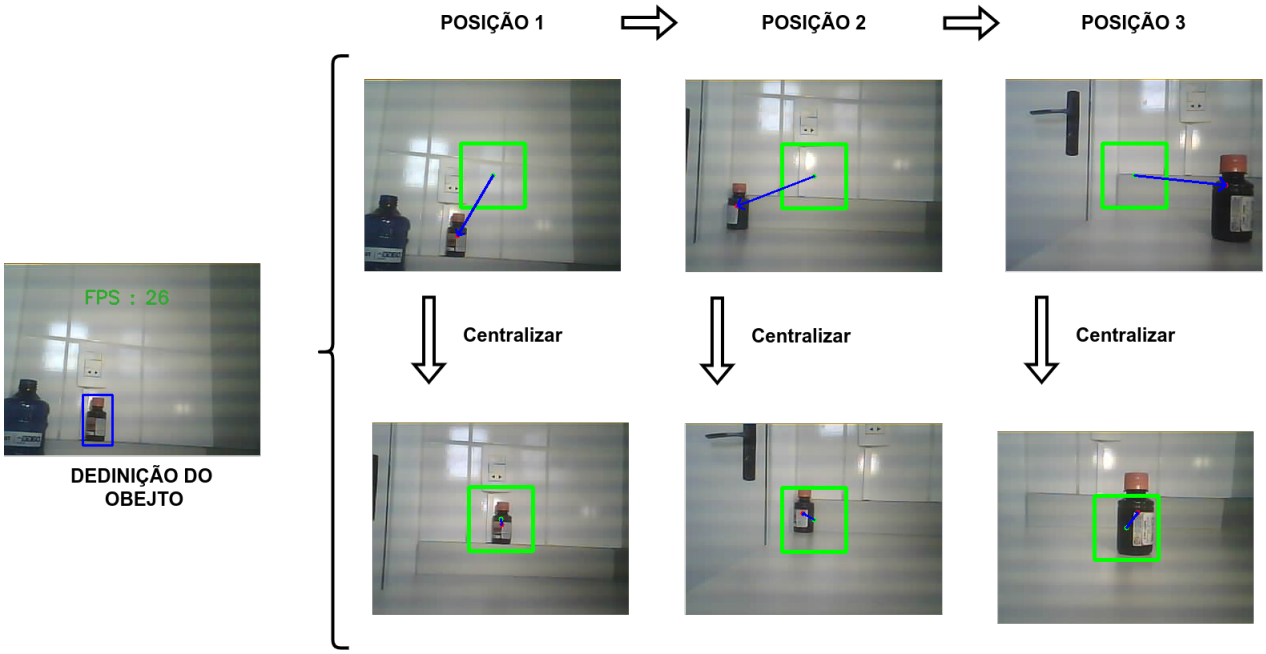
A Figura 58 mostra dois exemplos de imagens nítidas que o algoritmo detecta como borrada. A solução para ambos os casos foi adicionar mais informação na imagem, colocando outro elemento na cena. Foi diminuído também o limiar da variância para 200. Nota-se que no exemplo da segunda linha a variância do Laplaciano aumenta com o acréscimo da garrafa.

Figura 54 – Quatro exemplos de rastreio e centralização do objeto em posições diferentes .



fonte: feito pelo autor.

Figura 55 – Objeto sendo centralizado numa sequência de *frames* contínuos.

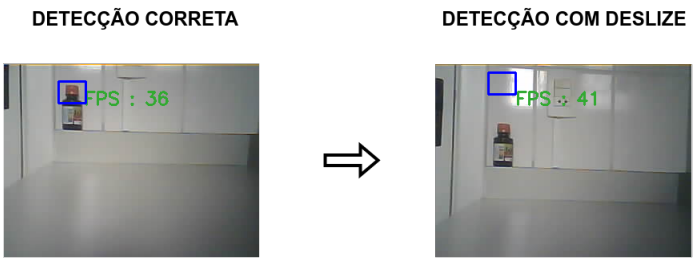


fonte: feito pelo autor.

Figura 56 – Exemplo de oclusão do objeto detectado.

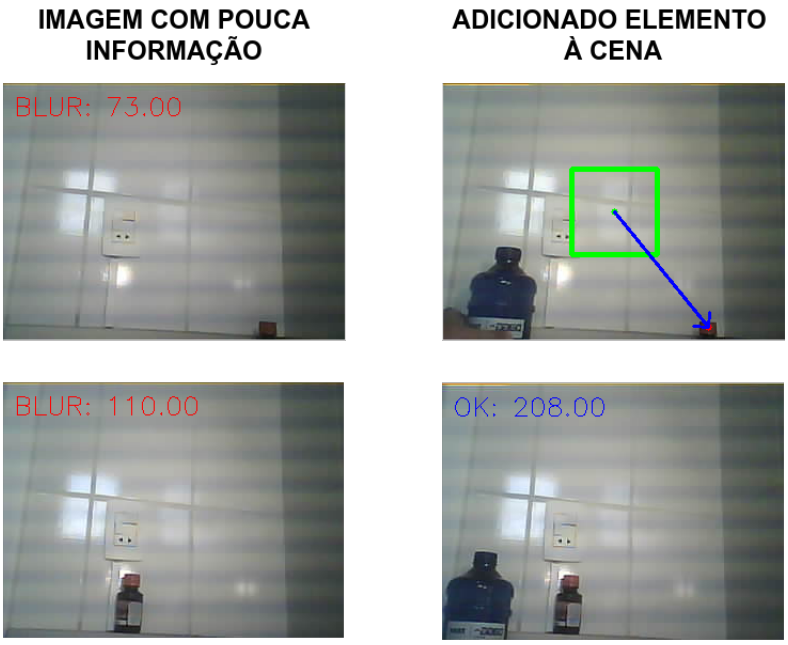
fonte: feito pelo autor.

Figura 57 – Exemplo de deslize indesejado do ROI durante rastreio.



fonte: feito pelo autor.

Figura 58 – Dois exemplos de falso negativo na detecção de foco.



fonte: feito pelo autor.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Neste trabalho, foi estudada a interface entre duas áreas da Engenharia Elétrica: Processamento de Imagens e Sistemas Embarcados. Foi desenvolvido um protótipo que combina um ESP32 e um PC para rastrear objetos na imagem capturada pelo módulo de câmera OV2640. O processamento de imagem foi realizado no computador, utilizando OpenCV e o algoritmo de rastreamento CSRT, que permitiu calcular os ângulos de *pan* e *tilt* necessários para centralizar o objeto na imagem.

O desenvolvimento do protótipo foi uma tarefa desafiadora, mas também extremamente gratificante, pois permitiu aplicar conceitos teóricos adquiridos durante a formação em Engenharia Elétrica. O resultado final demonstrou ser uma solução funcional e eficaz para o problema proposto, permitindo manter o objeto desejado sempre centralizado na imagem capturada pela câmera.

Ao longo deste projeto, foram estudados os desafios enfrentados pela geometria de câmeras em posições múltiplas. O estudo da formação da imagem na câmera foi crucial para compreender as dificuldades encontradas na modelagem de câmeras Pan-Tilt. A formulação desenvolvida para a projeção de câmeras rotacionando em dois eixos proporcionou uma base sólida para a modelagem do sistema e simplificou a correspondência de pontos idênticos em imagens distintas.

Foi possível verificar que o ESP32 é uma plataforma eficiente para a captura de imagens utilizando o módulo câmera OV2640 e para controlar um suporte PTU composto por dois servo motores. O sistema desenvolvido foi capaz de realizar a centralização de um objeto predefinido na cena, graças à integração da captura de imagem, processamento de imagem e controle de movimento. Além disso, a combinação de hardware e software permitiu que o sistema funcionasse de maneira fluida, sem interrupções ou atrasos. Estes resultados demonstram a capacidade de aplicação prática da teoria de geometria de câmeras em sistemas embarcados.

A utilização do algoritmo CSRT demonstrou ser uma opção eficaz para o rastreio de objetos em imagens, sendo capaz de localizar o objeto mesmo em situações com mudanças na velocidade e direção de movimento. Além disso, o uso da linguagem Python e da biblioteca OpenCV permitiu a implementação de um sistema inteiro de captura de imagens, processamento de imagens e rastreio de objetos de maneira relativamente simples e rápida.

Os testes realizados foram importantes para verificar a funcionalidade do sistema e identificar seus limites. A análise dos resultados obtidos permitiu concluir que o sistema é capaz de manter o objeto em foco mesmo em situações de movimentos rápidos e de oclusão do objeto. No entanto, é necessário destacar que a precisão do sistema pode ser afetada por fatores externos, como iluminação inadequada e sinal fraco de rede.

Nesta conclusão, é possível afirmar que os objetivos deste trabalho foram alcançados de forma satisfatória. O sistema de captura de imagens com o ESP32 foi programado para enviar imagens para o PC de forma rápida e confiável. Além disso, o código para processamento de imagens no computador utilizando OpenCV foi desenvolvido e aplicado de forma eficiente, permitindo controlar os servo-motores para centralizar o objeto na foto. Os testes realizados também apontam para a

viabilidade do sistema, comprovando a realização dos objetivos propostos.

Em resumo, o projeto foi uma oportunidade valiosa para desenvolver habilidades em programação, processamento de imagens e controle de sistemas embarcados. Além disso, o resultado final é uma solução inovadora que pode ser aplicada em diversas áreas, incluindo vigilância, segurança, monitoramento de produção, entre outras.

5.2 TRABALHOS FUTUROS

Em resumo, este trabalho apresenta o desenvolvimento de uma solução de rastreio de objetos utilizando o ESP32-cam e processamento de imagens em PC remoto com OpenCV. No entanto, ainda há muito espaço para otimizações e melhorias no sistema. Alguns dos trabalhos futuros que poderiam ser realizados incluem:

- Adicionar uma camada de filtro preditivo de Kalman para prever a próxima posição da localização encontrada pelo algoritmo de rastreio. Um protótipo dessa ideia já foi implementado junto com o sistema deste projeto. A figura Figura 60 do apêndice mostra esse teste.
- Estudar a formação e criação de imagens panorâmicas com o módulo PTU. Um exemplo teste gerado com o OpenCV é representado na figura Figura 61 do apêndice.
- Adicionar uma camada de detecção usando SIFT ou outro descritor local no algoritmo de rastreio para casos em que o objeto saia da cena ou que haja deslize.
- Fazer o processamento de imagens no ESP32, sem precisar de um PC. Essa ideia é possível simplificando o algoritmo de rastreio para localizar um objeto colorido que possa ser segmentado facilmente da cena.

Esses são apenas alguns dos muitos trabalhos futuros que poderiam ser realizados a partir da base desenvolvida neste projeto acadêmico.

REFERÊNCIAS

ALVAREZ, L.; GOMEZ, L.; SENDRA, J. R. Algebraic Lens Distortion Model Estimation. **Image Processing On Line,** v. 1, p. 1–10, 2010.

ARDUCAM. **OV2640 – Specs, Datasheets, Cameras, Features, Alternatives.** [Hong Kong, 2022]. Disponível em: https://www.arducam.com/ov2640/. Acesso em: 31 dez. 2022.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. **Lecture notes in computer science,** *[S.l.]*, v. 3951, p. 404–417, 2006.

BOLME, D. S. et al. Visual object tracking using adaptive correlation filters. **2010 IEEE computer society conference on computer vision and pattern recognition,** [*S.l*], p. 2544–2550, 2010.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, [*S.l*], 2000

CANTZLER, H. Random sample consensus (ransac). **Institute for Perception, Action and Behaviour, Division of Informatics at the University of Edinburgh**, Edimburgo, v. 3, 1981.

CARCAGNì, P. et al. Facial expression recognition and histograms of oriented gradients: a comprehensive study. **SpringerPlus,** [*S.l*], v. 4, p. 645, 2015.

DAVIS, J.; CHEN, X. Calibrating pan-tilt cameras in wide-area surveillance networks**. IEEE International Conference on Computer Vision,** [*S.l*], v. 2, p. 144–144, 2003.

DELLAERT, F.; THORPE, C. Robust car tracking using kalman filtering and bayesian templates. **Conference on intelligent transportation systems,** [*S.l*], v. 1, 1997.

DERPANIS, K. G. The harris corner detector. **York University Journal,** York, v. 2, p. 1–2, 2004

DETONE, D.; MALISIEWICZ, T.; RABINOVICH, A. Superpoint: Self-supervised interest point detection and description. **Proceedings of the IEEE conference on computer vision and pattern recognition workshops.** [*S.l.*], p. 224–236, 2018.

DURINI, D. **High performance silicon imaging: fundamentals and applications of cmos and ccd sensors.** Sawston: Woodhead Publishing, 2019.

EXNER, D. et al. Fast and robust camshift tracking. **2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops,** [*S.l.*], p. 9–16, 2010.

FELICIANO**, R. Micro servo 9G desmontado.** [São Paulo], 2019. Disponível em: https://www.youtube.com/watch?v=lYR1n-ssGdM. Acesso em: 31 dez. 2022.

FISHER, R. B.; OLIVER, P. **Multi-variate cross-correlation and image matching.** Edimburgo: University of Edinburgh, Department of Artificial Intelligence, 1995.

FLEET, D.; WEISS, Y. Optical flow estimation. **Handbook of mathematical models in computer vision.** [*S.l.*]: Springer, 2006.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais.** [*S.l.*]: Editora Blucher, 2000.

HARTLEY, R.; ZISSERMAN, A. **Multiple view geometry in computer vision.** [*S.l.*]: Cambridge University Press, 2003.

HESTER, C. F.; CASASENT, D. Multivariant technique for multiclass pattern recognition. **Applied Optics.** [*S.l.*], v. 19, n. 11, p. 1758–1761, 1980.

KIKUCHI, D. Y. **Sistema de controle servo visual de uma câmera pan-tilt com rastreamento de uma região de referência.** Tese (Doutorado) — Universidade de São Paulo, 2007.

LOWE, G. Sift-the scale invariant feature transform. **International Journal of Computer Vision,** [*S.l.*], v. 2, n. 91-110, p. 2, 2004.

LUKEZIC, A. et al. Discriminative correlation filter with channel and spatial reliability. **Proceedings of the IEEE conference on computer vision and pattern recognition,** [*S.l.*], p. 6309–6318, 2017.

MURTY, M. N.; DEVI, V. S. **Introduction to pattern recognition and machine learning.** [*S.l.*]: World Scientific, 2015.

MUSA, L. **CMOS Active Pixel Sensors.** [*S.l.*]: CERN Courier, 2018.

NAKAMURA, J. **Image sensors and signal processing for digital still cameras.** [*S.l.*]: CRC Press, 2017.

OGORZALEK, J. P. **Computer Vision Tracking of sUAS from a Pan/Tilt Platform.** Tese (Doutorado) — Virginia Tech, 2019.

OMNIVISION. **OMNIVISION Takes Lead in Mobile Cameras with New 2 Megapixel Sensor.** [*S.l.*]: 2005. Disponível em: https://www.ovt.com/press-releases/omnivision-takes-lead-in-mobile-cameras-with-new-2-megapixel-sensor/. Acesso em: 31 dez. 2022.

OMNIVISION. **OV2640 Color CMOS UXGA (2.0 MegaPixel) CAMERACHIP with OmniPixel Technology.** [*S.l.*]: 2006. Disponível em: https://www.uctronics.com/download/cam_module/OV2640DS.pdf. Acesso em: 31 dez. 2022.

PAPANIKOLOPOULOS, N. P.; KHOSLA, P. K.; KANADE, T. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision. **IEEE transactions on robotics and automation,** [*S.l.*], v. 9, n. 1, p. 14–35, 1993.

PECH-PACHECO, J. L. et al. Diatom autofocusing in brightfield microscopy: a comparative study. **International Conference on Pattern Recognition,** [*S.l.*], v. 3, p. 314–317, 2000.

RASHIDIAN, B. **Evolution of CMOS Technology.** [S.l.]: Teledyne, 2011.

ROBINSON**, A. Discriminative correlation filters in robot vision.** Tese (Doutorado) — Linköping University, 2021.

ROSEBROCK, A. **OpenCV Object Tracking.** [*S.l.*]: 2018. Disponível em: https://pyimagesearch.com/2018/07/30/opencv-object-tracking/. Acesso em: 20 dez. 2022.

RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. **2011 International conference on computer vision,** [*S.l.*], p. 2564–2571, 2011.

SARLIN, P.-E. et al. Superglue: Learning feature matching with graph neural networks. **CVF conference on computer vision and pattern recognition,** [*S.l.*], p. 4938–4947, 2020.

SCHINDELIN, J. et al. Fiji: an open-source platform for biological-image analysis. **Nature methods,** [*S.l.*], v. 9, n. 7, p. 676–682, 2012.

SEDRA, A. S. et al. **Microelectronic circuits.** [*S.l.*]: Oxford University Press, 2004.

SHI, J. et al. Good features to track. **IEEE conference on computer vision and pattern recognition.** [*S.l.*], p. 593–600, 1994.

SZELISKI, R. **Computer vision: algorithms and applications.** [*S.l.*]: Springer Nature, 2022.

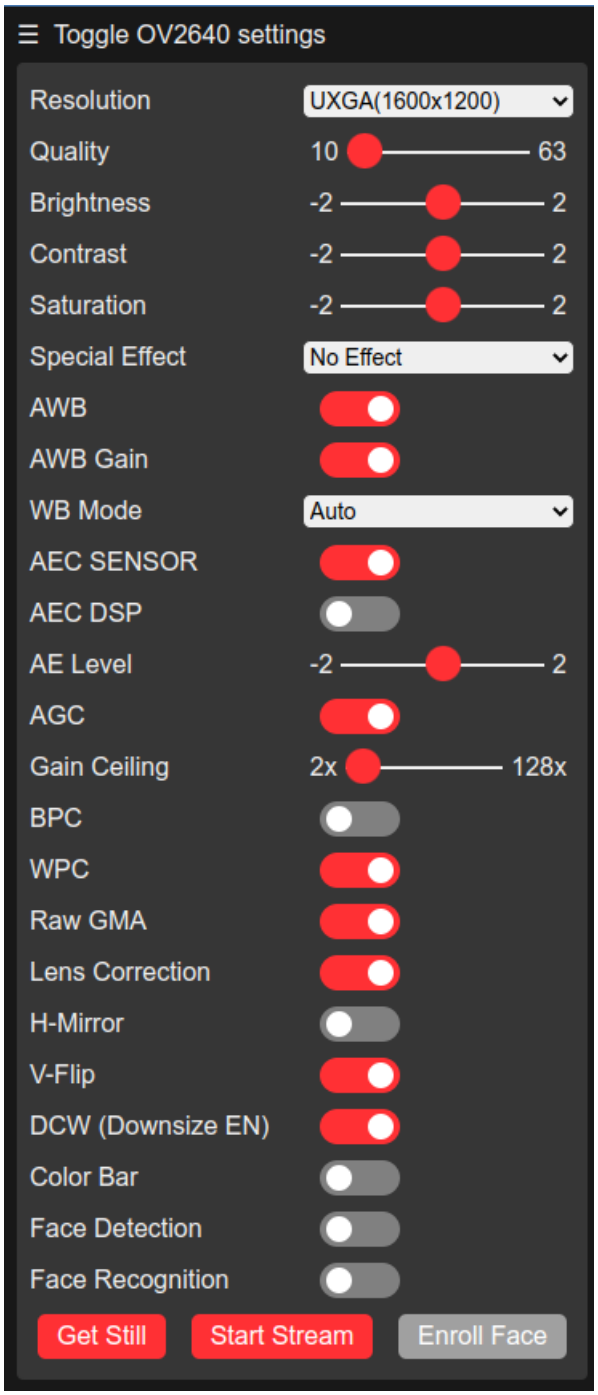
TURCHETTA, R.; SPRING, K**. Introduction to CMOS Image Sensors.** [*S.l.*]: 2015. Disponível em: https://micro.magnet.fsu.edu/primer/digitalimaging/cmosimagesensors.html/. Acesso em: 20 dez. 2022.

WEIJER, J. Van de; SCHMID, C. Applying color names to image description. **IEEE International Conference on Image Processing,** [*S.l.*], v. 3, p. 480–493, 2007.

Wikipedia contributors. **Distortion (optics).** [São Francisco, CA: Wikimedia Foundation, 2022]**.** Disponível em: https://en.wikipedia.org/w/index.php?title=Distortion_(optics)&oldid=1091172419. Acesso em: 20 dez. 2022.

APÊNDICE A – INTERFACE DE CONFIGURAÇÃO DA CÂMERA

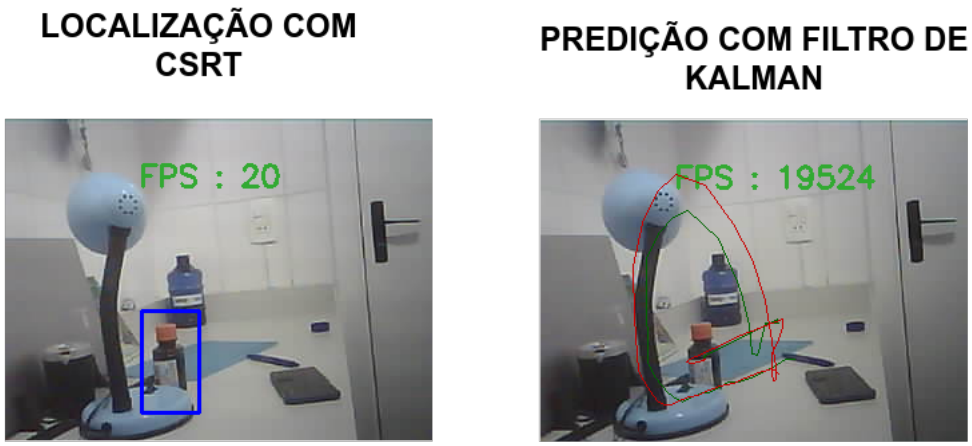
Figura 59 – Interface de configuração da câmera OV2640 gerada pelo servido ESP32.



fonte: Produção do próprio autor.

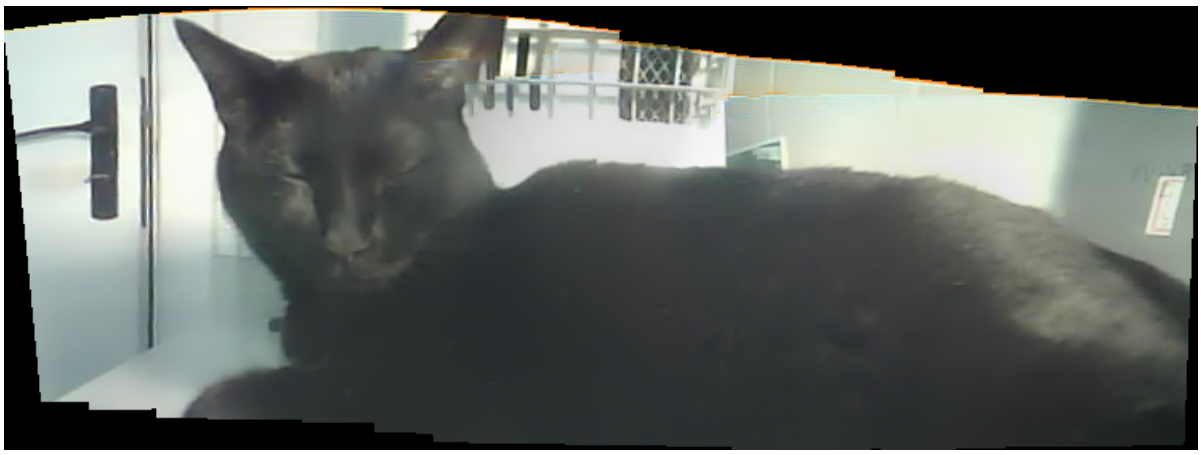
APÊNDICE B – PROTOTIPOS REALIZADOS PARA TRABALHOS FUTUROS

Figura 60 – Predição com filtro de Kalman em vermelho e CSRT em verde.



fonte: Produção do próprio autor.

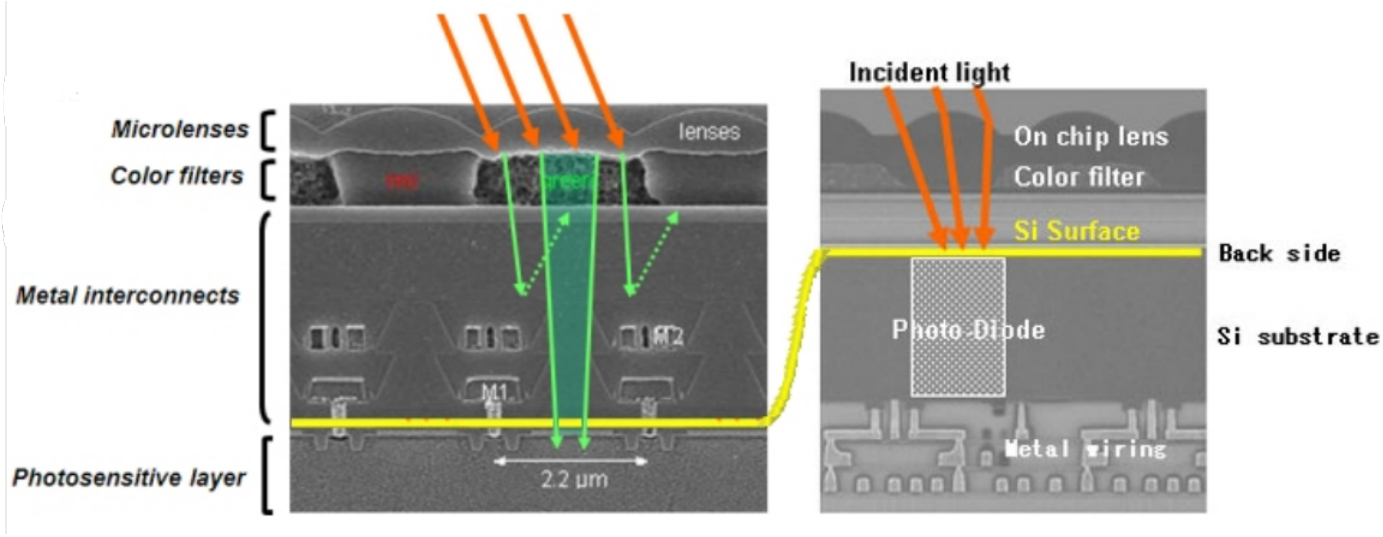
Figura 61 – Panorâmica gerado com módulo PTU e função do OpenCV.



fonte: Produção do próprio autor.

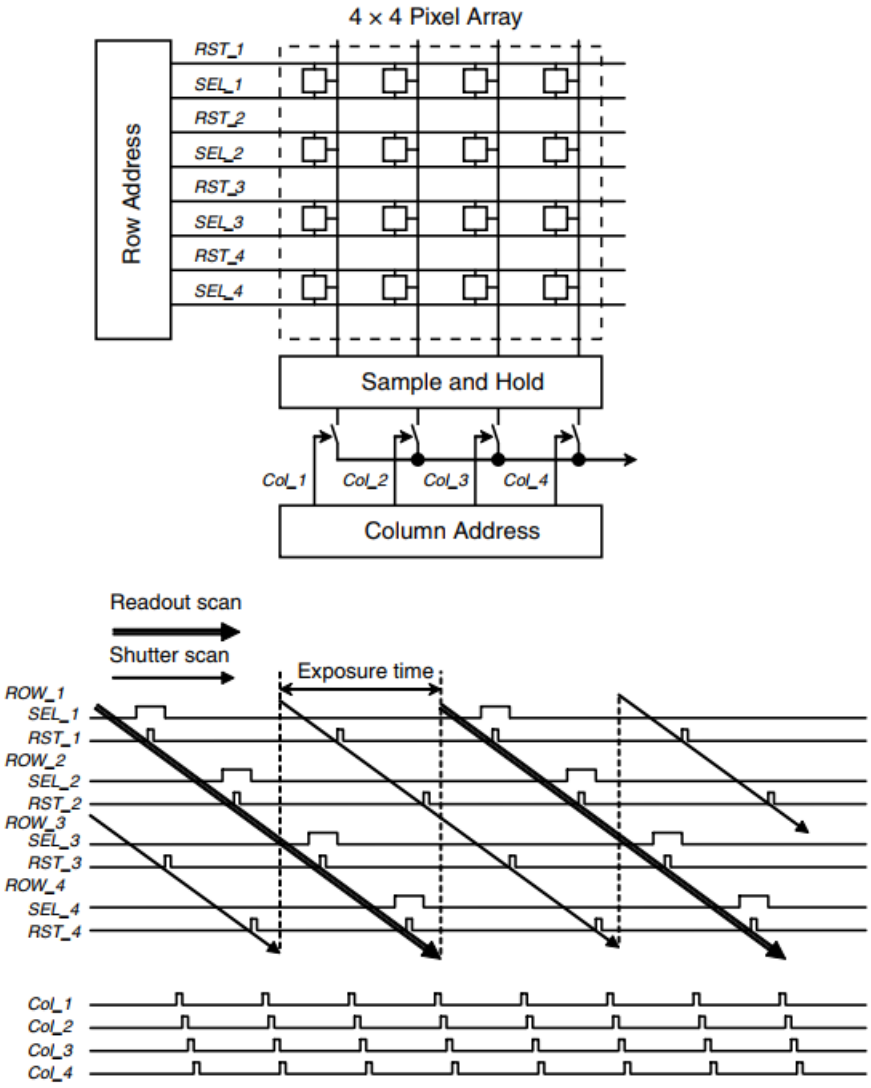
ANEXO A – FIGURAS COMPLEMENTARES SENSOR CMOS

Figura 62 – Imagens de microscópio eletrônico: vista lateral de unidade de fotocélula do sensor CMOS.
Dois tipos de sensores estão representados: o iluminado frontalmente (esquerda) e o retroiluminado. Enquanto que no primeiro caso há uma camada de metal entre a microlente e o sensor, no segundo o substrato está atrás do diodo.
Em ambos os casos estão traçados os raios luminosos que atingem o sensor. Pode-se notar que no sensor da esquerda a energia em forma de luz é dissipada no substrato intermediário.



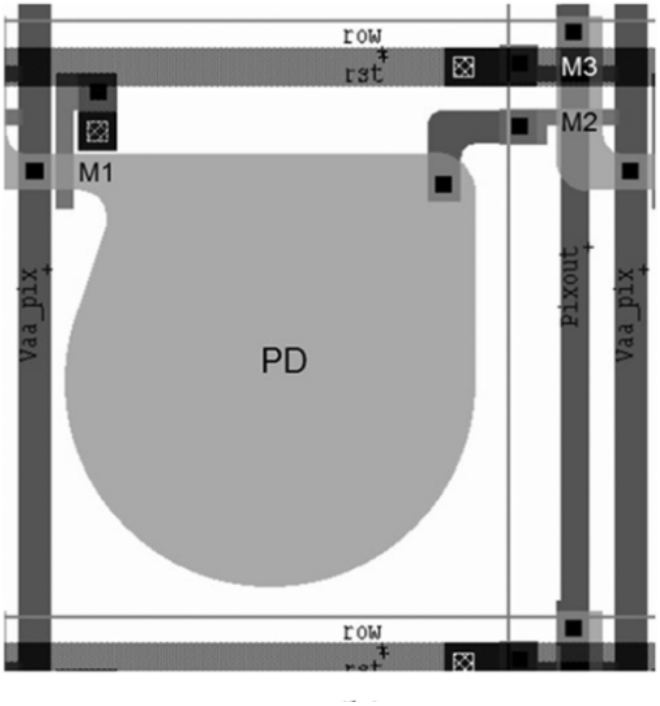
fonte: Musa (2018).

Figura 63 – Diagrama do processo de aquisição de imagem do tipo *Rolling Shutter*.



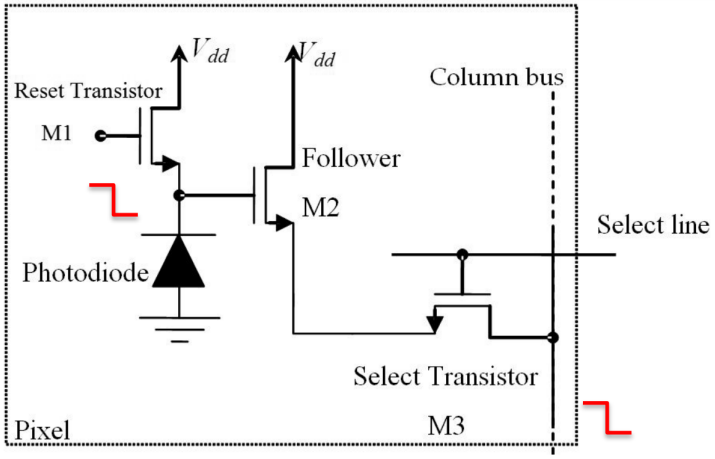
fonte: Nakamura (2017).

Figura 64 – Layout do circuito integrado para o modelo de pixel 3T. Na figura, nota-se as camadas que formam os três transistor, anotados com M1,M2 e M3. A camada central PD é o fotodiodo.



fonte: Musa (2018).

Figura 65 – Configuração mínima da fotocélula do sensor CMOS. Este circuito faz o controle de um pixel. O transistor M3 é chamado de seletor de coluna, M2 está como dreno comum e M1 é o gatilho para zerar a carga no diodo.



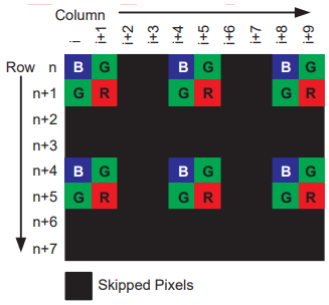
fonte: Musa (2018).

ANEXO B – FIGURAS COMPLEMENTARES MÓDULO OV2640

Figura 66 – Material de promoção do módulo OV2640 distribuído entre 2005 e 2009 pela OmniVision.

fonte: Arducam (2022).

Figura 67 – Disposição dos píxeis coloridos no sensor do módulo OV2640. A filtragem é feita de acordo com o mosaico de Bayer (BG/GR).
Existem espaços vazios porque o sensor está programado para capturar imagens do tipo VGA. Assim, com resolução menor há ganho de FPS, pois menos píxeis são capturados.



fonte: OmniVision (2006).