Universidade de Brasília - UnB
Faculdade do Gama - FGA
Engenharia Eletrônica

Sistema de Localização de Robôs Móveis Usando Marcadores ArUco

Autor: Hércules Ismael de Abreu Santos
Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF
2022

Hércules Ismael de Abreu Santos

Sistema de Localização de Robôs Móveis Usando Marcadores ArUco

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Universidade de Brasília - UnB

Faculdade do Gama - FGA

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2022

Hércules Ismael de Abreu Santos
Sistema de Localização de Robôs Móveis Usando Marcadores ArUco/ Hércules Ismael de Abreu Santos. – Brasília, DF, 2022-
51 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade do Gama - FGA , 2022.

1. Computer Vision. 2. Embedded Systems. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade do Gama. IV. Sistema de Localização de Robôs Móveis Usando Marcadores ArUco

CDU 02:141:005.6

Hércules Ismael de Abreu Santos

Sistema de Localização de Robôs Móveis Usando Marcadores ArUco

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 5 de novembro de 2021:

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Orientador

Prof. Dr. Diogo Caetano Garcia
Convidado 1

Prof. Dr. Jones Yudi Mori Alves da Silva
Convidado 2

Brasília, DF
2022

Agradecimentos

Quero agradecer primeiramente a minha família por me dar suporte no decorrer do curso, em especial a meus tios José Martins e Creuza Abreu. Agradeço a meu pai, que já não está entre os vivos, mas representa para mim o maior exemplo de homem que tive, e agradeço a minha mãe por me dar suporte e estar comigo.

Agradeço a meus colegas, que junto comigo enfrentaram os desafios de cursar um curso de graduação, e me ajudaram em diversas ocasiões.

Agradeço também ao meu orientador Daniel Mauricio Muñoz Arboleda, que me orientou sempre de forma atenciosa e instrutiva.

Resumo

A navegação de robôs móveis autônomos depende, entre outras coisas, da localização de robôs. Parte do processo de navegação consiste em adquirir dados sobre o ambiente em que o robô realizará a navegação. Estes dados podem ser adquiridos por uma diversidade de sensores, e uma das formas de adquirir estes dados é por meio de câmeras. O campo da visão computacional tem como objeto de estudo o desenvolvimento de técnicas para obter informações sobre o mundo real através de imagens utilizando a computação.

Uma técnica que pode ser usada para localizar robôs é a utilização de marcadores fiduciais nos robôs para identificá-los e localizá-los. Um algoritmo utilizado para realizar esse tipo de operação é o ArUco, que permite a identificação e localização de marcadores fiduciais.

O presente trabalho tem como objetivo o desenvolvimento de um sistema de localização de robôs baseado no algoritmo ArUco, utilizando uma arquitetura em hardware para implementar pontos chave deste algoritmo, são eles a operação de binarização de imagens em tons de cinza, a operação de extração de contornos e a operação de aproximação de polígonos.

O sistema proposto foi primeiramente testado utilizando uma solução realizada somente em software. Os testes realizados mostraram a eficiência do algoritmo, que identificou corretamente os marcadores ArUco. Depois foi testada a arquitetura em hardware proposta, e observou-se que as implementações da binarização de imagens, e a aproximação de polígonos não obteve uma aceleração do algoritmo, enquanto que a operação de extração de contornos acelerou o algoritmo, mas apenas para imagens menores do que 50 x 50 pixels.

Palavras-chaves: Localização de Robôs Móveis, Visão Computacional, FPGA, ArUco, HLS.

Abstract

The navigation of mobile robots depends, among other things, on the localization of robots. Part of the navigation process is the data acquisition about the environment in which the robot will navigate. These data can be acquired by using a diversity of sensors, and one way of doing it is by using cameras. The field of computer vision aims to study techniques for the extraction of information in images, by using computers.

One technique that can be used for the localization of robots is the use of fiducial markers on robots for their localization and identification. An algorithm that can accomplish this task is ArUco, identifying and tracking the fiducial markers.

The goal of this work is the development of a mobile robot localization system based on ArUco, using a hardware architecture that implements key points in this algorithm, respectively the binarization of a gray scale image, the contour extraction operation and the polygonal approximation operation.

The proposed system was tested utilizing a solution based only on software. The tests proved the efficiency of the algorithm, which could identify well the ArUco markers. Then was tested the proposed architecture for implementation of key points in the algorithm, in order to accelerate the algorithm. The tests of the architecture showed that this architecture was not able to achieve a lower execution time than the software solution's execution time.

Key-words: Mobile Robot Localization, Computer Vision, FPGA, ArUco, HLS.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMBA *Advanced Microcontroller Bus Architecture*

ASIC *Application Specific Integrated Circuit*

AXI *Advanced eXtensible Interface*

CMOS *Complementary metal-oxide-semiconductor*

CPU *Central Process Unit*

DMA *Direct Memory Access*

DRAM *Dynamic Random Access Memory*

DSP *Digital Signal Processor*

FPGA *Field Programmable Gate Arrays*

FSM *Finite State Machine*

GPS *Global Positioning System*

GPU *Graphic Process Unit*

HDL *Hardware Description Language*

HLS *High-Level Synthesis*

HSV *Hue Saturation Value*

IMU *Inertial Measurement Unit*

LfD *Learning from Demontration*

PIBIC *Programa Institucional de Bolsas de Iniciação Cientifica*

PLL *Phase-Locked Loop*

RAM *Random Access Memory*

RGB *Red Green Blue*

SIMD *Single Instruction, Multiple Data)*

SLAM *Simultaneous Localization And Mapping*

SoC *System on Chip*

SVM *Support Vector Machine*

VANT *Veículo Aéreo Não Tripulado*

VGA *Video Graphics Array*

Sumário

1 Introdução

1.1 Contextualização

A navegação autônoma de robôs é um campo de estudos com aplicações que trazem vantagens estratégicas em diversos sistemas. Ter um robô com a capacidade de se locomover de forma autônoma pode ser útil para automatizar tarefas em empresas como por exemplo tarefas relacionadas a logística. Uma outra aplicação da navegação autônoma é o caso em que é necessário realizar tarefas em lugares inóspitos, como por exemplo o caso de sondas espaciais (SHEN, 2011), ou casos de perigo como quando a polícia precisa desarmar uma bomba . Nestes casos um robô autônomo pode ser muito útil, pois ele permite que se realize tarefas de forma remota, sem a necessidade de ter humanos presentes no local, preservando a vida e a saúde do operador.

O desenvolvimento de sistemas de navegação autônoma para robôs móveis é uma atividade multidisciplinar, que depende de etapas que exploram campos como a eletrônica, o desenvolvimento de software, mecânica e até mesmo o estudo da cognição. Um sistema de navegação autônoma, de forma geral, envolve 5 etapas, que são: A leitura dos dados, o mapeamento do ambiente, a localização, o planejamento da rota e o acionamento de atuadores para executar a rota (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Um caso específico da navegação autônoma de robôs é o caso em que vários robôs cooperam para a execução de uma tarefa. Chama-se de sistema de robótica multiagente quando se tem vários robôs cooperando para a realização de uma tarefa. Um exemplo de aplicação deste tipo de tecnologia é o uso de múltiplos robôs para a patrulha de grandes áreas. Este tipo de sistema tem um desafio maior em sua implementação, pois além destes robôs executarem suas tarefas individuais, eles devem cooperar com os outros robôs, de forma que a tarefa de um robô não interfira a de outro.

Para solucionar o problemas na navegação autônoma de robôs móveis, por vezes é proposta uma solução baseada em algoritmos de aprendizado de máquina. De uma forma geral, tais algoritmos dependem de uma base de dados, e um método de aprendizagem para que o algoritmo possa aprender a tarefa de interesse. Tais métodos podem ser treinamentos supervisionados, ou treinamentos não supervisionados. Treinamentos supervisionados tendem a ser mais rápidos do que os treinamentos não-supervisionados. Isso porque o treinamento supervisionado tem a interferência de um ser humano no processo de aprendizagem (SEKAJ; CÍFERSKÝ; HVOZDÍK, 2019) (TANWANI; BILLARD, 2013).

Um tipo de treinamento supervisionado é o *Learning from Demonstration* (LfD), ou aprendizado por demonstração. Este tipo de treinamento se baseia na ideia de que

seres humanos raramente aprendem algo completamente do início, sem nenhuma referência ou ajuda de outras pessoas. Em geral, para aprender algo, os seres humanos partem de conhecimentos prévios, ou da ajuda de outras pessoas, que podem demonstrar como executar determinada tarefa, e a partir disso a pessoa consegue aprender a executar a tarefa de interesse (HAYES; DEMIRIS, 1994) (CALINON; GUENTER; BILLARD, 2007) (SHEN et al., 2018). Utilizando este princípio, é possível desenvolver um algoritmo que utilize a demonstração de um *"professor"* como referência, e a partir dela consiga, muitas vezes, fazer um treinamento que chegue à convergência mais rapidamente do que utilizando técnicas de treinamento não-supervisionado. Outros exemplos de técnicas usadas em treinamentos supervisionados são redes neurais, regressão linear, SVM(Support Vector Machine).

Estes tipos de técnicas podem ser utilizados em diversos módulos de um sistema de navegação autônoma. O presente trabalho se propõe a realizar um sistema de localização para ser utilizado em um sistema de navegação autônoma, que se baseia em LfD para o planejamento de rotas.

A etapa de localização, em um sistema de navegação autônoma, tem como objetivo localizar o robô dentro de um ambiente, fornecendo a posição e orientação do robô em relação a um mapa do ambiente que o robô se encontra. Esta etapa é importante pois fornece informações relevantes para os níveis mais altos de abstração da navegação, que envolvem tomada de decisão e planejamento de rotas.

Para realizar a localização do robô, pode-se utilizar uma diversidade de sensores e técnicas, desde sensores inerciais a sensores ultrassom e até GPS. Para este trabalho será explorado o uso da visão computacional, que é o campo de estudo que busca extrair informações sobre o mundo real através de imagens. Este tipo de técnica é promissora pois imagens carregam informações relevantes sobre o mundo real, não por coincidência, o ser humano e uma diversidade de animais utiliza-se da visão como o sentido principal para se localizar, e entender o ambiente em que estão.

Este trabalho dá prosseguimento a uma pesquisa iniciada em um Projeto de Iniciação Científica (SANTOS, 2021a) que foi realizado através do Programa Institucional de Bolsas de Iniciação Cientifica(PIBIC) da UnB. O projeto de iniciação científica teve como objetivo principal a análise e comparação de algoritmos para a localização de robôs. Foram analisados algoritmos que se baseiam em cores para localizar robôs, e um algoritmo que utiliza marcadores fiduciais para a localização e identificação de robôs, o ArUco. Os resultados da pesquisa indicaram o ArUco como o algoritmo de melhor desempenho, porém ao se analisar o tempo de execução do algoritmo observou-se que o tempo de execução é muito grande para permitir uma implementação em tempo real do sistema de localização em uma plataforma embarcada.

Neste trabalho dá-se o prosseguimento a pesquisa estudando a factibilidade da

implementação em hardware do algoritmo ArUco, para obter-se uma aplicação em que seja possível a execução em tempo real.

1.2 Justificativa

Sistemas embarcados tem uma limitação de desempenho maior do que aplicações que utilizam computadores de escritório ou notebooks por exemplo. O processamento de algoritmos de visão computacional pode ser custoso computacionalmente, pois exige um alto acesso a memória, além de cálculos utilizando matrizes, como multiplicação de matrizes, convoluções, entre outros. Esta limitação torna necessário o estudo de alternativas que acelerem a execução das tarefas envolvidas nas técnicas de localização de robôs móveis usando visão computacional, de forma que o processamento dos algoritmos de navegação possam ser executados em tempo real.

Uma forma de obter um tempo de execução adequado para estes algoritmos, é a aceleração destes algoritmos a nível de hardware. Isto pode ser obtido utilizando dispositivos FPGAs (*Field Programmable Gate Arrays*), em conjunto com processadores de software embarcados, tecnologia conhecida como Sistema em Chip (SoC). A FPGA permite a implantação do algoritmos a nível de hardware, com a possibilidade de paralelização das operações realizadas. Paralelizar operações quando se está manipulando imagens é uma forma eficiente de acelerar algoritmos, pois a manipulação de imagens incide naturalmente na manipulação de matrizes, que matematicamente depende de operações que se repetem em cada um dos elementos de uma matriz, e ocorrem paralelamente.

Desta forma, o estudo da implantação destes algoritmos em uma FPGA pode tornar viável a implementação de tais algoritmos em sistemas embarcados com recursos limitados de hardware.

Em particular, (SANTOS, 2021a) no projeto de iniciação científica realizado antes deste trabalho, foi verificado que os algoritmos implementados foram muito lentos para serem implementados em software em uma plataforma embarcada, e verificou-se que implementar partes do software em uma FPGA poderiam acelerar o algoritmo. Os estágios do algoritmo que mostraram-se viáveis de serem implementados foram o *Local Adaptive Threshold*, o algoritmo de Suzuki-Abe, e o algoritmo de Douglas-Peucker. Estes algoritmos serão explanados mais a fundo na seção de Fundamentação Teórica.

1.3 Objetivos

Desenvolver um sistema de localização de robôs móveis, utilizando uma arquitetura de hardware do algoritmo ArUco baseado em SoC FPGAs.

1.3.1 Objetivos Específicos

- Construção de uma base de dados a partir de imagens e vídeos de robôs móveis realizando tarefas coordenadas em uma arena de testes.
- Desenvolvimento de uma arquitetura em hardware de um módulo que implementa o algoritmo *Local Adaptive Threshold*.
- Desenvolvimento de uma arquitetura em hardware de um módulo que implementa o algoritmo *Suzuki-Abe* de extração de contornos.
- Desenvolvimento de uma arquitetura em hardware de um módulo que implementa o algoritmo de aproximação poligonal de Douglas-Peucker.

1.4 Contribuições do Trabalho

Este trabalho propõe arquiteturas em hardware para a implementação de uma parte dos algoritmos envolvidos na detecção de marcadores fiduciais ArUco. Essa implementação foca nas partes do código que tem o maior consumo de tempo na execução da aplicação. Para testar o desempenho da aplicação será utilizada uma base de dados com fotos coletadas durante a pesquisa, que contém caixas com marcadores ArUco, simulando robôs em uma arena. Esta base de dados é uma contribuição deste trabalho pois permite que pesquisas futuras se beneficiem destas imagens para realizar pesquisas no âmbito da navegação autônoma de robôs.

São fornecidas neste trabalho medidas de desempenho e um estudo comparativo da implementação em hardware dos algoritmos estudados, em contraste com a implementação em software.

1.5 Organização do Documento

Este documento é composto por 5 capítulos. No Capítulo 2, é exposta a revisão bibliográfica e o estado da arte que foram utilizados como referência para definir as soluções do problema. São ainda apresentados neste capítulo as descrições teóricas dos algoritmos de visão computacional a serem estudados. No Capítulo 3, apresenta-se a proposta de metodologia aplicada no trabalho. Também é descrito neste capítulo o procedimento de coleta de imagens para a base de dados e o procedimento de teste da implementação em hardware para comparação com a implementação via software. No Capítulo 4, são apresentados os resultados obtidos nos experimentos conduzidos, comparando-se as soluções estudadas, e discutindo-se o significado dos resultados obtidos. Finalmente, no Capítulo 5, são levantadas as conclusões.

2 Fundamentação Teórica

Neste capítulo, é apresentado o marco teórico do projeto a partir dos principais conceitos necessários para entender os algoritmos de segmentação por cores utilizados, o algoritmo de identificação de marcadores fiduciais da biblioteca ArUco e arquiteturas de hardware reconfigurável. Também será discutido neste capítulo o estado da arte de sistemas de visão computacional acelerados por hardware reconfigurável.

2.1 Definições Iniciais

Para este trabalho, se definirá imagem como sendo uma matriz com 3 dimensões, a primeira sendo o eixo x (horizontal), a segunda sendo o eixo y (vertical), e a terceira sendo o canal de cor.

Um pixel I(x, y) é da forma:

$$I(x, y) = (r_{xy}, g_{xy}, b_{xy}) \tag{2.1}$$

com r_{xy}, g_{xy} e b_{xy} assumindo valores de 0 a 255.

Para o caso de uma imagem em tons de cinza, ou uma imagem, o pixel I(x, y) será da forma:

$$I(x, y) = i_{xy} \tag{2.2}$$

com i_{xy} variando de 0 a 255.Para imagens binárias, i_{xy} será 0 ou 1.

No caso da imagem binária, dois tipos de regiões serão definidas. Regiões compostas por pixels de valor '1' serão chamados *objetos*, enquanto que *buracos* serão definidos

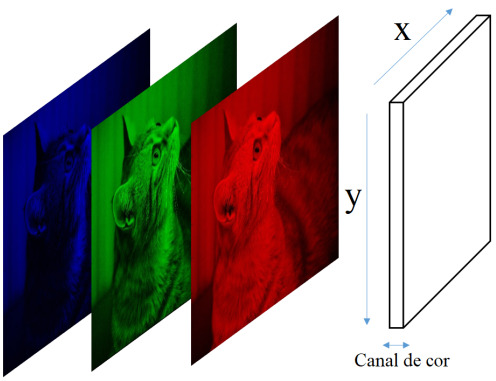

Figura 1 – Dimensões de uma imagem

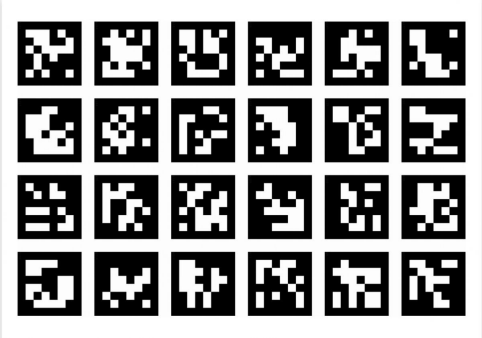

Figura 2 – Exemplos de marcadores fiduciais ArUco.

como regiões compostas por pixels de valor ’0’.

A imagem binária para os objetivos deste trabalho deve ter uma borda externa composta por ’0’s. De modo que a região de ’0’s que engloba a borda da imagem, é chamada de fundo da imagem.

Também será definido o contorno de uma região, como sendo composto de ’1’s, tanto para um objeto como para um buraco.

2.2 ArUco

Para se rastrear objetos em uma imagem, diversas técnicas podem ser utilizadas para extrair estes objetos de uma imagem. Um tipo de técnica que pode ser usada na visão computacional por exemplo é a segmentação de imagens. Segmentar uma imagem digital consiste em dividir esta imagem em regiões ou objetos, que são conjuntos de pixels. A partir destes segmentos pode-se, em muitos casos, facilitar a obtenção de informações de uma imagem.

Uma forma de se segmentar uma imagem é a limiarização de imagens. Esta técnica consiste basicamente em transformar uma imagem em tons de cinza para uma imagem binária a partir de um limiar. Por exemplo, pode-se escolher um limiar igual a 127 para se aplicar em uma imagem e a partir dele todos os pixels maiores que este limiar serão iguais a ’1’, e todos os pixels menores serão iguais a ’0’.

A partir de técnicas de segmentação de imagem, em conjunto com outras técnicas do processamento de imagens é possível fazer algoritmos mais complexos que são capazes de fazer reconhecimento de imagens por exemplo. Um exemplo de um algoritmo que une diversas técnicas do processamento de imagens e da visão computacional é o ArUco.

ArUco (MUÑOZ-SALINAS; MEDINA-CARNICER, 2020) (GARRIDO-JURADO et al., 2014), é uma biblioteca de visão computacional, que tem como objetivo fornecer ferramentas para identificação de marcadores fiduciais quadrados. Exemplos destes marcadores são mostrados na figura 2. O ArUco é capaz de identificar tais marcadores e

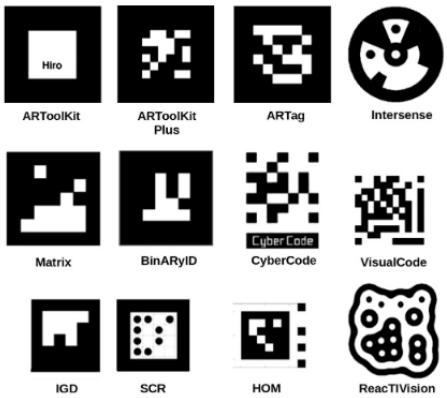

Figura 3 – Exemplos de outros tipos de marcadores fiduciais. (GARRIDO-JURADO et al., 2014)

fornecer a posição destes, a sua orientação, além de ter funcionalidades como a calibração de câmeras e aplicações em realidade aumentada. Os marcadores ArUco são compostos basicamente por um quadrado, este quadrado composto por uma borda negra, e externo a ele há também uma borda branca, que é necessária para o algoritmo funcionar de forma eficiente. Dentro do marcador ArUco, temos uma matriz de quadrados, que podem ser pretos ou brancos. Esta matriz tem tamanho variável, e depende do dicionário de marcadores que está sendo utilizado. Um dicionário de marcadores fiduciais, é uma coleção de marcadores. Ao identificar um possível marcador fiducial, o algoritmo deve comparar o marcador com o dicionário de marcadores, e se o marcador não pertence ao dicionário utilizado, não é considerado um marcador fiducial pelo algoritmo.

Além dos marcadores ArUco, existem outros tipos de marcadores fiduciais, como na Figura 3. Estes marcadores podem ser de várias formas. Para este trabalho será utilizado o ArUco, pois este *framework* fornece informação de pose, tem um algoritmo eficiente de correção de erro e é robusto à oclusão. Além disso, o ArUco é de código aberto, sendo parte da biblioteca de visão computacional OpenCV.

A figura 4 mostra um fluxograma do algoritmo utilizado na biblioteca ArUco para detectar um marcador fiducial. Por simplicidade, no diagrama foram omitidos os estágios de leitura e redimensionamento da imagem, embora tenham sido utilizados nos algoritmos desenvolvidos neste trabalho.

2.2.1 Imagem para tons de cinza

Após a leitura e redimensionamento da imagem, o primeiro passo é transformar a imagem para tons de cinza. É importante salientar que para o algoritmo ArUco, a informação de cor não é importante para detectar os marcadores.

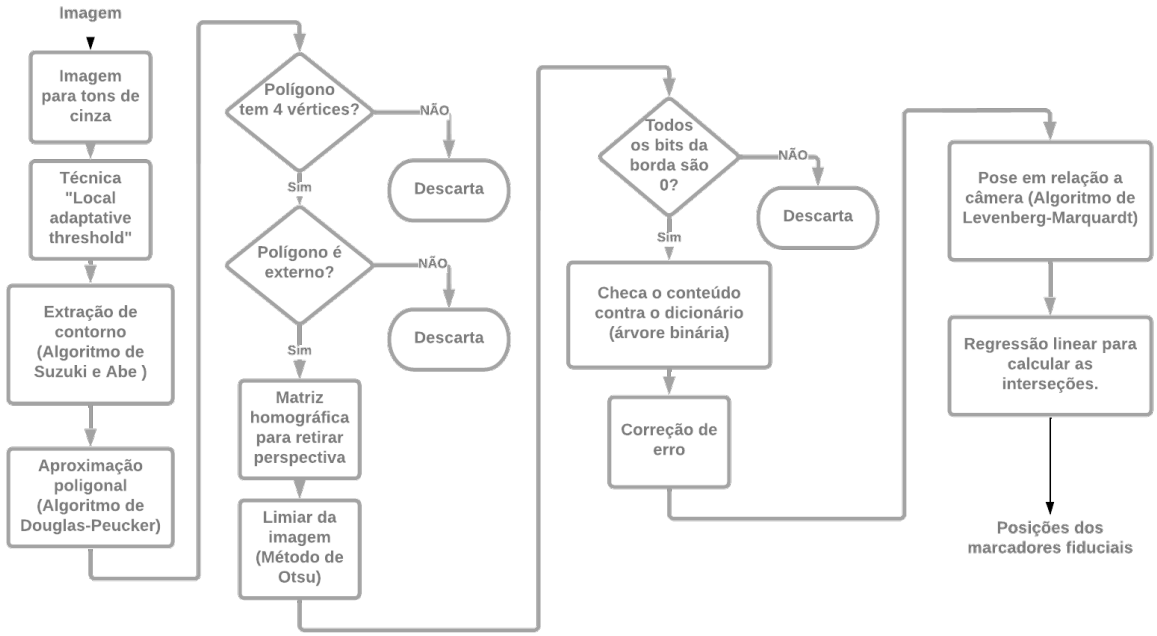

Figura 4 – Fluxograma do algoritmo ArUco

2.2.2 Local Adaptive Threshold

Após transformar a imagem em tons de cinza, o próximo passo é realizar uma transformação para uma imagem binária. Este processo é realizado por um algoritmo que usa uma abordagem do tipo *Local Adaptive Threshold*. Este tipo de abordagem é interessante pois usa limites diferentes para cada pixel para defini-lo como preto ou branco, baseado nos pixels adjacentes.

No caso deste trabalho, utiliza-se um método em que é analisado a vizinhança de cada pixel da imagem, e desta vizinhança é calculada a média dos valores destes pixels. A partir desta média, se obtém um *threshold*, ou um limite, para se decidir se o pixel será zero ou um, na imagem binária.

A fórmula que define o *threshold* para um dado pixel de uma imagem em tons de cinza é definida como:

$$T(x, y) = M(x, y) - C \tag{2.3}$$

onde M é a média dos pixels da vizinhança, e C é uma constante que pode ser escolhida conforme for conveniente para a aplicação.

2.2.3 Extração de Contorno

O próximo passo é extrair os contornos desta imagem binária. Este processo é feito pelo algoritmo de Suzuki e Abe (SUZUKI et al., 1985).

Este algoritmo tem o objetivo de encontrar os contornos de todos os objetos na imagem, e todos os contornos de buracos nas imagens.

O algoritmo se inicia escaneando a imagem, partindo da esquerda para direita, de cima para baixo, conforme a orientação padrão dos eixos x e y para imagens. Escaneia-se linha por linha, até se encontrar um pixel que seja um candidato ao contorno de um objeto, ou um buraco. Após encontrar um candidato, segue-se o contorno do objeto(ou buraco), marcando-se o contorno deste com um número identificador. Após fazer esta marcação, continua-se a procura por candidatos a bordas, até que se finde a imagem, e se encontre todos os contornos da imagem.

Para entender mais profundamente como são executados estes passos, serão apresentados os detalhes da execução destes passos. No primeiro passo, é buscado um pixel candidato a ser uma borda. Um pixel I(x,y) será considerado um candidato a borda de um objeto se ele cumprir os requisitos:

$$I(x, y) = 1 \tag{2.4}$$

$$I(x - 1, y) = 0 \tag{2.5}$$

No caso de um buraco, deve ser observado os requisitos:

$$I(x, y) \geq 1 \tag{2.6}$$

esta condição será $I(x, y) \geq 1$, e não $I(x, y) = 1$ porque conforme o algoritmo progride, números maiores que 1 aparecem, por causa da marcação do contorno. Por exemplo, a borda de um objeto pode ser marcada com '6', de forma que este '6' será maior do que 1, que satisfaz a condição.

$$I(x + 1, y) = 0 \tag{2.7}$$

Após encontrar um pixel que corresponde a um destes requisitos, se inicia o processo de seguir o contorno do objeto(ou buraco). Este processo é feito como em (ROSENFELD, 1970) . O candidato a borda é considerado como o início do contorno. A primeira coisa a se fazer é encontrar o final da borda, isso é feito buscando na vizinhança do primeiro pixel da borda um pixel '1'. Esta busca é ilustrada no passo 1 da figura 5 para o caso de um objeto, onde o primeiro pixel do contorno está circulado em vermelho. A busca começa do pixel circulado em azul, que da definição dada anteriormente para um

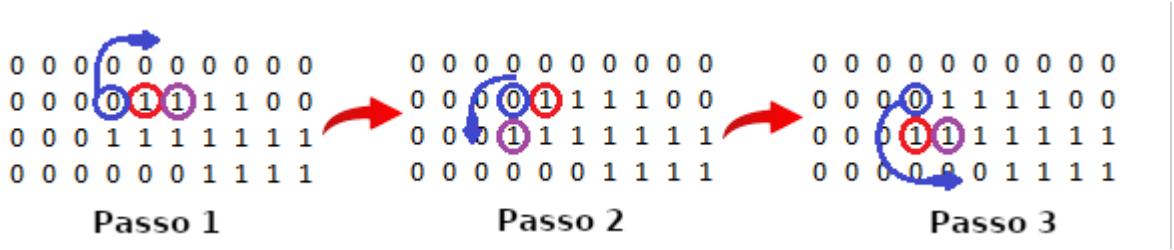

Figura 5 – Primeiro pixel do contorno em vermelho. Busca feita no sentido horário. Último pixel do contorno em roxo.

candidato a uma borda, sabe-se ser '0'. A busca segue no sentido horário, analisando os pixels da vizinhança do início do contorno até encontrar o primeiro '1', que na imagem está circulado em roxo. Terminado este processo, é obtido o último pixel de contorno. Para o caso de buracos, a lógica permanece, com a diferença de que a busca é iniciada no pixel a direita, em oposição ao caso de objetos, em que é iniciado a esquerda, isso porque pela forma que são definidos os buracos, o pixel a direita será sempre zero, para o primeiro pixel do contorno.

O resto da busca seguirá no sentido oposto ao usado para obter o último pixel do contorno,ou seja, a busca será feita no sentido anti-horário. O primeiro pixel a ser checado é o pixel do lado oposto do último pixel do contorno. No passo 2 da figura 5 este pixel é o pixel circulado em azul, que na vizinhança do pixel inicial, está do lado oposto do pixel obtido no passo anterior, que definimos como sendo o último pixel do contorno. A busca segue então até encontrarmos o próximo pixel do contorno, que na imagem está circulado em roxo.

O próximo pixel será buscado conforme o passo 3 da figura 5, seguindo a mesma lógica do passo anterior. Agora será analisado a vizinhança do segundo pixel do contorno. A busca por um pixel de valor '1', na vizinhança do pixel circulado em vermelho na imagem, deve agora se iniciar do pixel circulado em azul na figura. Olhando a vizinhança do pixel em vermelho, o pixel em azul está ao lado do pixel anterior do contorno, sendo o próximo pixel da vizinhança, seguindo no sentido anti-horário. Deve-se então, sempre iniciar a busca do pixel '1' na vizinhança do pixel do contorno analisado, a partir do próximo pixel da vizinhança depois do último pixel do contorno analisado. Esta regra é diferente para o primeiro pixel do contorno analisado, que sempre começa a busca pela esquerda quando se analisa objetos, e pela direita no caso de buracos.

Este passo será repetido até que o último pixel do contorno seja encontrado e o próximo pixel seja o primeiro pixel do contorno.

Em conjunto com o algoritmo para seguir o contorno, é realizada a marcação do contorno. Cada contorno tem um número que o identifica, e ao seguir o contorno de um

objeto ou buraco, cada elemento do contorno é substituído com o número correspondente ao número do contorno. Por exemplo, para o caso estudado para o seguimento do contorno, obteríamos o resultado:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 2 & -2 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 & 1 & 1 & 2 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Pode-se observar que em alguns pixels o valor é negativo, isso acontece por causa da política de marcação da borda usada. Para entender esta política, considere um pixel x_n, com uma vizinhança composto pelos pixels $y_0, y_1, ..., y_8$

$$\begin{matrix} y_3 & y_2 & y_1 \\ y_4 & x_n & y_0 \\ y_5 & y_6 & y_7 \end{matrix}$$

Seja x_n um pixel da borda, y_m o pixel anterior x_{n-1} da borda, e seja y_k o próximo pixel da borda x_{n+1}. Então

$$x_n = \begin{cases} -NBD & \text{se } (k-1) \mod 8 < m \\ NBD & \text{se } (k-1) \mod 8 \geq m, \text{ e } x_n = 1 \\ x_n & \text{caso contrário} \end{cases} \tag{2.8}$$

Sendo que NBD é o número da borda.

2.2.4 Aproximação Poligonal

Utilizando o algoritmo de Douglas-Peucker (DOUGLAS; PEUCKER, 1973), cada um dos contornos fechados da imagem são aproximados para polígonos. Estes polígonos por sua vez serão representados pelos seus vértices, que são pontos do contorno do objeto.

Este algoritmo funciona de modo a aproximar uma curva por meio de linhas retas. O quanto essas linhas se aproximam da curva original, depende da variável ϵ, que irá ditar o quanto essas linhas podem se distanciar da curva que está sendo aproximada.

Para entender o funcionamento do algoritmo, um exemplo será mostrado a seguir, descrevendo o procedimento de uma aproximação poligonal, dados pontos quaisquer de uma curva para ser aproximada. Para este exemplo, seja:

$$\epsilon = 1 \tag{2.9}$$

Na Figura 6, é apresentada a curva a ser aproximada. O primeiro passo é apresentado no passo 1 da Figura 6, traça-se uma linha entre o primeiro e o último ponto da

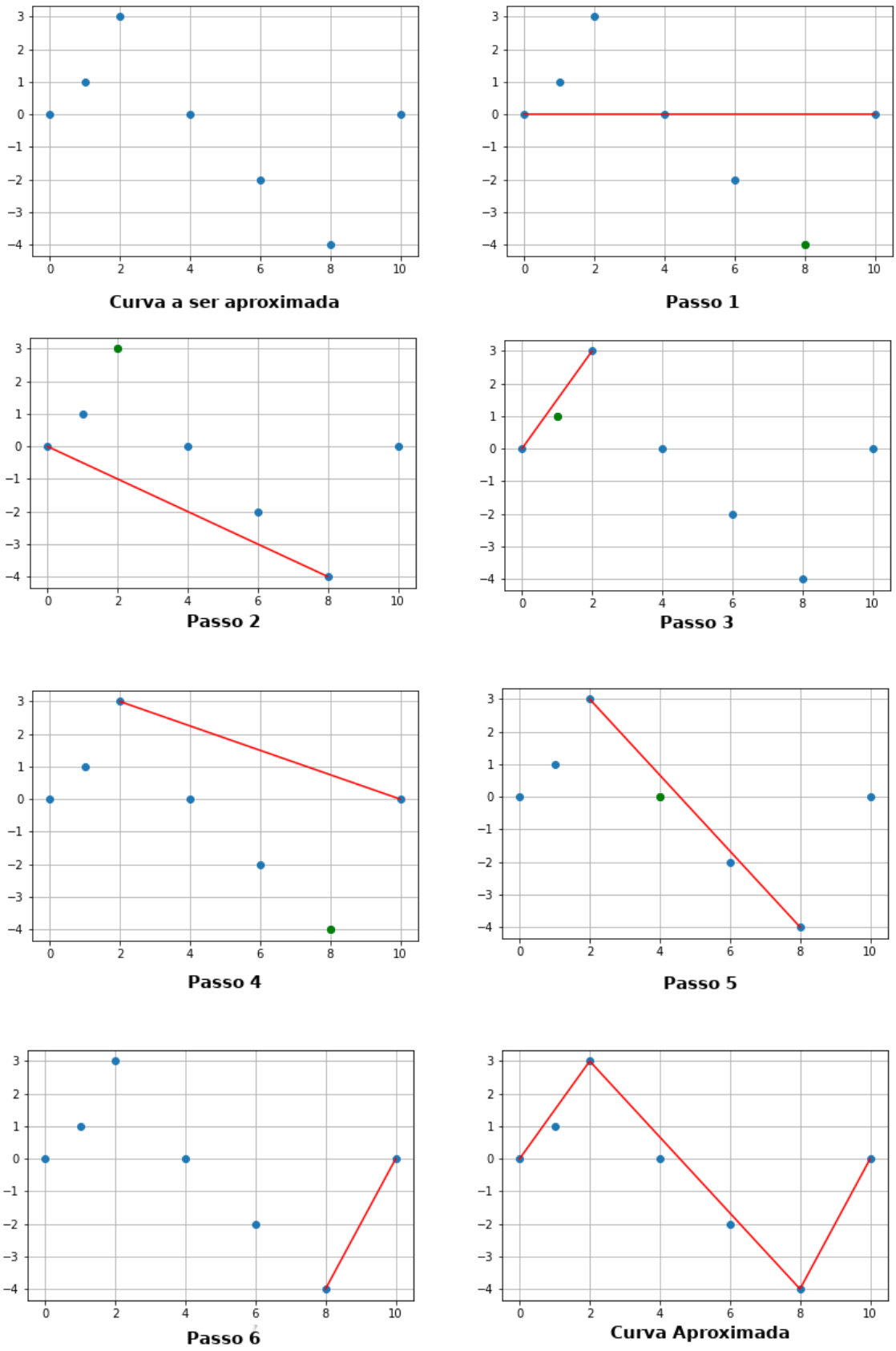

Figura 6 – Exemplo de execução do algoritmo de aproximação poligonal de Douglas-Peucker.

curva, e então analisa-se a distância de cada um dos pontos da curva em relação a esta linha traçada. A menor distância entre um ponto (x_0, y_0) e uma linha definida por dois pontos $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ pode ser obtida pela equação:

$$dist(P_1, P_2, (x_0, y_0)) = \frac{|(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \tag{2.10}$$

O ponto mais distante da linha está na cor verde. Compara-se a distância deste ponto da linha traçada com o ϵ. A distância entre este ponto e a linha é de 4 unidades, e $\epsilon = 1$. Isso significa que a distância entre o ponto e a linha é maior do que o valor de ϵ, que implica que para o valor de ϵ que escolhido, esta linha não é uma aproximação boa o suficiente.

Segue-se para o próximo passo, como mostrado no passo 2 da Figura 6. Traça-se agora uma linha entre o primeiro ponto da curva e o ponto obtido no passo anterior. Repete-se o mesmo procedimento de analisar a distâncias dos pontos da curva em relação a linha traçada, analisando sempre os pontos entre o início e o fim da linha. O ponto de maior distância é o verde, e novamente seu valor ultrapassa ϵ.

O passo 3 é apresentado na Figura 6. A linha é traçada entre o ponto inicial e o ponto obtido no passo anterior. O único ponto entre as extremidades da linha está em verde, e sua distância é menor que o valor de ϵ. Isso significa dizer que o segmento de reta obtido neste passo é uma aproximação boa o suficiente para os três primeiros pontos da curva. Assim, o ponto final deste segmento de reta é guardado como sendo um dos pontos a serem utilizados na aproximação.

No próximo passo, o passo 4, ilustrado na Figura 6, é traçada uma reta entre o ponto obtido no passo anterior, e o ponto final da curva. O ponto em verde é o ponto mais distante da linha traçada e sua distância ultrapassa ϵ. Deste modo, segue-se o mesmo procedimento dos passos anteriores, passando para o próximo passo e utilizando este ponto obtido como final da linha a ser traçada.

Na Figura 6 no passo 5. Nesta figura é observado que o ponto que tem maior distância da linha (em verde) tem uma distância abaixo de ϵ. Deste modo, essa linha obtida é uma boa aproximação para o trecho da curva entre o terceiro e o sexto ponto da curva.

No último passo(Figura 6) traça-se a linha entre o ponto final da linha obtida no passo anterior e o ponto final da curva que se quer aproximar. Não há mais nenhum ponto entre as extremidades desta linha, de modo que esta linha é uma boa aproximação entre o último e o penúltimo ponto da curva, já que esta linha descreve perfeitamente estes pontos.

O resultado final é ilustrado na Figura 6. Em suma este algoritmo obtém a aproxi-

mação de uma curva traçando-se retas entre os pontos da curva e analisando o ponto que mais se distancia desta reta. Se a distância deste ponto for maior que o ϵ escolhido, traça-se uma nova reta tendo como ponto final este ponto, caso contrário, esta reta é guardada, e traça-se uma nova reta partindo do fim da reta obtida e terminando no último ponto da curva que se quer aproximar. Assim que se obter uma reta que acabe no último ponto da curva, o algoritmo termina a execução, tendo obtido todos os pontos necessários para aproximar a curva com um erro máximo de magnitude ϵ.

2.2.5 Matriz Homográfica

Após a aproximação poligonal, devem ser encontrados candidatos a marcadores fiduciais, porém primeiro deve ser retirada a projeção em perspectiva das imagens. Isto é feito a partir da estimação dos parâmetros da matriz homográfica, possibilitando retirar a projeção em perspectiva deste retângulo, obtendo um quadrado plano sem influência da projeção em perspectiva.

2.2.6 Limiar de Otsu

Com o quadrado sem perspectiva busca-se então obter uma imagem binária para obter uma matriz com cada bit do marcador. Esta imagem é obtida utilizando-se o método de Otsu (OTSU, 1979), que fornece um valor de limiar ótimo para o caso da imagem ser bimodal, que é o caso para os marcadores fiduciais usados pelo algoritmo ArUco.

2.2.7 Identificação do marcador

Após estes passos, compara-se o conteúdo do marcador fiducial identificado com o dicionário utilizado. O conteúdo do marcador é representado por um número inteiro, que é obtido concatenando cada um dos bits da imagem. São gerados quatro números, um para cada rotação possível do marcador. Por sua vez, o dicionário é representado por uma árvore binária balanceada, para otimizar a busca. Pode ser deduzido que este processo tem complexidade logarítmica $O(4log_2(|D|))$, onde D é o tamanho do dicionário de símbolos, e o fator 4 se deve as quatro rotações possíveis do marcador.

Caso não seja encontrado um marcador correspondente no dicionário, é aplicado um método de correção. Considerando que a mínima distância entre dois marcadores em um dicionário é de d bits, então um erro de no máximo $\frac{d-1}{2}$ pode ser corrigido. Checa-se então o marcador com o dicionário e se for encontrado um elemento do dicionário com uma distância menor que $\frac{d-1}{2}$, então o marcador pode ser identificado como tal elemento.

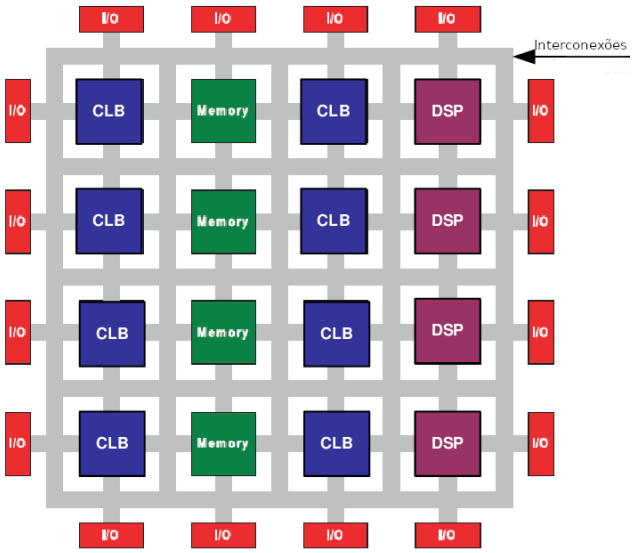

Figura 7 – Estrutura Básica de uma FPGA

2.3 Hardware Reconfigurável

Diversas abordagens podem ser utilizadas ao se desenvolver circuitos com circuitos integrados. Um tipo de abordagem é utilizar circuitos integrados com uma lógica fixa, por exemplo portas lógicas, flip-flops ou registradores. Outra abordagem é utilizar microprocessadores ou microcontroladores, de forma a tornar possível o desenvolvimento de um software personalizado para a aplicação de interesse, e assim é possível utilizar um mesmo microcontrolador, ou um microprocessador, para uma ampla gama de aplicações.

Uma outra abordagem porém, é a utilização de uma FPGA (Field Programmable Gate Array). Uma FPGA se diferencia de circuitos integrados como os discutidos anteriormente por ser um hardware reconfigurável. Diferente de microprocessadores, que permitem uma programação por software, uma FPGA permite uma reconfiguração do hardware, de forma que é possível rearranjar as conexões internas da FPGA para se executar uma lógica personalizada. Para entender este conceito é importante entender antes a estrutura de uma FPGA.

A estrutura básica de uma FPGA é representada na Figura 7. Esta estrutura é basicamente uma matriz de blocos lógicos, esses blocos na imagem são representados por quadrados azuis, e tem a capacidade de realizar operações lógicas simples, em geral este bloco se comporta como uma tabela verdade. Cada um destes blocos é cercado por canais de roteamento. Estes canais são configuráveis eletricamente, de modo que pode-se organizá-los para conectar os blocos lógicos a fim de obter uma lógica personalizada. Em volta desta matriz, temos um barramento de entrada e saída, que permite a interação da FPGA com outros dispositivos.

Além desses componentes mencionados, há mais dois componentes principais que

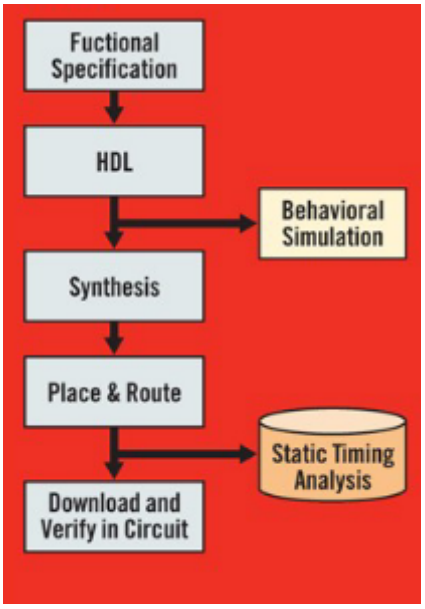

Figura 8 – Fluxo de Projeto para FPGA

compõem uma FPGA. O primeiro é a memória, que armazena a configuração de cada um dos canais de roteamento, que é chamado de *bitstream* e representa a interconexão entre os blocos da FPGA. O segundo componente, são blocos de circuitos especiais, por exemplo DSPs, RAM, PLLs e etc. Estes blocos também se interconectam com os blocos lógicos e permitem o desenvolvimento de lógicas complexas.

Há casos em que além desses blocos mencionados, temos também a presença de um processador junto a esses blocos. A esse tipo de sistema dá-se o nome de System-on-Chip (SoC), que é um sistema que agrega um sistema inteiro em um chip, com memória, processador, blocos especiais de processamento e em alguns casos blocos de rádio frequência, e adicionalmente uma lógica dedicada, específica da aplicação.

Para desenvolver sistemas em FPGA, o fluxo padrão de desenvolvimento é como na figura 8. Inicia-se o projeto fazendo uma descrição funcional do circuito que se quer desenvolver. Esta descrição deve ser focada nas funcionalidades que o bloco a ser desenvolvido irá ter. A partir desta especificação, descreve-se o circuito por meio de uma linguagem de descrição de hardware(ou HDL - Hardware Description Language). A partir desta descrição é possível fazer simulações, para aferir se o comportamento descrito na linguagem, é o comportamento esperado, dado as especificações funcionais. Se a simulação for bem sucedida, a próxima etapa será a síntese. Nesta etapa, através de softwares de desenvolvimento, é analisado a descrição em HDL, e a partir dela produz-se a uma descrição do sistema a nível de portas lógicas. A partir disto, a próxima etapa é a de *Place & Route*, onde será feito, também por software, o posicionamento e a interconexão da lógica do sistema na FPGA. Neste passo é possível fazer uma análise de tempo, para verificar se a lógica proposta para a FPGA escolhida cumpre as restrições de tempo necessárias.

E finalmente, se os testes tiverem resultados satisfatórios, pode-se produzir o arquivo de *bitstream* que pode ser baixado na FPGA, para fazer a verificação do circuito.

2.4 High-Level Synthesis

Com o crescente aumento na complexidade dos circuitos integrados, é necessário tornar o desenvolvimento de hardware cada vez mais eficiente, para que continue sendo viável o desenvolvimento desses sistemas. Para que isso aconteça, novas ferramentas são criadas, para maximizar a produtividade dos engenheiros envolvidos no desenvolvimento de hardware.

No fluxograma da figura 8, é gerado um código HDL a partir da especificação funcional do sistema. Uma forma de fazer isso é escrevendo o código HDL diretamente a partir das especificações, isto é, um engenheiro irá escrever diretamente o código, analisando as especificações. Estas especificações podem ser feitas de diversas formas, podendo-se utilizar um simples documento com uma descrição textual do que se deve implementar, até a produção de um modelo em software para descrever o comportamento do bloco de hardware que se quer desenvolver.

A partir do código HDL, as ferramentas de software conseguem sintetizar todo o circuito descrito, e implementar as conexões entre as células padrão da FPGA, ou mesmo produzir todo o leiaute de um circuito integrado do tipo ASIC (Application Specific Integrated Circuit). Isto permite a automação de grande parte dos processos envolvidos no desenvolvimento de circuitos integrados, e tem sido durante muitos anos a principal forma de se desenvolver tais sistemas.

Uma alternativa a esse fluxo de projeto, é utilizar ferramentas baseadas em HLS (High-Level Synthesis). Nesse tipo de fluxo de projeto, trabalha-se diretamente no nível mais alto de abstração de um projeto de circuito integrado, que na figura 8 corresponde a especificação funcional. Com o HLS é possível utilizar códigos em linguagens como SystemC ou ANSI C/C++, e gerar a arquitetura RTL, em HDL, de forma automática.

Esse tipo de abordagem tem a vantagem de acelerar o desenvolvimento de soluções em hardware, permitindo o desenvolvimento de estruturas complexas, de forma automatizada e rápida.

2.5 Protocolo AXI

Quando desenvolve-se uma arquitetura em hardware, uma forma de se lidar com a complexidade do sistema, é subdividir a arquitetura em blocos. Surge com isso, os chamados hardware IP's (Intellectual Property), que são blocos reutilizáveis de código HDL que podem ser usados na arquitetura em hardware. Para que seja possível utilizar

Figura 9 – Tipos de componentes AXI. Imagem de (LIMITED, 2021)

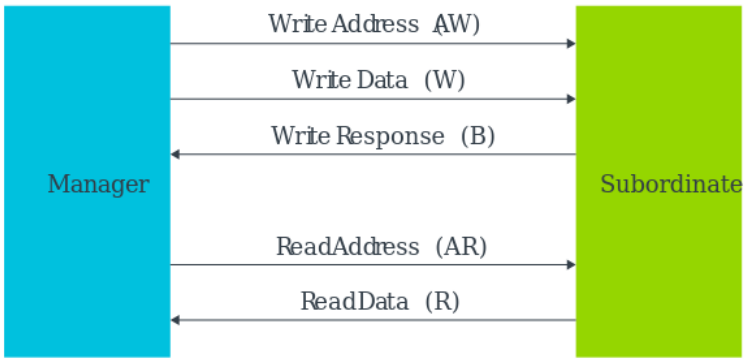

Figura 10 – Portas da interface AXI. Imagem de (LIMITED, 2021)

reutilizar estes IP's de forma genérica, é importante que haja uma interface padrão entre os componentes da arquitetura, para que se tenha uma arquitetura escalonável. É nesse contexto que um protocolo como o AXI se torna importante.

O protocolo AXI(Advanced eXtensible Interface) é uma especificação que padroniza as interfaces de um bloco IP. O AXI é parte do AMBA(Advanced Microcontroller Bus Architecture) que é um conjunto de padrões da ARM, que procura padronizar a comunicação entre IP's dentro de um circuito integrado.

Há dois tipos de interface AXI, o "subordinado"e o "administrador". O subordinado tem sempre que se conectar com um administrador e vice-versa, isso pode ser feito de forma direta, ou por meio de um bloco de conexão entre componentes AXI, o *AXI Interconnect*.

A figura 9 ilustra essa relação.

Na figura 10 é possível ver um diagrama simplificado com os sinais que um componente AXI manda para outro. Uma operação de escrita acontece da seguinte forma: O administrador(Manager) envia o endereço do registrador no qual quer escrever, através do sinal AW, e envia o dado que se quer escrever através do sinal W, após a operação o subordinado envia um sinal em B para reportar se a operação foi bem sucedida. Um canal de leitura pode ser observado na parte inferior da imagem, e funciona de maneira análoga ao canal de escrita, mas sem o sinal de resposta B.

Há também uma variação da interface AXI que é o AXI Stream. Esta variação não utiliza os sinais de endereço. Em vez disso usa um sinal "Ready"e um "Valid", que reportam respectivamente se o subordinado está pronto para receber dados e o outro diz se o administrador está enviando um dado válido. Esta variação é mais utilizada quando se precisa lidar com grandes volumes de informação, por exemplo em processamento de imagens, em que se precisa receber todos os pixels de uma imagem.

2.6 Estado da arte

Nesta seção será levantado o estado da arte para sistemas de localização utilizando visão computacional, com foco especial em sistemas que utilizam marcadores ArUco. Serão analisadas as plataformas utilizadas, os métodos utilizados, aplicações e também serão analisados trabalhos que utilizam aceleração por FPGA de algoritmos similares.

No trabalho (BURRELL et al., 2018) é proposto um sistema com um robô que corta canos no descomissionamento de uma usina nuclear. Para tanto, utiliza-se um robô com um kinect(Sensor de movimentos para Xbox 360, que utiliza câmera e sensor infravermelho para identificar os movimentos), que produz uma visão stereo para o robô, e utiliza-se um drone com uma câmera para complementar a visão do robô. Estes dispositivos são controlados remotamente por um notebook e um computador, e a partir destes dispositivos, conseguiu-se adquirir medidas de distância calculados a partir das imagens. O processamento das imagens utiliza marcadores ArUco e o SLAM(Simultaneous Localization And Mapping) para fazer estas medidas.

Em (MARUT; WOJTOWICZ; FALKOWSKI, 2019), é estudada a aplicação de marcadores ArUco em Veículos Aéreos Não Tripulados (VANT). O objetivo deste trabalho é desenvolver um sistema de pouso automático que utiliza marcadores ArUco como referência para se mensurar altura e distância do VANT estudado do lugar que se deseja pousar. Para implementar esse sistema utilizou-se um microcomputador de modelo ODROID-XU4 que contém um SoC Samsung Exynos 5422, que conta com uma CPU Cortex™-A15 2Ghz, uma CPU Cortex™-A7 Octa core e uma GPU Mali Mali-T628 MP6, além de contar com uma memória DRAM de 2 GigaBytes. Com este hardware, foi possível

atingir uma taxa de atualização de 10 quadros por segundo.

Um outro trabalho usando drones (SANI; KARIMIAN, 2017), buscou estudar o pouso automático de drones, utilizando marcadores ArUco. Este porém, utiliza os marcadores ArUco em conjunto com sensores inerciais (IMU) para conseguir realizar o procedimento de pouso de forma autônoma. Este trabalho utilizou um notebook DELL Studio 1558 para realizar o processamento, que conta com um processador Core i-7 Q720 e 6 GB de RAM.

A partir destes trabalhos é possível notar que é necessário um alto poder computacional para obter um desempenho razoável utilizando o algoritmo da biblioteca ArUco. No trabalho (MARUT; WOJTOWICZ; FALKOWSKI, 2019), citado anteriormente, chega-se a conclusão de que para aumentar o desempenho da aplicação seria necessário aumentar a resolução da câmera, e aliado a isso, seria necessário um hardware com maior desempenho. Uma forma de promover a aceleração destes algoritmos, como comentado em seções anteriores, é a utilização de uma FPGA.

Tendo isto em mente, em (GHORBEL et al., 2013), é proposto um sistema de localização de robô, com uma câmera instalada no topo de uma arena de testes. Este sistema é acelerado utilizando uma FPGA Xilinx Virtex 5ML507, e o algoritmo implementado se baseia na utilização de estágios de limiarização da imagem, erosão e o uso da transformada de Hough. Comparativamente, o mesmo algoritmo executado com o MATLAB, teve uma redução de 85% no tempo de execução, graças a otimizações feitas em hardware reconfigurável.

Outra aplicação de FPGA pode ser observada em (TIAN; WU; WANG, 2015), que propõe uma arquitetura de hardware, para operar uma imagem com um filtro Sobel. Esse filtro é um tipo de operador derivativo de primeira ordem. A arquitetura em hardware implementada utiliza uma FPGA Cyclone IV GX series, memórias DDR2, um sensor CMOS para obter a imagem, e um conector VGA para transmitir a imagem operada. Como resultado, esta arquitetura obteve uma performance em tempo real, reproduzindo as imagens operadas de resolução 640*480 no conector VGA, com uma velocidade de 60 quadros por segundo.

No trabalho de (JIN; CHO; JEON, 2006) é proposto um sistema de foco automático de câmeras. Esse sistema é proposto utilizando um algoritmo de *Adaptive Threshold*, que é utilizado também em um dos estágios do algoritmo ArUco. O sistema proposto se utiliza de uma FPGA para implementar o algoritmo, e a partir dele possibilita o ajuste automático do foco de uma câmera SonyXC-55. Este trabalho traz um resultado interessante, pois a partir dele observa-se ser viável a implementação em hardware reconfigurável de um dos estágios do algoritmo ArUco. Implementação essa que, em potencial, pode gerar uma aceleração do algoritmo ArUco.

A tabela 1 resume e complementa a discussão desta seção.

Tabela 1 – Tabela com resumo do estado da arte para sistemas de localização de robôs usando marcadores ArUco e implementações em FPGA de algoritmos de visão computacional.

Título	**Autor**	**Ano**	**Técnica**	**Plataforma Computacional**	**Principais Resultados**
Towards a cooperative robotic system for autonomous pipe cutting in nuclear decommissioning	Burrell, Thomas e West, Craig e Monk, Stephens D e Montezeri, Allahyar e Taylor, C James	2018	ArUco + SLAM + Visão Stereo	Notebook + Computador	Obteve medidas de distância com um erro médio de 225mm
ArUco markers pose estimation in UAV landing aid system	Marut, Adam e Wojtowicz, Konrad e Falkowski, Krzysztof	2019	ArUco	ODROID-XU4, Samsung Exynos 5422	Obteve medidas com erro de distância de menos de 10%. Obteve uma taxa de atualização de 10 quadros por segundo.
Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors	Sani, Mohammad Fattahi e Karimian, Ghader	2017	ArUco + IMU	DELL Studio 1558, processador Core i7 Q720	Conseguiu realizar o pouso automático de um drone, com um erro de 6 cm do lugar de pouso.
An FPGA based platform for real time robot localization	Ghorbel, Agnès and Jallouli, Mohamed and Amor, Nader Ben and Amouri, Lobna	2013	Transformada de Hough	FPGA Xilinx Virtex 5ML507	Obteve um erro de trajetória da navegação na ordem de 20 mm. E obteve uma redução de 85% no tempo de execução do algoritmo.
Realization of real-time sobel adaptive threshold edge detection system based on FPGA	Tian, Jie e Wu, Jingjing e Wang, Guanglong	2015	Implementação de um filtro Sobel	FPGA Cyclone IV GX series	Obteve o resultado com uma taxa de atualização da imagem de 60 quadros por segundo, com resolução de 640*480px.
FPGA based passive auto focus system using adaptive thresholding	Jin, Seung Hun e Cho, Jung Uk e Jeon, Jae Wook	2006	Adaptive Threshold	Xilinx xc4vlx200-ffl51	Implementa um sistema de foco automático para câmeras, obtendo uma melhor performance usando cenas bem iluminadas.
Sistema de Localização de Múltiplos Robôs de Pequeno Porte Baseado em Visão Computacional	Hércules I. A. Santos, Daniel M. Muñoz	2021	ArUco, Segmentação por Cores	Notebook i5-8265u, PYNQ-Z2	Compara algoritmos para a localização de robôs, obtendo o ArUco como solução mais viável.

3 Metodologia

Neste capítulo serão apresentados os métodos utilizados para o levantamento da base de dados, será definida a plataforma utilizada para os testes e serão apresentados os critérios adotados para medir o desempenho da arquitetura implementada.

3.1 Arquitetura em hardware proposta

Para acelerar o algoritmo ArUco, é proposta uma arquitetura em hardware conforme a Figura 11. É proposta uma implementação em hardware dos algoritmos de *Adaptative Threshold*, extração de contorno de Suzuki e a aproximação poligonal de Douglas-Peucker. Em resumo, a imagem, já em tons de cinza, é transferida do processador para a FPGA, e na FPGA é feito o processamento da imagem passando por todos os 3 estágios citados anteriormente, e no final do processamento é adquirido os contornos dos objetos da imagem já otimizados pelo algoritmo de aproximação poligonal.

Para que isso seja possível, é necessário utilizar interfaces AXI entre o bloco de processamento PS(*Processing System*) e o bloco de lógica progamável PL(*Programmable Logic*), isto porque o SoC ZYNQ utiliza o padrão AMBA de interconexões. Deste modo, 2

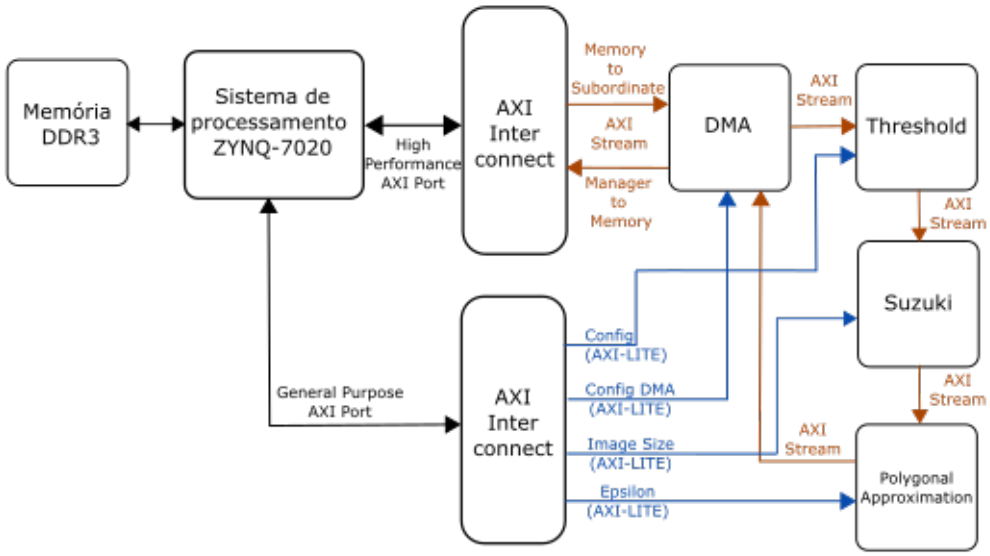

Figura 11 – Arquitetura proposta

blocos de interconexão AXI são utilizados, um para acessar as portas de alta performance (*High-Performance AXI Port*) e um para acessar as portas de uso geral(*General Purpose AXI Port*). As portas de alta performance dão acesso a memória DRAM, enquanto as portas de uso geral dão acesso a portas seriais, de propósito geral, e permitem o controle do bloco de processamento.

O DMA(*Direct Memory Access*) é um bloco que permite que outros componentes que não o processador tenham acesso direto a memória. Isso permite que o bloco de processamento realize outras tarefas enquanto espera o resultado do processamento dos blocos da lógica reconfigurável.

Este bloco DMA se conecta com a porta de alta performance AXI, que dá acesso a memória. Com isso o bloco DMA se habilita a fazer leituras e escritas na memória, por meio de interfaces AXI-Stream. Este bloco DMA é controlado pelo sistema de processamento por meio de uma porta AXI-Lite que configura o DMA. Deste bloco saem os pixels da imagem a ser processada para o bloco de Threshold, via AXI-Stream.

O bloco de Threshold tem uma interface AXI-Lite de configuração, que permite configurar o tamanho da janela usada na operação de binarização da imagem, a constante C que é subtraída conforme a equação 2.3, e também é utilizado um pino para configurar a dimensão da imagem.

Do bloco de Threshold a imagem passa para o bloco de extração de contorno Suzuki, também por meio da interface AXI-Stream. Este bloco tem também uma interface AXI-Lite para que seja configurada a dimensão utilizada da imagem.

Do bloco Suzuki saem os contornos da imagem, e através de mais uma interface AXI-Stream, o contorno é aproximado no bloco de aproximação poligonal. Este bloco tem também uma interface de configuração para configurar o valor ϵ do algoritmo de aproximação poligonal, que controla a precisão da aproximação. Por fim, do bloco de aproximação poligonal saem os contornos aproximados, que vão para o DMA, que por sua vez enviará esses dados para a memória, de modo que fique disponível para o sistema de processamento prosseguir com os próximos passos.

Os 3 blocos de processamento do ArUco, foram todos desenvolvidos em HLS, utilizando a linguagem C++. Por meio da descrição comportamental em C++ o software é capaz de inferir uma arquitetura em hardware capaz de implementar este algoritmo descrito em C++. Neste código foram utilizados os recursos mais simples do HLS, não sendo explorado o paralelismo e pipelines.

Figura 12 – Imagem da base de dados com marcadores ArUco com iluminação solar.

Figura 13 – Imagem da base de dados com marcadores ArUco em uma iluminação artificial.

3.2 Criação da base dados

Para testar os algoritmos, foi levantada uma base de dados, contendo fotos de caixas, simulando o formato dos robôs. As fotos foram obtidas com uma resolução de 4160 × 3120 pixels no centro de uma arena de testes de 3 × 3 metros a uma altura constante de 3 metros.

Cada caixa contém marcadores que serão rastreados pelo algoritmo. Para ser rastreado pelo ArUco, foram utilizados os marcadores fiduciais disponíveis em um dicionário chamado *aruco_mip_36h12_dict* (GARRIDO-JURADO et al., 2016), que é um dicionário desenvolvido especialmente para ser identificado pelo ArUco, alcançando maior eficiência do algoritmo. Exemplos de fotos da base de dados podem ser vistas nas figuras 12 a 14.

Esta base de dados foi criada com quatro tipo de iluminações diferentes em um cenário cujo chão é cimentado, e mais dois tipos de iluminações no Laboratório de Controle, da Faculdade UnB Gama, o qual tem um piso de porcelana branco. Para o primeiro

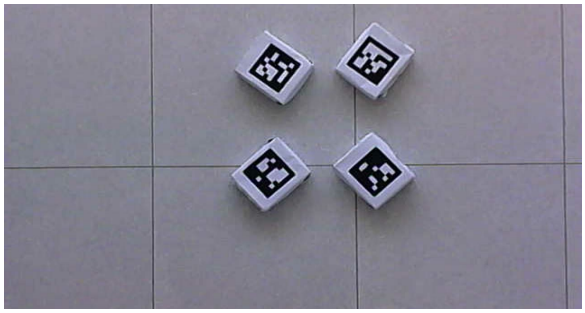

Figura 14 – Imagem da base de dados com marcadores ArUco no laboratório.

ambiente, com o chão cimentado, foram usadas os quatro tipos de iluminação: a) luz solar; b) luz artificial com incidência central; c) luz artificial com intensidade média e incidência lateral direita e; d) luz artificial de baixa intensidade com incidência lateral esquerda. Para o ambiente no laboratório, duas outras iluminações foram testadas, sendo a primeira uma iluminação utilizando luz natural e lâmpadas ao centro da arena, e uma segunda com luz natural e lâmpadas laterais à arena. Além das diferentes iluminações, foram coletadas fotos com diferentes configurações de robôs. Foram coletadas amostras com apenas um robô, alterando os símbolos que cada um carrega. Também foram obtidas amostras com quatro robôs, tendo cada robô sua própria identificação.

A base de dados reúne 222 imagens de robôs carregando marcadores ArUco, e pode ser acessada em (SANTOS, 2021b).

O algoritmo ArUco foi testado para cada uma das imagens contidas na base de dados, verificando se o algoritmo detecta corretamente os robôs e a posição estimada na imagem. Para validar a posição dos robôs é necessário ter uma medida de referência. Esta medida foi realizada observando-se as imagens coletadas, e apontando com o ponteiro do mouse para os cantos dos quadrados dos marcadores ArUco.

Uma observação a ser feita, é que para este trabalho não será calibrada a câmera. De modo que, se for observada a necessidade de calibração da câmera, em trabalhos futuros será feita esta análise, para obter um ganho de desempenho na localização dos robôs. Isto porque a câmera utilizada não tem uma distorção significativa de bordas, por causa das tecnologias mais atuais de desenvolvimento de câmeras.

3.3 Plataforma de desenvolvimento

Para este trabalho foi utilizado a plataforma de desenvolvimento PYNQ-Z2 (Figura 15). Esta plataforma é baseado no chip ZYNQ 7020 da Xilinx, e inclui uma memória RAM de 512 MB, 2 portas HDMI, 1 USB, 1 porta Ethernet, 2 entradas de áudio P2, uma série de portas de entrada e saída de uso geral, botões e uma entrada para cartão SD.

A Figura 16 ilustra o funcionamento do SoC. Este chip conta com um ARM Cortex-

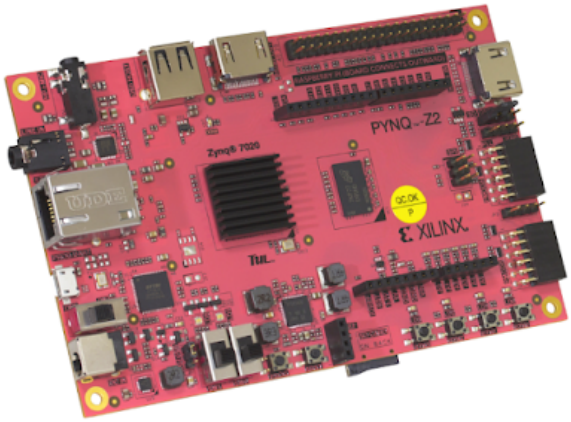

Figura 15 – Placa de desenvolvimento PYNQ-Z2.

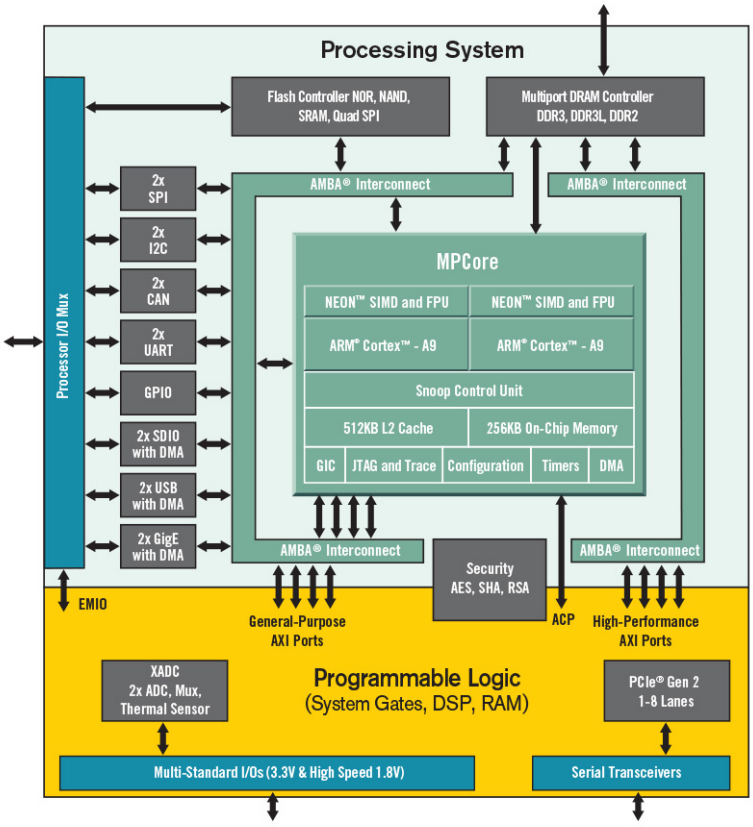

Figura 16 – Diagrama de blocos dos SoC's da série ZYNQ-7000.

A9 Dual-Core. Além deste processador, tem-se no mesmo chip uma FPGA Artix-7. A comunicação entre o processador e a FPGA se dá por um bloco de interconexão AMBA (AMBA Interconnect), pelo qual blocos com interface AXI podem se comunicar. Há 2 blocos de interconexão no diagrama, um à direita que dá acesso a memória RAM externa e um à esquerda que dá acesso a portas de uso geral, como portas GPIO, UART e SPI. Há também uma porta ACP entre a FPGA e o processador ARM, que permite o acesso da FPGA ao SCU(Snoop Control Unit) que controla a memória cache do processador e a memória *On-Chip*.

A PYNQ em geral é usada com um sistema operacional. Uma distribuição Linux baseada no Ubuntu é gravada no cartão de memória, e a partir dela é possível acessar as funcionalidades da placa. Um servidor é iniciado pela PYNQ, e é possível acessá-la via rede. O sistema operacional da PYNQ é um ambiente de desenvolvimento que permite o uso da linguagem Python para reprogramar a FPGA em tempo de execução e utilizar arquiteturas em hardware desenvolvidas com antecedência para implementar uma diversidade de funcionalidades.

3.4 Critérios de desempenho

Para avaliar a eficácia do ArUco como técnica para localizar robôs foram adotadas as seguintes métricas: a) porcentagem de detecção, que indica quantos acertos teve cada método de detecção de objetos com relação ao total; b) porcentagem de falsos positivos (FP) em relação ao número total de objetos a serem detectados na base de dados, indicando quantos objetos foram detectados em locais onde não tem um marcador fiducial, neste caso um FP=200% significaria que o algoritmo gerou duas vezes mais falsos positivos que o total de objetos presentes na base de dados; c) erro médio de posição; d) erro médio de orientação; e) tempo de execução médio usando duas plataformas computacionais, um laptop com processador Intel Core i5-8265U, 8GB RAM, 1.6 GHz, e uma plataforma de desenvolvimento PYNQ-Z2 com processador ARM Dual-core Cortex A9, 512 MB DDR3 RAM e 650 MHz. É importante notar que, para o tempo de execução, é necessário um tempo de processamento que permita a localização dos robôs em tempo real, de modo que uma taxa de processamento de aproximadamente 20 quadros por segundo seria um valor mínimo de tempo adequado.

3.5 Teste da arquitetura

Para avaliar a arquitetura em hardware proposta, foi testado cada um dos blocos individualmente, usando o DMA para acessar apenas um dos blocos do ArUco, por exemplo, para o threshold a arquitetura de teste é como na Figura 17.

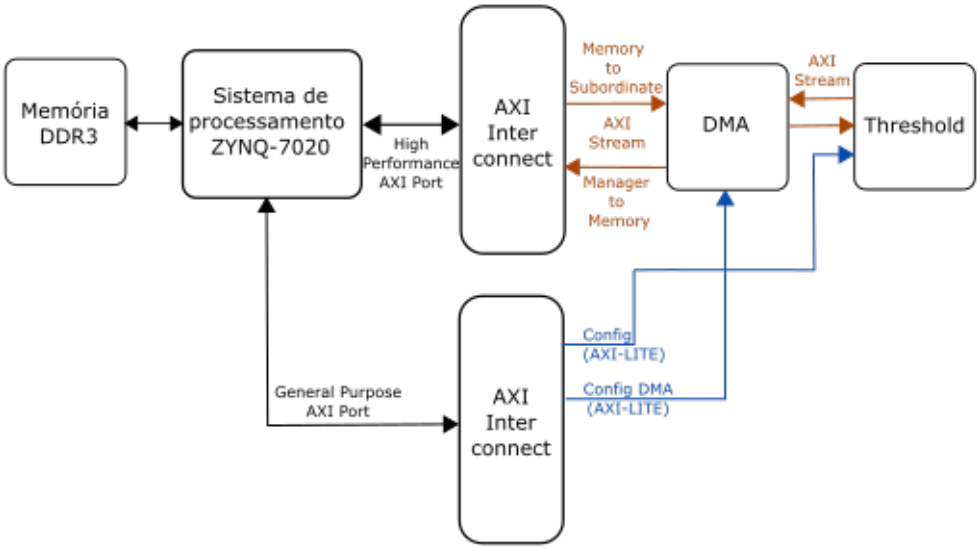

Figura 17 – Arquitetura para teste do threshold

Para cada um dos blocos foi medido o tempo de execução da solução em hardware, e comparou-se com o tempo de execução para a solução em software. Os testes foram feitos utilizando as imagens da base de dados, de modo que o teste foi feito em todas as imagens com marcadores ArUco da base.

4 Resultados Obtidos

Neste capítulo serão discutidos os resultados dos testes do algoritmo ArUco para a detecção de robôs móveis, analisando a performance do algoritmo para a base de dados.

4.1 Desempenho do Algoritmo ArUco

Como primeiro teste foi feita a análise de performance do algoritmo ArUco para a detecção de robôs móveis usando um computador de escritório. A partir das fotos da base de dados o algoritmo ArUco foi testado segundo as métricas de porcentagem de detecções, porcentagem de falsos positivos, erro médio de posição e erro médio de orientação

Na Tabela 2 é possível analisar os resultados das detecções de marcadores ArUco para as imagens da base de dados, separados pelo local em que foram testados, e o ID do marcador ArUco no dicionário. É possível ver que para quase todos os casos, a detecção foi de 100% do algoritmo, mostrando a robustez desta técnica.

A Tabela 3 mostra a porcentagem de falsos positivos detectados pelo ArUco. O resultado mostra que o ArUco não detectou nenhum falso positivo.

A Tabela 4 mostra o erro médio de posição do algoritmo testado. Mostrando resultados aceitáveis para aplicação de uma localização de robôs. As marcações em vermelho e verde representam respectivamente o valor mínimo e máximo de erro médio.

Tabela 2 – Porcentagem de detecções

	#0	**#1**	**#2**	**#3**
Chão cimentado(Luz solar)	96.88%	96.88%	93.75%	100.0%
Chão cimentado(Luz à esquerda)	100.0%	100.0%	100.0%	100.0%
Chão cimentado(Luz ao centro)	100.0%	100.0%	100.0%	100.0%
Chão cimentado(Luz à direita)	100.0%	100.0%	100.0%	100.0%
Chão de porcelana(Luz solar)	100.0%	100.0%	100.0%	100.0%
Chão de porcelana(Luz lateral)	100.0%	100.0%	100.0%	66.67%

Tabela 3 – Porcentagem de falsos positivos

	#0	**#1**	**#2**	**#3**
Chão cimentado(Luz solar)	0%	0%	0%	0%
Chão cimentado(Luz à esquerda)	0%	0%	0%	0%
Chão cimentado(Luz ao centro)	0%	0%	0%	0%
Chão cimentado(Luz à direita)	0%	0%	0%	0%
Chão de porcelana(Luz solar)	0%	0%	0%	0%
Chão de porcelana(Luz lateral)	0%	0%	0%	0%

Tabela 4 – Erro médio de posição

	#0	**#1**	**#2**	**#3**
Chão cimentado(Luz solar)	34.97mm	34.45mm	36.33mm	34.89mm
Chão cimentado(Luz à esquerda)	35.75mm	32.73mm	35.78mm	33.90mm
Chão cimentado(Luz ao centro)	33.80mm	33.31mm	33.18mm	32.13mm
Chão cimentado(Luz à direita)	32.20mm	32.70mm	33.30mm	32.57mm
Chão de porcelana(Luz solar)	8.19mm	6.85mm	7.39mm	5.55mm
Chão de porcelana(Luz lateral)	7.22 mm	6.74mm	-	-

Tabela 5 – Erro médio de orientação

	#0	**#1**	**#2**	**#3**
Chão cimentado(Luz solar)	8.29º	2.62º	8.78º	2.41º
Chão cimentado(Luz à esquerda)	2.16º	2.82º	3.23º	1.72º
Chão cimentado(Luz ao centro)	18.92º	2.12º	2.88º	2.86º
Chão cimentado(Luz à direita)	13.58º	4.01º	2.09º	2.40º
Chão de porcelana(Luz solar)	1.24º	1.04º	1.45º	1.26º
Chão de porcelana(Luz lateral)	1.26º	2.04º	1.45º	2.06º

Tabela 6 – Tempo de execução do Algoritmo ArUco executado em software, para um computador pessoal e a plataforma PYNQ-Z2

	Computador pessoal	**PYNQ-Z2**
Média	0.090 s	1.350 s
Mediana	0.088 s	1.340 s
Desvio Padrão	0.004	0.020

Na Tabela 5 é apresentado o erro médio de orientação para as detecções do algoritmo. Com erros aceitáveis. As marcações em vermelho e verde representam respectivamente o valor mínimo e máximo de erro médio.

A Tabela 6 apresenta o tempo de execução do algoritmo, para duas plataformas computacionais. Para um computador pessoal, o desempenho do algoritmo é aceitável dentro do mínimo de aproximadamente 20 quadros por segundo, para se executar a localização dos robôs em tempo real. Para a PYNQ-Z2 no entanto, o tempo de execução teve uma média de 1.350 s para a execução do algoritmo ArUco em uma única imagem. Este resultado para PYNQ-Z2 utiliza apenas o processador ARM, executando tudo em software, sem utilizar a FPGA de seu SoC.

Este resultado sugere o desenvolvimento de uma arquitetura em hardware para acelerar o algoritmo ArUco, para que ele atinja um tempo de execução aceitável para uma aplicação em tempo real, no caso de se utilizar uma PYNQ-Z2.

Na Figura 18 é possível ver um esquema com os tempos de execução de cada uma das funções. É possível notar que as funções de binarização da imagem(*Adaptive Threshold*), a transformação para tons de cinza e a extração de contornos(Suzuki) são as

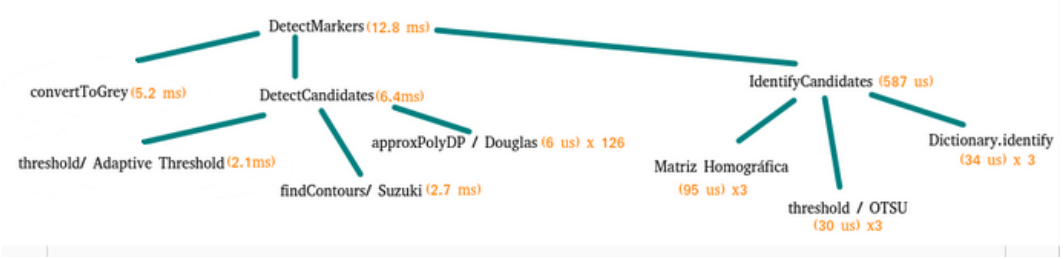

Figura 18 – Tempos de execução para as principais funções do algoritmo ArUco, testado em um computador pessoal para uma imagem de teste da base de dados.

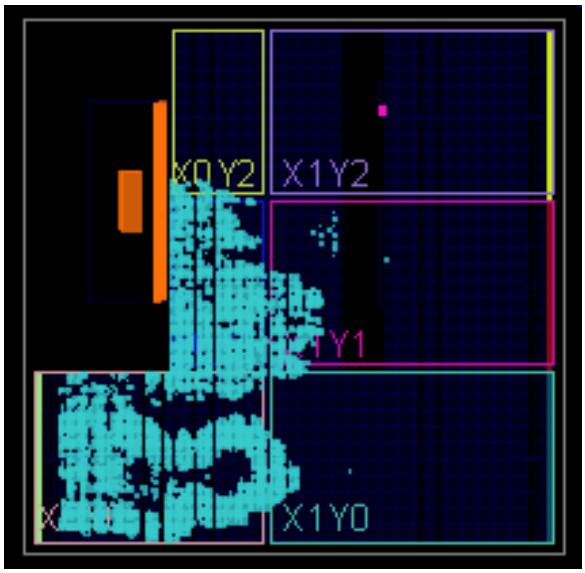

Figura 19 – Design implementado na FPGA para o teste do *Adaptive Threshold*.

funções mais demoradas do algoritmo. Essa análise de tempo justifica a aceleração destes estágios do algoritmo. Para este trabalho consideraremos uma imagem na memória, já em tons de cinza, pulando o primeiro passa de transformar a imagem para tons de cinza, portanto os estágios implementados em hardware são os 3 posteriores, que é a binarização da imagem, a extração de contornos e a aproximação poligonal.

4.2 Implementação do *Adaptive Threshold* em hardware

Para o teste do bloco de *threshold*, foram coletados o consumo de energia e o consumo de recursos da FPGA. A potência total dissipada para esta arquitetura é de 1.416 W. Para o consumo de recursos é possível analisar a Tabela 7. O layout desta implementação na FPGA fica populado conforme a Figura 19 e o consumo de energia é distribuído como na Figura 20.

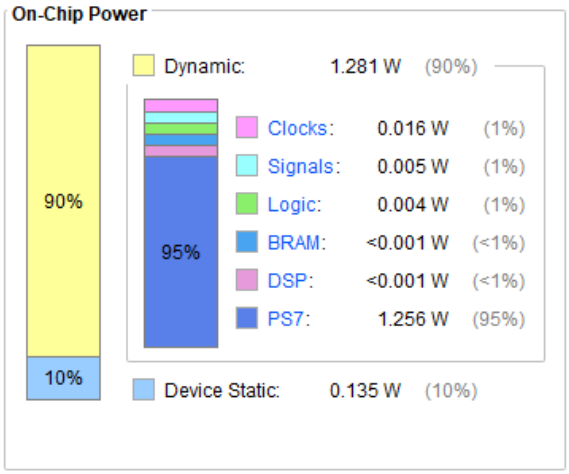

Figura 20 – Consumo de energia da arquitetura de teste para o algoritmo de binarização de imagens.

Tabela 7 – Tabela com o consumo de recursos da FPGA para o algoritmo de binarização.

Recurso	**Utilização**	**Disponível**	**Utilização %**
Look-Up Table	6657	53200	12.51
Look-Up Table RAM	259	17400	1.49
Flip-Flop	7800	106400	7.33
BRAM	3	140	2.14
DSP	8	220	3.64
Buffer do Clock	1	32	3,13

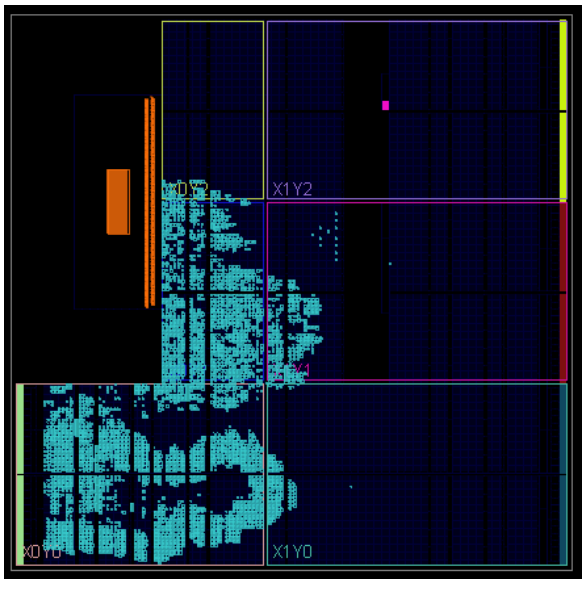

Figura 21 – Design implementado na FPGA para o teste da extração de contornos.

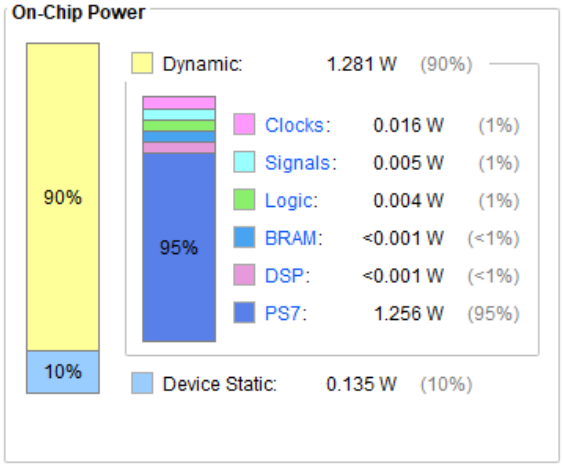

Figura 22 – Consumo de energia da arquitetura de teste para o algoritmo de extração de contornos.

4.3 Implementação da extração de contornos em hardware

Para o teste do bloco de extração de contornos, foram coletados o consumo de energia e o consumo de recursos da FPGA. A potência total dissipada para esta arquitetura é de 1.429 W. Para o consumo de recursos(Tabela 8), o consumo de BRAM se deve ao uso de um buffer para armazenar a imagem de 300x200 px, visto que é necessário ter a imagem completa para que o algoritmo de extração de contorno possa ser executado.

Esta arquitetura no entanto funcionou apenas para imagens de resolução baixa. Para imagens maiores que 50x50 px o algoritmo parou de funcionar. Dada a complexidade do algoritmo, é possível que ao se aumentar a resolução da imagem de teste, se obtenham circuitos combinacionais que não alcancem os parâmetros de temporização, e isso cause que a estrutura funcione de modo inesperado.

Uma possível análise para trabalhos futuros é testar esta arquitetura reduzindo-se a clock da FPGA e observar se a arquitetura continua falhando a partir do mesmo tamanho de imagem. Se o tamanho da imagem em que o algoritmo começa a falhar aumentar, isso indica que o problema pode estar relacionado com a temporização. Caso contrário, é possível que o problema seja estrutural, na estrutura do código escrito.

A arquitetura proposta permite a configuração da resolução da imagem via AXI-LITE. É possível testar soluções mais simples que não permitam essa funcionalidade, e verificar se o erro persiste.

Pode-se buscar também maneiras de simplificar os laços de repetição contidos nesta lógica, a fim de simplificar a arquitetura e eliminar os erros encontrados.

O layout desta implementação na FPGA fica populado conforme a Figura 21 e o consumo de energia é distribuído como na Figura 22.

Tabela 8 – Tabela com o consumo de recursos da FPGA para o algoritmo de de extração de contornos.

Recurso	**Utilização**	**Disponível**	**Utilização %**
Look-Up Table	6191	53200	11.64
Look-Up Table RAM	275	17400	1.58
Flip-Flop	7979	106400	7.50
BRAM	37.50	140	26.79
DSP	16	220	7.27
Buffer do Clock	1	32	3.13

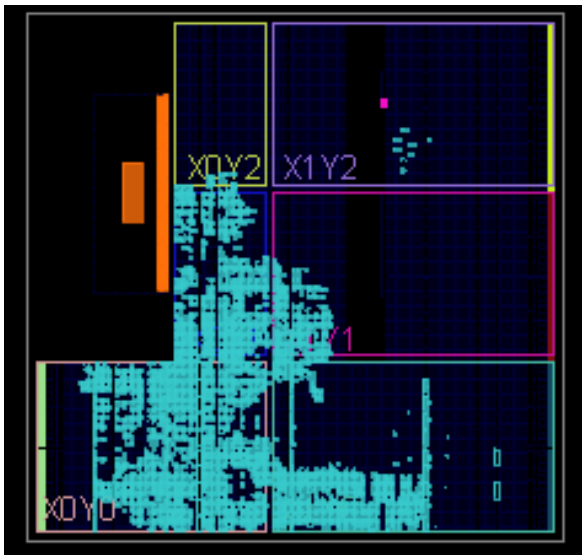

Figura 23 – Design implementado na FPGA para o teste da aproximação poligonal.

4.4 Implementação da aproximação poligonal em hardware

Para o teste do bloco de aproximação poligonal, foram coletados o consumo de energia e o consumo de recursos da FPGA. A potência total dissipada para esta arquitetura é de 1.493 W. Para o consumo de recursos(Tabela 9), é importante observar o consumo de BRAM, este alto consumo é devido a necessidade de se armazenar um buffer com um tamanho suficientemente grande para armazenar contornos arbitrariamente grandes dentro do limite possível em um imagem de 300x200 px. O layout desta implementação na FPGA fica populado conforme a Figura 23 e o consumo de energia é distribuído como na Figura 24.

4.5 Comparação de tempo

Para verificar se a arquitetura proposta obteve melhores resultados que a solução em software, foram feitos testes usando as imagens da base de dados de forma a testar todas as imagens da base com a execução dos 3 estágios que se deseja acelerar neste trabalho. Foram testadas as soluções implementadas em software, utilizando as funções

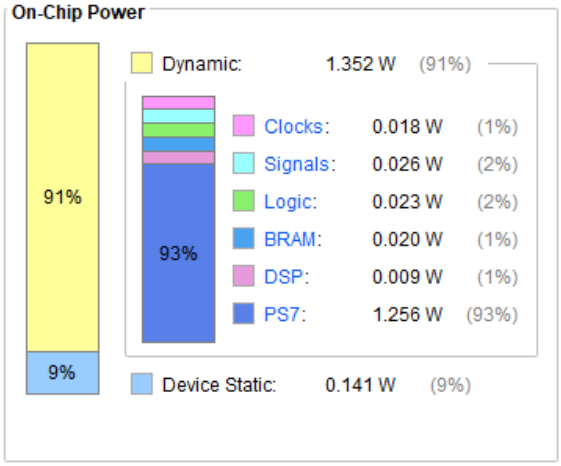

Figura 24 – Consumo de energia da arquitetura de teste para o algoritmo de aproximação poligonal.

Tabela 9 – Tabela com o consumo de recursos da FPGA para o algoritmo de aproximação poligonal.

Recurso	**Utilização**	**Disponível**	**Utilização %**
Look-Up Table	8109	53200	15.24
Look-Up Table RAM	412	17400	2.37
Flip-Flop	9099	106400	8.55
BRAM	53	140	37.86
DSP	18	220	8.18
Buffer do Clock	1	32	3.13

Tabela 10 – Tabela com tempos de execução dos algoritmos e taxa de "Ciclos / Pixel"de execução da arquitetura. O clock utilizado para a FPGA é de 50 MHz.

	Média	Mediana	Desvio Padrão	Ciclos/Pixel
Adaptive Threshold por sw	3,11 ms	3,03 ms	0,39 ms	-
Adaptive Threshold por hw	147,07 ms	147,07 ms	0,03 ms	122.55
Extração de contorno por sw	2,20 ms	2,23 ms	0,27 ms	-
Extração de contorno por hw	1,09 ms	1,08 ms	0,05 ms	545
Aproximação Poligonal por sw	138,00 ms	138,60 ms	17,45 ms	-
Aproximação Poligonal por hw	406,47 ms	399,11 ms	177,88 ms	338.725

implementadas na biblioteca de visão computacional OpenCV, e a solução em hardware proposta. Os dados dos tempos de execução coletados podem ser vistos na Tabela 4.5. Excepcionalmente para o algoritmo de extração de contornos, utilizou-se imagens com resolução de 10x10 px. Enquanto que para as demais imagens, a resolução utilizada foi de 300x200 px.

Como resultado, a solução em hardware, exceto para a extração de contorno, não conseguiu superar a solução já implementada em software, chegando, no caso do *threshold* a ficar várias vezes maior, o tempo de execução da solução proposta.

5 Conclusões

A partir dos resultados obtidos é possível inferir que o ArUco como forma de localização de robôs é uma técnica robusta, porém, precisa de melhorias para aplicações embarcadas.

Os resultados dos testes na base de dados, rastreando os critérios de desempenho propostos, mostram que o algoritmo ArUco tem a capacidade de identificar robôs com índices de detecções próximos a 100% para a base de dados levantada para este trabalho. Nenhum falso positivo foi identificado para este algoritmo, e os erros de orientação e posição se mostraram dentro de um limite razoável.

O tempo de execução desta aplicação na plataforma PYNQ-Z2 no entanto foi muito alto para ser utilizado em uma aplicação em tempo real. Uma arquitetura em hardware foi proposta para acelerar estágios do algoritmo que tem um custo computacional elevado. A arquitetura proposta utilizando o HLS não foi bem sucedida em acelerar a execução dos estágios do ArUco implementados em hardware. A arquitetura do algoritmo de extração de contornos também se mostrou instável ao se aumentar a resolução das imagens utilizadas para o teste, parando de funcionar em resoluções mais altas.

Portanto, para utilizar o ArUco como um sistema de localização de robôs, é necessário pesquisar mais a fundo o que está causando a instabilidade no algoritmo de Suzuki-Abe, se utilizando das orientações sugeridas na discussão dos resultados. Descoberta a fonte deste problema, pode-se partir para uma codificação dos blocos de hardware se utilizando de uma estratégia usando a linguagem VHDL ou o Verilog. Outra opção seria continuar utilizando o HLS, porém explorando mais a fundo as estruturas de paralelismo, utilizando as funcionalidades mais avançadas da ferramenta.

Referências

BURRELL, T. et al. Towards a cooperative robotic system for autonomous pipe cutting in nuclear decommissioning. In: IEEE. *2018 UKACC 12th International Conference on Control (CONTROL)*. [S.l.], 2018. p. 283–288. Citado na página 31.

CALINON, S.; GUENTER, F.; BILLARD, A. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 37, n. 2, p. 286–298, 2007. Citado na página 14.

DOUGLAS, D. H.; PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, University of Toronto Press, v. 10, n. 2, p. 112–122, 1973. Citado na página 23.

GARRIDO-JURADO, S. et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, Elsevier, v. 47, n. 6, p. 2280–2292, 2014. Citado 3 vezes nas páginas 7, 18 e 19.

GARRIDO-JURADO, S. et al. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, Elsevier, v. 51, p. 481–491, 2016. Citado na página 37.

GHORBEL, A. et al. An fpga based platform for real time robot localization. In: IEEE. *2013 International Conference on Individual and Collective Behaviors in Robotics (ICBR)*. [S.l.], 2013. p. 56–61. Citado na página 32.

HAYES, G. M.; DEMIRIS, J. *A robot controller using learning by imitation*. [S.l.]: University of Edinburgh, Department of Artificial Intelligence, 1994. Citado na página 14.

JIN, S. H.; CHO, J. U.; JEON, J. W. Fpga based passive auto focus system using adaptive thresholding. In: IEEE. *2006 SICE-ICASE International Joint Conference*. [S.l.], 2006. p. 2290–2295. Citado na página 32.

LIMITED, A. An introduction to amba axi. ARM, 2021. Citado 2 vezes nas páginas 7 e 30.

MARUT, A.; WOJTOWICZ, K.; FALKOWSKI, K. Aruco markers pose estimation in uav landing aid system. In: IEEE. *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. [S.l.], 2019. p. 261–266. Citado 2 vezes nas páginas 31 e 32.

MUÑOZ-SALINAS, R.; MEDINA-CARNICER, R. Ucoslam: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *Pattern Recognition*, Elsevier, v. 101, p. 107193, 2020. Citado na página 18.

OTSU, N. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, IEEE, v. 9, n. 1, p. 62–66, 1979. Citado na página 26.

ROSENFELD, A. Connectivity in digital pictures. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 17, n. 1, p. 146–160, 1970. Citado na página 21.

SANI, M. F.; KARIMIAN, G. Automatic navigation and landing of an indoor ar. drone quadrotor using aruco marker and inertial sensors. In: IEEE. *2017 international conference on computer and drone applications (IConDA)*. [S.l.], 2017. p. 102–107. Citado na página 32.

SANTOS, D. M. H. *Repositório do Sistema de Localização de Robôs*. 2021. Disponível em: <https://gitlab.com/lab154/aruco_robot_localization>. Citado 2 vezes nas páginas 14 e 15.

SANTOS, D. M. H. *Vídeo de apresentação do Sistema de Localização de Robôs para o Congresso de Iniciação Científica*. 2021. Disponível em: <https://youtu.be/CE0ZRFMOPdI>. Citado na página 38.

SEKAJ, I.; CÍFERSKỲ, L.; HVOZDÍK, M. Neuro-evolution of mobile robot controller. In: *Mendel*. [S.l.: s.n.], 2019. v. 25, n. 1, p. 39–42. Citado na página 13.

SHEN, T. et al. Optimized vision-based robot motion planning from multiple demonstrations. *Autonomous Robots*, Springer, v. 42, n. 6, p. 1117–1132, 2018. Citado na página 14.

SHEN, Y. Efficient normalized cross correlation calculation method for stereo vision based robot navigation. *Frontiers of Computer Science in China*, Springer, v. 5, n. 2, p. 227–235, 2011. Citado na página 13.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to autonomous mobile robots*. [S.l.]: MIT press, 2011. Citado na página 13.

SUZUKI, S. et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, Elsevier, v. 30, n. 1, p. 32–46, 1985. Citado na página 21.

TANWANI, A. K.; BILLARD, A. Transfer in inverse reinforcement learning for multiple strategies. In: IEEE. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2013. p. 3244–3250. Citado na página 13.

TIAN, J.; WU, J.; WANG, G. Realization of real-time sobel adaptive threshold edge detection system based on fpga. In: IEEE. *2015 IEEE International Conference on Information and Automation*. [S.l.], 2015. p. 2740–2743. Citado na página 32.