Insper

Curso: Engenharia Mecatrônica

Disciplina: Robótica Industrial / Visão de Máquina

Professores: Leonardo de Souza e Silva Tavares / Carlos Magno de Oliveira Valente

Integrando um sistema de visão e um robô colaborativo para jogar dominó

Gabriel Pinto Patrocinio – 15610367 – gabrielpp@al.insper.edu.br

Pedro Casella – 15610326 – pedroctr@al.insper.edu.br

Rodrigo Ronchail Gikas – 15610385 – rodrigo.rgk@hotmail.com

Sumário

1. Introdução

Este projeto consiste na integração de um robô colaborativo UR5 da Universal Robots A/S com um sistema de visão e o software Labview para criar um robô autônomo capaz de jogar dominó contra uma pessoa, desde identificação das peças, posição e orientação, até a realização da jogada e comunicação com o usuário.

Para isso se tornar possível foram utilizados conceitos de robótica, como transformações homogêneas e programação de robôs, comunicação por rede, fabricação, design, visão de máquinas, entre outros.

2. Robótica

Nesta seção serão discutidos todos os elementos da divisão de robótica do projeto, o que inclui desde o robô utilizado, o sistema de comunicação, as limitações, programação e as soluções elaboradas para resolver os problemas encontrados. A execução do projeto no âmbito da robótica não apresenta tanta dificuldade como a solução da seção de visão computacional, entretanto, também são necessários alguns cuidados e uma certa perspicácia na resolução dos problemas, além de um conhecimento de programação, redes industriais e geometria.

2.1. O Desafio da Robótica

Como foi mencionado anteriormente, o desafio da robótica para este projeto não oferece muitas complicações, ele se resume em:

i. Projetar e fabricar uma garra adequada para a aplicação
ii. Estabelecer a comunicação com o Labview para receber as informações da jogada
iii. Utilizar a comunicação com o Labview para enviar a informação do jogador
iv. Transformar os valores de posições recebidos em relação à câmera para serem em relação ao robô
v. Mover-se sem riscos de acidente para as posições recebidas

Para a realização destes objetivos são necessários diversos passos que serão relatados nos tópicos seguintes.

2.2. Robô Utilizado

O robô utilizado para a aplicação era desde o início um parâmetro já definido de projeto. Foi utilizado o Robô Colaborativo UR5 da Universal Robots, este robô é capaz de trabalhar com cargas uteis de até 5kg em um raio de 850mm, ocupando uma área de 149mm para sua instalação. Os seis graus de liberdades proporcionados pelas suas seis juntas rotativas fazem com que qualquer posição e orientação desejada possa ser atingida dentro da área de trabalho do robô, entretanto, é importante destacar que há uma limitação na rotação das juntas, assunto que será discutido no tópico 2.4.2.

O UR5 ainda possui além de algumas portas I/O digitais, a possibilidade de conexão via Modbus, TCP-IP Socket e Profinet. No projeto foi utilizado uma entrada digital e a conexão via Modbus, na qual o robô foi definido como Slave e o Labview como Master, assunto que será discutido no tópico 2.3. A figura abaixo, retirada do site do fabricante, mostra o UR5 utilizado. Além disso, está anexado neste relatório o Datasheet do robô, como apêndice 1.

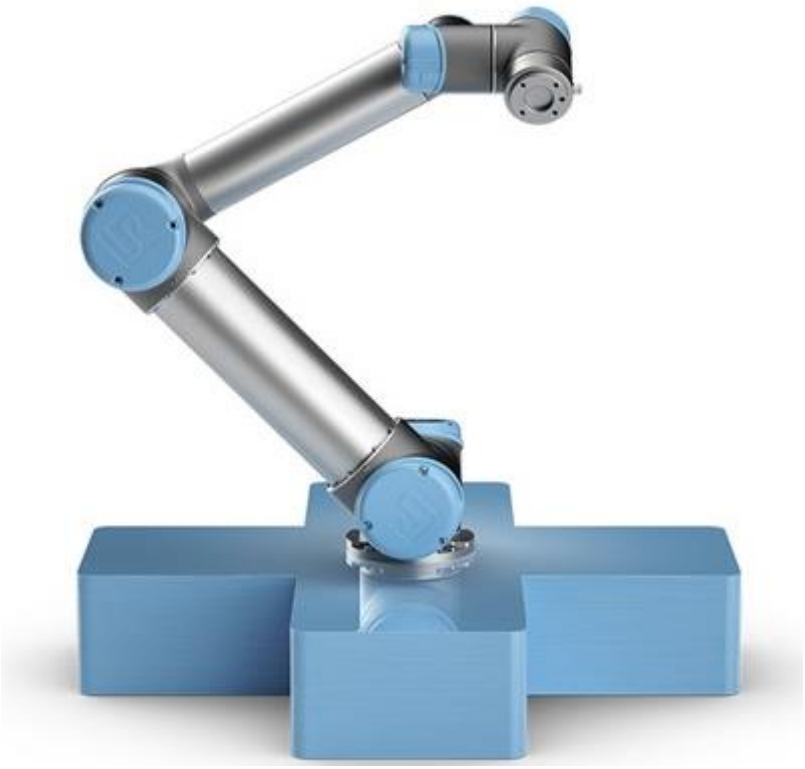


Figura 1 - UR5 (Imagem retirada de https://www.universal-robots.com/pt/produtos/ur5/)

A figura a seguir apresenta o UR5 com suas medidas, todas em milímetros:

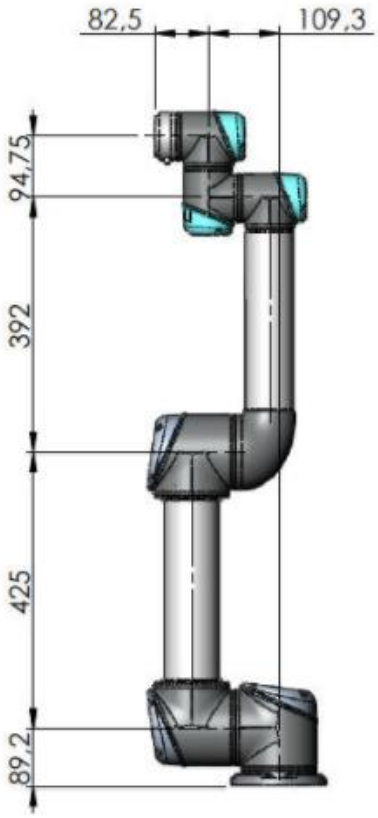


Figura 2- Medidas UR5 (Imagem da de: http://www.zacobria.com/universal-robots-zacobria-forum-hints-tips-how-to/script-client-server-example/)

2.3. Sistema de Rede

Como foi mencionado anteriormente, o UR5 possibilita a comunicação por rede de diversos tipos, contudo, o Labview utiliza o Modbus como método mais acessível, fazendo com que esta seja a rede escolhida para a aplicação. Por recomendação presente no manual da Universal Robots, o UR5 foi definido como Slave da comunicação Modbus, simplificando a captação de variáveis via script e o envio de informações para o Master, que seria o Labview.

2.4. Limitações e Soluções

Devido ao projeto ter começado com base nos equipamentos que havia disponível nos laboratórios, ou seja, não ter sido possível realizar a especificação adequada de todos os componentes do projeto, surgiram algumas limitações que necessitaram ser

resolvidas durante o andamento do projeto. Estas limitações e as soluções adotadas estão presentes nos subtópicos abaixo.

2.4.1. Curso da Garra

Figura 3 - Garra colaborativa Schunk – Robotic collaborative gripper EGP 40 CO ACT

Uma das limitações encontrada no projeto foi o curso das hastes da garra disponibilizada para uso. Estas apresentavam um curso máximo de $20mm$, o que para a aplicação idealizada seria inviável, uma vez que era necessário que as peças de dominó fossem de um tamanho razoável para facilitar a aplicação de visão de máquina no sistema, ainda assim tentando ser o mais fiel ao tamanho de uma peça normal de dominó. Além disso, um jogo normal de dominó necessita que as peças sejam posicionadas de dois jeitos diferentes, representados nas figuras 4 e 5 abaixo:

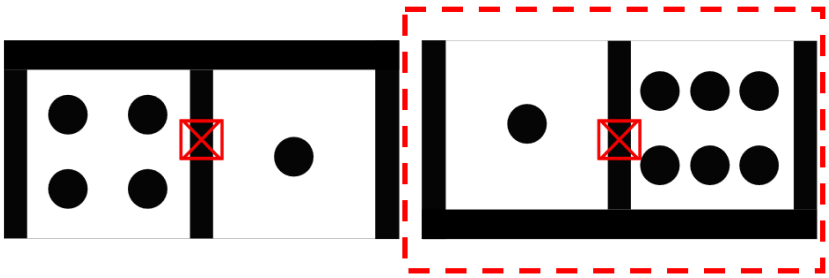


Figura 4 - Peça normal posicionada no tabuleiro

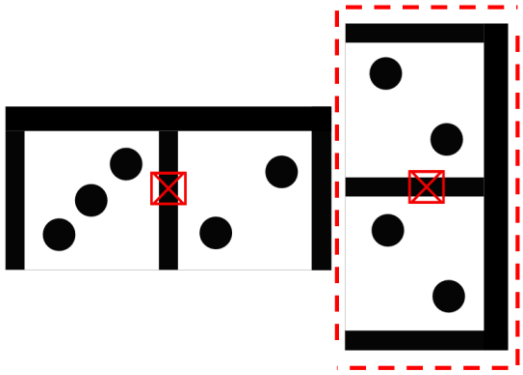


Figura 5 - Peça dobrão posicionada no tabuleiro

A peça indicada na Figura 5 necessitaria ser pega pela sua lateral direita para evitar que a garra, colocando a peça no tabuleiro durante a jogada, batesse na peça da ponta do jogo. Dessa forma o curso máximo é, novamente, um fator limitante, sendo que a diferença de comprimento entre a largura e o comprimento de uma peça normal de dominó é maior que os $20mm$ de curso disponíveis.

Diante dessas restrições, tornou-se indispensável a idealização de uma garra própria para a aplicação. Foram desenvolvidas hastes com duas espessuras diferentes para que dessa forma, ao acionar o mecanismo que fecha a garra, a distância entre elas seja levemente menor que a largura da peça em uma parte, para pegar a peça pelo meio, e levemente menor que o comprimento da peça na outra parte, para pegá-la pela lateral.

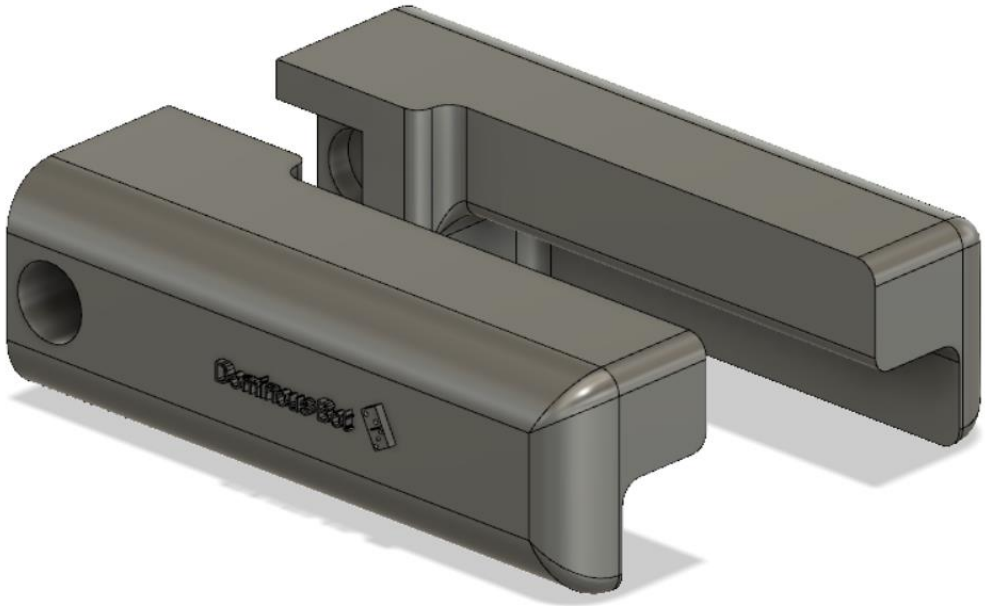


Figura 6 - Modelo da garra idealizado no Fusion360

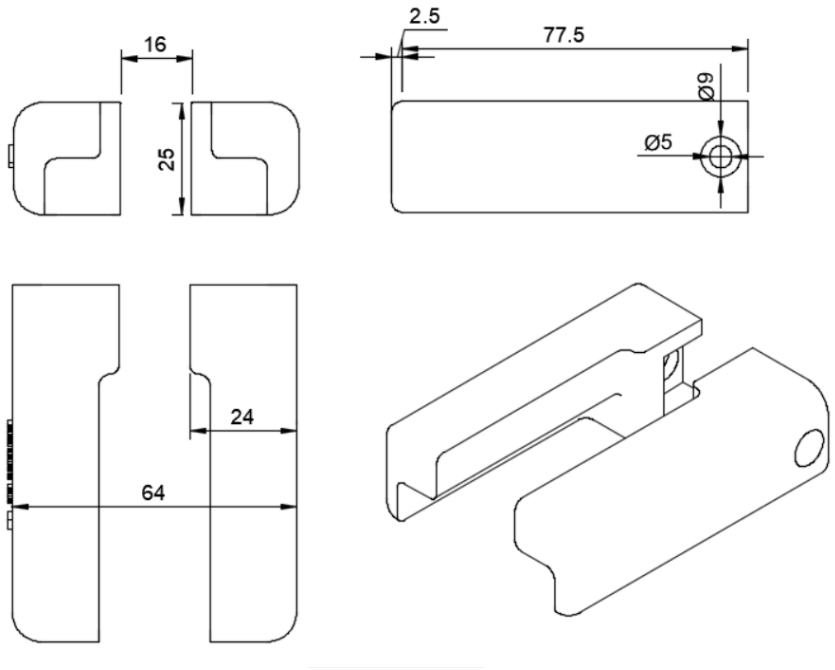


Figura 7 - Desenho técnico da garra idealizada

As imagens a seguir mostram as hastes prontas, já presa aos suportes na garra no braço e nas posições de pegar peça específicas de cada uma das disposições comentadas acima:

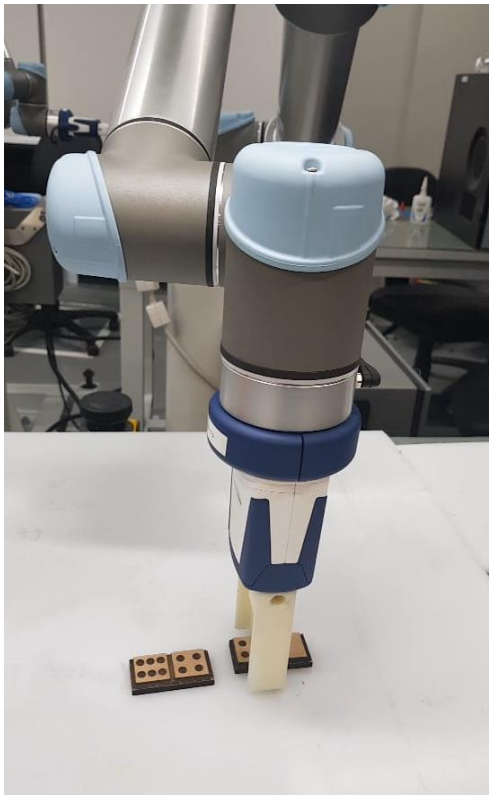


Figura 8 - Garra personalizada na posição de pega para uma peça normal

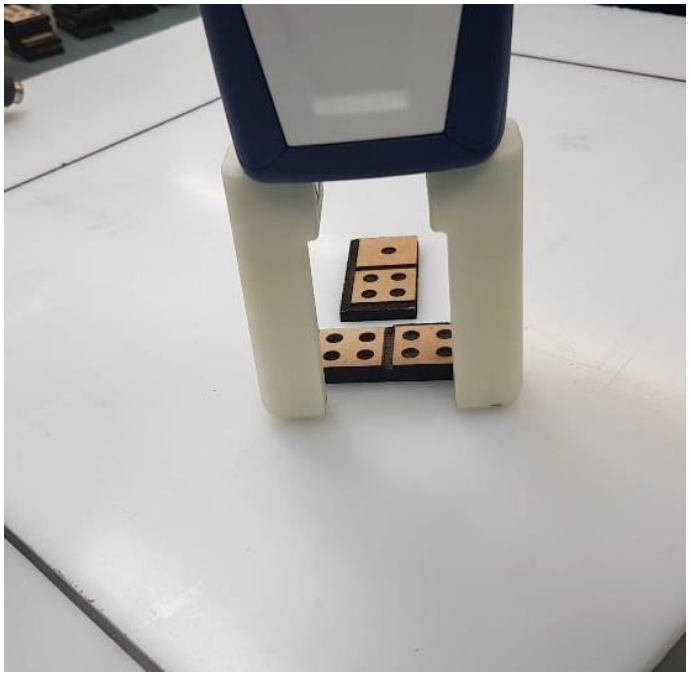


Figura 9 - Garra personalizada na posição de pega para uma peça dobrão

2.4.2. Ângulo de Rotação das Juntas

Uma outra limitação do sistema é o ângulo de rotação das juntas. Como especificado no Datasheet, cada junta possui a possibilidade de rotacionar $\pm 360°$, porém, como será explicado posteriormente na seção relacionada à obtenção do ângulo da peça no Labview, algumas vezes o robô receberá uma informação de rotação para o pulso que quando somada à sua rotação atual poderá exceder os 360° de rotação máxima, impossibilitando-o de atingir a posição desejada.

Para solucionar este problema, após cada movimento realizado pelo robô ele retorna para posições predefinidas anteriormente com ângulo do pulso referenciado em 0°, além disso, foi definido que o Labview envia os ângulos da peça sempre entre $\pm 180°$, evitando que o robô supere o limite de rotação do pulso. O lado negativo destas modificações é o impacto na velocidade da jogada, pois, o robô necessita após cada movimento voltar o pulso para a angulação predefinida ao invés de ir diretamente para a próxima posição, perdendo alguns segundos em cada movimento.

2.5. Fluxograma de Programação

No intuito de facilitar o entendimento da programação construiu-se um fluxograma para representar a lógica implementada no código. Este fluxograma possui uma função inicial (Init), uma função principal (Main) e algumas subfunções que são chamadas através da função Main quando necessário. Além disso há uma função que roda em paralelo com o programa principal, denominada Thread 1. Estes algoritmos podem ser vistos nas seções a seguir.

2.5.1. Função Inicial

Ao se iniciar o programa, algumas variáveis essenciais para o projeto são criadas com valores iniciais predefinidos para que o programa rode de maneira adequada na primeira iteração.

Logo em seguida, uma Popup aparece pedindo para que o usuário leve o robô para uma posição denominada “phome”. Esta é uma posição na qual o robô irá toda vez que completar sua jogada, ou seja, estiver aguardando o humano realizar a jogada dele. Assim que a pessoa levar o robô para a posição desejada, deverá apertar continuar na IHM e perceberá que o robô girará o pulso até que esteja em uma angulação de 0°.

Depois disso, internamente o robô guarda esta posição e retorna outra Popup para o usuário, pedindo dessa vez a posição “phand”. Esta posição pode ser entendida como a posição de abordagem para pegar a peça na mão do robô. É importante definir esta posição para que o robô não corra o risco de bater em nada durante sua jogada, uma boa sugestão seria definir esta posição aproximadamente no centro do local onde estão posicionadas as peças do robô.

Por fim, aparecerá na IHM a última Popup, desta vez pedindo para que o usuário leve o robô para a posição “pplay”, que tem a mesma função da posição de hand, só que desta vez para posicionar a peça no local adequado. Uma boa sugestão neste caso seria definir esta posição aproximadamente no centro do tabuleiro, onde o robô irá jogar suas peças.

Ao fim dessas definições, o robô irá mover-se para a posição de home definida inicialmente e chamará a função principal.

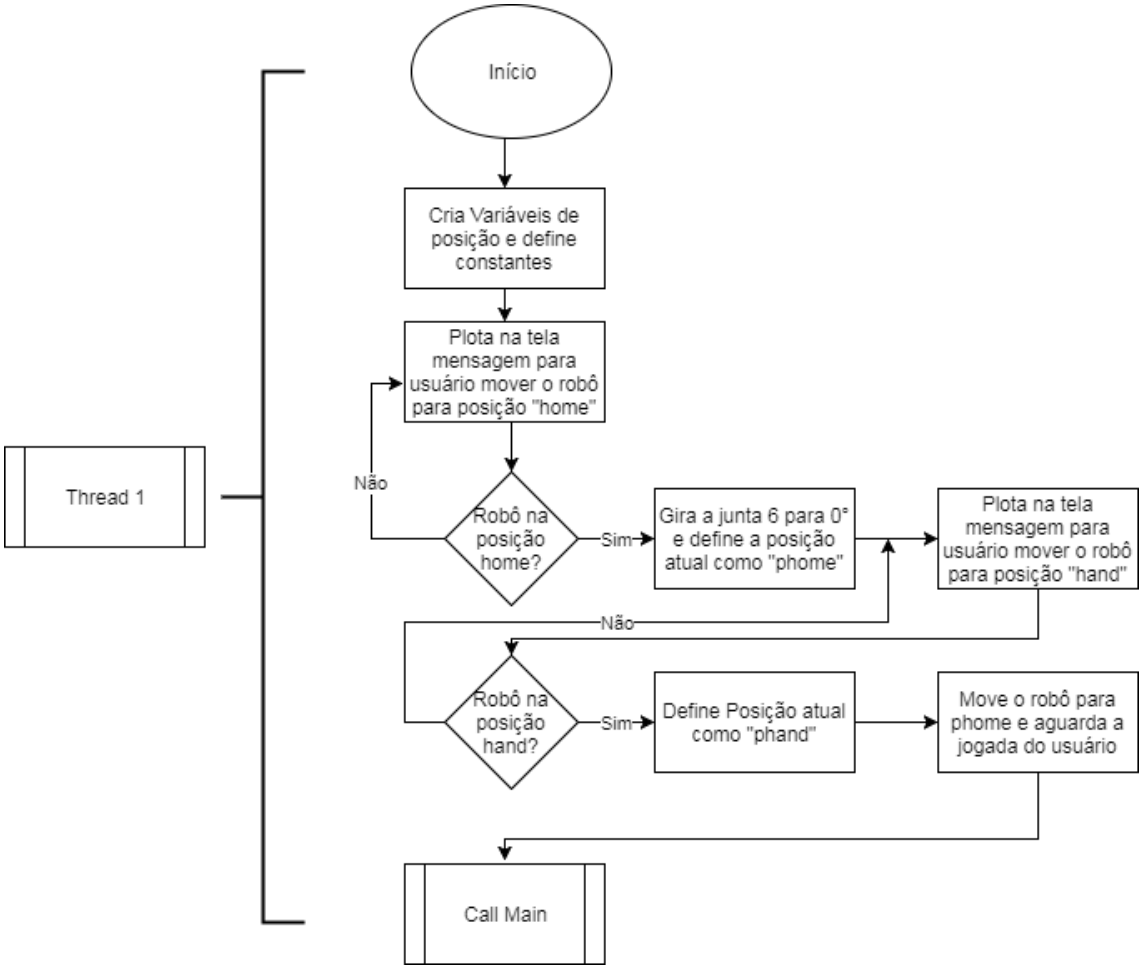


Figura 10 - Função Init

2.5.2. Função Principal

A função principal funciona utilizando uma lógica de estado de máquina, ou seja, ela fica checando o tempo todo os valores de algumas variáveis até que uma delas se altere e assim o algoritmo possa definir qual subfunção irá chamar.

Inicialmente checa-se se a variável de passar a vez é falsa, caso sim, uma variável denominada “pass_test” é definida como verdadeira e o algoritmo chama a função “Wait_Player”, que será detalhada posteriormente. No caso da afirmação ser verdadeira, ou seja, o robô não passe a vez e tenha uma peça para jogar, checa-se a variável “waiting_player”, que indica que o robô já finalizou sua jogada e está esperando o jogador realizar a dele. Caso esta variável seja verdadeira, checa-se se o robô já recebeu a confirmação do Labview indicando que o mesmo já processou as informações das câmeras e que o robô já pode realizar sua jogada. Se isso for verdade, o algoritmo irá chamar a função Pick_Hand (detalhada em 2.5.4). Caso contrário, o robô irá se

manter nesse laço esperando a confirmação do Labview de que pode finalmente realizar sua jogada.

No caso da variável waiting_player ser falsa, indicará que o robô estará no meio de sua jogada e já está com a peça desejada em sua mão, com isso, irá checar inicialmente se a variável piece_picked é verdadeira, caso seja, o algoritmo chamará a função place_piece (2.5.5).

Caso a variável piece_picked seja falsa, indicará que o robô também já posicionou a peça no local desejado, portanto, chama-se a função wait_player e finaliza-se a jogada retornando a variável robot_turn para falso, indicando que não está mais na vez do robô jogar e sim do humano.

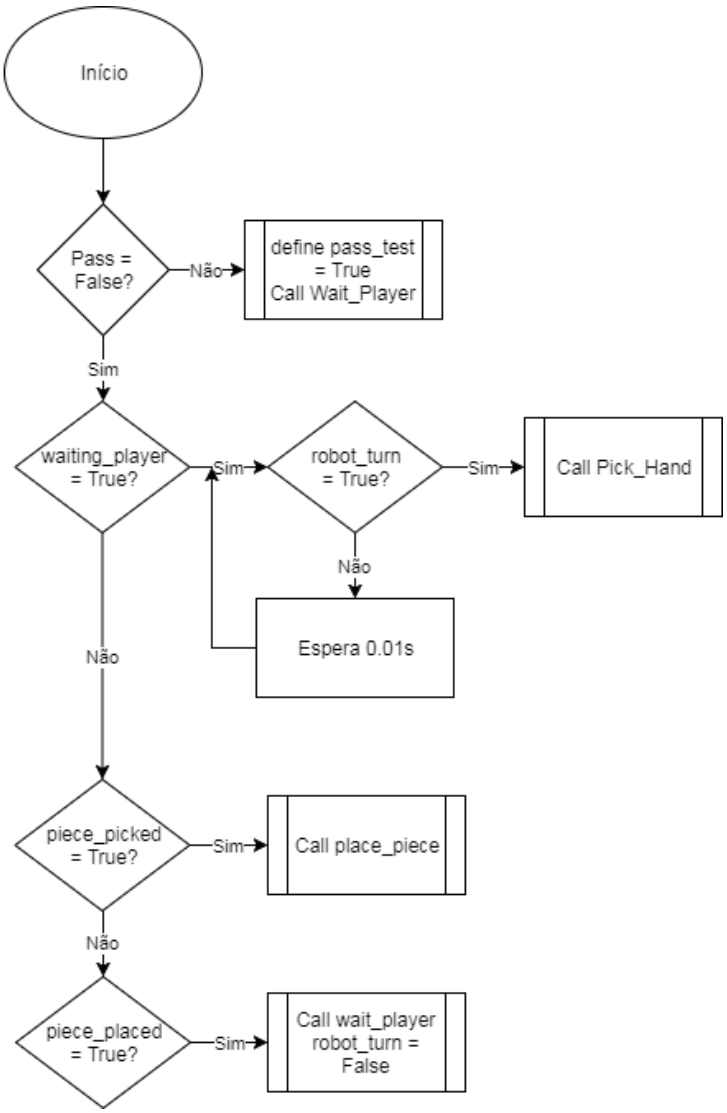


Figura 11- Função Main

2.5.3. Função de Espera

A função Wait_Player pode ser chamada de duas maneiras, quando o robô termina de realizar sua jogada, ou seja, colocou uma peça no tabuleiro, ou então quando o robô passa a vez.

Esta função inicialmente verifica duas variáveis para que, caso o robô tenha entrado nela por ter passado a vez e seja a primeira vez que ele entrou nessa função na jogada atual ele realizará uma rotina de movimentos no intuito de dar dois toques na mesa, indicando ao jogador que passou a vez.

Caso as variáveis sejam falsas, o robô simplesmente voltará para a posição de home e setará a variável waiting_player para True, indicando que finalizou sua jogada e aguarda novamente o jogador.

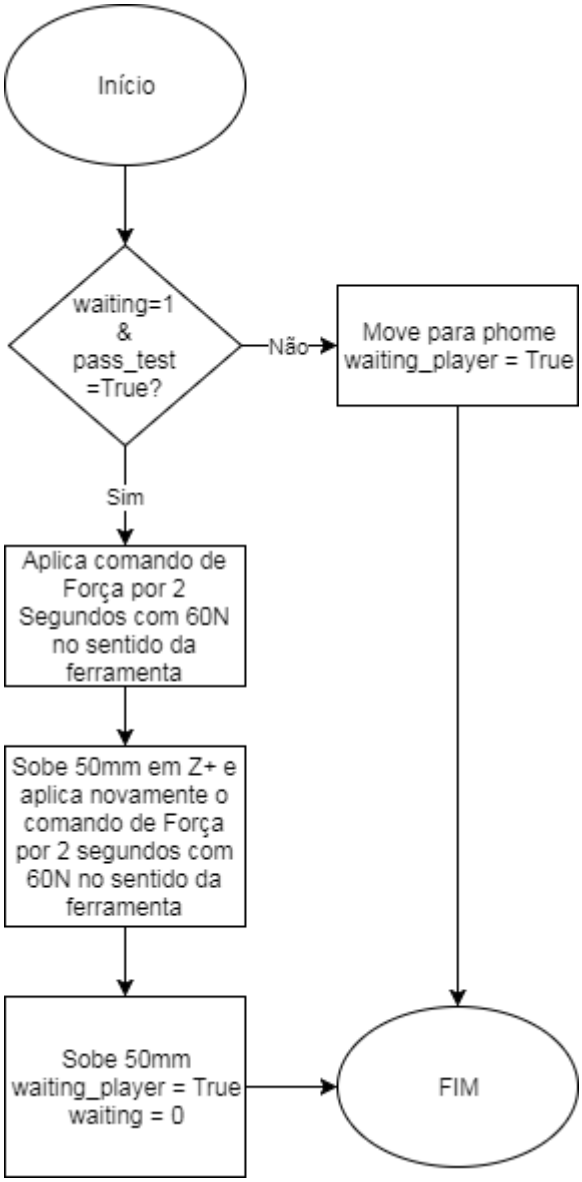


Figura 12 - Função Wait_Player

2.5.4. Função de Pegar a Peça

A função de pick_piece inicialmente muda a variável waiting_player para False, indicando que o robô não está mais esperando o jogador realizar sua jogada, e sim, ele próprio no meio da sua vez. Depois disso, moverá linearmente - para garantir que não corra o risco de bater em nada, causando acidentes – para a posição de hand, em seguida, abrirá a garra completamente e moverá para a posição transformada entre os valores obtidos pelo Labview e a posição da câmera da mão do robô, definida como uma Feature do programa. Uma melhor explicação desse processo de transformada de posição será feita na seção 2.6, mas por enquanto é importante saber que após essa

transformação ser realizada, o robô terá a posição e orientação da peça em relação à sua base, podendo se mover diretamente para a mesma e pegar a peça desejada.

Após se mover para a posição transformada, o algoritmo checa se a peça é um dobrão ou não, detalhe importante, pois, pela regra do dominó, caso a peça seja um dobrão deverá ser posicionada no tabuleiro de maneira diferente do que caso seja uma peça normal, detalhe explicado na seção de Visão, mais a frente neste documento.

Caso a peça não seja um dobrão o robô irá girar seu pulso no valor recebido e tratado pelo Labview, descerá um valor previamente estabelecido e fechará sua garra completamente. Em seguida subirá o mesmo valor descido previamente e retornará a angulação do pulso para 0° evitando o problema relatado na seção 2.4.2, relacionado à ângulo limite das juntas. Por fim, o robô mudará a variável piece_picked para True.

Caso a peça seja um dobrão, torna-se necessário uma correção na posição enviada pelo Labview, isso se dá pelo fato de que, ao construir a garra, precisou-se contar com a limitação detalhada na seção 2.4.1, o que gerou uma complicação ao se pegar e posicionar dobrões no tabuleiro. Essa correção será detalhada na seção 2.7 - Correção de Posição para Dobrões. Entretanto, após essa correção de posição ser feita o algoritmo se comportará da mesma maneira que no caso de não ser um dobrão, com a diferença no acréscimo de 90° na rotação do pulso para pegar a peça na orientação correta.

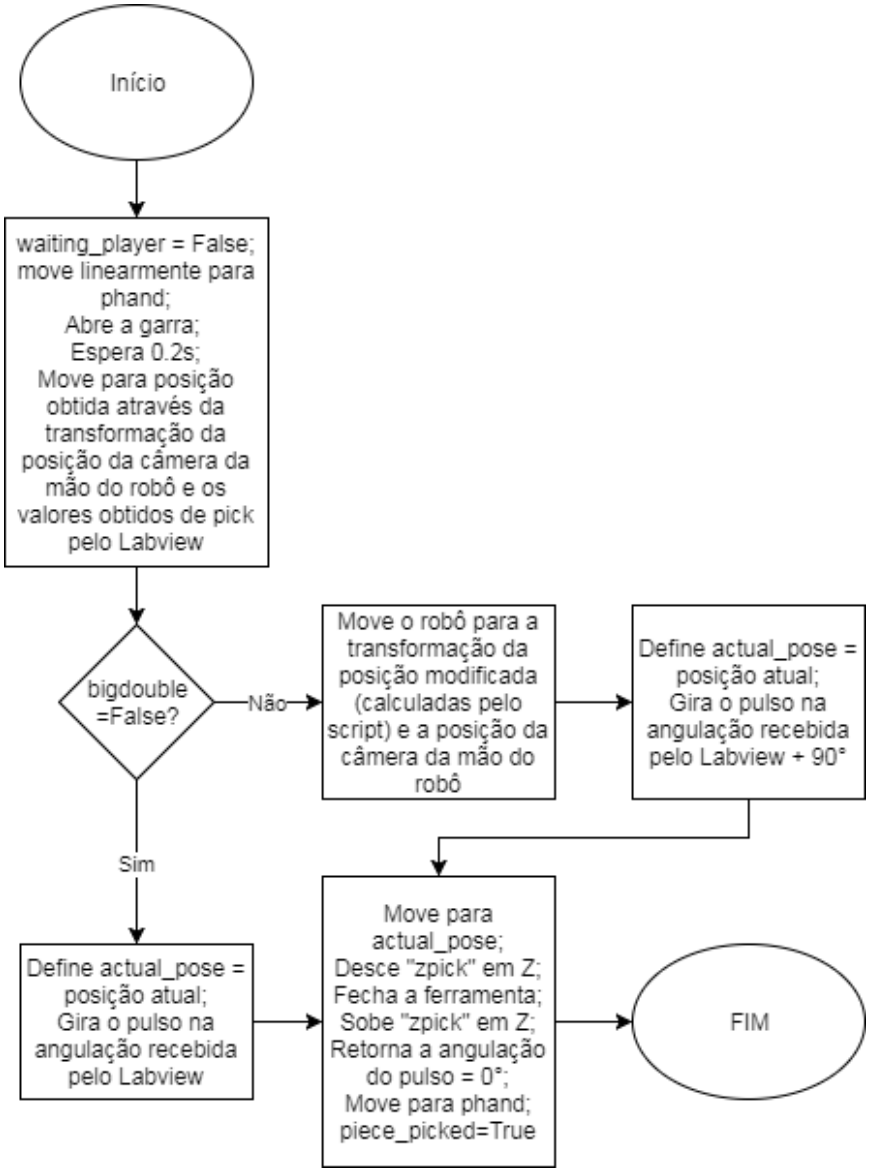


Figura 13 - Função Pick_Hand

2.5.5. Função de Colocar a Peça

A função de Place_piece se comporta de maneira muito semelhante a função Pick_piece, divergindo-se simplesmente no primeiro processo e na definição de algumas variáveis.

Inicialmente o algoritmo irá alterar a variável piece_picked para falso e mover também linearmente para a posição pplay. Em seguida irá se mover para a posição recebida pelo Labview e checará se a peça é ou não um dobrão. No caso de não ser, o robô girará a garra para a orientação na qual a peça deve ser colocada, descerá em Z e abrirá a garra, em seguida subirá em Z novamente e retornará a angulação da garra para 0°. Por fim definirá as variáveis game_over_var_1 e piece_placed para True.

No caso da peça ser um dobrão, o algoritmo fará as mesmas correções feitas em pick_piece e em seguida executará o mesmo processo que no caso da peça normal.

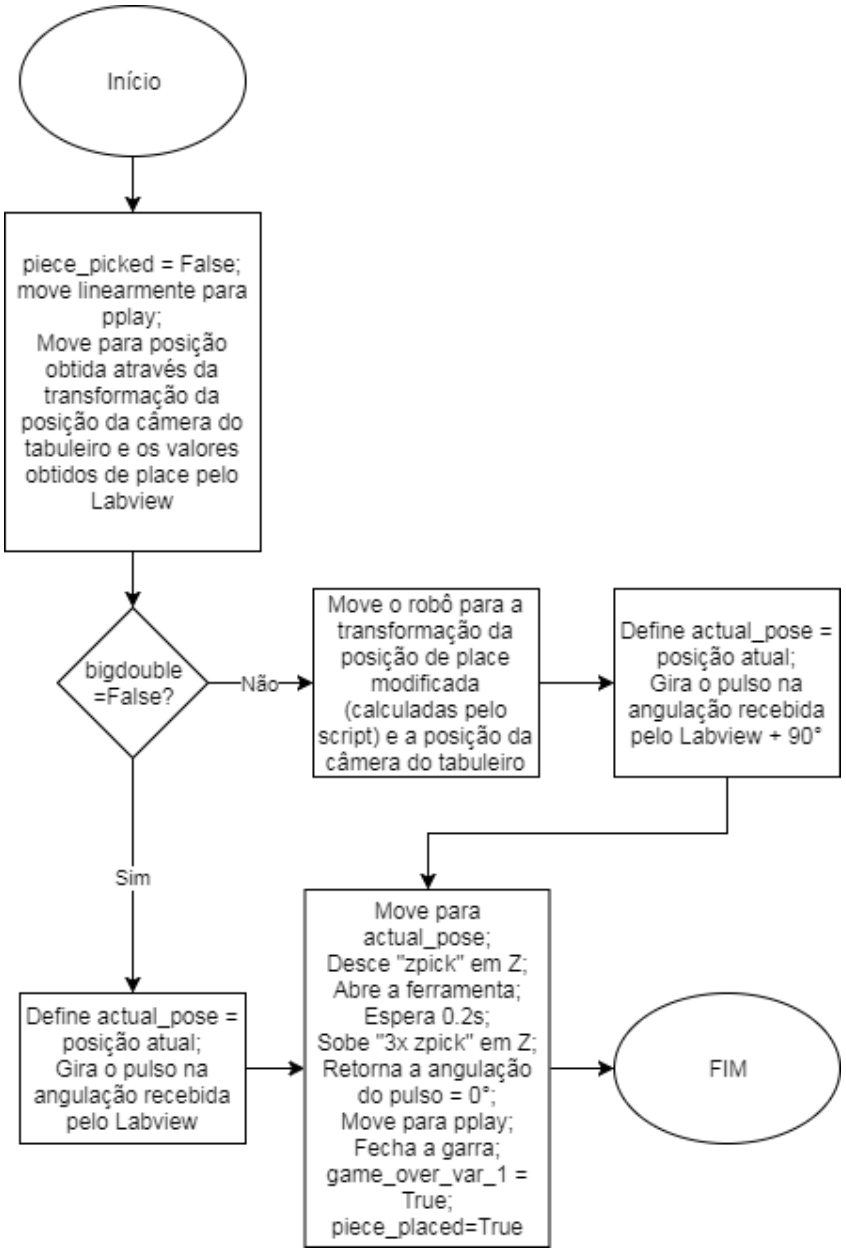


Figura 14 - Função Place_Piece

2.5.6. Função em Paralelo para comunicação com Labview

A função que estará sendo executada em paralelo com a rotina principal é de extrema importância para o funcionamento correto do programa. Esta função será responsável por se comunicar com o Labview obtendo as informações das câmeras já

processadas e corrigindo, quando necessário, essas informações através de scripts criado no próprio programa.

Inicialmente o programa cria a função converte_num e para entender sua necessidade primeiro é preciso saber que a comunicação Modbus realizada entre o UR5 e o Labview trabalha apenas com números inteiros e positivos, ou seja, quando um número positivo é enviado pelo Labview, ele pode atender os valores entre 0 e 32767. Caso o número enviado seja negativo, ele será processado como um valor entre 65536 e 32768, nessa ordem. Desta forma, o número "-1" quando enviado pelo Labview chega no robô como 65535.

Dito isso, fica simples entender a função converte_num, expressa abaixo como fluxograma:

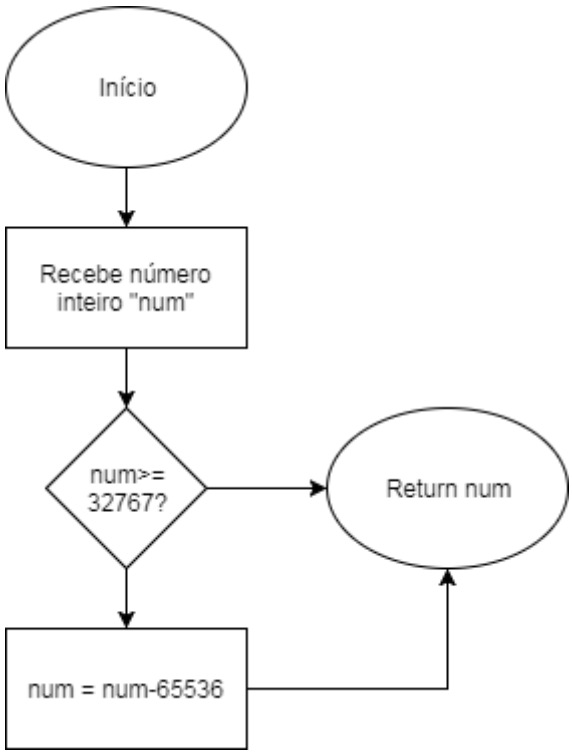


Figura 15 - Função converte_num

Em seguida, o programa chama as funções pick_values_modbus_check e calcula_tcp.

A primeira lê os valores de registradores recebidos pelo Modbus, aplica a função converte_num para obter o valor real e em seguida divide por 1000, para transformar o valor para metros, unidade padrão utilizada nos scripts de movimento. Além disso, ela

também lê os valores dos bits do Modbus e os designa para as variáveis adequadas. O código da função pode ser facilmente interpretado e está presente logo abaixo:

```
#LEITURA DOS REGISTRADORES

PPICKX = CONVERTE_NUM(READ_PORT_REGISTER(128))/1000

PPICKY=CONVERTE_NUM(READ_PORT_REGISTER(129))/1000

PPLACEX=CONVERTE_NUM(READ_PORT_REGISTER(130))/1000

PPLACEY=CONVERTE_NUM(READ_PORT_REGISTER(131))/1000

PICKTHETA=CONVERTE_NUM(READ_PORT_REGISTER(132))/1000

PLACETHETA=CONVERTE_NUM(READ_PORT_REGISTER(133))/1000

#LEITURA DOS BOOLEANOS

BIGDOUBLE=READ_PORT_BIT(17)

SINAL_ENVIADO=READ_PORT_BIT(20)

PASS=READ_PORT_BIT(18)

GAME_OVER=READ_PORT_BIT(19)
```

A função calcula_tcp será explicada na seção 2.7- Correção de Posição para Dobrões.

Continuando a rotina do Thread, o algoritmo irá checar se o botão do usuário foi pressionado (indicando que já realizou sua jogada) e se a variável piece_placed = True. Essa checagem dupla serve para garantir que mesmo que o usuário aperte o botão durante a realização da jogada do robô, ele irá concluir sua jogada normalmente e depois sim o usuário deverá realizar a sua jogada e apertar o botão.

Caso ambas as variáveis sejam verdadeiras, indicará que o Labview poderá capturar as imagens da mão do robô e do tabuleiro e analisa-las, portanto, primeiramente o algoritmo seta a variável piece_placed para falso e envia um bit ao Labview indicando que pode começar suas análises. Ao término do processamento do Labview, este retorna um bit indicando que já enviou os dados para o robô, este então

espera um tempo e chama novamente as funções de leitura e correção dos valores do Labview, atualizando todas as posições para a nova jogada.

Por fim, o algoritmo define as variáveis game_over_var_1 para False e robot_turn igual a True, fazendo com que a função Main ative seu “efeito cascata” realizando a jogada do robô.

Caso alguma das variáveis seja falsa, o robô checará se o Labview enviou um sinal de Game Over. Caso tenha enviado, o robô checará a variável game_over_var_1, que se torna verdadeira ao fim da jogada do robô. Essa checagem dupla ocorre para que, caso o robô vença o jogo, ele posicione a última peça corretamente antes de mostrar a Popup de Fim de Jogo.

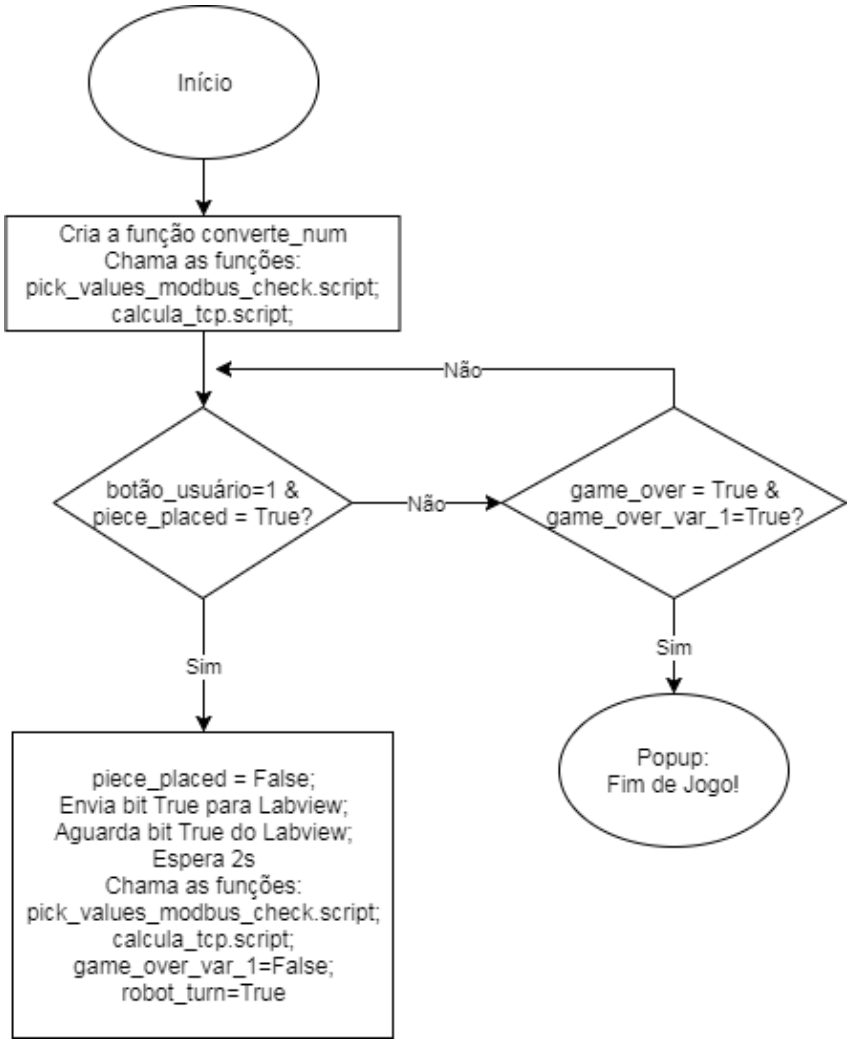


Figura 16 - Função Thread 1

2.6. Transformada de Posição Câmera – Robô

Um dos conhecimentos importantes para a realização deste projeto é o de Transformação Homogênea de posição. Como foi mencionado anteriormente, todas as posições calculadas pelo Labview relacionadas à posição das peças e onde as mesmas devem ser colocadas são feitas em relação à referência da câmera, e não ao robô, desta forma, torna-se necessário descobrir a posição e orientação da peça em relação ao robô, sabendo a posição da câmera e a posição da peça em relação à câmera. As figuras abaixo foram retiradas do site da fabricante do robô e descrevem muito bem o problema e qual a solução que poderá ser adotada, com a ressalva de que, nesse projeto em questão, o ponto de referência será o centro da câmera, pois é com base nele que o Labview realiza os cálculos necessários.

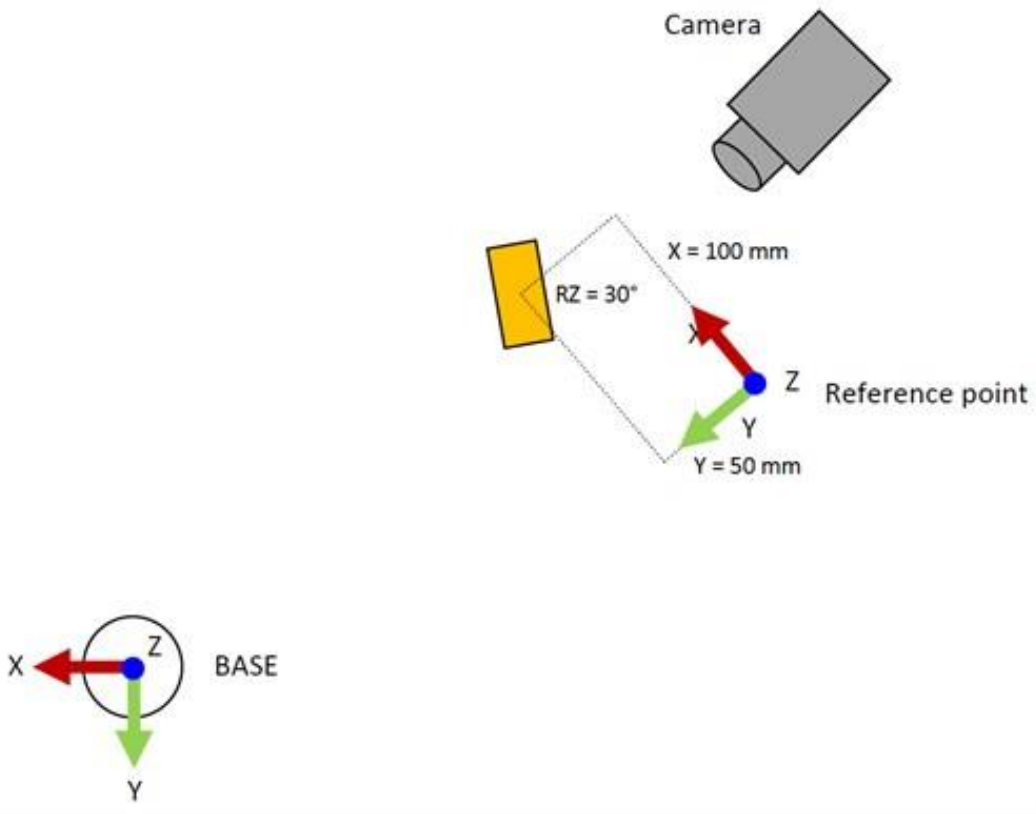


Figura 17 - Problema da Transformada (imagem retirada de: https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/simple-vision-system-23871/)

Na imagem acima tem-se o problema enfrentado na aplicação do dominó, na qual a câmera enxerga uma peça com uma posição e orientação baseado no seu sistema de referência, entretanto, o robô que está posicionado em outro sistema de referência

precisa pegar essa peça e para isso utilizará o conceito e comando apresentado na imagem abaixo:

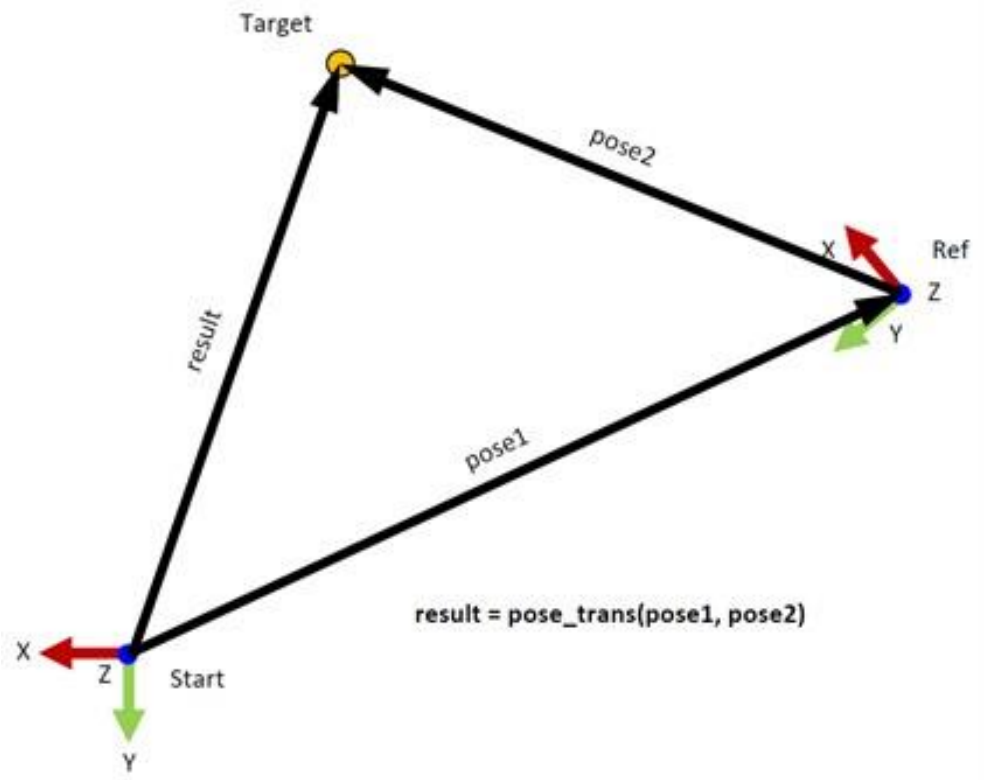


Figura 18 - Solução da Transformada (imagem retirada de: https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/simple-vision-system-23871/)

Conhecendo-se a posição do centro da câmera em relação ao robô, é possível utilizar o comando pose_trans passando como parâmetros esta posição e a posição calculada pelo Labview para obter por fim a posição e orientação da peça em relação ao robô.

2.6.1. Calibração da Posição das Câmeras em Relação ao Robô

Para realizar a calibração das câmeras em relação ao robô utilizou-se uma peça com uma marcação de um ponto pequeno no seu centro, como pode ser visto na imagem a seguir:

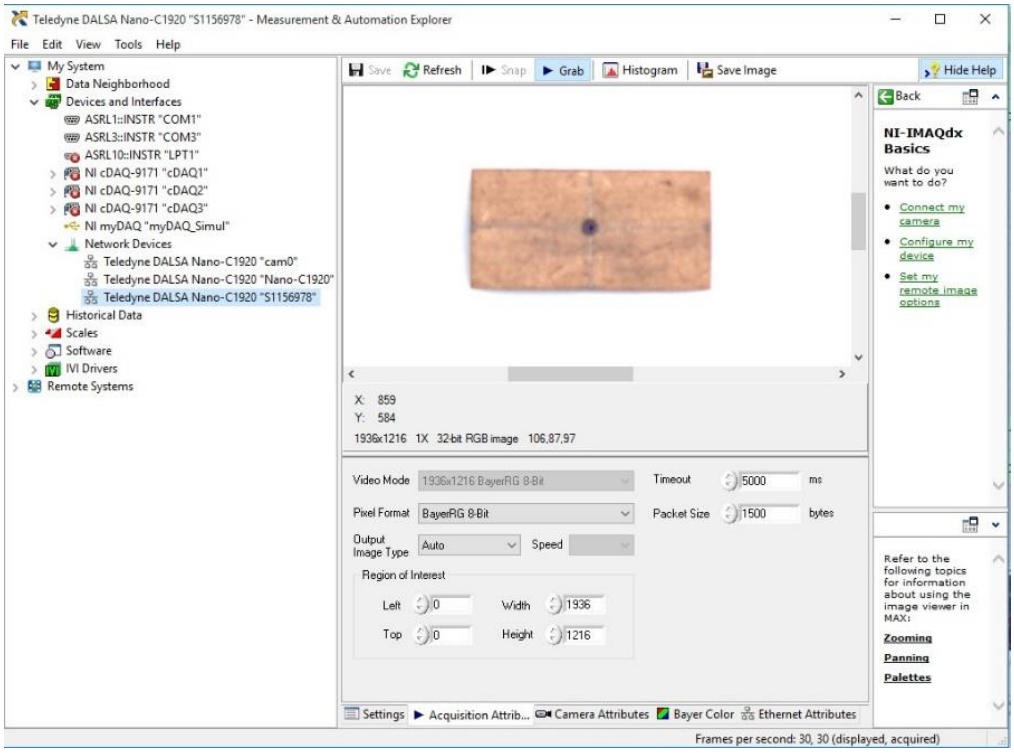


Figura 19 - Imagem da tela utilizada para calibração

Este ponto foi posicionado exatamente onde a câmera indicava as coordenadas 0,0 e em seguida posicionou-se o centro do TCP do robô no mesmo ponto, como pode ser visto nas imagens abaixo.

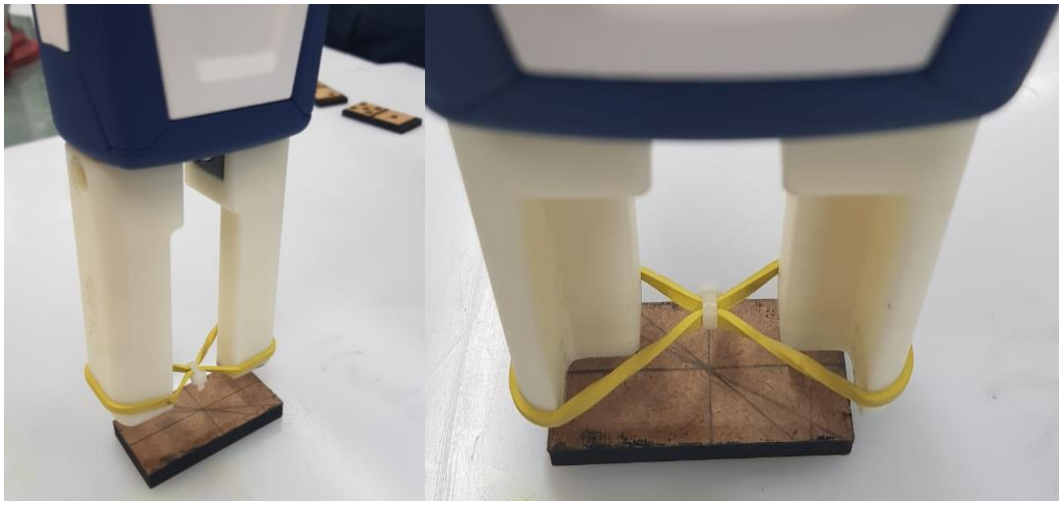


Figura 20 - Calibração das posições das câmeras

Como a altura das peças e da mesa é fixa, foi necessário calibrar somente as coordenadas x e y, podendo manter Z constante na aplicação. Esse processo foi realizado para as duas câmeras e os pontos obtidos para calibração foram definidos como um Feature no programa, chamados de PCam_Hand e PCam_Board, com isso, é

possível realizar as transformações mencionadas anteriormente utilizando essas poses e os valores recebidos pelo Labview.

2.7. Correção de Posição para Dobrões

Devido à adaptação realizada na garra, mencionada no tópico 2.4.1, há uma diferença no posicionamento da garra quando a peça é um dobrão ou não, que ocorre devido ao modo como a peça deve ser colocada no tabuleiro para cada caso. Quando a peça é normal, ela deve ser colocada como demonstrado na Figura 4, ao passo que quando a peça é um dobrão, ela deve ser colocada como pode ser visto na Figura 5, o que muda as coordenadas de pega e solta.

Para resolver este problema, definiu-se como padrão uma peça não dobrão, desta forma, as coordenadas passadas pelo software de identificação de peças será sempre considerando que a peça não é um dobrão, e, quando a peça for efetivamente um dobrão, o software do robô corrigirá utilizando um simples conceito de geometria.

A imagem abaixo representa o sistema padrão, envolvendo as coordenadas de um sistema qualquer e o robô utilizado.

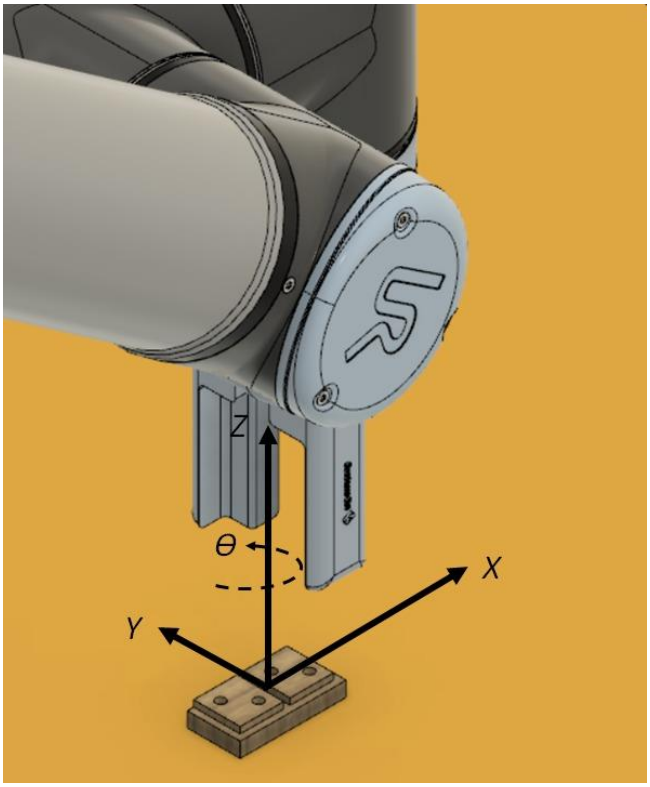


Figura 21 - Sistema peça - robô

É importante ressaltar que para esta demonstração será considerado o ângulo θ como o ângulo entre a peça e o eixo X.

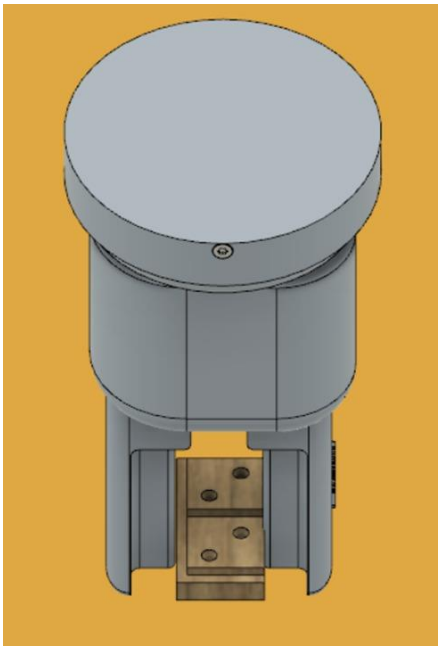


Figura 22 - Caso 1 - Não dobrão

Na figura acima, tem-se um caso padrão, na qual a garra pega uma peça que não é um dobrão. Já a figura abaixo tem como objetivo demonstrar que, caso a peça seja um dobrão, não é suficiente apenas girar a garra 90°, pois, devido a limitação de curso mencionada anteriormente, a garra precisa utilizar seu flanco maior para pegar um dobrão.

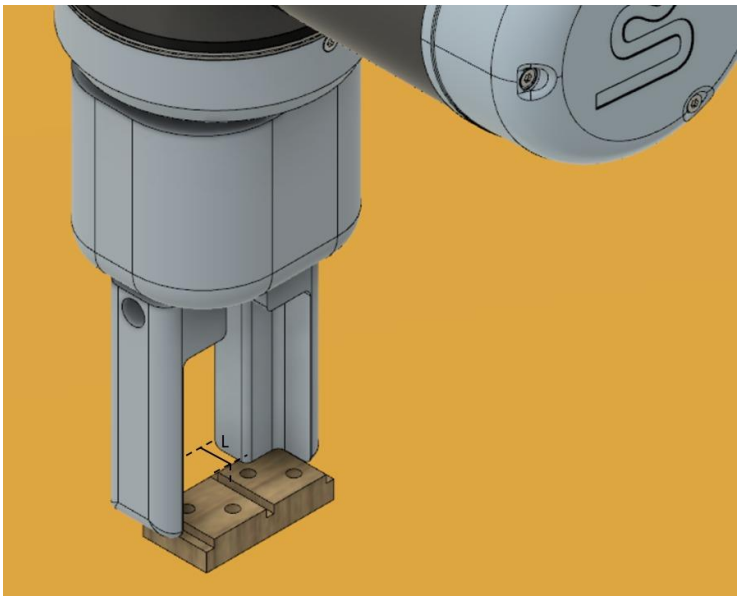


Figura 23 - Problema do dobrão

Para que a garra pegue o dobrão de maneira adequada, deve-se, além de girar 90° do ângulo enviado pelo software de imagem, deslocar-se nos eixos X e Y uma quantidade calculada, levando em consideração a medida “L” demonstrada na figura anterior, a fim de se obter o caso demonstrado na imagem abaixo.

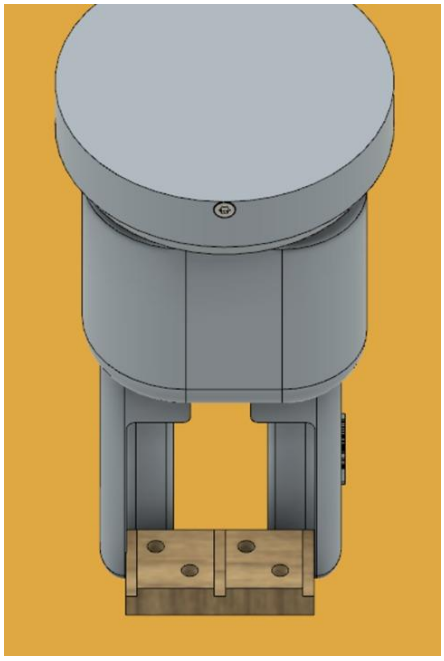


Figura 24 - Caso 2 – Dobrão

Na figura a seguir, pode ser entendido a relação geométrica utilizada para resolver o problema do dobrão. Vale lembrar que no caso estudado o deslocamento no eixo Z não interfere na análise, portanto serão utilizados apenas os eixos x e y referentes ao sistema de coordenadas da base do robô.

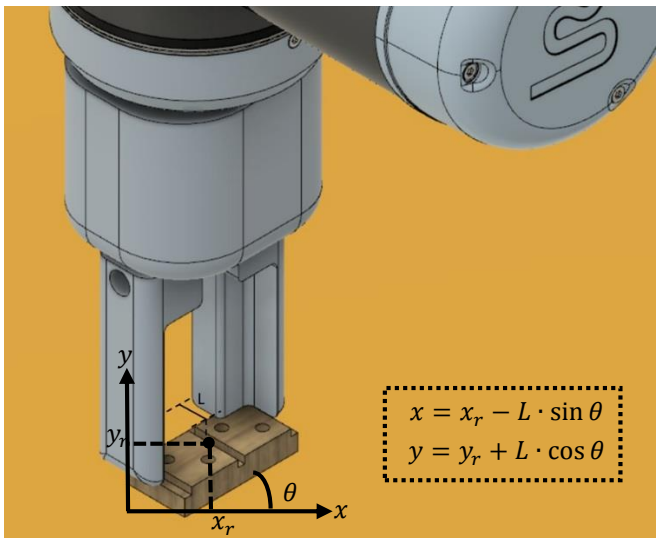


Figura 25 - Análise Geométrica

Com as equações demonstradas acima é possível obter, a partir de em uma posição e de uma orientação recebidas pelo software com base em uma peça normal, a posição e orientação que a garra deverá realmente atingir para que seja possível pegar um dobrão. Essa lógica foi implementada no software do robô através de um Script simples, resolvendo o problema apresentado.

3. Visão de máquina

Dando agora uma pequena introdução à parte de visão de máquinas deste projeto, o objetivo primário desta seção consiste em montar e programar um sistema de visão capaz de identificar as peças presentes tanto no tabuleiro do jogo quanto na região onde estariam as peças da mão do robô, reconhecer quais das peças podem ser selecionadas para a jogada e definir as posições e orientações de pega e solta corretas para serem enviadas ao robô.

Os tópicos a seguir discutem, primeiramente, as soluções encontradas na parte de estruturas e equipamentos para facilitar a aquisição e o processamento das imagens, e em seguida, o desenvolvimento da lógica de programação, focando-se em alguns pontos chaves que possibilitaram de fato o funcionamento eficaz do sistema.

3.1. Idealização e construção do sistema de visão

O sistema de visão empregado apresentou uma série de desafios ao longo do seu processo de concepção, seja pelas limitações de tempo na execução do projeto, seja pelas limitações de materiais e equipamentos. Não houve, por exemplo, a oportunidade de dimensionar de forma mais específica as câmeras e os conjuntos de lentes, tendo um conjunto de aquisição de imagens fixo para todos os grupos de projeto. Outro detalhe limitante foi a escassez de materiais para a construção das estruturas que sustentariam as câmeras e a iluminação, resultando em imprecisão na fixação das câmeras e em limitações na área de visão e na de atuação do manipulador robótico.

Diante do cenário descrito acima, foi concebido o seguinte sistema: foram utilizadas duas câmeras Genie Nano-C1920, cada uma com uma lente de distância focal de 12mm e abertura de íris de 1.4, presas cada uma em um braço em forma de L invertido junto a uma iluminação de LED branca difusa em forma de barra. Uma dessas estruturas foi posicionada sobre a região do tabuleiro, deixando uma distância de 63cm

de distância entre a mesa e a lente da câmera, a maior distância possível no caso, considerando o ajuste de foco da lente. A outra estrutura foi colocada sobre a área dedicada à mão do robô e, uma vez que o braço suporte teve que ser montado com barras mais curtas, na falta de matérias melhores, a distância entre a mesa e a lente da câmera ficou limitada a 43cm.

Ambas as câmeras, assim como a iluminação do tabuleiro, são alimentadas por uma fonte contínua de 24V, enquanto a iluminação da mão do robô é alimentada com 19V, visando reduzir o reflexo provocado na mesa devido à sua maior proximidade. Por último, a comunicação entre as câmeras e o programa de identificação no labview foi realizada via cabo de rede e configurada na programação usando o seguinte script:

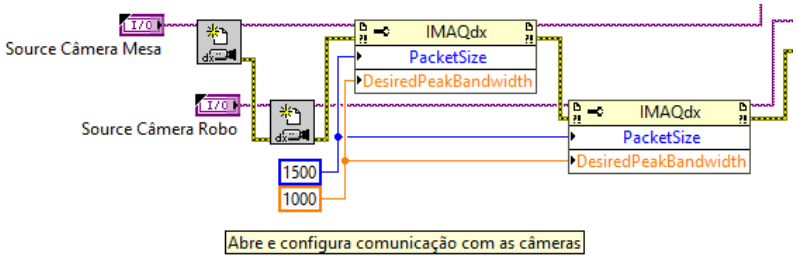


Figura 26 – Configuração da comunicação do labview com as câmeras

Nessa disposição, as imagens tiradas pelas câmeras acabaram apresentando certo efeito de perspectiva ou paralaxe dependendo da posição das peças na mesa, visto a abertura focal grande da lente, a proximidade das câmeras da mesa e a leve inclinação dos suportes em relação à vertical, o que infelizmente é algo que não pode ser corrigido mediante o prazo de entrega do projeto. Como forma de compensação, buscou-se outras formas e recursos para melhorar a qualidade das imagens e facilitar a identificação das peças.

A iluminação instalada no sistema já ajuda a melhorar a visibilidade das peças na mesa, porém, observou-se que as câmeras, sem nenhuma configuração pré-definida, possuíam uma coloração bem puxada para o verde, deixando as imagens escuras com detalhes menos distintos. Alterando as configurações de bayer pelo software do Labview foi possível ajustar o balanço de branco das imagens, o que tornou mais nítido o contorno das peças.

Outra observação realizada foi que as peças comuns de dominó mostram certa dificuldade de serem detectadas pelo programa de identificação com um nível alto de correspondência, independentemente do método usado. Possivelmente, ao que se concluiu após alguns testes práticos, essas peças não possuem muitos detalhes significativos que levariam a uma identificação mais concreta. Em resposta a isso, foi desenhado e fabricado um conjunto inteiro de peças de dominó com o desenho de um E marcado nelas, como pode ser visto na imagem abaixo, com uma peça exemplo:

Figura 27 – Desenho de uma peça de dominó personalizada

Isso por si só já ajudou bastante na identificação, como será mostrado nas próximas sessões. Um último fator na idealização desse sistema vem do fato de as peças personalizadas terem sido feitas de MDF, cuja cor não contrasta tão bem com o forro esverdeado das mesas do laboratório, detalhe esse facilmente resolvido com o uso de placas brancas e foscas como fundo para as imagens das câmeras. As imagens a seguir apresentam o Setup final do projeto.

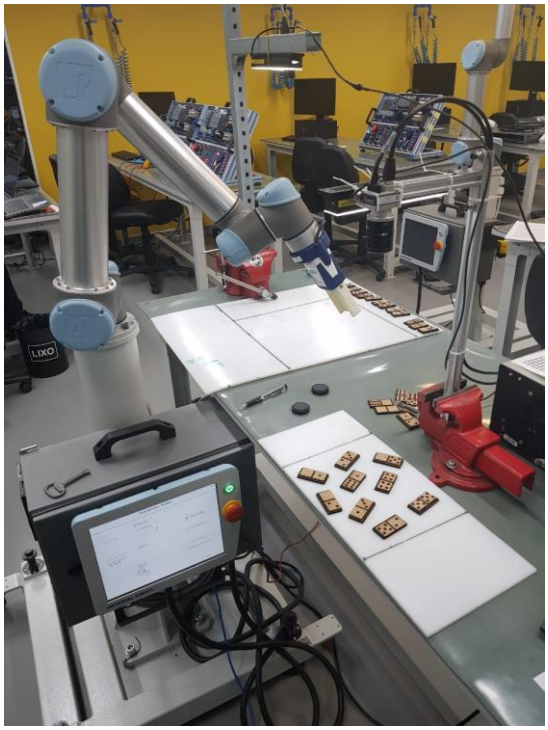


Figura 28 - Setup do Projeto - vista 1

Figura 29 - Setup do projeto - vista 2

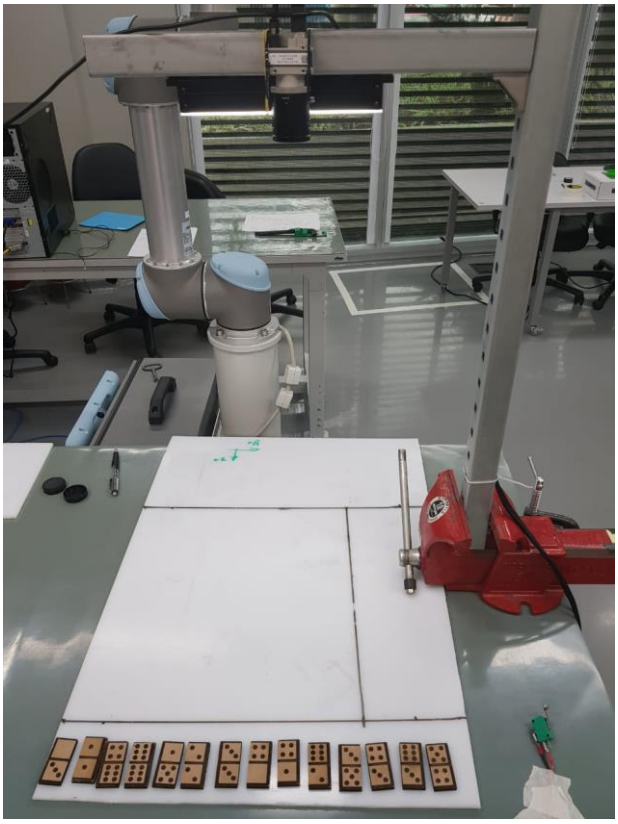


Figura 30 - Setup do projeto - Vista 3

3.2. Como identificar uma peça

Estabelecidas as estruturas para a aquisição de imagens, pode-se então começar o desenvolvimento da lógica de identificação e processamento dos dados na interface do Labview. Antes de qualquer outra coisa, é necessário saber como identificar cada uma das peças em ambas as imagens, sem no processo reconhecer erroneamente peças formadas pela junção das peças no tabuleiro ou peças que não existem no meio da imagem. Existem alguns métodos que podem ser usados para tal fim, mas o que se mostrou mais eficaz foi o por correspondência de padrão, no qual é definida uma imagem de base com aquilo que se deseja reconhecer:

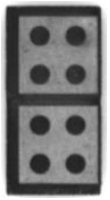


Figura 31 – Padrão para identificação por correspondência

A partir dessa base, o programa vasculha a imagem na qual se deseja identificar algo, procurando por correspondências pixel a pixel. Os resultados encontrados que estiverem acima de um valor mínimo de concordância são considerados como a saída da função. A maioria das peças no jogo provavelmente não combinará perfeitamente com o padrão apresentado pelo número de pontos diferente e pelo efeito de perspectiva comentado anteriormente, mas notou-se que ignorar os pixels dos pontos do padrão e considerar os pixels claros dentro e ao redor da peça, assim como a presença do E no entorno da peça, aumenta muito as chances de identificação de peças com correspondência alta.

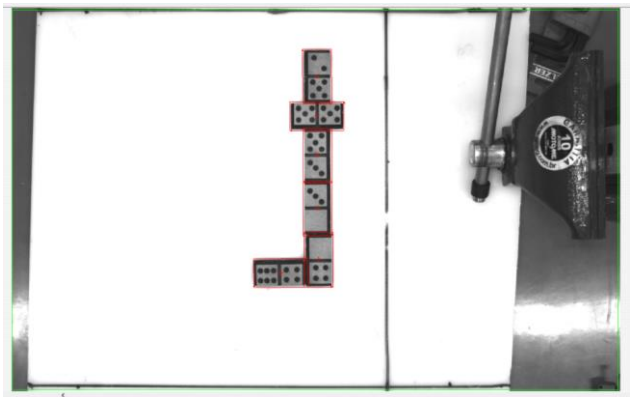
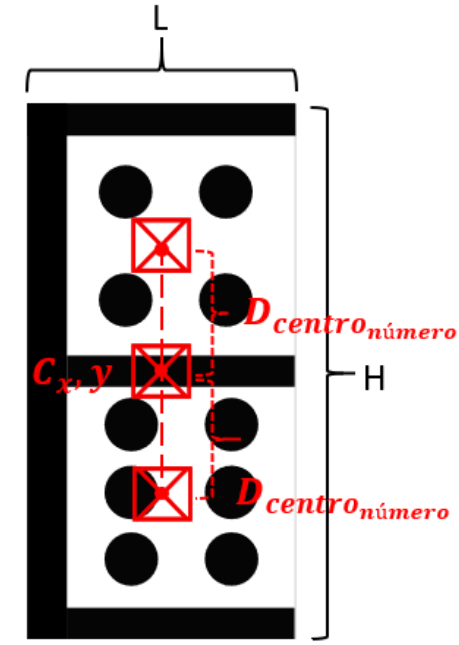


Figura 32 – Identificação das peças por correspondência de padrão

Após o processo de identificação das peças, a função retorna uma lista em que cada item é um conjunto de informações (cluster) sobre a peça identificada, como as coordenadas do centro da peça, o retângulo que envolve a peça, o ângulo de rotação da peça em relação ao padrão, dentre outras coisas que serão úteis ao longo do programa.

3.3. Como identificar qual peça está na ponta de um jogo



$L = comprimento\ peça = 25mm$
$H = altura\ peça = 50mm$
$D_{centro_{número}} = 12{,}5mm$

Figura 33 - Parâmetros das peças do jogo de dominó

Primeiramente toma-se como característica fixa e única uma distância limite que indica que uma peça de dominó esta junta de outra. Isso é possível visto que todas as peças possuem as mesmas dimensões. Além disso, verificou-se que independente da configuração da distribuição das peças no jogo, a distância máxima entre seus centros seria o tamanho da altura de uma peça e a menor distância seria metade da altura de uma peça mais $\frac{1}{4}$ de sua altura. Esta distância é nomeada como D_{lpj} $(distância\ limite\ peça\ junta)$. Os dois possíveis casos limites no jogo de dominó são:

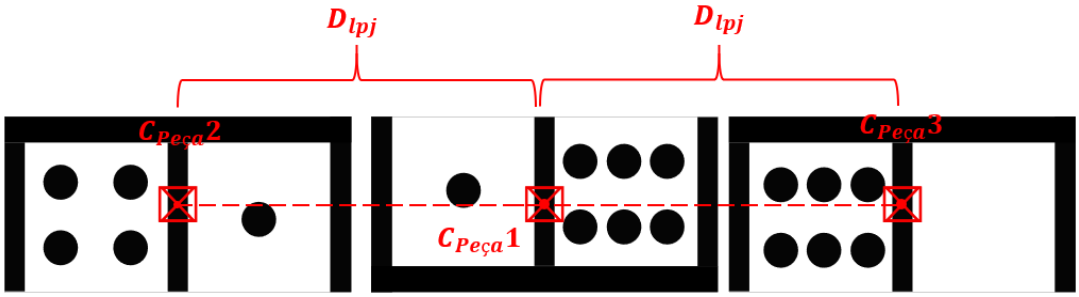


Figura 34 - Caso 1: maior distância possível entre os centros das peças de dominó até o centro da peça verificada $C_{peça}1$

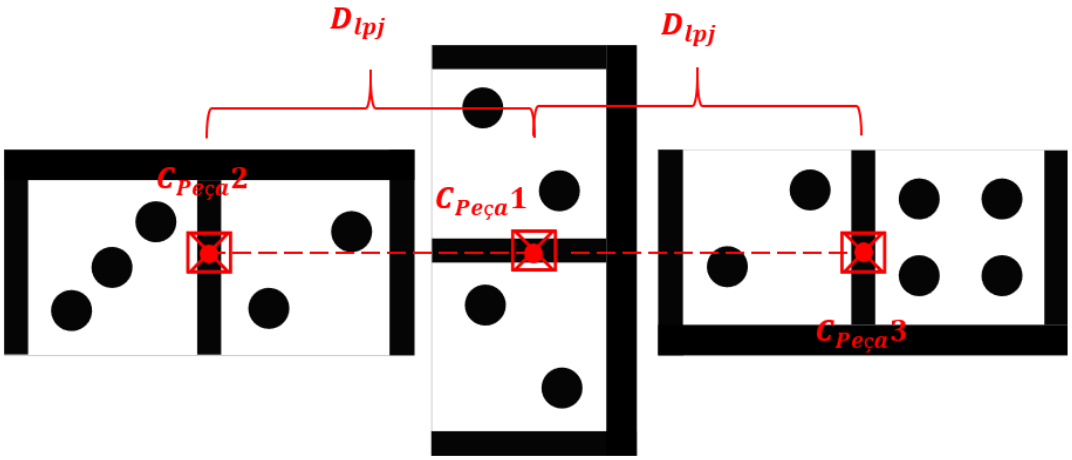


Figura 35 - Caso 2: menor distância possível entre os centros das peças de dominó até o centro da peça verificada $C_{peça}1$

Dado que o software em Labview consegue identificar as coordenadas dos centros de todas as peças do tabuleiro, ele consegue levantar todas as distâncias dos centros das peças em relação a peça selecionada para verificação. Compara com o D^*_{lpj} e verifica quantas peças estariam ligadas a ela. No caso sempre haverá um ou dois D^*_{lpj}. Quando há apenas um D^*_{lpj} é possível inferir que esta peça selecionada para verificação está na ponta.

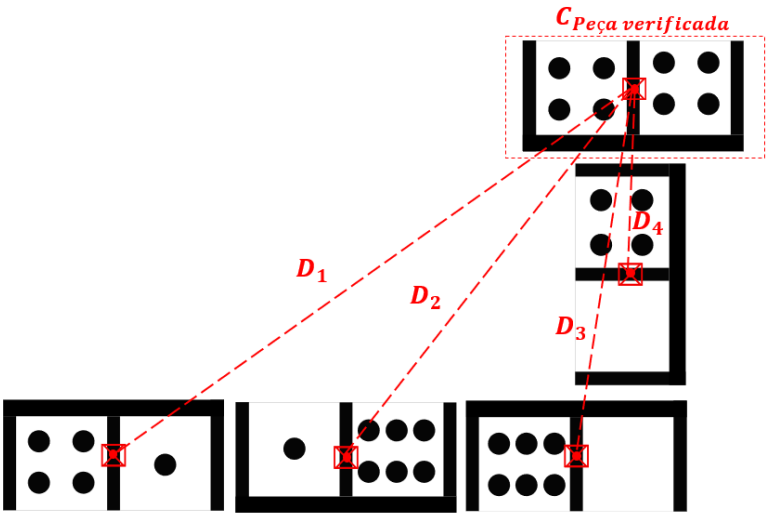


Figura 36 – Processo de verificação peça ponta

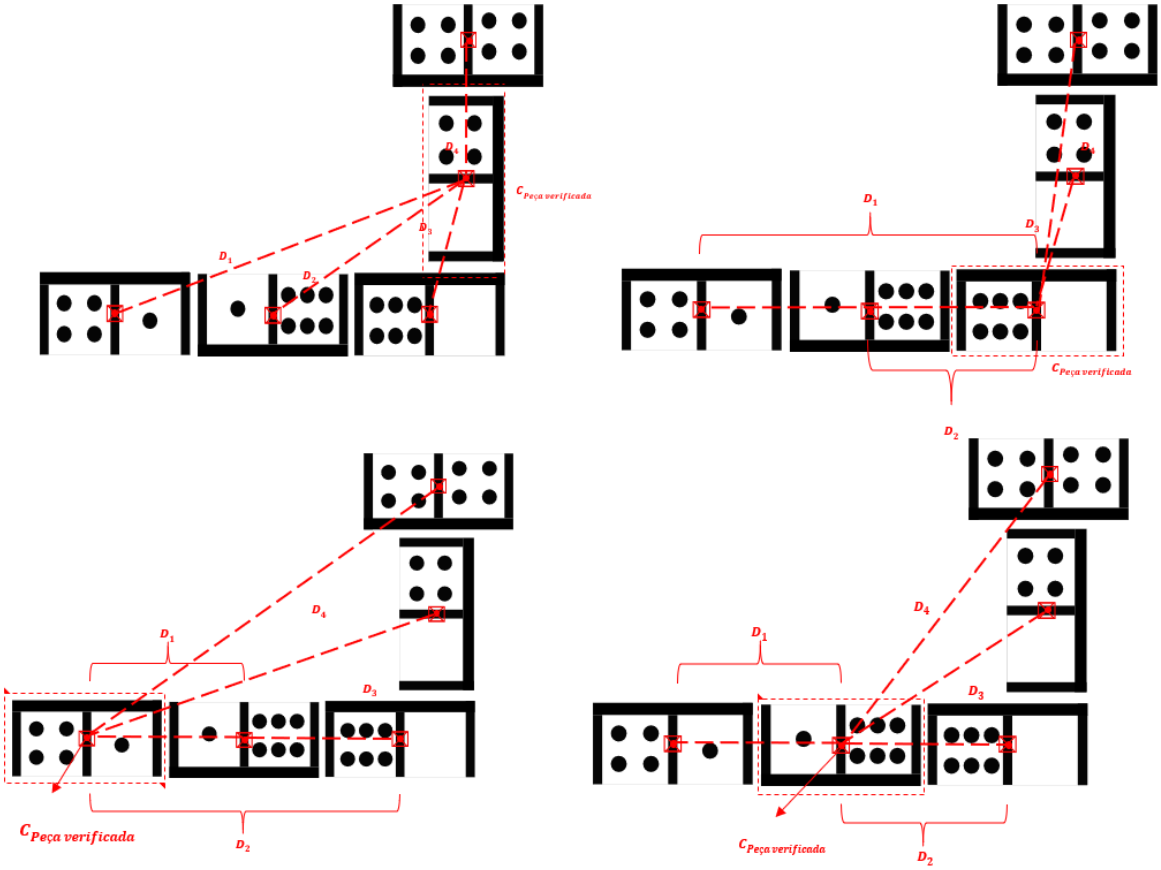


Figura 37 – Processo de verificação peça ponta para várias peças na mesma configuração

3.4. Como identificar qual o número que está na ponta de um jogo

O software após detectar a peça da ponta roda a rotina de identificar o número da ponta. Essa rotina basicamente levanta a distância dos centros números até o centro

da peça anterior e compara os tamanhos. O $C_{número}$ que apresentar a maior distancia, indica qual é o número da ponta, como é possível observar na figura abaixo:

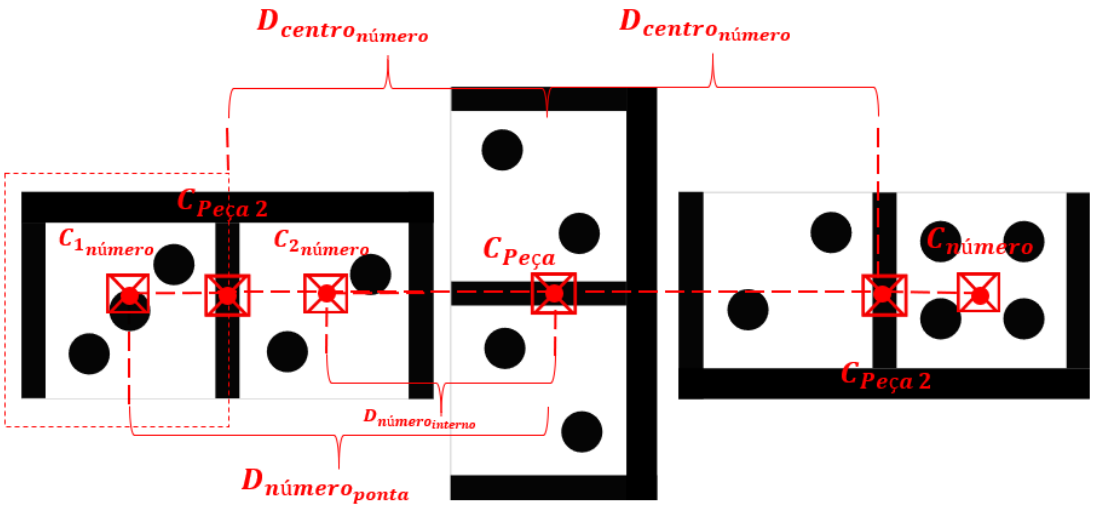


Figura 38 – Processo de verificação número ponta

Não se sabe inicialmente as coordenadas dos centros dos números da peça, mas podem facilmente serem calculadas a partir do retângulo que envolve a peça, obtido durante a identificação das peças na sessão 3.2, definindo primeiro um ponto médio em um dos lados mais logos do retângulo e depois mais dois pontos médios em relação ao primeiro e os outros dois vértices do retângulo. A figura a seguir exemplifica o método descrito:

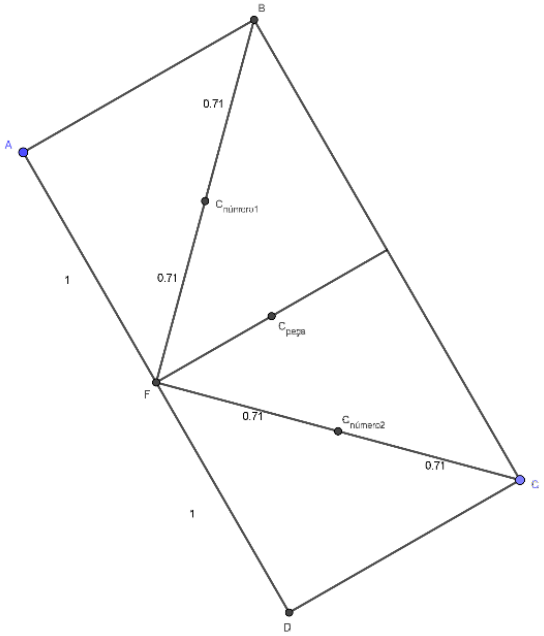


Figura 39 - Esquema para cálculo dos centros números

3.5. Como identificar os números da peça

Uma vez que as peças relevantes para a execução da jogada do robô foram identificadas nas sessões anteriores, busca-se agora o valor numérico de cada conjunto de pontos de cada uma dessas peças. Este processo por sua vez será dividido em duas etapas, a primeira voltada para a manipulação dos dados atuais da partida, visando facilitar a segunda etapa, que é a da identificação propriamente dita.

3.5.1. Construção das regiões de interesse dos números para identificação

Após analisar as informações obtidas no processo de identificação da sessão 3.2 e realizar alguns testes, concluiu-se que a forma mais eficaz para a verificação dos números das peças seria efetuar um recorte da imagem original contendo apenas os pontos de uma das metades da peça, podendo assim focar o processamento da imagem em uma região bem específica.

Na interface do labview, esse recorte pode ser feito usando uma máscara, que depende de uma região de interesse (ROI) da imagem, que por sua vez depende da posição e da orientação da peça, o que a princípio não seria uma limitação levando em conta que a identificação das peças já retornaria um retângulo que envolve a peça na imagem, contendo assim essas informações. Logo, seria possível utilizar esse retângulo no recorte, uma metade de cada vez. Contudo, construir essa região de interesse mostrou ser mais penoso do que o esperado, visto que o retângulo da identificação é um tipo de variável que não pode ser convertido diretamente para ROI.

A solução encontrada para contornar esse contratempo consiste em converter o retângulo identificado, uma lista de pontos correspondentes aos vértices do retângulo, em uma variável do tipo Retângulo, um formato especial de variável do módulo de visão de máquinas que pode ser convertido em ROI, composto por cinco valores que podem ser interpretados como as coordenadas dos pontos da diagonal do retângulo e o ângulo de rotação dele em relação à horizontal.

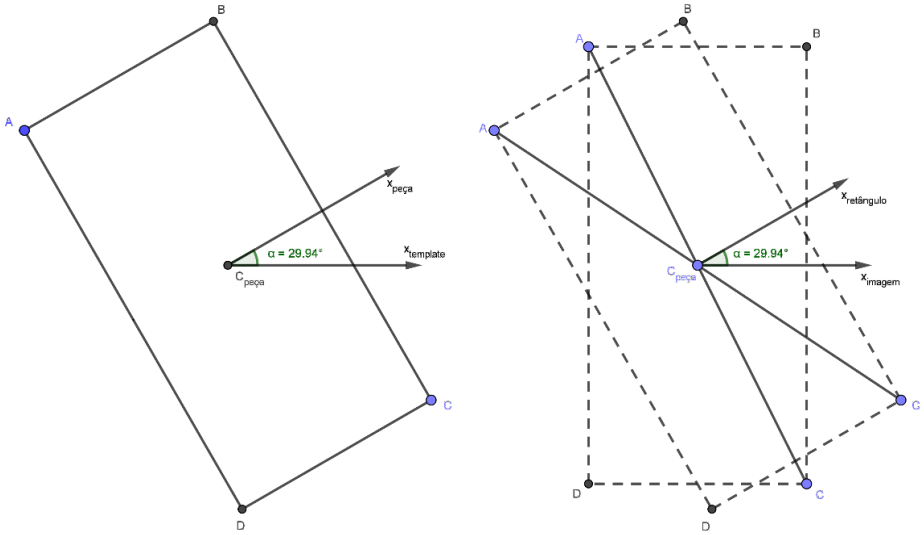


Figura 40 - – Representação retângulo identificado (à esquerda) e variável retângulo (à direita)

A variável Retângulo emprega a seguinte lógica para definir a sua orientação: as coordenadas definidas na sua declaração são as das extremidades da sua diagonal, do ponto mais próximo da origem da imagem para o mais distante, considerando o retângulo com rotação nula em relação ao eixo x da imagem e o ângulo de rotação define o quanto ele irá rotacionar em relação ao eixo x em torno do ponto médio da sua diagonal.

Dito isso, para obter uma região de interesse para cada conjunto de pontos da peça, deve-se primeiro calcular o comprimento referente à diagonal da metade do retângulo identificado, assim como o ângulo que essa diagonal compreende com algum dos lados do retângulo:

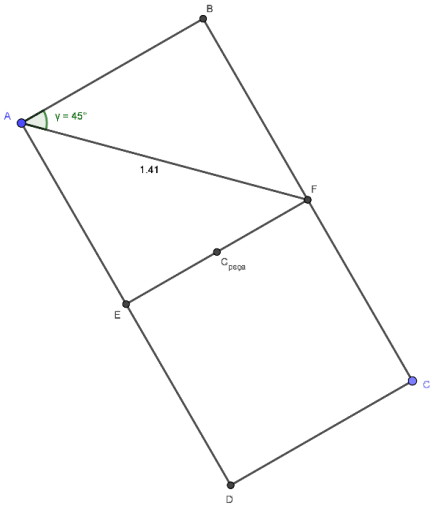


Figura 41 - Conversão retângulo identificado para ROI (parte 1)

Em seguida, são definidas as diagonais das duas áreas em relação aos centros dos números, empregando os valores adquiridos no passo anterior:

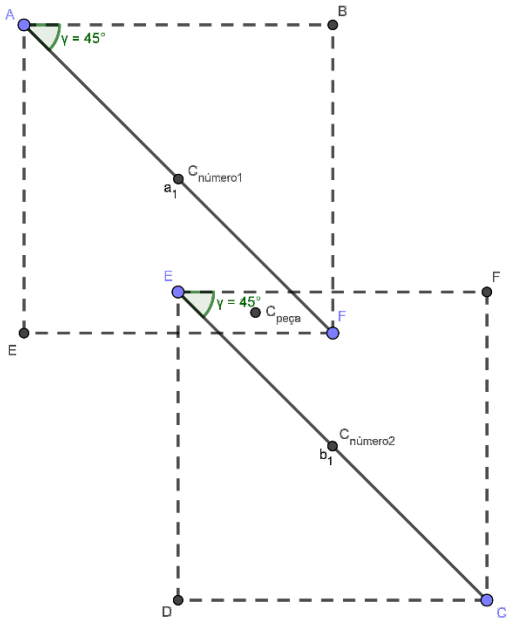


Figura 42 - Conversão retângulo identificado para ROI (parte 2)

Com as diagonais definidas, basta então rotacionar ambas em relação ao eixo x da imagem, considerando o mesmo ângulo obtido durante a identificação das peças, uma vez que o padrão empregado possui seu eixo x na mesma direção do da imagem original:

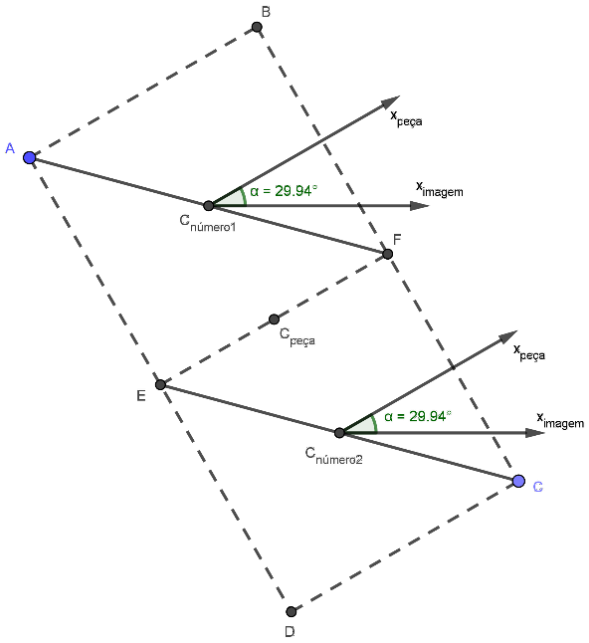


Figura 43 – Conversão retângulo identificado para ROI (parte 3)

3.5.2. Identificação dos números

Obtidas as regiões de interesse referentes a cada conjunto de pontos da peça, a identificação se inicia convertendo cada uma delas em uma máscara e aplicando-a na imagem original para recortar apenas metade da peça:

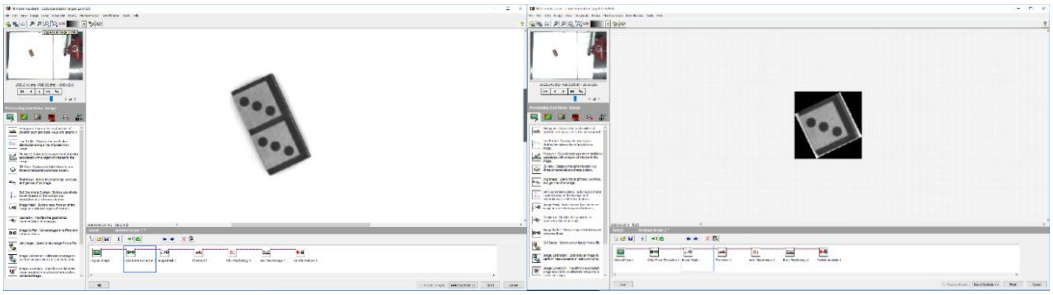


Figura 44 – Recorte do número da peça

A imagem recortada pode então ser processada, aplicando um threshold que deixe aparente os pontos da peça:

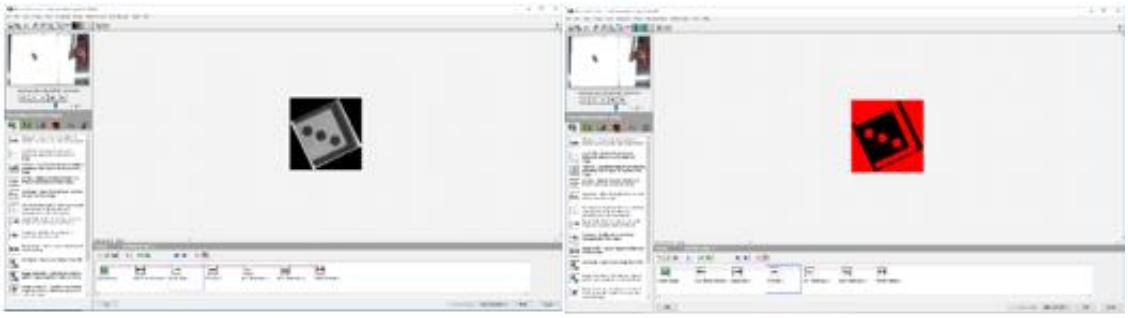


Figura 45 - Aplicação do threshold

Por sempre dividir a peça ao meio, o retângulo que envolve o número normalmente passa por cima da parte escura do E no meio da peça, e como o recorte da imagem geral gera regiões pretas nas bordas da imagem resultante fora desse retângulo, ao aplicar o threshold, o recorte do E na imagem tenderá a ficar conectado às regiões pretas. Em resumo, isso significa que ao aplicar um filtro à imagem que remove tudo que estiver conectado às bordas da imagem, sobrará apenas os pontos da peça e, possivelmente, algumas impurezas pequenas:

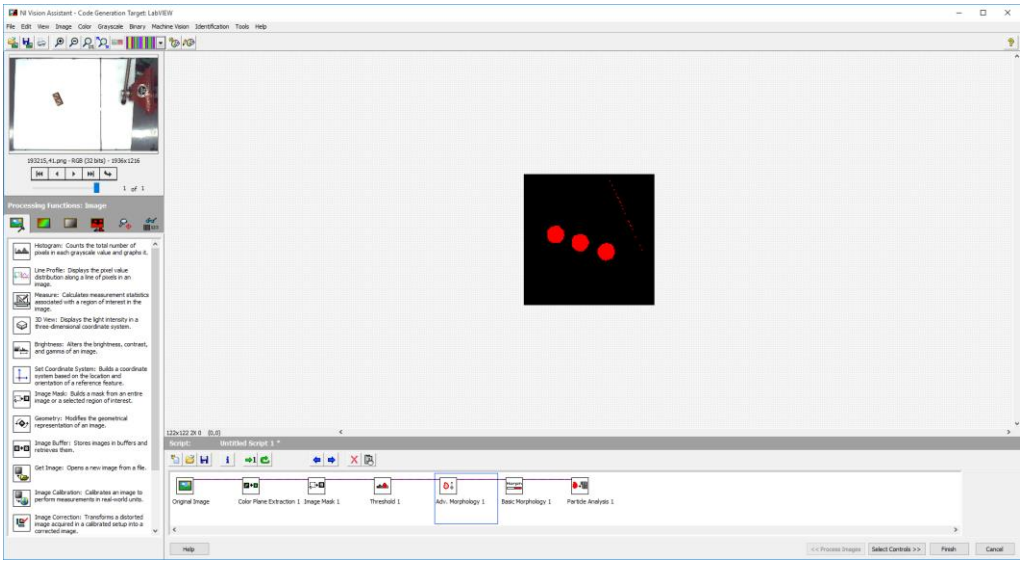


Figura 46 – Recorte após conteúdo das bordas ser retirado

Para eliminar eventuais impurezas que possam sobrar na imagem, é aplicado um filtro do tipo Open (que executa um processo de erosão e depois de dilatação na imagem):

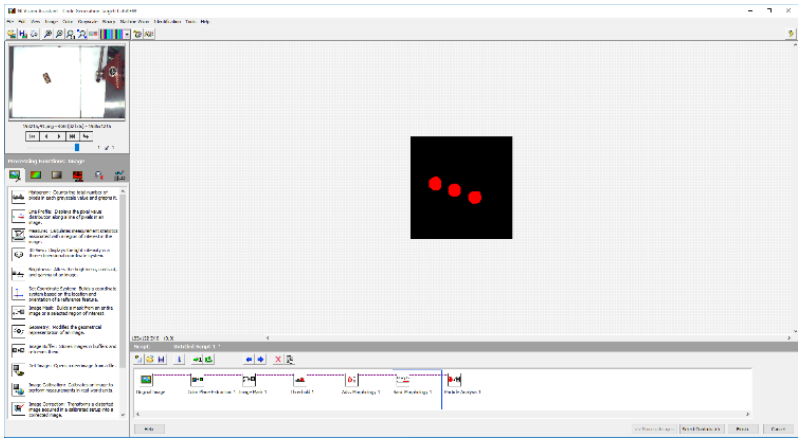


Figura 47 – Recorte após resíduos terem sido eliminados

Por último, uma função de análise de partículas é executada para contar quantas partículas (pontos), sobraram após os demais processos, obtendo assim o valor números de quantos pontos essa parte da peça possui:

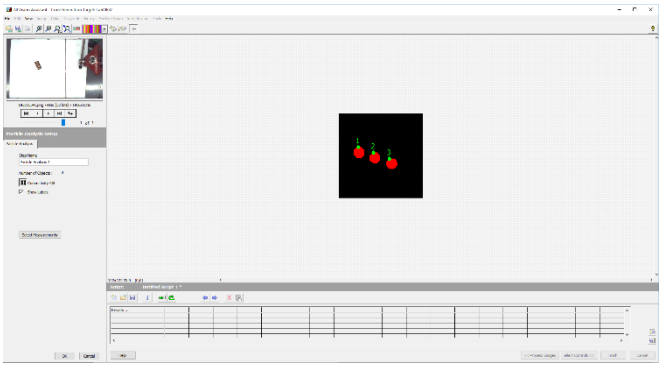


Figura 48 – Pontos da peça contados

3.6. Como escolher a peça certa para jogar

Até este ponto do projeto, foram obtidas todas as informações relevantes referentes à situação atual da partida, incluindo posição, orientação e números de cada uma das peças da mão do robô, além de onde podem ser colocadas no tabuleiro do jogo. Com esses dados à disposição, pode ser definida a lógica para decidir qual peça pode ser selecionada para realizar uma jogada.

Sabendo-se os números das peças da mão do robô e os números nas pontas do jogo no tabuleiro, a forma mais simples de verificar se algumas dessas peças pode ser jogada seria vasculhar essa lista de peças, procurando por uma correspondência com um dos números das pontas do jogo. Caso uma ou mais peças sejam identificadas como possíveis jogadas, por limitações de tempo, foi definido que a primeira dessas peças seria a escolhida.

3.7. Como calcular as posições e orientações de pegar e soltar peça

Selecionada a peça a ser jogada, basta então definir as coordenadas e a orientação da peça para o robô poder pegá-la, assim como a posição e a orientação desejadas da peça para que ela seja colocada corretamente no tabuleiro.

3.7.1. Calibração das câmeras

Os sistemas de coordenadas das câmeras, do modo como são fornecidos pelo software de identificação, não são muito úteis no que se refere a distâncias precisas, sendo que as origens dos sistemas estão sempre na extremidade superior esquerda das imagens, não importa a orientação das câmeras no espaço, além de quaisquer distâncias nas imagens serem medidas em pixels e não em centímetros ou milímetros.

Diante dessa situação, foi desenvolvido um método que permite calibrar as distâncias na imagem e deslocar a origem do sistema de coordenadas para o centro da imagem, facilitando os cálculos de posições relativas às câmeras.

Deslocar a origem consiste apenas em avaliar as dimensões máximas de cada imagem e dividi-las pela metade, adquirindo as coordenadas do ponto central da imagem que podem ser subtraídas das coordenadas normais da imagem, resultando em coordenadas relativas a esse ponto.

O processo de calibração, por sua vez, envolve obter um passo em milímetros equivalente a cada pixel da imagem, utilizando um padrão quadriculado de dimensões conhecidas. O usuário, ao início da execução da VI, é questionado se deseja calibrar as câmeras ou não:

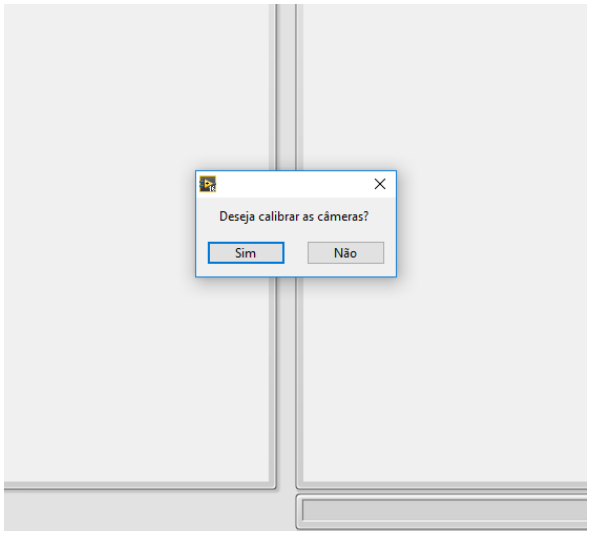


Figura 49 – IHM perguntando se usuário deseja calibrar as câmeras

Caso responda não, o programa carregará a última calibração feita, mas caso responda que sim, o programa mostrará a imagem da câmera da mesa em tempo real, com uma região delimitada onde o padrão deve ser posicionado:

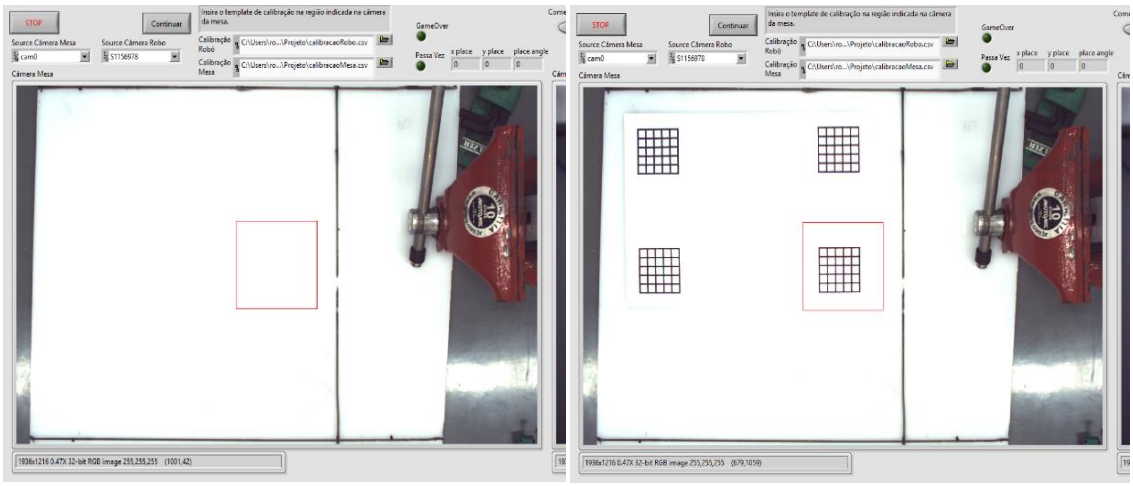


Figura 50 – Posicionamento do padrão de calibração na câmera da mesa

Após posicionar devidamente o padrão dentro da área definida e apertar o botão "Continuar", a última imagem tirada da câmera é então enviada para uma função análise, verificando primeiro as bordas de cada uma das linhas verticais e horizontais da imagem:

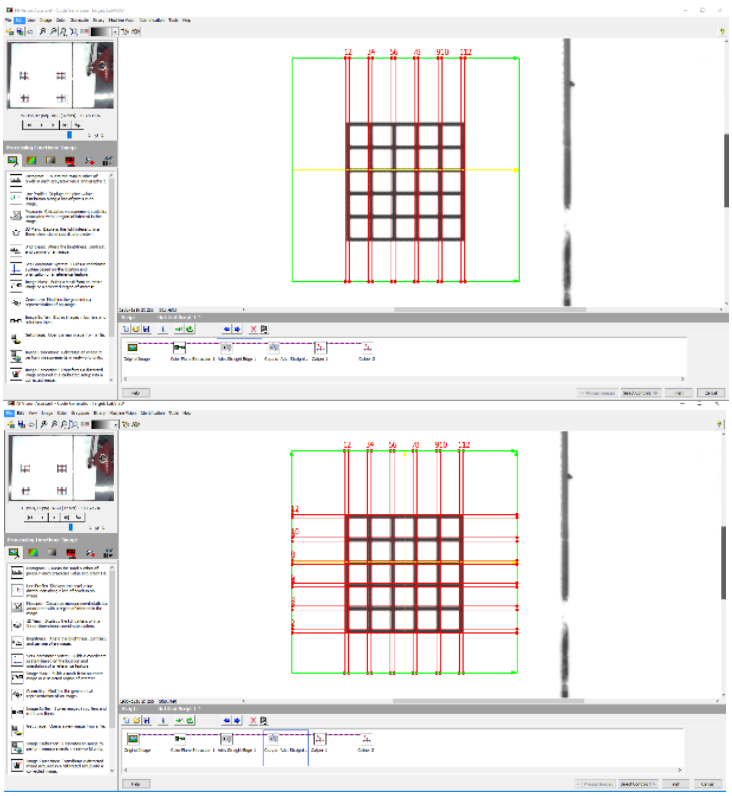


Figura 51 – Detecção das bordas do padrão de calibração

Em seguida, são definidas as retas médias entre as bordas detectadas, para tentar adquirir retas que passem bem próximas do centro das linhas do padrão:

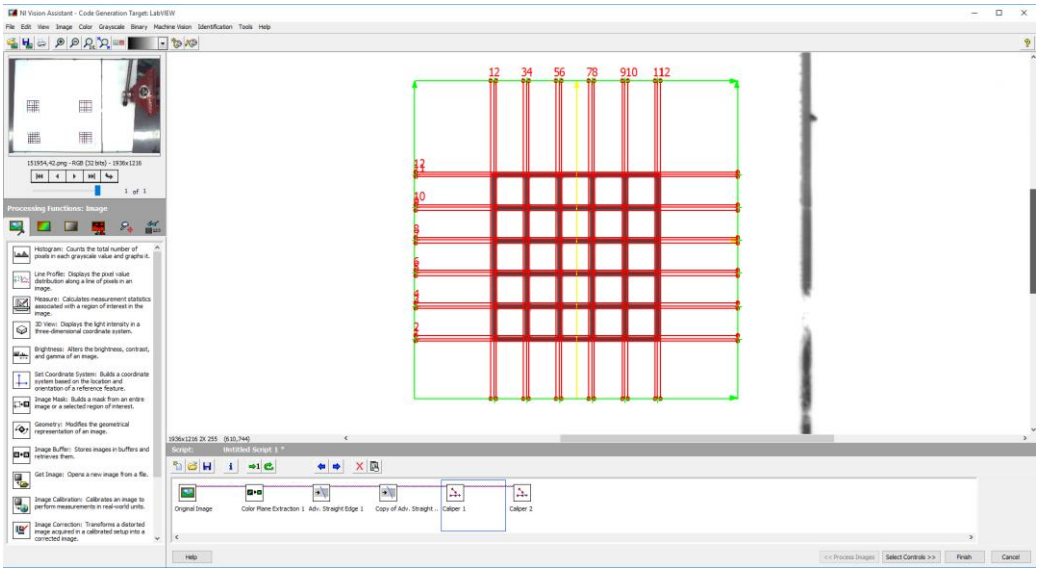


Figura 52 – Definição das retas médias entre as bordas detectadas

E por último, são medidas as distâncias de cada uma das dessas retas médias com a primeira reta identificada em cada direção, ou seja, a distância da primeira reta da vertical com a segunda da vertical, a da primeira com a terceira e assim por diante e depois repetir o mesmo processo com as retas na horizontal:

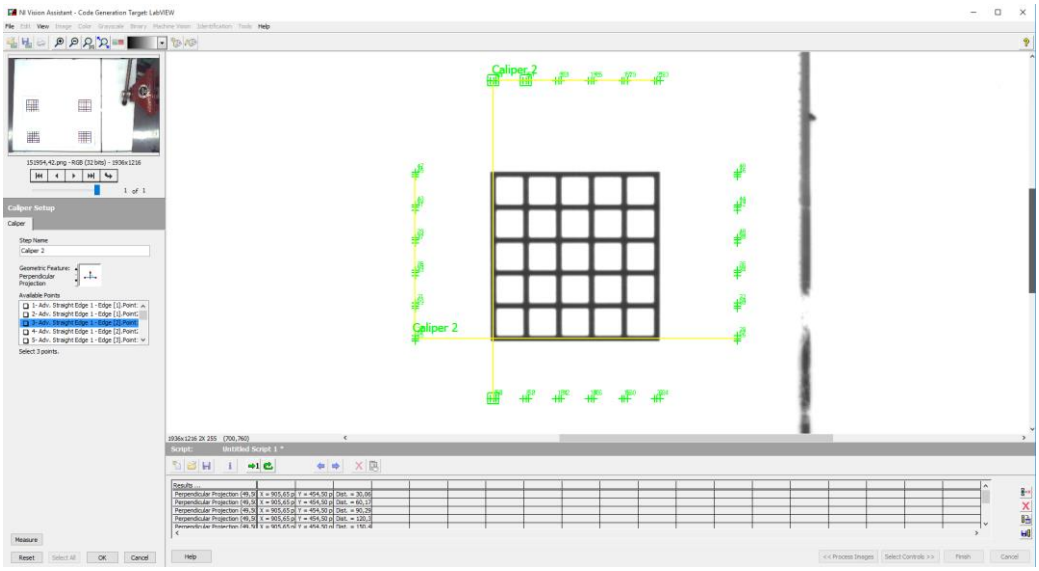


Figura 53 – Medição das distâncias entre as retas médias

Essa função retorna uma lista com as distâncias medidas, as quais são comparadas com as distâncias reais, obtendo assim um valor em milímetros a que cada pixel correspondente na imagem.

Concluída a calibração da câmera da mesa, o processo é repetido para a câmera da mão do robô.

3.7.2. Cálculo do ângulo de orientação

Identificar a orientação da peça por si só se mostrou ser um desafio a parte, uma vez que estariam disponíveis apenas o centro da peça e o centro do número da peça selecionado. O esquema abaixo servirá como referência visual para a explicação de como foi realizado este cálculo.

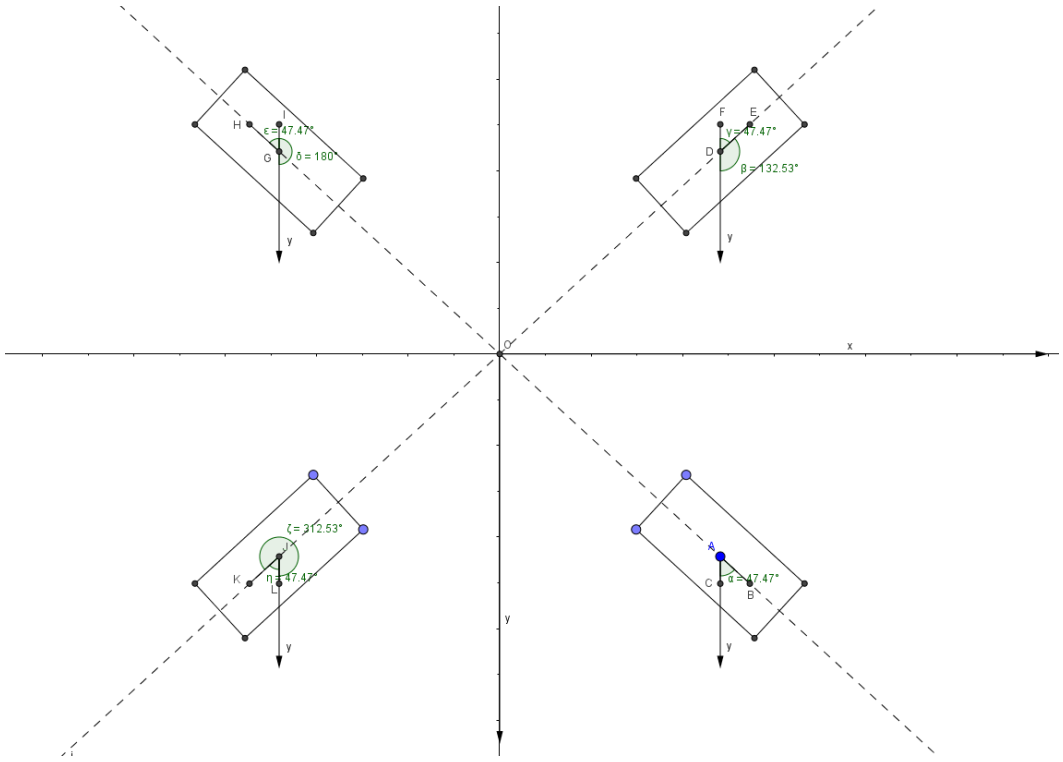


Figura 54 – Esquema para identificar orientação da peça

Os centros das peças não necessariamente estarão alinhados com a origem do sistema de coordenadas da imagem, mas, por questões didáticas, o esquema as representa dessa forma. Primeiramente, constrói-se um novo ponto cuja coordenada no eixo x é a mesma que a do centro da peça e cuja coordenada no eixo y é a mesma que a do centro do número. Dessa forma, o novo ponto indicará sempre a direção do eixo y passando pelo centro da peça, como pode ser visto na imagem na posição entre

os pontos AC, DF, GI e JL. A partir disso, forma-se um triângulo retângulo, por exemplo em ABC, do qual os comprimentos de AB e AC podem ser usados para calcular o ângulo CÂB na seguinte equação:

$$\alpha = \operatorname{acos}\left[\frac{dist(A,B)}{dist(A,C)}\right]$$

Esse processo resulta, segundo a Figura 54, nos ângulos $\alpha, \gamma, \varepsilon$ e η, relativos à vertical. Porém, está claro que todos esses ângulos possuem o mesmo valor, independente de qual quadrante do plano a peça está localizada. Para resolver esse problema, adota-se o sentido positivo do eixo y e o sentido positivo da rotação como anti-horário como base dos cálculos.

Os quadrantes no esquema, por sua vez, não indicam a posição de cada peça no plano, mas sim a direção e o sentido nos quais as peças estão orientadas. Uma peça que esteja no quadrante inferior direito do plano no esquema pode ter quaisquer coordenadas, positivas ou negativas, nos seus centros, desde que ela se mantenha apontada na direção e no sentido desse quadrante. Dito isso, podem ser tiradas algumas conclusões sobre a relação entre os centros de cada peça e a sua orientação a partir da observação da Figura 53:

- Se ambas as coordenadas do centro da peça forem menores do que as do centro do número, o ângulo de orientação é o que foi calculado acima;
- Se somente a coordenada x do centro da peça for menor, o ângulo de orientação é o ângulo suplementar do calculado, ou seja, 180^o menos o calculado;
- Se ambas as coordenadas forem maiores, o ângulo de orientação será 180^o mais o ângulo calculado;
- E por último, se somente a coordenada y for menor, o ângulo de orientação será o ângulo replementar do calculado, ou seja, 360^o menos o calculado.

Essa lógica resultou em uma sub-rotina que servirá de base para as demais operações:

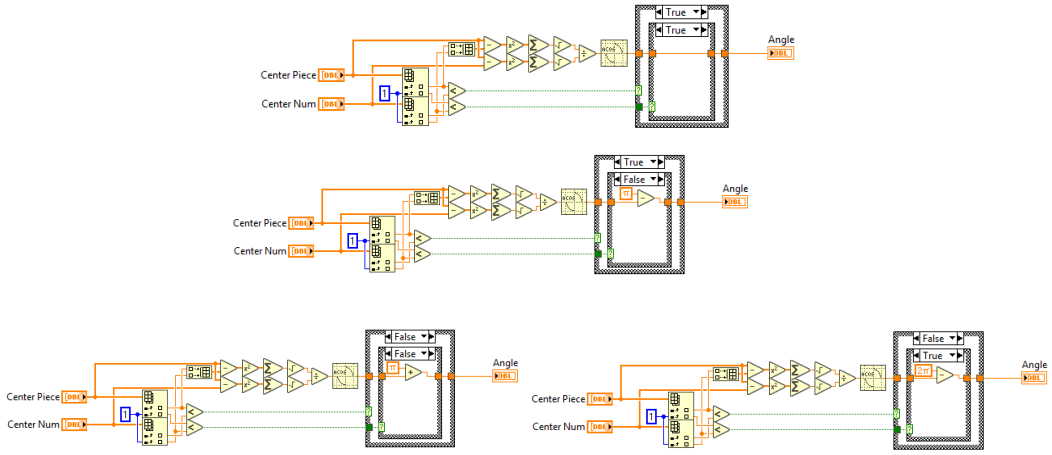


Figura 55 – Sub-rotina para identificação do ângulo de orientação das peças

3.7.3. Cálculo das posições de pegar e soltar

Tratando primeiro da posição de pegar peça, uma vez que o objetivo é mandar a garra do robô exatamente para onde a peça selecionada está na mão do robô, as coordenadas enviadas devem ser as do centro da peça e a orientação a mesma da peça.

Já a posição de soltar peça depende de se a peça selecionada é um dobrão ou não, pois a disposição na jogada varia drasticamente, necessitando de mais ou menos cálculos. O caso mais simples é o de quando a peça não for um dobrão, pois dessa forma os seus centros se alinham com os centros da peça na ponta do jogo, necessitando assim só de uma translação ao longo dessa reta para definir a posição desejada da peça na jogada:

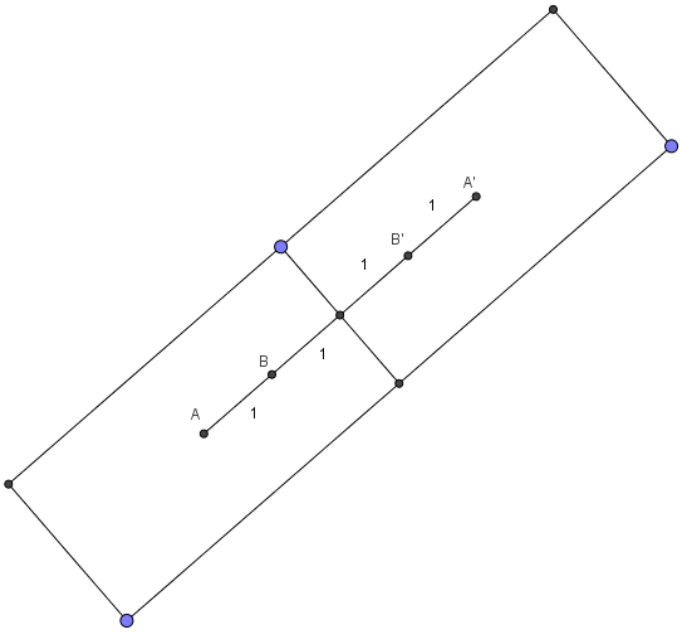


Figura 56 – Esquema da posição de soltar de uma peça normal

Logo, a posição do centro da peça desejado deve ser três vezes a distância entre o centro da peça e o centro do número na ponta do jogo, a partir do centro do número na ponta ou, em outras palavras, $A' = B + 3(B - A)$, sendo A e B pontos do tipo (x, y), enquanto a posição do centro do número desejado deve ser duas vezes essa distância, ou seja, $B' = B + 2(B - A)$, para alinhar os números na ponta do jogo.

No caso em que a peça for de fato um dobrão, o cálculo torna-se mais complexo, pois para qualquer angulação da peça da ponta do jogo, a posição de soltar deve estar perpendicular à peça da ponta.

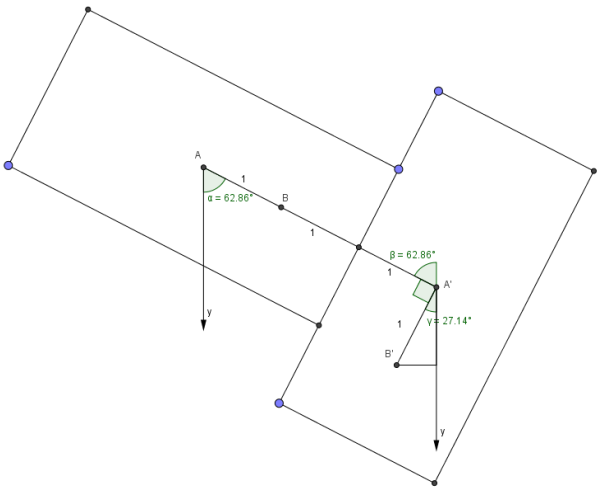


Figura 57 – Esquema da posição de soltar de uma peça dobrão

Analisando o esquema da Figura 13, percebe-se que a posição do centro da peça desejada para o caso do dobrão pode ser deduzida da mesma forma que para uma peça normal, efetuando $A' = B + 2(B - A)$. O centro do número desejado, por sua vez, é calculado deslocando o centro da peça A' em função do ângulo γ, que é igual ao ângulo complementar da orientação da peça da ponta do jogo, considerando x positivo para a direita e y positivo para baixo:

$$\begin{cases} B'_x = A'_x - dist(A,B)\sin(90^o - \alpha) \\ B'_y = A'_y + dist(A,B)\cos(90^o - \alpha) \end{cases}$$

$$\Downarrow$$

$$\begin{cases} B'_x = A'_x - dist(A,B)\cos(90^o - \alpha) \\ B'_y = A'_y + dist(A,B)\sin(90^o - \alpha) \end{cases}$$

Estabelecidas as coordenadas desejadas para o centro da peça e para o centro do número da peça selecionada, esses pontos são então introduzidos na sub-rotina descrita anteriormente para obter a orientação desejada da peça na posição de solta.

3.7.4. Correções de posição e orientação referentes à integração

Vale ressaltar que todos esses cálculos de posição e orientação foram feitos relativos aos sistemas de coordenadas das câmeras, o que sozinho não seria problemático vista a calibração das posições das câmeras em relação à base do braço robótico discutida anteriormente neste documento. Contudo, percebeu-se durante a integração das partes de robótica e de visão que os sistemas de coordenadas das câmeras estavam ou rotacionados ou invertidos em relação ao sistema da base do robô, provocando assim incoerências nos movimentos do braço.

Os esquemas abaixo mostram as disposições entre os sistemas de coordenadas identificadas:

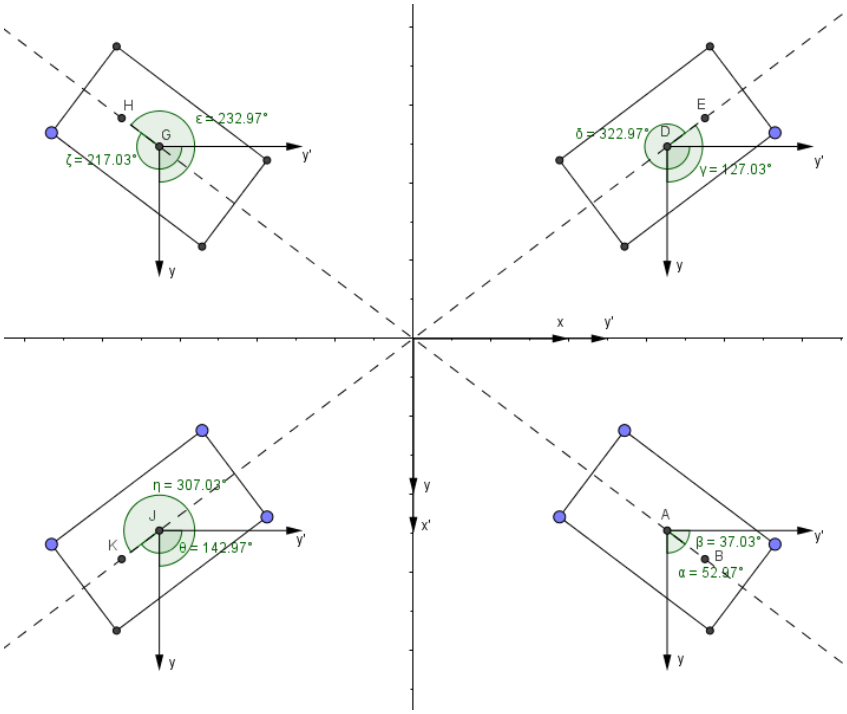


Figura 58 – Sistemas de coordenadas (câmera da mesa (x,y) vs. base do robô (x',y'))

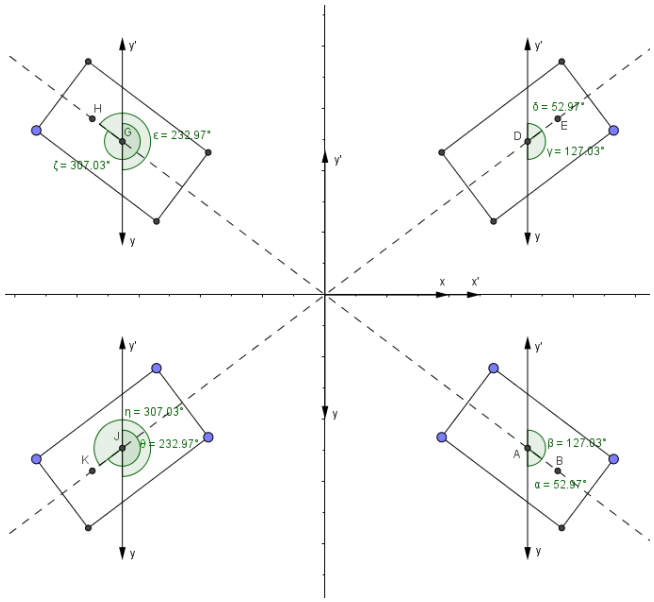


Figura 59 – Sistemas de coordenadas (câmera da mão do robô (x,y) vs. base do robô (x',y'))

Embora os esquemas pareçam complicados à primeira vista, a lógica por trás das conversões entre sistemas de coordenadas acabou por ser bem simples, empregando poucas operações matemáticas para isso. Quanto ao sistema da câmera da mesa (Figura 58), notou-se que a partir de 90^o de rotação em relação ao sistema original, o ângulo em relação ao novo sistema pode ser obtido calculando $360^o + (90^o - \alpha) = 450^o - \alpha$. Caso o ângulo seja menor que 90^o, o novo ângulo resultará em um valor maior que

360^o por essa conta, de modo que basta calcular o resto da divisão desse valor por 360^o para adquirir o valor correto. Já para o sistema da câmera da mão do robô, o processo de análise foi exatamente o mesmo, porém a referência para o cálculo não é mais 90^o, mas sim 180^o. Logo, as funções de conversão entre sistemas de coordenadas deduzidas foram:

$$\begin{cases} \alpha'_{mesa} = (450 - \alpha_{mesa}) mod(360) \\ \alpha'_{robô} = (640 - \alpha_{robô}) mod(360) \end{cases}$$

Como última correção, houve alguns problemas durante a integração do projeto na etapa de rotacionar o pulso da garra do robô para a posição certa, sendo que ela muitas vezes tendia a atingir o fim de curso do eixo do pulso na tentativa de atingir posições relativas impossíveis, questão que foi abordada anteriormente no tópico 2.4.2. Dentre as alterações efetuadas, optou-se por fornecer os ângulos de orientação das peças dentro de um intervalo de $\pm 180^o$ ao invés de um de 0^o a 360^o.

3.8. Como identificar que a partida acabou

Em meio aos possíveis cenários de desenvolvimento de uma partida de dominó, foram identificadas quatro situações nas quais o sistema deveria dar uma partida por encerrada:

- Se a partida foi iniciada com a mão do robô vazia, logo, a partida começaria já com o robô tendo ganho ela;
- Se o robô possui apenas uma peça e ela pode ser jogada, de modo que ao fim do movimento do braço de colocar a peça na mesa o jogo acaba;
- Se o número de peças na mesa e na mão do robô somadas der o total de peças em um jogo de dominó, indicando que a pessoa ficou sem peças e, consequentemente, ela ganhou a partida;
- E por último, se o número de peças na mesa e na mão do robô somadas for igual à da rodada anterior, indicando que tanto o robô quanto a pessoa passaram a vez.

3.9. Código do Labview

Devido ao detalhamento do software e principalmente ao fato de todos os métodos de implementação terem sido explicados detalhadamente anteriormente,

desde a identificação das peças até o cálculo das posições que as peças devem ser jogadas, não iremos comentar o código do Labview em sí, pois se trata apenas de programação, entretanto, as diversas seções do código podem ser vistas nas imagens abaixo, além disso, no código em sí existem comentários detalhados de cada seção, que podem ser vistos melhor abrindo o software.

3.9.1. Interface do usuário

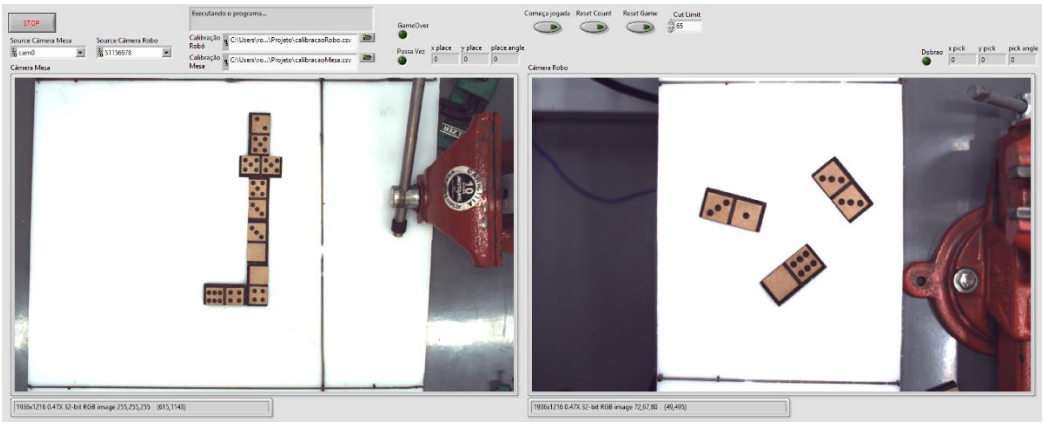


Figura 60 – Interface do usuário

3.9.2. SubVI de identificação de peça ponta e número ponta

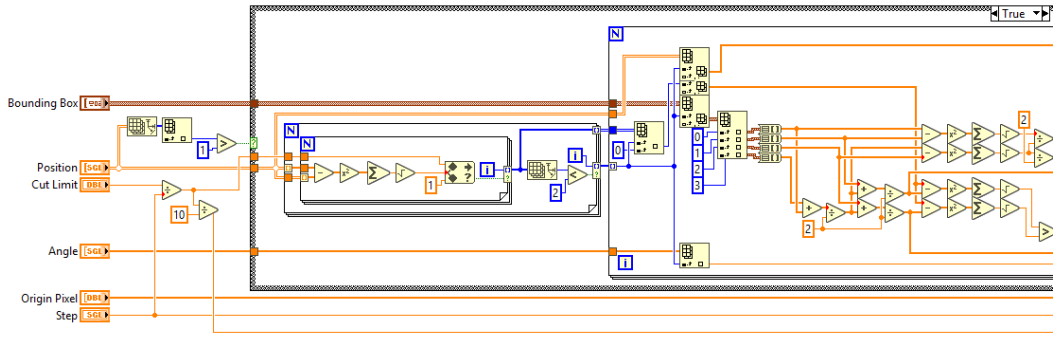


Figura 61 – SubVI getBorderNum (parte 1)

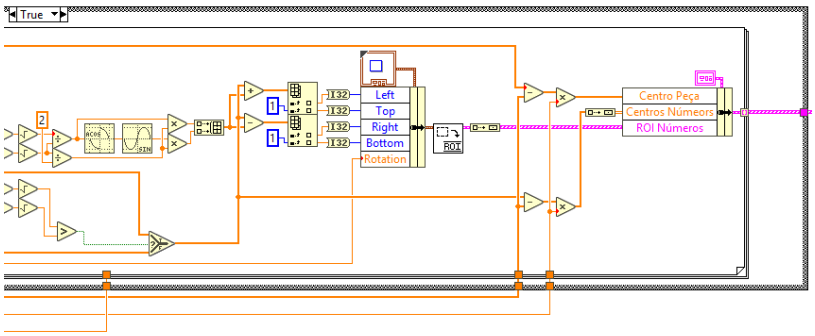


Figura 62 – SubVI getBorderNum (parte 2)

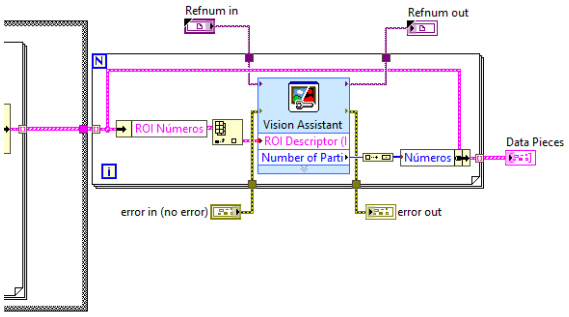


Figura 63 – SubVI getBorderNum (parte 3)

3.9.3. SubVI de identificação das peças da mão do robô e seu número

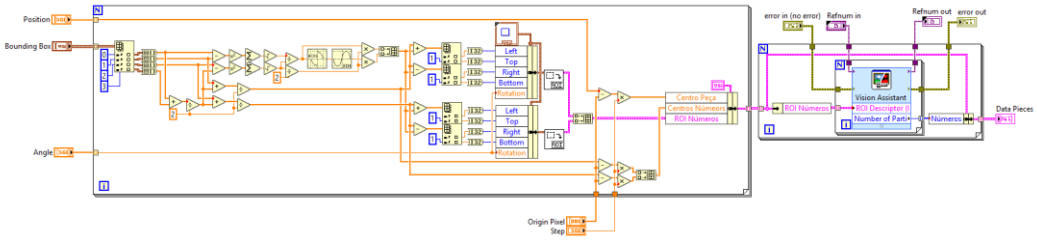


Figura 64 – SubVI getPieces

3.9.4. Programa principal

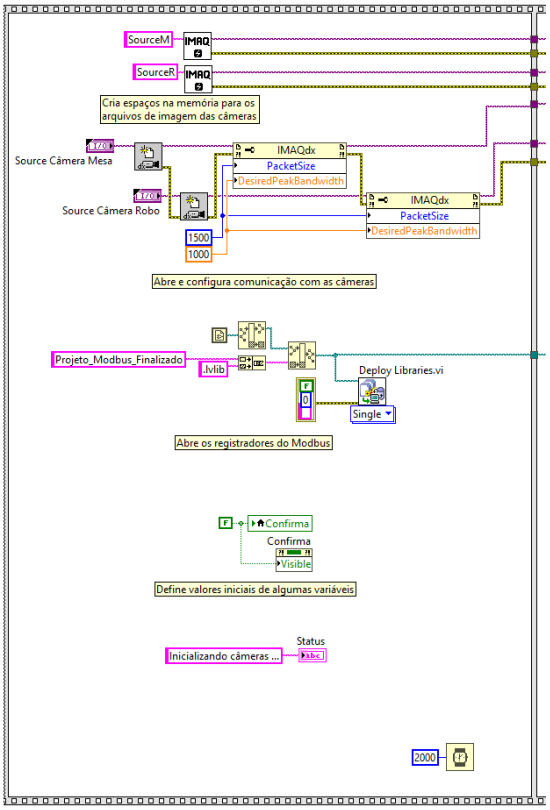


Figura 65 – Inicialização

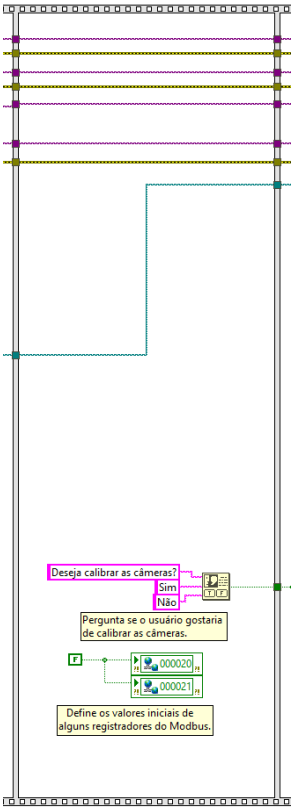


Figura 66 – Popup da calibração

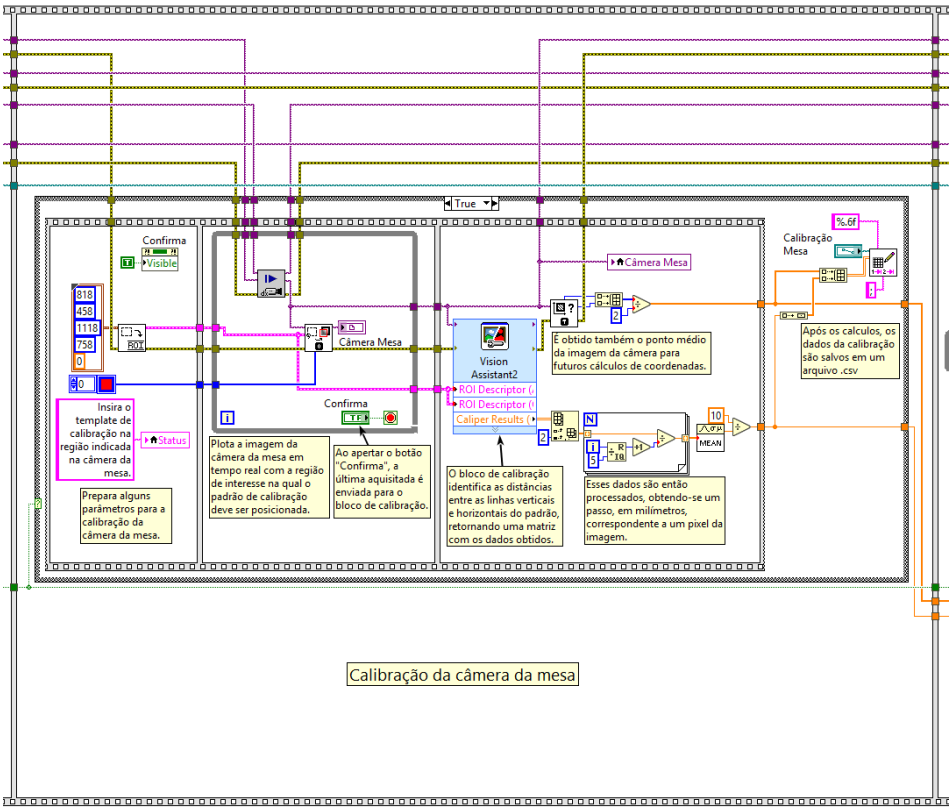


Figura 67 – Calibração da câmera da mesa (caso true)

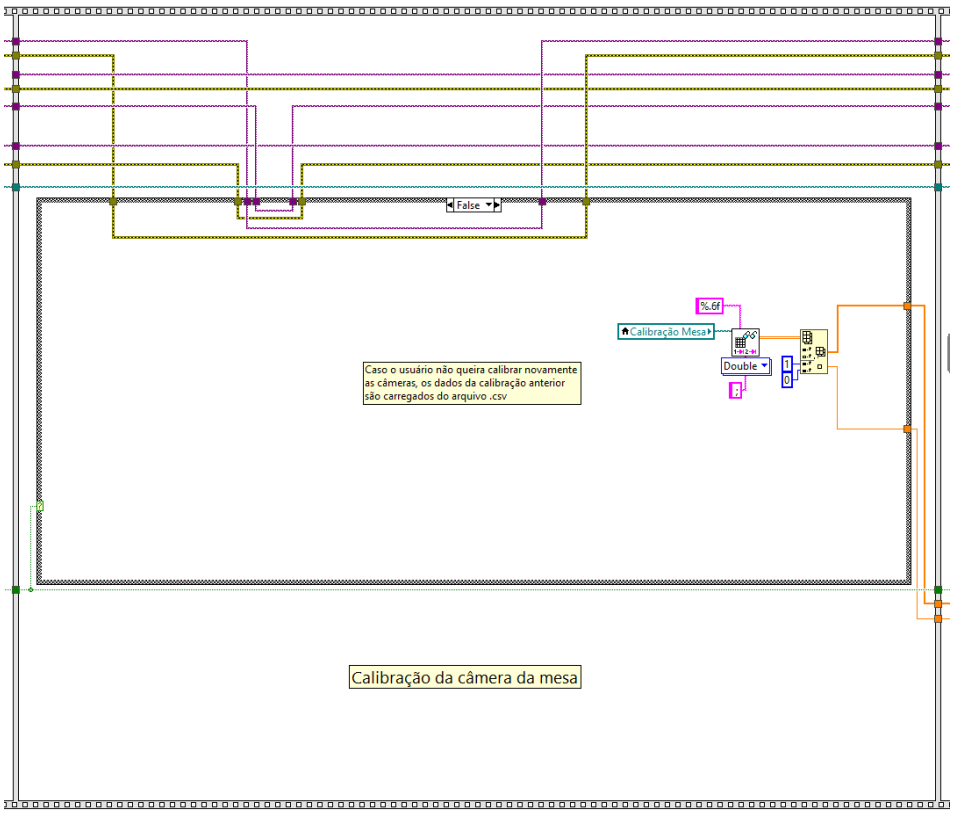


Figura 68 – Calibração da câmera da mesa (caso false)

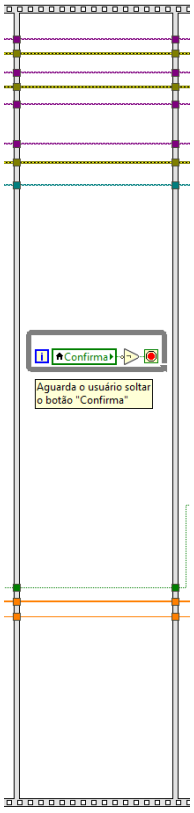


Figura 69 – Transição da calibração

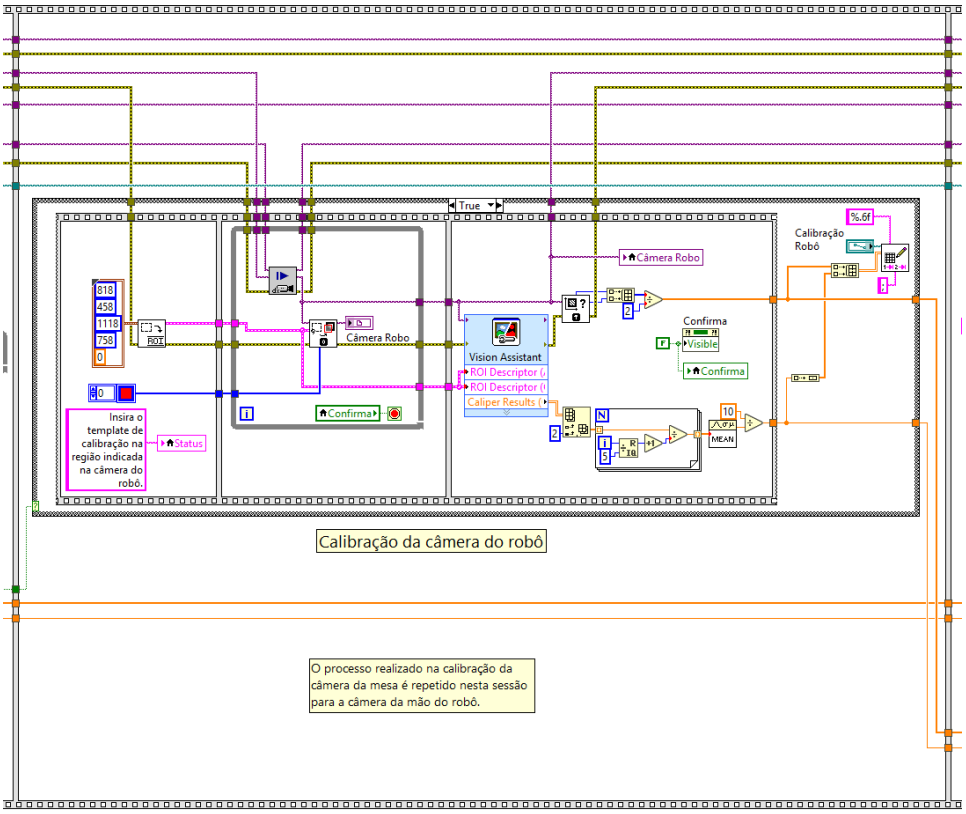


Figura 70 – Calibração da câmera da mão do robô

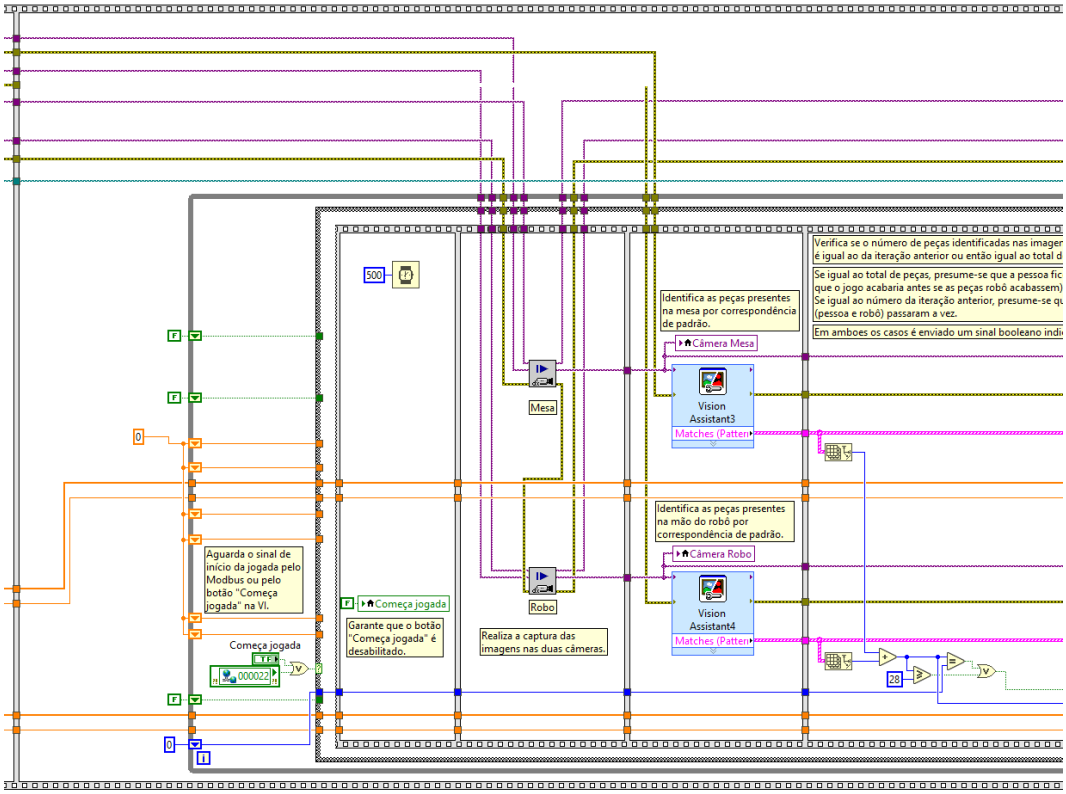


Figura 71 – Início do bloco principal (caso true)

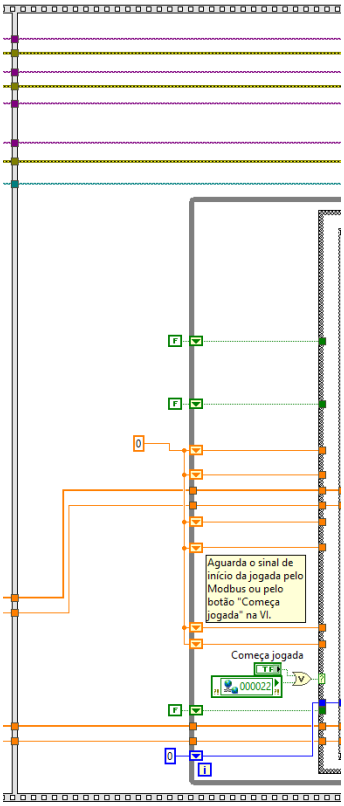


Figura 72 – Verificação de início de jodaga

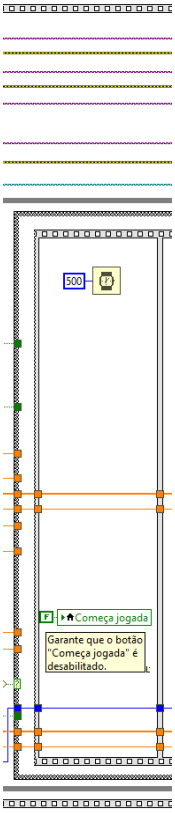


Figura 73 – Começo de jogada

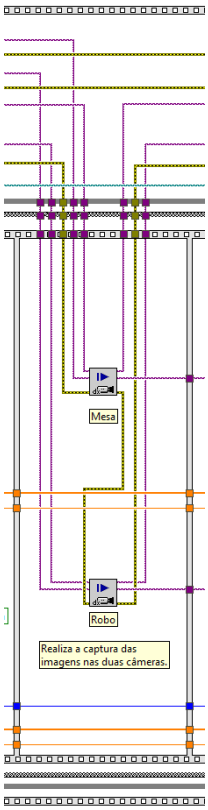


Figura 74 – Aquisição de imagens

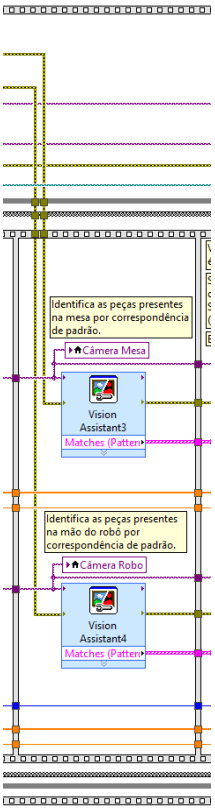


Figura 75 – Identificação das peças

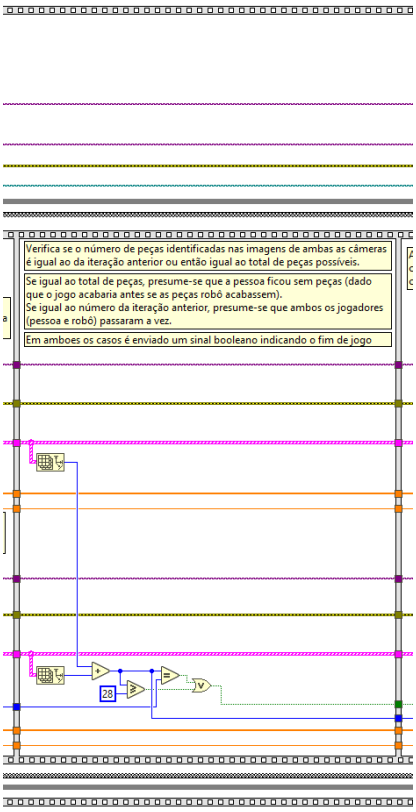


Figura 76 – Verificação de fim de jogo

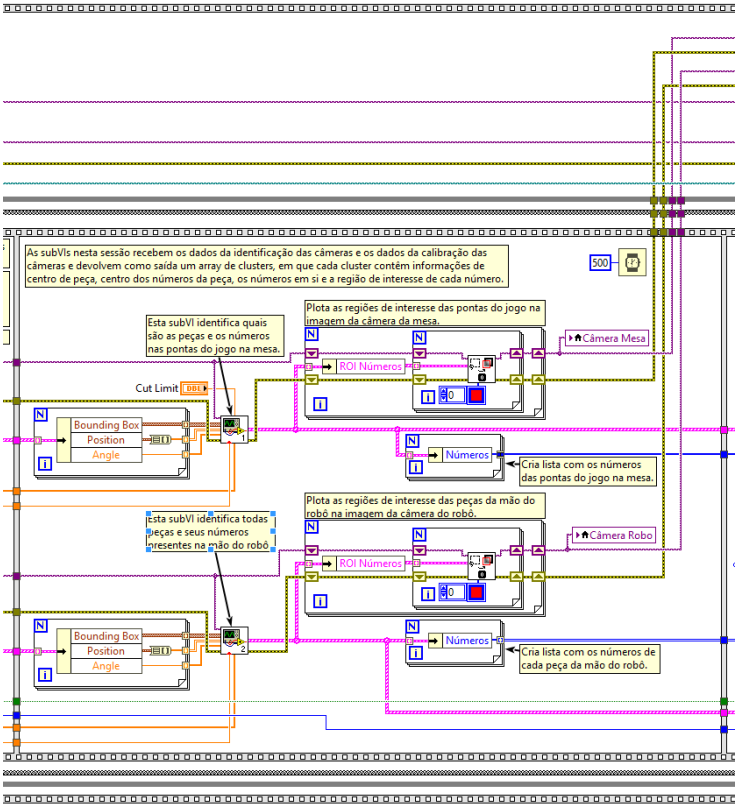


Figura 77 – Identificação dos números das peças

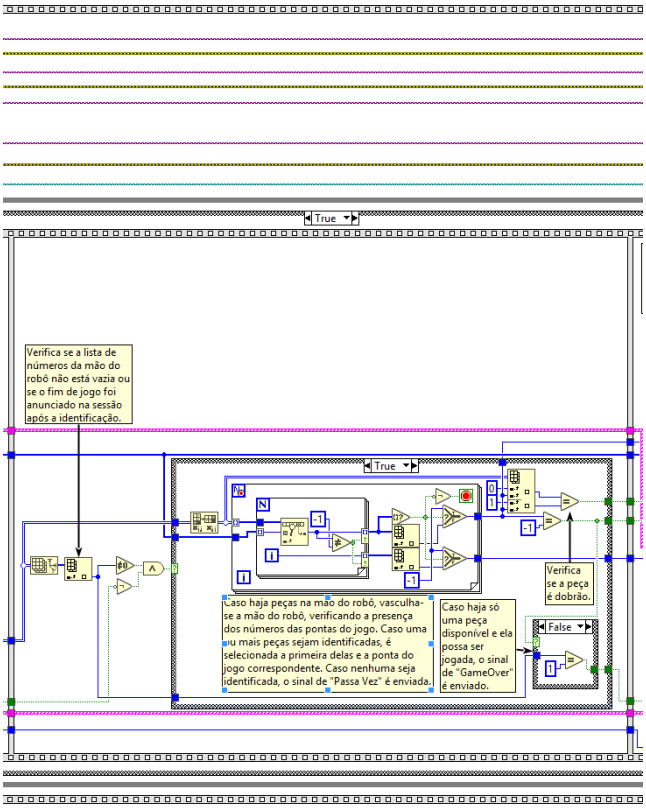


Figura 78 – Seleção da peça da jogada (caso true)

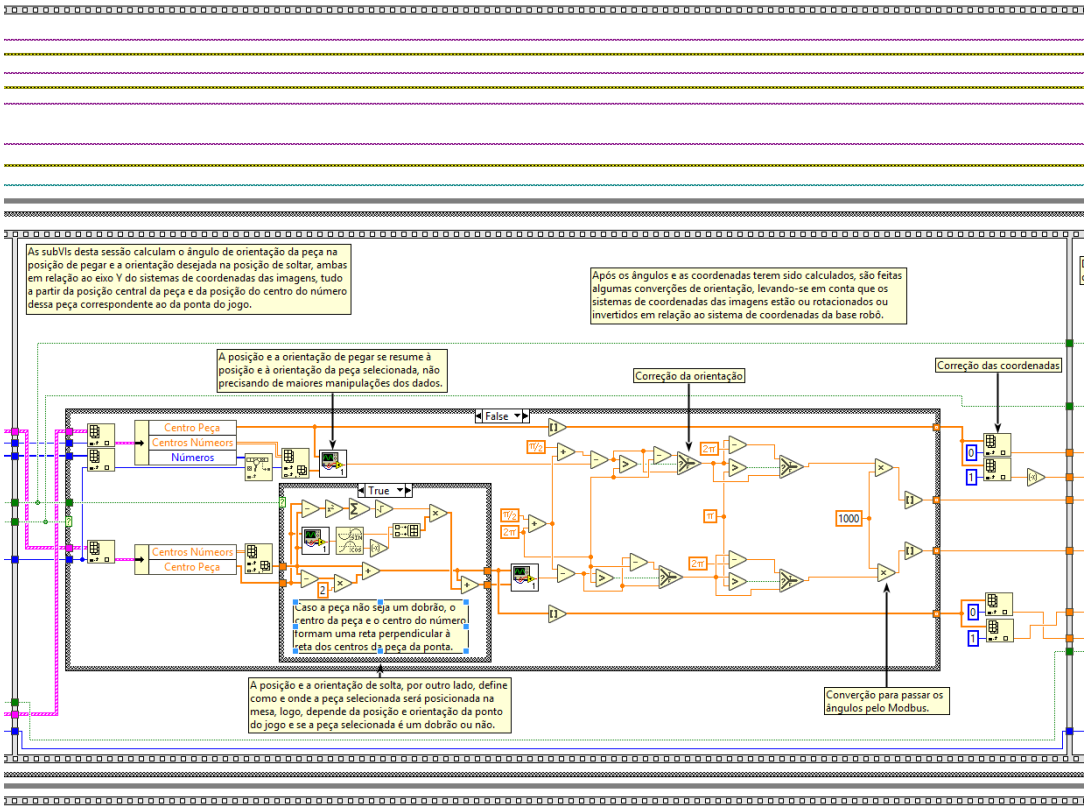


Figura 79 – Cálculo das coordenadas (caso true)

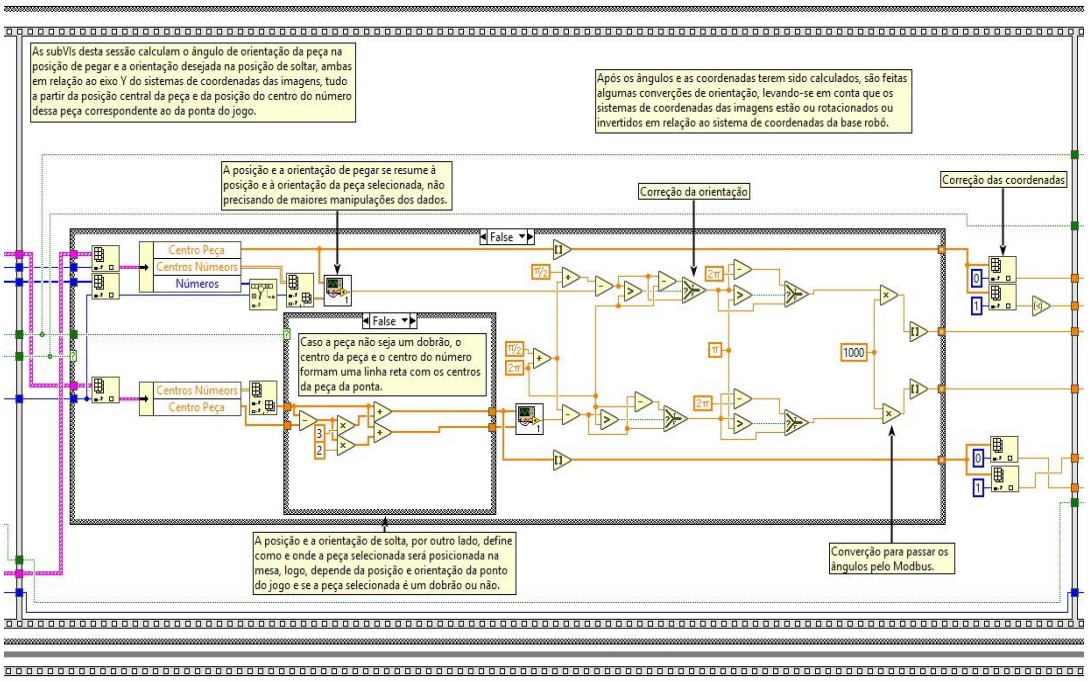


Figura 80 – Cálculo das coordenadas (caso false)

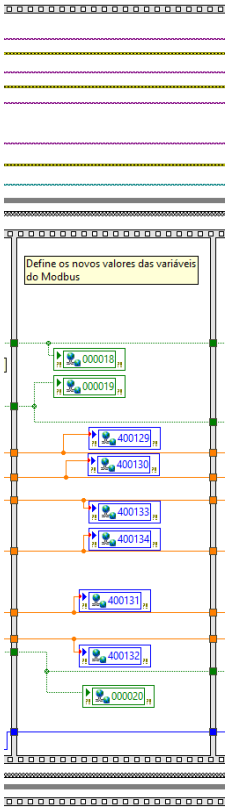


Figura 81 – Envio das coordenadas e booleanos para o Modbus

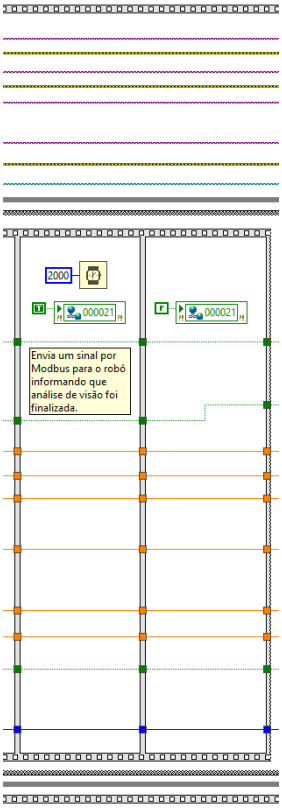


Figura 82 – Envio de aviso de finalização de processo para o Modbus

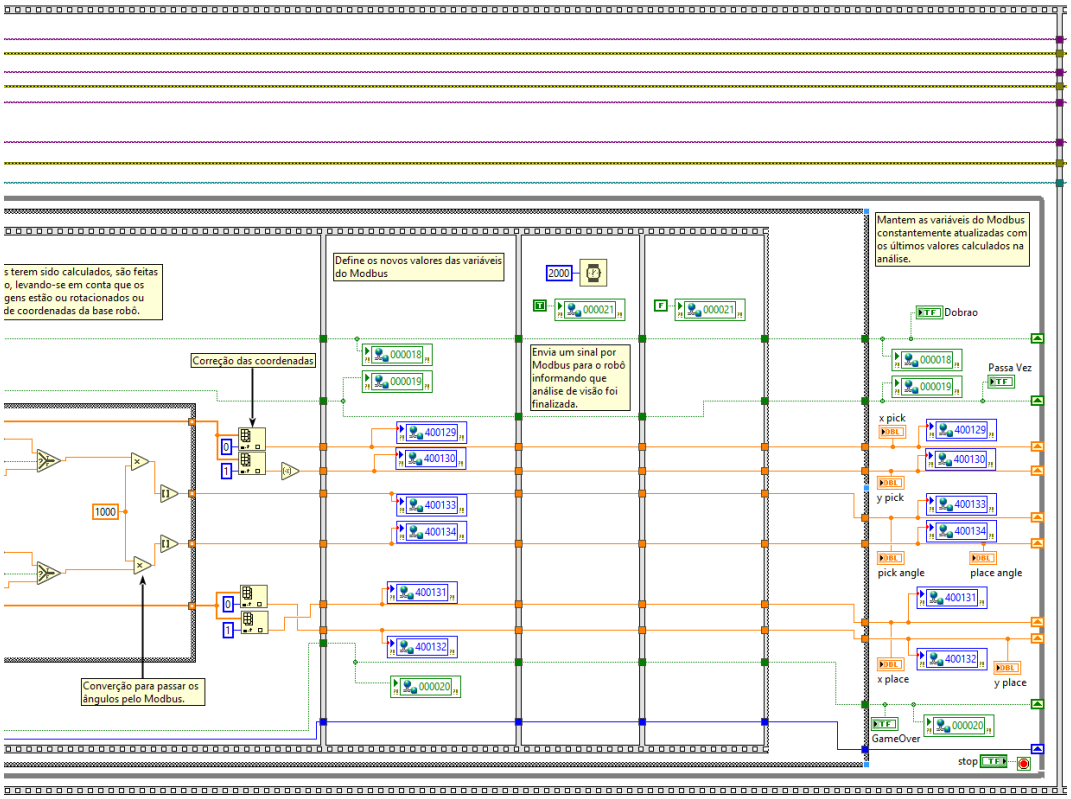


Figura 83 – Fim do bloco principal (caso true)

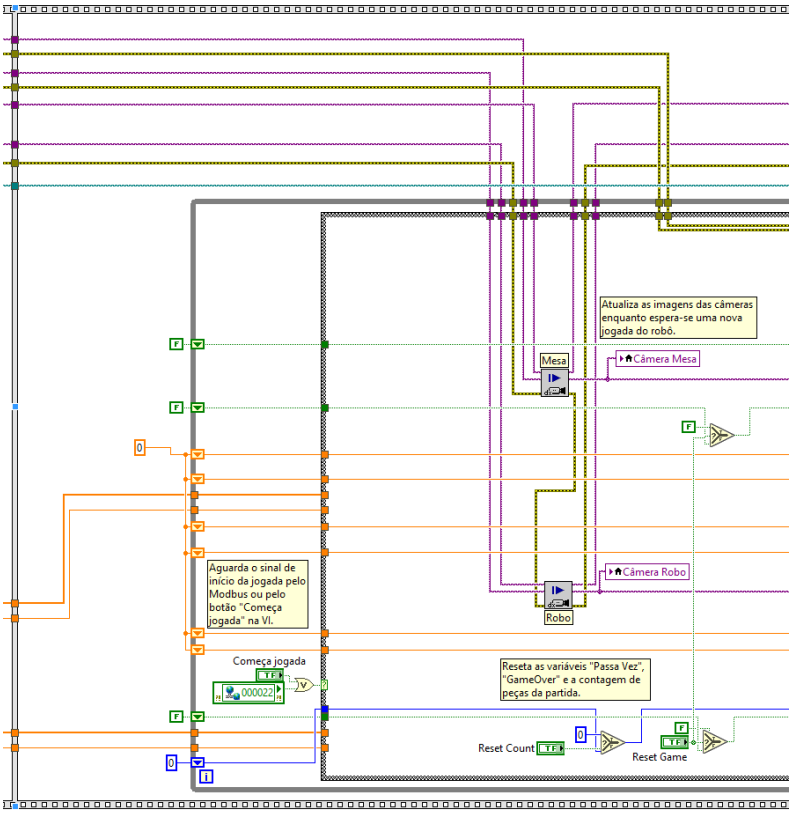


Figura 84 – Bloco Principal (caso false)

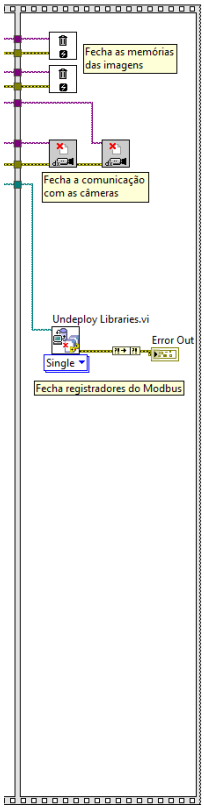


Figura 85 – Finalização

4. Vídeo demonstrativo do projeto finalizado

O grupo criou um vídeo para demonstrar o funcionamento do projeto depois de finalizado, este vídeo pode ser compartilhado livremente e pode ser visualizado clicando aqui.