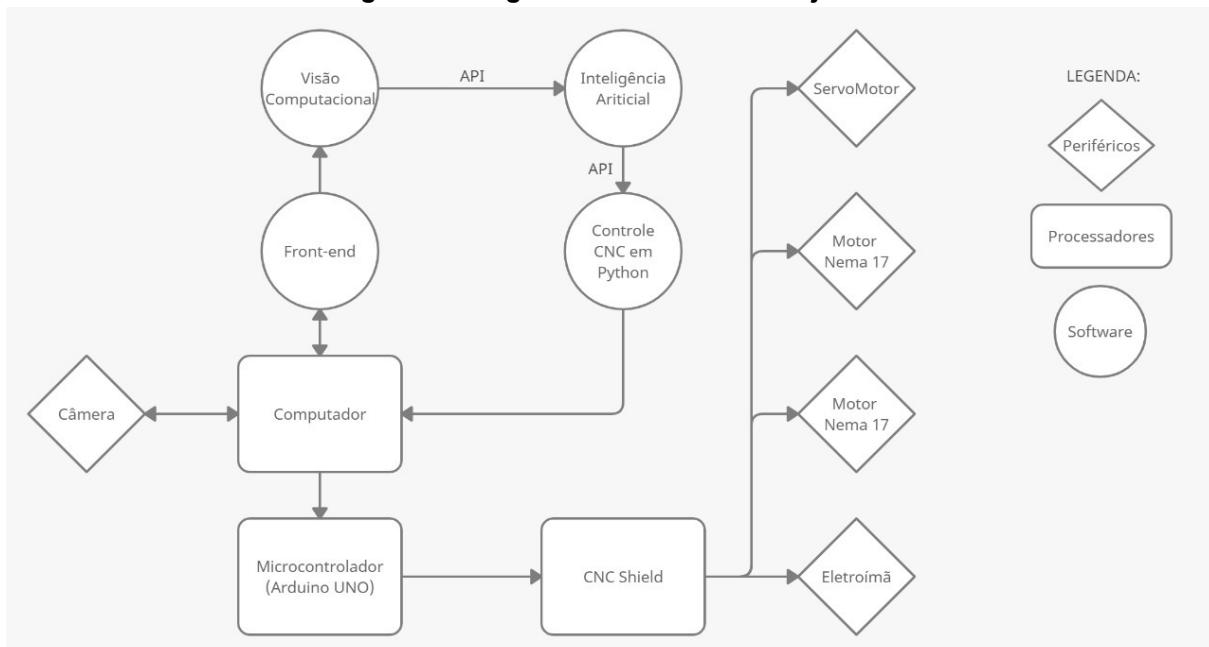


3 DESENVOLVIMENTO DO CONJUNTO EDUCACIONAL

O conjunto educacional proposto neste trabalho foi desenvolvido a partir dos conceitos apresentados no capítulo anterior. O agente deste projeto é um manipulador cartesiano de três graus de liberdade, composto por atuadores que são comandados por uma *CNC Shield*, a qual está conectada ao microcontrolador (Arduino UNO).

Figura 9 - Diagrama de blocos do conjunto



Fonte: Autoria própria (2021)

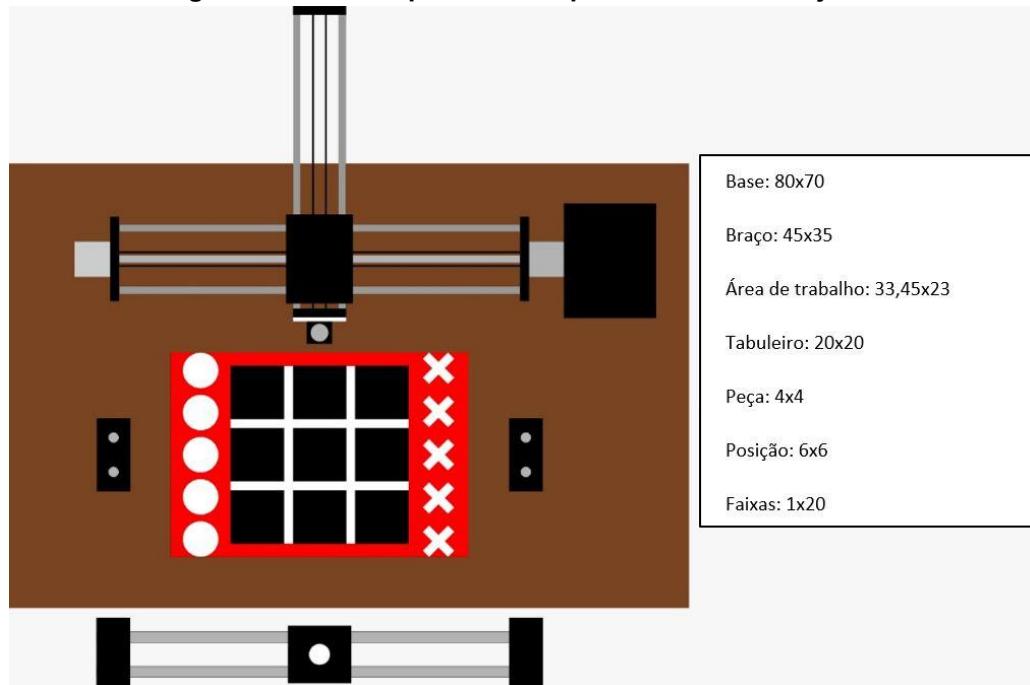
Conforme pode ser visto na Figura 9, o conjunto é dividido em *softwares*, que comunicam-se entre si através de um API, periféricos, responsáveis por obter imagens e movimentar o braço, e processadores, que fazem a integração entre o *software* e os periféricos.

Neste capítulo são apresentados em detalhes todos esses componentes do projeto, como a montagem da estrutura mecânica (Figura 10), as ligações dos componentes eletrônicos, o desenvolvimento dos programas de inteligência artificial, visão computacional e controle CNC e a integração destes sistemas através de API.

Além disso, visando a educação, o código desenvolvido foi disponibilizado em um repositório do Github com uma licença de uso comum para que qualquer

interessado possa analisá-lo e usá-lo e outros projetos, propagando o conhecimento adquirido aqui (CADAMURO, NASCIMENTO, VELHO, MIYACHI, 2021).

Figura 10 - Vista superior do esquema físico do conjunto

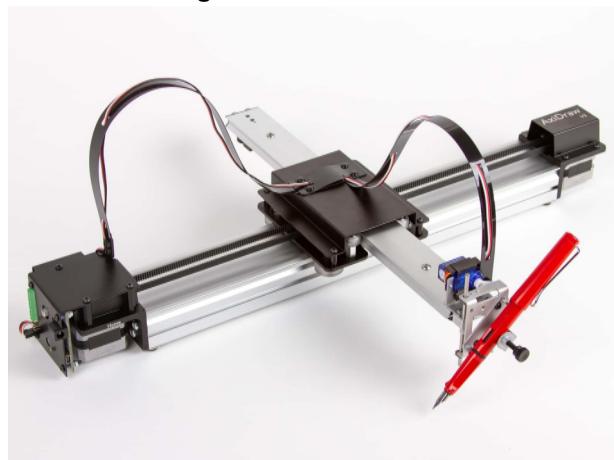


Fonte: Autoria própria (2021)

3.1 Estrutura

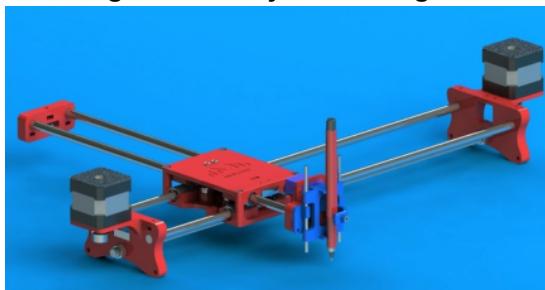
A estrutura do braço cartesiano foi baseada no projeto DrawingBot (MAKERC, 2016), que é uma versão de código aberto do modelo comercial AxiDrawV3. Ambos os projetos são plotters CNC de caneta, capazes de escrever ou desenhar em qualquer superfície plana. O projeto DrawingBot, da MakerC, está disponível na plataforma Thingiverse, sob licença *Creative Commons*. As Figuras 11 e 12 apresentam os modelos citados anteriormente (AxiDraw V3 e Projeto DrawingBot).

Figura 11 - AxiDraw V3



Fonte: Página da empresa AxiDraw (2021)

Figura 12 - Projeto DrawingBot



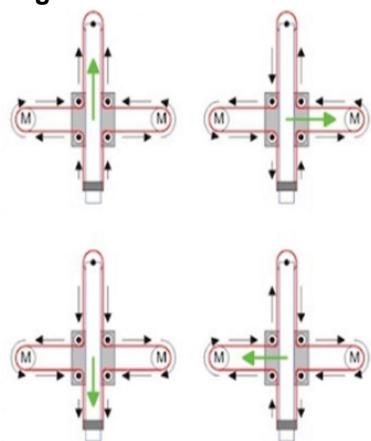
Fonte: Página da MakerC na plataforma Thingiverse (2021)

Para facilitar o entendimento, o projeto da estrutura mecânica foi dividido em duas partes: o braço cartesiano, abrangendo toda estrutura mecânica necessária para movimentar no plano XYZ; e a estrutura da câmera, responsável por manter a câmera em uma posição ideal para captura de imagens.

3.1.1 Estrutura Mecânica

O braço é composto por dois motores de passo Nema 17, responsáveis pelo movimento do braço cartesiano no plano XY. Os motores são dispostos em um arranjo mecânico chamado *CoreXY*, que é um sistema manipulador paralelo, ou seja, ambos os motores são estacionários.

Figura 13 - Estrutura cruzada



Fonte: Retirado do site test3dprints.com (2021)

A Figura 13 apresenta um diagrama do arranjo mecânico cruzado, o sistema possui apenas uma correia, que está posicionada no mesmo plano dos motores. Os motores estão fixados na direção do eixo X. Na parte traseira do braço (direção do eixo Y) existe um rolamento fixado, por onde a correia passa e na outra extremidade a correia é fixada, próximo a cabeça do braço robótico. Quando os motores se movimentam no mesmo sentido, a cabeça do braço cartesiano é movimentada para direita ou esquerda (dependendo se o sentido de rotação é horário ou antihorário). Já quando os motores se movimentam em sentidos opostos, a cabeça do braço cartesiano é movimentada para frente e para trás (dependendo do sentido de rotação de cada motor).

Dessa maneira, poucas mudanças foram executadas em relação ao projeto mecânico do braço cartesiano DrawingBot. Quando necessário, pequenas modificações foram executadas nos modelos 3D originais, utilizando a plataforma de modelagem 3D Tinkercad. Para adaptar o braço cartesiano para aplicação no presente trabalho, também foi necessário a modelagem 3D de novas peças, que foram elaboradas no software de modelagem Fusion 360. Todas as peças 3D utilizadas no projeto são apresentadas na Figura 14.

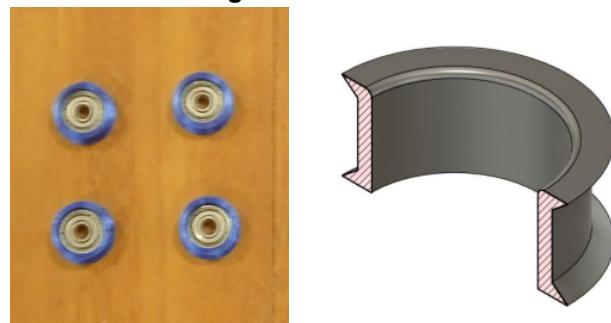
Figura 14 - Peças 3D impressas



Autoria própria (2021)

Uma polia foi modelada e impressa para evitar que a correia escapasse dos rolamentos de 624zz (rolamentos internos da estrutura). As quatro polias impressas e uma representação seccionada do desenho 3D são apresentadas na Figura 15.

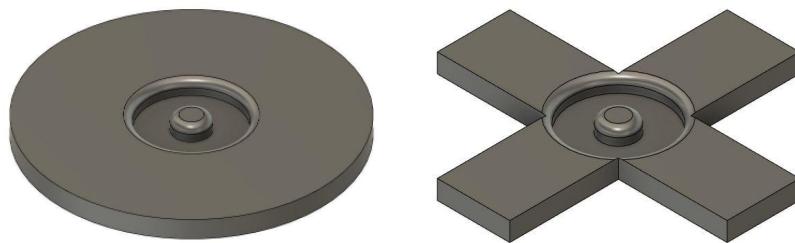
Figura 15 - Polia 3D



Fonte: Autoria própria (2021)

O tabuleiro é posicionado em uma superfície lisa, na qual também estão as peças “X” e “O”, que possuem um material metálico em seu interior, conforme apresentado na Figura 16.

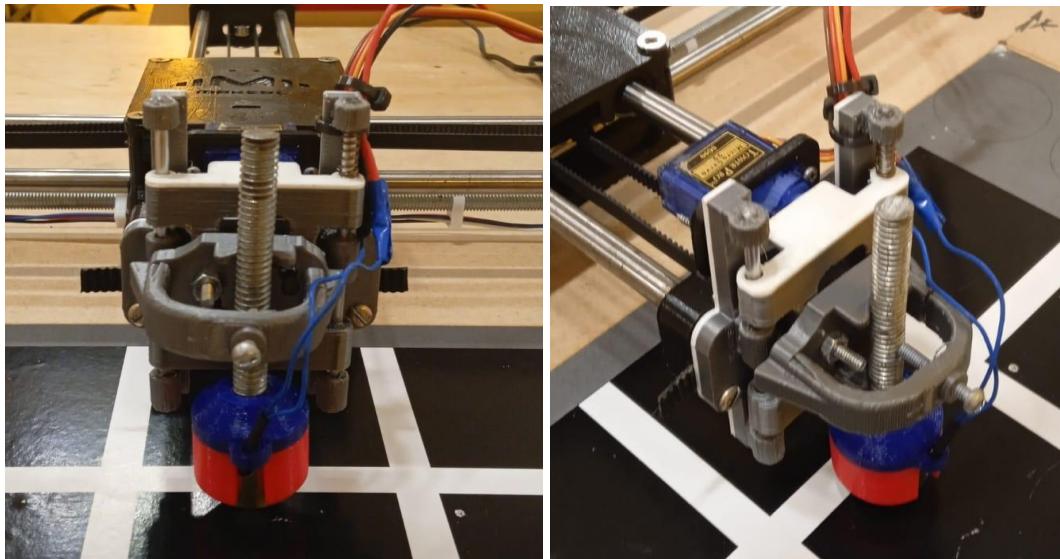
Figura 16 - Peças “O” e “X”



Fonte: Autoria própria (2021)

A priori o projeto DrawingBot possui uma caneta como ferramenta, que fica posicionada na cabeça do braço robótico. Uma peça 3D foi modelada para fixar o eletroímã do projeto na estrutura do eixo Z. O eixo Z permaneceu sem mudanças em relação ao projeto original, funcionando através do acionamento de um micro servo motor. Além disso, uma barra rosada de metal foi utilizada para conectar o eixo Z e o suporte 3D do eletroímã. Toda estrutura do eixo Z é apresentada na Figura 17, é possível controlar o eixo Z em duas posições, com o servo motor acionado ou desligado.

Figura 17 - Eixo Z e eletroímã

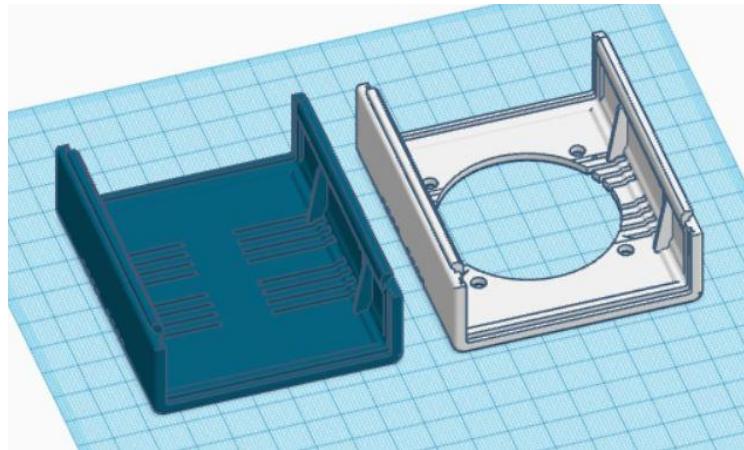


Fonte: Autoria própria (2021)

Algumas peças 3D do projeto original sofreram pequenas mudanças, como na caixa que fixa arduino e a *shield cnc*, onde foram adicionadas furações para fixação de um *cooler* e uma abertura para maior entrada de ar, uma vez que o

datasheet dos drivers a4988 indicavam a necessidade de ventilação forçada. A Figura 18 apresenta a tampa sem alteração, à esquerda, e a tampa da caixa alterada, à direita

Figura 18 - Alteração da tampa 3D



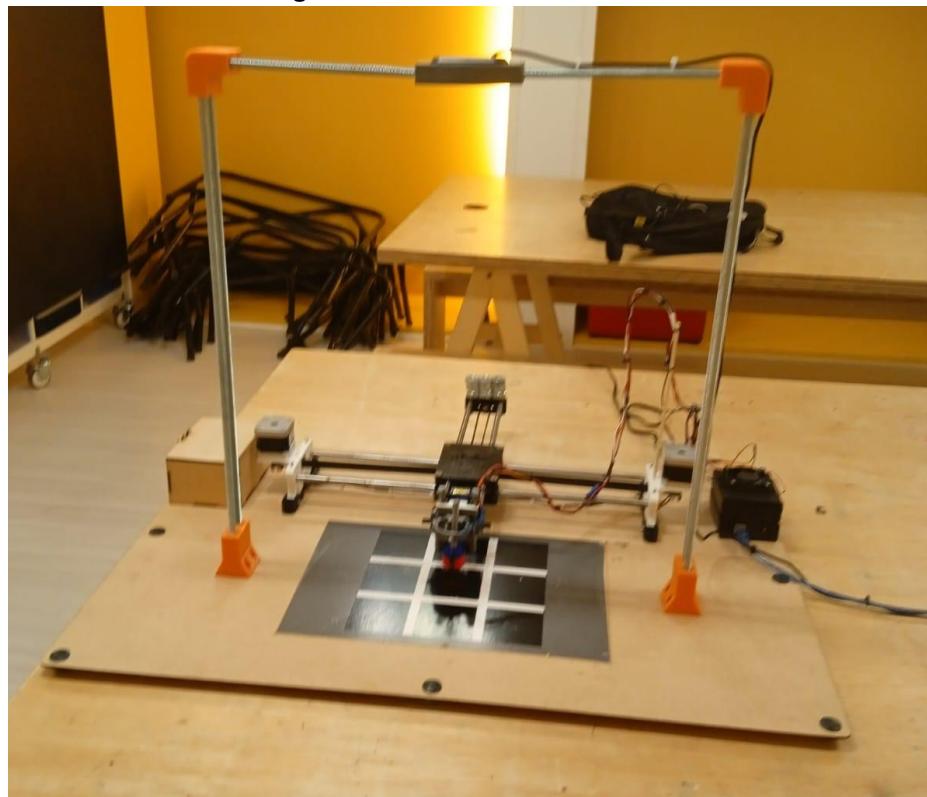
Fonte: Autoria própria (2021)

3.1.2 Estrutura da câmera

Com o intuito de facilitar a obtenção de imagens e evitar que ligações adicionais fossem feitas no Arduino, uma câmera WebCam foi utilizada. A câmera de vídeo Full HD possui uma resolução máxima de 1920x1080 com interface USB 2.0 e sistema Plug and Play, ou seja, o computador reconhece e configura a câmera automaticamente.

A câmera foi fixada na base de MDF do projeto, através de peças 3D que foram modeladas e impressas. Para conectar as peças 3D da estrutura barras com rosca M8 foram utilizadas. Toda estrutura de suporte para câmera é apresentada na Figura 19.

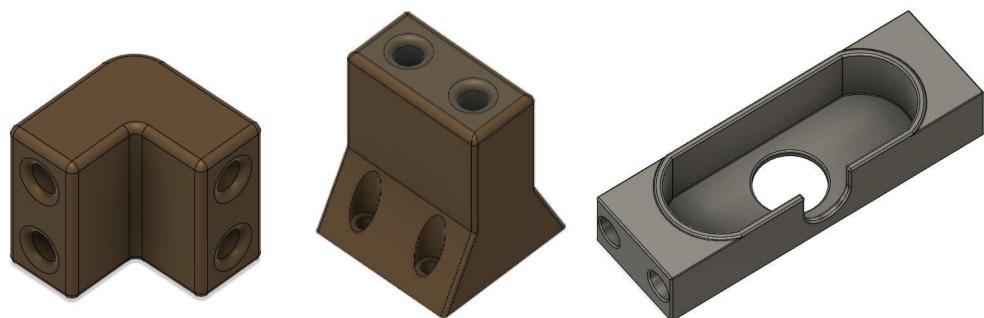
Figura 19 - Estrutura da câmera



Fonte: Autoria própria (2021)

A estrutura original do *plotter* DrawingBot não possui câmera, logo, todas as peças 3D foram modeladas utilizando o software Fusion 360 e posteriormente impressas em uma impressora 3D. As peças modeladas são: um suporte para câmera, onde a câmera foi fixada; duas peças para base da estrutura da câmera, que foram fixadas na base de MDF através de porcas e parafusos; e duas conexões em C, para conectar as peças da base e o suporte da câmera. As peças 3D modeladas são apresentadas na Figura 20.

Figura 20 - Peças 3D da câmera

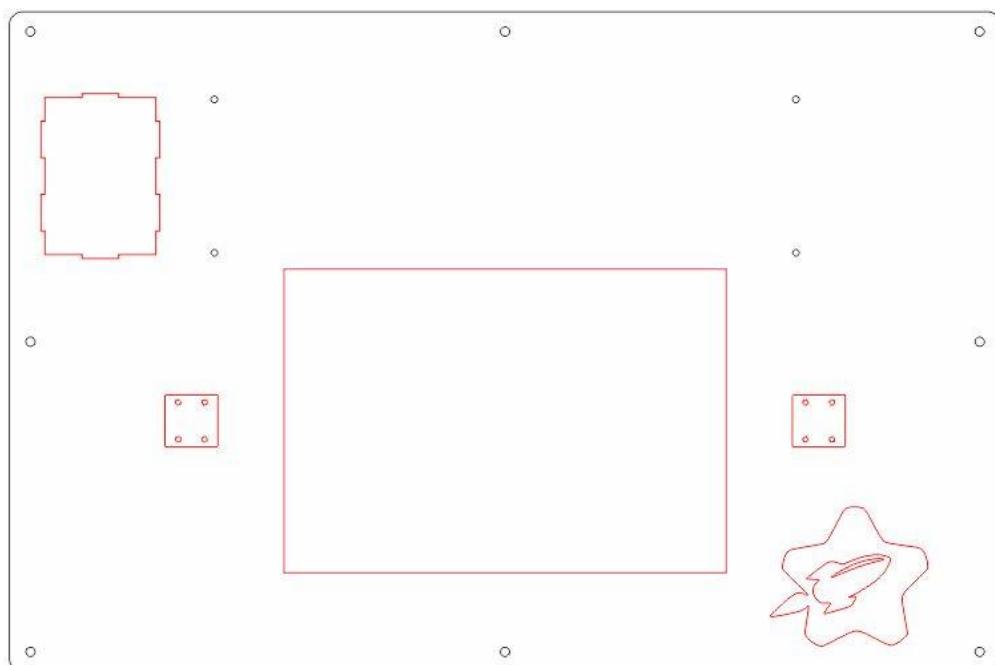


Fonte: Autoria própria (2021)

3.1.3 Estruturas adicionais

Além da estrutura mecânica e da câmera, também foi necessário elaborar uma base de MDF para fixá-las. A base de MDF foi desenhada com auxílio do software CorelDraw e posteriormente cortada em uma máquina CNC de corte a laser. A Figura 21 apresenta a base desenhada, as linhas pretas representam os locais de corte do MDF e as linhas vermelhas representam o contorno de gravação no MDF.

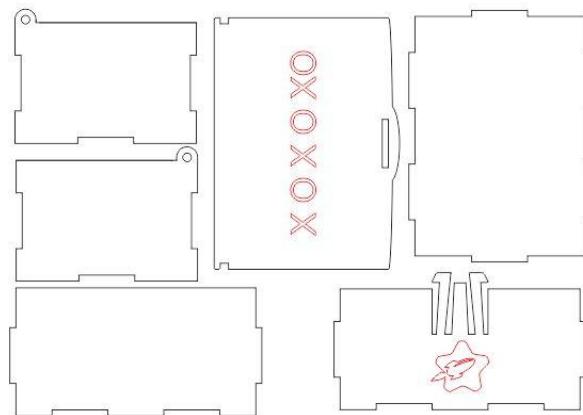
Figura 21 - Base de MDF



Fonte: Autoria própria (2021)

Para evitar que as peças ficassem espalhadas uma caixa de MDF com tampa também foi desenhada e cortada em uma CNC de corte a laser, a marcação vermelha no canto superior esquerdo da Figura 23 indica onde a caixa de MDF deve ser colada. O desenho da caixa de MDF é apresentado na Figura 22.

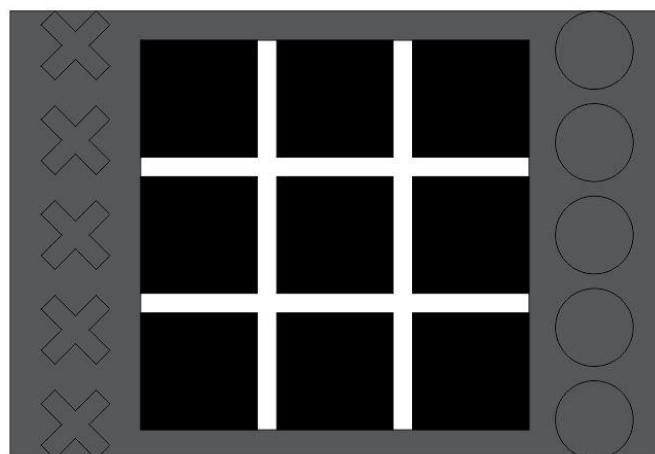
Figura 22 - Caixa de MDF



Fonte: Autoria própria (2021)

A área de trabalho do braço cartesiano possui uma área de 33,45x23 cm e é representada pelo retângulo vermelho no centro da base de MDF da Figura 23. A fim de aproveitar toda área de trabalho do braço, um adesivo contendo o tabuleiro e marcações para posicionamento das peças foi elaborado no software CorelDraw. O adesivo é apresentado na Figura 23.

Figura 23 - Tabuleiro



Fonte: Autoria própria (2021)

3.2 Eletrônica

A estrutura eletrônica do projeto é seccionada em duas partes isoladas: a primeira com a câmera, que é ligada diretamente no computador através de uma porta USB; e a segunda com todos componentes eletrônicos da máquina CNC, incluindo o Arduino UNO, a *shield* CNC e todos atuadores do projeto.

3.2.1 Arduino UNO

Para facilitar a aplicação dos microcontroladores, plataformas embarcadas foram criadas. Desse jeito, possibilitando a elaboração de projetos de maneira mais simples e modular. A plataforma Arduino possui diversos modelos de computadores de placa única. Estes computadores são projetados com um microcontrolador Atmel AVR, linguagem de programação baseada em C/C++, hardware e softwares livres. Um dos modelos mais difundidos no mercado é o Arduino Uno, esse dispositivo normalmente é aplicado em prototipagem de diferentes tipos de projetos pela sua versatilidade, baixo custo, simplicidade e alta compatibilidade com placas que oferecem recursos extras. A Figura 24 apresenta a placa Arduino Uno REV3.

Figura 24 - Arduino Uno REV3



Fonte: Página da plataforma Arduino (2021)

De acordo com as especificações técnicas descritas na página da licença *Creative Commons* da placa, o Arduino Uno é uma placa microcontrolada baseada no ATmega328P e possui 14 pinos de E/S, 6 entradas analógicas, um ressonador de cerâmica, conexão, uma conexão USB, um conector de alimentação de 6 a 12V, um conector ICSP e um botão de reinicialização.

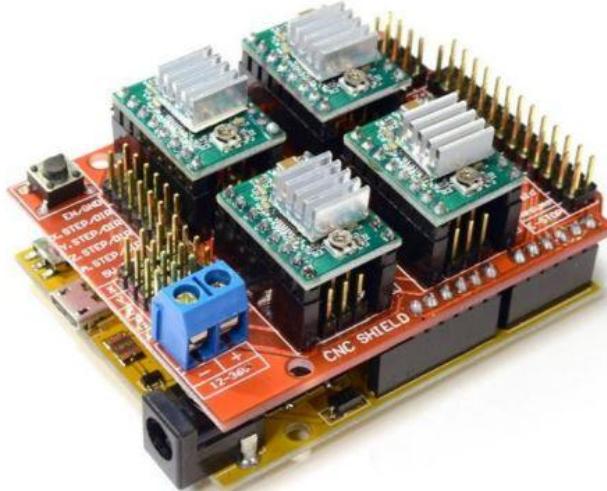
Uma placa Arduino UNO foi aplicada no projeto devido à possibilidade de programação em linguagem *grbl* e facilidade de controle de diversos atuadores quando integrada a shield CNC V3. O arduino Uno é conectado diretamente no computador através de um cabo USB-A e USB-B, toda comunicação serial é feita através desta conexão.

3.2.2 CNC Shield V3

Para aumentar a funcionalidade das placas Arduino, é possível integrar placas eletrônicas auxiliares que facilitam a execução destas funções extras. Estas placas são conectadas de maneira simples, pois possuem pinos que podem ser acoplados nos terminais de entrada e saída do Arduino. Existem diferentes tipos de shields, que podem incluir funções como acesso a bluetooth, Ethernet, GPS, telas sensíveis ao toque, monitores LCD e outros.

A placa eletrônica CNC Shield V3, que geralmente é aplicada em protótipos de máquinas de corte a laser, fresadoras CNC e impressoras 3D é apresentada na Figura 25. Conforme as especificações técnicas da *shield*, esta pode ser acoplada em um Arduino Uno para facilitar o controle de velocidade e direção de até quatro motores de passo, com uma corrente máxima de 2A por canal. Além disto, é possível medir a corrente elétrica em cada motor e executar outros recursos extras com funções de paragem, avanço e travagem. A *CNC Shield* é programada através da linguagem de programação *G-Code*, algo que é possível através do *software* GRBL, um *software* de código aberto e de “alto desempenho para controlar o movimento de máquinas” (CONRADO, 2017).

Figura 25 - CNC Shield V3 e drivers A4988



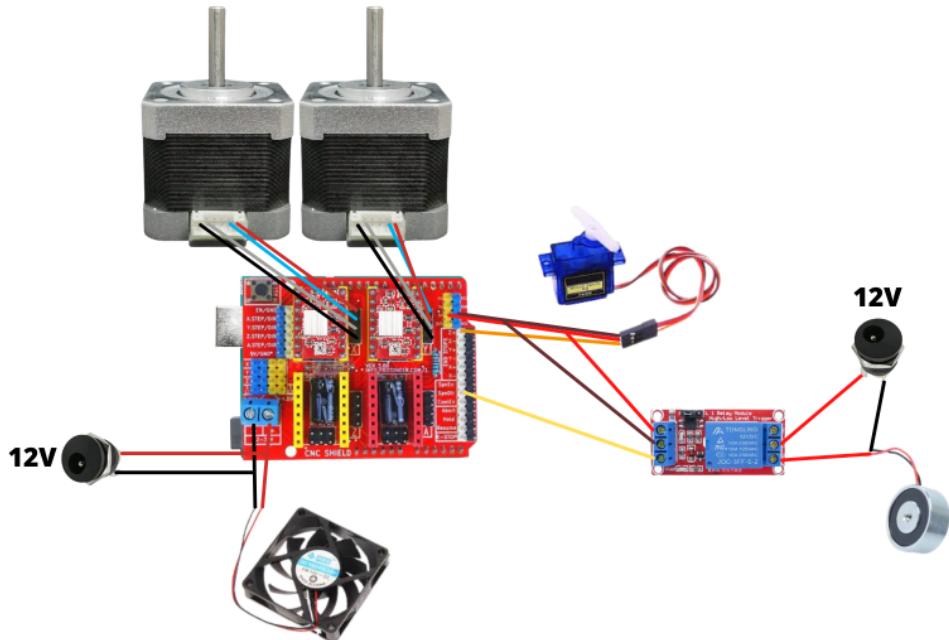
Fonte: Página da empresa Handson Technology (2021)

Cada motor controlado possui um driver A4988, que controla a corrente nas bobinas dos motores e pode ser removido. Além das entradas de alimentação das bobinas dos motores a CNC shield também possui entradas para ligar sensores de fim de curso, relés e até mesmo um micro servo motor.

É possível ligar os motores em diferentes configurações, que estão relacionadas à fração de passo ao qual o motor se movimenta. No projeto DrawingBot, os motores foram ligados na configuração *1/16 Step*, melhor dizendo, cada passo é equivalente a 1/16 de volta do motor. Para maior velocidade e menor precisão em relação ao projeto inicial, os motores do projeto proposto estão ligados na configuração *1/4 Step*, quer dizer, cada passo é equivalente a 1/4 de volta do motor.

Os componentes eletrônicos utilizados no projeto proposto são: um Arduino UNO, uma CNC Shield V3, dois motores de passo Nema 17, um micro servo motor SG90, um relé JQC-3FF-S-Z, uma ventoinha, um eletroímã 12V e duas fontes de alimentação 12V 5A. Uma representação do circuito eletrônico do braço robótico é apresentada na Figura 26.

Figura 26 - Circuito eletrônico



Fonte: Autoria própria (2021)

Os pinos à direita dos drivers A4988 são responsáveis pela alimentação dos motores de passo do eixo X e Y. A única diferença entre a ligação dos motores do projeto é que sempre um motor deve ter uma bobina invertida em relação ao outro motor. Uma fonte de alimentação 12V 5A é ligada diretamente à CNC shield, e possui a função de alimentar os motores de passo e a ventoinha. A ventoinha é necessária para refrigeração forçada dos drivers A4988, o ventilador fica ligado continuamente, dado que está ligado diretamente à fonte de alimentação. O micro servo motor e o relé são ligados na alimentação de 5V e no ground da CNC Shield. Por fim, o eletroímã está sujeito ao acionamento do relé e é alimentado diretamente por outra fonte 12V 5A.

3.3 Visão computacional

Para o desenvolvimento deste projeto foi utilizada a biblioteca OpenCV, mencionada precedentemente, com a linguagem Python. O objetivo deste código desenvolvido foi a captura de imagem do tabuleiro através de uma câmera,

processamento dessa imagem e, por fim, interpretação dela para identificar as peças em jogo.

Para capturar a foto do tabuleiro, foi utilizada a função a seguir:

```
def takePicture():
    cam = cv2.VideoCapture(0)

    camOn = True
    count = 0
    while camOn:
        ret, img = cam.read()

        if not ret:
            break

        if count == 2:
            imgName = "screenshot.png"
            cv2.imwrite(imgName, img)
            camOn = False

        count = count + 1
```

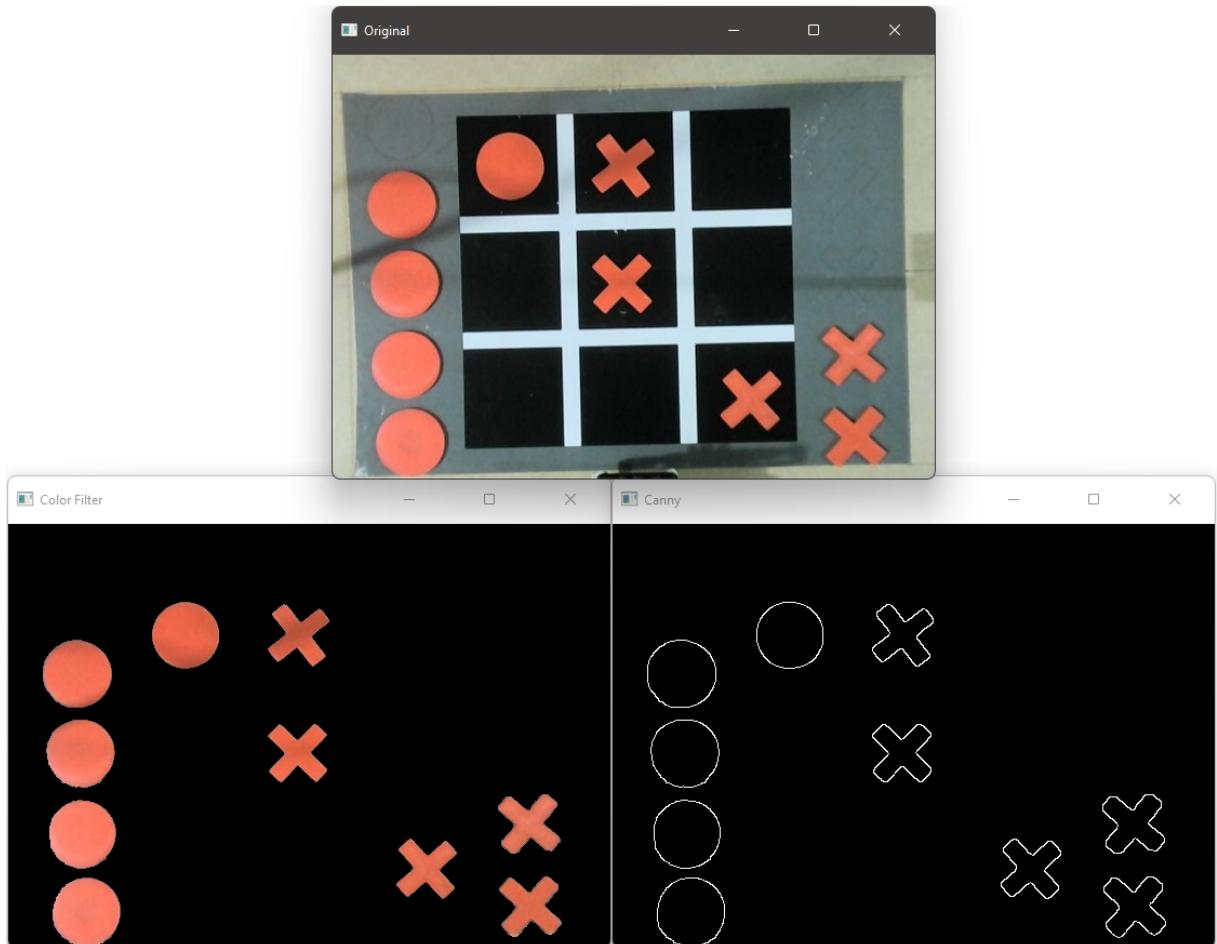
Utilizando a biblioteca mencionada, é capturado o vídeo da câmera ligada ao computador e salvo um *frame* como *screenshot.png*.

Esta imagem é então processada com a função *createPattern()* filtrando apenas a cor das peças do jogo e em seguida aplicando um filtro de detecção de bordas para depois ser possível identificar seus formatos. Obtendo assim o resultado mostrado na Figura 27.

```
def createPattern(img):
    imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lower = np.array([0, 79, 120])
    upper = np.array([179, 255, 255])
    mask = cv2.inRange(imgHSV, lower, upper)
    imgResult = cv2.bitwise_and(img, img, mask=mask)
    imgCanny = cv2.Canny(imgResult, 100, 500)

    return imgCanny
```

Figura 27 - Resultados do processamento da imagem



Fonte: Autoria própria (2021)

Em seguida esses formatos obtidos são comparados a um formato referência de 'X' e 'O' e assim as peças são rotuladas. Com isso é analisada a posição de cada peça para montar uma matriz representando o estado atual do tabuleiro, a qual é enviada para o *script* de minimax.

Neste *script* a seguir são obtidos os contornos das peças no tabuleiro e das referências mencionadas. E então comparados entre si para achar as combinações.

```

contours,           = cv2.findContours(img,           cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
cntX,             = cv2.findContours(patternX,       cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
cntO,             = cv2.findContours(patternO,       cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

```

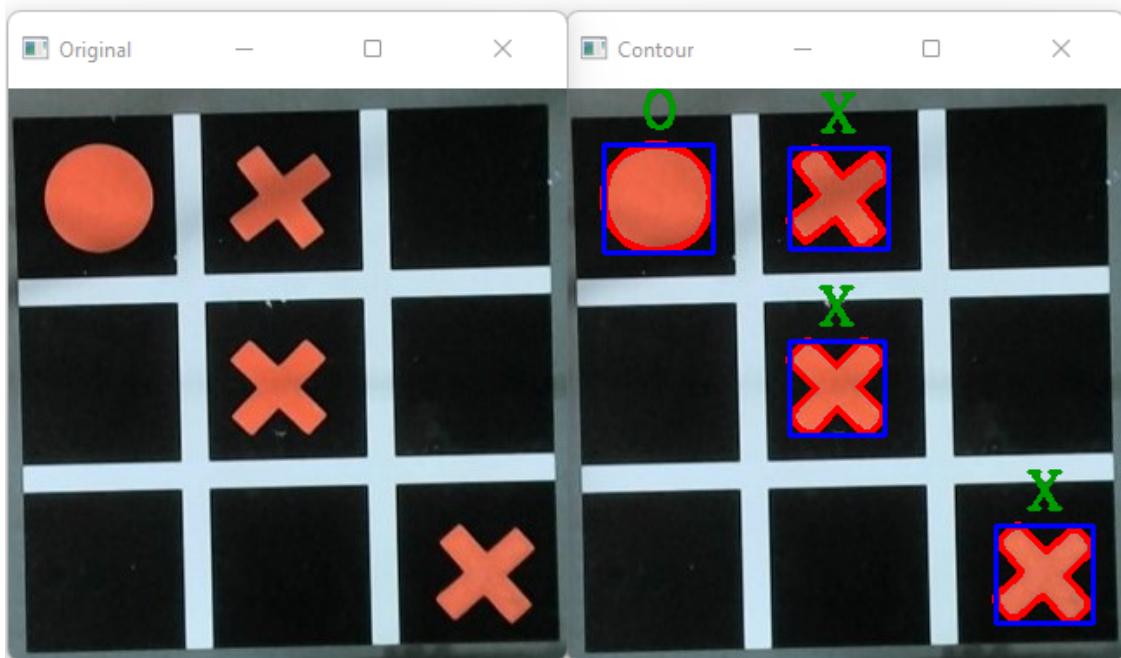
```

for cnt in contours:
    resX = cv2.matchShapes(cntX[0], cnt, 3, 0.0)
    resO = cv2.matchShapes(cntO[0], cnt, 3, 0.0)

```

A imagem é também recortada para obter apenas o que está dentro do tabuleiro, evitando possíveis ruídos de fora. O resultado pode ser visto na Figura 28 com a imagem original na esquerda e a final na direita.

Figura 28 - Resultado final da visão computacional



Fonte: Autoria própria (2021)

3.4 Inteligência artificial

Para a implementação da inteligência artificial, foi utilizado o algoritmo minimax. O algoritmo foi implementado utilizando a linguagem de programação python, porém poderia ter sido implementado em qualquer outra linguagem.

Para representar o estado atual do tabuleiro, foi criada uma variável "board" que é uma matriz 3x3.

```

board = [
    ["_", "_", "_"], # 0
    ["_", "_", "_"], # 1
    ["_", "_", "_"], # 2
] # 0 1 2

```

O jogador é representado pela letra "O" enquanto a IA é representada pela letra "X", e quando o espaço ainda não foi ocupado é utilizado o símbolo "_", como exemplo:

```
board = [
    ["X", "X", "_"], # 0
    ["_", "O", "_"], # 1
    ["_", "O", "X"], # 2
] # 0 1 2
```

Após definirmos o estado atual do tabuleiro, foi criada uma função "findBestMove" que é responsável por executar o algoritmo minimax, a fim de encontrar a melhor jogada para aquele estado do tabuleiro.

Esta função recebe o estado atual do tabuleiro como variável de entrada, e retorna duas variáveis, a variável "bestMove" que é uma lista com dois valores representando as coordenadas da linha e coluna respectivamente da melhor jogada (vale ressaltar que na linguagem python a posição inicial da lista é sempre 0) e a variável "bestVal", que pode ser -10 ou 10, sendo -10 significando a vitória do jogador e 10 significando a vitória da máquina.

```
# This will return the best possible move for the player
def findBestMove(board) :
    bestVal = -1000
    bestMove = (-1, -1)
    # Traverse all cells, evaluate minimax function for
    # all empty cells. And return the cell with optimal
    # value.
    for i in range(3) :
        for j in range(3) :

            # Check if cell is empty
            if (board[i][j] == '_') :

                # Make the move
                board[i][j] = player
                # compute evaluation function for this
                # move.
                moveVal = minimax(board, 0, False)
                # Undo the move
                board[i][j] = '_'
                # If the value of the current move is
                # more than the best value, then update
                # best/
                if (moveVal > bestVal) :
```

```

        bestMove = (i, j)
        bestVal = moveVal
    # print("The value of the best Move is :", bestVal)
    # if (isMovesLeft(board) == False):
    #     bestVal = -1
    if bestVal == -1000:
        bestVal = evaluate(board)

    return bestMove, bestVal, isMovesLeft(board)

```

Ao analisarmos a função, podemos perceber que ela funciona da seguinte maneira:

1. É definida uma variável "bestVal" que serve de referência para memorizarmos a melhor jogada. Possuindo um valor baixo para que possa ser sobreescrita logo na primeira iteração do algoritmo.
2. É definida uma variável "bestMove" que serve de referência para armazenarmos as coordenadas da melhor jogada. Assim como a "bestVal", tem um valor definido que será sobreescrito logo na primeira iteração do algoritmo.
3. É feita uma iteração sobre cada um dos espaços do tabuleiro.
4. Se o espaço estiver disponível, o algoritmo simula o valor do minimax, utilizando a função "minimax" para aquele espaço e calcula um score, que é armazenado na variável "moveVal".
5. O valor do "moveVal" é então comparado com o valor de referência "bestVal", caso seja superior, é um indicativo de que aquela é a melhor jogada, portanto sobreescrivemos as variáveis "bestVal" e "bestMove".

3.4.1 Função minimax

A função minimax é a função responsável por analisar todas as jogadas possíveis e retornar o *score* que é uma pontuação para a melhor jogada para o estado atual do tabuleiro. Recebendo três parâmetros:

- *board* - uma matriz 3x3 que representa o estado atual do tabuleiro.
- *depth* - número de quantidade de jogadas futuras que devem ser analisadas.
- *isMax* - booleano que indica se é a vez da máquina (maximizador) ou não.

Ao analisarmos a função, percebemos o seguinte comportamento:

1. O estado atual do tabuleiro é avaliado pela função "evaluate" que retorna um score.
2. O score é analisado e varia entre -10, 0 e 10. -10 indica que a máquina perdeu, 0 indica que nenhum dos jogadores venceu ou perdeu e 10 indica que a máquina venceu. Caso não haja vencedor, o script continua.
3. A estado atual do jogo é analisada por uma função isMovesLeft, que retorna se há ou não jogadas restantes. Esta função analisa todos os espaços da matriz e verifica se há ainda espaços disponíveis. Caso haja, o script continua.
4. Por fim, é feita a análise de máximos e mínimos. Caso seja a vez da máquina (maximizador). O algoritmo itera sobre cada uma das posições disponíveis da matriz, buscando a jogada com o maior score. Caso contrário, ele faz a iteração buscando a jogada com o menor score.

3.4.2 Função evaluate

A função `evaluate` é a função responsável por atribuir um score (pontuação) para o estado do jogo. Como explicado na função minimax (3.2.1), retorna um valor que pode variar entre -10, 10 e 0. Indicando, derrota, vitória ou nem derrota e nem vitória da máquina. Seu funcionamento consiste em analisar todas as possíveis combinações de vitória ou derrota do jogo da velha e comparar com o estado atual do jogo. Para isso ela faz análises por colunas da matriz, linhas e diagonais. Caso ocorra uma sequência de 3 elementos iguais consecutivos e alinhados, o jogo possui um vencedor.

```
def evaluate(b) :
    # Checking for Rows for X or O victory.
    for row in range(3) :
        if (b[row][0] == b[row][1] and b[row][1] == b[row][2]) :
            if (b[row][0] == player) :
                return player_win
            elif (b[row][0] == opponent) :
                return player_lose
    # Checking for Columns for X or O victory.
    for col in range(3) :
```

```

if (b[0][col] == b[1][col] and b[1][col] == b[2][col]) :

    if (b[0][col] == player) :
        return player_win
    elif (b[0][col] == opponent) :
        return player_lose
# Checking for Diagonals for X or O victory.
if (b[0][0] == b[1][1] and b[1][1] == b[2][2]) :

    if (b[0][0] == player) :
        return player_win
    elif (b[0][0] == opponent) :
        return player_lose
if (b[0][2] == b[1][1] and b[1][1] == b[2][0]) :

    if (b[0][2] == player) :
        return player_win
    elif (b[0][2] == opponent) :
        return player_lose
# Else if none of them have won then return 0
return 1

```

3.5 Integração

Para realizar a integração entre as 4 aplicações (visão computacional, controle CNC, interface gráfica e inteligência artificial) foram criadas APIs individuais. De forma que, a comunicação foi realizada utilizando a rede interna da máquina. Embora para este projeto tenhamos usado um único servidor, seria possível fazer toda a comunicação via internet, de forma que cada aplicação fosse executada em locais físicos diferentes.

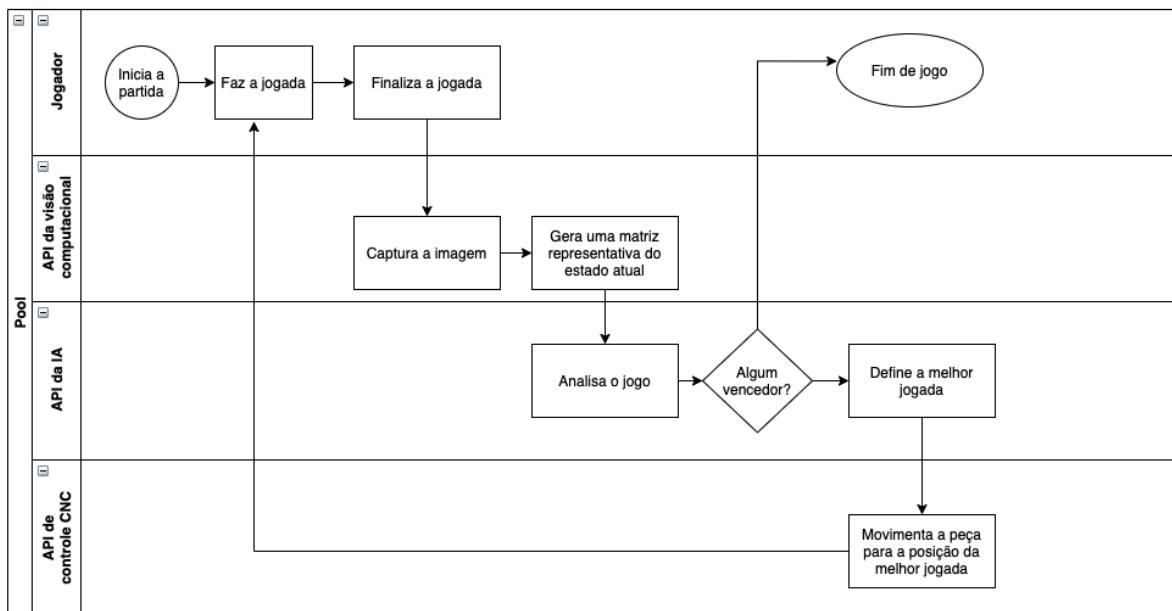
Para isto, utilizamos a biblioteca *Flask* que é um *micro web framework* para desenvolvimento de aplicações web, e a biblioteca *requests*, que é uma biblioteca que permite que sejam feitas requisições HTTP de forma simples.

3.5.1 Fluxo de ações

A comunicação ocorre da seguinte maneira. Primeiramente, a interface gráfica permite que o usuário interaja com a aplicação. A cada jogada é necessário que um botão seja acionado, indicando a finalização da jogada do usuário. A finalização da jogada é um gatilho que faz com que o processo de captura da imagem se inicie e a visão computacional interprete as informações. A visão computacional, então analisa

o estado atual do tabuleiro, gerando uma matriz 3x3 com as informações de cada peça. O estado atual do tabuleiro é enviado para a API da inteligência artificial, que processa as informações e aplica o algoritmo minimax para encontrar a melhor jogada, que é dada por uma linha e uma coluna da matriz. Ainda neste algoritmo, é feita uma análise para definir em qual posição do tabuleiro a CNC deve mover a peça e então envia o resultado para a API de controle da CNC, que por sua vez faz o movimento da jogada. Por fim, o jogador finaliza a jogada pela interface, fazendo com que o ciclo seja reiniciado. Todo este processo pode ser observado na Figura 29.

Figura 29 - Diagrama da integração



Fonte: Autoria própria (2021)

3.5.2 API da visão computacional

No código da API de visão computacional, o estado atual do tabuleiro é capturado e então enviado no formato de uma matriz 3x3.

```

def main(count):
    estado_inicial = visao.main()

    dados_de_envio = {
        'board': estado_inicial,
        'count': count
    }

```

```

print(dados_de_envio)
url_de_envio = ia_url
print('-----')
print('VISAO COMPUTACIONAL-----')
print('-----')
response = post_request(url_de_envio, dados_de_envio)
response = json.loads(response)

estado_final = response.get('novo estado')
print(estado_final)
vencedor = response.get('venceu')
return vencedor

```

3.5.3 API da inteligência artificial

No código da API da IA, o estado atual do tabuleiro é recebido e então é feita a análise da partida e da melhor jogada. Com base na linha e coluna da melhor jogada, é enviado um valor de 1 a 9 que representa as posições do tabuleiro. Este valor será interpretado pela API de controle CNC.

```

app = Flask(__name__)

@app.route('/', methods=['POST'])
def analisar_jogo():
    if request.method == 'POST':
        print('-----')
        print('IA DE ANALISE DE JOGADA---')
        print('-----')
        jogo = request.get_json().get('board')
        count = request.get_json().get('count')
        bestMove,bestVal,hasMove,actualVal,nextVal = findBestMove(jogo)

        vencedor = 0

        print(nextVal)
        if not isMovesLeft(jogo):
            vencedor = -1 # Deu velha

        if nextVal == 10:
            print('Máquina venceu')
            vencedor = 2

        if actualVal == -10:
            print('Jogador venceu')
            vencedor = 1

        result = { "novo estado": jogo, "venceu": vencedor }
        print(result)

        if hasMove and vencedor != 1:
            linha = bestMove[0]
            coluna = bestMove[1]

```

```

jogo[linha][coluna] = "X"

print(f'melhor jogada: {bestMove}')
coordenadas_para_mover = encontrar_coordenadas(linha, coluna)
dados_de_envio = {
    "coordenadas": coordenadas_para_mover,
    "count": count
}
post_request(mquina_cnc_url, dados_de_envio)

return result

def encontrar_coordenadas(linha, coluna):
    if (linha == 0 and coluna == 0):
        return 1
    elif (linha == 0 and coluna == 1):
        return 2
    elif (linha == 0 and coluna == 2):
        return 3
    elif (linha == 1 and coluna == 0):
        return 4
    elif (linha == 1 and coluna == 1):
        return 5
    elif (linha == 1 and coluna == 2):
        return 6
    elif (linha == 2 and coluna == 0):
        return 7
    elif (linha == 2 and coluna == 1):
        return 8
    elif (linha == 2 and coluna == 2):
        return 9

```

3.5.4 API do controle CNC

No código da API de controle CNC, é recebida a posição desejada para que seja realizada a movimentação da peça. Com base nesta posição, envia-se os comandos em GCode para que a máquina execute a seguinte sequência de movimentos:

1. Buscar a peça
2. Ligar o imã
3. Ir para a posição da jogada desejada (com a peça)
4. Desligar o imã
5. Voltar à posição inicial

```

# Open grbl serial port
# Port e Baud q usou no Universal GCode
s = serial.Serial('COM3',115200)

```

```

# Wake up grbl
s.write("\r\n\r\n".encode())
time.sleep(2)      # Wait for grbl to initialize
s.flushInput()     # Flush startup text in serial input

app = Flask(__name__)

@app.route('/', methods=['POST'])
def evaluate_board():
    if request.method == 'POST':
        print('-----')
        print('MAQUINA CNC-----')
        print('-----')
        coordenadas = request.get_json().get('coordenadas')
        count = request.get_json().get('count')

        execute(coordenadas, count)
        print(f'movendo para {coordenadas}')
        print(f'{count}ª jogada')
        return { "coordenadas": coordenadas }

    command_list = {
        "comando_volta": "G28",
        "pos_9": "G4 P3 G0 X11.4861391021805 Y-7.34701958057155",
        "pos_8": "G4 P3 G0 X38.182066565282 Y-12.571164914159",
        "pos_7": "G4 P3 G0 X61.6718333520725 Y-18.1935207175104",
        "pos_6": "G4 P3 G0 X-10.5707293685009 Y-12.4755441667009",
        "pos_5": "G4 P3 G0 X13.3375732879919 Y-18.0921634202235",
        "pos_4": "G4 P3 G0 X37.737393952929 Y-23.6694198686948",
        "pos_3": "G4 P3 G0 X-33.1875757375047 Y-18.0045726809352",
        "pos_2": "G4 P3 G0 X-8.56780022839084 Y-23.6978561274049",
        "pos_1": "G4 P3 G0 X12.3241809829099 Y-29.3168005286991",
        "pc_1": "G4 P3 G0 X0 Y0",
        "pc_2": "G4 P3 G0 X-12.9705627484771 Y-3.27123616632825",
        "pc_3": "G4 P3 G0 X-29.6482322781408 Y-7.19960717292019",
        "pc_4": "G4 P3 G0 X-44.618795026618 Y-11.1708433392484",
        "pc_5": "G4 P3 G0 X-61.9429111656884 Y-15.0206469257085",
    }

def execute(position, count):
    # comandos para cada posição do tabuleiro
    command_switcher = {
        1: (command_list["pos_1"], command_list["comando_volta"]),
        2: (command_list["pos_2"], command_list["comando_volta"]),
        3: (command_list["pos_3"], command_list["comando_volta"]),
        4: (command_list["pos_4"], command_list["comando_volta"]),
        5: (command_list["pos_5"], command_list["comando_volta"]),
        6: (command_list["pos_6"], command_list["comando_volta"]),
        7: (command_list["pos_7"], command_list["comando_volta"]),
        8: (command_list["pos_8"], command_list["comando_volta"]),
        9: (command_list["pos_9"], command_list["comando_volta"]),
    }

    commands = command_switcher.get(position, ("Inválido", "Inválido"))

    gCodeCommandIda = commands[0] + "\n"

```

```

gCodeCommandVolta = commands[1] + "\n"
gCodeCommandPeca = gcode_move_to_piece(count)

# print("Movendo para peça...")
turn_off_magnet()
s.write(gCodeCommandPeca.encode()) # Envia o comando para buscar
peça

# print("Fim do movimento")
turn_on_magnet()

# print("Movendo para jogada...")
s.write(gCodeCommandIda.encode()) # Envia o comando de ida
# print("Fim do movimento")

turn_off_magnet()

# print("Movendo para posição inicial...")
s.write("G4 P3\n".encode())
s.write(gCodeCommandVolta.encode()) # Envia o comando de volta
# print("Fim do movimento")

def gcode_move_to_piece(number):
    take_piece_command_name = f"pc_{number}"
    return command_list[take_piece_command_name]

def turn_on_magnet():
    # print("ligando imã...")
    s.write("M3\n".encode())
    s.write("S1\n".encode())
    # print("imã ligado")

def turn_off_magnet():
    # print("desligando imã...")
    s.write("S0\n".encode())
    s.write("M4\n".encode())
    # print("imã desligado")

```