DESENVOLVIMENTO DE UM SISTEMA ROBÓTICO COM MOVIMENTOS COMANDADOS POR GESTOS IDENTIFICADOS POR VISÃO COMPUTACIONAL[1]

Bruno Benini Lozaldo – loozza@gmail.com
Cauli Kolbe – caukolbe@gmail.com
Gabriel Gomes de Souza – gabriel.gs25@gmail.com
Prof. Dr. Bruno Luis Soares de Lima (Orientador) – bruno.lima@mackenzie.br

RESUMO

Atualmente, com o uso crescente de robôs, e pelo elevado nível educacional necessário para programar e interagir com essas máquinas, se faz necessário o desenvolvimento de plataformas que tornem a interação homem-máquina mais simples e precisa. No presente trabalho, desenvolveu-se um sistema de controle para manipuladores robóticos baseado em visão computacional e processamento de imagens. Foi desenvolvido um algoritmo em *Phyton* baseado na biblioteca de código aberto *OpenCV*, que identifica comandos gestuais capturados por uma webcam e, a partir de uma interface com o usuário, os transforma em movimentos específicos a serem realizados por um protótipo de manipulador que foi projetado e construído pelos autores. O protótipo, construído em MDF, possui alcance de 180 mm e 3 graus de liberdade, e seus movimentos são realizados a partir do acionamento de 3 motores de passo. O acionamento dos motores se dá a partir de um *firmware* que foi desenvolvido para o microcontrolador *Arduino*, o qual recebe o comando gestual que foi detectado. Foi possível estabelecer condições críticas de funcionamento do sistema desenvolvido com relação a luminosidade ambiente, distância do operador em relação a *webcam* e a precisão mecânica da resposta motora do manipulador. A partir das tecnologias aplicadas neste trabalho poderá ser possível o desenvolvimento de sistemas de controle por gestos para outros mecanismos robóticos.

Palavras-chave: Projeto de manipulador robótico. Construção de manipulador robótico. Controle gestual. Visão computacional com OpenCV.

DEVELOPMENT OF A ROBOTIC SYSTEM WITH MOVEMENTS COMMANDED BY GESTURES IDENTIFIED BY COMPUTER VISION

[1] Artigo do Trabalho de Conclusão de Curso, Graduação em Engenharia Elétrica, EE, UPM, São Paulo, 2019.

ABSTRACT

Nowadays, with the increasing use of robots, and the high level of education required to program and interact with these machines, it becomes necessary to develop platforms that make the human-machine interaction simpler and more precise. In the present work, a control system was developed for robotic manipulators based on computer vision and image processing. An algorithm was developed in *Phyton* based on the *OpenCV* open source library, which identifies gesture commands captured by a webcam and, from a user interface, transforms them into specific movements to be performed by a prototype manipulator that was designed and built by the authors. The prototype, built in MDF, has a range of 180 mm and 3 degrees of freedom, and its movements are made from the activation of 3 step motors. The activation of the motors takes place from a firmware that was developed for the *Arduino* microcontroller, which receives the gestural command that was detected. It was possible to establish critical operating conditions of the developed system in relation to the ambient brightness, distance from the operator to the webcam and the mechanical accuracy of the manipulator's motor response. From the technologies applied in this work it may be possible to develop control systems by gestures for other robotic mechanisms.
Keywords: Robotic arm design. Robotic arm building. Gesture control. Computer vision with OpenCV.

1 INTRODUÇÃO

Nas últimas décadas, os robôs vêm cooperando com o homem na execução de inúmeras tarefas, que vão desde tarefas simples de manipulação em chão-de-fábrica, até tarefas complexas, como no caso de cirurgias médicas. Em aplicações que envolvem alta periculosidade, por exemplo, é possível o acesso remoto através de braços robóticos ou robôs móveis, o que pode mitigar os riscos ao ser humano inerentes a essas aplicações. Em 2016, a Associação de Indústrias de Robótica (RIA) (ROBOTIC INDUSTRIES ASSOCIATION, 2016) descreveu quatro situações nas quais o acesso remoto de robôs se destaca: exploração marinha, exploração do espaço, soldagem e tarefas industriais, e na resposta a desastres, como desabamentos e inundações.

A indústria atual é baseada em grandes linhas de montagem para produção em massa. Isto ocasiona a necessidade de grandes complexos, que tendem a crescer conforme aumenta a demanda da sociedade por produtos.

Cada uma das grandes mudanças de paradigmas que ocorreram na história da indústria, devido às mudanças tecnológicas e à inovação, constituiu uma Revolução Industrial. Essas revoluções foram causadas (BENESOVÁ; TUPA, 2017) pela mecanização da produção (Primeira Revolução), pelo uso

da energia elétrica (Segunda Revolução) e pelo uso da Eletrônica e da Automação (Terceira Revolução).

Ainda segundo Benesová e Tupa (2017), uma nova grande mudança tem ocorrido na indústria com a digitalização e o uso acentuado da Robótica. Esse fenômeno tem sido chamado de "Quarta Revolução Industrial", ou "Indústria 4.0". Nesse contexto, uma profunda transformação na sociedade está ocorrendo devido a robotização da manufatura, permitindo a atuação em espaços confinados e um alto grau de precisão, a "Internet das Coisas (IoT)", a "Internet dos Serviços" e a "Internet das Pessoas", conectando à rede humanos e máquinas / objetos.

Esse cenário pode ser demonstrado, por exemplo, pela fábrica da montadora de veículos BMW em Leipzig, na Alemanha. O local possui a atuação de mais de mil manipuladores robóticos, e em 2018 contou com o investimento de mais 300 milhões de Euros para elevar a produção anual a 350 mil veículos (BLABST, 2018).

Segundo afirmação da Federação Internacional de Robótica (IFR), 3 milhões de robôs industriais serão utilizados em fábricas em todo o mundo até 2020. Isso significa que o estoque operacional irá mais que dobrar dentro de sete anos. (HEER, 2018).

Entretanto, um dos pontos negativos desta evolução decorre da necessidade de uma alta capacitação de operadores; sendo assim, apenas uma pequena parcela dos profissionais está qualificada a atuar.

De acordo com Owen-Hill (2016), os programadores de robôs utilizam diversas linguagens de programação para programar os movimentos do robô. Existem as linguagens proprietárias dos fabricantes de robôs, e basicamente cada fabricante possui a sua. O fabricante ABB possui a linguagem de programação RAPID, o fabricante Kuka possui a linguagem KRL (*Kuka Robot Language*), o fabricante Comau possui a linguagem PDL2, o fabricante Yaskawa possui a linguagem INFORM e o fabricante Kawasaki usa a linguagem AS. Pode-se citar também os fabricantes Fanuc (usa a linguagem Karel), Stäubli Robots (usa VAL3) e o fabricante Universal Robots (usa URScript). A diversidade de linguagens e a falta de padronização é considerado um problema. Linguagens de uso geral como C/C++, Java, Python e C#/.NET são muitas vezes utilizadas pelos programadores. Outra opção interessante é o uso da plataforma MATLAB e sua *toolbox* de Robótica. Por fim, ainda é possível a utilização de linguagens de descrição de hardware, como VHDL, ou ainda linguagens de baixo nível como Assembly.

Analogamente, a Robótica, dentro do campo da Visão Computacional, obteve altos investimentos nos últimos anos. A *Association for Advancing Automation* (A3) constatou um gasto de US$ 709 milhões no primeiro trimestre de 2018 (ARBOR, 2018).

Desta forma, com o contexto atual, de crescente automatização de processos industriais e transformação digital (ZEZULKA et al., 2016), se faz cada vez mais necessário o desenvolvimento

de plataformas robóticas controladas a distância, com relativo baixo custo e com interface de programação amigável.

A principal perspectiva sobre a indústria se dá pela ascensão das fábricas inteligentes. Em tais fábricas, suas máquinas, sensores e internet das coisas são conectadas por Sistemas Ciber-físicos (em inglês, CPS's). A quarta revolução está transformando não somente o chão-de-fábrica, mas também a educação e o mercado de trabalho; sendo assim, algumas profissões já estão desaparecendo (BENESOVÁ; TUPA, 2017).

A Figura 1 apresenta de maneira geral o sistema proposto no trabalho.

Figura 1 – Concepção geral do sistema proposto
Fonte: Elaborado pelos autores

O objetivo geral deste trabalho é desenvolver um sistema de controle inteligente de um manipulador robótico comandado através de gestos detectados por um algoritmo de visão computacional.

Os objetivos específicos serão:

- Definir a cinemática para projeto do braço robótico;
- Projeto e construção de braço robótico que deverá ser manipulado pelo sistema de controle a ser desenvolvido;
- Captura dos gestos realizados por um operador através de uma câmera *(webcam)* e desenvolvimento de um algoritmo que realize o reconhecimento dos gestos com a utilização da biblioteca de visão computacional *OpenCV*;
- Desenvolvimento de um algoritmo para acionamento do manipulador robótico a partir dos gestos que foram reconhecidos;
- Implementar o sistema de automação proposto e realizar sua validação.

Segundo a *Robotic Industries Association* (RIA), o robô industrial é definido como sendo um "manipulador multifuncional reprogramável projetado para movimentar materiais, partes, ferramentas ou peças especiais, através de diversos movimentos programados, para o desempenho de uma variedade de tarefas" (GROOVER et al., 1988).

Os robôs manipuladores industriais vêm sendo cada vez mais utilizados nas últimas décadas nos processos de automação programável e automação flexível. Notáveis avanços na tecnologia da robótica vêm sendo observados, com incrementos na confiabilidade, robustez, versatilidade e redução de custos.

As aplicações dos Robôs industriais são inúmeras e incluem operações de soldagem, montagem de componentes (como nas indústrias eletrônica e automotiva), operações de empacotamento, operações diversas de movimentação e transporte de peças, produção de placas de circuito impresso, operações de pintura de superfícies, operações de limpeza com jatos de água e abrasivos, operações de corte, fixação de parafusos e rebites, além de funções relacionadas a controle de qualidade, como inspeção e verificação dimensional de peças (GROOVER, 2011).

1.1 VISÃO COMPUTACIONAL E A BIBLIOTECA OPEN CV

Segundo Pulli et al. (2012), a Visão Computacional é um campo dedicado a análise, modificação e reconhecimento em alto nível de imagens. Seu objetivo é determinar o que está acontecendo em frente a uma câmera, e usar esse entendimento para controlar um sistema computacional ou robótico, ou prover às pessoas novas imagens que sejam mais informativas ou esteticamente mais úteis que as imagens originais.

Em aplicações de visão computacional, é largamente utilizada a biblioteca *OpenCV* (*Open Source Computer Vision Library*), pois trata-se de uma biblioteca de programação de código aberto, com inúmeros algoritmos de visão computacional disponíveis na literatura e vasta documentação. Ela foi desenvolvida inicialmente como um projeto de pesquisa pela empresa *Intel Corporation* em 1998, e está disponível desde 2000. (PULLI et al., 2012).

OpenCV provê as ferramentas necessárias para resolver problemas de visão computacional. A biblioteca contém um rol de funções de processamento de imagens em baixo nível e algoritmos em alto nível, tais como: detecção facial, detecção de pedestres, sensoriamento, entre outros (PULLI et al., 2012).

No processamento de imagens temos um processo que resulta em um conjunto de valores numéricos, a partir de uma entrada em forma de imagem. Esse resultado pode compor ou não uma outra figura. Entretanto, a visão computacional trabalha com o objetivo de reproduzir a visão humana. Portanto, é um procedimento que acarreta em uma interpretação da imagem colocada na entrada do sistema.

Marengoni e Stringhini (2009) apresentam um exemplo interessante que ilustra algumas diferenças entre o processamento de imagens e a visão computacional.

Figura 2 – Fotografia de um veículo antes e depois do processamento da imagem
Fonte: Marengoni e Stringhini (2009)

A Figura 2 mostra duas imagens. A imagem da esquerda mostra um veículo, contudo não é possível ler a placa do mesmo pois a imagem está muito escura. É então aplicado uma operação de processamento de imagens, resultando na imagem da direita, a qual é mais clara e permite a leitura da placa do veículo.

A *OpenCV* divide-se nos seguintes grupos: análise de movimento e rastreamento de objetos; processamento de imagens; análise estrutural; reconhecimento de padrões e calibração de câmera e reconstrução 3D (MARENGONI; STRINGHINI, 2009).

Ainda de acordo com Marengoni e Stringhini (2009), as aplicações de visão computacional precisam de uma etapa de pré-processamento, englobando o processamento da imagem. Na maioria das vezes, as imagens das quais se quer extrair alguma informação precisam ser convertidas para algum formato ou tamanho específico e precisam ainda ser filtradas para eliminar eventuais ruídos.

A Figura 3 apresenta o processo de segmentação, que consiste em particionar uma imagem em diversas regiões, usando como base características peculiares de cada região, como por exemplo, mudanças de cor. O grau de precisão possível de ser alcançado depende da resolução da imagem e de qual tarefa está sendo realizada (MARENGONI; STRINGHINI, 2009).

Uma maneira possível de realizar a operação de segmentação é a segmentação por detecção de borda, a qual detecta uma mudança abrupta no nível de intensidade dos pixels, e quando esses *pixels* estão próximos eles podem ser conectados formando uma borda ou um contorno, definindo assim uma região ou um objeto na imagem (MARENGONI; STRINGHINI, 2009).

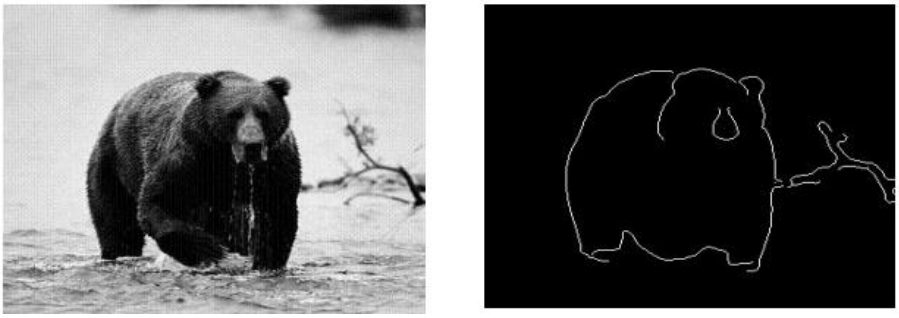

Figura 3 – Exemplo do processo de segmentação de uma imagem
Fonte: Marengoni e Stringhini (2009)

No presente trabalho, foi feita a implementação de uma aplicação de visão computacional, que consiste no desenvolvimento de um algoritmo baseado na biblioteca *OpenCV*, que permite o reconhecimento de gestos, o que tornou possível o envio de comandos específicos para a movimentação do braço robótico.

3. MATERIAIS E MÉTODOS

3.1 VISÃO GERAL E ARQUITETURA DO SISTEMA

3.1.1 Detecção da cor e posição do objeto

O presente trabalho aborda o desenvolvimento de um sistema de controle para manipuladores robóticos baseado em comandos gestuais, utilizando um objeto que será detectado por meio de visão computacional. Essa tecnologia aplicada possui como objetivo a obtenção de informações de uma imagem através da utilização de *hardware* e *software*, replicando a visão humana.

O objeto a ser detectado é uma bola verde que será posicionada em frente a uma *webcam*. Quando este objeto for identificado, de acordo com o seu posicionamento, um comando específico será enviado para os motores do manipulador, que realizará um movimento correspondente a este comando. A Figura 4 demonstra os quadrantes de ação disponíveis ao usuário.

Os quadrantes à esquerda (com o símbolo positivo) indicam o movimento do motor no sentido horário e os quadrantes à direita (com o símbolo negativo) indicam o movimento do motor no sentido anti-horário.

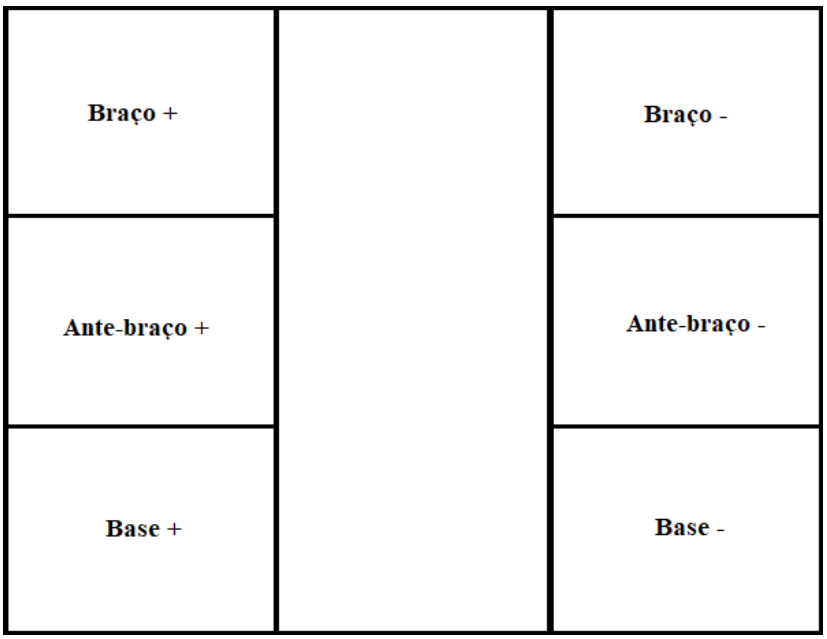

Figura 4 - Esquema dos quadrantes de comandos de movimento disponíveis ao usuário.
Fonte: Elaborado pelos autores

A Figura 5 apresenta um diagrama que ilustra o processo de acionamento do manipulador robótico a partir da detecção de uma imagem. Inicialmente, a bola verde é colocada diante de uma *webcam* e é usada para determinar o comando a ser executado, de acordo com os quadrantes mostrados anteriormente. O algoritmo de detecção de imagens escrito em *Phyton*, baseado na biblioteca OpenCV, torna possível a localização da posição da bola e seu respectivo centro. Um vez determinado o quadrante escolhido, o algoritmo retorna o comando apropriado, a ser enviado para o microcontrolador que realizará o acionamento dos motores.

O processo de detecção do objeto inicia-se com o uso da biblioteca *OpenCV*, pois entre as suas funcionalidades, esta biblioteca permite o processo de "mascarar" a imagem de forma a "discretizar" a mesma em *pixels* que corresponderão a uma certa regra, no qual resultará na divisão da imagem em regiões.

Realiza-se então o rastreamento dos *pixels* referentes a faixa de cores RGB (*Red*, *Green*, *Blue*) preestabelecida, que irá produzir o efeito binário na imagem em que o *pixel* pertencente a regra possuirá o valor de 1, permanecendo branco, e o pixel não pertencente terá o valor de 0, permanecendo preto.

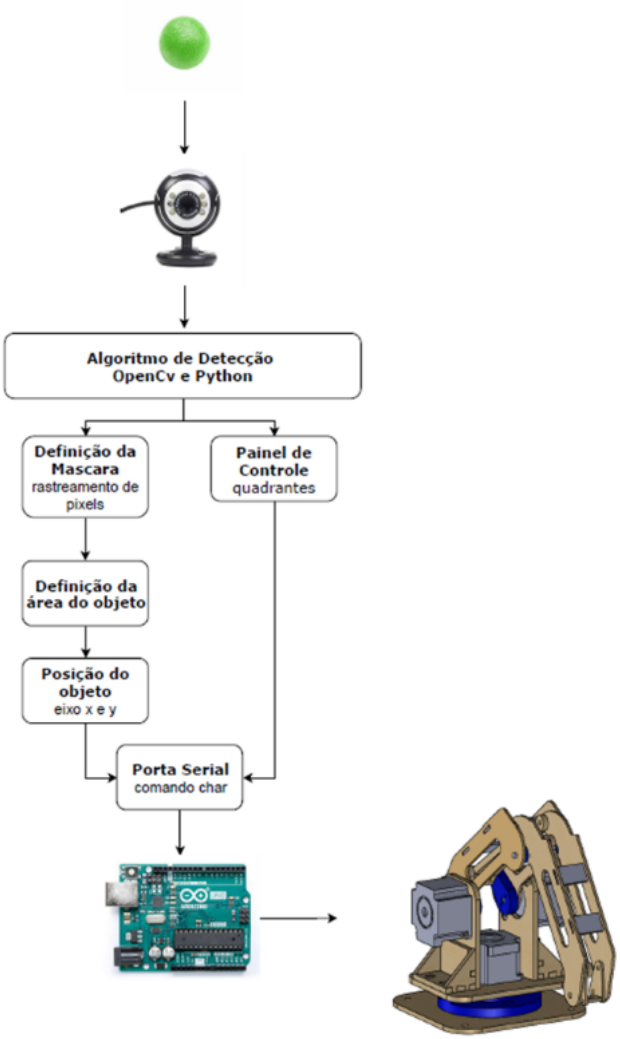

Figura 5 – Visão geral do processo de detecção da imagem, levando ao acionamento do manipulador robótico
Fonte: Elaborado pelos autores

A partir do agrupamento dessas regiões se produzirá a segmentação da imagem, processo aplicado em visão computacional para a simplificação da interpretação de imagem e utilizado para localização dos contornos de objetos. Esse processo torna possível a aplicação do conceito de momento invariante da imagem, técnica presente em processamento de imagens que permite, através de cálculos, a definição da área total de um objeto e consequentemente o centro geométrico do corpo referenciado nos eixos X e Y.

Dessa forma, a identificação da posição do objeto localizado no quadrante alvo sucederá em um comando a ser escrito na porta serial do microcontrolador, que por sua vez acionará os motores, levando a um movimento específico do robô.

Conforme apresentado, o controle realiza a conversão do posicionamento do objeto em uma variável *char*, e essa é enviada pela porta serial ao microcontrolador, e, pela aplicação da modulação de largura de pulso, posicionará corretamente o manipulador.

Outra frente de desenvolvimento do trabalho foi a construção de um manipulador robótico para efeito de testes do controle inteligente. Este manipulador robótico, totalmente desenvolvido pelo grupo desde o seu projeto até a sua construção, será controlado por um microcontrolador que deverá obter comandos através da porta serial.

3.1.2 Componentes eletrônicos, interfaces e módulos

Um *notebook* captura as imagens através de sua *webcam* e ainda implementa o algoritmo de detecção do objeto. A partir da detecção realizada, um caractere específico é enviado pela porta serial ao microcontrolador.

O microcontrolador utilizado neste projeto é o *Arduino*, mais especificamente o modelo *Arduino Uno*. O Arduino é uma plataforma de *hardware* e *software open-source* projetada para ser relativamente fácil de usar. Neste projeto, o microcontrolador realiza a leitura do caractere recebido através da sua interface serial, e de acordo com o mesmo, realiza o acionamento da *CNC SHIELD* de acordo com o algoritmo de acionamento desenvolvido. Ele é conectado a três módulos TB6600, que serão responsáveis por converter esses comandos nas tensões e correntes adequadas ao acionamento dos três motores de passo *NEMA* 23 de 2A nominais.

Os módulos são alimentados por uma fonte chaveada 12 VDC de até 10A nominais. A Figura 6 mostra a arquitetura do sistema e a aparência dos componentes.

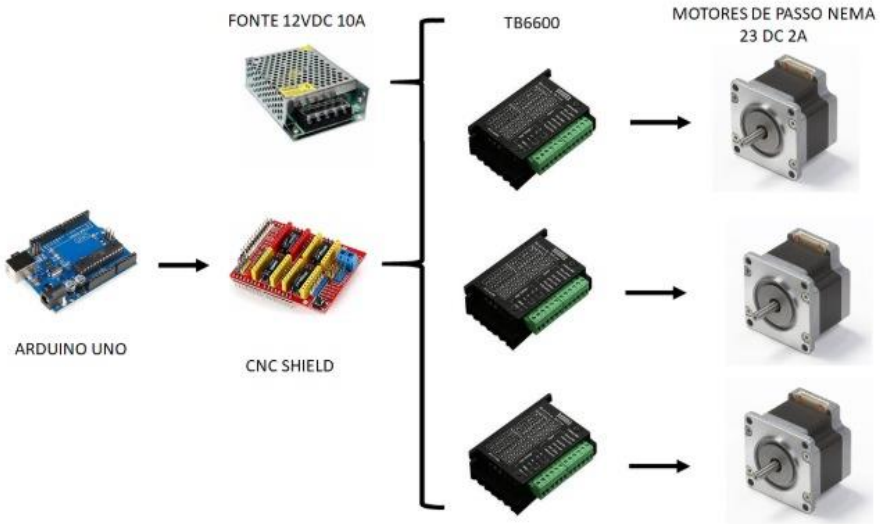

Figura 6 – Arquitetura geral dos componentes eletrônicos e dos motores de passo utilizados no projeto
Fonte: Elaborado pelos autores

3.2 PROJETO E FABRICAÇÃO DO MANIPULADOR ROBÓTICO

A construção do manipulador robótico para teste viabilizou o desenvolvimento do controle inteligente. Isso por ter possibilitado testes, ajustes e aplicação do mesmo.

Neste tópico são apresentadas a cinemática e modelamento e construção do manipulador robótico.

3.2.1 Cinemática e Modelamento

Para estudo das dimensões, modelamento e movimentação foi utilizado o *software CAD 3D Solidworks*, caracterizado por uma interface gráfica objetiva e inúmeras ferramentas.

A estrutura principal do protótipo é formada pela base, braço, antebraço e suporte de fixação do atuador. Também possui como sua estrutura principal três graus de liberdade. O primeiro grau para rotação do manipulador em sua base, o segundo grau para movimentação do braço e o terceiro grau para movimentação do antebraço.

Com o objetivo da manutenção do suporte de fixação do atuador paralelo ao piso, realizou-se um esboço no *Solidworks* definindo assim o comprimento do braço em 190 mm e os seus entre-furos com 142 mm, antebraço com 212 mm e seus entre-furos com 170 mm.

A Figura 7 apresenta o dimensionamento e dimensões em milímetros do manipulador robótico.

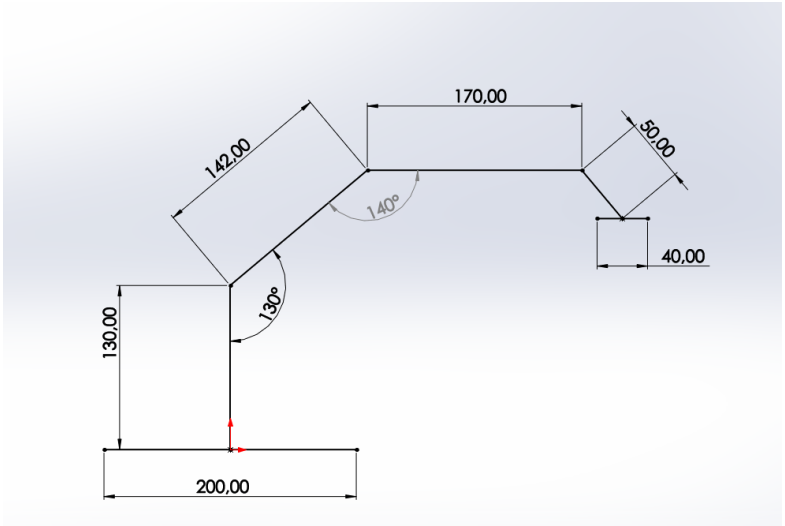

Figura 7 – Dimensionamento do tamanho das estruturas principais, formadas pela base, suporte dos motores, braço e antebraço.
Fonte: Elaborado pelos autores

Dispondo do tamanho das estruturas principais, realizou-se o modelamento do manipulador robótico no software de *CAD 3D*.

A Figura 8 à esquerda apresenta uma concepção do manipulador, com a localização dos motores evidenciando os três graus de liberdade para os movimentos do robô.

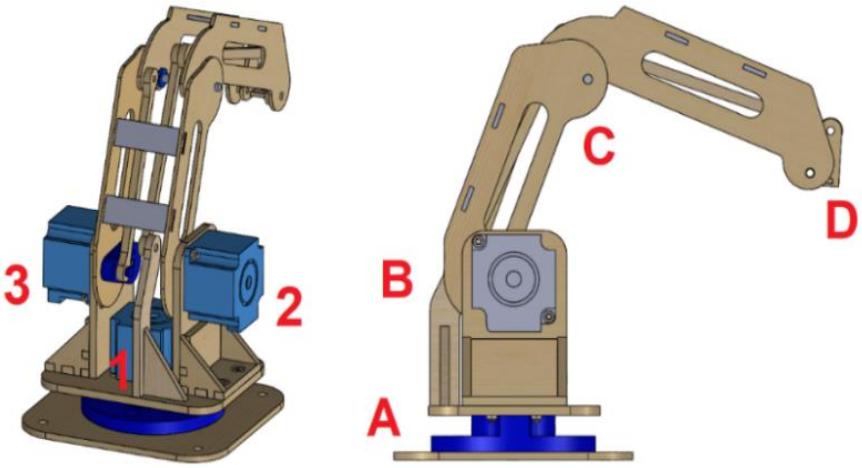

Figura 8 – Manipulador robótico modelado e os três motores destacados referentes aos três graus de liberdade (esq.) e segmentação do braço para cálculo dos torques (dir.)
Fonte: Elaborado pelos autores

A partir desta estrutura apresentada, adicionou-se uma estrutura secundária formada por suportes de fixação e três terminais de rótulas. Dois desses terminais possuem comprimento 140 mm e o terceiro possui comprimento 142 mm. Estes componentes permitiram a de fixação do motor referente ao terceiro grau de liberdade, na base, paralelo ao motor do segundo grau de liberdade.

Na Figura 8, à direita, mostra-se a segmentação do braço para o cálculo dos torques. Na Tabela 1 são apresentados os dados necessários para o cálculo dos torques presentes nas articulações: o peso em gramas e o comprimento em milímetros de cada um dos segmentos.

Tabela 1 – Dados para o cálculo dos torques presentes nas articulações

LIGAÇÕES	PESO (GRAMAS)	COMPRIMENTO (MM)
AB	800	140
BC	90	142
CD	100	170

Fonte: Elaborado pelos autores

De posse desses valores, os torques dos motores 1, 2 e 3 foram calculados como 16,2; 9,71 e 3,37 N.m, respectivamente. A partir disto, foi definido para montagem do manipulador robótico um motor de 5 kgf ou aproximadamente 50 N.m, ou seja, houve um superdimensionamento. Este fato deve-se a disponibilidade de motores comerciais com esta especificação, não foram encontrados motores com torque mais próximos dos calculados. Com o torque definido é possível obter uma movimentação adequada para o braço.

3.2.2 Construção do manipulador robótico

Através das informações e modelamento obtido no item anterior, realizou-se a fabricação do manipulador. Para tal, foram utilizadas duas placas de *MDF* com 3 e 6 mm de espessura, respectivamente. As placas foram cortadas na máquina a *laser* do Laboratório de Mecânica da Universidade Presbiteriana Mackenzie.

Além do *MDF*, utilizou-se pinos confeccionados pela impressora *3D* do Laboratório de Engenharia Elétrica da universidade, bem como parafusos e porcas. Essas estruturas localizam-se nas junções e suportes do protótipo, sendo dimensionadas para evitar a presença de folga.

O protótipo como resultado, o manipulador obteve um alcance de 180 mm, com altura máxima da estrutura em 280 mm e peso total em 1,2 Kg.

3.3 ALGORITMO PARA DETECÇÃO DE IMAGENS

Este item detalha a metodologia utilizada para o desenvolvimento do algoritmo de detecção de um objeto, conforme sua cor e seu posicionamento. Desenvolvido em linguagem de programação *Python* (FOUNDATION, 2016a), o algoritmo desenvolvido faz uso das diversas bibliotecas (também chamadas de módulos) disponíveis para a linguagem. Estas bibliotecas são capazes de auxiliar nas mais diversas tarefas, como: contar o tempo, localizar determinadas cores em espaços de cor distintos no quadro, desenhar circunferências, detectar centro geométrico de formas presentes, redimensionar o tamanho do quadro, processar a imagem para facilitar a detecção, entre outros.

O algoritmo desenvolvido coleta os comandos para movimentar o braço robótico, primeiro identificando a cor parametrizada que determinará o movimento a ser detectado. Após isto, o algoritmo identifica a imagem e desenha no quadro uma circunferência aproximada. Em seguida, transmite o comando para o microcontrolador *Arduino* via porta *serial*, e então ocorre o acionamento do braço robótico.

O objeto a ser detectado pelo algoritmo, para determinar os movimentos, é uma bola de cor verde. O movimento deste objeto em relação aos seis quadrantes exibidos na tela determinará os movimentos do braço robótico.

Na Figura 9, no início do fluxograma à esquerda, mostra-se a preparação do ambiente básico para o funcionamento do algoritmo desenvolvido, que consiste na importação das bibliotecas necessárias e na criação de variáveis, como o limite superior e inferior *HSV* (*Hue*, *Saturation* e *Value*, em inglês) da cor verde. O sistema *HSV* é definido por três valores de variável: matiz, saturação e valor. A Matiz corresponde a mistura de cores identificada; a Saturação é um parâmetro que determina a qualidade do matiz, e o Valor é o parâmetro que define o brilho da cor.

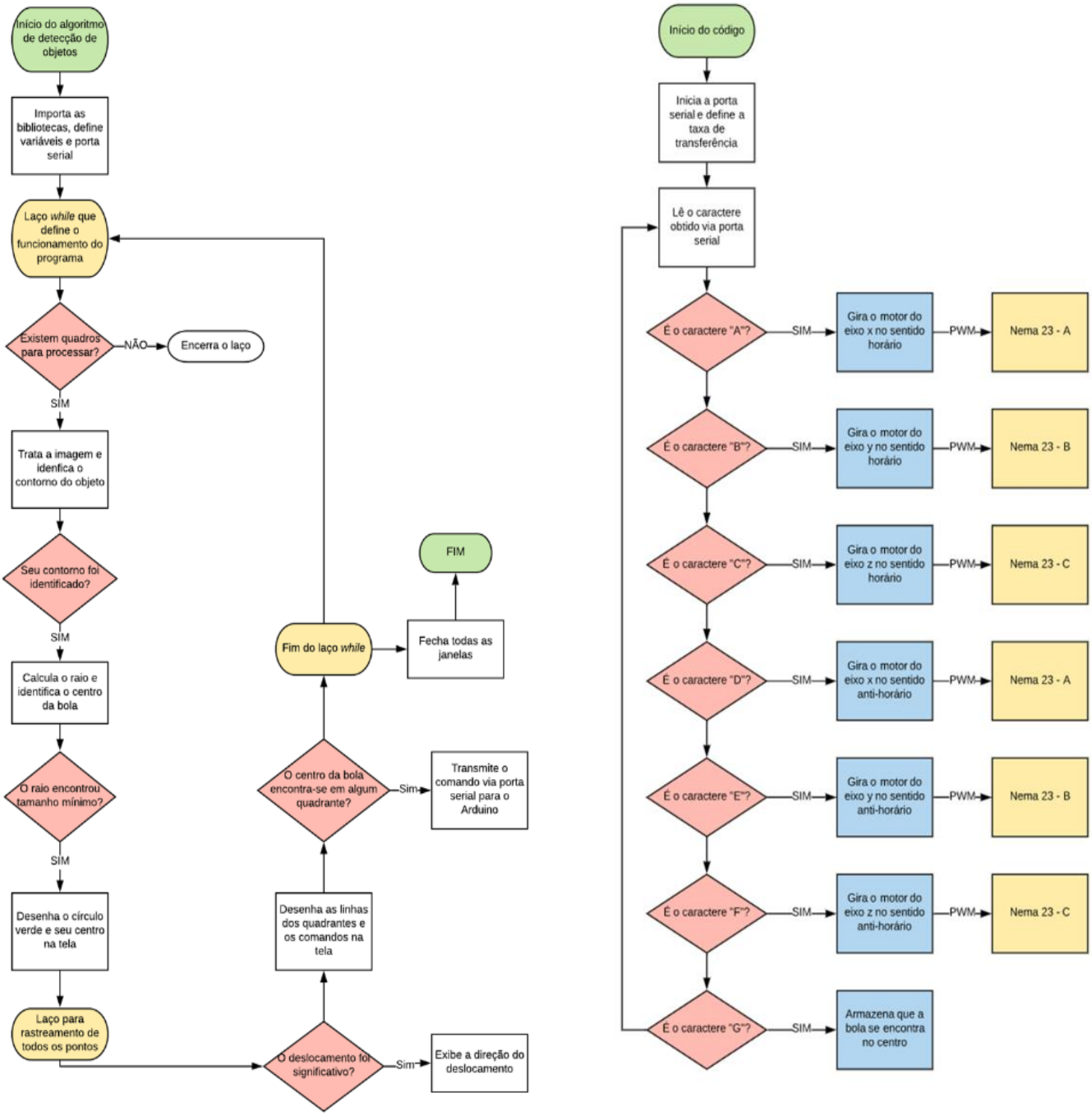

Figura 9 – Fluxogramas que descrevem a modelagem dos softwares desenvolvidos. À esquerda, o *software* escrito em *Phyton* para a detecção de imagens. À direita, o *firmware* desenvolvido para o microcontrolador *Arduino*, para o acionamento dos motores.
Fonte: Elaborado pelos autores

Nos fluxogramas, os balões em branco representam ações, os balões em vermelho representam as condições a serem analisadas pelos algoritmos, os balões em azul representam as decisões tomadas pelo *firmware* do microcontrolador e os balões em amarelo representam, no fluxograma da direita, os acionamentos dos motores do robô, e, no fluxograma da esquerda, o início e o fim dos laços do algoritmo. Os balões em verde representam o início ou o fim dos algoritmos.

Definiu-se que o objeto a ser detectado possui a cor verde, pois essa cor possui fácil percepção na maioria dos ambientes (DRISCOLL, 2017). Existem inúmeras combinações de valores que são reconhecidas como algum tipo de verde. Sendo assim, é definido um intervalo de valores – um limite inferior e um limite superior – dentro do qual se considera a como sendo a cor verde.

A Figura 10 apresenta, na parte de cima, dois quadros obtidos a partir da câmera. O primeiro mostra, à esquerda, a imagem bruta e o painel de controle já detalhado anteriormente. À direita é mostrada a máscara obtida.

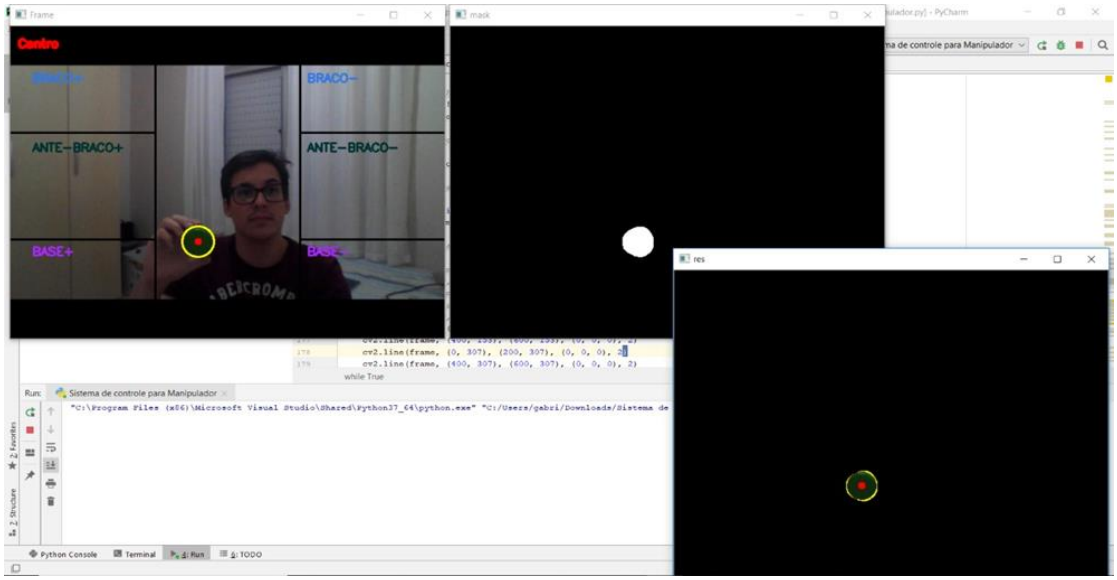

Figura 10 – Captura final dos quadros processados pelo algoritmo. Da esquerda para a direita: a saída principal, com o desenho dos quadrantes e dos comandos disponibilizados; a máscara de identificação de cores, e a máscara de identificação com o objeto recortado.
Fonte: Elaborado pelos autores

Em seguida, o raio da circunferência é calculado e o seu centro geométrico (centroide) é identificado. O programa coleta inúmeras amostras de raio do objeto identificado, já que algumas das leituras coletadas podem apresentar ruídos. Assim, as leituras de raio são comparadas de forma a se obter o menor raio possível para objeto. Com raio e centro obtidos, desenha-se no quadro a circunferência verde no entorno do objeto identificado, assim como o ponto respectivo ao centro da figura. Esse fato pode ser observado na Figura 10, na parte de baixo, à direita.

Após a determinação do centro do objeto, toma-se a decisão de informar na tela a direção do movimento realizado pelo usuário que comanda o sistema através da *webcam* (apenas caso houver deslocamento significativo realizado pela bola). O algoritmo reconhece comandos dados pelo usuário a partir do movimento do objeto verde (bola) em 4 direções, sendo: norte, sul, leste e oeste.

São desenhadas ao todo quatro linhas para delimitar os quadrantes; sendo assim, são nove quadrantes ao todo. Os quadrantes adjacentes às bordas laterais representam os comandos a serem realizados pelo braço robótico, totalizando sete comandos – dois para cada um dos motores presentes

no braço. Já os três quadrantes centrais emitem a variável *G* para sinalizar o algoritmo quando a bola se encontrar nos três quadrantes centrais.

Na última decisão apresentada no fluxograma à esquerda da Figura 9, encontra-se um dos aspectos principais do desenvolvimento do algoritmo, que consiste em identificar se a bola se encontra em algum dos quadrantes determinados. Caso seja verdade, é transmitido um caractere via porta *serial* para a plataforma *Arduino.* Foi estabelecido um protocolo de comunicação entre o algoritmo de detecção de imagens e movimentos para a placa o microcontrolador responsável por enviar os comandos aos motores do manipulador robótico. Como apresentado na Tabela 2, cada caractere representa a identificação de um comando a ser dado aos motores do manipulador robótico de acordo com a figura do painel do usuário apresentado nas Figuras 4 e 10.

Tabela 2 – Relação entre os comandos enviados via porta *serial* e os movimentos realizados pelo manipulador robótico. O caractere *G* não se relaciona com o motor.

CARACTERE DE COMANDO	MOTOR	SENTIDO
A	Eixo X	Horário
B	Eixo Y	Horário
C	Eixo Z	Horário
D	Eixo X	Anti-horário
E	Eixo Y	Anti-horário
F	Eixo Z	Anti-horário
G	-	-

Fonte: Elaborado pelos autores

3.4. *FIRMWARE* PARA ACIONAMENTO DO ROBÔ

O *firmware* desenvolvido para o microcontrolador *Arduino Uno* tem o objetivo de acionar o manipulador robótico. O algoritmo recebe como entrada os dados obtidos via porta *serial*. Como visto no item anterior, o algoritmo desenvolvido em *Python* será responsável pelo envio dos seis comandos possíveis identificados através da imagem da *webcam.*

A plataforma *Arduino* permite a fácil prototipagem e implementação das mais diversas aplicações. É uma placa que apresenta diversas portas de entrada e saída, tanto analógicas quanto digitais, baixo custo de aquisição e é uma plataforma de código aberto (D'AUSILIO, 2011).

Conforme a Figura 9, no fluxograma à direita, a partir da leitura do caractere de texto transmitida via porta *serial*, o programa entra em um laço para determinar o que deve acontecer com

os três motores de passo acoplados ao manipulador robótico e seus respectivos sentidos de rotação. Os comandos são transmitidos via *PWM* (*Pulse Width Modulation* – Modulação por Largura de Pulso), capaz de transformar uma saída digital em analógica (SHULTZ; VAN VUGT, 2015). São esses sinais que serão direcionados aos motores para acioná-los.

4 RESULTADOS E DISCUSSÃO

4.1. ALGORITMO PARA DETECÇÃO DE IMAGENS PARA IDENTIFICAÇÃO DOS COMANDOS DADOS AO BRAÇO

Este item apresenta em detalhe o algoritmo desenvolvido para fazer a detecção da bola verde apresentada pelo usuário à *webcam*, de modo que seja capaz de enviar comandos específicos para a manipulação do braço robótico conectado ao *Arduino*.

A função *ArgumentParser* permite analisar, através de argumentos fornecidos pelo usuário, qual será a fonte de fornecimento de vídeo para o programa. As duas opções são: acessar a transmissão da *webcam* ou trabalhar com um vídeo fornecido pelo usuário do sistema.

As linhas de código *'greenLower = (29, 86, 6)'* e *'greenUpper = (64, 255, 255)'* atribuem os limites, inferiores e superiores, respectivamente, *HSV (Hue, Saturation and Value,* em inglês*)* da cor verde.

A seguir, na linha de código *'pts = deque(maxlen = args["buffer"])'* é inicializado o *deque* e também se define o número de pontos que o traço da bola deixa na tela. *Deques*, são reconhecidos como filas de dados de duas pontas da biblioteca *collections*, e permitem que sejam criados vetores onde as leituras podem começar tanto do começo quanto do final da fila de valores (FOUNDATION, 2016b), trazendo mais agilidade ao processo. Conforme a linha de código *'ap.add_argument("-b", "—buffer", type=int, default=32, help="Maximo tamanho de buffer")'* o máximo valor do segmento será de 32 pontos (*pixels*).

Após isso, se dá continuidade ao processo de escolha da referência de vídeo. Se não houver um caminho para o arquivo de vídeo definido, a *webcam* disponível é acessada. A seguir, é provocado um pequeno atraso no sistema para que o vídeo da *webcam* ou o arquivo de vídeo possam inicializar, através da função *time.sleep*.

É dado início então ao laço *while* que determina quando o programa se encerra. Uma das condições é caso a tecla '*q*' for acionada pelo usuário no teclado, ou caso não encontre nenhum arquivo ou caminho de vídeo e acabam-se os quadros. Inicia-se a leitura de transmissão de vídeo através da função *vs.read* e se atribui a variável de referência do quadro (em inglês, *frame*).

Dessa forma, também é definida a forma de leitura do quadro, de forma que seja criada uma *tupla* (uma lista ou vetor imutável do *Python*, de dois elementos). O primeiro deles é uma variável

booleana (variáveis que retornam apenas dois tipos de valor: verdadeiro ou falso) que indica se o quadro foi lido ou não. O segundo elemento consiste do próprio quadro da tela. Na linha seguinte é decidido o caminho a seguir, caso seja o arquivo de vídeo ou a transmissão via *webcam*.

Caso a leitura do vídeo estiver sendo feita e não estiver sendo bem-sucedida então é o sinal de que o vídeo acabou e se deve quebrar o ciclo de leitura e fechar o programa.

Na linha de código *'frame = imutils.resize(frame, width = 600)'*, o primeiro argumento *frame* seleciona o quadro a ser trabalhado e o segundo determina a largura do quadro *width*, sendo definida a largura máxima de 600 *pixels*, de forma a se ter uma imagem menos "pesada" e com menos *pixels* para processar, o que traz mais rapidez e agilidade para o sistema, de forma a aumentar o número de quadros por segundo processados pelo programa.

Nas linhas seguintes é mostrado o pré-processamento do quadro para a detecção da cor. Na linha '`blurred = cv2.GaussianBlur(frame, (11, 11), 0)`' se "borra" a imagem do quadro (diminuindo a nitidez) através da função *cv2.GaussianBlur* para diminuir o ruído de cores "estouradas" (causados principalmente por focos de luz presentes no ambiente). A seguir, é trocado o espaço de cor em que a imagem se encontra, do espaço *RGB* (o espaço mais comumente utilizado, definido pelos valores da cor vermelho, verde e azul) para o *HSV* anteriormente descrito, através da função *cv2.cvtColor*: '`hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)`'.

Na linha *'mask = cv2.inRange(hsv, greenLower, greenUpper)'*, mostra-se a criação da máscara *(mask*, do inglês*)* da imagem original. Nessa máscara está contida apenas a imagem binária do quadro. Os pontos em preto do quadro são onde o programa não identificou o verde e os pontos em branco são onde o objeto verde se encontra. A função que permite tal processo chama-se *cv2.inRange*, e dentro de seus argumentos se tem o limite inferior e o superior da cor a ser identificada. Logo após, são aplicadas duas iterações de erosão na imagem com as funções *cv2.erode* e *cv2.dilate* para que seu contorno se torne mais nítido, e também são utilizadas duas iterações da dilatação, para que sejam removidos ruídos e falsos positivos da imagem gerada. Isso é visto nas linhas '`mask = cv2.erode(mask, None, iterations=2)`' e '`mask = cv2.dilate(mask, None, iterations=2)`'.

Com o auxílio da função *cv2.findCountors*, é alterada a variável *mask* mais uma vez, mas desta vez retirando seu diâmetro externo máximo, e dentre seus argumentos é utilizada uma função de aproximação simples da biblioteca. Com a utilização da biblioteca *imutils*, é identificado o contorno da bola verde e armazenado na variável *cnts*. Também é criada uma variável nula para alocar o centroide (centro geométrico) da bola com os valores *x* e *y*.

É criada então uma estrutura lógica condicional *if*, que se certifica de que pelo menos um valor de centro (ou centroide) foi coletado. Após isto, na linha de código *'c = max(cnts, key=cv2.contourArea)'* é mostrado o que é feito na sequência: procura-se pelo maior valor de

contorno da bolha de elementos positivos vista na máscara. É computado então o valor mínimo do círculo que se encaixa na bolha e, na linha seguinte, o centroide em *x* e *y* da bola, realizado na linha *'center = (int(M["m10] / M["m00"]), int(M["m01"] / M["m00"]))'*.

Logo após, é feita uma checagem rápida do valor do raio para se certificar que é coerente com o valor encontrado pela função, verificando se ele é maior que 10 *pixels*. Na sequência, duas bolas são desenhadas na tela, uma para o contorno externo da bola e outra, menor, em seu centroide para sinalização. Feito isso, é adicionada a localização do centro da bola para a lista de pontos *pts*.

Um laço *for* é feito para realizar a leitura dos pontos rastreados. Caso nem o atual nem o anterior dos pontos rastreados sejam nulos (o que indica que a bola não foi identificada devidamente no quadro), continua-se no laço para computar a largura do rastro deixado pela bola na tela com a variável *thickness* (em português, espessura). A partir disso, desenha-se o rastro.

Com o auxílio da função *cv2.line*, são desenhadas as linhas, horizontais e verticais, que determinam os quadrantes de atuação do usuário com os movimentos do braço robótico, sendo a resolução do quadro obtido de 600x460 *pixels*. É desenhada então uma linha vertical a cada 200 *pixels* e uma linha horizontal a cada 153 *pixels*. Também são escritos os comandos dos motores nos lados esquerdo e direito da tela, um para cada motor e sua respectiva direção (tanto positiva como negativa). Logo após, é feito o "selo" da comunicação da *webcam* com o *Arduino* e seus motores de passo acoplados. Com o uso da função *ser.write*, são escritas variáveis de sinal para cada situação de movimento possível, e enviadas ao *Arduino*, como pode ser observado na estrutura *if* presente na linha *'if 0 <= xx <= 200 and 154 <= yy <=307:'*. Junto a isso, para informar o usuário do sistema, é escrito no canto superior esquerdo o comando atual do manipulador robótico com o auxílio da função *cv2.putText*.

Na linha *'if key == ord("q"):'* é mostrado o quadro para o usuário e adicionada a interrupção instantânea (*break*, do inglês) do algoritmo caso a tecla '*q*' for acionada.

Fora do laço *while* principal, é questionado pelo algoritmo se está sendo utilizado o arquivo de vídeo, e caso sim, a reprodução é cessada. Caso a fonte seja uma transmissão de vídeo proveniente da *webcam*, esta também é cessada.

4.2 FIRMWARE PARA ACIONAMENTO DO ROBÔ

Este item se propõe a apresentar o algoritmo de leitura e transmissão de comandos desenvolvido para a plataforma *Arduino*. O algoritmo tem como principal função ler os caracteres enviados pelo algoritmo escrito em *Python* para a detecção do objeto e tomar ações para seus determinados motores de passo. No começo do código, são importadas as bibliotecas necessárias. Nas linhas de código *'AccelStepper Xaxis(1, 2, 5);'*, *'AccelStepper Yaxis(1,*

3, 6);' e *'AccelStepper Zaxis(1, 4, 7);'*, são declaradas as pinagens dos três motores de passo presentes no manipulador robótico, sendo um pino para a alimentação do motor no primeiro argumento da função, o segundo para o passo desejado e o último para direção selecionada.

AccelStepper é uma biblioteca com foco no desempenho da aceleração do motor de passo. Ela trabalha com dois valores: a velocidade máxima que o motor deve alcançar durante o trajeto, e sua devida aceleração (MCCAULEY, 2010a). Isto pode ser percebido pelas linhas de código *'int Vel = 50;'* e *'int Acel = 100'*, com as variáveis *Vel* para velocidade e *Acel* para aceleração.

Em seguida, é declarada a variável *data* para armazenar onde a bola esteve anteriormente, auxiliando no processo criado junto com o caractere *G*, proveniente da porta *serial*. Este processo delimita o quadrante central para ser identificado. Cria-se também a variável de caractere (*char*) justamente para o armazenamento do comando recebido via *Python*.

Logo após, inicializa-se a porta *serial* e a função de preparação (*setup*, no inglês) do algoritmo geral atribuindo acelerações e velocidades máximas de cada motor, através das funções *setMaxSpeed* e *setAcceleration* conforme, por exemplo, as linhas de código *'Xaxis.setMaxSpeed(Vel);'* e *'Xaxis.setAcceleration(Acel);'* para o motor do eixo X. Procedimento similar é feito para cada um dos motores de eixo presentes no sistema.

No laço principal do programa, *void loop*, é checado se a porta *serial* se encontra disponível, e após isso se faz a leitura da variável de texto contida, e então é atribuída à variável *userInput*.

Na estrutura lógica *if* presente na linha *'if (userInput == 'A' && data != 1)'* e subsequentes, estão contidas as linhas de código para a execução do movimento, com o auxílio da função *move*, a qual determina os passos a serem dados pelo motor. Dentro da função, através de uma estrutura lógica, é decidido se o motor deve continuar rotacionando ou não, com as funções *distanceToGo* (que se certifica de que há distância a ser percorrida), e *runToPosition* (que mantém a rotação do motor). Também se altera o valor da variável *data*, que se certifica que, se o comando foi executado previamente e não deve ser executado novamente a menos que o usuário do sistema mude de comando ou passe pelo quadrante central.

4.3 PROVA DE CONCEITO DO PROTÓTIPO DE DESENVOLVIDO

Para a realização dos testes de conceito do sistema de manipulação do braço robótico, foram estipulados movimentos dos eixos nele presentes, de forma a verificar a exatidão dos movimentos comandados a partir do algoritmo de visão computacional. Foram definidos os deltas dos ângulos para os movimentos: 5, 10, 20 e 40 graus, em cada um dos eixos X, Y e Z. Estes ângulos refletem a necessidade de movimentos comandados mais precisos, também com maior passo e consequentemente mais velozes.

As Figura 11 demonstra um par de testes realizados, com ângulo de 5 graus nas fotos à esquerda (de 85 a 90 graus no eixo Y) e com ângulo de 40 graus nas fotos à direita (130 a 90 graus no eixo X). A posição inicial da referência de ângulo é mostrada à esquerda em cada par e em seguida, mostra-se à direita, em cada par, a referência após o movimento realizado pelo manipulador robótico

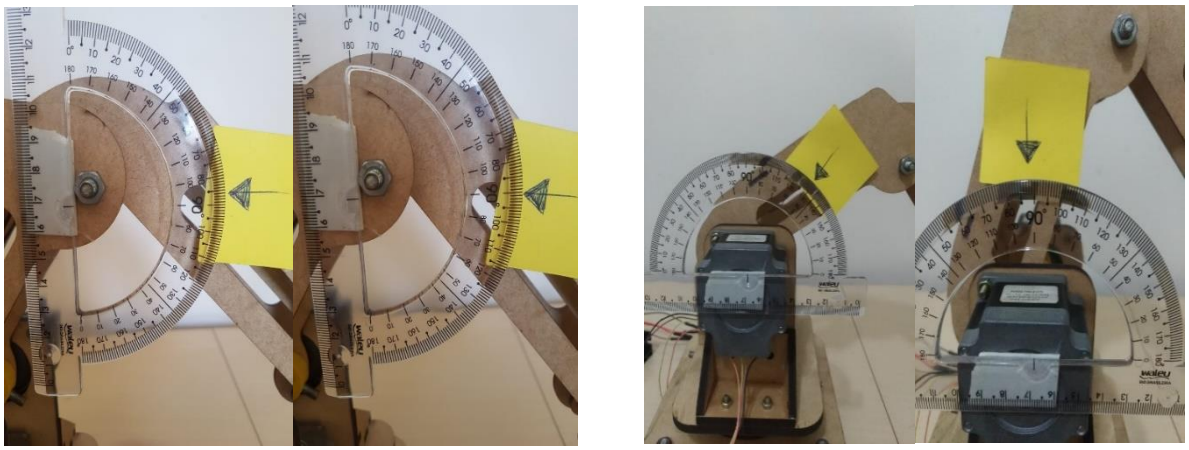

Figura 11 – Demonstração do movimento realizado pelo motor de passo do eixo Y, onde percebe-se o movimento do motor na posição de 85º para 90º (à esquerda) e movimento de 130º para 90º no eixo X (à direita)
Fonte: Elaborado pelos autores

Os testes realizados elucidam os pontos positivos, mas também os aspectos que permitem melhorias nas ações do projeto, apesar da precisão obtida. Folgas e pontos estruturais são decisivos na execução com êxito das variações de ângulos desejadas.

Sendo os comandos transmitidos via algoritmo de detecção, verificou-se com sucesso a replicação das instruções no manipulador robótico.

Para verificar o funcionamento do protótipo em relação a luminosidade ambiente, fator que afeta o algoritmo para detecção de comandos, foram realizados os testes aferindo a luminosidade ambiente utilizando um sensor de luminosidade fotossensitivo P7 da fabricante *GBK Robotics*. Esse modulo faz uso do sensor fotocélula *LDR* (“*Light Dependent Resistor*”), que é um resistor cuja resistência varia de acordo com a luminosidade e luz incidente .

Foram realizados testes variando a luminosidade ambiente, medindo a mesma através do sistema descrito e verificando a eficácia na detecção dos movimentos do objeto responsável por acionar o braço robótico.

Obteve-se então as faixas de operação ideais de iluminância para projeto entre 450 e 550 lux, faixa que é observada em instalações industriais convencionais.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi desenvolvido um sistema de controle inteligente para um manipulador robótico através da utilização de visão computacional. Foi possível desenvolver o *firmware* do microcontrolador para acionar o braço robótico, o algoritmo de visão computacional e construir o próprio manipulador robótico.

Inicialmente, a proposta era utilizar o *Kinect* para reconhecimento de gestos. Devido a dificuldades com suporte devido a descontinuidade do *Kinect* e as suas bibliotecas, optou-se pelo uso de uma *webcam* aliado a biblioteca *OpenCV*. Assim, foi possível a detecção de objeto, a extração de contornos e o rastreamento e posicionamento real. Dessa forma, tornou-se possível a aplicação do painel de controle, a interface com usuário do sistema proposto, que recebe os comandos gestuais e verifica com sucesso a replicação destes no manipulador robótico construído.

O trabalho permitiu definir uma metodologia e um conjunto de tecnologias e ferramentas de desenvolvimento que poderão ser aplicadas ao desenvolvimento de outras aplicações robóticas com comandos gestuais.

No contexto industrial, onde a capacitação de programadores e operadores é um desafio, a proposta do trabalho poderá permitir a programação ou envio de comandos a um mecanismo robótico de maneira amigável.

No futuro, o trabalho poderá evoluir para o contexto de um robô colaborativo onde pessoas, operadores e outros robôs poderão colaborar para execução de uma tarefa industrial. Para tal desenvolvimento será necessário um sistema com sensores e atuadores para prover a segurança ao operador e pessoas próximas aos robôs.

REFERÊNCIAS

ARBOR, Ann. **North American Machine Vision Market Grows 19% in First Quarter of 2018, Sets New Records.** Association of Advancing Automation, 2018. Disponível em: <https://www.a3automate.org/north-american-machine-vision-market-grows-q1-2018/>. Acesso em: 17 nov. 2018.

BENESOVÁ, Andrea; TUPA, Jirí. Requirements for Education and Qualification of People in Industry 4.0. **Procedia Manufacturing**, [s.l.], v. 11, p.2195-2202, 2017. Elsevier BV. http://dx.doi.org/10.1016/j.promfg.2017.07.366. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2351978917305747>. Acesso em: 16 nov. 2018.

BLABST, Michael. **Work Begins on BMW Plant Leipzig Extension.** BMW Group, 2018. Disponível em: <https://www.bmwgroup-plants.com/leipzig/en/news/Plant_extension.html#sectioncontainer_parsys_layoutcontainer_layoutcontainercontent_nachricht>. Acesso em: 17 nov. 2018.

D'AUSILIO, Alessandro. **Arduino: A low-cost multipurpose lab equipment**. Behavior Research Methods, [s.l.], v. 44, n. 2, p.305-313, 25 out. 2011. Springer Nature. http://dx.doi.org/10.3758/s13428-011-0163-z.

DRISCOL, Ed. **The Keys To Chromakey: How To Use A Green Screen**. Disponível em: <https://www.videomaker.com/article/f5/13055-the-keys-to-chromakey-how-to-use-a-green-screen>. Acesso em: 10 mai. 2019.

FOUNDATION, Python Software. **Python Language Reference, version 3.7**. Disponível em: <https://docs.python.org/3/library/index.html>.
Acesso em: 16 abr. 2019.

FOUNDATION, Python Software. **Python Language Reference, version 3.7**. Disponível em: <https://docs.python.org/3/library/collections.html#collections.deque>.
Acesso em: 16 abr. 2019.

GROOVER, Mikell P. et al. **Robótica: Tecnologia e Programação.** São Paulo: Mcgraw-hill, 1988.

GROOVER, Mikell P. **Automação Industrial e Sistemas de Manufatura.** 3. ed. São Paulo: Pearson Prentice Hall, 2011.

HEER, Carsten. **Robots Double Worldwide by 2020:** 3 Million Industrial Robots Use by 2020. IFR.ORG, 2018. Disponível em: <https://ifr.org/ifr-press-releases/news/robots-double-worldwide-by-2020>. Acesso em: 17 nov. 2018.

MARENGONI, Maurício; STRINGHINI, Denise. Tutorial: Introdução à Visão Computacional usando OpenCV. **Rita**: Revista de Informática Teórica e Aplicada, Porto Alegre, v. 16, n. 1, p.125-160, jul. 2009. Disponível em: <https://www.seer.ufrgs.br/rita/article/view/rita_v16_n1_p125/7289>. Acesso em: 27 abr. 2019.

MCCAULEY, Mike. **AccelStepper library for Arduino**. Disponível em: <https://www.airspayce.com/mikem/arduino/AccelStepper/>. Acesso em: 15 abr. 2019.

OWEN-HILL, Alex. **What is the Best Programming Language for Robotics?** Robotiq, 2016. Disponível em <https://blog.robotiq.com/what-is-the-best-programming-language-for-robotics>. Acesso em: 21 nov. 2018.

PULLI, Kari et al. Real-time computer vision with OpenCV. **Communications Of The Acm**, [s.l.], v. 55, n. 6, p.61-69, 1 jun. 2012. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/2184319.2184337. Disponível em: <http://lvelho.impa.br/ip08/reading/rt-ocv.pdf>. Acesso em: 26 abr. 2019.

ROBOTIC INDUSTRIES ASSOCIATION. **Dangerous Robot Jobs.** Robotics Online Blog, 2016. Disponível em: <https://www.robotics.org/blog-article.cfm/dangerous-robot-jobs/15>. Acesso em: 21 nov. 2018.

SCHULTZ, Benjamin G.; VAN VUGT, Floris T.. **Tap Arduino: An Arduino microcontroller for low-latency auditory feedback in sensorimotor synchronization experiments**. Behavior Research Methods, [s.l.], v. 48, n. 4, p.1591-1607, 5 nov. 2015. Springer Nature. http://dx.doi.org/10.3758/s13428-015-0671-3.

ZEZULKA, F. et al. Industry 4.0 – An Introduction in the Phenomenon. **Ifac-Papersonline,** [s.l.], v. 49, n. 25, p.8-12, 2016. Elsevier BV. http://dx.doi.org/10.1016/j.ifacol.2016.12.002. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405896316326386>. Acesso em: 16 nov. 2018.