Development of Robotic Arm Control System Using Computational Vision

G. Oliveira, G. Oliveira, J. Luna, C. Egoavil, *Member, IEEE*, and C. Carvalho Jr., *Member, IEEE*

***Abstract*—Industrial robots are present in most of today's industries. Their benefits to industrial production make them the main research's target and constant development. However, are limited to sequential and repetitive movements without any decision making. Therefore, one of the current researches and development strands is the robot's integration with computer vision. Which seeks to enable them to see as humans, and thus, make them even more useful, efficient and independent. In this context, a PID control system was developed for a robotic arm. It is integrated to a computer vision system using Kinect®, in order to capture objects - within 6 possible positions - according to their color. For that, an Arduino MEGA 2560 was used for control system, and Software was developed in Processing programming language for integration, systems operation, and 6-position definition. The SimpleOpenNI and OpenCV for Processing libraries were used to work with Kinect®, and computer vision, respectively. Finally, the results were satisfactory, and the lowest percentage of catches made successfully among all positions was 83.33%.**

***Index Terms*—Computer Vision, Control, *Kinect*®, *Arduino*, PID.**

I. Introdução

EM 2015 foram investidos 2,5 bilhões de dólares na indústria de robótica, e até 2030 as linhas de produção em geral deverão produzir 55% a mais do que foi produzido em 2015 [1]. Além disso, em uma sociedade com um mercado de trabalho que se torna mais competitivo a cada dia e que valoriza cada vez mais a ciência, viu-se a necessidade de se passar menos tempo executando tarefas básicas, porém essenciais, e passar mais tempo com estudos e tarefas específicas que se tornaram mais valorizadas. Dessa forma, a sociedade passou a entender que as máquinas não seriam eficientes e úteis apenas nas indústrias, mas em diversas outras áreas onde poderiam tanto executar determinadas tarefas de forma mais eficiente, quanto trazer mais conforto aos humanos, executando, portanto, diversos serviços não especializados, repetitivos ou de força bruta, como robôs separadores de estoque de pequenos e grandes produtos, limpadores de casas, atendentes de lojas, etc. [1]. Ademais, a eficiência da robótica começou a ser utilizada também em serviços especializados com a operação de um profissional, pois diversos tipos de robôs estão sendo usados onde são necessárias precisões milimétricas, como em cirurgias difíceis, ou para manusear substâncias nocivas aos seres humanos, como os elementos radioativos [2].

G. S. Oliveira, Universidade Federal de Rondônia, Porto Velho, Rondônia, Brasil, glaufesantos@gmail.com.

G. B. Oliveira, Universidade Federal de Rondônia, Porto Velho, Rondônia, Brasil, gladistone.batista@unir.brfortes.

J. D. F. O. Luna, Instituto Federal de Rondônia, Porto Velho, Rondônia, Brasil, jdiogoforte@hotmail.com.

C. J. Egoavil, Universidade Federal de Rondônia, Porto Velho, Rondônia, Brasil, ciro.egoavil@unir.br.

C. A. T. Carvalho Jr., Universidade Federal de Rondônia, Porto Velho, Rondônia, Brasil, tenorio@unir.br.

É notório que a visão é um dos principais – se não o principal – sentidos usados pelos seres humanos, o qual permite identificar objetos, pessoas e animais, bem como provê noção de profundidade e localização. Sendo assim, devido a importância e versatilidade da visão, integrar o reconhecimento preciso de objetos a robôs automatizados traz uma maior eficiência e independência para estes robôs na indústria; como consequência, o processo industrial como um todo se torna mais lucrativo e competitivo.

Com o intuito, então, de tornar os robôs capazes de reproduzir as ações humanas de formas mais eficientes, chegou-se a uma das maiores dificuldades da robótica: como reproduzir a visão humana em robôs?

Assim, como principal objetivo deste trabalho foi simular um ambiente industrial utilizando um braço robótico didático, *Softwares* gratuitos e sistemas embarcados, integrado a um sistema de visão computacional. O sistema de visão computacional deverá ser capaz de identificar a posição de objetos específicos dentro de seis posições pré-definidas e, então, orientar o braço robótico a pegar o objeto e colocá-lo em outra posição pré-definida – as posições e os objetos são definidos através de um software. Um diferencial contido no presente trabalho na integração de um robô e um sistema de visão computacional é a utilização de *Softwares* e sistemas embarcados gratuitos e de código aberto, ao contrário, por exemplo, de *Softwares* como *Matlab*, que são amplamente utilizados por estudantes e profissionais no âmbito da visão computacional e robótica [3]–[7].

Considerando os aspectos mencionados anteriormente, foi desenvolvido um *Software* em linguagem de programação *Processing* utilizando seu ambiente de desenvolvimento integrado e as bibliotecas *SimpleOpenNI* [8] e *OpenCV* [9], e o sistema de controle do braço robótico OWI-535 (OWI Inc.) utilizado foi implementado a partir de uma placa *Arduino* MEGA 2560 [10]. Não obstante, outro diferencial apresentado neste trabalho é a utilização do *Kinect* (controle de videogame baseado em visão computacional comercializado pela empresa *Microsoft*), que tem como principal vantagem seu sensor de profundidade que possibilita o reconhecimento da posição de objetos em três dimensões, sem a necessidade de câmeras adicionais, como no método da estéreo-visão, ou um conhecimento prévio da cena capturada, como no método monocular [11].

II. Materiais e Métodos

A. Braço Robótico OWI-535

O Braço Robótico OWI-535, Figura 1, utilizado neste trabalho é um manipulador robótico criado pela empresa OWI Inc. do tipo articulado com quatro graus de liberdade que usa motores CC como atuadores, sendo que o sentido da movimentação dos elos é regido pela polarização dos motores. Ele é composto por uma base com uma rotação horizontal de 270 graus, ombro com rotação vertical de 180 graus, cotovelo com rotação de 300 graus, pulso com rotação de 120 graus e um efetuador tipo garra. Sua carga máxima suportada é de 100 gramas. O Braço Robótico OWI-535 não é dotado de sensores para realimentação de posição dos elos ou efetuador.

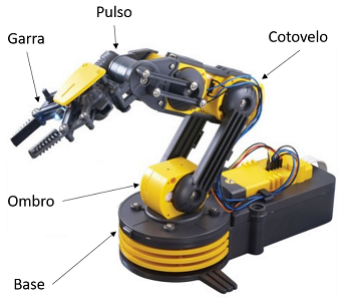

Fig. 1. Braço robótico OWI-535.

B. Visão Computacional

Em 2010, a *Microsoft* trouxe um novo tipo de acessório para os seus jogos: o *Kinect* para *Xbox 360*. Com o sucesso do *Nintendo WII* e a necessidade de prolongar a vida útil do seu console, a *Microsoft* levou ao mercado um produto que tornava o jogador capaz de jogar usando apenas o seu próprio corpo, com gestos físicos e comandos de voz [12].

O Kinetic, Figura 2, é composto por uma câmera RGB, um sensor de profundidade contendo um projetor de luz infravermelha, uma câmera infravermelha, um conjunto de microfones, um motor que altera a angulação da câmera e do sensor (-27^o a 27^o) e um acelerômetro que indica a inclinação do sensor.

Com essa arquitetura, o *Kinect* devolve os seguintes dados ao sistema ao qual está conectado [13]:

a) *Image Stream* (imagem): a câmera RGB captura imagens em uma resolução de 1280×960 a 15 *frames* por segundo (fps), ou a 30 *frames* por segundo em uma resolução de 640×480.
b) *Depth Stream* (informação de profundidade): o projeto infravermelho emite pulsos de luz infravermelha, e a câmera infravermelha os lê quando refletidos de volta. Assim, através da leitura desses pulsos, o equipamento consegue fazer leitura de profundidade e medir a distância entre os objetos à frente do aparelho, e o sensor a partir de 1,2 até 3,5 metros. Além da imagem em RGB, é possível, ainda, obter as imagens da câmera infravermelha com 640x480 e 30 fps, sendo que cada *pixel* equivale à profundidade dos objetos na imagem.
c) *Audio Stream* (áudio): com um conjunto de 4 microfones e anulação de ruído e eco, o *Kinect* pode ser utilizado para gravação de áudio e reconhecimento de fala, como é utilizado pela *Microsoft* em seus consoles.

Fig. 2. Arquitetura básica do *Kinect* [13].

Além disso, desde o seu lançamento, cientistas e pesquisadores viram potencial em sua tecnologia e, então, o *Kinect* vem sendo usado em diversas áreas do conhecimento, dentre elas: robótica, arqueologia e medicina [14], [15]. Com isso, a *Microsoft* lançou oficialmente o *SDK* (Kit de Desenvolvimento de *Software*) para *Kinect*, hoje em sua versão 1.8 para a versão de *Xbox 360* do *Kinect*, com vistas a integrar o mesmo com computadores que utilizam *Windows* 7, 8, 8.1 ou 10, e facilitar o desenvolvimento e a pesquisa com o aparelho.

C. Sistema de Controle

Um sistema de controle em malha fechada atua comparando o valor da variável de processo com um valor desejado. A diferença entre estes valores é usada para computar, por meio de uma lei de controle, um sinal de correção que é aplicado ao processo. A lei de controle mais tradicional é a PID, onde o sinal de correção é obtido ponderando o erro atual, o erro acumulado e a variação do erro [16]–[21].

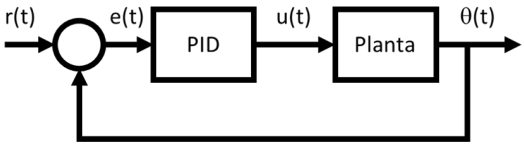

Fig. 3. Descrição simplificada do sistema de controle.

D. Processing

É uma linguagem de programação e ambiente de desenvolvimento integrado flexível para desenvolvimento gráfico de *Softwares*. É totalmente grátis e *Open Source* (código aberto), tendo sido criado por Ben Fry e Casey Reas, inicialmente, apenas para facilitar o desenvolvimento gráfico de *Softwares* e ensinar fundamentos de programação dentro de contextos visuais; porém, com o passar do tempo, evoluiu também para uma ferramenta de desenvolvimento para profissionais [22].

Devido a seu status de *Open Source*, *Processing* tem uma comunidade de desenvolvimento e várias bibliotecas que facilitam o trabalho em áreas como visão computacional, visualização de dados, composição musical, entre outras [22].

Seu grande diferencial é a facilidade de desenvolvimento gráfico. Desta forma, trabalhar com imagens ou *Interfaces* gráficas para *Softwares* se torna muito simples e intuitivo, assim como a interação do *Software* com os periféricos do computador, como mouse e teclado. Diante disso, prototipação de *Softwares* e visualização de dados são duas das mais importantes áreas para desenvolvedores que utilizam *Processing*, uma vez que companhias grandes, como Google e Intel, têm usado *Processing* para desenvolvimento inicial de novas *Interfaces* e serviços.

E. OpenNI

OpenNI é um *SDK Open Source* usado para o desenvolvimento de bibliotecas e aplicativos *Middleware* (*Software* que se encontra entre o sistema operacional e os aplicativos nele executados, funcionando como uma camada oculta de tradução de detecção 3D) [8]. O *site* da *OpenNI* oferece uma comunidade ativa de desenvolvedores, ferramentas e suporte. Dentre os projetos elaborados na *OpenNI* está o *SimpleOpenNI*, uma biblioteca desenvolvida para *Processing* por Max Rheiner que permite a integração com o *Kinect* e acesso às suas funcionalidades principais, como sua câmera e seu sensor de profundidade [8].

F. OPENCV

OpenCV (*Open Source Computer Vision Library*) é uma biblioteca *Open Source* de visão computacional e *Machine Learning* (aprendizado de máquina). É escrita em C e C++, e tem desenvolvimento ativo em várias linguagens, como *Python*, *Ruby*, *Matlab* e *Processing* [9]. A biblioteca tem mais de 2.500 algoritmos otimizados, dentre os quais alguns exemplos de aplicabilidade são: detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmera, rastrear objetos em movimento, extrair modelos 3D de objetos, encontrar imagens semelhantes de um banco de dados, acompanhar movimento dos olhos, etc. [9].

G. Controle do Braço Robótico OWI-535

Para que o braço robótico fosse capaz de capturar os objetos com precisão, foi necessário um sistema de controle da posição angular de seus elos, quando escolheu-se implementar um controle em malha fechada com ação PID. Porém, devido as limitações do braço robótico, fez-se necessário realizar adaptações para obtenção das realimentações para o sistema de controle.

A primeira adaptação foi a alocação de potenciômetros de $100k\Omega$ posicionados para que seus valores variassem conforme a movimentação dos elos. Assim, foram colocados três potenciômetros: um no ombro do braço, outro no cotovelo, e o último no pulso, como pode ser visto na Figura 4.

Com isso, a leitura da posição angular dos elos é feita em 10 *bits*, e varia entre valores de 0 a 1023. Ainda, implementou-se a média para a leitura de cada potenciômetro de modo a atenuar o ruído de medição. O sinal, então, é sobre amostrado e são tomadas 30 leituras das quais se obtém uma média.

A segunda adaptação foi incluída com a intenção de obter a posição angular da base do braço. Como não foi possível a instalação de um potenciômetro, ou outro tipo de sensor que fizesse a leitura da posição conforme sua variação, decidiu-se, pois, limitar o braço a se deslocar horizontalmente por quatro posições definidas, dentre elas três posições dentro do campo de visão do *Kinect* (onde é possível o resgate dos objetos) e uma fora do campo de visão (para deixar os objetos capturados).

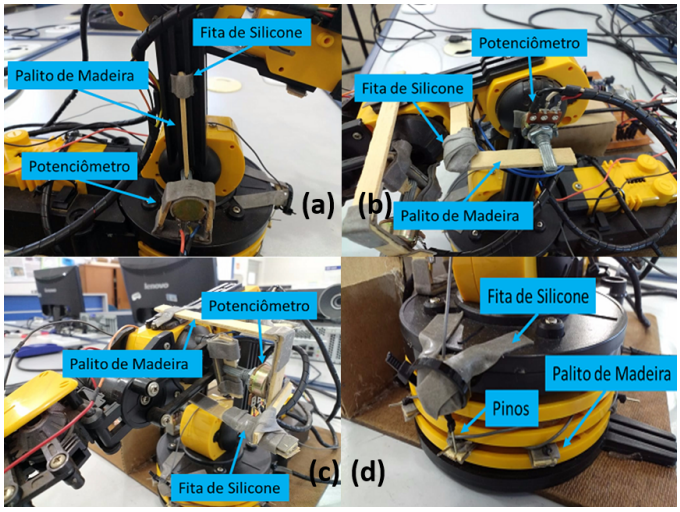

Fig. 4. (a) Potenciômetro instalado no ombro do braço robótico. (b) Potenciômetro instalado no cotovelo do braço robótico. (c) Potenciômetro instalado no pulso do braço robótico. (d) Pinos instalados para controle da posição angular horizontal do braço.

Para isso, instalaram-se cinco pinos na base (Figura 4(d)): um na parte frontal da base, um do lado direito, dois espaçados igualmente à direita e à esquerda do pino frontal, e o último na parte frontal do braço, deslocando-se junto com ele. Este último é conectado aos 5 Volts da placa microcontroladora, enquanto os outros quatro estão conectados a quatro portas digitais. Quando o braço se desloca horizontalmente, o pino preso à sua frente entra em contato com os pinos da base, e a porta digital conectada ao pino da base, que está em contato com o pino do braço, recebe os 5 Volts, enquanto os outros três permanecem desligados. Dessa forma, é possível saber em qual das quatro posições o braço está, e controlar esse deslocamento.

A terceira e última adaptação foi a instalação de um sensor de pressão piezoelétrico no efetuador tipo garra do braço, Figura 5, o qual detecta alterações de pressão quando a garra é fechada ou captura um objeto. Assim, o microcontrolador recebe essa informação e, então, ela é utilizada para saber quando a garra capturou algo e controlar a força aplicada ao objeto, ou, se simplesmente a garra se fechou.

H. Identificação do Sistema

Sabe-se, a partir da modelagem fenomenológica, que a posição angular de um motor DC tem uma dinâmica composta por um integrador e um par de polos reais, sendo que, em geral, o polo rápido pode ser desconsiderado visto que

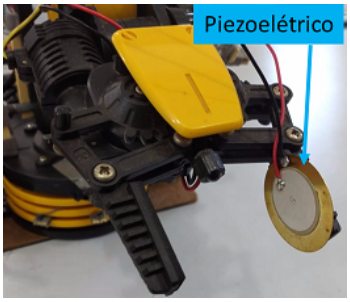

Fig. 5. Piezoelétrico instalado na garra do braço robótico.

a dinâmica mecânica do motor é muito mais lenta que a dinâmica elétrica [16], podendo ser descrito por:

$$\frac{Y(s)}{U(s)} = \frac{K}{s(s\tau + 1)} \tag{1}$$

O processo para identificar um sistema com um polo real e um integrador através de sua resposta ao impulso é análogo a identificar um sistema de primeira ordem por sua resposta ao degrau. Portanto, para levantar os modelos foi utilizado o método de Hägglund modificado para o caso integrador e, deste modo, pode-se encontrar o valor de τ sabendo que o mesmo equivale ao tempo que o sistema leva para alcançar 63,2% do valor de regime permanente após a entrada em impulso, enquanto K é o ganho calculado através da razão entre a variação na saída e a magnitude do impulso. Sendo assim, para realizar a identificação do modelo para cada junta, o respectivo motor foi ligado com 5 Volts durante 200 milissegundos para garantir a unitariedade do impulso, e a variação nos potenciômetros foi lida. Logo, obtiveram-se os modelos: Ombro:

$$\frac{Y(s)}{U(s)} = \frac{10}{0,188s^2 + s} \tag{2}$$

Cotovelo:

$$\frac{Y(s)}{U(s)} = \frac{11}{0,157s^2 + s} \tag{3}$$

Pulso:

$$\frac{Y(s)}{U(s)} = \frac{10}{0,332s^2 + s} \tag{4}$$

I. Implementação dos Controladores PID

Em posse das funções de transferência de cada motor, utilizando *Softwares* de projeto assistido por computador e de forma heurística, encontrou-se os controladores PID para cada motor, sendo eles:
Ombro:

$$u(t) = 2,4e(t) + 0,36\int_0^t e(\tau)d\tau + 0,09\frac{de(t)}{dt} + 401 \tag{5}$$

Cotovelo:

$$u(t) = 4,4e(t) + 0,36\int_0^t e(\tau)d\tau + 0,06\frac{de(t)}{dt} + 401 \tag{6}$$

Pulso:

$$u(t) = 4,5e(t) + 0,38\int_0^t e(\tau)d\tau + 0,13\frac{de(t)}{dt} + 401 \tag{7}$$

Com isso, partiu-se para a implementação destes controladores no braço robótico através da placa microcontroladora *Arduino* MEGA 2560. Começando pela parte física, em se tratando de motores de corrente contínua como atuadores, empregaram-se pontes H para se inverter a rotação dos motores quando necessário e, assim, controlar a posição angular dos elos, como também para funcionarem como uma chave eletrônica, permitindo o controle dos motores de acordo com o sinal de saída do *Arduino*, mas alimentando-os através de uma fonte externa de 5 Volts e 6 Amperes, tendo em vista que o microcontrolador não tem capacidade para alimentar todos os motores por si só.

Dessa forma, escolheu-se o módulo de ponte H L298N que trabalha com até dois motores e com saídas PWM. A Figura 6 demonstra o esquemático elétrico geral do trabalho após concluídas as instalações.

Na Figura 7 é apresentado um teste de resposta ao degrau para as juntas do ombro, do cotovelo e do pulso operando em malha fechada; as posições angulares são apresentadas nas curvas em vermelho sólido, enquanto os $setpoints$ são apresentados em azul tracejado, ambos em valores normalizados. As respostas do ombro e do pulso apresentaram tempo de acomodação da ordem de 5s, enquanto a junto do cotovelo teve tempo de acomodação da ordem de 6s. As três juntas apresentaram respostas sem sobressinal.

J. Implementação do Sensor Kinect

Primeiramente, como a versão do *Kinect* utilizada é a versão para *Xbox 360*, fez-se uso de um adaptador USB para a conexão do mesmo ao computador. A fonte é conectada a uma tomada comum (127 ou 220 VAC/60 Hz), o conector USB no computador e o outro conector ligado ao *Kinect*; ainda, para a conexão do *Kinect* ao computador, foi instalado o *SDK* para *Kinect* versão 1.8, o qual contém todos os *drivers* necessários para a integração do mesmo ao *Windows* 10 [23]. Com o *Kinect* devidamente conectado, parte-se para a obtenção da imagem em RGB 640×480 e matriz de profundidade capturada por aquele. Assim, a biblioteca *SimpleOpenNI* foi instalada, e obteve-se acesso aos dados requeridos. Utilizou-se a versão 2.2.1 do *Processing* que é compatível com a *SimpleOpenNI* e a *OpenCV*.

Como último ajuste para uso do *Kinect*, teve-se a certeza de usá-lo sempre a uma distância mínima de 1,5 metros, levando em conta a distância mínima para um bom reconhecimento de profundidade.

K. Reconhecimento do Objeto

A partir da imagem RGB capturada, iniciou-se o processo de reconhecimento dos objetos. A primeira etapa foi a instalação da biblioteca *OpenCV* para *Processing*, em sua versão 0.5.4. Concluída a instalação e tendo-se escolhido a cor do objeto (vermelho ou azul), partiu-se para o processamento da imagem.

A imagem em RGB e a cor do objeto selecionado foram convertidas para o modelo HSV. Após a conversão, realizou-se uma etapa de *thresholding* multinível, onde a imagem em HSV é segmentada utilizando a função *inRange*, excluindo,

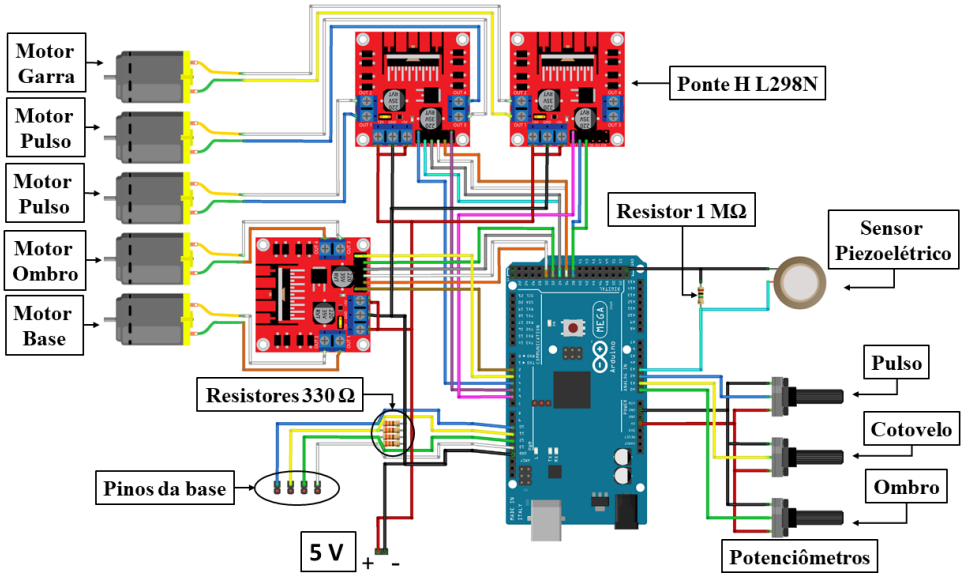

Fig. 6. Esquemático elétrico do projeto usando Arduino MEGA 2560 e Ponte H L298N.

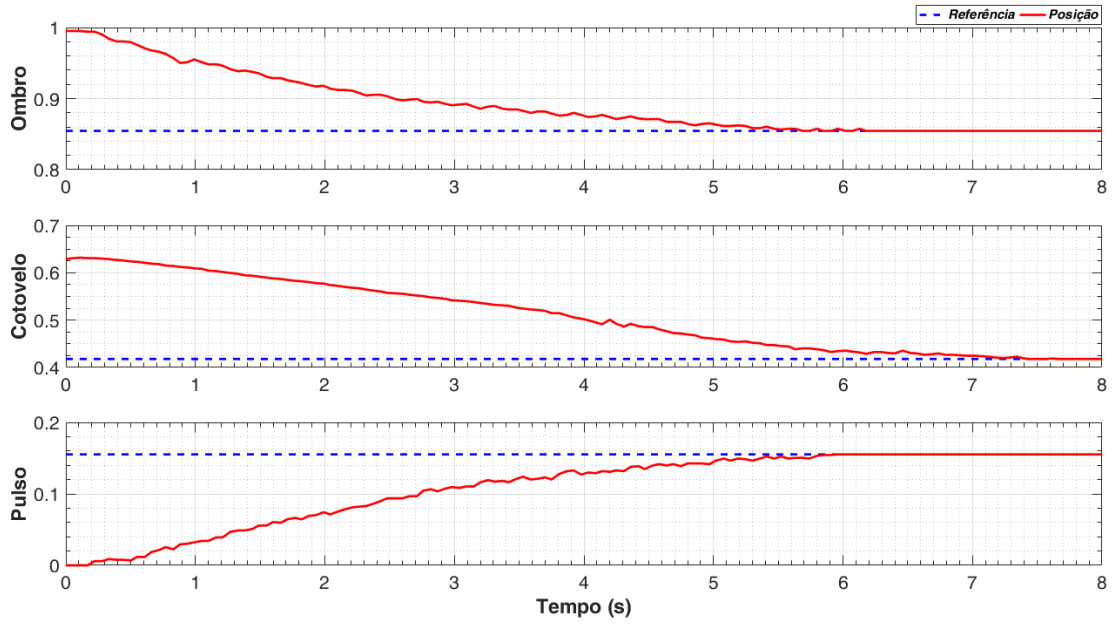

Fig. 7. Resposta normalizada ao degrau em malha fechada durante 8 segundos.

portanto, tudo que esteja fora do intervalo entre a matiz da cor selecionada -1 e a matiz da cor selecionada $+1$, independente de brilho ou saturação. Então, foi aplicado o método de suavização através da função *blur* e a segmentação reforçada através da função *threshold*. Por fim, foram aplicados filtros morfológicos de erosão e dilatação através das funções erode e dilate, respectivamente, os quais foram utilizados nos processos de abertura e fechamento, sendo aplicado, primeiramente, a abertura, e, posteriormente, um fechamento [24].

Feito isto, é encontrada a maior quantidade de *pixels* agrupados que está dentro do intervalo de *threshold*, a qual caracteriza o objeto reconhecido. Com isso, através das coordenadas do centro desse agrupamento de *pixels*, a distância entre o *Kinect* e o objeto é encontrada na matriz de profundidade, completando o reconhecimento da posição do mesmo.

L. Implementação do Software

Objetivando integrar o reconhecimento do objeto e a ação do sistema de controle, desenvolveu-se um *Software* com *interface* gráfica mostrando as imagens capturadas em tempo real pelo *Kinect* e a possibilidade de se escolher qual objeto (azul ou vermelho) o braço deverá capturar. Quando escolhidos, os objetos passam a ser rodeados por um retângulo com um círculo no centro para demonstração ao usuário. Além disso, o *Software* foi desenvolvido para que as cores utilizadas no reconhecimento dos objetos sejam selecionadas ou ajustadas através de cliques do mouse na imagem dos objetos. De forma a manter o funcionamento do reconhecimento, mesmo após

variações de luz ou ambiente, os dados das cores são salvos em um arquivo CSV (*Comma-Separated Values*) e importados sempre que o programa é aberto. Ademais, decidiu-se por definir seis posições fixas onde a peça poderia ficar. Sendo assim, foi adicionado na *interface* gráfica do *Software* uma forma de ajustar e reajustar as posições da peça sem a obrigação de mexer no código. Logo, os dados de cada posição são, também, salvos em um arquivo CSV. Adicionalmente, foi acrescentada no *Software* a possibilidade de alterar a posição dos elos da garra ao capturar os objetos, salvando estes dados na memória EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) do microcontrolador e resgatando-os sempre que forem ligados, assim, às seis posições adaptáveis dentro das limitações do braço robótico.

M. Objetos de Identificação

As peças a serem reconhecidas e capturadas nesse trabalho são mostradas na Figura 8 com suas devidas dimensões.

As dimensões e a geometria dos objetos foram escolhidas levando em consideração o tamanho do braço, bem como o grau de liberdade do pulso do braço robótico.

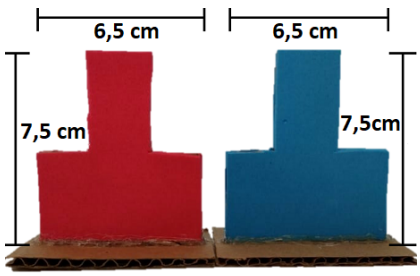

Fig. 8. Objetos de $5,32g$ à serem reconhecidos e capturados.

III. Resultados E Discussão

A. Controle do Braço Robótico

O primeiro problema do sistema de controle foram as leituras instáveis dos potenciômetros utilizados nas realimentações de posição. A Figura 9 demonstra a diferença entre 10 segundos de leitura sem o filtro e, após, com o filtro.

Nestes gráficos é possível perceber o nível de ruído de medição presente na leitura sem o filtro, e como o filtro atenuou o problema. Em termos de SNR (*Signal to Noise Ratio*), os resultados do filtro aplicado podem ser vistos na Tabela I.

TABELA I
Resultados do filtro aplicado em termos de SNR

SNR (Potênciometro)	Sem Filtro (dB)	Com Filtro (dB)
Ombro	42,74	58,31
Cotovelo	35,29	52,19
Pulso	29,69	45,25

Por fim, é fato que o filtro não resolveu completamente o problema, pois, mesmo que pequena, a variação ainda existe. Porém, por ser tão pequena, essa variação não consegue mover os elos para tentar corrigir o erro, e, dessa forma, permite que o sistema funcione bem e não apresente *jitter* na variável controlada.

Para testar a eficácia do sistema de controle PID, foi realizado um procedimento em que, por 10 vezes, foram escolhidos valores aleatórios de posições angulares para os elos – respeitando, é claro, os limites do braço –, e após a ação do sistema de controle PID tomou-se os valores finais e comparou-os com os valores desejados.

Em resumo, o erro médio entre todos os elos e testes foi de 0,52% em relação a escala de 0 a 1023, o que, na prática, não interfere no posicionamento e nem para a captura dos objetos em questão. Além disso, houve apenas dois erros expressivos: um de 1,85% e outro de 1,46% que, apesar de expressivos na posição angular dos elos, ainda assim são variações pequenas de posição.

A razão para a ação integral do controlador PID não ter tirado o erro em regime se dá pelo fato de o controlador, neste estudo, estar trabalhando em um tempo limite de 9 segundos – tempo que é suficiente para que o braço robótico cumpra seu objetivo.

Posteriormente, com os testes e a movimentação do braço, percebeu-se que, eventualmente, o pino instalado à frente do braço perde o contato com todos os pinos da base. Por essa razão, o sistema perde a referência de posição angular horizontal do braço e, por motivos de segurança, impede seu movimento horizontal. Dessa forma, para retornar a funcionar, era necessário ligar o motor da base de outra forma, até o pino à frente do braço entrar em contato com algum da base.

Para resolver esse problema passou-se a salvar, sempre na memória EEPROM do microcontrolador, o último pino da base em que o pino do braço esteve em contato, permitindo, assim, uma forma de localizar a posição do braço horizontalmente e, então, movê-lo para o pino desejado a qualquer instante.

B. Reconhecimento do Objeto

A Figura 10 mostra o reconhecimento dos objetos vermelho e azul através do processamento da imagem, sendo a imagem em RGB e, ao lado a imagem filtrada e segmentada.

Ademais, o reconhecimento da posição dos objetos funcionou conforme a expectativa. As peças foram comutadas dentre as 6 posições cerca de 50 vezes, e o sistema reconheceu cada posição em menos de 1 segundo, contando desde o momento em que a peça era colocada na devida posição; mesmo em momentos onde as peças estavam próximas umas das outras, o sistema não falhou. Contudo, vez ou outra, ao comutar o objeto a ser reconhecido, ocorreram demoras entre 5 a 10 segundos. Portanto, o resultado do reconhecimento dos objetos através da cor se mostrou eficiente e satisfatório.

C. Software de controle e Interface

No software é possível escolher a cor da peça, iniciar a execução do sistema de controle e selecionar, ou ajustar, as cores dos objetos. Para isso, basta clicar na cor que quer selecionar ou ajustar, e após clicar no objeto correspondente a essa cor. Assim, o sistema salvará os dados e reconhecerá aquele objeto sempre que for escolhido, ainda que o *Software*

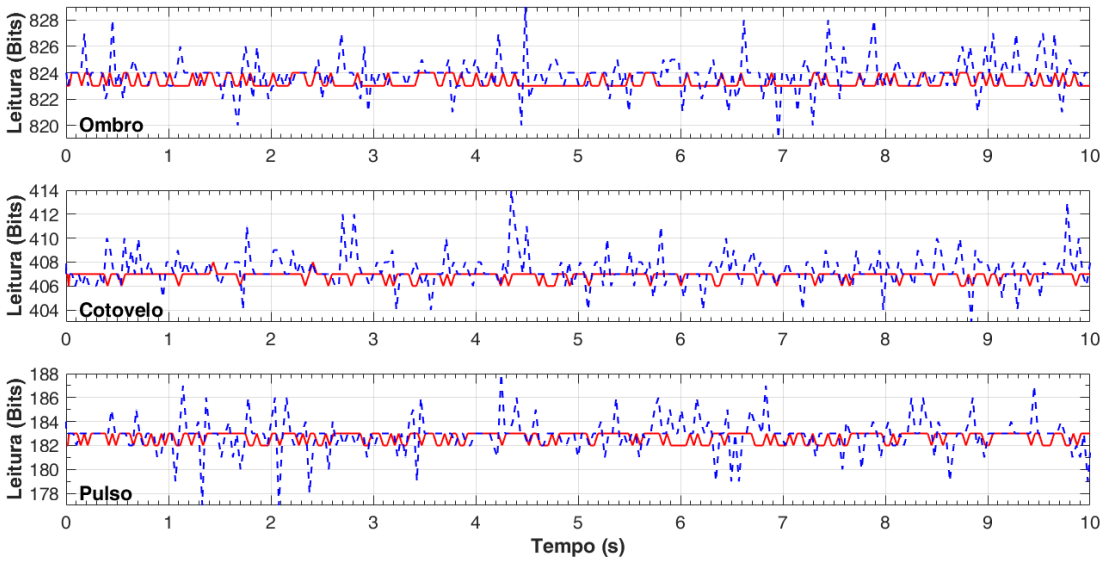

Fig. 9. Leitura dos potenciômetros fixados no braço robótico por 10 segundos.

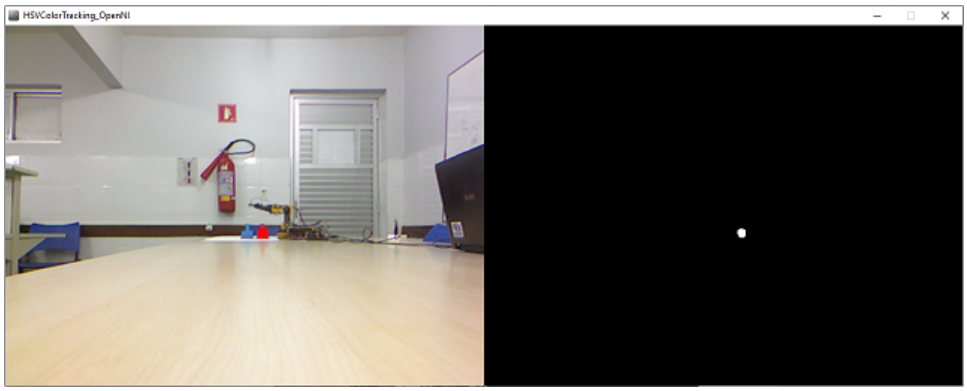

Fig. 10. Reconhecimento do objeto vermelho e imagem filtrada correspondente à uma distância de 2 metros do *Kinetc*.

seja reiniciado. Para ajustar as posições, basta clicar no botão "Posições Peça" que o *Software* irá ser direcionado à *interface* de calibração.

Em seguida, selecionar a posição que se quer ajustar, posicionar o objeto a ser capturado na posição em questão e apertar no botão "Calibrar". Logo, os dados de posição do objeto adquiridos pelo sistema de visão computacional são salvos.

Finalmente, para finalizar os ajustes das posições basta clicar no botão "Posições Garra" que irá direcionar o *Software* à uma terceira interface. A Interface para calibração das posições do braço, após selecionar a posição a ser definida, movimentar a garra através do teclado do computador, posicioná-la da melhor forma de capturar o objeto naquela posição e apertar o botão "Calibrar". Assim sendo, o microcontrolador lê os valores indicados nos potenciômetros, bem como em qual pino da base a garra está e, na sequência, os dados desta posição são salvos e recuperados sempre que é ligado.

D. Aspecto de Geral de Funcionamento

Na área de trabalho criada para a realização dos testes finais foram configuradas 6 posições para a captura dos objetos e duas posições para a soltura dos mesmos conforme a cor, o que pode ser observado na Figura 11. Além disso, o *Kinect* foi alocado a uma distância média de 2 metros da área de trabalho, e as posições, tanto da garra quanto das peças, foram ajustadas utilizando o *Software* desenvolvido.

Foram realizadas sucessivas tentativas de captura em cada posição, começando pela posição 1 e seguindo em ordem crescente até que se completasse 15 em cada uma. Os resultados podem ser vistos na Tabela II.

TABELA II
RESULTADOS DOS TESTES FINAIS

Posição	Tentativas	Capturas	Falhas	Erro (%)
1	16	15	1	6,25
2	16	15	1	6,25
3	16	15	1	6,25
4	16	15	1	6,25
5	17	15	2	11,76
6	18	15	3	16,67

Com estes resultados pode-se comprovar que o sistema de controle apresentou uma eficiência satisfatória ao que se

propõe, mas observou-se, também, que o mesmo começa a apresentar falhas após muitas repetições ininterruptas. Tais falhas são atribuídas ao uso de contatos na junta da cintura, dada a impossibilidade de adaptar um potenciômetro para esta junta. Os contatos, por sua natureza discreta, causavam erros de quantização na posição que acabavam se acumulando ao longo das repetições.

Finalmente, constatou-se que tanto o software como a integração entre o sistema de visão computacional e o sistema de controle funcionaram exatamente como esperado, bem como os dados a serem salvos em arquivos externos e na memória EEPROM do microcontrolador foram corretamente salvos e importados quando necessário.

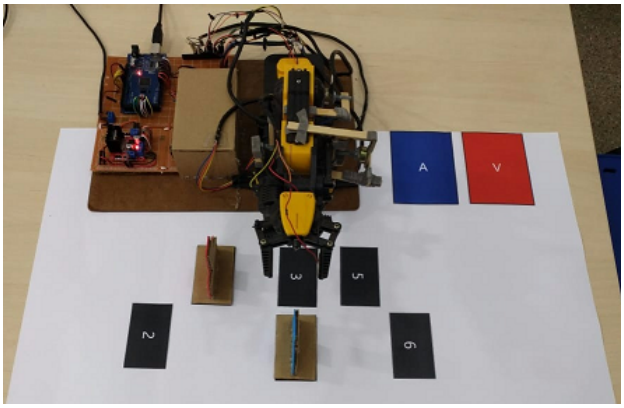

Fig. 11. Área de trabalho construída para a realização dos testes.

IV. Conclusão

Neste trabalho foi desenvolvido um sistema de controle PID para um braço robótico integrado a um sistema de visão computacional utilizando *Kinect*. Foi desenvolvido, também, um *Software* para operação e integração dos dois sistemas.

O presente trabalho teve significativa relevância para um melhor entendimento nas áreas de controle, automação e robótica, presentes no curso de Engenharia Elétrica, assim como na integração da computação com a engenharia. Além disso, o próprio se mostrou bastante didático na sua forma de tratar o tema, principalmente pelo uso de plataformas gratuitas ou de baixo custo, abrindo, ainda, muitas portas e possibilidades para seu aperfeiçoamento e uso em escala industrial.

Os resultados mostraram que o sistema de visão computacional, o software, o sistema de controle PID e a comunicação entre eles funcionaram de forma adequada. Ainda assim, houve falhas quanto a repetibilidade do sistema causadas pelo método implementado para detecção de posição angular da cintura.

Portanto, os objetivos foram alcançados com sucesso, propondo-se, ainda, como novas implementações, a utilização de braços robóticos com mais graus de liberdade e com realimentação própria da posição dos elos e efetuador, o uso de cinemática direta e inversa para se obter um sistema mais inteligente e com mais liberdade, a utilização de linguagens de programação mais robustas, mas ainda assim *Open Sources*, como *Python*, e, finalmente, a completa automatização do braço robótico para determinados serviços baseados na classificação de objetos através da visão computacional.

Agradecimentos

Agradecemos ao apoio e suporte nas atividades pelo Grupo de Pesquisa em Modelagem de Sistemas Elétricos – GPMSE, DEE/NT/UNIR.

Referências

[1] BRANCO, L. Dados & Ideias - A economia só tem a agradecer aos robôs, Revista Exame, Ed. Abril, 15 de março, 2018.
[2] ROMANO, V. F. Robótica Industrial - Aplicações na indústria de manufatura e de processos. ISBN: 8521203152, Ed. Blucher, São Paulo, SP, 2002.
[3] BACKES, A.; JUNIOR, J. J. d. M. S. Introdução à Visão Computacional Usando *Matlab*. Rio de Janeiro, RJ, Ed. Alta Books, 2016.
[4] SENA, H. J. d. S. Controle Robótico Referenciado por Sistema de Visão Computacional Utilizando o *Kinect*. Trabalho de Final de Curso, Faculdade Assis Gurgacz, Cascavel, PR, 2011.
[5] LADISLAO M., A. Caverzasi; F. Saravia; G. Gómez; J. Pedroni. Detection of Human-Robot Collision using Kinetic, Vol.:11, Iss.:1, ISSN: 1548-0992, IEEE Latin America Transactions, Feb. 2013.
[6] CORREA, M. V. d. C. Controle de robôs móveis utilizando *Kinect*, Trabalho de Conclusão de Curso, Escola de Engenharia de São Carlos, USP, São Carlos, SP, 2014.
[7] JARDIM, Laercio Arraes. Sistema de visão robótica para reconhecimento e localização de objetos sob manipulação por robôs industriais em células de manufatura. 110 f. Dissertação (Mestrado em Sistemas Mecatrônicos) Universidade de Brasília, Brasília, DF, 2006.
[8] OPENNI. *OpenNI* - The standard framework for 3D sensing. Disponível em: http://openni.ru/index.html.
[9] OPENCV. Open Source Computer Vision Library. 2018. Disponível em: https://opencv.org/about.html.
[10] *Arduino*. *Arduino* MEGA 2560 & Genuíno MEGA 2560. 2018. Disponível em: https://www.arduino.cc.
[11] BELLON, Olga Regina Pereira. Visão computacional: um sistema para localização de objetos poliédricos no espaço 3 D, [106]f. Dissertação (mestrado) - UNICAMP, FEE, Campinas, SP, 1990.
[12] RAMOS, D. A sucessão de erros e problemas que acompanha a história do *Kinect*, Dez., 2015. Disponível em: https://canaltech.com.br/games/a-sucessao-de-erros-e-problemas-que-acompanha-a-historia-do-kinect-55031/
[13] PAULA, B. C. d. Adaptando e desenvolvendo jogos para uso com o *Microsoft Kinect*. XI SBGames – Brasília – DF – Brazil, November 2nd - 4th, 2012.
[14] CHANG, Y.-J.; CHEN, S.-F.; HUANG, J.-D. A Kinect-based system for physical rehabilitation: A pilot study for young. Elsevier, p. 2566–2570, 2011.
[15] CHEN, C.-T. Linear System Theory and Design. New York: Oxford University Press,1999.
[16] OGATA, K. Engenharia de Controle Moderno. São Paulo: Pearson Prentice Hall, 2010.
[17] SILVA, G. J.; DATTA, A.; BHATTACHARYYA, S. P. New Results on the Synthesis of PID Controllers. IEEE Transactions on Automatic control, v. 47, n. 2, p. 241–252, 2002.
[18] KNOSPE, C., PID control, in IEEE Control Systems Magazine, doi: 10.1109/MCS.2006.1580151, vol. 26, no. 1, pp. 30-31, Feb. 2006.
[19] ASTROM, K.; HAGGLUND, T. PID controllers: theory, design and tuning. Research Triangle Park: Instrument Society of America, 1995. 343 p. ISBN 1556175167.
[20] FREITAS, C. M. Controle PID em sistemas embarcados. 2014. Disponível em: https://www.embarcados.com.br/controle-pid-em-sistemas-embarcados/.
[21] TORRES, W. L., Araujo. I. B. Q., Filho. J. B. M., and COSTA Jr. A. G., Mathematical Modeling and PID Controller Parameter Tuning in a Didactic Thermal Plant, in IEEE Latin America Transactions, vol. 15, no. 7, pp. 1250-1256, doi: 10.1109/TLA.2017.7959343, 2017.
[22] Processing Foundation. Overview. A short introduction to the Processing software and projects from the community. Disponível em: https://processing.org/overview/.

[23] HIJDEN, P. v. d. How to Connect a Kinect. 2015. Disponível em: http://www.instructables.com/id/How-to-Connect-a-Kinect/.
[24] OLIVEIRA, B. A. S. et al. Detecção e Rastreamento de Objetos coloridos em vídeo utilizando o *OpenCV*. VII Semana de Ciência e Tecnologia IFMG – campus Bambuí; VII Jornada Científica e I Mostra de Extensão, out. 2014.