Dispositivo de Selagem para Manipulador Industrial

Lucas Caliento Barone [1] e Roberto Santos Inoue [2]

***Resumo*— Há gerações o ser humano tem buscado maneiras de facilitar seu trabalho, sendo a automação industrial uma delas. Neste contexto, os braços robóticos são utilizados para reproduzir tarefas manuais com maior eficiência, como no processo de selagem utilizado na aeronáutica. O processo de selagem, ao ser aplicado em junções, rebites, furos e dobras; é responsável por confinar fluidos, manter a pressurização e adicionar suavidade aerodinâmica à superfícies expostas, como nas asas do avião. Assim, a automatização desse processo é de suma importância a fim de que ela desempenhe corretamente suas funções. Desta forma, este trabalho propõe desenvolver algoritmos que sejam capaz de automatizar este processo para qualquer tipo de asa. Utilizou-se o MATLAB® e o sensor de laser Hokuyo UST-10LX® para a identificação do ponto de interesse a ser selado; e o Arduino IDE®, Python integrado ao ROS® (Robot Operating System) e células de carga, para mensurar a força no bico de aplicação de selagem. Os resultados dos testes do laser pelo MATLAB® e dos testes dos sensores de força, demonstram tecnologias que podem auxiliar na selagem automática. Essas soluções poderão futuramente ser integrados em manipuladores robóticos para se automatizar o processo por inteiro de selagem.**

Palavras-Chave - Manipulador Industrial, ROS®, Hokuyo UST-10LX®, Células de carga, Selagem Automática.

I. Introdução

A industrialização em conjunto com a globalização, propiciou o acelerado avanço tecnológico, aumentando a competição entre industrias, criando a necessidade destas se modernizarem rapidamente e de oferecem diferenciais em seus produtos; como o aumento de qualidade, redução de custos e preços mais acessíveis. Esse conjunto de processos e técnicas seria posteriormente denominado de ”automação industrial” [1].

Neste contexto, os braços robóticos são utilizados na automação industrial por reproduzirem tarefas repetitivas, as quais, normalmente são realizadas manualmente. Estes possuem vantagens, tais quais, o aumento na escala e velocidade de produção, na precisão, na uniformidade de produção e na segurança do trabalho ao eliminar funções perigosas. Exemplos de atividades exercidas eficientemente por braços robóticos são: soldagem, separação de peças ou materiais, montagem mecânica e eletrônica, entre outros [2].

Diante dessas aplicações na indústria, o processo de selagem, ao ser aplicado em junções, rebites, furos e dobras; é responsável por confinar fluidos, manter a pressurização e adicionar suavidade aerodinâmica à superfícies expostas. Neste contexto, a automatização nesse processo é de suma importância a fim de que ele desempenhe corretamente suas funções [3].

O presente projeto, realizará partes do procedimento de selagem automática a ser utilizado futuramente em um manipulador industrial, utilizando o periférico laser Hokuyo UST-10LX® para mapear em tempo real as regiões de interesse. Uma vez mapeado, um algoritmo no ambiente MATLAB® será responsável por transcrever estes dados em distância até os pontos observados.

A fim de determinar a força a ser aplicada no bico de selagem e viabilizar o envio de informação, será utilizado células de carga, as quais são sensores de força cujo funcionamento se baseia no uso de extensômetros ligados em circuito tipo ponte. Este circuito promove um desbalanço de tensão proporcional a variação de resistência, consequente da deformação elástica sofrida por uma força aplicada [4].

Posteriormente, as células de carga serão embarcadas em placa Arduino Leonardo® com programação no Arduino IDE® e enviadas via comunicação serial para um programa desenvolvido em Python, assim futuramente poderão ser controladas com *offset* desejado.

Além disso, para que futuramente ocorra comunicação do laser com as células de carga, será utilizado a plataforma de código aberto ROS® (Robot Operating System). Esta plataforma fornece bibliotecas e ferramentas prontas auxiliando o desenvolvimento de softwares aplicáveis em robótica, sendo importante para a troca de mensagens do laser e sensores de força com o manipulador robótico [5].

Sendo assim, o presente projeto, tem como objetivo utilizar o processo de automatização para viabilizar um modelo de selagem, capaz de produzir filetes para qualquer aplicação, sem a necessidade da criação de algoritmos específicos para cada uma.

Este trabalho está organizado da seguinte forma: na Seção II é apresentado os materiais e métodos utilizados para desenvolver o algoritmo do laser e o sensor de força, na Seção III é apresentado os resultados e discussões obtidos com a leitura do laser e da força, e na Seção IV é apresentado a conclusão do projeto.

II. Materiais e Métodos

Os materiais e métodos foram divididos em duas seções. Sendo a Seção II-A, a metodologia utilizada para a obtenção de um algoritmo capaz de identificar automaticamente as regiões de interesse a serem seladas; na Seção II-B, a metodologia realizada para medição de força do aplicador de selagem e sua integração com o ROS®. O sistema operacional escolhido no desenvolvimento do presente projeto foi

[1] Autor é aluno do Departamento de Engenharia Elétrica, Universidade Federal de São Carlos, UFSCar, São Carlos, Brasil lucascbarone@hotmail.com.

[2] Orientador e professor adjunto do Departamento de Computação, Universidade Federal de São Carlos, UFSCar, São Carlos, Brasil rsinoue@gmail.com.

o Ubuntu, por apresentar compatibilidade com a plataforma ROS®.

A. Algoritmo do Laser

O desenvolvimento do algoritmo de detecção automática da região de interesse para aplicação da selagem foi realizado utilizando o software de programação MATLAB® com o periférico Hokuyo UST-10LX® (Figura 1). O motivo de sua escolha foi devido à sua precisão, alcance, compactabilidade e rápida resposta serem compatíveis com o que era necessário para o projeto [6].

Fig. 1. Hokuyo UST-10LX® [7] .

O laser possui faixa máxima de leitura de 270º, contudo através de testes em tempo real em que se visualizava a leitura do laser em um gráfico (x, y), determinou-se a faixa para este projeto em 75º.

Com a faixa definida, foi possível determinar o método necessário para que o algoritmo encontrasse automaticamente e em tempo real o ponto de interesse.

Para isso, partiu-se do pressuposto de que o ponto de interesse a ser detectado seria o ponto mais distante do laser. E sendo as coordenadas (x) a distância até o ponto de colisão e (y) a distância horizontal até o mesmo ponto, como ilustrado pela Figura 2, foram utilizadas funções no MATLAB® a fim de encontrar o máximo valor de (x) e utilizá-lo para encontrar o valor correspondente de (y), plotando em gráfico com indicador visual esse ponto de interesse, veja Apêndice I.

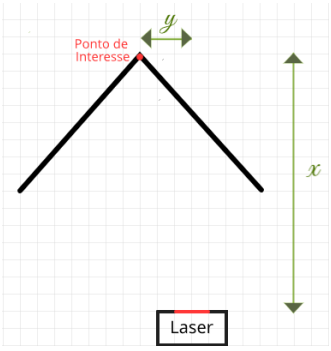

Fig. 2. Esquemático do ponto de interesse.

B. Sensor de Força

Para a implementação do algoritmo de leitura de força, foi utilizado o software do Arduino IDE® para programar um Arduino Leonardo®, visto na Figura 3. O software desenvolvido seria responsável por mensurar a força dos sensores e enviá-las via comunicação serial, para que um programa em Python recebesse esta informação e transmitisse para o manipulador industrial utilizando ROS®. Por isso, o Arduino Leonardo® foi escolhido por conter memória e portas digitais suficientes para o projeto, tendo como diferencial seu tamanho compacto e preço mais acessível do que outros modelos.

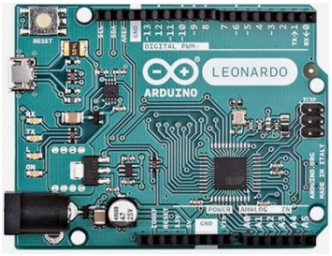

Fig. 3. Arduino Leonardo® [8] (adaptado) .

O sensor de força é composto por 3 células de carga com sensibilidade de até 2kg. Cada uma dessas células foi conectada a um amplificador e conversor Analógico/Digital próprio, o HX711. A célula escolhida e o HX711 podem ser vistos pela Figura 4 e Figura 5, respectivamente.

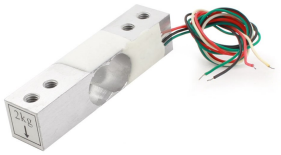

Fig. 4. Célula de Carga de 2kg [9] .

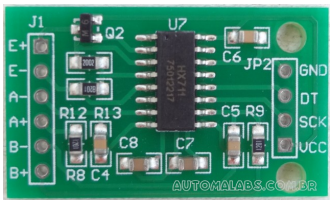

Fig. 5. Conversor A/D HX711 [10] .

Dessa maneira, as células puderam ser conectadas ao Arduino® conforme diagrama mostrado na Figura 6. Os fios vermelho e preto da célula de carga se conectam ao HX711 pelos pinos E+ e E-, respectivamente, que serão responsáveis por alimentá-la. Já os fios verde e cinza são conectados aos pinos A+ e A- e serão responsáveis pela leitura do sinal da célula. Os pinos VCC e GND serão conectados ao Arduino® nos pinos de 5V e GND, respectivamente, e serão

responsáveis por alimentar o circuito. Por fim os pinos SCK e DOUT são conectados aos pinos digitais do Arduino® para que possa ser efetuada a medição da célula por software desenvolvido no Arduino IDE®.

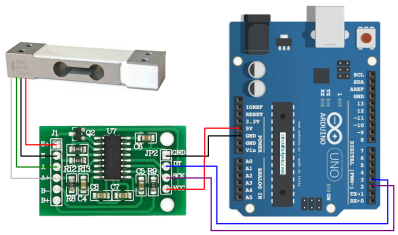

Fig. 6. Diagrama de ligação da célula de carga com o conversor A/D, e do conversor A/D com o Arduino® [11] .

No entanto, ao interligar o conjunto de células, conversores e Arduino®, o sistema se apresentou mais propício a falhas devido aos *jumpers*. Desta forma, optou-se por desenvolver um *shield* para o Arduino®, utilizando o software Eagle®. O *shield* desenvolvido no software e sua placa impressa podem ser vistos pela Figura 7.

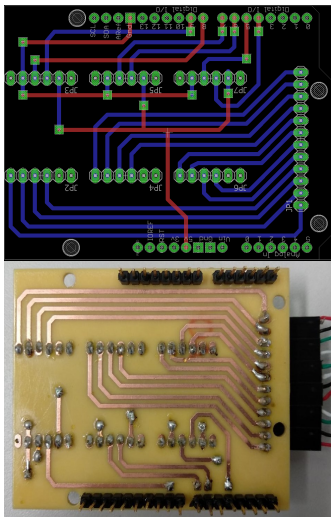

Fig. 7. *Shield* desenvolvido para o Arduino Leonardo® (em cima) e *shield* impresso (embaixo).

O *shield* foi projetado com o intuito de ter a mesma dimensão do Arduino Leonardo®, podendo acomodar os conversores HX711 encaixados nele e as células de carga encaixadas com *jumpers* fêmea na placa. Desta forma, a placa do *shield* foi impressa e os componentes foram soldados de acordo com seus respectivos lugares.

Para que fosse possível se trabalhar com as células de carga, foi necessário a adição da biblioteca "HX711 ADC" ao Arduino IDE®. Além de possuir ferramentas necessárias para se trabalhar com as células em conjunto com o HX711, essa biblioteca apresenta um código pronto para calibração das células e um código para a leitura de até 2 células de carga simultaneamente [12].

Para a calibração das células, foram aplicados pesos conhecidos, alterando seus ganhos no algoritmo até que a leitura obtida correspondesse ao valor esperado.

Calibrada as células e usando o algoritmo de leitura de 2 células de carga como base, foi desenvolvido um algoritmo capaz de realizar a leitura de 3 células de carga, veja Apêndice II. Para montar a palavra com as forças medidas na mensagem serial, foi utilizado a classe "Union". Essa classe tem como característica permitir a alocação de um espaço de memória para mais de um tipo de variável. No caso, foi utilizado os tipos "int" e "char" [13].

Uma variável do tipo "int" quando declarada ocupa 2 bytes, enquanto que uma variável do tipo "char" ocupa apenas 1 byte. Podendo representar uma variável do tipo "int" com 2 "char" [13].

Utilizando esta classe, foi multiplicado a variável "float" de força de cada célula por um fator de 10, e em seguida armazenado em novas variáveis com tipo "int". Desta maneira, foi possível criar um vetor de tipo "char" e armazenar as forças ao transformar cada variável de força "int" em 2 "char" correspondentes.

Assim, a mensagem foi armazenada no vetor com o formato $xxyyzz#, no qual $ e #, são caracteres que identificam o início e fim da mensagem respectivamente. Além disso, os caracteres xx, yy e zz são os valores de força de cada célula.

Uma vez que o vetor foi corretamente preenchido com as forças, foi enviado via comunicação serial para o computador executando ROS®.

Com o auxílio do ROS®, foi implementado um algoritmo em Python que fosse capaz de interpretar a mensagem enviada via serial pelo Arduino®, veja Apêndice III.

Para conseguir interpretar a mensagem, foi utilizada a função "unpack". Essa função recebe como parâmetros de entrada a maneira que se deve considerar cada byte da mensagem recebida, assim como o vetor que contém a mensagem em questão. A maneira com que se desejou interpretar os bytes recebidos é representado por uma letra "h" para cada uma das 3 forças. Esta sigla indica que a mensagem será lida como "int" de 2 bytes. O vetor contendo a mensagem em questão foi declarado como "inMessage" [13].

O argumento de saída da função é um vetor de 3 posições, cada uma contendo um dos 3 valores de força enviados pelo Arduino®. Em seguida cada força foi dividida por um fator de 10 para que retornassem ao valor correto de leitura. Por fim, utilizou-se a mensagem do tipo *wrench* do ROS® para publicar simultaneamente os valores das 3 forças, tal como uma média aritmética destas. *Wrench* é um subtipo de mensagem do tipo *geometry_msgs* do ROS®, capaz de representar forças no espaço separado em suas partes linear e angular [14]. Este tipo de mensagem pode armazenar dados como "force" ou "torque", cada uma contendo 3 variáveis (x, y, z). Assim, as 3 forças foram armazenadas nas variáveis (x, y, z) do "torque", e a média aritmética destas na variável (z) do "force".

As 3 células de carga foram fixadas em uma peça circular de metal, dispostas 120º umas das outras e com suas pontas voltadas para o centro. Esse conjunto foi acoplado a um aplicador de selagem, de maneira com que a aplicação de

força no bico do aplicador transmitisse essa força de forma centralizada ao conjunto de células de carga, sendo possível medir esta força. Veja Figura 13.

III. Resultados e Discussões

Os resultados obtidos foram divididos nas Seções III-A e III-B de acordo com a metodologia aplicada. A Seção III-A apresenta os resultados obtidos com a simulação do laser pelo MATLAB®, enquanto que a Seção III-B, apresenta os resultados obtidos com a simulação dos sensores de força no terminal do Ubuntu®.

A. Leitura do Laser

A leitura do laser foi efetuada de acordo com o método descrito na Seção II-A, e o código desenvolvido no MATLAB® pode ser visualizado no Apêndice I.

Na Figura 8 é possível visualizar uma leitura teste realizada com o laser para verificar seu campo de visão completo de 270º. Nesta imagem, o laser se situa na origem $(0, 0)$ e o ponto de interesse aproximadamente em $(0, 0.3)$.

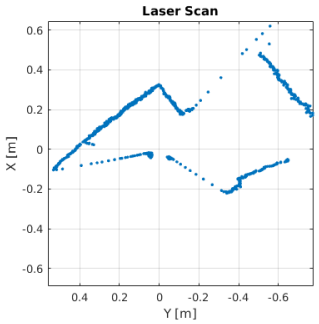

Fig. 8. Visão Completa do Hokuyo®.

Já na Figura 9 é possível visualizar o gráfico plotado pelo código do Apêndice I. Com o ângulo de visão reduzido para uma visão frontal de aproximadamente 75º, fica evidente o ponto de interesse, o qual é demonstrado visivelmente em tempo real pelo símbolo ”*”.

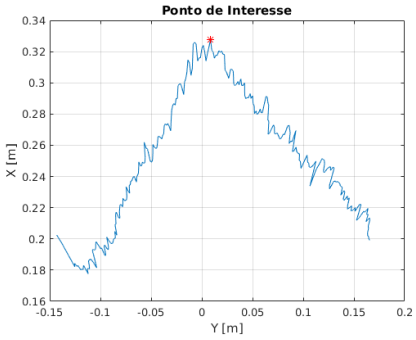

Fig. 9. Visualização de Ponto de Interesse com Hokuyo®.

B. Leitura da Força

Como projetado de acordo com o método descrito na Seção II-B, é possível visualizar o *shield* por cima e pelo lado na Figura 10, sendo possível verificar seu encaixe no Arduino® e as conexões com as células de carga.

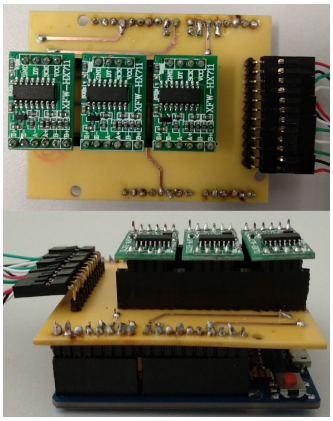

Fig. 10. *Shield* visto por cima (em cima) e vista lateral do conjunto (embaixo).

Na Figura 11 é possível visualizar todo o conjunto dos circuitos com as células de carga, e o código desenvolvido no Arduino IDE® para leitura das forças e seu envio via serial pode ser visualizado no Apêndice II.

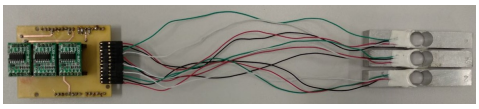

Fig. 11. Vista superior do conjunto.

A seguir, na Figura 12 é possível visualizar um teste da leitura das forças pelo terminal do Ubuntu®. A mensagem “wrench.force.z” apresenta a média aritmética das 3 forças lidas pelas 3 células de carga, as quais cada uma tem sua força apresentada por “x”, “y” e “z” na mensagem “wrench.torque”. O código para leitura das forças no terminal foi desenvolvido em Python utilizando ROS® e pode ser visualizado no Apêndice III.

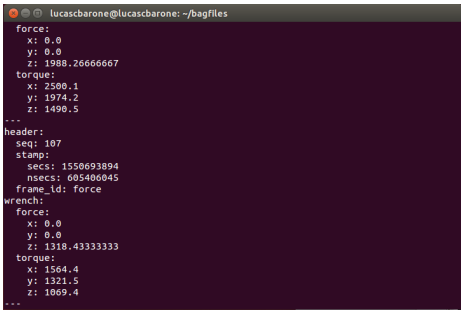

Fig. 12. Leitura de força pelo terminal do Ubuntu®.

Com o acoplamento do conjunto de células de carga na estrutura do aplicador de selagem, temos uma visão geral de sua forma final na Figura 13.

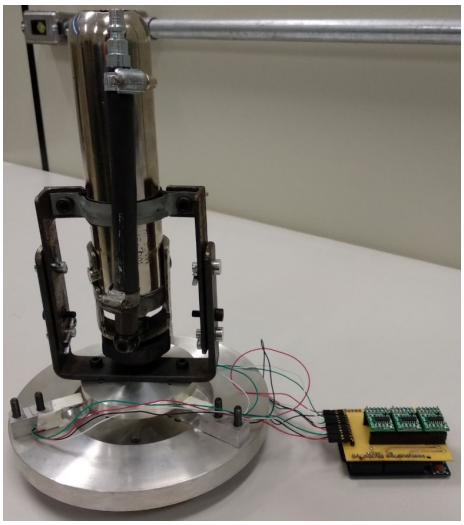

Fig. 13. Conjunto final com o aplicador.

Fica evidente como o design do *shield* foi de grande importância tanto para o valor estético do projeto, quanto para o funcional. Desta maneira, reduziu-se consideravelmente o espaço que o conjunto antes ocupava, eliminou-se o uso de *jumpers* e organizou-se para que as células se conectassem ordenadamente na borda da placa. Assim, foi possível conectar as células no aplicador de maneira organizada e com espaçamento suficiente para seus fios.

Por fim, na Figura 14 é possível visualizar um gráfico plotado de uma simulação das células com o aplicador ao se aplicar variadas forças ao longo do tempo em seu bico.

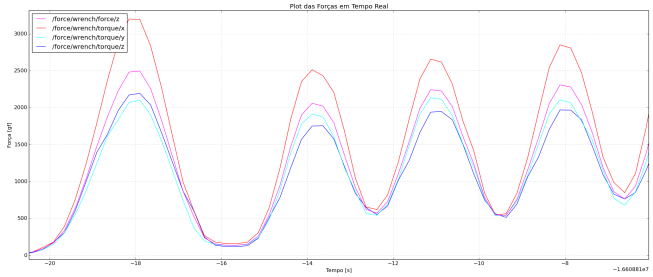

Fig. 14. Plotagem das forças em tempo real.

IV. Conclusões

Em suma, o presente projeto demonstra o sucesso no desenvolvimento de dois algoritmos, um capaz de obter as coordenadas do ponto de interesse para a selagem e outro de mensurar a força no bico de selagem, ambos necessários para o processo de automatização da selagem aeronáutica. Além disso, foram integrados com o ROS® para que futuras pesquisas possam interligá-los com o braço robótico. Entretanto, estudos adicionais mensurando a confiabilidade e a precisão do processo poderão ser realizados de acordo com as necessidades do projeto.

Apêndice I
Código do Laser

```
scansub = rossubscriber('/scan10');
while 1
    scan = receive(scansub)
    cartScanData = scan.readCartesian;
    cartScanData(:,3) = 0;
    x = cartScanData(400:700,1);
    y = cartScanData(400:700,2);
    [maxx, indexOfMaxX] = max(x);
    yAtMaxX = y(indexOfMaxX);
    plot (y,x)
    hold on
    plot (yAtMaxX,maxx,'r*')
    title('Ponto de Interesse')
    xlabel('Y [m]')
    ylabel('X [m]')
    grid on
    pause(.05)
    clf
end
homScanData = cart2hom(cartScanData);
scandata = rosmessage('sensor_msgs/
    LaserScan')
```

Apêndice II
Código do Arduino Para Leitura das Células de Carga®

```
#include <HX711_ADC.h>

//HX711 constructor
HX711_ADC LoadCell_1(5, 4); //HX711 1 (
    sck pin, dout pin)
HX711_ADC LoadCell_2(7, 6); //HX711 2 (
    sck pin, dout pin)
HX711_ADC LoadCell_3(8, 9); //HX711 3 (
    dout pin, sck pin)
union Data {
  int i;
  char b[2];
  };

long t;

void setup() {
  Serial.begin(9600);
  Serial.println("Wait...");
  LoadCell_1.begin();
  LoadCell_2.begin();
  LoadCell_3.begin();
  long stabilisingtime = 2000;
  byte loadcell_1_rdy = 0;
  byte loadcell_2_rdy = 0;
  byte loadcell_3_rdy = 0;
  while ((loadcell_1_rdy +
      loadcell_2_rdy + loadcell_3_rdy) <
      3) {
    if (!loadcell_1_rdy) loadcell_1_rdy
        = LoadCell_1.startMultiple(
```

```
        stabilisingtime);
    if (!loadcell_2_rdy) loadcell_2_rdy
        = LoadCell_2.startMultiple(
        stabilisingtime);
    if (!loadcell_3_rdy) loadcell_3_rdy
        = LoadCell_3.startMultiple(
        stabilisingtime);
  }
  LoadCell_1.setCalFactor(812.0); //
      user set calibration factor (float)
  LoadCell_2.setCalFactor(920.0); //
      user set calibration factor (float)
  LoadCell_3.setCalFactor(896.0); //
      user set calibration factor (float)
}

void loop() {
  LoadCell_1.update();
  LoadCell_2.update();
  LoadCell_3.update();

  //get smoothed value from data set +
      current calibration factor
  if (millis() > t + 250) {
    float force1 = LoadCell_1.getData();
    float force2 = LoadCell_2.getData();
    float force3 = LoadCell_3.getData();
    outputMsg (force1,force2,force3);
    t = millis();
  }

  //receive from serial terminal
  if (Serial.available() > 0) {
    float i;
    char inByte = Serial.read();
    if (inByte == 't') {
      LoadCell_1.tareNoDelay();
      LoadCell_2.tareNoDelay();
      LoadCell_3.tareNoDelay();
    }
  }
}

//Send Output Message

void outputMsg (float force1, float
    force2, float force3){
  char outputChar[10];
  union Data {
  int i ;
  char b [ 2 ] ;
} ;
  Data PrintFor1, PrintFor2, PrintFor3;

  PrintFor1.i = (int)(force1*10);
  PrintFor2.i = (int)(force2*10);
  PrintFor3.i = (int)(force3*10);

  outputChar[0] = '$';
  outputChar[1] = PrintFor1.b[0];
  outputChar[2] = PrintFor1.b[1];
  outputChar[3] = PrintFor2.b[0];
  outputChar[4] = PrintFor2.b[1];
  outputChar[5] = PrintFor3.b[0];
  outputChar[6] = PrintFor3.b[1];
  outputChar[7] = '#';
  outputChar[8] = '\n';
  outputChar[9] = '\0';

  Serial.write(outputChar,10);
}
```

APÊNDICE III
CÓDIGO DO ROS®

```
import serial
import rospy
import tf
from std_msgs.msg import String
from struct import unpack, pack
from geometry_msgs.msg import
    WrenchStamped

DEVICE = rospy.get_param("~device",'/dev
    /ttyACM0')
BAUD = rospy.get_param("~baud",9600)
usbPort = serial.Serial (DEVICE, BAUD ,
    timeout=5)


def ForceSensor():
    force_pub = rospy.Publisher("force",
     WrenchStamped, queue_size=1)
     rospy.init_node('force_publisher',
    anonymous=True)

     rate = rospy.Rate(100) #10Hz


     last_time = rospy.Time.now()
     force1 = 0.0
     force2 = 0.0
     force3 = 0.0
     force  = 0.0


     while not rospy.is_shutdown():
         current_time = rospy.Time.now()
         inChar = usbPort.readline(1)
         if inChar == "$" :
             inMessage = usbPort.read(7)
             usbPort.flushInput()

             if  inMessage[6] == "#":
```

```
                forceSensors = unpack ("
    hhh", inMessage[0:6])
                force1 = forceSensors
    [0]*0.1
                force2 = forceSensors
    [1]*0.1
                force3 = forceSensors
    [2]*0.1
                force = (force1 + force2
     + force3)/3

            else:
                usbPort.flushInput()
        else:
            usbPort.flushInput()

        # next, we'll publish the force
    message over ROS
        force_msg = WrenchStamped()
        force_msg.header.stamp =
    current_time
        force_msg.header.frame_id = "
    force"

        # set the force
        force_msg.wrench.force.x = 0.0
        force_msg.wrench.force.y = 0.0
        force_msg.wrench.force.z = force

        # set the forces
        force_msg.wrench.torque.x =
    force1
        force_msg.wrench.torque.y =
    force2
        force_msg.wrench.torque.z =
    force3

        # publish the message
        force_pub.publish(force_msg)

        last_time = current_time
        rate.sleep()

if __name__ == '__main__':
    try:
        ForceSensor()
    except rospy.ROSInterruptException:
        pass
```


AGRADECIMENTOS

A meus pais por me dar a oportunidade e por incentivar os estudos.

Aos meus amigos e colegas de curso pela amizade e pelo companheirismo, sem os quais não teria chegado aonde cheguei.

Aos professores do departamento de engenharia elétrica pelos ensinamentos tanto dentro quanto fora de sala de aula.

Ao Roberto Santos Inoue por me orientar neste trabalho e dispor de seu tempo e esforço para me auxiliar neste período.

REFERÊNCIAS

[1] J. Rosario, *Automação industrial*. BARAUNA. [Online]. Available: https://books.google.com.br/books?id=YsUHLcHdbh4C

[2] V. Carrara, "Introdução à robótica industrial," *Instituto Nacional de Pesquisas Espaciais (INPE)*, p. 1, 2015.

[3] U. S. F. A. A. United States. Flight Standards Service, *Airframe and powerplant mechanics airframe handbook [electronic resource]*. U.S. Dept. of Transportation, Federal Aviation Administration, Flight Standards Service, 1976.

[4] J. C. P. Beck, "Projeto, construção e análise de células de carga de placa e de anel," *Programa de Pós-Graduação em Engenharia Metalúrgica e dos Materiais (PPGEMM), Universidade Federal do Rio Grande do Sul,*, p. 1, 1983.

[5] "About ros," 2019, acessado: 2019-06-15. [Online]. Available: http://www.ros.org/about-ros

[6] M. Mirdanies and R. Saputra, "Experimental review of distance sensors for indoor mapping," *Journal of Mechatronics, Electrical Power, and Vehicular Technology*, vol. 8, no. 2, 2017.

[7] "Ust-10lx," 2016, acessado: 2019-06-15. [Online]. Available: https://www.hokuyo-usa.com/products/scanning-laser-rangefinders/ust-10lx

[8] "Arduino leonardo with headers," 2019, acessado: 2019-06-15. [Online]. Available: https://store.arduino.cc/usa/leonardo

[9] "2kg rectangle aluminium alloy parallel beam electronic scale load cell," 2014, acessado: 2019-06-15. [Online]. Available: https://www.amazon.co.uk/Rectangle-Aluminium-Alloy-Parallel-Electronic/dp/B00PZYSEJU

[10] "Módulo para leitura de células de carga hx711," 2018, acessado: 2019-06-15. [Online]. Available: https://www.automalabs.com.br/modulo-para-leitura-de-celulas-de-carga-hx711

[11] "Hx711 load cell amplifier interface with arduino," 2016, acessado: 2019-06-15. [Online]. Available: https://circuits4you.com/2016/11/25/hx711-arduino-load-cell

[12] "Arduino library for the hx711 24-bit adc for weight scales," 2019, acessado: 2019-02-11. [Online]. Available: https://github.com/olkal/hx711_adc

[13] V. I. Sgrignoli, "Prototipagem de um robô de tração diferencial," *Projeto de Conclusão de Curso - Programa de Graduação em Engenharia Elétrica, Universidade Federal de São Carlos, São Carlos*, pp. 27–28, 2017.

[14] "geometry_msgs/wrench message," 2019, acessado: 2019-08-30. [Online]. Available: http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Wrench.html