CENTRO DE ENGENHARIA, MODELAGEM E CIÊNCIAS SOCIAIS APLICADAS

PROGRAMA DE GRADUAÇÃO EM ENGENHARIA DE INSTRUMENTAÇÃO, AUTOMAÇÃO E ROBÓTICA

CRISTIANO MORAES BILLACHI AZARIAS, EDUARDO GASQUES MONTEIRO, LETICIA CIPRIANO DE SIQUEIRA

Orientador: Prof. Dr. Mário Gazziro

RECONSTRUÇÃO DE DIGITAL TWINS UTILIZANDO LIDAR: UMA ABORDAGEM PARA A AFERIÇÃO ANTROPOMÉTRICA

Santo André, SP

2022

UNIVERSIDADE FEDERAL DO ABC
CENTRO DE ENGENHARIA, MODELAGEM E CIÊNCIAS SOCIAIS APLICADAS
PROGRAMA DE GRADUAÇÃO EM ENGENHARIA DE INSTRUMENTAÇÃO,
AUTOMAÇÃO E ROBÓTICA

CRISTIANO MORAES BILLACHI AZARIAS, EDUARDO GASQUES MONTEIRO,
LETICIA CIPRIANO DE SIQUEIRA

RECONSTRUÇÃO DE DIGITAL TWINS UTILIZANDO LIDAR: UMA ABORDAGEM PARA A AFERIÇÃO ANTROPOMÉTRICA

Monografia apresentada ao Curso de Engenharia de Instrumentação, Automação e Robótica da Universidade Federal do ABC como parte dos requisitos necessários para a obtenção do grau de Engenheiro(a) de Controle e Automação.

Orientador: Prof. Dr. Mário Gazziro

Santo André, SP
2022

Cristiano Moraes Billachi Azarias, Eduardo Gasques Monteiro, Leticia Cipriano de Siqueira

RECONSTRUÇÃO DE DIGITAL TWINS UTILIZANDO LIDAR: UMA ABORDAGEM PARA A AFERIÇÃO ANTROPOMÉTRICA

Monografia apresentada ao Curso de Engenharia de Instrumentação, Automação e Robótica da Universidade Federal do ABC como parte dos requisitos necessários para a obtenção do grau em Engenheiro(a) de Controle e Automação.

Aprovada em Santo André, 04 de Novembro de 2022.

Prof. Dr. Mário Gazziro
Universidade Federal do ABC
Orientador

Prof. Dr. Agnaldo Aparecido Freschi
Universidade Federal do ABC - UFABC
Examinador

Dedicamos esse trabalho às nossas famílias sejam elas as de sangue ou as adquiridas ao longo de nossas vidas.

Agradecimentos

Agradecemos a todos os amigos e professores que tornaram possível a realização do curso.

Todas as vitórias ocultam uma abdicação. (Simone de Beauvoir)

Resumo

O presente projeto procurou desenvolver um escâner tridimensional afim de obter um gêmeo digital (*digital twin*), ou seja, uma réplica virtual do corpo sendo escaneado. Para isso, foi utilizado um sensor LiDAR fixo integrado a uma plataforma giratória. Os dados obtidos pelo sensor foram compilados de forma a unificar todos os registros em uma única malha de pontos. Essa malha então foi tratada para remover ruídos e por fim reconstruída para gerar um modelo digital, reparado via algoritmo *Alpha Shapes*.

Com o objetivo de desenvolver uma forma fácil, acessível e precisa de escanear uma pessoa, consolidamos essa ideia para contribuir com os avanços da área. No entanto, com as diversas limitações encontradas ao longo dos testes realizados para se obter o modelo, constatou-se que o sensor escolhido é falho em captar detalhes, além de ter limitações relacionas à cores nas superfícies escaneadas.

Palavras-chave: Escaneamento 3D. Modelagem. Medidas. Gêmeo Digital. Digital Twin. LiDAR. Plataforma giratória.

Abstract

The following project aims to develop a three-dimensional scanner to recreate a digital twin, i.e., a virtual copy of a scanned body. We used a standing-still LiDAR sensor attached to a rotating platform to achieve this goal. The sampled data were compiled and unified into a single point cloud. This point cloud, afterwards, had its outliers removed and then reconstructed into an ordered and integrated mesh using an Alpha Shape reconstruction.

Our broader goal was to develop an easy, accessible and accurate way to scan a person and also contribute to advances in the area. However, we faced some limitations during the testing phases. These limitations were primarily due to the sensor used once they were failures while recording the data, and the lack of information for some surface colours.

Keywords: 3D Scanning, Modeling, Measurements, Digital Twin, LiDAR, Rotating Platform.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

1D	Unidimensional
2D	Bidimensional
3D	Tridimensional
CI	Circuito Integrado
CLK	Clock
CSn	Chip Serial Number
DC	Direct Current
DO	Dados
FOV	Field Of View
FPS	Frames por segundo
GPIO	General Purpose Input/Output
Hz	Hertz
ICP	Iterative Closest Point
Kd	Ganho derivativo
Ki	Ganho integral
Kp	Ganho proporcional
LIDAR	Light Detection And Ranging
MHz	Mega Hertz
PCB	Printed Circuit Board
PID	Proporcional, Integral, Derivativo
PoP	Package on Package
PWM	Pulse Width Modulation
RAM	Random Access Memory
SoC	System on Chip

ToF	Time of Flight
USB	Universal Serial Bus

Sumário

1 Introdução

1.1 Justificativa

Um *digital twin* (do inglês, gêmeo digital) é definido como a réplica virtual de um produto, sistema ou ser vivo que são continuamente atualizados com dados de sua contraparte física, bem como do ambiente no qual está inserido. Ele é o elo entre o mundo virtual e as entidades físicas e, por isso é considerado um dos pilares da Indústria 4.0 e, segundo Jiang et al. (2021), é a força motriz para futuras inovações.

Embora o uso desse tipo de tecnologia já venha sendo difundido, ele ainda é limitado à empresas de grande porte. Nessas empresas, o gêmeo virtual dos sistemas físicos é utilizada para fazer predições sobre condições adversas de funcionamento, testes de colisão e de aerodinâmica, para citar alguns exemplos de aplicações. Além disso, essa tecnologia também vem sendo implementada por companhias cinematográficas e de efeitos visuais com auxílio de câmeras e sensores do tipo LiDAR ToF, que possibilitam a reconstrução de forma quase instantânea e com alta precisão do ambiente e objetos.

O presente projeto propõe desenvolver um método de escaneamento 3D utilizando um sensor ToF do tipo LIDAR e, com isso, promover o avanço na tecnologia de escâneres 3D. Atualmente, estes sensores vem se mostrado bons ao detectar objetos com alta fidelidade e de forma robusta, permitido que veículos autônomos possam navegar (LI; IBANEZ-GUZMAN, 2020), auxiliando na caracterização de plantações (CANATA; MOLIN; SOUSA, 2019) e, mais atualmente, no emprego de escâneres.

O objeto para escaneamento escolhido foi o corpo humano para que dessa forma, cada pessoa tenha suas medidas tiradas com precisão, auxiliando assim na escolha de tamanhos em compras on-line e na fabricação de roupas, por exemplo. Tendo em vista o potencial de alta fidelidade e a robustez nas medições, podemos obter cópias digitais de boa resolução a um baixo custo. Assim, de forma pragmática, também há um grande potencial desse tipo de tecnologia nas áreas de engenharia de produto, prototipagem rápida e na confecção de próteses médicas de áreas grandes ou pequenas.

1.2 Objetivos

O objetivo do projeto é desenvolver um método de aquisição baseado no escanamento com o sensor LiDAR para obter um modelo 3D com medidas fiéis de uma pessoa. A motivação principal deste objetivo sendo de que esse método possa ser utilizado em um provador de roupas virtual, em que uma pessoa possa ter um modelo do próprio corpo e com ele poder experimentar

roupas virtualmente. O presente trabalho foi pensado para auxiliar no desenvolvimento desta tecnologia, para com o passar dos anos e mais esforços como o do grupo, o escaneamento de pessoas possa se tornar mais acessível e confiável.

Dos objetivos específicos, destacam-se três: o desenvolvimento do controle da plataforma, a aquisição dos dados e reconstrução da malha a partir das nuvens de pontos obtidas pelo sensor. Em relação à plataforma giratória, desejamos que seu movimento seja suave o suficiente e contínuo para que o usuário não fique incomodado. Na aquisição de dados, desejamos que o LIDAR e a plataforma tenham os dados integrados de maneira robusta e precisa. Por fim, findada a aquisição, buscamos reconstruir a nuvem de pontos compilada na forma de uma malha tridimensional suave e com detalhes suficientes para a obtenção de medidas corporais.

1.3 Contextualização

O uso do escaneamento para tirar medidas do corpo de pessoas tem se tornado cada vez mais popular, seja para usos médicos e até mesmo estéticos. Com essas medidas, pode-se calcular o índice corporal de gordura, fazer levantamento de medidas morfológicas ou, até mesmo, para se ter medidas mais acuradas na hora de comprar vestimentas.

Ao se comprar roupas, uma grande dificuldade é a incerteza se as medições apresentadas são compatíveis com a da peça presentada, e com o crescimento da modalidade de compra online, tal problema vem afetando cada vez mais pessoas. Esse problema é uma consequência da forma como a industria de vestimentas funciona, o modelo de fabricação é baseado em um formato de corpo como modelo e tendo-se variações de tamanho respeitando a mesma proporção, o que gera roupas que não correspondem ao biotipo da maioria das pessoas.

De acordo com Madureira (2021), estão sendo incluídos outros dois biotipos na norma que rege tamanhos de roupas femininas, devido a grande problemática gerada por usar padrões de corpo que não correspondem a realidade demográfica brasileira. De acordo com a matéria também se pode confirmar o incentivo de empresas no ramo de escaneamento 3D para se garantir um melhor aferimento dessas medidas.

A escolha do escâner tipo ToF LiDAR foi motivada pela sua popularidade e famigerada robustez na detecção de objetos. Sendo esta a principal característica pela qual esse tipo de dispositivo encontra-se em satélites, carros autônomos e drones. Além disso, recentemente, o mesmo encontra-se também presente *smartphones* de alto padrão sendo utilizado em recursos de realidade aumentada e melhor autofoco nas câmeras.

2 Revisão Bibliográfica

2.1 Trabalhos Relacionados

Lançamos mão de alguns trabalhos como o de Franca et al. (2005) para compor a fundamentação teórica do nosso trabalho. Nele, são apresentados métodos mais efetivos de se usar a triangulação com lasers no escaneamento 3D, afim de se obter imagens de alta resolução, em diferentes ângulos, profundidade e cores. Ainda, de acordo com os pesquisadores, pode-se usar a triangulação integrada a um *encoder*, um laser e uma câmera.

2.2 Fundamentação Teórica

2.2.1 Sensores ToF

Um sensor ToF (*Time of Flight*, do ingles, Tempo de Vôo) funciona calculando o tempo gasto por um laser emitido pelo sensor chegar a superfície e voltar para o mesmo. Esse tipo de sensor faz uso de raios de infravermelho o que traz os benefícios de poder operar em ambientes com pouca luz e sem a interferência da luz visível (FREEDMAN et al., 2014).

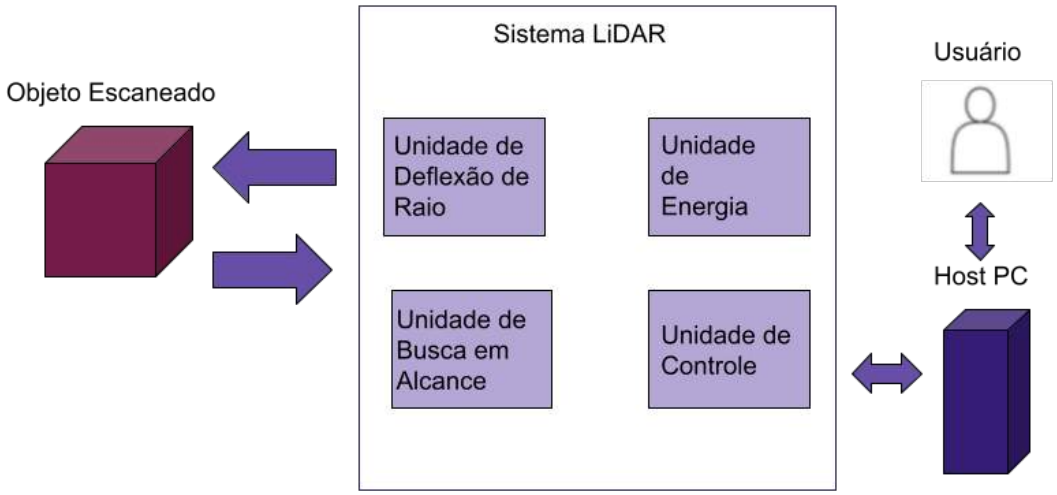
2.2.2 LiDAR

Um dispositivo LiDAR é composto por vários sub-sistemas que variam de instrumento para instrumento, mas, como base, todos tem uma unidade de busca de alcance, uma unidade de controle, uma unidade de energia e uma unidade de deflexão de raio, sendo essa um diferencial que não está presente nos leitores de apenas 1D. A arquitetura básica de um LiDAR pode ser vista na Fig 2.1.

A unidade de busca em alcance é a chave para um sistema LiDAR, que contém os componentes responsáveis por gerar, transmitir e receber os pulsos de laser. Esses componentes são normalmente um diodo laser, um fotodiodo e um amplificador de trans-impedância (TIA). o unidade de controle é responsável por realizar um processamento do sinal, controle dos sinais gerados e estabelecer comunicação com um computador. A unidade de energia tem como função suprir a necessidade energética do LiDAR e o design dessa unidade varia de acordo com a tensão e corrente necessárias. Por fim, a unidade de deflexão de raio é responsável pela aquisição de informação espacial. Ela é a principal diferença entre os escâner unidimensionais, bidimensionais ou tridimensionais de acordo com a quantidade de dados que pode captar (RAJ et al., 2020).

Em função da característica ativa desses sensores, existem algumas especificações que devem ser seguidas. Podemos elencar, por exemplo, a distância dado que existe um intervalo

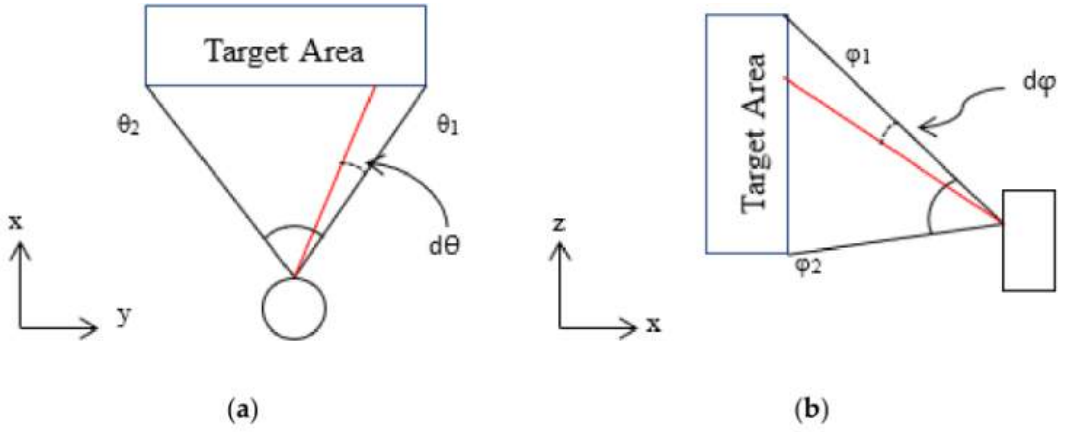
Figura 2.1 – Arquitetura LiDAR



Fonte: Adaptado de Raj et al. (2020)

mínimo para que a leitura tenha uma boa resolução. Considerando isso, pode-se chegar em uma das principais variáveis consideradas nesse trabalho que é área capturada pelo LiDAR, ou FOV (*field of view*, do inglês, campo de visão). Como podemos ver na Fig. 2.2, o ângulo máximo de captação do sensor define a área útil e o cone de captação discutidos na seção 3

Figura 2.2 – FOV - Field of View, a) visão frontal; b)visão lateral



Fonte: Raj et al. (2020)

2.2.3 Escaneamento 3D

O escaneamento 3D via laser é uma tecnologia que permite a obtenção de nuvens de pontos a partir da varredura das superfícies dos objetos por feixes de raios laser, permitindo a captura automática de uma grande quantidade de dados em um curto período de tempo (GROETELAARS; AMORÍN, 2011).

Existem métodos e técnicas usados para tornar o escaneamento possível, sendo esses métodos o escaneamento ativo e passivo. No escaneamento passivo, o sensor não emite iluminação, ao invés disso se utiliza da luz ambiente. Escâneres ópticos que não exigem contato podem ser agrupados de acordo com a iluminação controlada necessária para o seu funcionamento (TAUBIN; MORENO; LANMAN, 2014).

Um exemplo de escâner passivo muito utilizado é o de imagens estetoscópicas. Ele estima a posição de uma cena em 3 dimensões usando triangulação. Nesse método, as câmeras identificam a projeção 2D dos pontos usando objetos de calibração conhecidos, com o posicionamento calibrado de forma a ajudar, a geração de imagens de cada câmera é estimada, utilizando uma única linha em 3 dimensões "desenhada" entre o centro de cada projeção das câmeras por meio desse ponto. Com o encontro dessas linhas se pode encontrar a profundidade dos pontos.

Um grande impedimento deste método é a obrigatoriedade de uma referência, ou um objeto correspondente, por isso existem diversas alternativas podem ser usadas para tentar contornar essa situação, sendo inclusive usados diversos algoritmos, em muitos casos sendo necessário algorítimos de visão computacional mais robustos, e em situações controladas, como quando se tem o ambiente conhecido, e objetos identificados.

Como uma solução para o problema de correspondência vinda dos sensores passivos, existem os escâneres ativos. Em comparação com o método visto anteriormente, neste método se tem uma fonte de iluminação ativa, sendo esta muito mais sensível para reconhecer a superfície dos objetos escaneados. Mesmo assim, ainda existem objetos muito refletivos ou translúcidos que necessitam de medidas adicionais (TAUBIN; MORENO; LANMAN, 2014).

Para se realizar o escaneamento ativo pode-se usar uma fonte de luz como um laser em movimento, mas tal técnica é dispensável com o uso de um projetor de luz estruturada, que consegue direcionar a luz em toda a superfície. Com o avanço desses métodos é possível a realização do escaneamento com sensores de luz estruturada como a presente no trabalho.

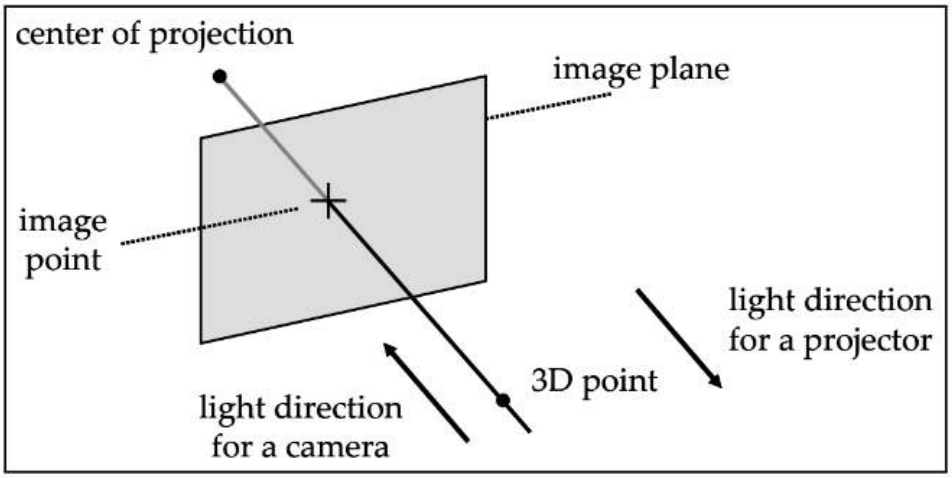
2.2.4 Triangulação

A triangulação para escaneamento representa o alinhamento de câmeras e fontes de luz de forma a se ter uma visão em 3 dimensões do objeto escaneado. Sendo que a triangulação pode ser feita com câmeras, ou como no caso do presente trabalho, com laser. O objetivo é obter as coordenadas tridimensionais do objeto.

Existem diversos métodos de triangulação. Um meio bem conhecido e simples, é o "Pinhole Model". Esse modelo é composto por um plano e um ponto externo a ele. Este plano é referenciado como sendo o plano da imagem, e o ponto como centro da projeção, como é possível ver na Fig. 2.3.

Em uma câmera, todos os pontos em 3 dimensões, exceto o centro de projeção, criam uma linha única que passa pelo centro de projeção, e que caso não seja paralela ao plano da

Figura 2.3 – Pinhole Model



Fonte: Taubin, Moreno e Lanman (2014)

imagem, deve interceptar o plano em um único ponto. Para resumir a luz percorre o caminho indo de um projetor até uma câmera, ao longo de uma uma linha que conecta o ponto em três dimensões com a perspectiva em 2D projetada no centro do plano da imagem (TAUBIN; MORENO; LANMAN, 2014). Um exemplo de triangulação pode ser visto no trabalho de Franca et al. (2005) no qual se faz uso de um encoder, uma câmera e um laser para se ter três pontos do objeto escaneado.

2.2.5 Nuvem de pontos

O modelo geométrico da nuvem de pontos é a representação mais básica obtida diretamente pelo escâner 3D. Cada ponto que forma a nuvem de pontos é representado por suas coordenadas cartesianas (x, y, z) e um ou mais atributos associados ao mesmo (GROETELAARS; AMORÍN, 2011).

O registro dos dados consiste em posicionar várias cenas, ou imagens coletadas pelo sensor, em único arquivo com o mesmo referencial. E então, com a associação das n imagens, com área de sobreposição acima de 30%, realizar operações para otimizar as nuvens. As operações consistem em fazer filtragem dos pontos relevantes e re-amostragem, eliminando informações redundantes.

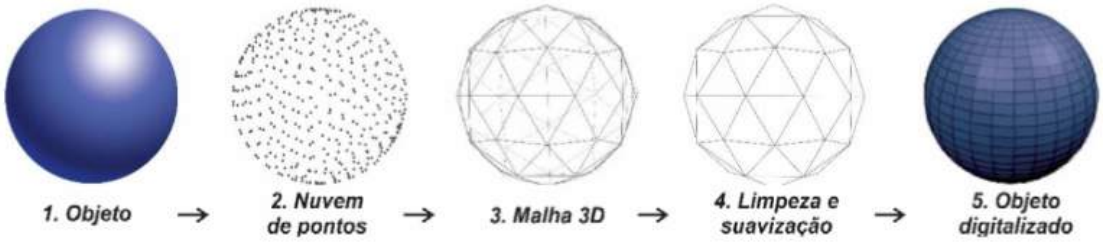
2.2.6 Malha

Ao unirmos os pontos de uma nuvem de maneira coesa e coerente, obtemos um objeto que pode ser manipulada de acordo com o que se pretende analisar. No geral, definimos uma malha como uma matriz de conexões que compõe um grafo. Neste caso, as coordenadas desses

nós são obtidas através do escaneamento, e as conexões através de algoritmos de reconstrução que estimam as conexões entre os nós.

O processo de obtenção de um objeto tridimensional consiste em duas etapas: no processamento da nuvem de pontos em uma rede de conexões, i.e., uma malha tridimensional, e no pós-processamento dessa malha bruta onde serão feitas suavizações, preenchimento de buracos, e eventuais decimações ou sobreamostragens. O processo de confecção da malha 3D pode ser visto na Fig. 2.4.

Figura 2.4 – Processo de obtenção de malha 3D



Fonte: (SIERRA; OKIMOTO, 2021)

A qualidade final do escaneamento é influenciada por diversos fatores, indo desde a qualidade do sensor usado, da precisão dele e tecnologia usada, além de fatores de hardware, cenário usado e protocolo de coleta (SIERRA; OKIMOTO, 2021). Por esses e outros fatores, a malha escaneada sempre irá apresentar algum tipo de erro, sendo pequeno ou não. Por isso exitem diversas técnicas para esse processo de limpeza.

2.2.7 Reconstrução da Malha

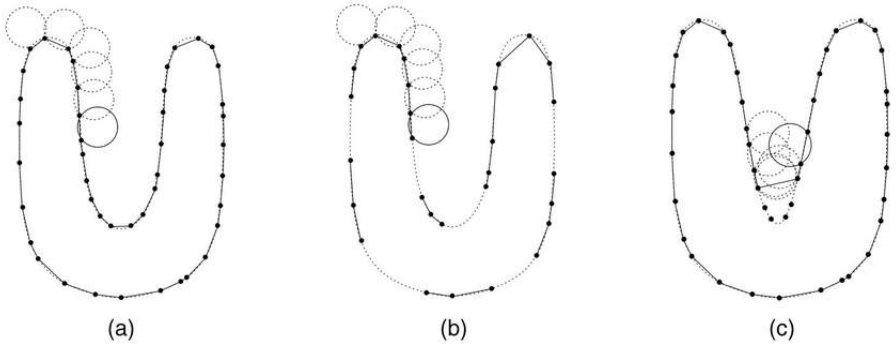
2.2.7.1 Ball Pivoting

Ball Pivoting é um algorítimo que reconstrói a superfície de uma malha triangular a partir de um conjunto de pontos dispersos em 3 dimensões. Para entender como funciona o algoritmo devemos pensar em uma amostra P que esta em uma superfície de um objeto 3D M. Ele reconstrói M a partir de P, desde que P seja suficientemente densa, com a condição de que a bola-r não pode passar por P sem tocar 3 pontos. O inicio do algorítimo se inicia usando essa bola em contato com três pontos, e a bola é rotacionada em torno de 2 desses três pontos, de forma que se fique em contato com dois pontos e se rotacione em torno do eixo que eles formam até que se toque outro ponto (DIGNE, 2014) .

2.2.7.2 Alpha Shapes

O algorítimo Alpha shapes é um subsequência da "triangulação de Delaunay" que tem como objetivo conectar pontos ao seus vizinhos mais próximos, mas o Alpha Shapes além disso

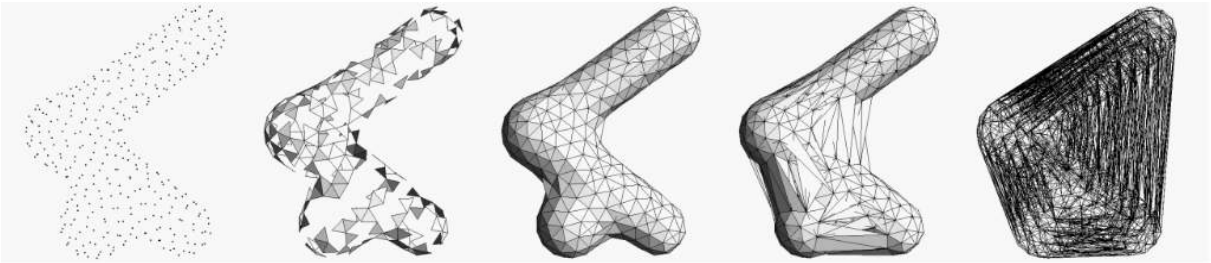
Figura 2.5 – Ball Pivoting



Fonte: Digne (2014)

cria triângulos para produzir uma tetraedrização completa do espaço. O método pode ser usado para reconstruir a superfície de um polígono usando como entrada um conjunto de amostras pontuais. Isso pois ele usa distancia entre os pontos para criar triângulos, conectando esses pontos (TEICHMANN; CAPPS, 1998). O uso do algorítimo pode ser visto na **??**.

Figura 2.6 – Alpha shapes



Fonte: Teichmann e Capps (1998)

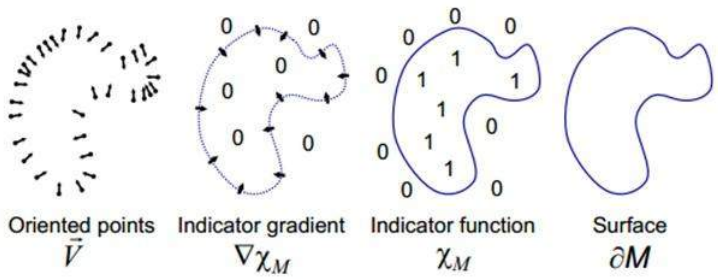
Algumas desvantagens de se usar este algorítimo são que existem algumas limitações, como quando se há intersecções, juntas, ou ainda quando dois objetos distintos estão próximos um do outro, sendo assim necessário utilizar alguma outra técnica unida a esse algorítimo.

2.2.7.3 Reconstrução de Poisson

Para a reconstrução de superfície de Poisson se usa um grupo de pontos orientados, de forma que se obtenha o gradiente, uma função e então a reconstrução da superfície. É um ótimo algorítimo a ser usado quando se tem os pontos vindos de uma única fonte, como é o caso do sensor usado.

No entanto, o problema com usar esse algorítimo é que existe um parâmetro chave que deve ser usado para a reconstrução, variando de uma superfície perfeita para uma muito suavizada. (KAZHDAN; BOLITHO; HOPPE, 2006)

Figura 2.7 – Reconstrução de superfície de Poisson



Fonte: Kazhdan, Bolitho e Hoppe (2006)

2.2.8 Motor DC

Os motores que funcionam a corrente contínua, ou DC, são os mais comuns quando se quer transformar energia elétrica em energia mecânica. Um motor DC é constituído por duas partes, o Rotor ou Armadura, que é uma parte móvel girante, montada sobre o eixo da máquina, feita de um material ferromagnético envolto em um enrolamento chamado de enrolamento de armadura.

A outra parte é fixa e, por sua vez, chamada de Campo ou Estator. Este é montado em volta do rotor, de forma que o mesmo possa girar internamente. Também constituído de material ferromagnético envolto em um enrolamento chamado de enrolamento de campo, que tem a função de produzir um campo magnético fixo para interagir com o campo da armadura. As principais características do motor são a sua tensão nominal, corrente nominal, potência, binário e velocidade. Velocidade essa que varia de forma proporcional com a tensão de alimentação (MATOS, 2008).

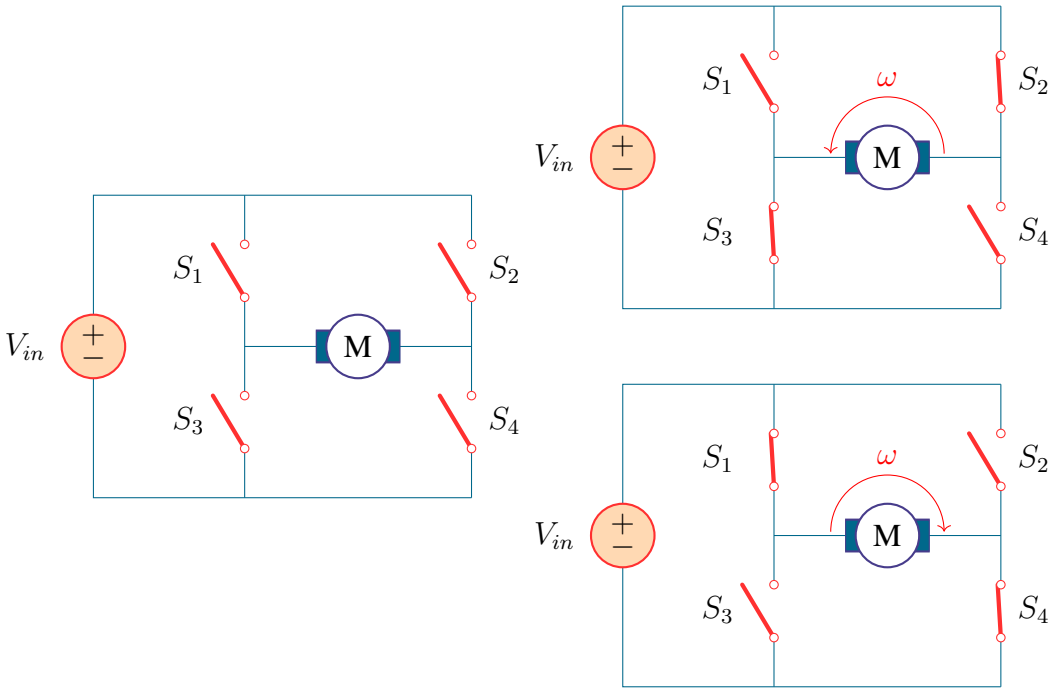
2.2.9 Ponte H

A ponte H é um circuito utilizado tanto para controlar o sentido de rotação de um motor DC, quanto a sua velocidade. Como a tensão e corrente solicitadas pelo motor são muito mais altas que as suportadas pelo microcontrolador nós o utilizamos somente para chavear digitalmente o acionamento do motor.

Na Fig 2.8 podemos ver o diagrama simplificado de acionamento da ponte H. Ao acionarmos os pares de chaves S_1 e S_4 permitimos que a corrente flua através do motor acionado-o em um sentido. De maneira análoga, ao acionarmos S_2 e S_3 o sentido da corrente se altera fazendo com que o sentido de rotação se inverta. As demais combinações de pares de chaves não acionarão o motor, uma vez que a corrente não fluirá pelo motor.

Para o controle da velocidade devemos variar a tensão aplicada no rotor do motor. Embora a simples sobretensão ou subtensão aumente ou diminua a velocidade, respectivamente, tensões diferentes da nominal levarão a um desgaste acelerado do componente até que ocorra

Figura 2.8 – Esquema simplificado de uma ponte H



Fonte: Autoria Própria

a falha. Dessa forma, uma solução é aplicar um PWM (*pulse with modulation*, do inglês, pulso com modulação) onde a tensão média fornecida ao motor irá variar alterando assim a sua velocidade.

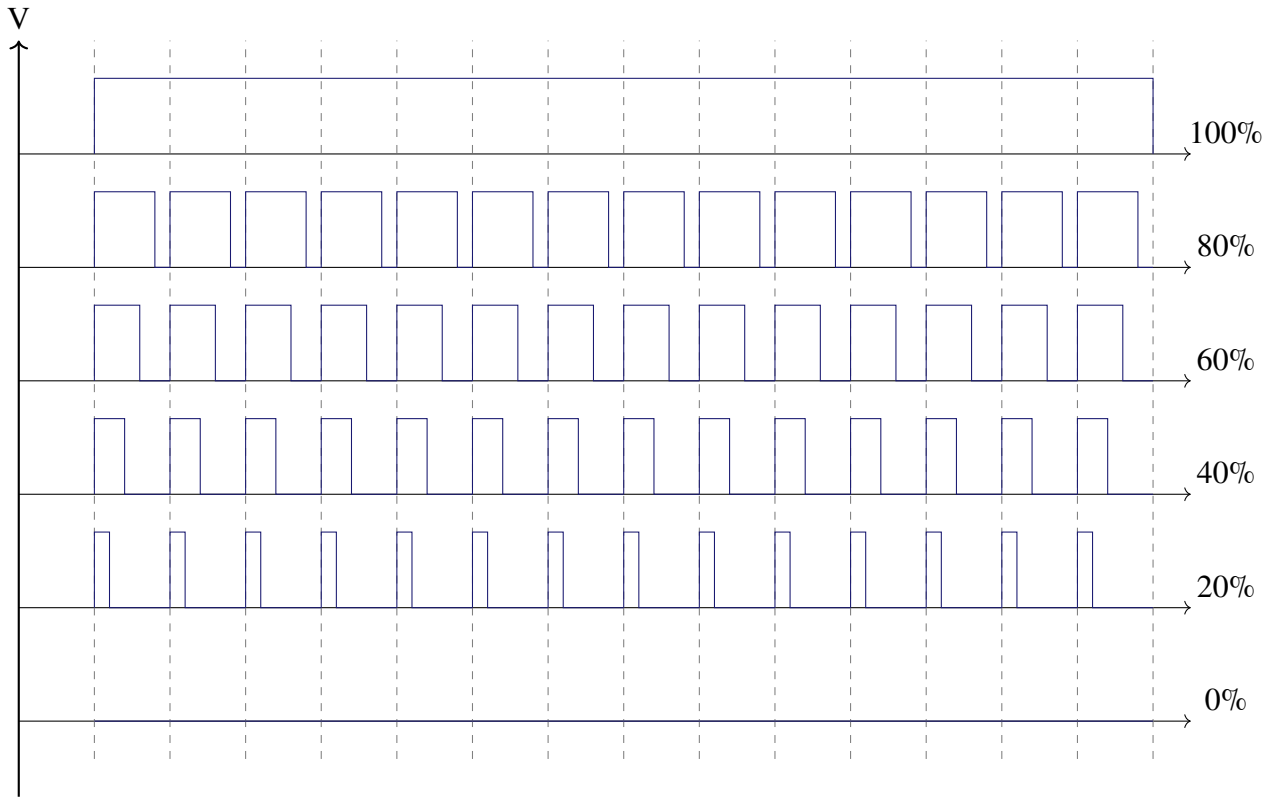
Um PWM é definido pelo seu *duty cycle* (ciclo de trabalho). *Duty cycles* maiores implicam em tensões médias mais próxima a nominal, enquanto duty cycles menores implicam em tensões médias próximas a zero. Como mostrado na Fig. 2.9, a gradação entre tensões médias dependerá, principalmente, do tempo levado em consideração. Contudo, a resolução temporal dependerá dos registradores temporais do microcontrolador sendo empregado ou do gerador de sinais.

Microcontroladores da família ATmega328, possuem dois registradores temporais de 8 bits e um de 16 bits. Dessa forma, a resolução da tensão de saída será a razão entre a tensão nominal e 255 ou 1023 a depender do registrado em questão.

2.2.10 Transdutores de Posição Angular

Um transdutor de posição angular, ou *encoder*, é um dispositivo eletrônico capaz de transformar a rotação de um corpo em informação elétrica. Esses transdutores podem ser divididos em incrementais, que detectam apenas a mudança de posição, e absolutos, que indicam a posição real do objeto (OLIVEIRA et al., 2017). No projeto, foi utilizado o *encoder* absoluto AS5040.

Figura 2.9 – Tipos de PWM com duty cycle associado



Fonte: Autoria Própria

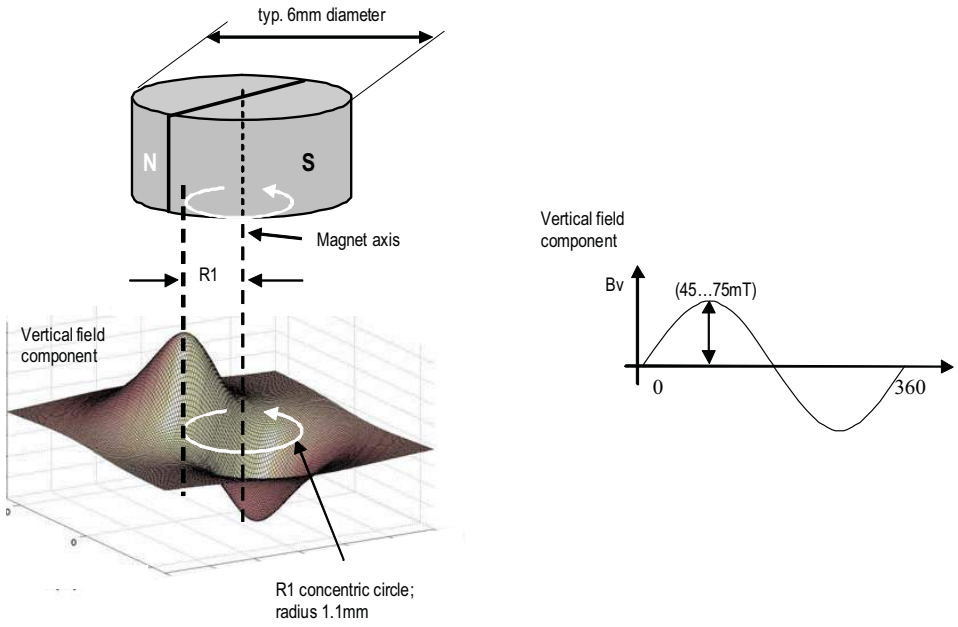
O AS5040 é um *encoder* magnético rotativo que mede, de maneira acurada, ângulos dentro de uma faixa de $360°$. Ele é um sistema embarcado com elementos Hall, front-end analógico e processamento de sinal digital. Além disso, sua resolução é de 0,35ř totalizando 1024 posições por revolução.

Para medir a posição angular de um corpo fixamos um dipolo permanente sobre ou sob o circuito integrado (CI) de forma centralizada. Como neste CI os elementos Hall estão distribuídos ao redor do centro do chip, ao girarmos o ímã os mesmos fornecem uma representação elétrica do campo magnético na superfície do dispositivo. Com essa representação, o AS5040 infere o deslocamento e a posição angular atual. A Fig. 2.10 mostra, de maneira esquemática, o funcionamento do *encoder*.

2.2.11 Placa de Desenvolvimento e Microncontrolador

As placas de desenvolvimento são dispositivos que consistem em diversos itens. Geralmente são placas de circuito impresso (PCB), que contêm um SoC (System-on-Chip), em que por conseguinte contém um módulo PoP (Package-on-Package). Neste módulo em questão, existe uma memória RAM, a qual acaba não podendo ser alterada justamente por estar introduzida no módulo PoP.

Figura 2.10 – Diagrama funcional do *encoder* absoluto AS5040



Fonte: ams OSRAM Group (2022)

Costumeiramente possuem diversas entradas e saídas, seja em forma de pinos, ou por conexão USB. As placas de desenvolvimento geralmente trazem também conexões serial, que podem vir através de pinagem própria, ou pela própria USB da placa (USB over Serial). Quase em sua totalidade, essas placas de desenvolvimento contam com pinos de GPIO (Entrada e Saída de Propósito Geral). Esses pinos são necessários para que haja interação microeletrônica, como a utilização de módulos, interface serial, ou sensores (MARGOLIS; JEPSON; WELDIN, 2020; SHILOH; BANZI, 2022).

Atualmente existem diversas placas de desenvolvimento variando desde código aberto até as patenteadas. Para o propósito do projeto escolheu-se uma placa Arduino Nano (ARDUINO, 2022) uma vez que ela possui um microcontrolador ATMega328 SMD (ATMEL, 2016) integrado, e um tamanho relativamente pequeno quando comparado as demais placas da família Arduíno.

As vantagens atreladas a se trabalhar com uma placa de desenvolvimento estão principalmente na facilidade de prototipagem rápida a um baixo custo. O Arduíno Nano, bem como as demais Arduíneos, conta com um ambiente de desenvolvimento integrado (IDE) de código aberto e com uma linguagem própria baseada em C e C++ (ROBÓTICA, 2012).

2.2.12 Controle

Ao lidar com máquinas ou sistemas de máquinas é possível se utilizar de dispositivos ou técnicas para gerenciar o comportamento esperado. Para isso se precisa primeiramente fazer a modelagem matemática do sistema, ou planta. No caso deste projeto foi usado o modo de

controle PID.

Controle PID é uma forma de controle de um sistema, que significa controle Proporcional-Integrativo-Derivativo. Ele é composto por três coeficientes: Kp (ganho proporcional), Ki (ganho integral), Kd (ganho deriviativo).Os controle PID podem ser empregados em sistemas de malha fechada, ou seja, sistemas com retroalimentação que são sistemas onde a partir de uma comparação entre o estado esperado na saída com o seu estado real, o controlador recebe os dados e atua de forma a fazer com que a saída seja alterada para o seu valor esperado, no melhor caso, ou o mais próximo possível (FILHO; ARAÚJO,).

O controle PID pode ser escrito na forma da equação abaixo.

$$PID = K_p\left(1 + \frac{1}{T_i S} + T_d S\right) = K_p + \frac{K_i}{S} + K_d S \tag{2.1}$$

O termo proporcional produz um resultado proporcional ao erro estacionário, sendo que esse pode ser ajustado ao se multiplicar o erro pela constante Kp. Quanto maior o ganho do termo proporcional maior a mudança no resultado pelo erro. Sendo que se o mesmo é grande demais o sistema fica muito instável, mas também quando ele fica muito baixo o controlador fica menos responsivo, respondendo de forma desproporcional a perturbações no sistema. (ANG; CHONG; LI, 2005)

O termo integral influência tanto a magnitude quanto o tempo de duração do erro. Esse termo é o resultado da soma do erro instantâneo ao longo do tempo, por conta disso ele faz com que o valor presente sofra um sobressinal, i.e., um sinal com valor excedente ao esperado.

O termo derivativo é achado determinando a inclinação do erro ao longo do tempo, e multiplicando pelo termo de ganho derivativo. Com este termo se tem uma previsão do comportamento do sistema, sendo possível melhorar o tempo de acomodação a estabilidade do sistema (CRENGANIS; BOLOGA,). De forma resumida, o efeito das constatantes no sistema pode ser analisado de acordo com a tabela 2.1.

Tabela 2.1 – Efeito dos Parâmetros em um sistema

	Tempo de subida	Overshoot	Tempo de acomodação	Erro	Estabilidade
Kp	Diminui	Aumenta	Indiferente	Diminui	Piora
Ki	Diminui	Aumenta	Aumenta	Elimina	Piora
Kd	Indiferente	Diminui	Diminui	Sem efeito	Melhora se Kd é pequeno

Fonte: Autoria Própria

E para a aplicação desse controle, pode se utilizar diversos métodos, inclusive o método manual. O qual se executa deixando tanto Ki quando Kd sendo 0, e então se aumenta Kp até que a curva oscile. Com isso se diminui o valor de Kp pela metade, para se ter uma resposta na qual a amplitude da curva cai 25% do tamanho. Com isso se aumenta Ki até que a resposta seja

corrigida, mas não se aumenta demais para não causar instabilidade. Por fim se aumenta Kd de forma que o sistema fique com o ganho e estabilidade esperados (CRENGANIS; BOLOGA,).

2.2.13 Filtros Digitais

Ao colhermos dados provenientes de sensores, muitas vezes temos ruídos e artefatos no sinal de interesse. Esses ruídos podem ser provenientes de harmônicas do próprio sinal, interferências mecânicas, interferência de campos ou até mesmo derivas causadas por processos de integração (SALMONY, 2021a; SALMONY, 2021b; SALMONY, 2021c). Logo, para se obter um sinal limpo, faz-se necessária a remoção desses sinais espúrios, i.e., precisamos filtrar esses sinais.

Por vezes, para mitigar os efeitos desses ruídos, acoplamos circuitos com resistores, capacitores e indutores aos sensores. Embora ainda hajam casos que essa aplicação física seja necessária, hoje também há a possibilidade de implementar esses circuitos de forma digital, i.e., um filtro digital. Na área de processamento de sinais, um filtro digital é um sistema que realiza operações matemáticas num sinal discretizado no tempo para reduzir ou reforçar aspectos do sinal.

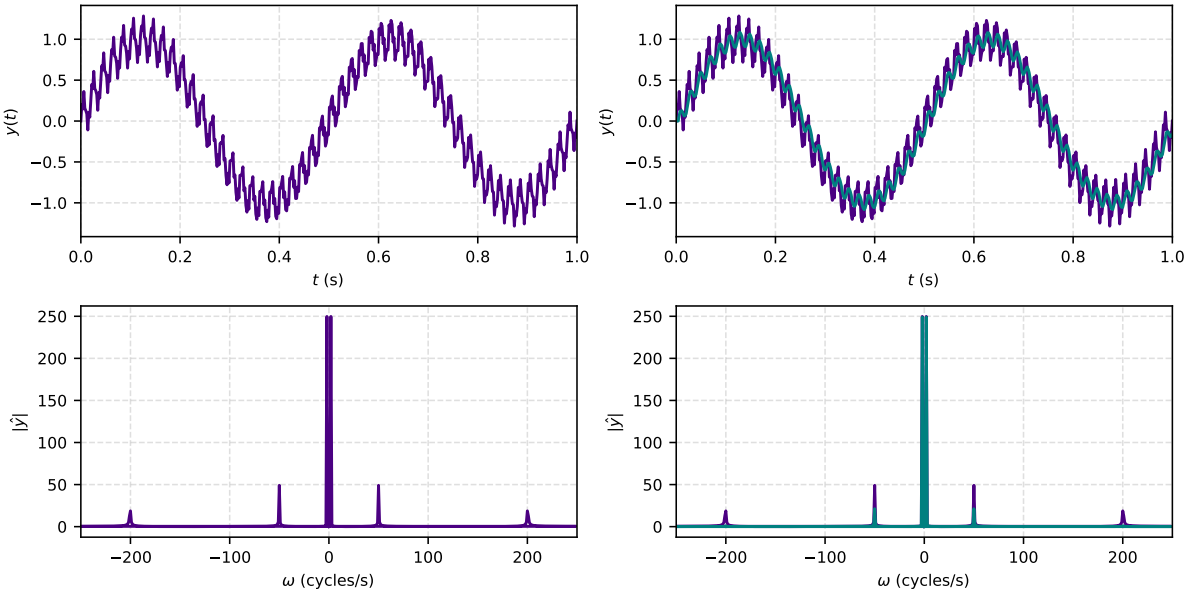
Para que o filtro seja implementado, devemos definir uma função de transferência bem como a faixa de funcionamento. No geral, utilizamos as funções de transferências dos filtro analógicos contínuos e em seguida discretizamos a função de transferência contínua em função da nossa taxa de amostragem. Assim como para filtros analógicos, o filtro digital pode operar sobre diversas faixas de frequência recebendo as denominações passa-baixa, passa-altas, passa-faixa e rejeita-faixa.

O filtro passa-baixa apresenta um limite superior de acordo com a frequência desejada (frequência de corte), e limite inferior nulo. Neste caso, frequências inferiores a frequência de corte serão conservadas, enquanto as superiores serão atenuadas. O filtro passa-alta tem o efeito oposto ao passa-baixa, permitindo que frequências maiores que a de corte sejam mantidas enquanto as inferiores sejam anuladas. O filtro passa-faixa, por sua vez, possui tanto um limite inferior quanto um superior, permitindo que somente uma faixa de frequências seja mantida. Por outro lado, o filtro rejeita-faixa tem limite inferior e superior determinados para conservar qualquer faixa de frequência que não seja a que se deseja rejeitar (MARIN, 2014).

A Fig. 2.11 mostra o funcionamento de um filtro passa baixa. Podemos ver, na coluna à esquerda, que o sinal no domínio do tempo possui uma componente principal de baixa frequência e outras componentes de maior frequência. Ao fazermos a transformada de Fourier, constatamos uma componente próxima a zero, uma em 50 rad/s e outra em 200 rad/s. Como neste caso desejava-se eliminar as componentes de mais alta frequência, definiu-se uma frequência de corte em 25 Hz. Na coluna à direita vemos o resultado da aplicação do filtro digital. Inicialmente podemos ver um pequeno atraso do sinal filtrado em relação ao sinal original e, além disso,

pode-se constatar uma oscilação de alta frequência bem mais atenuada. Ao vermos o sinal pela ótica da transformada de Fourier constatamos uma eliminação completa da componente em 200 rad/s e uma forte atenuação em 50 rad/s.

Figura 2.11 – Efeito de um filtro passa-baixa



Fonte: Autoria Própria

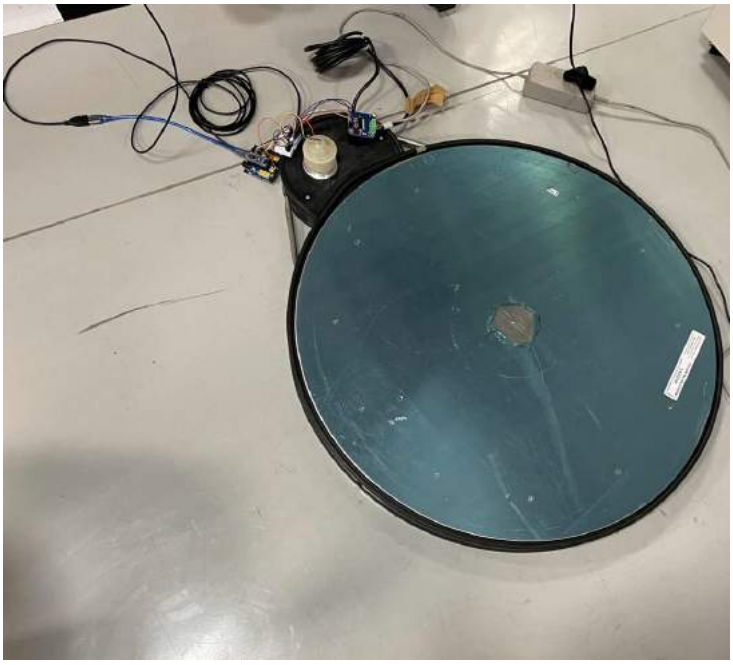
Podemos elencar como benefícios da implementação de filtros digitais a possibilidade de se implementar, alterar e ajustar esses filtro de maneira mais fácil dado que no geral serão ajustes de software. Além disso, podemos implementar filtros de maior ordem sem grande aumento da complexidade diferentemente dos dispositivos eletrônicos que aumentaram proporcionalmente em tamanho e implicarão numa maior dissipação energética.

3 Métodos

3.1 Plataforma giratória

A plataforma giratória utilizada é formada por dois discos de alumínio espaçados por um compensado de madeira, fixados por parafusos do tipo Torx. Em contato com o compensado de madeira, há uma correia dentada que transfere a rotação do motor para a plataforma. Na Fig. 3.1, podemos ver a plataforma giratória e o motor que fica armazenado dentro do compartimento preto.

Figura 3.1 – Plataforma giratória utilizada na rotação dos indivíduos durante o escaneamento.



Fonte: Autoria Própria

Para realizar a alimentação do motor se utilizou de um fonte de corrente contínua com tensão nominal de 12 V e corrente nominal de 3 A. Em detrimento à necessidade de acionar o motor através do microcontrolador, ligou-se a fonte e o motor à ponte H. Por meio dessa interface de controle e potência foi possível acionar o motor, controlar a sua direção assim como sua velocidade.

Afim de aferir a rotação do motor, posicionamos o transdutor de posição absoluta sobre o ímã fixado no eixo do motor. Este transdutor foi então enclausurado em um compartimento impresso em 3D pelo qual o fluxo magnético era capaz de fluir livremente por meio de um orifício na parte inferior.

Por fim, o Arduíno Nano foi conectado ao computador por meio de um cabo USB. Neste caso, a porta USB fornece tanto a alimentação quanto permite estabelecer a comunicação serial bidirecional.

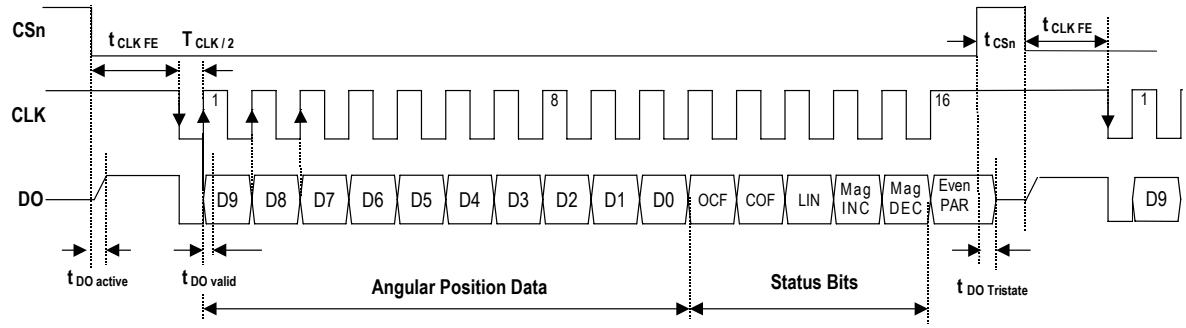
3.1.1 Aferição da Posição Angular

Uma vez que tínhamos todos os componentes em posição foi possível iniciar o desenvolvimento do software de controle da plataforma. Iniciamos pela codificação do *encoder* usando a amostragem da posição angular do motor que nos fornece a posição e a velocidade de rotação da base.

Ao verificar o *datasheet* do AS5040, foi decidido seguir com a implementação do protocolo de *synchronous serial interface* (do inglês, interface serial síncrona). Para essa configuração, precisamos conectar o *encoder* a dois pinos de alimentação e três pinos de sinal do microcontrolador. Dos pinos de sinal temos um pino para selecionar o chip (CSn), um pino para enviar o *clock* (CLK) e um último pino para fazer a leitura dos dados (DO).

Para que a aferição da posição angular fosse efetuada, devem ser seguidas algumas etapas apresentadas na Fig. 3.2. Essa figura mostra a configuração de cada um dos pinos no microcontrolador ao longo de uma aferição. Inicialmente temos CSn e CLK em nível lógico alto e dizemos que o DO está num estado tri-estado de alta impedância.

Figura 3.2 – Protocolo de aferição de posição angular.



Fonte: ams OSRAM Group (2022)

Para inciarmos o ciclo de aferição movemos o CSn para nível lógico baixo e aguardamos um período. Em seguida, iniciamos a sequência de pulsos modulados a um *duty cycle* de 50% a uma taxa de 1 MHz. Ao longo dos pulsos, na borda de subida, os dados são movidos para o pino DO que os lê. Como a leitura inicia-se pelo bit mais significativo fazemos um deslocamento do mesmo para a esquerda. Este processo repete-se dez vezes.

Ao fim do primeiro ciclo, inicia-se a leitura de outras informações complementares ao dado. Essas informações servem para dizer se o dado lido no primeiro ciclo é válido ou não.

Dessa forma, podemos eliminar a medição caso a mesma falhe ao atender os critérios dos bits complementares.

Por fim, cessamos o *clock* retornando-o ao nível lógico alto e, então, retornamos o CSn ao nível lógico alto também. Para iniciarmos uma nova aferição devemos repetir o mesmo processo, mas com um interstício de no mínimo 500 nanosegundos.

3.1.2 Processamento Paralelo

Como os microcontroladores ATMega328 possuem somente uma unidade de processamento lógico, eles não suportam a execução de tarefas em paralelo, ou *threads*. Todavia, é possível alternar a execução das tarefas de forma condicional e, assim, simularmos uma execução em paralelo, ou *protothreads* (DUNKELS et al., 2006).

Para que a execução condicional ocorra precisamos que o código opere de forma não blocante, i.e., sem que hajam funções que travem o fluxo normal de operação. Sendo assim, operações como os atrasos deveriam ser substituídos por condicionais que checam se um dado intervalo de tempo foi atingido. Dessa forma, enquanto o intervalo de tempo é inferior ao esperado, o microcontrolador pode desviar o fluxo normal do código e executar outras operações.

O método integrado com as interrupções temporais do microcontrolador em questão traz um controle temporal de alta precisão. Essa precisão é proveniente da capacidade de operarmos na faixa dos microsegundos, checando se há necessidade de executar ou não as funções programadas. Soma-se a isso o uso de programação orientada a objeto, e temos uma forma robusta de atualizarmos dados de interesse e os recuperarmos de maneira fácil e segura.

Tendo isso posto, utilizamos o processamento paralelo para a maioria das funções que foram implementadas. Neste caso, o *encoder* era a função com o menor intervalo entre atualizações a fim de se obter um dado confiável, seguido da atualização da velocidade do motor e o envio de dados via serial. Por fim, somente a função de recebimento de dados via serial ficou sem o uso do processamento paralelo uma vez que a mesma deveria ser executada sempre a fim de processar qualquer solicitação proveniente do computador.

3.1.3 Controle de Velocidade

Ao nos debruçarmos sobre objetivos do escaneamento 3D, recaímos em dois problemas: a quantidade de dados e o tempo de escaneamento. Ao rotacionarmos o objeto lentamente, podemos obter uma nuvem de pontos mais detalhada, uma vez que as pequenas variações poderão ser detectadas, contudo, o tempo de escaneamento será elevado. Por outro lado, ao rotacionarmos rapidamente o objeto, perderemos nuances de sua superfície obtendo um escaneamento mais simples, embora mais rápido.

Para obtermos uma taxa de variação constante e ajustável precisamos implementar um

controle da velocidade do motor. Dado que os corpos submetidos ao escaneamento possuem momentos de inércia diferentes, possíveis trepidações e eventuais oscilações do fornecimento de energia, faz-se necessário um sistema de controle por retroalimentação.

Por tratar-se de um sistema relativamente simples, recorremos ao uso do controle Proporcional, Integral e Derivativo (PID) afim de controlarmos o sistema motor-plataforma. Assim, desenvolvemos um sistema de controle PID digital. Neste sistema de controle, variamos a velocidade do motor em função da tensão de alimentação que pode ser ajustada pelo PWM(do inglês, *pulse with modulation*) aplicado à ponte H.

A velocidade angular da plataforma pode ser obtida com a relação de transmissão entre a plataforma e o motor. Dessa forma, a cada atualização da posição angular do motor era somado a um acumulador a posição total da plataforma e com isso, calculávamos a derivada primeira de maneira discreta. Mais uma vez utilizamos do processamento paralelo para atualizar o controlador e a atuação dele sobre o sistema.

Para realizar o ajuste dos parâmetros de controle K_p, K_i e K_d utilizamos um método heurístico implementado por bissecção. Começamos por ajustar o ganho proporcional e, enquanto monitorávamos o erro estacionário, aumentávamos o ganho integral. Por fim, incluímos um ganho derivativo para lidar com oscilações repentinas de velocidade da plataforma, e melhorar o tempo de resposta ao se alterar a velocidade via comunicação serial.

Embora as constantes finais tenham atendido aos nossos requisitos, ainda observava-se uma variação errática da velocidade em função de ruídos de leitura do transdutor de posição angular. Para reduzir os efeitos dos ruídos de alta frequência, implementamos um filtro passa baixas.

3.1.4 Filtro Passa Baixas

A ideia por trás dos filtros é remover uma parte do sinal de forma a diminuir o ruído na aferição da posição angular. Como o motor empregado não possui uma velocidade elevada e estamos trabalhando com a corrente abaixo da nominal, vamos colocar a frequência de corte em 25 Hz. Dessa forma, velocidades muito superiores serão atenuadas.

Filtros reais são modelados pela função de transferência dada na Eq. 3.1. Nesta equação, o parâmetro ω_0 é a frequência de corte do filtro.

$$H(s) = \frac{\omega_0}{s + \omega_0} \tag{3.1}$$

Dela seguimos fazendo a transformação bilinear onde

$$s = \frac{2}{\Delta t}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \text{ , para } \Delta t = \frac{1}{f_s} \tag{3.2}$$

Ao fazermos a transformação bilinear para uma taxa de amostragem de $f_s = 500\,\text{Hz}$ obtermos a função de transferência discreta desejada (ver Eq. 3.3).

$$H(z) = \frac{\omega_0}{\frac{2}{\Delta t}\frac{1-z^{-1}}{1+z^{-1}} + \omega_0} = \frac{\Delta t\omega_0(z+1)}{(\Delta t\omega_0 + 2)z + \Delta t\omega_0 - 2} = \frac{0.135z + 0.135}{z - 0.728} \tag{3.3}$$

Para que possamos utilizar essa função de transferência discreta, lançaremos mão de uma equação a diferenças. De forma geral, podemos escrever qualquer função de transferência discreta em termos da Eq.3.4

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{N} b_k x[n-k] \tag{3.4}$$

na qual, ao tomarmos a transformada z obtemos a seguinte função de transferência discreta

$$Y(z)\left(\sum_{k=0}^{N} a_k z^{-k}\right) = X(z)\left(\sum_{k=0}^{N} b_k z^{-k}\right) \tag{3.5}$$

$$\Rightarrow H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} \tag{3.6}$$

Portanto, ao unirmos os resultados obtidos na Eq. 3.3 com a Eq. 3.4 obtemos a equação do nosso filtro digital a ser implementado (ver Eq. 3.7).

$$y[n] = 0.135x[n] + 0.135x[n-1] + 0.728y[n-1] \tag{3.7}$$

3.1.5 Comunicação Serial

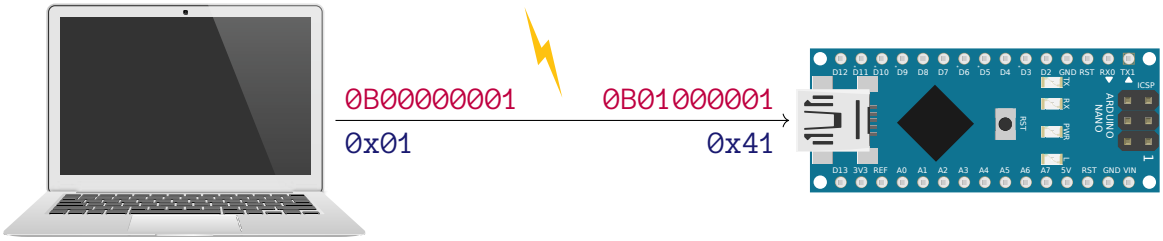
Para estabelecer uma comunicação entre o Arduino e o computador, utilizamos a porta serial desses dispositivos. A vantagem de se trabalhar com esse tipo de comunicação recai principalmente na facilidade de implementá-la, contudo ela não é muito robusta e nem confiável sendo propensa a erros e ruídos na transmissão dos dados.

No nosso caso, como estamos trabalhando com motores, há um campo eletromagnético próximos aos nosso cabos. Em função das correntes induzidas no fio, poderia haver distorções nos dados enviados. Dessa forma, poderíamos receber um dado alterado durante a comunicação. A Fig. 3.3 ilustra essa interferência no envio do dado. Nela, inicialmente enviamos o dado `0B00000001` e, por conta da interferência do ambiente, o dado é alterado chegando como `0B01000001`.

Dada uma situação hipotética onde deseja-se controlar a velocidade do motor, sairíamos de uma velocidade de 1 para 65. Ou seja, um aumento percentual de 25% ao sair do repouso.

Embora o nosso motor não possua altas velocidades, essa aceleração substancial poderia causar um certo desconforto.

Figura 3.3 – Comunicação serial e implementação do protocolo de comunicação



Fonte: Autoria Própria

Para mitigar esse tipo de problema, utilizamos um protocolo de comunicação. Protocolos de comunicação podem ser implementados de diversas formas, como por exemplo, utilizando expressões regulares em textos (AHO, 1990) ou empacotamentos dos dados. A primeira forma, embora altamente confiável, requer uma demanda computacional elevada, podendo causar travamentos no microcontrolador. A segunda alternativa, que embora mais simples, é igualmente confiável e requer menos recursos de memória e de processamento. Por conta desses fatores, lançamos mão do método de empacotamentos dos dados.

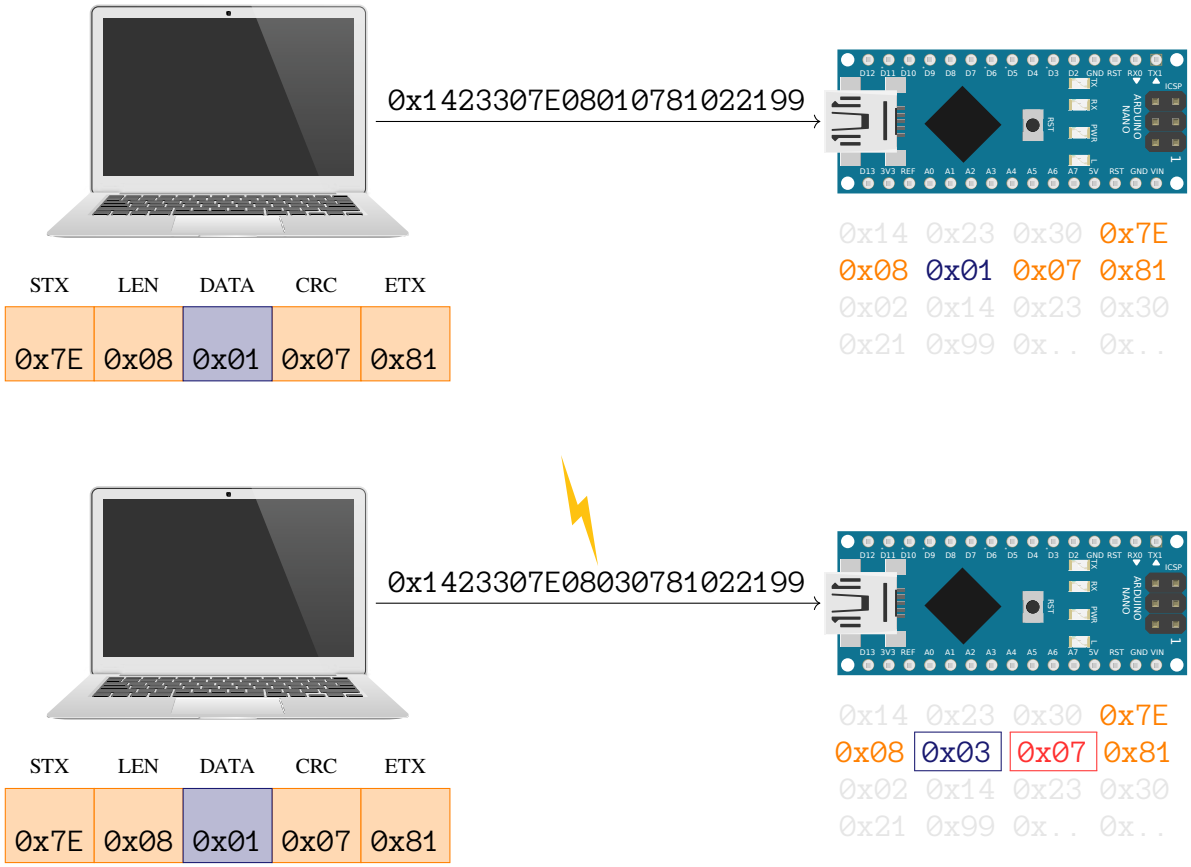
O empacotamento consiste na formação de uma mensagem utilizando marcadores conhecidos e uma arquitetura recorrente. De maneira geral, na formação da mensagem, definimos marcadores de início e fim, marcadores de tamanho, i.e., quantos bytes comporão o dado, e um marcador de validação, onde foi empregado uma verificação cíclica de redundância (SOBOLEWSKI, 2003). Na Fig. 3.4, podemos ver esse marcadores em laranja. Após a definição desses marcadores, podemos montar nossa mensagem como o byte inicial representando o marcador de início, seguido pelo tamanho do dado, o dado *per se*, o validador e, por fim, o marcador de fim.

Ao implementarmos essa estrutura em ambos os dispositivos, podemos deixá-los monitorando a via serial em busca dos dados. Quando o byte representando o marcador de início é lido, o dispositivo começa a armazenar os próximos dados. Ele também irá monitorar o tamanho dos bytes do dado a serem guardados, o dado e em seguida o validador, findando a mensagem no marcador de fim.

Se em algum desses momentos o número de bytes esperados for maior que o número de bytes recebido a mensagem é eliminada. O mesmo ocorre para o análogo oposto. Por fim, temos a situação onde o tamanho da mensagem enviada e a mensagem recebida são iguais. Neste caso usaremos o validador para checar se o dado foi corrompido ou não.

Os validadores funcionam como um dígito de verificação. Ao aplicarmos uma sequência de operações lógicas podemos, por exemplo: obter o validador ou tornar o resultado par. A

Figura 3.4 – Protocolo com envio de dado único



Fonte: Autoria Própria

construção dessas operações é feita de forma que somente a sequência de bits original resultaria numa tautologia com o validador. Na parte inferior da Fig. 3.4 exemplificamos um caso onde o dado foi corrompido.

3.2 Escaneamento

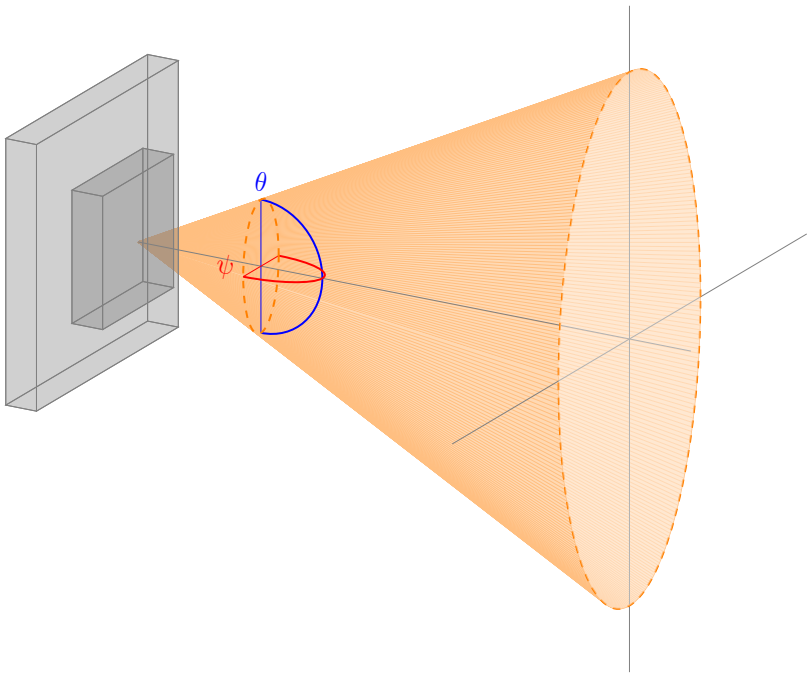
3.2.1 Posicionamento Adequado

O posicionamento tanto da plataforma, quanto do LiDAR deu-se de maneira iterativa. Ao longo das tentativas, buscávamos encontrar uma área livre atrás da plataforma, assim como um posição suficientemente próxima para conectarmos os cabos para a comunicação serial.

Embora dependente da posição da plataforma, o posicionamento do LiDAR precisava atender tanto as especificações técnicas, quanto as do nosso projeto. Segundo o fabricante, o Onion Tau LiDAR Camera possuía uma resolução de 160×60 pixels e um campo de visão de 81×30 graus, além de uma distância mínima de 10 cm e uma máxima de 4,5 metros. Como buscávamos escanear pessoa com até 2 metros tínhamos as variáveis de contorno definidas.

A Fig. 3.5 nos mostra um diagrama esquemático para o posicionamento inicial do LiDAR. Tendo em vista que a resolução maior estaria na horizontal, nós invertemos a orientação do LiDAR afim se obter mais pontos na vertical. Assim, o ângulo θ representa o campo de visão de 81 graus e o ângulo ψ o de 30 graus. Para prosseguir com os cálculo tomamos a distância entre o sensor e o objeto escaneado como d, o maior raio como h_1 e o menor como h_2.

Figura 3.5 – Campo de visão do LiDAR



Fonte: Autoria Própria

Tomando a altura desejada teríamos uma distância mínima dada por

$$\tan\theta = \frac{h_1}{d} \Leftrightarrow d = \frac{\frac{2}{2}}{\tan\left(\frac{81}{2}\right)} = 1,17 \text{ m} \tag{3.8}$$

Contudo, com essa distância, somente poderíamos medir distâncias entre mãos de

$$\tan\phi = \frac{h_2}{d} \Leftrightarrow h_2 = 1,17 \cdot \tan\left(\frac{30}{2}\right) = 0,313 \text{ cm} \tag{3.9}$$

portanto, 63 cm.

Em amostragem piloto, vimos que a distância entre mão ficava dentro de uma facha de 75 cm. Dessa forma, foi necessário aumentar a distância. Partindo dos 75 cm, obtivemos uma distância igual a 2,8 metros.

$$\tan\phi = \frac{h_2}{d} \Leftrightarrow d = 0,75 \cdot \tan\left(\frac{30}{2}\right) = 2,8 \text{ m} \tag{3.10}$$

$$\tan\theta = \frac{h_1}{d} \Leftrightarrow h_1 = 2,8 \cdot \tan\left(\frac{81}{2}\right) = 2,39 \text{ m} \tag{3.11}$$

Assim, para garantirmos a altura do LiDAR utilizamos um tripé com um suporte acoplado à sua extremidade (ver Fig 3.6). O tripé foi posicionado com uma altura fixa de 1,5 m de altura. Embora o LiDAR intercepte o chão nessa altura, garantimos que nossa zona de amostragem fique dentro do cone de visão e longe das extremidades assim como exemplificado na Fig. 3.5.

Figura 3.6 – Posicionamento inicial da plataforma e suporte do LiDAR e a clausura

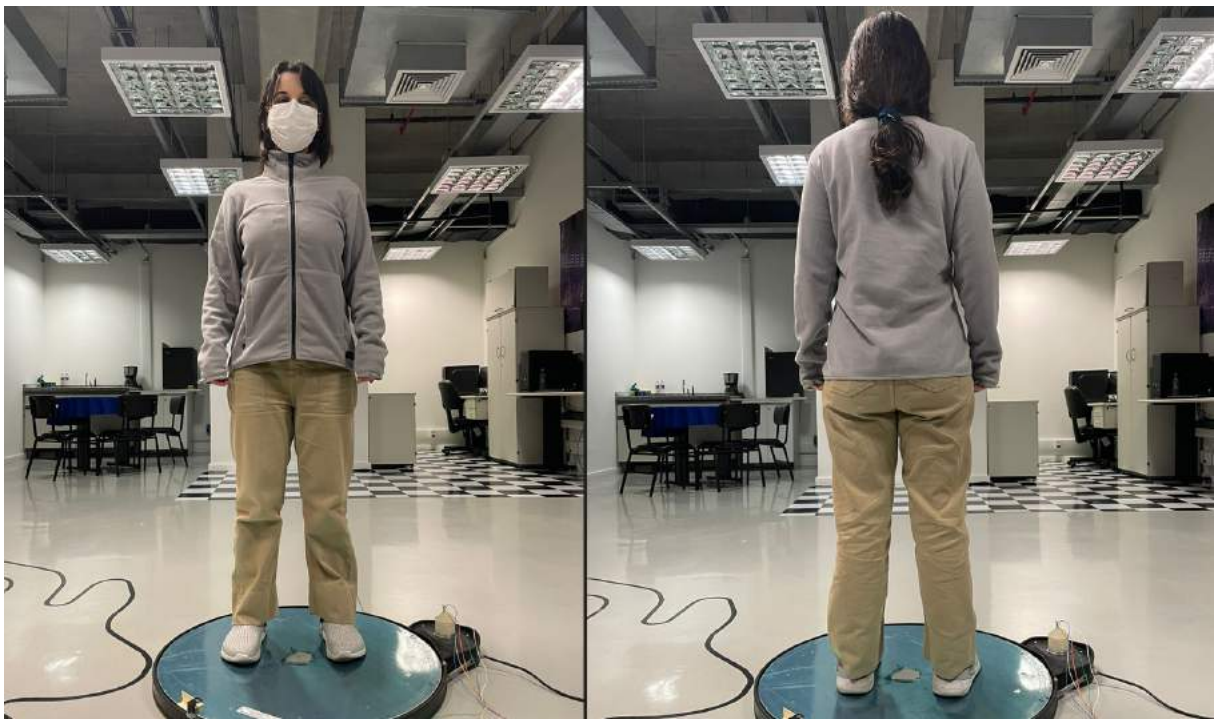
Fonte: Autoria Própria

3.2.2 Amostragem dos Dados

Diante à implementação das etapas prévias, começamos o processo amostragem das nuvens de ponto. Para que obtivéssemos uma nuvem de pontos representativa, deveríamos garantir algumas etapas, das quais podemos elencar: 1) garantir que não haviam objetos no fundo da imagem interferindo na coleta dos dados; 2) as vestimentas das pessoas, bem como a cor dos objetos deveriam ser claras; 3) caso estivéssemos escaneando uma pessoa, a pose padrão seria com os baços levemente afastados.

Para iniciar a amostragem dos dados, enviávamos um comando para o Arduíno através da via serial indicando a velocidade da plataforma, bem como a liberação do movimento. Em seguida, o Arduíno começava a enviar os dados da posição angular ao mesmo tempo que o LiDAR fazia a aquisição da nuvem de pontos. Todos os dados eram coletados e salvos num arquivo binário por meio de um *script Python*. Ao fim de duas voltas, era enviado um novo comando ao Arduíno para que o mesmo cessasse o movimento.

Figura 3.7 – Exemplificação do processo de escaneamento



Fonte: Autoria Própria

O processo de escaneamento pode ser visualizado na Fig. 3.7. Nesta figura, a pessoa escaneada foi girada duas vezes afim de se obter uma maior quantidade de dados finais. Vale notar que o cenário no fundo está distante do plano de interesse, e a pessoa utiliza vestimentas claras.

3.3 Processamento dos Dados

3.3.1 Remoção de Ruídos da Aquisição

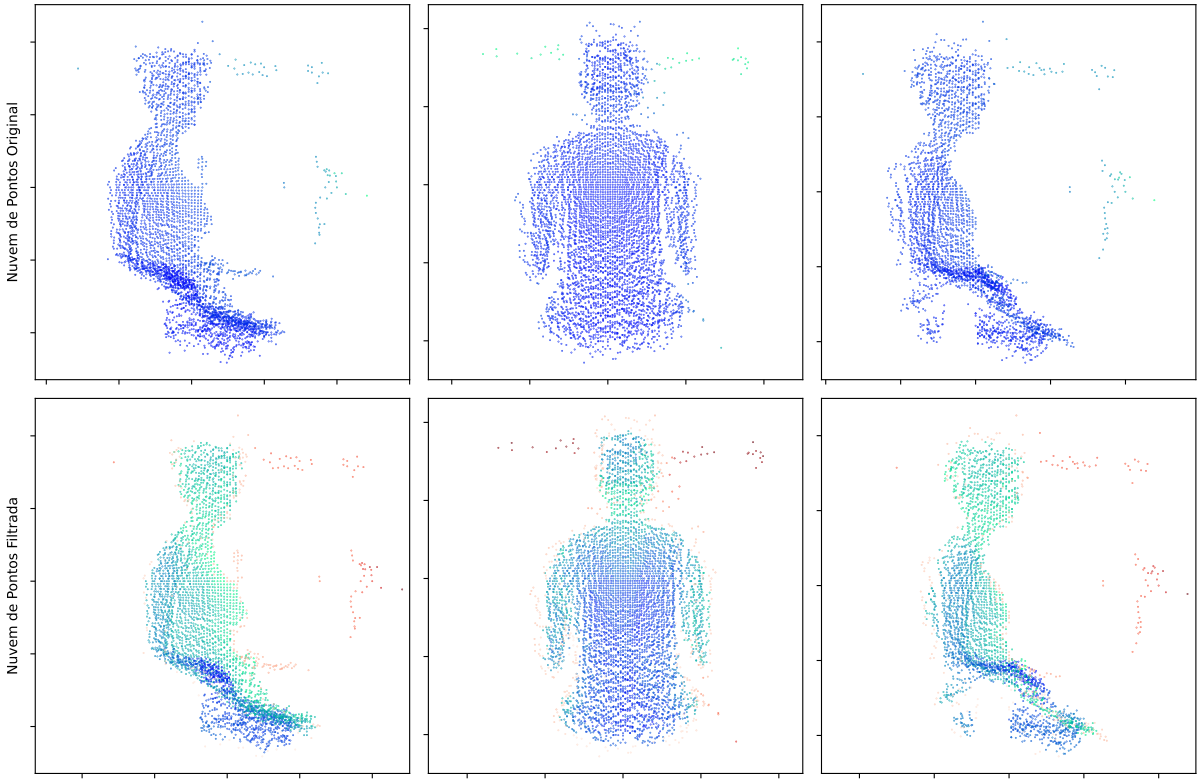
Durante o escaneamento, o LiDAR detecta alguns pontos que são ruídos de medida, ou *outliers*. Portanto, para realizar a união das malhas e a reconstrução da malha faz-se necessária remoção destes registros. Embora existam diversos métodos para a detecção de *outliers*, utilizamos um método simples que detecta o número de pontos vizinhos dentro de um certo raio. Caso o ponto não atenda aos critérios estabelecidos, nós o removemos.

Como o LiDAR permanecia fixo e o cenário de fundo idealmente também, definimos um critério que atendesse aos nossos critérios de outliers. Assim, após ajuste no número de vizinhos a serem considerados, bem como a distância, chegamos num filtro que julgamos ser suficientemente agressivo para a remoção de *outliers* até mesmo próximos ao corpo submetido ao escaneamento.

Na Fig. 3.8 podemos ver, na parte superior, as nuvens amostradas sem nenhum trata-

mento prévio. Com elas percebemos que existem diversos pontos ao redor da pessoa sendo escaneada, mas que não pertencem a ela. No parte inferior, por outro lado, estão as nuvens de pontos após a remoção dos *outliers*. Podemos ver que os pontos que estavam mais afastados do corpo foram identificados como *outliers* e foram destacados em vermelho. Em ambas as nuvens de pontos o mapa de cores varia de azul escuro até verde conforme a distância do ponto em relação ao sensor.

Figura 3.8 – Tratamento de Outliers



Fonte: Autoria Própria

3.3.2 Alinhamento da Nuvem de Pontos

Diferentemente de alguns escâneres portáteis em que o objeto se mantém fixo enquanto o escâner navega pelo espaço, no nosso o objeto submetido ao escaneamento que é rotacionado. Neste cenário, todas os dados adquiridos são tangentes ao sensor sendo necessárias transformações homogêneas *a posteriori* para que tenhamos uma nuvem rotacionada entorno de um ponto central.

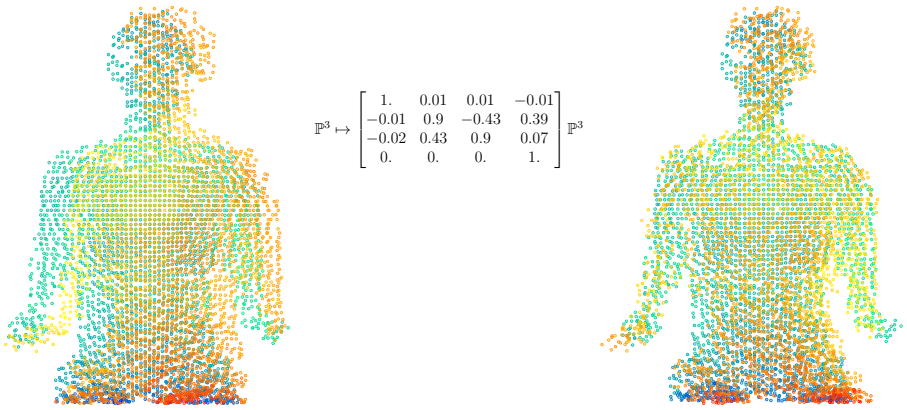
Dessa forma, para que fosse feito o alinhamento entre as malhas, utilizamos um método não supervisionado chamado ICP (*Iterative Closest Point*, do inglês, ponto mais próximo iterativo). O algorítimo de ICP pode ser resumido pela minimização da equação 3.12. Seja um ponto p_i pertencente a um conjunto de pontos P e de forma semelhante, q_i pertencente ao conjunto Q. O algorítimo busca alinhar as nuvens calculando iterativamente a transformada T que minimiza

a distância entre p_i e q_i. O resultado deste método fornece uma nova nuvem de pontos que é o resultado do alinhamento das nuvens em que o algorítimo é aplicado.

$$\overline{e} = \frac{1}{N} \sum_{i=1}^{N} \|Tp_i - q_i\|^2 \tag{3.12}$$

A Fig. 3.9 mostra duas nuvens de pontos submetidas ao alinhamento pelo ICP. Como na amostragem os corpos eram rotacionados lentamente, havia uma grande sobreposição dos dados o que facilitava a convergência do método. Como podemos ver, mesmo com nuvens razoavelmente desalinhadas, o algoritmo é capaz de fornecer uma transformação que minimize a distância entre as nuvens de pontos. Vale ressaltar que esse método trata as nuvens de pontos como corpos rígidos e, dessa forma, não podem sofrer escalonamentos somente translações e rotações.

Figura 3.9 – Alinhamento Nuvem de Pontos



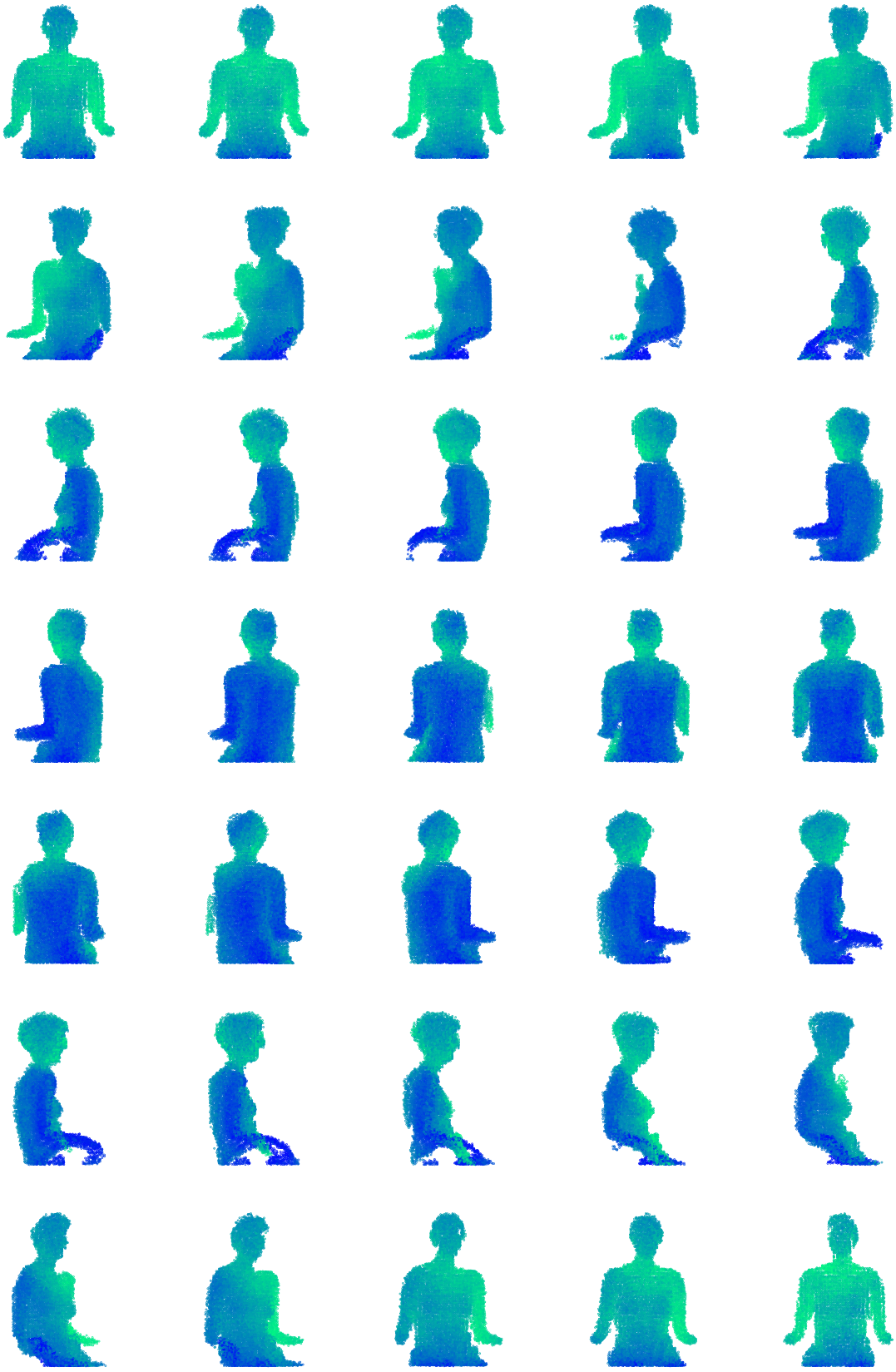
Fonte: Autoria Própria

Após o escaneamento, haviam cerca de 350 nuvens de pontos distintas distribuídas ao longo de uma revolução. Ao alinharmos uma nuvem de pontos em relação a sua anterior, concatenávamos mais pontos a uma única nuvem afim de obter-se os 360 graus do objeto escaneado. Em função das diversas concatenações o alinhamento das nuvens seguintes tornava-se mais custoso computacionalmente e, por seguinte, mais demorado.

Para que o processo de união e alinhamento fosse menos custoso, nós o dividimos em alguns passos de alinhamento. Inicialmente uníamos as nuvens em grupos de dez garantindo que a última nuvem e a primeira do grupo seguinte fossem iguais para aumentarmos a sobreposição para um próximo passo. Os resultados dessa primeira rodada de alinhamento pode ser visto na Fig. 3.10.

Para o caso específico exemplificado obtivemos 35 nuvens intermediárias. Com elas podemos já ver o delineado da pessoa bem como ter a referência espacial em função das cores. No

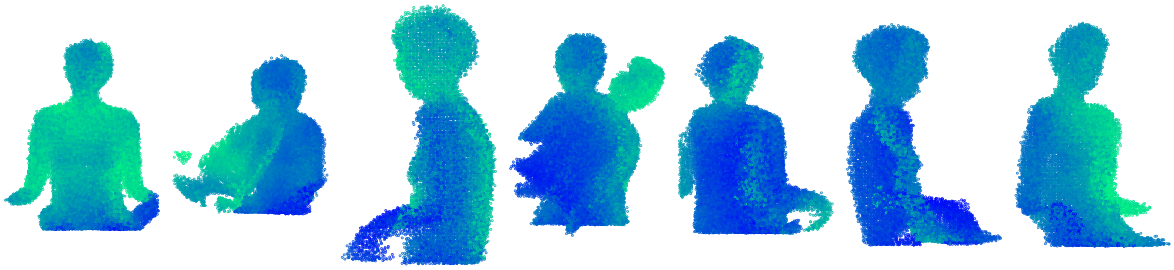
Figura 3.10 – Nuvem de pontos com agrupamento de dez vistas distintas



Fonte: Autoria Própria

passo seguinte, alinhamos em grupos de cinco da mesma forma que no primeiro passo. Os resultados podem ser vistos na Fig. 3.11. Nesta figura podemos ver que para as nuvens mais laterais o método diverge retornando resultado muito aquém do esperado.

Figura 3.11 – Resultado do alinhamento após união das 5 nuvens



Fonte: Autoria Própria

Por se tratar de um método não supervisionado há uma dificuldade inerente ao método em prevenir certos resultados, logo seriam necessárias outras formas de se alinhas ou funções de otimização diferentes que penalizassem rotações excessivas, por exemplo. Como o ICP não nos fornece uma maneira trivial aplicarmos essas restrições, utilizamos as nuvens de pontos das pontas, i.e., a primeira e última para prosseguirmos com a geração da malha tridimensional. O resultado pode ser visto na Fig. 3.12.

Figura 3.12 – Nuvem de pontos final utilizada no trabalho

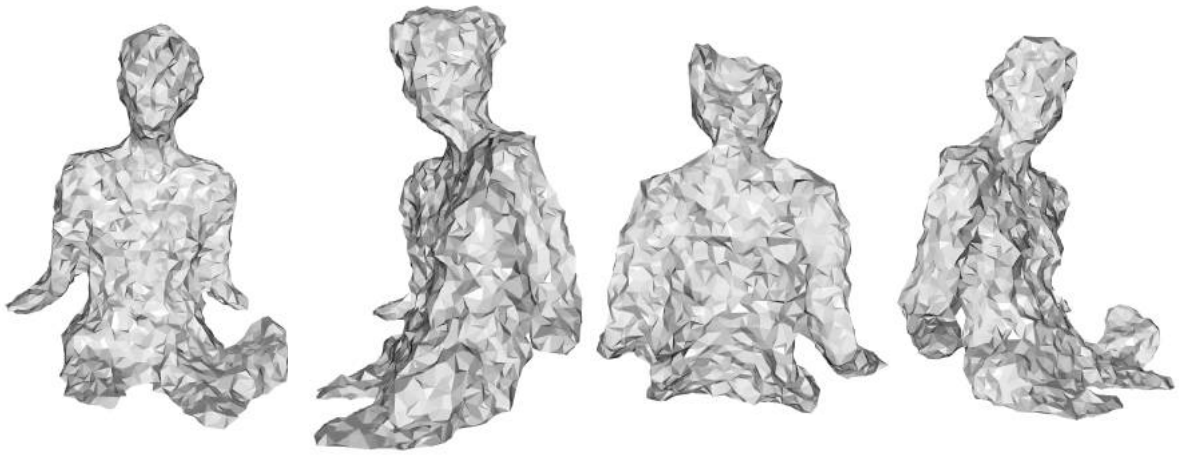
Fonte: Autoria Própria

3.3.3 Malha

Após a obtenção da última nuvem de pontos utilizamos alguns métodos diferentes para fazer a reconstrução da nuvem de pontos em uma malha formada por triângulos. Dessa forma, utilizamos três métodos diferentes com o que julgamos serem os melhores parâmetros para a

solução do problema. O primeiro caso consiste na aplicação do algoritmo de *Alpha Shapes* onde obtivemos a malha mostrada na Fig. 3.13.

Figura 3.13 – Reconstrução da malha utilizando *Alpha Shapes*

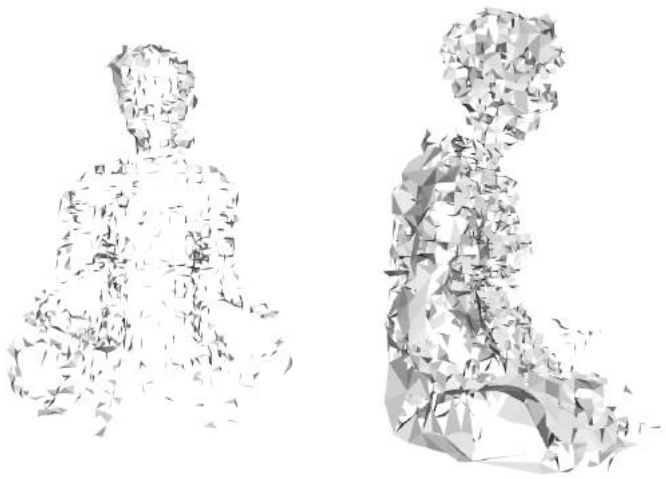


Fonte: Autoria Própria

Esse primeiro algoritmo nos retornou resultado muito satisfatórios nos quais podemos ver a silhueta sendo muito bem reproduzida. Além disso, vemos que as formas gerais do corpo estão presentes, embora com muitas arestas vivas. Contudo, podemos suaviza essa malha com a aplicação de um filtro análogo ao filtro passa baixa, suavizando as arestas.

O segundo método implementado foi a reconstrução utilizando o método de pivotamento de bolas. Os resultados estão dispostos na Fig. 3.14. Como podemos ver, o fechamento da nuvem de pontos não converge para um fechamento total. Mesmo ao alterarmos tanto o tamanho, quanto o número de possíveis raios, o método nos retorna um resultado bem inferior ao esperado.

Figura 3.14 – Reconstrução da malha utilizando *Ball Pivoting*



Fonte: Autoria Própria

Por fim, foi utilizado o método de reconstrução de Poisson. Por ser um dos estado da arte, esperávamos que o seu desempenho superasse os métodos anteriores, contudo, esse método culminou no pior resultado. O método não é capaz nem mesmo de aderir-se à nuvem de pontos, resultando numa superfície secante aos dados. Em função disso, mantivemos o método de *Alpha Shapes*, com a aplicação de um filtro para suavizar os cantos vivos.

4 Resultados e Discussão

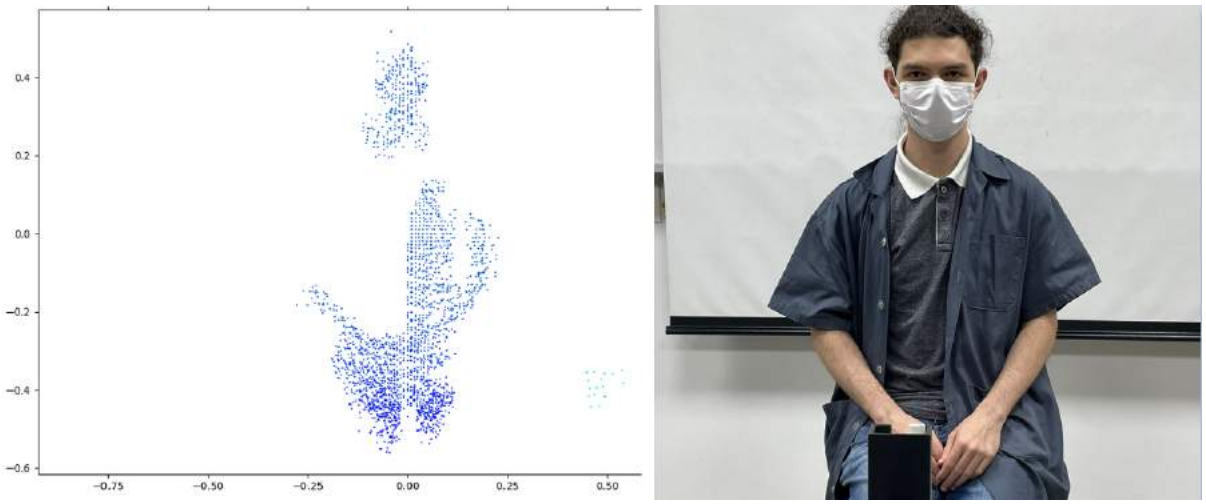
Podemos observar melhores resultados com o método de alpha shapes mostrado na Fig. 3.13. Apesar disso, o resultado está longe do ideal. A malha reconstruída possuí uma resolução baixa ao ponto de que a forma geral a pessoa escaneada é identificável, mas há falhas como no reconstrução do rosto e na captura das mãos.

Foi verificado que o sensor apresenta algumas limitações que dificultam a coleta de dados, entre elas identificamos problemas como a própria resolução do equipamento, o efeito da cor/tecido das roupas, pele e cabelo da pessoa sendo escaneada e até o ambiente. Estas limitações serão discutidas a seguir.

Um importante aspecto que foi levado em consideração foi a viabilidade econômica do projeto. O sensor utilizado no projeto, o Onion Tau LiDAR, é um sensor intermediário com resolução de 160×60. Parte do resultado pouco fiel obtido na malha reconstruída (Fig. 4.4) se deve em boa parte a baixa resolução do sensor, sendo que nesse escaneamento foi tomado em condições ideais. Naturalmente, um resultado bastante fiel poderia ter sido obtido com um sensor de alta resolução (1920×1080), no entanto, como mencionado na seção 1.2, um dos objetivos era construir um sistema que fosse replicável e acessível.

No caso de vestimentas pretas a leitura não era executada corretamente. Portanto é recomendado fazer o procedimento com roupas claras para evitar a absorção por parte do tecido. Além disso, alguns tecidos não puderam ser escaneados de forma a se ter um bom resultado, como visto no teste feito com um membro do grupo, na Fig. 4.1, mesmo a camisa e o jaleco sendo de tecidos escuros, o jaleco era possível de ser escaneado enquanto a camisa, não.

Figura 4.1 – Problema com Escaneamento - Roupa



Fonte: Autoria Própria

Também foi constatado pelo grupo que diversas superfícies escuras não são lidas corretamente pelo sensor, sendo necessário usar uma touca para que o couro cabeludo não fosse suprimido, e por consequência a cabeça da pessoa escaneada não fosse "cortada"no resultado final por conta do cabelo escuro.

Outro ponto notado é que se a pessoa estiver na frente de uma parede, o fundo interfere na leitura caso hajam muitos objetos que devem ser removidos, por isso a escolha de um lugar com um fundo de uma parede lise. Na Fig. 4.2 pode-se ver o problema causado por esse posicionamento falho.

Figura 4.2 – Problema com Escaneamento - Superfície

Fonte: Autoria Própria

Afim de realizar mais testes ainda trouxemos um convidado com um tom de pele mais escura do que dos membros já escaneados. Esta tentativa foi de grande ajuda para sabermos o quanto a cor da pele da pessoa escaneada influência na leitura.

Como é possível ver na Fig. 4.3, a leitura não é insatisfatória. Foi possível ajustar a leitura alterando o tempo de integração do sensor para se poder ser superfícies mais escuras, no entanto com isso se captou muito ruído, sendo inviável a leitura dessa forma.

Uma das dificuldades encontradas foi que não se tem os parâmetros do motor usado para movimentar a plataforma, para saber exatamente os coeficientes para executar o controle PID, por isso foi necessário achar tais valores por tentativa e erro, tomando um tempo considerável.

Apesar da alta complexidade do trabalho foi possível se obter um modelo final, o qual é apresentado na figura 4.4.

O modelo foi obtido após diversas tentativas e erros, sendo possível contornar-los via aplicação do algoritmo *Alpha Shapes*, o qual segundo o que foi mencionado nas referências,

Figura 4.3 – Problema com Escaneamento - Pele

Fonte: Autoria Própria

Figura 4.4 – Reconstrução da malha utilizando *Alpha Shapes*

Fonte: Autoria Própria

foi capaz de criar triângulos para produzir uma tetraedrização do espaço. Pode-se constatar que detalhes do rosto da pessoa escaneada não são possíveis de serem identificados no modelo final, apenas o contorno externo do corpo.

5 Conclusão

O resultado obtido com o sensor LiDAR usado se mostrou insatisfatório para o objetivo deste projeto, uma vez que para a aplicação ser funcional a mesma deve poder ser replicada para o maior número possível de pessoas, sendo isso impossível já que cada pessoa, ou a cada grupo com o mesmo tom de pele, exige uma configuração própria.

Com o resultados dos testes realizados, pode-se aferir que o método usado neste trabalho não poderia ser facilmente aplicado conforme o objetivo do grupo tendo em vista as diversas limitações, as inconsistências encontradas entre os tipos de roupa considerados dificultam, e em alguns casos, inviabilizam as medições pelo motivo do sensor não ser capaz de detectar certos tecidos e tonalidades de cor. Como indicação ao próximo trabalho a ser realizado, é recomendado o uso de outro sensor.

Apêndices

APÊNDICE A – Código Arduíno

Listing A.1 – Código Principal Arduino

```
#include <TimerOne.h>
#include "AS5040.h"
#include "ThreadController.h"
#include "Thread.h"
#include "SerialTransfer.h"
#include "PID.h"
#include "Platform.h"

#define CSpin 12
#define CLKpin 10
#define DOpin 11

#define PWM 5
#define LPWM 6
#define RPWM 7

float prevAngle = 0;
float prevVel = 0;
float prevT = 0;
float vFilt = 0;

// Instantiate communicaton
SerialTransfer dataTransfer;
struct __attribute__((packed)) STRUCT {
  unsigned short opt;
  double data;
} serialData;


// Instantiate sensor thread
AS5040 encoder = AS5040(CLKpin, CSpin, DOpin);

// Instantiate platform
Platform platform = Platform(PWM, LPWM, RPWM);

// Instantiate platform PID
PID platformPID;

// Instantiate platform control thread
Thread platformControl = Thread();

// Instantiate communication thread
Thread tx_communication = Thread();

// Instantiate a new Thread Controller
ThreadController controller = ThreadController();

// This is the callback for the Timer
void timerCallback() {
  controller.run();
}

```

```
53 void setup() {
54   // Open serial communication
55   Serial.begin(115200);
56   dataTransfer.begin(Serial);
57
58   // Initialize encoder
59   if (!encoder.begin())
60     Serial.println("Error setting up AS5040");
61
62   // Configure platform thread
63   platformControl.onRun(runPlatform);
64   platformControl.setInterval(100);  // Milliseconds
65   platformControl.enabled = false;   // Initially disabled
66
67   // Configure communication thread
68   tx_communication.onRun(sendPosition);
69   tx_communication.setInterval(15);  // Milliseconds
70   tx_communication.enabled = false;  // Initially disabled
71
72   // Configure encoder thread
73   encoder.setInterval(2);  // Milliseconds
74   encoder.enabled = true;  // Initially enabled
75
76   // Add all threads to thread controller
77   controller.add(&encoder);
78   controller.add(&platformControl);
79   controller.add(&tx_communication);
80
81   // Attach timer interrupt
82   Timer1.initialize(1000);                // Microseconds
83   Timer1.attachInterrupt(timerCallback);  // Run controller
84   Timer1.start();
85
86   // H-Bridge pins setup
87   pinMode(PWM, OUTPUT);
88   pinMode(LPWM, OUTPUT);
89   pinMode(RPWM, OUTPUT);
90
91   // PID setup
92   platformPID.setParams(3., 20., 0., -255., 255.);
93 }
94
95 void loop() {
96   if (dataTransfer.available()) {                      // If data received
97     uint16_t recSize = 0;                              // Received data index
98     recSize = dataTransfer.rxObj(serialData, recSize);  // Decode recived data
99     if (recSize != 6) sendData(0, 0);
100    else{
101      switch (serialData.opt) {
102        /* Available options
103        * 1 : Enable Motor (True/False)
104        * 2 : Change angular speed
105        */
106        case 1:
107          if ((int)serialData.data == 255) {
108            platformControl.enabled = true;
109            tx_communication.enabled = true;
110          } else {
111            platform.turnOff();
112            platformControl.enabled = false;
```

```
113             tx_communication.enabled = false;
114           }
115           break;
116         case 2:
117           platformPID.setTarget((float)serialData.data);
118           break;
119         case 3:
120           sendData(4, millis());
121         break;
122       }
123     }
124   }
125 }
126
127 void sendPosition() {
128   sendData(3, encoder.totalAngle);
129 }
130
131 void sendData(int opt, double data) {
132   serialData.opt = opt;                                  // Options (Python defined)
133   serialData.data = data;                                // Data to be sent
134   uint16_t dataSize = 0;                                 // Serial stream data size
135   dataSize = dataTransfer.txObj(serialData, dataSize);   // Encode message
136   dataTransfer.sendData(dataSize);                       // Send data
137 }
138
139 void runPlatform() {
140   platform.setSpeed(getPID());
141   platform.motorRun();
142 }
143
144 float getPID() {
145   unsigned long currT = micros();                      // Current Time
146   float totalAngle = encoder.totalAngle;               // Get current angle from encoder
147   float deltaT = ((float)(currT - prevT)) / 1.0e6;     // Delta time since last call
148   float velocity = (totalAngle - prevAngle) / deltaT;  // Angular Velocity
149
150   vFilt = 0.728 * vFilt + 0.136 * velocity + 0.136 * prevVel;  // Low pass filter
151   prevAngle = totalAngle;                       // Update previous angle
152   prevT = currT;                                // Update previous time
153   prevVel = velocity;
154
155   return platformPID.eval(vFilt, deltaT);  // Evaluate PID
156 }
```

Listing A.2 – Código AS5040

```
1  /*
2   *
3   * AS5040.h - Arduino library for AMS AS5040 magnetic rotary encoder chip
4   *
5   */
6
7  #ifndef AS5040_h
8  #define AS5040_h
9
10 #include "Arduino.h"
11 #include "Thread.h"
12
13 // defines for 5 bit _status value
```

```
14 #define AS5040_STATUS_OCF 0x10
15
16 class AS5040 : public Thread {
17 public:
18   float currAngle = 0;
19   float totalAngle = 0;
20
21   AS5040(byte pinCLK, byte pinCS, byte pinDO) {
22     _pinCLK = pinCLK;
23     _pinCS = pinCS;
24     _pinDO = pinDO;
25     _status = 0xFF;  // invalid status
26   }
27
28   boolean begin() {
29     return begin(0);
30   }
31   boolean begin(byte mode) {
32     return begin(mode, false, 0);
33   };
34   boolean begin(byte mode, boolean reverse, unsigned int offset) {
35     pinMode(_pinCLK, OUTPUT);
36     digitalWrite(_pinCLK, HIGH);
37     pinMode(_pinCS, OUTPUT);
38     digitalWrite(_pinCS, HIGH);
39     pinMode(_pinDO, INPUT_PULLUP);
40
41     byte count = 0;
42     while (read(), (_status & AS5040_STATUS_OCF) == 0) {
43       if (count > 30)
44         return false;  // failed to initialize
45       delay(1);
46       count++;
47     }
48
49     _startAngle = convertAngle(read());
50     return true;
51   }
52
53   void run() {
54     currAngle = convertAngle(read());
55     correctAngle();
56     checkQuadrant();
57     runned();
58   }
59
60   byte status() {
61     return _status;
62   }
63
64   boolean valid() {
65     return _parity == 0 && (_status & 0x18) == 0x10 && (_status & 3) != 3;
66   }
67
68   int Zaxis() {
69     switch (_status & 0x3) {
70       case 0:
71         return 0;
72       case 1:
73         return -1;
```

```
74        case 2:
75          return +1;
76        default:
77          return 0;
78      };
79    }
80
81  private:
82    byte _pinCLK;
83    byte _pinCS;
84    byte _pinDO;
85
86    byte _status;
87    byte _parity;
88    float _startAngle;
89    float _noTurns = 0.;
90    float _normAngle = 0.;
91    int _quadrant[2] = { 0, 0 };
92
93    byte even_parity(byte val) {
94      val = (val >> 1) ^ val;
95      val = (val >> 2) ^ val;
96      val = (val >> 4) ^ val;
97      return val & 1;
98    }
99
100   unsigned int read() {
101     digitalWrite(_pinCS, LOW);
102     unsigned int value = 0;
103     for (byte i = 0; i < 10; i++) {
104       digitalWrite(_pinCLK, LOW);
105       digitalWrite(_pinCLK, HIGH);
106       value = (value << 1) | digitalRead(_pinDO);
107     }
108     byte status = 0;
109     for (byte i = 0; i < 6; i++) {
110       digitalWrite(_pinCLK, LOW);
111       digitalWrite(_pinCLK, HIGH);
112       status = (status << 1) | digitalRead(_pinDO);
113     }
114     digitalWrite(_pinCS, HIGH);
115     _parity = even_parity(value >> 2) ^ even_parity(value & 3) ^ even_parity(status);
116     _status = status >> 1;
117     return value;
118   }
119
120   float convertAngle(int value) {
121     return (float)value * 360. / 1023;
122   }
123
124   void correctAngle() {
125     _normAngle = currAngle - _startAngle;
126     _normAngle = (_normAngle < 0) ? _normAngle + 360 : _normAngle;
127   }
128
129   void checkQuadrant() {
130
131     //Q1
132     if (_normAngle >= 0 && _normAngle <= 90) _quadrant[1] = 1;
133
```

```
134     //Q2
135     else if (_normAngle > 90 && _normAngle <= 180) _quadrant[1] = 2;
136
137     //Q3
138     else if (_normAngle > 180 && _normAngle <= 270) _quadrant[1] = 3;
139
140     //Q4
141     else if (_normAngle > 270 && _normAngle < 360) _quadrant[1] = 4;
142
143     if (_quadrant[0] == 1 && _quadrant[1] == 4) {
144       _noTurns--;
145     } else if (_quadrant[0] == 4 && _quadrant[1] == 1) {
146       _noTurns++;
147     }
148
149     _quadrant[0] = _quadrant[1];
150     totalAngle = ((_noTurns * 360.) + _normAngle);
151   }
152 };
153 #endif
```

Listing A.3 – Código Header Packet

```
1  /*
2  01111110 00000000 11111111 00000000 00000000 00000000 ... 00000000 10000001
3  |      | |      | |      | |      | |      | |      | | | |      | |______|__Stop byte
4  |      | |      | |      | |      | |      | |      | | | |______|___________8-bit CRC
5  |      | |      | |      | |      | |      | |      | |_|_____________________Rest of payload
6  |      | |      | |      | |      | |      | |______|_________________________2nd payload byte
7  |      | |      | |      | |      | |______|___________________________________1st payload byte
8  |      | |      | |      | |______|_____________________________________________# of payload
       bytes
9  |      | |      | |______|_____________________________________________________COBS Overhead
       byte
10 |      | |______|___________________________________________________________Packet ID (0 by
       default)
11 |______|_____________________________________________________________________Start byte (
       constant)
12 */
13
14 #pragma once
15 #include "Arduino.h"
16 #include "PacketCRC.h"
17
18
19 typedef void (*functionPtr)();
20
21
22 const int8_t CONTINUE = 3;
23 const int8_t NEW_DATA = 2;
24 const int8_t NO_DATA = 1;
25 const int8_t CRC_ERROR = 0;
26 const int8_t PAYLOAD_ERROR = -1;
27 const int8_t STOP_BYTE_ERROR = -2;
28 const int8_t STALE_PACKET_ERROR = -3;
29
30 const uint8_t START_BYTE = 0x7E;
31 const uint8_t STOP_BYTE = 0x81;
32
33 const uint8_t PREAMBLE_SIZE = 4;
```

```
34 const uint8_t POSTAMBLE_SIZE = 2;
35 const uint8_t MAX_PACKET_SIZE = 0xFE;  // Maximum allowed payload bytes per packet
36
37 const uint8_t DEFAULT_TIMEOUT = 50;
38
39
40 struct configST {
41   Stream* debugPort = &Serial;
42   bool debug = true;
43   const functionPtr* callbacks = NULL;
44   uint8_t callbacksLen = 0;
45   uint32_t timeout = __UINT32_MAX__;
46 };
47
48
49 class Packet {
50 public:  // <<----------------------------------------//public
51   uint8_t txBuff[MAX_PACKET_SIZE];
52   uint8_t rxBuff[MAX_PACKET_SIZE];
53   uint8_t preamble[PREAMBLE_SIZE] = { START_BYTE, 0, 0, 0 };
54   uint8_t postamble[POSTAMBLE_SIZE] = { 0, STOP_BYTE };
55
56   uint8_t bytesRead = 0;
57   int8_t status = 0;
58
59
60   void begin(const configST& configs);
61   void begin(const bool& _debug = true, Stream& _debugPort = Serial, const uint32_t& _timeout
       = DEFAULT_TIMEOUT);
62   uint8_t constructPacket(const uint16_t& messageLen, const uint8_t& packetID = 0);
63   uint8_t parse(const uint8_t& recChar, const bool& valid = true);
64   uint8_t currentPacketID();
65   void reset();
66
67
68   /*
69    uint16_t Packet::txObj(const T &val, const uint16_t &index=0, const uint16_t &len=sizeof(T)
       )
70    Description:
71    ------------
72     * Stuffs "len" number of bytes of an arbitrary object (byte, int,
73     float, double, struct, etc...) into the transmit buffer (txBuff)
74     starting at the index as specified by the argument "index"
75
76    Inputs:
77    -------
78     * const T &val - Pointer to the object to be copied to the
79     transmit buffer (txBuff)
80     * const uint16_t &index - Starting index of the object within the
81     transmit buffer (txBuff)
82     * const uint16_t &len - Number of bytes of the object "val" to transmit
83
84    Return:
85    -------
86     * uint16_t maxIndex - uint16_t maxIndex - Index of the transmit buffer (txBuff) that
       directly follows the bytes processed
87     by the calling of this member function
88   */
89   template<typename T>
90   uint16_t txObj(const T& val, const uint16_t& index = 0, const uint16_t& len = sizeof(T)) {
```

```
91      uint8_t* ptr = (uint8_t*)&val;
92      uint16_t maxIndex;
93
94      if ((len + index) > MAX_PACKET_SIZE)
95        maxIndex = MAX_PACKET_SIZE;
96      else
97        maxIndex = len + index;
98
99      for (uint16_t i = index; i < maxIndex; i++) {
100       txBuff[i] = *ptr;
101       ptr++;
102     }
103
104     return maxIndex;
105   }
106
107
108   /*
109    uint16_t Packet::rxObj(const T &val, const uint16_t &index=0, const uint16_t &len=sizeof(T)
       )
110    Description:
111    ------------
112     * Reads "len" number of bytes from the receive buffer (rxBuff)
113     starting at the index as specified by the argument "index"
114     into an arbitrary object (byte, int, float, double, struct, etc...)
115
116    Inputs:
117    -------
118     * const T &val - Pointer to the object to be copied into from the
119     receive buffer (rxBuff)
120     * const uint16_t &index - Starting index of the object within the
121     receive buffer (rxBuff)
122     * const uint16_t &len - Number of bytes in the object "val" received
123
124    Return:
125    -------
126     * uint16_t maxIndex - Index of the receive buffer (rxBuff) that directly follows the bytes
        processed
127     by the calling of this member function
128   */
129   template<typename T>
130   uint16_t rxObj(const T& val, const uint16_t& index = 0, const uint16_t& len = sizeof(T)) {
131     uint8_t* ptr = (uint8_t*)&val;
132     uint16_t maxIndex;
133
134     if ((len + index) > MAX_PACKET_SIZE)
135       maxIndex = MAX_PACKET_SIZE;
136     else
137       maxIndex = len + index;
138
139     for (uint16_t i = index; i < maxIndex; i++) {
140       *ptr = rxBuff[i];
141       ptr++;
142     }
143
144     return maxIndex;
145   }
146
147
148 private:  // <<--------------------------------------//private
```

```
149   enum fsm {
150     find_start_byte,
151     find_id_byte,
152     find_overhead_byte,
153     find_payload_len,
154     find_payload,
155     find_crc,
156     find_end_byte
157   };
158   fsm state = find_start_byte;
159
160   const functionPtr* callbacks = NULL;
161   uint8_t callbacksLen = 0;
162
163   Stream* debugPort;
164   bool debug = false;
165
166   uint8_t bytesToRec = 0;
167   uint8_t payIndex = 0;
168   uint8_t idByte = 0;
169   uint8_t overheadByte = 0;
170   uint8_t recOverheadByte = 0;
171
172   uint32_t packetStart = 0;
173   uint32_t timeout;
174
175
176   void calcOverhead(uint8_t arr[], const uint8_t& len);
177   int16_t findLast(uint8_t arr[], const uint8_t& len);
178   void stuffPacket(uint8_t arr[], const uint8_t& len);
179   void unpackPacket(uint8_t arr[]);
180 };
```

Listing A.4 – Código Packet

```
1  #include "Packet.h"
2
3
4  PacketCRC crc;
5
6
7  /*
8   void Packet::begin(const configST& configs)
9   Description:
10  ------------
11    * Advanced initializer for the Packet Class
12  Inputs:
13  -------
14    * const configST& configs - Struct that holds config
15    values for all possible initialization parameters
16  Return:
17  -------
18    * void
19  */
20 void Packet::begin(const configST& configs) {
21   debugPort = configs.debugPort;
22   debug = configs.debug;
23   callbacks = configs.callbacks;
24   callbacksLen = configs.callbacksLen;
25   timeout = configs.timeout;
```

```
26 }
27
28
29 /*
30  void Packet::begin(const bool& _debug, Stream& _debugPort, const uint32_t& _timeout)
31  Description:
32  ------------
33   * Simple initializer for the Packet Class
34  Inputs:
35  -------
36   * const bool& _debug - Whether or not to print error messages
37   * Stream &_debugPort - Serial port to print error messages
38   * const uint32_t& _timeout - Number of ms to wait before
39   declaring packet parsing timeout
40  Return:
41  -------
42   * void
43 */
44 void Packet::begin(const bool& _debug, Stream& _debugPort, const uint32_t& _timeout) {
45   debugPort = &_debugPort;
46   debug = _debug;
47   timeout = _timeout;
48 }
49
50
51 /*
52  uint8_t Packet::constructPacket(const uint16_t& messageLen, const uint8_t& packetID)
53  Description:
54  ------------
55   * Calculate, format, and insert the packet protocol metadata into the packet transmit
56   buffer
57  Inputs:
58  -------
59   * const uint16_t& messageLen - Number of values in txBuff
60   to send as the payload in the next packet
61   * const uint8_t& packetID - The packet 8-bit identifier
62  Return:
63  -------
64   * uint8_t - Number of payload bytes included in packet
65 */
66 uint8_t Packet::constructPacket(const uint16_t& messageLen, const uint8_t& packetID) {
67   if (messageLen > MAX_PACKET_SIZE) {
68     calcOverhead(txBuff, MAX_PACKET_SIZE);
69     stuffPacket(txBuff, MAX_PACKET_SIZE);
70     uint8_t crcVal = crc.calculate(txBuff, MAX_PACKET_SIZE);
71
72     preamble[1] = packetID;
73     preamble[2] = overheadByte;
74     preamble[3] = MAX_PACKET_SIZE;
75
76     postamble[0] = crcVal;
77
78     return MAX_PACKET_SIZE;
79   } else {
80     calcOverhead(txBuff, (uint8_t)messageLen);
81     stuffPacket(txBuff, (uint8_t)messageLen);
82     uint8_t crcVal = crc.calculate(txBuff, (uint8_t)messageLen);
83
84     preamble[1] = packetID;
85     preamble[2] = overheadByte;
```

```
86      preamble[3] = messageLen;
87
88      postamble[0] = crcVal;
89
90      return (uint8_t)messageLen;
91    }
92  }
93
94
95  /*
96   uint8_t Packet::parse(const uint8_t& recChar, const bool& valid)
97   Description:
98   ------------
99    * Parses incoming serial data, analyzes packet contents,
100   and reports errors/successful packet reception. Executes
101   callback functions for parsed packets whos ID has a
102   corresponding callback function set via
103   "void Packet::begin(const configST configs)"
104  Inputs:
105  -------
106   * const uint8_t& recChar - Next char to parse in the stream
107   * const bool& valid - Set if stream is "available()" and clear if not
108  Return:
109  -------
110   * uint8_t - Num bytes in RX buffer
111 */
112
113 uint8_t Packet::parse(const uint8_t& recChar, const bool& valid) {
114   bool packet_fresh = (packetStart == 0) || ((millis() - packetStart) < timeout);
115
116   if (!packet_fresh)  //packet is stale, start over.
117   {
118     if (debug)
119       debugPort->println("ERROR: STALE PACKET");
120
121     bytesRead = 0;
122     state = find_start_byte;
123     status = STALE_PACKET_ERROR;
124     packetStart = 0;
125
126     return bytesRead;
127   }
128
129   if (valid) {
130     switch (state) {
131       case find_start_byte:  ////////////////////////////////////////
132         {
133           if (recChar == START_BYTE) {
134             state = find_id_byte;
135             packetStart = millis();  //start the timer
136           }
137
138           break;
139         }
140
141       case find_id_byte:  /////////////////////////////////////////
142         {
143           idByte = recChar;
144           state = find_overhead_byte;
145           break;
```

```
146         }
147
148       case find_overhead_byte:  ////////////////////////////////////
149         {
150           recOverheadByte = recChar;
151           state = find_payload_len;
152           break;
153         }
154
155       case find_payload_len:  //////////////////////////////////////
156         {
157           if ((recChar > 0) && (recChar <= MAX_PACKET_SIZE)) {
158             bytesToRec = recChar;
159             payIndex = 0;
160             state = find_payload;
161           } else {
162             bytesRead = 0;
163             state = find_start_byte;
164             status = PAYLOAD_ERROR;
165
166             if (debug)
167               debugPort->println("ERROR: PAYLOAD_ERROR");
168
169             reset();
170             return bytesRead;
171           }
172           break;
173         }
174
175       case find_payload:  ////////////////////////////////////////////
176         {
177           if (payIndex < bytesToRec) {
178             rxBuff[payIndex] = recChar;
179             payIndex++;
180
181             if (payIndex == bytesToRec)
182               state = find_crc;
183           }
184           break;
185         }
186
187       case find_crc:  ////////////////////////////////////////////////
188         {
189           uint8_t calcCrc = crc.calculate(rxBuff, bytesToRec);
190
191           if (calcCrc == recChar)
192             state = find_end_byte;
193           else {
194             bytesRead = 0;
195             state = find_start_byte;
196             status = CRC_ERROR;
197
198             if (debug)
199               debugPort->println("ERROR: CRC_ERROR");
200
201             reset();
202             return bytesRead;
203           }
204
205           break;
```

```
206         }
207
208       case find_end_byte:  ////////////////////////////////////////
209         {
210           state = find_start_byte;
211
212           if (recChar == STOP_BYTE) {
213             unpackPacket(rxBuff);
214             bytesRead = bytesToRec;
215             status = NEW_DATA;
216
217             if (callbacks) {
218               if (idByte < callbacksLen)
219                 callbacks[idByte]();
220               else if (debug) {
221                 debugPort->print(F("ERROR: No callback available for packet ID "));
222                 debugPort->println(idByte);
223               }
224             }
225             packetStart = 0;  // reset the timer
226             return bytesToRec;
227           }
228
229           bytesRead = 0;
230           status = STOP_BYTE_ERROR;
231
232           if (debug)
233             debugPort->println("ERROR: STOP_BYTE_ERROR");
234
235           reset();
236           return bytesRead;
237           break;
238         }
239
240       default:
241         {
242           if (debug) {
243             debugPort->print("ERROR: Undefined state ");
244             debugPort->println(state);
245           }
246
247           reset();
248           bytesRead = 0;
249           state = find_start_byte;
250           break;
251         }
252     }
253   } else {
254     bytesRead = 0;
255     status = NO_DATA;
256     return bytesRead;
257   }
258
259   bytesRead = 0;
260   status = CONTINUE;
261   return bytesRead;
262 }
263
264
265 /*
```

```
266  uint8_t Packet::currentPacketID()
267  Description:
268  ------------
269   * Returns the ID of the last parsed packet
270  Inputs:
271  -------
272   * void
273  Return:
274  -------
275   * uint8_t - ID of the last parsed packet
276 */
277 uint8_t Packet::currentPacketID() {
278   return idByte;
279 }
280
281
282 /*
283  void Packet::calcOverhead(uint8_t arr[], const uint8_t &len)
284  Description:
285  ------------
286   * Calculates the COBS (Consistent Overhead Stuffing) Overhead
287   byte and stores it in the class's overheadByte variable. This
288   variable holds the byte position (within the payload) of the
289   first payload byte equal to that of START_BYTE
290  Inputs:
291  -------
292   * uint8_t arr[] - Array of values the overhead is to be calculated
293   over
294   * const uint8_t &len - Number of elements in arr[]
295  Return:
296  -------
297   * void
298 */
299 void Packet::calcOverhead(uint8_t arr[], const uint8_t& len) {
300   overheadByte = 0xFF;
301
302   for (uint8_t i = 0; i < len; i++) {
303     if (arr[i] == START_BYTE) {
304       overheadByte = i;
305       break;
306     }
307   }
308 }
309
310
311 /*
312  int16_t Packet::findLast(uint8_t arr[], const uint8_t &len)
313  Description:
314  ------------
315   * Finds last instance of the value START_BYTE within the given
316   packet array
317  Inputs:
318  -------
319   * uint8_t arr[] - Packet array
320   * const uint8_t &len - Number of elements in arr[]
321  Return:
322  -------
323   * int16_t - Index of last instance of the value START_BYTE within the given
324   packet array
325 */
```

```
326 int16_t Packet::findLast(uint8_t arr[], const uint8_t& len) {
327   for (uint8_t i = (len - 1); i != 0xFF; i--)
328     if (arr[i] == START_BYTE)
329       return i;
330
331   return -1;
332 }
333
334
335 /*
336  void Packet::stuffPacket(uint8_t arr[], const uint8_t &len)
337  Description:
338  ------------
339   * Enforces the COBS (Consistent Overhead Stuffing) ruleset across
340   all bytes in the packet against the value of START_BYTE
341  Inputs:
342  -------
343   * uint8_t arr[] - Array of values to stuff
344   * const uint8_t &len - Number of elements in arr[]
345  Return:
346  -------
347   * void
348 */
349 void Packet::stuffPacket(uint8_t arr[], const uint8_t& len) {
350   int16_t refByte = findLast(arr, len);
351
352   if (refByte != -1) {
353     for (uint8_t i = (len - 1); i != 0xFF; i--) {
354       if (arr[i] == START_BYTE) {
355         arr[i] = refByte - i;
356         refByte = i;
357       }
358     }
359   }
360 }
361
362
363 /*
364  void Packet::unpackPacket(uint8_t arr[], const uint8_t &len)
365  Description:
366  ------------
367   * Unpacks all COBS-stuffed bytes within the array
368  Inputs:
369  -------
370   * uint8_t arr[] - Array of values to unpack
371   * const uint8_t &len - Number of elements in arr[]
372  Return:
373  -------
374   * void
375 */
376 void Packet::unpackPacket(uint8_t arr[]) {
377   uint8_t testIndex = recOverheadByte;
378   uint8_t delta = 0;
379
380   if (testIndex <= MAX_PACKET_SIZE) {
381     while (arr[testIndex]) {
382       delta = arr[testIndex];
383       arr[testIndex] = START_BYTE;
384       testIndex += delta;
385     }
```

```
386     arr[testIndex] = START_BYTE;
387   }
388 }
389
390
391 /*
392  void Packet::reset()
393  Description:
394  ------------
395   * Clears out the tx, and rx buffers, plus resets
396   the "bytes read" variable, finite state machine, etc
397  Inputs:
398  -------
399   * void
400  Return:
401  -------
402   * void
403 */
404 void Packet::reset() {
405   memset(txBuff, 0, sizeof(txBuff));
406   memset(rxBuff, 0, sizeof(rxBuff));
407
408   bytesRead = 0;
409   status = CONTINUE;
410   packetStart = 0;
411 }
```

Listing A.5 – Código Header Packet CRC

```
1  #pragma once
2  #include "Arduino.h"
3
4
5  class PacketCRC {
6  public:  // <<----------------------------------------//public
7    uint8_t poly = 0;
8
9
10   PacketCRC(const uint8_t& polynomial = 0x9B, const uint8_t& crcLen = 8) {
11     poly = polynomial;
12     crcLen_ = crcLen;
13     tableLen_ = pow(2, crcLen);
14     csTable = new uint8_t[tableLen_];
15
16     generateTable();
17   }
18
19   void generateTable() {
20     for (uint16_t i = 0; i < tableLen_; ++i) {
21       int curr = i;
22
23       for (int j = 0; j < 8; ++j) {
24         if ((curr & 0x80) != 0)
25           curr = (curr << 1) ^ (int)poly;
26         else
27           curr <<= 1;
28       }
29
30       csTable[i] = (byte)curr;
31     }
```

```
32   }
33
34   void printTable() {
35     for (uint16_t i = 0; i < tableLen_; i++) {
36       Serial.print(csTable[i], HEX);
37
38       if ((i + 1) % 16)
39         Serial.print(' ');
40       else
41         Serial.println();
42     }
43   }
44
45   uint8_t calculate(const uint8_t& val) {
46     if (val < tableLen_)
47       return csTable[val];
48     return 0;
49   }
50
51   uint8_t calculate(uint8_t arr[], uint8_t len) {
52     uint8_t crc = 0;
53     for (uint16_t i = 0; i < len; i++)
54       crc = csTable[crc ^ arr[i]];
55
56     return crc;
57   }
58
59
60 private:  // <<----------------------------------------//private
61   uint16_t tableLen_;
62   uint8_t crcLen_;
63   uint8_t* csTable;
64 };
65
66
67 extern PacketCRC crc;
```

Listing A.6 – Código Header Serial Transfer

```
1  #pragma once
2  #include "Arduino.h"
3  #include "Packet.h"
4
5
6  class SerialTransfer {
7  public:  // <<----------------------------------------//public
8    Packet packet;
9    uint8_t bytesRead = 0;
10   int8_t status = 0;
11
12
13   void begin(Stream& _port, const configST configs);
14   void begin(Stream& _port, const bool _debug = true, Stream& _debugPort = Serial, uint32_t
       _timeout = DEFAULT_TIMEOUT);
15   uint8_t sendData(const uint16_t& messageLen, const uint8_t packetID = 0);
16   uint8_t available();
17   bool tick();
18   uint8_t currentPacketID();
19   void reset();
20
```

```
21
22  /*
23   uint16_t SerialTransfer::txObj(const T &val, const uint16_t &index=0, const uint16_t &len=
       sizeof(T))
24   Description:
25   ------------
26    * Stuffs "len" number of bytes of an arbitrary object (byte, int,
27    float, double, struct, etc...) into the transmit buffer (txBuff)
28    starting at the index as specified by the argument "index"
29   Inputs:
30   -------
31    * const T &val - Pointer to the object to be copied to the
32    transmit buffer (txBuff)
33    * const uint16_t &index - Starting index of the object within the
34    transmit buffer (txBuff)
35    * const uint16_t &len - Number of bytes of the object "val" to transmit
36   Return:
37   -------
38    * uint16_t maxIndex - uint16_t maxIndex - Index of the transmit buffer (txBuff) that
       directly follows the bytes processed
39    by the calling of this member function
40  */
41  template<typename T>
42  uint16_t txObj(const T& val, const uint16_t& index = 0, const uint16_t& len = sizeof(T)) {
43    return packet.txObj(val, index, len);
44  }
45
46
47  /*
48   uint16_t SerialTransfer::rxObj(const T &val, const uint16_t &index=0, const uint16_t &len=
       sizeof(T))
49   Description:
50   ------------
51    * Reads "len" number of bytes from the receive buffer (rxBuff)
52    starting at the index as specified by the argument "index"
53    into an arbitrary object (byte, int, float, double, struct, etc...)
54   Inputs:
55   -------
56    * const T &val - Pointer to the object to be copied into from the
57    receive buffer (rxBuff)
58    * const uint16_t &index - Starting index of the object within the
59    receive buffer (rxBuff)
60    * const uint16_t &len - Number of bytes in the object "val" received
61   Return:
62   -------
63    * uint16_t maxIndex - Index of the receive buffer (rxBuff) that directly follows the bytes
       processed
64    by the calling of this member function
65  */
66  template<typename T>
67  uint16_t rxObj(const T& val, const uint16_t& index = 0, const uint16_t& len = sizeof(T)) {
68    return packet.rxObj(val, index, len);
69  }
70
71
72  /*
73   uint8_t SerialTransfer::sendDatum(const T &val, const uint16_t &len=sizeof(T))
74   Description:
75   ------------
76    * Stuffs "len" number of bytes of an arbitrary object (byte, int,
```

```
77     float, double, struct, etc...) into the transmit buffer (txBuff)
78     starting at the index as specified by the argument "index" and
79     automatically transmits the bytes in an individual packet
80    Inputs:
81    -------
82     * const T &val - Pointer to the object to be copied to the
83     transmit buffer (txBuff)
84     * const uint16_t &len - Number of bytes of the object "val" to transmit
85    Return:
86    -------
87     * uint8_t - Number of payload bytes included in packet
88   */
89   template<typename T>
90   uint8_t sendDatum(const T& val, const uint16_t& len = sizeof(T)) {
91     return sendData(packet.txObj(val, 0, len));
92   }
93
94
95 private:  // <<---------------------------------------//private
96   Stream* port;
97   uint32_t timeout;
98 };
```

Listing A.7 – Código Serial Transfer

```
1  #include "SerialTransfer.h"
2
3
4  /*
5   void SerialTransfer::begin(Stream &_port, configST configs)
6   Description:
7   ------------
8    * Advanced initializer for the SerialTransfer Class
9   Inputs:
10  -------
11   * const Stream &_port - Serial port to communicate over
12   * const configST configs - Struct that holds config
13   values for all possible initialization parameters
14  Return:
15  -------
16   * void
17 */
18 void SerialTransfer::begin(Stream& _port, const configST configs) {
19   port = &_port;
20   packet.begin(configs);
21 }
22
23
24 /*
25  void SerialTransfer::begin(Stream &_port, const bool _debug, Stream &_debugPort)
26  Description:
27  ------------
28   * Simple initializer for the SerialTransfer Class
29  Inputs:
30  -------
31   * const Stream &_port - Serial port to communicate over
32   * const bool _debug - Whether or not to print error messages
33   * const Stream &_debugPort - Serial port to print error messages
34  Return:
35  -------
```

```
36   * void
37 */
38 void SerialTransfer::begin(Stream& _port, const bool _debug, Stream& _debugPort, uint32_t
       _timeout) {
39   port = &_port;
40   timeout = _timeout;
41   packet.begin(_debug, _debugPort, _timeout);
42 }
43
44
45 /*
46  uint8_t SerialTransfer::sendData(const uint16_t &messageLen, const uint8_t packetID)
47  Description:
48  ------------
49   * Send a specified number of bytes in packetized form
50  Inputs:
51  -------
52   * const uint16_t &messageLen - Number of values in txBuff
53   to send as the payload in the next packet
54   * const uint8_t packetID - The packet 8-bit identifier
55  Return:
56  -------
57   * uint8_t numBytesIncl - Number of payload bytes included in packet
58 */
59 uint8_t SerialTransfer::sendData(const uint16_t& messageLen, const uint8_t packetID) {
60   uint8_t numBytesIncl;
61
62   numBytesIncl = packet.constructPacket(messageLen, packetID);
63   port->write(packet.preamble, sizeof(packet.preamble));
64   port->write(packet.txBuff, numBytesIncl);
65   port->write(packet.postamble, sizeof(packet.postamble));
66
67   return numBytesIncl;
68 }
69
70
71 /*
72  uint8_t SerialTransfer::available()
73  Description:
74  ------------
75   * Parses incoming serial data, analyzes packet contents,
76   and reports errors/successful packet reception
77  Inputs:
78  -------
79   * void
80  Return:
81  -------
82   * uint8_t bytesRead - Num bytes in RX buffer
83 */
84 uint8_t SerialTransfer::available() {
85   bool valid = false;
86   uint8_t recChar = 0xFF;
87
88   if (port->available()) {
89     valid = true;
90
91     while (port->available()) {
92       recChar = port->read();
93
94       bytesRead = packet.parse(recChar, valid);
```

```
95        status = packet.status;
96
97        if (status != CONTINUE) {
98          if (status < 0)
99            reset();
100
101         break;
102       }
103     }
104   } else {
105     bytesRead = packet.parse(recChar, valid);
106     status = packet.status;
107
108     if (status < 0)
109       reset();
110   }
111
112   return bytesRead;
113 }
114
115
116 /*
117  bool SerialTransfer::tick()
118  Description:
119  ------------
120   * Checks to see if any packets have been fully parsed. This
121   is basically a wrapper around the method "available()" and
122   is used primarily in conjunction with callbacks
123  Inputs:
124  -------
125   * void
126  Return:
127  -------
128   * bool - Whether or not a full packet has been parsed
129 */
130 bool SerialTransfer::tick() {
131   if (available())
132     return true;
133
134   return false;
135 }
136
137
138 /*
139  uint8_t SerialTransfer::currentPacketID()
140  Description:
141  ------------
142   * Returns the ID of the last parsed packet
143  Inputs:
144  -------
145   * void
146  Return:
147  -------
148   * uint8_t - ID of the last parsed packet
149 */
150 uint8_t SerialTransfer::currentPacketID() {
151   return packet.currentPacketID();
152 }
153
154
```

```
/*
 void SerialTransfer::reset()
 Description:
 ------------
  * Clears out the tx, and rx buffers, plus resets
  the "bytes read" variable, finite state machine, etc
 Inputs:
 -------
  * void
 Return:
 -------
  * void
*/
void SerialTransfer::reset() {
  while (port->available())
    port->read();

  packet.reset();
  status = packet.status;
}
```

Listing A.8 – Código Header PID

```
class PID {
private:
  float _kp, _ki, _kd;       // Parameters
  float _eprev, _eintegral;  // Storage
  float _umin, _umax;        // Anti-wind up clamp limits
  float _target = 0;

public:
  // Constructor
  PID()
    : _kp(1), _ki(0), _kd(0), _eprev(0.0),
      _eintegral(0.0), _umin(-1), _umax(1) {}

  // A function to set the parameters
  void setParams(float kpIn, float kiIn, float kdIn, float uminIn, float umaxIn) {
    _kp = kpIn;
    _ki = kiIn;
    _kd = kdIn;
    _umin = uminIn;
    _umax = umaxIn;
  }

  float setTarget(float target) {
    _target = target;
  }

  // A function to compute the control signal
  float eval(float value, float dt) {
    // Compute error
    float e = _target - value;
    float dedt = (e - _eprev) / (dt);
    _eprev = e;  // store previous error

    float eintegralUpdate = _eintegral + e * dt;

    // Evaluate the control signal
    float u = _kp * e + _ki * eintegralUpdate + _kd * dedt;
```

```
38
39      // Anti-wind up clamping
40      // If either of the first two conditions are true
41      // The integral is not updated
42      if (u < _umin) {
43        u = _umin;
44      } else if (u > _umax) {
45        u = _umax;
46      } else {
47        // Otherwise update the integral
48        _eintegral = eintegralUpdate;
49      }
50      return u;
51    }
52  };
```

Listing A.9 – Código Header Platform

```
1  class Platform {
2  public:
3    Platform(byte PWM, byte LPWM, byte RPWM) {
4      _pinPWM = PWM;
5      _pinLPWM = LPWM;
6      _pinRPWM = RPWM;
7    }
8
9    void motorRun() {
10     analogWrite(_pinPWM, abs(_valSpeed));  // PWM signal on H-Bridge signal
11     if (_valSpeed < 0) {                   // CW Rotation
12       digitalWrite(_pinRPWM, HIGH);
13       digitalWrite(_pinLPWM, LOW);
14     } else if (_valSpeed > 0) {  // CCW Rotation
15       digitalWrite(_pinRPWM, LOW);
16       digitalWrite(_pinLPWM, HIGH);
17     } else {  // Disable Rotation
18       digitalWrite(_pinRPWM, LOW);
19       digitalWrite(_pinLPWM, LOW);
20     }
21   }
22
23   void turnOff() {
24     analogWrite(_pinPWM, 0);  // PWM signal on H-Bridge signal
25     digitalWrite(_pinRPWM, LOW);
26     digitalWrite(_pinLPWM, LOW);
27   }
28
29   void setSpeed(float speed) {
30     _valSpeed = speed;
31   }
32
33 private:
34   byte _pinPWM;
35   byte _pinLPWM;
36   byte _pinRPWM;
37
38   float _valSpeed = 0;
39 };
```

Listing A.10 – Código Header Thread

```
/*
  Thread.h - An runnable object

  Thread is responsable for holding the "action" for something,
  also, it responds if it "should" or "should not" run, based on
  the current time;

  For instructions, go to https://github.com/ivanseidel/ArduinoThread

  Created by Ivan Seidel Gomes, March, 2013.
  Released into the public domain.
*/

#ifndef Thread_h
#define Thread_h

#if defined(ARDUINO) && ARDUINO >= 100
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

#include <inttypes.h>

/*
  Uncomment this line to enable ThreadName Strings.

  It might be usefull if you are loging thread with Serial,
  or displaying a list of threads...
*/
// #define USE_THREAD_NAMES 1

class Thread {
protected:
  // Desired interval between runs
  unsigned long interval;

  // Last runned time in Ms
  unsigned long last_run;

  // Scheduled run in Ms (MUST BE CACHED)
  unsigned long _cached_next_run;

  /*
    IMPORTANT! Run after all calls to run()
    Updates last_run and cache next run.
    NOTE: This MUST be called if extending
    this class and implementing run() method
  */
  void runned(unsigned long time);

  // Default is to mark it runned "now"
  void runned() {
    runned(millis());
  }

  // Callback for run() if not implemented
  void (*_onRun)(void);
```

```
59
60 public:
61
62   // If the current Thread is enabled or not
63   bool enabled;
64
65   // ID of the Thread (initialized from memory adr.)
66   int ThreadID;
67
68 #ifdef USE_THREAD_NAMES
69   // Thread Name (used for better UI).
70   String ThreadName;
71 #endif
72
73   Thread(void (*callback)(void) = NULL, unsigned long _interval = 0);
74
75   // Set the desired interval for calls, and update _cached_next_run
76   virtual void setInterval(unsigned long _interval);
77
78   // Return if the Thread should be runned or not
79   virtual bool shouldRun(unsigned long time);
80
81   // Default is to check whether it should run "now"
82   bool shouldRun() {
83     return shouldRun(millis());
84   }
85
86   // Callback set
87   void onRun(void (*callback)(void));
88
89   // Runs Thread
90   virtual void run();
91 };
92
93 #endif
```

Listing A.11 – Código Thread

```
1 #include "Thread.h"
2
3 Thread::Thread(void (*callback)(void), unsigned long _interval) {
4   enabled = true;
5   onRun(callback);
6   _cached_next_run = 0;
7   last_run = millis();
8
9   ThreadID = (int)this;
10 #ifdef USE_THREAD_NAMES
11   ThreadName = "Thread ";
12   ThreadName = ThreadName + ThreadID;
13 #endif
14
15   setInterval(_interval);
16 };
17
18 void Thread::runned(unsigned long time) {
19   // Saves last_run
20   last_run = time;
21
22   // Cache next run
```

```
23   _cached_next_run = last_run + interval;
24 }
25
26 void Thread::setInterval(unsigned long _interval) {
27   // Save interval
28   interval = _interval;
29
30   // Cache the next run based on the last_run
31   _cached_next_run = last_run + interval;
32 }
33
34 bool Thread::shouldRun(unsigned long time) {
35   // If the "sign" bit is set the signed difference would be negative
36   bool time_remaining = (time - _cached_next_run) & 0x80000000;
37
38   // Exceeded the time limit, AND is enabled? Then should run...
39   return !time_remaining && enabled;
40 }
41
42 void Thread::onRun(void (*callback)(void)) {
43   _onRun = callback;
44 }
45
46 void Thread::run() {
47   if (_onRun != NULL)
48     _onRun();
49
50   // Update last_run and _cached_next_run
51   runned();
52 }
```

Listing A.12 – Código Header Thread Controller

```
1 /*
2   ThreadController.h - Controlls a list of Threads with different timings
3
4   Basicaly, what it does is to keep track of current Threads and run when
5   necessary.
6
7   ThreadController is an extended class of Thread, because of that,
8   it allows you to add a ThreadController inside another ThreadController...
9
10  For instructions, go to https://github.com/ivanseidel/ArduinoThread
11
12  Created by Ivan Seidel Gomes, March, 2013.
13  Released into the public domain.
14 */
15
16 #ifndef ThreadController_h
17 #define ThreadController_h
18
19 #include "Thread.h"
20 #include "inttypes.h"
21
22 #define MAX_THREADS 15
23
24 class ThreadController : public Thread {
25 protected:
26   Thread* thread[MAX_THREADS];
27   int cached_size;
```

```
28 public:
29   ThreadController(unsigned long _interval = 0);
30
31   // run() Method is overrided
32   void run();
33
34   // Adds a thread in the first available slot (remove first)
35   // Returns if the Thread could be added or not
36   bool add(Thread* _thread);
37
38   // remove the thread (given the Thread* or ThreadID)
39   void remove(int _id);
40   void remove(Thread* _thread);
41
42   // Removes all threads
43   void clear();
44
45   // Return the quantity of Threads
46   int size(bool cached = true);
47
48   // Return the I Thread on the array
49   // Returns NULL if none found
50   Thread* get(int index);
51 };
52
53 #endif
```

APÊNDICE B – Python - Aquisição

Listing B.1 – Código Amostragem de Dados - Plataforma

```
#%%
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

from TauLidarCommon.frame import FrameType, Frame
from TauLidarCamera.camera import Camera
from TauLidarCamera.constants import VALUE_20MHZ
from TauLidarCommon.color import ColorMode

from time import sleep
from pySerialTransfer import pySerialTransfer as txfer

import pickle as pk

#%%
Camera.setColorMode(ColorMode.DISTANCE)    ## use distance for point color
Camera.setRange(0, 4000)                   ## points in the distance range to be colored

camera = Camera.open()
cameraInfo = camera.info()
print("\nToF camera opened successfully:")

print("    model:      %s" % cameraInfo.model)
print("    firmware:   %s" % cameraInfo.firmware)
print("    uid:        %s" % cameraInfo.uid)
print("    resolution: %s" % cameraInfo.resolution)
print("    port:       %s" % cameraInfo.port)

camera.setModulationFrequency(VALUE_20MHZ) ## frequency: 20MHZ
camera.setModulationChannel(0)             ## autoChannelEnabled: 0, channel: 0
camera.setMode(0)                          ## Mode 0, wide fov
camera.setHdr(0)                           ## HDR off
camera.setIntegrationTime3d(0, 1000)        ## set integration time 0: 1000
camera.setMinimalAmplitude(0, 5)          ## set minimal amplitude 0: 80
camera.setOffset(0)                        ## set distance offset: 0
camera.setRoi(0, 0, 159, 59)               ## set ROI to max width and height

# %%
if __name__ == '__main__':
    link = txfer.SerialTransfer('/dev/cu.usbserial-14140', baud=115200)
    link.open()
    sleep(5)

    for i in range(100):
        sendSize = 0
        sendSize = link.tx_obj(1, start_pos=sendSize, val_type_override='H')
        sendSize = link.tx_obj(255., start_pos=sendSize, val_type_override='f')
        link.send(sendSize)

    for i in range(100):
        sendSize = 0
```

```
53          sendSize = link.tx_obj(2, start_pos=sendSize, val_type_override='H')
54          sendSize = link.tx_obj(80., start_pos=sendSize, val_type_override='f')
55          link.send(sendSize)
56
57      try:
58          encoder = []
59          scan = []
60          while True:
61              if link.available():
62                  frame = camera.readFrameRawData(frameType=FrameType.DISTANCE)
63                  scan.append(frame)
64
65                  recSize = 0
66                  _ = link.rx_obj(obj_type='H', start_pos=recSize)
67                  recSize += txfer.STRUCT_FORMAT_LENGTHS['H']
68                  encoder.append(link.rx_obj(obj_type='f', start_pos=recSize))
69
70      except KeyboardInterrupt:
71          with open('scan4.pkl', 'wb') as fpkl:
72              pk.dump(scan, fpkl)
73
74          with open('encoder4.pkl', 'wb') as fpkl:
75              pk.dump(encoder, fpkl)
76
77      finally:
78          for i in range(100):
79              sendSize = 0
80              sendSize = link.tx_obj(1, start_pos=sendSize, val_type_override='H')
81              sendSize = link.tx_obj(0., start_pos=sendSize, val_type_override='f')
82              link.send(sendSize)
83  # %%
```

Listing B.2 – Código Amostragem de Dados - Visualização

```
1  #%%
2  from ast import Pass
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import matplotlib
6
7  from TauLidarCommon.frame import FrameType, Frame
8  from TauLidarCamera.camera import Camera
9  from TauLidarCamera.constants import VALUE_20MHZ
10 from TauLidarCommon.color import ColorMode
11
12 from time import sleep
13 from pySerialTransfer import pySerialTransfer as txfer
14
15 import pickle as pk
16 import time
17
18 #%%
19 Camera.setColorMode(ColorMode.DISTANCE)    ## use distance for point color
20 Camera.setRange(0, 4000)                   ## points in the distance range to be colored
21
22 camera = Camera.open()
23 cameraInfo = camera.info()
24 print("\nToF camera opened successfully:")
25
26 print("    model:      %s" % cameraInfo.model)
```

```
print("    firmware:   %s" % cameraInfo.firmware)
print("    uid:        %s" % cameraInfo.uid)
print("    resolution: %s" % cameraInfo.resolution)
print("    port:       %s" % cameraInfo.port)

camera.setModulationFrequency(VALUE_20MHZ) ## frequency: 20MHZ
camera.setModulationChannel(0)             ## autoChannelEnabled: 0, channel: 0
camera.setMode(0)                          ## Mode 0, wide fov
camera.setHdr(0)                           ## HDR off
camera.setIntegrationTime3d(0, 600)        ## set integration time 0: 1000
camera.setMinimalAmplitude(0, 60)          ## set minimal amplitude 0: 80
camera.setOffset(0)                        ## set distance offset: 0
camera.setRoi(0, 0, 159, 59)

#%%
from matplotlib.animation import FuncAnimation

def my_function(i):
    # clear axis
    #for ax in axs: ax.cla()
    axs.cla()

    # plot cpu
    frames.append(camera.readFrame(frameType=FrameType.DISTANCE))
    array = np.array(frames[-1].points_3d)[:,:3]

    axs.scatter(array[:,1], array[:,0], s=.5, cmap='winter', c=array[:,-1])
    #axs[0].scatter(array[:,1], array[:,0], s=.5, cmap='winter', c=array[:,-1])
    #axs[1].scatter(array[:,2], array[:,1], s=.5)
    #axs[2].scatter(array[:,2], array[:,0], s=.5)

    # fix axis
    axs.axis('equal')
    #for ax in axs: ax.axis('equal')

if __name__=='__main__':
    try:
        frames = []
        fig, axs = plt.subplots(1,1)
        ani = FuncAnimation(fig, my_function, interval=80)
        plt.show()

    except:
        with open(f'{time.time_ns()}.pkl', 'wb') as fpkl:
            pk.dump(frames, fpkl)

    finally:
        with open(f'{time.time_ns()}.pkl', 'wb') as fpkl:
            pk.dump(frames, fpkl)
```

APÊNDICE C – Python - Processamento

Listing C.1 – Código Processamento Nuvem de Pontos

```
#%%
from TauLidarCommon.frame import FrameType, Frame
from TauLidarCamera.camera import Camera
from cv2 import transform
import matplotlib.pyplot as plt
import pickle as pk
import numpy as np
import open3d as o3d
import copy

from tqdm.auto import tqdm

treg = o3d.t.pipelines.registration
FORMAT = 'pdf'

#%%
with open('PCDs/1665258639409838000.pkl', 'rb') as fpkl:
    loaded_data = pk.load(fpkl)

#%%
def showPointCloud(frame):
    array = np.array(frame.points_3d)[:,:3]
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(array)
    #o3d.visualization.draw_geometries([pcd])
    return pcd

pcds = []
for frame in loaded_data:
    pcds.append(showPointCloud(frame))

#%% Down Sample
def down_sample(pcd):
    pcd_down = pcd.voxel_down_sample(voxel_size = 0.08)
    o3d.visualization.draw_geometries([pcd_down])
    return pcd_down

# %%
def display_inlier_outlier(pcd, ind):
    inlier_cloud = pcd.select_by_index(ind)
    outlier_cloud = pcd.select_by_index(ind, invert = True)
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    #o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

# %%
pcd_no = []
for n, pcd in enumerate(pcds):
    c1, ind = pcd.remove_statistical_outlier(nb_neighbors = 30, std_ratio = .3)
    if n % 5 == 0: display_inlier_outlier(pcd, ind)
    pcd_no.append(pcd.select_by_index(ind))

```

```
53 pcd_no_cut = []
54 for n, pcd in enumerate(pcd_no):
55     points = np.asarray(pcd.points)
56     pcdCut = o3d.geometry.PointCloud()
57     pcdCut.points = o3d.utility.Vector3dVector(points[(points[:,2]>0.63)&(points[:,0]>-.35)])
58     pcd_no_cut.append(pcdCut)
59
60 #%%
61 def alignPCD(source, target, down_sample=0.005, std=10, nb=20, ansatz=np.eye(4)):
62     source = copy.deepcopy(source)
63     target = copy.deepcopy(target)
64
65     criteria_list = [treg.ICPConvergenceCriteria(i, i, int(j)) for i,j in zip(10*np.linspace
       (-1.3,-7,100), 10*np.linspace(2,4,100))]
66     voxel_sizes = o3d.utility.DoubleVector((10**np.linspace(-0.3, -2, 100)).tolist())
67     max_correspondence_distances = o3d.utility.DoubleVector((10**np.linspace(-.22,-1.7,100)).
       tolist())
68     estimation = treg.TransformationEstimationPointToPlane()
69
70     target.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.05,
       max_nn=500))
71     source.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.05,
       max_nn=500))
72     target.orient_normals_consistent_tangent_plane(k=500)
73     source.orient_normals_consistent_tangent_plane(k=500)
74
75     temp1 = o3d.t.geometry.PointCloud().from_legacy(target)
76     temp2 = o3d.t.geometry.PointCloud().from_legacy(source)
77
78     registration_ms_icp = treg.multi_scale_icp(temp2, temp1, voxel_sizes,
79                                         criteria_list,
80                                         max_correspondence_distances,
81                                         ansatz, estimation, True)
82
83     transformation = registration_ms_icp.transformation.numpy()
84     temp2.transform(registration_ms_icp.transformation)
85     source = temp2.to_legacy()
86
87     points = np.concatenate((np.asarray(target.points), np.asarray(source.points)), axis=0)
88     target = o3d.geometry.PointCloud()
89     target.points = o3d.utility.Vector3dVector(points)
90     c1, ind = target.remove_statistical_outlier(nb_neighbors = nb, std_ratio = std)
91     target = target.select_by_index(ind)
92     target = target.voxel_down_sample(down_sample)
93     return target, transformation
94
95 #%%
96 stepSize = 10
97 final_pcd = []
98 for n in tqdm(range(0, len(pcd_no_cut), stepSize)):
99     target = pcd_no_cut[n]
100    ansatz = np.eye(4)
101    for pcd in tqdm(pcd_no_cut[n+1:n+1+stepSize], leave=False):
102        try:
103            target, ansatz = alignPCD(pcd, target, down_sample=0.01, ansatz=ansatz)
104        except:
105            continue
106    final_pcd.append(copy.deepcopy(target))
107
108 #%%
```

```
fig, axs = plt.subplots(7, 5, figsize=(10,15))
axs = axs.flatten()

for ax,pcd in zip(axs,final_pcd):
    points = np.asarray(pcd.points)
    ax.scatter(points[:,1],points[:,0],c=points[:,2],cmap='winter',s=.1)
    ax.axis('equal')
    ax.axis('off')

plt.savefig(f'pointcloud-reco1-10.{FORMAT}', bbox_inches='tight', dpi=1000)

#%%
stepSize = 5
iter_pcd = []
for n in tqdm(range(0, len(final_pcd), stepSize)):
    target = final_pcd[n]
    ansatz = np.eye(4)
    for pcd in tqdm(final_pcd[n+1:n+1+stepSize], leave=False):
        try:
            target, ansatz = alignPCD(pcd, target, down_sample=0.01, ansatz=ansatz)
        except:
            continue
    iter_pcd.append(copy.deepcopy(target))

#%%
fig, axs = plt.subplots(1, 7, figsize=(10,3))
axs = axs.flatten()

for ax,pcd in zip(axs,iter_pcd):
    points = np.asarray(pcd.points)
    ax.scatter(points[:,1],points[:,0],c=points[:,2],cmap='winter',s=.1)
    ax.axis('equal')
    ax.axis('off')

plt.subplots_adjust(hspace=0, wspace=0)
plt.savefig(f'pointcloud-reco1-5.{FORMAT}', bbox_inches='tight', dpi=1000)

#%%
target = iter_pcd[-2]
ansatz = np.eye(4)
for pcd in [iter_pcd[-1], iter_pcd[0]]:
    target, ansatz = alignPCD(pcd, target, down_sample=0.015, ansatz=ansatz)

#%%
fig, axs = plt.subplots(1, 3, figsize=(6,2))

for ang, axs in zip([0, -np.pi/2, np.pi], axs):
    R = copy.deepcopy(target).get_rotation_matrix_from_xyz((ang, 0, 0))
    points = np.asarray(copy.deepcopy(target).rotate(R, center=target.get_center()).points)
    axs.scatter(points[:,1],points[:,0],c=points[:,2],cmap='winter',s=.1)
    axs.axis('equal')
    axs.axis('off')
plt.subplots_adjust(hspace=0, wspace=0)
plt.savefig(f'pointcloud-reco1-1.{FORMAT}', bbox_inches='tight', dpi=1000)

#%%
target.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.05, max_nn
    =500))
target.orient_normals_consistent_tangent_plane(k=500)
c1, ind = target.remove_statistical_outlier(nb_neighbors = 20, std_ratio = 10)
```

```
168 target = target.select_by_index(ind)
169 target = target.voxel_down_sample(0.02)
170
171 # %% Alpha Shapes
172 tetra_mesh, pt_map = o3d.geometry.TetraMesh.create_from_point_cloud(pcd)
173 mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(pcd, 0.02, tetra_mesh,
        pt_map)
174 mesh.compute_vertex_normals()
175 o3d.io.write_triangle_mesh('alphaMesh.ply', mesh)
176 o3d.visualization.draw_geometries([mesh], mesh_show_back_face=True)
177
178 #%% Ball Pivoting
179 radii = np.linspace(0.001, 0.02, 5)
180 rec_mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting(
181     target, o3d.utility.DoubleVector(radii))
182 o3d.io.write_triangle_mesh('ballMesh.ply', mesh)
183 o3d.visualization.draw_geometries([rec_mesh])
184
185 #%% Poisson
186 with o3d.utility.VerbosityContextManager(o3d.utility.VerbosityLevel.Debug) as cm:
187     mesh, densities = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(target, depth
        =2)
188 o3d.io.write_triangle_mesh('poissonMesh.ply', mesh)
189 o3d.visualization.draw_geometries([mesh])
190
191 #%%
192 import trimesh
193 mesh = trimesh.load_mesh('alphaMesh.ply')
194 mesh.fix_normals()
195 trimesh.smoothing.filter_humphrey(mesh).subdivide().subdivide().show()
```

Referências

AHO, A. V. Algorithms for finding patterns in strings. In: *Algorithms and Complexity*. Elsevier, 1990. p. 255–300. Disponível em: <https://doi.org/10.1016/b978-0-444-88071-0.50010-2>.

AMS OSRAM GROUP. *10-Bit 360ž Programmable Magneti Rotary Encoder*. Tobelbader Strasse 30, 8141 Premstaetten, Austria-Europe, 2022. V3-00. Disponível em: <https://ams.com/documents/20143/36005/AS5040_DS000374_5-00.pdf>.

ANG, K. H.; CHONG, G.; LI, Y. Pid control system analysis, design, and technology. *IEEE transactions on control systems technology*, IEEE, v. 13, n. 4, p. 559–576, 2005.

ARDUINO. *Arduino Nano*. Via Andrea Appiani 25, 20900 MONZA MB, Italy, 2022. Disponível em: <https://docs.arduino.cc/static/07bb0689db5f51912df3f8e53e625137/A000005-datasheet.pdf>.

ATMEL. *8-bit AVR Microcontrollers*. 1600 Technology Drive, San Jose, CA 95110 USA, 2016. Disponível em: <https://www.ic-components.hk/files/47/ATMEGA328-MU.pdf>.

CANATA, T. F.; MOLIN, J. P.; SOUSA, R. V. de. A MEASUREMENT SYSTEM BASED ON LiDAR TECHNOLOGY TO CHARACTERIZE THE CANOPY OF SUGARCANE PLANTS. *Engenharia Agrícola*, FapUNIFESP (SciELO), v. 39, n. 2, p. 240–247, abr. 2019. Disponível em: <https://doi.org/10.1590/1809-4430-eng.agric.v39n2p240-247/2019>.

CRENGANIS, M.; BOLOGA, O. Pid controller for a differential steering mobile platform.

DIGNE, J. An analysis and implementation of a parallel ball pivoting algorithm. *Image Processing On Line*, v. 4, p. 149–168, 2014.

DUNKELS, A.; SCHMIDT, O.; VOIGT, T.; ALI, M. Protothreads. In: *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*. ACM Press, 2006. Disponível em: <https://doi.org/10.1145/1182807.1182811>.

FILHO, A. S. de L.; ARAÚJO, A. E. P. de. Utilização de controle pid na robótica.

FRANCA, J.; GAZZIRO, M.; IDE, A.; SAITO, J. A 3d scanning system based on laser triangulation and variable field of view. In: *IEEE International Conference on Image Processing 2005*. IEEE, 2005. Disponível em: <https://doi.org/10.1109/icip.2005.1529778>.

FREEDMAN, D.; SMOLIN, Y.; KRUPKA, E.; LEICHTER, I.; SCHMIDT, M. Sra: Fast removal of general multipath for tof sensors. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2014. p. 234–249.

GROETELAARS, N. J.; AMORÍN, A. Tecnologia 3d laser scanning: Características processos e ferramentas para manipulação de nuvens de pontos [3d laser scanning technology: characteristics, processes and point cloud tools]. CUMINCAD, 2011.

JIANG, Y.; YIN, S.; LI, K.; LUO, H.; KAYNAK, O. Industrial applications of digital twins. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, The Royal Society, v. 379, n. 2207, p. 20200360, ago. 2021. Disponível em: <https://doi.org/10.1098/rsta.2020.0360>.

KAZHDAN, M.; BOLITHO, M.; HOPPE, H. Poisson surface reconstruction. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. [S.l.: s.n.], 2006. v. 7.

LI, Y.; IBANEZ-GUZMAN, J. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, Institute of Electrical and Electronics Engineers (IEEE), v. 37, n. 4, p. 50–61, jul. 2020. Disponível em: <https://doi.org/10.1109/msp.2020.2973615>.

MADUREIRA, D. Roupa da brasileira terá que respeitar biótipos 'retângulo' e 'colher'. Folha de São Paulo, 2021. Disponível em: <https://www1.folha.uol.com.br/mercado/2021/12/roupa-da-brasileira-tera-que-respeitar-biotipos-retangulo-e-colher.shtml>.

MARGOLIS, M.; JEPSON, B.; WELDIN, N. R. *Arduino cookbook*. 3. ed. Sebastopol, CA: O'Reilly Media, 2020.

MARIN, J. L. *Implementação de filtros digitais utilizando microcontroladores*. Tese (Doutorado) — Instituto Federal de Educação, Ciência e Tecnologia da Bahia, 2014.

MATOS, B. G. G. d. *Controlador e accionador para motor DC em malha fechada*. Tese (Doutorado), 2008.

OLIVEIRA, B. Q. de; FARIAS, J. L.; FERREIRA, R. K. P.; COSTA, T. A. d. O. L.; OLIVEIRA, I. N. de; FILHO, A. C. Tipos e aplicações de sensores na robótica. *Caderno de Graduação-Ciências Exatas e Tecnológicas-UNIT-ALAGOAS*, v. 4, n. 1, p. 223–223, 2017.

RAJ, T.; HASHIM, F. H.; HUDDIN, A. B.; IBRAHIM, M. F.; HUSSAIN, A. A survey on lidar scanning mechanisms. *Electronics*, Multidisciplinary Digital Publishing Institute, v. 9, n. 5, p. 741, 2020.

ROBÓTICA, G. de. Introduçao ao arduino. *Notas de aula, Universidade Federal do Mato Grosso do Sul*, p. 10, 2012.

SALMONY, P. *Accelerometers and Gyroscopes - Sensor Fusion #1 - Phil's Lab #33*. 2021. Disponível em: <https://youtu.be/RZd6XDx5VXo>.

SALMONY, P. *Complementary Filter - Sensor Fusion #2 - Phil's Lab #34*. 2021. Disponível em: <https://youtu.be/BUW2OdAtzBw>.

SALMONY, P. *Extended Kalman Filter - Sensor Fusion #3 - Phil's Lab #37*. 2021. Disponível em: <https://youtu.be/hQUkiC5o0JI>.

SHILOH, M.; BANZI, M. *Getting started with arduino 4e*. Sebastopol, CA: Maker Media, 2022.

SIERRA, I. de S.; OKIMOTO, M. L. Proposta de protocolo de análise da malha escaneada (map). *Design e Tecnologia*, v. 11, n. 22, p. 83–92, 2021.

SOBOLEWSKI, J. S. *Cyclic Redundancy Check*. GBR: John Wiley and Sons Ltd., 2003. 476479 p. ISBN 0470864125.

TAUBIN, G.; MORENO, D.; LANMAN, D. 3d scanning for personal 3d printing. In: *ACM SIGGRAPH 2014 Studio on - SIGGRAPH '14*. ACM Press, 2014. Disponível em: <https://doi.org/10.1145/2619195.2656314>.

TEICHMANN, M.; CAPPS, M. Surface reconstruction with anisotropic density-scaled alpha shapes. In: IEEE. *Proceedings Visualization'98 (Cat. No. 98CB36276)*. [S.l.], 1998. p. 67–72.