UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Movimentação de Objetos via Internet

Trabalho de Conclusão de Curso

João Pedro Santos Quintino

São Cristóvão – Sergipe

2019

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

João Pedro Santos Quintino

Movimentação de Objetos via Internet

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(a): Antonio Ramirez Hidalgo
Coorientador(a): Ricardo José Paiva de Britto Salgueiro

São Cristóvão – Sergipe

2019

Resumo

Com a evolução da eletrônica, especialmente da microeletrônica, tornou-se possível, e cada vez mais comum, o desenvolvimento de aplicações que permitem uma atuação física de forma remota. O acesso remoto à dispositivos é uma funcionalidade não só desejável como também, às vezes, necessária, seja por dificuldade de acesso ou por segurança. É perceptível a carência de soluções tecnológicas modernas em situações que demandam controle à distância. Com o atual alcance e suporte da internet, a aplicação da mesma para o controle remoto e comunicação entre dispositivos está se tornando cada vez mais fácil e frequente. Nesse sentido, aplicações como manipulação de material radioativo, desarmamento de explosivos, exploração em áreas inacessíveis, dentre tantas outras, se beneficiam com um sistema controlado em tempo real via internet. Neste trabalho será desenvolvido um estudo de caso para movimentação de dispositivos via internet, utilizando um braço robótico para realizar movimentos determinados por um operador em tempo real.

Palavras-chave: Internet, Wi-Fi, Braço Robótico, ESP32, Sensores, MQTT.

Abstract

With the evolution of electronics, especially microelectronics, it has become possible, and increasingly common, the development of applications that allow a physical performance remotely. The remote access to devices is not only desirable but also sometimes necessary, whether due to access difficulties or to security. It is noticeable the lack of modern technological solutions in situations that require remote control. With the current reach and support of the internet, its application for remote control and communication between devices, is becoming increasingly easy and frequent. In this way, applications such as manipulation of radioactive material, disarming explosives, exploration of inaccessible areas, among many others, benefit from a real-time controlled system via the Internet. In this work, a case study for moving objects through the internet will be developed, using a robotic arm to perform movements determined by an operator in real time.

Keywords: Internet, Wi-Fi, Robotic Arm, ESP32, Sensors, MQTT.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AD	Analógico-Digital
ADC	Analog-to-Digital Converter, em português, “Conversor Analógico-Digital”
BLE	Bluetooth Low Energy, em português, “Bluetooth de baixa energia”
DA	Digital-Analógico
DAC	Digital-to-Analog Converter, em português, “Conversor Digital-Analógico”
DMP	Digital Motion Processor, em português, “Processador digital de movimento”
GPIO	General Purpose Input/Output, em português, “Entrada/Saíde de propósito geral”
I^2C	Inter-Integrated Circuit, em português, “Circuito Inter-Integrado”
IDE	Integrated Development Environment, em português, “Ambiente de desenvolvimento integrado”
IoT	Internet of Things, no português, “Internet das Coisas”
LTE	Long Term Evolution, em português, “Evolução de Longo Prazo”
MCU	Micro Control Unit, em português, “Unidade microcontrolada”
MQTT	Message Queuing Telemetry Transport
PWM	Pulse Width Modulation, em português, “Modulação por Largura de Pulso”
SPI	Serial Peripheral Interface, em português, “Interface Serial Periférica”

Sumário

1 Introdução

Com o avanço da tecnologia tornou-se cada vez mais frequente o uso de máquinas em setores como a indústria e a construção civil. A partir da década de 1970, quando surgiram os primeiros microcontroladores, ficou cada vez mais fácil e barata a implantação de sistemas controlados, fazendo com que esse uso se expandisse além das áreas industriais.

Juntamente com o surgimento dos microcontroladores, a internet surgiu, na década de 1970, e se expandiu, principalmente na década de 1990. Atualmente, a rede de internet alcança grande parte da população mundial. De acordo com (SANOU, 2016) no referido ano de publicação, tinha-se que 95% da população global vivia em uma área coberta pela rede de internet, conforme ilustra a Figura 1, representando essa evolução de 2007 até 2016.

O gráfico da Figura 1 mostra a quantidade de pessoas que vivem em uma área coberta por tecnologias de redes móveis. Em 2007 cerca de 5,2 bilhões de pessoas viviam em uma área coberta por redes móveis, dessas, cerca de 4,2 bilhões contavam apenas com a tecnologia 2G e cerca de 1 bilhão já contava também com a tecnologia 3G. Em 2016, cerca de 6,5 bilhões de pessoas viviam em uma área coberta por redes móveis, onde dessas, cerca de 1 bilhão contava apenas com a tecnologia 2G, cerca de 2 bilhões contava além do 2G com a tecnologia 3G e cerca de 3,5 bilhões vivia em uma área coberta também pela tecnologia LTE (4G ou superior).

O surgimento dos microcontroladores permitiu, entre outras coisas, que fossem desenvolvidos sistemas de manipulação de objetos controlados por humanos, possibilitando que pessoas controlem as máquinas em situações que necessitam de mais força, precisão ou em condições insalubres.

Em algumas aplicações, como manipulação de elementos radioativos e desarmamento de bombas, é desejável que não haja presença humana, pois são atividades que trazem um alto risco à vida, mas ao mesmo tempo é necessário que alguma ação seja tomada.

O controle de objetos via internet proporciona a interação com ambientes hostis sem

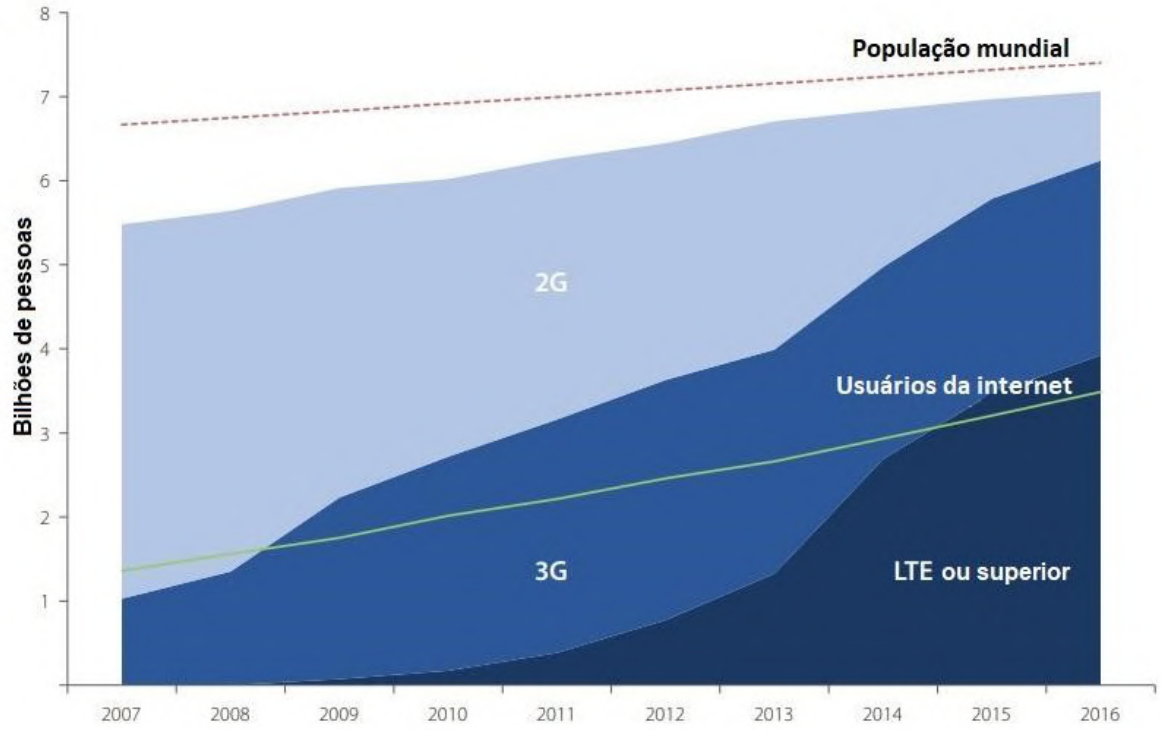

Figura 1 – Gráfico da cobertura de redes móveis e tecnologias em evolução

Fonte: (SANOU, 2016)

que haja a necessidade da presença humana no local, permitindo que ações sejam tomadas à distância, controlando uma situação de perigo sem o risco envolvido em uma ação humana.

O fato de a internet ser tão acessível atualmente, e o consequente barateamento do seu acesso, bem como de dispositivos de controle, como os microcontroladores, faz com que esse tipo de controle possa ser realizado também em ambientes industriais, comerciais, e até mesmo residenciais.

Considerando-se o barateamento do acesso e a imensa área de alcance da internet, observa-se uma ampla vantagem com relação a tecnologias de acesso remoto de curta distância, como o bluetooth, pois a movimentação de atuadores via internet propicia ao operador mais segurança e tranquilidade para efetuar a sua tarefa, além de poder executá-la praticamente de qualquer lugar do planeta.

Neste trabalho será desenvolvido um sistema de manipulação remota de objetos através de uma conexão de internet, o que possibilita a sua utilização em aplicações como: movimentação de robôs, controle de manipuladores, controle de motores, movimentação de carga, acionamento de alarmes e atuadores, entre outras.

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver um sistema que possibilite ao usuário o controle de objetos utilizando a internet.

1.1.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- O desenvolvimento de um estudo de caso para o sistema de controle de movimento via internet, utilizando-se de um braço robótico como dispositivo controlado e uma luva equipada com sensores para a detecção de movimentos da mão, que será utilizada como controle. O sistema deverá possuir um monitoramento do ambiente onde o braço robótico estará instalado, utilizando uma câmera IP;
- Aprender o funcionamento e utilização do ESP32
- Aprender a fazer a comunicação via internet com o ESP32;
- Estudar a aplicação de câmera IP para monitoramento de ambiente;
- Estudar o funcionamento dos servo-motores que movimentam o braço robótico;
- Estudar o uso dos sensores de captação de movimento da luva;
- Desenvolver um software para a captura de movimentos da luva;
- Desenvolver um software para movimento do braço robótico;
- Projetar a luva para captação de movimentos da mão;
- Estudar o funcionamento e utilização do módulo MPU-6050 e do sensor flex, utilizados para captura de movimentos da mão;
- Revisão da linguagem Java para implementação de software de exibição da câmera e exibição de movimentos da luva;
- Aprender o protocolo MQTT e a arquiterura Publish-Subscribe para comunicação via internet;

1.1.2 Metodologia

No processo de desenvolvimento deste trabalho são realizadas pesquisas, utilizando como referência materiais das mais diversificadas fontes, tais como: livros, artigos científicos, dissertações de mestrado, sites, trabalhos acadêmicos e o conhecimento adquirido durante os anos de graduação.

Para o melhor entendimento do funcionamento dos componentes de hardware são feitos estudos de datasheets dos componentes, pesquisa em sites dos fabricantes e de terceiros, pesquisas de vídeos demonstrativos e explicativos e também pesquisa em livros.

Para o projeto do software é levado em conta o conhecimento em linguagens de programação adquirido na graduação, a aplicabilidade das linguagens e também a afinidade com os ambientes e linguagens de programação. Neste trabalho são utilizadas diversas figuras ilustrativas e citações com o objetivo de trazer confiabilidade e melhor compreensão do assunto.

2

Trabalhos Relacionados

O objetivo deste capítulo é contextualizar o trabalho proposto com trabalhos contemporâneos e relevantes a fim de confirmar a importância do tema proposto. São apresentados em seções a proposta e os resultados de cada trabalho.

O trabalho de (SILVA, 2002) merece destaque por apresentar um sistema embutido baseado em microcontrolador em uma plataforma prática e adequada para a interconexão de dispositivos elétricos em redes TCP/IP. Porém, o mesmo não foi citado nas seções por não ser atual.

Os trabalhos de (OLIVEIRA, 2017) e (SANTOS, 2017) também merecem citação, apesar de terem um fim específico que destoa do tema deste trabalho. Ambos tratam da aplicação de um sistema de automação residencial utilizando o microcontrolador ESP8266. Os trabalhos utilizam um sistema via internet para comunicação com o dispositivo controlador (smartphone, tablet, computador, entre outros.

Outros trabalhos foram considerados, mas não estão descritos nas seções a seguir, ou por não serem atuais ou por focarem em aplicações específicas que fogem do tema proposto.

2.1 Braço Robótico Operado Através de Movimentos Reais de um Braço Humano

No trabalho realizado por (OLIVEIRA; SILVA, 2016) é desenvolvido um sistema de controle que utiliza como sinal de entrada a leitura de sensores, anexados a um braço humano, para manipular os movimentos de um braço robótico.

A solução conta com um braço robótico, que possui 4 graus de liberdade, com sensores flexíveis e sensores inerciais (acelerômetro e giroscópio), e tem como objetivo a implantação de um sistema local para controle de movimento.

Como resultado, foi obtido com êxito um sistema útil para manipulação de pequenos objetos, possibilitando um controle estável. O sistema aplicado na manipulação mostrou-se capaz de realizar a leitura através dos movimentos reais do braço humano e aplicar o controle realizando os movimentos desejados.

2.2 Controle de um Braço Robótico Via Web

Neste trabalho é apresentada uma solução para o controle de um braço robótico, utilizando sinais recebidos de uma página web como entrada para o controle do mesmo. O autor utiliza a plataforma BeagleBone Black para exibição da página web e envio dos sinais para o braço robótico.

Esta solução, proposta por (SILVA et al., 2016) conta com um braço robótico, que possui 4 graus de liberdade, com um microcontrolador BeagleBone Black, o qual será o cérebro da aplicação, e conta também com uma página web desenvolvida com as linguagens HTML e PHP. A página web apresenta botões, os quais representam o controle de direção de cada articulação do braço robótico.

Como resultado, foi observada com êxito a aplicabilidade de controle do dispositivo via internet, utilizando controle em tempo real. Além de sua aplicabilidade, também observam-se vantagens, como a facilidade de acesso, a forma intuitiva como o sistema é utilizado e a portabilidade, visto que uma página web pode ser acessada praticamente de qualquer lugar do planeta.

2.3 Estudo de Caso: Utilização do Arduino Para Um Sistema de Controle Remoto de Dispositivos via Internet

Neste trabalho, proposto por (TOLENTINO; TSUKAMOTO; NOMURA, 2013), é proposto o desenvolvimento de um sistema de monitoramento remoto da temperatura de um ambiente, com a funcionalidade de enviar uma mensagem de alerta ao usuário, via Internet, quando a temperatura exceder o limite pré-estabelecido. Além disso, o sistema proposto deve aceitar comandos do usuário, recebidos via Internet, para que seja feito o controle de temperatura do ambiente.

O sistema conta com uma implementação de hardware utilizando um Arduino UNO e um módulo Ethernet Shield para prover ao Arduino o acesso à internet. O sistema conta também com a implementação do software responsável pela transmissão e recepção de dados via internet.

Como resultado, foi obtida uma plataforma versátil e de baixo custo, tornando possível o monitoramento da temperatura ambiente e o alerta via e-mail, constituindo, assim, um sistema bastante útil e com muitas aplicações tanto em segmentos comerciais e industriais quanto no uso doméstico.

2.4 PROTOBOT - Protótipo De Braço Robótico Comandado Por Comunicação Sem fio

Este trabalho, proposto por (VANÇAN, 2017), trata-se de uma plataforma educacional, voltada ao ensino de robótica e programação para ensinos fundamental médio e técnico. O trabalho constitui um guia explicativo para a construção, programação e execução de um braço robótico comandado por controle sem fio.

No trabalho, é feita a construção lógica de funções que controlam a posição dos servo-motores, buscando obter um bom controle energético, a fim de permitir um modo de operação com economia de energia. A comunicação remota do dispositivo é feita com a utilização de rádio NRF24L01. O microcontrolador utilizado para realização do processamento foi um Arduino UNO.

O trabalho gera como resultado um tutorial explicativo do processo de construção e programação de um braço robótico. Alguns problemas que ocorreram no trabalho foram: sincronismo, bugs indefinidos na utilização do rádio, e a dificuldade de encontrar os componentes e materiais necessários.

2.5 Braço Articulado Com Controle Proporcional De Movimento Comandado Via Bluetooth Por Um Aplicativo Desenvolvido Para Plataformas Android

Este trabalho propõe um sistema de controle de movimentos de um braço robótico via celular com comandos seriais enviados via Bluetooth.

Neste trabalho, proposto por (FERREIRA; ALVES, 2013), é feito um controle proporcional de movimento através de um aplicativo desenvolvido para a plataforma Android. O braço robótico utilizado no trabalho conta com 6 graus de liberdade. No trabalho é utilizado um microcontrolador MSP430 com o auxílio de um módulo Bluetooth para movimentação dos servo-motores do braço robótico, utilizando PWM. É utilizada também a plataforma App Inventor, do Google, para criação do aplicativo para Android, pois o mesmo não exige habilidades de programação.

Como resultado, foram desenvolvidos com êxito, um aplicativo Android para controle do sistema, a estrutura mecânica do braço robótico e o software de controle do microcontrolador.

3

Protocolo MQTT

O MQTT (do inglês, Message Queuing Telemetry Transport) é um protocolo de mensagem desenvolvido pela IBM por volta dos anos 90. Foi desenvolvido com base na pilha TCP/IP e tem suporte para a comunicação assíncrona entre as partes. Segundo (YUAN, 2017), uma das vantagens para o uso do MQTT é que, por ser um protocolo de sistema de mensagens assíncrono, desacopla o emissor e o receptor da mensagem tanto no espaço quanto no tempo e, portanto, é escalável em ambientes de rede que não são confiáveis. Essa característica faz com que ele seja bastante utilizado em aplicações de IoT.

Um dos principais diferenciais do MQTT com relação ao HTTP, que é o protocolo mais comum em conexões de internet, é que o mesmo funciona sobre o paradigma Publish-Subscribe, enquanto o HTTP utiliza o Request-Response em que o cliente faz uma requisição ao servidor e este responde a solicitação do cliente.

3.1 O paradigma Publish-Subscribe

No paradigma Publish-Subscribe existem três elementos principais para que haja a comunicação. Estes elementos são: Publisher (Publicante), Subscriber (Assinante) e Broker (Corretor).

O Broker funciona como um servidor, que irá intermediar a comunicação entre o Publisher e o Subscriber. Ele fornece um canal para que as partes se comuniquem através de tópicos e mensagens. Um tópico é um identificador de um canal que será utilizado para postagem de mensagens, nele podem ser postadas diversas mensagens. O Publisher é o dispositivo que faz a publicação das mensagens. Este se conecta ao Broker, através da sua URL e porta, e faz a publicação de uma mensagem em um tópico. Um Publisher pode publicar diversos tópicos no Broker e várias mensagens em cada tópico.

O Subscriber, também conhecido como cliente MQTT, é o elemento que faz a leitura

das mensagens. Ele se conecta ao Broker e informa quais tópicos deseja assinar. Desta forma, quando um cliente MQTT estiver assinando um tópico, sempre que o Publisher publicar alguma mensagem nesse tópico, o Subscriber receberá esta mensagem.

Em uma conexão via MQTT um mesmo dispositivo pode ser um Publisher, um Subscriber, ou os dois ao mesmo tempo. A figura 2 exemplifica o modo como ocorre a troca de mensagens no paradigma Publish-Subscribe.

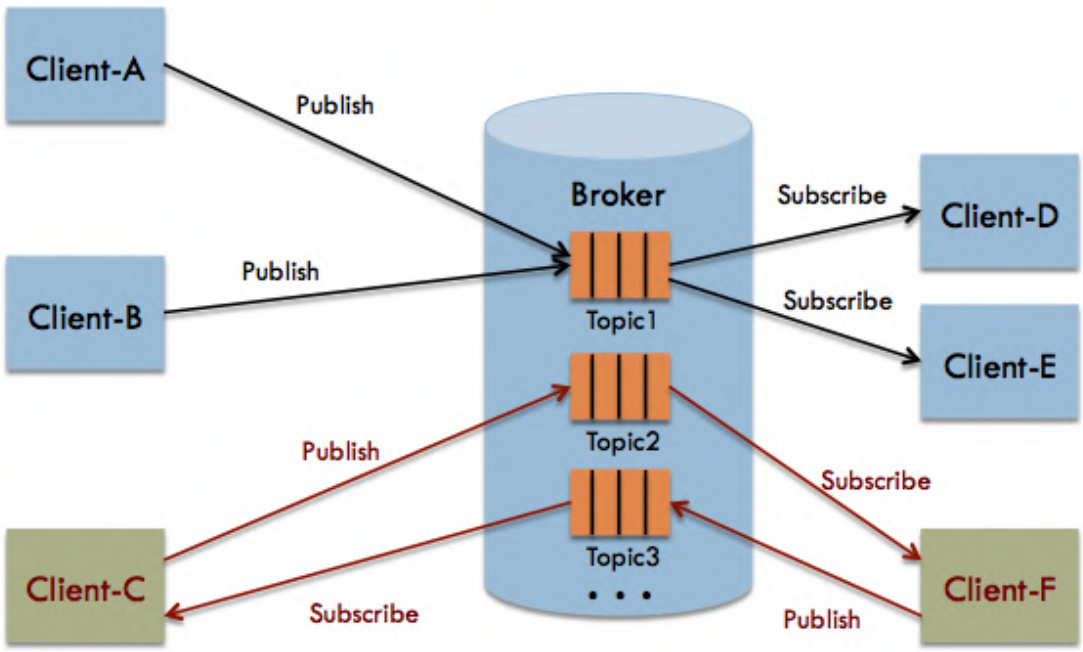

Figura 2 – Troca de mensagens no Publish-Subscribe

Fonte: (FISCHER, 2016)

Na figura 2 os clientes A e B operam como publishers, os clientes D e E operam como subscribers, e os clientes C e F operam em ambas as funções.

4

Projeto de Sistema de Hardware

Para exemplificar o funcionamento do sistema de movimentação, bem como as suas conexões, será feito o uso de diagrama de blocos, a fim de tornar explícito e intuitivo o funcionamento do sistema de hardware utilizado.

4.1 Diagrama de blocos do sistema

Na Figura 3 estão ilustradas todas as conexões entre os blocos do sistema de hardware, contendo todos os dispositivos que serão utilizados na montagem do sistema de movimentação. O sistema é formado por dois blocos principais. O primeiro bloco, chamado Bloco Central, corresponde à operação local do sistema, nele estão o operador, a luva que captará os movimentos, um computador e um roteador. No segundo bloco, chamado Bloco Remoto, se encontram o braço robótico, equipado com um microcontrolador, uma câmera para monitoramento e um roteador. A conexão entre os dois blocos principais será feita via internet.

4.1.1 Bloco Central

Este bloco é responsável pela captura dos movimentos da mão, transmissão dos dados e recepção das imagens da câmera do bloco Remoto. As conexões internas do bloco Central estão ilustradas na Figura 4.

Dentro do Bloco Central temos um computador, um roteador, uma luva contendo dois sensores e um microcontrolador.

O computador fará a interface entre o usuário e o resto do sistema através de um software, desenvolvido em Java, que receberá via internet as informações do microcontrolador embutido na luva e exibirá os movimentos realizados em interface gráfica. Além disso, o computador

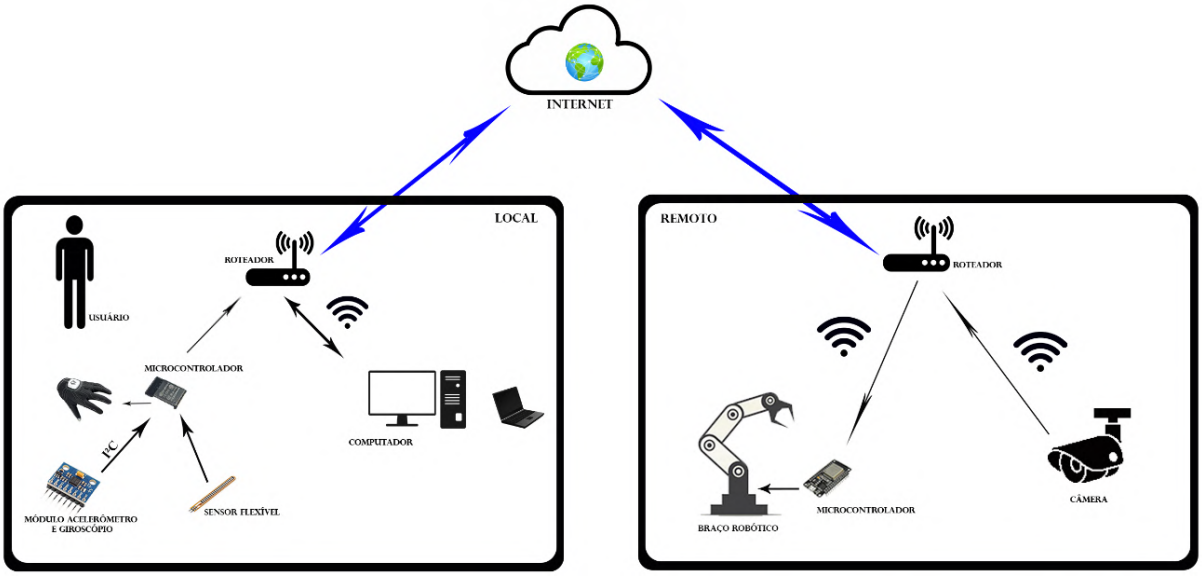

Figura 3 – Diagrama de blocos do sistema de hardware

Fonte: ELABORADA PELO AUTOR

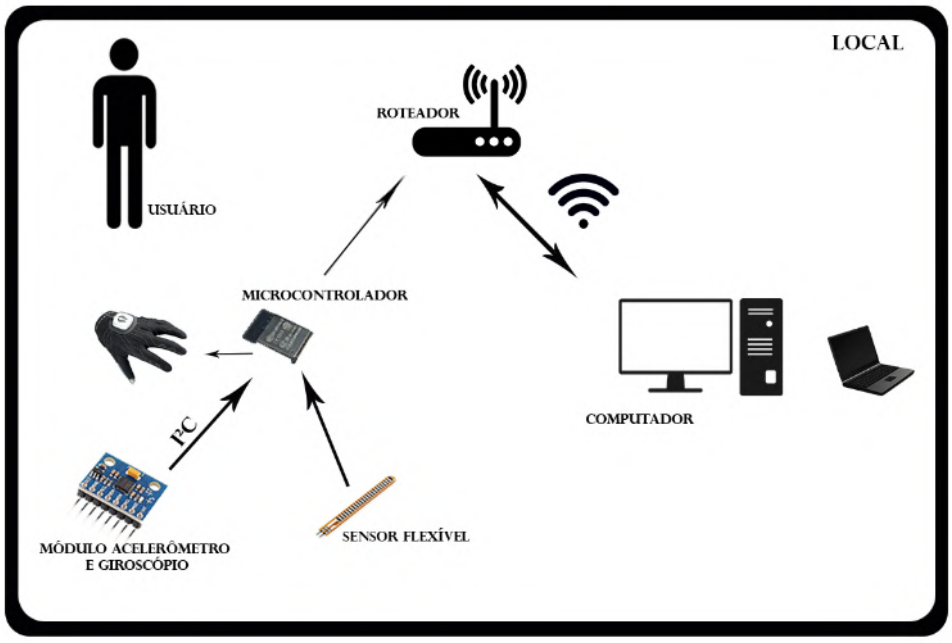

Figura 4 – Bloco Central

Fonte: ELABORADA PELO AUTOR

receberá do roteador as imagens da câmera contida no Bloco Remoto. O computador deverá estar previamente conectado ao roteador.

O roteador fará a comunicação com o Bloco Remoto utilizando uma conexão de internet para efetuar o envio dos dados da movimentação e o recebimento das imagens da câmera. Para executar a sua tarefa, o roteador precisa estar conectado à uma rede de internet.

O microcontrolador do bloco Central enviará os dados de movimentação para o bloco Remoto através de um broker MQTT. O microcontrolador do bloco Remoto fará a leitura dos dados no broker MQTT à medida que os dados forem sendo postados. Dessa forma ocorre a comunicação entre dois dispositivos em redes separadas. Essa comunicação está ilustrada Figura 5.

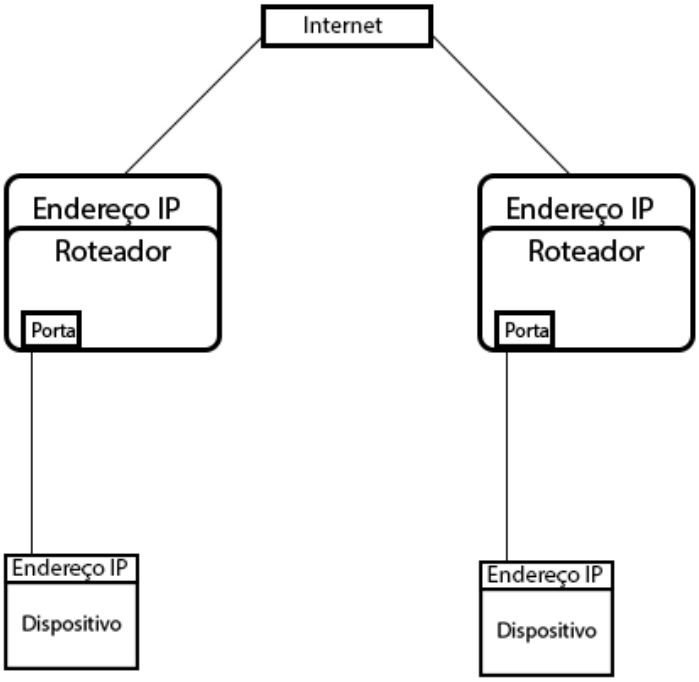

Figura 5 – Conexão entre duas redes separadas

Fonte: ELABORADA PELO AUTOR

A luva, que será responsável por capturar os movimentos do usuário, servirá de suporte para o microcontrolador e os sensores. Um dos sensores contém um acelerômetro e um giroscópio embutidos, estes sensores juntos fornecerão a informação de ângulo e movimentação da mão. O outro é um sensor flexível, que tem sua saída alterada conforme é flexionado, esse sensor fornecerá a informação de abertura ou fechamento da mão.

O microcontrolador, anexado à luva, processará os dados dos sensores e transformará em informações de movimentação. Para isso é necessário um cálculo trigonométrico para obtenção dos ângulos do acelerômetro e uma integração discreta para obtenção dos ângulos do giroscópio. Em seguida é feita a fusão dos dois sensores utilizando a técnica de filtro complementar, onde dados de dois ou mais sensores são combinados a fim de proporcionar melhor precisão no valor medido.

Além disso, o microcontrolador também fará a comunicação entre o sistema da luva e o computador utilizando o mesmo canal que será enviado para o bloco Remoto.

A luva conta também com uma fonte de alimentação para o microcontrolador e os sensores. O esquema eletrônico da Figura 6 ilustra a conexão entre os sensores e o microcontrolador

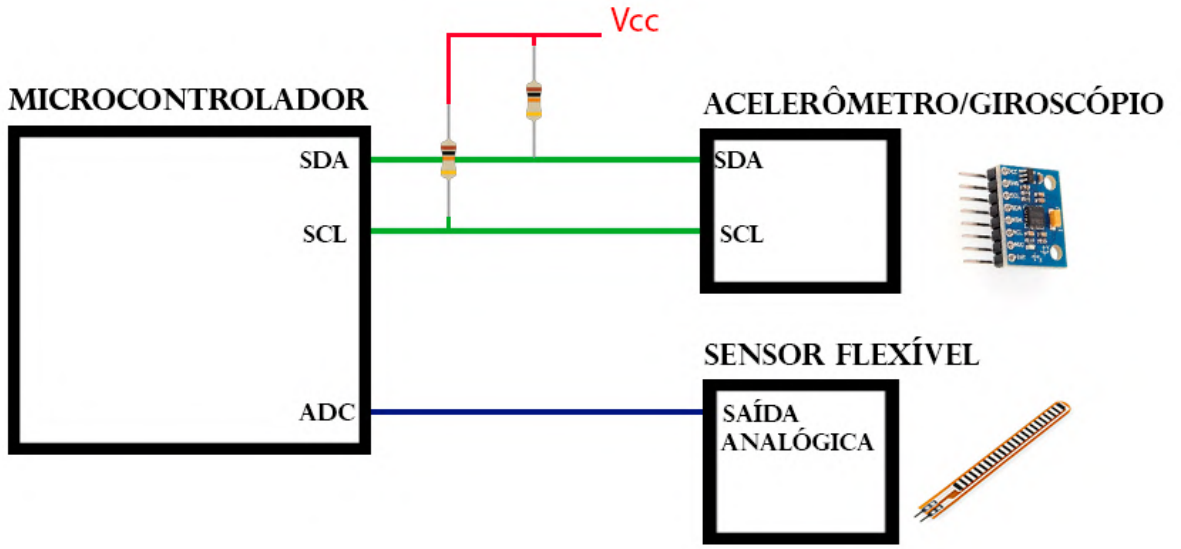

Figura 6 – Esquema eletrônico dos componentes da luva

Fonte: ELABORADA PELO AUTOR

da luva.

A comunicação entre o microcontrolador e o acelerômetro/giroscópio é feita através do protocolo síncrono I^2C, que conta com duas conexões: SDA, para transmissão de dados, e SCL, para clock de sincronização.

O sensor flexível é composto por uma resistência, que é alterada conforme o seu uso. Assim, é colocada uma tensão em um dos terminais do sensor e no outro terminal é colocado um resistor, a fim de provocar uma divisão de tensão. Essa divisão é diretamente proporcional ao valor de resistência do sensor, portanto a variação de resistência do sensor resultará em uma tensão analógica a ser lida pelo microcontrolador, que precisa de um conversor AD para realizar a leitura.

4.1.2 Bloco Remoto

Este bloco é responsável pelo recebimento das informações de movimentação e envio das imagens da câmera para o Bloco Central e também pela movimentação do sistema controlado.

Neste bloco estão contidos um roteador, um microcontrolador, um braço robótico (que será o nosso estudo de caso) e uma câmera com conexão Wi-Fi. As conexões internas do Bloco Remoto estão ilustradas na Figura 7.

O roteador fará a conexão com o Bloco Central pela internet, obtendo as informações de movimentação e enviando as imagens da câmera. Para isso, o roteador precisa estar conectado à internet.

O microcontrolador estará conectado ao roteador por Wi-Fi e será responsável por processar as informações de movimentação, transformando-as em sinais de PWM (Pulse Width

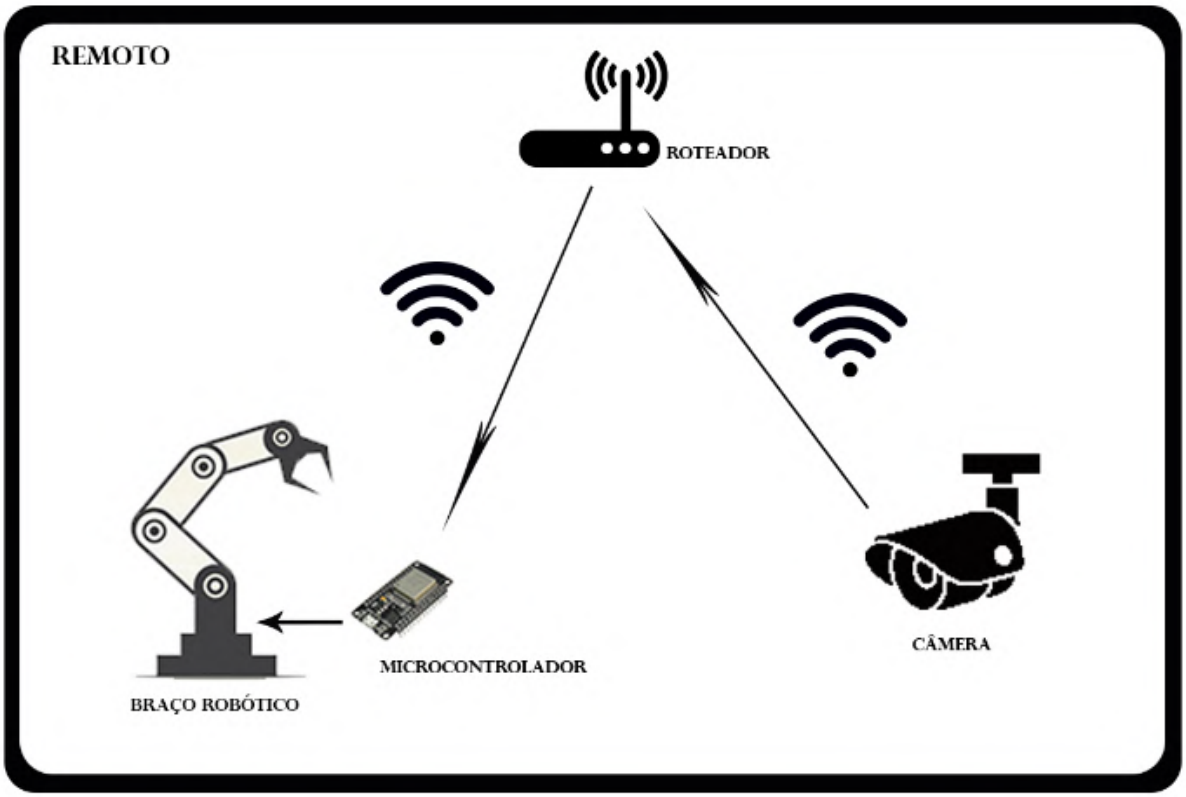

Figura 7 – Bloco Remoto

Fonte: ELABORADA PELO AUTOR

Modulation – Modulação por largura de pulso) para o braço robótico. Para isto, é feito o uso da biblioteca Servo do ambiente de desenvolvimento do Arduino, que irá converter uma informação de ângulo, variando de 0º a 180º, em uma informação de Duty Cycle (Tempo que o sinal PWM é positivo com relação ao período do sinal), variando de 3% a 12%.

Também é feito um cálculo para determinar a posição da garra, utilizando-se da informação do sensor Flex. Considerando um conversor AD de 12 bits, a entrada lida do sensor Flex varia de 0 a 4095, desta forma é feito o cálculo segundo a equação 4.1, onde flex é o valor medido do sensor Flex, para converter o intervalo de variação para um de 0 a 180.

$$flexao = flex \times (\frac{180}{4095}) \tag{4.1}$$

O braço robótico recebe as informações do microcontrolador e executa os movimentos, através de servo-motores.

A câmera estará conectada ao roteador por Wi-Fi e será responsável pelo monitoramento do ambiente onde estará instalado o braço robótico, para que possa ser visualizada a ação que se deseja efetuar.

O esquema eletrônico da Figura 8 ilustra a conexão entre o microcontrolador e o braço robótico.

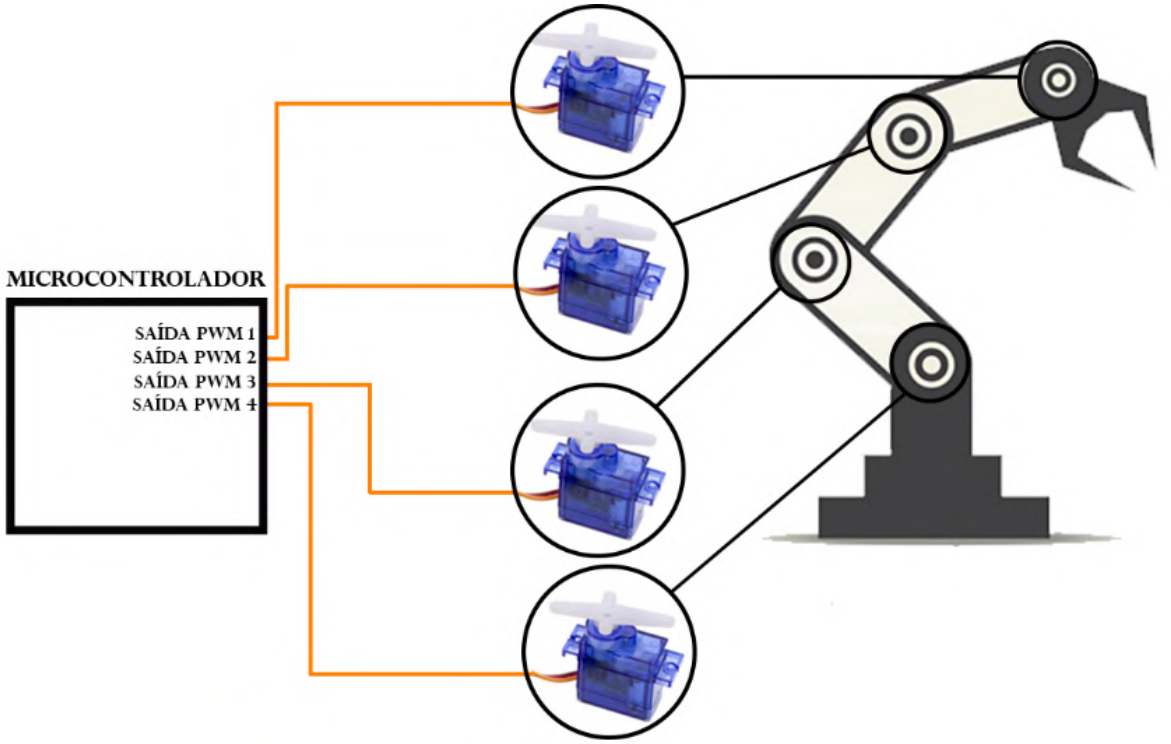

Figura 8 – Esquema eletrônico dos componentes do braço robótico

Fonte: ELABORADA PELO AUTOR

O braço robótico se movimenta através de servo-motores. Estes funcionam com uma entrada de sinal de PWM para determinar o ângulo de rotação dos motores, portanto, o microcontrolador gera esses sinais de PWM para executar os movimentos do braço robótico.

5

Descrição dos Componentes de Hardware

Para tornar claro o desenvolvimento do sistema, será feita a descrição de todos os componentes de hardware utilizados no projeto.

5.1 Roteador

O roteador é um dos dispositivos mais importantes do sistema, pois é o responsável por interligar a central de processamento local com o ambiente remoto, tornando possível todo o controle via internet.

O roteador é um dispositivo que interliga redes separadas e encaminha pacotes de dados entre essas redes. Um roteador é conectado a duas ou mais linhas de dados de redes diferentes. Quando um pacote de dados chega em uma das linhas, o roteador lê a informação de endereço no pacote para determinar o seu destino final. Em seguida, usando a informação na sua tabela de roteamento, ele direciona o pacote para a próxima rede em sua viagem. Os roteadores são os responsáveis pelo "tráfego"na Internet. Um pacote de dados é normalmente encaminhado de um roteador para outro através das redes que constituem a rede até atingir o destino. E, portanto, o roteador é tipicamente um dispositivo da camada 3 (camada de rede) do Modelo OSI. (MORIMOTO, 2011)

Uma visão de alto nível da arquitetura de um roteador é mostrada na Figura 9.

Segundo (KUROSE J; ROSS, 2013), é possível identificar quatro componentes da arquitetura de um roteador:

- Portas de entrada: Realiza funções da camada física de terminar um enlace físico de entrada em um roteador. Realiza também funções da camada de enlace necessárias para interoperar com as funções da camada de enlace do outro lado do enlace de entrada. Exerce ainda a

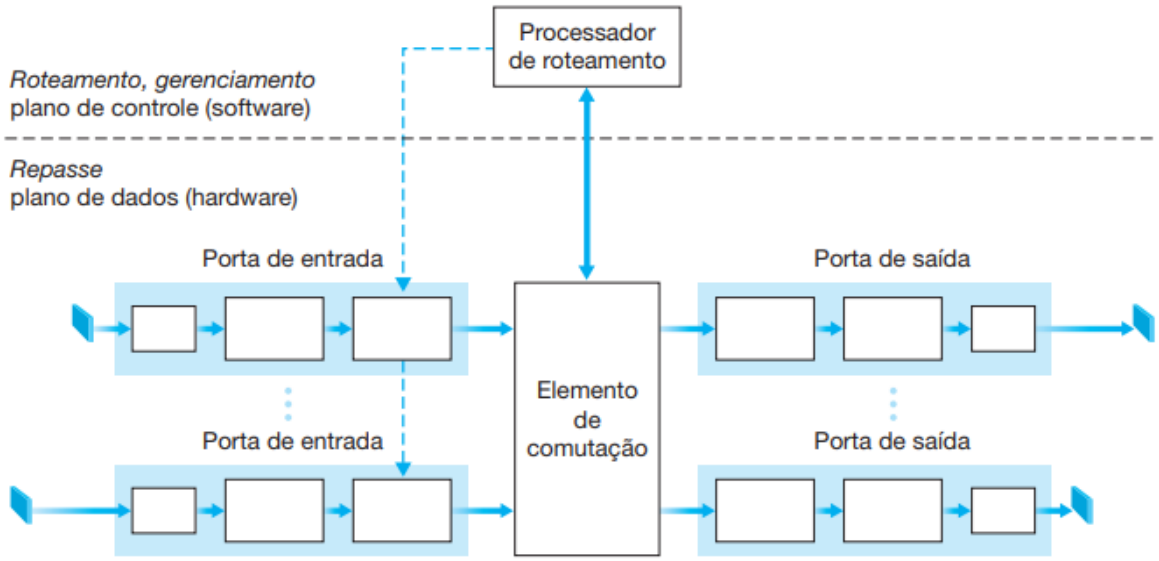

Figura 9 – Arquitetura de um roteador

Fonte: (KUROSE J; ROSS, 2013)

função de exame, onde a tabela de repasse é consultada para determinar a porta de saída do roteador à qual um pacote que chega será repassado por meio do elemento de comutação;

- Elemento de comutação: É responsável por conectar as portas de entrada do roteador às suas portas de saída;
- Portas de saída: Uma porta de saída armazena os pacotes que foram repassados a ela através do elemento de comutação, e então, os transmite até o enlace de saída, realizando as funções necessárias da camada de enlace e da física;
- Processador de roteamento: O processador de roteamento executa os protocolos de roteamento, mantém as tabelas de roteamento e as informações de estado do enlace, e calcula a tabela de repasse para o roteador.

5.2 Sensores

Os sensores são responsáveis por fazer a leitura de movimento da mão e transformar essa informação em sinais eletrônicos, que serão lidos pelo microcontrolador. No projeto será utilizado um módulo contendo acelerômetro e giroscópio e será usado também um sensor flex.

5.2.1 Módulo MPU-6050 - Acelerômetro e giroscópio

O MPU-6050, exibido na Figura 10, é um dispositivo criado pela Invensense que contém sensores de acelerômetro, giroscópio e termômetro. Além disso, contem também um processa-

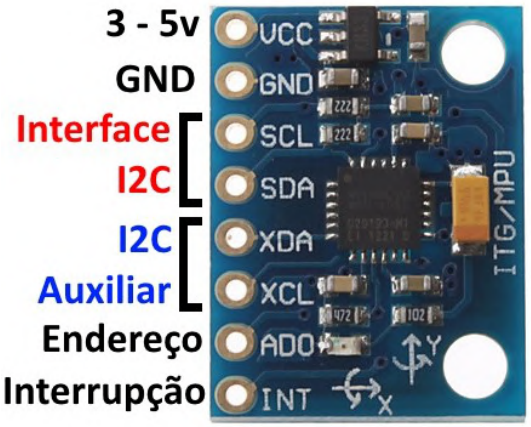

Figura 10 – Módulo MPU-6050

Fonte: (THOMSEN, 2014)

dor digital de movimento (DMP – Digital Motion Processor), responsável por fazer cálculos complexos com os sensores e cujos dados podem ser usados para sistemas de reconhecimento de gestos, navegação, jogos e diversas outras aplicações.

A comunicação do MPU6050 com o microcontrolador usa a interface I^2C, por meio dos pinos SCL e SDA do sensor. O módulo possui ainda pinos de I^2C auxiliares, que permitem a conexão de outro dispositivo, como um magnetômetro por exemplo, através dos pinos XDA e XCL. A alimentação do módulo pode variar entre 3V e 5V, mas para melhores resultados e precisão recomenda-se utilizar 5V.

Por padrão, o endereço I^2C do módulo é 0x68, mas o mesmo possui um pino AD0 para alterar o endereço para 0x69 caso o pino esteja acionado com 5V. Essa mudança de endereço permite que dois módulos MPU-6050 sejam conectados em um mesmo circuito.

O MPU-6050 possui um pino de interrupção, que permite que o usuário faça leituras apenas quando um dado novo estiver disponível, ou seja, quando as informações mudarem. Desta forma reduz-se o uso de processamento.

5.2.1.1 Acelerômetro

O sensor acelerômetro mede aceleração nos eixos x, y e z. Utilizando a informação de aceleração nesses eixos é possível obter a informação de inclinação nos eixos x e y através de manipulação trigonométrica, utilizando as equações 5.1 e 5.2.

Considerando que o acelerômetro estará em repouso, então, a única força que está agindo sobre ele é a força da gravidade. Dessa forma, a medida apresentada pelo acelerômetro corresponde à uma decomposição da aceleração da gravidade nos três eixos do acelerômetro. Já que a aceleração da gravidade é constante, a variação na medida de aceleração corresponderá a uma variação no ângulo de inclinação do acelerômetro. A Figura 11 mostra essa decomposição em um cenário de 2 dimensões.

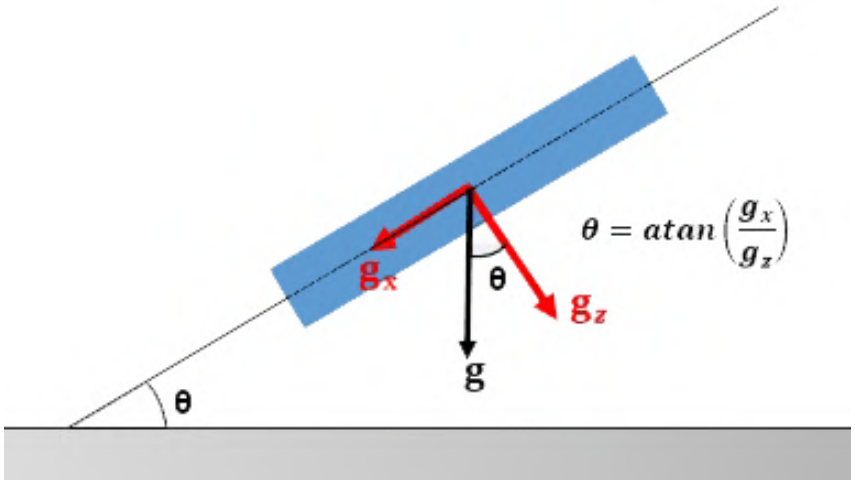

Figura 11 – Decomposição da aceleração da gravidade em cenário 2D

Fonte: (NAYLAMP, 2016)

Porém, como o cenário real é em 3D é preciso levar em consideração a decomposição também no eixo y. Desse modo chega-se às equações 5.1 e 5.2 para os eixos x e y, respectivamente.

$$\theta_x = \arctan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right) \times \left(\frac{180}{\pi}\right) \tag{5.1}$$

$$\theta_y = \arctan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right) \times \left(\frac{180}{\pi}\right) \tag{5.2}$$

Onde θ_x é a inclinação no eixo x, θ_y é a inclinação no eixo y, ambos em graus. a_x, a_y e a_z são as informações de aceleração nos eixos x, y e z, respectivamente.

5.2.1.2 Giroscópio

O sensor giroscópio mede velocidade angular nos eixos x, y e z. Para obter a informação de posição angular é necessário fazer uma multiplicação da velocidade angular pelo tempo, como sugere a equação 5.3, onde θ é o ângulo, ω é a velocidade angular, e t é o tempo.Desta forma, é possível obter a informação de ângulo dos três eixos utilizando-se de uma integração discreta, conforme as equações 5.4, 5.5 e 5.6.

$$\theta = \omega \times t \tag{5.3}$$

$$\theta_{xg} = (g_x \times dt + \theta_{xg\_prev}) \tag{5.4}$$

$$\theta_{yg} = (g_y \times dt + \theta_{yg\_prev}) \tag{5.5}$$

$$\theta_{zg} = (g_z \times dt + \theta_{zg\_prev}) \tag{5.6}$$

Onde θ_{xg}, θ_{yg} e θ_{zg} são, respectivamente, a rotação em torno dos eixos x, y e z. g_x, g_y e g_x são as velocidades angulares dos eixos x, y e z, respectivamente. θ_{xg_prev}, θ_{yg_prev} e θ_{zg_prev} são as informações anteriores de rotação nos três eixos. E dt é o tempo em segundos entre uma medida e outra.

5.2.2 Sensor Flex

O sensor flexível (ou sensor Flex), exibido na Figura 12 é um sensor resistivo que tem o seu valor de resistência alterado conforme vai sendo dobrado. Em repouso, o sensor apresenta uma resistência fixa, e, ao ser dobrado a resistência diminui ou aumenta, a depender do sentido da flexão. Uma ilustração da variação de resistência pode ser observada na Figura 13.

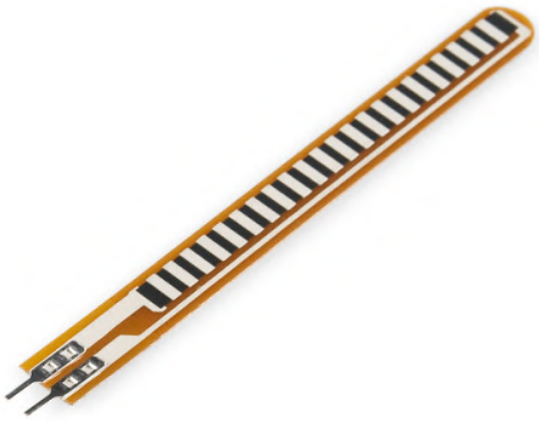

Figura 12 – Sensor Flex

Fonte: (SPARKFUN, 2019)

Como a saída desse sensor é uma variação de resistência, é necessário utilizar um circuito com ele, para transformar a informação de resistência em informação de tensão, objetivando medir essa tensão com o microcontrolador. O circuito utilizado para medir tensão em um sensor Flex é um divisor resistivo, exibido na Figura 14.

A tensão pode ser medida utilizando a equação 5.7.

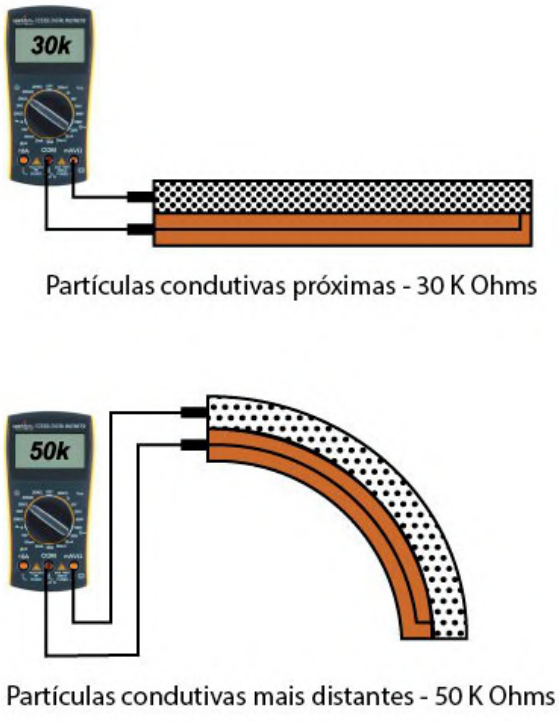

Figura 13 – Variação de resistência sofrida pelo sensor Flex durante dobramento

Fonte: ADAPTADA PELO AUTOR

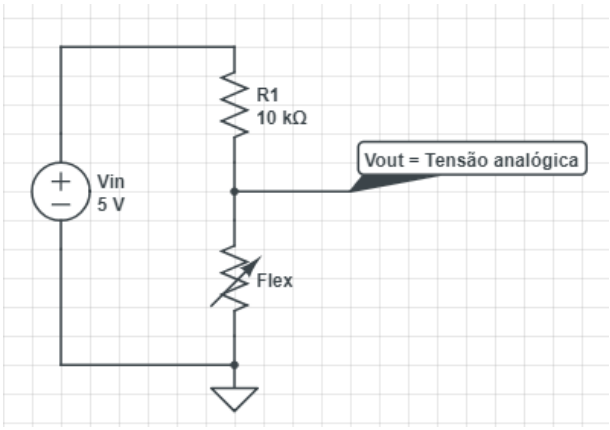

Figura 14 – Divisor Resistivo

Fonte: ELABORADA PELO AUTOR

$$V_{OUT} = V_{IN} \times \frac{R1}{R1 + Flex} \tag{5.7}$$

Dessa forma, é possível utilizar uma entrada analógica do microcontrolador para ler a informação do sensor, obtendo, para um conversor AD de 12 bits, uma variação de 0 a 4095 para uma entrada de 0V até a tensão de referência do microcontrolador, que no caso é de 3.3V.

5.3 Microcontrolador ESP32

Dentre tantos microcontroladores disponíveis no mercado, o ESP32 foi escolhido principalmente por sua conectividade Wi-Fi embutida e também por ser um microcontrolador moderno e com bom poder de processamento.

O ESP32, apresentado na Figura 15, é um dispositivo desenvolvido pela Espressif Systems que atende a diversas necessidades da IoT, pelo seu tamanho reduzido, baixo consumo de energia, alta integração, e principalmente, pela sua conectividade Wi-Fi e Bluetooth. O dispositivo possui as seguintes características:

- CPU: Xtensa® Dual-Core 32-bit LX6;
- ROM: 448 KB
- Flash: 4MB
- Clock Ajustável de 80MHz a 240 MHz
- Wireless padrão 802.11 b/g/n
- Conexão Wi-Fi 2.4Ghz a 2.5Ghz
- Modos de Operação: Cliente, Ponto de Acesso, Cliente + Ponto de Acesso
- Antena embutida
- Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode e P2P Power Management
- Bluetooth BLE 4.2
- 36 GPIOs
- GPIO com funções de PWM, I^2C, SPI
- 18 Conversores analógico digital (ADC)
- 2 Conversores digital analógico (DAC)
- Tensão de operação 3.3V

Figura 15 – ESP32-WROOM-32

Fonte: (DIGIKEY, 2019)

No projeto será utilizado o módulo ESP32 DEVKIT V1, apresentado na Figura 16, contendo o ESP32 da Figura 15, adicionando pinos para facilitar o uso, reguladores de tensão, conector micro USB para alimentação e programação, resistores, LEDs e outros dispositivos para facilitar a interface do desenvolvedor com o dispositivo. O módulo ESP32 DEVKIT V1 pode ser programado utilizando a IDE do Arduino.

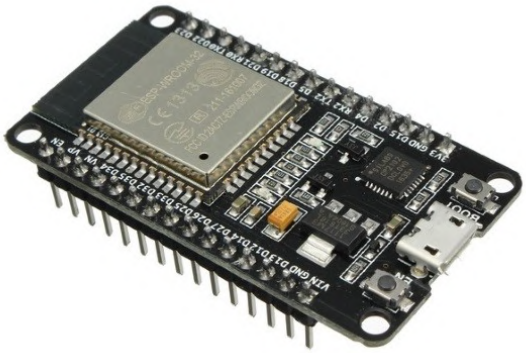

Figura 16 – ESP32 DEVKIT V1

Fonte: (KOYANAGI, 2017)

Na Figura 17 é exibida a pinagem do módulo ESP32 DEVKIT V1, contendo a descrição de todos os pinos de entrada e saída.

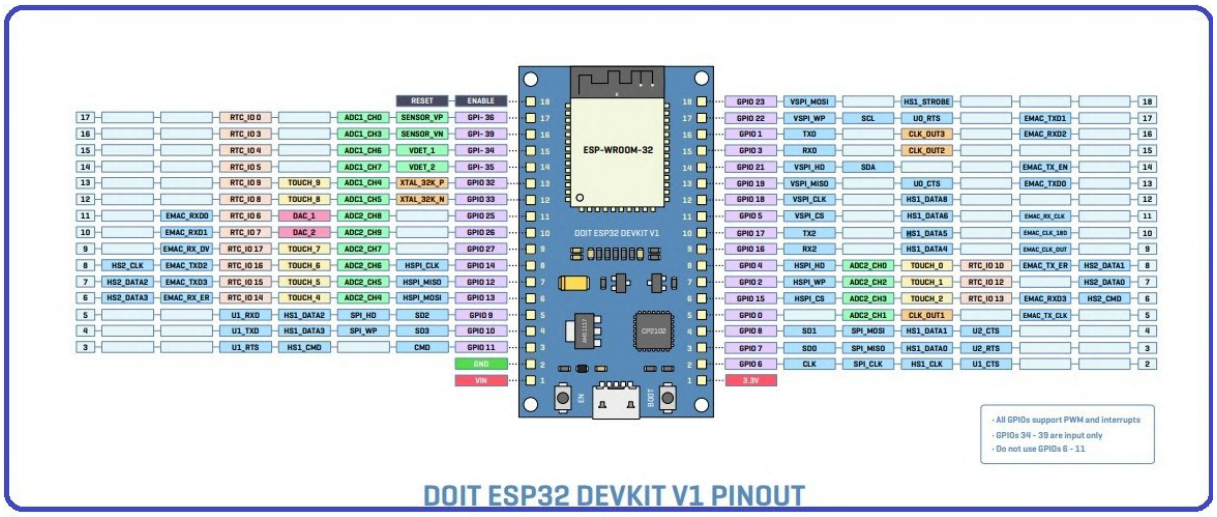

Figura 17 – Pinagem do ESP32 DEVKIT V1

Fonte: https://microcontrollerslab.com/esp32-pinout-use-gpio-pins/

5.4 Braço robótico

O braço robótico utilizado no projeto conta com quatro servo-motores, permitindo que sejam feitos movimentos de rotação da base, controle de abertura da garra, avanço e elevação do braço.

O modelo escolhido para o projeto, apresentado na Figura 18, é fabricado em MDF (Medium Density Fiberboard - Palca de Fibra de Média Densidade) e foi escolhido por ser compacto, leve e de baixo custo. A sua montagem é simples, visto que as peças já vêm cortadas, restando apenas parafusá-las nos seus respectivos lugares e posicionar os servo-motores.

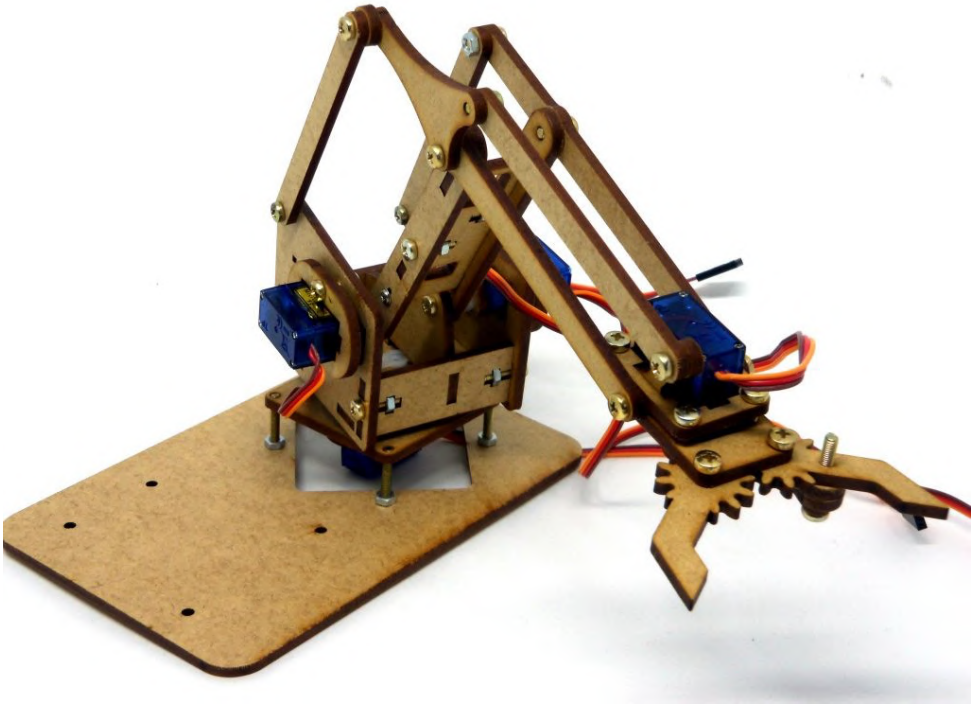

Figura 18 – Braço Robótico

Fonte: (VIDAL, 2017)

5.4.1 Servo-motor

O Servo-motor (ou somente Servo) é um dispositivo eletromecânico que possui um sistema de controle embutido. O servo utilizado no projeto possui controle de posição, portanto, possui um dispositivo que lê a sua posição angular e realimenta o controle do motor, para que este vá para uma determinada posição recebida pelo sistema de controle.

No projeto será utilizado um micro servo-motor SG90 da marca Tower Pro, exibido na Figura 19 que possui como características:

- Tensão de operação: 3.3V a 7.2V;
- Ângulo de rotação: 180 graus;
- Velocidade: 0,12 seg/60Graus (4,8V) sem carga;
- Torque: 1,2 kg.cm (4,8V) e 1,6 kg.cm (6,0V);
- Temperatura de Operação: -30C +60C;
- Tipo de Engrenagem: Nylon
- Dimensões: 32 x 30 x 12mm
- Peso 9g

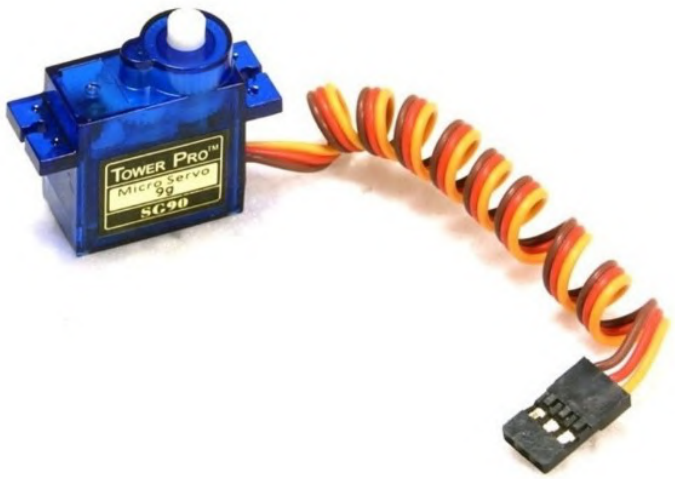

Figura 19 – Servo-motor Tower Pro SG90

Fonte: (FILIPEFLOP, 2019)

O servo-motor utilizado funciona com um sinal de entrada modulado por PWM. Para manter o servo na posição desejada, é utilizado um potenciômetro no motor, esse funciona como um sensor de posição. O sinal do potenciômetro é retornado para o controle do servo, a fim de garantir que o motor fique na posição desejada.

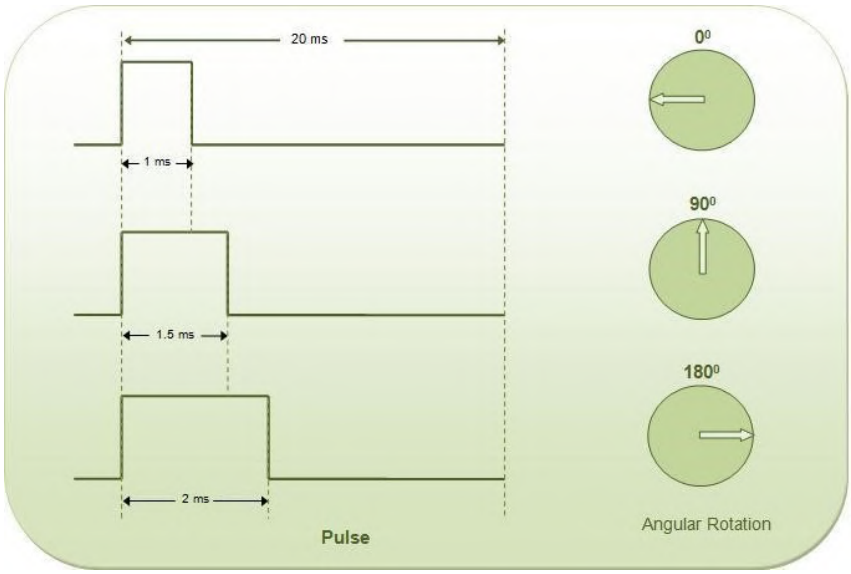

Figura 20 – Controle de um servo via PWM

Fonte: (MOTA, 2017)

Como é possível ver na Figura 20, a posição do servo motor é proporcional ao Duty Cycle do sinal PWM.

Diferentemente de motores comuns de corrente contínua ou motores de passo, que podem girar indefinidamente, o eixo de um servo apenas possui liberdade de 180 graus. Esse é o motivo pelo qual ele é muito utilizado na robótica, pois consegue determinar posição de forma rápida e precisa.

5.5 Câmera IP

Denomina-se como câmera IP uma câmera de vídeo que pode ser acessada e controlada remotamente de qualquer rede IP, podendo ser via LAN, Intranet ou Internet.

Esses dispositivos combinam as capacidades de filmagem, de uma câmera, e de armazenamento, de um computador, pois são equipados com um servidor interno e uma placa de processamento. Uma câmera IP, ilustrada na Figura 21, não necessita de softwares ou placas adicionais, o que torna a sua instalação e manuseio fácil dentro de uma rede, pois elas possuem seu próprio endereço IP.

Figura 21 – Câmera IP Wireless interna da marca Foscam

Fonte: (FOSCAM, 2019)

Depois de configurada, a câmera é encontrada na rede como qualquer outro dispositivo.

6

Projeto de Sistema de Software

Para operação do sistema de hardware proposto é necessário desenvolver um software que dê suporte às ações que serão realizadas pelo sistema. Pensando nisso, o projeto de software será desenvolvido de forma subdividida, conforme o posicionamento dos dispositivos do sistema.

O projeto de software pode ser subdividido em três principais partes:

- Software do Computador;
- Software da Luva;
- Software Remoto;

6.1 Software do Computador

Situado no bloco Central, o computador terá um software que fará a comunicação com o com o roteador local, para recepção de dados. Pensando nessa necessidade foi feito um esboço de tela para exemplificar a interface do software do computador, conforme a Figura 22.

Para o desenvolvimento do software do computador será utilizada a linguagem de programação Java, por conta da familiaridade com a linguagem e da aplicabilidade da mesma.

Na tela do computador serão exibidas informações dos sensores da luva, em forma de rastreamento 3D para o MPU-6050, e em forma de barra de intensidade para o sensor flexível, visando obter maior controle dos movimentos que estão sendo realizados.

Adicionalmente, o software presente no computador também irá lançar a aplicação CMSClient, que é um software de monitoramento de câmeras. Será utilizado este software pois ele é recomendado pelo fabricante da câmera utilizada.

O software CMSClient ficará com maior destaque na tela, a fim de dar mais notoriedade ao ambiente monitorado, pois é esse o alvo do controle remoto.

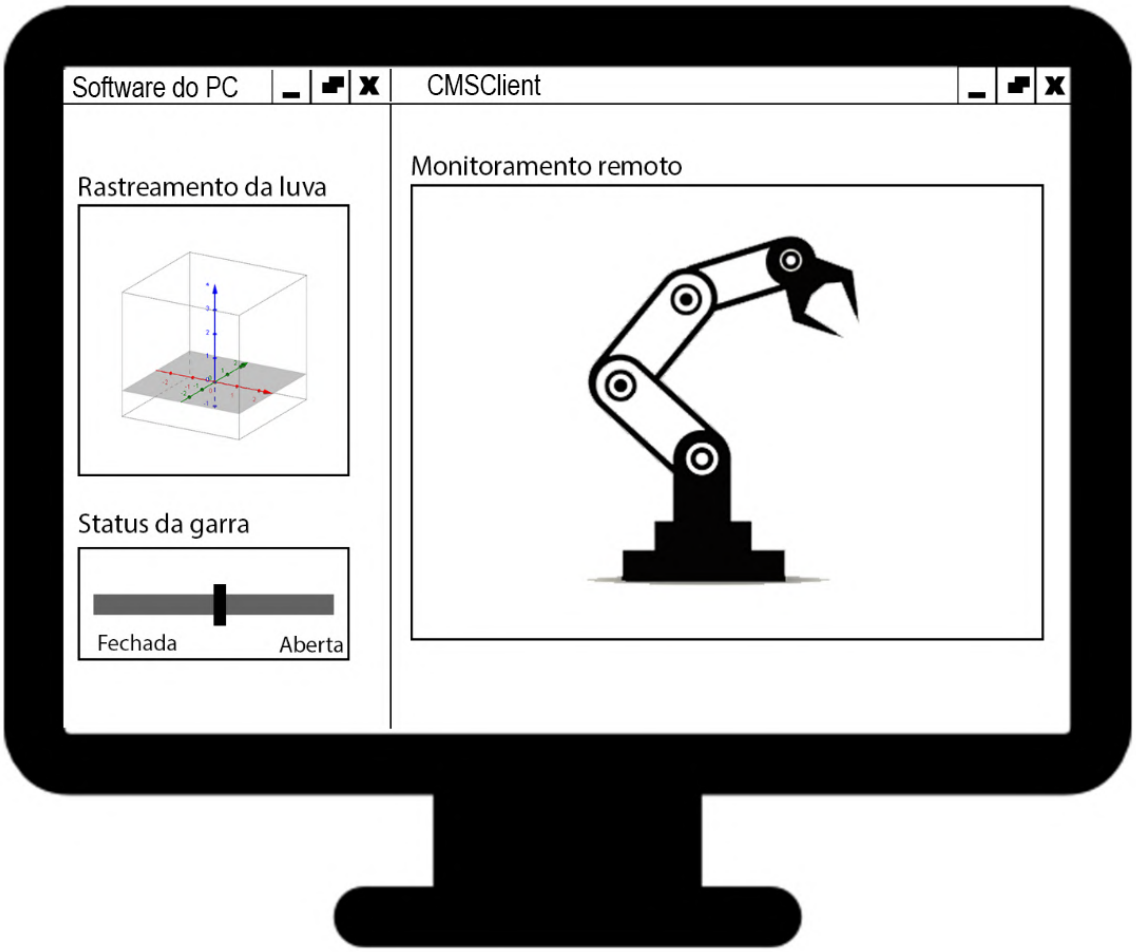

Figura 22 – Esboço de tela do software do computador

Fonte: ELABORADA PELO AUTOR

O software do computador irá receber as informações de movimentação da luva via internet, através do protocolo MQTT. Na Figura 23 estão expostas, em forma de diagrama de blocos, as informações que chegam até o software do computador.

Figura 23 – Informações que chegam ao software do computador

Fonte: ELABORADA PELO AUTOR

6.2 Software da Luva

A luva está contida no bloco Central e possuirá um software dedicado, desenvolvido para o microcontrolador ESP32 anexado à mesma.

A função do software da luva é ler as informações dos sensores e enviar essa informação ao sistema remoto e ao computador, para que ela possa ser exibida.

As informações são enviadas ao Broker através do protocolo MQTT. Desta forma, não é necessário enviar os dados para os dois dispositivos, bastando apenas que os dispositivos assinem o tópico em que serão publicadas as informações. Assim, para cada dado novo, somente é feito um envio ao Broker.

Na Figura 24 está a representação em diagrama de blocos das etapas realizadas pelo software da luva.

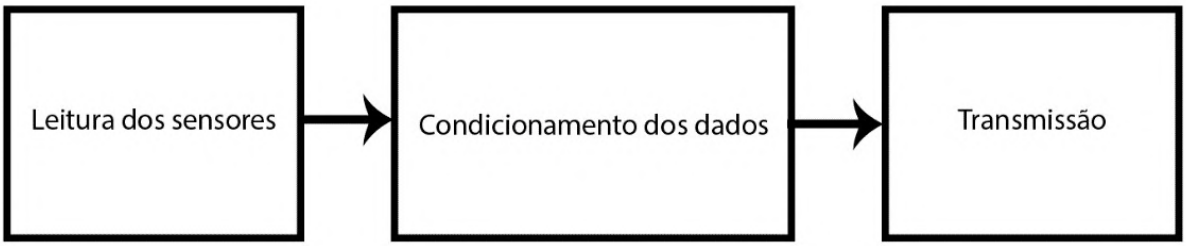

Figura 24 – Fluxo do software da luva

Fonte: ELABORADA PELO AUTOR

Na etapa de leitura é feita a captura da leitura atual dos sensores utilizados. Na etapa de condicionamento dos dados é feita a preparação dos dados brutos dos sensores, transformando-os em informação visual. As informações de aceleração e velocidade angular passam por alterações, conforme as equações 5.1, 5.2, 5.4, 5.5 e 5.6, a fim de transformar essas informações em informação de ângulo. Também é nessa etapa que serão feitas filtragens nos sensores, com o objetivo de eliminar ruídos e aproveitar melhor a característica de cada sensor. No sensor Flex foi utilizado um filtro de média, a fim de eliminar oscilações rápidas e ruídos de alta frequência. Nos sensores acelerômetro e giroscópio do MPU-6050 foi utilizada uma abordagem de fusão de sensores, denominada filtro complementar, a fim de eliminar ruídos dos dois sensores.

6.2.1 Fusão de Sensores

Os valores de aceleração lidos do MPU-6050 apresentam um ruído de alta frequência, visto que o acelerômetro tem elevada sensibilidade a variações de aceleração. Além disso, o acelerômetro também está sujeito a acelerações lineares, o que faz com que ocorram variações na leitura de ângulo somente com movimentação linear.

A leitura do giroscópio é dada em velocidade angular, desta forma é necessário aplicar uma integração para obter os dados de posição angular. Por conta desta integração é adicionado

à leitura de ângulo um ruído de baixa frequência, conhecido como drift.

Para solucionar estes problemas, utiliza-se a abordagem de fusão de sensores, onde entre as mais conhecidas estão: Filtro de Kalman, Direction Cosine Matrix Filter (Filtro de matriz de cosseno de direção), Filtro de Sebastian Madgwick e Filtro Complementar. Dentre as opções, é escolhida a de filtro complementar, devido à simplicidade do mesmo e a aplicabilidade no sistema proposto.

O Filtro Complementar utiliza a propriedade dos sensores para utilizá-los em conjunto, fazendo com que um sensor elimine o erro do outro. Como o acelerômetro possui ruído de alta frequência, ele é passado por um filtro passa-baixas fazendo com que o ruído seja atenuado e a leitura seja mais precisa, no entanto, as informações de alta frequência, dadas por variações rápidas, ficam comprometidas visto que o filtro irá atenuar essas informações também. Já o giroscópio, por ter um ruído de baixa frequência associado à integração, é passado por um filtro passa-altas, fazendo com que o drift de ângulo seja reduzido e, desta forma, a leitura seja mais precisa, porém, devido ao comportamento do filtro, as variações lentas na movimentação serão atenuadas para a leitura do giroscópio. O Filtro Complementar, então, utiliza as duas leituras juntas, para compor uma única informação, conforme a figura 25, visto que em frequências baixas o acelerômetro tem boa precisão e em frequências altas o giroscópio tem boa qualidade na medida. Desta forma, o acelerômetro e o giroscópio são complementares. Este comportamento em frequência é ilustrado na figura 26.

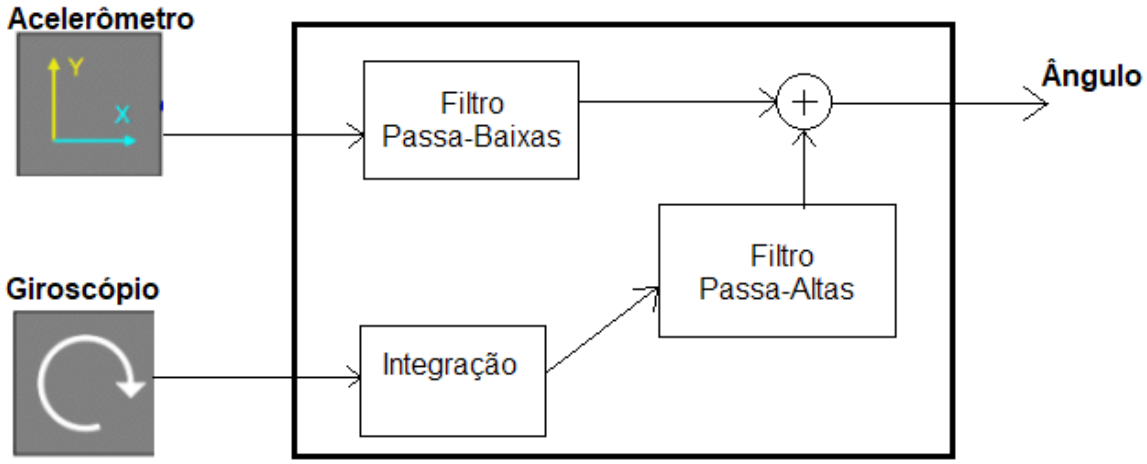

Figura 25 – Diagrama de blocos do Filtro Complementar

Fonte: ELABORADA PELO AUTOR

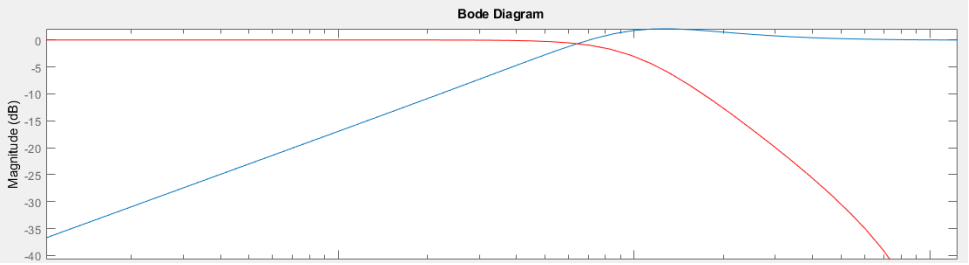

Figura 26 – Resposta em frequência de Filtro Complementar

Fonte: ELABORADA PELO AUTOR

6.3 Software Remoto

Na parte remota do sistema, o software será desenvolvido para o segundo microcontrolador ESP32, e terá a função de receber as informações do bloco Central e executar os movimentos o braço robótico. Essas funcionalidades estão expostas no diagrama de blocos da Figura 27.

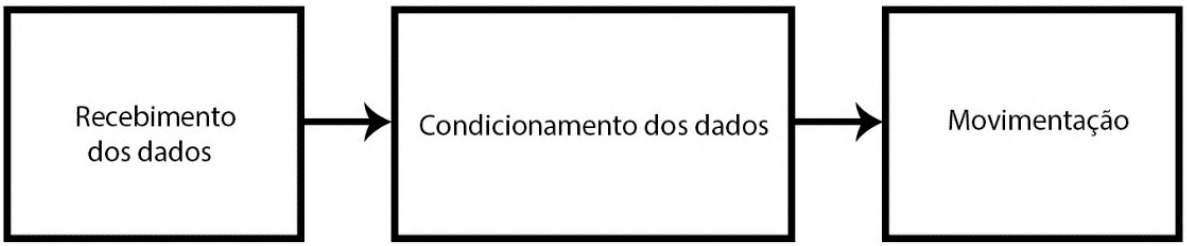

Figura 27 – Fluxo do software remoto

Fonte: ELABORADA PELO AUTOR

Na etapa de recebimento dos dados é feita a leitura dos dados que foram recebidos através da internet. Esses dados contêm informação de movimentação da luva e de abertura e fechamento da mão. Para que ocorra a movimentação, esses dados precisam ser interpretados e transformados em PWM para controle dos servo-motores do braço robótico. Esse trabalho de transformação é feito na etapa de condicionamento dos dados, utilizando a biblioteca Servo do ambiente de desenvolvimento do Arduino.

7

Implementação

Para tornar claro o funcionamento do sistema, bem como as suas conexões e fluxo de execução, é feito uso de diagramas de bloco, fluxogramas e imagens, além da disponibilização dos códigos-fonte desenvolvidos.

7.1 Implementação do Hardware

Para a implementação do hardware serão implementados os blocos Local e Remoto, descritos anteriormente, em uma única rede, apenas a fim de evitar o uso de um segundo roteador. Apesar de conectados à mesma rede, a comunicação entre os dispositivos é feita via internet utilizando o protocolo MQTT para a conexão entre os dois microcontroladores ESP32, bem como entre o microcontrolador local e o computador local, que utiliza da mesma conexão para ler os dados de movimentação da mão. O diagrama de blocos que representa essa conexão é apresentado na figura 28.

Para a montagem da luva foi utilizada uma placa de prototipagem, além do ESP32, o MPU-6050, sensor flex, alguns leds e resistores. A montagem foi feita sobre uma luva com abertura na parte superior. Foi montado o ESP32 junto com o MPU-6050 na parte superior da luva, conforme a figura 29.

O sensor Flex foi instalado na parte interna da luva, de modo a ser flexionado conforme o movimento de fechamento da mão. A instalação deste sensor é vista na figura 30.

Para alimentação da luva foi utilizado um pack de duas baterias de íon de lítio, de 3.5V cada, totalizando 7V. Foi utilizado também um regulador de tensão L7805V para ajustar a tensão de alimentação para 5V e um conector micro USB para alimentação do ESP32. A figura 31 mostra o pack de baterias montado. Na figura 32 é possível ver a luva ligada com a alimentação do pack de baterias.

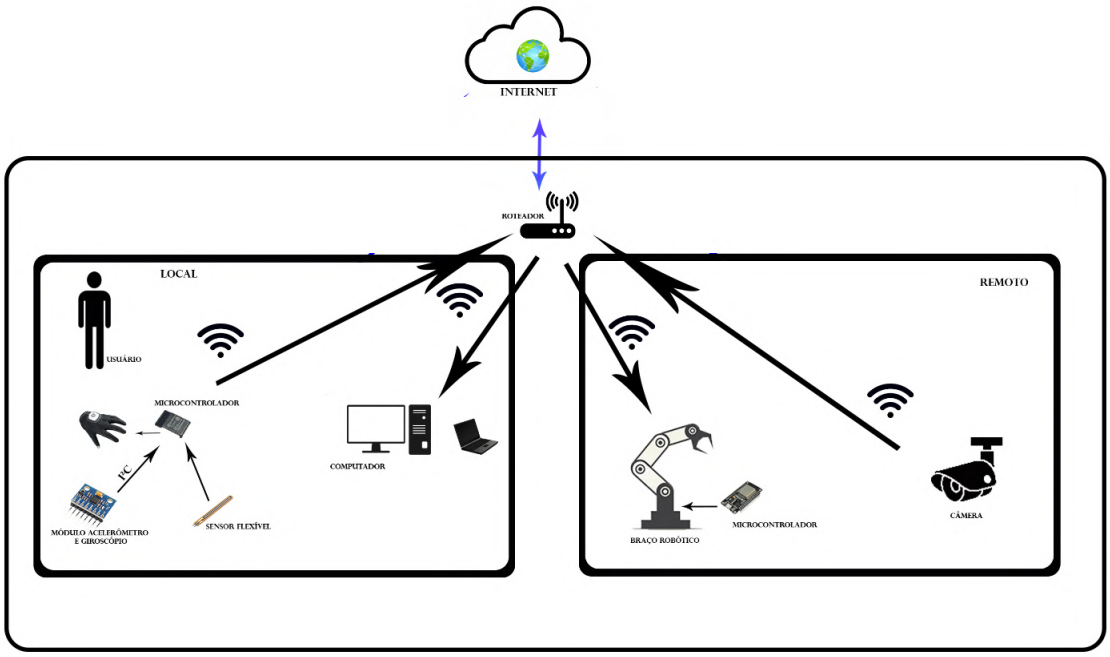

Figura 28 – Diagrama de blocos com sistema em rede local

Fonte: ELABORADA PELO AUTOR

Figura 29 – Luva adaptada para sistema de movimentação

Fonte: ELABORADA PELO AUTOR

Figura 30 – Sensor Flex posicionado na parte interna da luva

Fonte: ELABORADA PELO AUTOR

Figura 31 – Pack de baterias com regulador de tensão e conector micro USB

Fonte: ELABORADA PELO AUTOR

Figura 32 – Luva ligada à fonte de alimentação

Fonte: ELABORADA PELO AUTOR

O braço robótico foi montado segundo manual de instruções do fabricante, que disponibiliza as peças em MDF já cortadas além dos parafusos para fixação do mesmo e os servomotores que dão movimento ao braço. O braço foi montado junto à uma protoboard, onde é feita a sua ligação com o microcontrolador ESP32 e a sua alimentação a uma fonte de 5V. A figura 33 exibe o braço robótico, utilizado neste trabalho, montado.

A câmera IP foi posicionada atrás do braço robótico, conforme a figura 34 para que possa observar os movimentos na mesma direção que o operador está movimentando, ou seja, para que não houvesse espelhamento do movimento. Desta forma, a imagem observada é parecida com a da figura 35, onde é mostrada a visão que deverá ser vista através da câmera.

A câmera foi devidamente configurada utilizando o software CMSClient, recomendado pelo fabricante.

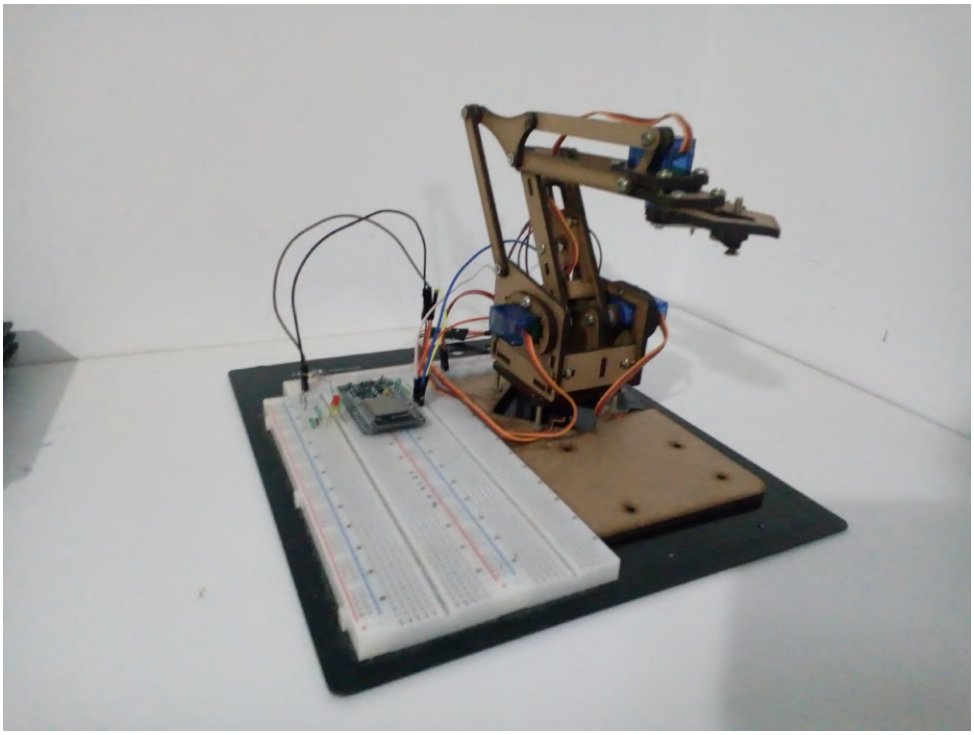

Figura 33 – Braço robótico montado junto ao microcontrolador

Fonte: ELABORADA PELO AUTOR

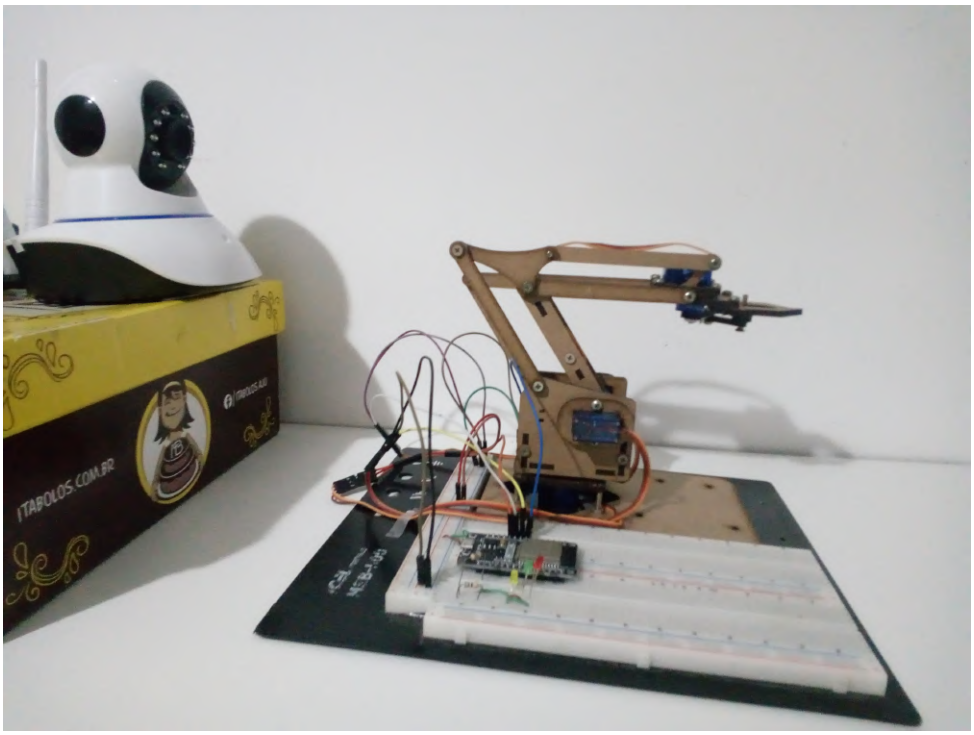

Figura 34 – Posicionamento da câmera

Fonte: ELABORADA PELO AUTOR

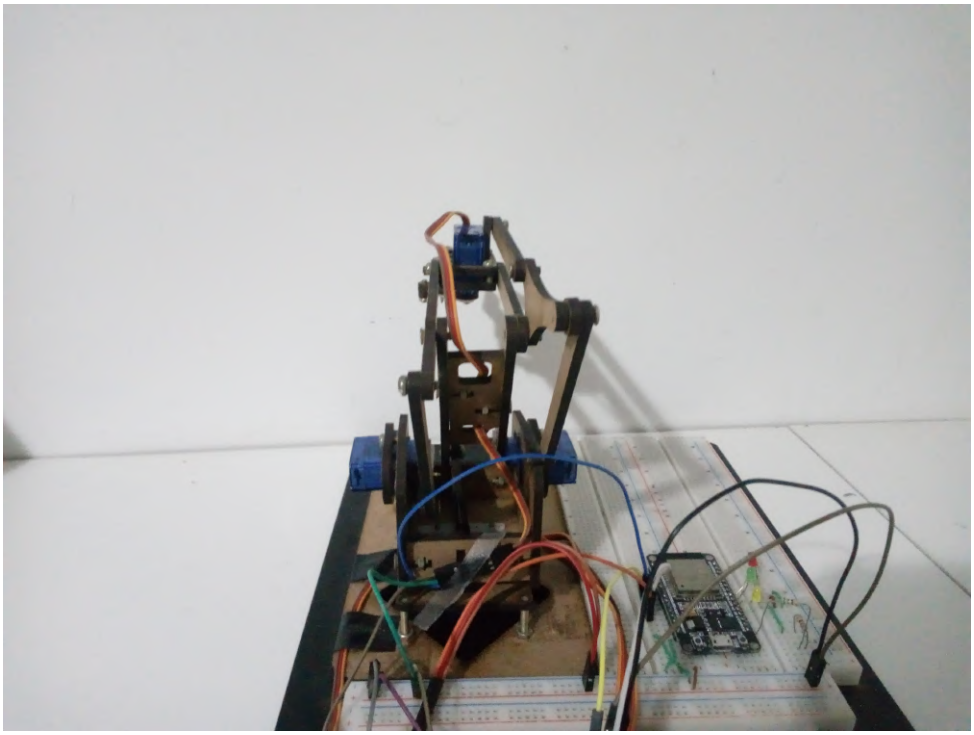

Figura 35 – Visão observada pela câmera

Fonte: ELABORADA PELO AUTOR

7.2 Implementação do Software

7.2.1 Software da luva

O software da luva conta com as seguintes funcionalidades: Leitura de sensores, filtragem dos sensores, conetividade Wi-Fi, Publisher MQTT. Na sua implementação foram utilizadas as bibliotecas WiFi, para fornecer uma forma simples de conectar o ESP32 à rede Wi-Fi, PubSubClient, para fornecer conexão com o Broker MQTT e envio de mensagens, MPU6050, Wire e I2Cdev, para fornecer acesso à comunicação I^2C e aos registradores do MPU6050.

Para melhor entendimento da sequência de execução do software embarcado na luva será feito o uso de um fluxograma, apresentado na figura 36.

Segundo o fluxograma da figura 36, no início são feitas configurações de pinos e conexões ao Wi-Fi e ao broker MQTT, em seguida é feita a inicialização dos sensores. Após a inicialização é iniciado um bloco que ficará em loop infinito. Nesse bloco é feita a leitura dos sensores e a filtragem dos seus dados, logo em seguida são verificadas as conexões MQTT e Wi-Fi. Na sequência é checado se alguma informação lida pelos sensores foi diferente da leitura anterior, para que só então sejam publicados os dados dos sensores no broker MQTT. Caso nenhuma leitura tenha sido diferente da anterior, os dados não são publicados no broker, desta forma evita-se que sejam feitas publicações desnecessárias com o mesmo valor em todos os sensores.

O código do software da luva está disponível no apêndice A.

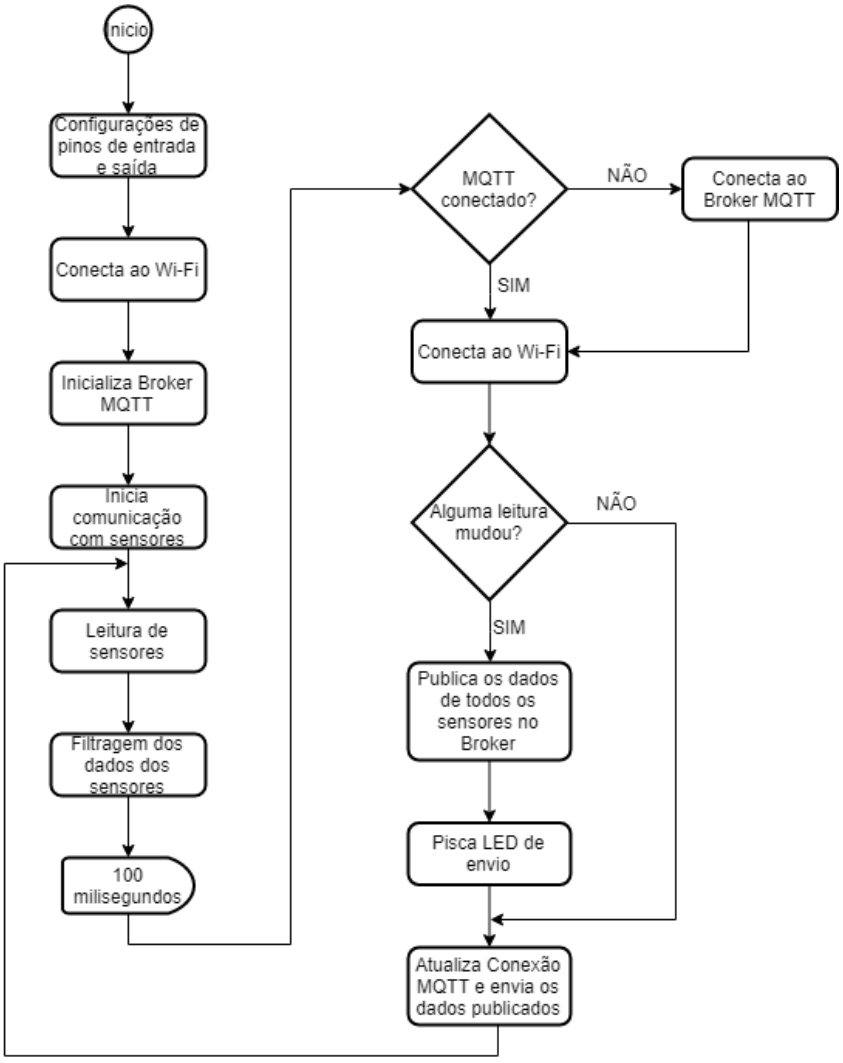

Figura 36 – Fluxograma do software da luva

Fonte: ELABORADA PELO AUTOR

7.2.2 Software do Computador

O software do computador, desenvolvido em linguagem de programação Java, conta com as seguintes funções: Comunicação MQTT, Abertura de programa para streaming de vídeo, renderização de objeto 3D.

Para realizar a comunicação MQTT é utilizado o projeto Eclipse Paho, que dispõe de implementações open-source de clientes MQTT.

Para realizar a abertura do programa de terceiros foi utilizada a classe Runtime do Java, que permite que a aplicação java tenha uma interface com o ambiente em que ela está sendo executada.

Para realizar a renderização 3D são utilizadas as bibliotecas do Java3D, que fornecem ferramentas para criação de objetos 3D com interação em interface gráfica. Também foram utilizadas algumas bibliotecas nativas do Java para adaptação de gráficos, conversão de objetos e

para execução de funções diversas.

Para melhor entendimento da sequência de execução do software que executa no computador será feito o uso dos fluxograma apresentados nas figura 37 e 38.

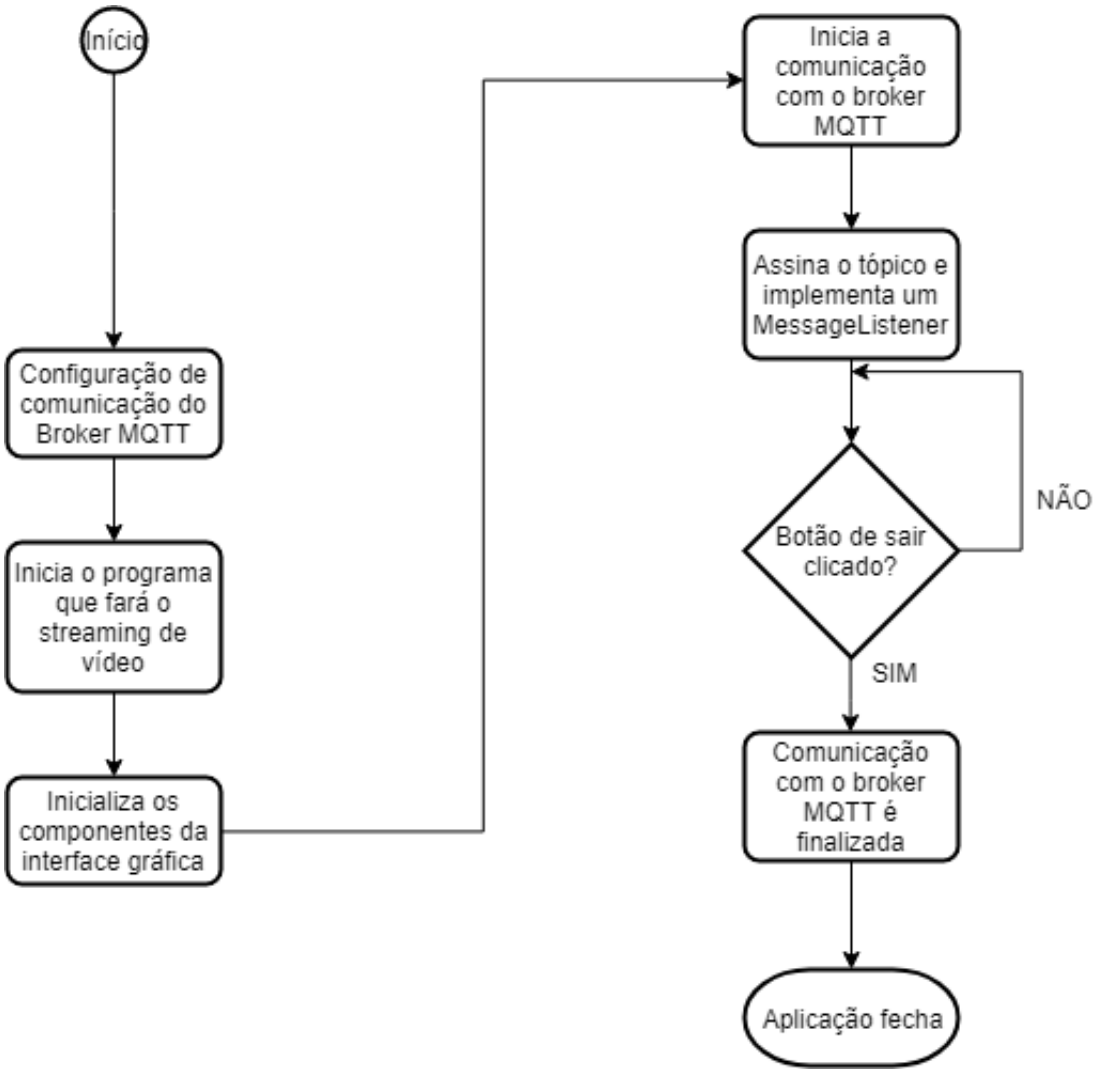

Figura 37 – Fluxograma do software do computador

Fonte: ELABORADA PELO AUTOR

De acordo com o fluxograma da figura 37, inicialmente são feita as configurações de comunicação o broker MQTT. Na sequência é iniciada a aplicação responsável por fazer a conexão com a câmera IP. Após a inicialização do programa da câmera é carregada a interface gráfica, neste momento é renderizada a figura 3D que representa a movimentação da luva. Com a interface gráfica já renderizada é iniciada a comunicação com o broker MQTT, para então ser feita a assinatura no tópico de interesse, a fim de receber os dados de movimentação da luva. A implementação da assinatura do tópico é feito em um Message Listener (Ouvinte de Mensagem), que funciona de forma assíncrona e é chamado sempre que uma mensagem chega no tópico assinado. Conforme o fluxograma da figura 38, quando uma mensagem é recebida,

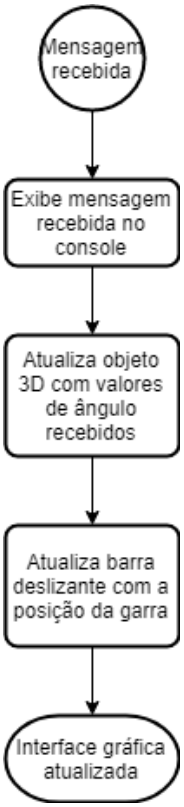

Figura 38 – Fluxograma do recebimento de mensagens pelo software do computador

Fonte: ELABORADA PELO AUTOR

ela é convertida e os componentes da interface gráfica são atualizados para representar os dados recebidos.

Caso o usuário encerre a aplicação a comunicação com o broker é encerrada e a aplicação é encerrada. Caso o usuário não encerre a aplicação, ela continua executando sem tempo limite para encerramento.

O código do software do computador está disponível no apêndice C.

7.2.3 Software Remoto

No software remoto, desenvolvido para o microcontrolador ESP32, foram utilizadas as bibliotecas WiFi, para fornecer uma forma simples de conectar o ESP32 à rede Wi-Fi, PubSubClient, para fornecer uma conexão com o Broker MQTT e possibilitar a assinatura de tópicos e recebimento de mensagens. Foi utilizada também a biblioteca Servo, para simplificar a tradução de informações de ângulo em PWM para os servo-motores do braço robótico. Para melhor entendimento da sequência de execução do software embarcado no sistema remoto será feito o uso de um fluxograma, apresentado na figura 39.

No fluxograma da figura 39, observa-se que inicialmente são feitas configurações de pinos e servo-motores, bem como conexões ao Wi-Fi e ao broker MQTT. Além disso, também

é definida a função que será executada quando chegar uma mensagem no tópico em que o subscriber estiver assinando. Na sequência é executado um bloco que ficará em loop infinito, consistindo de verificação das conexões Wi-Fi e MQTT, e atualização e processamento dos dados da mensagem recebida.

Ainda no fluxograma é exibido um fluxo do recebimento de pacote, este fluxo é executado sempre que um pacote é recebido via MQTT. Nele é sinalizado, por um LED, que chegou uma mensagem e então esses valores são convertidos e utilizados para realizar a movimentação dos servo-motores do braço robótico.

O código do software remoto está disponível no apêndice B.

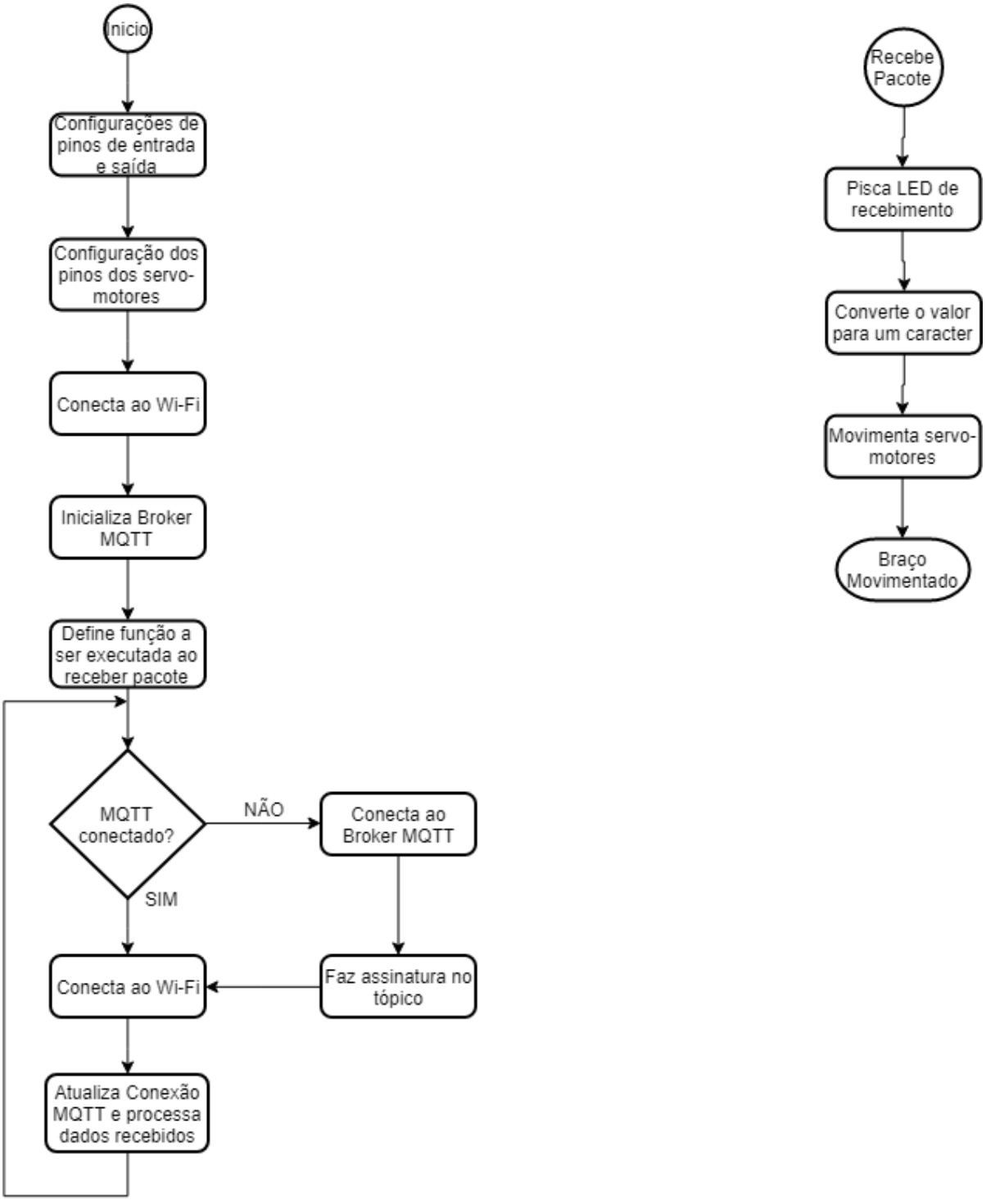

Figura 39 – Fluxograma do software do ambiente remoto

Fonte: ELABORADA PELO AUTOR

8

Resultados e Discussão

Durante os testes, notou-se que havia uma latência entre a movimentação da mão e a execução do movimento do braço robótico, apesar de o protocolo utilizado ser otimizado para reduzir este tipo de problema. Isso se deve principalmente à latência da própria conexão de internet e ao fato de o Broker gratuito possuir o processamento compartilhado entre todos os usuários. Observando a latência média da rede no site rjnet.com.br , notou-se que esta era de aproximadamente 255 ms, conforme a figura 40, já o atraso observado do sistema de movimentação foi de cerda de 500 ms. Em alguns momentos do teste ocorreram atrasos superiores a 1 segundo. Isso se dá principalmente pela variação da latência da rede e pelo compartilhamento do serviço do Broker.

O atraso das imagens da câmera, por sua vez, tiveram um atraso maior, chegando em alguns casos a mais de 2 segundos.

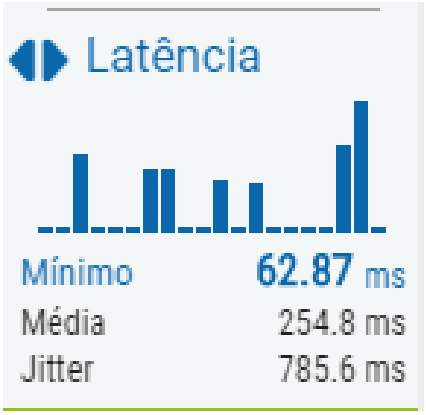

Figura 40 – Latência da rede utilizada nos testes

Fonte: http://www.rjnet.com.br/velocimetro.html

A movimentação do braço robótico corresponde com precisão aos valores esperados, oriundos da leitura dos sensores da luva. Os dados de ângulo foram validados apresentando um

desvio máximo de 1 grau, tomando como parâmetro os sensores de um smartphone, conforme as figuras 41 e 42, onde é possível ver o ângulo medido pelo celular (circulado em vermelho) e o ângulo correspondente medido pelo MPU-6050 utilizando filtro complementar (circulado em azul).

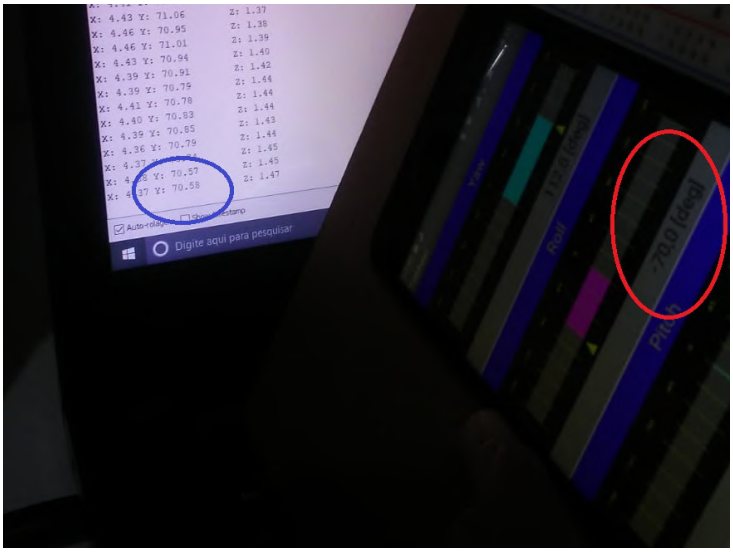

Figura 41 – Validação de ângulos junto à um smartphone com a medida em 70 graus

Fonte: ELABORADA PELO AUTOR

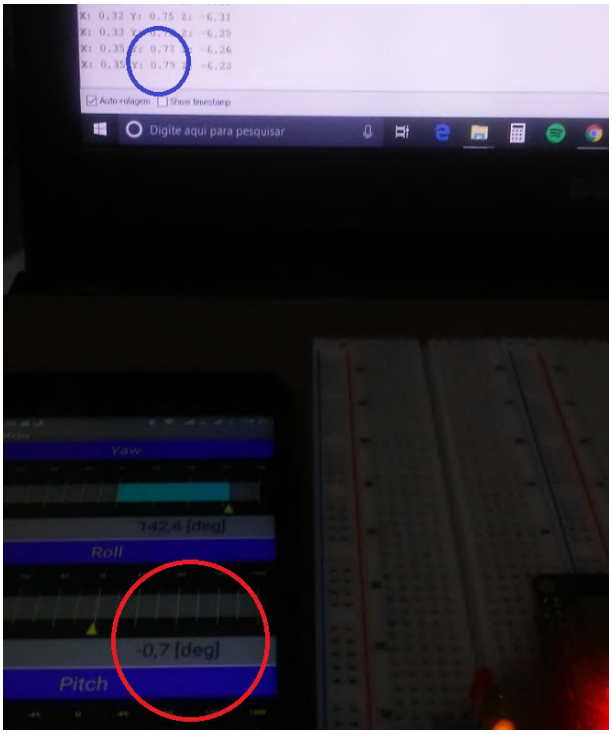

Figura 42 – Validação de ângulos junto à um smartphone com a medida em 0.7 graus

Fonte: ELABORADA PELO AUTOR

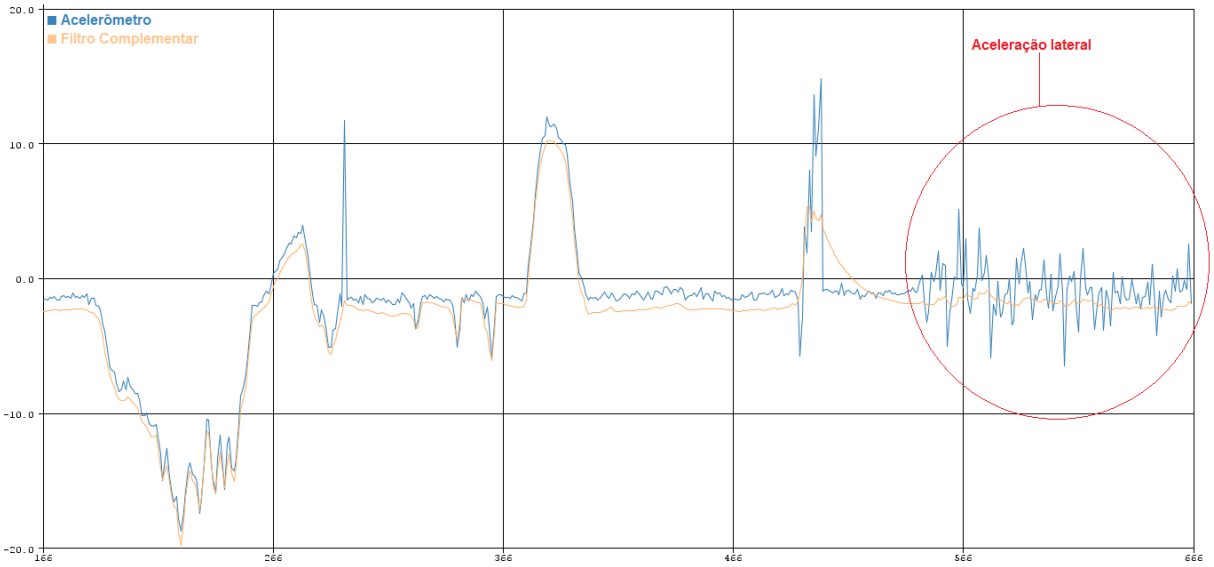

Figura 43 – Comparação entre acelerômetro e filtro complementar

Os dados dos sensores acelerômetro e giroscópio apresentaram bastante ruído, por isso foi implementada o filtro complementar. Tal abordagem permite, de forma simples, reduzir o ruído dos dois sensores e obter uma medida mais robusta. Na figura 43 é feita uma comparação dos dados de ângulo medidos em um teste utilizando o acelerômetro e utilizando o filtro complementar com os dois sensores.

Pelo gráfico da figura 43, é possível ver que a leitura somente pelo acelerômetro (em azul) apresenta mais ruídos e variações indevidas do que o filtro complementar (em amarelo). Além disso, é possível notar no círculo em vermelho que o filtro complementar é menos suscetível a variações advindas de aceleração lateral, diferente do acelerômetro, que apresenta uma variação de cerca de 5 graus para uma pequena aceleração lateral testada.

O filtro complementar corrigiu as imperfeições nos eixos x e y, porém, como não é possível obter o ângulo de rotação do eixo z com o acelerômetro, não foi possível implementar o filtro complementar para o eixo z. Desta forma, a medida de ângulo neste eixo ainda apresentava o erro, conhecido como drift de ângulo. Para corrigir este erro foi implementado um filtro passa-altas no eixo z do giroscópio. O uso deste filtro faz com que as componentes de baixa frequência sejam atenuadas, assim, quando em repouso, a medida de ângulo tende a voltar para a posição inicial. Este erro foi reduzido ao máximo, mas não pôde ser solucionado utilizando apenas giroscópio e acelerômetro, tendo como possível solução a implementação de um magnetômetro junto aos sensores, para assim corrigir o drift no eixo z.

Os servomotores do braço robótico apresentaram um bom desempenho, fornecendo torque suficiente para movimentar o mesmo. Em alguns momentos ocorreram trepidações, por insuficiência de fornecimento de corrente da fonte utilizada, mas após a troca da fonte o problema parou de acontecer.

8.1 Custos

O custo total do projeto foi calculado com base no preço atual de mercado dos componentes utilizados somado ao valor do frete, quando aplicável. Conforme a listagem da tabela 1, o custo total ficou em R$ 400,00.

Tabela 1 – Custo do protótipo

Componente	**Custo unitário (R$)**	**Quantidade**	**Custo Final (R$)**
ESP32	40,00	2	80,00
Braço Robótico	100,00	1	100,00
MPU-6050	30,00	1	30,00
Sensor Flex	90,00	1	90,00
Câmera IP	100,00	1	100,00

Foram utilizados ainda protoboards e jumpers, mas seus custos não foram contabilizados pois não são necessários para a implementação do sistema, podendo ser substituídos por uma placa de circuito impresso, cujo custo deve ser estimado baseado no projeto.

9 Conclusão

Este trabalho teve como finalidade o desenvolvimento de um sistema de movimentação de objetos via internet, visando que não houvesse necessidade de o operador estar no mesmo ambiente que o sistema movimentado.

O estudo de caso desenvolvido, operando um braço robótico via internet através de leitura de sensores de movimento, explorou, além da movimentação via internet, o uso de sensores de flexão e inerciais, bem como o tratamento dos seus dados e explorou uma estratégia de filtragem.

O sistema desenvolvido foi capaz de realizar a conexão entre os dois microcontroladores ESP32 através da internet, utilizando o protocolo MQTT. Foi capaz ainda de realizar a leitura e filtragem dos sensores, de receber via MQTT informações em um software desenvolvido para computador e exibir tais informações em tela.

A construção deste sistema possuiu um forte caráter didático: foram realizados estudos de protocolo de rede, arquitetura de rede, sensores, filtros e desenvolvidos trabalhos nas áreas de computação e eletrônica. Os conceitos utilizados neste projeto podem ser importados para qualquer outro sistema de movimentação baseado em uma conexão remota.

9.1 Trabalhos Futuros

Para os próximos trabalhos seria interessante fazer a gravação dos dados de movimentação e de imagem para processamento e implementação de aprendizado de máquina, fazendo com que o sistema movimentado aprendesse os movimentos e pudesse realizá-los de forma autônoma.

A utilização de ferramentas de visão computacional para identificar objetos, permitindo que o sistema pudesse reconhecer e selecionar objetos.

Implementação do recebimento das imagens da câmera dentro do software desenvolvido, eliminando a necessidade de um software de terceiros e fazendo com que toda a aplicação de exibição de movimentos rode dentro de um único software.

Também é desejável em trabalhos futuros que seja implementada uma interface para configurar a conexão Wi-Fi dos microcontroladores, bastando selecionar uma rede Wi-Fi dentre as disponíveis no local e inserir a senha. Desta forma não seria necessário configurar no firmware do microcontrolador sempre que estiver em uma rede Wi-Fi diferente.

Uma outra possível melhoria é a adição de um sensor magnetômetro junto do giroscópio para corrigir o drift de ângulo no eixo z, já que o magnetômetro fornece essa informação, tornando possível o uso da fusão de sensores neste eixo.

Por fim, uma outra melhoria sugerida é a utilização do bluetooth, que já é uma das ferramentas do ESP32, para realizar o envio das informações para o computador. Desta forma o retorno visual do movimento realizado seria mais rápido.

Referências

DIGIKEY. *ESP32-WROOM-32*. 2019. Disponível em: <https://www.digikey.sg/product-detail/en/espressif-systems/ESP32-WROOM-32/1904-1010-1-ND/8544305>. Citado na página 32.

FERREIRA, E.; ALVES, N. Braço articulado com controle proporcional de movimento comandado via bluetooth por um aplicativo desenvolvido para plataformas android. *Sao José dos Campos - São Paulo*, 2013. Citado na página 16.

FILIPEFLOP. *Micro Servo 9g SG90 TowerPro*. 2019. Disponível em: <https://www.filipeflop.com/produto/micro-servo-9g-sg90-towerpro/>. Citado na página 35.

FISCHER, J. *Messaging with MQTT*. 2016. Disponível em: <https://micropython-iot-hackathon.readthedocs.io/en/latest/mqtt.html>. Citado na página 18.

FOSCAM. *Câmera IP Wireless FI19851P (HD)*. 2019. Disponível em: <http://www.foscam.com.br/camera-ip-fi9851w-hd.html>. Citado na página 36.

KOYANAGI, F. *Introdução ao ESP32*. 2017. Disponível em: <https://www.fernandok.com/2017/11/introducao-ao-esp32.html>. Citado na página 32.

KUROSE J; ROSS, K. *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. São Paulo: Pearson Education do Brasil, 2013. ISBN 978-85-430-1443-2. Citado 2 vezes nas páginas 25 e 26.

MORIMOTO, C. E. *Redes, Guia Prático 2ª*. Porto Alegre: Sul Editores, 2011. Citado na página 25.

MOTA, A. *O que é servomotor? | Controlando um servo com arduino*. 2017. Disponível em: <https://portal.vidadesilicio.com.br/o-que-e-servomotor/>. Citado na página 35.

NAYLAMP. *Tutorial MPU6050, Acelerômetro e Giroscópio*. 2016. Disponível em: <https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html>. Citado na página 28.

OLIVEIRA, D. H. M.; SILVA, R. d. O. Braço robótico manipulado através de movimentos reais de um braço humano. Rio verde - GO, 2016. Citado na página 14.

OLIVEIRA, R. R. Uso do microcontrolador esp8266 para automação residencial. Rio de Janeiro - RJ, 2017. Citado na página 14.

SANOU, B. Ict facts and figures 2016. *International telecommunications union. Geneva*, 2016. Citado 2 vezes nas páginas 10 e 11.

SANTOS, L. M. d. Automação residencial utilizando iot. São Cristóvão - SE, 2017. Citado na página 14.

SILVA, A. M. et al. Controle de um braço robótico via web. Uberlândia - MG, 2016. Citado na página 15.

SILVA, J. B. Monitoramento, aquisição e controle de sinais elétricos, via web, utilizando microcontroladores. Florianópolis, SC, 2002. Citado na página 14.

SPARKFUN. *Flex Sensor*. 2019. Disponível em: <https://www.sparkfun.com/products/10264>. Citado na página 29.

THOMSEN, A. *Tutorial: Acelerômetro MPU6050 com Arduino*. 2014. Disponível em: <https://www.filipeflop.com/blog/tutorial-acelerometro-mpu6050-arduino/>. Citado na página 27.

TOLENTINO, G. C.; TSUKAMOTO, D. B.; NOMURA, S. Estudo de caso: Utilização do arduino para um sistema de controle remoto de dispositivos via internet. Uberlândia - MG, 2013. Citado na página 15.

VANÇAN, N. B. M. Protobot - protótipo de braço robótico comandado por comunicação sem fio. Londrina - PR, 2017. Citado na página 16.

VIDAL, V. *Kit Braço Robótico MDF com Arduino*. 2017. Disponível em: <http://blog.eletrogate.com/kit-braco-robotico-mdf-com-arduino/>. Citado na página 34.

YUAN, M. *Conhecendo o MQTT*. 2017. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Citado na página 17.

Apêndices

APÊNDICE A –

Código do software da luva

```
1
2  #include <WiFi.h>
3  #include <PubSubClient.h>
4  #include <MPU6050.h>
5  #include <Wire.h>
6  #include <I2Cdev.h>
7
8  #define pinBotao1 12  // Pino do botao que sera lido
9  #define ledState  LED_BUILTIN // Led para sinalizacao de leitura do
       botao
10 #define ledWiFi   14  // Led para sinalizacao da conexao WiFi
11 #define ledBroker 27  // Led para sinalizacao da conexao com Broker
       MQTT
12 #define ledSend   26  // Led para sinalizacao de publicacao de dado
13 #define flex      15  // Pino para sensor flex
14
15 #define ID_MQTT "jpsq_061"        // Id do dispositivo
16 #define TOPIC_PUBLISH "b_1jpsq"   // Topico do pacote enviado
17 #define CLOUDMQTT_USER   "iatrqott"    // Usuario para o broker
       cloudmqtt
18 #define CLOUDMQTT_PASSWORD  "gW7u8bMQzjOm"// Senha para o broker
       cloudmqtt
19 // Macros para filtragem dos dados do MPU6050
20 #define SCALE_GYRO  131         // Escala de sensibilidade do
       giroscopio  | 250-> 131  | 500-> 65.5|1000-> 32.8|2000->16.4|
21 #define SCALE_ACCEL 16384       // Escala de sensibilidade do
       acelerometro| 2g -> 16384| 4g -> 8192| 8g -> 4096| 16g->2048|
22 #define ALPHA_X     0.08
23 #define X_OFFSET    0.00
24 #define ALPHA_Y     0.08
25 #define Y_OFFSET_ACCEL    0.0
```

```
26 #define Y_OFFSET_GYRO     0.0
27 #define ALPHA_Z     0.008
28 #define Z_OFFSET_GYRO     0.0
29
30 //Credenciais para conexao WiFi
31 const char* SSID = "Link_One_420240";
32 const char* PASSWORD = "quintino";
33 WiFiClient wifiClient;
34
35 //Endereco e porta do MQTT Server
36 const char* BROKER_MQTT = "broker.hivemq.com";
37 int         BROKER_PORT = 1883;
38
39 PubSubClient MQTT(wifiClient);
40
41 //declaracao de funcoes utilizadas
42 void mantemConexoes();
43 void conectaWiFi();
44 void conectaMQTT();
45 void enviaPacote();
46
47 MPU6050 mpu;
48 int16_t ax, ay, az, gx, gy, gz, flexvalue, flexvalue_old, flex_prev
      ;
49 unsigned long time_prev = 0;
50 float ax_deg, ay_deg,   // Valores do acelerometro em angulo(graus)
      .
51       ax_deg_old,ay_deg_old,az_deg_old,
52       gx_dps, gy_dps, gz_dps, // Valores do giroscopio em grau/s
53       gx_old, gy_old, gz_old, // Valores da medida anterior de
            g_dps
54       gx_deg, gy_deg, gz_deg, // Valores do giroscopio em angulo(
            graus).
55       gx_deg_old, gy_deg_old, gz_deg_old, // Valores da medida
            anterior de g_deg
56       ang_x, ang_y, ang_z, dt,  // Valores de angulo apos filtro
            complementar
57       ang_x_old, ang_y_old, ang_z_old; // Valores da medida
            anterior de ang_x
58
59 void setup() {
60   pinMode(pinBotao1, INPUT_PULLUP);
```

```
61    pinMode(ledState, OUTPUT);
62    pinMode(ledWiFi,OUTPUT);
63    pinMode(ledBroker,OUTPUT);
64    pinMode(ledSend,OUTPUT);
65    pinMode(flex, INPUT);
66    Serial.begin(115200);
67    Wire.begin();
68
69    conectaWiFi();
70    MQTT.setServer(BROKER_MQTT, BROKER_PORT);
71
72    mpu.initialize();
73
74  }
75
76  void loop() {
77    mpu.getMotion6(&ax,&ay,&az,&gx,&gy,&gz);
78    flexvalue = analogRead(flex);
79    flexvalue = 0.15*flexvalue + 0.85*flexvalue_old;
80    flexvalue_old = flexvalue;
81    gx_dps = (float)gx/SCALE_GYRO;  // Conversao dos dados do
        giroscopio para grau/s
82    gy_dps = (float)gy/SCALE_GYRO;
83    gz_dps = (float)gz/SCALE_GYRO;
84
85    dt = (float)(micros() - time_prev)/1000000.0; // calculo de tempo
         entre medidas em segundos
86  // Inicio do calculo dos angulos pelo giroscopio
87    gx_deg += 0.5*(gx_dps + gx_old)*dt; // Calculo integral de
        posicao angular no eixo x;
88    gx_old = gx_dps;
89    gx_deg = (1*gx_deg - ALPHA_X*gx_deg_old);// + X_OFFSET; // Filtro
         Passa-Altas
90    gx_deg_old = gx_deg;
91
92    gy_deg += 0.5*(gy_dps + gy_old)*dt;    // Calculo integral de
        posicao angular no eixo y;
93    gy_old = gy_dps;
94    gy_deg = (1*gy_deg - ALPHA_Y*gy_deg_old) + Y_OFFSET_GYRO; //
        Filtro Passa-Altas
95    gy_deg_old = gy_deg;
96
```

```
97    gz_deg += 0.5*(gz_dps + gz_old)*dt;  // Calculo integral de
          posicao angular no eixo z;
98    gz_old = gz_dps;
99    gz_deg = (1*gz_deg - ALPHA_Z*gz_deg_old) + Z_OFFSET_GYRO;
100   gz_deg_old = gz_deg;
101
102   time_prev = micros(); // Atualizacao da referencia de tempo para
          calculo de dt
103 // Fim do calculo dos angulos pelo giroscopio
104 // Inicio do calculo dos angulos pelo acelerometro
105   ax_deg = atan(ay/sqrt(pow(ax,2) + pow(az,2)))*(180.0/3.14); //
          Calculo dos angulos do acelerometro por arcotangente
106   ay_deg = atan(-ax/sqrt(pow(ay,2) + pow(az,2)))*(180.0/3.14);
107
108   ax_deg = (ALPHA_X*ax_deg + (1.0 - ALPHA_X)*ax_deg_old);  //
          Filtro Passa-Baixas
109   ay_deg = (ALPHA_Y*ay_deg + (1.0 - ALPHA_Y)*ay_deg_old) +
          Y_OFFSET_ACCEL;  // Filtro Passa-Baixas
110   ax_deg_old = ax_deg;
111   ay_deg_old = ay_deg;
112 //  Valores definitivos de angulo
113   ang_x = ax_deg + gx_deg; // Complemento
114   ang_y = ay_deg + gy_deg; // Complemento
115   ang_z = gz_deg; // Angulo no eixo z nao possui complemento
116 // Fim do calculo dos anguloa pelo acelerometro
117
118
119   delay(100);
120   mantemConexoes();
121   enviaPacote();
122   MQTT.loop();
123 }
124
125 void mantemConexoes(){
126   if(!MQTT.connected()){
127     conectaMQTT();
128   }
129   conectaWiFi();  // Se nao ha conexao WiFi, a conexao e refeita
130 }
131
132 void conectaWiFi(){
133   if(WiFi.status() == WL_CONNECTED){
```

```
134     digitalWrite(ledWiFi, HIGH);
135     return;
136   }
137
138
139   WiFi.begin(SSID, PASSWORD);
140   while(WiFi.status() != WL_CONNECTED){
141     digitalWrite(ledWiFi, LOW);
142     delay(200);
143
144 }
145
146 void conectaMQTT(){
147 while(!MQTT.connected()){
148
149   if(MQTT.connect(ID_MQTT)){//, MQTT_USER, MQTT_PASSWORD)){
150     digitalWrite(ledBroker, HIGH);
151   }
152   else{
153     digitalWrite(ledBroker,LOW);
154     delay(5000);
155   }
156 }
157
158 }
159
160 void enviaPacote(){
161   static unsigned long delay_send;
162   int ang_x_i = 0, ang_y_i = 0, ang_z_i = 0;
163   int flexao = map(flexvalue, 1800,2400,0,180);
164   char angulo[4] = {0};
165   if( ((int)ang_y != (int)ang_y_old)||((int)ang_x != (int)ang_x_old
        )||((int)ang_z != (int)ang_z_old)||(flexao > (flex_prev+20))
        ||(flexao < (flex_prev-20)) ){
166     ang_x_i = ang_x;
167     angulo[0] = char(ang_x_i + 90);
168     ang_x_old = ang_x;
169     ang_y_i = ang_y;
170     angulo[1] = char(ang_y_i + 90);
171     ang_y_old = ang_y;
172     ang_z_i = ang_z;
173     angulo[2] = char(ang_z_i + 90);
```

```
174     ang_z_old = ang_z;
175     angulo[3] = char(flexao); // Sensor flex
176     flex_prev = flexao;
177
178     if(MQTT.publish(TOPIC_PUBLISH, angulo)){
179       Serial.print("X: ");Serial.print((int)angulo[0]);
180       Serial.print("\tY: ");Serial.print((int)angulo[1]);
181       Serial.print("\tZ: ");Serial.print((int)angulo[2]);
182       Serial.print("\tFlex: ");Serial.println((int)angulo[3]);
183       digitalWrite(ledSend, HIGH);
184       delay_send = millis();
185     }
186   }
187   if((millis()-delay_send) > 100)
188   {
189     digitalWrite(ledSend,LOW);
190     delay_send = millis();
191   }
192 }
```

APÊNDICE B –

Código do software remoto

```
1
2  #include <WiFi.h>
3  #include <PubSubClient.h>
4  #include <Servo.h>
5
6  #define ledState  LED_BUILTIN // Led para sinalizacao de dado
       recebido
7  #define ledWiFi   5  // Led para sinalizacao da conexao WiFi
8  #define ledBroker 18  // Led para sinalizacao da conexao com Broker
       MQTT
9  #define ledReceive   19  // Led para sinalizacao de recebimento de
       dado
10 #define servo1Pin    14  // Servo da base
11 #define servo2Pin    27  // Servo 1 do braco
12 #define servo3Pin    26  // Servo de alcar braco
13 #define servo4Pin    25  // Servo da garra
14 #define ID_MQTT "jpsq062"         // Id do dispositivo
15 #define TOPIC_SUBSCRIBE "b_1jpsq" // Id do pacote assinado
16
17 unsigned long delay_receive;
18 //Credenciais do WiFi
19 const char* SSID = "Link_One_420240";
20 const char* PASSWORD = "quintino";
21 WiFiClient wifiClient;
22
23 //Endereco e porta do MQTT Server
24 const char* BROKER_MQTT = "broker.hivemq.com";
25 int         BROKER_PORT = 1883;
26
27 PubSubClient MQTT(wifiClient);
28
```

```
29 Servo servo1, servo2, servo3, servo4;
30
31 //declaracao de funcoes utilizadas
32 void mantemConexoes();
33 void conectaWiFi();
34 void conectaMQTT();
35 void recebePacote(char* topic, byte* payload, unsigned int length);
36 void moveArm(int servo, int pos);
37
38
39 void setup() {
40   pinMode(ledState, OUTPUT);
41   pinMode(ledWiFi,OUTPUT);
42   pinMode(ledBroker,OUTPUT);
43   pinMode(ledReceive,OUTPUT);
44   servo1.attach(servo1Pin);
45   servo2.attach(servo2Pin);
46   servo3.attach(servo3Pin);
47   servo4.attach(servo4Pin);
48   Serial.begin(115200);
49
50   conectaWiFi();
51   MQTT.setServer(BROKER_MQTT, BROKER_PORT);
52   MQTT.setCallback(recebePacote);
53
54 }
55
56 void loop() {
57   mantemConexoes();
58   MQTT.loop();
59   if((millis() - delay_receive) > 100) digitalWrite(ledReceive, LOW
       );
60 }
61
62 void mantemConexoes(){
63   if(!MQTT.connected()){
64     conectaMQTT();
65   }
66   conectaWiFi();  // Se nao ha conexao WiFi, a conexao e refeita
67 }
68
69 void conectaWiFi(){
```

```
70    if(WiFi.status() == WL_CONNECTED){
71      digitalWrite(ledWiFi, HIGH);
72      return;
73    }
74
75
76    WiFi.begin(SSID, PASSWORD);
77    while(WiFi.status() != WL_CONNECTED){
78      digitalWrite(ledWiFi, LOW);
79      delay(200);
80  }
81
82  void conectaMQTT(){
83    while(!MQTT.connected()){
84      digitalWrite(ledBroker, LOW);
85      if(MQTT.connect(ID_MQTT)){
86        MQTT.subscribe(TOPIC_SUBSCRIBE, 1);
87        digitalWrite(ledBroker, HIGH);
88      }
89      else{
90        digitalWrite(ledBroker,LOW);
91        delay(5000);
92      }
93    }
94  }
95  void recebePacote( char* topic, byte* payload, unsigned int length)
       {
96    char msg[3];
97    digitalWrite(ledReceive, HIGH);
98    delay_receive = millis();
99    for(int i = 0; i<length;i++)
100   {
101     msg[i] = (char)payload[i];
102   }
103   Serial.print("theta_x: ");
104   Serial.print((int)msg[0]);Serial.print("\t");
105   Serial.print("theta_y: ");
106   Serial.print((int)msg[1]);Serial.print("\t");
107   Serial.print("theta_z: ");
108   Serial.println((int)msg[2]);
109   moveArm(1,(int)msg[2]);
110   moveArm(2,(int)msg[1]);
```

```
111   moveArm(3,(int)msg[0]);
112
113 }
114
115 void moveArm(int servo,int pos){
116   switch(servo)
117   {
118     case 1:
119       servo1.write(map(pos, 45, 135, 15, 170));
120       break;
121     case 2:
122       servo2.write(map(pos, 45, 135, 45, 135));
123       break;
124     case 3:
125       servo3.write(map(pos, 45, 135, 30, 125));
126       break;
127     case 4:
128       servo4.write(map(pos, 0, 90, 15, 45));
129       break;
130     default:
131       break;
132   }
133 }
```

APÊNDICE C – Software do Computador

C.1 Classe JFMain.java

```
1  package mqtt;
2
3
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import javax.swing.JPanel;
7  import javax.swing.Timer;
8  import com.sun.j3d.utils.geometry.Box;
9  import com.sun.j3d.utils.universe.SimpleUniverse;
10 import java.awt.BorderLayout;
11 import java.awt.Dimension;
12 import java.awt.GraphicsConfiguration;
13 import java.io.UnsupportedEncodingException;
14 import javax.media.j3d.Appearance;
15 import javax.media.j3d.BranchGroup;
16 import javax.media.j3d.Canvas3D;
17 import javax.media.j3d.ColoringAttributes;
18 import javax.media.j3d.PolygonAttributes;
19 import javax.media.j3d.Transform3D;
20 import javax.media.j3d.TransformGroup;
21 import javax.swing.JSlider;
22
23 /**
24  *
25  * @author Joao Pedro Quintino
26  */
27 public class JFMain extends javax.swing.JFrame {
28
```

```
29      static ClienteMQTT clienteMQTT = new ClienteMQTT("tcp://broker.
           mqttdashboard.com:1883", "iatrqott", "gW7u8bMQzjOm");
30      private static View objeto = new View(0);
31      private static Integer y = 0, x = 0, z=0, ga = 0;
32      private static char aux;
33
34      public static void setLabel( byte c[]) throws
           UnsupportedEncodingException {
35          aux = (char) c[0];
36          x = (int) aux;  // Valor de angulo no eixo x
37          x += (90 - x) * 2;  //Ajuste para corrigir espelhamento
38
39          aux = (char)c[1];
40          y = (int)aux;   // Valor de angulo no eixo y
41          y += 180;       // Ajuste para corrigir orientacao
42
43          aux = (char)c[2];
44          z = (int)aux;   // Valor de angulo no eixo z
45
46          aux = (char) c[3];
47          ga = ((int) aux);   // Valor de abertura da garra
48
49          objeto.rotate(x, y, z); // Rotaciona objeto 3D
50      }
51      TimerToLabel timer_label;
52
53      public JFMain() {
54          initComponents();
55          this.timer_label = new TimerToLabel(100, jSGarra);//
               Atualiza Slider a cada 100ms
56          this.timer_label.init_timer();
57
58      }
59
60      @SuppressWarnings("unchecked")
61      // <editor-fold defaultstate="collapsed" desc="Generated Code">
62      private void initComponents() {
63
64          jPMain = new javax.swing.JPanel();
65          panel3D = new javax.swing.JScrollPane(objeto);
66          jLabel4 = new javax.swing.JLabel();
67          jPanel2 = new javax.swing.JPanel();
```

```
68          jSGarra = new javax.swing.JSlider();
69          jBExit = new javax.swing.JButton();
70
71          setDefaultCloseOperation(javax.swing.WindowConstants.
               EXIT_ON_CLOSE);
72          setMaximizedBounds(new java.awt.Rectangle(0, 0, 270, 800));
73          setMinimumSize(new java.awt.Dimension(280, 527));
74          setSize(new java.awt.Dimension(0, 0));
75          addWindowListener(new java.awt.event.WindowAdapter() {
76              public void windowClosed(java.awt.event.WindowEvent evt
                   ) {
77                  formWindowClosed(evt);
78              }
79          });
80
81          jPMain.setBackground(java.awt.SystemColor.textInactiveText)
               ;
82          jPMain.setAlignmentX(0.0F);
83          jPMain.setAlignmentY(0.0F);
84          jPMain.setMaximumSize(new java.awt.Dimension(270, 1000));
85          jPMain.setPreferredSize(new java.awt.Dimension(270, 527));
86
87          panel3D.setHorizontalScrollBarPolicy(javax.swing.
               ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
88          panel3D.setVerticalScrollBarPolicy(javax.swing.
               ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);
89          panel3D.setMaximumSize(new java.awt.Dimension(240, 240));
90          panel3D.setMinimumSize(new java.awt.Dimension(240, 240));
91          panel3D.setPreferredSize(new java.awt.Dimension(240, 240));
92
93          jLabel4.setText("Movimentacao:");
94
95          jPanel2.setMaximumSize(new java.awt.Dimension(240, 90));
96          jPanel2.setPreferredSize(new java.awt.Dimension(240, 90));
97
98          javax.swing.GroupLayout jPanel2Layout = new javax.swing.
               GroupLayout(jPanel2);
99          jPanel2.setLayout(jPanel2Layout);
100         jPanel2Layout.setHorizontalGroup(
101             jPanel2Layout.createParallelGroup(javax.swing.
                   GroupLayout.Alignment.LEADING)
102             .addGroup(jPanel2Layout.createSequentialGroup()
```

```
103                 .addContainerGap(javax.swing.GroupLayout.
                        DEFAULT_SIZE, Short.MAX_VALUE)
104                 .addComponent(jSGarra, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 222, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
105                 .addContainerGap())
106         );
107         jPanel2Layout.setVerticalGroup(
108             jPanel2Layout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
109             .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                    jPanel2Layout.createSequentialGroup()
110                 .addContainerGap()
111                 .addComponent(jSGarra, javax.swing.GroupLayout.
                        DEFAULT_SIZE, 27, Short.MAX_VALUE)
112                 .addGap(52, 52, 52))
113         );
114
115         jBExit.setText("Sair");
116         jBExit.setHorizontalTextPosition(javax.swing.SwingConstants
                .CENTER);
117         jBExit.addMouseListener(new java.awt.event.MouseAdapter() {
118             public void mouseClicked(java.awt.event.MouseEvent evt)
                    {
119                 jBExitMouseClicked(evt);
120             }
121         });
122         jBExit.addActionListener(new java.awt.event.ActionListener
                () {
123             public void actionPerformed(java.awt.event.ActionEvent
                    evt) {
124                 jBExitActionPerformed(evt);
125             }
126         });
127
128         javax.swing.GroupLayout jPMainLayout = new javax.swing.
                GroupLayout(jPMain);
129         jPMain.setLayout(jPMainLayout);
130         jPMainLayout.setHorizontalGroup(
131             jPMainLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
132             .addGroup(jPMainLayout.createSequentialGroup()
```

```
133                 .addGap(0, 0, Short.MAX_VALUE)
134                 .addComponent(jLabel4)
135                 .addGap(167, 167, 167))
136             .addGroup(jPMainLayout.createSequentialGroup()
137                 .addContainerGap(14, Short.MAX_VALUE)
138                 .addGroup(jPMainLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.TRAILING)
139                     .addComponent(jBExit)
140                     .addGroup(jPMainLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.TRAILING,
                            false)
141                         .addComponent(jPanel2, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 242, javax.
                                swing.GroupLayout.PREFERRED_SIZE)
142                         .addComponent(panel3D, javax.swing.
                                GroupLayout.DEFAULT_SIZE, javax.swing.
                                GroupLayout.DEFAULT_SIZE, Short.
                                MAX_VALUE)))
143                 .addContainerGap(14, Short.MAX_VALUE))
144         );
145         jPMainLayout.setVerticalGroup(
146             jPMainLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
147             .addGroup(jPMainLayout.createSequentialGroup()
148                 .addContainerGap(44, Short.MAX_VALUE)
149                 .addComponent(jLabel4)
150                 .addPreferredGap(javax.swing.LayoutStyle.
                        ComponentPlacement.RELATED)
151                 .addComponent(panel3D, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
152                 .addGap(59, 59, 59)
153                 .addComponent(jPanel2, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
154                 .addGap(40, 40, 40)
155                 .addComponent(jBExit)
156                 .addContainerGap(javax.swing.GroupLayout.
                        DEFAULT_SIZE, Short.MAX_VALUE))
157         );
```

```
158
159         javax.swing.GroupLayout layout = new javax.swing.
                GroupLayout(getContentPane());
160         getContentPane().setLayout(layout);
161         layout.setHorizontalGroup(
162             layout.createParallelGroup(javax.swing.GroupLayout.
                    Alignment.LEADING)
163             .addComponent(jPMain, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE
                    , javax.swing.GroupLayout.PREFERRED_SIZE)
164         );
165         layout.setVerticalGroup(
166             layout.createParallelGroup(javax.swing.GroupLayout.
                    Alignment.LEADING)
167             .addComponent(jPMain, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                    Short.MAX_VALUE)
168         );
169
170         pack();
171     }// </editor-fold>
172
173     private void formWindowClosed(java.awt.event.WindowEvent evt) {
174         clienteMQTT.finalizar(); // Finaliza a comunicacao com o
                Broker
175         System.exit(1);
176     }
177
178     private void jBExitActionPerformed(java.awt.event.ActionEvent
            evt) {
179
180     }
181
182     private void jBExitMouseClicked(java.awt.event.MouseEvent evt)
            {
183         clienteMQTT.finalizar(); // Finaliza a comunicacao com o
                Broker
184         System.exit(1);
185     }
186
187     public static void main(String args[]) throws Exception {
188
```

```
189         //<editor-fold defaultstate="collapsed" desc=" Look and
               feel setting code (optional) ">
190         /* If Nimbus (introduced in Java SE 6) is not available,
               stay with the default look and feel.
191          * For details see http://download.oracle.com/javase/
               tutorial/uiswing/lookandfeel/plaf.html
192          */
193         try {
194             for (javax.swing.UIManager.LookAndFeelInfo info : javax
                   .swing.UIManager.getInstalledLookAndFeels()) {
195                 if ("Nimbus".equals(info.getName())) {
196                     javax.swing.UIManager.setLookAndFeel(info.
                           getClassName());
197                     break;
198                 }
199             }
200         } catch (ClassNotFoundException ex) {
201             java.util.logging.Logger.getLogger(JFMain.class.getName
                   ()).log(java.util.logging.Level.SEVERE, null, ex);
202         } catch (InstantiationException ex) {
203             java.util.logging.Logger.getLogger(JFMain.class.getName
                   ()).log(java.util.logging.Level.SEVERE, null, ex);
204         } catch (IllegalAccessException ex) {
205             java.util.logging.Logger.getLogger(JFMain.class.getName
                   ()).log(java.util.logging.Level.SEVERE, null, ex);
206         } catch (javax.swing.UnsupportedLookAndFeelException ex) {
207             java.util.logging.Logger.getLogger(JFMain.class.getName
                   ()).log(java.util.logging.Level.SEVERE, null, ex);
208         }
209         //</editor-fold>
210 
211 
212         try {
213             //Executa o CMSClient
214             Runtime r = Runtime.getRuntime();
215             r.exec("C://Program Files (x86)/CMSClient/CMSClient.exe
                   ");
216 
217         } catch (java.io.IOException e) {
218             e.printStackTrace();
219         }
220 
```

```
221         java.awt.EventQueue.invokeLater(new Runnable() {
222             @Override
223             public void run() {
224                 new JFMain().setVisible(true);//Inicia a interface
225             }
226         });
227 
228         clienteMQTT.iniciar();  //Inicia a comunicacao MQTT
229         new Ouvinte(clienteMQTT, "b_1jpsq", 0); //Assina Topico no
               Broker
230     }
231 
232     //Classe para atualizar o Slider constantemente
233     private class TimerToLabel implements ActionListener {
234 
235         private Timer timer;
236         private final JSlider sliderGarra;
237         private final int delay;
238 
239         public TimerToLabel(final int delay, final JSlider garra) {
240             this.delay = delay;
241             this.sliderGarra = garra;
242         }
243 
244         public void init_timer() {
245             timer = new Timer(delay, this);
246             timer.start();
247         }
248 
249         // Atualiza os valores do slider no tempo configurado
250         public void actionPerformed(final ActionEvent e) {
251             sliderGarra.setValue(ga);
252             sliderGarra.updateUI();
253         }
254 
255     }
256 
257     // Variables declaration - do not modify
258     private javax.swing.JButton jBExit;
259     private javax.swing.JLabel jLabel4;
260     private javax.swing.JPanel jPMain;
261     private javax.swing.JPanel jPanel2;
```

```
262     private javax.swing.JSlider jSGarra;
263     private static javax.swing.JScrollPane panel3D;
264     // End of variables declaration
265 }
```

C.2 Classe ClienteMQTT.java

```
1  package mqtt;
2
3  /**
4   *
5   * @author Joao Pedro Quintino
6   */
7  import java.util.Arrays;
8  import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
9  import org.eclipse.paho.client.mqttv3.IMqttMessageListener;
10 import org.eclipse.paho.client.mqttv3.IMqttToken;
11 import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
12 import org.eclipse.paho.client.mqttv3.MqttClient;
13 import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
14 import org.eclipse.paho.client.mqttv3.MqttException;
15 import org.eclipse.paho.client.mqttv3.MqttMessage;
16 import org.eclipse.paho.client.mqttv3.persist.
      MqttDefaultFilePersistence;
17
18 public class ClienteMQTT implements MqttCallbackExtended {
19
20     private final String serverURI;
21     private MqttClient client;
22     private final MqttConnectOptions mqttOptions;
23
24     public ClienteMQTT(String serverURI, String usuario, String
          senha) {
25         this.serverURI = serverURI;
26
27         mqttOptions = new MqttConnectOptions();
28         mqttOptions.setMaxInflight(200);
29         mqttOptions.setConnectionTimeout(3);
30         mqttOptions.setKeepAliveInterval(10);
31         mqttOptions.setAutomaticReconnect(true);
32         mqttOptions.setCleanSession(false);
33
```

```
34          if (usuario != null && senha != null) {
35              mqttOptions.setUserName(usuario);
36              mqttOptions.setPassword(senha.toCharArray());
37          }
38      }
39
40      public IMqttToken subscribe(int qos, IMqttMessageListener
            gestorMensagemMQTT, String... topicos) {
41          if (client == null || topicos.length == 0) {
42              return null;
43          }
44          int tamanho = topicos.length;
45          int[] qoss = new int[tamanho];
46          IMqttMessageListener[] listners = new IMqttMessageListener[
                tamanho];
47
48          for (int i = 0; i < tamanho; i++) {
49              qoss[i] = qos;
50              listners[i] = gestorMensagemMQTT;
51          }
52          try {
53              return client.subscribeWithResponse(topicos, qoss,
                    listners);
54          } catch (MqttException ex) {
55              System.out.println(String.format("Erro ao se inscrever
                    nos topicos %s - %s", Arrays.asList(topicos), ex));
56              return null;
57          }
58      }
59
60      public void unsubscribe(String... topicos) {
61          if (client == null || !client.isConnected() || topicos.
                length == 0) {
62              return;
63          }
64          try {
65              client.unsubscribe(topicos);
66          } catch (MqttException ex) {
67              System.out.println(String.format("Erro ao se
                    desinscrever no topico %s - %s", Arrays.asList(
                    topicos), ex));
68          }
```

```
69      }
70
71      public void iniciar() {
72          try {
73              System.out.println("Conectando no broker MQTT em " +
                    serverURI);
74              client = new MqttClient(serverURI, String.format("
                    cliente_java_%d", System.currentTimeMillis()), new
                    MqttDefaultFilePersistence(System.getProperty("java.
                    io.tmpdir")));
75              client.setCallback(this);
76              client.connect(mqttOptions);
77          } catch (MqttException ex) {
78              System.out.println("Erro ao se conectar ao broker mqtt
                    " + serverURI + " - " + ex);
79          }
80      }
81
82      public void finalizar() {
83          if (client == null || !client.isConnected()) {
84              return;
85          }
86          try {
87              client.disconnect();
88              client.close();
89          } catch (MqttException ex) {
90              System.out.println("Erro ao desconectar do broker mqtt
                    - " + ex);
91          }
92      }
93
94      public void publicar(String topic, byte[] payload, int qos) {
95          publicar(topic, payload, qos, false);
96      }
97
98      public synchronized void publicar(String topic, byte[] payload,
            int qos, boolean retained) {
99          try {
100             if (client.isConnected()) {
101                 client.publish(topic, payload, qos, retained);
102                 System.out.println(String.format("Topico %s
                        publicado. %dB", topic, payload.length));
```

```
103             } else {
104                 System.out.println("Cliente desconectado, nao foi
                       possivel publicar o topico " + topic);
105             }
106         } catch (MqttException ex) {
107             System.out.println("Erro ao publicar " + topic + " - "
                   + ex);
108         }
109     }
110 
111     @Override
112     public void connectionLost(Throwable thrwbl) {
113         System.out.println("Conexao com o broker perdida -" +
               thrwbl);
114     }
115 
116     @Override
117     public void connectComplete(boolean reconnect, String serverURI
           ) {
118         System.out.println("Cliente MQTT " + (reconnect ? "
               reconectado" : "conectado") + " com o broker " +
               serverURI);
119     }
120 
121     @Override
122     public void deliveryComplete(IMqttDeliveryToken imdt) {
123     }
124 
125     @Override
126     public void messageArrived(String topic, MqttMessage mm) throws
           Exception {
127     }
128 
129 }
```

C.3 Classe Ouvinte.java

```
1 
2 package mqtt;
3 
4 /**
5  *
```

```
6   * @author Joao Pedro Quintino
7   */
8  import org.eclipse.paho.client.mqttv3.IMqttMessageListener;
9  import org.eclipse.paho.client.mqttv3.IMqttToken;
10 import org.eclipse.paho.client.mqttv3.MqttMessage;
11
12 public class Ouvinte implements IMqttMessageListener {
13
14     public Ouvinte(ClienteMQTT clienteMQTT, String topico, int qos)
           {
15         clienteMQTT.subscribe(qos, this, topico);
16     }
17
18     @Override
19     public void messageArrived(String topico, MqttMessage mm)
           throws Exception {
20         byte c[]={0,0,0,0};
21         c[0] = mm.getPayload()[0];
22         c[1] = mm.getPayload()[1];
23         c[2] = mm.getPayload()[2];
24         c[3] = mm.getPayload()[3];
25         System.out.println("Mensagem recebida:");
26         System.out.println("\tX: " + c[0] +"\tY: "+ c[1] + "\tZ: "
               + c[2] + "\tGarra: " + c[3]);
27         System.out.println("");
28         JFMain.setLabel(c);
29     }
30
31 }
```