Desenvolvimento de uma Interface de Monitoração Remota para o Sistema Robótico ROBIX, Integrando o Protocolo MQTT e oROS

Daniel Zolett
Universidade do Vale do Itajaí - UNIVALI
daniel.zolett@edu.univali.br

Alejandro Rafael Garcia Ramirez
Universidade do Vale do Itajaí - UNIVALI
ramirez@univali.br

ABSTRACT

The history of the industry has been undergoing many changes, improving production methodologies and setting new goals. The industry is currently facing a new challenge called Industry 4.0. This is the result of technological developments in areas such as the Internet of Things (IoT), Information and Communication Technology (ICT) and Automation Systems. The goal is to define a more dynamic and efficient production line, improving production in terms of resources and time. However, in the current scenario, there are some difficulties to overcome. One of these difficulties is the existence of various communication protocols, being many proprietary and closed, which hinders the interoperability between machines. For this reason, there is a need to standardize communication protocols, facilitating the integration of all elements in the production line. Given this context, this paper presents the integration of the MQTT protocol with the ROS tool to control a robotic arm via Web. The angles of the robot links movements are informed via web interface. The interface communicates with the robotic arm via the MQTT protocol. The arm control software was developed using the ROS tool as well as its simulation environment. The software is implemented on the Raspbarry PI 3 development board and the robot servo motors are driven by the Arduino Mega kit processor. Data exchange between these elements is done through serial communication. Tests were designed to assess communication latency to validate the feasibility of using the MQTT protocol for this scenario. The tests proved that the latency average of the MQTT protocol, considering the worst case scenario, was 300 milliseconds, which is within the range of the studies consulted in the literature.

KEYWORDS

Indsstria 4.0, Protocolo IoT, ROS

1 INTRODUÇÃO

No decorrer do desenvolvimento industrial, existiram revoluções tecnológicas que resultaram no que conhecemos hoje como indústria. Essas revoluções introduziram novas formas de produzir, buscando melhorar e dinamizar o processo de produção. Tendo como base a última revolução industrial, pode-se observar a introdução da automação baseada na eletrônica e tecnologias de internet [1]. Este fato trouxe para a indústria, tecnologias que permitiram facilitar o controle de múltiplas ferramentas industriais (Controlador Lógico Programável) [2]. A evolução das tecnologias da internet levou a indústria para uma nova revolução, denominada de indústria 4.0.

A indústria 4.0, foi um termo criado na Alemanha em 2011, com o objetivo de impulsionar o desenvolvimento industrial do país. Este conceito surgiu com o objetivo de introduzir a ideia de uma indústria totalmente integrada [3]. Sendo esse objetivo possível pelo desenvolvimento de tecnologias como: *Internet of Things* (IoT), sistemas ciber-físicos (CPS), automação e computação em nuvem, dentre outras tecnologias ligadas à internet. Desta forma, a indústria 4.0 está relacionada ao desenvolvimento das tecnologias da Internet no ambiente fabril [4]. Os objetivos a serem alcançados são: a interoperabilidade, virtualização, descentralização, capacidade em tempo real, orientação a serviços, modularidade e atingir níveis mais altos de produtividade e automação.

Para alcançar os objetivos da indústria 4.0 existem diversos desafios a serem superados. Um destes desafios se apresenta na comunicação entre as máquinas e a internet. Esse desafio surge pelo legado introduzido na revolução anterior, na qual muitos protocolos proprietários foram desenvolvidos com o objetivo de melhorar a eficiência da comunicação entre as máquinas. Porém esses protocolos não atendem as novas necessidades da indústria 4.0 e nem apresentam uma padronização a ser seguida. Isso porque são protocolos proprietários de código fechado e não foram desenvolvidos com o intuito de oferecerem uma comunicação eficiente com a internet. Sendo assim, surge a necessidade de uniformizar os protocolos de comunicação dentro da indústria, a fim de estabelecer uma mesma "linguagem" para todas as máquinas [5].

No nível de comunicação entre a máquina e a internet, há diversos protocolos já consolidados no cenário da IoT, tais como o MQTT (*Message Queuing Telemetry Transport*), CoAP (*Constrained Application Protocol*) e o AMQP (*Advanced Message Queuing Protocol*), as quais são opções a serem consideradas. Isso porque estes protocolos foram desenvolvidos para serem mais rápidos, leves e totalmente escaláveis. Já no nível de criação e suporte de soluções robóticas, ferramentas como ORIN (*Open Resource Interface for the Network*) e ROS (*Robot Operating System*) oferecem padronização, modularização e abstração de hardware para as soluções. Tais ferramentas permitem que os softwares desenvolvidos sejam robustos e facilmente adaptáveis.

Na literatura, são encontrados estudos que buscam comparar protocolos consolidados no ambiente IoT para cenários de indústria 4.0. Os protocolos mais abordados na literatura para esta finalidade são o MQTT, CoAP e o AMQP. O estudo de [6] apresenta a implementação de testes para evidenciar as características destes protocolos. Os testes feitos se concentraram em definir as seguintes métricas: Tamanho da mensagem, latência e confiabilidade. Dentro dos resultados alcançados é possível concluir que o protocolo MQTT, apresenta melhores métricas para o cenário de indústria 4.0. Isso se deve pelo fato de o protocolo apresentar um menor cabeçalho fixo. [7] reforça essa afirmação em seu artigo. Este protocolo apresenta menor consumo de energia para mensagens maior que 1 KB. Esta afirmação também é encontrada em [8]. O MQTT também

apresentou uma maior confiabilidade na entrega das mensagens [7].

Os estudos feitos por [9–11] evidenciam o desempenho do protocolo MQTT. Nestes estudos, são utilizadas métricas relacionadas ao tempo de ida e volta das mensagens. Sendo aplicando para isso, um teste no qual o objetivo é enviar múltiplas mensagens de cliente para cliente, sobre diferentes situações de rede e tamanho de mensagem. Por fim, os estudos apresentam um tempo médio de latência para o MQTT de 500 milissegundos.

Estudos da implementação do ROS para o cenário industrial, são demonstrados em [2] e [12]. Nestes trabalhos são evidenciados as funcionalidades e os benefícios da ferramenta ROS, como alternativa para a padronização no desenvolvimento de soluções robóticas. No entanto, a utilização da ferramenta foi explorada apenas em nível de simulação, não sendo aplicada em um cenário real com robôs.

À luz destes estudos, este artigo busca apresentar a implementação do protocolo MQTT integrado ao ROS. O objetivo é evidenciar a capacidade do protocolo MQTT de se tornar uma possível solução para a comunicação com robôs na indústria 4.0. Além de demonstrar os benefícios da utilização da ferramenta ROS. Assim sendo, este estudo buscou construir uma interface de monitoramento remoto, via WEB, para um braço robótico ROBIX, apresentando uma implementação prática em um cenário real das tecnologias envolvidas.

Este artigo está dividido da seguinte forma: na seção 2 se apresenta a fundamentação teórica, relatando as tecnologias utilizadas neste projeto. Na seção 3 são apresentados os trabalhos que ajudaram a fundamentar o desenvolvimento deste projeto. Na seção 4, Desenvolvimento, se descreve a implementação. A seção 5, apresenta os resultados obtidos nos testes de latência. Por fim, na seção 6 são apresentadas as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEORICA

Nesta seção é descrito os conceitos das tecnologias utilizadas neste artigo. O objetivo desta seção e evidenciar as características do protocolo MQTT e da ferramenta ROS.

2.1 MQTT

O protocolo MQTT foi proposto pelo Dr. Andy Stanford-Clark, da IBM, e Arlen Nipper, da Arcom (agora Eurotech), em 1999. O objetivo desta proposta era de desenvolver um protocolo aberto, simples e fácil de implementar [13]. No entanto, o protocolo só se tornou público em 2010 com sua versão 3.1, disponibilizada pela IBM [14]. E em outubro de 2014 veio a se tornar um padrão OASIS com sua versão 3.1.1 [15].

O MQTT é um protocolo de camada de aplicação que opera sobre o protocolo TCP na camada de transporte. Foi desenvolvido para ser aplicado em ambientes com redes e dispositivos restritos. O funcionamento do protocolo é baseado em uma arquitetura de publicação/assinatura. Neste tipo de arquitetura existem dois elementos principais: o broker e o cliente. O broker pode ser visto como um controlador e é o responsável por coordenar o fluxo de mensagem. Esse controle é feito através dos tópicos, que podem ser vistos como endereços de memória especiais [16]. Os clientes são os nós, finais responsáveis por gerar e consumir os dados [17]. Um sistema MQTT funciona da seguinte maneira: Os clientes se anunciam ao broker, indicando qual é sua função (publicar ou assinar) e qual o tópico. Em sequência, o broker cria o tópico e gerencia o fluxo de mensagens entre o publicador e o assinante. Tudo feito sobre o protocolo TCP na camada de transporte. A Figura 1 exemplifica o fluxo de comunicação em uma rede MQTT. Pode-se observar a interação entre os nós, sendo possível através do broker.

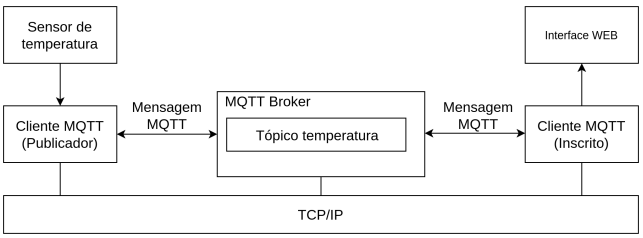

Figura 1: Fluxo de dados do protocolo MQTT.

A comunicação dentro de um sistema MQTT é feita através de troca de massagens. As mensagens do protocolo são compostas por três campos: o cabeçalho fixo, o variável e o payload. O cabeçalho fixo tem o tamanho fixo de 2 bytes e está sempre presente nas mensagens. Este cabeçalho, tem a finalidade de comportar as informações da mensagem como o tipo, flags de configuração e o tamanho máximo da mensagem. O cabeçalho variável comporta informações de controle, para sinalizar as ações que o broker deve executar. Este cabeçalho existe em alguns tipos de mensagens e está localizado entre o cabeçalho fixo e o payload. Por fim, a mensagem MQTT apresenta um payload, no qual os dados então contidos. Uma mensagem MQTT pode ter um tamanho máximo de 256 MB. Vale apena ressaltar, que apenas o cabeçalho fixo é obrigatório nas mensagens, os outros campos variam com o tipo da mensagem [15]. A Figura 2, demonstra como os campos de uma mensagem MQTT estão organizados. Nota-se que as configurações importantes das mensagens estão no cabeçalho fixo, deixando o resto da mensagem opcional para tipos específicos. Deste modo, uma mensagem MQTT pode ter um tamanho mínimo de 2 Bytes.

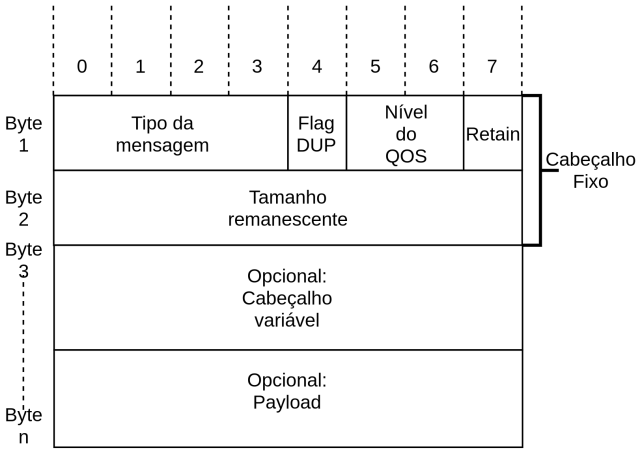

Figura 2: Formato da mensagem MQTT.

O protocolo MQTT disponibiliza três níveis de QoS. Uma mensagem MQTT pode ser configurada para atender um destes níveis.

Esses níveis garantem diferentes confiabilidades para o envio e o recebimento da mensagem. O nível zero, garante apenas o envio, não se preocupando com o recebimento. O nível um, exige que o recebimento seja confirmado pelo menos uma vez, não impedindo múltiplos recebimentos. O nível dois, garante que a mensagem seja recebida exatamente uma vez, impedindo múltiplos recebimentos. Esses QoSs, influenciam diretamente na latência de comunicação entre os clientes, por exigir que exista uma política de confirmação resposta que será ministrada pelo broker. No entanto, esses níveis de QoSs funcionam como um mecanismo de segurança no envio das mensagens [15].

2.2 ROS

O ROS é um conjunto de bibliotecas e ferramentas que visa oferecer funcionalidades de um sistema operacional para soluções robóticas. Ao utilizar o ROS, obtém-se recursos de sistemas operacionais como: a abstração de hardware, gestão de pacotes, passagem de mensagens entre processos, controle de entrada e saída, entre outras [18]. A conveniência de se utilizar o ROS, está na padronização e na modularização do software de controle desenvolvido. Padronização por permitir que com poucas alterações, a mesma solução seja utilizada para cenários semelhantes. Modularização por ser uma ferramenta composta por módulos, sendo assim a solução utilizara apenas o necessário. Por esse motivo o ROS pode ser visto como uma ferramenta de suporte para robôs.

O funcionamento do ROS se assemelha ao MQTT por trabalhar sobre uma arquitetura de publicação/assinatura, e utilizar versões ROS dos protocolos TCP e UDP. Ambas versões chamadas de TCPROS e UDPROS. Devido a utilização desta arquitetura, o ROS é composto por dois elementos. O mestre, que funciona como o broker MQTT e os nós, que funcionam como os clientes. Assim como o MQTT, o ROS controla o fluxo de dados através dos tópicos que são coordenados pelo mestre [19]. No entanto, um sistema ROS apresenta diferenças quanto a sua forma de comunicação e características do broker. A comunicação é feita através de tópicos com tipagem definida. Assim sendo, um tópico será criado com um único tipo e recebera inscritos e publicadores apenas deste tipo.

O ROS pode suportar o fluxo de dados de tipos primitivos como boolean, inteiro, float. Também suporta a utilização de estrutura que combina esses tipos primitivos. Assim, o ROS permite criar tópicos específicos para o fluxo de determinada estrutura de dados. Quanto ao mestre ROS, ele apresenta maiores funcionalidades que o broker MQTT. O mestre ROS pode executar serviços, tarefas, apresentar ferramentas de monitoramento e simulação, entre outras funcionalidades presentes em sistemas operacionais. A Figura 3 demonstra o fluxo básico de comunicação em um sistema ROS. Nesta Figura é exemplificado uma câmera, que obtêm os dados e os publica em um tópico onde os nós interessados podem obtê-los. É interessante notar que em um primeiro momento, os nós se registram ao mestre, informando o seu objetivo.

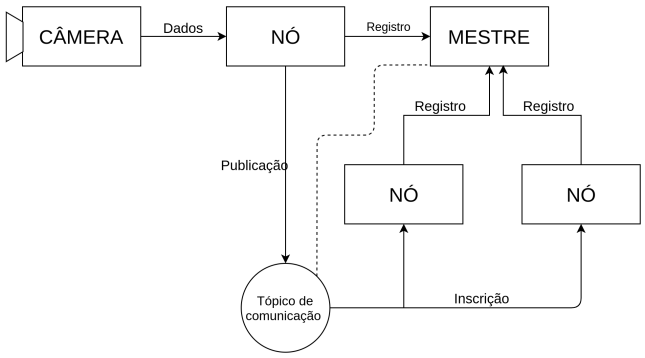

Figura 3: Fluxo de dados em um sistema ROS.

3 TRABALHOS CORRELATOS

Nesta seção, são descritos trabalhos que dão base ao desenvolvimento deste projeto. Neles são evidenciados a utilização do protocolo MQTT e da ferramenta ROS dentro de um contexto semelhante ao deste projeto. Para a seleção destes trabalhos foram utilizadas as seguintes bases de dados: IEEE Xplore, Scopus, Google Scholar e ACM.

Em [20], é apresentado um projeto de integração entre um veículo de quatro rodas e um interface WEB através do protocolo MQTT. No veículo foi usado um Raspberry pi 3, responsável por conectar o veículo a internet. Para implementar a interface WEB e o broker, foi utilizado um computador pessoal. A comunicação foi feita através da troca de dados em formato JSON. Os testes de latência foram elaborados através da criação de 10 seções, na qual efetuavam a transmissão contínua de dados aleatórios para o veículo. Para determinar o tempo de transmissão, os autores utilizaram a função time do Python, implementando-a no veículo. Como conclusão, os autores obtiveram uma latência média de comunicação de 75,1 milissegundos.

No trabalho de [21], é proposto uma solução baseada no protocolo MQTT, para operar remotamente um braço robótico. O objetivo foi efetuar a operação do robô pelo giroscópio do smartphone. Os autores implementaram a solução desenvolvida utilizando dois servidores para hospedar o MQTT, público e privado. Para verificar a latência dos servidores, os autores aplicaram teste de ida e volta junto a um script em Python, para o registro de tempo. Por fim, os autores concluem que a latência em um servidor público é maior, tendo uma média de 500 milissegundos. No entanto, em um servidor privado a latência média foi de 80 milissegundos.

Os autores de [2], demonstraram a utilização do sistema ROS com o pacote de software *Move it*! Integrado ao MQTT. O objetivo dos autores é permitir que um cliente envie dados para a simulação através de uma interface WEB. Para medir a latência de comunicação, os autores mediram a diferença do tempo de envio com o tempo de recebimento da mensagem. Este teste foi executado para 1200 mensagens. Com isso os autores obtiveram uma latência média de 319,6 milissegundos.

Em [12], é demonstrada a implementação do ROS juntamente com o ROS-I (ROS-Industrial) e o *Move it*!. O projeto tem por finalidade efetuar o planejamento e o movimento do robô SDA10F. Nesse artigo, os autores demonstram as funcionalidades que podem ser usadas para programar robô através da utilização do ROS. É evidenciado também, as funcionalidades das ferramentas de simulação, tanto para visualização quanto para o planejamento de

Tabela 1: Trabalhos correlatos

Autor	Camada de aplicação	ROS	Latência(ms)
[20]	MQTT	Não	75,1
[21]	MQTT	Não	500/80
[2]	MQTT	Sim	319,6
[12]	-	Sim	-
Este trabalho	MQTT	Sim	300/40

movimentos. Os autores concluem o artigo evidenciando as facilidades obtidas com a utilização do ROS em comparação com uma solução tradicional.

Com base nos trabalhos descritos anteriormente, este projeto aborda uma implementação não explorada por esses autores. Nota-se que nestes trabalhos não houve uma integração direta entre MQTT, ROS e as ferramentas de simulação com um sistema físico. O trabalho que mais se aproxima a este, é o de [2], porém o autor se limita ao contexto da simulação. Por tanto, este trabalho aborda a integração destas tecnologias junto a um robô físico. Com isso, busca-se evidenciar os resultados da integração entre as tecnologias, apresentando a viabilidade da utilização do MQTT na indústria 4.0 através da métrica de latência. Na Tabela 1, é apresentado de forma resumida, as tecnologias utilizadas em cada trabalho citado assim como a latência obtida nas implementações que utilizam MQTT.

4 DESENVOLVIMENTO

Esta seção se dedica a evidenciar a arquitetura do projeto e demonstrar como as tecnologias foram integradas, descrevendo o fluxo de dados. A arquitetura da solução implementada consiste em dois módulos: a interface WEB e o módulo do robô. A Figura 4, apresenta a arquitetura implementada neste projeto.

O módulo da interface tem a finalidade de obter os parâmetros informados pelo usuário, e os enviar através do cliente MQTT. Esta escolha de projeto, torna a interface totalmente independente do sistema robótico a ser controlado. Além disso, a interface também conta com um sistema de retorno baseado em stream de vídeo. Os dados da stream são adquiridos e transmitidos pelo módulo do robô.

O módulo robô foi projetado para receber os dados da interface pelo protocolo MQTT e os converter para o ambiente ROS. Com as informações no ambiente ROS, é possível direcioná-las para os tópicos ROS de interesse e assim acionar os respectivos motores. O acionamento dos motores é feito através de uma interface baseada em Arduino. Esta condição é uma consequência já existente no robô ROBIX devido a projetos anteriores e por ser uma ferramenta amplamente usada no ensino de robótica. Por esses motivos, esse projeto usufrui do sistema de acionamento local já existente (Plataforma Arduino), alterando unicamente a forma de controle (Firmware). Neste módulo, também é gerado os dados para a stream de video através de uma câmera USB. Estes dados são transmitidos para a interface, utilizando o protocolo HTTP. Sendo assim, neste projeto apenas os dados de controle do robô são enviados por MQTT.

A interface WEB é desenvolvida com o auxílio de duas bibliotecas para linguem Python: o Flask e Paho MQTT. O Flask, é um framework para facilitar o desenvolvimento de aplicações WEB. Neste projeto, ele serve para construir a interface WEB e integrá-la

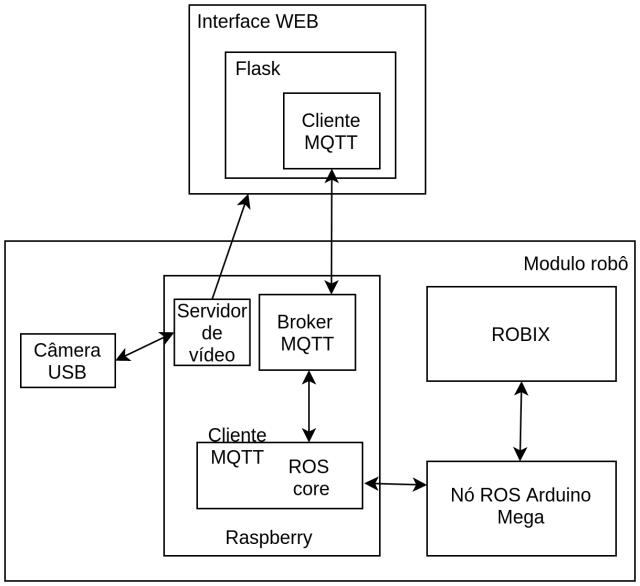

Figura 4: Arquitetura utilizada.

ao MQTT. O Paho MQTT, oferece suporte a implementação do cliente MQTT, permitindo que este cliente faça ações de publicação e assinatura nos tópicos. Esta biblioteca foi utilizada junto a interface, para permitir que os dados informados pelo cliente possam ser transmitidos ao broker MQTT. A Figura 5 demonstra a interface desenvolvida. Nela é possível observar que exitem 6 sliders na parte superior da página, no qual cada um deles representa um movimento do robô. Abaixo dos sliders, existe o botão de publicação, que é responsável por enviar os dados. Por fim, na parte inferior da página é localizado a stream de vídeo, que dá o retorno ao usuário sobre os efeitos dos comandos enviados.

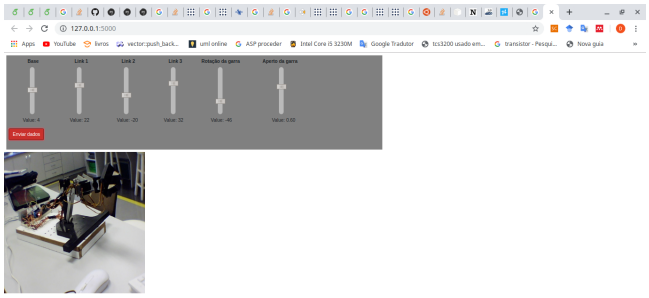

Figura 5: Interface desenvolvida.

O módulo robô é desenvolvido sobre a plataforma Raspberry pi 3 b+ e Arduino Mega. No Raspberry pi, é implementado o broker e o cliente MQTT, o ROS core (mestre), o simulador, o servidor de vídeo e um script em Python de tradução de comandos. Para isso optou-se pela utilização do sistema operacional Ubuntu 16.04, disponibilizado pela *Ubiquity Robotics*. Esta imagem do sistema é

dedicada a processadores ARM, implementados no Raspberry, e traz já instalado o sistema ROS *kinetic* básico. Ela também contém todas as ferramentas necessárias para a utilização da linguagem Python. Portanto, a utilização desta imagem garante para o projeto uma versão estável e funcional do ROS para a plataforma Raspberry. A interface Arduino Mega, é utilizada para servir de acionamento para os servomotores e assim efetuar os movimentos do braço Robix.

O broker MQTT é implementado com a utilização da ferramenta Mosquitto, disponibilizada pela Eclipse. A configuração inicial oferecida pela instalação do Mosquitto, é suficiente para atender os requisitos deste projeto. Para o consumo de dados, foi implementado um cliente MQTT, também utilizando a biblioteca paho-maqtt, para consumir os dados da interface. Desta forma, os dois clientes trocam dados através de um tópico ministrado pelo broker Mosquitto.

O simulador foi construído utilizando a ferramenta Rviz, disponível no ROS. Essa ferramenta, utiliza a descrição do robô para representar de forma visual os links e as ligações de um robô. Tal descrição é feita no formato URDF (*Unified Robot Description Format*), e pode ser comparada a uma descrição XML. Com o Rviz, é possível simular os movimentos executados pelo robô, no entanto este simulador não se atém a questões físicas atreladas ao movimento. Para este projeto, a simulação desenvolvida traz a representação simples do robô, sendo que o objetivo principal é a simulação dos movimentos. Deste modo, o simulador funciona em conjunto ao robô físico, executando o mesmo movimento. Neste projeto, o simulador é apresentado no módulo do robô, não sendo transmitido para a interface WEB. A comunicação com o simulador é feita através de tópicos específicos do Rviz. A Figura 6 apresenta o simulador desenvolvido.

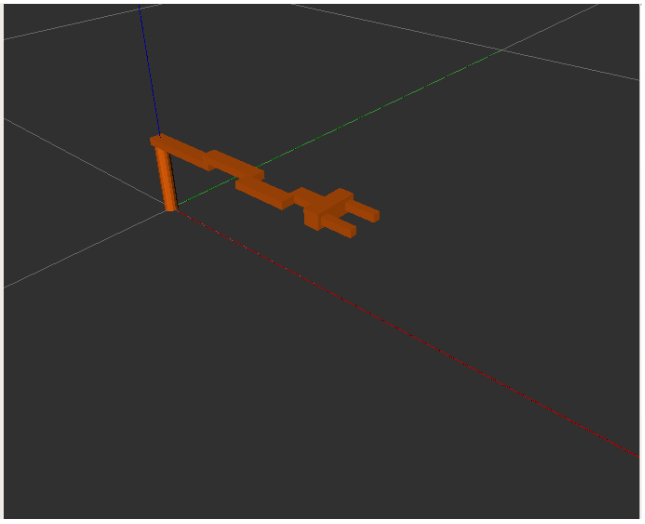

Figura 6: Representação do ROBIX no simulador Rviz.

A stream de vídeo é construída sobre uma ferramenta do ROS denominada *web-video-server*. Ela é responsável por construir um servidor local de imagens utilizando um tópico específico para isso. Sendo assim, esta ferramenta cria um servidor de transmissão de vídeo, onde é possível obter esses dados através da URL do servidor. O tópico que esta ferramenta administra, precisa receber dados de uma fonte. Neste projeto, a fonte de dados é fornecida pela câmera USB através da ferramenta *usb-cam*, também disponível no ROS. Portanto, a ferramenta *usb-cam* fornece os dados para um tópico onde a *web-video-server* constrói um servidor e disponibiliza os dados através de uma URL.

O fluxo de dados neste projeto pode ser definido da seguinte maneira: Os parâmetros de movimento do robô são informados pelo usuário através de sliders na interface WEB. Esses dados são formatados em JSON e posteriormente enviados pelo cliente MQTT da interface. O envio de dados só é possível se o usuário clicar no botão de publicar, caso contrário os dados não são formatados e nem enviados. Com os dados publicados em um tópico MQTT, é possível que o cliente no robô possa obtê-los se inscrevendo no mesmo tópico. Cabe ressaltar que no protocolo MQTT, os tópicos podem ser criados pelos clientes. Dentro do contexto deste projeto, o tópico de comunicação entre a interface e o robô, é criado no momento em que exista uma inscrição ou publicação. Sendo assim, neste caso o tópico está sendo criado na inscrição do robô, já que a publicação do usuário é mais demorada.

No módulo do robô, os dados obtidos pelo cliente MQTT são enviados para um script em Python. Neste estágio, os dados são convertidos para informações válidas no ambiente ROS. Essa conversão é necessária, pois os tópicos ROS utilizam tipos de dados diferentes para os tópicos. Para o tópico de simulação, os dados são convertidos para o tipo *joint_state_publisher*. Este tipo de dado, é uma estrutura que comporta o nome dos links, em string, e o valor do deslocamento a ser aplicado, em float64. Para a serialização, os dados são convertidos com a utilização da biblioteca Python *ros_serial*. Através dela, os dados de qualquer tipo do ambiente ROS são convertidos para estruturas interpretáveis pela plataforma Arduino.

A comunicação com o robô físico é feita através da porta serial, conectada ao processador do kit Arduino mega (ATMEGA1280) pelo protocolo RS-232. O Arduino é programado para se transformar em um nó ROS. Isso é feito pela importação da biblioteca *ROS.h*, criada pelo próprio ROS utilizando o módulo *rosserial-arduino*. Com isso, é possível inscrever o Arduino no tópico, onde serão publicados os dados de forma serializada. O código no Arduino destina-se a receber esses dados e acionar os servos motores do robô físico. Este acionamento é feito por Modulação por Largura de Pulso (PWM), e é suportado pela biblioteca nativa do Arduino *Servo.h*.

A transmissão de dados para a stream de vídeo é construída a partir da criação do tópico */image_raw*. Este tópico, é criado pela ferramenta *usb-cam* disponível no ambiente ROS. O objetivo desta ferramenta é capturar os dados gerados pela câmera USB e publicá-los no tópico */image_raw*. Com o tópico criado, a ferramenta *web-video-server* é utilizada para gerenciar esse dados, com o objetivo de construir um servidor de dados. Por fim, os dados do servidor são enviados para a interface do usuário através do protocolo HTTP. A Figura 7 representa graficamente o fluxo de dados deste projeto.

Para este projeto, foi utilizada uma estrutura mecânica do kit de desenvolvimento Robix^{TM} RCS-6, que é destinado ao ensino da robótica. Esse kit, conta com 6 servos motores Hitec HS422 e 18 links, que são utilizados para montar várias estruturas que atendem a diferentes propósitos educacionais. Para este projeto, a estrutura mais adequada é a *Chemist*, por se assemelhar mais a um robô

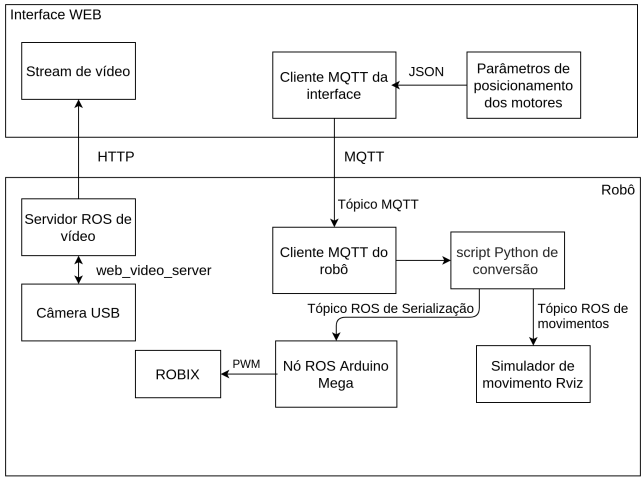

Figura 7: Fluxo de dados da solução proposta.

industrial. Esta estrutura apresenta cinco graus de mobilidade e uma pinça, que pode acomodar um objeto de 23 mm. A Figura 8 demonstra a estrutura usada para este projeto.

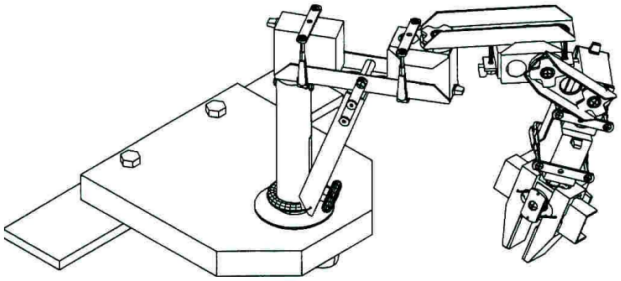

Figura 8: Estrutura do Robix *Chemist*.

5 RESULTADOS

Esta seção destina-se a apresentar os resultados dos testes de latência aplicados ao protocolo MQTT e a correspondência de monimentos entre o simulador e o robô. Nela, demonstra-se a viabilidade da utilização deste protocolo no cenário da indústria 4.0.

A compatibilidade entre a movimentação do robô físico e do simulador, foi testada de forma qualitativa, sem o uso de métricas físicas. Para os teste, foram enviados comandos para o robô, e observou-se os resultados gerados no robô físico e no simulador. A Figura 9 demonstra um dos resultados avaliados nos testes. Nesta imagem, é possível observar a compatibilidade entre as poses.

Os testes desenvolvidos para medir a latência do MQTT, foram implementados utilizando a biblioteca time do Python. Com ela é possível obter o tempo atual do sistema em segundos. Para os testes serem válidos, foi necessária uma sincronização de relógio. Isso serve para que o sistema que comporta a interface WEB e o Raspberry pi possam ter a mesma métrica de tempo. A latência do protocolo MQTT é muito dependente da disponibilidade que

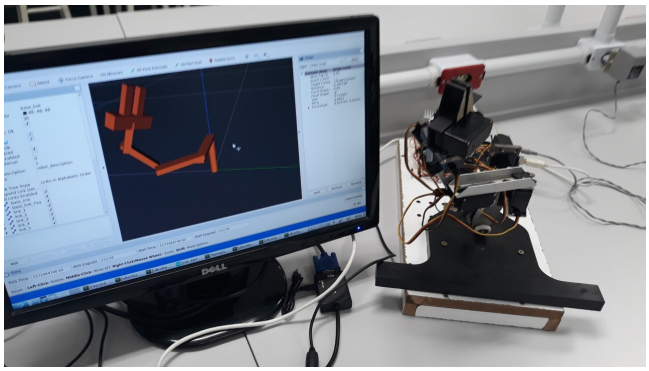

Figura 9: Demonstração da correspondência entre o simulador e o robô físico.

o broker tem para gerenciar o fluxo de dados. Por este motivo, os testes desenvolvidos avaliaram os impactos gerados pela utilização de um broker público, privado e local. Tais testes demostram a possível viabilidade da aplicação deste projeto para um cenário de computação em nuvem.

O broker público é um elemento fornecido por empresas, para permitir o teste de aplicações IoT. Ele é limitado em questão de recursos e pode se tornar instável com o tempo de uso. O broker privado, é um elemento construído sobre serviços de nuvem, como AWS da Amazom ou a Azure da Microsoft. Estes serviços têm por finalidade oferecer uma largura de banda e uma conexão mais segura e estável dentro dos requisitos do plano assinado. Isso porque esses serviços oferecem um endereço de conexão único para cada usuário. O broker local é um dispositivo que tem implementado uma ferramenta como o Mosquitto, e através dela este dispositivo dispõe da capacidade de operar como broker.

Os testes consistiram em uma rotina de envio de mensagens, da interface WEB para o robô. As mensagens geradas pela interface respeitaram os seis graus de movimentos do robô. Desta maneira, as mensagens geradas em formatação JSON continham os nomes dos links e os ângulos de movimento de cada link. Para os testes, foi acrescentada a mensagem o tempo do sistema em que ela foi gerada. Com este tempo, é possível calcular a diferença quando a mensagem é recebida no Raspberry pi. Portanto, a rotina desenvolvida executava o envio de 100 mensagens, na qual os ângulos variavam dentro dos valores limites. No Raspberry pi, os tempos de recebimento são armazenados em um vetor para posteriores cálculos.

Os testes foram realizados em cinco brokers diferentes, no qual dois deles são públicos, dois privados e um local. A Tabela 2 apresenta os valores de latência máximo, mínimo e médio para cada broker testado.

Analisando os resultados obtidos, é possível observar que o melhor cenário é a implementação local do broker MQTT. Este resultado é esperado devido a localidade do broker e o mesmo não recebe grandes influências do trefego de rede. No entanto, é possível notar que com a utilização de um broker público ou privado, a latência média não ultrapassa o valor de 300ms. Tal fato reforça a viabilidade da utilização deste protocolo para o cenário em estudo. Quanto a principal diferença entre broker público e privado, tem-se o tempo

Tabela 2: Resultados dos testes de latência

bkoker	Média (ms)	Máximo (ms)	Mínimo (ms)
test.mosquitto.org (Público)	0.26277	0.49991	0.23488
broker.hivemq.com (Público)	0.29732	0.54778	0.27431
soldier.cloudmqtt.com (Privado)	0.17156	0.32085	0.15813
mqtt.flespi.io. (Privado)	0.26314	0.47958	0.24961
Local	0.03477	0.03996	0.03372

máximo de recebimento. Nos resultados, observa-se que a latência em um broker público está em torno de 523ms, enquanto no broker privado essa latência decai para 400ms. Vale apena ressaltar, que o plano contratado para a utilização dos brokers privados foram de licença de teste. Portanto, a plataforma de serviço disponibiliza uma configuração mínima de recursos a serem utilizados. No entanto, é evidente a melhora na latência com a utilização de um broker privado. Isso é caracterizado pela disponibilidade do broker. Dentro de um cenário público, o broker deve gerenciar mais informações, que não necessariamente estão ligadas à aplicação de interesse. Em um broker privado, existe apenas a gerência das mensagens de interesse da aplicação em questão.

6 CONCLUSÃO

Neste artigo é apresentado a viabilidade da utilização do protocolo MQTT e do ROS, como sendo alternativas para a padronização da comunicação na indústria 4.0. Essas tecnologias foram escolhidas por apresentarem características que se enquadram no cenário da indústria 4.0. O MQTT tem características de ser um protocolo de comunicação rápido e leve, sendo amplamente implementado na área do IoT. Por esse motivo, esse protocolo se torna uma possível escolha para a indústria. O ROS é um conjunto de ferramentas que tem a finalidade de oferecer suporte ao desenvolvimento de soluções robóticas. A implementação do ROS na indústria, traz benefícios ligados a adaptabilidade de soluções já desenvolvidas. Isso porque o ROS funciona com a utilização de módulos, o que permite uma adaptação mais fácil da solução.

Para este artigo, foi utilizada uma arquitetura que divide a interface do usuário e o robô físico. Esta abordagem foi escolhida para facilitar a integração entre os módulos. Deste modo, é possível que a interface se comunique com outros robôs, assim como o robô receba dados de outras fontes.

Os testes executados sobre diferentes brokers, comprovam a eficiência da implementação do MQTT, utilizando um broker em nuvem. Esta comprovação é justificada pela média de latência nos brokers testados estar na média de 300ms. Na literatura são encontrados uma média de latência também de 300ms. Isso reforça a ideia de que, a implementação do protocolo MQTT em uma plataforma de nuvem está sujeita a uma latência desta magnitude. No entanto, para o desenvolvimento deste artigo, escolheu-se a aplicação de um broker local. Esta escolha se justifica pela latência alcançada ser de no máximo 40ms, e pelo controle total sobre a implementação do broker. A latência obtida com essa abordagem é menor que a obtida por [20], que em seu artigo afirma que a latência de 75,1ms é o suficiente para uma aplicação em tempo real.

Com os resultados alcançados, é evidente os benefícios de se utilizar a ferramenta ROS para o desenvolvimento da solução robótica. Com ela é obtida uma maior abstração de hardware, tornando o desenvolvimento mais dinâmico. Para a comunicação, pode-se concluir que o protocolo MQTT apresenta uma latência baixa e que no pior dos cenários ele pode chegar a 500ms.

Para trabalhos futuros, existe a necessidade da integração do simulador com a interface WEB, utilizando as ferramentas do ROS WEB tools. Essa integração, torna possível a utilização da interface para planejamento de movimentos antes de serem enviados ao robô físico. Outra contribuição que pode ser feita a este trabalho, é a integração destes sistemas com outros robôs, explorando ainda mais o ROS e suas propriedades.

REFERÊNCIAS

[1] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10, 2017.
[2] Rachmad Andri Atmoko and Daoguo Yang. Online monitoring & controlling industrial arm robot using mqtt protocol. In *2018 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics)*, pages 12–16. IEEE, 2018.
[3] Erik Hofmann and Marco Rüsch. Industry 4.0 and the current status as well as future prospects on logistics. *Computers in Industry*, 89:23–34, 2017.
[4] Rainer Drath and Alexander Horch. Industrie 4.0: Hit or hype?[industry forum]. *IEEE industrial electronics magazine*, 8(2):56–58, 2014.
[5] Diego RC Silva, Guilherme MB Oliveira, Ivanovitch Silva, Paolo Ferrari, and Emiliano Sisinni. Latency evaluation for mqtt and websocket protocols: an industry 4.0 perspective. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 01233–01238. IEEE, 2018.
[6] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE international systems engineering symposium (ISSE)*, pages 1–7. IEEE, 2017.
[7] Daniel Bezerra, Rafael Roque Aschoff, Geza Szabo, and Djamel Fawzi Hadj Sadok. An iot protocol evaluation in a smart factory environment. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pages 118–123. IEEE, 2018.
[8] Dae-Hyeok Mun, Minh Le Dinh, and Young-Woo Kwon. An assessment of internet of things protocols for resource-constrained applications. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 555–560. IEEE, 2016.
[9] Burak H Çorak, Feyza Y Okay, Metehan Güzel, Şahin Murt, and Suat Ozdemir. Comparative analysis of iot communication protocols. In *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2018.
[10] Ivan Heđi, I Špeh, and A Šarabok. Iot network protocols comparison for the purpose of iot constrained networks. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 501–505. IEEE, 2017.
[11] Stefan Profanter, Ayhun Tekat, Kirill Dorofeev, Markus Rickert, and Alois Knoll. Opc ua versus ros, dds, and mqtt: Performance evaluation of industry 4.0 protocols. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, 2019.
[12] Carol Martinez, Nicolas Barrero, Wilson Hernandez, Cesar Montaño, and Iván Mondragón. Setup of the yaskawa sda10f robot for industrial applications, using ros-industrial. In *Advances in Automation and Robotics Research in Latin America*, pages 186–203. Springer, 2017.
[13] Valerie Lampkin. What is mqtt and how does it work with websphere mq? *DeveloperWorks App. Integration Middleware Support Blog*, 2012.
[14] Markel Iglesias-Urkia, Adrián Orive, Marc Barcelo, Adrian Moran, Josu Bilbao, and Aitor Urbieta. Towards a lightweight protocol for industry 4.0: An implementation based benchmark. In *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pages 1–6. IEEE, 2017.
[15] Andrew Banks and Rahul Gupta. Mqtt version 3.1. 1. *OASIS standard*, 29:89, 2014.
[16] Yiming Xu, V Mahendran, and Sridhar Radhakrishnan. Towards sdn-based fog computing: Mqtt broker virtualization for effective and reliable delivery. In *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–6. IEEE, 2016.

[17] Krešimir Grgić, Ivan Špeh, and Ivan Heđi. A web-based iot solution for monitoring data using mqtt protocol. In *2016 International Conference on Smart Systems and Technologies (SST)*, pages 249–253. IEEE, 2016.

[18] Amanda Dattalo. Ros/introduction, 2019. URL http://wiki.ros.org/ROS/Introduction.

[19] Benjamin B Rhoades, Disha Srivastava, and James M Conrad. Design and development of a ros enabled can based all-terrain vehicle platform. In *SoutheastCon 2018*, pages 1–6. IEEE, 2018.

[20] Sujoy Ghosh, Subarna Ghosh, and Meet Dayani. Design of an unlimited range web browser controlled robot with self-adapting fuzzy logic controller. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1294–1300. IEEE, 2018.

[21] Mohamad Khairi Ishak and Ng Mun Kit. Design and implementation of robot assisted surgery based on internet of things (iot). In *2017 International Conference on Advanced Computing and Applications (ACOMP)*, pages 65–70. IEEE, 2017.