DOI: 10.48011/asba.v2i1.1143

Controle remoto de ferramentas de aperto usando o *Open Protocol*

Andre N. Makoski * **Egberto N. Silva** ** **Amauri A. Assef** ***

* *CPGEI, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, Brasil, (e-mail: andre.makoski@gmail.com)*
** *Faculdade de Engenharia Elétrica, Centro Universitário da FEI, São Paulo, Brasil (e-mail: egbmail@gmail.com)*
*** *CPGEI/DAELT, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, Brasil, (e-mail: amauriassef@utfpr.edu.br)*

Abstract: The increase in demand for exclusive products with high quality and low cost has been doing companies to innovate on their production processes. The automotive industry is no different. In the industrial sector there are several types of manufacturing, assembly and tightening tools. Controlling and tracking these parameters ensures an efficient production, suitable for the quality and safety programs required by law and general market. This journal aims especially to summarize, based on the manufacturer's guide, and test through a Python designed program a way to remotely control assembly tools that use open protocol communication developed by Atlas Copco. The proposed model fits in the concept of 4.0 industry, also known as smart facility.

Resumo: O aumento da demanda por produtos exclusivos com alta qualidade e baixo custo tem feito as empresas inovarem nos processos produtivos. Na indústria automotiva não é diferente. No setor industrial há diversos tipos de ferramentas de fabricação, montagem e aperto. Controlar e rastrear esses parâmetros garante uma produção eficiente, adequada aos programas de qualidade e segurança exigidos pela lei e pelo mercado consumidor. Este artigo visa especificamente resumir, baseado no manual do fabricante, e testar, através de um conjunto de códigos desenvolvidos em linguagem de programação Python, uma maneira a controlar remotamente ferramentas de aperto que utilizam o protocolo de comunicação *Open Protocol*, desenvolvido pela empresa Atlas Copco. Para os testes de avaliação dos comandos de torque e ângulo foram utilizados dois controladores de marcar diferentes e os resultados entre as interfaces desenvolvidas de usuário e Web mostraram perfeita concordância. O modelo proposto poderá ser explorado por estudantes e profissionais para otimizar recursos e desempenho na área de manufatura no âmbito da indústria 4.0, também conhecida como fábrica inteligente.

Keywords: Open Protocol; Tightening Controller; Traceability; Torque; Tightening.
Palavras-chaves: Open Protocol; Controladores de Aperto; Rastreabilidade; Torque; Aperto.

1. INTRODUÇÃO

A inovação tecnológica traz mudanças significativas nos processos de fabricação. Além de possibilitar maior produtividade, flexibilidade, eficiência e competitividade, as ações de inovação colaboram fortemente para o aumento da qualidade dos produtos (Aldea et al., 2018). Atualmente, a qualidade pode ser compreendida como o oferecimento de valor ao cliente e é essencial para a sobrevivência e crescimento de qualquer empresa (Barrera et al., 2019).

Recentemente, tecnologias emergentes, como a Internet das Coisas (IoT), computação em nuvem (*Cloud Computing*), *Big data*, redes de sensores sem fio (*Wireless Sensor Networks*) e sistemas embarcados (*Embedded Systems*) têm sido introduzidas no ambiente da manufatura tornando as fábricas inteligentes (Shiyong et al., 2016).

Em um mercado competitivo e globalizado, a indústria 4.0 é amplamente utilizada em todo o mundo, principalmente para garantir maior desempenho com melhor qualidade de produto e maior competitividade, estendendo para a para as fábricas a maneira como vivemos o nosso cotidiano, conectados. (Mumtaz et al., 2017).

Na manufatura automobilística, inúmeras operações de apertos e rosqueamento, relacionadas aos parâmetros específicos de força, torque e ângulo, são realizados por veículo. Falhas nos processos operacionais de aperto (e.g., aperto insuficiente ou excessivo de elementos de fixação e componentes), introduzidas pelos profissionais envolvidos, podem comprometer a qualidade dos produtos e causar falhas graves. Efetuá-las com eficiência, garantir os padrões de aperto e manter a rastreabilidade dos mesmos, podem reduzir os custos de fabricação, retrabalho de peças e inclusive evitar ou restringir eventuais *recalls* (John and Xiaowen, 2018).

Uma ferramenta auxiliar nesse controle são as apertadeiras eletrônicas e seus controladores. Os controladores de

aperto inicialmente foram desenvolvidos para enviar potência elétrica para as ferramentas e, ao mesmo tempo, garantir que os limites de torque e ângulo desejados ou estratégias complexas sejam alcançadas. Embora a importância da potência elétrica tenha diminuído com a chegada das ferramentas sem fio, com bateria, a garantia da qualidade do processo decorrente da aplicação da estratégia de aperto correta ainda é a principal função dos controladores.

Diversas empresas têm adotado ferramentas e metodologias nas linhas de montagem com foco na redução de custos e ganho de desempenho. Tipicamente, são empregados protocolos específicos para troca de dados e comandos com as ferramentas de montagem, como, por exemplo, o *Open Protocol*, criado pela empresa Atlas Copco para comunicação com os controladores de aperto. O *Open Protocol* foi uma resposta à demanda da indústria para integração de sistemas de gestão de manufatura com os controladores de aperto no "chão de fábrica"(Boyes et al., 2018).

Sistemas integrados e inteligentes vêm se firmando nas indústrias automobilisticas da região de Curitiba, onde existem diversas montadoras de veículos, porém as empresas especializadas no *Open Protocol* são poucas, elevando assim o custo das soluções. Ademais, são escassas na literatura informações sobre o assunto.

Dessa maneira, o objetivo desse artigo é apresentar e avaliar as principais funções do *Open Protocol* para controle de ferramentas de aperto através de um conjunto de *scritps* desenvolvidos em linguagem *Python*. Os resultados desse trabalho poderão ser utilizados como base referencial tanto pela comunidade científica quanto por outros profissionais da indústria que desejam explorar seus recursos.

O artigo está organizado como segue: após a introdução com a contextualização do trabalho, na segunda seção apresenta-se um resumo do *Open Protocol* e suas variações. Na terceira seção, faz-se uma abordagem em relação à indústria 4.0. Em seguida, na quarta seção, apresentam-se os resultados utilizando os *scripts* desenvolvidos. Na quinta seção, discutem-se os resultados e, por fim, na última seção descreve-se a conclusão do trabalho.

2. OPEN PROTOCOL

O *Open Protocol* é uma interface para a criação de aplicativos voltada ao controle remoto de controladores de aperto (Atlas-Copco, 2015). É independente de plataforma e suporta conexão serial RS 232/RS-485 e Ethernet *full duplex*. Para iniciar a comunicação, o aplicativo deve conectar-se com o controlador, sendo esse o servidor e aquele o cliente. De acordo com o manual do fornecedor, a aplicação servidor é chamada de controlador e a aplicação cliente de integrador. Uma vez conectado, o integrador comanda o controlador selecionando o aperto a ser realizado, podendo ser via as estratégias *Parameter Set* (Pset) ou receita (Job).

2.1 PSETs e JOBs

Cada estratégia individual de aperto existente no controlador é chamada de Pset. Os Psets são definidos pelo usuário no momento da programação do controlador, correspondendo às estratégias e/ou parâmetros de aperto que a ferramenta irá aplicar durante a operação. Para cada Pset configurado, será atribuído um código único, que deverá ser utilizado para selecionar estratégias e executar apertos.

No entanto, os processos de aperto na indústria normalmente aplicam mais de um parafuso por peça ou posto de montagem. Assim, as estratégias de aperto requeridas para cada parafuso podem ser iguais, diferentes ou mistas. Ou seja, pode-se repetir um mesmo Pset em um grupo de parafusos ao passo que se utilizam outros Psets individualmente nos parafusos restantes. Para agrupar todas estas operações de seleção de Psets, foi criado o conceito de Job, que nada mais é que uma receita de produção completa na qual se configura quais Psets serão executados, a ordem de execução e quantidade de repetições. Assim, como nos Psets, cada Job configurado terá um identificador único que poderá ser utilizado posteriormente para seleção.

Neste contexto, o uso do *Open Protocol* está relacionado principalmente com o envio de comandos para seleção de Psets e Jobs e à leitura de dados de aperto. Adicionalmente, a seleção de Pset ou Job esta diferenciada na estrutura das mensagens trocadas.

2.2 Estrutura de Mensagens

A maioria das mensagens trocadas pelo integrador e controlador são em formato ASCII e consistem em três partes: *Header*, *Data Field* e *Message End*.

O campo *Header*, apresentado na Fig. 1, é composto pelos 20 primeiros *bytes* e não há obrigatoriedade de todos os campos estarem preenchidos.

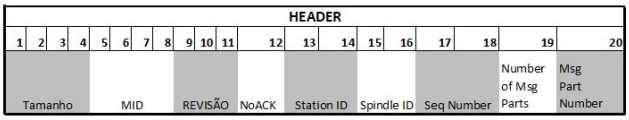

Figura 1. Campo Header composto por 20 *bytes*.

Os quatro primeiros *bytes* especificam o tamanho da mensagem somando os campos *Header* + *Data Field*. O bloco *Message End* não entra na soma.

O *Message identification* (MID) é a informação mais importante de todo o dado, pois indica como interpretar a mensagem, ou seja, o que se deseja fazer: selecionar um aperto, bloquear uma ferramenta ou ler dados de aperto entre outros.

A revisão da MID é usada em casos em que existem diferentes versões para a mesma mensagem, sendo que a revisão padrão é a 001. As diferentes revisões surgiram conforme os controladores foram se modernizando. A primeira revisão (001) aceitava 99 Jobs diferentes e a segunda passou para 9999. A mensagem que mais recebeu revisões foi a MID 0061, que indica o último resultado de aperto, pois novos dados foram surgindo e precisavam ser extraídos do controlador. Entre essas informações estão o torque de arrasto, torque de encosto e suporte a múltiplos estágios de aperto.

O *NoACK* é um sinal a ser usado apenas em mensagens de subscrição de eventos por parte do integrador. Este *flag* especifica quando o controlador irá aguardar ou não por

mensagens de reconhecimento, após disparar um evento para o integrador. O modo padrão é deixar esse campo com espaço em branco ou "0", indicando que o controlador deverá sim aguardar por mensagem de reconhecimento para cada evento disparado.

O *StationID* é utilizado para representar o identificador, ID, da estação em controladores que implementam mais de um ID. Na prática, esse campo não mais é usado, pois os novos controladores que implementam múltiplas estações virtuais, abrem portas TCP adicionais para cada estação virtual em execução, como exemplo: 4546, 4547 e assim sucessivamente.

Os *bytes Seq Number* são utilizados na versão 2.0 do *Open Protocol* e estão relacionados com o reconhecimento na camada de *link*.

As informações *Number of Msg Parts* e *Msg Part Number* são utilizados a partir da versão 2.0 do *Open Protocol* e estão associados com o fracionamento de mensagens maiores que 9999 *bytes*.

Na sequência do campo *Header*, o *Data Field* é o campo de dados que carrega as informações de interesse para as aplicações. Como exemplo, podem-se enviar atributos de comandos ou valores provenientes das operações de aperto, os quais contêm tamanhos dinâmico e, inclusive, podem ser nulas em algumas MIDs.

O campo *Message end* é o indicador de fim de mensagem, em ASCII, representado por NULL.

2.3 Tipos de mensagens

No *Open Protocol* existem cinco tipos básicos de mensagens: *Request*, *Command*, *Subscription*, *Acknowledge* e *Special*, sendo os duas últimas tratadas como exceções.

As mensagens do tipo *Request* são utilizadas pelo integrador para recuperar informações de interesse presentes no controlador, como, por exemplo, a lista de programas e estratégias de aperto configurados. Para recuperar tal informação o integrador precisa enviar a MID 0010 e aguardar resposta do controlador, que poderá ser a MID 0011 trazendo a lista de programas, ou a MID 0004 indicando que algum erro ocorreu. Tanto a mensagem de requisição partindo do integrador, quanto a resposta enviada pelo controlador pertencem a mesma classe.

Uma mensagem do tipo *Command* indica uma ação mais imperativa, como, a seleção de uma estratégia de aperto, liberação ou bloqueio da ferramenta dentre dezenas de outras possibilidades. As mensagens de comando obrigatoriamente retornarão uma mensagem de confirmação de comando realizado com sucesso, MID 0005, ou que algum erro ocorreu, MID 0004.

As mensagens de *Subscription* são aquelas que permitem aos integradores se cadastrem em diversos serviços de notificação automática disponíveis no controlador. Dessa forma, sempre que um evento de interesse do integrador ocorrer, esse será notificado prontamente pelo controlador. Como principal exemplo, existem as notificações de resultado de aperto, nas quais o controlador envia a MID 0061 sempre que uma operação de aperto é realizada pela ferramenta. Neste grupo constam as mensagens de solicitação de cadastro (*subscribe*), as mensagens de cancelamento de cadastro (*unsubscribe*), as mensagens do evento propriamente e as mensagens de reconhecimento de evento recebido.

A maioria das requisições ou comandos enviados durante a comunicação irão desencadear algum tipo de reconhecimento, ou *Acknowledge*, seja negativo (MID 0004) ou positivo (MID 0005). No caso de reconhecimento positivo, a MID 0005 trará em seu campo de dados o código MID do comando que foi aceito. No caso de reconhecimento de falha, a MID 0004 trará em seu campo de dados o código MID da requisição ou comando que desencadeou a falha, e também um código de dois caracteres informando a natureza do erro.

No caso de mensagens do tipo *Special* é possível citar como exemplos as MIDs para iniciar e parar o protocolo (MID 0001, MID 0002 e MID 0003) e o *KeepAlive* (MID 9999).

2.4 Inicialização da comunicação

Apesar de suportar comunicação serial, atualmente a maioria dos sistemas implementam o *Open Protocol* usando o modelo TCP-IP. Dessa maneira de comunicação, o controlador mantém uma porta aberta para receber conexões de um ou mais aplicativos integradores. A porta 4545 é a padrão, porém a muitos equipamentos permitem que esta seja definida pelo usuário.

Após o estabelecimento da comunicação entre cliente e servidor na camada TCP-IP, o protocolo deve ser iniciado através das MIDs especiais, ou seja, para iniciar a troca de informações e comando remoto a primeira mensagem que o integrador deve enviar é a MID 0001, o controlador responderá com a MID 0002, caso aceite a conexão, ou a MID 0004, caso não aceite. Uma vez iniciado o protocolo, o integrador deverá garantir que não ocorra silêncio na comunicação superior a 15 segundos, sob pena de ter a conexão cancelada pelo controlador. Para evitar tal situação o aplicativo deverá enviar a mensagem *KeepAlive* (MID 9999) a cada 10 segundos.

As Figs. 2 e 3 mostram o fluxo de mensagens e o alocamento de *bytes* para a MID 0001 entre o aplicativo e o controlador.

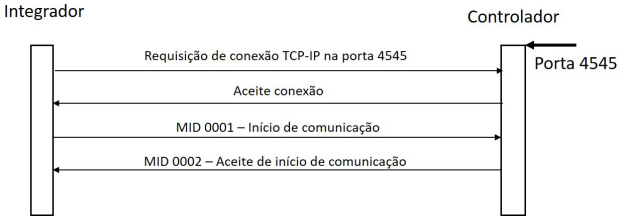

Figura 2. Fluxo início das mensagens.

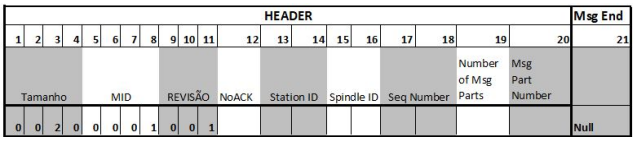

Figura 3. Exemplo de mensagem MID 0001 do *Header*.

Ao receber a MID 0001, o controlador responde com a MID 0002 (Fig. 4), caso aceite a conexão. A Fig. 5 mostra um exemplo do campo *Data Field* com caracteres ASCII. Caso a conexão não seja aceita, a MID00 04 é enviada, ilustrada da Fig. 6.

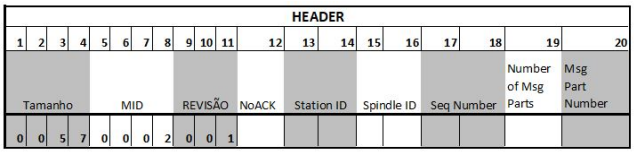

Figura 4. Exemplo de mensagem MID 0002 do *Header* para aceite de comunicação.

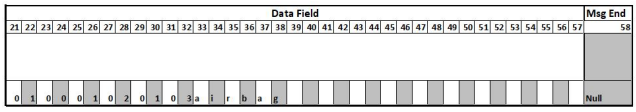

Figura 5. Campo *Data Field* da MID 0002.

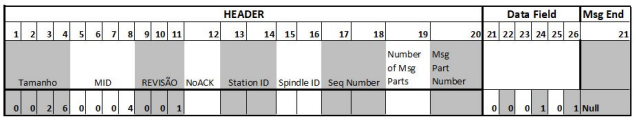

Figura 6. Mensagem de recusa de conexão - MID 0004.

É possível verificar no campo *Data Field*, que o erro está relacionado com a MID 0001 e que o código 01, de acordo com o manual, significa *Invalid data.*

2.5 Fluxos básicos de operação com o Open Protocol

Conforme descrito anteriormente, as unidades básicas que representam estratégias individuais de trabalho ou receitas completas são respectivamente os Psets e Jobs. No exemplo da Fig. 7, existe a necessidade de se apertarem quatro parafusos – dois deles com 50 Nm e os outros dois com 100 Nm. Nos parâmetros de configuração do aperto, definem-se os limites aceitáveis para a ação, como as tolerâncias de torque e ângulo, entre outras.

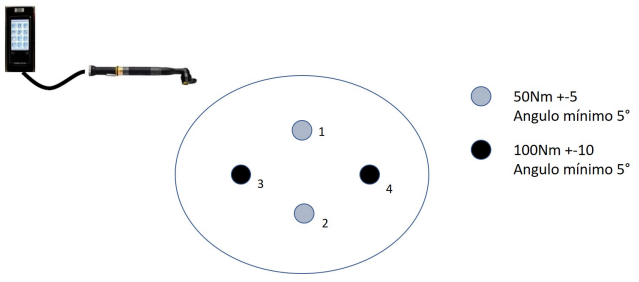

Figura 7. Exemplo de configuração de aperto.

Para realizar um aperto através do Pset, pode-se configurar o Pset 1 com parâmetros de torque alvo de 50 Nm, tolerância de ± 10% e ângulo mínimo de 5°. Também seria preciso criar um o Pset 2 com torque alvo de 100 Nm, tolerância de ± 10% e ângulo mínimo de 5°. Para deixar o controlador apto a operar, também via seleção de Job, deve-se criar o Job 1 cuja receita será repetir duas vezes o Pset 1 e, em seguida, repetir duas vezes o Pset 2.

A maneira de se realizar o aperto por Job ou Pset difere-se pelo envio da MID0018 e MID 0038, respectivamente. Além disso, é importante enviar a mensagem *Abort Job*, MID 0127, para cancelar um eventual Job na mémoria do controlador. As Figs 8 e 9 apresentam os fluxos de mensagens para as estratégias Jop e Pset, respectivamente.

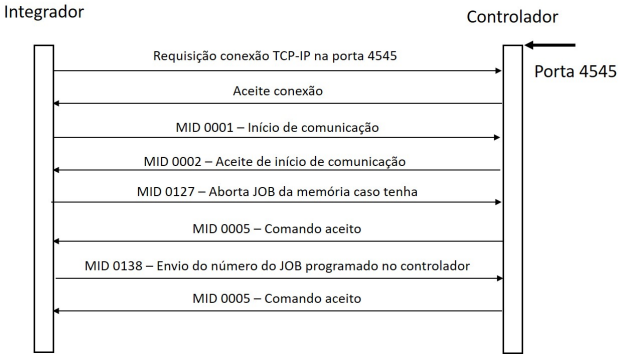

Figura 8. Seleção de aperto através de Job.

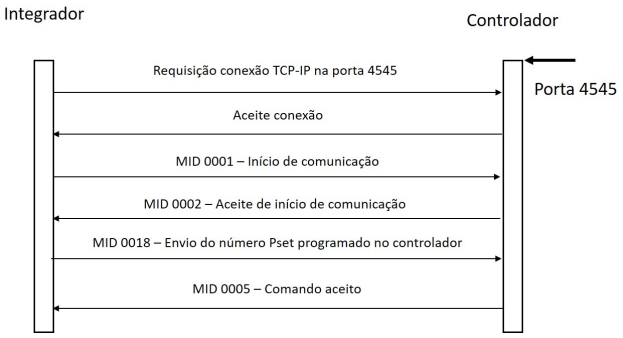

Figura 9. Seleção de aperto através de Pset.

2.6 Leitura de dados de aperto

Após a realização do aperto pela ferramenta, o controlador enviará todos os dados relacionados ao aperto (Fig. 10) através da MID 0061, desde que o integrador tenha se cadastrado previamente nesse serviço de notificação (MID 0060). Após receber a MID 0061, o integrador enviará uma resposta de confirmação positiva para o controlador (MID 0062). Caso contrário, o controlador tentará retransmitir a mensagem três vezes e, por fim, encerrará a conexão.

É possível desativar o envio da mensagem de confirmação por parte do integrador configurando para "1" o *bit NO-ACK* no cabeçalho da mensagem MID 0060.

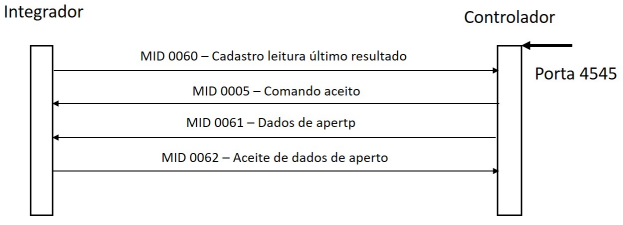

Figura 10. Leitura de dados de aperto.

Conforme a Fig. 11, a MID 0003 precisa ser enviada para encerramento da conexão com o controlador. Após isso, o controlador para de responder a qualquer comando, exceto a MID 0001 de início da comunicação.

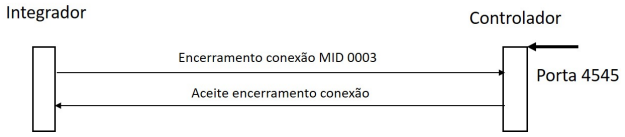

Figura 11. Encerramento da conexão.

2.7 Demais informações do Open Protocol

A última versão do manual, *Release 2.5 Revision 5*, em suas 301 páginas traz inúmeras informações e mensagens possíveis de serem utilizadas em diversas aplicações. Entre elas as mensagens de requisições para resgatar a lista previamente configurada na memória (MID 0010 e MID 0011) para Pset e (MID 0030 e MID 0031) para Job.

Além dessas, ressalta-se as mensagens de habilitar e desabilitar ferramentas, MID 0043 e MID 0042, respectivamente, as quais são úteis no intertravamento entre processos; e as MIDs relacionadas às informações de identificação do produto (VIN) que podem ser vinculadas à rastreabilidade e consequentemente à qualidade.

3. OPEN PROTOCOL EM RELAÇÃO À INDÚSTRIA 4.0

A literatura referente a indústria 4.0 aborda diversas tecnologias, também conhecidas como pilares, que caracterizam um processo fábril como pertencente a 4ª revolução industrial. (Vaidya et al., 2018). É possível caracterizar as ferramentas de aperto e controladores modernos com pelo menos três destes pilares: IoT, *Big Data* e *System Integration*.

Do ponto de vista de IoT, pode-se considerar que uma ferramenta de aperto enviando e recebendo dados para outros equipamentos, desde que esses alcancem a internet ou intranet, já satisfaz os requisitos mínimos para ser considerada um dispositivo IoT (Hanes et al., 2017). Contudo, a utilização de *gateways* IoT que convertam o *Open Protocol* para protocolos de mensagens mais modernos como MQTT (*Message Queue Telemetry Transport*), é uma alternativa para integrar as ferramentas de aperto na iniciativa de IoT da fábrica (Hillar, 2017).

Em relação à integração plena de sistemas e processos, o *Open Protocol* permite que as ferramentas sejam controladas e monitoradas remotamente pelos *softwares* integradores através dos comandos básicos apresentados neste artigo. Contudo, o protocolo possui também MIDs avançadas que permitem a construção de Jobs e Psets dinamicamente. Este recurso poderia, por exemplo, ser usado em conjunto com algoritmos de inteligência artificial que ajustariam atributos da estratégia de aperto com base nos resultados recebidos das operações anteriores (Li et al., 2017). Ou seja, o *Open Protocol* é compatível com iniciativas de sistemas *cyber*-fisicos. Além disso, o *Open Protocol* possui centenas de mensagens e eventos que lhe garantem uma riqueza muito grande de dados. Por se tratar de um protocolo aberto e por utilizar o formato ASCII para construção das mensagens e representação dos resultados, a migração destes dados para um *data lake* é possível, tornando o *Open Protocol* uma tecnologia compatível com iniciativas de *Big Data* (Miloslavskaya and Tolstoy, 2016).

No entanto, em relação à segurança cibernética, o *Open Protocol* não traz recursos que o caracterizem como um protocolo seguro. Pelo contrário, o fato de o controlador aceitar qualquer conexão externa e por não existir nenhum mecanismo de autenticação dos aplicativos integradores, o protocolo e, consequentemente, os controladores e até mesmo o processo, se mostram vulneráveis sob o ponto de vista da segurança cibernética (Craigen et al., 2014). Uma alternativa para esta situação é a utilização de *firewalls* de bloqueio no chão de fábrica a fim de evitar *cyber* ataques (Tsuchiya et al., 2018).

4. RESULTADOS

Os *scripts* a seguir foram desenvolvidos de modo a ilustrar a comunicação básica entre aplicativos integradores e os controladores de aperto. A linguagem *Python* foi escolhida neste trabalho por ser uma linguagem de código aberto, de alto nível e muito utilizada. Além disso, possibilita que mesmo pessoas sem conhecimento em programação possam entender ou até mesmo testar os *scripts* (Kumar and Panda, 2019). Existem diversas interfaces de desenvolvimento e editores de código para linguagem *Python*. Entretanto, neste trabalho foi adotado o MU, que é um editor de código aberto de fácil utilização, para o desenvolvimento dos três *scripts* de comunicação e comando.

Para realização dos testes foram configurados dois controladores de marcas diferentes. Um deles é o PowerFocus 6000 da própria empresa Atlas Copco e o outro um MPRO400GC da empresa Cleco. Tanto os controladores quanto as ferramentas de aperto devem estar configurados com os dados de aperto e o *Open Protocol* ativado.

O *script* OpenProtocolJOBSelector.py (Lista 1) deve ser o primeiro a ser executado, selecionando assim o Job desejado.

Listing 1. Arquivo OpenProtocolJOBSelector.py.

```
#OpenProtocolJOBSelector.py
import socket
#Cria e conecta um socket na porta padrao
with socket.socket(socket.AF_INET, socket.
    SOCK_STREAM) as sock:
  sock.connect(('192.168.4.10', 4545))
  #Configurar IP da ferramenta
  #Envia MID0001 para iniciar protocolo
  sendMID = '00200001001000000000' + chr(0)
  sock.sendall(sendMID.encode())
  #Aguarda resposta do controlador e imprime
  data = sock.recv(1024).decode()
  receivedMID = data[4:8]
  print('Received MID ', receivedMID)
  #Confirma se o MID0001 foi aceito
  #ou seja, se recebeu MID0002
   if receivedMID == '0002':
   #Solicita entrada numero do JOB desejado
   # e envia para o controlador
     pset = input('Qual JOB deseja selecionar ?')
```

```
    #Primeiramente aborta um eventual JOB (se
        existir)
    sendMID = '00200127001000000000' + chr(0)
    print(sendMID)
    sock.sendall(sendMID.encode())
    #Aguarda resposta do controlador e imprime
    data = sock.recv(1024).decode()
    receivedMID = data[4:8]
    #Confirma se abortar JOB foi aceito
    if receivedMID == '0005':
      print('Comando abortar JOB aceito')
      #Envia comando para selecionar JOB
      sendMID = '00220038001000000000' + pset.
          zfill(2) + chr(0)
      print(sendMID)
      sock.sendall(sendMID.encode())
      #Aguarda resposta do controlador e imprime
      data = sock.recv(1024).decode()
      receivedMID = data[4:8]
      print('Received MID ', receivedMID)
      #Imprime mensagem de confirmacao
      # (negativa ou positiva)
       if receivedMID == '0005':
         print('Selecao JOB aceita')
       else:
         print('Erro ao selecionar JOB')
        else:
         print('Erro ao Abortar JOB')
       #Encerra Open Protocol
       sendMID = '00200030001000000000' + chr(0)
       print(sendMID)
       sock.sendall(sendMID.encode())
```

Na Fig. 12 são mostradas as mensagens resultantes impressas na tela do usuário. A primeira linha traz a MID 0002 que representa o aceite de comunicação em resposta à MID 0001. A segunda linha indica a pergunta do Job desejado ao usuário. Nesse caso, o Job 2 foi a opção desejada. A terceira linha mostra a mensagem enviada ao controlador pelo programa, "00220038001000002", sendo: 0022 a quantidade de *bytes*, 0038 a MID de seleção de Job, 001 a revisão da MID e 02 o Job escolhido. A quarta e quinta linha mostram a resposta de aceite MID 0005 e a informação de seleção aceita, respectivamente.

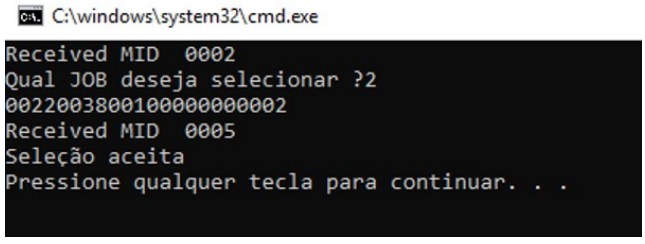

Figura 12. Exemplo de tela do usuário para a estratégia Job.

Após selecionado o Job, é preciso o cadastro para receber os dados de aperto. Assim, executa-se o *script* OpenProtocolLastResult.py (Lista 2). Da mesma maneira, envia-se a MID 0001, aguarda-se a resposta MID 0002 e faz-se a inscrição para receber os apertos através da MID 0060. Caso aceita a inscrição, o controlador envia a MID 0005 e, então, o integrador aguarda os dados dos apertos. Toda vez que um aperto é realizado o controlador envia a MID 0061 com os respectivos dados. É necessário o envio da mensagem *KeepAlive* (MID 9999) para manter a comunicação ativa.

Listing 2. Código do arquivo OpenProtocolLastResult.py.

```
#OpenProtocolLastResult.py
import socket
import time, threading
def KeepAlive():
  #Envia MID9999 KeepAlive
  sendMID = '00209999001000000000' + chr(0)
  sock.sendall(sendMID.encode())
  #Inicia timer para disparar novamente apos 10s
  threading.Timer(10, KeepAlive).start()
#Cria e conecta um socket na porta padrao OP
with socket.socket(socket.AF_INET, socket.
    SOCK_STREAM) as sock:
  sock.connect(('192.168.4.10', 4545))
  #Envia MID0001 para iniciar protocolo
  sendMID = '00200001001000000000' + chr(0)
  sock.sendall(sendMID.encode())
  #Aguarda resposta do controlador e imprime
  data = sock.recv(1024)
  receivedMID = data[4:8].decode()
  print('MID Recebido: ', receivedMID)
  #Confirma se o MID0001 foi aceito
   if receivedMID == '0002':
    #Agenda keepalive para ocorrer apos 10s
    threading.Timer(10, KeepAlive).start()
    #Solicita cadastro no servico de ultimo
        resultado de aperto
    sendMID = '00200060001000000000' + chr(0)
    sock.sendall(sendMID.encode())
    #Aguarda resposta do controlador e imprime
     data = sock.recv(1024).decode()
     receivedMID = data[4:8]
     print('Received MID ', receivedMID)
     #Confirma se controlador aceitou cadastro
     if receivedMID == '0005':
       print('Cadastro aceito')
       while True:
        #Aguarda MID de resultado de aperto
        data = sock.recv(1024).decode()
        receivedMID = data[4:8]
        print('Received MID ', receivedMID)
#Confirma se recebeu resultado de aperto MID 0061
          if receivedMID == '0061':
           #Envia Acknowledge para o controlador
           sendMID = '00200062001000000000' + chr
               (0)
           sock.sendall(sendMID.encode())
           #Extrai informacoes desejadas e imprime
           torque = str(float(int(data[140:146])
               /100))
           angulo = str(data[169:174])
           status = str(data[107])
           print('Torque: ', torque)
           print('Angulo: ', angulo)
           print('Status:', 'OK' if status == '1'
               else 'NOK')

    else:
    print('Erro. Cadastro falhou')
```

Nesse exemplo, foram extraídos os dados de torque, ângulo e o *status*, que indica se o aperto está aprovado ou reprovado, de acordo com os parâmetros configurados no controlador. Na Fig. 13 é possível verificar que os dados

coletados são exatamente os mesmos, tanto na interface do usuário quanto na interface Web do controlador.

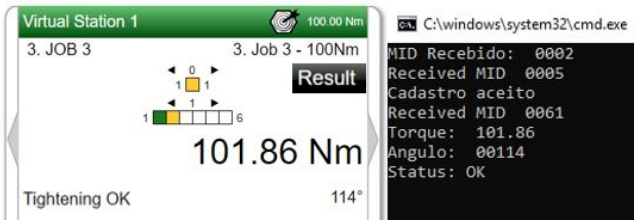

Figura 13. Dados recebidos ao realizar o aperto desejado via Job.

Para a seleção do Pset, o controlador deve ser programado e a MID 0018 precisa ser enviada ao invés da MID 0038, conforme ilustrado na Lista 3, referente ao *script* OpenProtocolPsetSelector.Py.

Listing 3. Arquivo OpenProtocolPsetSelector.py.

```
#Selecao Pset
import socket
#Cria e conecta um socket na porta padrao OP
with socket.socket(socket.AF_INET, socket.
    SOCK_STREAM) as sock:
  sock.connect(('127.0.0.1', 4545))
  #Envia MID0001 para iniciar protocolo
  sendMID = '00200001001000000000' + chr(0)
  sock.sendall(sendMID.encode())
  #Aguarda resposta do controlador e imprime
  data = sock.recv(1024).decode()
  receivedMID = data[4:8]
  print('Received MID ', receivedMID)
  #Confirma se recebeu MID0002
  if receivedMID == '0002':
   #Solicita entrada numero do PSet desejado e
       envia para o controlador
   pset = input('Qual PSet deseja selecionar ?')
   sendMID = '00230018001000000000' + pset.zfill
       (3) + chr(0)
   print(sendMID)
   sock.sendall(sendMID.encode())
   #Aguarda resposta do controlador e imprime
   data = sock.recv(1024).decode()
   receivedMID = data[4:8]
   print('Received MID ', receivedMID)
   #Imprime mensagem de confirmacao
    if receivedMID == '0005':
     print('Selecoo PSet aceita')
    else:
     print('Erro ao selecionar PSet')
    #Encerra Open Protocol
    sendMID = '00200003001000000000' + chr(0)
    print(sendMID)
    sock.sendall(sendMID.encode())
```

Similarmente, cadastra-se para receber os dados de aperto através da MID 0060 utilizando o *script* OpenProtocol-LastResult.py. Também, verifica-se a igualdade nos dados recebidos entre a interface do usuário e a interface Web do controlador através da Fig. 14.

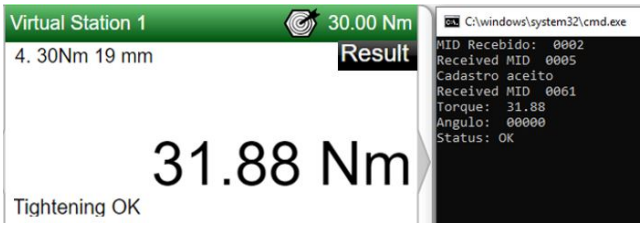

Figura 14. Dados recebidos ao realizar o aperto desejado via Pset.

5. DISCUSSÃO

Para interação com ferramentas eletrônicas de aperto usando o *Open Protocol* é necessário conhecer e aplicar algumas das principais mensagens de comando e configuração. Assim, esse trabalho além de trazer informações básicas de como realizar a comunicação entre aplicativo (servidor) e controlador (cliente), também mostrou como fazê-lo através de exemplos desenvolvidos em linguagem de programação *Python*. Acredita-se, então, que os resultados apresentados à comunidade acadêmica e profissionais da área poderão ajudar na integração de sistemas de gestão de manufatura, encurtando o tempo de desenvolvimento de aplicações de aperto e rosqueamento, uma vez que as informações compartilhadas são escassas na literatura.

Como já previsto, apesar de funcionais, os *softwares* apresentados não desempenham todas as funcionalidades e requisitos mínimos de robustez que um aplicativo integrador comercial necessita, sobretudo por implementar comunicação síncrona, na qual o fluxo do programa fica interrompido até que uma mensagem seja recebida pelo *socket*. Para aplicativos comerciais usando *Python*, bibliotecas de estrutura de rede assíncrona (Schreiber et al., 2017) e fluxo de *multithreading* (Nemirovsky and Tullsen, 2013) seriam mais apropriadas.

Uma alternativa ao método explanado nesse artigo é a utilização do padrão OPC (*Open Platform Communications*) e o *driver Ehternet Torque Tool* (PTC-Inc, 2019). O OPC é um padrão para comunicação de dados no mundo industrial permitindo que equipamentos como controladores lógicos programáveis e sistemas de controle distribuídos, entre outros, conectem-se com *softwares* de operação e gestão da produção (Zhu et al., 2017). Dessa maneira, o OPC funcionaria com um intermediário entre o integrador e o controlador, sendo os dados tratados como *tags* e não mais como *bytes*. O artigo (Merchán et al., 2017) demonstra uma aplicação utilizando o OPC para comunicação com dispositivos de controle. Apesar de facilitar a comunicação, o *driver* é comercializado e tem preços elevados, o que dependendo da quantidade de controladores pode ser inviável financeiramente.

Em relação ao assunto financeiro é saudável discutir a possibilidade de futuramente não existir a necessidade do controlador, podendo o integrador conversar diretamente com a ferramenta, desde que essa comunique-se via *Open Protocol* o que na prática reduziria consideravelmente o custo da solução.

Por fim, não foram observadas divergências de resultado em relação aos 2 controladores. Ambos tiveram as mesmas respostas e desempenho.

6. CONCLUSÃO

O estudo apresentado teve o objetivo de resumir as principais informações para realizar o controle remoto de ferramentas de aperto utilizando o *Open Protrocol*. Foram trazidas informações básicas para comunicação, as principais MIDs e, por fim, alguns *scripts* para envio e recebimento de dados, os quais foram validados utilizando dois controladores de diferentes fabricantes. Cobre-se assim um *gap* na literatura e criam-se novas oportunidades para estudantes, profissionais e até empresas, que hoje não possuem em seu portfólio a comunicação com ferramentas de aperto, passem a oferecer esse recurso, aumentando a oferta e reduzindo o custo dessas soluções na indústria. Como trabalhos futuros, pode-se aprimorar a questão da rastreabilidade, a utilização de Psets e Jobs dinâmicos descentralizando as informações do controlador ou ainda o desenvolvimento de ferramentas *wireless*, as quais não precisem de controlador como intermediário e passem a se comunicar diretamente usando o *Open Protrocol*.

7. AGRADECIMENTOS

Os autores agradecem ao gerente Edrey Damasio por fornecer o suporte e recursos necessários, além das horas de trabalho para realização dos testes dos programas.

REFERÊNCIAS

Aldea, A., Iacob, M., Wombacher, A., Hiralal, M., and Franck, T. (2018). Enterprise architecture 4.0 – a vision, an approach and software tool support. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, 1–10.

Atlas-Copco (ed.) (2015). *Open Protocol Specification*. Atlas Copco Industrial Technique AQ, Specification Release 2.5. Release 2.5 Revision 5.

Barrera, A.B.C., Parreño, M.F., and J.C.Q.Flores (2019). Waste reduction using lean manufacturing tools: A case in the manufacturing of bricks. In *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*., 1285–1289.

Boyes, H., Hallaq, B., Cunningham, J., and Watson, T. (2018). The industrial internet of things IIoT: An analysis framework. *Computers in Industry*, 101, 1–12.

Craigen, D., Diakun-Thibault, N., and Purse, R. (2014). Defining cybersecurity. *Technology Innovation Management Review*, 4(10).

Hanes, D., Salgueiro, G., Grossetete, P., Barton, R., and Henry, J. (2017). *IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things*. Cisco Press.

Hillar, G.C. (2017). *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd.

John, N. and Xiaowen, H. (2018). Discovery-to-recall in the automotive industry: A problem-solving perspective on investigation of quality failures. *Journal of Supply Chain Management*, 54, 71–95.

Kumar, A. and Panda, S.P. (2019). A survey: How python pitches in it-world. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 248–251.

Li, B.h., Hou, B.c., Yu, W.t., Lu, X.b., and Yang, C.w. (2017). Applications of artificial intelligence in intelligent manufacturing: a review. *Frontiers of Information Technology & Electronic Engineering*, 18(1), 86–96.

Merchán, D.F., Peralta, J.A., Vazquez, A., Minchala, L.I., and D.S.Astudillo (2017). Open source scada system for advanced monitoring of industrial processes. In *2017 International Conference on Information Systems and Computer Science (INCISCOS)*, 160–165.

Miloslavskaya, N. and Tolstoy, A. (2016). Big data, fast data and data lake concepts. *Procedia Computer Science*, 88(300-305), 63.

Mumtaz, S., Alsohaily, A., Pang, Z., Rayes, A., Tsang, K.F., and Rodriguez, J. (2017). Massive internet of things for industrial applications: Addressing wireless iiot connectivity challenges and ecosystem fragmentation. *IEEE Industrial Electronics Magazine*, 11(1), 28–33.

Nemirovsky, M. and Tullsen, D. (2013). *Multithreading Architecture*. Morgan Claypool Publishers, San Rafael, California.

PTC-Inc (ed.) (2019). *Torque Tool Ethernet Driver*. PTC Inc.

Schreiber, A., Scullin, W., Spotz, B., Terrel, A., Basermann, A., Chen, Y., Foley, S., Harrison, C., Hinsen, K., Klemm, M., Kloeckner, A., Ling, M., and Muller, M. (2017). Pyhpc2016: 6th workshop on python for high-performance and scientific computing. *Proceedings of PyHPC 2016: 6th Workshop on Python for High-Performance and Scientific Computing - Held in conjunction with SC16: The International Conference for High Performance Computing, Networking, Storage and Analysis*, v–vi.

Shiyong, W., Jiafu, W., Di, L., and Zhang, Z. (2016). Implementing smart factory of industrie 4.0: An outlook. *International Journal of Distributed Sensor*, 2016, 1–10.

Tsuchiya, A., Fraile, F., Koshijima, I., Ortiz, A., and Poler, R. (2018). Software defined networking firewall for industry 4.0 manufacturing systems. *Journal of Industrial Engineering and Management (JIEM)*, 11(2), 318–333.

Vaidya, S., Ambad, P., and Bhosle, S. (2018). Industry 4.0–a glimpse. *Procedia Manufacturing*, 20, 233–238.

Zhu, M., Lu, S., Du, H., and Zhu, Z. (2017). Design and application of field equipment information acquisition system based on opc. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 7422–7426.