Uma Arquitetura Distribuída para Sistemas de Tempo Real Baseados em VANT´s

A Distributed Architecture for UAV-based Real-Time Systems

DOI:10.34117/bjdv7n3-438

Recebimento dos originais: 17/02/2021
Aceitação para publicação: 17/03/2021

Celso Maciel da Costa
Doutorado em Computação
Universidade Estadual do Rio Grande do Sul (Uergs)
Guaíba– RS – Brazil
E-mail: celso-costa@uergs.edu.br

Philippe Silveira
Graduação em Engenharia de Computação
Universidade Estadual do Rio Grande do Sul (Uergs)
Guaíba– RS – Brazil
E-mail: phil.silveira@hotmail.com

Alleff Dymytry
Graduação em Engenharia de Computação
Universidade Estadual do Rio Grande do Sul (Uergs)
Guaíba– RS – Brazil
E-mail: alleffdymytry@gmail.com

Maike Bressan
Graduando em Engenharia de Computação
Universidade Estadual do Rio Grande do Sul (Uergs)
Guaíba– RS – Brazil
E-mail: maike-bressan@uergs.edu.br

Adriane Parraga
Doutorado em Engenharia Elétrica
Universidade Estadual do Rio Grande do Sul (Uergs)
Guaíba– RS – Brazil
E-mail: adriane-parraga@uergs.edu.br

Letícia Vieira Guimarães
Doutorado em Ciência de Computação e Engenharia de Sistemas
Universidade Estadual do Rio Grande do Sul (Uergs)
Guaíba– RS – Brazil
E-mail:leticia-guimaraes@uergs.edu.br

RESUMO
A evolução ocorrida no desenvolvimento dos VANT´s fez surgir alguns desafios, sendo que um deles está relacionado ao ambiente distribuído de suporte à aplicação. Esse artigo apresenta uma arquitetura distribuída para suporte à construção de sistemas baseados em

VANT´s. A arquitetura proposta considera a existência de uma base em terra, de um servidor, e de um cliente embarcado no VANT Com o uso do conceito de middleware, as funcionalidades do sistema são visíveis aos usuários sob a forma de uma API. Para validar a arquitetura proposta, foi realizada uma implementação, na qual um VANT captura imagens e as envia para uma base em terra, em tempo real. A comunicação entre cliente e servidor é feita com o uso de uma rede wireless. As imagens são capturadas por uma câmera acoplada a um Raspberry embarcado no VANT. O sistema se encontra operacional e o uso do conceito de middleware se mostrou adequado ao desenvolvimento de sistemas baseados em VANT´s.

Palavras-chave: Veículos aéreos não tripulados, arquitetura distribuída, sistemas embarcados de tempo real, captura e envio de imagens em tempo real

ABSTRACT
The evolution of the development of the Drones has given rise to some challenges, one of which is related to the distributed environment of the application support. This article presents a distributed architecture to support the construction of systems Drones based. The proposed architecture considers the existence of a ground base, the server, and a client boarded on the Drone equipped with a camera. The client captures images and sends them to the server in real time, which receives and displays them. Using the concept of middleware, the features are visible to users in the form of an API and allow the users to receive images in the form of photos, stream and record video. To validate the proposed architecture, an implementation was performed in which the communication between client and server is made using a wireless network. The images are captured by a camera attached to a Raspberry boarded in the Drone. The system is operational and allows users to view real-time images captured by the Drone.

Keywords: Unmanned aerial vehicles, distributed architecture, embedded real-time systems, capture and send images in real time

1 INTRODUÇÃO

À medida que a tecnologia avança, os Veículos Aéreos Não Tripulados (VANTs) tendem a alcançar progressivamente velocidades mais elevadas, transportar cargas maiores e cobrir distâncias maiores. Os desafios envolvem diversas áreas, incluindo automação, visão computacional, desenvolvimento de software embarcado e engenharia mecânica [Luca Mottola, 2015]. Diversos trabalhos recentes mostram a importância dos VANTs em diferentes tipos de aplicações, que não seriam possíveis com outra tecnologia.

O uso de VANTs para a coleta de imagens em baixas altitudes se mostra uma opção viável para uso em diferentes aplicações. Os atuais VANTs evoluíram de forma a permitir que os vôos possam ser feitos com base em localização obtida por GPS com mínima intervenção de um piloto, o qual programa a missão, manda o VANT executá-la e monitora sua execução desde a decolagem até o pouso automático. Câmeras de captura de imagens acopladas aos VANTs permitem o registro de imagens no espectro visível e adjacentes. Em

muitas aplicações, o processamento das imagens ocorre posteriormente, e os resultados são obtidos, em geral, algumas horas ou dias depois.

Para uma classe de aplicações, o desenvolvimento de um sistema embarcado em um VANT, para a captura e o envio de imagens em tempo real, para uma plataforma em terra, se faz necessário, pois permite que se façam monitoramentos otimizados e permite que informações sobre regiões de interesse sejam repassadas aos operadores da missão durante o próprio voo. Exemplos de aplicações são o monitoramente e fiscalização de áreas de proteção ambiental, o monitoramente de desastres ambientais, operações de busca e de salvamento, etc. Nestes casos, sistemas embarcados de processamento de imagens acoplado aos VANTs podem sobrevoar regiões de interesse e dar um retorno instantâneo a respeito da área total de desmoronamento, de desmatamento, de alagamento, de queimadas, etc. Considerando a complexidade de tais sistemas embarcados, formado por diferentes dispositivos e funcionalidades, o uso do conceito de Middleware se mostra altamente adequado, pois viabiliza a integração modular de dispositivos heterogêneos, tais como câmeras, microcomputadores e Notebooks e oferece uma interface através da qual os recursos do sistema podem ser acessados.

Este artigo apresenta uma arquitetura distribuída, que utiliza o conceito de middleware [González, A. et al. 2011], [Douglas C. Schmidt 2002], e [Douglas C. Schmidt et al. 2004] para o desenvolvimento de aplicações baseadas em VANTs. Foi realizada a implementação de um sistema, baseado no modelo cliente/servidor, no qual VANTs, sobrevoando áreas de interesse, capturam imagens e as enviam para uma base em terra. As imagens são disponibilizadas aos usuários sob a forma de fotos, vídeos ou stream. O artigo está organizado da seguinte forma: a seção 2 apresenta algumas aplicações que utilizam VANT´s, na seção 3 uma visão geral do sistema desenvolvido é apresentada, a seção 4 descreve a arquitetura proposta, a seção 5 mostra detalhes da implementação e a seção 6 é dedicada às considerações finais.

2 SISTEMAS BASEADOS EM VANT´S

[Yoo et al., 2015] apresentam uma plataforma formada por uma esquadrilha de VANTs, utilizada em aplicações que necessitam que os VANTs troquem informações. Para o gerenciamento dos VANTs, é estabelecida e mantida uma comunicação entre a esquadrilha de VANTs e a estação de controle no solo. Além disso, cada VANT da esquadrilha pode se comunicar com os demais VANTs. Com as comunicações estabelecidas, a plataforma de gerência da esquadrilha de VANTs pode controlar a esquadrilha inteira e obter as

informações necessárias para o usuário. A plataforma construída, NetDrone, foi implementada e utilizada em uma aplicação de mapeamento de paisagens e em uma aplicação de prestação de serviços, na qual quatro VANTs se conectam a outros dispositivos além dos próprios VANTs VANTs (vídeo disponível em https://www.youtube.com/watch?v=lFaWsEmiQvw&feature=youtu.be).

O trabalho apresentado por [Yao-Hu, Yu-Ren and Ling-Jyh, 2015] descreve Krypto, um VANT para auxiliar nas operações de busca e salvamento. Krypto sobrevoa a área de desastre para detectar sinais wireless a partir de telefones celulares, o que possibilita a localização de possíveis vítimas. O sistema procura maximizar a área de pesquisa, minimizando os erros de localização, fazendo buscas por diferentes caminhos. Para procurar por sinais, Krypto cria um access point para receber conexões de celulares. Quando um celular se conecta ao access point, o sistema analisa o endereço IP e o MAC address e, continuamente, faz ping no endereço IP para obter pacotes de resposta e armazena os sinais recebidos com a posição corrente do VANT, obtida por GPS. Como Krypto voa ao redor da área do desastre de acordo com uma trajetória de vôo predeterminada, os sinais recebidos de diferentes locais são usados para estimar a localização do telefone móvel.

Um sistema de entrega de produtos é apresentado por [Mark, 2015]. No projeto, foi definido e implementado um sistema de entrega utilizando um VANT, usando hardware comercial e software livre, requisitos importantes do projeto. Foi usado um VANT, disponível comercialmente, modelo Robotics Iris Quadcopter, que possui o firmware Pixhawk (http://pixhawk.org/), open-source e transceptores de rádio para a comunicação com um computador (estação de controle de solo). A aplicação de controle de solo open source selecionada foi desenvolvido em Python, linguagem adequada por permitir uma rápida prototipagem. O resultado do trabalho realizado é um sistema de entregas que pode pegar e entregar pequenos pacotes, em uma distância de até mil metros. Também foi desenvolvido um protótipo de aplicativo de telefone para comunicação com o VANT, que possibilita o envio e recebimento de coordenadas e mensagens de status.

O artigo de [Eric, 2014] é relacionado ao uso de VANTs para realizar trabalhos “sujos” ou perigosos em locais remotos ou hostis aos homens. Os VANTs são potencialmente adequados ao estudo de fenômenos atmosféricos, tais como temporais, rajadas de vento, furacões, deslizamento de terras, incêndios em matas, etc. No caso das grandes tempestades e tornados, que ocorrem rapidamente e não duram muito tempo, os VANTs podem ser usados para capturar novas informações e permitir melhor compreensão desses fenômenos. O artigo apresenta os grandes desafios associados ao projeto de um

sistema para estudar grandes tempestades. Estes desafios foram divididos em três categorias principais. Em primeiro lugar, desafios regulatórios são descritos com base nas regras para a operação VANTs. Segundo, desafios logísticos são descritos com base na natureza nômade dos VANTs e em terceiro lugar, os desafios técnicos multidisciplinares são apresentados.

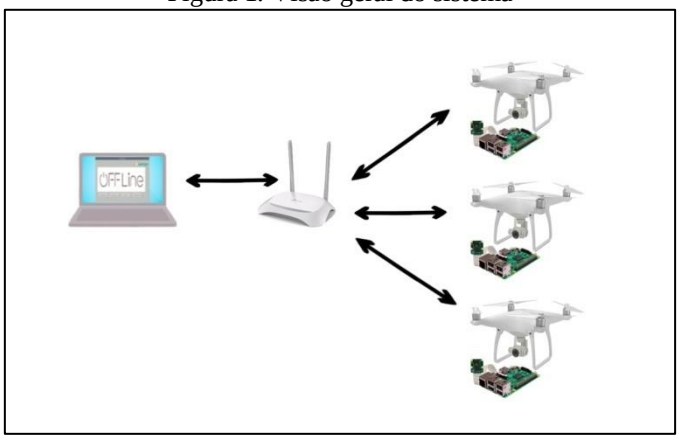
Nos sistemas apresentados, pode-se observar que todos possuem uma organização semelhante. O VANT sobrevoa áreas de interesse, sob o controle de uma base em terra. Considerando o modelo cliente/servidor, o VANT pode ser o cliente recebendo instruções e executando ações solicitadas por um servidor (a base) em terra. Todos os serviços oferecidos pela aplicação podem ser acessíveis por um conjunto de primitivas (API), implementadas por um middleware distribuído, rodando no cliente e no servidor.

3 VISÃO GERAL DO SISTEMA PROPOSTO

O Middleware de Comunicação em Tempo Real (MCTR) é uma ferramenta criada para facilitar o desenvolvimento de aplicações que tenham como base a transmissão de imagens entre dois dispositivos. Seu funcionamento é similar ao de um sistema operacional, onde o serviço é solicitado por meio de "chamadas de sistemas". Quando solicitado, o MCTR executa a ação desejada e retorna seu status. Ele contém dois níveis lógicos: nível usuário e nível de sistema. O primeiro é usado para realizar solicitações de serviço ao MCTR. No Nível sistema é onde são, de fato, executadas as ações solicitadas pelo usuário. O MCTR tem as seguintes funcionalidades: tirar uma foto, tirar fotos periódicas, mostrar imagem da câmera em tempo real e gravar vídeo. O sistema roda em Linux e foi inteiramente escrito em linguagem C/C++. A comunicação entre os dispositivos ocorre via uma rede wireless e foi implementada com o uso de sockets. Com o intuito de facilitar a manipulação de imagens e a conexão com a câmera, foi usada a ferramenta "OpenCV".

O sistema desenvolvido é baseado no modelo cliente/servidor. O servidor executa em um Laptop com sistema operacional Linux, que é a base na terra. O cliente roda em um Raspberry embarcado no VANT, que possui uma câmera acoplada. O sistema operacional usado no Raspberry é o Raspbian, uma distribuição Linux criada para rodar no Raspberry Pi. A Figura 1 apresenta uma visão geral do sistema.

Figura 1. Visão geral do sistema



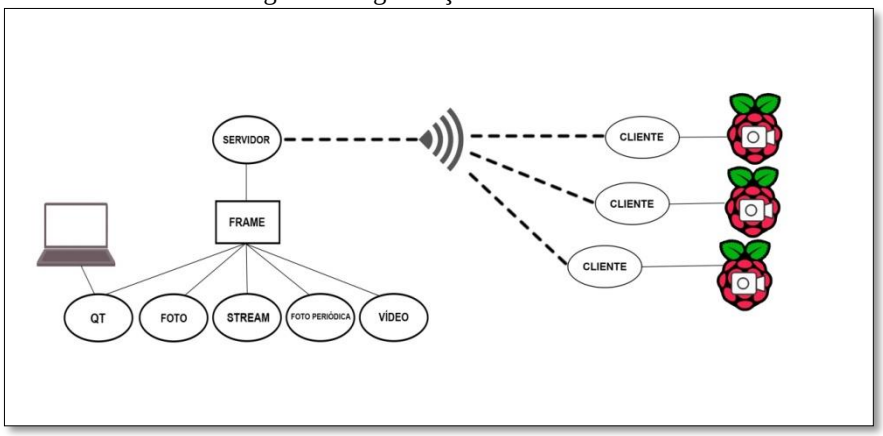
A comunicação cliente/servidor é realizada por intermédio de um roteador wireless. O cliente executa nos Raspberry e o servidor no computador base em terra. Na fase de inicialização, o servidor envia uma mensagem, via broadcast, que será recebida por todos os clientes dentro da área de cobertura da rede. Uma vez estabelecida à conexão, para cada cliente será criada uma thread servidor, para receber as imagens capturadas e enviadas pelo cliente. Essas imagens poderão ser visualizadas pelo usuário sob a forma de um stream, como uma foto ou poderá ser gerado um vídeo. Essas funções são selecionadas pelo usuário, a partir de uma interface gráfica, integrada ao programa servidor.

O programa cliente, que roda nos Raspberry embarcados nos VANT´s, continuamente aciona a câmera fotográfica e envia a imagem para o servidor. O servidor recebe cada foto e executa a função selecionada pelo usuário.

4 ARQUITETURA DO MIDDLEWARE

A organização do Middleware é apresentada na Figura 2. O servidor possui um conjunto de threads, uma para receber as imagens do cliente, uma que implementa a interface com o usuário e uma para cada funcionalidade disponível ao usuário. O cliente possui uma thread, que continuamente, em um intervalo parametrizado, captura uma imagem e a envia para a thread servidor, que a armazena em um buffer. Sincronamente, as threads, cujas funções estão habilitadas pelo usuário (ex. gravar fotos a cada segundo e mostrar as imagens continuamente (stream), são acionadas e executam suas funções. A thread servidora volta a receber uma nova imagem quando for sinalizada, o que ocorre quando todas as funções selecionadas pelo usuário tiverem sido executadas.

Figura 2. Organização do Middleware



A seguir serão apresentadas as threads usadas, juntamente com a descrição de suas funcionalidades.

4.1 THREADS PRESENTES NO SERVIDOR

- th_server_master: Esta thread é a responsável por estabelecer as configurações iniciais do sistema e iniciar o serviço da thread th_server. Inicialmente manda uma mensagem broadcast informando que se trada de um dispositivo servidor e necessita dos serviços de um dispositivo cliente. Posteriormente estabelece as configurações inicias do sistema e dispara a threadth_server.
- th_server: É responsável por receber as imagens transmitidas pelo cliente e também por iniciar ou finalizar os demais serviços oferecidos pelo servidor. É responsável por iniciar e finalizar as threads th_stream, th_photo_period e th_video.
- th_stream: É responsável por exibir, em uma janela, o fluxo de frames recebidos pelo servidor, possibilitando ao usuário acompanhar em tempo real as imagens capturadas pela câmera.
- th_photo_period: Responsável por salvar fotos em intervalos constantes. O intervalo de tempo entre uma foto e outra é passado para essa thread por um parâmetro, que define o intervalo de tempo, em segundos.
- th_video: Responsável por gravar o vídeo transmitido para o servidor. O vídeo começa a ser gravado com a chamada da função video_start e é finalizado com a chamada da função video_stop. O video gravado é salvo com a extensão “.avi”.
- QT: é uma thread implementada na linguagem QT que implementa a interface com o usuário. Exibe as imagens no monitor, de acordo com a opção selecionada pelo usuário (stream, foto, etc.).

4.2. THREADS PRESENTES NO CLIENTE

- th_client_master: Esta thread é a responsável por definir as configurações iniciais do cliente e iniciar o serviço da thread th_client. Define as configurações iniciais e, na sequência, dispara a thread thr_cli.
- thr_cli: É a thread responsável por capturar a imagem da câmera e posteriormente transmiti-la ao servidor.

5 DETALHES DA IMPLEMENTAÇÃO

O Middleware possui dois níveis lógicos, um que aparece aos usuários como um conjunto de funcionalidades fornecidas pelo sistema, e outro, que implementa as funções do Middleware.

5.1 NÍVEL USUÁRIO

O nível lógico do usuário contém o conjunto de serviços suportados pelo Middleware e foi implementado em C/C++, disponível aos usuários sob a forma de um mecanismo semelhante às chamadas de sistema, em um sistema operacional. É descrito em um conjunto de dois arquivos: usercall.hpp e usercall.cpp. O usercall.hpp contém os cabeçalhos das "chamadas de sistema" do MCTR, que são init_server(), intphoto_imed(), int photo_period(int period), int video_start(); int video_stop(), int stream_start() e int stream_stop(). A estrutura básica dessas chamadas de sistema é o seu nome e seu conjunto de parâmetros, específico de cada serviço. Abaixo segue o exemplo de uma chamada de sistema:

Figura 3. Código fonte da função photo_period

```
int photo_period(int period)
{
Parameters arg;
arg.callnumber = PHOTO_PERIOD;
arg.p0 = (unsigned char *)period;
return (int) invoke(&arg);
}
```

Na primeira linha da função, aparece o tipo parameters, que é uma struct declarada em "invoke.hpp", que contêm um identificador de chamada de sistema (callnumber) e seus parâmetros (p0, p1, p2, p3). Na última linha da função, aparece a chamada da função invoke, que tem como o objetivo realizar a troca de contexto. A troca de contexto nada

mais é que a interrupção momentânea do programa de aplicação para que seja solicitado o serviço do MCTR.

5.2. NÍVEL MIDDLEWARE

É o nível lógico do sistema que executa os serviços solicitados. Possui dois arquivos: “invoke.hpp” e “invoke.cpp”. A invoke.hpp contém os cabeçalhos das funções e a declaração de variáveis e estruturas de dados. A invoke.cpp executa o serviço solicitado. Recebe como parâmetro o número e os parâmetros da chamada de sistema.

Figura 4. Código fonte da função invoke

```
int invoke(parameters *arg){
switch (arg->callnumber) {
case SERVER_MASTER_INIT:
pthread_create(&serv_master, NULL, server_master, (void *) arg);
break;
case CLIENT_MASTER_INIT:
pthread_create(&cli_master, NULL, client_master, (void *) arg);
break;
case STREAM_START:
pthread_create(&t1, NULL, stream, (void *) arg);
break;
case STREAM_STOP:
control.stream = OFF;
break;
case STREAM_QT:
stream_qt();
break;
case PHOTO_IMED:
photo();
break;
case PHOTO_PERIOD_START:
delay = (long)*(arg->p0);
pthread_create(&t2, NULL, photo_periodical, (void *) arg);
break;
case PHOTO_PERIOD_STOP:
control.photo = OFF;
break;
case VIDEO_START:
pthread_create(&t3, NULL, video, (void *) arg);
break;
case VIDEO_STOP:
control.video = OFF;
break;
default:
break;
  }
}
```

A exceção da função photo, as demais funcionalidades são implementadas por threads, que serão apresentadas a seguir.

A thread stream, Figura 5, exibe em tempo real, os frames recebidos pelo servidor em terra.

Figura 5. Código fonte da função stream

```
void *stream(void *arg)
{
char window[] = {"-===<| CAM |>===-"};
namedWindow(window, WINDOW_NORMAL);
while(control.stream == ON) {
sem_wait(&sem_stream);
imshow(window, frame);
waitKey(20);
}
pthread_exit(0);
}
```

A thread vídeo (código na Figura 6), grava os frames recebidos pelo servidor em um arquivo .avi.

Figura 6. Código da thread video

```
void *video(void *arg)
{
while (control.video == ON) {
sem_wait(&sem_video);
video.write(frame);
video.release();
  }
pthread_exit(0);
}
```

A thread photo_periodical (código na Figura 7), controla a aquisição de fotos periodicamente. O tempo de espera entre a aquisição de fotos, em segundos, e configurável pelo usuário.

Figura 7. Código fonte da função photo_periodical

```
void *photo_periodical(void *arg){
parameters *par = (parameters *)arg;
while(control.photo == ON) {
photo();
sleep((int)delay);
}
}
```

A thread photo (código na Figura 8), salva o último frame recebido pelo servidor em um arquivo jpg. Esse arquivo possui a data e a hora na qual foi tirada a foto.

Figura 8. Código da função photo

```
void * photo(){
sem_wait(&sem_foto);
“escreve no arquivo data e a  hora da foto”
imwrite(foto, frame, foto_parametros);
return;
}
```

A comunicação entre o cliente, rodando no Raspberry embarcado no VANT e o servidor, executando no Laptop, é realizada por uma thread rodando no cliente, thr-cli() e outra no servidor, thr_server(). A thread thr_cli() (Figura 9) captura a imagem e a envia, para a thread thr_server() (Figura 10), usando o protocolo UDP/IP. A imagem recebida pela thread é disponibilizada para as demais threads que rodam no servidor, e que implementam as funcionalidades do sistema.

Figura 9. Estrutura da thread cliente

```
void *thr_cli () {
while(1) {
#ifdef ENABLE_RASPICAM
CAM.grab();
CAM.retrieve(frame); //faz a leitura do frame
#endif
#ifdef ENABLE_USBCAM
cam.read(frame); //faz a leitura do frame
#endif
bytes = sendto(s1, frame,sizeof(frame),0,(struct sockaddr *)&cli, slen);
}
```

Figura 10. Estrutura da thread servidor

```
void *thr_server(void *arg)
{
for (;;) {
     recvMSG = recvfrom(s2,&iBUFF,65540,0,(sockaddr *)&serv, &slen);
     frame = iBUFF ;
     “acorda todas as threads que implementam os servicos”
     “fica em wait e será acordada pela última thread”
    }
}
```

6 CONSIDERAÇÕES FINAIS

Middleware é um conceito estratégico no desenvolvimento de sistemas distribuídos embarcados de tempo real. Serve como elemento de união entre os programas de aplicação e os sistemas operacionais. Se utiliza da pilha de protocolos de rede para fornecer mecanismos necessários à implementação dos serviços de comunicação.

A implementação realizada possui dois níveis lógicos, nível usuário, que disponibiliza a API, a partir da qual os serviços do sistema são acessíveis, e o nível sistema, que contém a implementação das funcionalidades do sistema. A utilização do conceito de Middleware, com sua estrutura modular, permite que correções, adição de novas funcionalidades, melhorias nas funções existentes, etc. sejam mais facilmente realizadas. Permite também que diferentes componentes sejam produzidos por equipes diferentes, visto que as interfaces são uniformes.

O sistema distribuído, com um servidor executando em um Laptop e os clientes rodando embarcados em um Raspberry acoplado em um VANT, está operacional. É possível gravar fotos, gravar vídeos e ver imagens em tempo real. A velocidade obtida na transferência das imagens garante a visualização em tempo real. À distância percorrida pelo VANT está limitada ao alcance do roteador wireless utilizado, que no caso é de aproximadamente 120 metros. Para distâncias maiores é necessário utilizar roteadores com maior área de cobertura de sinal.

Para a troca de mensagens entre o servidor e os clientes, inicialmente foi implementada uma versão do sistema usando socketes TCP/IP. Essa versão não atendia os requisitos de tempo real da aplicação. Uma nova versão, usando UDP/IP foi desenvolvida, e se mostrou cerca de três vezes mais rápida que a versão anterior. Essa versão permite a exibição de imagens em tempo real. O sistema pode ser utilizado em operações de busca e salvamento, vigilância, monitoramento de rodovias, etc.

Como trabalhos futuros, pretende-se substituir o roteador wireless pela tecnologia LoRa[Martin, 2016], que possibilita a comunicação sem fio de longo alcance, com baixo consumo de energia. Além disso, essa nova versão deverá enviar os frames para um servidor na nuvem, onde serão armazenados em uma base de dados e poderão ser visualizados por notebooks ou smartphones.

REFERÊNCIAS

Celso Maciel da Costa, Philippe Silveira, Alleff Dymytry, Maike **B**ressan, Adriane Parraga, Letícia Vieira Guimarães (2018) Uma Arquitetura Distribuída para Sistemas de Tempo Real Baseados em VANT´s**. Vídeo:** https://drive.google.com/file/d/11vthtGjUMuxLbU9OedHzsUeyUnZVUG-v/view

Douglas C. Schmidt (2002) Middleware for real-time and embedded systems. Communication ACM 45, 6 (June 2002), 43-48. DOI: https://doi.org/10.1145/508448.508472

Douglas C. Schmidt, Aniruddha Gokhale, Richard E. Schantz, and Joseph P. Loyall (2004) Middleware R&D challenges for distributed real-time and embedded systems. SIGBED Rev. 1, 1 (April 2004).

Eric Frew (2014). Storm-chasing drones. XRDS 20, 3 (March 2014), 18-23.

González, A., Mata, W., Villaseñor, L. et al. (2011) µDDS: A Middleware for Real-time Wireless Embedded Systems. Journal of Intelligent & Robotic Systems, 2011.

Luca Mottola (2015) Real-world Drone Sensor Networks: A Multi-disciplinary Challenge. In Proceedings of the 6th ACM Workshop on Real World Wireless Sensor Networks (RealWSN '15). ACM, New York, NY, USA.

Mark O. Milhouse (2015) Framework for Autonomous Delivery Drones. In Proceedings of the 4th Annual ACM Conference on Research in Information Technology (RIIT '15). ACM, New York, NY, USA,

Martin Bor, John Vidler, and Utz Roedig (2016). LoRa for the Internet of Things. In Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN ’16). Junction Publishing, USA, 361–366.

Seungho Yoo, Kangho Kim, Jongtack Jung, Albert Y. Chung, Jiyeon Lee, Suk Kyu Lee, Hyung Kyu Lee, and Hwangnam Kim (2015) A Multi-Drone Platform for Empowering Drones’. The 21st Annual International Conference on Mobile Computing and Networking. MobiCom’15, September 7–11, 2015, Paris, France.

Yao-Hua Ho, Yu-Ren Chen and Ling-Jyh Chen (2015) Krypto: Assisting Search and Rescue Operations using Wi-Fi Signal with UAV. In Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet '15). ACM, New York, NY, USA.