Protótipo para Deteção de Contentores de Resíduos com Recurso a Técnicas de Visão Computacional

Miguel Valente[1], Hélio Silva[2], João Caldeira[3], Vasco Soares[3], Pedro Gaspar[4]

[1] Instituto Politécnico de Castelo Branco, Portugal
mvalente@ipcbcampus.pt
[2] EVOX Technologies, Portugal
helio.silva@evox.pt
[3] Instituto Politécnico de Castelo Branco, Instituto de Telecomunicações, Portugal
{jcaldeira,vasco.g.soares}@ipcb.pt
[4] Universidade da Beira Interior, Portugal
dinis@ubi.pt

Resumo. Este artigo apresenta o desenho e implementação de um protótipo funcional para avaliar e validar a utilização de técnicas de visão computacional, na identificação de contentores de resíduos no contexto de uma cidade inteligente. Este protótipo recorre à utilização de uma rede neuronal convolucional YOLO e de um microcomputador Jetson Nano da Nvidia. Comparativamente ao método atual de identificação de contentores de resíduos por radiofrequência, esta abordagem é mais ágil e diminui os recursos necessários para implementação, contribuindo para poupar nos gastos logísticos e de implementação da gestão inteligente de resíduos.

Palavras Chave: Cidades Inteligentes, Gestão de Resíduos, Visão Computacional, Deteção de Contentores de Resíduos.

1 Introdução

As cidades inteligentes são hoje uma realidade. Resultam de iniciativas que combinam tecnologia atual e emergente com infraestruturas existentes ou novas, sensores IoT, e plataformas para trabalhar as informações recolhidas. Procuram oferecer soluções a desafios tais como melhorar a qualidade de vida das pessoas, a competitividade económica e a sustentabilidade.

A gestão inteligente de resíduos é um tema pertinente no contexto das cidades inteligentes. Tem por objetivo, entre outros, monitorizar o estado de enchimento dos contentores de resíduos e otimizar as rotas dos veículos de recolha em função do mesmo. Para tal, é fundamental a correta identificação dos contentores de resíduos. A abordagem atual, baseada em etiquetas para identificação por radiofrequência (RFID), apresenta um conjunto de limitações. Nomeadamente, a necessidade de instalar antenas nos veículos de recolha e etiquetas RFID nos contentores de resíduos. Para concretizar esta operação, torna-se necessário recolher, limpar e modificar os contentores existentes para adicionar o módulo RFID. Este é um processo demorado, complexo e de custo

elevado. Além disso, a detecção de dispositivos RFID não operacionais apenas é possível durante a sua utilização, ou seja, no local da sua instalação. Esta condicionante limita, por exemplo, a detecção remota de equipamentos não operacionais.

Neste âmbito, este artigo dá continuidade à investigação conduzida pelos mesmos autores, que demonstrou e avaliou a utilização de diferentes técnicas de visão computacional para substituir o método de identificação de contentores de resíduos via RFID [1, 2, 3]. Neste artigo, apresenta-se o desenho, implementação e avaliação de um protótipo funcional para demonstração deste conceito. O protótipo recorre à utilização de uma rede neuronal convolucional YOLOv2 [4] e de um microcomputador Jetson Nano da Nvidia [5].

Este artigo encontra-se estruturado da seguinte forma. A Secção 2 apresenta o desenho e implementação do protótipo. De seguida, na Secção 3 é avaliado o desempenho do protótipo, discutindo-se os resultados obtidos. Finalmente, na Secção 4, apresentam-se as conclusões e trabalho futuro.

2 Protótipo

Esta secção apresenta o desenho e implementação do protótipo funcional para demonstração do conceito, cuja arquitetura se apresenta na Fig. 1. O protótipo contém quatro componentes essenciais para o seu funcionamento: a rede neuronal YOLOv2, o Jetson Nano, o servidor e a base de dados. A rede neuronal é usada de duas formas: para deteção de contentores e para o treino de novas iterações (da rede neuronal). Estas ações são realizadas em componentes distintos [1], [2], no Nano e no servidor. O Nano, que deve ser montado num camião de recolha de resíduos, é o componente responsável pela deteção de contentores. O servidor é responsável pelo treino de novas iterações da rede neuronal, bem como para servir de ponto de comunicação entre o Nano e a base de dados. Na base de dados será guardada toda a informação transmitida pelo Nano, nomeadamente, as deteções efetuadas, a precisão da deteção, os *frames* onde ocorreu deteção e a localização geográfica no momento em que ocorre a deteção.

Esta secção encontra-se dividida em duas subsecções: na primeira é apresentado o *hardware* envolvido na construção do protótipo e na segunda é descrita a implementação de *software* do protótipo.

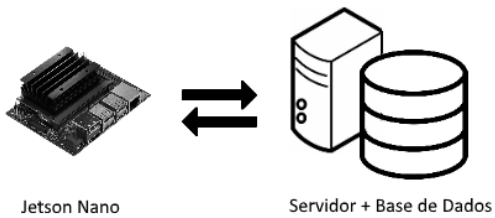

Fig. 1. Diagrama da arquitetura do protótipo.

2.1 Componente de *hardware*

O protótipo apresentado tem por base um Jetson Nano [5] sendo este o pilar central responsável por executar a rede neuronal para realizar a deteção, bem como os processos secundários de comunicação com o servidor. O Jetson Nano está montado numa *baseboard* que vem juntamente com o *Developer Kit*. O Nano requer atenção especial em relação ao método de como a energia lhe é fornecida; existem dois métodos principais, o primeiro faz uso da interface Micro-USB podendo esta receber até 5V⎓2A (Volts/Amperes). O segundo método utiliza um conector DC (*direct current*) recebendo até 5V⎓4A. Para selecionar o segundo método de alimentação é necessário colocar um *jumper* no conector par J48 da *baseboard*. No contexto deste protótipo, o fornecimento de energia é feito utilizando o segundo método, isto devido às necessidades energéticas do Nano durante os uso das redes neuronais.

O outro componente de hardware que é preciso ter em consideração é a câmara; esta pode ser CSI (*Camera Serial Interface*) ou câmara USB (*Universal Serial Bus*). No protótipo é utilizada uma Raspberry Pi Camera Module v2, a qual é conectada na porta J13 da *baseboard*.

O protótipo funcional pode ser visto na Fig.2. O ecrã está presente apenas para efeitos demonstrativos e de validação de conceito. No protótipo final o ecrã é desnecessário, pois toda a interação é feita através de *headless* SSH (*Secure Shell*) [6], ou seja, toda a interação com o Nano é feita sem periféricos.

Para o servidor considera-se uma máquina com um sistema operativo baseado em GNU/Linux [7] tal como Ubuntu Server [8] ou Ubuntu Bionic [9]. Neste protótipo a base de dados é simplificada e partilha o espaço lógico com o servidor, funcionando através de diretorias com um destino específico; uma diretoria onde estão guardadas as iterações das redes neuronais, outra guarda os ficheiros de texto com informação sobre deteções e localização geográfica, e finalmente outra para guardar as *frames* obtidas durante a deteção.

Fig. 2. Protótipo funcional.

2.2 Componente de *software*

O protótipo a executar no Jetson Nano utiliza o sistema operativo de raiz disponibilizado pela NVIDIA, o JetPack 4.2 [10]. O sistema operativo é baseado em Ubuntu Bionic e contem as drivers necessárias para utilizar o Nano assim como CUDA [11] e cuDNN [12] pré-instaladas. CUDA é uma plataforma desenvolvida pela NVIDIA para processamento paralelo utilizando uma GPU. cuDNN é uma biblioteca de primitivas para *deep neural networks* que fornece rotinas altamente otimizadas para ações relacionadas com a utilização de redes neuronais. Este *software* é essencial para permitir uma utilização eficiente de redes neuronais, tirando partido do poder de processamento paralelo de uma GPU.

Para realizar deteções e treinar novas iterações de redes neuronais, recorre-se ao *software darknet* [13], pelo que para utilizar este *software* foi também necessário instalar biblioteca OpenCV 4.0 [14]. A *darknet* é utilizada pelo servidor e pelo Nano – no servidor para treinar novas iterações de redes neuronais e no Nano para efetuar as deteções dos contentores de resíduos. De modo a manter a iteração mais recente da rede neuronal, é feita uma comunicação com o servidor na diretoria onde se encontram as iterações das redes neuronais. Para automatizar este processo foi utilizado o *software rsync*, *cron* e chaves RSA (Rivest-Shamir-Adleman).

Rsync é um programa usado para fazer a sincronização de ficheiros e pastas, que funciona localmente mas também funciona através da rede. De modo a garantir a segurança da transferência o *rsync* faz uso do protocolo SSH. *Cron* é um serviço Unix que executa comandos agendados na *crontab*, um ficheiro em que cada linha equivale a um *cronjob* com o respetivo comando ou *script* a executar. As chaves RSA são usadas para garantir autenticação sem ser necessário a introdução de uma *password*.

Para fazer a atualização da rede neuronal é executado um *bash script* no Jetson Nano denominado de "sincroniza" (ver Fig. 3), este *script* utiliza o *rsync* para verificar se houve alguma mudança na diretoria do servidor onde está guardada a última versão da rede neuronal. Se existir alguma alteração, o *rsync* é responsável por transferir a nova versão da rede neuronal. Sempre que o *rsync* é executado, é gerado um ficheiro *log* com informação da transferência e verificação de ficheiros, que por sua vez é guardado no Jetson Nano. Este ficheiro contém detalhes sobre a transferência e pode ser consultado para fazer *debugging* ou para verificar se a transferência foi completada corretamente. Para automatizar este processo recorreu-se ao uso de um *cronjob* que se encontra escrito na *crontab* (ver Fig. 4). Este *cronjob* executa o *script* "sincroniza" de hora em hora, e caso ocorra um erro este é escrito no ficheiro *erro.log*. Para este processo ser automático é ainda necessário que o acesso ao servidor seja feito sem necessitar de autenticação através de *password*. Para este feito é gerado um par de chaves RSA no Nano. A chave publica é enviada através de *ssh-copy-id* para o servidor (ver Fig. 5). Desta maneira, o processo de atualização da rede neuronal é completamente automático sem ser necessário interferência por parte de um utilizador.

```
#!/bin/sh
/usr/bin/rsync -avhPS --chmod=a+rwx --delete servidor@servidores.org:'"/home/transfer_files_serv/"' "/home/transfer_fil
es_nano" 2>&1 > /home/sincroniza_logs/detecao.log
```

Fig. 3. *Bash* script "sincroniza" para transferir rede neuronal.

```
*/60 * * * * /usr/bin/flock -xn /tmp/flocktmp.lock -c "/home/scripts/sincroniza" >> /home/erro.log 2>&1
```

Fig. 4. *Cronjob* para executar o *bash script*.

```
:~$ sudo ssh-copy-id -i /root/.ssh/id_rsa.pub servidor@servidores.org
```

Fig. 5. Comando para copiar a chave publica usando *ssh-copy-id.*

A *darknet* fornece um conjunto de opções, permitindo fazer a deteção através de múltiplas imagens, vídeo ou *input* de uma câmara. Durante as deteções é produzido um *output* de texto contendo: os *frames* processados por segundo (FPS), os *frames* onde ocorreu deteção e a informação sobre os contentores detetados com as posições e percentagens de precisão correspondentes, sendo a posição em relação ao *frame* capturada (ver Fig. 6). Cada vez que se inicia um processo de deteção no Nano, é criado um ficheiro de texto onde é escrito o *output* gerado, assim como o *frame* de deteção correspondente. Estes ficheiros são guardados numa diretoria onde ocorre um processo semelhante à atualização da rede neuronal, mas desta vez do lado do servidor.

No servidor existe um outro *script* "sincroniza" que verifica a diretoria onde os ficheiros das deteções se encontram. Do mesmo modo, existe um *cronjob* que executa o *script* a cada minuto, verificando se existe alguma diferença na diretoria e nesse caso transferindo os novos ficheiros.

```
FPS:0.0
Objects:

indeferenciado: 93%     (left_x:  182   top_y:  108   width:   65   height:   67)
indeferenciado: 75%     (left_x:  110   top_y:  114   width:   65   height:   53)
verde: 65%      (left_x:  133   top_y:   85   width:   46   height:   24)
azul: 78%       (left_x:  189   top_y:   86   width:   54   height:   17)
amarelo: 75%    (left_x:   72   top_y:   83   width:   63   height:   57)
amarelo: 53%    (left_x:   98   top_y:   83   width:   46   height:   27)

FPS:-0.0

 cvWriteFrame
Objects:

indeferenciado: 93%     (left_x:  182   top_y:  108   width:   65   height:   67)
indeferenciado: 75%     (left_x:  110   top_y:  114   width:   65   height:   53)
verde: 65%      (left_x:  133   top_y:   85   width:   46   height:   24)
azul: 78%       (left_x:  189   top_y:   86   width:   54   height:   17)
amarelo: 75%    (left_x:   72   top_y:   83   width:   63   height:   57)
amarelo: 53%    (left_x:   98   top_y:   83   width:   46   height:   27)
```

Fig. 6. *Output* gerado durante o processo de deteção.

3 Resultados

Nesta secção apresentam-se os resultados do protótipo, na execução da rede neuronal YOLOv2 com vista à deteção de contentores de resíduos. A rede neuronal utilizada nestes testes contem um nível de precisão de deteção na ordem dos 92% [1], [2]. Considera-se que esta precisão se aplica ao vídeo analisado. A análise dos resultados foca-se na capacidade de processamento de imagens do Nano, utilizando a rede neuronal e não na precisão em si, visto que esta já está estabelecida.

De modo a obter resultados num ambiente controlado, foi utilizado um vídeo com resolução de 1920*1080 codificado em mp4, que contem vários exemplos de contentores (ver Fig.7). O tempo de duração do vídeo não é uma métrica importante, pois a leitura deste é feita através dos *frames* e não é relevante o tempo de exposição de cada *frame*. Neste caso o vídeo utilizado contém 3306 *frames*. No caso do YOLOv2 a resolução original do vídeo não interessa, pois o algoritmo redimensiona a imagem durante o processo de deteção, para o tamanho descrito no seu ficheiro de configuração, que neste caso é de 416*416. Devido a esta mecânica do algoritmo YOLOv2, independentemente da resolução dos *frames*, o tempo de processamento é o mesmo. Isto é demonstrado na Tabela 1.

No decurso dos testes, todos os ficheiros (*outputs* de deteções e vídeos) estão armazenados no Nano. Toda a análise também foi efetuada no Nano. Para obter estes dados foi seguido um procedimento – redimensionar o vídeo original em quatro resoluções diferentes utilizando *ffmpeg* (um utilitário Linux). De seguida, utilizou-se a *darknet* para fazer a deteção de contentores em cada um dos vídeos, sendo guardado o *output* gerado (ver Fig.6), assim como o vídeo com as deteções (ver Fig.8). Para obter a média de FPS de cada processo de deteção, separou-se a informação sobre os FPS utilizando os utilitários Linux *grep* e *cut*, e calculou-se a média utilizando o *awk*. A Fig. 9 demonstra a utilização destes utilitários. Para a obtenção do tempo usou-se o utilitário Linux *time*, em concreto o *output real*, que reflete o tempo total que demora a analisar o vídeo.

Desta análise conclui-se que demora cerca de 0,21 segundos para processar um *frame*, identificando os contentores presentes. Este tempo aproxima-se do *real-time* e demonstra a aplicabilidade do protótipo aqui apresentado no mundo real. Salienta-se que este é um valor obtido utilizando uma rede neuronal, que apesar de ser considerada leve na literatura da computação visual [15], [16], tem versões *light* destinadas à computação móvel. No entanto, estas abdicam de precisão para não exigirem tanto processamento.

Tabela 1. Tempo de processamento e média de FPS de vídeos com diferentes resoluções.

	TEMPO	**MÉDIA DE FPS**
VÍDEO (1920*1080)	2m32,796s	4.8
VÍDEO (960*540)	2m38,780s	4.8
VÍDEO (480*270)	2m37,879s	4.7
VÍDEO (240*136)	2m36,749s	4.7

Fig. 7. Frames do vídeo.

Fig. 8. O mesmo *frame* de vídeos com diferentes resoluções.

```
mvalente@jetnano:~/tools/darknetAB/FPS$ cat VID_2_resized_D.txt | grep FPS | cut -d ':' -f 2 > exemplo.txt
mvalente@jetnano:~/tools/darknetAB/FPS$ awk '{ counter+=1 } { sum += $1 } {avg=sum/counter} END { print avg } ' exemplo.txt
4.8345
mvalente@jetnano:~/tools/darknetAB/FPS$
```

Fig. 9. Processo de obtenção da média de FPS em cada processo de deteção.

4 Conclusões e Trabalho Futuro

O trabalho aqui apresentado resulta de um projeto de investigação para proposta de uma solução para a deteção de contentores de resíduos, utilizando técnicas de visão computacional. Nesse sentido, teve como principal objetivo mostrar o desenho,

implementação e avaliação de um protótipo funcional para demonstração do conceito. Considerou-se a utilização de uma rede neuronal convolucional YOLOv2 e de um microcomputador Jetson Nano da Nvidia.

Os resultados obtidos confirmam que a abordagem proposta é capaz de identificar diferentes tipos de recipientes de resíduos, usando imagens, vídeo ou captura de vídeo em tempo real. Comparativamente ao método atual de identificação por radiofrequência, esta abordagem, além de eficaz e eficiente, é ágil e reduz os custos de implementação e manutenção.

Para trabalho futuro pretende-se substituir todo o processo que envolva comunicação entre o servidor e o Nano por *web services* com um *front-end* do lado do servidor. O *front-end* permitirá configurar o modo como a atualização da rede neuronal é feita, bem como se processa a transferência de ficheiros entre o Nano e o servidor.

Agradecimentos

Os autores expressam o seu agradecimento às empresas EVOX Technologies e InspiringSci, Lda pelo interesse e contribuição, determinantes para a concretização deste trabalho.

Referências

1. M. Valente et al., “Detection of Waste Containers Using Computer Vision,” Appl. Syst. Innov., vol. 2, no. 1, p. 11, Mar. 2019.
2. M. Valente, H. Silva, J. M. L. P. Caldeira, and V. N. G. J. Soares, “Técnicas de Visão Computacional para a Deteção de Contentores de Residuos,”. in 14th Iberian Conference on Information Systems and Technologies (CISTI'2019), Coimbra, Portugal. 2019.
3. M. Valente, H. Silva, J. M. L. P. Caldeira, and V. N. G. J. Soares, “A New Approach Towards Waste Container Detection in Smart Cities,”. in 11th Conference on Telecommunications (CONFTELE 2019), Lisboa, Portugal. 2019.
4. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.
5. NVIDIA Corporation, JETSON NANO DEVELOPER KIT. NVIDIA Corporation, 2019.
6. T. Ylonen, “SSH manual page.” 2012.
7. F. S. Foundation, “GNU/Linux,” 2019. [Online]. Available: https://www.gnu.org/software/bash/manual/. [Accessed: 10-Jun-2019].
8. U. D. Team, “Ubuntu Server,” 2019. [Online]. Available: https://help.ubuntu.com/lts/serverguide/index.html. [Accessed: 10-Jun-2019].
9. U. D. Team, “Ubuntu Bionic,” 2019. [Online]. Available: https://help.ubuntu.com/lts/ubuntu-help/index.html. [Accessed: 10-Jun-2019].
10. NVIDIA, “JetPack 4.2,” 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetpack. [Accessed: 10-Jun-2019].
11. NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture Programming Guide. NVIDIA Corporation, 2007.
12. S. Chetlur et al., “cudnn: Efficient primitives for deep learning,” arXiv Prepr. arXiv1410.0759, 2014.

13. A. Bezlepkin, “darknet,” GitHub repository. GitHub, 2019.
14. G. Bradski, “The OpenCV Library,” Dr. Dobb’s J. Softw. Tools, 2000.
15. S. Achatz, “State of the Art of Object Recognition Techniques,” Sci. Semin. Neurocientific Syst. Theory, 2016.
16. Z.-Q. Zhao, P. Zheng, S. Xu, and X. Wu, “Object detection with deep learning: A review,” arXiv Prepr. arXiv1807.05511, 2018.