University of Verona

DEPARTMENT OF COMPUTER SCIENCE
Master Degree in Computer Science and Engineering

Model-based design of at-the-edge heterogeneous Robotics Applications based on ROS

Candidate:
Simone Girardi

Thesis advisor:
Prof. Nicola Bombieri

Research supervisor:
Prof. Franco Fummi

Thesis submitted in 2020

Abstract

Developing distributed robotics applications on embedded devices, we have to deal with the diversity of the applications and the different platforms where these applications run. At the state of the art there are some solitutions that allow us to develop robotics applications and deploy them on embedded boards. The problem is that none of these solutions allows us to be sufficiently accurate to guarantee the functioning of the entire system, esapecially if we want to increase its complexity. To solve the problem we must to take account of the necessary reasources to run the applications and the constraints imposed by the limits of the devices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis outline

This thesis is organised in two main workflows: *Verification* and *Methodology*. Chapter 2 is an overview over the research contribution in the field of deep learning and the other tools discussed in the next chapter of this thesis. The first workflow is described in chapter 3, where a complete inspection and verification was made on the ORB SLAM algorithm to garantee a deterministic behaviour in a sequential contex. The second workflow described in chapter 4 proposes a solution to the problem of integrating heterogeneous robotic applications. With the support of edge computing platforms like Kubeedge, some experiments was made to deploy automatically our integrated system from the host (cloud) to the edge. Finally in chapter 5 the experimental results are showed and discussed.

Chapter 2

Background

2.1 Deep Learning

2.1.1 Motivations

Deep learning has recently been highly successful in machine learning across a variety of application domains, including computer vision, natural language processing, and big data analysis, among others. For example, deep learning methods have consistently outperformed traditional methods for object recognition and detection in the ISLVRC Computer Vision Competition since 2012 [26]. However, deep learning's high accuracy comes at the expense of high computational and memory requirements for both the training and inference phases of deep learning. Training a deep learning model is space and computationally expensive due to millions of parameters that need to be iteratively refined over multiple time epochs. Inference is computationally expensive due to the potentially high dimensionality of the input data (e.g., a high-resolution image) and millions of computations that need to be performed on the input data.

2.1.2 Definitions

As described in [12], the modern term "deep learning" goes beyond the neuroscientific perspective engineering applications on the current breed of machine learning models. It appeals to a more general principle of learning *multiple levels of composition*, which can be applied in machine learning frameworks that are not necessarily neurally inspired. Deep learning is a subset of AI and machine learning and differs in that they can automatically

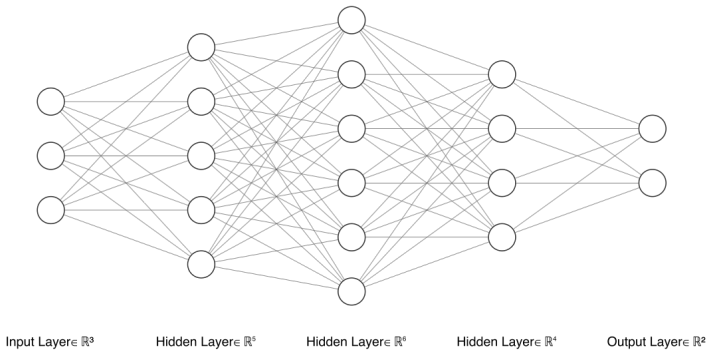

Figure 2.1: DNN example with image classification

learn representations from data such as images, video or text, to be used for classification without introducing hand-coded rules or human domain knowledge. Their highly flexible architectures can learn directly from raw data and can increase their predictive accuracy when provided with more data. A deep learning prediction algorithm, consists of a number of layers, as shown in Fig. 2.1.

In deep learning *inference*, the input data pass through the node's layers in sequence, and each layer performs matrix multiplications on the data. The output of a layer is usually the input to the subsequent layer. After data are processed by the final (fully connected) layer, the output is either a feature or a classification value. When the model contains many layers in sequence, the neural network is known as a deep neural network (DNN). When the matrix multiplications include convolutional filter operations, the model is named convolutional neural networks (CNNs), which is common for image and video processing contexts. There are also DNNs designed especially for time series prediction; these are called recurrent neural networks (RNNs), which have loops in their layer connections to keep state and enable predictions on sequential inputs.

In deep learning *training*, the computation proceeds in reverse order. Given the ground-truth training labels, multiple passes are made over the layers to optimize the parameters of each layer of matrix multiplications,

starting from the final layer and ending with the first layer. The algorithm used is typically stochastic gradient descent (SGD). In each pass, a randomly small subset of N input data ("mini-batch") from the training data set, is selected and used to update the gradients in the direction that minimizes the training loss (where the training loss is defined as the difference between the predictions and the ground truth). One pass through the entire training data set is called a training epoch [25].

There are some considerations to take into account: the first is that there are a large number of parameters in the matrix multiplications, resulting in many computations being performed and thus the latency issues that we see on end devices. The second is that there are many choices (hyper-parameters) on how to design the DNN models (e.g., the number of parameters per layer, and the number of layers), which makes the model design more of an art than a science. Different DNN design decisions result in tradeoffs between system metrics; for example, a DNN with higher accuracy likely requires more memory to store all the model parameters and will have higher latency because of all the matrix multiplications being performed. On the other hand, a DNN model with fewer parameters will likely execute more quickly and use less computational resources and energy, but it may not have sufficient accuracy to meet the application's requirements.

2.1.3 Performance Measurement

How can we evaluate the performance of a neural network? Deep learning can be used to perform both supervised learning and unsupervised learning. The metrics of success depend on the particular application domain where deep learning is being applied. For example, in object detection, the accuracy may be measured by the mean average precision (mAP) [26], which measures how well the predicted object location overlaps with the ground-truth location, averaged across multiple categories of objects. In machine translation, the accuracy can be measured by the bilingual evaluation understudy score metric [18], which compares a candidate translation with several groundtruth reference translations. Other general system performance metrics not specific to the application include throughput, latency, and energy. These metrics are summarized in Table 2.1. Designing a good DNN model or selecting the right DNN model for a given application is challenging due to the large number of hyperparameter decisions.

Metric	Unit
Latency	s
Energy	mW, J
Concurrent Requests Served	#
Network Bandwidth	Mbps
Accuracy	Application Specific

Table 2.1: Neural Network Perfomance Metrics

Machine learning research typically focuses on accuracy metrics, and their system performance results are often reported from powerful server testbeds equipped with GPUs. For example, Huang et al. [14] compared the speed and accuracy tradeoffs when running on a high-end gaming GPU (NVIDIA Titan X). The YOLO DNN model [23], which is designed for real-time performance, provides timing measurements on the same server GPU. Specifically targeting mobile devices, Lu et al. [16] provided the measurements for a number of popular DNN models on mobile CPUs and GPUs (Nvidia TK1 and TX1). Ran et al.[22] further explored the accuracy-latency tradeoffs on mobile devices by measuring how reducing the dimensionality of the input size reduces the overall accuracy and latency. DNN models designed specifically for mobile devices, such as MobileNets [13], report system performance in terms of a number of multiply–add operations, which could be used to estimate latency characteristics and other metrics on different mobile hardware, based on the processing capabilities of the hardware. Once the system performance is understood, the application developer can choose the right model.

2.1.4 Frameworks

Several open-source software libraries are publicly available for deep learning inference and training on end devices and edge servers. Google's TensorFlow [4], released in 2015, is an interface for expressing machine learning algorithms and an implementation for executing such algorithms on heterogeneous distributed systems. Tensorflow's computation workflow is designed as a directed graph and utilizes a placement algorithm to distribute computation tasks based on the estimated or measured execution time and communication time [5]. The placement algorithm uses a greedy

approach that places a computation task on the node that is expected to complete the computation the soonest. Tensorflow can run on edge devices, such as Raspberry Pi and smartphones. TensorFlow Lite was proposed in the late 2017 [3], which is an optimized version of Tensorflow for mobile and embedded devices, with mobile GPU support added in early 2019. Tensorflow Lite only provides on-device inference abilities, not training, and achieves low latency by compressing a pre-trained DNN model. Caffe [15] is another deep learning framework, originally developed by Jia, with the current version, Caffe2, maintained by Facebook. It seeks to provide an easy and straightforward way for deep learning with a focus on mobile devices, including smartphones and Raspberry Pis. PyTorch [19] is another deep learning platform developed by Facebook, with its main goal differing from Caffe2 in which it focuses on the integration of research proto- types to production development. Actually Facebook is working on the merge of Caffe2 and PyTorch frameworks. GPUs are an important key element in efficient DNN inference and training. NVIDIA provides GPU software libraries to make use of NVIDIA GPUs, such as CUDA [7] for general GPU processing and cuDNN [8] which is targeted toward deep learning. While such libraries are useful for training DNN models on a desktop server, cuDNN and CUDA are not widely available on current mobile devices such as smartphones. To utilize smartphone GPUs, Android devel- opers can currently make use of Tensorflow Lite, which provides experimental GPU capabilities. To experiment with edge devices other than smartphones, researchers can turn to edge-specific development kits, such as the NVIDIA Jetson TX2 development kit for experimenting with edge computing, with NVIDIA-provided SDKs used to program the devices.

2.1.5 Challenges

Because of the required competences and effort to choose the best neural network and the best parameters that better fit the applications of our interest, there has also been much recent studies in automated machine learning, which uses artificial intelligence to choose which DNN model to run and tune the hyperparameters. For example, Tan et al. [28] and Taylor et al. [29] proposed using reinforcement learning and traditional machine learning, respectively, to choose the right hyperparameters for mobile devices, which is useful in edge scenarios. As described in [6] many challenges remain in

deploying deep learning on the edge, not only on end devices but also on the edge servers and on a combination of end devices, edge servers, and the cloud. For example parameters like latency, energy consumption and migration still the main challenges in the field of deep learning applied to the edge computing.

2.2 Robotics Tools and Platforms

This section will cite and recap the main robotics platforms discovered and explored during the development of this thesis.

2.2.1 ROS

The Robot Operating System (ROS) [21] is a open-source framework based on the component-based software engineering paradigm that provides the middleware for inter-process communication. Initially developed by the Stanford Artificial Intelligence Laboratory its development continued at Willow Garage, a robotics research institute, and now it's maintained and improved under the action of the ORSF foundation [21]. As a meta-operating system, ROS offers features such as hardware abstraction, low-level device control, implementation of commonly used functionalities, message communication between processes and package management. It uses an asynchronous publish/subscribing mechanism made possible by message standardisation and encapsulation that make the external interface of every node as general as possible, allowing quick nodes exchange and, thus, great architectural flexibility. Each independent block, called *node*, executes a particular task of a process and can communicates with other nodes through *topics*. This allows to create complex architectures by aggregating many simpler entities and simplifies the use of different tasks or different methods for the same task. Additionally to the message-passing system, the core ROS component, called *roscore*, maintains a global execution time for the nodes to achieve synchronisation. Each node executes separately with its own internal clock driven by the set execution rate. At every message sent/received, containing the internal time information, the core component updates the global time by following the execution status provided by these messages. For more clarity about how ROS communication works an example is provided in figure 2.2. When a publishing node will announce to the

master that it is publishing over a topic and the subscribing node will say to the master that it want to listen to a topic. Then if the publisher and the subscriber are on the same topic, the master will transfer data in order for the two other node to communicate directly via TCP/IP or UDP.

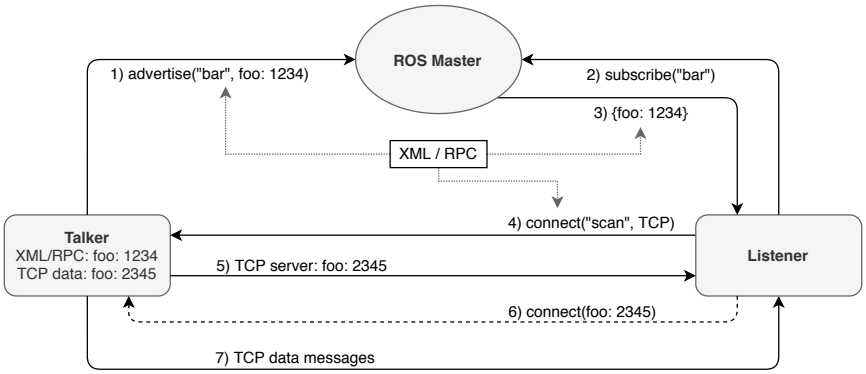

Figure 2.2: Example of nodes and topic communication

The reason to choose ROS for the development of robotics applications resides mainly in its high modularity. Furthermore it is a well known standard in the scientific community of the robotics researches and developers.

2.2.2 NVIDIA Isaac SDK

The Isaac SDK is the main software toolkit provided by NVIDIA and allows developers to brought new opportunities for developing robotics solutions as well as researching topics in this field. As described in [9] it's comprised of the following:

- Isaac Robot Engine: A framework which allows you to easily write modular applications and deploy them on your robots.
- Isaac GEMs: A collection of robotics algorithms from planning to perception, most of them GPU-accelerated.
- Applications: Various example applications from basic samples which show specific features to applications that facilitate complicated robotics use cases.
- Isaac Sim for Navigation: A powerful virtual robotics laboratory and a high-fidelity 3D world simulator that accelerates research, design, and development by reducing cost and risk.

The main reasons we would consider to use this platform reside in its high modularity and high performance solutions that it is able to offer on NVIDIA Jetson platforms. Others important reasons to adopt this solution are related the **C API** that allows the communication with Isaac apps from languages other than C++, and the **ROS bridge** that offers the capability to interact with ROS nodes applications.

2.2.3 CoppeliaSim

From the creators of V-Rep [10], CoppeliaSim [24] seems to be the definitive solution for development of robotics applications. The robot simulator CoppeliaSim, with integrated development environment, is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, ROS nodes, BlueZero nodes, remote API clients, or a custom solution. This makes CoppeliaSim very versatile and ideal for multi-robot applications. Controllers can be written in C/C++, Python, Java, Lua, Matlab, Octave or Urbi. CoppeliaSim can be used as a stand-alone application or can easily be embedded into a main client application: its small footprint and elaborate API makes CoppeliaSim an ideal candidate to embed into higher-level applications. An integrated Lua script interpreter makes CoppeliaSim an extremely versatile application, leaving the freedom to the user to combine the low/high-level functionalities to obtain new high-level functionalities.

2.3 Edge Computing

2.4 Docker

2.5 Kubernetes

2.6 Kubeedge

Chapter 3

Verification

This chapter will focus on description and verification of ORB-SLAM2 [17] improved solution proposed in IROS 2019 conference [27]. Primary it will describe how ORB-SLAM2 works. Secondly it will show an accurate inspection necessary to find some potential bugs that cause an unexpected behaviour of the algorithm in a sequential environment.

3.1 System description

As described in [27], ORB-SLAM2 is a Simultaneous Localization And Mapping system that can works with data coming from monocular, stereo, and RGB-D cameras. A system of this kind has the purpose to allow both map reconstruction and navigation in the most common environment without the support of a GPS. The system consists of the following three main blocks (see figure 3.1):

Tracking and localization This block is in charge of computing visual features, localizing the robot in the environment, and, in case of significant discrepancies between an already saved map and the input stream, communicating updated map information to the mapping block. The frames per second (FPS) that can be computed by the whole system strongly depends on the performance of this block.

Mapping It updates the environment map by using the information (map changes) sent by the localization block. It is a computational time consuming block and its execution rate strictly depends on the agent speed. However, considering the actual agent speed of the KITTI

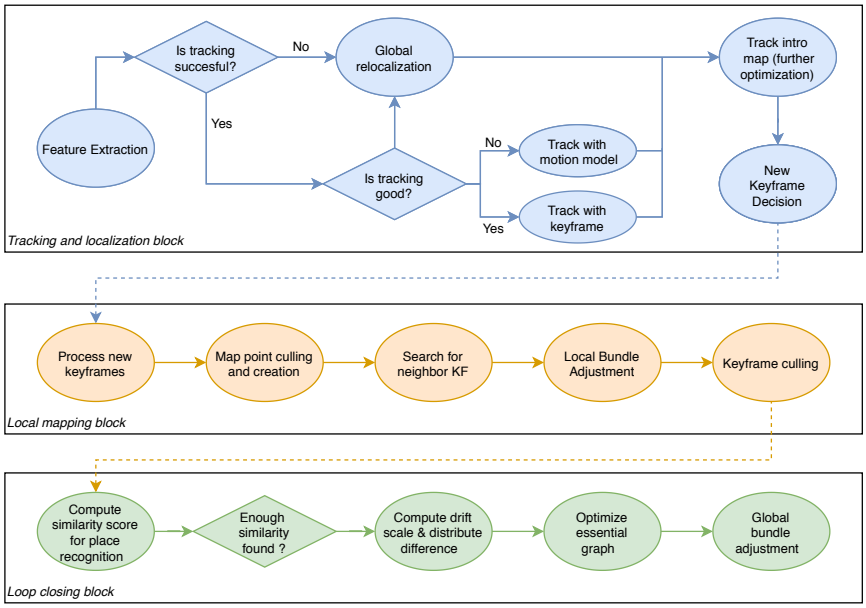

Figure 3.1: Main blocks of the ORB-SLAM2 algorithm.

datasets analysed in this work [11], it does not represent a system bottleneck.

Loop closing It aims at adjusting the scale drift error accumulated during the input analysis. When a loop in the robot pathway is detected, this block updates the mapped information through a high latency heavy computation, during which the first two blocks are suspended. This can lead the robot to loose tracking and localization information and, as a consequence, the robot to get temporary lost. The computation efficiency of this block (running on-demand) is crucial for the quality of the final results.

The system is organized on three parallel threads, one per block. The use of parallel threads allows for obtaining real-time processing on an Intel Core i7 desktop PC with 16GB RAM [17]. The work done in [27] ...

3.2 Unexpected behaviour

After the porting and the optimizations of the ORB-SLAM2 algorithm on the NVIDIA Jetson TX2 to achieve real-time performance, the experimental results conducted in [27] shown a massive improvement of the execu-

Figure 3.2: An illustration of Track Lost soon after initialization (left), and Track Lost state (right)

tion times on some KITTI dataset sequences. Unfortunately, despite these performance improvements, the algorithm seems to exhibit some unexpected behaviours. More specifically, it get into the Tack Lost state soon after initialization, and another Track Lost state after crossing the threshold of 5 elaborated Keyframes. (see fig: **??**)

Trying to solve this problem a bottom-up workflow has been followed,

3.2.1 The problem of the non-determinism

3.2.2 The problem of the concurrency

3.3 Discussion

In this chapter the problem of non deterministic behaviour has been addressed and some of the possible solutions to overcome the problem of ... have been presented.

The strategy adopted to achieve the goal of ... could reduce the ... in the system.

Chapter 4

Methodology

This chapter will present a multi-level model design flow for the integration of heterogeneous applications and their deployment to the edge on embedded low power devices like NVIDIA Jetson TX2.

4.1 A ROS based robotic system

The choice of the most suitable integration platform is one of the main problems to be faced when integrating heterogeneous applications such as robotic software systems. This choice will have implications on performance and communication of the whole system. Furthermore it will be a key factor during the design phase of the entire architecture. After an accurate comparison among all available robotics platforms, for the purpose of our specific application we decided to use ROS [21] in favour of its standard on communication among other applications.

4.1.1 Architecture at L1

At this architecture level we put the whole system on a single Intel x86 powered machine that we will conventionally call *Host*. This is the simplest and the most common architecture used by robotics researches. As we can see in figure 4.1, in this case we have two main ROS nodes that communicate between them. Specifically speaking, in this case we have both ORB-SLAM2 and VOVD [20] running as ROS nodes to perform a mobile robot navigation task.

Because of the most used robotics platforms used today support the

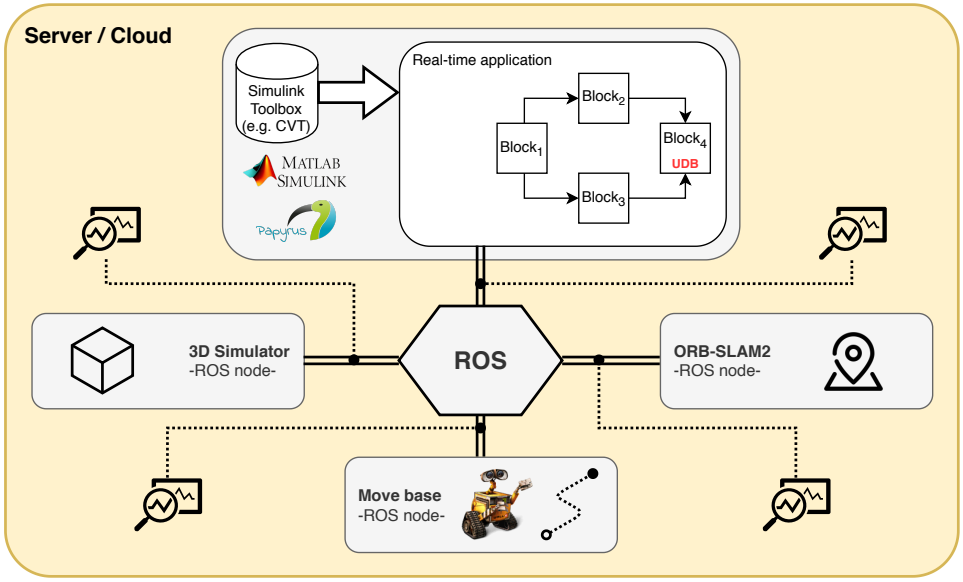

Figure 4.1: L1 architecture

Melodic Morenia [2] version of ROS, and because of VOVD was written with the Kinetic version, it has been decided to convert VOVD in the relative Melodic ROS version. This task required a redefinition of some functions related to the ROS *tf2* package [30] due to the deprecation of the *tf* package functions previously used. Another modification done talking about VOVD is related to the 3D map used by Gazebo [1]. The problem was that it came with a featureless map generated from an URDF file, and because of ORB-SLAM2 works only if some features are detected during its execution, it has been necessary to add a material to the 3D model of the map. Talking about ORB-SLAM2, it comes already provided with the necessaries functions to run in a ROS environment, so no others operations have been required to complete our first level architecture.

4.1.2 Architecture at L2

The next step requires the split of our system in two parts: Host (simulator) and Device/s (Edge Computing node/s).

4.1.3 Architecture at L3

Host - Device/s - Robot ...

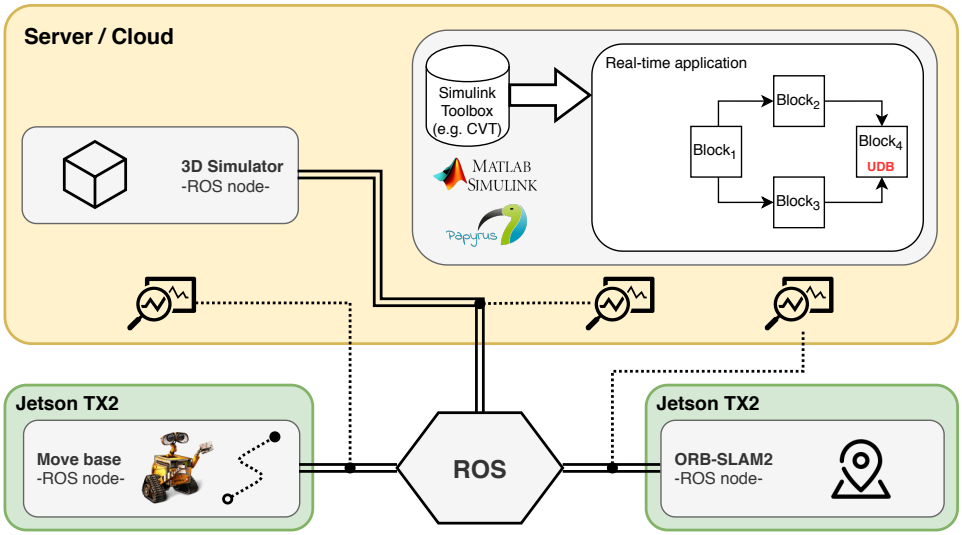

Figure 4.2: L2 architecture

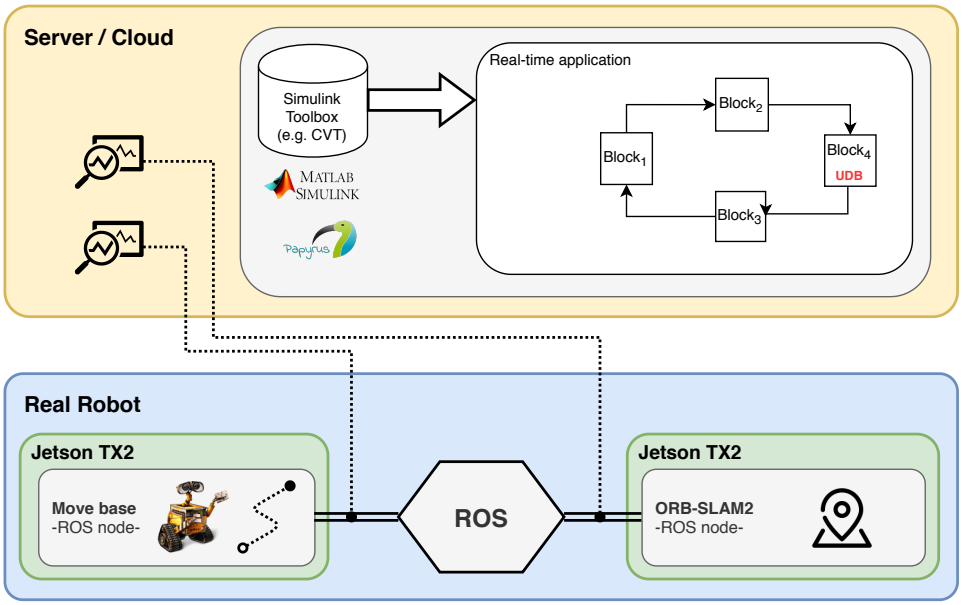

Figure 4.3: L3 architecture

4.2 Deployment from the cloud to the edge

As mentioned in 2.6, Kubeedge is built upon Kubernetes and provides fundamental infrastructure support for network, app. deployment and metadata synchronization between cloud and edge. As we know, Kubernetes uses containers to run isolated, packaged applications across its cluster nodes. To run on Kubernetes, your applications must be encapsulated in one or more container images and executed using a container runtime like Docker. While containerizing your components is a requirement for Kubernetes, it also allow easy scaling and management. For instance, containers provide isolation between the application environment and the external host system, support a networked, service-oriented approach to inter-application communication, and typically take configuration through environmental variables and expose logs written to standard error and standard out. Containers themselves encourage process-based concurrency and help maintain dev/prod parity by being independently scalable and bundling the process's runtime environment. These characteristics make it possible to package your applications so that they run smoothly on Kubernetes. In this section it will described the process used in our project to prepare both VOVD and ORB-SLAM2 applications for the edge computing through the support of Kubeedge.

4.2.1 VOVD at the edge

4.2.2 ORB-SLAM2 at the edge

4.3 The whole system on Kubeedge

4.4 Discussion

Chapter 5

Experimental Results

In this chapter we present the experimental setup on which the proposed architecture has been developed, implemented and tested.

5.1 Setup

The main technicals specifications of the Host are shown in table 5.1.

Hardware	Model
Processor	Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz
Operating System	18.04.1-Ubuntu
Storage	SATA 500GiB
Memory	16GiB
GPU	NVIDIA GeForce GTX 980
Software	Version
OpenCV	3.4.7
CUDA	10.0
ROS	Melodic Moreina

Table 5.1: Technical specifications of the Host

5.2 Mobile robots

5.2.1 Hybrid local/global planner

5.3 Results

Setup	Odom VOVD	Track Pipe	Decompression	Frame Elaboration	Feature Extraction	Publish Pose	FPS Track	FPS Feature Extraction	FPS Supported
orb(j1)	-	43,41	13,08	49,21	43,62	0,02	23,04	20,32	20,32
vovd(j1)	1,70	-	-	-	-	-	-	-	-
(orb;vovd)(j1)	2,58	73,49	18,69	62,70	54,76	0,17	13,61	15,95	13,61
(orb(j1);vovd(j2))	1,60	49,81	13,53	49,25	43,01	0,08	20,08	20,30	20,08

Table 5.2: Performance Measurement: camera 10 Hz

Setup	Odom VOVD	Track Pipe	Decompression	Frame Elaboration	Feature Extraction	Publish Pose	FPS Track	FPS Feature Extraction	FPS Supported
orb(j1)	-	54,82	14,47	49,40	44,35	0,02	18,24	20,24	18,24
vovd(j1)	1,70	-	-	-	-	-	-	-	-
(orb;vovd)(j1)	2,85	93,15	23,96	67,99	60,41	0,21	10,74	14,71	10,74
(orb(j1);vovd(j2))	1,60	56,20	14,00	49,32	43,95	0,09	17,79	20,27	17,79

Table 5.3: Performance Measurement: camera 20 Hz

Setup	Odom VOVD	Track Pipe	Decompression	Frame Elaboration	Feature Extraction	Publish Pose	FPS Track	FPS Feature Extraction	FPS Supported
orb(j1)	-	47,28	12,78	40,52	36,02	0,01	21,15	24,68	21,15
vovd(j1)	1,30	-	-	-	-	-	-	-	-
(orb;vovd)(j1)	1,40	82,47	15,10	53,54	48,09	0,23	12,12	18,68	12,12
(orb(j1);vovd(j2))	1,38	50,83	13,98	43,80	38,04	0,33	19,67	22,83	19,67

Table 5.4: kubeedge - Performance Measurement: camera 10 Hz

Setup	Odom VOVD	Track Pipe	Decompression	Frame Elaboration	Feature Extraction	Publish Pose	FPS Track	FPS Feature Extraction	FPS Supported
orb(j1)	-	50,20	16,20	42,16	34,82	0,10	19,92	23,72	19,92
vovd(j1)	1,21	-	-	-	-	-	-	-	-
(orb;vovd)(j1)	1,43	70,54	15,02	55,60	50,24	0,22	14,18	17,98	14,18
(orb(j1);vovd(j2))	1,26	58,14	13,52	42,34	37,17	0,32	17,20	23,62	17,20

Table 5.5: Kubeedge - Performance Measurement: camera 20 Hz

Conclusions And Future Works

In this thesis a .... has been presented to address the issue of providing ...

Bibliography

[1] Gazebo robot simulation. `http://gazebosim.org/`.

[2] Ros melodic morenia. `http://wiki.ros.org/melodic`.

[3] Tensorflow lite. `https://www.tensorflow.org/lite`.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.

[6] J. Chen and X. Ran. "deep learning with edge computing: A review". *in Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[7] NVIDIA Corporation. Cuda zone. `https://developer.nvidia.com/cuda-zone`.

[8] NVIDIA Corporation. cuDNN deep neural network library. `https://developer.nvidia.com/cudnn`.

[9] NVIDIA Corporation. Isaac sdk. `https://developer.nvidia.com/isaac-sd`.

[10] Marc Freese, Surya Singh, Fumio Ozaki, and Nobuto Matsuhira. Virtual robot experimentation platform v-rep: A versatile 3d robot simulator. In *Proceedings of the Second International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, SIMPAR'10, page 51–62, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] Andreas Geiger. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, page 3354–3361, USA, 2012. IEEE Computer Society.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[14] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors, 2016.

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*,

MM '14, page 675–678, New York, NY, USA, 2014. Association for Computing Machinery.

[16] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 1663–1671, New York, NY, USA, 2017. Association for Computing Machinery.

[17] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[18] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach et al., editor, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[20] Nicola Piccinelli and Riccardo Muradore. Hybrid motion planner integrating global voronoi diagrams and local velocity obstacle method. pages 26–31, 06 2018.

[21] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009.

[22] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. pages 1421–1429, 04 2018.

[23] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[24] "E. Rohmer, S. P. N. Singh, and M. Freese". "coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework". In *"Proc. of The International Conference on Intelligent Robots and Systems (IROS)"*, "2013".

[25] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[27] D. D. Bloisi S. Aldegheri, N. Bombieri and A. Farinelli. Data flow orb-slam for real-time performance on embedded gpu boards. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5370–5375, 2019.

[28] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile, 2018.

[29] Ben Taylor, Vicent Sanz Marco, Willy Wolff, Yehia Elkhatib, and Zheng Wang. Adaptive selection of deep learning models on embedded systems, 2018.

[30] Wim Meeussen Tully Foote, Eitan Marder-Eppstein. tf ros package. `http://wiki.ros.org/tf`.