DALHOUSIE UNIVERSITY

ENSTA Bretagne

ENSTA Bretagne
2, rue François Verny
29806 BREST cedex
FRANCE
Tel +33 (0)2 98 34 88 00
www.ensta-bretagne.fr

report

CI2020
second year

July 29, 2019

2019 Summer Internship Dalhousie University, Halifax, Nova Scotia, Canada

Robust Deep Learning Tracking Robot

written by ARNAUD KLIPFEL, Robotics specialization at ENSTA-BRETAGNE
supervised by Dr. THOMAS TRAPPENBERG, DALHOUSIE UNIVERSITY

Contents

Ackowledgement

Resume

Introduction

This document is the report of the project on which I worked during my summer internship in 2019 at Dalhousie University. The main goal was to build a tracking robot with the middleware ROS and with deep learning techniques applied to tracking.

The first part presents how to setup the robot.

Keywords

stereo-vision, tracking, *ROS*, mechatronics, mobile robotics, RC car, ground robot, deep learning.

Part 1

Setup

THIS part presents the material needed and the necessary conditions to replicate the results of this project. Should you need some explanations on the project, please refer to parts 2 on the facing page, 3 on page 9, 4 on page 11, and 5 on page 13.

Part 2

Project Contextualization

2.1 Project Specification

2.1.1 Motivations

The main goal of this project was to build a *Deep Learning Tracking Robot*. Many tracking robots have been implemented in the previous years, few have used the adaptive, versatile, and now standard robotic Middleware *ROS*[1]. In addition, even fewer have ventured combining *ROS* and *Deep Learning* techniques.

Why *ROS* precisely? As defined by Anis Kouba in [Kou16], *ROS* is an « open-source *middleware* » that provides a robust and reliable framework to build robots. *ROS* is a voguish tool to create robots with advanced functionalities so much so that it has become the standard to develop robots. *ROS2* is already in part operational. In this project though, it has been decided that the first version of *ROS* will be used since *ROS2* is still in heavy development and *ROS* is better documented than its counterpart.

Why is *Deep Learning* suitable for robotic applications? In general, *AI* [2] is really adapted for critical algorithms, such as embedded codes on robots, since the performances are much higher than with traditional implementations. By traditional implementations, one could for instance consider iterative algorithms where for a particular task all processes are hand-implemented. The conventional solution that comes to mind regarding tracking is an *RGB* tracking, that is to say an algorithm that tracks the color in a frame[3] by applying color filters. *AI* enables much finer and more robust implementations since the *model*, which can be regarded as the applied processes, can evolve by itself through diverse methods such as training.

2.1.2 Requirements

In order to realize the tracking robot, several requirements had to be met in this project. First of all, the robot had to reuse an old *Race Car* platform comprising the chassis, the wheels . . . all the mechanics. The platform is the *H1 model* of the *monster Car* [Car]. The brain of the robot, that is to say the embedded computer had to be the *Jetson Nano* designed by *Nvidia* [MSV19]. By and large, to build the tracker the material of the laboratory had to be used as much as possible, and the equipment bought had to remain affordable.

2.2 Strategy

2.2.1 Different periods

Developing a robot is a critical task, and has to be conducted thoroughly. In this regard, *ROS* builds a framework for developing robots in a more organized way. For each task to implement, the work-flow has always been the following: research, simulation, isolated tests, integration.

[1] Which stands for *Robot Operating System*.

[2] Artificial Intelligence, which comprises *Deep Learning*.

[3] image.

Before even implementing something and integrating it, one should research all the possibilities that are offered to them. Then one of the easier or maybe most sustainable options could be chosen. Yet, even the tests, simulation, and integration has to be thought and foreseen beforehand.

Simulating robots, without depending on the hardware is a needed step. A specific hardware has always some specificities that could hide some issues and unpredictable behaviors. As a matter of fact, simulating algorithms that will be integrated in the robot afterward in a controlled environment prevents any hardware based issue to shadow our understanding of the basic implementation of the *software*.

Once the simulation is in place, it is then recommended to test the simulation as much as possible in order to unveil some detrimental singularities. In the case of the hardware, each component has to be tested independently before integration [4].

The final step is to integrate the work in the robot, and to test it to see if the robot behaves as expected or if there is any regression.

A robot is not just a piece of software, it is a system where the interaction of the software and the hardware is by definition the future behavior of the robot.

2.2.2 Different aspects

The development of a robot encapsulates different areas or type of work. In the construction of the tracker, principally two aspects have been tackled, the hardware, and the software.

In the first place, the hardware has to be well-chosen beforehand. The motor had to replaced, the *wifi* access had to be added, batteries had to be chosen . . . to fit the requirements of the tracking robot.

In the second place, the sofware has to be implemented. Precisely, the tracking algorithm had to be chosen, and the *ROS* architecture had to be designed.

For the entire project, the philosophy was to try to have a working first prototype as soon as possible. In a nutshell, having a hardware and software which are compatible, and then refining the existing platform. Indeed, *ROS* provides here another crucial boon, for it enables us to have an evolutive architecture where each part can be replaced easily without undermining the overhaul process. For instance, once an architecture is working, the tracking technique could be easily replaced.

Throughout this report, which presents the results of the project, three questions will be answered:

- How was the hardware selected? You can find the answers to this question in the part 4 on page 11.
- How was the tracking algorithm implemented? Which belongs more to the software. You can find the answers to this question in the section 5.1 on page 13.
- How was the *ROS* architecture designed? Which belongs more to the software. You can find the answers to this question in the section 5.2 on page 13.

[4]Especially, some driver issues are sometimes really hard to pinpoint.

Part 3

State of the art

In order to build the tracker, the first step was to choose the hardware of the robot. Secondly, the *ROS* architecture had to be decided and thirdly the tracking algorithm had to be implemented. This part presents a brief overview of different existing projects and solutions that could have been selected throughout the project. All the projects presented and listed here will then be compared and the choices made will be underpinned in the parts 4 on page 11 and 5 on page 13.

3.1 Building an electric car

Several projects have aimed to realize an electric car robot, sometimes autonomous. Having in mind these projects can give some piece of advice on how it is common to design such systems.

Some projects have used the on-board computer *Jetson Nano*. One can name the following robots : *Jetbot* [nvia], *Kaya* [nvib]. These robots are both designed by *Nvidia*, the designers of the *Jetson* board series[1]. It is notably interesting to take as example their hardware selection since they use the same embedded computer the tracker use.

Other *RC-cars*[2] use other types of embedded computers. However, the hardware architecture with regard to *RC-cars* is almost always the same. With either the *DeepRacer* of *Amazon* [Ama], or the *Pi-Car raspberry* powered car [Sun], or *Ghost* car [Dan], or the *Roscar* [nai], or the *F1tenth* car [F1T], or the *Donkey* car [Diy], the architecture follows some fundamental principles.

The selection of the hardware is discussed further in the part 4 on page 11.

3.2 Tracking

Multiple ways of tracking a target exist, in this project the most common one was used : a *visual tracking*. Visual tracking is basically based on the processing of images or frames taken by a camera.

More precisely, the mission of the tracker implemented in this project is to follow a specific object in a frame [3], to be able to learn the object in that process, and even to be able to recover the target after any occlusion. Looking at the deep learning models and algorithms which have been developed in the recent years, some could be more labeled as detection algorithms, do not really learn the specificities of the target, and even track multiple targets. For instance, the network *Yolo* [Bje16] is rather designed for multi-tracking and not to follow a specific target.

Considering deep learning models that are able to track a specific target the database *GOT-10k* has produced an enumeration and ranking of the current most effective techniques [GOTB19].

For sure, in order to automate the robot, a detection algorithm could precede the tracking.

[1] Jetson Nano, Xavier, TX1, TX2 ...

[2] *Remote Control Car.*

[3] Given by a camera for instance.

3.3 Using ROS

Developing with *ROS* gives also an incredible advantage to any roboticist. In fact, some *ROS packages* which are totally open-source already provide cutting-edge algorithms such as sensor fusion by Kalman filtering, *SLAM* [4]. Unfortunately, applied to specific object tracking few *ROS* packages were developed, and the existing ones do not use deep learning techniques at all.

Nevertheless, some *ROS interfaces* were implemented for *ROS*, which can allow one to use a tracking algorithm compatible with the *ROS interface*, also often named *bridge* [AS18]. The main problem is mainly that the implemented programs are not up-to-date which renders the integration in *ROS* sometimes almost impossible due to software dependencies.

Apart from that, several projects, almost all shared on *github*, implement tracking solutions. However, they either do not use deep learning, or are not supported by *ROS* anymore. The book *ROS Robotics Projects* presents a well-documented tracking project, though no deep learning techniques are used [Jos17]. Yet, the *ROS* architecture inspired some of the realization of the own architecture of this project.

[4]Simultaneous Localization And Mapping.

Part 4

Hardware

FIRST of all, before even thinking about the tracking process, and the behavior of the robot, the existing platform had to be fixed, and if needed adapted to the current needs of the project. This part starts by presenting the latest architecture of the tracking robot, an then breaks this architecture apart by giving an overview of each challenge which had to be tackled.

4.1 Hardware architecture

It is crucial to chose wisely the hardware of the robot in the first place, since the software could be seen then as an exploitation of the hardware of the robot. The initial hardware of the robot did comprise a servomotor, a *brushed DC motor*[1], and a motor controller, or *ESC*, which stands for *Electronic Speed Controller*.

This initial architecture had to be adapted for several reasons. The latest hardware architecture of the tracking robot is presented on figure ??, which shows each component and how each interacts with each other.

Let's explain a bit what are the crucial parts that constitute a tracking robot. The target tracking process used in this project is based on image and depth inputs, but to be more precise, on a *stereo camera*. A *stereo camera* provides basically two types of information : depth and color. For this purpose the *ZED camera* of *Stereolabs* was used [ste]. The processing of the frames given by the camera is done on the embedded computer *Jetson Nano*. Regarding the actuators, a servomotor controls the bearing of the car, and a motor controls the speed of the car through an *ESC*. In order to communicate with the motor controller, or *ESC* a *PWM shield* was needed. Basically, a *PWM* signal, which stands for *Pulse Width Modulation* in this case, is a certain type of signal which are characterized by their duty cycle. This fundamental parameter is equivalent to the command sent to the *ESC*. At this end, the *PWM shield* PCA9685 of Adafruit was used [Ada]. Ultimately, since in the field of mobile robotics, robots tend to be mobile by definition, the tracking robot had to be powered on its own. To meet that constraint two batteries were added.

4.2 Low speed control

4.2.1 Motor

After having fixed the existing *RC car*, the speed was unfortunately too high to meet the likely processing speed of the embedded computer. The next step was then to find a way to slow down the robot.

The first constraint was to avoid changing the mechanics as much as possible. Changing the gear box, without buying a new *ESC* and a new motor could have been possible. Yet, it would have required to alter the entire mechanics of the *RC car* platform. Thus, this option was not

[1]More is explained on that in the section 4.2.

considered.

How is it possible then not to modify the mechanics and at the same time to decrease the speed? If the mechanics was to be kept identical, that is to say that the power chain should remain untouched, the motor has to remain of the same dimensions. This is why as new motor, a *brushed DC motor* was in part chosen. Economically talking, a *brushed* motor is far cheaper in comparison to its counterpart the *brushless* motor. Yet, the speed has still not been decreased.

To understand how it is actually possible to slow down a *DC motor* it is essential to fathom the existing relation between the torque applied to the charge and the *RPM*[2], or in other words the angular speed of the shaft of the motor. Basically, the angular speed is inversely proportional to the torque. Thus, in order to decrease the angular speed, a *brushed DC motor* with more torque had to be chosen. [McC18; Sca15]

4.2.2 Motor Controller

A *DC motor* without a controller is not very useful. At this stage, a controller had to be selected in order to set by commands the angular speed of the motor. Numerous strategies exist to regulate the speed of a *DC motor*. For instance, it is possible to use specific motor controllers [F1T], transistors, or *ESCs*. From all the projects introduced in the state or the art part 3.1 on page 9, a majority uses *ESCs*.

Yet, the race car of the *MIT*[3] employs *VESC*, which stands for *Vedder Electronic Speed Controller* [MIT]. Such *ESCs* were designed to provide more flexibility in the control of the low speeds. However, their price were far too high to be considered in this project.

All in all, a new brushed motor *ESC* was bought.

4.3 Power consumption

4.3.1 How many?

The first question that should came to mind is not which type of battery should be bought, or even how much power the hardware requires, but how many batteries the robot needs. It is common in robotic applications and more extensively in any system to split the power supply into two parts.[Jet]

Actuators, that is to say for instance the motor or the servomotor, tend to require far more power than the embedded computer. Besides, should the actuators draw too much current[4], the embedded computer will switch off. Thus, as a matter of safety, it is crucial to isolate the power source of the actuators from the one of the embedded computer. This explains the choice of two separate batteries.

4.3.2 Which one?

4.4 Remote control and processing

4.5 Performances of the embedded computer

4.6 Unit Tests

[2]Revolutions Per Minutes.

[3]Massachusetts Institute of Technology, USA.

[4]It often happens that the actuators may induce current peaks.

Part 5

Software

5.1 Tracking

5.2 ROS

Conclusion

Bibliography

[Ada] Adafruit. *Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685.* https://www.adafruit.com/product/815 (cit. on p. 11).

[Ama] Amazon. *Amazon Deep Racer.* https://aws.amazon.com/deepracer/ (cit. on p. 9).

[AS18] Martin Jagersanda Abhineet Singha. "Modular Tracking Framework: A unified approach to registration based tracking". In: *Computer Vision and Image Understanding, Elsevier* (2018) (cit. on p. 10).

[Bje16] Marko Bjelonic. *YOLO ROS: Real-Time Object Detection for ROS.* https://github.com/leggedrobotics/darknet_ros. 2016–2018 (cit. on p. 9).

[Car] Monster Cars. *Instructions manual.* link. H1 model. (cit. on p. 7).

[Dan] Steven Daniluk. *Ghost II : controlling an RC car with a computer.* https://medium.com/hackernoon/ghost-ii-controlling-an-rc-car-with-a-computer-b1d1849d9e43 (cit. on p. 9).

[Diy] DiyRoboCars. *Donkey Car.* https://docs.donkeycar.com/ (cit. on p. 9).

[F1T] University Of Pennsylvania F1Tenth. *F1Tenth race car.* http://f1tenth.org/build.html (cit. on pp. 9, 12).

[GOTB19] GOT 10k : Generic Object Tracking Benchmark. *Leaderboard : Up-to-date generic object tracking performance.* http://got-10k.aitestunion.com/leaderboard. 2019 (cit. on p. 9).

[Jet] JetsonHacks. *RACECAR/J Initial Assembly.* https://www.jetsonhacks.com/2018/01/28/racecar-j-initial-assembly/ (cit. on p. 12).

[Jos17] Lentin Joseph. *ROS Robotics Projects.* Chapter 2. Packt, 2017 (cit. on p. 10).

[Kou16] Anis Koubaa. *Robot Operating System : the complete reference (volume 1 and 2).* Springer, 2016 (cit. on p. 7).

[McC18] Gordon McComb. *Robot Builder's Bonanza, 5th Edition.* McGraw-Hill Education, 2018 (cit. on p. 12).

[MIT] MIT. *MIT racecar.* github project & official documentation (cit. on p. 12).

[MSV19] Janakiram MSV. *NVIDIA Brings Affordable GPU to the Edge with Jetson Nano.* https://thenewstack.io/nvidia-brings-affordable-gpu-to-the-edge-with-jetson-nano/. 2019 (cit. on p. 7).

[nai] Github repository by the user naisy. *ROSCAR.* https://github.com/naisy/roscar (cit. on p. 9).

[nvia] nvidia. *Jetbot : an educational robot.* https://github.com/NVIDIA-AI-IOT/jetbot/wiki (cit. on p. 9).

[nvib] nvidia. *Kaya.* https://docs.nvidia.com/isaac/isaac/apps/tutorials/doc/assemble_kaya.html (cit. on p. 9).

[Sca15] Matthew Scarpino. *Motors for Makers: A Guide to Steppers, Servos, and Other Electrical Machines.* Que Pub, 2015 (cit. on p. 12).

[ste] stereloabs. *ZED camera documentation.* https://www.stereolabs.com/docs/getting-started/ (cit. on p. 11).

[Sun] SunFounder. *Raspberry Pi Smart Robot.* https://www.sunfounder.com/picar-s-kit.html (cit. on p. 9).