ROS2 Developers' GUIDE

by **Ricardo Téllez**, *ROS teacher at Master Industrial Robotics 4.0, Technical University of Catalonia, U.P.C.*

Your starting point to create the robots of the future

The Construct

Written by Ricardo Tellez, ROS teacher at Master of Industry 4.0, Technical University of Catalonia, Spain
Cover design by Lorena Guevara
Infographics by Yuhong Lin and Lorena Guevara

INTRODUCTION

If you are reading this is because you are tired of not having the robots promised on the movies. Where are our robot companions? Where are the robots that do all the hard work for us? Where are the robots that take the difficult and dangerous tasks for us?

Nowhere!

We are tired of those fake robots that fool journalists and the general public, looking like they understand everything, when the reality is that they understand nothing at all. Those are just bullshit robots!

What is happening here? Why don't we have useful autonomous robots yet?

The bottleneck that is preventing us from having clever robots is in the robot software. Robot hardware is already good enough to perform many of the tasks we would like a robot to do for us, like cleaning a room or preparing the meals, build a car from start to end, harvest the fields, or explore the spaces of Mars. Robotics hardware is already ready to do all those things.

The problem is the software: there is no software for robots that allows them to do any of those tasks autonomously. What can we do?!?!

WE NEED YOU!

And here is where you get into the picture. There are not enough good robotics programmers (also called **robotics developers**).

The robots of the future are waiting for good robotics developers, they are waiting for you!

We need audacious software engineers that master programming robots with novel and intelligent algorithms. Beyond a conceptual or theoretical algorithm, we need developers that bring those ideas into reality. We need modern developers that master programming at the highest level… are you in?

If you are, keep reading this guide. Here we are going to give you the steps you need to become…

…the first robotics developer that will build a conscious robot!

WHY NOT ENOUGH GOOD DEVELOPERS FOR ROBOTICS?

In general, software developers do not like to deal with the hardware.

It is very likely that you are a good software developer and never thought about entering in the robotics field. You probably think that by programming for robots you would need to know about electronics and may be even mechanics. You probably think that hardware and software are too much coupled in robots that you cannot touch one thing without touching the other. And sometimes, it is…

Some years ago, we had to develop the navigation system for a robot. However, our navigation program was not working at all. We thought that it was something wrong with the program, but after extensive review and test in simulations, we found that it actually was a problem with the electronics of the laser scanner that we used to localize the robot. There were some micro-interruptions in the voltage level that made the laser reboot. In order to find that error, we had to go to the basics and find where the problem within the physical laser was. For that we needed to mess with the electronics. We had to take the laser out from the robot, put it on the table and start doing experiments. Different voltages, different interruptions to the power, in order to try to reproduce the effect in a controlled environment. That is a lot of interaction with the hardware.

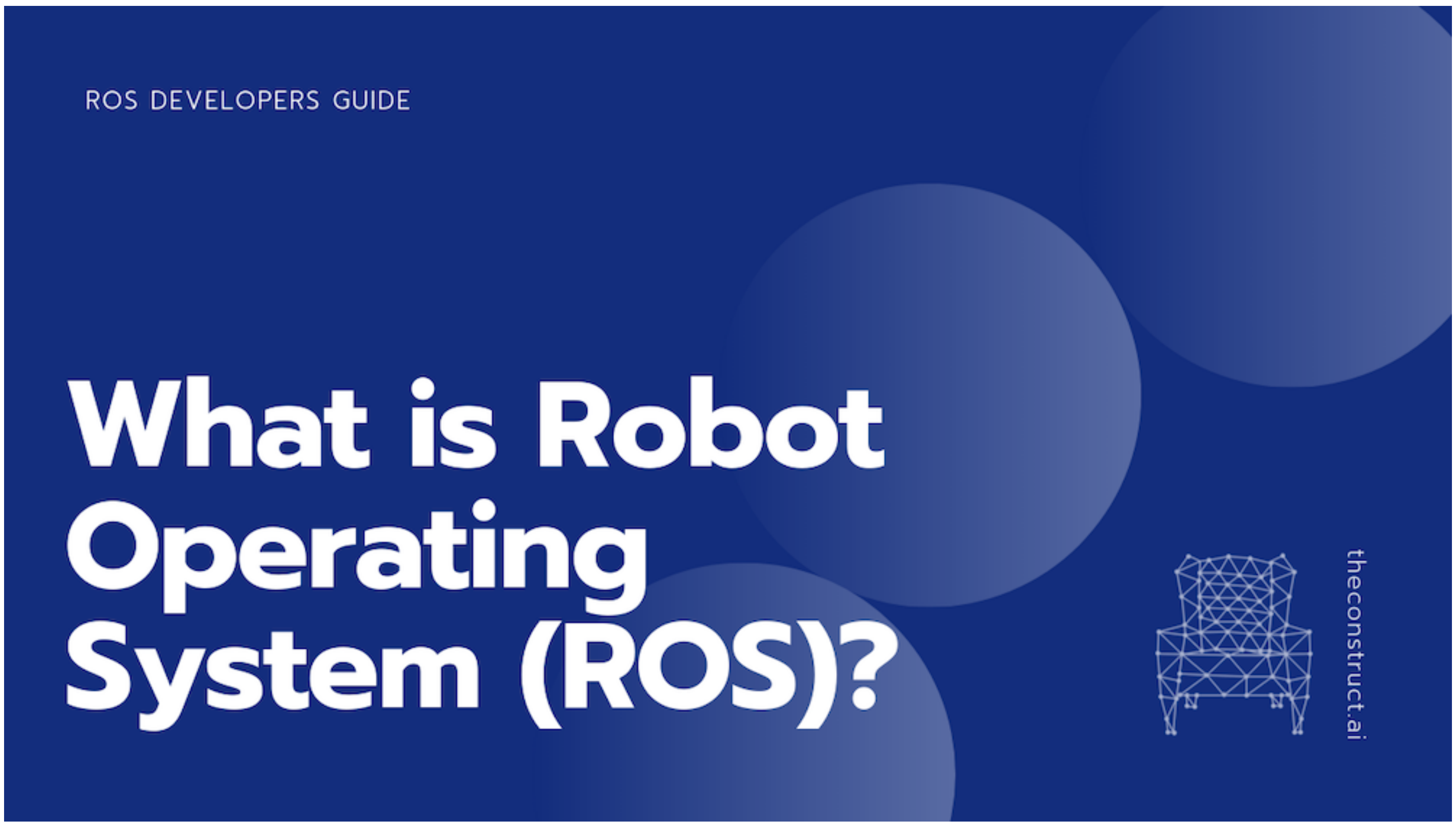
That interaction with the hardware is something that many software developers don't like. After all, they decided to become software developers, not hardware ones!!

So, if we want good software developers to program robots, we need to provide two things:

1. A framework to abstract the hardware of the robot to the software developer
2. This framework must be quite common among robots. Otherwise we would need to learn a different framework for each type of robot.

Fortunately, we already have this framework, and it is called ROS.

Thanks to the Robot Operating System, ROS, you can abstract completely the hardware from the software in most of the cases, so you can program a robot just by knowing the robot ROS API and having a simulation of the robot for testing purposes. By using the ROS API, you can forget about the hardware and just concentrate on the software that makes the robot do what you want.



Also, the framework is so well designed and distributed that it is becoming the standard in robotics programming. This allows us to re-use code from one robot to another, preventing us from having to re-invent the wheel continuously.

WHAT IS THE ROBOT ROS API?

The ROS API is the list of ROS topics, services, action servers and messages that a given robot is providing to give access to its hardware, that is, sensors and actuators in a standard way.

If you are not familiar with ROS, you may not understand what those terms mean. But simply put in the developers language, topics/services/messages are like the software functions you can call on a robot to get data from the sensors or make the robot take an action. It also includes the parameters you can pass to those functions.

Most modern robot builders are providing off-the-shelf ROS APIs, like for example ROS-Components shop (http://www.roscomponents.com) that provides all their robot hardware running with a ROS API.

If the robot you want to work with does not run ROS you can make it work by ROSifying it. To ROSify means to adapt your robot to work with ROS, so you provide its ROS API. To ROSify a robot usually requires knowledge to access the hardware. You need to learn how to communicate with the electronics that provide the sensor data or access the motors of the robot. In this book, we are not dealing with that subject because it gets out of scope for developers. We assume you have somebody else doing this job for you.

So for the rest of the book, we will assume that you have access (or willing to have) to a robot that is already ROSified. That is actually the trend in robotics.

WHAT IS ROS ANYWAY?

ROS stands for Robot Operating System (http://www.ros.org). Even if it says so, ROS is not a real operating system since it goes on top of Linux Ubuntu (also on top of Mac, and recently, on top of Windows). ROS is a framework on top of the O.S. that allows to abstract the hardware from the software. This means, you can think in terms of software for all the hardware of the robot. And that are good news for you because this implies that you can actually create programs for robots without having to deal with the hardware. Yeah!

ROS works on Linux Ubuntu or Linux Debian. Experimental support already exists for OSX, Gentoo and Windows, but we really don't recommend you to use them yet unless you are and expert. Check this page (https://tinyurl.com/y6z36nru) for more information about how to use ROS on those systems.

A HISTORY OF ROS

Let us introduce you with a short history of how ROS started, the main reasons behind it, and its current status.

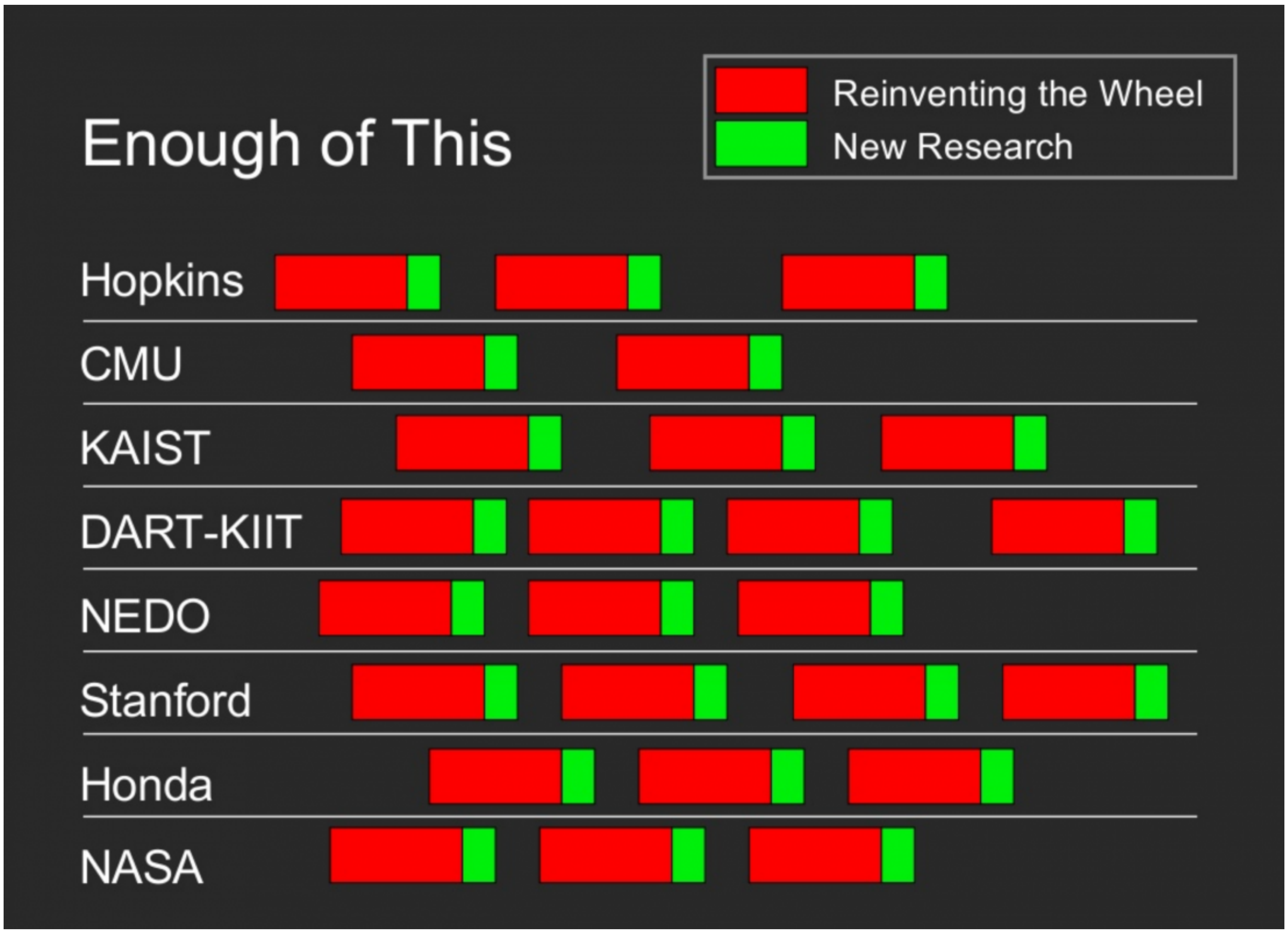
The Stanford Period

ROS started as a personal project of Keenan Wyrobek and Eric Berger while at Stanford, as an attempt to remove the *reinventing the wheel* situation from which robotics was suffering. Those two guys were worried about **the most common problem of robotics** at the time:

- **too much time dedicated to re-implementing the software infrastructure** required to build complex robotics algorithms (basically, drivers to the sensors and actuators, and communications between different programs inside the same robot)
- **too little time dedicated to actually building intelligent robotics programs** that were based on that infrastructure.

Even inside the same organization, the re-invention of the drivers and communication systems was re-implemented for each new project. This situation was beautifully expressed by Keenan and Eric in one of their slides used to pitch investors.

Most of the time spent in robotics was reinventing the wheel (slide from Eric and Keenan pitch deck)



In order to attack that problem, Eric and Keenan created in 2006 a program at Stanford called the Stanford Personal Robotics Program, with the aim to build a **framework** that allowed processes to communicate with each other, plus some tools to help create code on top of that. All that framework was supposed to be used to create code for a robot they also would build, the **Personal Robot**, as a testbed and example to others. They would build 10 of those robots and provide them to universities so that they could develop software based on their framework.

NOTE: People more versed in ROS will recognize in those the precursors of ROS-comm libraries and the Rviz, rqt_tools and the like of current modern ROS distributions. Also, the Personal Robot was the precursor of the famous PR2 robot.

PR1 robot, picture by IEEE Spectrum

Similar frameworks at the time

The idea of such a system for robotics was not new. Actually, there were some other **related projects** already available for the robotics community, Player/Stage, one of the most famous in the line of open source, and URBI in the line of proprietary systems. Even Open-

R, the system developed by Sony which powered the early Aibo robots of 1999, was a system created to prevent that problem (a shame that Sony canceled that project, as they could have become the leaders by now. Ironically, in 2019, Sony launched a new version of the Aibo robot… which runs ROS inside!). Finally, another similar system developed in Europe was YARP.

Actually, one of the leaders of the Player/Stage research project was Brian Gerkey, who later went to Willow Garage to develop ROS, and is now the CEO of Open Robotics, the company behind the development of ROS at present. On its side, URBI was a professional system led by Jean-Christoph Baillie, which worked very well, but could not compete with the free-ness of ROS.

That is an important point to discuss: URBI was (at least) as good as ROS. I used it for many research tasks while doing my Ph.D., for example, this code about making Aibo robot walk using neural networks (in 2005!), and making Aibo dance, and do some other tricks. But URBI failed when competing with ROS. URBI had as many tools for debugging, and as much documentation as ROS. So, why is it that URBI failed against ROS?

The fastest readers will jump to the point that URBI was not free. Actually, it was quite expensive. Was the price what killed URBI? I don't think so. In my opinion, what killed URBI was the lack of community. It takes some time to build a community, but once you have it, it acts like changing gears. URBI could not build a community because it relied on an (expensive) paid fee. That made it so that only people that could buy it were accessing the framework. That limits the amount of community you can create. It is true that ROS was free. But that is not the reason (many products that are free fail). The reason is that they built a community. Being free was just a strategy to build that community.

Switching gears

While at Stanford, Keenan and Eric received a $50k funding and used it to build a PR robot and a demo of what their actual project was. However, they realized that in order to build a really universal system and to provide those robots to the research groups, they would need additional funding. So they started to pitch investors.

At some point around 2008, Keenan and Eric met with Scott Hassan, investor and the founder of Willow Garage, a research center with a focus on robotics products. Scott found their idea so interesting that he decided to fund it and start a Personal Robotics Program inside Willow Garage with them. The Robot Operating System was born and the PR2 robot with it. Actually, the ROS project became so important that all the other projects of Willow Garage were discarded and Willow Garage concentrated only on the development and spread of ROS.

Willow Garage takes the lead

ROS was developed at Willow Garage for around 6 years, until Willow shut down, back in 2014. During that time, many advancements in the project were made. It was this push during the *Willow time* that skyrocketed its popularity. It was also during that time that I acknowledged its existence (I started with ROS C-turtle in 2010) and decided to switch from Player/Stage (the framework I was using at that time) to ROS, even if I was in love with Player/Stage (I don't miss you because ROS is so much better in all aspects… sorry, Player/Stage, it is not me, it is you).

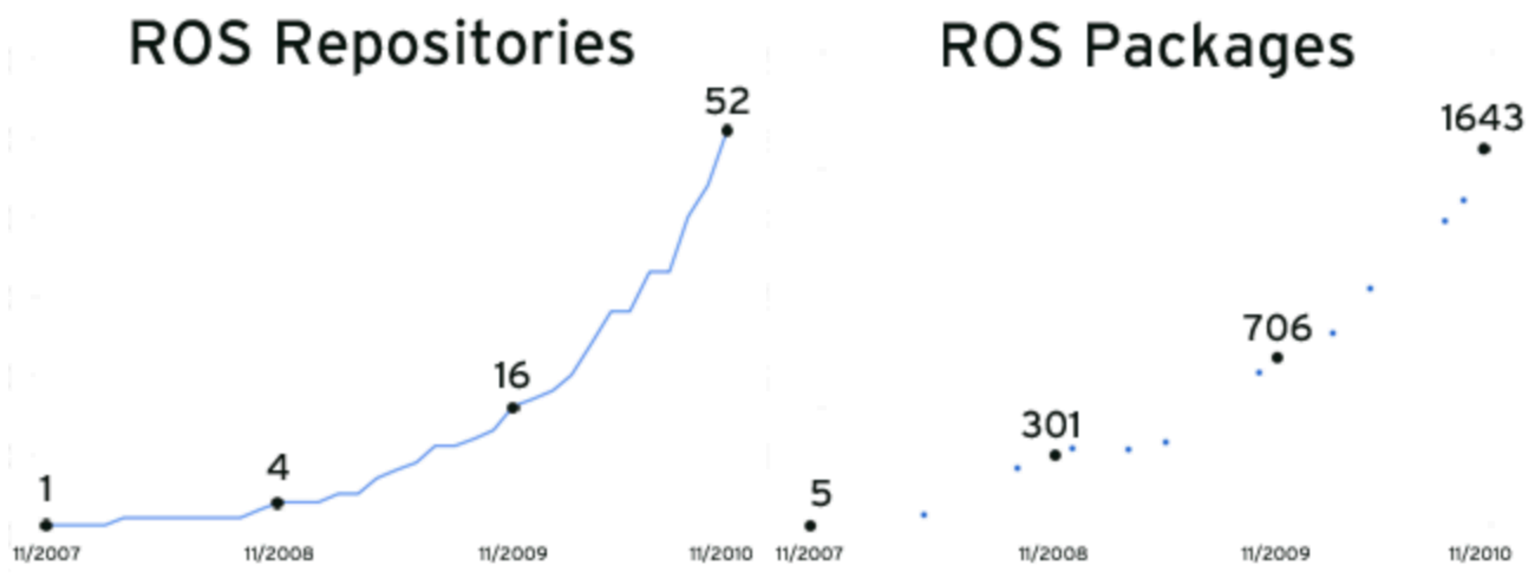
In 2009, the first distribution of ROS was released: **ROS Mango Tango,** also called **ROS 0.4**. As you can see, the name of the first release had nothing to do with the current naming convention (for unknown reasons to this author). The release 1.0 of that distribution was launched almost a year later in 2010. From that point, the ROS team decided to name of the distributions after turtle types. Hence, the following distributions and release dates were done:

- Box Turtle, in 2010
- ROS C-Turtle, in 2010
- Diamond Back, in 2011
- ROS Electric Emys, in 2011
- ROS Fuerte Turtle, in 2012
- ROS Groovy Galapagos, in 2012

Around that time, other even also happened:

1. In 2009, they built a second version of the Personal Robot, the PR2
2. In 2010, they launched ROS Answers, the channel to answer technical questions about ROS.
3. The first edition of the ROSCON was in 2012. The ROSCON became the official yearly conference for ROS developers.
4. In 2010, they built 11 PR2 robots and provided them to 11 universities for robotics software development using ROS (as the original idea of Eric and Keenan). At that point, the PR2 robot was for sale, so anybody in the world could buy one (if enough money was available ;-)).
5. In 2011, as robot simulations started to become very important, the team decided to incorporate Gazebo, the 3D robotics simulator from the Player/Stage project, into ROS. Gazebo became the default 3D simulator for ROS.

As ROS was evolving, all the metrics of ROS were skyrocketing. The number of repositories, the number of packages provided, and of course, the number of universities using it and of companies putting it into their products.



Evolution of ROS in the early days (picture from Willow Garage)

Another important event that increased the size of the ROS community was that, in 2011 Willow Garage announced the [release of Turtlebot robot](), **the most famous robot for ROS developers**. Even if PR2 was the intended robot for testing and developing with ROS, its complexity and high price made it non-viable for most researchers. Instead, the Turtlebot was a simple and cheap robot that allowed anybody to experiment with the basics of robotics and ROS. It quickly became a big hit, and is used even today, in its Turtlebot2 and Turtlebot 3 versions.

In 2013, [Willow Garage 'announced' that the company would dissolve]() that year.

I remember when we received the news that Willow Garage was closing. I was working at that time at Pal Robotics. We at Pal Robotics were all very worried. What would happen with ROS? After all, we had changed a lot of our code to be working with ROS. We removed previous libraries like Karto for navigation (Karto is software for robot navigation, which at present is free, but at that time, we had to pay for a license to use it as the main SLAM and path planning algorithms of our robots).

The idea was that the newly created Open Source Robotics Foundation would take the lead of ROS development. And many of the employees were absorbed by Suitable Technologies (one of the spin-offs created from Willow Garage, which ironically does not use ROS for their products ;-)). The customer support for all the PR2 robots was absorbed by another important company, Clearpath Robotics.

Under the Open Source Robotics Foundation umbrella

Under the new legal structure of the OSRF, ROS continued to develop and release new distributions.

- ROS Hydro Medusa, in 2013
- ROS Indigo Igloo, in 2014
- ROS Jade Turtle, in 2015
- ROS Kinetic Kame, in 2016
- ROS Lunar Loggerhead, in 2017
- ROS Melodic Morenia, in 2018
- ROS Noetic, in 2020

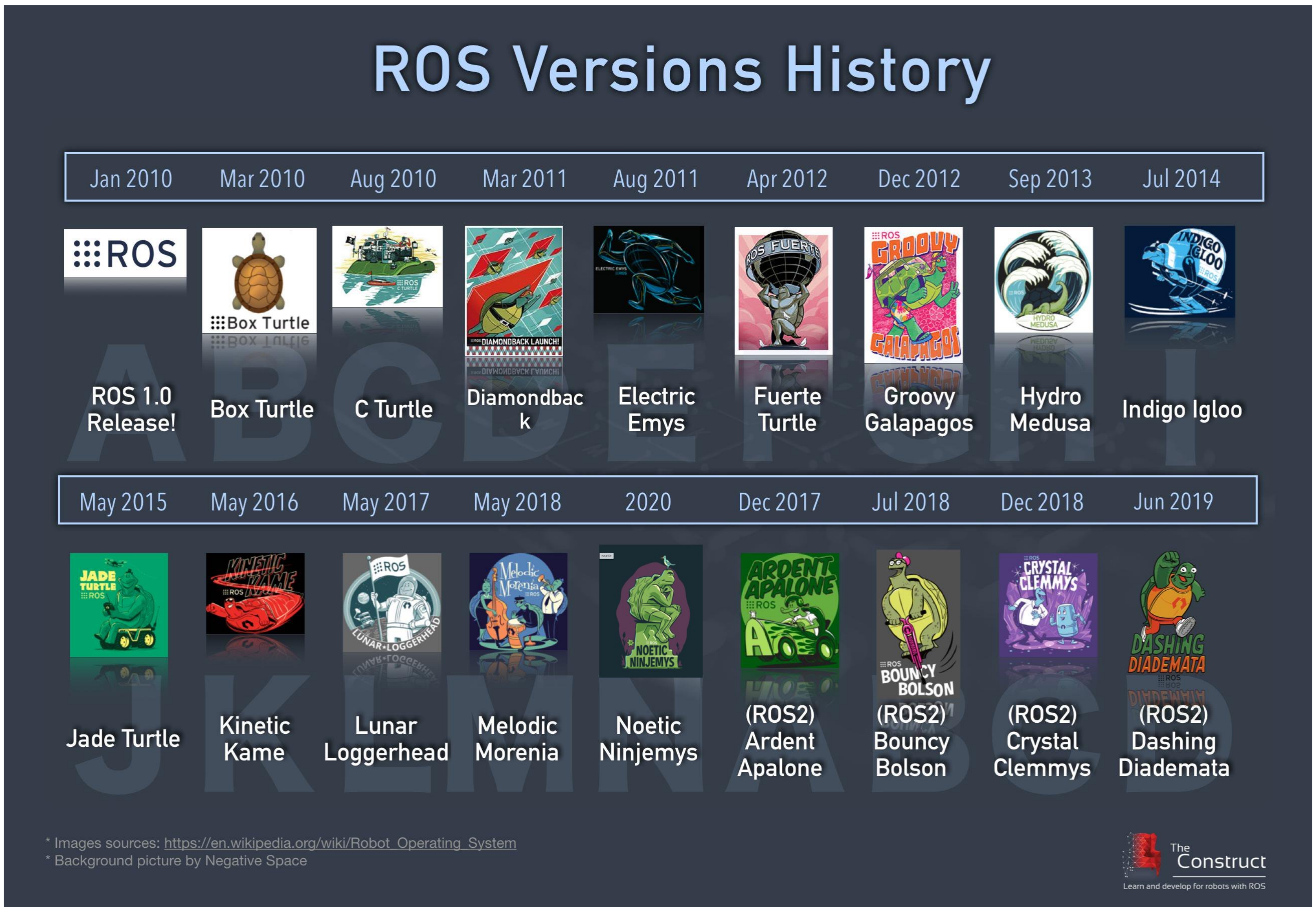
The reports created after each year are publicly available here under the tag ROS Metrics.

Having reached this point, it is important to state that the last and final distribution of ROS1 was released in 2020. It is called ROS Noetic, and it is based on Python 3, instead of Python 2 as all the previous ones were. From that date, no more ROS1 distributions will be released, and the full development will be directed to ROS2 development.

But wait a second, what is ROS2?

ROS 2.0

Around 2015, the **deficiencies of ROS for commercial products** were manifesting very clearly. **Single point of failure** (the roscore), lack of **security**, and **no real-time** support were some of the main deficiencies that companies argued for not supporting ROS in their products. However, it was clear that, if ROS had to become the standard for robotics, it had to reach the industrial sector with a stronger voice than that of the few pioneer companies already shipping ROS in their products.



ROS Versions History

Jan 2010	Mar 2010	Aug 2010	Mar 2011	Aug 2011	Apr 2012	Dec 2012	Sep 2013	Jul 2014
ROS 1.0 Release!	Box Turtle	C Turtle	Diamondbac k	Electric Emys	Fuerte Turtle	Groovy Galapagos	Hydro Medusa	Indigo Igloo
May 2015	May 2016	May 2017	May 2018	2020	Dec 2017	Jul 2018	Dec 2018	Jun 2019
Jade Turtle	Kinetic Kame	Lunar Loggerhead	Melodic Morenia	Noetic Ninjemys	(ROS2) Ardent Apalone	(ROS2) Bouncy Bolson	(ROS2) Crystal Clemmys	(ROS2) Dashing Diademata

* Images sources: https://en.wikipedia.org/wiki/Robot_Operating_System
* Background picture by Negative Space

The Construct
Learn and develop for robots with ROS

In order to overcome that point, the OSRF took the effort to create ROS 2.0, a complete rewriting of ROS1 with solutions for all those problems in mind.

ROS 2.0, even if still not fully at the same level as ROS1, it has already reached its sixth distribution in May 2021 with the release of Galactic Geochelone. This means that it is becoming mature to be used in many applications.

For instace, ROS2 is used in Apex.OS for the operating system of autonomous cars. Apex.OS has been adopted by Toyota for their line of autonomous cars.

Recent movements in the ROS ecosystem

- In 2017, the Open Source Robotics Foundation changed its name to Open Robotics, in order to become more of a company than a foundation, even if the foundation branch still exists (some explanation about it can be found in this post and in this interview of Tully Foote).
- Recently, Open Robotics has opened a new facility in Singapore and established a collaboration with the government there for development. As far as I know, they are working to apply ROS2 to control robots inside a hospital.
- Local ROS conferences have been launched:
  - ROSCON France
  - ROSCON Japan

- In the last couple of years, big players like Amazon, Google, and Microsoft have started to show interest in the system, and show support for ROS.
- New versions of ROS2 for other architectures have been created like ARM u-controllers

That is definitely a sign that ROS is in good health (I would say in better health than ever) and that it has a bright future in front of it. Sure, many problems will arise, but I'm 100% sure that we, and by we I mean the whole ROS community, will solve them and build on top of them.

ROS FOR SERVICE ROBOTS

ROS is becoming the standard in robotics programming, at least in the service robots sector. Initially, ROS started at the Universities and was mainly used for research, but now it is quickly spreading into the companies world. Every day more and more companies and startups are basing their business in ROS (check this article about the top ten robotics companies based on ROS: https://tinyurl.com/y37upbj6).

Before ROS, every robot had to be programmed using the manufacturer's own API. This means that if you changed to another robot, you had to start the whole software again, apart from having to learn the new API. Furthermore, you had to know a lot about how to interact with the electronics of the robot in order to understand how your program was doing. The situation was like with the computers in the 80s when every computer had its own operating system and you had to create the same program for every type of computer. ROS is for robots like Windows is for PCs or Android for the phones.

By having a ROSified robot, that is, a robot that runs on ROS, you can create programs that can be shared among different robots. You can build a navigation program, that is a program to make a robot move around without colliding, for a four wheels robot built by company A and then use the same exact code to move a two wheels robot built by company B… or even use it on a drone from company C with minor modifications.

ROS FOR INDUSTRIAL ROBOTS

ROS is being used in many of the service robots that are created at present. On the other side, industrial robotics companies are still not completely convinced about using it, mainly because they will loose the power of having a proprietary system. However, four years ago an international group called ROS-Industrial (https://rosindustrial.org/) was created whose aim is to make industrial manufacturers of robots, understand the good chance that ROS is for them, since they will be able to use off the shelf all the software that other people has created for other ROS robots.

I made a very interesting interview to Jorge Nicho about building hybrid ROS1 & ROS2 applications for industrial robots. I recommend you to listed to it for getting an idea of the possibilities we have of ROS for industrial robots.

ROS FOR AGRICULTURAL ROBOTS

On the same line as ROS-Industrial, ROS-Agriculture (http://rosagriculture.org/) is another international group that aims to introduce ROS for agriculture robots. I highly recommend that you follow this group if you are interested because they are very well motivated team that can do crazy things with several tons machines, by using ROS. Check by instance this video (https://youtu.be/obu_Ru2gDHk) about an autonomous tractor running ROS, built by Kyler Laird of the ROS Agriculture group.

HOW TO DEVELOP FOR ROBOTS WITH ROS

Now, if you are convinced of becoming a ROS developer, in this guide you are going to find the steps that you can take to become one. We have divided the book in the following sections that should cover the whole development process:

- Pre-requisites
- Setting Up a ROS2 Development Environment
- Learning ROS2
- Coding in ROS2
- Testing ROS2 programs
- Practicing with ROS2 real robots
- Post-requisites

Let's go one by one.

CHAPTER 1: PRE-REQUISITES

We do not know which experience you have as software developer. We found that many people who come new to ROS lack the basic skills necessary to start programming in ROS. That is why we have included here a section of pre requisites that you must meet before trying to get into ROS.

If you want to develop for ROS based robots, you need to know in advance how to program either in C++ or Python. Also you need to be comfortable using the Linux shell. **Those two are mandatory!**

WHY LINUX?

You may be wondering if you could create ROS programs using Windows or even Mac, instead of Linux. Our suggestion is that you go to Linux unless you are really an expert software developer. The Linux version of ROS is the most mature of all of them, which means that you will have **less trouble trying to make something to work in ROS**. The problem here is that learning ROS has a very stepped curve, so you don't need to add more confusion to your learning than the one of ROS itself.

The Windows version of ROS (https://tinyurl.com/y3kv8wmh) is still in a very early state, so you can find confusing to try to make basic ROS things work properly. If you are new to ROS, you don't need to add more confusion to the one that ROS actually has.

The Mac version is more mature than the Windows one (https://tinyurl.com/y59b5mgy), but again, less than the Linux one. Additionally, there is less support in the community list of questions, so unless you are an expert in both Mac system and ROS, you may have a lot of trouble finding errors.

Believe us when we say that your safest bet is to learn Linux and go with ROS with it. Actually, learning the basics of Linux is quite simple and quick. However it is up to you!

In case you don't know Linux and want to give it a try, check this free online course we have created that teaches you the specific subjects of Linux that you are going to need for robotics with ROS:

- **Linux for robotics free online course**: https://tinyurl.com/2vesr3fw

LINUX FOR ROBOTICS
Linux for Robotics
Learn the Linux fundamentals you'll need for robotics development

SHOULD I LEARN PYTHON OR C++ ?

As you may know, **you can create ROS programs mainly in two programming languages: Python and C++**. Mostly, whatever you can do in ROS with C++, you can do it also with Python. Furthermore, **you can have C++ ROS programs talking to other Python ROS programs in the same robot**. Since the nodes communicate using standard ROS messages, the actual language implementation of the nodes is not affecting their communication. That is the beauty of ROS.

If that is the way of working of ROS, then, why not to let everybody program in the language they want? Well, because programming in one language or another has its consequences, especially when you are learning ROS. Let's have a look at them.

Pros of programming ROS in Python

- Is faster to build a prototype. You can create a working demo of your node very fast if you use Python, because the language takes care of a lot of things by itself so the programmer doesn't have to bother.
- You don't need to compile, and spend endless hours trying to catch a hidden bug. Even if you can have bugs in Python, the nature of them are a lot easier and faster to catch.
- You can learn Python very fast.
- You can make really short programs for complex things
- The final code of your node is quite easy to read and understand what it does.

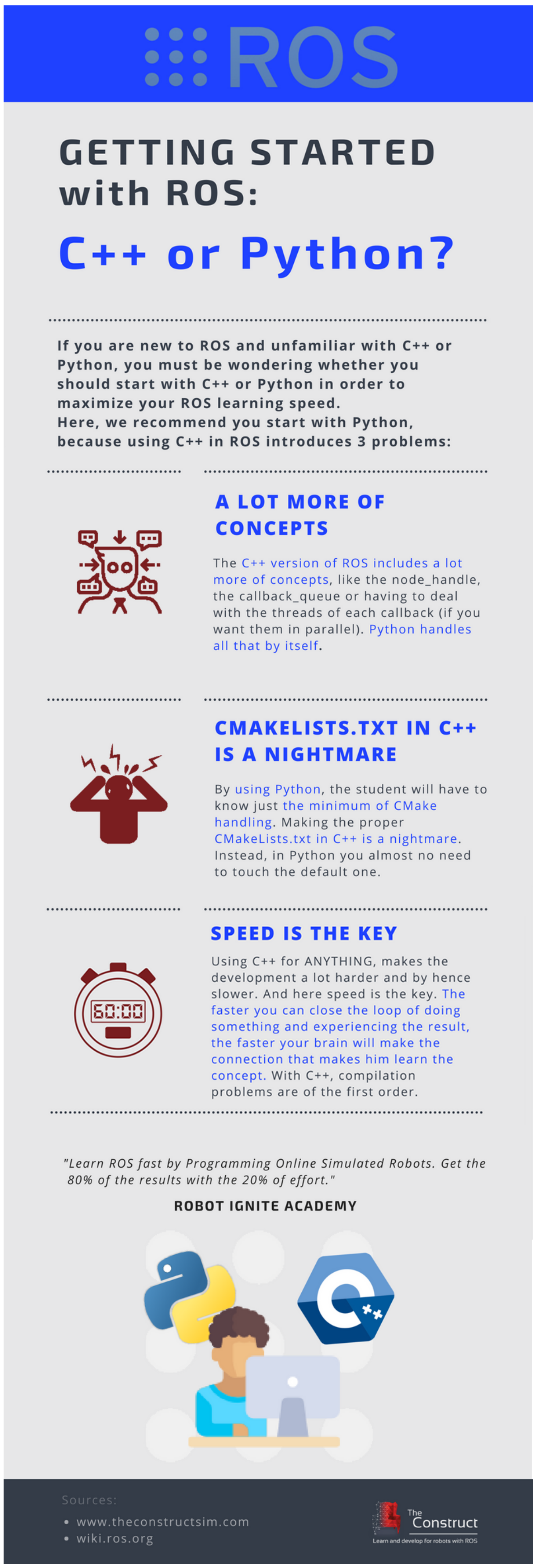
- You can do anything with Python. Python is a very powerful language with libraries for anything you want.
- It is easier to integrate it with web services based on Django. Since Django is based on Python, you can integrate ROS functions easily in the server calls.
- It is easier to understand some ROS concepts if you use the Python API, because some complex concepts are hidden for the developer in the ROS Python API. For example, things like the *Callback Queue* are handled directly by the ROS Python API.

Cons of programming ROS in Python

- It runs slower. Python is an interpreted language, which means that it is compiled in run time, while the program is being executed. That makes the code slower.
- Higher chances of crashing in run time, that is, while the program is running on the robot. You can have a very silly mistake in the code that won't be cached until the program runs that part of the code (in run time).
- Unless you define very clear procedures, you can end with a messy code in a short time if the project grows. I mean, you need to impose some rules for developing, like indicating every class you import from which library is, force strong types on any definition of a variable, etc.
- Doesn't allow real time applications.

Pros of programming ROS in C++:

- The code runs really fast. Maybe in your project, you need some fast code.
- By having to compile, you can catch a lot of errors during compilation time, instead of having them in run time.



ROS

GETTING STARTED
with ROS:

C++ or Python?

If you are new to ROS and unfamiliar with C++ or Python, you must be wondering whether you should start with C++ or Python in order to maximize your ROS learning speed.
Here, we recommend you start with Python, because using C++ in ROS introduces 3 problems:

A LOT MORE OF CONCEPTS

The C++ version of ROS includes a lot more of concepts, like the node_handle, the callback_queue or having to deal with the threads of each callback (if you want them in parallel). Python handles all that by itself.

CMAKELISTS.TXT IN C++ IS A NIGHTMARE

By using Python, the student will have to know just the minimum of CMake handling. Making the proper CMakeLists.txt in C++ is a nightmare. Instead, in Python you almost no need to touch the default one.

SPEED IS THE KEY

Using C++ for ANYTHING, makes the development a lot harder and by hence slower. And here speed is the key. The faster you can close the loop of doing something and experiencing the result, the faster your brain will make the connection that makes him learn the concept. With C++, compilation problems are of the first order.

"Learn ROS fast by Programming Online Simulated Robots. Get the 80% of the results with the 20% of effort."

ROBOT IGNITE ACADEMY

Sources:
- www.theconstructsim.com
- wiki.ros.org

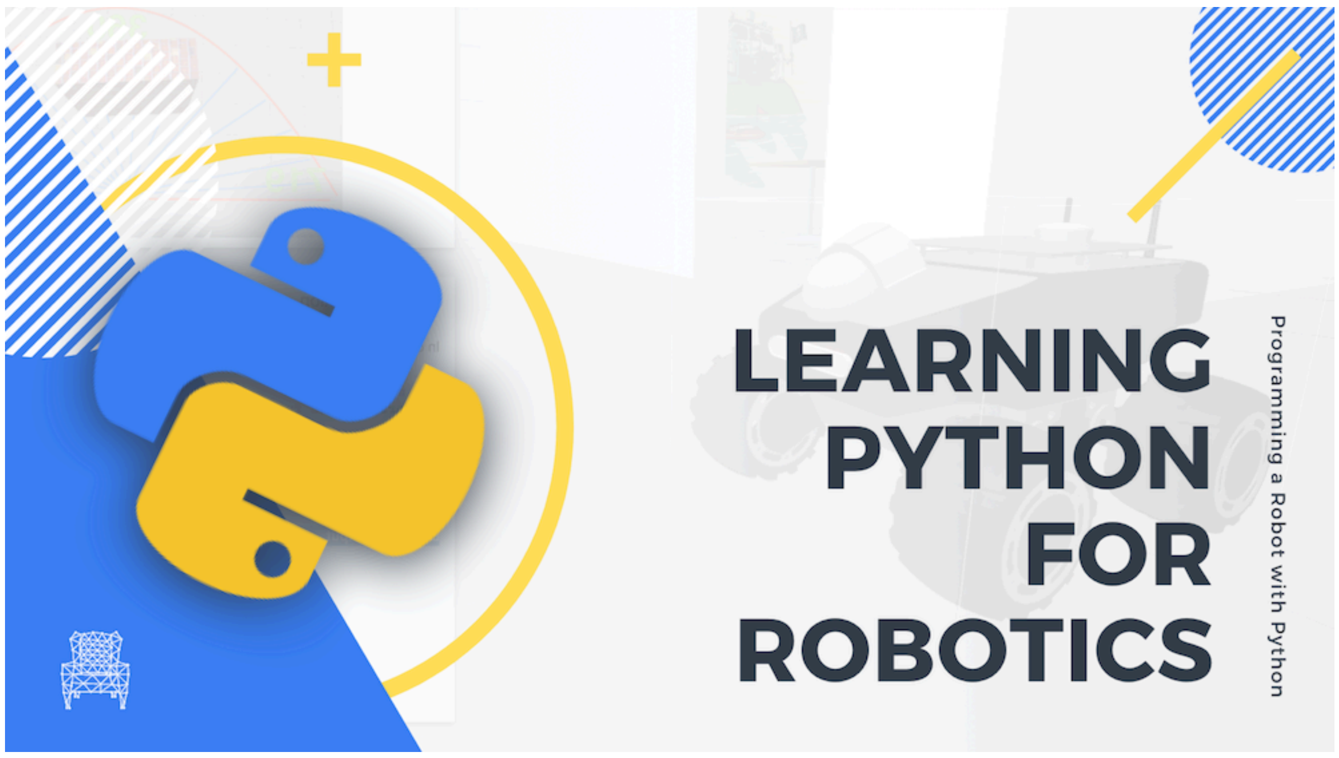
The Construct
Learn and develop for robots with ROS

- C++ has an infinite number of libraries that allow you to do whatever you want with C++.
- It is the language used in the robotics industry, so you need to master it if you want to work there.
- It is necessary for real time code.

Cons of programming ROS in C++:

- C++ is a lot more complex to learn and master. A LOT
- Just creating a small demo of something requires to create a lot of code.
- To understand what a C++ program does can take you a long time.
- Debugging errors in C++ is complex and takes time.

One situation that we get all the time, especially at the University, is that students that come to learn ROS do not know neither C++ nor Python (actually, most of them not even know about Linux shell, but that is another matter). In that case, we strongly recommend start learning ROS using Python. Really, believe me, it is too much to try to learn ROS at the same time that you learn C++. I've seen many times. People get frustrated and complain about the difficulty of ROS. Yes, ROS is difficult but the thing is that you tried to swallow too much by learning C++ and ROS at the same time.



Which Python version uses ROS?

All previous ROS distributions until 2020 have been based on Python 2.7. This means that all the code already available that runs for those distributions needs to run on Python 2.7.

However, the final distribution of ROS 1 (*ROS Noetic* (http://wiki.ros.org/Distributions/ReleasePolicy), released in 2020) is fully based on Python 3, as indicated by main ROS developer Tully Foote in this interview (https://tinyurl.com/y43akq33).

Additionally, the next generation of ROS, that is, *ROS 2*, works only on Python 3, so I think that the answer is clear from the point of view of ROS: learn Python 3.

Which Python version for other important libraries?

Apart from ROS, you will have to use many other libraries to build your robot programs.

For example, if you want to do image manipulation, then you will probably use OpenCV. If you want to use machine learning with the robot, you will probably use Scikit, OpenAI-Baselines or Tensorflow/Keras. If you want to use numerical computation you may use scipy, pandas, or numpy. Except for OpenAI-Baselines, all of them work with both Python versions.

Check this page for a complete list of the most important Python libraries ready for Python 3.

Additional considerations

On top of all said, **Python 2.7 is going to reach its end of life in 2020**. That doesn't mean that your Python 2.7 programs will stop working on that date! It just means that no additional development will be done by the organization that develops Python (that is, The Python Software Foundation). So all the Python software already existing will continue to work, and you will still be able to develop also for it, but no new bug fixing

Finally, clarify that from a beginner point of view, the differences between Python 2.7 and Python 3 are very small. My advice is, if you are a beginner, it will be a safe bet to start with Python 3, and then, learn the small differences that affect your personal case when they arise.

So if you are finally convinced to learn Python, have a look at the free online course we have created which specifically targets Python for robotics.

- **Learn Python for robotics** (online free course): https://tinyurl.com/wa8yupaj

As we already mentioned, ROS can be programmed by C++ or Python. However, if you don't know C++ very well, do not try to get into ROS with C++. If that is your situation, please get yourself to learn ROS with Python. Of course, you can start learning C++ now because C++ is the language used in the robotics industry and you will need to do the transition from ROS Python to ROS C++ later. But your initial learning of ROS should be done programming in Python. We know you are going to think that you can handle it and take both things (learning C++ and ROS) at the same time. Based on our experience, we do not agree at all. Bad decision. Good luck. Still, if you decide to go on that direction, take this free online course that we have created that specifically targets C++ for Robotics (take into account that it is just an introduction. C++ is really huge and you will keep learning it while actually programming the robots):

- **Learn C++ for robotics** (online free course): https://tinyurl.com/3tte9w6h

Final comment, there exist also bindings for other languages like Prolog, Lisp, Nodejs or R. Find a complete list of currently supported languages here (http://wiki.ros.org/Client%20Libraries). We do not recommend you to learn ROS with any of those languages. ROS is complicated enough to having to complicate it more with experimental languages. Also, you will have plenty of time to practice with those once you know ROS with Python.

SHOULD I LEARN ROS1 OR ROS2

That is the billion dollars question!

And as always, the answer depends on your goals. I discussed this questions long enough in this Youtube video (https://youtu.be/KcDCsx5sgX8), but let me summarize here the reasons for one or the other:

1. Use ROS1 if you need something fully functional right now that takes advantage of what already exists in ROS.

2. Use ROS2 if you are building or doing research on something with a longer term. ROS1 is going to die in 5 years, so at the end ROS2 will be the one, so you better prepare now.

3. ROS1 has functionalities that still have not been migrated to ROS2, if you need those, either you develop them yourself for ROS2 or you will need to use ROS1. The exact same happens with ROS2. It has some functionality that ROS1 doesn't have and will not have because they are lacks of the original design. If you need those features, you are going to need ROS2.

In any case, deciding this here, given that you are a beginner, is a difficult question for you. Hence, I'm going to decide for you, following the same reasoning as I did for Python2.7 or Python 3. You are going to learn ROS2!

ROS2 is the future of ROS. Hence, the sooner you start the better. It is also starting to be mature, so you can do many things with it. Then, learn ROS2 and in case you need some functionality from ROS1, learn that in ROS1 and use the ROS_bridge to connect your ROS2 programs with the other ROS1 programs (you don't need to worry now about what is the ROS_bridge, you will learn what it is and how to use it in any basic ROS2 course).

CHAPTER 2: SETTING UP A ROS2 DEVELOPMENT ENVIRONMENT

After having met the pre-requisites to start learning ROS2, next thing you need to do is to set your system up so you can get ROS2 installed in a machine in which to develop your programs and practice while learning. Let's see how to set up a machine for programming for robots with ROS2. You basically have two options:

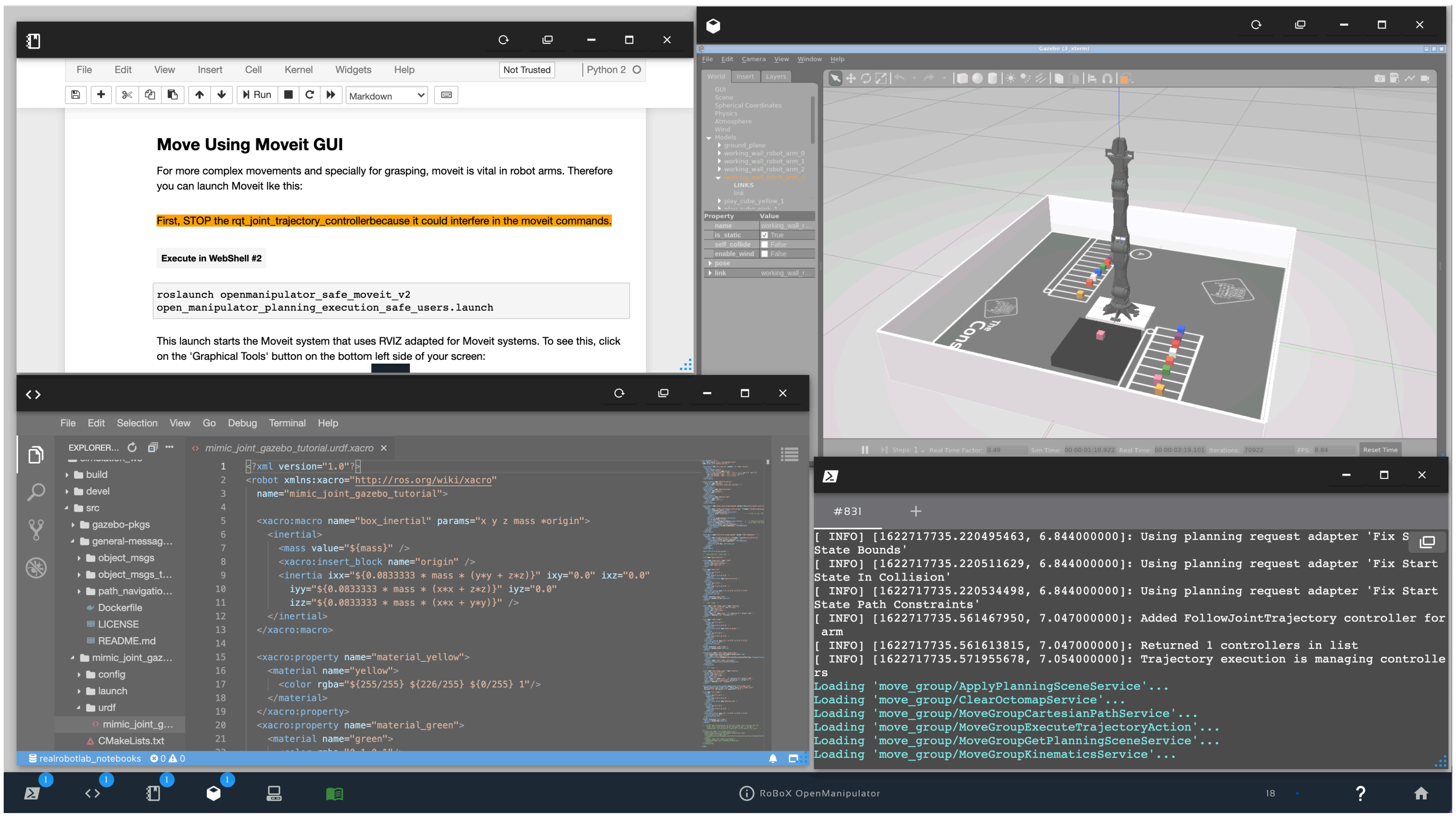
1. **Use our online ROS Development Studio** which already provides everything installed and setup for any type of computer, and requires only a web browser.

2. **Install ROS2 in your local computer**. Several options are available here.

Let's cover now those two options.

USING THE ROS DEVELOPMENT STUDIO

This is by far the easiest way to start with ROS. The ROS Development Studio (ROSDS) is an online development platform provided by The Construct to allow people develop for ROS fast and without caring about the details under the hood.



Move Using Moveit GUI

For more complex movements and specially for grasping, moveit is vital in robot arms. Therefore you can launch Moveit lke this:

First, STOP the rqt_joint_trajectory_controllerbecause it could interfere in the moveit commands.

Execute in WebShell #2

```
roslaunch openmanipulator_safe_moveit_v2
open_manipulator_planning_execution_safe_users.launch
```

This launch starts the Moveit system that uses RVIZ adapted for Moveit systems. To see this, click on the 'Graphical Tools' button on the bottom left side of your screen:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro"
  name="mimic_joint_gazebo_tutorial">

  <xacro:macro name="box_inertial" params="x y z mass *origin">
    <inertial>
      <mass value="${mass}" />
      <xacro:insert_block name="origin" />
      <inertia ixx="${0.0833333 * mass * (y*y + z*z)}" ixy="0.0" ixz="0.0"
        iyy="${0.0833333 * mass * (x*x + z*z)}" iyz="0.0"
        izz="${0.0833333 * mass * (x*x + y*y)}" />
    </inertial>
  </xacro:macro>

  <xacro:property name="material_yellow">
    <material name="yellow">
      <color rgba="${255/255} ${226/255} ${0/255} 1"/>
    </material>
  </xacro:property>
  <xacro:property name="material_green">
    <material name="green">
```

```
[ INFO] [1622717735.220495463, 6.844000000]: Using planning request adapter 'Fix S
State Bounds'
[ INFO] [1622717735.220511629, 6.844000000]: Using planning request adapter 'Fix Start
State In Collision'
[ INFO] [1622717735.220534498, 6.844000000]: Using planning request adapter 'Fix Start
State Path Constraints'
[ INFO] [1622717735.561467950, 7.047000000]: Added FollowJointTrajectory controller for
 arm
[ INFO] [1622717735.561613815, 7.047000000]: Returned 1 controllers in list
[ INFO] [1622717735.571955678, 7.054000000]: Trajectory execution is managing controlle
rs
Loading 'move_group/ApplyPlanningSceneService'...
Loading 'move_group/ClearOctomapService'...
Loading 'move_group/MoveGroupCartesianPathService'...
Loading 'move_group/MoveGroupExecuteTrajectoryAction'...
Loading 'move_group/MoveGroupGetPlanningSceneService'...
Loading 'move_group/MoveGroupKinematicsService'...
```

You only need a web browser to access the ROSDS at http://app.theconstructsim.com. Once you are in, you will be able to start a new ROS project (we call it a rosject ;-) and start developing using the integrated IDE, the Linux shells and even the provided simulations of the robots. Please refer to this Youtube playlist (https://tinyurl.com/yx9lx5nb) for a series of Live Class example videos showing how to use the ROSDS.

The ROSDS provides a complete development environment as if you had everything properly configured in your computer. The advantages are:

- You don't need to install anything in your computer
- You can use Windows, Mac or Linux based computers. So don't worry about the system of your students' computer.
- If something you write breaks the ROS system, you don't care because just by re-starting the ROSject, the ROS system gets re-started to its initial state.
- You can create projects for many different distributions: Kinetic, Melodic, Dashing, Foxy, etc…
- You can share your ROS code with a single link and the code will work the same way to anybody who receives the link (including simulations, datasets and documentation). Here, there are a couple of examples:
    - This is an example of one of a ROS Live Class delivered by Alberto from The Construct, where he teaches how to do ROS topics statistics with ROS2: https://app.theconstructsim.com/#/l/d3a5164/ (click on the link to get a full copy of the code and documentation).
    - This is another example where Alberto teaches Basic Robotics probability with a simulated ROSbot: https://app.theconstructsim.com/#/l/327e8e7d/ (click on the link to get a full copy of the code and documentation).

The Construct's ROSDS is widely used by Universities and research centres worldwide because they don't require the students to install anything in their computers, specially when that requires to partition the hard disk with different operating systems.

You can skip the rest of this chapter if you decide to use the ROSDS.

INSTALLING ROS2 IN YOUR MACHINE

The section explains how to install ROS2 in Linux. ROS2 works on Linux Ubuntu, Linux Debian or Linux Gentoo as well as MacOS and Windows. We are only going to cover the installation of Linux Ubuntu because it is simplest and the one more mature. Remember, you must remove problems when dealing with ROS, especially if you are beginner.

We will assume that you don't even have Ubuntu installed, so let's start by installing the operating system.

1) Installing Ubuntu

To have Ubuntu working on your Windows machine, you have two options:

a) Install Ubuntu on the machine

What we recommend you is to install one of the latest versions of Ubuntu in your computer. As up today, June 2021, we recommend you to install Ubuntu 20.04. Check this page (https://www.ubuntu.com/download/desktop) for a guide on how to install Ubuntu on a windows computer. It covers two options: removing completely the Windows partition substituting it by Ubuntu, or making a dual boot where you can have both systems working on the same machine (you decide which operating system to use on boot time).

b) Use a virtual machine with Ubuntu installed

Still, if you are not convinced about installing Ubuntu in your machine and keep thinking about using Windows or Mac for ROS development you can use a virtual machine with Ubuntu installed on your home O.S. (either Mac or Windows). Here you can find a post (https://linuxhint.com/install_ubuntu_virtualbox_2004/) about how to install Ubuntu 20.04 on Virtual Box. I really don't recommend this option because you are adding an additional layer of complexity to a subject that is quite complex on the first time. Also, you will see that the whole system will go very slow.

2) Installing ROS2

Once you have an Ubuntu system working in your computer, you need to install ROS2 on it.

We are going to cover how to install the latest release, ROS2 Galactic. Those instructions are a simplification from the original instructions published by Open Robotics here: https://bit.ly/3idFsUp

Step 1 – Configuration

The first step is setup your package environment. Type the following commands:

```
sudo apt install software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install curl gnupg lsb-release
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key  -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Step 2 – ROS2 Installation

Download the latest build of ROS2 Galactic by doing the following command:

```
cd ~
wget https://github.com/ros2/ros2/releases/download/release-galactic-20210523/ros2-galactic-20210523-linux-focal-amd64.tar.bz2
```

Then unpack the package downloaded:

```
mkdir -p ~/ros2_galactic
cd ~/ros2_galactic
tar xf ~/Downloads/ros2-galactic-20210523-linux-focal-amd64.tar.bz2
```

Then update all the package system in your Ubuntu and install ROS2 dependencies

```
sudo apt update
sudo apt install -y python3-rosdep
sudo rosdep init
rosdep update
rosdep install --from-paths ~/ros2_galactic/ros2-linux/share --ignore-src --rosdistro galactic -y --skip-keys "console_bridge fastcdr fastrtps osrf_testing_tools_cpp poco_vendor rmw_connextdds rti-connext-dds-5.3.1 tinyxml_vendor tinyxml2_vendor urdfdom urdfdom_headers"
sudo apt install -y libpython3-dev python3-pip
```

Now you are ready to use ROS2. For that, you need to source it on any shell you want to launch a ROS2 command. Do the following command to source ROS2:

```
. ~/ros2_galactic/ros2-linux/setup.bash
```

After that you will have ROS2 installed in your computer. Now you are ready to use ROS2 in your computer. But before you can start creating your ROS2 programs, you need to learn how ROS2 works. That is the subject of the next chapter.

CHAPTER 3: LEARNING ROS2

Now, with all the previous background and setup, it is the moment to start learning ROS2.

FIVE METHODS TO LEARN ROS2

1- The official tutorials: ROS2 wiki

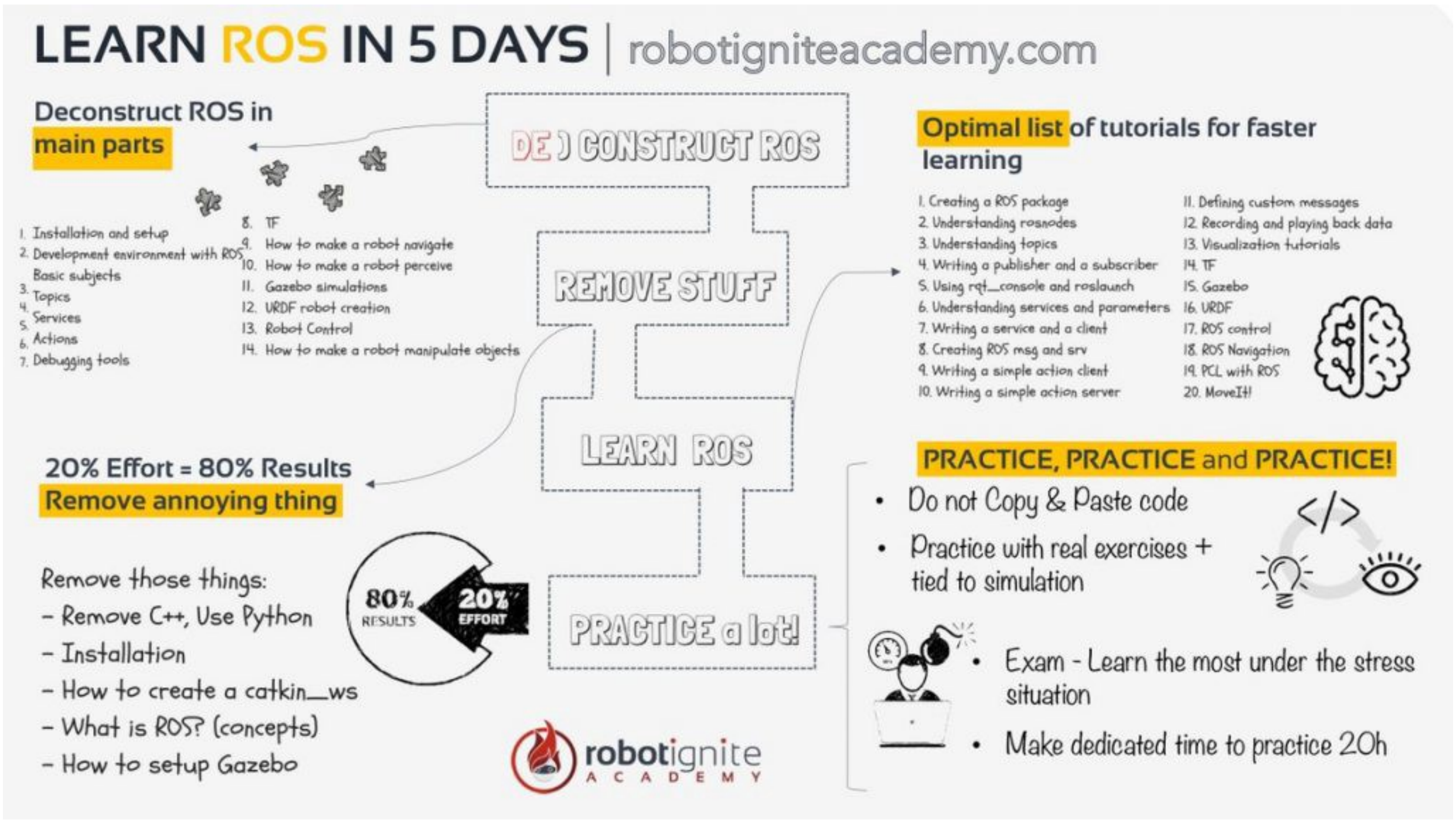
The official ROS2 tutorial website (http://docs.ros.org/en/galactic/Tutorials.html) provided by Open Robotics (https://www.osrfoundation.org/), that is, the organization that builds and maintains ROS, is very comprehensive and it is available in multiple languages. It includes details for installation, how tos, documentation of ROS2, etc. and it's completely free. Just follow the tutorials provided on ROS2 Wiki page, and get started.

This type of tutorial belongs to the traditional academic learning materials. They start by describing concepts one by one, following a well defined hierarchy. It's good material but easy to get lost while reading, and it takes time to grasp the core spirit of ROS.

There is a long list of tutorials there. The list is so big that it can be overwhelming. If you decide to use this method to learn ROS, then we recommend you that you follow the following order for the tutorials for optimal learning:

- Creating a workspace: https://bit.ly/3uU6pzq

- Creating your first ROS2 package: https://bit.ly/3piEAPY
- Creating a launch file: https://bit.ly/3cc3BXx
- Understanding ROS2 topics: https://bit.ly/2SSJUxt
- Understanding ROS2 services: https://bit.ly/3cfYYMi
- Understanding ROS2 parameters: https://bit.ly/3chSywh
- Understanding ROS2 actions: https://bit.ly/3icmsph
- Creating custom ROS2 Msg and Srv files: https://bit.ly/2SWBKnr
- Composing multiple nodes in a single process: https://bit.ly/3wRMaUg
- Using colcon to build packages: https://bit.ly/3w10dXm
- Using TF2 with ROS2: https://bit.ly/2SVvE77
- Using URDF with robot_state_publisher: https://bit.ly/3vNMVh7
- Management of nodes with managed lifecycles: https://bit.ly/3fKvsjZ
- ROS2 Navigation: https://navigation.ros.org/
- ROS2 Control: https://ros-controls.github.io/control.ros.org/



Deconstruct ROS in main parts

1. Installation and setup
2. Development environment with ROS
   Basic subjects
3. Topics
4. Services
5. Actions
6. Debugging tools
8. TF
9. How to make a robot navigate
10. How to make a robot perceive
11. Gazebo simulations
12. URDF robot creation
13. Robot Control
14. How to make a robot manipulate objects

Optimal list of tutorials for faster learning

1. Creating a ROS package
2. Understanding rosnodes
3. Understanding topics
4. Writing a publisher and a subscriber
5. Using rqt_console and roslaunch
6. Understanding services and parameters
7. Writing a service and a client
8. Creating ROS msg and srv
9. Writing a simple action client
10. Writing a simple action server
11. Defining custom messages
12. Recording and playing back data
13. Visualization tutorials
14. TF
15. Gazebo
16. URDF
17. ROS control
18. ROS Navigation
19. PCL with ROS
20. MoveIt!

20% Effort = 80% Results
Remove annoying thing

Remove those things:
- Remove C++, Use Python
- Installation
- How to create a catkin_ws
- What is ROS? (concepts)
- How to setup Gazebo

PRACTICE, PRACTICE and PRACTICE!

- Do not Copy & Paste code
- Practice with real exercises + tied to simulation
- Exam - Learn the most under the stress situation
- Make dedicated time to practice 20h

2- Video tutorials

Video tutorials provide a unique presentation which show how programs are created and ran, in a practical way. It allows you to learn how to carry a ROS project from the professional or instructor, which alleviate a beginner's fear to start learning ROS to a certain degree. It is plenty of tutorials of ROS on Youtube. Just search for *ROS tutorial* and select among all the videos shown.

But there is a drawback that anyone can create a video, this means not require any sort of qualification to publish their content, credibility might be shifty. Also it is difficult to find an order on the videos, that provide a full learning path. We only recommend to use this method when you want to learn about a very specific point of ROS, or when you need to find the answer to a very specific question.

One of the ROS video tutorial course provided by Dr. Anis Koubaa from Prince Sultan University, is a great starting point to learn ROS (https://tinyurl.com/yxulr3g4). Course combines a guided tutorial, different examples and exercises with increasing level of difficulty along with the use of an autonomous robot.

Another of the great resource is our Youtube channel (https://tinyurl.com/y3bvvcmn). Sorry to mention like that, but it is true, based on the comments people leave for us! In our channel we teach about ROS basics, ROS2 basics, Gazebo basics, ROS Q&A, and even real robot projects with ROS. Please have a look at it and let us know if you actually find it useful or not.

3- ROS face-to-face training

Face-to-face instructional courses are the traditional way of teaching. It builds strong foundations of ROS into students.

ROS training is usually a short course, it requires you to focus on learning ROS in a particular environment and a period of time. With the interaction with instructors and colleagues which allows you to get feedback directly. Under the guidance and supervision of instructors, it definitely encourages a better result.

The following are some of the institutions that are holding live ROS training or summer courses on a regular basis:

- **FH Aachen Summer Schools** (https://tinyurl.com/yxsjyjs6)
- **ROS China Summer School** (http://www.roseducation.org/)
- **ROS Industrial Trainings**: a series of trainings on ROS with specific emphasis on ROS Industrial. They have several trainings distributed along the three continents Europe, America and Asia (https://rosindustrial.org/events-summary)
- **Mastering Mobile Manipulators Training**: a full week of learning how to build full ROS applications for mobile manipulators, including the navigation, arm control, grasping, robot behavior and user interfaces (https://bit.ly/3ci6IgW)

When we talk about ROS face-to-face training we are not including the conferences, since the learning that you can get there is very little and specific. ROS conferences are more to be up to date of the latests progress in the field, rather than learning ROS. So it is supposed that you go to the conferences with the minimum ROS knowledge to take any profit from it.

Some of the ROS conferences around the world:

- The official **ROSCON** https://roscon.ros.org/
- **ROSCON France** https://roscon.fr/
- **ROSCON Japan** http://roscon.jp/
- The **ROS Developers Day** online http://rosdevday.com/

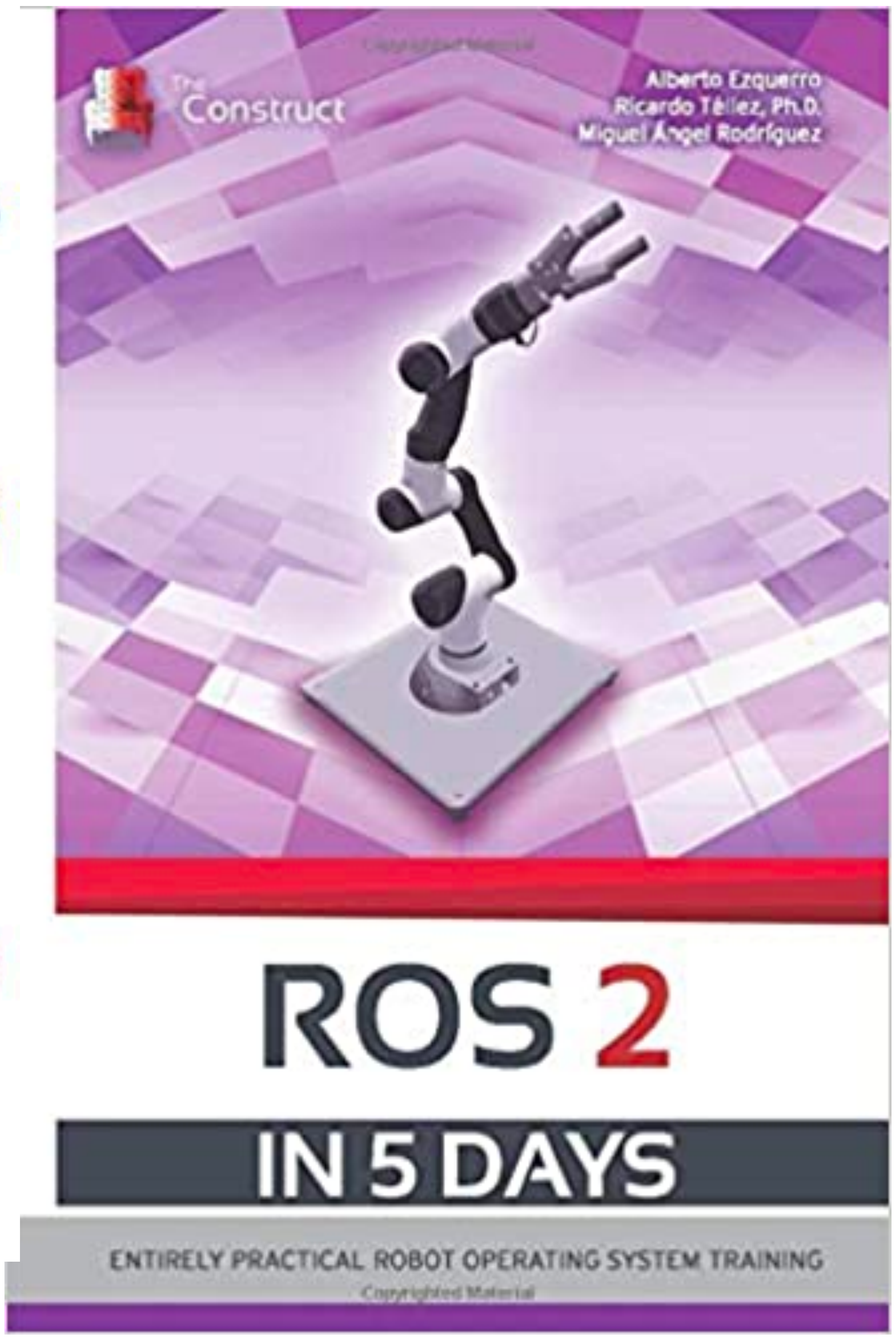
4- ROS Books

ROS books are published by experienced roboticists. They extract the essence of ROS and present a lot of practical examples.

Books are good tools for learning ROS, however, it requires a high self-discipline and concentration so as to achieve the desired result. (but they are only as good as the person using them, it depends on many factors. It allows many distractions to easily affect your progress, unless the strong self-discipline to ensure is paying full attention at all times).

Here you have a list of ROS2 books so far:

- ROS2 in 5 Days (https://amzn.to/2l5DkTE)
- Starting with ROS2 (https://www.youtalk.jp/get-started-ros2/)



5- Integrated ROS2 learning platform – Robot Ignite Academy

We have created integrated learning platform as a new way of learning ROS fast. Compared to other learning methods, it provides a more comprehensive learning mechanism. It is the easiest and fastest of all the methods.

In the platform, you will learn ROS2 by practicing on simulated robots. You will follow a step by step tutorial and will program the robots while observing the program's result on the robot simulation in realtime. The whole platform is integrated into a web page so you don't have to install anything. You just connect by using a web browser from any type of computer and start learning.

Well, perhaps the only drawback is it is not free. You can, though, try the platform with a free account at www.robotigniteacademy.com

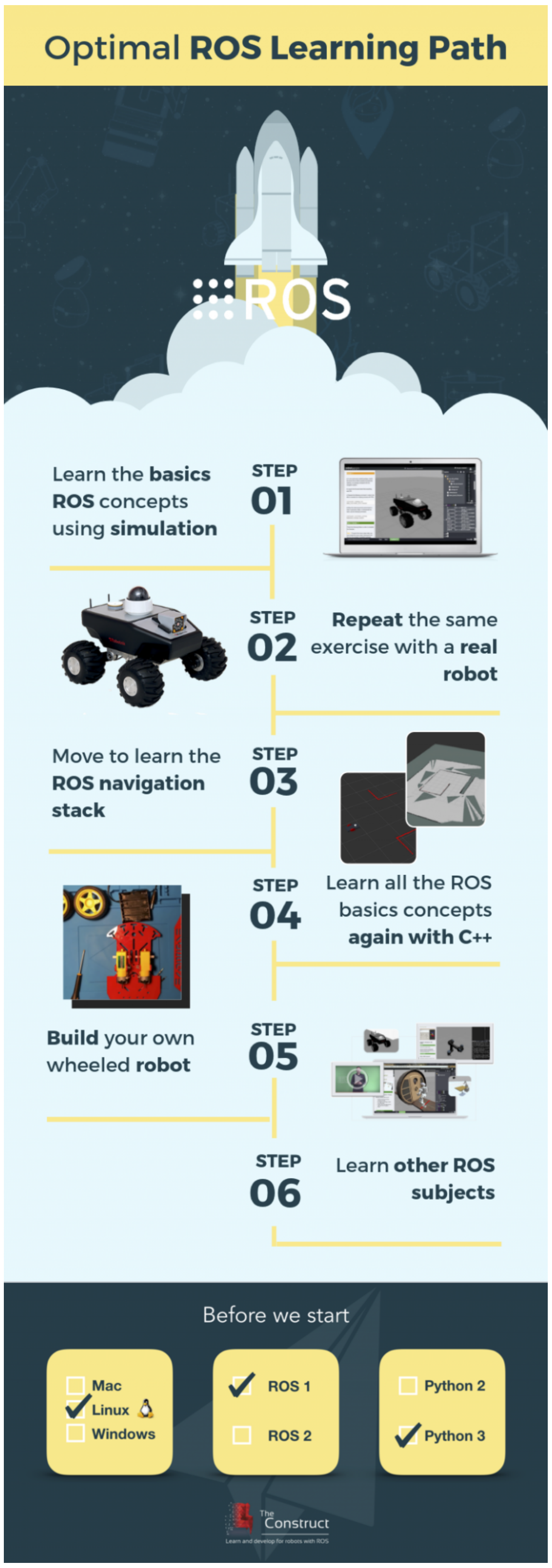
We have to tell you that many Universities around the world are using that academy to teach their students about ROS and robotics with ROS. For instance: University of Tokyo, University of Michigan, University of Sydney, University of Reims, University of Luxembourg, etc.. as well as companies like Softbank, 3M, Robotnik or HP.

Recently, we have added a series of Remote Real Robot Labs, to which students can connect remotely and practice with real robots from their location. The labs include a wheeled robot and an arm robot, to practice all basic robotics concepts with real robots.

HOW TO ASK QUESTIONS

It will happen that while learning or creating your own ROS programs, you will have doubts. Things that doesn't work as the documentation indicates, strange errors that appear even if you checked that everything is correct,... you will need to get support from the community in order to keep improving. For that purpose, you can use two different channels to ask questions:

- **ROS answers** (http://answers.ros.org/): that is the official forum for questions about ROS. Use that forum to **ask your technical questions about ROS**. Also you can use it for checking for previous answers to similar questions to yours. Finally, use it to help others by answering the questions that you know about.
- **ROS discourse** (https://discourse.ros.org/): this is a second official forum. However, this forum is not about answering technical questions. Instead it is about announcing things related to ROS. If you have created a new package that interfaces ROS with neural networks, or you are holding an event about ROS, or you are releasing a product about ROS, that is the place to post it. **Do not use this forum for technical questions**, but for announcements or questions related to the ROS community.


Optimal ROS Learning Path
ROS
Learn the basics ROS concepts using simulation
STEP 01
STEP 02
Repeat the same exercise with a real robot
Move to learn the ROS navigation stack
STEP 03
STEP 04
Learn all the ROS basics concepts again with C++
Build your own wheeled robot
STEP 05
STEP 06
Learn other ROS subjects
Before we start
Mac
Linux
Windows
ROS 1
ROS 2
Python 2
Python 3
The Construct

CONCLUSION

From all the methods presented there, I'm going to recommend you our online Robot Ignite Academy because it is by no means, the fastest and more comprehensive route to learn ROS. It is not something that I say, but what our customers say. Our online academy has a price but it will speed considerably your learning of ROS.

In case you don't want to spend money and you are not in a hurry, then the best option would be to follow the wiki tutorials. They are a little bit difficult to follow and it is a slow path to learn ROS, but it definitely works. After all, that is the method the author of this guide used to learn ROS.

As a final recommendation, Alberto from The Construct delivers a **free live online class about ROS/ROS2 every Tuesday** at 18:00 CEST/CET (the **ROS Developers Live Class**, https://tinyurl.com/y3bvvcmn). We recommend you to attend since it is 100% practice based and deals with a new ROS2 subject every week.

CHAPTER 4: CODING IN ROS2

In order to create ROS programs, you will need a C++ or Python code editor. In this chapter we are going to show you a list of integrated environments for programming ROS with those languages. Many others do exist, but we are putting here the most complete and easy to start with.

Note: since ROS only fully works on Linux (at least at present), all the IDEs included here are Linux environments, even if Windows versions may exist.

CLOUD IDES

Using the ROS Development Studio as your IDE

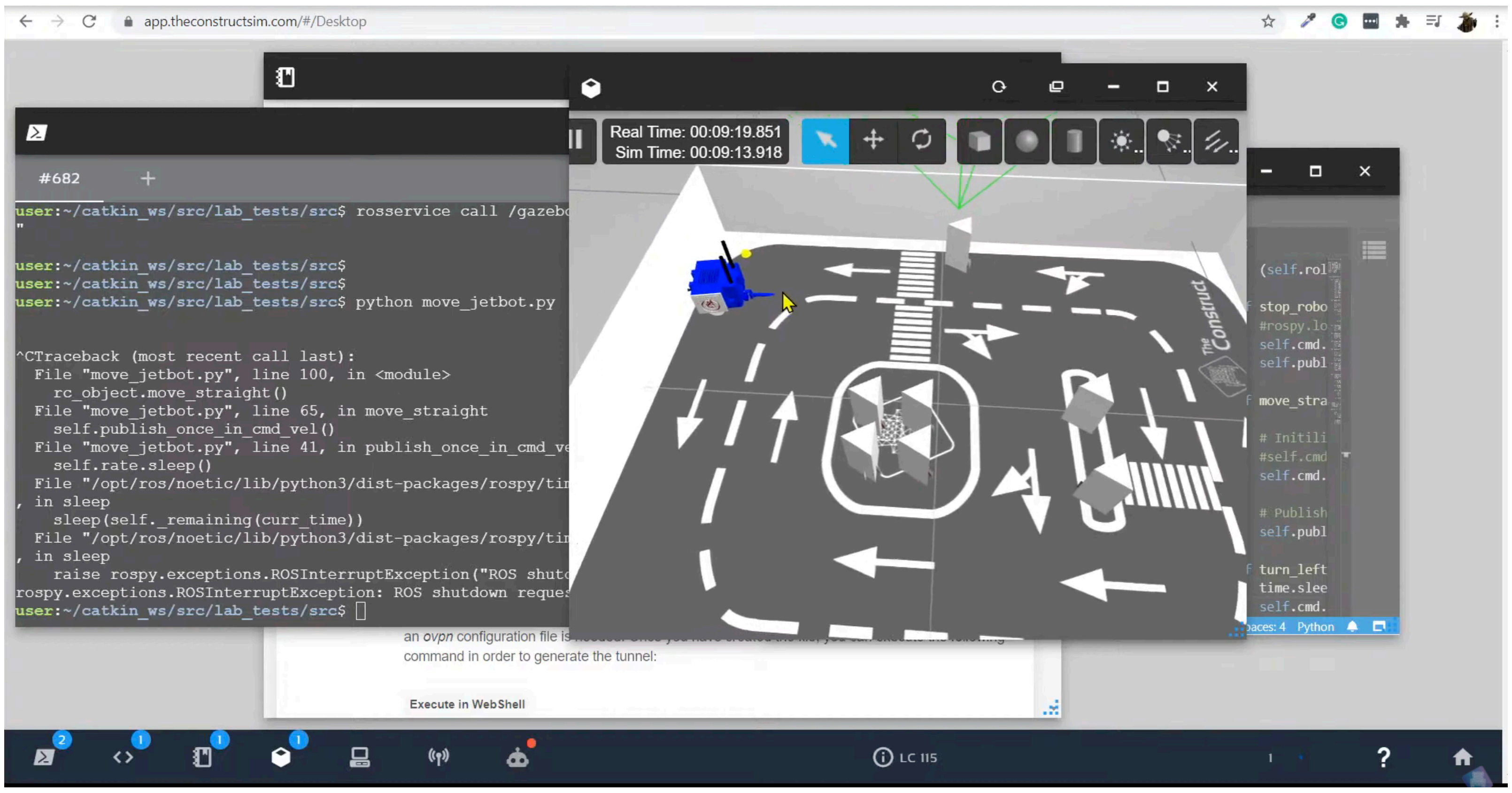
You can simplify your life when working with ROS by using the ROS Development Studio (http://rosds.online). **The ROSDS comes off-the-shelf with ROS, Gazebo, and IDE already installed. You need no installation in your computer**, just a web browser. The main advantage of the ROSDS is that it allows you to develop on ANY operating system. It also allows programming in both C++ or Python with autocomplete functionality.

Additionally, **ROSDS provides running simulations of the robots**, so you can quickly test your programs in realistic simulations of robots and check that your program is actually doing what is supposed to do. This is a very important step while developing for robots (as you will see in the next chapter). You have a full set of tutorials about how to use ROSDS here.

ROSDS allows you to **share your already working ROS projects** with your peers with a single click. By sharing your project, your mates will receive an exact copy of what you shared. This implies that your peers will be able to run it the exact same way as you. This is interesting for replicating research results, for providing exercises to students, for robot competitions based on simulations or for just sharing code with non-experts.

Finally, let us mention that ROSDS can **deploy into your real robot**. You can switch from testing in the simulation to running in the robot with a couple of clicks.

If you choose to use the ROSDS then you can skip the rest of this chapter.



```
user:~/catkin_ws/src/lab_tests/src$ rosservice call /gazeb
"

user:~/catkin_ws/src/lab_tests/src$
user:~/catkin_ws/src/lab_tests/src$
user:~/catkin_ws/src/lab_tests/src$ python move_jetbot.py


^CTraceback (most recent call last):
  File "move_jetbot.py", line 100, in <module>
    rc_object.move_straight()
  File "move_jetbot.py", line 65, in move_straight
    self.publish_once_in_cmd_vel()
  File "move_jetbot.py", line 41, in publish_once_in_cmd_ve
    self.rate.sleep()
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/tin
, in sleep
    sleep(self._remaining(curr_time))
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/tin
, in sleep
    raise rospy.exceptions.ROSInterruptException("ROS shuto
rospy.exceptions.ROSInterruptException: ROS shutdown reques
user:~/catkin_ws/src/lab_tests/src$
```

```
(self.rol
stop_robo
#rospy.lo
self.cmd.
self.publ
move_stra
# Initili
#self.cmd
self.cmd.
# Publish
self.publ
turn_left
time.slee
self.cmd.
```

an *ovpn* configuration file is
command in order to generate the tunnel:

LOCAL IDES

What follows is a list of some of the best editors for each language (Python and C++) that you can install in your local computer.

Eclipse

- **Languages supported:** C++ and Python
- **O.S.:** Linux, Windows, macOS
- **Download from:** https://www.eclipse.org/downloads/
- **How to configure it for ROS:** http://wiki.ros.org/IDEs#Eclipse

CLion
- **Languages supported:** C++
- **O.S.:** Linux, Windows, macOS
- **Download from:** https://www.jetbrains.com/clion/
- **How to configure it for ROS:** http://wiki.ros.org/IDEs#ROS-Robot_Operating_System

NetBeans
- **Languages supported:** Python
- **O.S.:** Linux, Windows, macOS
- **Download from:** http://www.netbeans.org/
- **How to configure it for ROS:** http://wiki.ros.org/IDEs#NetBeans

QTCreator
- **Languages supported:** C++
- **O.S.:** Linux, Windows, macOS
- **Download from:** https://www.qt.io/download
- **How to configure it for ROS:** http://wiki.ros.org/IDEs#QtCreator

PyCharm
- **Languages supported:** Python
- **O.S.:** Linux, Windows, macOS
- **Download from:** https://www.jetbrains.com/pycharm/download/#section=mac
- **How to configure it for ROS:** http://wiki.ros.org/IDEs#PyCharm_.28community_edition.29

Visual Studio Code
- **Languages supported:** Python
- **O.S.:** Linux, Windows, macOS
- **Download from:** https://code.visualstudio.com/download
- **How to configure it for ROS:** https://marketplace.visualstudio.com/items?itemName=ms-iot.vscode-ros

CHAPTER 5: TESTING

How can you test the programs you are developing for a robot? If you are building a program for a robot, you have to make sure that it works properly and for that you need some tests.

If you think that you should test your programs directly on the robot, think again. Testing on the real robot is only the last of the steps you will need to do in a testing procedure. Why you should test on the real robot first?:

- Because you can break the robot or harm somebody. When you start testing your code, many things can be wrong in your program that can make the robot go crazy and crash against something. We are not talking about mobile apps here!! We are talking about physical stuff that can be broken or harm somebody.
- Because testing on the robot takes a ot more time than testing on your local machine. To test on the real robot, usually you will need to have a physical setup that you need to rebuild everytime that you test (because by doing the test, the physical robot will change position or things of the environment that need to be reset).

Testing with the real robot is mandatory for the creation of a program for a robot (at least at present). However, it is the last of the steps in the testing procedure.

The proper procedure for developing for robots works as follows:

1. You create the ROS program in your computer
2. You test someway in your computer that the program works

3. Based on the test results, you modify the program until the local testing works properly
4. Once you are sure that your program works correctly on your computer, then you test on the real robot.
5. You will see that in the real robot, the results differ from your local tests. Then you need to modify the code again, as well as the local tests in order to reflect the points that were not working on the robot
6. Repeat the whole testing cycle until it works on your local machine first, and then on the real robot

IMPORTANT: if you want to develop fast and with as less hassle as possible, you should never go and try your code on the robot until it works properly in the local test environment. **If the code doesn't work in your local tests, is never going to work on the real robot!**

Then the question is, how do you test your programs in your local machine?

Well, you have several of options to test your ROS programs without having to use a robot.

LEVELS OF TESTING

(this section information was extracted from ROS wiki)

Level 1. Library unit test (unittest, gtest): a library unit test should test your code sans ROS (if you are having hard time testing sans ROS, you probably need to refactor your library). Make sure that your underlying functions are separated out and well tested with both common, edge, and error inputs. Make sure that the pre- and post- conditions of your methods are well tested. Also make sure that the behavior of your code under these various conditions is well documented.

Level 2. ROS node unit test (rostest + unittest/gtest): node unit tests start up your node and test its external API, i.e. published topics, subscribed topics, and services.

Level 3. ROS nodes integration / regression test (rostest + unittest/gtest): integration tests start up multiple nodes and test that they all work together as expected.

Level 4. Functional testing. At this level, you should test the full robot application as a whole. Check that the robot is able to navigate, to grasp, to recognise people, etc... You will need to devise some tests that allow you to check that your code is still working on those functionalities. Usually the tests are done using simulations or bag files.

CREATING TESTS OF LEVELS 1,2 AND 3

In order to create tests for those levels, you will need to use the following libraries and packages:

- Specifically for Level 1 of tests
  - *gtest*: also know as Google Test, it is used to build unit tests in C++. You can get the full documentation about using *gtest* with ROS in this link.
  - *unittest*: is the framework provided for doing unit tests in Python. You can get a full documentation about using *unittest* with ROS in this link.
- Additionally, for levels 2 and 3 of tests:
  - *rostest*: it is a package provided by ROS that allows the creation of nodes with specific configurations to test that your whole ROS program is working properly inside the ROS framework by executing unit tests from within. That is why, for the creation of tests for those two levels we need to combine *rostest* with either *gtest* (C++ case) or *unittest*

(Python case). You can get a full documentation about using *rostest* in this link.

CREATING TESTS OF LEVEL 4

Level 4 tests are the ones that allow us to test if the program is actually doing what is supposed to do. For testing that we have three different option, from further to closer to reality.

Using mocks for testing

If you are a developer, you already know what mocks are. You can create your own mocks that emulate the connection to the different parts of the ROS API of your robot. In this page, you will find some information about how to use mocks in ROS.

Working with mocks in ROS is not an easy option since it requires a lot of preparation work. Also it is of very limited usefulness since can only produce what you have put on it previous. Using mocks is an option that I don't really recommend for developing. Use it only if you cannot use any of the options below or if you are creating unit tests for your code. On my personal experience, I have never used them, always used any of the next two options.

Anyway, if you want to use mocks for your testing, then I recommend you to use the Google Test environment https://github.com/google/googletest

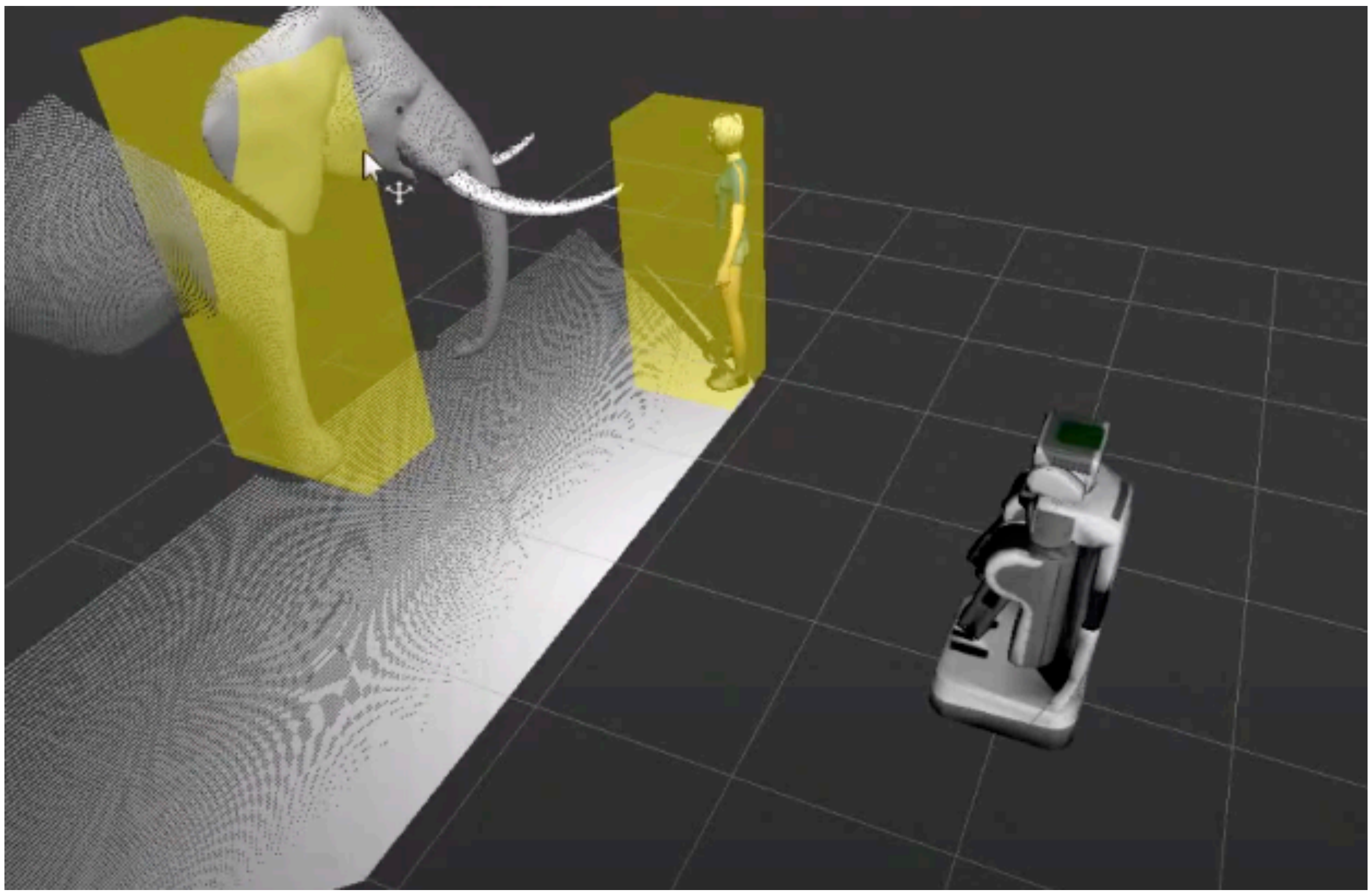
Using ROS bags for testing

ROS provides a way to record in a log file the full ROS API of a robot while running on a real life situation, and then run it back later on another computer. This log file is called a ROS bag. When you run a ROS bag on another computer, the computer will be showing to your programs the same ROS API that had when it was recorded.

You can learn how to use ROS bags here: record and replay of ROS bags (http://wiki.ros.org/rosbag).

ROSbags are a limited system in the sense that you can only use them for the case that you want to create algorithm that does something from sensor data. This means, you cannot generate commands to the robot because it is only a reproduction of data, so you can get the same data as the robot got when recorded, but you cannot decide new actions for the robot.

Using simulations for testing



So if you want to go pro without having to use a real robot, you should use simulations of robots. Simulations is the next step in software development. This is like having the real robot on your side but without having to care for the electronics, hardware and physical space. Roboticists consider simulations as the ugly brother of robotics. Roboticists usually hate to use the simulation because it is not the real robot. Roboticists like the contact with the real thing. But fortunately, we are talking here with the opposite kind of people, those who want to keep their hands off the hardware. For those, simulations are key.

Let me tell you one thing: even if roboticists do not admit, robot simulations are the key to intelligent robots. More on that in future posts, but remember you read this first here! (https://www.theconstructsim.com/simulations-key-intelligent-robots/)

In the case of simulations, you have a simulation of the robot running in your computer that you can run and act like the real robot. Once the simulation is running, your computer will present to your programs the same ROS API that you would had if you were in the computer of the real robot. That is exactly the same as in the case of the ROS bags, with the advantage that now you can actually send commands to the robot and the simulated robot will reply accordingly to your commands. That is awesome!

To use robot simulations, ROS provides the Gazebo simulator (http://gazebosim.org/) already installed. You only need to have the simulation of the robot you want to program for running. Usually, the companies creators of the robot provide already the simulation of their robot that you can download and run on the Gazebo simulator.

Installing simulations and making them run could be a little more of hassle. In case that you want to avoid all that work, I recommend you to use our ROS Development Studio that contains the simulations ready to be launched with a single click, as well as having ROS, IDE and other useful tools.

DEBUGGING

You will also need to debug the code. Most of the IDEs we have presented provide integrated debugging tools on them. However, you can also use all the standard debugging tools used for typical C++ code, including but not limited to:

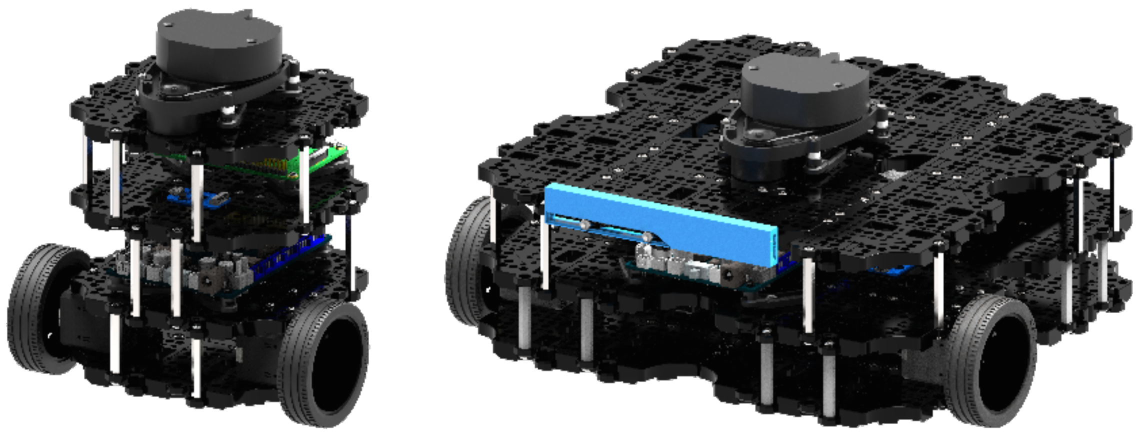
- GDB (http://sourceware.org/gdb/)
- Oprofile (http://oprofile.sourceforge.net/news/)
- Valgrind (http://valgrind.org/)

CHAPTER 6: LIST OF ROS2 ROBOTS TO PRACTICE WITH

At a certain point, you will need to start practicing with real robots. You can start with any of the following ones, which are reasonably accessible.

Turtlebot 3, by Robotis

This was the first commercial robot able to run ROS2 on it. Turtlebot3 was designed to run ROS1 by default, so if you buy it, it will come with ROS1 pre-installed. However, the guys from Robotis quickly prepared a tutorial about how to make your Turtlebot3 run ROS2. The full tutorial is here.



ROSbot, by Husarion.

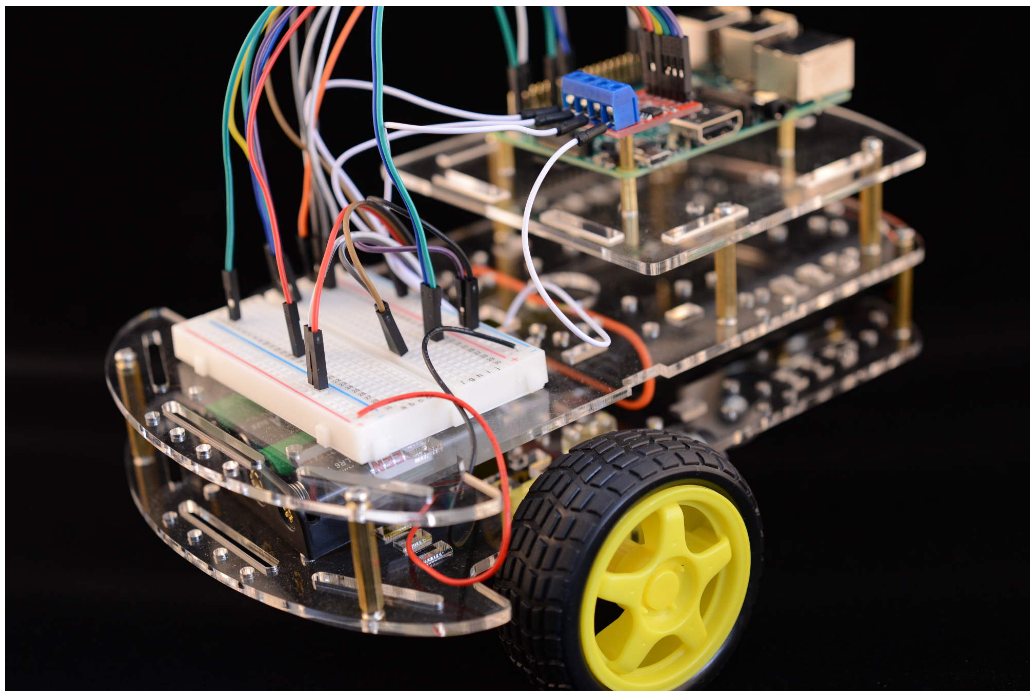
Again, this robot default system is ROS1 but Husarion has released a full tutorial about how to change the default ROS version to ROS2. The full tutorial is here.

Rover Zero, by Rover Robotics

Same procedure as the previous 2. Default system is ROS1 but Rover provides a git repo with the code and instructions about how to install the ROS2 drivers for that robot. Check it out here.

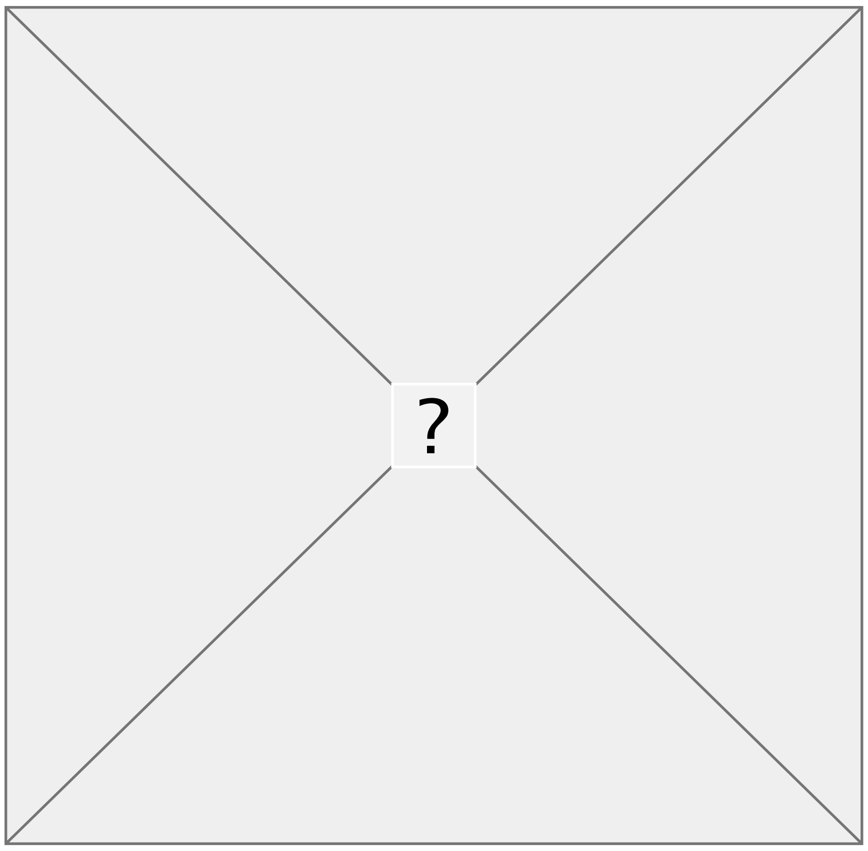
Hadabot, by Hadabot

This is the first ROS2 native robot I have seen. It is a very simple robot with the main purpose of learn ROS2 while applying it to the real robot. Basically, just a pair of wheels with encoders, more than enough for learning basic ROS2. Check it out here.



e-puck 2, by GCTronic.

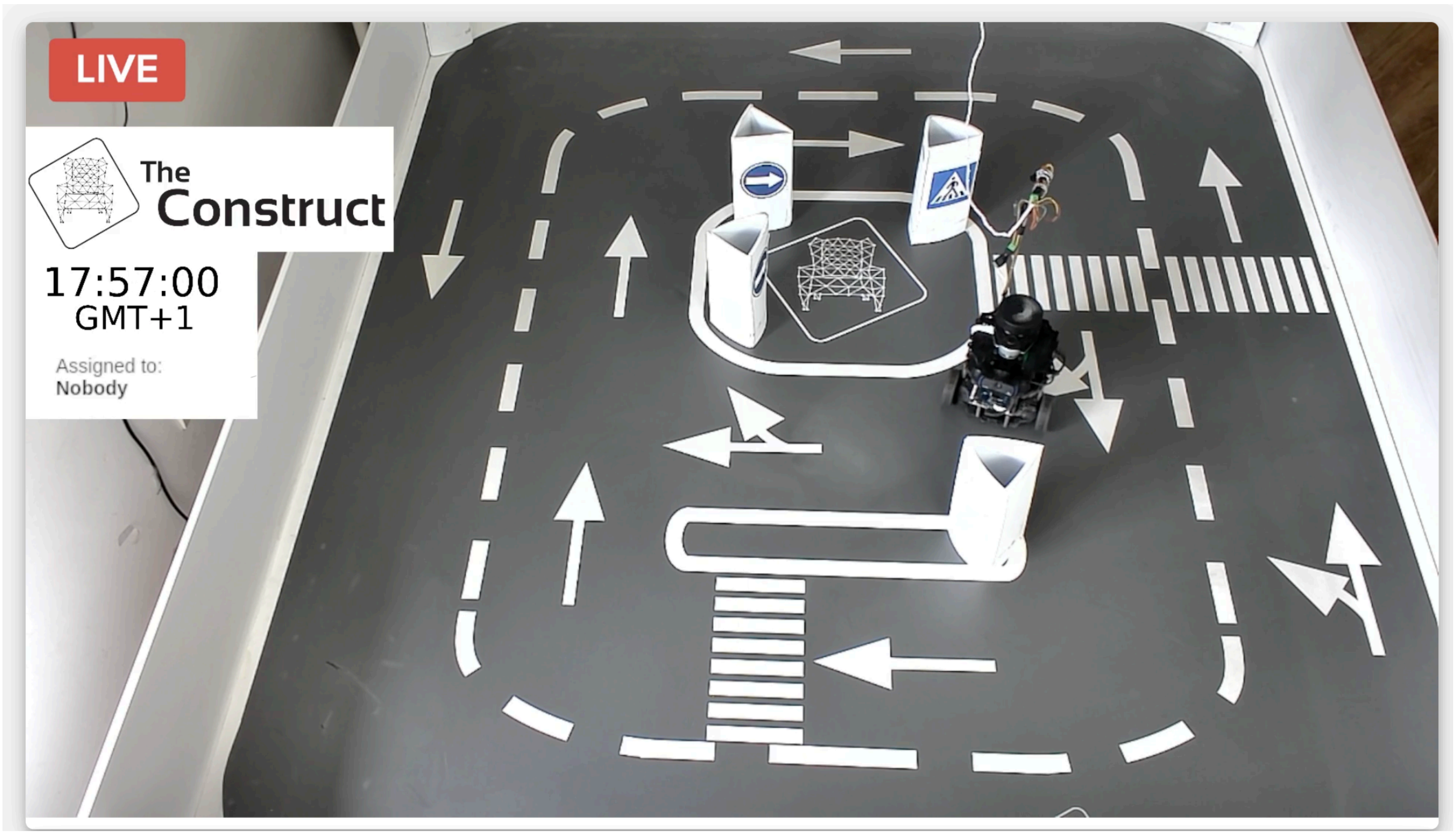
The e-puck 2 robot is a very small and cool robot with lots of sensors. The e-puck 2, from conception, it has nothing to do with ROS. However, recently Cyberbotics company created a ROS2 driver for that robot so we can now use this excellent robot with our favorite ROS. Here the drivers with tutorials.

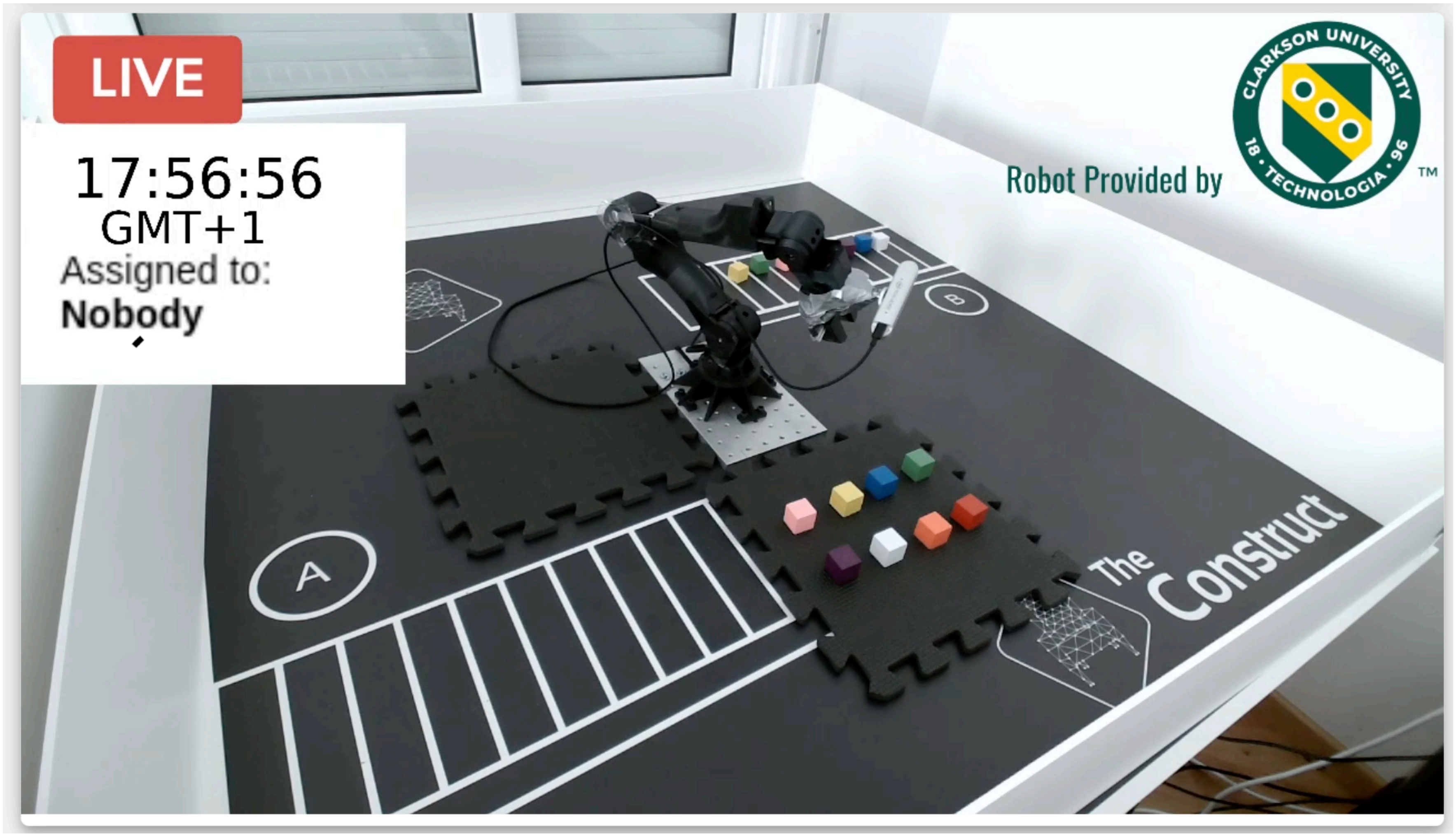

Open Manipulator, by Robotis

This is the only arm robot running ROS2 that I'm aware of. It is produced by the creators of the Turtlebot3 and, again, the robot boots initially with ROS1. However, the Robotis team have created a detailed tutorial about how to set this manipulator work with ROS2. **Find the tutorial here**.

Using a remote real robot lab

In case you cannot afford to buy a robot, you can use the remote real robot labs that we at The Construct have created for students a set of Remote Real Robot Labs, to which students can connect remotely and practice with real robots from their location. The labs include a wheeled robot and an arm robot, to practice all basic robotics concepts with real robots.





CHAPTER 7: ADDITIONAL LEARNING

Final step to become a robotics developer is to master some robotics related subjects.

Robots are living things, I mean, things that move around. That entails all a series of new concepts that you need to learn (your program is not going to stay on a fixed place anymore). I'm sure that as a software engineer you already have understood many of the subjects above because they closely relate to the normal job of a software developer. However, there are still some concepts closely tied to robotics that a software engineer needs to master (in the same sense as if you need to program for invoices, you need to understand some concepts of accounting).

What follows is a list of concepts that you need to understand to be able to create programs for robots. I have included a link to online courses that teach those subjects:

- Basic mobile robots kinematics (https://app.theconstructsim.com/#/Course/46)
- Robot arm kinematics (https://app.theconstructsim.com/#/Course/51)
- Robot body dynamics (https://app.theconstructsim.com/#/Course/49)
- URDF robot model description (https://tinyurl.com/y2wrjmvx)
- Robot navigation (including SLAM, localization, path planning and obstacle avoidance) (https://tinyurl.com/y68qso54)
- Robot manipulation (https://app.theconstructsim.com/#/Course/66)
- What is a frame reference system (https://tinyurl.com/y5en2dtf)
- Robot joint control (https://tinyurl.com/y6h4s4oz)
- Robotics perception (https://tinyurl.com/y6ycvcfo)
- Artificial intelligence: that is the key to create intelligent robots, combined with good programming skills:
    - Machine Learning (https://app.theconstructsim.com/#/Course/47)
    - Deep Learning Basics (https://app.theconstructsim.com/#/Course/71)
    - Reinforcement Learning (https://app.theconstructsim.com/#/Course/68)

APPENDIX

Here you are going to find additional information about how to program for ROS. Those are additional resources that can make you learn and improve your ROS level as a developer.

- The ROSCON, the official annual ROS conference created by the creators of ROS, the Open Source Robotics Foundation. https://roscon.ros.org

- The ROS Developers Conference (http://www.rosdevcon.com). This is an annual online ROS conference created by The Construct for ROS developers where the attendants practice at the same time that the speaker is explaining. You can watch the presentations of previous editions at The Construct Youtube channel

- The ROS Developers Podcast. A weekly podcast with insights about ROS from real ROS experts around the world. (http://www.theconstructsim.com/category/ros_developers_podcast/)

- ROS Best practices. A series of online documents that will help you create better ROS code. (http://wiki.ros.org/BestPractices)

- ROS C++ Style guide. (http://wiki.ros.org/CppStyleGuide)

- The Construct public repo of **robot simulations** (check the branch name for different ROS versions). (https://bitbucket.org/account/user/theconstructcore/projects/PS)

- Online shop called ROS Components. (https://www.roscomponents.com/en/)

- ROS.org official tutorials for learning ROS (wiki.ros.org/ROS/Tutorials)

- How to build a ROS based robot, **series of Youtube videos. (**https://tinyurl.com/yxtmfmqj**)**