GIANLUCA BARDARO

ROBOTS, SOFTWARE AND OTHER STUFF

bip, bop.

— A robot

ACKNOWLEDGMENTS

Thanks all.

ABSTRACT

Some kind of abstract

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

1 INTRODUCTION

Robotics popularity is growing everywhere. Not only in the academic world, where a lot of research is now applied to robotics, but also in the industrial world, where new companies are providing commercial solutions involving robots. Even the general public is now more used to a society where robot coexist and collaborate with humans. The first model of iRobot Roomba was introduced in 2002, this means that today there are teenagers that only know a world where robots as part of the household are the norm.

The current evolution of robotics as a field is similar, in a way, to the growth in popularity of mobile phones, first, and smartphones, later. Originally, the idea of a personal wireless communication system was only possible in science fiction, then scientific progress and new technology made it possible. At fist only for very few applications, i. e., the military, the railroad system, but later it grew exponentially and today it is part of our everyday life. Many factors made this leap possible: first of all, technological advancements, like miniaturisation, battery life extension, increase in display quality, cheaper computational power, additionally, a sense of need, people felt that a mobile phone was a great addition to their life, lastly, standardization, shared communication platforms, accessible development environments, and multiple abstraction layers.

The same was for robots, originally no more than toys, mechanical puppets and mysterious automata. They existed,

as truly autonomous agents, only in the minds and works of writers and directors, and even today we are not able to match those visions. As soon as technology made it possible, the first autonomous arms were developed. Initially applied to heavy industry to replace human in dangerous and highly specialized tasks, later, technical refinements and functionality extensions made them suitable for healthcare and the military. From here it was an explosion of different technologies, shapes and applications. Autonomous arms evolved in precision, power and dexterity, from the massive industrial arms, to the agile surgical robots. Soon after the development of the first complex arms, many researchers tried to realize the vision of a full humanoid robot, but, even today, after many progress we are not able to fully replicate the complexity of the human body. Mobile platforms were the next logical step, robots able to autonomously explore and navigate the environment, robots able to reason on what they detect and to react accordingly.

In the last two decades, robotics have been applied in numerous fields and robots assumed a myriad of shapes and functionalities. In industry, robots are used for welding, painting, drilling, cutting, handling dangerous materials, moving heavy objects, pick-and-place, inventory management. In healthcare, today, surgical robots are the norm, but advancement in soft robotics made robots suitable for rehabilitation and elderly care. Most of the recent discoveries of planetary science exist thanks to rovers, autonomous mobile robots that can, unassisted, explore the surface of planets, asteroids and comets. Moreover, maintenance in outer space is extremely dangerous for humans and often impossible, only robotic arms and autonomous probes can perform them. Back on Earth, in our houses and cities, robots are not an unusual sight. There are robotic vacuum cleaners and lawn

mowers, autonomous robots deliver packages directly to the front door of the house and self-driving public transportation is a reality in various cities. Fully autonomous cars are still only prototypes, however not because of technological limitations, but mostly for economical, social and legal reasons. Thanks to the recent progresses in human-robot interaction, the sight of a robotic waiter or concierge, while marvellous, is not completely unexpected. Lastly, unmanned autonomous vehicles (UAVs), e. g., off-road vehicles, drones, boats, submarines, have been used successfully in search and rescue missions and to operate in environments dangerous or unreachable by humans, like volcanos, mountains, disaster zones, contaminated areas.

In this brief history of robots, most of the progress and technological advancements seems related to hardware. More responsive motors, more precise and reliable sensors, cheaper electronic and computational power. All these advancements contributed to what is robotics today. However, software has always been one of the main concerns of any roboticist. The implementation, the logic, is what makes the difference between a mechanism and an intelligent robot. Since their inception, robots have spawned a series of software solutions to implement their functionalities. For example, the Stanford Research Institute Problem Solver, better known by it acronym STRIPS, is an automated planner developed for Shakey that became the foundation of modern action languages. Modern robots have software architecture far more complex than Shakey, they coordinate multiple sensors and actuators, moreover they implement different functionalities and are expected to operate in real time. This is why, more recently, i. e., the last twenty-five years, a lot of efforts in robotic software revolved around the design of a solution to streamline and simplify the development process.

The answer was the introduction of robotic middleware and framework and to rely on component-based designs. This approach fits perfectly the necessities of robotics, components encapsulate functionalities and promote reusability, while a pre-defined communication layer free the developer from the burden of micromanaging the low-level interactions. After the first wave of ad hoc implementations, few frameworks rose in popularity and become standard defacto for robotic software development. Today, depending on the specific application, a developer can choose various framework or middleware: OROCOS (or its derivation RoCK), for hard real-time application, SmartMDSD, for a more complete and structured development environment, YARP, for a more light-weight and data-centric approach, or ROS, for more extensive support and development freedom.

Middlewares and frameworks have fuelled the progress of robotic systems, creating the current scenario of robot design and development. Thousand of components are already available to anyone who wants to implement his own robot and experts can setup the most common functionalities (i. e., teleoperation, mapping, indoor localization and navigation) of a new system in a matter of days. However, the learning curve to reach this kind of expertise is quite steep and extending the functionalities of a robot beyond what is currently available requires a considerable effort not strictly related to the new functionality itself. By doing again the parallel between robotics and smartphones, we are currently in robotics in the same situation developers were before the standardization introduced by Android. Today, an Android developer can bootstrap and deploy a new application on millions of devices in few steps, thanks to abstraction layers that separate the development environment to the underlying hardware and operating system and thanks to advanced

design, development and simulation tools. Of course smartphones are not robots, while there is a great variability from one device to another (e. g., screen size, quality and number of cameras, sensors availability, type of mobile network, etc.), they cannot be compared to the incredible range of sensors, actuators, shapes and functionalities that exists in robotics. For this reason, while robotics can aim to achieve the same streamlined development of smartphones, the approach needs to be different.

1.1 MOTIVATIONS

Middlewares and frameworks created the present development environment of robotics, but current approaches are not suitable any more for a constantly advancing robotic field. The personal experience of a all-around robotic expert still drives robotic software design and development. When developing a new robotics system or application it is expect that a developer has total expertise on the low-level functionalities provided by the underlying framework and the high-level functionalities to be implemented; while this was possible in the past, it is an unsustainable approach today. Not only it is necessary to create a distinction between different roles in the design and development process of a robot, but is is also necessary to provide to these roles the right tools to fulfil their tasks.

The *system designer* needs tools to outline the architecture of the system and describe the high-level interactions and requirements of components. This can be achieved using a modelling language to describe components and their inner workings in an agnostic way with respect to the underlying framework. This approach, not only provides the right

environment for the designer, but it also provides early detection of errors, an architectural overview of the system and system-level reusability.

The *component developer* should focus only on the implementation of the internal logic and not on the structure of the component itself, since this is the role of the designer. To do so, the component developer needs an environment that abstract from the framework-related boilerplate code and provides a contained development space. Potentially, the logic implemented should be portable from one component to another, even if they are not based on the same framework, given they share the same design principles. Building on top of the modelling language used by the system designer, it is possible to achieve the ideal development environment by delegating to an automatic code generator most of the boilerplate implementation, and by defining a bounded reference component that can be used by the component developer as a starting point.

The *application developer* implements high-level functionalities, that should be independent from the underlying architecture of the robot. In practices, this means it should exist an abstraction layer between the low-level capabilities provided by one or more components and the high-level applications. There is a plethora of robots, with different configurations and implementations, however it is possible to abstract most of the capabilities independently from the system. An example could be teleoperation: by defining linear and angular velocity of the mobile platform it is possible to control any robot, independently from their physical configuration. Using these general interfaces the application developer should be able to implement high-level functionalities for multiple robots with minimal modifications. In order to achieve this it is necessary to define the concept

of capabilities, to identify them in a robot architecture and to provide a framework-independent way to interact with them.

1.2 THESIS CONTRIBUTIONS

Our proposed approach revolves around two key factors: formalise the design and development of robotic software and streamline the implementation process for the different kind of experts involved. In order to achieve this we developed a toolchain.

Part I

IT BEGINS