SLIM ROBOTICS: ROBOTICS IN A SMALL TEAM WITH SPACE REQUIREMENTS

Virtual Conference 19–23 October 2020

Edmundo Ferreira*, Aaron Pereira, Andrei Gherghescu, Levin Gerdes, Lukas Hann, Thomas Krueger

Human Robot Interaction Laboratory, ESA-ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, Netherlands
Email: first_name.last_name@esa.int

ABSTRACT

In this paper, we present the rationale behind our development philosophy and explain which elements we adopted from general software engineering practices of industry, the open source community, embedded, and cloud technologies. We identify aspects of development practices and consider the transition from prototype to product and the difficulties the use of frameworks entails. Along with the explanations, examples, and use cases given, we will relate directly to a payload development for the International Space Station (ISS) with all the intricacies of qualification such as external requirements, and constraints. We hope to add to the scope of roboticists and software/framework developers alike by showing how we developed and improved robotics systems software from early prototypes to production grade-maturity.

1 INTRODUCTION

The Human Robot Interaction (HRI) laboratory is an engineering research lab at the European Space Agency (ESA) that supports research, development, and validation of human-robot interaction technology, with a primary focus on telerobotics and haptics. The laboratory's goal is to support novel spaceflight projects by developing and demonstrating critical technologies [1]. The HRI lab has conducted several experiments on board the ISS within the Multi-Purpose End-To-End Robotic Operation Network (METERON) project [2, 3]; Haptics-1 (2014) [4], Haptics-2 (2015) [5], Interact (2015) [6, 7, 8], and most recently Analog-1 (2019) [9, 10].

In all these experiments, and in our daily activities of the laboratory, we (like many other institutes and companies in the field of robotics) face challenges from several areas of expertise. Some predominant domains are mechatronics, telecommunications, control, and software development. In this paper, we mainly focus on the software aspects of the robotics systems and the burden they impose on small teams. Most of the opinions expressed here are based on the experience accumulated over several years. These opinions stem from the lessons learned when updating the systems from one experiment to another and when required to extend our robots' capabilities.

In Section 2, we briefly mention some of our most important requirements for the latest flight experiment. In Section 3 we outline five hypotheses that we believe will always arise in one form or another when developing software for robotics systems in a small team. We then discuss these hypotheses and some of their implications and consequences. In Section 4, we detail some of the decisions made at subsystem/component level, taken in accordance with the hypotheses we introduced. Conclusions of our approach are detailed in Section 5.

2 REQUIREMENTS & CONSTRAINTS

We state the requirements and constraints of our ISS experiments as they provide context for the hypotheses introduced in Section 3. Here, we present the requirements for our most recent experiment, the Analog-1 mission, as it surpasses all of the laboratory's previous missions in scope. The development of the prior missions have been a gradual increase in complexity, that culminated in the current system. The current system, as used in the Analog-1 mission, includes either subsystems or components from previous experiments. These requirements encapsulate not only the most recent mission but also most of the HRI laboratory's past ISS experiments.

The entire mission system can be divided into two parts: the subsystem on-board the ISS which was used by the astronaut to give commands and receive feedback, and the rover subsystem on earth [9]. We now enumerate some of the mission's key requirements and constraints.

R0 The mission centered around rover navigation and exploration, as well as geology sampling. Concrete tasks mainly focused on teleoperation

*corresponding author, co-authors are listed in alphabetical order

and telemanipulation and limited autonomy features.

R1 Short time to mission, by ESA standards. Initial mission concept and proposal (Experiment Science Requirements Document (ESR)) to actual mission of ∼18 months.

R2 Small team; (1 mechatronics engineer, 1 industrial engineer, 3 software engineers, 1 principal investigator).

R3 Re-use of already upmassed components onboard the ISS: HP ZBook laptop for cost minimization.

R4 Extensive usage of Commercial Off-The-Shelf (COTS) components, including but not limited to: KUKA Light Weight Robot (LWR), Sigma7 [11], and Ambot rover platform [12].

R5 Ability to integrate systems developed by Institute of Robotics and Mechatronics of the German Aerospace Center (DLR), specifically the KUKA LWR [13] robot manipulator, and time-domain passivity control algorithms required for safe operation. This flexibility to integrate external software algorithms and modules in our systems, was mainly a requirement due to limitations of the robot's Original Equipment Manufacturer (OEM) software.

R6 Real-time capabilities with hardware in the loop. Although many of the hard real-time constraints are run in firmware, we required a system capable of handling periodic tasks of about ∼1 ms with bounded jitter. Such requirements are especially important for haptics control loops and for the robot control loops.

R7 Security restrictions on the Operating System (OS) and applications on the final software image. This is often described as "hardening" the software system and is a requirement of the ESA Security Board. Hardening implies that the system must be scanned automatically by Security Content Automation Protocol (SCAP) tools as defined by the Center for Internet Security (CIS). Depending on its version, each application used in the root filesystem must be searched for known Common Vulnerabilities and Exposures (CVE) and addressed. Hardening also implies the configuration of specific and quite restrictive firewall and network settings. The hardening requirement applies to any computer system on-board or any on ground segment device that "interacts directly" with the ISS payload.

R8 Constrained communication between ISS and rover (2 Mb/s bandwidth, ∼850 ms round trip delay, and up to ∼4 % packet loss [5]).

R9 Reliable operation with minimum astronaut intervention. Ability to update the system on board with automated Over The Air (OTA) updates as astronaut time on ISS is an extremely valuable resource. Also due to **R1** it would have been impossible, by our estimates, to deliver the software with a traditional approach where the entire software systems would need to be completed up to 6 months before the rocket launch delivering the experiment.

3 PROBLEM DESCRIPTION

From the requirements specified in Section 2, it is a given that the development team needs to operate on the entire software stack, from the robotics software down to the OS level. The lessons learned from all our past experiments led us to postulate the following hypotheses that we believe apply to any small team which designs and implements robotics software from the ground-up:

H0 There is neither a silver bullet [14], nor a one size fits all solution.

H1 Choose your community or take ownership.

H2 All updates have hidden costs.

H3 Small teams cannot afford dedicated roles.

H4 Fundamental knowledge is reusable.

These realizations might seem completely obvious and trivial to produce any useful guidelines. However, when reflecting upon our past decisions, it showed that they have taken us down an unusual path, and produced some indirect and unanticipated solutions. For example, we borrowed some solutions from cloud computing, embedded software, and IoT, which are not typically applied to mainstream robotics. These hypotheses highlight what we believe to be frequently overlooked issues; faced not only by roboticists in small research teams but also by startups in the robotics landscape of today. The following paragraphs describe the hypotheses in detail.

H0 The complexities of modern robotics software systems are unlikely to be fulfilled by any one tool, library or framework without the need of external dependencies. These external dependencies may be other tools, libraries or even people. The search for

a framework that single-handedly satisfies all our requirements and constraints, without major modification, continues to this day. We firmly believe that if your set of requirements is greater than zero, and your project has even the simplest goal, a few libraries, frameworks, OS's, and workflows will need to be chosen, combined, and eventually tailored.

H1 Given the need for software dependencies, it is critical to minimize them, and to have a judicious selections of the communities that maintain them. When investigating which communities fit the project best, the following key criteria weigh in the most on our decision: a track record of dependability, fast pace, responsive attitude, and a like-minded approach to software. After the selection and a trial phase, serious investment into knowing the details of how these dependencies function at a fundamental level is crucial for a successful integration. However, there may be cases where the external solution or the community engagement do not align with mission requirements, deadlines, or development principles. In these cases, taking ownership and developing an internal project to address your needs might prove to be a better and faster solution. Although an internally owned solution may lack features, the benefits of being able to quickly fix bugs, gain an understanding, and increased reliability outweigh those shortcomings.

H2 The tasks of maintaining and updating the systems at any level is unavoidable for most robotics teams. In our experience, a recurring pattern emerged at the adoption of any new dependency. An initial period of big gains is eventually followed by a large workload needed at the moment that updating is crucial. Typically, the task of updating dependencies tends to be met with two attitudes: ignore or undertake. The "ignore" approach, i.e. updating infrequently, has the drawback that when an update is finally unavoidable, the dependencies of the dependencies, which need to be updated themselves, result more often than not in an unavoidable update of the entire software stack. Additional work might also be required to reconcile the new update with the previous configuration of the system. At this moment the initial payoff from the adoption of the external dependency vanishes. We believe the hidden cost of the update is shown at this singular moment. The alternative update approach consists of continuously engaging in the update tasks. In this model, if the dependency is well established, the increments will be much smaller and "digestible" by a small team. This model tends to generate frequent and small updates, where second tier dependency problems are usually more contained and can be promptly resolved. Here the hidden cost of the update is spread out over time. Another important factor in decreasing the team's workload, is their ability to automate the workflows that generate a new system. In the first model some automation could be done but since it's sparsely executed it also tends to break compatibility and not generate the desired result on the following update cycle. However on the fast update model, the automation can more easily be maintained and exercised. This concept of Continuous Integration and Continuous Deployment (CI/CD) has gained traction in general software development and is prevalent alongside modern version control systems.

H3 From past experiences, and from collaborations with other institutions, we observed a clear distinction between two types of engineers who contribute to robotics systems. On one hand there are software engineers who focus on the higher level algorithms for robotics and are exclusively users of the chosen framework. On the other hand there are software engineers who maintain, improve, and update the operating system and all of its dependencies and frameworks. Big companies or big departments may not view this as a problem, because their pool of engineers is large enough to address software issues from both areas. In small teams, however, this becomes a problem as the team cannot afford to choose between working solely on the high level algorithms with complete disregard of the system level and dependency updates. In a small team, we cannot afford the luxury of choosing between value-adding robotics features, and robustness and stability of the underlying operating system and system libraries.

H4 In many situations it has become evident that a lack of fundamental understanding of basic principles leads to catastrophic assumptions about how a system functions. This is especially true for adopted dependencies where a lack of knowledge investment can lead to a situation of complete dependency or blind trust. Studying the fundamentals may be relatively slow, difficult, or expensive, but the benefits of doing so manifest in an easy transfer of knowledge between different layers of the software stack, OS, libraries, frameworks, robotics applications and thereby creates a virtuous cycle. Additionally, this brings educational benefits to trainees and interns who work in our laboratory, which they can reuse in their future projects with other employers.

The following section describes the typical systems developed by the HRI laboratory and the current

structure of the robotics systems and deployment workflow.

4 SYSTEM DESCRIPTION

Previous HRI experiments involved teleoperation tasks with multi-DOF robotic manipulators on a rover driving on moon-like terrain. Our current teleoperation setup comprises a master control station, a slave robot which executes the master's commands for manipulation and grasping tasks, a wheeled platform for mobility, dedicated hardware for video processing and streaming, and optionally a dedicated communications node. In a typical ISS experiment, any infrastructure-connected device needs to have a hardened OS and the payloads need to meet specific security requirements. All these systems need to be constantly maintained and upgraded for each new experiment. The next section describes two alternative ways of arriving and maintaining such hardened operating systems.

4.1 Operating System

In most cases, the selection of a framework determines the selection of the OS itself (or the flavor of the Linux distribution), including system dependencies. In order to meet mission requirements, this system will need to be heavily customized to available packages, system level configurations, firewalls, etc. to meet mission objectives as well as security requirements for the production environment. This culminates in a "golden image", a result of manual alterations of the original system which transformed from generic to specific. If at any point in time a need arises to update either the framework or the entire OS in order to upgrade the kernel or address any well known CVEs, an almost insurmountable (to small teams) amount of work is required to reconcile the new update with the previous configuration.

The above scenario is not hearsay, but rather a cautionary tale from experience. All of the HRI laboratory's experiments before Analog-1 have been performed using a carefully tuned Debian[1] Linux image running the Xenomai[2] kernel framework. With every new experiment the team's effort switched from robotics system development to OS maintenance work, upgrading Debian and Xenomai, in most updates from scratch, leading us to our views expressed in **H2** and **H3**.

With the Analog-1 experiments on the horizon, it was decided to invest in fundamental knowledge and automation in favor of manual tuning, see **H4**. Inspired by the embedded community, the Buildroot[3] system was adapted to build a custom Linux distribution with all of the requirements (real-time patches and package hardening) in place. Even though the initial amount of effort for this approach was higher than for customizing an existing Debian image, it yielded a substantial return on investment. With a full DevOps cloud infrastructure setup in place, our OS images are automatically built on every change and in the span of few hours they are readily available for deployment on all robotic platforms, embodying CI/CD practices. To be clear, the current system still requires frequent updates, but a substantial amount of work is automated (see **H2**), leaving the developers free to focus on other aspects of the infrastructure and robotics software.

Lastly, another welcomed side-effect of using a technology such as Buildroot is the introduction of toolchains and Software Development Kits (SDKs) into our workflow. This allows all robotics software engineers to implement software for the robotics systems in a rather straightforward manner with respect to their development environment, hence the statements in **H3** and **H4**. Although the usage of toolchains and SDKs is ubiquitous in other software engineering fields, it is our belief that it is not leveraged sufficiently in the field of robotics.

4.2 Deployment and Updates

There are two commonly known ways of making a custom OS available to a device. The first one is shipping a device running the OS, and the second one is installing it remotely. While intuitively a remote installation appears easier, in practice most payloads are shipped to the ISS because this requires no crew time.

As payload developers, our choice was further complicated. When physically shipping the system, the software and OS hardening need to be completed early (software "freeze"), approved by the Software Authentication Board, and installed on flight hardware, before it can be "literally" shipped to the ISS. This process requires all software to be completed months ahead of the experiment date. However, our preferred method of delivering our system was a remote installation, as it allowed us to continue to develop, update, and test the software much closer to the date of the experiment. This established the need for a remote, unattended installation on board the ISS.

With its specific requirements, ESA's ISS Columbus module networking infrastructure imposed

[1] https://www.debian.org/distrib/

[2] https://gitlab.denx.de/Xenomai/xenomai/-/wikis/home

[3] https://buildroot.org/

unique challenges to remote software updates and installation. Specifically, any remote installation method has to operate with the following restrictions: no Dynamic Host Configuration Protocol (DHCP), no Trivial File Transfer Protocol (TFTP), no Preboot Execution Environment (PXE) support, and mandatory SSH File Transfer Protocol (SFTP). These are non-standard restrictions in *bare-metal* deployment infrastructure, which reinforced our belief in **H0**.

The solution, in line with **H1**, was to develop our own remote deployment tool compatible with restrictions imposed by the Columbus module. Inspired by the cloud infrastructure development community, and based entirely on Golang[4], our in-house tool *Golang ESA Provision Image Controller* (GoEPIC) is a Linux Text-based user interface (TUI) application. Under nominal circumstances, the tool requires only that the device is powered on, and a single user input, an Enter keystroke. The latter requirement can even be bypassed if *ssh* is allowed and available.

The added benefit of taking ownership and developing such a *bare-metal provisioning* tool is, as per **H4**, the reuse of gained knowledge. For all the HRI infrastructure, be it rover or flight hardware, the tool is constantly used for fresh OS deployment or updates. Our updating mechanism is based on the A/B system updates principle[5] used by Android and DevOps for cloud-infrastructure. This allows us to deploy a new OS image and revert back to a previous installation in a matter of seconds.

4.3 Robotics Systems

Since we are working on one-off projects, we try to incorporate as many off-the-shelf components as possible to save time and reduce costs. However, many consumer or industry-grade components that seem to meet our requirements are unsuitable because we want to operate them in a different way than intended by the manufacturer. For example, the robotic manipulators (KUKA LWR) that we use because they can sustain the necessary loads are designed for industrial applications that demand high precision, repeatability, and high speed. In contrast, our projects are focused on adaptability and dynamic control, but have similar mechanical requirements to the industrial tasks. We found that the OEM software limited the capabilities of the robots, which restricted them from operating in accordance with our requirements. Furthermore, the control boxes were too bulky and heavy to fit in our mobile platform and have high power consumption, as they were designed to be stationary. Therefore, we took advantage of our partnership with DLR who designed the robot arms and licensed the technology to the OEM, but had their own control system which is more suitable to our requirements. This cooperation greatly enhanced our robotics capabilities (**H1**).

[4] `https://golang.org/`

[5] `https://source.android.com/devices/tech/ota/ab`

5 CONCLUSIONS

In this paper, we have presented our experiences with space robotics for a small team of engineers over several years and projects and how these experiences, lessons learned, and requirements have shaped our approach to software engineering practices.

To be concise, we have distilled these findings into five hypotheses and provided examples for each of them. Of course, there is some overlap and our hypotheses, or beliefs, do correlate with each other in most cases. Of course, **H0** (there is no silver bullet) also holds true for these principles of ours. Nevertheless, we hope that our findings can be of use to other engineers in similar situations and spark some insightful discussions.

Acknowledgment

All of these developments and insights are not created in a vacuum. We want to thank our project partners of the directorate of Human and Robotic Exploration for project cooperation and to enable us to apply these developments in real mission contexts. In addition, our gratitude goes to DLR's Institute of Robotics and Mechatronics for the longstanding and close cooperation, which above all means exchange of ideas, co-development of technologies, and accomplishing projects together.

List of Abbreviations

CI/CD Continuous Integration and Continuous Deployment

CIS Center for Internet Security

COTS Commercial Off-The-Shelf

CVE Common Vulnerabilities and Exposures

DHCP Dynamic Host Configuration Protocol

DLR Institute of Robotics and Mechatronics of the German Aerospace Center

DOF Degree(s) of Freedom

ESA European Space Agency

ESR Experiment Science Requirements Document

HRI Human Robot Interaction

IoT Internet of Things

ISS International Space Station

LWR Light Weight Robot

METERON Multi-Purpose End-To-End Robotic Operation Network

OEM Original Equipment Manufacturer

OS Operating System

OTA Over The Air

PXE Preboot Execution Environment

SCAP Security Content Automation Protocol

SDK Software Development Kit

SFTP SSH File Transfer Protocol

TFTP Trivial File Transfer Protocol

TUI Text-based user interface

References

[1] Dr Jessica Grenouilleau (HRE-XSR), Phillipe-Schoonejans (HRE-XSR), Giorgio Magistrati (HRE-XE), Dr Thomas Krueger (TEC-MMA) and Kjetil Wormnes (TEC-MMA) (2020) *The contributions of the Human-Robot Interaction Laboratory to the past, present, and future exploration programs*. White paper under review, European Space Agency.

[2] European Space Agency (2019) . METERON Project. `https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Automation_and_Robotics/METERON_Project`. [Online].

[3] Schmaus P, Leidner D, Krueger T, Schiele A, Pleintinger B, Bayer R and Lii NY (2018) Preliminary Insights From the METERON SUPVIS Justin Space-Robotics Experiment. In: *IEEE Robotics and Automation Letters*, 3(4):pp.3836–3843.

[4] Schiele A, Aiple M, Krueger T, van der Hulst F, Kimmer S, Smisek J and den Exter E (2016) Haptics-1: Preliminary Results from the First Stiffness JND Identification Experiment in Space. In: F Bello, H Kajimoto and Y Visell (Editors), *Haptics: Perception, Devices, Control, and Applications*, Springer International Publishing, Cham. ISBN 978-3-319-42321-0, pp.13–22. doi:10.1007/978-3-319-42321-0_2.

[5] Schiele A, Krueger T, Kimmer S, Aiple M, Rebelo J, Smisek J, den Exter E, Mattheson E, Hernandez A and van der Hulst F (2016) Haptics-2 — A system for bilateral control experiments from space to ground via geosynchronous satellites. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp.000892–000897. doi:10.1109/SMC.2016.7844354.

[6] European Space Agency (2015) . Meet ESA's Interact Rover. `https://www.esa.int/ESA_Multimedia/Videos/2015/09/Meet_ESA_s_Interact_Rover`. [Online; posted September 2015].

[7] European Space Agency (2015) . Andreas Mogensen Controls Ground Rover From Space. `https://www.esa.int/esatv/Videos/2015/09/Andreas_Mogensen_controls_ground_rover_from_space`. [Online; posted 7 September 2015].

[8] European Space Agency (2019) . Analog-1 Interact rover. `https://www.esa.int/ESA_Multimedia/Images/2019/11/Analog-1_Interact_rover`. [Online; posted November 2019].

[9] European Space Agency (2019) . Astronaut Luca feeling the force, to advance rover control. `https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Astronaut_Luca_feeling_the_force_to_advance_rover_control`. [Online; posted 29-November-2019].

[10] Krueger T, Pereira A, Ferreira E, Gherghescu A, Hann L, den Exter E, van der Hulst F, Gerdes L, Cencetti L, Singh H, Panzirsch M, Hulin T, Balachandran R and Lii N (2020) Designing and Testing a Robotic Avatar for Space-to-Ground Teleoperation: The Developers' Insights. In: *Proc. International Astronautical Congress*. To appear.

[11] Force Dimension (2020) . Sigma7. `https://www.forcedimension.com/products/sigma`. [Online].

[12] American Robot Company (2020) . GRP 4400. `http://www.ambot.com/ip-wheel.shtml`. [Online].

[13] KUKA (2019) . KUKA LWR 4+ Brochure. `https://www.coboticsworld.com/wp-content/uploads/2019/05/KUKA-LWR-Brochure.pdf`.

[14] Brooks (1987) No Silver Bullet Essence and Accidents of Software Engineering. In: *Computer*, 20(4):pp.10–19.