An Experience Report on Challenges in Learning the Robot Operating System

Paulo Santos[1,2], Miguel Tavares[1], Ricardo Cordeiro[1], Alcides Fonseca[1], Christopher S. Timperley[2]

[1] LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

[2] Institute for Software Research, Carnegie Mellon University, United States of America

reliable software systems

Motivation

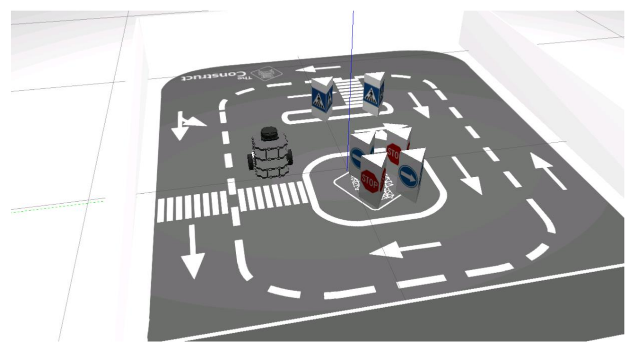
The **Robot Operating System** (ROS) allows developers to build valuable robots by configuring and reusing off-the-self-components. However, despite the advantages, the lack of documentation can present a challenge to novice users.

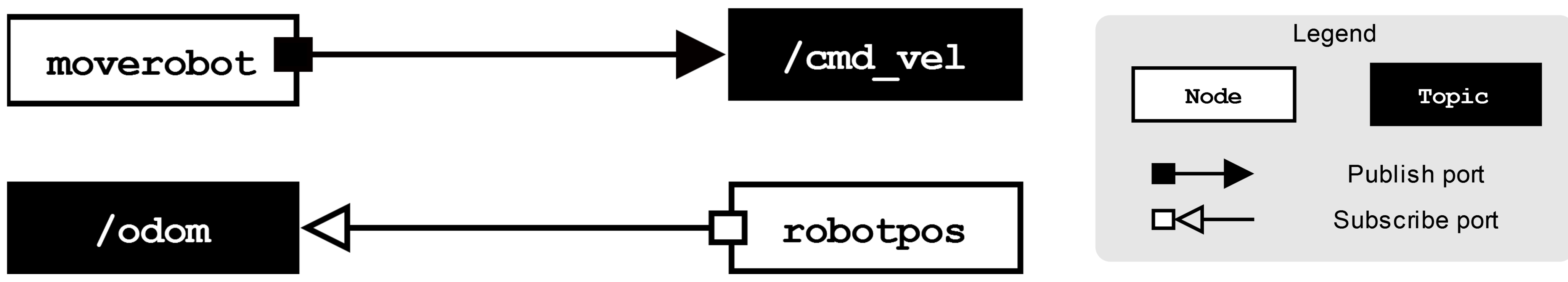
This work aims to identify what **challenges do newcomers to ROS face, to improve the development experience**.

The detection of the most frequent and time-consuming challenges can guide the development of approaches to improve the usability and correctness of ROS systems.

The Robot Operating System

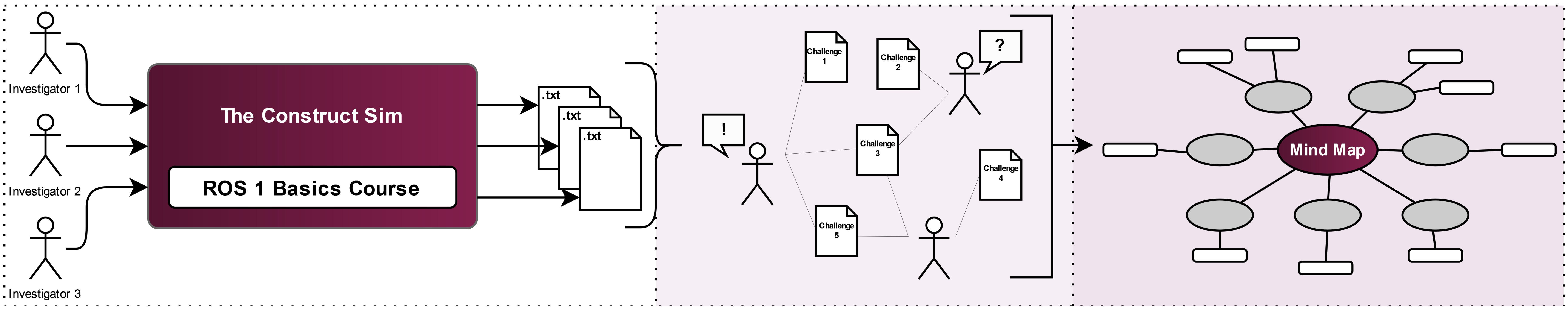
ROS allows different software components to exchange the information through **messages** in a **Publisher-Subscriber model**. Nodes communicate between each other by publishing and subscribing to messages to topics.





Methodology

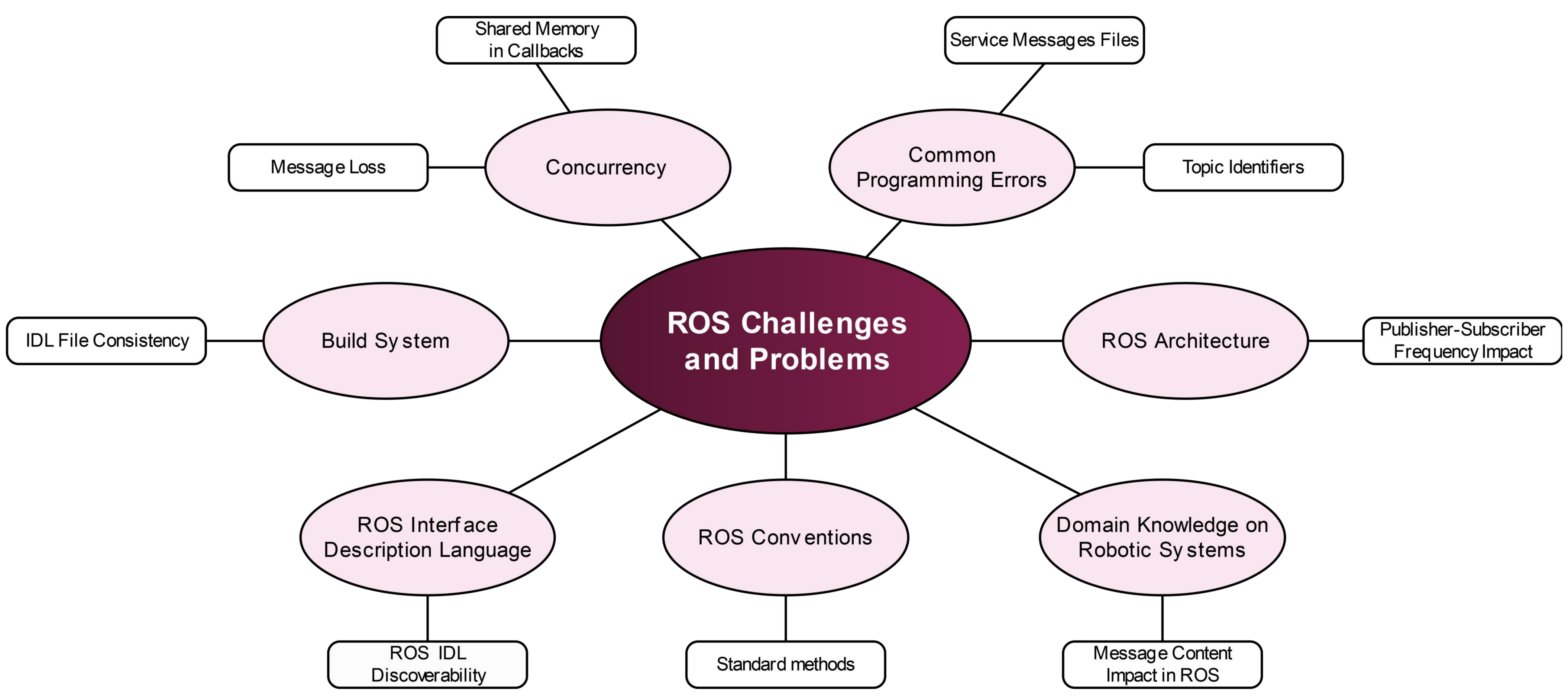
Stage 1 | **Stage 2** | **Stage 3**



Each of three investigators take the ROS Basics in 5 Days (Python) course at **The Construct Sim** and maintain notes on any difficulties encountered.

The unorganized notes are categorized and the investigators discuss the shared challenges.

The identified challenges are consolidated into a mind map consisting of seven top-level categories.



ROS Architecture

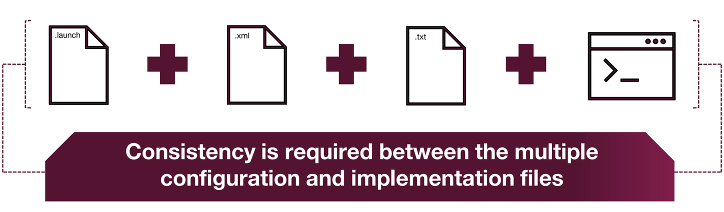
Publisher-Subscriber Frequency Impact (●●●)

- ROS allows developers to connect different components that may have different event frequencies.
- The first challenge appears in the definition of the publishing rate and adequate queue size. Both ROS publisher and subscriber place their messages on a **bounded queue** at a specific **publication rate**.
- The investigators found it difficult to set the correct queue size and rate and understand their impact in the robotic system.

Build System

IDL File Consistency (●●○)

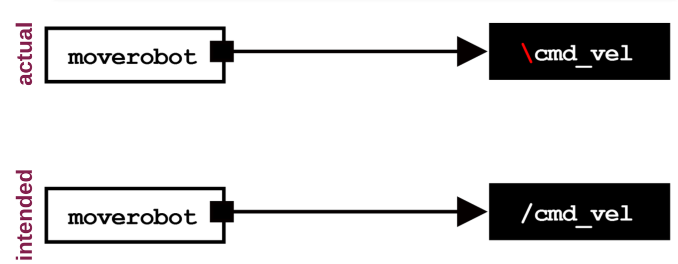
- The process for defining new message formats requires changing code in multiple locations, thus increasing the probability of introducing errors.
- The **lack of sanity** checks by ROS can lead to mismatch between identifiers defined in different files when creating a new node.



Common Programming Errors

Topic Identifiers (●●○)

- In ROS, to publish or subscribe to information one needs to provide the topic name as a string.



- The most common error is the **mistyping of topic names**. Since no verification is done, the system compiles and runs but does not behave as intended.

Service Message Files (●○○)

- In ROS, messages and services are allowed to have the same name.
- However, if both types are used in the same node, the system emits errors that are **not easy to trace** back to the different entities.

Concurrency

Shared Memory in Callbacks (●○○)

```python
sensor = list()

# Callback for scan topic
def scan(scan_msg):
  sensor = scan_msg.ranges

# Callback for odometry topic
def odom(msg):
  position : Pose = msg.pose.pose.position
  if sensor[90] < 1.0:
    print(f"Robot close to wall: {sensor[90]}")
    print(f"Robot Position: {position}")

sub1 = rospy.Subscriber('/odom', Odometry, odom)
sub2 = rospy.Subscriber('/scan', LaserScan, scan)
rospy.spin()
```

Message Loss (●●○)

- A common problem faced by the investigators is the loss of messages, leading the robot to an idle state.
- This problem may occur when a publisher publishes to a topic only once before a subscriber is listening.
- If the connection is not *latched*, the order in which the subscriber and publisher are initiated matters.

ROS Interface Description Language

ROS IDL Discoverability (●●○)

- ROS provides components for different common tasks in robots. Nevertheless, it is challenging for newcomers to **identify the components responsible** for providing certain information.
- Furthermore, it is not explicit how each message and its parameters impact the execution of the robotic systems due to a **lack of documentation**.

ROS Conventions

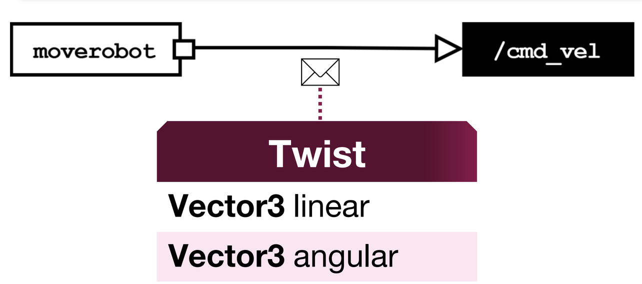
Standard Methods (●●○)

- The investigators found it common not to follow expected good practices in ROS.
- In ROS, the lack of good practices can lead to an unintended behaviour of the system.
- One example is forgetting to implement callbacks and hook methods, typically required for the good functioning of the robotic systems.
- However, there is **no warning or clear message** identifying this issue is in ROS.

Domain Knowledge on Robotic Systems

Message Content Impact in ROS (●●●)

- The abstraction model of ROS hides the dependency on the domain knowledge and the implementation details, hindering the connection between high-level code and its impact in the simulation.



ULISBOA | Ciências ULisboa | FCT Fundação para a Ciência e a Tecnologia | LASIGE | isr institute for SOFTWARE RESEARCH | Carnegie Mellon Portugal

Acknowledgements:
This work was supported by *Fundação para a Ciência e Tecnologia* (FCT) in the LASIGE Research Unit under the ref. (UIDB/00408/2020 and UIDP/00408/2020), and the CMU-Portugal Dual Degree PhD Program (SFRH/BD/151469/2021), by the CMU-Portugal project CAMELOT, (POCI-01-0247-FEDER-045915), the RAP project under the reference (EXPL/CCI-COM/1306/2021), and the U.S. Air Force Research Laboratory (\OSR-4066).

The authors are grateful for their support. Any opinions, findings, or recommendations expressed are those of the authors and do not necessarily reflect those of the US Government.