Increasing ROS Reliability and Safety through Advanced Introspection Capabilities

Andreas Bihlmaier, Heinz Wörn

Institute for Anthropomatics and Robotics (IAR)
Intelligent Process Control and Robotics Lab (IPR)
Karlsruhe Institute of Technology (KIT)
Engler-Bunte-Ring 8
76131 Karlsruhe
andreas.bihlmaier@kit.edu
woern@kit.edu

Abstract: The Robot Operating System (ROS) brought together many different robotics research groups by providing a common software ecosystem. One important part of ROS is the publish-subscribe messaging infrastructure that facilitates building modular and reusable distributed robotics systems. However, the inherent complexity – flexibility, diversity and dynamics – of such a system opposes properties essential to robotics: reliability and safety. The paper details how to gain control of a ROS-based robotics system through the extension of ROS by advanced introspection capabilities. Once the introspection based on distributed gathering of metadata is in place, we are able to define a reference state for the whole ROS system. A monitoring node continuously compares the reference to the actual state. In case of a deviation the operator is alerted through a dashboard, in addition appropriate countermeasures can be initiated by the robotics system itself.

1 Introduction

The Robot Operating System (ROS) can be seen as consisting of a middleware, generic supporting tools, robotics specific libraries and a large collection of robotics software based on the former. Our focus is on the first two components: ROS as middleware together with its supporting tools. Without recounting the middleware architecture [QCG+09], it is important to recall the general components. Overall, the ROS middleware is a publish/subscribe architecture [TvS07]. The ROS master provides the naming and registration services for all ROS nodes. If node A publishes a topic T, it registers T with the master. When node B wants to subscribe to T, it queries the master for the network address of T and contacts A at that address. Afterwards the data is directly sent between A and B, without further involvement of the master.

This architecture allows dynamic addition and removal of nodes, flexible communication patterns and modular robotics software design. However, it can also become difficult to track down problems in such a complex system. The complexity is due to the high

diversity and dynamics of the system. There are many different nodes communicating on many different topics with communicating patterns varying during runtime – e.g. involved nodes, numbers of publishers and subscribers, frequency and latency. Fortunately, a utility visualizing the current ROS graph is part of the ROS support tools. It shows which nodes exist, the connections between them and the message type of each topic. Unfortunately, there are two important shortcomings: First, it is not possible to get information about the data, if any, which is sent between the nodes. Second, there is no tool available, which allows the definition a good reference state of the ROS (sub)graph that *should* be the case and check it against the actual state. We will show in the remainder of the paper, how these shortcomings can be overcome and how this can benefit reliability and safety of ROS-based robotic systems.

The following section (2) goes through all the components that are part of our proposal for advanced ROS introspection capabilities. In order to put these capabilities in concrete terms, we present a case study on a robotic surgery research platform in section 3, which is based on ROS. Section 4 concludes the paper with a discussion about advantages provided by advanced introspection and the obstacles that have to be overcome in order to utilize it.

2 Advanced ROS Introspection

Our proposed extension to existing ROS introspection features (see Fig.1) consists of four elements: First, distributed measurement and publication of metadata about a node's communication. Second, a format to define reference states of a ROS subgraph in a manner that preserves runtime flexibility. Third, a monitoring node that collects all metadata, compares it to the reference and runs predefined actions in case of a deviation. Fourth, a rqt GUI to provide clear visual feedback about the robot systems state, utilizing the output of the monitoring node. Each component will be detailed in the remainder of this section.

2.1 Acquiring Metadata

The data transport of ROS, usually TCPROS, already provides all required metadata on receiving or sending a message either in its header fields or through the sockets API. Utilized network bandwidth, message frequency, jitter and even latency can be acquired for many messages with negligible computation. Furthermore, additional metadata can be generated, such as the correlation between incoming and outgoing messages. These correlations support the reasoning about the common ROS pattern of publishing messages after doing computation on a received message. By way of specifying a new ROS msg type, all these measures can be published for each topic using readily available ROS components. Since only a few floating point numbers are required for the metadata of each topic, the additional network traffic comprises only a small fraction of the data traffic.[1]

[1] In future versions of ROS, based on recent work by [FHM14], it will be possible to collect this metadata, as */statistics*, for all C++ and Python nodes without any modification.

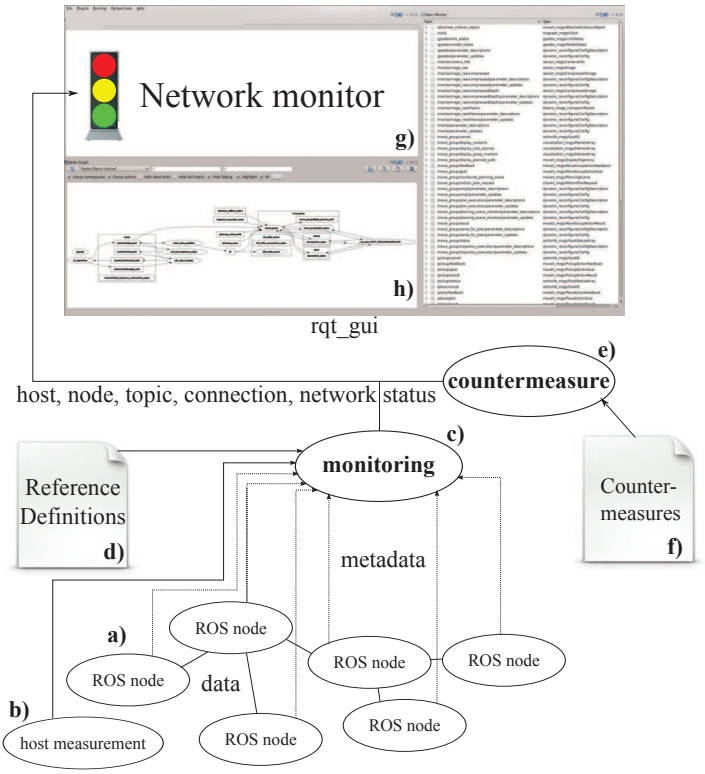

Figure 1: Overview of the advanced introspection capabilities for ROS: a) Gathering and publishing of metadata; b) Nodes to gather host specific data; c) Monitoring node to compare actual metadata against d) reference definitions; e) Node to automatically execute countermeasures defined in f); g) rqt_gui plugin to inspect host, node, topic and connection metadata and to alert about errors; h) rqt_graph extended with metadata information.

In addition to acquiring metadata about message passing traffic, it is also beneficial to measure host and node related features. Such features comprise – per host and per node – CPU, memory and raw network utilization as well as hardware health properties, e.g. CPU and GPU temperatures. The current introspection and monitoring features of ROS for these properties (the rqt process monitor plugin) provide only limited access to them and just for a single host. Given the fact that ROS-based systems are usually distributed across multiple machines, taken together with the flexibility to decide on which host a node runs during startup, operating system level tools regularly prove themselves as insufficient.

2.2 Defining network reference states

The format for reference definitions must allow their definition in a modular and orthogonal way. As a result it is possible to define constraints for different parts of the ROS graph and different – not contradicting – constraints for the same set of nodes. All attributes must support wildcards or value ranges in order to keep flexibility while attaining reliability through monitoring and self-diagnosis. In contrast to other approaches [ZSM+13][MWV14], we propose a simple manual or at most semi-automatic approach. Because automatic approaches are – at least for a research setting – either too sensitive, i.e. have far too many false-positive alarms, or they require too much fine tuning effort and then only work for a very specific setup that does not last long. Therefore we manually define multiple partial constraints that are known to be *necessary* conditions for a working system.

The case study (section 3) will exemplify how the above considerations can be applied to actual systems. Yet, there are also some monitoring rules that can be reasonably applied to most systems. They largely relate to hosts and nodes seen as processes running on a certain host. Often memory and network utilization should stay below some threshold to avoid swapping for the former and package loss for the latter resource. Otherwise both can result in sudden, highly non-linear performance degradation. For CPU (and GPU) utilization, we have to distinguish two cases: On the one hand, there are nodes that consume only a fixed amount of computational resources in the absence of errors. On the other hand, some nodes use up all available computation power to generate as many results per time unit as possible. For the limited consumers, we can define a reference state as usual. For the unlimited case, one can argue that it is ill advised in the first place to have components of unknown performance in a robotic system at all. Nevertheless, it can still make sense to at least define a lower bound for the activity and result frequency of such a node and define falling below this threshold as a system error.

2.3 Monitoring through Metadata

As noted previously, due to the low bandwidth and computation demands, it is possible to collect the *meta*data of all ROS nodes at a central node, even for a large system. This central monitoring node is responsible for comparing the metadata against all constraints given by the reference state. Again, using a custom ROS message type, the result of the comparison is published for each connection, topic, node and host computer. The data can be either used for a robot dashboard (cf. the following subsection) or reacted to by another node in order to increase reliability or at least safety. For example, if the latency to the node for some robot's main environment sensor, such as a RGBD camera, significantly increases or the node completely fails to deliver data, an emergency stop could be initiated instead of the robot crashing into an obstacle. This does not solve the notorious "ROS is not a realtime framework" problem, but at least a higher level of fail-safety can be guaranteed, in case time constraints are violated.

The definition of countermeasures is based on the evaluated metadata (cf. Fig.1). Each metadata property is classified by the monitoring node as either unknown, valid, too-low or too-high. A countermeasure action consists of a logical conditional that is matched against these states and an action that is performed when a match occures.

2.4 Robot status dashboard

Even when running a moderate number of servers, it is good practice to use IT infrastructure monitoring solutions, such as the well-known software Nagios [Jos13]. These software packages enable an administrator to know that a problem has arisen and, more important, what kind of problem it is and where it occurred. Instead of "something is not working right", the monitoring software at least points out "component A, D and X failed" – usually through some kind of dashboard GUI. Our monitoring node allows to transfer this state of affairs from IT infrastructure to distributed robotics systems. Within the ROS ecosystem the adequate implementation of such a dashboard is a rqt[2] GUI plugin. The monitoring plugin provides an overview of the general system state and aggregated access to the metadata. In order to remain usable for larger systems and at the same time provide all details required to track down the source of an error, the system is presented in hierarchical groups. This way multiple hosts, the nodes running on each of these hosts, the various published and subscribed topics of each node and the individual connections making up each topic can be inspected.

3 Case study: Robotic Surgery

The platform to evaluate the advanced ROS Introspection capabilities is a modular robotic system for minimally-invasive surgery (MIS) [NBK+13]. Automated endoscopic camera guidance [BW14] is the particular application targeted in the case study.

3.1 System Overview

The core components for the camera guidance task are as follows (Fig.2): On the perception side we either use an industrial camera with an Ethernet interface or a dedicated endoscope camera through a frame grabber. The system's action is achieved by either – in order of decreasing degrees of freedom – a KUKA LWR4+, a Universal Robots UR5 or a TRUMPF ViKY motorized endoscope holder. However, these are not the only active ROS components in the target scenario, where one robot guides the endoscope and two other robots, equipped with MIS instruments, are telemanipulated. In addition, our robotic surgery platform contains a variety of data-intensive sensors monitoring the OR room.

[2] http://wiki.ros.org/rqt

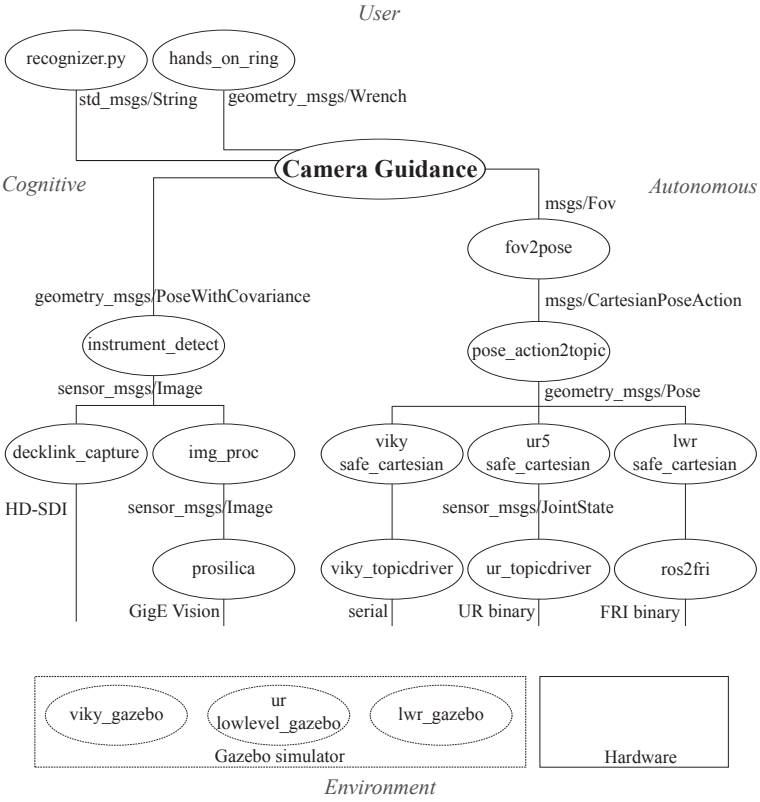

Figure 2: This figure shows part of the ROS graph that makes up the infrastructure for a robotic automated endoscopic camera guidance system (cf. [BW14]).

Given the metadata for all hosts, nodes, topics and connections is published, the first step is to define a number of constraints on the valid state of each ROS component. To illustrate the utilization of the advanced introspection capabilities, we define a reference state for the higher-level camera sensor nodes (see left part of Fig.2) and the lower-level actuator nodes (see right-hand side). In top to bottom direction, we first define that the instrument_detect node should send at least 25 Pose messages per second. Also, we define a maximum delay and jitter value for the connection to the camera guidance node. Going further down in the graph, we define a reference state for the Image carrying topic. Here we have two possibilities: First, to define a different, specific reference for each image source. Second, to define a common, less rigorous reference that is valid for both image sources. In our case one camera provides images at exactly 30 frames per second (FPS) and the other varies between 50 and 60 FPS. Taking into account the big picture of the endoscope guidance system, the important property is to have at least 25 Poses per second. Therefore it is enough to monitor for the same number of images per second. Without going into more detail for the actuator nodes, we want to point out that, based on the same considerations,

we either define a generic rule that holds for all the actuators in use or is actuator specific. By taking advantage of ROS launch files, both cases are easy to deal with. The reference constraints are uploaded from the launch file to the monitoring node via the parameter server.

3.2 Reliability and Safety

Reliability and safety issues do not only occur in surgical robotics, but they become quite obvious when a robot has to control a 30 cm long steel shaft within the patient during surgery. It is, of course, not enough to depend on the ROS systems self-diagnosis in order to guarantee safety. Yet, just as in IT infrastructure monitoring, it is an essential requirement that the operator of the system does not have to scrutinize all parts of the system in order to find out what the problem is. Rather it is commendable that the system points him right at it.

4 Discussion and Outlook

We have detailed why advanced introspection capabilities are important for the reliability and safety of robotics systems based on a distributed architecture such as ROS. Our account then showed how this can be achieved by the generation and analysis of per node metadata, also providing a ROS implementation. One open question concerns the amount of effort required to write down reference definitions that detect anomalies properly, i.e. have a high accuracy, but are general enough not to impair on ROS flexibility. This and further questions related to the real world practicability of self-diagnosis through advanced introspection, are part of our ongoing research into surgical robotics and ROS.

Acknowledgment

This research was carried out with the support of the German Research Foundation (DFG) within project I05, SFB/TRR 125 “Cognition-Guided Surgery”.

References

[BW14] Andreas Bihlmaier and Heinz Wörn. Automated Endoscopic Camera Guidance: A Knowledge-Based System towards Robot Assisted Surgery. In *Proceedings for the Joint Conference of ISR 2014 (45th International Symposium on Robotics) Und ROBOTIK 2014 (8th German Conference on Robotics)*, pages 617–622, 2014.

[FHM14] Dariush Forouher, Jan Hartmann, and Erik Maehle. Data Flow Analysis in ROS. In

Proceedings for the Joint Conference of ISR 2014 (45th International Symposium on Robotics) Und ROBOTIK 2014 (8th German Conference on Robotics), pages 643–648, 2014.

[Jos13] David Josephsen. *Nagios: Building Enterprise-Grade Monitoring Infrastructures for Systems and Networks*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2013.

[MWV14] Valiallah Monajjemi, Jens Wawerla, and Richard Vaughan. Drums: A Middleware-Aware Distributed Robot Monitoring System. In *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pages 211–218, May 2014.

[NBK+13] P. Nicolai, T. Brennecke, M. Kunze, L. Schreiter, T. Beyl, Y. Zhang, J. Mintenbeck, J. Raczkowsky, and H. Wörn. The OP:Sense surgical robotics platform: first feasibility studies and current research. *International Journal of Computer Assisted Radiology and Surgery*, 1(8):136–137, 2013.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop On Open Source Software*, volume 3, 2009.

[TvS07] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems*. Pearson, Prentice Hall, Upper Saddle River, NJ, 2 edition, 2007.

[ZSM+13] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej, and S. Uran. An integrated model-based diagnosis and repair architecture for ROS-based robot systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 482–489, May 2013.