KI - Künstliche Intelligenz (2019) 33:143–150
https://doi.org/10.1007/s13218-019-00581-6

TECHNICAL CONTRIBUTION

Catering to Real-Time Requirements of Cloud-Connected Mobile Manipulators

Christoph Walter[1] · **Julian-Benedikt Scholle**[1] · **Norbert Elkmann**[1]

Received: 7 December 2018 / Accepted: 27 March 2019 / Published online: 12 April 2019

Abstract
In this contribution, we explore real-time requirements of mobile manipulators, a class of intelligent robots, in the context of the ongoing fast-robotics (https://de.fast-zwanzig20.de/industrie/fast-robotics/) project. The project aims at implementing such robots based on (edge-) cloud-services using wireless communication in order to make them more capable and efficient. Instead of trying to universally achieve hard real-time in such a system, we present a mixed real-time approach with an application centered fault tolerance scheme based on transition points and pre-computed alternate plans. We argue that deliberatively addressing uncertainties in timing is similarly important than handling uncertainties e.g. in perception for future intelligent robots.

Keywords Edge computing · Architecture · Robotics · Real-time · Time-aware · Ultra-low-latency · Wireless

1 Introduction

Robots are inherently flexible machines and are thus essential parts of smart production systems. Here we can expect carefully engineered processes and workspaces, even in somehow flexible or adaptable production settings. This includes readily available models suitable for detailed planning of robot actions. Flexibility could be categorized as follows: (1) the ability to deal with local uncertainties like variations in part positions or wear of tools. (2) Overcoming larger process variations like random placement of parts (e.g. bin picking), unexpected obstacles in shared workspaces, or new part variations. (3) Supporting implementation of new processes. We consider category 3 to be relevant during design time, when commissioning, or even during ramp-up or training phases. Category 1 and 2 are run-time properties and thus relevant functions of a suitable control system. While the category 1 can be achieved by simple sense-act scheme using locally installed sensors, category 2 is better served using sense-plan-act as an approach. Even though planning is a broad topic, it involves data or computationally intensive tasks like managing or updating models [12], as well as running planning algorithms. When considering mobile robots, planning tasks are obvious candidates to be offloaded to remote servers. However, sensing and motion generation can also be computationally intensive. Offloading computation from the mobile robot would save space, weight and battery power. The ANNIE robot[1] we use in our experiments requires about 250W for the onboard computers. Assuming a lithium ion battery with a specific energy of 140 Wh/kg is used to power the system over an 8 h operating cycle, we see that ANNIE needs 14 kg of battery weight just for onboard computations. Designing a distributed system, especially one that incorporates heterogeneous hard- and software while still being supportive in addressing key challenges in domain specific application development, is an extensive undertaking requiring considerations in many different directions. This paper is limited in scope to an initial motivation for using a mixed real-time system in conjunction with standard and advanced communication and IT-concepts, a classification of typical manipulation tasks dependent on elasticity concerning quality of service, and an analysis of an exemplary use-case.

✉ Julian-Benedikt Scholle
Julian-Benedikt.Scholle@iff.fraunhofer.de

Christoph Walter
Christoph.Walter@iff.fraunhofer.de

Norbert Elkmann
Norbert.Elkmann@iff.fraunhofer.de

[1] Fraunhofer Institute for Factory Operation and Automation, Magdeburg, Germany

[1] https://www.iff.fraunhofer.de/en/business-units/robotic-systems/research-platform-annie.html

This paper is structures as follows: After the discussion of the related work, in particular concerning progress in wireless networking as well as related software, we motivate timeliness from the point of view of a dynamic robot application. In Sect. 4 we present a scheme for fostering a distributed soft real-time system in conjunction with a fault tolerance scheme in order to implement a control task. This is followed by an in-depth analysis of a concrete application scenario in Sect. 5.

2 Related Work

Our work is the result of an ongoing interdisciplinary project with experts from both the robotics domain as well as from communication and cloud systems. One aspect is the integration of technologies. Thus, it is necessary to briefly summarize the state of the art in both communication- and IT-systems as well as in the robotics and automation software domain for a broader perspective.

2.1 (Wireless) Networking

In data center networking we have seen a convergence of applications to use a common network infrastructure based on Ethernet. Here, support for timing sensitive applications was improved by targeting the elimination of packet loss on the network as well as introducing prioritization. IEEE 802.3x introduced layer 2 flow control, 802.1p priorities for expedited forwarding, while later 802.1Qbb implements a combination of both. 802.1Qat tackles end-to-end management of streams and 802.1Qav introduced credit-based traffic shaping. With 802.1Qat there is a method for reserving network resources. Apart from time synchronization using 802.1AS, which is also available on 802.11 wireless networks, these improvements are limited to the wired backend. Mobile robots usually require some kind of wireless connection. Wireless networking is commonly used for interacive and multimedia applications. In this context wireless perfomance has been improved regarding both latency [11] and as well as packet loss rate [10].

Here, we are using a novel implementation of EchoRing, a wireless industrial network with real-time capabilities. It is a data link layer protocol that extends the well-known token-passing principle with a modified error recovery mechanism in order to deal with the dynamic nature of wireless channels [3]. At the moment, the implementation is bandwidth limited so we combined it with a conventional WLAN. Cellular networks are another wireless communication technology. 5G networks will support network slicing to address the needs of different application classes simultaneously [8]. Classes include machine-type communication as well as ultra-reliable low-latency communication (URLLC). Low-latency required the use of the 5G new radio implementation (NR) which is still in its infancy. However, field tests have shown radio segment delays of less than 1 ms [6].

2.2 Software

Robots can be approached from different points of view. One such perspective is industrial automation. In automation equipment, real-time communication has also transitioned to Ethernet-based communication. With implementations such as Profinet (rt/irt), EtherCAT, SERCOS 3, or Ethernet/IP, we see both largely proprietary solutions with very good real-time properties including isochronous communication and solutions featuring co-existence or interoperability with other Ethernet protocols.

Time sensitive networking (TSN) is a current development based on the already established standards mentioned in the previous section but with a clear emphasis on highly deterministic real-time and automation (IEC/IEEE 60802). It introduces a number of new or refined technologies into Ethernet while enabling interoperability with other traffic. An analysis [4] of TSN also shows that it has the potential to displace all many other bus systems in the industial robotics.

However, TSN does not say anything about the minimal possible latencies of the software over the network, but merely specifies an upper expected limit for them. For the overall performance, the speed of the network must therefore also be taken into account. Since the speed of the network also depends on the network stack of the operating system, we have searched for analyses of this.

Due to the real-time software still available on the robot, such as the motor control and the evaluation of the safety laser scanners, the operation system must be real-time capable and due to the simple availability Linux is used here together with the RT-Preempt patches. The investigation [1] shows that the RT-Linux kernel can even contribute to the reduction of network latencies and even shows significant improvement through over adaptations.

The popular robotics software framework ROS is currently transitioning to the new communication middleware DDS [7]. This should open possibilities regarding better integration especially with timing improving networking technologies.

2.3 Summary

While the mentioned technologies improve determinism of communication channels, we still face basic challenges:

- Admission of a (communication) tasks may fail in a dynamic system.
- Unknown or hard to estimate worst-case execution times of high-level computation tasks.

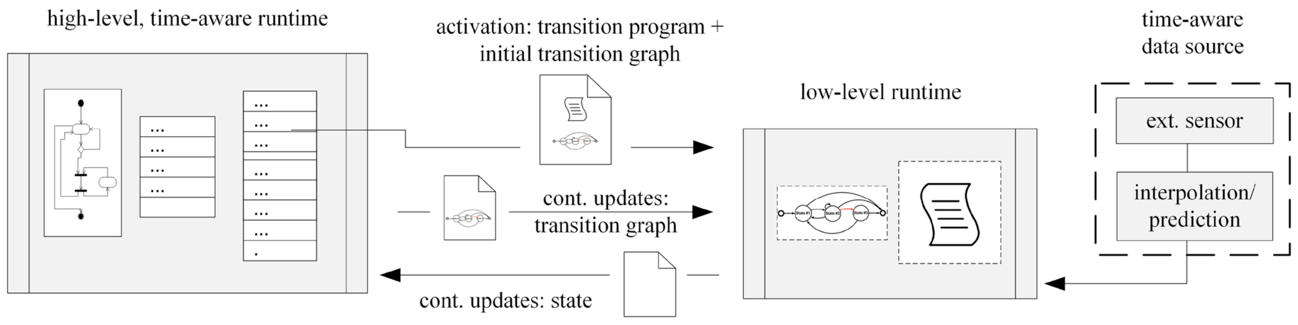
high-level, time-aware runtime
activation: transition program + initial transition graph
cont. updates: transition graph
cont. updates: state
low-level runtime
time-aware data source
ext. sensor
interpolation/ prediction

Fig. 1 Basic system structure

- Conservative timing guaranties (when available).
- Variable bandwidth of wireless communication over time.

This stipulates the necessity of designing robot applications which are aware of the timing of their communication and computation systems and that can react to it in a similar way as they would react to observations of the environment. When it commes to robot control architecture, the availability ot new these improved wireless communication systems grants access to vast computational ressources. This in turn enables a better integration of high-level tasks like planning and low-level execution by the means of a massive ammount of pre-computed behavior variations.

3 Motivation for Timeliness

Robots implement interaction with the physical world and are by themselves physical objects. This implies real-time constraints on their control systems. On the other hand, the advent of ROS [9] has proven the usefulness of software systems with weak or no real-time support in this area. When taking a look at robot applications we can see different tasks or sub-tasks, some of which are sensitive to timing in some way, others are not. Timing-sensitive tasks can be found in applications where the process to be carried out is timing sensitive. An example is glue application. Here, deviations from a carefully engineered trajectory will cause imperfect results. The application process is dynamic and a temporary loss of control results in undesired behavior. Other processes are not inherently timing sensitive. For example, bin picking: The objects in the bin do not move on their own, the robot system can be completely implemented with best-effort timings. In this case, the robot would move incrementally whenever a new waypoint is generated. The system would be quite slow and may not be feasible for commercial application. There is however a limit on how fast a best-effort robot can become. The transition to swift and fluid movement of the robot again introduces a dynamic process where temporary loss of control again results in undesired behavior. Robotic control architectures can be classified in hierarchical, behavioral, hybrid approaches. All of which are organised in higher and lower level subtasks. The low-level part can only act upon locally available information. This part can easily made real-time compliant, in contrast to high-level tasks. Furthermore, reactions are either very limited or become autonomous, which can be undesirable because of the limited access to relevant knowledge on the lower level. In the remainder of this paper we discuss a solution whereby we continue controlling the robot in case of a timing fault of the computer or the communication system. This is done for the most part deliberatively based on pre-planned motion at a higher level. We also incorporate data for fast reaction from a soft-real-time sensor based on sensor prediction.

4 A Fault Tolerance Scheme Based on Transition Points

Our basic architectures (see. Fig. 1) consist of a high-level, time-aware runtime, a low-level runtime, and additional time-aware data-sources. The low-level runtime implements a time-triggered control loop and is local to the robot. It is hard real-time and has access to local information from select sensors. It furthermore subscribes to data sources with a similar time scope from the event driven high-level system, if necessary, to make decisions. It is only concerned with actions relevant to the current situation. The high-level runtime implements the overarching robot program. It is time-aware in the sense that it knows timing requirements of both the low-level runtime and the application task either explicitly from the user provided program or implicitly derived from models. It continuously communicates with the low-level runtime during task execution and can be consolidated onto a remote server. This is most preferably an edge server with bounded communication latency, but implementation on a core server with soft real-time communication link to the robot is conceptually also possible. Time aware data sources are endpoints in event-driven processing chains

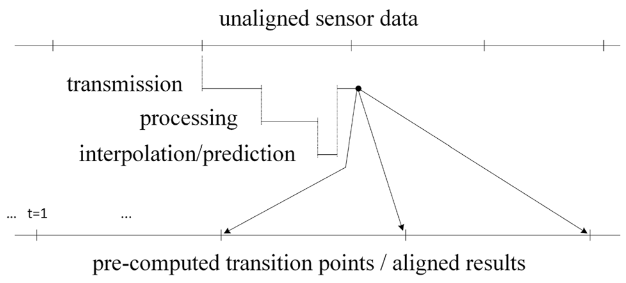

Fig. 2 A time-aware data source is a chain of tasks (transmission, processing, interpolation/prediction) and provides results aligned to transition points

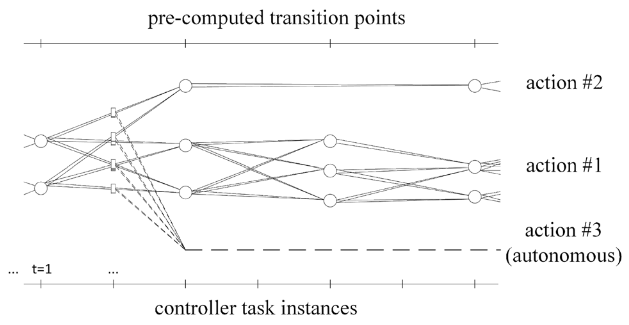

Fig. 3 Concept of transition points and branching within and between actions. The number of controller task instances has been reduced for clarity. Dashed lines represent autonomous actions and transitions. Circles represent nodes in the pre-computed transition graph, while rectangles show implicit transitions to autonomous actions for controller task instances when no transition point is available (only one instance is shown to avoid clutter). Nodes are connected with multiple lines because blending causes different results for different transitions

which provide useful information. The emission of such events can be aperiodic, but because the source is aware of the timing requirements of the subscriber, information is processed to best fit the subscribers needs. This is being achieved by interpolation or prediction onto the required points in time.

In order to have the benefits of global knowledge gained from various data processing services including object detection, recognition and tracking, as well as demanding algorithms like collision-free motion planning and optimization, we want these tasks to be performed on off-board. As a layered control architecture, we need a scheme for mediation between higher-level (slow, soft real-time, event-based) and lower-level (fast, hard real-time, cyclic) layers.

While it is possible to trigger a sequence of otherwise autonomous behaviors implemented at the lower level, this approach does not fully benefit from detailed knowledge and models available only at a higher level. Thus, we implemented a hybrid approach. The concept is based on a large number of pre-computed movement variations to form an overall deliberative action. Tightly spaced transition points enable the change between trajectories in order to be able to still react swiftly to external events. This included real-time guarantees. We propose a spacing of 5–20 ms depending on the application as a suitable amount of transition points. This is also dependent on the capabilities of external data sources needed to make decisions. A time-aware data source, such as a camera system with object tracking capability, is expected to provide aligned data, if necessary based on predictions as input to the low-level runtime (see Fig. 2) as well as for the high-level runtime for planning.

The various movement variations can be pre-planned using parallelization on powerful off-board servers with a larger granularity of about ten transition points. We will present values for a concrete example in Sect. 5.2. The number of variations grows rapidly over time so it is important to limit the number of transition points as well as the number of alternative planning goals to a number necessary for the application. This requires a better understanding of the application (sub-) task that we want to implement, but avoids computationally expensive overprovisioning.

The movement is encoded in the form of a graph structure where transition points are a group of nodes labeled with the same timestamp, t (see Fig. 3). Each node represents an intermediate pose, i.e. an intermediate goal configuration. The edges represent trajectory segments to reach the next set of nodes at the next transition point. The maximum number of trajectory segments between transitions is $n_{t-1} \cdot n_t \cdot n_{t+1}$ with n as number of nodes. Not all transitions are possible in all cases, because they exceed the dynamic limits of the robot. Other nodes can lead to paths where no alternatives are available or where alternatives have a different spacing of transition points. Autonomous behaviors are still necessary for implementing fastest transitions to desired actions, e.g. reactions to collisions. Special care must be taken for transitioning back from autonomous to pre-planned actions. The simplest approach to this problem is to do it when stopped. Another is to use motion generation at the low-level runtime for a brief period until a node in the deliberatively planned graph structure is reached. This also requires the use of relative timestamps for nodes because execution time of the transition is be variable. Actions group steps or branches of the program with distinct purpose (see Fig. 3). They are similar to traditional behaviors in the sense that activation and execution is autonomous based on locally available information to make decisions. However, the comparison can also be misleading since, in our case, we emphasize deliberatively planned movement variations using global models. The decision process within the local real-time controller is realized through a user defined script provided at run-time initialization. This transition program is executed with every cycle of the control loop and is implemented

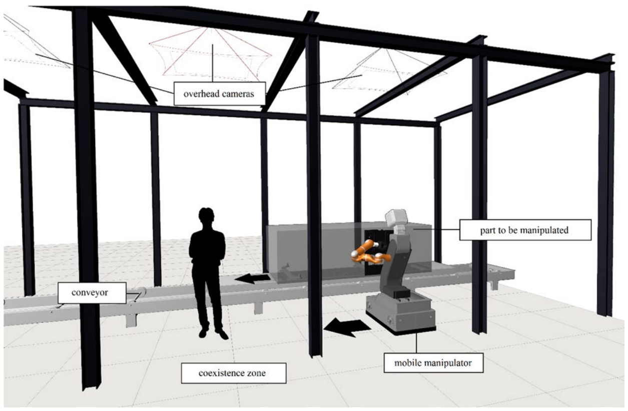

Fig. 4 Experimental setup conceptual overview

with the lightweight embedded language, Lua version 5.x [5], which we adapted for real-time use. In every cycle the transition program checks which transitions are available for the current action and on the current cycle. It then evaluates their necessity based on locally available information or on the lack of information due to communication interruption. Transitions to autonomous behavior may occur on any cycle. Transitions to pre-computed alternatives only occur on specific transition points: points in time for which nodes are available in the graph structure. At these points transitions between pre-planned trajectories are available.

5 Analysis of an Application Scenario

In this section, we discuss a scenario as a use-case for our system architecture. It consists of an assembly line with multiple work stations in sequence. The work stations are designed for human employees to operate. We want to alternatively operate one workstation using an anthropomorphic mobile manipulator. This implies not only a collaborative setting, i.e. co-existence [2] with human co-workers at neighboring workplaces, but also the requirement of completing tasks at the same speed as a human workers.

5.1 Physical Setup, Application, and Sub-Tasks

The physical setup is shown in Fig. 4. The whole system is about 10 m long and 5 m wide. We are using a mobile manipulator based on an omnidirectional mobile platform with Mecanum wheels, a head-mounted sensor array including a fast monochrome camera, wireless communication, and a robot arm mounted on an adjustable torso. The robot arm controller requires a 1 ms cyclic task execution, while the mobile platform is controlled in 5 ms cycles. These real-time functions are implemented on a x86 Intel Pentium M platform. The robot's reach is roughly comparable to that

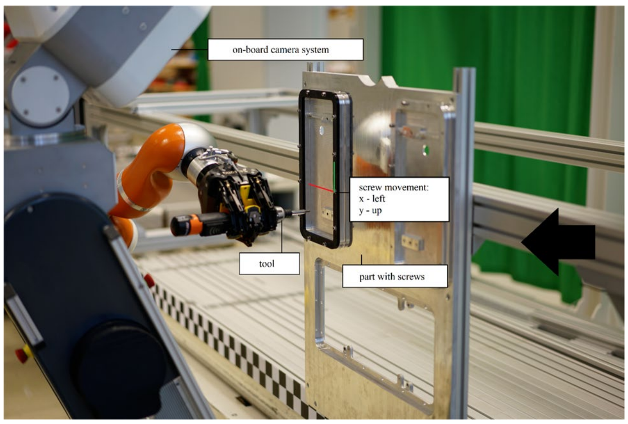

Fig. 5 Experimental setup with robot and tool for manipulating the part

of a small-sized person. For situational awareness we use additional overhead cameras dedicated to the identification of approaching persons while the robot is focusing its attention on the work process. The task of the robot is to approach parts passing by on the conveyor, and fasten a number of pre-placed screws on the part using a battery powered, hand-held screwdriver. The robot therefore needs to track the conveyor when working on the part and thus may come close to a human co-worker at the next station. Workers may also come close to the robot on their own, requiring the robot to react accordingly. Off-board servers are placed near the station and have a direct, wired connection to the wireless transmitters mounted on a support frame in the working area. The basic sub-tasks can be summarized as follows:

- Detection of the approaching part.
- Platform motion towards the part and synchronization with conveyor movement.
- Locating the screws and moving the arm into a suitable start configuration.
- Inserting the tip of the tool into a screw.
- Tighten screw and repeat the previous three steps until all screws are fastened.
- Retract the arm and drive to home position.

While several of the sub-tasks are interesting candidates for a detailed discussion, we will limit our analysis to a single core task, the insertion of the tool into the screw. This task involves tracking both tool and screw via the on-board camera system. The image processing and the motion generation will be executed by the off-board servers. The movement cannot be pre-programmed, because the position of the part and the robot are expected to vary between instances. The robot must also react to approaching workers and inadvertent collisions by retracting the arm to avoid a dangerous situation and then stopping (Fig. 5).

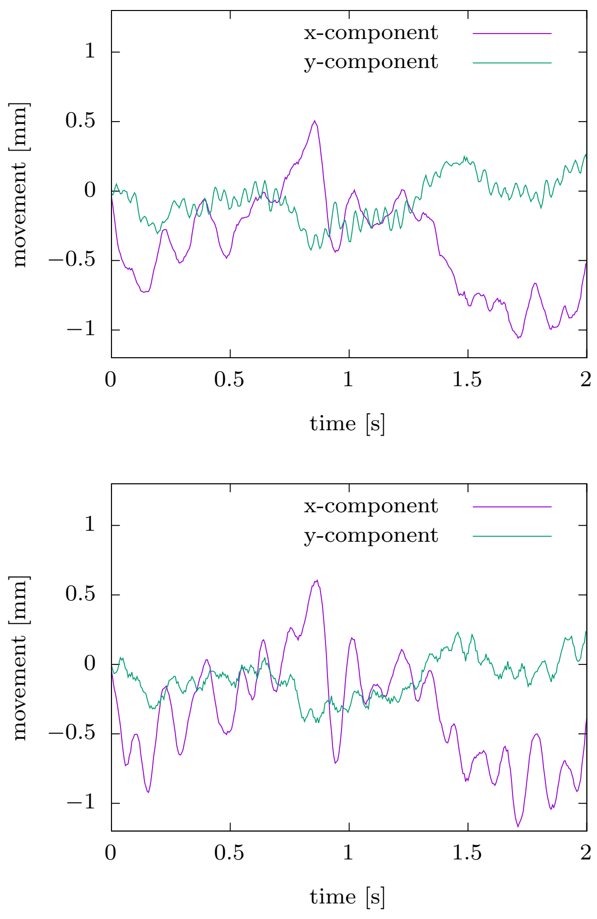

Fig. 6 Movement of screw (upper) and screw in relation to tool (lower) during a 2 s period as observed by the head mounted camera system

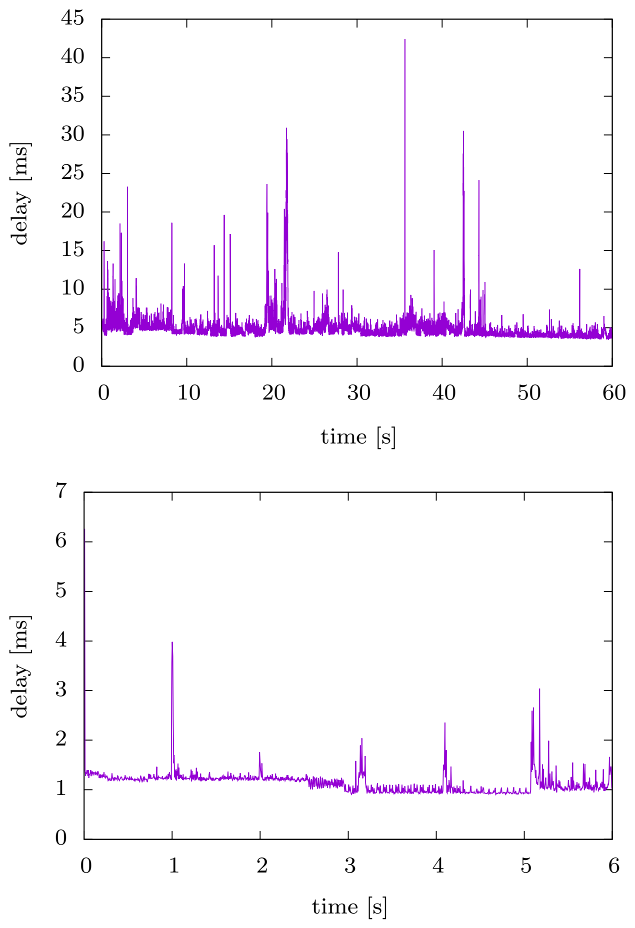

Fig. 7 Communication delays when transmitting image data from the robot to an external server over a period of 60 s (upper), processing time for object tracking over a period of 6 s (lower)

5.2 Implementation of Tool Insertion Sub-Task

As previously mentioned, we focused our work on the sub-task of inserting the tool into a screw at a part which is fixed to a moving skid, requiring motion of both the manipulator arm and the mobile platform to complete the task. While this example does not fully show the potential of higher-level motion planning for the quite small overall movement of approximately 5 cm, it is the sub-task which requires the fastest reaction times and the largest amount of planning alternatives, i.e. nodes and transitions in the graph structure. This made it a suitable candidate for studying the feasibility of our approach. At first, we built a physical process model by analyzing the dynamics of the process using test runs. As a physical process, the movement of tool and screw in relation to the observing sensor shows periodic properties. In test runs, the x-component showed oscillations with a frequency of approximately 7 Hz while the y-component oscillated with a frequency of around 23 Hz (see Fig. 6). While the mobile platform in general tracks the skid satisfactory, dynamic position variations of the screw in relation to the tool clearly need to be accounted for when planning the insert motion.

We also need to determine the capability of the robot to compensate for the movement of the screw in relation to the tool. By experimental trials we observed in the worst case (full reversal of direction) that in our setup the robot was able to compensate 1.1 mm in 1/23 s, which is about twice as fast as needed according to preceding analysis of the process.

Next, we characterized the communication channels. We decided to use WLAN to transfer image data to a remote server because of bandwidth requirements of about 3 MBytes per second. This already includes area of interest cropping with subsequent compression. Figure 7 shows the communication delays measured at our test location with 8 other wireless networks in the vicinity. The average latency was below 5 ms with peaks in the range up to 40 ms. Result transfer back to the robot took an average of 1.8 ms with peaks up to 7 ms. All other communication including binary encoded graph structures are estimated to not exceed 200 kBytes per second and can be transferred using EchoRing which provides a bounded transmission time of 10 ms. The processing times vary slightly with an average of 1.1 ms and spikes below 7 ms.

Based on this understanding of the process and communication and computational systems, we generate predictions of most likely part movements in order to be able to plan ahead in time. We use a simple autoregressive model to predict both x- and y-components separately and combine those to find a likely 3D goal position. Based on test data, the mean prediction error 50 ms ahead in time is about 0.09 mm, while 83.9% of prediction were within a distance of 0.15 mm from ground truth. We consider that distance to be to the threshold where larger deviations result in a failed attempt. Ground truth is generated by the camera system. The camera has 1280×1024 pixels, which results in a resolution 0.2 mm per pixel and a sub-pixel tracking performance of 0.02 mm at the working distance. We assume this to be sufficiently accurate for this analysis. To increase the likelihood of success we increase prediction coverage by generating alternate goal positions. Based on our test dataset, which contains 1258 measurements, planning only two alternates resulted in a complete coverage. Regarding the influence of the data source on live decisions, we estimate that in 1.8 percent of instances prediction error for a 10 ms interval will lead to a failed tool insertion. In addition, timing variance will lead to instances where less accurate predictions have to be used. This increases the percentage of instances that will fail to 2.4. This is still a significant amount, which needs further error handling. However, a failure to insert the tip properly can easily be detected by a verification movement in combination with repeating the sub-task. In this case the error rate can be considered acceptable. The main factor is the quality of the predictions, which can be improved by a more elaborate model or shorter overall latencies.

For the implementation of the motion planning in the high-level runtime, we therefore designed the following fault tolerant application: We choose to generate trajectories to the most likely position of the screw, with between 2 and 4 alternatives for the insertion action. Further transitions are also generated when certain events occur. If a human is predicted to come too close to the robot, a Retraction movement to a safe position triggered. If the screw moves outside the region covered by predictions and therefore cannot be reached by a pre-computed trajectory or if a timing fault occurs and sensor- or planning data becomes unavailable, a transition is triggered to a slightly retracted hold position. The final transition is to an autonomous reaction in case of a premature collision being detected. That reaction gets triggered based on local torque sensors only and can happen on any controller cycle without the need for a pre-computed transition point. The reaction involves switching to a joint compliance mode which can be transitioned out of as soon as the arm comes to a stop when external forces have subsided. Overall, we have deliberate transition points every 5 ms in conjunction with a planning horizon of 50 ms. In the worst case, this amounts to 1750 trajectory fragments to by planned this time. For this sub-task, two data sources are required by the low-level runtime: The camera-based tracking of the distance between screw and tool, and a minimum distance to the closest human worker in the vicinity.

6 Summary and Outlook

Real-time requirements in robotics depend on the specific tasks of an application. Some tasks (Sect. 3) require hard real-time control and a fixed update rate; others are more flexible but still sensitive to timing faults at relevant execution speed. We have presented a scheme of realizing such tasks using a distributed runtime which deliberatively plans many alternatives of concrete behavior to relax time constraints on higher-level planning. As a basis for choosing between alternatives in a timely manner we used aligned predictions from asynchronous event driven sensor data processing which also reduces real-time constraints on sensing. Overall our approach represents a tradeoff between computational resources and communication bandwidth on the one hand and latency as well as jitter in processing and communication on the other hand. In future work we plan on using the presented scheme in the implementation of other sub-tasks of the application. An interesting question is the possibility of performance improvements in the discussed sub-task for tool insertion by automatically adjusting the planning horizon based on the observed dynamics of the process. Further automatic adjustments include the timing properties of the off-board, time-aware data sources, in particular the communication channel. In this work we had to identify these manually. However, we expect this to be an automatic feature, especially in next generation cellular networks where even short-term predictions are possible.

Acknowledgements This work was partially funded by the Federal Ministry of Education and Research of Germany under Grant agreement no. 03ZZ0528A.

References

1. Abeni L, Kiraly C (2013) Investigating the network performance of a real-time Linux kernel *. http://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=754BB29FF6DFD09D0007FC2787DC6044?doi=10.1.1.702.7571
2. Behrens R, Saenz J, Vogel C, Elkmann N (2015) Upcoming technologies and fundamentals for safeguarding all forms of human–robot collaboration. In: Deutsche Gesetzliche Unfallversicherung e.V. -DGUV-: 8th international conference safety of industrial automated systems. SIAS, pp 18–23
3. Dombrowski C, Gross J (2015) Echoring: a low-latency, reliable token-passing mac protocol for wireless industrial networks. In: Proceedings of European wireless 2015; 21th European wireless conference, pp 1–8
4. Gutiérrez CSV, Juan LUS, Ugarte IZ, Vilches VM (2018) Time-sensitive networking for robotics. Computing research repository (CoRR). https://arxiv.org/corr
5. Ierusalimschy R, de Figueiredo LH, Filho WC (2005) The implementation of lua 5.0. J Univ Comput Sci 11(7):1159–1176
6. Iwabuchi M, Benjebbour A, Kishiyama Y, Ren G, Tang C, Tian T, Gu L, Takada T, Kashima T (2017) 5g field experimental trials on urllc using new frame structure. In: 2017 IEEE Globecom workshops (GC Wkshps), pp 1–6. https://doi.org/10.1109/GLOCOMW.2017.8269130
7. Maruyama Y, Kato S, Azumi T (2016) Exploring the performance of ros2. In: 2016 international conference on embedded software (EMSOFT), pp 1–10. https://doi.org/10.1145/2968478.2968502
8. Popovski P, Trillingsgaard KF, Simeone O, Durisi G (2018) 5g wireless network slicing for embb, urllc, and mmtc: a communication-theoretic view. IEEE Access 6:55765–55779
9. Quigley M, Conley K, Gerkey BP, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: an open-source robot operating system. In: ICRA workshop on open source software
10. Sheshadri RK, Koutsonikolas D (2017) On packet loss rates in modern 802.11 networks. In: IEEE INFOCOM 2017—IEEE conference on computer communications, pp 1–9. https://doi.org/10.1109/INFOCOM.2017.8057130
11. Wellnitz O, Wolf L (2010) On latency in IEEE 802.11-based wireless ad-hoc networks. In: IEEE 5th international symposium on wireless pervasive computing 2010, pp 261–266. https://doi.org/10.1109/ISWPC.2010.5483719
12. Zaheer S, Jayaraju M, Gulrez T (2015) Performance analysis of path planning techniques for autonomous mobile robots. In: 2015 IEEE international conference on electrical, computer and communication technologies (ICECCT), pp 1–5. https://doi.org/10.1109/ICECCT.2015.7226205