공학석사학위논문

System optimization of ROS-based open source autonomous driving platform

ROS기반의 오픈소스 자율주행 플랫폼 시스템 최적화

2022년 08월

서울대학교 대학원
컴퓨터공학부
박 병 규

System optimization of ROS-based open source autonomous driving platform

ROS기반의 오픈소스 자율주행 플랫폼 시스템 최적화

지도교수 이 창 건

이 논문을 공학석사 학위논문으로 제출함

2022년 06월

서울대학교 대학원

컴퓨터공학부

박 병 규

박병규의 공학석사 학위논문을 인준함

2022년 08월

위 원 장 하 순 회 (인)

부위원장 이 창 건 (인)

위 원 김 지 홍 (인)

Abstract

System optimization of ROS-based open source autonomous driving platform

Byungkyu Park
Department of Computer Science and Engineering
The Graduate School
Seoul National University

The open-source robot operating system(ROS) is being studied in complex system such as autonomous driving. Many studies have made efforts to port real-time to ROS for complex systems based on ROS. However, these methods are not user-friendly because they are difficult to use and require complicated procedures. This paper focused on the response time to improve ROS performance without modifying the ROS structure or adding other complicated procedures. We found that one of the characteristics of ROS causes response time delay. In this paper, we describe a method to improve response time with user convenience by using the characteristics of ROS. Finally, We show the performance of the method presented in this paper through several experiments.

keywords : Autonomous Driving, Optimization, System Profiling

Student Number : 2020-22934

Contents

List of Figures

1 Introduction

The open-source robot operating system(ROS), which has been rapidly growing in recent years, is being studied in various fields[1]. As the field of research expands, ROS has come to implement in more complex and high-performance systems, such as autonomous driving. Several studies, such as [2][3][4], have tried to port real-time systems to improve the performance and robustness of ROS. However, since ROS cannot guarantee real-time, the real-time ROS proposed in several papers requires a complex process, so it is not user-friendly due to low portability.

This paper proposes a method that can improve ROS performance compared to the default setting of ROS in an easily portable way. We try not to modify the structure of ROS for user convenience, and for this, we pay attention to the system's response time.

We model ROS as a Directed Acyclic Graph(DAG), and we measure the worst-case response time of all DAG paths in the system. We designate the path with the highest worst-case response time as the critical chain.

We analyze the ROS to reduce the response time of the critical chain, and we find that there is a delay that occupies a large portion of the response time. Furthermore, the characteristics of ROS cause the delay. We call this alignment delay and propose a method to increase the spin rate to reduce the delay. The spin rate is one of the characteristics of ROS and is a parameter. Adjusting the spin rate can be said to have high portability because it does not require complicated processes such as ROS modification or separate application installation. Of course, increasing the spin rate puts a load on the system. However, we show that the load is not large through the ex-

periment and that we cannot infinitely increase the spin rate through the experiment. Our approach is to model the response time of ROS to consider this load and find an appropriate spin rate.

The contribution of this paper is to improve the response time through a heuristic algorithm by modeling the response time of the ROS. And the way found through this algorithm can significantly reduce the response time of the critical chain compared to the initial setting.

To prove this, we select a target system Autoware, one of the ROS-based autonomous driving systems[5] and reveal how much performance improvement it through measured response time and a simulation result. This paper is organized as follows. Section.2 briefly describes the background required for the paper. In Section.3, we are going to introduce about Instance Chain and our observation. Then, Section.4 explains our proposed algorithm. Section.5 reports our experiment results. Finally, Section.6 concludes the paper.

This paper is organized as follows. Section 2 briefly describes the background required for the paper. In Section 3, we are going to introduce about Instance Chain and our observation. Then, Section 4 explains our proposed algorithm. Section 5 reports our experiment results. Finally, Section 6. concludes the paper.

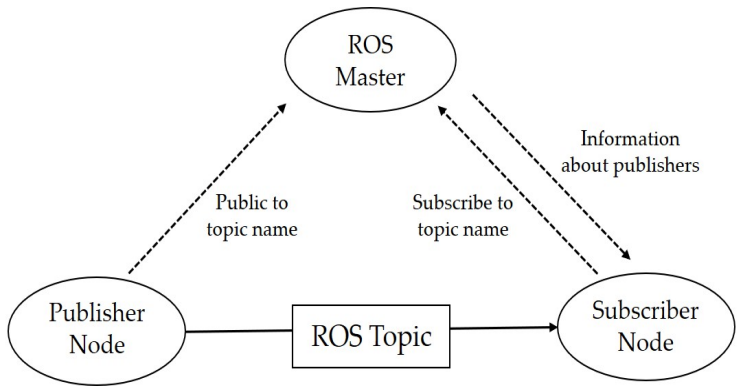
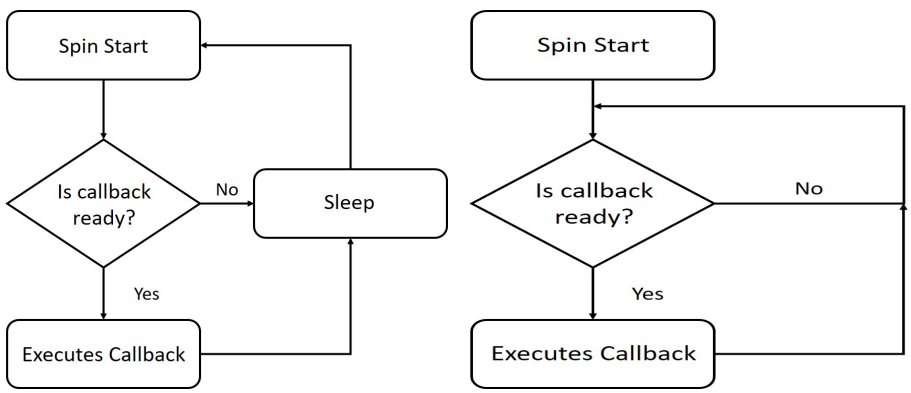


Figure 1: ROS Structure



(a) Synchronous Spinner (b) Asynchronous Spinner

Figure 2: Spinner

2 Backgound

Autoware is based on Robot Operating System(ROS). ROS has three characteristics. One is a structure; another is the position of ROS in the system layer; the other is a spinner that runs a program of ROS.

2.1 ROS Structure

Figure 1 shows the ROS structure. The ROS uses the expression node as a program unit. There are 3 types of node. One is master node, another is publisher node, the other is subscriber node. Each node is under the management of the master node, and data is exchanged between nodes in the form of topic. When data is exchanged between nodes, the ROS forms a relationship between a publisher and a subscriber node. The publisher node generates data and publishes, and the subscriber node subscribes the topic and performs processing on the data of the topic. A node can be both subscriber and publisher. Because of this structure, you might misunderstand ROS for event-driven. However, ROS doesn't work as event-driven.

2.2 Node

The node's elements are a spin rate, callback, callback queue, and spinner. Basically, the node works at a spin rate given by the user. A node can have one or more callbacks, and each callback has its callback queue, which stores the subscribed topic data. The spinner performs the node at a given spin rate.

As figure 2 shows, there are two kinds of the spinner in ROS. One is a synchronous spinner; the other is an asynchronous spinner. The synchronous spinner is the standard spinner used by most nodes. It sleeps and wakes up repeatedly at the rate set by the user. On the other hand, the asynchronous spinner is a busy waiting spinner. It executes a callback as soon as data enters the callback queue, and the behavior is the same as setting the rate to the maximum value in the synchronous spinner.

The execution of the node is as follows: When the node wakes up, the spinner checks the callback queue. If there is data in the callback queue, the spinner executes the

callback. Otherwise, it sleeps and wakes up at the given rate. If there are two data in the callback queue, only one data at a time is processed in the callback, so the second data is executed in the next cycle.

ROS uses the spinner, which runs a node at a given spin rate, as seen in the node execution flow. This attribute makes the node runs independently rather than event-driven.

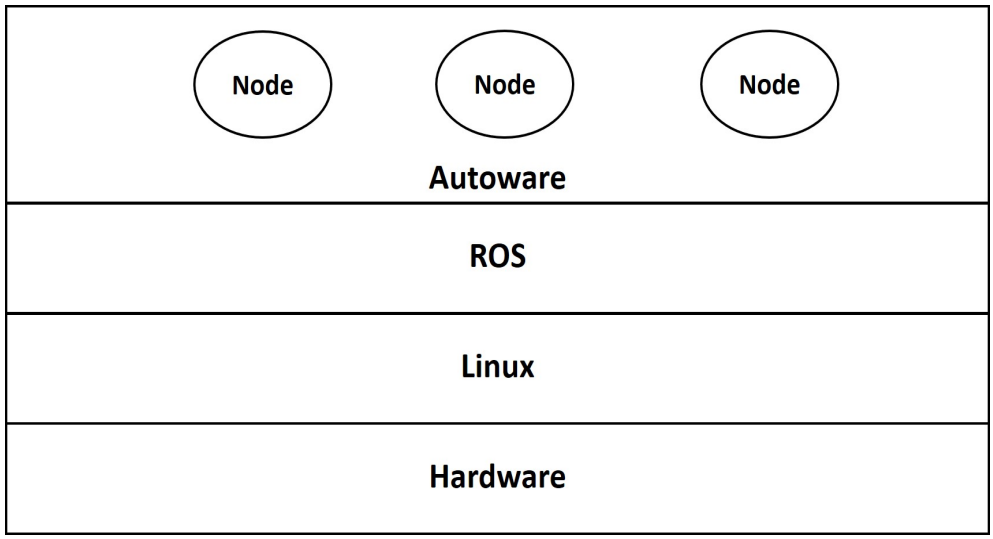


Figure 3: ROS System Layer

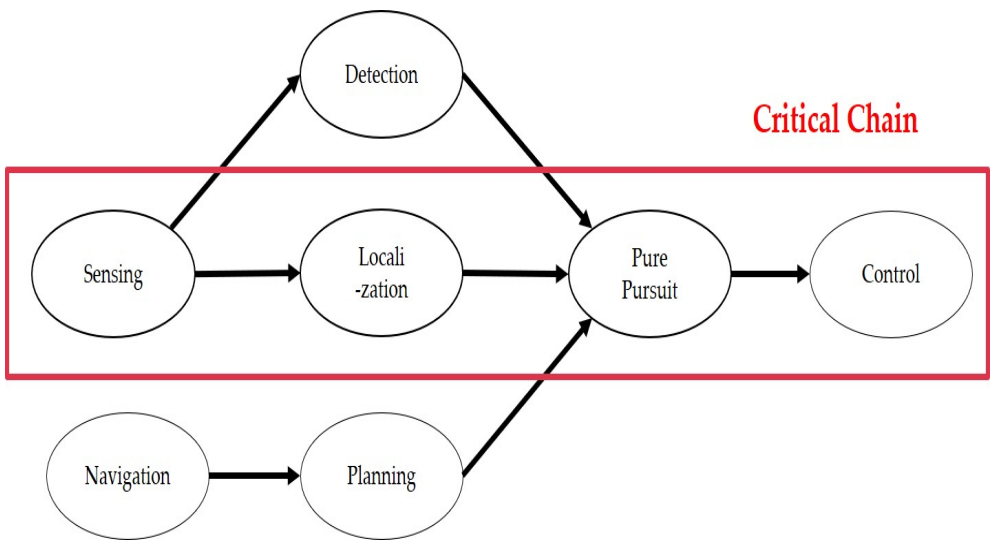


Figure 4: Autoware

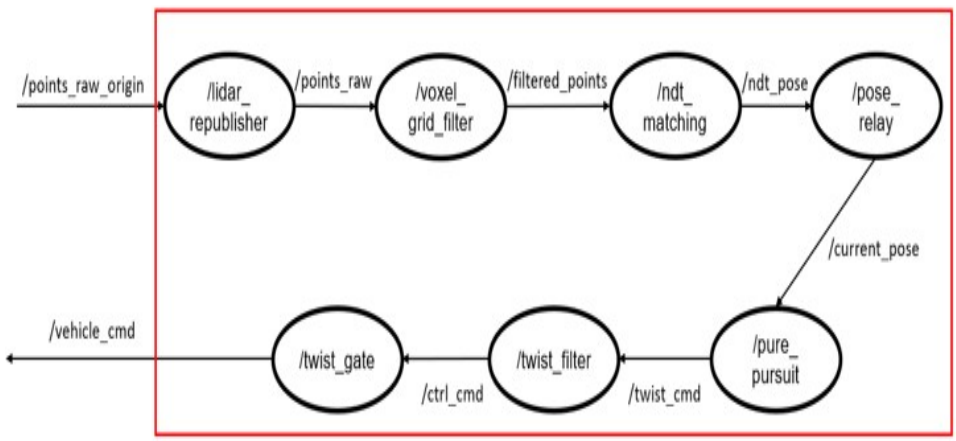


Figure 5: Critical Chain

3 Problem Description

To improve Autoware, a complex autonomous driving system, profiling is necessary to diagnose the system. However, It is challenging to precisely and accurately profiles the entire system. Because Autoware performs on ROS, and ROS is middleware that works on Linux, as seen in Figure 3.

3.1 Critical Chain

To profile Autoware, We designate the most important chain as a critical chain among several chains constituting Autoware. Figure 4 shows the approximate structure of Autoware. Here, directly involved in vehicle control is a chain leading to sensing, localization, and control. We will define this as an Critical Chain. The nodes composing the Critical Chain and the topics published by each node are as shown in figure 5.

We profile the critical chain for one sensor data defined as an instance. And we clas-

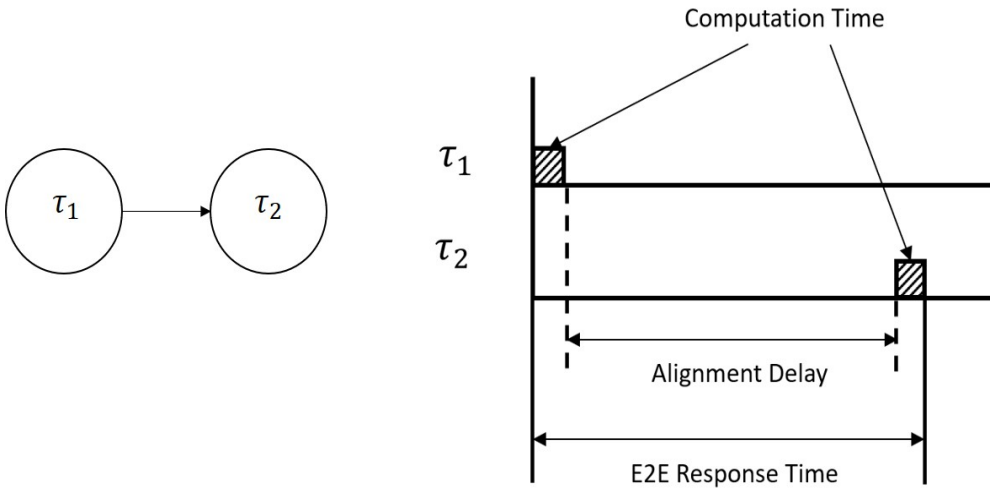


Figure 6: Profiling Example

sify the profiling result into three types: Critical Chain end-to-end(E2E) Response time, Alignment Delay, and Total Node Computation time.
I will explain Figure. 6 as an example. Assuming that there are two nodes, τ_1 and τ_2, in the critical chain.

- Critical Chain E2E Response Time

Critical Chain E2E Response Time is measured from the start of τ_1, the first node of the chain, to the end time of τ_2, the last node of the chain, for one instance. The E2E response time consists of the sum of the alignment delay and the computation time of each node.

- Alignment Delay

The alignment delay is measured from the end of the predecessor node to the start time of successor node. The alignment delay occurs because each node operates in-

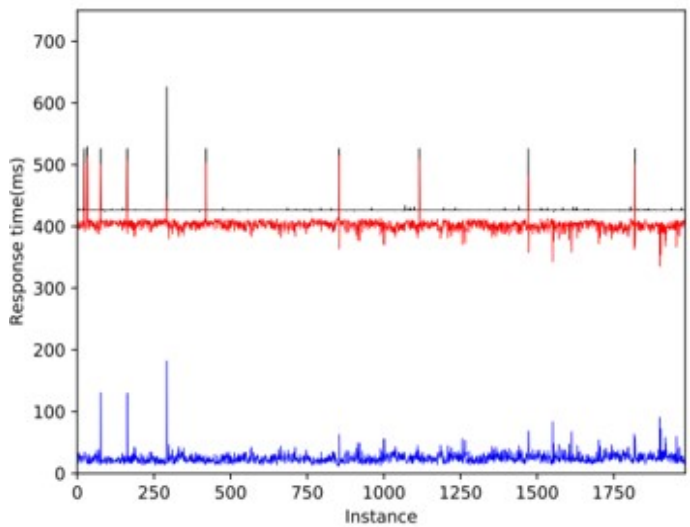


Figure 7: Obesrvation

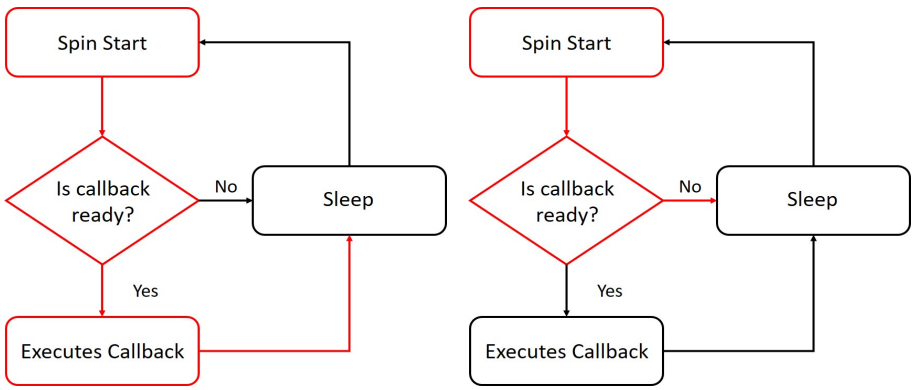
dependently, not event-driven, as described in Section 2.

- Total Node Computation Time

Each node's computation time is measured from start of the node to the end of the node. Total node computation time is a sum of all node's computation time in the critical chain for an instance.

3.2 Observation

The current Autoware is configured that the lidar sensor is coming in at 10Hz. It means instance's period is 100ms. The result of profiling Autoware through the Critical Chain is as shown in the figure 7. We observe the alignment delay has a large proportion to the E2E response time.



(a) Callback Execution (b) Callback Check Time

Figure 8: Node Execution Type

Based on this observation, our study aims to create a faster response system by increasing the given spin rate to reduce the alignment delay. Before increasing the spin rate, it is necessary to verify how a spin rate higher than the existing one can affect the system.

As shown in figure 8, execution of a node can be divided into two types. One is callback execution, the other is callback check. When a node wakes up, the spinner check the callback queue. If there is a topic data in callback queue, spinner executes the callback and sleeps. This is the callback execution. If not, just sleep. This is the callback check. And the time it takes to only check the callback is called the callback check time. If the spin rate is increased, the number of times to check the callback queue increases. It means the callback check time increases, which puts a load on the system. As a result of the measurement, the callback check time is small enough with an average of 0.000273ms, so it can be said that it does not put a significant load on the system even if the node's spin rate is increased.

However, it does not mean that the spin rate can be increased infinitely. It can be

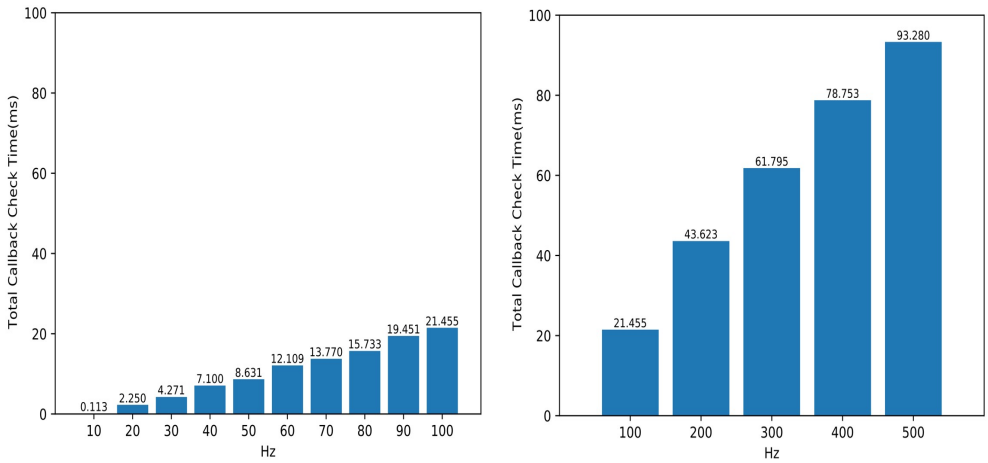


Figure 9: Total Overhead Time(Voxel Grid Filter)

checked through figure 9 and figure 10. figure 9 is the total sum of callback check times that occurred while executing 750 instances in the Voxel grid filter node according to the spin rate. And figure 10 measures the ratio of callback check time and callback execution time that occurred while executing 750 instances in the same Voxel grid filter node as figure 9 according to the spin rate.

Through the above two figures, a non-negligible amount of callback check time occurred at a spin rate over a certain Hz, which means that the spin rate cannot be increased indefinitely.

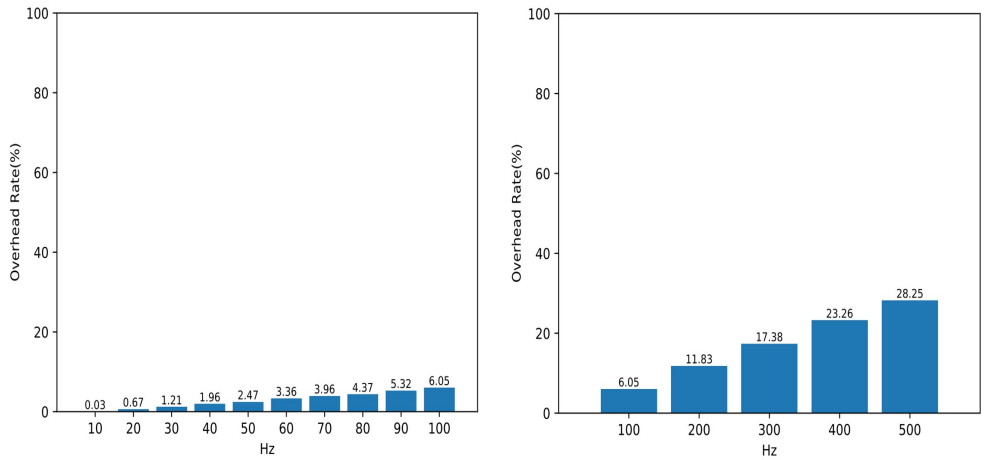


Figure 10: Overhead Rate(Voxel Grid Filter)

4 Proposed Approach

Our approach is to increase the spin rate of all the critical chain nodes to be the same to reduce the alignment delay. And what we ultimately want to achieve through this is to reduce the E2E response time of the critical chain. In Section. 3, we observe the callback check time is not negligible above a certain Hz. We model an objective function and propose an algorithm to find an appropriate spin rate through it.

4.1 Assumption

As we mentioned in Section 3, it is challenging to precisely and accurately profiles Autoware. So, some assumptions are needed to model the objective function.

- All nodes in the Autoware can preempt the current node :

  Autoware performs on Completely Fair Scheduler(CFS), the Linux kernel's default scheduler. Since CFS operates similarly to the round robin method, it assumes that all nodes can preempt the current node.

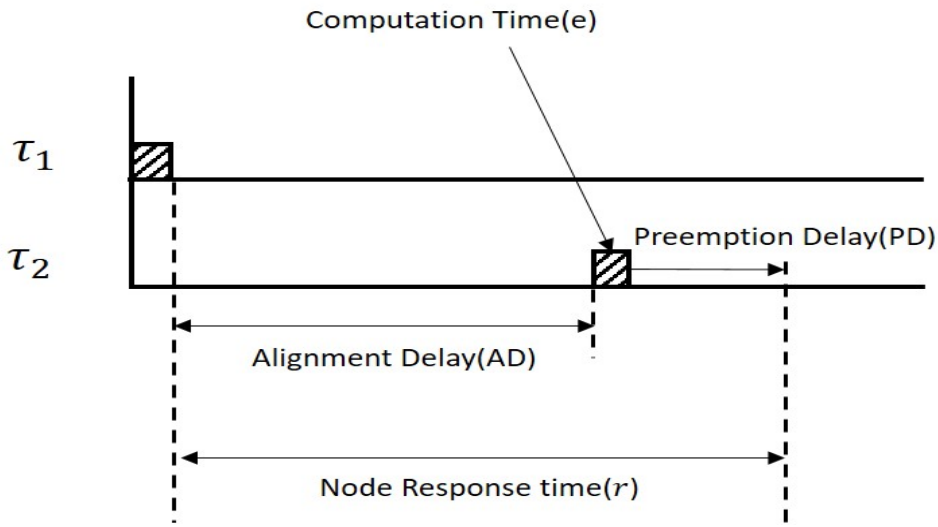


Figure 11: Node Response Time

- Autoware performs on m multi-core hardware environment.
- There are n nodes in the Autoware system $\tau = (\tau_1, \tau 2, \tau 3, \cdots, \tau_n)$
- There are k nodes in the Critical Chain $\tau_c = (\tau_c 1, \tau_c 2, \tau_c 3, \cdots, \tau_c k),\ \tau_c \subset \tau$
- Node's p is defined by spin rate and node's e is computation time.
- There is a callback check time set of τ_c,

$$e_{cbc} = (e_{cbc1}, e_{cbc2}, e_{cbc3}, \cdots, e_{cbck})$$

4.2 Objective Function

The E2E response time of the critical chain can be expressed by the following objective function. Through this objective function, we get the E2E response time according to the spin rate(SR). The E2E response time consists of the sum of the response

time of critical chain nodes.

$$R(SR) = \sum_{i \in \tau_c} r_i(SR) \tag{1}$$

$$r_i(SR) = e_i + AD_i(SR) + PD_i(SR) \tag{2}$$

Figure 11 shows the node response time. The response time of the ith critical chain node is composed of computation time, alignment delay function(AD), and preemption delay function(PD).

$$AD_i(SR) = \begin{cases} 0 & if, i = \tau_{c1} \\ \frac{1000}{SR} & otherwise \end{cases} \tag{3}$$

The spin rate determines the alignment delay. However, the alignment delay of the first node in the critical chain is 0 because no alignment delay occurs.

$$\begin{gathered} Initial\ PD_i = e_i, \\ PD_i \leftarrow e_i + \frac{1}{m} \sum_{j \in (\tau - \tau_i)} \left(\left\lceil \frac{PD_i}{p_j} \right\rceil e_j \right) \\ (e_j = \begin{cases} e_{cbcj} & if, j \in \tau_c \\ e_j & otherwise \end{cases}, p_j = \begin{cases} \frac{1000}{SR} & if, j \in \tau_c \\ p_j & otherwise \end{cases}) \end{gathered} \tag{4}$$

The equation (4) is the preemption delay function. We compute the approximate preemption delay of each node using a method inspired by response time analysis[5][6][7][8].

Algorithm 1 Proposed Algorithm

Input: τ(entire system node set), τ_c(critical chain node set), e_{cbc}(callback check time set of critical chain), m(cores)

Output: spin rate of critical chain

```
1: min_R = 10000
2: min_SR = 0
3: for SR = 10 to 1000 do
4:     find a response time(R) for SR using Eq (1)
5:     if R < min_R then
6:         min_R = R
7:         min_SR = SR
8:     end if
9: end for
10: return min_SR
```

The initial value of this function is the ith node's computation time. With this initial value, calculate the preemption delay recursively. According to the assumption, all nodes of Autoware can preempt each other, so the computation time and period used in this function vary depending on whether the corresponding node is a critical chain node. In the case of computation time, if it is a critical chain node, it is determined by the callback check time set mentioned in the assumption; otherwise, it has the computation time of the node. In the case of period, if it is a critical chain node, it is determined by spin rate; otherwise, it has the period of the node. This recursive function terminates when the result is no longer incremented.

4.3 Proposed Algorithm

In summary, our proposed heuristic algorithm can be represented as Algorithm 1. The algorithm increases the spin rate by 10 from 10 to 1000, and uses Eq(1) to find the E2E response time for a given spin rate. The algorithm finds the spin rate that minimizes the E2E response time while repeating the for loop.

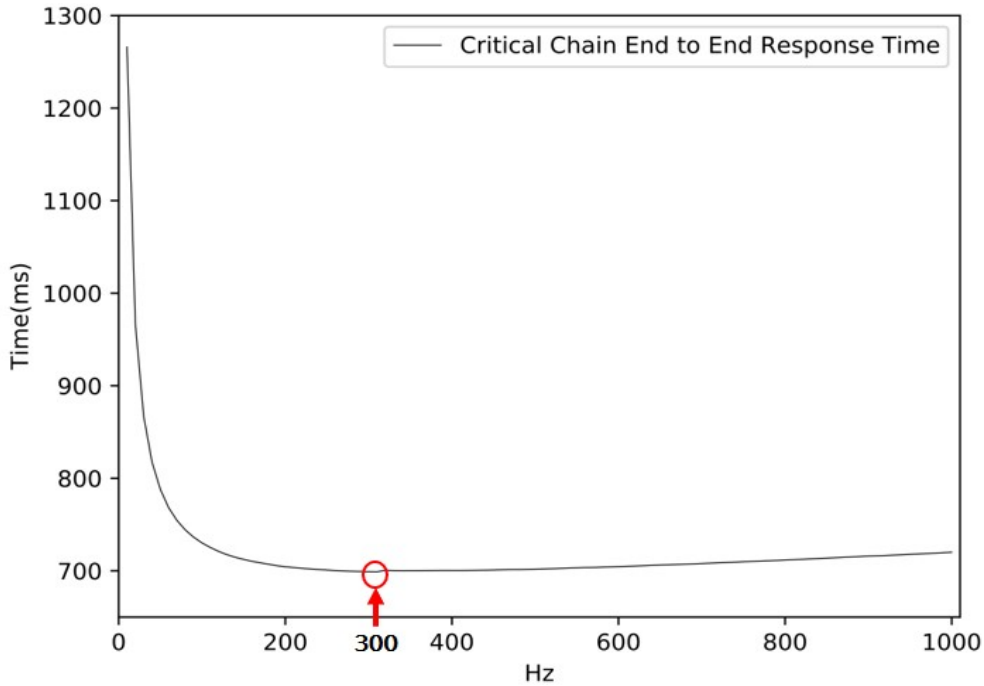


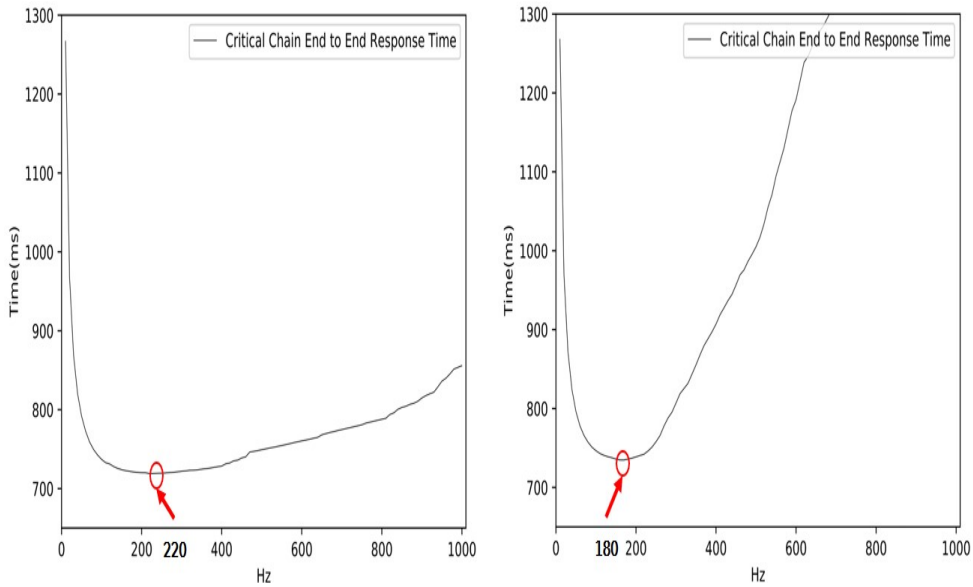
Figure 12: Algorithm Result

5 Evaluation

In this section, we verify the proposed algorithm and apply it to Autoware based on this result. And compare the difference between the improved and existing systems in E2E response time evaluation and a simulator environment.

5.1 Objective Function Evaluation

The result for the critical chain obtained through the proposed algorithm are shown in figure 12. The spin rate of the critical chain nodes obtained through the proposed algorithm is 300Hz. So, in the experiment below, we adjust the spin rate of all critical chain nodes to 300Hz. And to verify the algorithm, we experiment with the case



(a) Experiment 1(0.5ms) (b) Experiment 2(1.0ms)

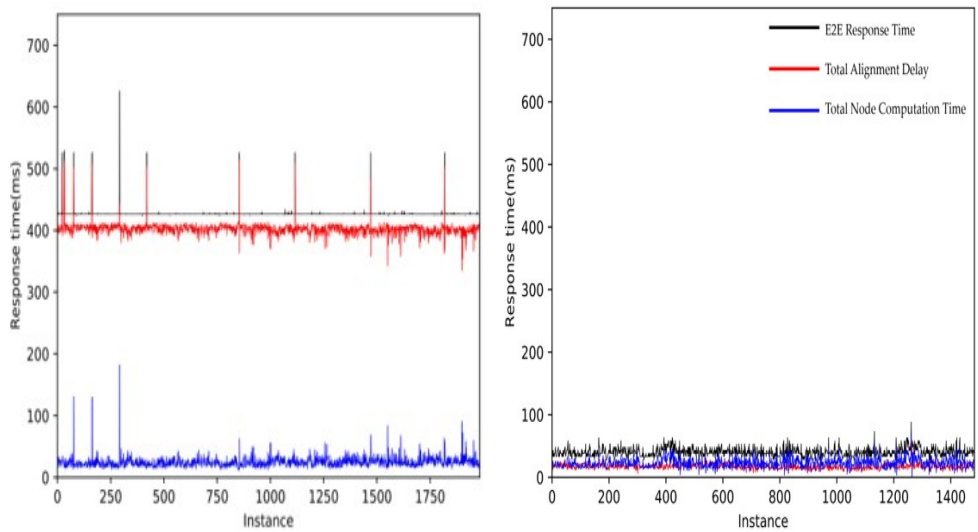
Figure 13: High Callback Check Time Experiment

where the callback check time of a certain node is large.

In the first experiment, there is a node with a 0.5 ms callback check time in the critical chain. On that condition, as you can see on 13(a), the spin rate of all critical nodes is 220Hz. In the second experiment, there is a node with a 1.0 ms callback check time in the critical chain. On that condition, as you can see on 13(b), the spin rate of all critical nodes is 180Hz. As shown in figure 13, the larger the callback check time, the lower the Hz, so the proposed algorithm works well.

5.2 Response Time Evaluation

The E2E Response time of the Initial System has been shown in figure 6. Figure 14 compares the existing system and the system to which our method is applied. In the case of the existing system, the average E2E response time is 426.93ms, the aver-



(a) Initial Spin rate (b) Assigned Spin rate

Figure 14: E2E Response Time Evaluation

age alignment delay is 402.27ms, and the average total computation time is 24.66ms. On the other hand, in the case of the Optimized system, the E2E response time is 45.53ms, the average alignment delay is 21.23ms, and the average total computation time is 24.30ms. The system in which the proposed method is applied shows a performance improvement of about 18 times in the alignment delay and about nine times in the case of the E2E response time. Below, we would like to report what kind of performance improvement was actually made through simulation.

5.3 Autonomous Driving Evaluation Setup

Experimental environment configuration is as in figure 15. The desktop computer consists of an Intel i7-8700 CPU, NVIDIA GTX 1080 GPU, and 32GB of RAM. The simulator and Autoware run on a single desktop computer and top of the Linux operating system. The simulator sends the sensor information to Autoware, and Au-

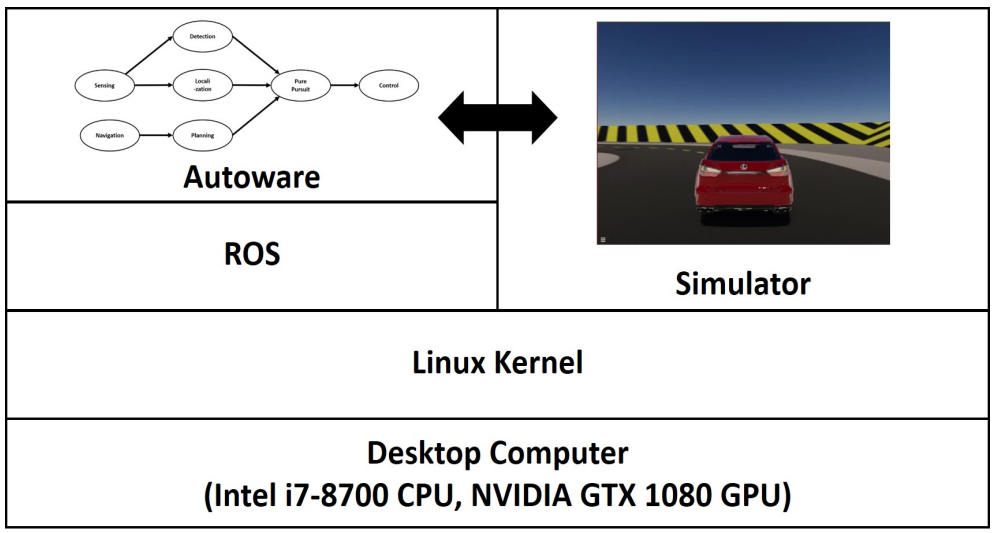
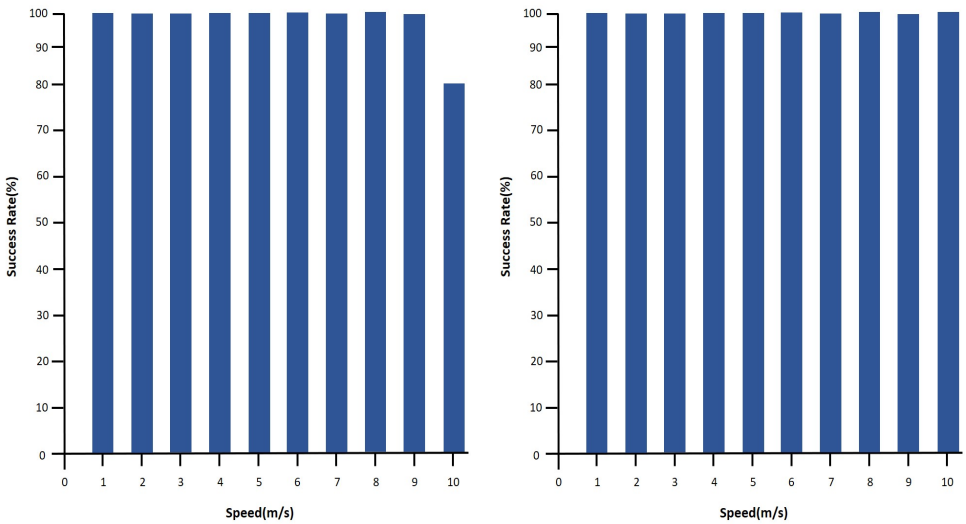


Figure 15: Experiment Environment

toware sends the control information calculated by the received sensor information to the simulator. At this time, the simulator and Autoware communicate through TCP/IP. The experiment is conducted on a circular track of four corners, and a simulation vehicle drives on the way. The simulation vehicle drives to verify how the response time affects the system. The simulation experiment tests whether Lane Keeping is successful with all Autoware nodes running. The success criterion is verifying whether Lane keeping was successful for 1400 instances. Ignore stepping on the inside line when turning corners because it is a limitation of the control algorithm.

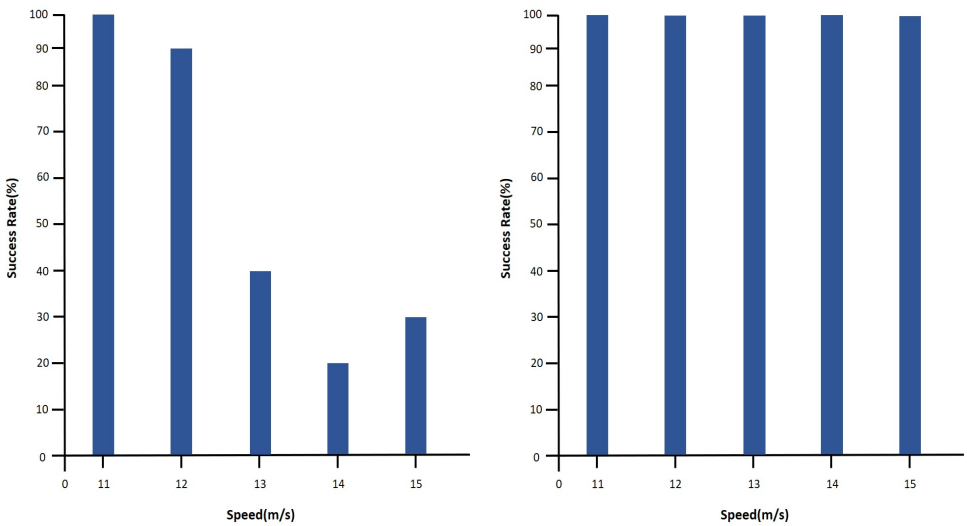
5.4 Autonomous Driving Evaluation Result

Figure 16 shows the success rate according to the speed when increases the speed by 1m/s from 1m/s to 10 m/s, and figure 17 shows the success rate from 11 m/s to 15 m/s. Both systems succeed without failure in most cases, up to 10 m/s. However, as can be seen from the figure 17, the success rate of the existing system is less than



(a) Initial Spin rate (b) Assigned Spin rate

Figure 16: Simulation Evaluation(10m/s)



(a) Initial Spin rate (b) Assigned Spin rate

Figure 17: Simulation Evaluation(15m/s)

half at the speed of 13m/s or higher, but the system to which the proposed method is applied succeeds at all speeds.

6 Conclusion

This paper proposed a method to improve Autoware, a ROS-based autonomous driving platform. Our proposed algorithm is to improve performance by suggesting a critical chain among the nodes constituting Autoware and optimizing the spin rate of the critical chain nodes. We explained how our proposed method improved the existing system's performance through the E2E response time and the simulation results. In the case of response time, the proposed system improvement approach improves the existing system by about 9times or more. In the case of the simulation, the performance improved by about 40% compared to the current system.

References

[1] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[2] Hongxing Wei, Zhenzhou Shao, Zhen Huang, Renhai Chen, Yong Guan, Jindong Tan, and Zili Shao. Rt-ros: A real-time ros architecture on multi-core processors. *Future Generation Computer Systems*, 56:171–178, 2016.

[3] Yukihiro Saito, Futoshi Sato, Takuya Azumi, Shinpei Kato, and Nobuhiko Nishio. Rosch: real-time scheduling framework for ros. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 52–58. IEEE, 2018.

[4] Hongxing Wei, Zhen Huang, Qiang Yu, Miao Liu, Yong Guan, and Jindong Tan. Rgmp-ros: A real-time ros architecture of hybrid rtos and gpos on multi-core processor. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2482–2487. IEEE, 2014.

[5] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 149–160. IEEE, 2007.

[6] Lars Lundberg. Multiprocessor scheduling of age constraint processes. In *Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No. 98EX236)*, pages 42–47. IEEE, 1998.

[7] Alan Burns and Andrew J Wellings. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

[8] Björn Andersson and Jan Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proc. of the IEEE Real-Time Systems Symposium, Work-in-Progress Session*, page 19, 2000.

요약(국문초록)

오픈소스 로봇 운영체제(ROS)는 자율주행과 같은 복잡한 시스템에서 연구되고 있다. ROS를 기반으로 하는 복잡한 시스템을 위해 ROS에 실시간 이식을 위한 많은 연구가 이루어졌다. 그러나 이러한 방법은 사용하기 어렵고 복잡한 절차가 필요하기 때문에 사용자 친화적이지 않다. 이를 위해 본 논문에서는 ROS의 구조를 수정하거나 다른 복잡한 절차를 추가하지 않고 ROS의 성능을 높이기 위하여 응답 시간에 주목하였으며, ROS의 특성 중 하나가 응답시간 지연을 발생시키는 것을 발견하였다. 본 논문에서는 ROS의 특성을 이용하여 사용자의 편의성을 갖춘 응답 시간 향상 방법을 설명한다. 마지막으로 본 논문에서 제시하는 방법의 성능을 몇 가지 실험을 통해 보인다.

주요어 : 자율주행, 최적화, 시스템 분석
학 번 : 2020-22934