Sophisticated Offline Analysis of Teams of Autonomous Mobile Robots

Dirk Thomas, Dorian Scholz, Simon Templer, Oskar von Stryk

Simulation, Systems Optimization and Robotics Group, Technische Universität Darmstadt
Hochschulstr. 10, 64289 Darmstadt, Germany
{dthomas|scholz|templer|stryk}@sim.tu-darmstadt.de

***Abstract*— Debugging control software for an autonomous mobile robot is a difficult and time consuming task. But it gets even harder, when analyzing a whole team of robots and their team behavior. The quality of the robots' decisions based on their current knowledge cannot be judged anymore by merely looking at their actions from the outside.**

In this paper an approach for collecting intrinsic and extrinsic data of a team of robots during full operation and analyzing this data offline is described. Since the amount of data to be collected is quite large a method for automated and semi-automated analysis is shown - making it possible to detect known problems in an automated process and mark potentially interesting events for manual review.

Furthermore a solution to reuse existing single robot debugging tools on teams of robots, without rewriting each tool, is presented.

I. Introduction

The software enabling robots to act autonomously is becoming increasingly complex. Different algorithms from various domains are being integrated to perform a single task or scenario. Debugging tools for such systems are essential for building working solutions. In the context of teams of mobile agents this task becomes even more complex due to the distribution and limitation of the used platforms and infrastructure.

In this paper a method for logging various different intrinsic and extrinsic data and making it available for debugging tools is presented. Since it cannot always be assumed that a global system time is available special attention has to be payed to data synchronization.

Especially in the scenario of multiple cooperating autonomous robots in a dynamic environment, the amount of intrinsic information generated during robot operation is tremendous. Analyzing and reviewing all this data manually is extremely time consuming and error-prone, and quickly becomes unmanageable and inefficient. Algorithms for automated analysis of the information have to be applied in order to make the review process more efficient. An important aspect of the described approach is how existing debugging tools, designed for single robot applications, can be reused in the context of teams of agents.

The remainder of this paper is structured as follows: In section 2 the need for debugging complex applications is motivated and the specific requirements for the scenario of teams of mobile robots are given. In section 3 the authors' implementation of the intrinsic and extrinsic data logging is presented as well as the approach to synchronize this information. After this, an improved concept to ease navigation around the large amount of data and automatically analyze the information is presented in section 4. The section continues describing the approach to reuse existing debugging functionality and presents some debugging tools specific to teams of agents. The paper closes with concluding remarks and an outlook in section 5.

II. Motivation and Requirements

As background for specifying the requirements for the described work, the scenarios where the method was applied are described in short.

A. Scenarios

The authors' group is participating in two leagues at the RoboCup competition, which is an international research and education initiative. The first scenario is an autonomous soccer game played by humanoid robots. This provides an environment of standardized benchmarks for autonomous robots to foster artificial intelligence and robotics research where a wide range of methodologies can be examined and integrated. The ultimate goal of RoboCup Soccer is to develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer. The authors are participating in this competition as part of the team Darmstadt Dribblers [1], which is a two time champion in the Humanoid Kid-Size League, winning it in 2009 and 2010.

Additionally the work presented here was applied by Team Hector [2] participating in the RoboCup Search And Rescue League since 2009.

Both scenarios have several aspects in common which are similarly present in any complex autonomous systems: The developed robot software applications integrate several different domain specific algorithms and functionality to provide an integrated overall system. The functional components vary based on the targeted objectives and cover different approaches for processing and filtering sensor data, self localization and mapping as well as behavior and motion control. Each scenario has its own focus but since the requirements are continuously increasing the robot application software becomes more and more complex. The successful combination of all these different parts in an integrated robot application software

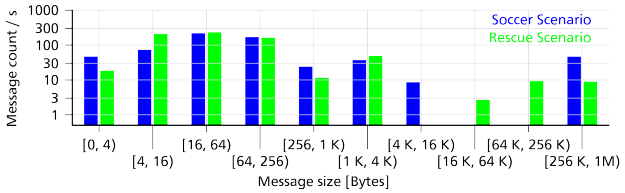


Fig. 1: The number of messages exchanged per second between the numerous components inside the robot control software

is a significant challenge and requires a great amount of debugging.

B. Storing Intrinsic Data Locally

The software for both scenarios is based on `RoboFrame` [3] which provides message-oriented middleware (MOM) functionality. Due to the complexity of the application the amount of data exchanged between the different functional components is tremendous. Fig. 1 shows a rough overview how many messages are communicated in the two scenarios and how much data is involved. All of this data may be necessary for thorough debugging capabilities. Because of limited wireless network connectivity this information has to be logged locally on each agent. Since even the local storage is quite limited in size and write speed depending on the scenario it may only be possible to log a subset of the data for later review. The recording must be performed as efficient as possible in order not to affect the runtime behavior of the robot control software which would render the results useless. The logged data from multiple robots must be synchronized later on for a combined offline analysis.

C. Navigating Large Amounts of Data

In the considered scenarios during ten minutes of operation approximately half a million messages are exchanged and potentially recorded. Due to the large amount of data the navigation through this collection is also of high relevance. Besides simple timeline-based navigation methods it should be possible to mark important points in the log automatically using different algorithms. This supports the developer in reviewing somehow interesting or relevant periods manually.

D. Reusing Single-User Debugging Tools

In an existing project various different debugging tools have already been developed since every algorithm and data requires its own graphical user interface for visualization and debugging. Most of them are unable to work properly when receiving data from multiple agents simultaneously, e.g., a dialog displaying the camera images usually only handles data from a single robot. Rewriting or adapting them for the use with teams of robots would be a time-consuming and repeated task and should therefore be avoided.

E. Enhancement with Extrinsic Information

Since even the comprehensive intrinsic data from all robots may not be sufficient to gain a good overview of the environment during the analysis, the integration of additional external sensors is important. In the described method multiple external video cameras are used to record the environment from external points of view. This provides valuable information for the developer when debugging and comparing the intrinsic data with the real situation of the environment.

F. Existing Approaches

In this area only a small number of publications exist since the application complexity needs to be quite high until debugging tools of this sort are required. In many simpler scenarios and applications the debugging solution for single robots are sufficient.

But there are some approaches especially from the context of RoboCup where debugging of teams of agents or integration of external data is done. E.g. the LogViewer [4] of the former GermanTeam [5] in the Four-Legged League is capable of integrating a single external video to assist the debugging. But the debugging capabilities are limited to the functional component for behavior control and do not cover other data collected in the applications.

Another solution called Vizard [6] was implemented by the team 'The Ulm Sparrows' who participated in the Middlesize League of RoboCup. Although their tool helps in debugging and analysis of the robots software, several aspects are still to be addressed. Synchronized playback for distributed recorded data and the integration of extrinsic information need to be added to work with teams of autonomous robots acting in a highly dynamic environment.

On the other side there are tools like the Interaction Debugger [7] which is used to analyze human-robot interaction. It enables integrating various different intrinsic and extrinsic data and replaying and visualizing them in a synchronized way. The drawback of that approach is the need for a central server for logging the data and a separate capturing PC to record videos which is impracticable for mobile robots restricted to wireless communication.

III. Logging Data

A. Intrinsic Data

As the utilized message-oriented middleware is based on the publish-subscribe paradigm all intrinsic data is passed as messages between functional components. This simplifies integrating an additional subscriber which receives all data of interest and stores it, e.g., in a binary logfile. But the amount of data to be exchanged is quite significant - in particular the raw data of the camera images amounts to nearly one gigabyte per minute.

The used hardware platforms have three limitations relevant to the data recording: First the overall space for data storage is limited to a few gigabytes; Second the throughput to write data to the storage medium is quite restricted so that the amount of data stored per second must be reduced; Third and most

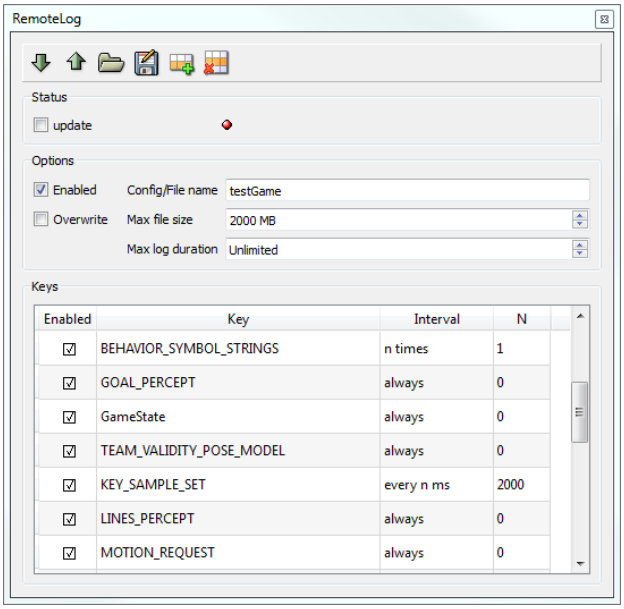


Fig. 2: The dialog for configuring logging functionality.

importantly the processing power needed to log the data should not decrease the control software performance. Depending on the debugging focus various different kinds of information are relevant. Therefore, the type of messages to be recorded can be configured as well as their frequency.

This functionality was implemented on top of `RoboFrame` using the inherent feature for requesting specific messages with a given frequency. If the communication layer had not provided the functionality for publisher-side throttling, a huge overhead of published data had decreased the performance and therefore limited the amount of logged messages even further.

Besides this some usability features have been integrated like stopping the recording when the storage device is nearly full or triggering specific logging functionality through a remotely connected graphical user interface (see Fig. 2).

B. Synchronization

If the multiple robots involved do not share a global time, which is the case in the mentioned scenario, the need for synchronizing the logged information arises. Instead of doing this manually for every single logfile, a method to determine the time offsets between the teammates was implemented. Therefore, every robot continuously broadcasts its current time every N seconds and responds to each broadcast of a teammate with a unicast using the wireless LAN interface and UDP protocol. Based on this information a robot can calculate the offset between both clocks by assuming a similar delay in sending the broadcast and receiving the answer. The procedure is derived from the simple network time protocol (SNTP) [8]. But instead of adjusting the system time of one host, the calculated offset between the two hosts is written to the logfile for offline interpretation.

The calculated time offset between the two robots is additionally broadcasted to all teammates to assure that each teammate is aware of all pairwise time offsets. This becomes important when you consider the case of substituting robot A with another robot C during operation, e.g., due to hardware or software problems. Since the robots A and C never operated at the same time they were not able to exchange timestamps. This makes it impossible to synchronize their logfiles automatically, without using the propagated offsets of robot B with each teammate A and C. But with this information it becomes possible to only load the logfiles from robot A and C and still synchronize them automatically, as they contain the time offsets between A and B as well as between B and C.

The accuracy achieved by this simple method is high enough for the mentioned scenario but may not be for other areas. Obviously, a time drift is not addressed by this method as it was not required in the described scenarios due to the short length per mission of only several minutes.

C. Extrinsic Data

In addition to the intrinsic data of the robots, arbitrary extrinsic data can be collected and used for later analysis. In the described scenario a video camera was use to record the environment of the team of robots. This data was recorded on a general purpose camera with a resolution of 720p in MPEG-4 format. The video can be replayed synchronously with the intrinsic data as can be seen in Fig. 3

Extrinsic data from multiple videos or cameras can easily be integrated as long as the output format can be handled by the multimedia library FFMPEG. But for the described scenarios one camera was sufficient to show the relevant parts of the environment. Since synchronizing the video with the intrinsic data would require complex algorithms this has yet to be done manually *once* per video.

IV. Automated Analysis

Due to the quantity of messages that is archived even in short periods of time, the task of looking through the information becomes increasingly complex. The chronological data can be navigated using a time- or frame-based time line. But since the number of messages can easily become overwhelming, a different concept for navigating through this data is necessary to allow an efficient review of the available information.

To ease the process of navigating through the data additional information must be provided to the user. This can be achieved through an automated extraction of prominent points in time, which supports the user in browsing the information. These points in time are named *events* in the following. Arbitrary algorithms can be used to extract events offline based on the recorded data.

In Fig. 4 a partial view of a list of several hundred events is shown whereas the complete list of messages would have been a thousand times longer.

In the described scenario of soccer playing robots the game state (including the playtime, the score, penalties etc.) is continuously recorded to the logfile. A custom algorithm is used to generate events at the points in time where the

Fig. 3: Model viewer showing the robots' and ball's positions (top) synchronized with the external camera image (bottom) including some overlays generated from the logged data.

Player Events

All locations | All generators

	Time	Location	Event	Generator
5	156.53	Global (Player3_17)	Ready	GameStateEvents
6	167.03	Global (Player3_17)	Set	GameStateEvents
7	223.57	Global (Player3_17)	Playing	GameStateEvents
8	231.32	testGame_1.mp4	started recording	Video
9	241.36	Player3_17	fell on his side	RobotStatusEvents
10	253.32	Player3_17	fell on his face	RobotStatusEvents
11	299.13	Global (Player3_17)	Ready	GameStateEvents
12	299.13	Global (Player3_17)	0:1	GameStateEvents
13	318.14	Global (Player3_17)	Set	GameStateEvents
14	321.14	Global (Player3_17)	Ready	GameStateEvents
15	335.14	Global (Player3_17)	Set	GameStateEvents

Fig. 4: List of events easing navigation of recorded data.

state changed, which enable a better navigation through the recorded data.

A. *Identify Known Problems Automatically*

Besides events which are only used for a better overview of the recorded data additional events are generated to identify known issues automatically. Such information permits the user to focus on relevant periods and efficiently review the recorded data as these events identify situations which are most relevant for manual analysis. Several different algorithms have been implemented to make the analysis tasks of the selected scenarios more efficient.

B. *Discontinuity of Self Localization*

The task of self localization for mobile robots is to determine the current position and orientation of the robot. As long as the mobile platform is not manually repositioned, it can only move in space in a continuous manner and with a limited speed. Therefore, if the position of the robot changes considerably in a short timeframe, the determined location must either have been wrong before or afterwards. An algorithm detects such discontinuities using predefined thresholds and generates events at these points in time.

C. *State Oscillation in Behavior Control*

Another example, which shows the gain of the automated analysis, is from the domain of behavior control. In the considered scenarios the behavior is implemented using hierarchies of finite state machines defined in XABSL [9]. A common problem with this approach exists when the state machine continuously oscillates between multiple states. This is usually due to insufficient hysteresis in the transition constraints, which is especially valid for robotics due to uncertainties.

An algorithm has been implemented to detect such occurrences in the behavior generating corresponding events. These problems are especially difficult to find manually as the timeframe of the occurrence is very short. Thus the time to identify these cases is reduced significantly and users can concentrate on tracking down the source of the issues.

V. Reusing Existing Debugging Tools

For existing projects various different debugging tools exist which are normally suited for communicating with a single robot only. For example a dialog in the graphical user interface for displaying the image of the robot's camera. The dialog might provide additional support for debugging the internals of the used image processing. But for obvious reasons it is not capable of handling data from multiple robots simultaneously (see Fig. 5).

The optimal reuse of the existing functionality would be to instantiate it multiple times and let every instance just receive data from one specific robot. This could be achieved by altering all existing dialogs and implement some controls for choosing the data source to use. Since this process is very time consuming and repetitive, a solution provided by the middleware for all existing dialogs is preferred. But currently

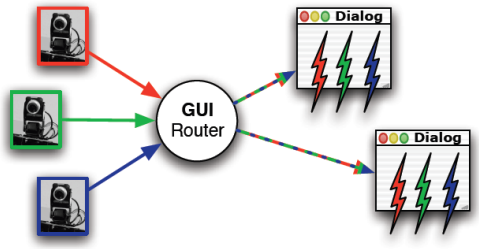


Fig. 5: Visualization of intrinsic data from multiple robots without filtering.

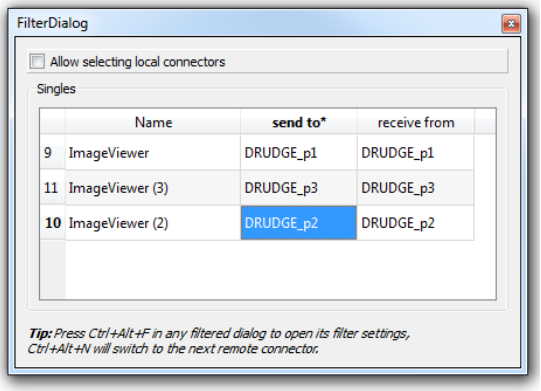


Fig. 6: The central view for specifying the correlation between views and particular robots

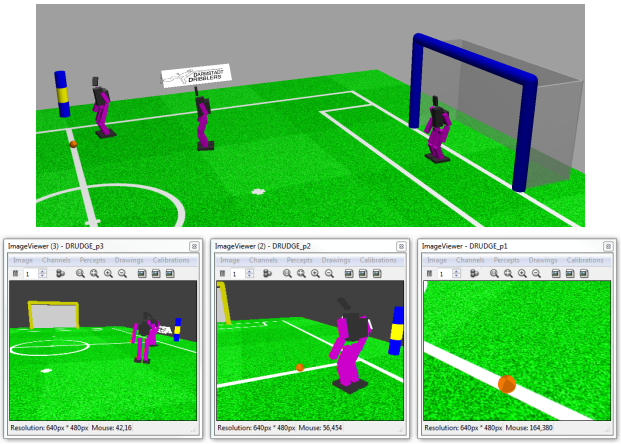


Fig. 7: Multiple instances of a single view, each communicating with a particular robot

TABLE I: Performance impact of recording large messages in the soccer application measured on an AMD Geode

Recorded Messages	Avg. Frame Rate	Avg. Data Rate
None	16.43 Hz	-
All percepts and models	16.32 Hz	142 Kb/s
- plus one image every 5 seconds	15.80 Hz	262 Kb/s
- plus one image every second	13.07 Hz	742 Kb/s

nearly none of the existing message-based middleware provide the necessary meta-information and allow to redirect the data flow accordingly without touching any dialog implementation.

In `RoboFrame` the interaction between each dialog and the middleware is handled by a gateway. The details about the specific API design of the communication layer to support such approaches is described in another paper [10]. Following the concept of dependency injection, the gateway is exchanged with a custom implementation which restricts each dialog to communicate with a single robot. The setup which dialog should communicate with which particular robot is done using a central configuration view (see Fig. 6).

This policy is then enforced by the custom gateways altering the information exchange between the dialogs and the middleware. Therefore, the views only receive messages from a single robot as depicted in Fig. 7, which shows multiple instances of the same view, each displaying the camera image of a particular robot.

For increased usability the configuration cannot only be changed using the aforementioned view but also using keyboard shortcuts. Thereby, the correlation of the currently selected view of a particular robot can be altered without switching to another view by using a single keystroke only.

VI. Results

The described methods for collecting intrinsic and extrinsic information on distributed systems and automatically synchronizing these data sources for offline analysis have been used since 2009 in the authors group.

A. Performance

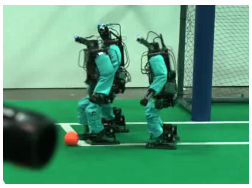
Since a large amount of data is logged locally on the robot the performance impact has to be considered. Extensive measurements of the performance depending on the set and frequency of recorded data have been made and are summarized in TABLE I.

On the previously used AMD Geode 500 MHz it was not feasible to record e.g. all raw images processed. Mainly the serialization of the large amount of data required too many CPU resources and therefore had a negative impact on the processing speed of the application.

In the meantime the PC hardware of the humanoid robots has been upgraded to an Intel ATOM with 1.6 GHz, using an USB stick as storage for the logged data. On this new platform the amount of recorded data could even been increased without harming the application performance. It is processing images at a constant rate of 30 frames per second which is the limit of the used USB camera. The main limiting factor for logging on this configuration is the data throughput rate that can be constantly achieved when writing to the USB stick. To avoid any performance penalties through wait states when writing log data, only one image every five seconds is recorded in addition to the complete intrinsic data.

(a)

(b)



(c)

Fig. 8: Difficulty to identify the source of problems in complex applications. All three robots seem to approach the ball obstructing each other in (b) and (c)

B. Automation of Analysis

The required efforts for analyzing the comprehensive amount of data have been significantly reduced. First the distributed recorded data is synchronized automatically which frees the user from this time-consuming task. Secondly several tasks for the analysis are automated using application specific algorithms. They are used to generate events to improve navigation and identify known problems. This enables efficient analysis and permits an objective and reliable evaluation.

C. Multiple Robots

Reuse of the existing debugging tools for teams of autonomous mobile robots enables thorough manual analysis of the collected data. It has been successfully applied in both described scenarios at RoboCup and can be seen in use when analyzing the final game of RoboCup 2010 [11]. These tools are used in every test run of the robots and even during real competitions. They enable the developers to track down issues which would be impossible to resolve without the combination of the information about the internal state of the applications and the extrinsic data. Some issues can only be identified when all data from multiple teammates can be analyzed synchronously.

For example, in the situation depicted in Fig. 8, at first glance all robots seem to approach the ball obstructing each other. The initial assumption of the developers was either a error in the dynamic role assignment, which should avoid multiple robots approaching the ball concurrently, or a problem with the team communication, whereby each robot would perform as if it was the only agent on the field.

Based on the comprehensive information available for debugging the behavior, it was possible to quickly identify the reason for the problem. Both team communication and dynamic role assignment worked flawlessly. The problem showed to be that the target positions of the supporting robots were set to unreasonable coordinates near their own penalty area. This led to the observed clustering and obstruction of the goalie which intended to clear the ball. Based on the insight obtained through the analysis tools, this issue was resolved easily.

VII. Conclusions

The continuous growth in complexity in teams of autonomous mobile robots puts high demands on debugging tools. Without a comprehensive solution for this task the creation of robust and error-free software is unfeasible.

In this paper a method for logging arbitrary intrinsic and extrinsic data in a flexible way has been presented. In this process the necessary information for automatically synchronizing the intrinsic data have been collected to ease later utilization.

These methods have been implemented as part of `RoboFrame` and were successfully applied in multiple scenarios. Extensive measurements of the performance depending on the amount of logged data were made to find a good trade-off between performance impact and completeness of available intrinsic data for later debugging.

Furthermore, a concept for navigating the extensive amount of recorded data using events has been presented. Various algorithms extracting different types of events have been implemented for the considered scenarios, which demonstrate the reduced time required for reviewing the data manually and enable identifying known issues reliably and efficiently.

References

[1] (2010) The Darmstadt Dribblers website. [Online]. Available: http://www.dribblers.de/

[2] (2010) The Team Hector website. [Online]. Available: http://www.gkmm.de/rescue

[3] S. Petters, D. Thomas, and O. von Stryk, "RoboFrame - A Modular Software Framework for Lightweight Autonomous Robots," in *Proceedings of the Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA, October 29 2007.

[4] M. Risler, "Behavior control for single and multiple autonomous agents based on hierarchical finite state machines," Ph.D. dissertation, Technische Universität Darmstadt, May 15 2009.

[5] (2010) The GermanTeam website. [Online]. Available: http://www.germanteam.org/

[6] H. Utz, G. Mayer, and G. K. Kraetzschmar, "Middleware Logging Facilities for Experimentation and Evaluation in Robotics," in *27th German Conference on Artificial Intelligence*, Ulm, Germany, September 2004, workshop on Methods and Technology for Empirical Evaluation of Multiagent Systems and Multirobot Teams.

[7] T. Kooijmans, T. Kanda, C. Bartneck, H. Ishiguro, and N. Hagita, "Interaction debugging: an integral approach to analyze human-robot interaction," in *HRI '06: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. New York, NY, USA: ACM, 2006, pp. 64–71.

[8] D. Mills, J. Martin, J. Burbank, and W. Kasch, "RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification," June 2010. [Online]. Available: http://tools.ietf.org/html/rfc5905

[9] M. Lötzsch, M. Risler, and M. Jüngel, "XABSL - A Pragmatic Approach to Behavior Engineering," in *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*, Beijing, China, October 9-15 2006, pp. 5124–5129.

[10] D. Thomas and O. von Stryk, "Efficient communication in autonomous robot software," in *Proc. IEEE/RSJ Intl. Conf. on Inteligent Robots and Systems (IROS)*, 2010, p. accepted.

[11] (2010) The video analysis of Darmstadt Dribblers playing the final at robocup 2010 on YouTube. [Online]. Available: http://www.youtube.com/watch?v=puL3XH_So2k