Mutual Influences in Self-adaptive and Autonomously Learning Systems

Dissertation zur Erlangung des Doktorgrades
der Fakultät für Angewandte Informatik
der Universität Augsburg

Stefan Rudolph

Gutachter:	Prof. Dr. Jörg Hähner
	Prof. Dr. Bernhard Bauer
	Prof. Dr. Bernhard Sick
Tag der mündlichen Prüfung:	20. August 2019

Abstract

Since the 1990s, we see an incremental miniaturization of computers, and, consequently, a computerization of everyday objects. This trend has strongly increased over the last decades and there is no sign of a reversal. The Internet of things (IoT) and cyber-physical systems (CPS) are current domains, which are a direct continuation of this development.

Most of the time it is not possible to model such a high number of devices at design-time. Therefore, Organic Computing aims to overcome the issues that appear with the control of systems with high complexity by implementing the concepts of self-adaption and self-organization in technical systems, i.e., enable the systems to adapt and manage themselves during runtime.

However, due to the complexity and size of modern systems, it is often not clear to whom the systems should adapt to or organize with. Such complexity appears because of high numbers of autonomous systems, which can bear a high degree of complexity in itself, but even more as a result of the non-trivial interactions among these systems. This leads to mutual influences which can be either direct, i.e., easily observable, or indirect, i.e., hard to observe or hidden. Not easily observable influences arise especially because of indirect couplings or influential paths that are not perceptible by the individual systems, e.g., because the environment reacts to several autonomous systems operating on it at the same time. To address these mutual influences, it is necessary to uncover them during runtime and enable the systems to self-adapt to them.

Therefore, the basic theme of this thesis is a computational approach to acquire knowledge about mutual influences among systems or system parts that allow them to be controlled more efficient by enabling interaction and averting to interfere with each other. Such a methodology should allow for backwards compatibility as well as applicability in heterogeneous systems and utilization for autonomous application during the runtime of a system.

In this thesis, the algorithmic approach to influence detection and several aspects of its application are covered. The proposed method is based on stochastic dependency measures that are applied to quantify how important the other sys-

tems are for the outcome of the system itself based on past experience, i.e., the systems measure the correlation between their reward signal and the configuration of other systems. Especially, the following dependency measures are evaluated for this purpose: the Pearson correlation, the Kendall rank correlation, the Spearman rank correlation, the distance covariance, the mutual information, and the maximal information coefficient.

The basic methodology is expanded in four ways: (i) the incorporation of mutual reactions between several components in the system by conditioning the measurement of the influence by these other components, (ii) the detection of delayed influences, i.e., influences that do not manifest immediately but after a time period, by measuring the reward signal against past configurations of other systems, (iii) the application at runtime by relying on partially randomized configurations, and (iv) the adaption of the systems to the influences on each other by employing reinforcement learning algorithms.

The approaches are evaluated on elementary use cases and two real-world applications. The latter are smart camera networks from the IoT domain and smart factories as a representative of CPS. The results show that the method can detect all influences on each of the applications. Eventually, practical advice on the application of influence detection in different system classes are given.

Keywords:
Mutual Influences, Self-Organization, Self-Integration, Self-Adaption, Machine Learning, Organic Computing, Interwoven Systems

Zusammenfassung

Seit den 1990er-Jahren beobachten wir eine zunehmende Miniaturisierung von Computern und daraus folgend eine zunehmende Computerisierung von Alltagsgegenständen. Dieser Trend hat sich über das letzte Jahrzehnt erheblich verstärkt und bisher gibt es kein Zeichen der Umkehrung. Das Internet of Things (IoT, zu dt. Internet der Dinge) und Cyber-Physical Systems (CPS, zu dt. Cyber-physikalische Systeme) sind aktuelle Bereiche, die direkte Folgen dieser Entwicklung sind.

In vielen Fällen ist es nicht möglich eine solche Anzahl an Geräten zur Entwurfszeit zu modellieren. Deswegen ist es das Ziel des Organic Computing die Probleme, die bei der Regelung von Systemen mit hoher Komplexität auftreten, zu lösen, indem Konzepte wie Selbstorganisation und Selbst-Adaption in technischen Systemen realisiert wird, also Systeme in der Lage sind sich selbst zur Laufzeit zu managen und anzupassen.

Bedingt durch diese Komplexität und Größe von modernen Systemen ist es oft nicht klar, womit ein System sich organisieren und abstimmen und an wen sich ein System anpassen sollte. Dies liegt an der großen Anzahl an autonomen Systemen, welche in nicht-trivialer Weise interagieren und so nur schwer vorhersehbare Gesamtergebnisse erzeugen. Diese Einflüsse kategorisieren wir als direkt, wenn sie leicht zu beobachten sind, oder indirekt, wenn sie schwer zu beobachten oder versteckt sind. Diese schwer beobachtbaren Einflüsse entstehen insbesondere bei indirekter Kopplung und Einflusspfaden, die nicht durch die Systeme wahrnehmbar sind, beispielsweise, wenn die Umgebung auf mehrere autonome Systeme, die gleichzeitig auf sie einwirken, reagiert. Um diese Einflüsse adäquat zu behandeln, ist es notwendig sie zur Laufzeit sichtbar zu machen und passende Adaptionsalgorithmen einzusetzen.

Daher ist das Thema dieser Arbeit ein algorithmischer Ansatz, der Wissen über Einflüsse und Abhängigkeiten zwischen Systemen erlangt, welches eine effizientere Nutzung durch gezielte Interaktion und das Abwenden von gegenseitigen Behinderungen ermöglicht. Die vorgestellte Methode ist abwärtskompatibel und kann in heterogenen Systemen angewendet werden. Zudem erlaubt sie die autonome Verwendung von Systemen während der Laufzeit des Systems.

Die angesprochene Methode zur Erkennung von Einflüssen basiert auf statistischen Abhängigkeitsmaßen, die verwendet werden, um auf Basis von vorhergehenden Erlebnissen zu quantifizieren wie wichtig Systeme für einander sind. Dies wird realisiert, indem die Systeme die Korrelation zwischen ihrem Belohnungssignal und der Konfiguration der anderen Systeme ermitteln. Insbesondere werden hierfür die folgenden Abhängigkeitsmaße herangezogen: Den Korrelationskoeffizient nach Pearson, die Rangkorrelationskoeffizienten nach Kendall und Spearman, die Distanzkovarianz, die Transinformation und der Maximal Information Coefficient.

Des weiteren wird dieses Grundprinzip um vier Aspekte erweitert: (i) Die Berücksichtigung von Wechselwirkungen zwischen mehreren Komponenten in einem System, indem die Einflussmessung unter den anderen Komponenten bedingt wird, (ii) die Erkennung von verzögerten Einflüssen, also Einflüssen die sich erst nach einer gewissen Zeitspanne manifestieren, indem die Messung zwischen dem Belohnungssignal und vergangenen Konfigurationen vorgenommen wird, (iii) die Messung der Einflüsse zur Laufzeit durch das Hinzufügen von zufallsbehafteten Konfigurationen und (iv) die Selbst-Adaption der Systeme an die Einflüsse, die auf sie wirken, mit Hilfe von Reinforcement Learning (zu dt. Bestärkendes Lernen) Techniken.

Die verschiedenen Aspekte werden in elementare Anwendungsfälle und zwei Echtweltanwendungen evaluiert. Letztere sind zum einen Smart Camera Networks (zu dt. intelligente Kameranetzwerke) als Stellvertreter der IoT-Anwendungsfälle und Smart Factories (zu dt. intelligente Fabriken) als Repräsentant der CPS. Die Ergebnisse zeigen, dass die vorgestellte Methode alle Einflüsse in diesen Anwendungen erkennen kann. Darüber hinaus werden Ratschläge zur praktischen Anwendung in verschiedenen Klassen von Systemen gegeben.

Schlüsselwörter:
Gegenseitige Einflüsse, Selbstorganization, Selbst-Integration, Selbst-Adaption, Maschinelles Lernen, Organic Computing, Interwoven Systems

Contents

List of Abbreviations

AC	Autonomic Computing
CAS	Collective Adaptive System
CPS	Cyber-Physical Systems
CS	Configuration Space
ICT	Information and Communication Technology
IT	Information Technology
IwS	Interwoven Systems
KAOS	Knowledge Acquisition in Automated Specification
LCS	Learning Classifier System
ld	Logarithmus Dualis
MAPE	Monitor-Analyse-Plan-Execute
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent Systems
MI	Mutual Information
MIC	Maximal Information Coefficient
ML	Machine Learning
MLOC	Multi-Level Observer/Controller Framework
MRMR	Minimum-Redundancy-Maximum-Relevance
O/C	Observer/Controller
OC	Organic Computing
OSS	Organic Security System
PTZ	Pan-Tilt-Zoom
RL	Reinforcement Learning
SC	Smart Camera
SCN	Smart Camera Network
SoS	System of Systems
SuOC	System under Observation and Control
UC	Ubiquitous Computing
XCS	Extended Classifier System

List of Figures

List of Own Publications

[1] S. Rudolph, S. Tomforde, and J. Hähner, “Mutual influence-aware runtime learning of self-adaptation behavior,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 14, no. 1, Sep. 2019.

[2] S. Rudolph, “Influence detection,” in *Organic Computing – Technical Systems for Survival in the Real World*. Springer International Publishing, 2018, pp. 385–405.

[3] K. Bucher, T. Blome, S. Rudolph, and S. von Mammen, “Vreanimate ii: Training first aid and reanimation in virtual reality,” *Journal of Computers in Education*, pp. 1–26, 2018.

[4] S. Rudolph, R. Hihn, S. Tomforde, and J. Hähner, “Towards discovering delayed mutual influences in organic computing systems,” in *ARCS 2017; 30th GI/ITG International Conference on Architecture of Computing Systems*, VDE, 2017, pp. 39–46.

[5] S. Rudolph, “Mutual influences in organic computing systems,” in *Organic Computing Doctoral Dissertation Colloquium 2016*, kassel university press GmbH, Kassel, 2017, pp. 69–80.

[6] T. Blome, A. Diefenbach, S. Rudolph, K. Bucher, and S. von Mammen, “VReanimate - non-verbal guidance and learning in virtual reality,” in *2017 9th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, Sep. 2017, pp. 23–30.

[7] M. Bieshaar, A. Calma, C. Gruhl, S. Rudolph, and A. Stein, “Machine learning in organic computing: A brief clarification of terms and concepts,” in *Organic Computing Doctoral Dissertation Colloquium 2016*, kassel university press GmbH, Kassel, 2017, pp. 113–130.

[8] O. Meisch, G. Peet, S. Rudolph, J. Haehner, and S. von Mammen, “Pick again: Self-adaptive warehouse commissioning,” in *ARCS 2017; 30th GI/ITG International Conference on Architecture of Computing Systems*, VDE, 2017, pp. 1–7.

[9] A. Stein, S. Rudolph, S. Tomforde, and J. Hähner, “Self-learning smart cameras – harnessing the generalization capability of XCS,” in *Proceedings of International Joint Conference on Computational Intelligence IJCCI 2017*, Funchal, Madeira - Portugal, Nov. 2017, pp. 129–140.

[10] S. Rudolph, S. Tomforde, and J. Hähner, “A mutual influence-based learning algorithm,” in *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016), Volume 1, Rome, Italy, February 24-26, 2016.*, 2016, pp. 181–189.

[11] S. Rudolph, S. von Mammen, J. Jungbluth, and J. Hähner, “Design and evaluation of an extended learning classifier-based starcraft micro AI,” in *Applications of Evolutionary Computation - 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Part I*, 2016, pp. 669–681.

[12] S. Rudolph, R. Hihn, S. Tomforde, and J. Hähner, “Comparison of dependency measures for the detection of mutual influences in organic computing systems,” in *Architecture of Computing Systems - ARCS 2016 - 29th International Conference, Nuremberg, Germany, April 4-7, 2016, Proceedings*, 2016, pp. 334–347.

[13] S. Rudolph, J. Kantert, U. Jänen, S. Tomforde, J. Hähner, and C. Müller-Schloer, “Measuring self-organisation processes in smart camera networks,” in *ARCS 2016; 29th International Conference on Architecture of Computing Systems*, Apr. 2016, pp. 1–6.

[14] H. Heck, S. Rudolph, C. Gruhl, A. Wacker, J. Hähner, B. Sick, and S. Tomforde, “Towards autonomous self-tests at runtime,” in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Sep. 2016, pp. 98–99.

[15] S. Rudolph and S. Tomforde, “A taxonomy for organic computing systems regarding mutual influences,” Informatik, Tech. Rep. 2016-03, 2016.

[16] S. Tomforde, S. Rudolph, K. L. Bellman, and R. P. Würtz, “An organic computing perspective on self-improving system interweaving at runtime,” in *2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17-22, 2016*, 2016, pp. 276–284.

[17] S. Rudolph, S. Tomforde, B. Sick, and J. Hähner, “A mutual influence detection algorithm for systems with local performance measurement,” in *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems, Cambridge, MA, USA, September 21-25, 2015*, 2015, pp. 144–149.

[18] S. Rudolph, S. Tomforde, B. Sick, H. Heck, A. Wacker, and J. Hähner, “An online influence detection algorithm for organic computing systems,” in *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems. Proceedings*, Mar. 2015, pp. 1–8.

[19] A. Al-Anbuky, S. Rudolph, J. Hähner, and S. Tomforde, “Public space ambient intelligence systems: Benefits, approaches and challenges,” in *Architecture of Computing Systems. Proceedings, ARCS 2015-The 28th International Conference on*, VDE, 2015, pp. 1–6.

[20] S. Rudolph, S. Edenhofer, S. Tomforde, and J. Hähner, “Reinforcement learning for coverage optimization through PTZ camera alignment in highly dynamic environments,” in *Proceedings of the International Conference on Distributed Smart Cameras, ICDSC ’14, Venezia Mestre, Italy, November 4-7, 2014*, 2014, 19:1–19:6.

[21] S. Rudolph, “A distributed controller for organic computing applications,” in *Proceedings of the First Organic Computing Doctoral Dissertation Colloquium (OC-DDC’13)*, Universität Augsburg, 2013, pp. 8–11.

[22] J. Hähner, S. Rudolph, S. Tomforde, D. Fisch, B. Sick, N. Kopal, and A. Wacker, “A concept for securing cyber-physical systems with organic computing techniques,” in *ARCS 2013 - 26th International Conference on Architecture of Computing Systems 2013, Workshop Proceedings, Feburary 19-22, 2013, Prague, Czech Republic.*, 2013.

[23] F. Bagnoli, D. Borkmann, A. Guazzini, E. Massaro, and S. Rudolph, “Modeling epidemic risk perception in networks with community structure,” in *Bio-Inspired Models of Network, Information, and Computing Systems - 7th International ICST Conference, BIONETICS 2012, Lugano, Switzerland, December 10-11, 2012, Revised Selected Papers*, 2012, pp. 283–295.

[24] F. Bagnoli, D. Borkmann, A. Guazzini, E. Massaro, and S. Rudolph, “Modeling risk perception in networks with community structure,” *CoRR*, vol. abs/1212.0657, 2012.

[25] D. Borkmann, A. Guazzini, E. Massaro, and S. Rudolph, “A cognitive-inspired model for self-organizing networks,” in *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*, 2012, pp. 229–234.

[26] B. Hurling, S. Tomforde, S. Rudolph, J. Hähner, and C. Müller-Schloer, “Evolving network protocols,” in *Proceedings of the 8th International Conference on Autonomic Computing, ICAC 2011, Karlsruhe, Germany, June 14-18, 2011*, 2011, pp. 173–174.

1 Introduction

Since the 1990s, we see an incremental miniaturization of computers, and, consequently, a computerization of everyday objects which is often called *ubiquitous computing* (UC) [27]. Even though this trend is still underway, it is unquestionable that more and more computers surround us when we look back at the last 25 years. Considering the personal and office area, we see several examples such as smart phones and tablets [28], smart watches and fitness trackers [29], personal assistants, or smart lights [30]. Extrapolating the developments, we will see them pervade more and more of our day-to-day lives.

In other areas, we can see a similar development, e.g., in the production sector with smart factories [31] or the transport sector with self-driving cars [32]. These areas show potential applications for *cyber-physical systems* [33], i.e., systems that interact with the physical world but are controlled by an embedded computation unit and are connected via a communication network. A further area of interest for the recent developments in the computer domain is the provision of computational resources where we witness the rise of *cloud computing* [34]. Here, we see massive amounts of individual servers distributed over the entire planet that have to work seamlessly to provide computational resources via the Internet. These systems have to be managed and scaled in order to adapt to failures and current use which leads to complex structures within the system.

All these developments increase the demand for algorithms that are capable to manage and coordinate the large number of devices. Therefore, *Organic Computing* [35, 36] aims to establish the concepts of *self-adaption* and *self-organization* in technical systems for the control of systems with high complexity. Several other research initiatives have similar goals or techniques, e.g., *Autonomic Computing* [37], *Proactive Computing* [38], *Multi-Agent Systems* [39], or *Collective Adaptive Systems* [40]. However, due to the complexity and size of modern systems, it is often

not clear to whom the system should adapt to or organize with. Such complexity appears as a result of large number of autonomous subsystems which can bear a high degree of complexity in itself but even more because of the non-trivial interactions among these subsystems. This results in mutual influences, i.e., influences that have their origin in the actions and configuration of other autonomous subsystems and change the quality of the outcome of another subsystem. Please note that the *mutual* is added to clarify that these influences stem from other autonomous subsystems and not the environment or possibly other sources. However, throughout this thesis, we omit the *mutual* as long as the meaning is self-explanatory. Such mutual influences can be either direct or indirect. Direct influences are often based on interactions or negotiations with other subsystems. Due to their explicit nature they are rather easy to observe. Indirect influences arise especially because of indirect couplings or influential paths that are not perceptible by the individual systems, e.g., because the environment reacts to several autonomous entities operating on it at the same time. Since the causal path from the origin of the influence and the influenced subsystems is often not observable and cannot be easily understood without hand-crafting a model these influences are difficult to trace. To address these influences, it is necessary to uncover them during runtime and enable the systems to adapt to these influences. Therefore, we formulate the topic of this thesis as follows:

> The basic theme of this thesis is a computational approach to acquire knowledge about mutual influences among systems and system parts that allows them to be controlled more efficiently by enabling interaction and averting to interfere with each other.

1.1 Motivation

In the following, we introduce two concrete examples that outline the idea of influences between technical systems. We will see that it is necessary to address these issues by developing techniques that allow to identify such dependencies during design or runtime.

Cloud computing (CC) is a common term for the recent development in the provision of computer related resources, such as computing time, storage space, or bandwidth [41]. An important concept for CC is to hide the complexity of the system from the user, i.e., she accesses an application programming interface (API)

and does not have to manage the internals. For instance, the system scales automatically and allocates the data to optimal data centers regarding the latency. An important part of CC is the scalability which requires a constant provisioning and releasing of instances within the cloud. Every relevant player from the information and communications technology (ICT) industry provides such a service today. We use a recent incident regarding the Amazon Simple Storage Service[1] (S3) as a representative in this field. To ensure an availability of the services close to 100% of the time it is necessary to introduce automated mechanisms that react to failures. This introduces dependencies and influences between the server systems resulting in highly complex structures. A recent example for such complex dependencies and possible devastating consequences, is the break-down of the S3 on February 28th 2017[2]. The incident started with an operator entering a command to shut down a certain number of servers, but, unfortunately, a typing error in the command caused the shutdown of a significantly higher number of instances than intended. Even though the system was designed to adjust the removal of a large number of instances the index system failed and had to be restarted. As a consequence, the placement subsystem failed which led to a major drop out for many users of the system. The recovery of these two systems took about 1.5 hours. Parts of the remaining systems took longer to recover due to the resulting backlog. The key insight of this example is not that wrong commands may be entered or that the program removed too many instances in a too short amount of time, but that there was a chain reaction that has not been foreseen by the designers of the system due to influences and interdependencies resulting in a high complexity.

Vehicular traffic control is constantly under discourse in personal anecdotes and politics. Common goals of traffic engineers are to reduce travel time and carbon emission [42]. Besides the efforts towards the development of self-driving cars [32], we witness the rise of technologies changing the possibilities to optimize traffic. The omnipresence of sensors, e.g., embedded in the street surface or smartphone, allows to monitor traffic in real-time. Closing the loop, the flow can be controlled by adapting traffic light phase cycle or by sending warnings to route guidance systems [43]. A common technique to allow faster travel and less carbon production are progressive signal systems. This means that traffic lights on a highly frequented road will be synchronized in a way that it is not necessary for the cars to stop at the

[1] See `https://aws.amazon.com/de/s3/` for details on the service (accessed on 2017-05-12)

[2] See `http://aws.amazon.com/en/message/41926/` for Amazon's official report (accessed on 2017-05-02).

next traffic light since it shows green when the cars arrive. This is done by adjusting the traffic light phase cycle with respect to the neighboring intersections [44]. However, today, this process is time intensive and often neglected or only applied to few major roads in a city. If it has been done, it is mostly left untouched for several years. Therefore, we want to point out the usefulness of an automated dependency detection to identify important neighbors to adjust to. This could be done at design-time supporting the traffic engineer or during runtime allowing the reaction to changing dependencies for example in the case of failure.

Following these concrete examples, we introduce two exemplary trends in the research and business area that will lead to a disruptive increase in the number of devices and autonomous systems and therefore in the number of possibly influencing systems. Without computational support it will be infeasible to analyze systems manually in terms of their interdependencies.

Industry 4.0 origins from a German government initiative[3], but it is more widely used, today [45]. Similar initiatives have been introduced by other countries, e.g., the USA[4]. The term aims at indicating that we are on the edge of a 4th industrial revolution, but there is no strict definition that is commonly agreed on. Therefore, manifold attempts for interconnection and the further automation of industrial facilities are targeted under this term. They reach from industrial devices that can be analyzed and operated via a tablet computer[5] up to fully autonomous machines. Picking up on the idea of complete interconnectedness of autonomous, reconfigurable production machines that is on the rise, novel techniques are needed to allow for an optimal behavior of these devices. More specific, we need methods that identify necessary partners and enable a co-adaption of the machines. This is especially challenging due to the large amount of devices and the flexible usage, i.e., the high number of situations and goals that often do not allow a complete analysis at design time [46]. To address these issues a method for the automated detection of influences and interdependencies in such systems is necessary.

Internet of things has been established as term in the early 2000s to capture the research and business trends towards a fully connected environment with a large amount of devices [47]. It mainly aims at the development of interconnected

[3] See `http://www.plattform-i40.de/` (accessed on 2017-09-20) and `https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html` (accessed on 2016-05-18)

[4] See `http://www.iiconsortium.org` (accessed on 2017-05-02)

[5] See for example `https://goo.gl/UG0TK7` (German, accessed on 2017-05-03)

embedded devices in terms of every-day objects that are equipped with computational devices that enable complementary functionality and therefore is similar to the vision of UC. The outlined application areas are manifold, including environmental monitoring [48], smart infrastructure [49], energy management [50], or healthcare [51]. The probably most advanced example is home automation: today, smart light bulbs, robotic vacuums and ventilation systems are already available. Due to sheer amount of devices that are present in such highly technical environments, interdependencies and influences between the systems appear that have not been foreseen by the designer. Therefore, a mechanism that can detect these influences is necessary to support the designer or automate the optimized utilization at runtime.

Concluding the previous examples, we have seen that an increasingly complex structure between technical systems leads to interweavement where designers cannot see through on all occasions [52]. This results in *hidden* and *indirect* influences which can lead to suboptimal or even fatal behavior. Thus, it is necessary to put the accountability for the detection of such influences from the systems designer to an algorithmic approach that can guide the designer or can be integrated in the system itself.

1.2 Problem Statement

Today, we face a large number of technical systems which can contain up to thousands of devices. Often, such systems employ legacy standards and algorithms that are not designed to manage the high complexity that appears through the direct and indirect interactions. Such systems can be characterized as *Interwoven Systems* [16].

If coordination is introduced between the systems, we face hand-crafted solutions that are highly specialized, e.g., the coordination of traffic lights [53] or smart cameras [54]. Such solutions can often only be created through a time-intensive trial-and-error procedure, especially if not all influencing factors are known. The influences are typically difficult to identify since they can be hidden in complex dependency structures or arise through indirect coupling. We characterize such influences as *hidden*, *implicit*, or *indirect*.

Therefore, the focus shifts towards the paradigm of *self-integration* [55] which aims at creating systems that can fully automatically integrate in a landscape of systems. A key challenge to reach this goal is to find algorithms that analyze the

influence structure between systems. This leads to the possibility to let systems self-organize well by selecting partners for cooperation.

Such a methodology should fulfill several requirements that are outlined in the following:

- **Runtime capability**: Due to the complexity that is present in modern systems, it is a more and more challenging task to foresee all situations a system will face during its lifetime. It becomes virtually impossible because of the appearance and vanishing of other systems that affect the outcome of certain processes. Therefore, an influence detection algorithm has to be executable during the runtime of a system to allow an adaption to not foreseen interaction partner, such as new devices from other stakeholders, manufacturers, or owners.

- **Heterogeneity**: To reach self-integration, a system faces the challenge to interact with several other systems that can be very heterogeneous in several aspects such as scale, virtual/physical components, connectedness, etc. A special focus in this thesis is on the heterogeneity induced by the ownership of system which limits the access and control of the other systems.

- **Compatibility**: Today, we see several long-term systems that are in use for over a decade. Replacing such systems would be costly and time intensive. Therefore, to integrate with such systems, it is necessary to keep the influence detection compatible to a wide-range of systems. This especially includes systems that do not have a cooperation mechanism, are non-adaptive, or use different adaption techniques.

- **Autonomy**: In recent times, we see an increase in the number of systems that leads to an exponential increase in complexity. These structures of systems bear new challenges since it is much more difficult to predict their behavior. To avoid time- and cost-intensive maintenance of these systems, we look for an influence detection that can be implemented in autonomous systems, i.e., that can be integrated in a system and afterwards decide on its own which influences have to be addressed.

Concluding these requirements, we can find that the generality of a proposed algorithm is the key factor to make it attractive for a potential use. Today, we have several specialized standards and solutions, but these are tied to special applications

and, therefore, lack the possibility of being applied without adaption by the designer. Following this requirements, it is necessary to find a methodology that can be applied in manifold cases.

1.3 Contribution

In the following, the main contributions of this thesis are briefly summarized:

- An architectural perspective of considering mutual influence detection based on a reinforcement learning based system model.
- A taxonomy for systems regarding their characteristics relevant to mutual influences.
- A methodology and a general workflow for the detection of mutual influences among self-adaptive and autonomous systems.
- An extension to the method that allows to detect multi-component influences, which allows to identify if configuration components influence a system in common.
- An enhancement of the methodology that covers delayed mutual influences, i.e., influences that do not become apparent immediately but after a period of time.
- A variant of the influence detection that is runtime capable and consequently distinguishes correlations in utility and performance with other's configuration from noisy or coincidental effects.
- An incorporation of this information within a reinforcement learning algorithm that is applied to the problem of self-adapting the system.
- The application of the methodology to various scenarios from the smart camera network and smart factory domain.

1.4 Overview of the Thesis

In this thesis, we see how an influence detection algorithm can be realized. The motivation and contribution are set out in *Chapter 1*. We see how it is embedded in the system landscape and architectural approaches in *Chapter 2* and how the

approach relates to other works in *Chapter 3*. It follows the core of the thesis, which is the actual methodology for detection of dependencies in technical systems in *Chapter 4* and extension to the approach in *Chapter 5*, 6, 7, and 8. Practical advice are given in *Chapter 9*. Finally, the thesis is concluded in *Chapter 10*. In the following, a more detailed overview of the chapters is given. Please note that literal and paraphrased quotations of own previous published works have not been marked in this thesis to allow for a better readability. Instead, the previous publications that are relevant for each chapter are given in the following summaries.

The contents of the individual chapters are:

1. *Chapter 1* introduces in the general topic and motivates the work. Several current developments such as Industry 4.0, cloud computing, vehicular traffic control and the Internet of things illustrate the potential application fields of this thesis. Furthermore, a problem statement is formulated that sketches the requirements of a widely applicable algorithm and an overview of the thesis is given. Parts of this chapter are based on previously published works [1, 2, 16].

2. *Chapter 2* defines the target systems which mainly relies on a configuration space and a local reward and how the approach is embodied in a multi-layer observer/controller architectural framework. Additionally, the relation to the domain of Multi-Agent Reinforcement Learning is pointed out and the example applications examined in this thesis are introduced. Finally, a taxonomy for systems regarding the characteristics important for the influence detection is introduced. Parts of this chapter are based on previously published works [1, 2, 10, 15, 16, 20–22]

3. *Chapter 3* presents related approaches. Parts of this chapter are based on previously published works [1, 2, 4, 5, 10, 12, 17, 18, 21, 22].

4. *Chapter 4* depicts the general methodology to detect influences, i.e., which information is gathered and how they are processed and how the dependency measures are utilized. Furthermore, candidates for the measurement are briefly discussed. The chapter includes examples and evaluations on an exemplary use case and the smart camera network application. Parts of this chapter are based on previously published works [1, 2, 5, 10, 12, 15–18, 20, 21].

5. *Chapter 5* extends the previous introduced methodology to capture influences that are only visible if the other configuration components are respected. It introduces the methods with examples followed by an evaluation on an exemplary use case and the smart camera network application. Parts of this chapter are based on previously published works [1, 2, 5, 10, 12, 15–18, 20, 21].

6. *Chapter 6* shows how delayed influences that affect subsystems after a period of time can be included in the analysis. It is evaluated on a smart factory scenario. Parts of this chapter are based on previously published works [1, 2, 4, 5, 10, 12, 15–18, 20, 21].

7. *Chapter 7* depicts how the measurement can be realized at runtime and what the implications of a runtime detection are. It is evaluated on a smart camera scenario. Parts of this chapter are based on previously published works [1, 2, 4, 5, 10, 12, 15–18, 20, 21].

8. *Chapter 8* introduces an exemplary method to integrate the influence information within a reinforcement learning algorithm. The approach is evaluated on scenarios from the smart camera domain. Parts of this chapter are based on previously published works [1, 2, 4, 5, 10, 12, 16–18, 20, 21].

9. *Chapter 9* gives practical advice on how to apply the influence detection under various circumstances. Parts of this chapter are based on a previously published work [1].

10. *Chapter 10* concludes the thesis and gives an outlook on possible future research directions. Parts of this chapter are based on a previously published work [1].

2 | System Model

In this chapter, the system model is introduced. Here, we outline the requirements a system has to fulfill to allow the application of the influence detection (Section 2.1). Afterwards, we embed these requirements in an optional architectural framework for self-organizing systems (Section 2.2). Furthermore, we introduce three example applications (Section 2.3) for the methodology. These are (i) elementary use cases (Section 2.3.1) that help to introduce the notion of mutual influences and the methods for influence detection, (ii) smart camera networks (Section 2.3.2), a common real-world application in the OC context, and (iii) smart factories (Section 2.3.3), an emerging technology that still bears several challenges in the context of self-organization. Finally, a taxonomy for systems regarding the characteristics for the influence detection is introduced and the example applications are classified (Section 2.4).

2.1 Target Systems

After introducing the notion of mutual influences and the types of systems that will benefit the most from the methods presented in this work (cf. Chapter 1), the utilized system model is discussed which is a formalization of the requirements we are facing in order to apply the quantification of influences. We require an overall system that is a composition of subsystems $A_1, \ldots, A_n$ in a virtual or physical environment. Here, we use the term *(sub)system* following the OC terminology, and, for a better readability, omit the *sub* if it is clear from the context that not the overall system is meant. However, one might prefer *entity* or *agent*[1] which could be used as well. Each subsystem in the overall system can assume different configurations. Such a configuration typically consists of different components. We define the whole

[1] According to the definition of the term *agent* in the Multi-Agent System domain [39]

configuration space of a subsystem A_i as Cartesian product $C_i = c_{i1} \times \cdots \times c_{im}$, where c_{ij} are the components of the configuration. Such a configuration space is a typical assumption in the Organic Computing context [56] and is (implicitly) present in most applications that are relevant to the field of adaptive systems.

Consider, e.g., a router A_1 in a computer network as an illustrating example. It can take varying configurations into account, such as the processed network protocol or parameter settings. E.g., an interval $c_{11} = [0, 100]$ for the timeout parameter in seconds and the set $c_{12} = \{1, 2, \ldots, 16\}$ for the buffer size in kilobyte. The entire configuration space of system A_1 would then be $C_1 = [0, 100] \times \{1, 2, \ldots, 16\}$.

A further assumption is that the particular configurations of individual systems are non-overlapping, meaning each subsystem has its own set of configurations and cannot control the configurations of other subsystems. This does not mean that the configuration components have to be completely disjoint in structure and values of the contained variables. For instance, two subsystems might have the capabilities, which would lead to the same set of possible configurations in these attributes but for different subsystems. Such a relation is explicitly allowed within the model.

Besides the configuration space, we need to consider a further element: the local reward. In order to apply the proposed method, each subsystem has to estimate the success of its decisions at runtime – as a response to actions taken before. This is realized based on a feedback mechanism – with feedback possibly stemming from the environment of the subsystem (i.e. direct feedback) or from manual reward assignments (i.e. indirect feedback). This resembles the classic reinforcement model [57], where the existence of such a reward is one of the basic assumptions, as well as, the idea of a *utility function* present in the OC domain [35]. If there is no obvious way to create such a signal it can be useful to apply more structured approaches from the field of goal-oriented requirements engineering [58], such as the *knowledge acquisition in automated specification* (KAOS) goal model [59] that has been applied to similar tasks [60]. Looking at the router example, a useful local reward can have different forms depending on the application scenario. For instance, a useful measure of success could be the throughput.

2.1.1 Relations to Reinforcement Learning

As mentioned before, the model of the target systems is inspired by a reinforcement learning [57] (RL) perspective. Therefore, we introduce the basic RL model and the extension to multi-agent reinforcement learning [61] (MARL) and show how it relates

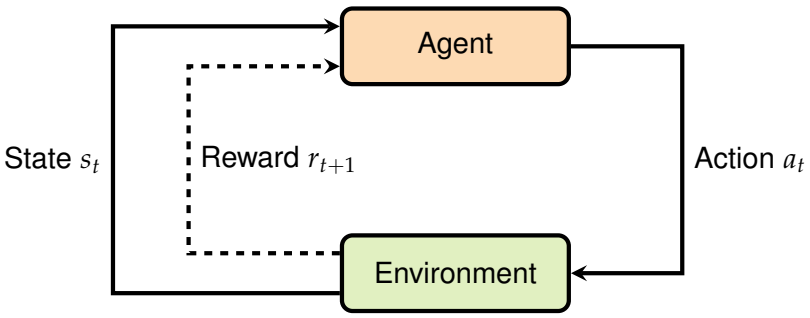


Figure 2.1: The basic reinforcement learning model.

to the model used here.

In Figure 2.1, we see the basic model for RL. It consists of an agent that interacts with its environment. It has the ability to sense the current state of the environment and manipulate it by applying an action to it. Furthermore, a reward is provided to the agent which reflects the usefulness of the applied action, which is possibly related to the current or resulting state and can have stochastic components. The goal of the agent is to maximize the expectation of this reward over the runtime of the system by choosing the best actions available in each state. To find such a strategy several approaches have been presented [62]. The model is usually formalized as a Markov Decision Process (MDP), which is a tuple (S, A, T, r) with:

- a set of states S,
- a set of actions A,
- a transition function $T : S \times A \times S \to [0,1], T(s', a, s) \mapsto p(s'|s, a)$, i.e., a function that gives the probability that action a in the state s results in the state s'.
- a reward function $r : S \times A \times S \to \mathbb{R}$, which gives a reward for each state transition.

Its basic form assumes discrete time steps $t \in \mathbb{N}$. In each time step t, the agent senses the environment's state $s_t \in S$. It then selects and applies an action $a_t \in A$. In the next step, $t+1$, the agent receives a reward r_{t+1} which reflects the quality of its action in the given state. It then senses the new state s_{t+1} and starts over again. Looking at the transition function T, we see that the probability of the appearance

of a specific state can depend on the previous state and the selected action. The agent's goal is to find a *policy* $\pi(s_t)$ that maximizes the expected discounted reward

$$\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}), \tag{2.1}$$

where $0 < \gamma < 1$ is a discount factor for the future rewards.

A common reinforcement learning algorithm is Q-learning. It is well known and studied intensively and has originally been proposed by Watkins *et al.* [63]. Like all RL techniques, Q-learning tries to solve the general RL Problem, i.e., to find an optimal policy for a given problem with respect to the long-term reward. The main idea is to find a Quality-function $Q : S \times A \to \mathbb{R}$ that approximates the reward for each state-action pair and takes into account the future rewards. To reach this goal, the value for each state-action-pair is initialized according to some of the various proposed methods, e.g., they are all set to a fixed value or they are set to a random value, and afterwards updated according to the rule

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right), \tag{2.2}$$

where $Q_t(s, a)$ denotes the old *Q*-Value and $Q_{t+1}(s, a)$ the new one, each for a given state-action pair (s, a). Furthermore, r_{t+1} denotes the reward received in time step $t + 1$ and therefore is the immediate reward for the action a_t taken in time step t. The discount factor $\gamma \in [0, 1)$ determines the fraction of estimated future rewards that is taken into account in the present step. The learning rate $\alpha \in (0, 1]$ determines how much the current experience, i.e. the current reward, is taken into account for approximating the *Q*-value.

An extension to the single-agent RL is MARL [61], where several agents interact with a single environment. Often each of the agents can only observe and manipulate specific parts of the environment. In general, the agents will therefore influence the reward of each other. The formal approach is a generalization of the MDP and is called a stochastic game (SG). It is defined by a tuple $(S, A_1, \ldots, A_n, T, r_1, \ldots, r_n)$ where n is the number of agents and the components are defined analogous to the single-agent case. Such SGs are classified depending on the reward structure. They are called fully cooperative if $r_1 \equiv \ldots \equiv r_n$ and fully competitive if $r_1(s', a_1, \ldots, a_n, s) + \ldots + r_n(s', a_1, \ldots, a_n, s) = 0$, where a_i is the action of the i-th agent. Games between these border cases are called mixed games.

Regarding the previously introduced model of the target systems, we see several similarities to the RL and MARL domain if we observe the previously described subsystems and the agents. The beforehand discussed configuration is similar to the action set since both are encoding the *choice* of the agent/subsystem, and, in fact, it is possible that both are equal. However, while actions are often encoded relative, the configuration space has to be encoded absolute. E.g., a robot could have an action *turn left* which would result in a different orientation depending on its previous orientation. For the configuration, an absolute encoding is necessary, e.g., the orientation as cardinal direction. This can result in an implicit integration of the state in the the configuration, e.g., when the current orientation is part of the state. Furthermore, the local reward is a concept adopted from the reward in the RL domain. But the difference is the focus on the locality. E.g., in a case where the local reward of each agent is equal (as in a fully cooperative game), the influence detection can still be applied but might be of limited use since it is only possible to measure which agent has influence on the overall result. These results can still be interesting but, in this thesis, the focus is on studying systems where the notion of a local reward is matched.

2.2 Architectural Framework

In the Organic Computing domain, the *observer/controller* architecture [64] is widely used since it gives a framework for the realization of self-X characteristics such as self-adaption, self-optimization, or self-learning. The basic scheme is given in Figure 2.2. There, we see three components: the *system under observation and control* (SuOC), the *observer*, and the *controller*. The SuOC is an *OC-ready* system [56], i.e., the system provides interfaces for the observation and configuration by an observer/controller pair. The observation is realized through sensors that can (at least partial) track the situation the system faces. Common tasks of an observer range from simple to complex, e.g., from preprocessing or averaging data to novelty detection [65]. The controller is the deciding entity when it comes to the configuration of the SuOC. It picks up on the information from the observer to optimize the system behavior and selects an appropriate observation model for the observer. Often, it employs a rule-based learning system [43]. Furthermore, a user can induce goals in the system or possibly change them during runtime. This basic observer/controller structure is the simplest form, but if necessary it is not limited to

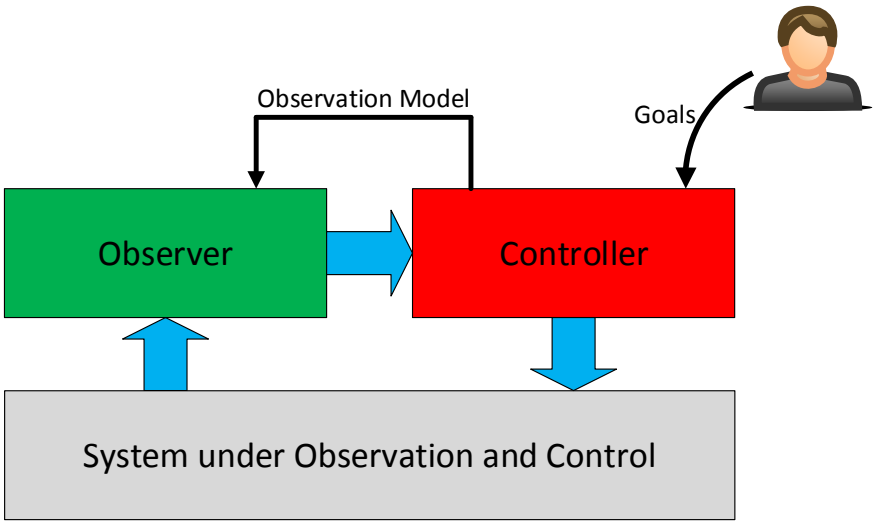


Figure 2.2: The basic observer/controller architecture.

one observer/controller pair. There are several possibilities to combine them, e.g., by using individual controller for different subsystems, stacking observer/controller pairs over each other, or use more complex hierarchical structures [66].

In the context of this thesis, we rely on a 3-layered structure[2] as depicted in Figure 2.3. Such an observer/controller stack together with the SuOC forms a subsystem in the sense of Section 2.1. In the figure, at the bottom, we see a SuOC that can be accessed via sensors and actuators. These actuators form an *action space* which is quite similar to the previously introduced configuration space (cf. Section 2.1) and, in fact, can be equal to the configuration space. However, there is a slight difference since for the configuration, we always assume an absolute encoding that is not relative to the current situation. E.g., for the router example, an action could be *increase buffer size* which is relative to the current buffer size, but for the configuration this should be the value that results from this increase which is an absolute value.
The sensors/actuators are connected to three stacked observer/controller pairs. Each of the layers can work autonomously without the interference of the higher layers, e.g., the reaction layer can still control the SuOC, if their is a failure of the

[2]This specific architecture includes an interface for users to change the goals of the system and security mechanisms [22] that are left out here since they are not in the focus of this thesis.

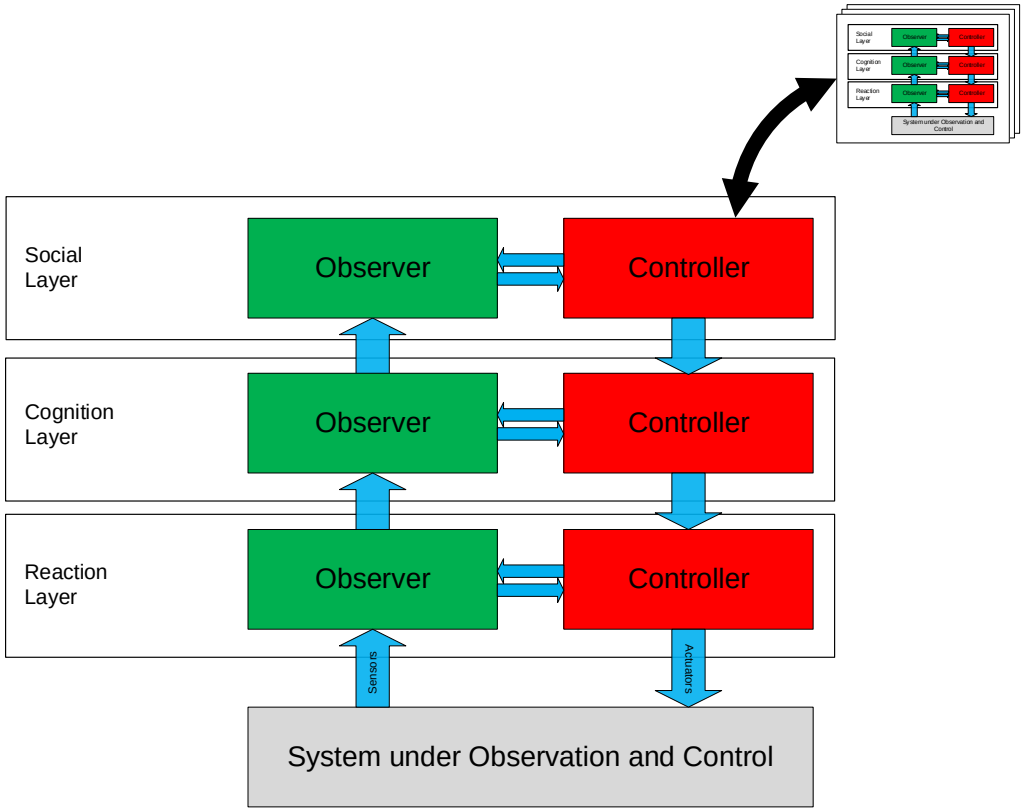


Figure 2.3: The architectural framework for this thesis. Such a multi-layered observer/controller structure is assumed for each participant in the system.

cognition layer. In the following, we introduce the purpose of each of the layers.

⇒ *Reaction layer:* This layer works on a reactive basis and ensures a fast answer to the systems needs. The observer monitors the SuOC through sensors and the controller configures it. The observer processes the raw sensor data from the SuOC, e.g., by extracting features. The controller is usually equipped with rules that allow to choose an appropriate action for the situation described by the observer.

⇒ *Cognition layer:* The layer is responsible for the awareness capabilities of the system. Especially, it can recognize situations that have not been foreseen at the design time and consider appropriate reactions. These reactions can then be included in the reaction layer. The observer mainly detects *anomalies*, e.g., *novelty* (new circumstances appear in the environment). Since these

anomalies are not covered by the rule set in the reaction layer, the main purpose of the controller is to generate new rules that cover these situations. These rules are injected into the controller on the reaction layer.

⇒ *Social layer:* Again, this layer enables awareness capabilities of the system. However, in contrast to the cognition layer, it does not focus on the SuOC and the environment but on other systems. Furthermore, it is the part of the system that is responsible for the communication (e.g., a general information exchange or teaching mechanisms) and collaboration (e.g., negotiated behavior) with other systems.

The influence detection methods introduced in Chapter 4 are therefore part of the controller of the social layer. The adaption to influencing systems as introduced in Chapter 8 can be triggered by this controller and carried out in the cognition layer.

Even though this thesis is grounded on the observer/controller architecture, the basic method is not limited to this type of system. For example, there are several architectural approaches for autonomous systems that can (at least in principle) benefit from an influence detection, e.g., the MAPE cycle [67] in Autonomic Computing [37]. For a list of approaches and a brief introduction to them see Tomforde [68].

2.3 Example Applications

In the following, we introduce several example applications. On the one hand, they serve as a basis for further clarifications of the mutual influence notion and the system model. On the other hand, they are used for the evaluation in the remainder of the thesis. First, we start with two elementary use cases (Section 2.3.1) that allow for a simple description and introduction in the appearing challenges. Afterwards, we look at the before mentioned emerging technology domains CPS and IoT (cf. Chapter 1). We selected two real-world applications from these fields. First, smart camera networks (Section 2.3.2) which can be located in the IoT sector as well as supporting devices in the CPS domain. The second are smart factories (Section 2.3.3) which are a core application for CPS.

2.3.1 Elementary Use Cases

In the following, we introduce two elementary use cases in which mutual influences appear: the collaborative box manipulation and the two-man saw application. The collaborative box manipulation is probably one of the simplest problems for an influence detection algorithm. In the two-man saw example, we encounter the issue that the own configuration of a system has to be taken into account to find an appropriate detection algorithm.

Collaborative Box Manipulation

The collaborative box manipulation is rather simple. It includes two robots and a heavy box. The robots can be configured to push or pull the box, but none of them is able to move the box alone since it is too heavy. Therefore, the robots have to cooperate, i.e., both push or pull the box, i.e., the configuration space is $\{PUSH, PULL\}$. The robots receive a local reward that is 1 if the box moves forward or 0 otherwise. For a human, it is easy to derive from this setting that the configuration of one robot has an influence on the other.

Two-Man Saw

The two-man saw example is similar to the box application. Again, it includes two robots, but, in comparison to the box example, a two-man saw is operated instead of pushing a box. The robots can be configured to push or pull the two-man saw (which results in the same configuration space as before), but they cannot handle the saw alone. Therefore, the robots have to cooperate, i.e., one has to push and the other one has to pull the saw (and vice versa). The robots receive a local reward that is 1 if the saw moves or 0 otherwise. Again, a human can easily recognize that the robots influence each other. The main difference between the two-man saw and the collaborative box manipulation from the viewpoint of the influence detection is that it is necessary to consider the configurations of both robots to find the influence. A detailed explanation of this effect can be found in Chapter 5.

2.3.2 Smart Camera Network

Smart cameras possess a build-in computation unit that can be utilized for several tasks such as image processing, object localization and object tracking. Also, most smart cameras have pan, tilt and zoom capabilities and the computation unit is used to determine beneficial alignments for the camera. Beyond, the smart cameras are

Figure 2.4: An exemplary smart camera.

equipped with wired or wireless communication devices that allow for communication with neighboring cameras, and, therefore, form a smart camera network [69].
Such smart cameras can be utilized to achieve different goals. Exemplary, this can be the tracking of objects, the identification of new objects or the 3D reconstruction of objects. In order to apply the before presented methodology of detection of mutual influences to smart cameras a local reward is obligatory. Obviously, this reward function heavily depends on the chosen goal. This is why different goals have been considered (see below). Regarding the configuration, as depicted in the system model (cf. Section 2.1), we consider the three adaptable parameters of the camera's alignment (pan, tilt, zoom).
In order to cover different classes of possible mutual influences in OC systems several goals can be considered:

- the maximization of observed space,
- the coverage of fast changing environments [20],
- the detection of before unknown objects [18] and
- the 3D-reconstruction of objects [17].

To further the diversification of dependencies in the systems, we consider different scenarios that include different positions and numbers of cameras as well as obstacles, e.g., walls, and objects, e.g., different movement patterns. Based on the many different application scenarios created in the smart camera network domain, we tested the general method for the detection of influences in technical systems. Furthermore, we show that a learning algorithm that considers the dependencies in the system can lead to a global reward close to the optimum.

Concluding the former discussion and mapping this application on the model of the target system (cf. Section 2.1), it is a realistic assumption that the cameras' configuration space is composed of their pan, tilt and zoom configuration. Allowing a pan angle between 0 and 360 degree, a tilt angle between 120 and 180 degree and a zoom[3] between 12 and 18 this would be $[0, 360) \times [120, 180] \times [12, 18]$ for each camera. As stated before, the configuration spaces of the cameras are identical in structure, but each camera can assume an individual configuration. For the local reward we can choose one of the before presented.

2.3.3 Smart Factory

In this case study, a scenario that is based on a real-world application in the domain of cyber-physical systems is considered. A currently trending subdomain of cyber-physical systems is *industry 4.0*. The term has been initially introduced in 2011 at Hanover Fair[4] and has then been picked up by the *Bundesministerium für Bildung und Forschung*, which created a research initiative[5]. There is no clear definition of the term, but it is a conglomerate of techniques and concepts that allow for a further automation of industrial tasks, such as *individualization of production*, *end-to-end digital integration*, and *horizontal integration in collaborative networks* [45]. An important part of the industry 4.0 domain are smart factories, i.e., production places that rely on a highly automatized manufacturing process.
The specific application for the case study has been chosen since it gives us new interesting characteristics that are common for cyber-physical systems. First, this are heterogeneous nodes in the system that allow us to analyze our developed techniques in a more complex scenario, e.g., with different types of machines. Second,

[3] Please notice that the actual parameter adapted here is the focal length of the camera and is typically measured in millimeter. However, to give a more intuitive understanding what a change in the configuration causes, we stick to *zoom* with origin in the term PTZ camera.

[4] `http://www.hannovermesse.de/home` (accessed on 2017-08-15)

[5] `https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html` (accessed on 2016-05-18)

there is the opportunity to examine scenarios where we face heterogeneous communication capabilities, e.g., communication within one factory and communication over the internet. Third, we see delayed influences in this application that can be examined.

The smart factory can arguably be seen as an extension of the (previously depicted) smart camera network application. This is since it is expected that future smart factories will include a number of cameras that are used as additional sensors to ensure security and well-function of the machines. However, the most important conceptual difference in the context of the influence measurement is the appearance of delayed influences. In the smart camera application, we face mostly an influence that appears immediately, i.e., only the current configuration has an impact on the outcome. Here, we face that the configuration has a delayed impact, i.e., the choice of the drill can be turn out to be of importance some time after the actual application of it.

The basic application is inspired by Beckhoff's[6] smart factory demonstrator. Even though this basic demonstrator only consists of two autonomous parts, i.e., a transport system and a workstation, it is already intended to use several of such autonomous systems [70]. Therefore, we accommodate this expected development and create scenarios with multiple entities that are based on Beckhoff's demonstrator but exceed the possibilities of it in some facets in order to reflect the expected developments in the area of smart factories. The focus of research is on two main components, that are the *flexible transport system* and the workstations. The workstations are able to use different tools, such as drills, saws, or die cutters. They have the possibility to change their configuration, i.e., which of the different tools are used during runtime and form the points in the smart factory where the work pieces are actually processed. The flexible transport system can be used in different positions, i.e., vertical or horizontal, and can be equipped with different moving components on the edge. In this thesis, we focus on transport systems that have several *mover*s, i.e., trolleys, that hold work pieces and can move at individual speeds on a designated track. The track connects two workstations and the pick and place robot, i.e., a robotic arm that is able to pick up work pieces and can put them in other places, e.g., another workstation.

[6] Beckhoff Automation GmbH & Co. KG is a company with focus on the Industry 4.0 domain.

Starting from this basic setting, the complexity of the scenarios can be varied in several dimensions. These are the number of nodes, the node's capabilities in terms of different tools and the heterogeneity by adding more machine types, the communication structure due to the location in the same or different facilities and different capabilities of the network. This allows to analyze the usefulness of the new developed techniques under manifold circumstances.
Concluding the former and mapping this application on the model of the target system (cf. Section 2.1), a realistic, exemplary configuration space is composed of the type of drill used and the pressure that is applied to the drill. Allowing 100 types of drills a pressure between 0 and 1 this would be $\{0,\ldots,99\} \times [0,1]$ for tool at the workbench. These capabilities could of cause differ between the tools, e.g., by having a different set of drills available. As local reward most likely a measure of success is used, e.g., the outcome of the operation applied is considered.

2.4 Taxonomy

Beforehand, the target systems and example applications have been introduced. They define which systems are in the scope of this work and on which systems the influence detection was evaluated. In the following, we summarize the characteristics of OC systems, which are of particular interest in the context of the influence detection. This taxonomy is meant to guide through the application of the influence detection by identifying key characteristics of the systems that help to decide in which form the influence detection should be applied to gather optimal results. The system size and characterization of subsystems are discussed in Section 2.4.1 and different types of influences are addressed in Section 2.4.2. Furthermore, the two real-world applications are exemplary classified in Section 2.4.3.

2.4.1 System Size and Characterization of the Subsystems

Obviously, the system size in terms of the number of subsystems is an interesting characteristic for the influence detection. We can roughly identify three categories of OC systems:

- *Small systems*, i.e., systems with few subsystems.
- *Middle-size systems*, i.e., systems with less than a few hundred subsystems.
- *Large-scale systems*, i.e., systems with more than a few hundred subsystems.

Another characteristic related to the subsystems is their configuration space. As highlighted before, the system model allows for multiple configuration components (cf. Section 2.1) that can have different types. Please notice that the evaluation in this thesis focuses on small to mid-sized systems, but, advice on how to handle larger systems is given in Chapter 9. Another interesting criterion is simply the number of configuration components. Furthermore, we identified the following types of configuration components:

- *nominal*: The different values can be categorized, but there is no order for the categories. An instance are the categories left and right.
- *ordinal*: The categories can be ordered. An instance are the categories low, medium and high, or 1, 2, 3.
- *infinite real-valued*: An infinite number of values can be assumed. For instance, this could be the interval [0,1].

The types *nominal* and *ordinal* can be further characterized by the number of categories. For the *infinite real-valued* class, we always assume the set of values to be infinite.

Consider the smart camera example again: Current test systems comprise from 2 up to 100 cameras, i.e., we face a small to middle-sized system. Thus, the computational complexity is not a major factor in the choice for an optimal dependency measure. Furthermore, the configuration components of the cameras are infinite real-valued. This excludes the discrete mutual information as a viable choice.

2.4.2 Influences

The last category are the characteristics of the influence itself. The aspects introduced in the following mainly describe the “nature” of the influence and have to be considered for an optimal detection.

Origin: A first characteristic is especially interesting in the context of high communication costs: the origin of the influence. An influence can originate from an agent in the neighborhood or from a more remote one that can only be reached over multiple hops. Even though this does not change how the influence can be detected, it can have an impact on the communication cost attached to it. For instance, in an ad hoc network the communication cost increases linearly with the number of hops.

Joint influence: Another characteristic of an influence is the possibility that multiple subsystems have to act jointly in order to exert their influence. For example, two robotic arms have to hold a workpiece in place while another drills a hole in it. Analyzing a single holding arm alone is not sufficient to detect the influence on the drilling robot.

Strength: The strength of an influence can be characterized by two aspects. The first is the power of the dependency measures. Some of them are limited to linear or monotone dependencies, but the more powerful ones can measure stochastic dependencies (which corresponds to the most general class of dependencies). The second aspect is the sample set size: For some applications, a small sample set can already lead to an appropriate detection rate, but in other cases a much bigger sample set might be required.

Time behavior: An influence can have an immediate effect meaning that the results are experienced by the target subsystem right after the configuration is assumed. However, there are also cases where the subsystem is affected after a delay.

Regarding the smart cameras, we do not observe any joint influences or delays. Moreover, the origin of an influence can be assumed to be rather close since it requires a shared observation area.

2.4.3 Classification of Example Applications

Following the taxonomy description, we assess the two real world applications, smart camera networks and smart factories (cf. Section 2.3.2 and 2.3.3).

Smart Camera Networks

In the following, we classify the depicted smart camera networks regarding the taxonomy and start with the subsystem characteristics. If we look at the first characteristic, that is the number of subsystems, we do not have a clear classification since SCN can have different sizes from few cameras to few hundred cameras. Large-scale systems are also possible, but, looking at the current development, this is not expected for the near future. The number of configuration components in this domain is three, i.e., the pan, tilt, and the zoom. Each of the configuration components is infinite real-valued.

Regarding the communication, we can face different instances of SCN. As men-

tioned before, smart cameras can be connected wired or wireless. In the case of a wireless ad-hoc network, we face high communication costs at least for multi-hop communication. If we face wired connections, the situation is not as demanding as in wireless communication but still limited.

Considering the influences in such a system, we find that the possible influences are limited to the spatial neighborhood. Due to the nature of the reward function the cameras can only be influenced by other cameras that share the potential field of view. But we cannot suspect that the influences can be detected by linear or monotonic measures, therefore we categorize them as stochastic. Moreover, we cannot find instances in which the influence only becomes apparent if several neighbors act in common. Furthermore, we do not see a temporal influence in the system since previous configurations of the camera do not play a role for the other cameras. This is because the cameras only get higher rewarded if they observe the same area of interest at the same time.

Smart Factory

Regarding the subsystem characteristics, we can assume that such a smart factory consists of a rather small system with few subsystems. This reflects the current state-of-the-art in this area. However, in principle, a smart factory could also be a middle-sized system if several facilities are considered. In the conducted experiments, the configuration space of the workstation is composed of only a single configuration component that is nominal with a small number of categories, but depending on the actual hardware it could as well have infinite real-valued configuration components, e.g., the pressure used for drilling.

Considering the communication, one can assume that it is rather cheap since the workstations are stationary. Moreover, a designated area can easily be connected by wire and could also be controlled by virtual systems that are emulated on a central computer. The situation changes if the system consists of several closed up areas that might not be connectable that easy.

Since in systems with a smaller number of nodes we can assume that all subsystems are neighbored, for instance, as virtual systems in a central device, we do not have to consider multi-hop influences here. In contrast, the situation is more

complex regarding the influences that only become apparent if several subsystems act in common, e.g., two robotic arms have to hold a workpiece in place while another drills a hole in it. Since the configuration space of the different subsystems in a smart factory is rather small, it might be possible to use linear or monotonic dependency measures for this purpose. What can be clearly stated here is that the influences in the system will be temporal since the workpieces move through the system and will cause a delay in the occurrence of influences.

2.5 Summary

Summarizing the system model used in this thesis, we started by introducing a model for the systems that are targeted by the presented methods. The key aspects of this model are the configuration space and local reward of the systems. Additionally, the methods are embedded in an architectural framework namely a multi-layer observer/controller architecture with a reaction, cognition, and social layer. Afterwards, four example applications have been introduced. Two of them are elementary use cases and the other two are realistic applications from the IoT and CPS domain. Besides a general introduction in the domain, each of the examples has been set in a context of the model for the target system. Finally, a taxonomy for systems regarding the influence detection has been introduced and the two real-world applications have been exemplary classified regarding this taxonomy.

In the following, we will see how the influence detection works and how it can be applied under various circumstances.

3 | Related Work

In the following, the related work in the area of influence formalization and quantification is presented. Further related work can be found in Section 4.1.1 where several dependency measures are introduced. Additionally, we have seen how this thesis relates to the field of OC in Section 2.2 and RL in Section 2.1.1.

In literature, several approaches can be found that try to formalize the mutual influences. Most of these approaches focus on the influence through direct or indirect interactions. For instance, a model for interactions is proposed by Keil *et al.* [72], but a method to detect the implicit interactions is not provided. Another common approach is to use *stit* logic for modeling the interactions in Multi-Agent Systems [73–75]. The focus of this works is on the system specification and verification and therefore differs much from the focus of our work where a goal is adaption at runtime.

Multi-agent reinforcement learning (MARL) is an active field of research [61, 76] which is related to this thesis. An insightful overview and a useful taxonomy has been introduced by Busoniu *et al.* [61], for instance. Following their taxonomy, the approach presented here can be useful in fully competitive and mixed games (each static or dynamic). It can be used in fully cooperative games as well, especially if the global payoff is a function of the local payoffs of the agents (cf. the smart camera application). As stated by the authors the complexity resulting from coordination is a major issue in MARL systems. The influence detection can keep this complexity to a minimum since it allows to only coordinate with relevant partners.

There are some interesting approaches in the area of coordination in MARL. For example, in [77], an algorithm for the learning of organizational roles has been

introduced. This means the agents are heterogeneous and the approach allows to find the best fit of agent capabilities and tasks.

Kok *et al.* [78] presented a work that is based on so-called coordination graphs and an approach to solve the global coordination problem on a local basis if it is possible to decompose the global payoff function into a sum of local payoff functions. A restriction of this method is that it relies on inference rules that are hand-crafted. Furthermore, they focus on discrete state variables which is not the case in the approach presented in this thesis. But since they assume a given graph the influence detection could be used to infer such a graph based on the roles of the agents.

An extension to this approach has been presented in [79]. There, similar to the influence based approach, the coordination graph is inferred at runtime creating a transition from independent learners to coordinated action selection. It is based on a t-test between the maximally possible expected reward when the agents act in common and the expected reward from independent decisions. The approach is for general sum games and is limited to discrete state and action spaces and cannot be simply adopted to a continuous case. Furthermore, the work only shows the applicability for a Q-learning-based approach and there is no trivial way to make it applicable to a wide range of algorithms. Methodologically, a major difference here is that the method is focused on finding the states in which agents should collaborate. This requires that all agents are always willing and able to cooperate.

De Hauwere *et al.* [80] demonstrated a specialized solution for mazes with two robots. They used a generalized learning automata that uses the distance to the other robot to learn how to avoid a collision by identifying states in which they have to coordinate. While this approach might be adapted for other tasks it would be necessary to hand-craft the inputs of the learning algorithm each time.

Later, they presented a work with the main focus on a method to generalize the learned behavior for a single state over several states. The presented approach needs to identify the states were coordination is necessary [81]. Their method is similar to Kok's. However, they assume that the agents have already learned an optimal policy if acting alone since this is necessary to find states that need coordination which renders it unusable for learning at runtime. Furthermore, the work is focused on sparse interaction, i.e., it identifies the states for cooperation and not the systems. This means the approach is not appropriate to find useful collaborators from a set of agents. Furthermore, the approach is limited to discrete

state and action sets. Another difference to the approach presented here is that it relies on the Kolmogorov–Smirnov test used as a goodness-of-fit test, i.e., it tests if the distribution of points fits a model. However, this approach depends on a model, which is not always present, e.g., if not Q-learning is used but a policy-based RL approach, which is contrary to the requirement of independence from the control mechanism of the system.

The above approach has been extended to solve delayed coordination problems [82]. However, the limitation regarding the discrete state and action sets persists.

Vrancx *et al.* [83] presented a further extension to Coordinating Q-Learning, where they employ transfer learning similar to the approach taken in this thesis but on a state basis, i.e., only states where coordination is necessary are extended. This approach is not applicable here since they use an agent centric view to generalize over different situations because it cannot be applied to problems where it is necessary to distinguish between the other systems. Furthermore, they focus on discrete situations and actions and the methods cannot by applied to continuous state and action spaces because the tests and extensions are applied state wise. Moreover, they use the situation of the other agent which can be insufficient because it does not take into account which actions are applied by the other systems. In contrast, in this thesis, we rely on the configuration of the other systems which includes the chosen action.

Lanctot *et al.* [84] proposed a measure called joint policy correlation. This has been prototypically applied to a two-agent laser-tag scenario. It is based on the repetition of the same scenario with several seeds that lead to different strategies of the agents. Afterwards, a matrix is formed that compares the average rewards of the agents against the agents from the other repetitions. The values against the initial opponent and the other opponents are then aggregated to create a measure that allows to see how much an agent has overfitted to its initial opponent. The goal of the work is very different from the one presented here, since it wants to create a measure on how much an agent overfits to the behavior of other agents. A measure of influence cannot be directly derived from this approach. Furthermore, in this thesis, we focus on a runtime learning approach that is not possible if the experiment has to be rerun several times.

The approach presented in this thesis has similarities with feature selection

methods. These are categorized as *filter*, *wrapper* or *embedded* methods. Wrapper methods are not applicable here since they require multiple repetitions of the learning tasks which is contrary to the goal of learning at runtime. In the field of filter methods, the minimum-redundancy-maximum-relevance [85] (MRMR) is frequently used. However, it relies on the elimination of redundancies between the features which is not desired here since it reduces the information one gets from the influence detection. Furthermore, it comes at an extensive computational cost compared to the methods proposed in this thesis. If it appears that too much configurations are identified as influencing, the control algorithm can employ a technique for dimension reduction such as auto-encoders, which have been successfully used with RL [86].

Regarding the embedded methods, the most prominent instance is the regularized least-squares policy [87–89]. These methods are tied to a specific learning algorithm and therefore breach the restriction of an applicability independent from the control algorithm. There are several other works that employ feature selection for reinforcement learning tasks [90–93], but they focus on a single learner and not the interaction of autonomous entities. Additionally, the works do not focus on feature selection at runtime. Even though Bishop *et al.* [91] call their approach *online*, they do not share our understanding of the term, since they apply their algorithm to the same problem multiple times for the purpose of learning.

Regarding the statistical methods, a selection of frequently used dependency measures is outlined in Section 4.1.1. Additionally, the cross-correlation [94] could be considered, which has been for example used in the control theory for system identification tasks. However, the cross-correlation can only guarantee the detection of linear associations and additionally does not allow an online calculation. More advanced approaches in the field of non-linear system identification use techniques that rely on the estimation of a function and are therefore not applicable here [95].

Furthermore, another calculation method for the continuous mutual information could have been considered [96], but each measure has been limited to the most common approach. Moreover, parametric methods [97] could be considered, but since assumptions of the distributions of the reward function of the system are often not possible, we omit these techniques. Additionally, statistical tests [98] could be used instead of dependency measures, but preliminary results have not been promising and therefore we stick to dependency measures in this thesis.

There are related works on collaborative learning in distributed environments,

e.g., by knowledge exchange. Transferred to the problem description presented here this approach would correspond to an exchange of rules in the sense of the observer/controller architecture (cf. Section 2.2) [99, 100] which can be used additional to the methods presented in this thesis.

Furthermore, there are attempts to combine self-organizing systems with learning algorithms, e.g., Boes *et al.* [101] introduced a MAS for the control of technical systems that has self-X characteristics. The method is called Escher and aims to deconstruct complex control problems by using an acyclic network of interacting agents that are able to learn from experiences. The algorithm has been applied to toy problems and an engine calibration. The approach differs from the one presented here, since they created a MAS to control a single technical system, while here we focus on several systems that are autonomous and have to adapt to the behavior of other systems. But, the application of the influence detection to the individual controller agents of Escher can be helpful if the different inputs of the controlled system have to be coordinated to reach optimal results.

Concluding the related work, it follows a list that summarizes the limitations that have not been addressed in previous works but are possible with the method presented in this thesis:

- Automated detection of influences: Some works have mentioned or embraced or focused on mutual influences, but they do not provide a method to find them in an automated way [72–75].
- Independence from control algorithm: The approaches rely on a specific reinforcement learning algorithm and are not easily portable to other algorithms because their influence detection relies on this model, e.g., Q-Learning [78, 79, 81, 82] or least-squares policy [88, 89] .
- Continuous state and action spaces: The algorithms are based on the idea of sparse interaction and therefore use statistical methods to identify states in which collaboration is necessary [78, 79, 81–83]. These methods are not portable to settings with continuous state and action spaces.
- Runtime capability: Some algorithms, especially the feature selection approaches, assume a fixed set of candidates that can have an influence on the learning algorithm [85, 88, 89], but they cannot be used in a stage wise fashion that allows the application at runtime.

4 | Mutual Influences

In this chapter, the method for the detection of influences among systems is presented. The general workflow is depicted in Section 4.1 and the basic measuring method in Section 4.1. This includes examples (Section 4.1.2) and an evaluation (Section 4.1.3).

In the following chapters, this basic method will be extended, and we have a look at more complex tasks: the consideration of own configurations, delayed influences, and possibilities to adapt to detected influences.

4.1 Methodology for Detection

Based on the described target systems (cf. Section 2.1), we can now define the methodology for mutual influence measurement. The goal is to identify those components of the configuration of the other systems (i.e., value range and considered variables) that have influence on the system itself. After the identification of influencing configuration components, they can be addressed by a designer, e.g., by considering them in their control algorithms, or by a self-adapting system itself, e.g., using a learning algorithm.

In general, we are typically interested in the question whether a system as a whole is influencing other systems. However, to be more precise in the description of the influence, we want to detect those parameters where the optimal configuration values are somehow influenced by the current settings of the other systems. The basic idea of the following approach is to make use of stochastic dependency measures that estimate associations and relations between the configuration components of a system and the reward of a second system. The basic method assumes that the mutual influence between the systems is instantaneous, meaning that the reward of the system reacts to the configurations of the other systems in the same time step.

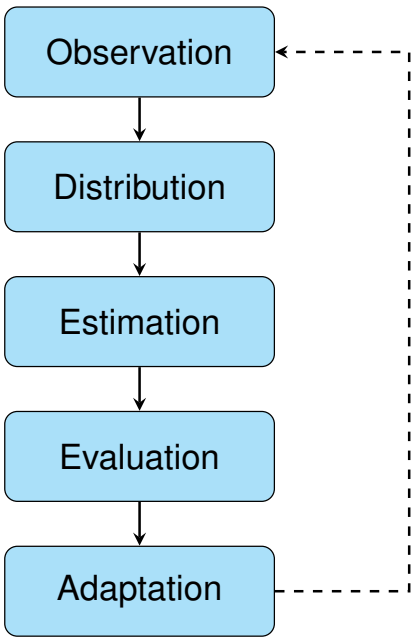


Figure 4.1: The general workflow of influence detection.

However, the approach can be extended in order to detect delayed influences as well by measuring the dependency between the configuration components and a later reward which is depicted in detail in Chapter 6. In this section, we introduce the general workflow on the variant without delayed influences.

In general, dependency measures are designed to find correlations between two random variables. In the following, we model the configurations of distributed systems as such random variables, which reflects the system model where we face autonomous systems whose actions can be uncertain due to non-deterministic behavior or incomplete information, for instance.

Before we introduce some candidates for the dependency measures (cf. Section 4.1.1), we outline the general workflow for an influence detection for each subsystem in the overall system that is depicted in Figure 4.1.

1. *Observation:* Continuously observe the configuration and estimate the goal achievement of a productive system, e.g. the pan, tilt, and zoom of a camera and the corresponding reward in terms of the given goals. These observations will typically be done by the system itself in the observer side of the multi-layered observer/controller architecture. But they could as well come from an external entity.

2. *Distribution:* Gather configurations from other systems and provide your own one to them, i.e., the observations have to be send to the physical location where the calculation of influences takes place. Regarding the system model, this is the device which is responsible for the social layer of the systems.

3. *Estimation:* Estimate the dependency value by relating the own reward to the configurations of the other systems. The basic idea of the algorithm is to make use of stochastic dependency measures that estimate associations between the reward of a system A and the configuration components of a second system B. These dependency measures are designed to identify correlations between two random variables X and Y. The reward of A is identified with a random variable X and the configuration of entity B with a random variable Y. This mapping implies that if the association between X and Y is high, we also have a high influence of B on A since it reflects that the configurations of B *matter* for the reward of A. Vice versa, if the association is low, we do not see an influence. There are several dependency measures available which might be suitable depending on the application. An overview for the most interesting measures is given in Section 4.1.1.

4. *Evaluation:* Compare the values calculated for the different (other) systems and their configuration components. Regarding this step, we have several possibilities. The first would be to compare the influence values to a fixed threshold, which is a valid option but can be difficult since an appropriate threshold might not be available for each application. Second, the influence values can be compared on a relative basis between the systems. This leads to a ranking of systems according to their influence. A third possibility is to simulate an independent system with the same configuration space and compare the “artificial” value to the real values of the other systems. This allows to decide on a basis that only includes one other system. Additionally, it can be useful to calculate the p-value, i.e. a value that allows to estimate how likely the given outcome under the assumption of independence is.

5. *Adaptation:* Address the influences in the control strategy. This can take various forms; especially when applied at design-time, the designer can decide on a case-by-case basis. For a self-adaptive solution, we propose to use a learning algorithm that includes the configuration of the influencing systems in the situation description during runtime (cf. Chapter 8).

This process can be used either during design-time with a prototype or continuously during runtime. For the design-time variant, the system runs for a certain time while the configurations and reward are logged. Afterwards, the calculation and decision process starts. During runtime, the steps have to be considered as a loop that runs continuously but one can still decide on how long samples are gathered since a distribution and recalculation may not be justified for each newly gathered sample.

4.1.1 Discussion of Dependency Measures

As previously laid out, dependency measures are utilized for the influence detection. Therefore, it follows a short overview of the most prominent measures and their advantages and drawbacks:

- *Pearson correlation:* The probably most prominent instance is the Pearson correlation coefficient – sometimes just called "correlation coefficient" [102]. The main advantages are its simple implementation and its fast calculation. In the context of influence detection as discussed in this section, the major drawback is that only linear correlations can be detected, i.e., it can fail in case of more complex dependencies. Moreover, it is necessary to calculate the distance between realisations of the random variable which might make it not well suited for some problems. It assumes values between -1 and 1 where -1 indicates a perfect negative linear correlation and 1 a perfect positive correlation. 0 means that there is no linear correlation. Because of this, if comparing the influence of different systems, it should be considered to use the absolute values. The Pearson correlation coefficient r is defined as:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}\,, \tag{4.1}$$

  where the x_i and y_i are the gathered samples, i.e. the configuration components and the reward. n is the number of samples and $\overline{x}, \overline{y}$ denote the mean values of the random variables.

- *Kendall rank correlation:* Another measure that is based on calculating the ranks of the gathered samples has been introduced by Kendall in 1938 [103]. The measure can be computed rather fast. It can detect monotone dependencies, which is better than just linear dependencies but still can be not

sufficient for many applications. It assumes values between -1 and 1; where -1 indicates a perfect monotone declining relationship and 1 a perfect monotone increasing relationship. For independent variables, a value around 0 should be considered. The Kendall rank correlation τ is calculated using:

$$\tau = \frac{(\#\ concordantPairs) - (\#\ discordantPairs)}{n(n-1)/2}\ . \tag{4.2}$$

A concordant pair are two samples (x_i, y_i), (x_j, y_j) where the ranks of the elements agree, i.e. if $x_i > x_j$ then $y_i > y_j$ or if $x_i < x_j$ then $y_i < y_j$. The opposite of a concordant pair is called discordant. Cases with $x_i = x_j$ or $y_i = y_j$ are neither discordant or concordant and are not handled in the basic variant.

- *Spearman rank correlation:* The rank correlation after Spearman is similar to Kendall's since it is based on ranks, too. Here, the ranks of the samples are calculated and instead of a comparison between the samples the Pearsons correlation coefficient is calculated on the ranks. Therefore, the values range from -1 to 1 with 0 meaning that no dependency has been detected. The Spearman correlation is calculated using:

$$\rho = \frac{\sum_{i=1}^{n}(rg(x_i) - \overline{rg_x})(rg(y_i) - \overline{rg_y})}{\sqrt{\sum_{i=1}^{n}(rg(x_i) - \overline{rg_x})^2}\sqrt{\sum_{i=1}^{n}(rg(y_i) - \overline{rg_y})^2}}\ , \tag{4.3}$$

where x_i and y_i denote the samples and n the number of samples. $rg(x)$ is short for the rank of x, i.e. the position of x if all samples are ordered by their value. $\overline{rg_x}$ is the average rank of the samples x_i and $\overline{rg_y}$ for y_i, respectively.

- *Distance covariance:* This measure is an extension to the Pearson correlation that takes the distance between the samples into account. Since these distances have to be calculated for the whole sample set, it is not suitable for an online calculation; the advantage of this method is that it is not limited to linear dependencies but can find all types of dependencies [104]. The distance covariance is calculated as follows: We first derive the Euclidean distances $a_{j,k} = \|x_j - x_k\|$ and $b_{j,k} = \|y_j - y_k\|$. Afterwards, we calculate $A_{j,k} := a_{j,k} - \overline{a_{j.}} - \overline{a_{.k}} + \overline{a_{..}}$ and $B_{j,k} := b_{j,k} - \overline{b_{j.}} - \overline{b_{.k}} + \overline{b_{..}}$ where $\overline{a_{j.}}$ is the mean of the j-th row, $\overline{a_{.k}}$ is the mean of the k-th column and $\overline{a_{..}}$ is the mean of the whole matrix:

$$\frac{1}{n^2} \sum_{j=1}^{n} \sum_{k=1}^{n} A_{j,k} B_{j,k} \tag{4.4}$$

- *Mutual information:* A quite different approach has been taken by Shannon as part of his work in the context of information theory [105]. The basic variant can be used for discrete random variables and calculated online. Furthermore, all types of dependencies can be found. A small disadvantage in this context is that the maximal possible value depends on the structure of the random variables, i.e., the values can be normalized between 0 and 1, but the comparability to other variables with different structures might be limited. The mutual information is defined as:

$$I(X;Y) := \sum_{x \in X} \sum_{y \in Y} p(x,y) \operatorname{ld} \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{4.5}$$

  where $p(x,y)$ is the joint probability of the events x and y and $p(x)$ and $p(y)$ are the marginal probabilities of x and y. The probabilities can be easily approximated using a frequency counting (i.e., a maximum likelihood approach based on discrete finite random variables). There exists also a continuous variant of the mutual information measure. Here, the calculation is much more complicated since the estimation from samples needs more advanced techniques. The most common approach is based on a k-nearest neighbor method [106]. Another approach is the manual binning of the values in order to use the discrete version, which can bear problems because the results might depend highly on the chosen binning parameters. An automatic binning can be done with the maximal information coefficient (see below).

- *Maximal information coefficient:* This measure is an extension to the mutual information for real-valued data. It automatically calculates the binning that results in the maximal mutual information for a given dataset. A drawback is that there is no method for the online calculation. The basic formula is:

$$MIC(X,Y) := \max_{n_x n_y < B} \frac{I(X;Y)}{\log(\min(n_x, n_y))} \tag{4.6}$$

  where n_x and n_y denote the number of bins for X and Y. This means that the number of bins is limited by a threshold B that is by default determined

Reward of system A (f_A)		Configuration of system A (C_A)	
		PUSH	PULL
Configuration of system B (C_B)	PUSH	1	0
	PULL	0	0

Table 4.1: The reward of system A resulting from different configurations of system A and B in the box example.

depending on the sample size. The denominator serves for the normalization of the value. However, it is computational expensive to determine the values for all possible binnings. Therefore, a heuristic is introduced. For details please see [107].

It should be mentioned that the choice of a dependency measure does not have to be exclusive. It can be useful to calculate several measures in parallel or to perform an iterative process where first computationally light-weighted measures are used, and only if necessary, the more powerful and computationally expensive methods are used.

4.1.2 Examples

In the following, we see an example calculation of the influences in the collaborative box manipulation which has been introduced in Section 2.3.1. The example aims to clarify step 3 of the workflow, the estimation of influences (cf. Figure 4.1).

To recap briefly, the setting in the collaborative box manipulation consists of two robots that aim to move a box which is too heavy to move alone. Therefore, it only moves if both of the robots use the configuration $PUSH$. In this case each of them receives a reward of 1 while otherwise they receive a reward of 0.

We chose this elementary use cases for the demonstration of the methodology since it is a very simple scenario. For demonstration purposes, we assume that a table with the reward resulting from a combination of configurations of the robots is known (cf. 4.1) which is explicitly not the case in real world applications. The estimation of the influences from samples is shown later in Section 4.1.3.

For a human, it is easy to derive from this setting that the configuration of one robot has an influence on the other. Therefore, we expect the algorithm to measure an influence greater than zero. Since it is relatively easy to calculate (if

the probabilities of the different events are known), we chose the mutual information dependency measure for this example calculation. However, other dependency measures will work as well for this simple example (cf. 4.1.3).

In Table 4.1, we see the resulting reward for system A given the different combinations of configurations of the two systems. In order to find the influence of system B on system A, we have to calculate the mutual information between the configuration of system B (C_B) and the reward of system A (f_A). The mutual information is defined as:

$$I(X;Y) := \sum_{x \in X} \sum_{y \in Y} p(x,y) \operatorname{ld} \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{4.7}$$

This means, we have to derive the marginal probability distributions of f_A and C_B as well as their joint probability distributions. For the calculation, we assume that each system chooses the PUSH or PULL configuration with a probability of 0.5 independent of each other. This means $p(C_A = \text{PUSH}) = p(C_A = \text{PULL}) = p(C_B = \text{PUSH}) = p(C_B = \text{PULL}) = 0.5$. Since we have four combinations of PUSH and PULL, where each one appears with a probability of 0.25, we can derive from the table that $p(f_A = 1) = 0.25$ and $p(f_A = 0) = 0.75$. Next, we have to calculate the joint probabilities $p(f_A = 1, C_B = \text{PUSH})$, $p(f_A = 1, C_B = \text{PULL})$, $p(f_A = 0, C_B = \text{PUSH})$ and $p(f_A = 0, C_B = \text{PULL})$. In the table, we find that f_A will never be 1, if $C_B = \text{PULL}$, hence $p(f_A = 1, C_B = \text{PULL}) = 0$ and $p(f_A = 0, C_B = \text{PULL}) = 0.5$. If $C_B = \text{PUSH}$, which happens with a probability 0.5, we have two cases. In the first case, we see $C_A = \text{PUSH}$ and $f_A = 1$, in the second, $C_A = \text{PULL}$ and $f_A = 0$. This means $p(f_A = 1, C_B = \text{PUSH}) = p(f_A = 0, C_B = \text{PUSH}) = 0.25$. Now, we can substitute the values in Equation 4.7 and get

$$\begin{aligned}
I(f_A; C_B) = {} & p(f_A = 1, C_B = \text{PULL}) \operatorname{ld} \left(\frac{p(f_A = 1, C_B = \text{PULL})}{p(f_A = 1)p(C_B = \text{PULL})} \right) \\
& + p(f_A = 1, C_B = \text{PUSH}) \operatorname{ld} \left(\frac{p(f_A = 1, C_B = \text{PUSH})}{p(f_A = 1)p(C_B = \text{PUSH})} \right) \\
& + p(f_A = 0, C_B = \text{PULL}) \operatorname{ld} \left(\frac{p(f_A = 0, C_B = \text{PULL})}{p(f_A = 0)p(C_B = \text{PULL})} \right) \\
& + p(f_A = 0, C_B = \text{PUSH}) \operatorname{ld} \left(\frac{p(f_A = 0, C_B = \text{PUSH})}{p(f_A = 0)p(C_B = \text{PUSH})} \right) \\
= {} & 0 \operatorname{ld} \left(\frac{0}{0.25\ 0.5} \right) + 0.25 \operatorname{ld} \left(\frac{0.25}{0.25\ 0.5} \right)
\end{aligned} \tag{4.8}$$

$$+ 0.5 \operatorname{ld}\left(\frac{0.5}{0.75\ 0.5}\right) + 0.25 \operatorname{ld}\left(\frac{0.25}{0.75\ 0.5}\right)$$

$$\approx 0.31$$

The result is 0.31. This value shows us that there is a correlation between the configuration of system A and the reward of system B. Since the configuration is independent and uniformly distributed this implies that system B influences system A.

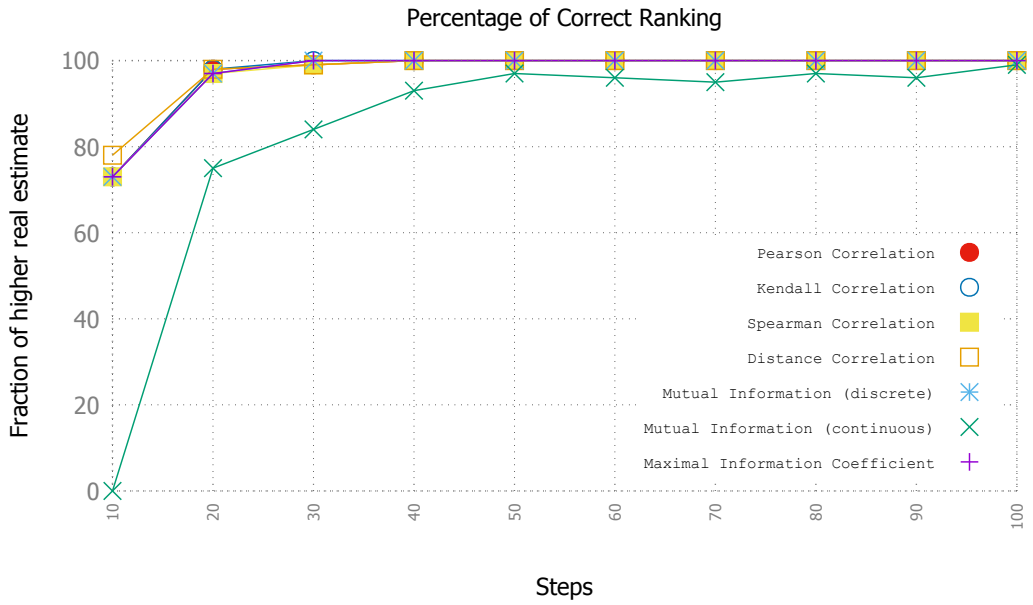
4.1.3 Evaluation

In the last sections, an introduction to the general method of influence detection has been given and we have seen an example calculation. Such a calculation is possible if the distributions of the configurations and rewards are known. In reality these distributions are unknown in most applications. Therefore, we look at the influence detection in two applications where the influence has to be estimated from samples. The first one, the collaborative box manipulation, has previously been discussed on a theoretical level. The second one, the smart camera network, shows the applicability in a real-world setting. We focus on these two examples since they can be solved with the general method introduced so far. The other examples will be evaluated in the following chapters where more appropriate methods for these examples are introduced.

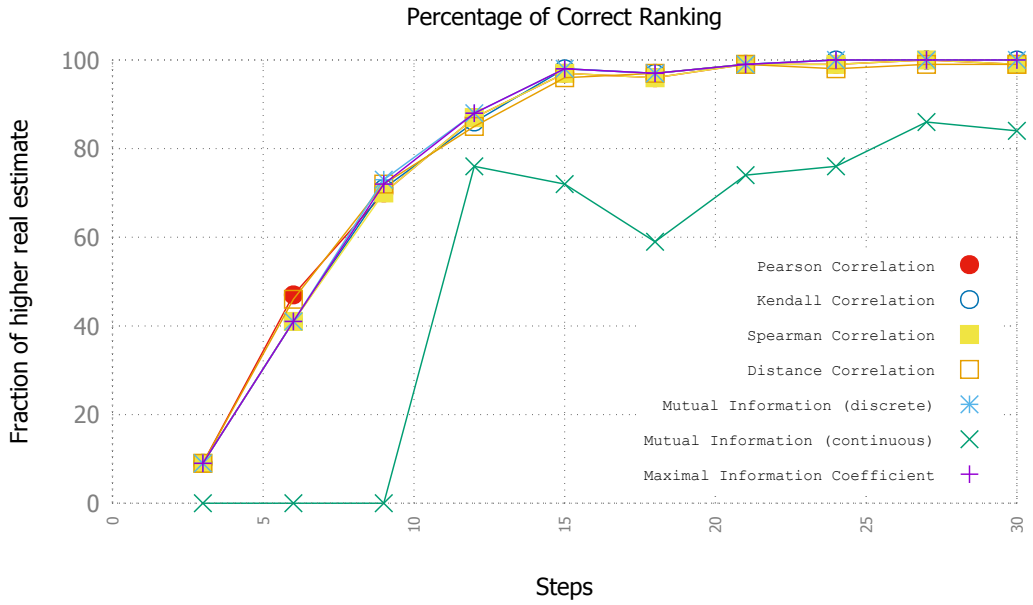
Collaborative Box Manipulation

For the evaluation, we first consider one of the the previously introduced elementary use cases (cf. Section 2.3.1), the collaborative box manipulation. Let us recall, briefly: In this application, there are two robots with the task of pushing a box. Each of them can either $PUSH$ or $PULL$ the box and due to the nature of the problem the box only moves if both of them $PUSH$. This leads to a reward of 1 while all other combinations lead to a reward of 0. Since some of the dependency measures do not allow for categorical inputs, such as the here used $PUSH$ and $PULL$, the configuration has been mapped to the numbers 0 and 1.

For the evaluation of this application, we conducted 100 experiments and measured the influence using each of the previously introduced dependency measures (cf. Section 4.1.1). For each run and each measure, two values have been calculated. The first one is the influence of robot B on robot A. The second one is the



(a) An overview of the full 100 steps evaluated.



(b) A more detailed view on the first 30 steps.

Figure 4.2: The results for the collaborative box manipulation. The graphs show the fraction of 100 runs in which the influence of robot B is detected higher as the influence of the notional robot.

influence of a notional robot. This notional robot has the same capabilities as the real robot B, but his actions do not influence robot A at all. The value is calculated to find out how reliable the detection of influencing robots is. This can be done by comparing in which percent of the runs robot B has been found more influential than the notional robot.

The results are depicted in Figure 4.2. There, we see that most of the measures perform similar in this scenario. Except for the continuous MI approximated with the Kraskov method 40 steps are sufficient to distinguish the influencing robot from a non-influencing in each of the 100 runs. The vast majority is already correct detected after 15 steps. The continuous MI eventually finds the influence in each run. However, the detection speed is rather slow. This is due to continuous MI having problems with the discrete values that are assumed in the problem. Furthermore, the continuous MI has not been calculated for less than ten steps since a minimum number of samples is required. Therefore, the first three data points should be disregarded.

Concluding the results, the influence in this elementary use case can be detected quite easily with the previously introduced method. Even though the continuous mutual information shows a little slower detection the selected measure for the task is not too important since the detection is over all quite fast.

Smart Camera Network

As a second application for the evaluation, we consider a small example from the smart camera network domain (cf. Section 2.3.2). To recap briefly, smart cameras are surveillance cameras that are equipped with computational capabilities that can be used for several tasks including image processing. Furthermore, these cameras are interconnected via a network that allows them to exchange data and coordinate. For this thesis, we stick to so-called PTZ cameras that allow for an automatic adjustment of the pan angle, tilt angle, and zoom of the camera. As previously described, there are several reasonable goals for a smart camera network. For this evaluation, we stick to the goal of a 3D-reconstruction of the observed targets. This means it is necessary to observe the targets from different perspectives at the same time. To achieve this, the cameras get a reward of 1 for each object that is observed by at least two cameras in a time step.

A top-down view on scenario SCN 1 is depicted in Figure 4.3. There, we see three black dots which represent one camera each. Around the dots, there are colored circles. Each of them represents the area that is potentially observable by one of the cameras if it chooses an according PTZ configuration. Furthermore, we see exemplary areas that are currently observed marked by red lines. Camera 1 and Camera 2 share a common area which can be observed by both at the same time. In contrast, Camera 3 is isolated and does not share a common area with one of the others. The yellow arrow marks the entry point and direction for objects that move through the scene in the area between Camera 1 and Camera 2.

Ten independent runs of this scenario have been conducted in a Mason[1] simulation, where each of the cameras assumes a uniformly sampled PTZ configuration in each time step. The pan angels are between 0 and 360 degree, the tilt angle between 120 and 180 degree, and the zoom between 12 and 18. In each one, we compared the influence of Camera 2 on Camera 1 and the influence on Camera 3 on Camera 1. For each of the measures, there are three comparisons: one for the pan, one for the tilt, and one for the zoom of the cameras. It is expected that there is a clear trend towards a higher influence of Camera 2 over Camera 3 in general and especially for the pan and tilt since these configuration determine if it is possible to gather a reward for Camera 1 or not.

The results are depicted in Figure 4.4. There, we see that after a few 100 steps most measures allow a correct detection in 100% of the cases. The continuous mutual information and maximal information coefficient on the other hand take a little longer to reach this level of certainty. For the pan, we see that the distance correlation shows the best result ahead of the other measures. However, the other measures work as well with a higher sample size. Even though the MIC finds an influence quite reliable the other measure do not show a definite result which can be explained with the minor influence of this configuration components, i.e., in most cases, the zoom does not determine if there is a positive reward at all but only the height of this positive reward.

Concluding the results, we have seen that it is possible to detect the influences in simple examples based on real-world applications, such as smart camera networks.

[1] Mason is a multi-agent simulation framework for Java [108].

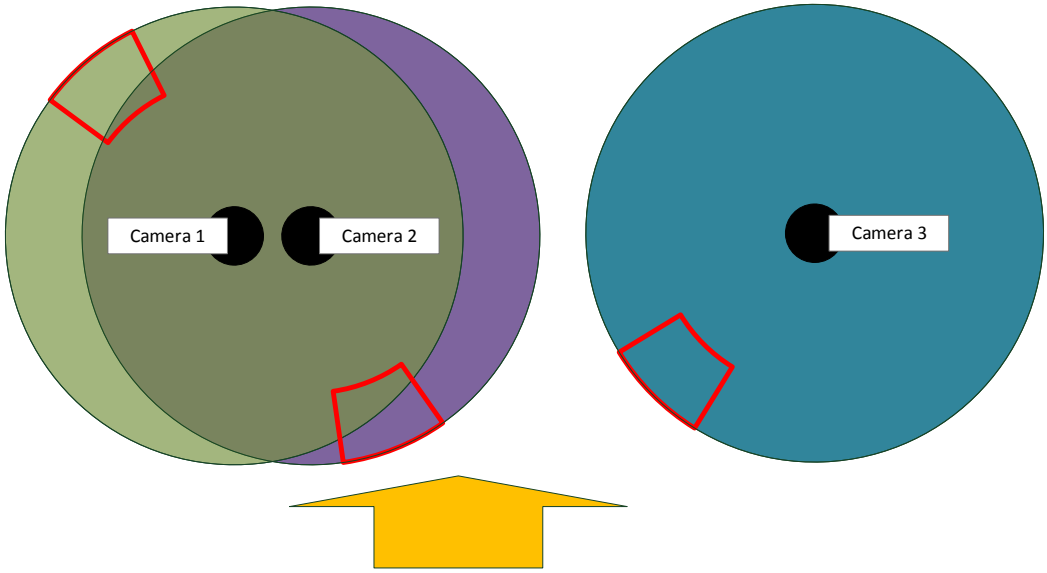
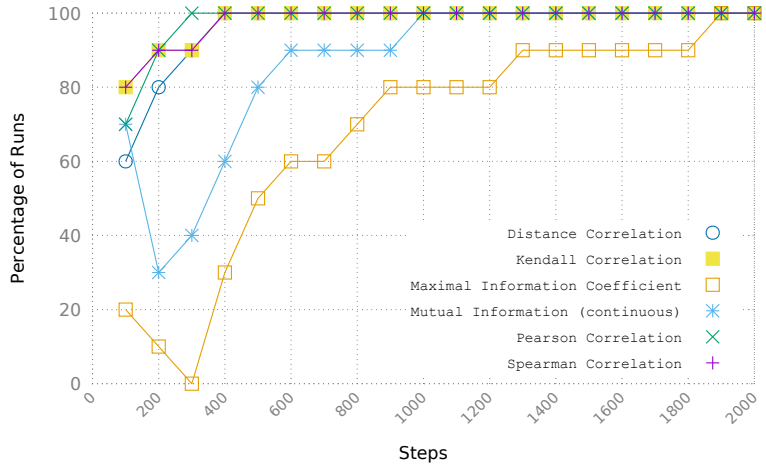
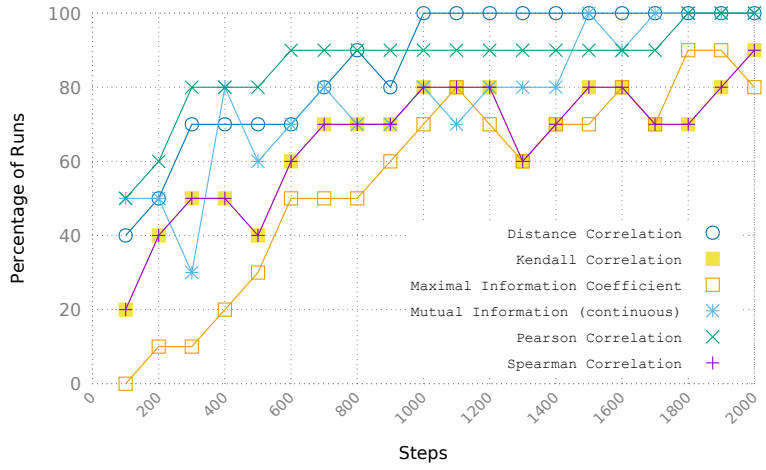


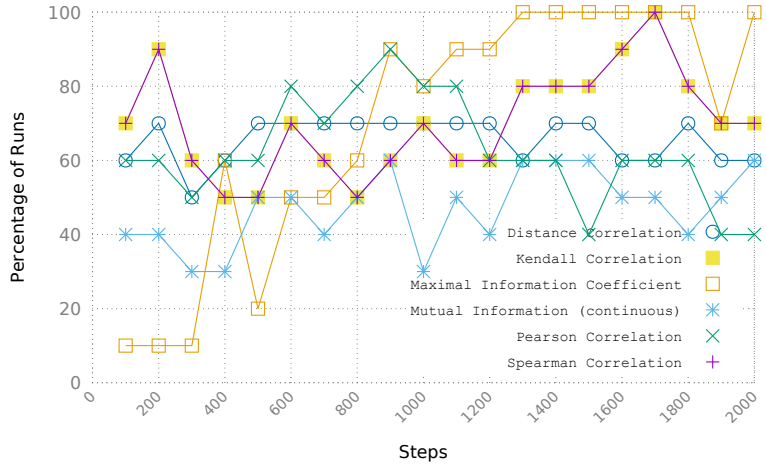
Figure 4.3: *SCN 1*. A top-down view on a smart camera network. The black dots depict cameras surrounded by a circle that marks their potential observable area. The red shapes show the field of view for an exemplary PTZ configuration. The yellow arrow indicates from where and in which direction the objects of interest move.



(a) The results for the pan.



(b) The results for the tilt.



(c) The results for the zoom.

Figure 4.4: The results for scenario *SCN 1* using the general method. Each graph shows the fraction of runs in which the influence of the configuration component (pan, tilt, or zoom) of Camera 1 on Camera 0 is detected to be higher than the influence of Camera 2.

Over the next section, we will see how to use it in scenarios with more complex requirements.

4.2 Summary

This chapter covers the main method of influence detection. In Section 4.1, the workflow that describes the sequence of events that take place if the influence detection is applied is introduced. Afterwards, the estimation step is specified and the candidates for the dependency measure are presented in Section 4.1.1. The calculation of the influence is further clarified by applying it exemplary to the collaborative box manipulation in Section 4.1.2. Finally, in Section 4.1.3, the method has been applied to show its usefulness. Concluding the chapter, how the general method of influence works and that it can detect the influences between systems.

In the following chapters, we will extend the general method by three means. The first is the consideration of other configuration components that will allow to find influences that only come to light for specific combinations of configuration components in Chapter 5. Second, dalyed influences, i.e., influences that reveal itself after a period of time, are given attention to in Section 6. Furthermore, in Section 7 it will be explored how the method can be applied during the runtime of the system. Furthermore, in Section 8, it is described how the evaluation and adaption step from the workflow are realized during runtime using reinforcement learning algorithms.

5 | Multi-component Influences

In Section 4.1, we introduced the general workflow for the influence detection, saw examples, and evaluations for simple scenarios. In this section, we cover an extension of it that can be useful for more complex cases. That is a possibility for the consideration of other configuration components, e.g., if the influence pattern is only visible if a system examines the past experiences with respect to the other configuration components. To achieve this, we introduce a more advanced estimation of the influence, i.e., adapt the technique used in the evaluation step of the workflow.

5.1 Methodology for Detection

If using the so far proposed general method (cf. Section 4.1) alone it can lead to unsatisfying results for some scenarios. This is since the other configuration components are not taken into account. For example, in some smart camera network scenarios it might not be possible to identify the influence by only looking at the pan angle isolated, but it is necessary to consider certain combinations of the pan angle and the tilt angle of a camera. In the following, we do not distinguish if this other configuration is part of the influencing system or the influenced system since the approach is equivalent in both cases, but we stick to the case where each of the configuration components belong to different systems. In this case, if we want to measure the influence of a system A_1 on a system A_2, we condition the calculation with the configuration of A_2. This means we calculate the dependency of the reward and the configuration for each configuration of A_2 separately. This works for discrete random variables, but it becomes infeasible for systems with infinite number of configurations for the component since it is not possible to calculate an infinitely number of values. Therefore, we split the configuration components in two (or possibly

more) parts and calculate the dependency separately. This ensures a rough approximation of the dependency under each configuration and ensures a high sample size.

Let us consider the detection of influences of a Camera 2 on a Camera 1, where we face n samples $(f_1, p_2)_t$. Here, f_1 is local reward of Camera 1 at time t, and p_2 is the pan of Camera 2. Following the before introduced method, we estimate the dependency between this random variables from the samples. However, this might not lead to a satisfying result in some cases where the outcome depends on multiple configuration components (see next section for an example). To resolve this issue, the samples are sorted in two (or more) categories depending on the other configuration component at time t. E.g., we sort the samples $(f_1, p_2)_t$ by the value of $p_1 \in [0, 360)$ at time t which is the pan of Camera 1, i.e., all samples where $p_1 \in [0, 180)$ are collected in one bucket and all samples where $p_1 \in [180, 360)$ are collected in a second bucket. This results in two sets of samples that have roughly the size $n/2$ (assuming the pan of Camera 1 is uniformly distributed). Afterwards, the dependency is estimated for both sets independently and the both values are aggregated for example by using the average. Technically spoken this calculation results in a rough approximation of the dependency between the reward and the configuration component conditioned by the other configuration component. If necessary the approximation can be made more accurate by using more parts. However, in this work, we stick to two parts since it resolves the issue and at the same time the sample size for each estimation remains the highest.

5.2 Example

In contrast to the example for the general workflow (cf. Section 4.1.2), we do not rely on the collaborative box manipulation but on the two-man saw application (cf. Section 2.3.1). A brief reminder: this use case has two robots that operate a two-man saw by $PUSH$ing or $PULL$ing it. The saw can be in two positions: the left or the right position. If it is in the left position, the left robot has to $PUSH$ and the right has to $PULL$ to move the saw and generate a reward of 1. Otherwise the reward is 0. If the saw is in the right position they have to apply the contrary configurations to move the saw. The reward for each outcome is depicted in Table 5.1 and 5.2. Again, a human can easily recognize that the robots influence each other. In the following, we will see that the algorithm can fail to detect this influence if applied inappropriately and how a correct application can be realized, which returns a value

Reward of left system (f_L) if the saw is in left position		Configuration of left system (C_L)	
		PUSH	PULL
Configuration of right system (C_R)	PUSH	0	0
	PULL	1	0

Table 5.1: The reward of left system L resulting from different configurations of system L and R in the two-man saw example if the saw is in the left position.

Reward of left system (f_L) if the saw is in right position		Configuration of left system (C_L)	
		PUSH	PULL
Configuration of right system (C_R)	PUSH	0	1
	PULL	0	0

Table 5.2: The reward of left system L resulting from different configurations of system L and R in the two-man saw example if the saw is in the right position.

greater than zero.

In this example, we see that the basic method fails to detect the influence in some scenarios. This can be resolved when the own configurations of the systems are taken into account. For the calculations we need to derive the probability distribution of the reward of the left robot f_L, the joint distribution of f_L and the configuration of the right robot C_R from Table 5.1 and 5.2. The deduction works similar to the one in the box example and is therefore left to the reader. We only face a slightly different situation since the correct direction depends on the current position of the saw. To find the distributions, we have to adapt the distributions for the left and right position, i.e., after deriving the (conditional) probabilities for each position we can calculate $p(X) = 0.5p(X|\text{saw is in left position}) + 0.5p(X|\text{saw is in right position})$. We use the *weighting* of 0.5 since we can safely assume that the saw is in the left position in half the cases, because of the distributions of PUSH and PULL[1]. Expressed in a more technical way, we face a *prior probability* to be in a certain position. When we

[1]This can be can be extended to cases with more states where the weighting will be $1/n$ as long as the probability to switch from one state to another is uniformly distributed.

calculate the probabilities in this manner, we get:

$$\begin{aligned}
&p(C_R = \text{PUSH}) = p(C_R = \text{PULL}) = 0.5 \\
&p(f_L = 1 | C_L = \text{PUSH}) = 0.25 \\
&p(f_L = 0 | C_L = \text{PUSH}) = 0.75 \\
&p(f_L = 1, C_R = \text{PULL} | C_L = \text{PUSH}) = 0.25 \\
&p(f_L = 0, C_R = \text{PULL} | C_L = \text{PUSH}) = 0.25 \\
&p(f_L = 1, C_R = \text{PUSH} | C_L = \text{PUSH}) = 0 \\
&p(f_L = 0, C_R = \text{PUSH} | C_L = \text{PUSH}) = 0.5
\end{aligned} \tag{5.1}$$

Using these probabilities, we can calculate the mutual information as in the box example:

$$\begin{aligned}
I(f_L; C_R) &= p(f_L = 1, C_R = \text{PULL}) \operatorname{ld} \left(\frac{p(f_L = 1, C_R = \text{PULL})}{p(f_L = 1) p(C_R = \text{PULL})} \right) \\
&+ p(f_L = 1, C_R = \text{PUSH}) \operatorname{ld} \left(\frac{p(f_L = 1, C_R = \text{PUSH})}{p(f_L = 1) p(C_R = \text{PUSH})} \right) \\
&+ p(f_L = 0, C_R = \text{PULL}) \operatorname{ld} \left(\frac{p(f_L = 0, C_R = \text{PULL})}{p(f_L = 0) p(C_R = \text{PULL})} \right) \\
&+ p(f_L = 0, C_R = \text{PUSH}) \operatorname{ld} \left(\frac{p(f_L = 0, C_R = \text{PUSH})}{p(f_L = 0) p(C_R = \text{PUSH})} \right) \\
&= 0.125 \operatorname{ld} \left(\frac{0.125}{0.25\ 0.5} \right) + 0.125 \operatorname{ld} \left(\frac{0.125}{0.25\ 0.5} \right) \\
&+ 0.375 \operatorname{ld} \left(\frac{0.375}{0.75\ 0.5} \right) + 0.375 \operatorname{ld} \left(\frac{0.375}{0.75\ 0.5} \right) \\
&= 0
\end{aligned} \tag{5.2}$$

Our expected result here would be that the calculation gives a value higher than zero since we clearly have influences in the described scenario. The reason for the apparent problem is that it is necessary to consider the configurations of the left robot for the detection of the influence. This can be done by conditioning the probabilities with the configurations of the left robot, i.e., we calculate

the conditional mutual information $I(f_L; C_R|C_L) = p(C_L = \text{PUSH})I(f_L; C_R|C_L = \text{PUSH}) + p(C_L = PULL)I(f_L; C_R|C_L = \text{PULL}) = 0.5I(f_L; C_R|C_L = \text{PUSH}) + 0.5I(f_L; C_R|C_L = \text{PULL})$.

$I(f_L; C_R|C_L = \text{PUSH})$ can be calculated as follows:

$$
\begin{aligned}
I(f_L; C_R|C_L = \text{PUSH}) = {} & p(f_L = 1, C_R = \text{PULL}|C_L = \text{PUSH}) \\
& \text{ld}\left(\frac{p(f_L = 1, C_R = \text{PULL}|C_L = \text{PUSH})}{p(f_L = 1|C_L = \text{PUSH})p(C_R = \text{PULL}|C_L = \text{PUSH})}\right) \\
& + p(f_L = 1, C_R = \text{PUSH}|C_L = \text{PUSH}) \\
& \text{ld}\left(\frac{p(f_L = 1, C_R = \text{PUSH}|C_L = \text{PUSH})}{p(f_L = 1|C_L = \text{PUSH})p(C_R = \text{PUSH}|C_L = \text{PUSH})}\right) \\
& + p(f_L = 0, C_R = \text{PULL}|C_L = \text{PUSH}) \\
& \text{ld}\left(\frac{p(f_L = 0, C_R = \text{PULL}|C_L = \text{PUSH})}{p(f_L = 0|C_L = \text{PUSH})p(C_R = \text{PULL}|C_L = \text{PUSH})}\right) \\
& + p(f_L = 1, C_R = \text{PUSH}|C_L = \text{PUSH}) \\
& \text{ld}\left(\frac{p(f_L = 0, C_R = \text{PUSH}|C_L = \text{PUSH})}{p(f_L = 0|C_L = \text{PUSH})p(C_R = \text{PUSH}|C_L = \text{PUSH})}\right) \\
& = 0.25\,\text{ld}\left(\frac{0.25}{0.25\ 0.5}\right) + 0\,\text{ld}\left(\frac{0}{0.25\ 0.5}\right) \\
& + 0.25\,\text{ld}\left(\frac{0.25}{0.75\ 0.5}\right) + 0.5\,\text{ld}\left(\frac{0.5}{0.75\ 0.5}\right) \\
& \approx 0.31
\end{aligned}
\tag{5.3}
$$

The value of $I(f_L; C_R|C_L = \text{PULL})$ can be deducted in a similar manner as before, and the calculations are therefore omitted. For the influence value, we use the sum of each of the parts $I(f_L; C_R|C_L = \text{PULL}) + I(f_L; C_R|C_L = \text{PUSH})$. In this case, we receive a value of approximately 0.62 from the calculations. This value shows us that there is a correlation between the configuration of the right robot and the reward of the left robot. Since the configuration is independent and uniformly distributed this implies that the right robot influences the left robot. Consequently, the influence

detection works as expected also in more complicated scenarios.

5.3 Evaluation

In the previous section, we have seen an example calculation that shows us the importance of the consideration of other configurations for the influence detection. In the following, we discuss the topic from a more practical point of view and show the effect of such cases on the influence detection. We exemplary evaluate an elementary use case, the two-man saw, and a smart camera network scenario and see the effect of the other configurations on the detection. For both of the evaluations, we compare the standard method and the method of conditioned measurement.

5.3.1 Two-man Saw

First, we evaluate the conditioned measurement in an elementary use case, the two-man saw. It has been introduced in Section 2.3.1 and theoretically analyzed in the previous Section 5.2. The example is inspired by two robots that operate a saw that only moves if both of them move it in the right direction, i.e., one $PUSH$es and the other $PULL$s or vice versa depending on the current position of the saw. If the saw moves each robot gets a reward of 1 otherwise 0. Again, the categorical configurations $PUSH$ and $PULL$ have been mapped on numerical values to make all dependency measures applicable.

The results are depicted in Figure 5.1. As for the collaborative box manipulation, the actual influence has been calculated using the seven dependency measures and compared with a measurement of a notional robot that assumes uniformly distributed random configurations but has no influence on the actual outcome of the experiment. The figure shows in how many of the 100 independent runs the influencing robot has been found to be more influential than the notional (not-influencing) robot. The first graph shows the result for the method used in the previous evaluations, i.e., without a consideration of other configurations. The detection is between 30% and 50% which is below the expected value of 50%. This due to the fact that the run will only be counted as correct detected if the value is higher but not if both values are equal which is quite often the case in this scenario. The second graph shows the results with the consideration of the other configuration, i.e., the calculation of the dependencies is conditioned under the configuration of the first robot. We see that influence is nearly perfect detected after 30 steps which is close to the result in the

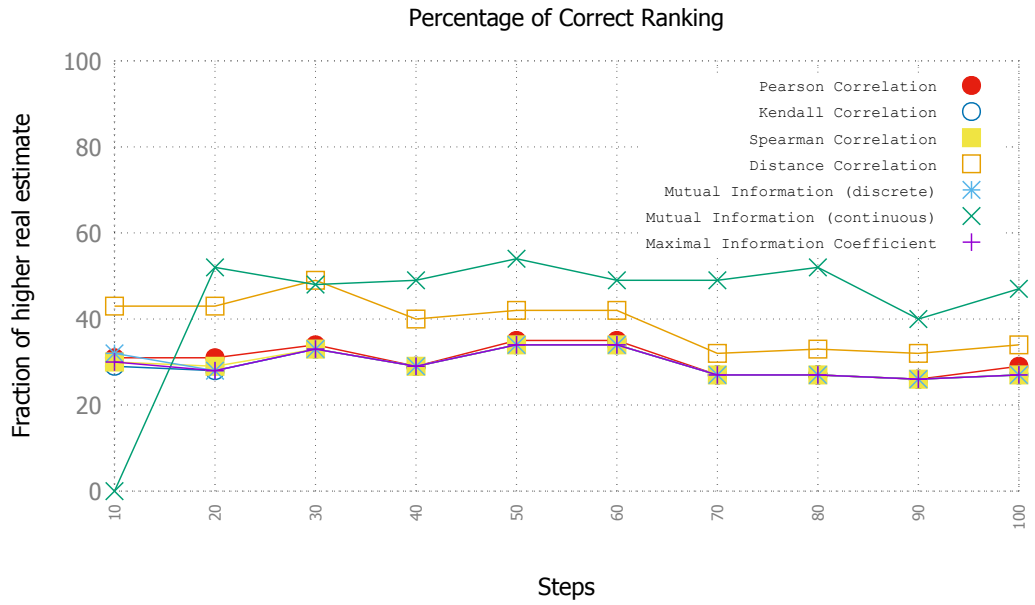
collaborative box manipulation. Also similar is the small weakness of the continuous MI which needs about 90 steps to catch up.

Concluding the evaluation of the two-man saw application, we have seen that even in relatively simple applications the problem with other configurations can lead to influences that can not be detected with the basic variant of the influence detection. However, the improvement that includes the other configurations by conditioning the measurement shows very good results for such cases as well.

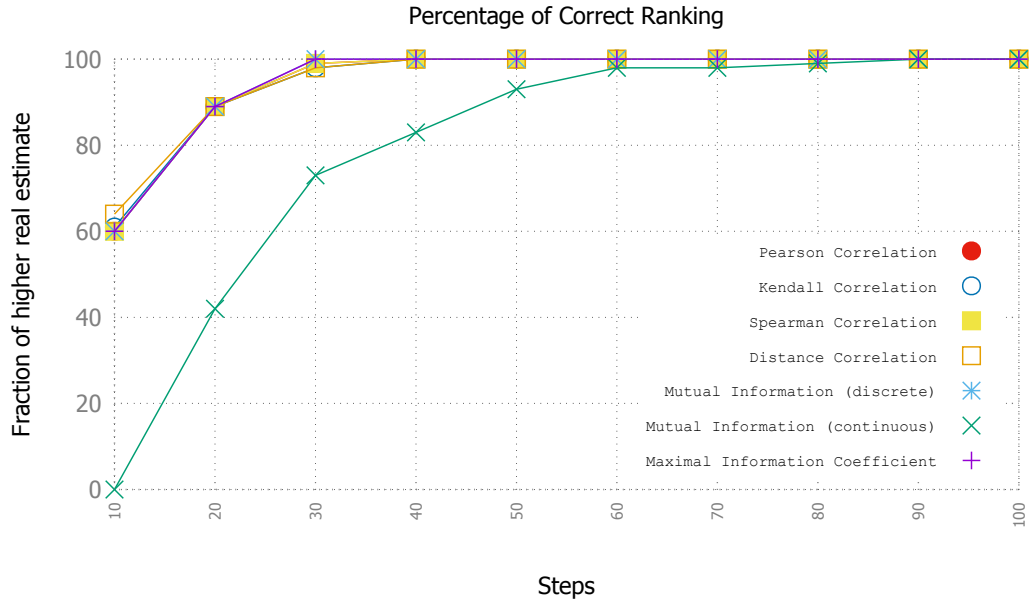
5.3.2 Smart Camera Network

After the elementary use case, we focus on a more complicated use case from the smart camera network domain. A top-down view on *Scenario 2* is depicted in Figure 5.2. It is similar to scenario SCN 1, which we used for the evaluation in Section 4.1.3. In the figure, we see three black dots that mark the position of the cameras. Camera 1 and Camera 2 are quite close and potentially share a common field of view which is marked by the colors areas around the cameras. Camera 3 on the other hand is separated and cannot observe the same areas as the other two cameras. The only difference to the previously evaluated scenario is where the objects of interest appear and move. In the last scenario they were limited to the area between Camera 2 and 3 appearing from the south. Here, we see that they appear from the north and the south on the whole area around Camera 1 and 2, i.e., the two cameras are surrounded by objects. The influence from Camera 2 and Camera 3 on Camera 1 has been measured for the pan, the tilt, and the zoom. We expect that the pan and the tilt of Camera 2 have the most influence. Since a reward greater than 0 is much more likely in this scenario than in the previous, we also expect the zoom to play a more important role.

The results are depicted in Figure 5.3 and 5.4. As before, 10 independent runs have been conducted and we see the ratio of runs in which the configuration components of Camera 2 have been identified as more influencing that of Camera 3. In Figure 5.3, we see the results for the previous method. Even though the camera placement is identical to the previous scenario, we see that the pan is not detected within the first 2000 steps. Furthermore, the tilt cannot be detected by the Pearson correlation. This is because it is necessary to consider the other configuration components in this scenario. Therefore, we adopted the conditioned calculation, i.e., we split each of the other configurations in two parts and sort the points in different buckets depending on which configuration has been assumed. Here, we



(a) The detection if a single estimator is used and the own configuration is not considered.



(b) The detection if the own configuration is considered, i.e., there is one estimator for each of the configurations (Push and Pull).

Figure 5.1: The results for the two-man saw use case.

face 5 other configurations since we analyze the pan and the other configuration components are the tilt and zoom of the influencing camera and the pan, tilt, and zoom of the influenced camera. Each of the configuration components will be split in two parts and the samples are sorted into buckets that are determined by each of the configuration components values. This leads to $2^5 = 32$ buckets. The values calculated for the buckets are then added up to get an aggregated result. The results achieved by this method are depicted in Figure 5.4. We see that using this method the issues experienced in the first experiment do not appear and a flawless detection is ensured.

Concluding the results, we have seen that small changes in the setting can make it necessary to consider the other configuration components in the influence detection. A fast and reliable detection can be achieved by applying the method proposed in this chapter.

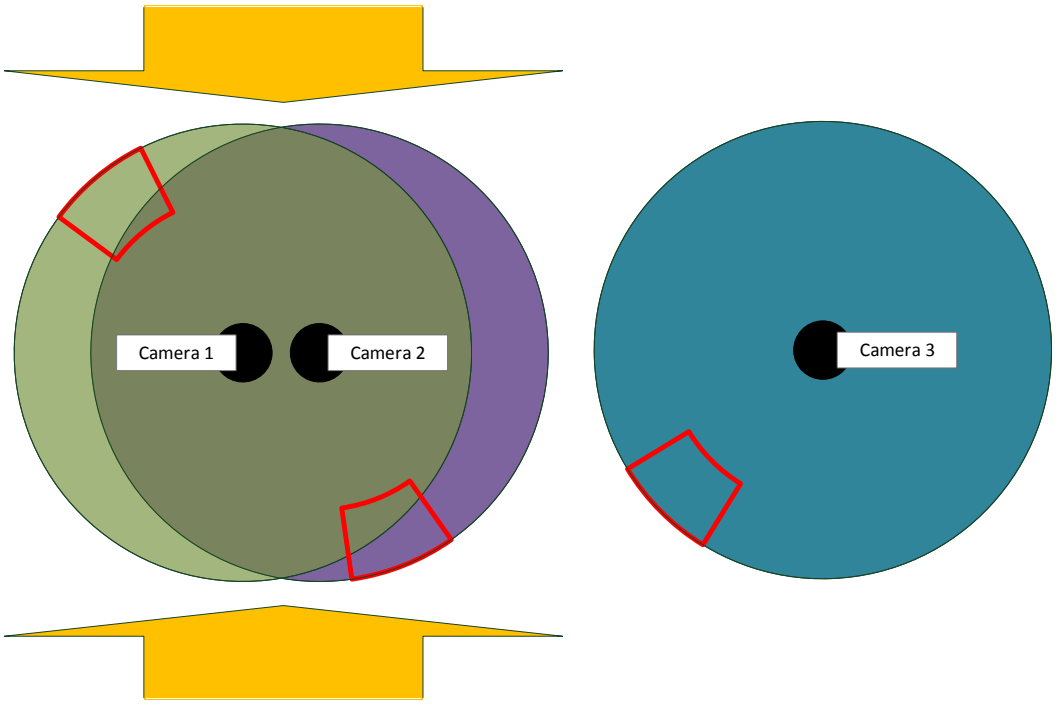
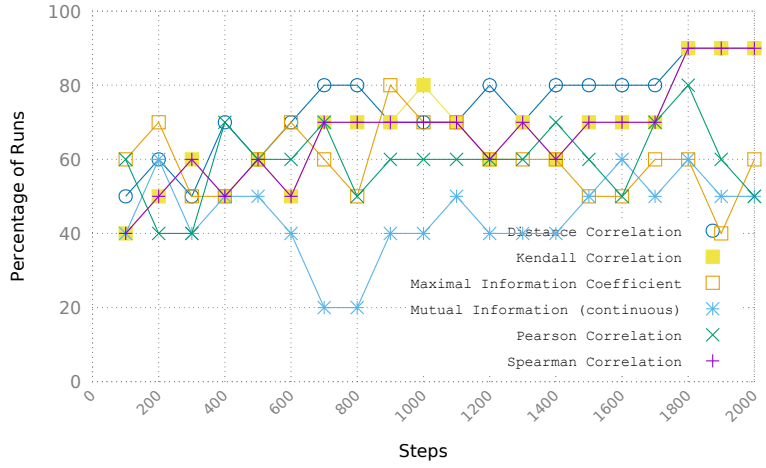
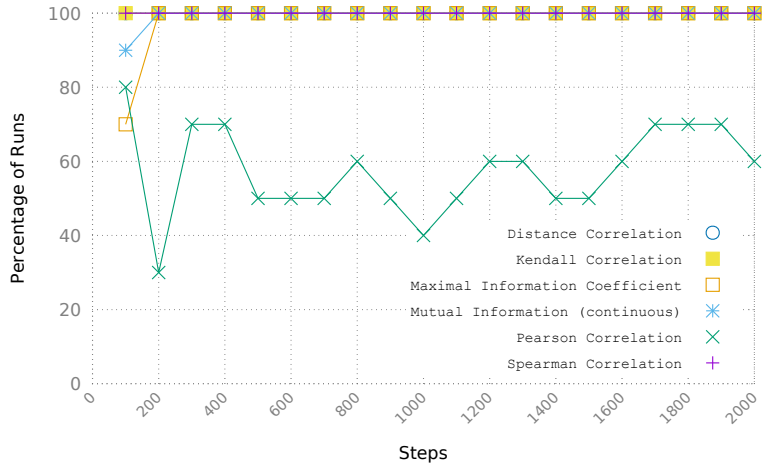


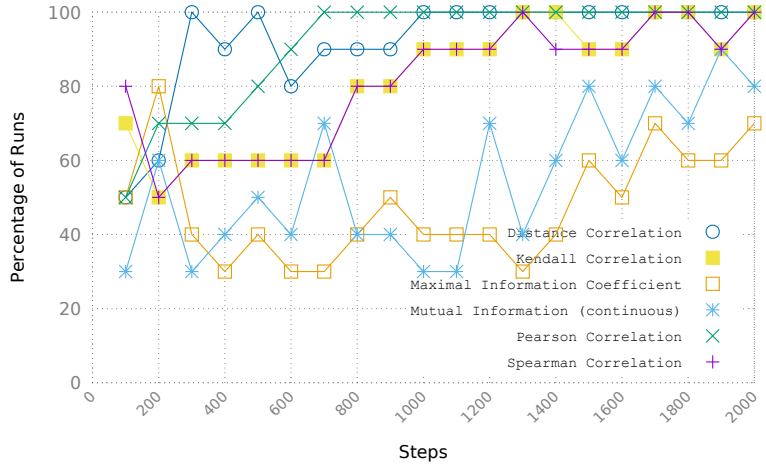
Figure 5.2: *SCN 2*. A top-down view on a smart camera network. This scenario is only slightly changed from SCN 1 by adjusting the flow of targets represented by the yellow arrows. The black dots depict cameras surrounded by a circle that marks their potential observable area. The red shapes show the field of view for an exemplary PTZ configuration.



(a) The results for the pan.

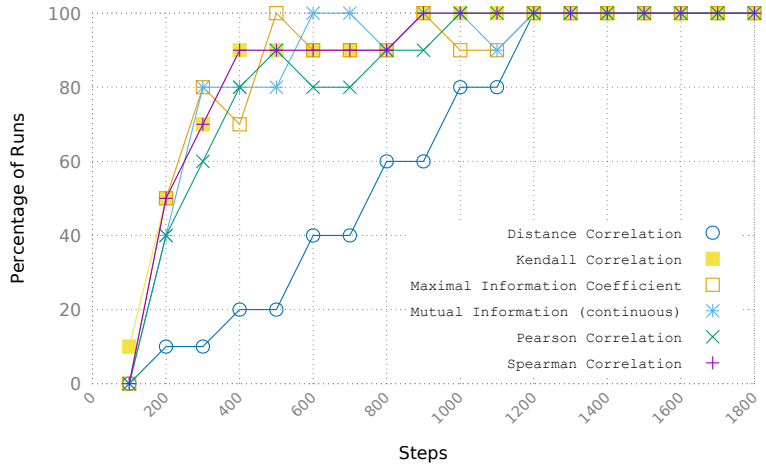


(b) The results for the tilt.

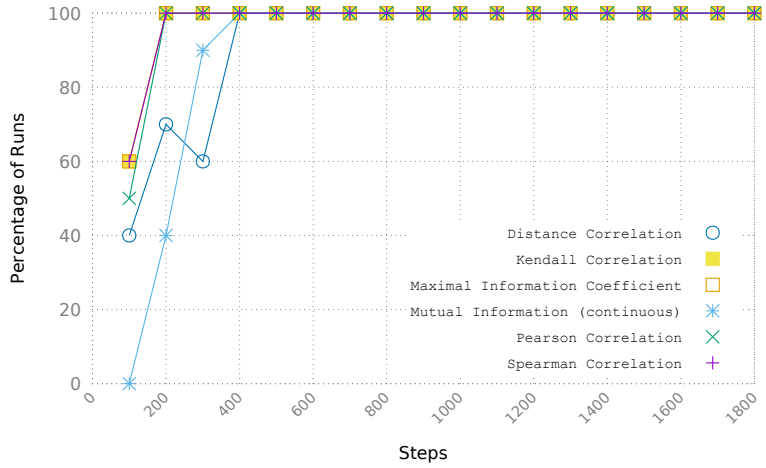


(c) The results for the zoom.

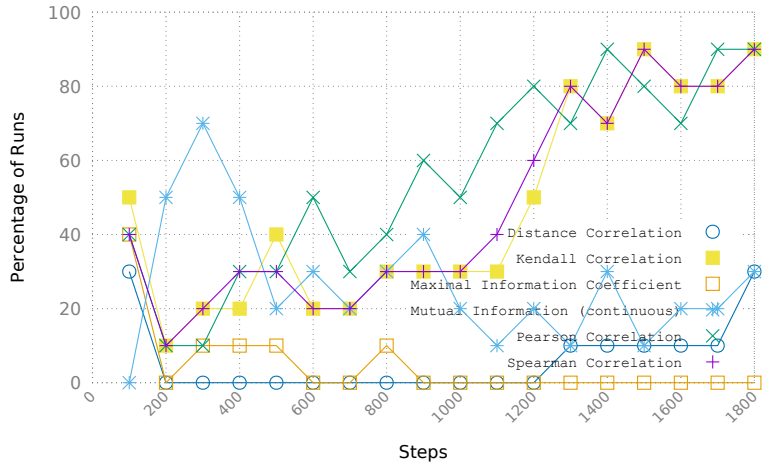
Figure 5.3: The results for the scenario SC2 using no conditioning. Each graph shows the fraction of runs in which the influence of the configuration component (pan, tilt, or zoom) of Camera 1 on Camera 0 is detected to be higher than the influence of Camera 2.



(a) The results for the pan.



(b) The results for the tilt.



(c) The results for the zoom.

Figure 5.4: The results for the scenario SC2 using two parts for conditioning. Each graph shows the fraction of runs in which the influence of the configuration component (pan, tilt, or zoom) of Camera 1 on Camera 0 is detected to be higher than the influence of Camera 2.

5.4 Summary

In Chapter 4, the general methodology for the influence detection has been introduced. In this chapter, this methodology has been extended to find the influences that are dependent on multiple configuration components by refining the estimation step of the general workflow (cf. 5.1). We have seen how influences from multiple configuration components can skew the measurement possibly to a degree that the influence cannot be detected and how this issue can be addressed in the two-man saw application in Section 5.2. Furthermore, in Section 5.3, the method has been evaluated in the smart camera application, where it shows good results in a practical setting.

In the following chapters, the estimation step in the general workflow of the influence detection will be further extended and examined regarding the detection of delayed influences in Chapter 6 and the applicability at runtime in Chapter 7. Furthermore, the evaluation and adaption step will be in the focus in Chapter 8.

6 | Delayed Influences

In this section, we extend the basic methodology (cf. Section 4.1) by additionally regarding the configuration that has not appeared in the same time step but in previous time steps. For instance, if we are interested in the influence that will appear after 5 time steps, we analyze the association between the current reward of a system A and the configuration that has been assumed by a system B 5 steps before. Using this extension, we can discover the influences that appear after a period of time.

Additionally, we compare this basic calculation method with a more complex one that allows to take the configuration of system A into account, which in the basic process only contributes the reward to the calculation. This is done by conditioning the calculation under these configurations what has been described in Chapter 5.

6.1 Methodology for Detection

Before we show how such delayed influences can be handled, we clarify when they appear by examining a thought experiment. Considering a big river that is used by industrial facilities, we can imagine that at one point there is a factory that produces cars. If the sun is shining the factory runs by solar power but if it is cloudy or night-time it uses an electrical generator that is run by petroleum. The generator will discharge petrochemicals which lead to water pollution. A few kilometers down the river a new fish farm opens and notices that the chemicals harm their fish. The fish farmer install an electric filter system that can be turned on or off. Since the fish farm has to pay the expenses for the electric energy it is desirable to minimize the time the filter system is active. Here, we see a simple system in which a delayed influence with a delay of time x that the chemicals need to travel from the factory to

the fish farm is apparent. The optimal behavior of the filter system would be to turn on at $t_s + x$ where t_s is the start time of the generator and turn of at $t_e + x$ where t_e is the time the generator stops. In the following, we see how the influence and the x can be determined by extending the before introduced techniques. To achieve this, we introduce a more advanced estimation of the influence, i.e., adapt the technique used in the evaluation step of the workflow.

The main idea is to introduce several influence detectors; one for each delay to analyze. For example, if influences with a delay of up to 10 time steps should be detected, 10 additional influence detectors are used and each of them will measure the correlation between reward and configuration but with a different offset. The previously introduced influence detector has a delay of 0, i.e., it catches instantaneous influences. At each time step t a sample (r_t, c_t) is created from the reward r_t and the configuration c_t at time step t and used for the detection. Now, we introduce a separate detector for every possible delay. As depicted in Figure 6.1, the detector for a delay of 1 uses the sample (r_t, c_{t-1}) and the detector for a delay of 2 uses the sample (r_t, c_{t-2}) and so on. In Table 6.1, we see which samples are created in which time step for which detector regarding the detection of influences from system A on system B. A delay of k can naturally only be detected if at least $k + x$ steps have been done by the system where x is the number samples needed to detect the influence.

6.2 Evaluation

In the following, we present the results obtained in an Industry 4.0 domain. This application has been introduced initially in Chapter 2.3.3. Let us recall briefly, the smart factory application builds up current developments in the Industry 4.0 domain. The scenarios are composed from pick and place robots and configurable work benches that are placed next to a transport system. Two such workbenches that are interconnected with a transport system form what we call a workstation. The work pieces that travel via the transport system can be picked up from one workstation and placed on another by the robots.

The actual scenario used for the evaluation is depicted in Figure 6.2. There, we see two workstations, on the left S_L and on the right S_R and a pick and place robot R. Each of the stations consists of two workbenches that are connected via movers. The workbenches are named B_{iU} and B_{iL}, for upper and lower, where

Time	Reward System B	Config. System A	Sample no delay	Sample delay 1	Sample delay 2	Sample delay 3	...	Sample delay k
1	r_1	c_1	(r_1, c_1)	-	-	-	...	-
2	r_2	c_2	(r_2, c_2)	(r_2, c_1)	-	-	...	-
3	r_3	c_3	(r_3, c_3)	(r_3, c_2)	(r_3, c_1)	-	...	-
4	r_4	c_4	(r_4, c_4)	(r_4, c_3)	(r_4, c_2)	(r_4, c_1)	...	-
5	r_5	c_5	(r_5, c_5)	(r_5, c_4)	(r_5, c_3)	(r_5, c_2)	...	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
k	r_k	c_k	(r_k, c_k)	(r_k, c_{k-1})	(r_k, c_{k-2})	(r_k, c_{k-3})	...	-
$k+1$	r_{k+1}	c_{k+1}	(r_{k+1}, c_{k+1})	(r_{k+1}, c_k)	(r_{k+1}, c_{k-1})	(r_{k+1}, c_{k-2})	...	(r_{k+1}, c_1)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
$n-k$	r_{n-k}	c_{n-k}	(r_{n-k}, c_{n-k})	(r_{n-k}, c_{n-k-1})	(r_{n-k}, c_{n-k-2})	(r_{n-k}, c_{n-k-3})	...	(r_{n-k}, c_{n-2k})
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
$n-2$	r_{n-2}	c_{n-2}	(r_{n-2}, c_{n-2})	(r_{n-2}, c_{n-3})	(r_{n-2}, c_{n-4})	(r_{n-2}, c_{n-5})	...	(r_{n-2}, c_{n-k-2})
$n-1$	r_{n-1}	c_{n-1}	(r_{n-1}, c_{n-1})	(r_{n-1}, c_{n-2})	(r_{n-1}, c_{n-3})	(r_{n-1}, c_{n-4})	...	(r_{n-1}, c_{n-k-1})
n	r_n	c_n	(r_n, c_n)	(r_n, c_{n-1})	(r_n, c_{n-2})	(r_n, c_{n-3})	...	(r_n, c_{n-k})

Table 6.1: Composition of samples for the estimators with different delays. n denotes the number of time steps the system has run and $k < n$ is the maximum delay that will be analyzed. On the left side, the data points gathered at a specific time if interested in the influence from system A on system B are depicted. On the right side the samples for the influence detection with delays that can be composed in each time step are shown. The *sample for* $t - i$ are the samples that will be used to detect influences with a delay of i steps.

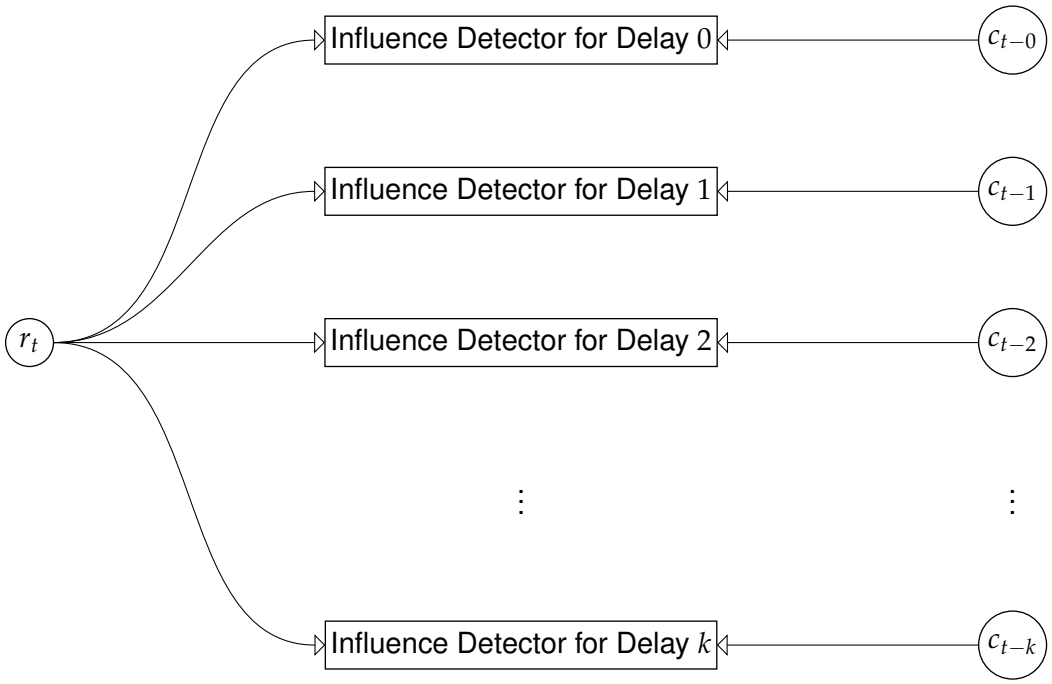


Figure 6.1: The concept for the detection of delayed influences.

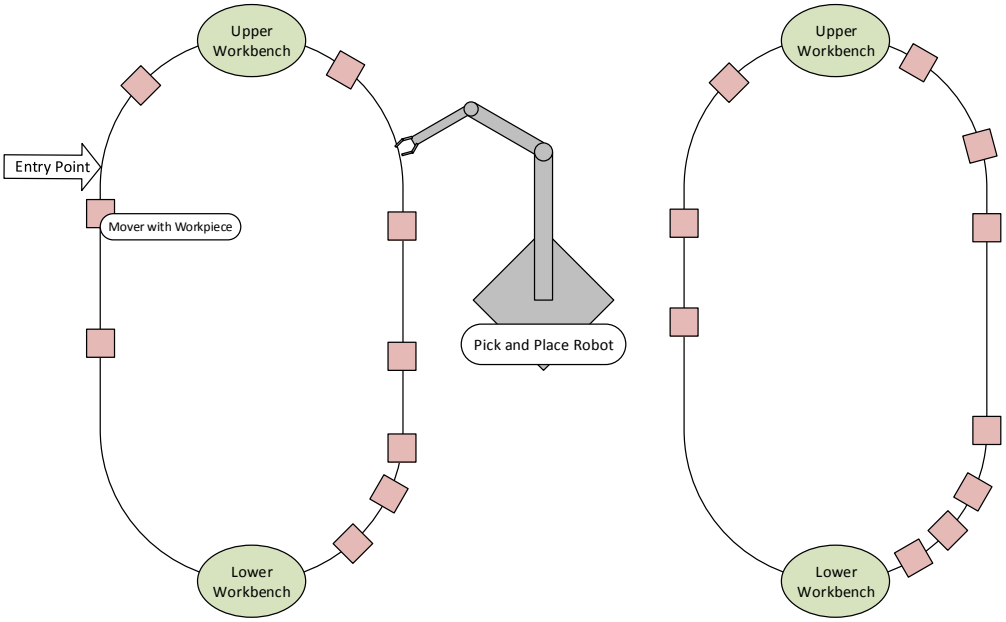


Figure 6.2: A top-down view on the smart factory scenario. The green ovals mark workbenches. Two each are interconnected by movers, depicted as red squares, which can transport workpieces clockwise on the track, shown as black line. In the middle, in gray, there is a robot that can pick the work pieces from one mover up and place them on empty movers at dedicated spaces.

$i \in \{L, R\}$ stands for left or right. Each of the benches has different possibilities to configure themselves: B_{LU} can chose from $\{0,1,2\}$, B_{LL} from $\{0,1,3\}$, B_{RU} from $\{0,5,6\}$, and B_{RL} from $\{0,3,7\}$ where the configuration 0 means that the bench is not working. The other configurations may be different drills or saws, for instance. For the purpose of measuring the influence in the system, we assume that the work benches and the pick and place robot take a random configuration in each step, where the configurations are uniformly distributed. Each time step, a work piece is introduced in the simulation at the entry point. All of these work pieces have to be handled with configuration 2 and afterwards with configuration 5 to be useful for the further process. For the reward, we add a function to the work benches that gives a value of 1 if the correct configuration for the work piece has been chosen and 0 otherwise. For this scenario, we expect to find an influence of work bench B_{LU} on B_{RU} with a delay of 5 time steps. This is because the travel time from bench B_{LU} to B_{RU} via the pick and place robot is 5.

For the results, we present the values of association measured between the reward of B_{RU} and the past configurations of B_{LU}. Additionally, the values of association have been compared with a value that has been created independently and with no association at all. The evaluation comprises the standard measurement and the measurement with the consideration of other configuration components from Chapter 5. The first is not considering the own configuration of B_{RU} and the second does consider these configurations. The results are based on 100 independent runs that are simulating the first 700 steps each. The results for a delay of 0-9 time steps have been measured with seven different dependency measures. The simulation has been conducted in a Mason simulation and each of the workstations chooses a configuration from a uniform distribution in each step.

In Figure 6.3-6.8, we see the results for the measured values of three different dependency measures. These three measures have been selected exemplary: the maximal information coefficient as a representative of the modern, very powerful measures, the Pearson correlation as a representative of the classical, simpler approaches that need less computational power, and the continuous mutual information as an inappropriate choice for this system type. The results for the other measures can be found in Appendix A. Each of the figures shows two graphs; the first is a comparison between the influence estimation from B_{LU} on B_{RU} and the estimation of the influence from a notional, independent workbench. In Figure 6.3-6.5, we see the results if the configuration of B_{RU} is not considered, i.e., the general method from Section 4.1 is applied. In the remaining Figure 6.6-6.8, this configuration has been considered, i.e., the method described in Section 5.

Considering the second graph in each figure, as expected, we can see that the values for the delay of 0-4 time steps and 6-9 time steps are rather low and on an equal level for each dependency measure. For a delay of 5 steps, we see that the values for the maximal information coefficient and the Pearson correlation are significantly higher than for the rest of the steps. An exception is the value of the continuous mutual information that is approximated with the Kraskov method where the values remain on an equal level that can be seen in Figure 6.5 and 6.8.

In the upper graphs, we see the percentage of runs in which the measured value is higher than the comparison value that is calculated using an independent random variable. We see that the values of a delay of 0-4 and 6-9 steps are at about 50%. This means that for this delay we do not see that an influence has been detected here. In contrast, we see that for a delay of 5 steps the maximal

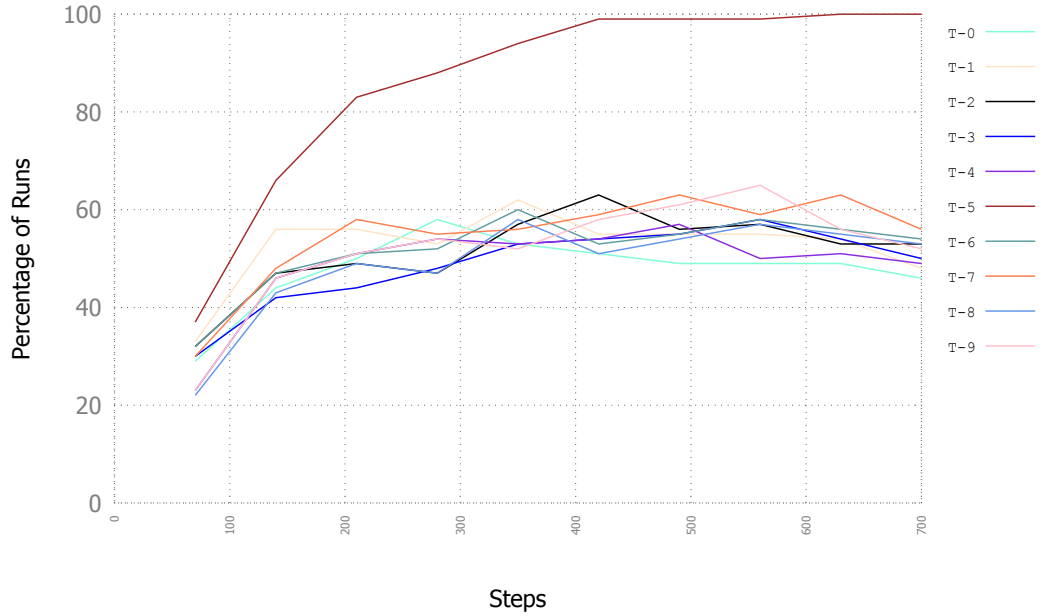
information coefficient and the Pearson correlation find an influence, i.e., the values are higher in 100% of the runs. Again, we see an exception for the Kraskov method which does not recognize the influence.

Concluding the results, we have seen that the method for the detection of delayed influences works as good as on the immediate influences. Nearly each measure can very reliable find the influence within the 700 steps and the differences between them are quite marginal. One exception is the continuous mutual information that again shows that it is not useful for discrete configuration spaces. Furthermore, we see that the configuration of B_{RU} does not have to be considered here but if taken into account does change the quality of detection only marginally.

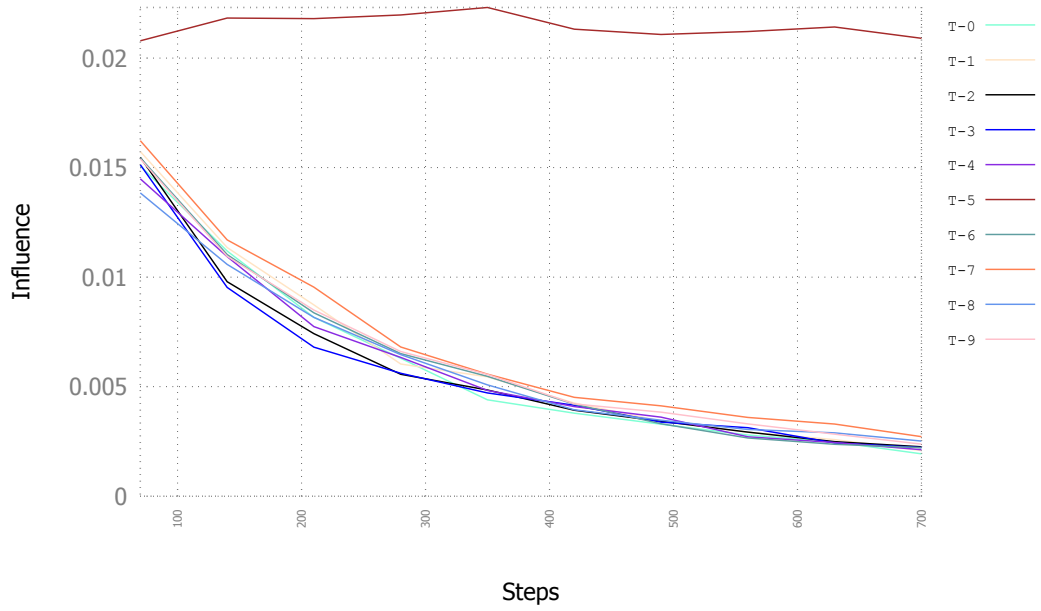
6.3 Summary

Beforehand, the general method of influence detection has been introduced in Chapter 4.1 and extended in Chapter 5. In this chapter, we have build up on this method and enhanced it with a mechanism to detect influences that affect other systems after a period of time. Therefore, we first introduced how the estimation step of the workflow has been adapted to cover such cases in Section 6.1. Afterwards, we showed how the extended method performs on smart factories settings in Section 6.2. We saw that the delayed influences can be properly detected.

In the following, we will explore how influences can be detected at runtime in Chapter 7, and how influences can be addressed by means of reinforcement learning algorithms in Chapter 8.

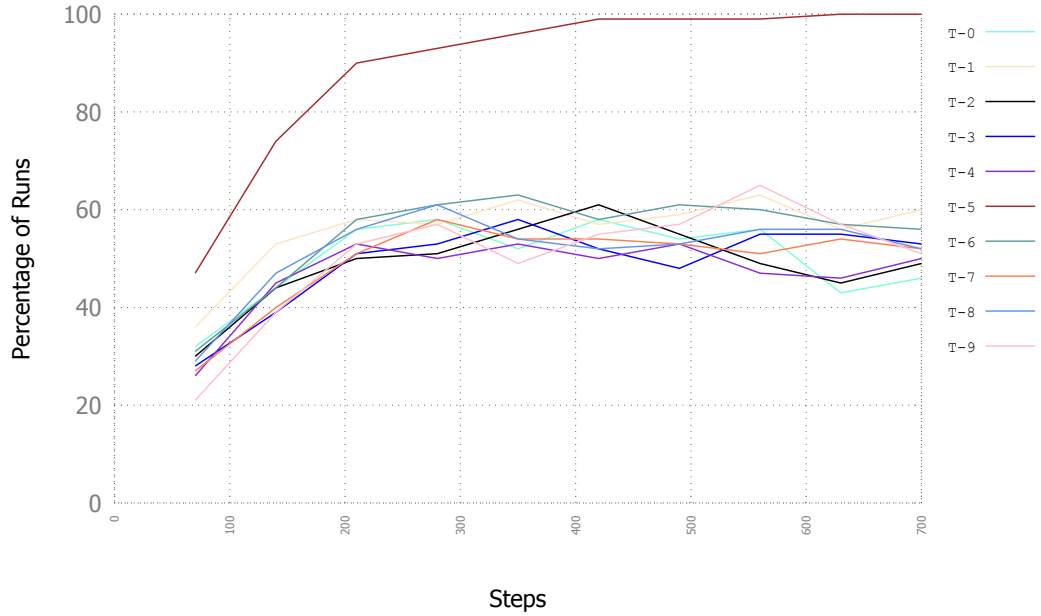


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

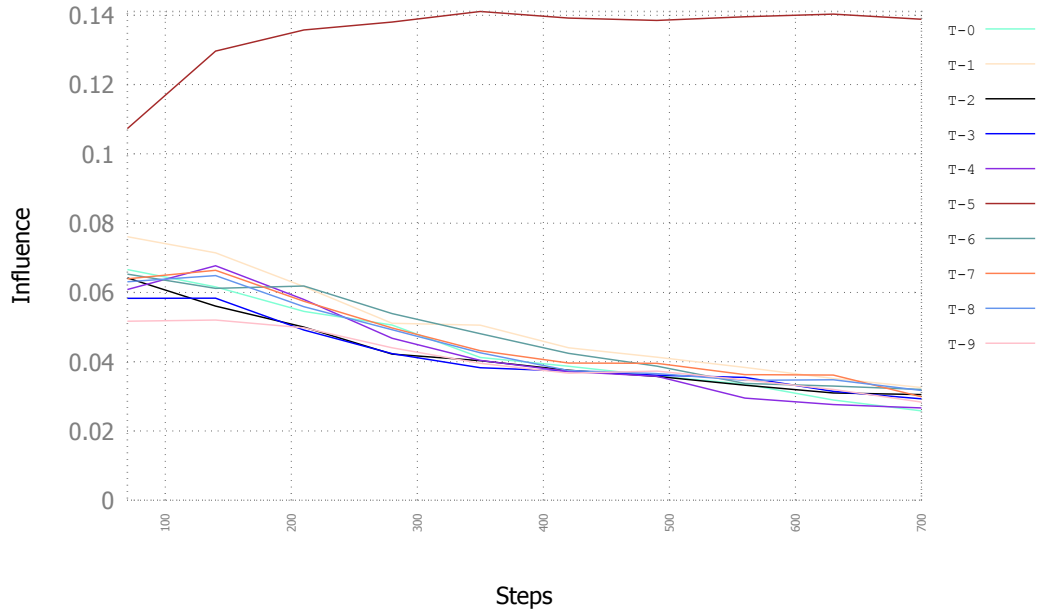


(b) The averaged influence values.

Figure 6.3: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Maximal Information Coefficient**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

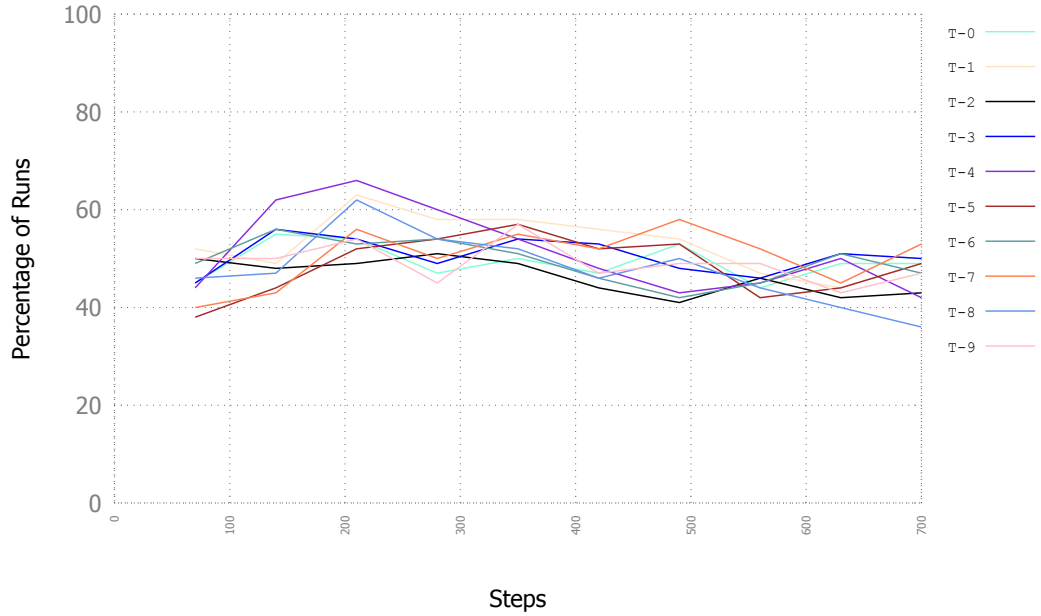


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

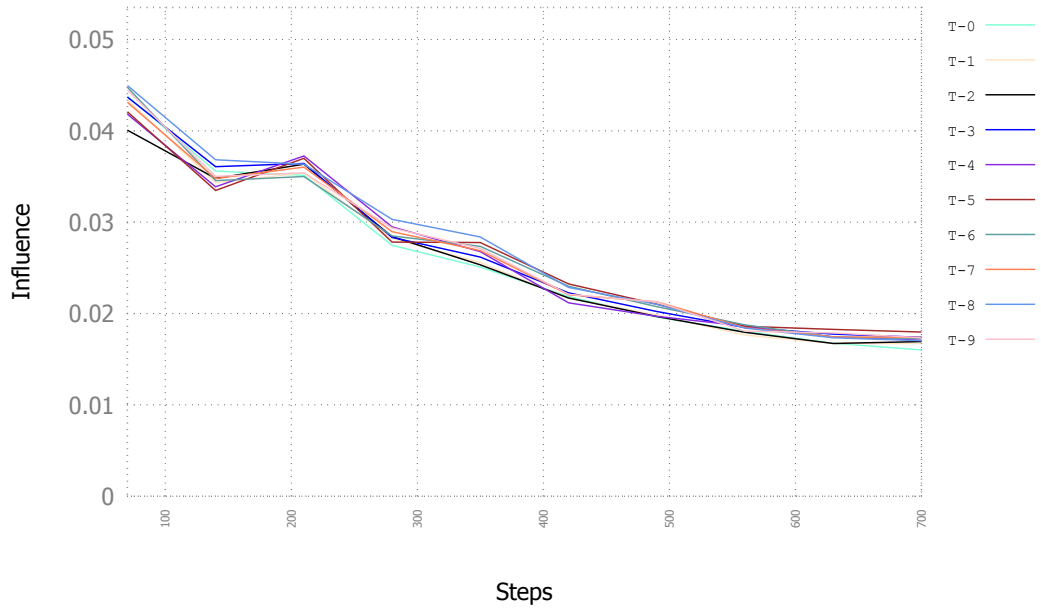


(b) The averaged influence values.

Figure 6.4: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Pearson correlation coefficient**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

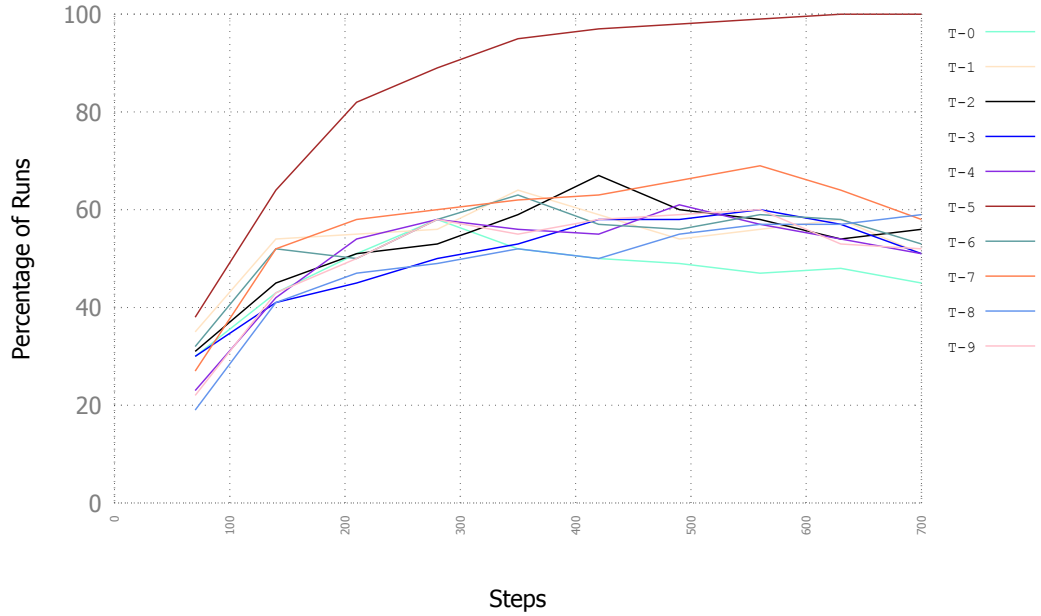


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

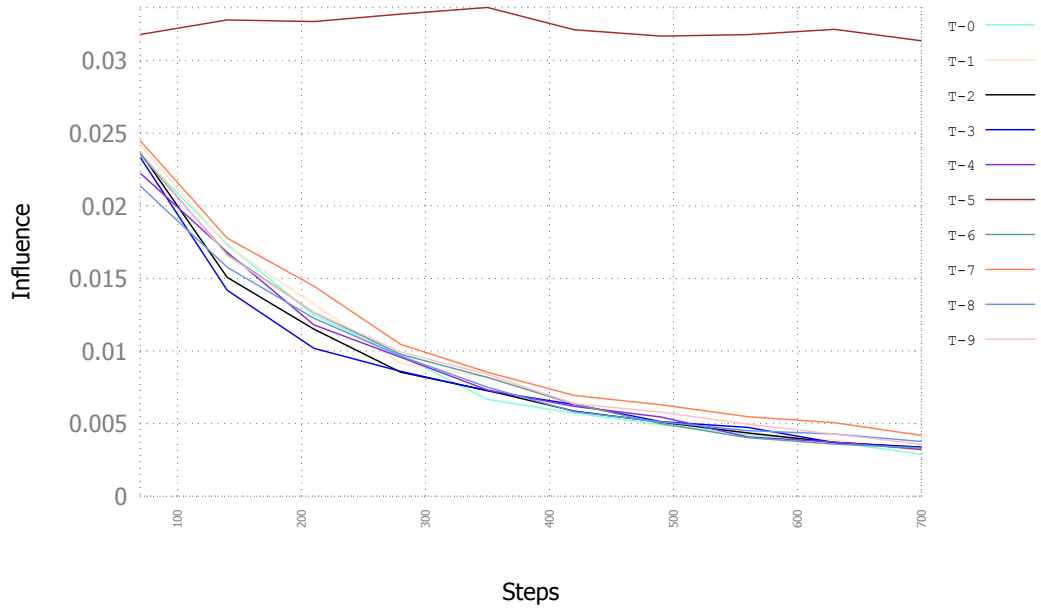


(b) The averaged influence values.

Figure 6.5: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Continuous Mutual Information**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

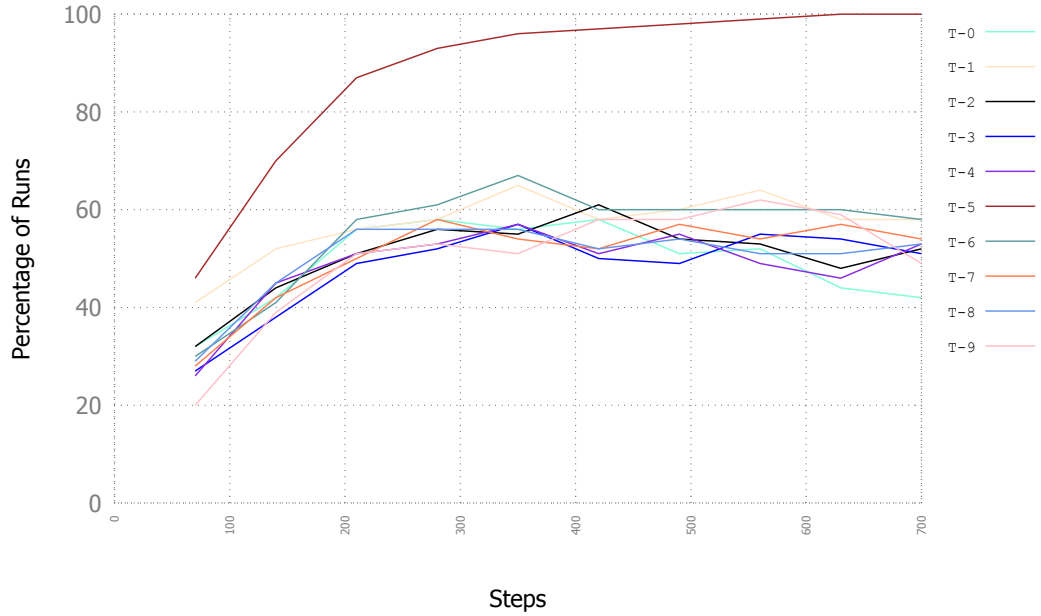


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

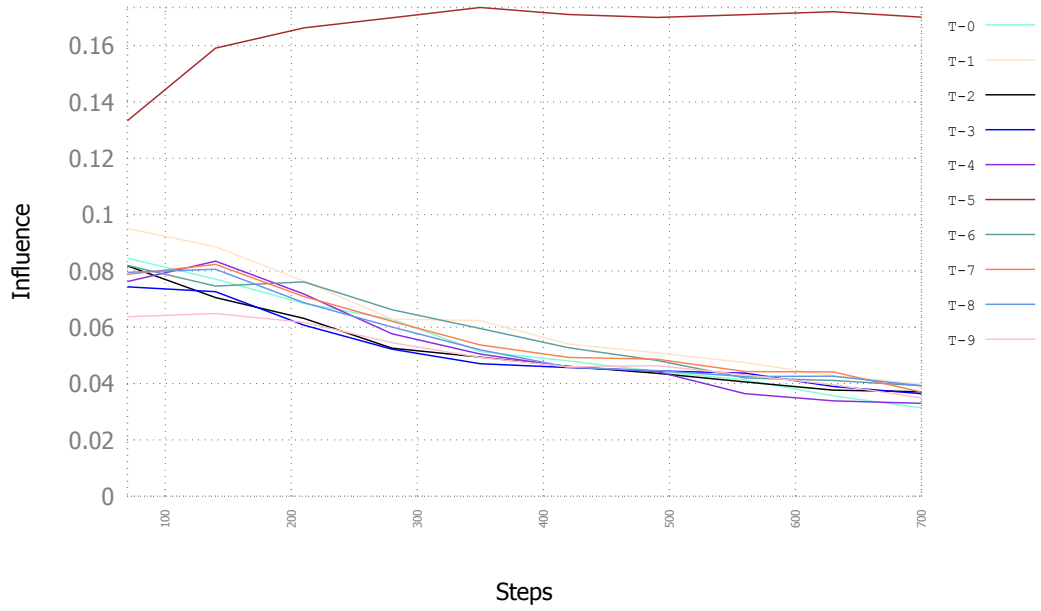


(b) The averaged influence values.

Figure 6.6: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Maximal Information Coefficient**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.

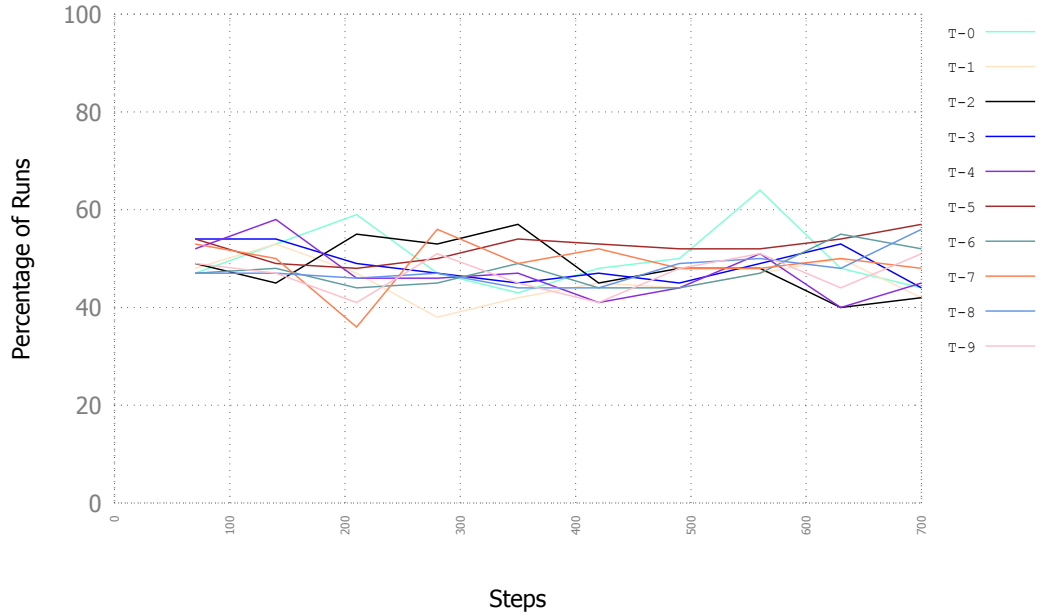


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

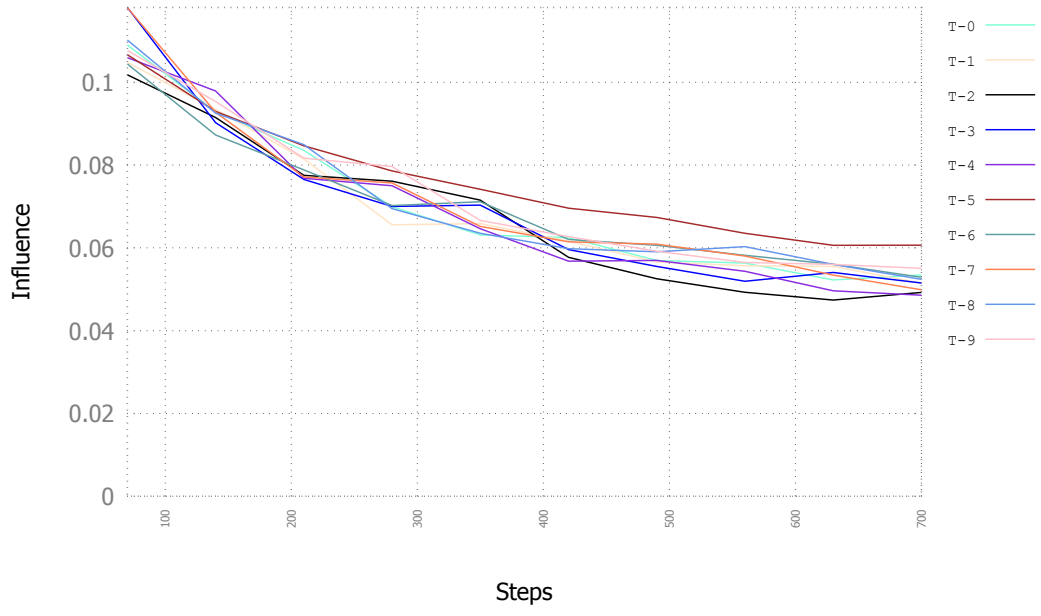


(b) The averaged influence values.

Figure 6.7: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Pearson correlation coefficient**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.



(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.



(b) The averaged influence values.

Figure 6.8: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Continuous Mutual Information**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.

7 Influence Detection at Runtime

Modern ICT systems often cannot be tested entirely since not all circumstances the system faces can be foreseen due to a lack of awareness or time constraints. This applies especially to the type of systems that this thesis focuses on. E.g., if another system is under control of another stakeholder or the systems should be able to adapt to new goals or environmental changes it is especially difficult to foresee all situations a system will face. Therefore, in this section, we look into the detection of influences at runtime. A special examination of this is satisfied because of the problem of static behavior of the systems that introduces disturbances in the measurement by causing correlations that are not based on causality but coincidence.

7.1 Methodology for Detection

Regarding the detection at runtime, we have to consider that the proposed influence detection algorithm relies on dependency measures that estimate the correlation between two random variables. If the method is applied at runtime, this can lead to wrongly detected influences because such correlations can appear without an underlying causality. In the previous experiments, it was possible to infer a causality from the correlation since we enforced the configurations to be randomly selected from independent uniform distributions. However, for the detection at runtime it has to be factored in that we can face autocorrelations and other disturbances within the configurations of the systems.

The approach to avoid this is to rely mostly on randomized configurations that

appear naturally in applications which use reinforcement learning during runtime. This is due to the need of exploration in this tasks to find optimal strategies and avoiding to "get stuck" in a local optimum. This can happen quite often with greedy approaches because the algorithm will only try one behavior and if it is "good" it will stick to it and miss out on other strategies that will lead to better results. There are different strategies to avoid this behavior. An easy and reliable approach is to use a ϵ-greedy action selection [57], i.e., the algorithm will stick to the action that it has evaluated as the best one so far most of the time but with a probability of ϵ it will use a random action regardless of the so far evaluated usefulness of it. This means a natural approach to the issue of correlation without causality is to only use the samples that are formed from such exploration steps. But this would lead to a significant lower amount of samples. To avoid this we will also examine in detail how much "randomness" is necessary to allow the measurement to function properly by conduction experiments with different levels of ϵ. Finding the lowest level of ϵ will ensure the fastest detection of influences without running the risk of to falsify the measurement.

One might be suspicious that a very high level is crucial for a successful detection. However, the correlation mainly appears due to the repetition of specific pattern in both systems in the same frequency. If such patterns are brought out of sync, a correct detection is possible despite the repetition, i.e., a high autocorrelation does not necessarily mean that the measurement is flawed. For instance, two systems A and B each switch back and forth between their two configurations 1 and 2. The system A gets a reward of 0.5 if it applies 1 and 0.8 if it applies 2. If both systems switch between two configurations in each time step the measurement will be falsified because system B's configuration will correlate with the reward of system A. If one of the systems randomly do not switch their configuration the behavior becomes "asynchronous" and the measurement works well.

7.2 Evaluation

In the following, the evaluation of the runtime detection on scenarios from the smart camera domain.

We start on SCN 2 that has been introduced in Chapter 5 and Figure 5.2. Briefly recapped, we have three cameras, where Camera 1 and Camera 2 influence each other because of a overlap in their potential observable space. Camera 3 does not overlap with one of the other cameras, and, therefore, it does not influence the other

Step	Start	$10k$	$20k$	$30k$	$40k$	$50k$	$70k$	$80k$
ϵ	1.0	0.9	0.81	0.73	0.66	0.59	0.53	0.48
Step	$90k$	$100k$	$110k$	$120k$	$130k$	$140k$	$150k$	$160k$
ϵ	0.43	0.39	0.35	0.31	0.28	0.25	0.23	0.20
Step	$170k$	$180k$	$190k$	$200k$	$210k$	$220k$	$230k$	$240k$
ϵ	0.19	0.17	0.15	0.14	0.12	0.11	0.1	0.09
Step	$250k$	$260k$	$270k$	$280k$	$290k$	$300k$	$310k$	$320k$
ϵ	0.08	0.07	0.06	0.06	0.05	0.05	0.05	0.05

Table 7.1: The values for ϵ in the ϵ-greedy action selection. It starts at 1.0 and decreases by 10% every $10k$ steps until it reaches 5%.

two. Previously, we saw how the detection works when the configuration is chosen randomly from the full configuration space.

In the following, the experiments are based on control mechanisms that use Q-learning (cf. Section 2.1.1) and use discretized states and relative actions. Q-learning has been chosen since it is widely used and well understood. For the concrete implementation, we limit the alignment of the camera to 12 pan angles, i.e., it is from $s_p = \{0, 30, 60, \ldots, 300, 330\}$, to 3 tilt angles, i.e., it is from $s_t = \{120, 150, 180\}$, and to 2 zoom levels, i.e., it is from $c_z = \{12, 18\}$. This results in $12 \cdot 3 \cdot 2 = 72$ states each camera has. The camera can increase, decrease or leave the pan, tilt and zoom in each time steps resulting in $3 \cdot 3 \cdot 3 = 27$ actions that can be applied. This means that the configuration space is $C = s_p \times s_t \times s_z$ for each camera and is identical to the state space.

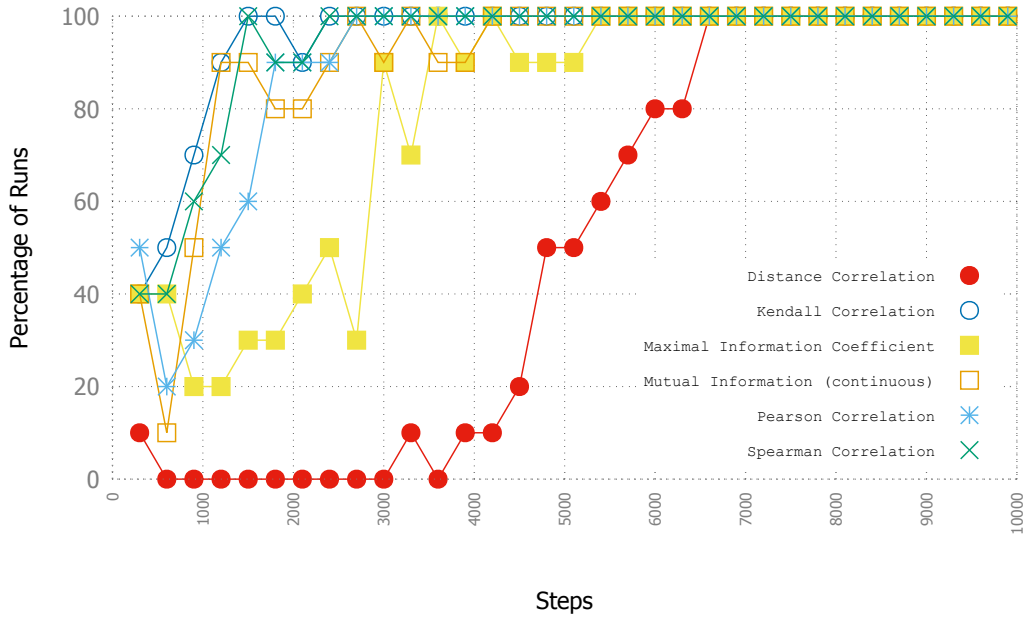
The first evaluation step is to find out how the limitation of the configuration space and the order of states that are visited affect the influence detection. The results can be found in Figure 7.1 and 7.2. There, the system has run for 10000 steps by applying random actions and show the percentage of runs in which Camera 1 detects the Camera 2 as more influencing then Camera 3. The results for the pan and zoom are depicted in Figure 7.1. There, we see that each measure can find the influences very reliable within the 10000 steps with the distance correlation being much more unreliable using less samples. In Figure 7.2, the results for the tilt are depicted. We can see that only few steps needed to find the influence, i.e., in only 600 steps each of the measures can detect very reliable the influence.

As the next step we look at how the detection rate changes if we switch from a purely random action selection to ϵ-greedy. The ϵ will be varied between 1 and 0.05 according to Table 7.1. It reflects that the ϵ is set to 1 at start and is then decreased by 10% each 10000 steps. The results can be found in Figure 7.3 and 7.4. There, we see the detection rate if single-agent Q-learning with an ϵ-greedy action selection is used and the last 10000 samples are used for the detection. The Q-learning algorithm uses a low $\alpha = 0.1$ to handle fluctuations in the reward signal and a high $\gamma = 0.9$ to allow the learning of a sequence of actions. In Figure 7.3a, we see that for the pan the detection works well for the first 50000 steps, i.e., for $\epsilon > 0.66$. Afterwards, the detection works worse for some of the measures with an average of about 80%. The continuous mutual information sticks out here with a perfect detection even with low values of ϵ. For the tilt, in Figure 7.3b, we see a similar effect, but it is way less distinct and shows only for lower ϵ. In Figure 7.4, the graph for the zoom is shown. It shows way more sensitive regarding ϵ and makes the measurement unreliable.

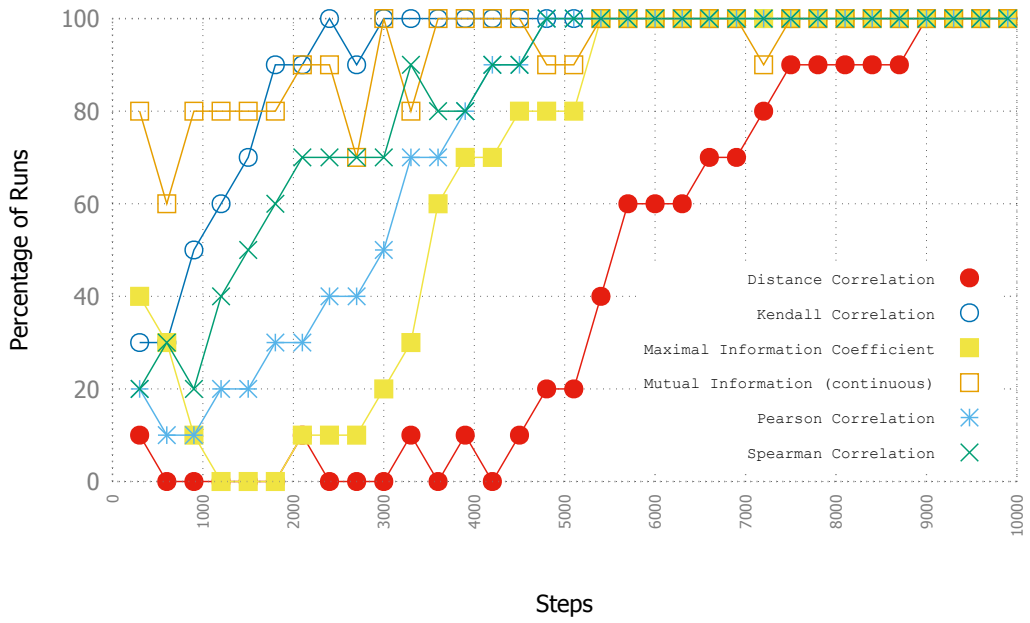
Concluding the results, we have seen how the application of a control algorithm affects the influence detection in contrast to a random selection of configurations. Due to the disturbances and autocorrelations, it is possible that influences are not detected correctly. However, if it is ensured that an adequate amount of "randomness" is used to select the actions it results in proper results. For the example examined here, we have seen that the usage of 66% randomly selected actions allows a precise identification of influencing systems. If the usage of less randomly selected actions is necessary, an amount of greedy actions should be removed from the set for the calculation of influences until the required amount of "randomness" can be ensured.

7.3 Summary

In the previous chapters, we have seen how the influence detection can be extended to cover multi-component influences (cf. Chapter 5) and delayed influences (cf. Chapter 6). In this chapter, we have focused on the detection of influences during the runtime of a system. The main idea is to use the exploration steps in reinforcement learning algorithms to avoid measuring correlations that appear due to the control algorithms rather then influences between the subsystems which has been described in detail in Section 7.1. The approach is then evaluated in the smart camera domain

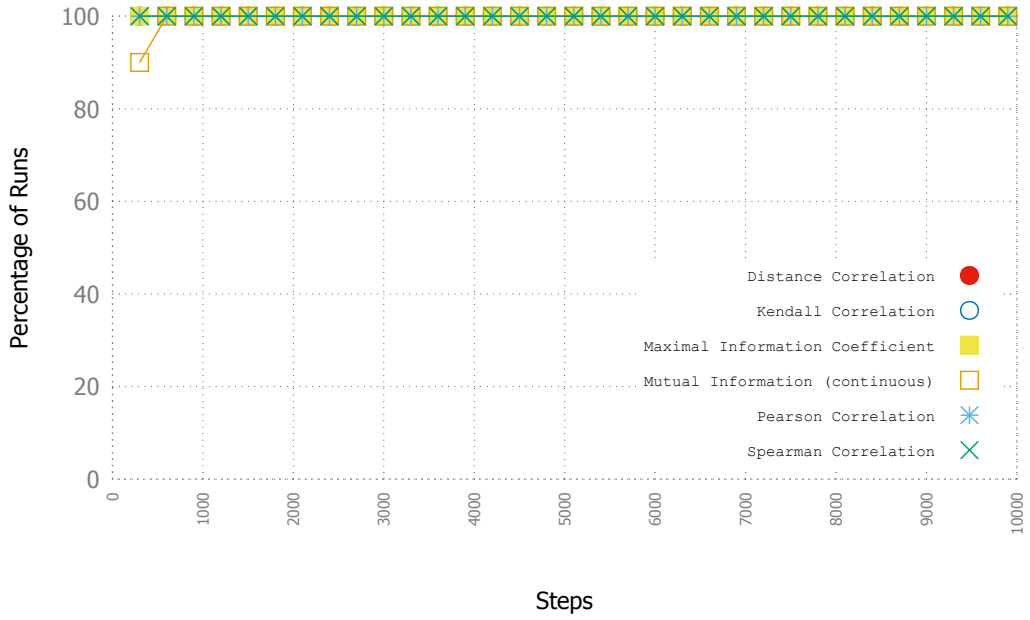


(a) The results for the pan.

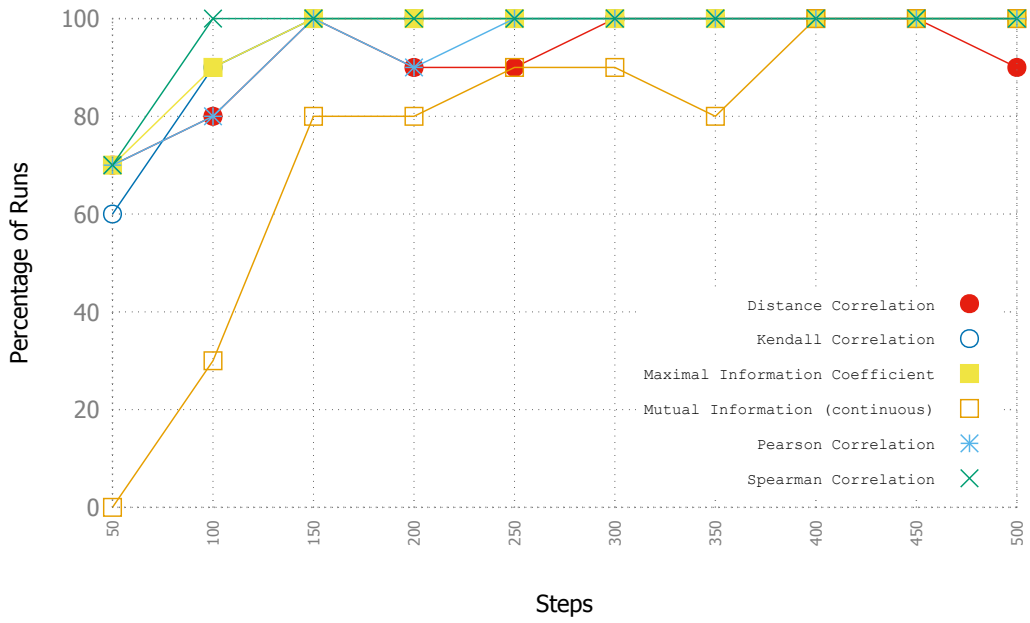


(b) The results for the zoom.

Figure 7.1: The results for the detection of influences in SCN 2. The influence detection rate for the **pan** and **zoom** of Camera 1 versus the Camera 2 based on 10 independent runs. The measurement is similar to the graphs before but with the discretization of the state space and a Q-learning that applies the actions randomly. The graphs show the number of runs, in which the influence of Camera 1 is measured higher than the influence of Camera 2, for different numbers of samples.

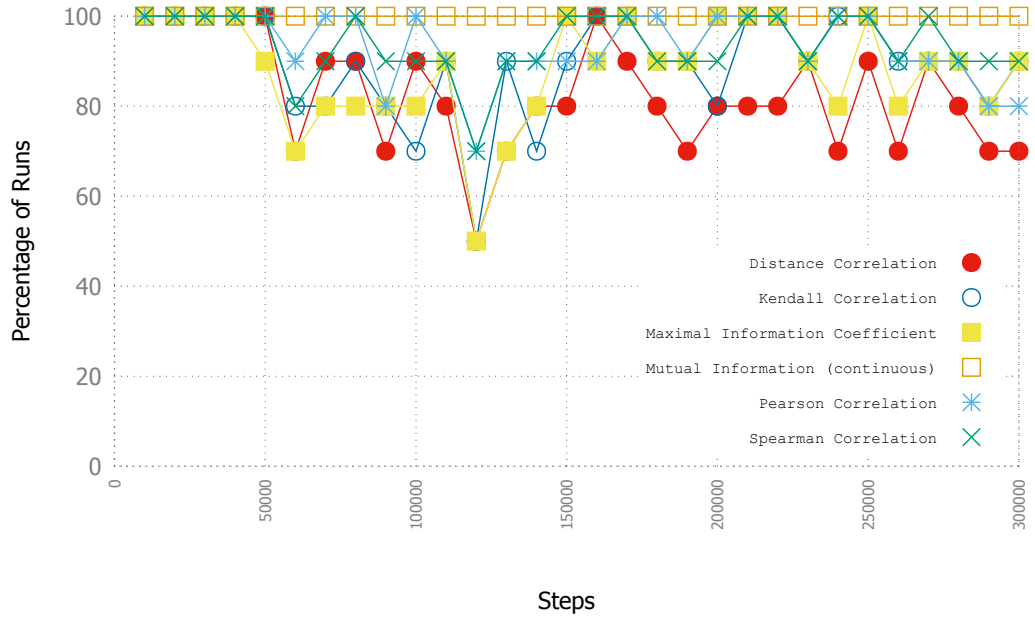


(a) The results for 300-9900 samples.

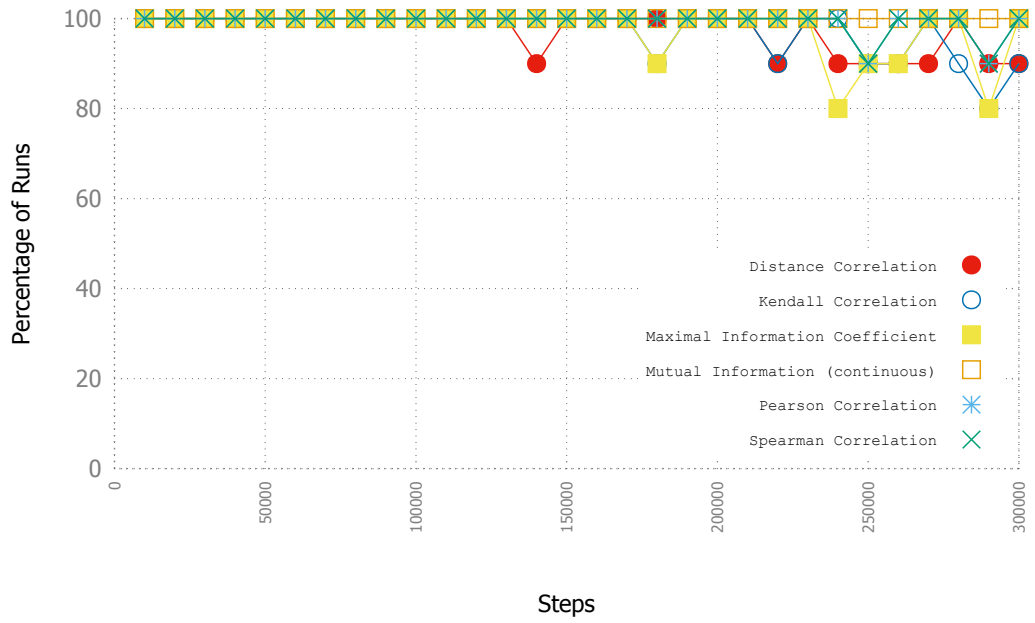


(b) A detailed few for 50-500 samples.

Figure 7.2: The results for the detection of influences in SCN 2. The influence detection rate for the **tilt** of Camera 1 versus the Camera 2 based on 10 independent runs. The measurement is similar to the graphs before but with the discretization of the state space and a Q-learning that applies the actions randomly. The graphs show the number of runs, in which the influence of Camera 1 is measured higher than the influence of Camera 2, for different numbers of samples.



(a) The results for the pan.



(b) The results for the tilt.

Figure 7.3: The results for the detection of influences in SCN 2. The influence detection rate for the **pan** and **tilt** of Camera 1 versus the Camera 2 based on 10 independent runs. The measurement is similar to the graphs before but here the camera is controlled by a Q-learning algorithm with ϵ-greedy strategy, where the ϵ is falling from 1 to 0.05 in 10%-declines each $10k$ steps. The graphs show the number of runs, in which the influence of Camera 1 is measured higher than the influence of Camera 2, using the last $10k$ steps as samples.

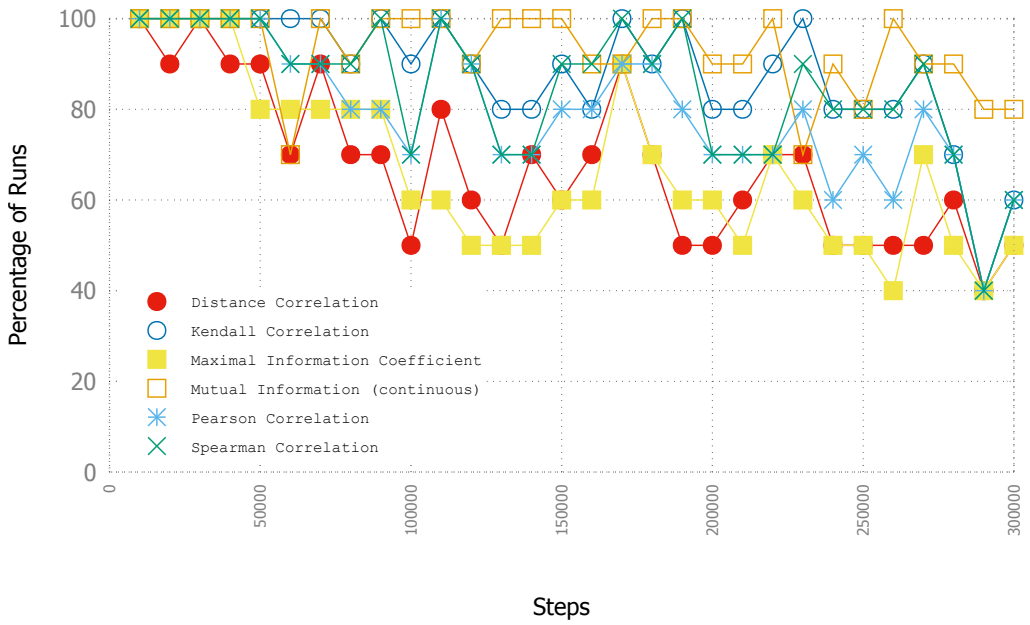


Figure 7.4: The results for the detection of influences in SCN 2. The influence detection rate for the **zoom** of Camera 1 versus the Camera 2 based on 10 independent runs. The measurement is similar to the graphs before but here the camera is controlled by a Q-learning algorithm with ϵ-greedy strategy, where the ϵ is falling from 1 to 0.05 in 10%-declines each $10k$ steps. The graphs show the number of runs, in which the influence of Camera 1 is measured higher than the influence of Camera 2, using the last $10k$ steps as samples.

in Section 7.2. There, we see for which exploration rates we can expect useful results from the measurement.

In the following, in Chapter 8, we will see how the evaluation and adaption step can be realized by means of reinforcement learning algorithms.

8 | Self-adapting to Influences

In the previous chapters, we focused on the detection on the influences, which is a crucial to step to the goal of optimal behavior. In this chapter, we introduce an approach to self-adaption to the influences, i.e., the exploitation of the influences if they have been discovered. To achieve this, we focus on the evaluation step and the adaption step of the workflow. The focus is not on the presentation of a single algorithm, but giving a general methodology based on the basic RL model and realize it with the prominent Q-learning.

8.1 Methodology for Self-adaption

In general there is a variety of options to address the influences since the detection is designed to work independently from the control algorithm of the system. Possible reaction to influences go from a hand-crafted solution during design time over a system that adapts during runtime in static patterns to a full self-learning behavior. Potential candidates for such a learning algorithm can be found in in the area of MARL. Depending on the problem, for example, distributed W-learning [109] can be useful. However, in this section, we rely on a basic principle that unifies three properties:

1. it is possible to start out with single learners and later on learn the cooperation,
2. it can be applied to every reinforcement learning algorithm,
3. it can be used with systems that have heterogeneous configuration spaces and control algorithms, e.g., some of the systems have a static behavior.

The basic adaption principle is depicted in Figure 8.1. There, we adopt the basic reinforcement learning model introduced in Section 2.1.1. We assume that previous

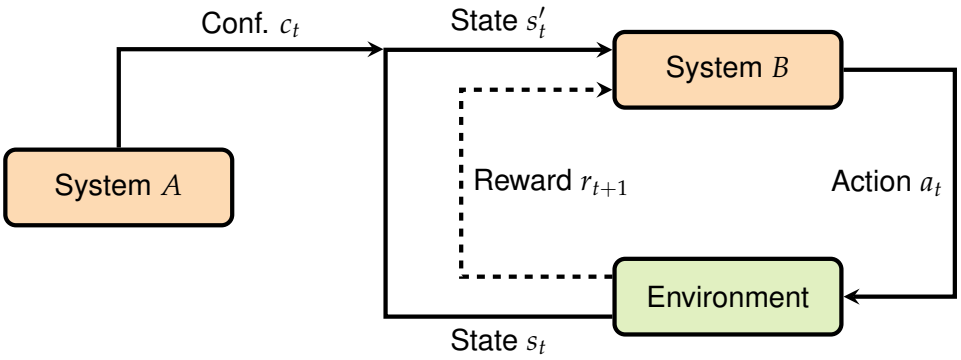


Figure 8.1: The proposed method to adapt to other influencing agents in terms of the reinforcement learning model.

measurement has resulted in a detection of an influence from system A on system B meaning B should react properly regarding the configuration of A. Therefore, we integrate the value of the configuration c_t at time t into the state s_t resulting in a state s'_t. For example, if B's states are given through a position represented as (x, y)-coordinate between 0 and 1 each, its state space is $S = [0,1] \times [0,1]$. Assuming the influence detection has found A's configuration component c_B, which represents the speed of B and is between -1 and 1, as influencing, the state space S will be extended to a new state space $S' = S \times [-1,1] = [0,1] \times [0,1] \times [-1,1]$.

There are two points to discuss in this approach. The first one is how the influenced system B will know the current configuration of A. This is not an issue with delayed influences but only with immediate. It can be resolved by letting the influenced system wait until it can observe the decision either by a message of the influencing system or through sensors. If several influences in the system are detected this can lead to chains of systems that wait on each other and it is necessary to check for each of the integrations in the state space that the graph they form is acyclic.

As mentioned before this method is in principle independent from the learning algorithm as long it fits the RL model. But the extension of the state space can be seen as a special instance of transfer learning in the RL domain [110]. This makes algorithms that have a strong transfer capability more suitable for this task. In this thesis, we rely on Q-learning since it gives an easy and natural way to realize the transfer during runtime that will be explained in the following: Assuming that c_B can take k different values from each state s, there will be k new states s'_i, $i \in \{1, \ldots, k\}$.

Before the expansion of the state space, the system holds a current Q-value for $Q(s, a_j)$, where $a_j \in A$ are the possible actions. To transfer the already learned knowledge, the new Q-values are set to $Q(s'_i, a_j) := Q(s, a_j)$, for each $i \in \{1, \ldots, k\}$ and $a_j \in A$. The advantage here is that in states that are nor affected by the other system the Q-value will remain at the correct value and in states where the system is influenced the values will be update to their true value using the upcoming experiences.

If the state space is continuous the standard Q-learning can not be used. An alternative would be for example the extended classifier system (XCS-R) with continuous-valued states [111, 112] that has good online learning capabilities and is transferable [113].

A further issue is that in the previous experiments about runtime detection, we have seen that a small amount of runs might detect the independent systems as more influencing then they actual are. This behavior is temporary and can be corrected by using a bigger sample size. However, during the runtime of the system it is not possible to determine if it is a malicious detection. Therefore, we introduce a factor by that the real measurement has to be higher than the calculated notional independent counterpart which is used as baseline. Furthermore, the different configuration parts are ranked and after each influence calculation only highest ranked configuration is integrated in the state space of each camera. Even though it is very unlikely this cannot prevent erroneous integrations entirely but, if a wrong integration is detected later, it can be reverted easily.

8.2 Evaluation

In the next part, we examine how the influence measurement does react to the adaption that has been introduced previously. To do so, we integrated the three configuration components of Camera 2 in the state of Camera 1 at the start of the simulations. In Figure 8.2 and 8.3, we see similar results as for the independent learning with especially bad results for the maximal information coefficient and the distance correlation.

As a last comparison on this scenario, we have a look at the rewards received with the different method: the independent learners in Figure 8.4a, the integration of the configuration of Camera 2 in the state of Camera 1 in Figure 8.4b, and the

dynamic integration of the configurations at runtime in Figure 8.5. We can see that the independent learners are not able to reach an expectable result since they lack a coordination mechanism. The integration at start can reach an optimal result of about 25 after $250k$ steps. The dynamic expand can reach a similar result but learns slower. This is because the transfer mechanism cannot play out its advantages here since the Q-values differ for all states in this example.

In Figure 8.6, we see scenario SCN 3, another scenario from the smart camera domain. The figure shows a top-down view on a smart camera network with six cameras depicted as black dots. Each of the cameras has a corresponding partial circle around it that visualizes the space potential observable by the cameras. The circle is only partial since the vision is blocked by pillars that are shown as gray squares. They are several streams of targets entering and crossing the scene from all sites marked by yellow arrows. We see that not each pair of cameras share a common area both can observe. Therefore, only the pairs $(1,2)$, $(2,4)$, $(3,4)$, $(3,5)$, $(5,6)$ influence each other and have to consider the configuration of the other camera.

Again, we apply the Q-learning algorithm to this scenario but with a small adaption. This time, we do not use relative action, e.g., increase or decrease, but absolute ones, i.e., each of the defined states can be accessed from every other state leading to 72 actions. Since this does not require to learn sequences of action to reach one alignment from the previous the discount factor is set to $\gamma = 0$.

The results are shown in Figure 8.7 and 8.8. The single-agent learning case in Figure 8.7 is clearly worse than the other variants only reaching a reward of 2 on average since the cameras do not coordinate well. We see that the algorithms gets stuck in local optima when ϵ drops low at around $250k$ steps since single configurations are valued the highest due to the lack of knowledge of the influencing systems. The other to variants show the dynamic integration in the state space. In Figure 8.8a, we see how the system performs when 3000 samples are used for the detection; in Figure 8.8b, 10000 samples are used. The graphs are nearly identical and both reach an optimal behavior that gives an average reward of above 3.

Finally, in Figure 8.9, we see the cooperation network that evolved after 200000 steps in an exemplary run. Each arrow means that the configuration parts of the camera that is pointed at are integrated in the state space of the first camera. We see that the structure that formed reflects the structure of influence described before.

Concluding the results, we have seen how single learner compete with a priori

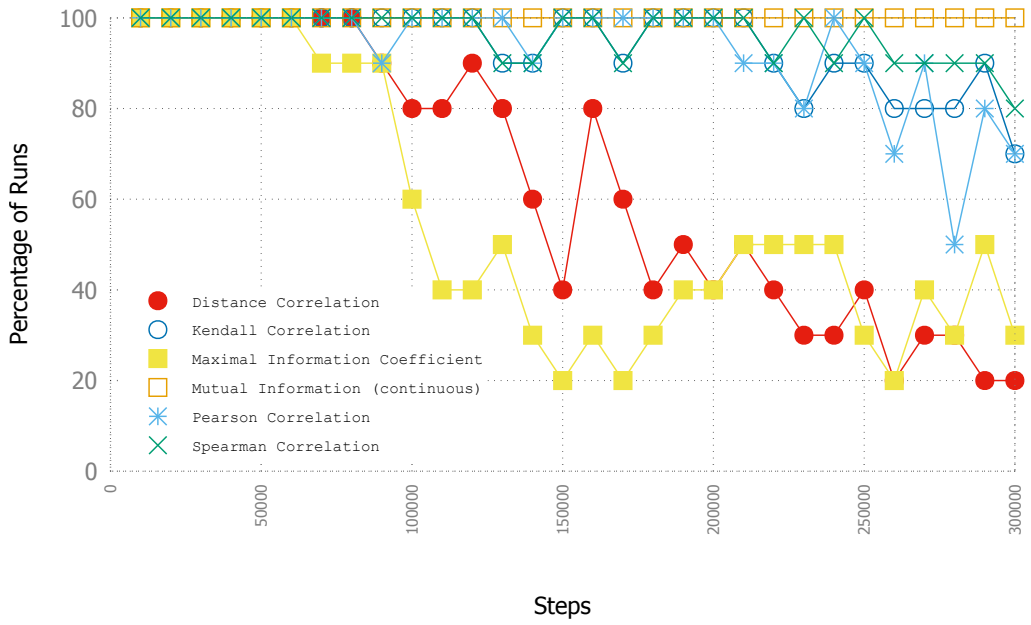
adapted systems and a stagewise integration of influencing configurations in the state space of a RL algorithm. We have seen that the single learner can not compete on the same level as influence respecting variants. The influence detecting approaches have shown similar results, which makes them useful options for the design of systems.

8.3 Summary

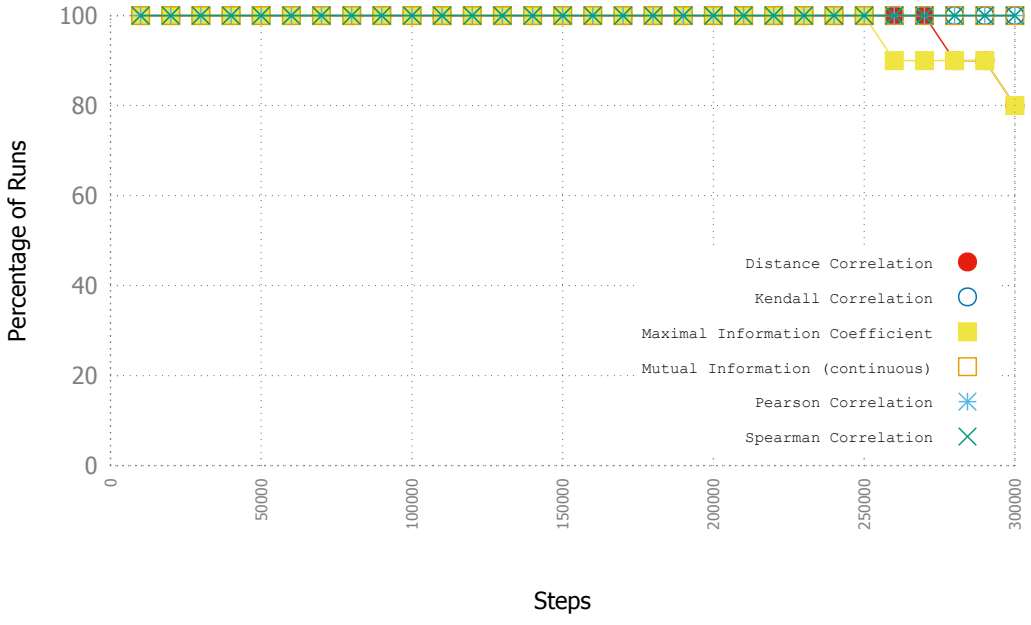
In the previous chapters, we introduced the general influence detection (cf. Chapter 4.1), and extensions of it to make make it applicable to a wider range of cases by refining on the estimation step of the general workflow. We explored multi-component influences (cf. Chapter 5), delayed influences (cf. Chapter 6), and the influence detection at runtime (cf. Chapter 7).

In this chapter, the main focus is on the evaluation and the adaption step of the general workflow. Especially, a method for the self-adaption of subsystems to influences from other subsystems has been introduced. The main idea is to adapt the the classical single learner reinforcement learning model by adding the configurations of influencing agents to the state space during the runtime of the system (cf. Section 8.1). The adaption capability of the systems has been demonstrated in the smart camera domain (cf. Section 8.2).

In the following, practical advice regarding the influence detection are given in Chapter 9 and the thesis is concluded in Chapter 10.



(a) The results for the pan.



(b) The results for the tilt.

Figure 8.2: The results for the detection of influences in SCN 2. The influence detection rate for the **pan** and **tilt** of Camera 1 versus the Camera 2 based on 10 independent runs. The cameras are controlled with the same algorithms as before but the pan, tilt, and zoom configuration of Camera 2 are integrated in the state space of Camera 1 at the start of the runs. The graphs show the number of runs, in which the influence of Camera 1 is measured higher than the influence of Camera 2, using the last $10k$ steps as samples.

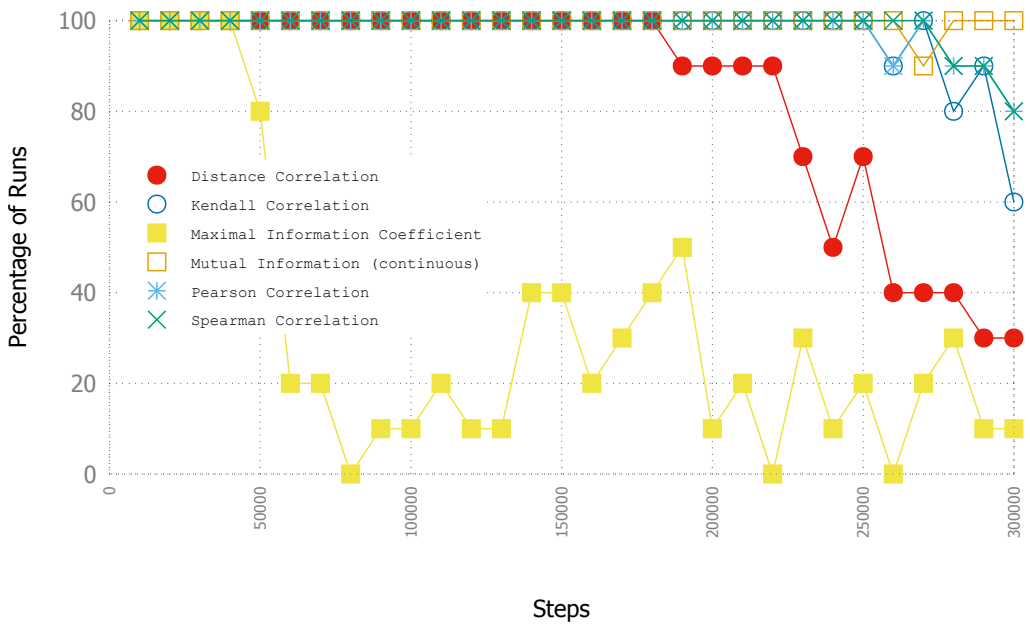
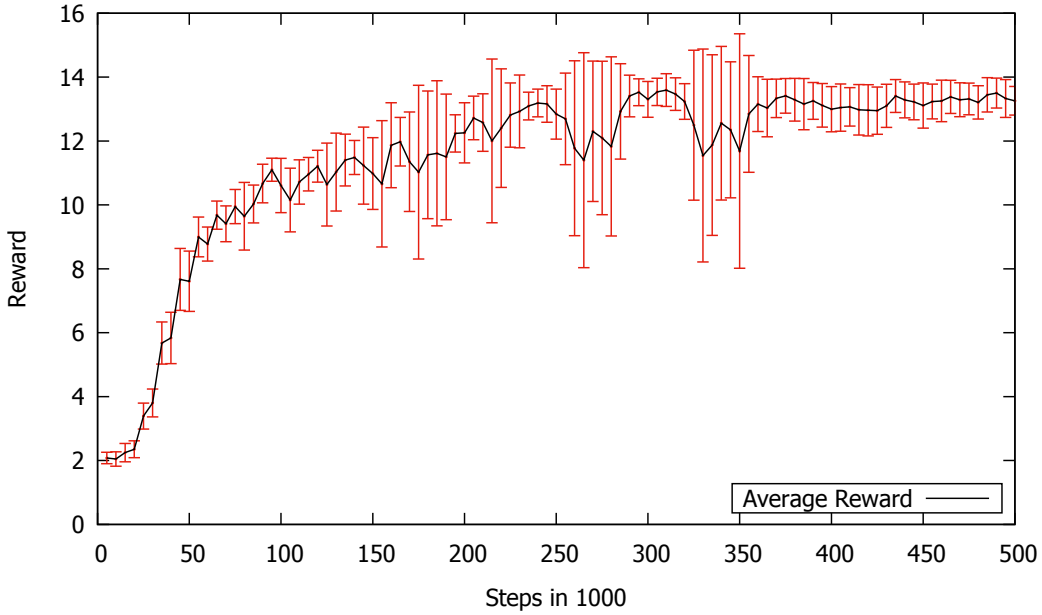
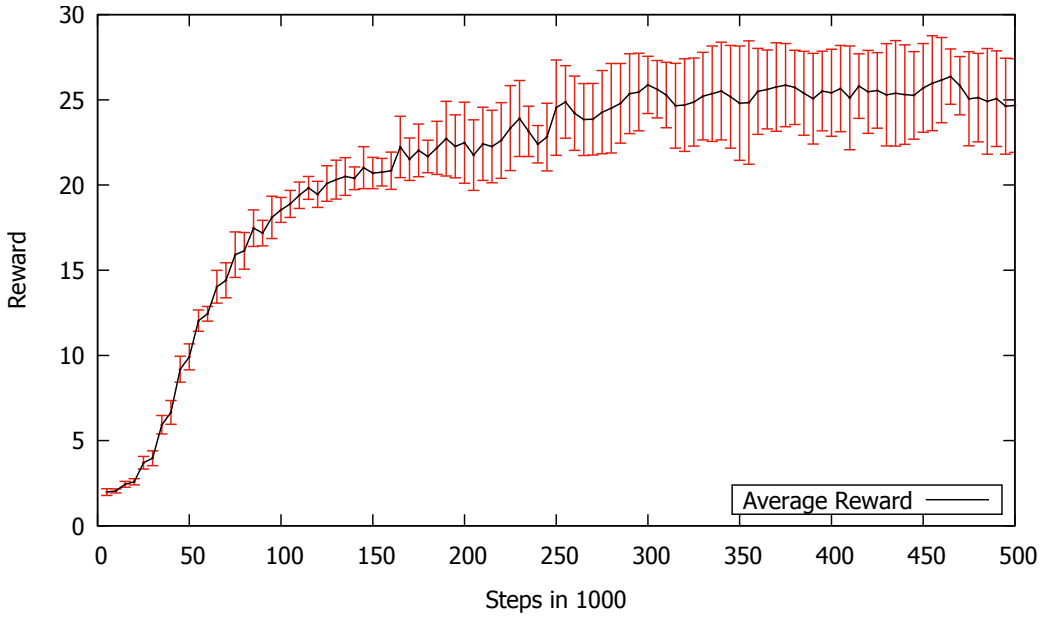


Figure 8.3: The results for the detection of influences in SCN 2. The influence detection rate for the **zoom** of Camera 1 versus the Camera 2 based on 10 independent runs. The cameras are controlled with the same algorithms as before but the pan, tilt, and zoom configuration of Camera 2 are integrated in the state space of Camera 1 at the start of the runs. The graphs show the number of runs, in which the influence of Camera 1 is measured higher than the influence of Camera 2, using the last $10k$ steps as samples.



(a) The results with single learners.



(b) The results if a the pan, tilt, and zoom configurations of Camera 2 are integrated in the state space of Camera 1.

Figure 8.4: The figure shows the rewards received by the system if the Q-learning algorithm is applied to SCN 2 with single learners and with an integration at the start.

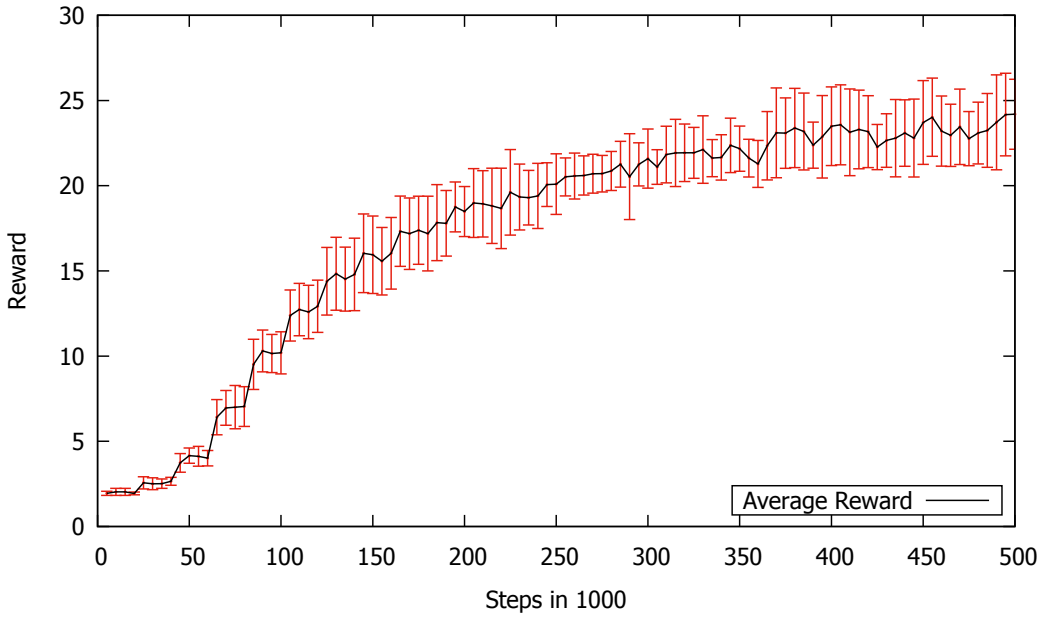


Figure 8.5: The figure shows the rewards received by the system if the Q-learning algorithm is applied to SCN 2 and the configurations are integrated dynamically during runtime.

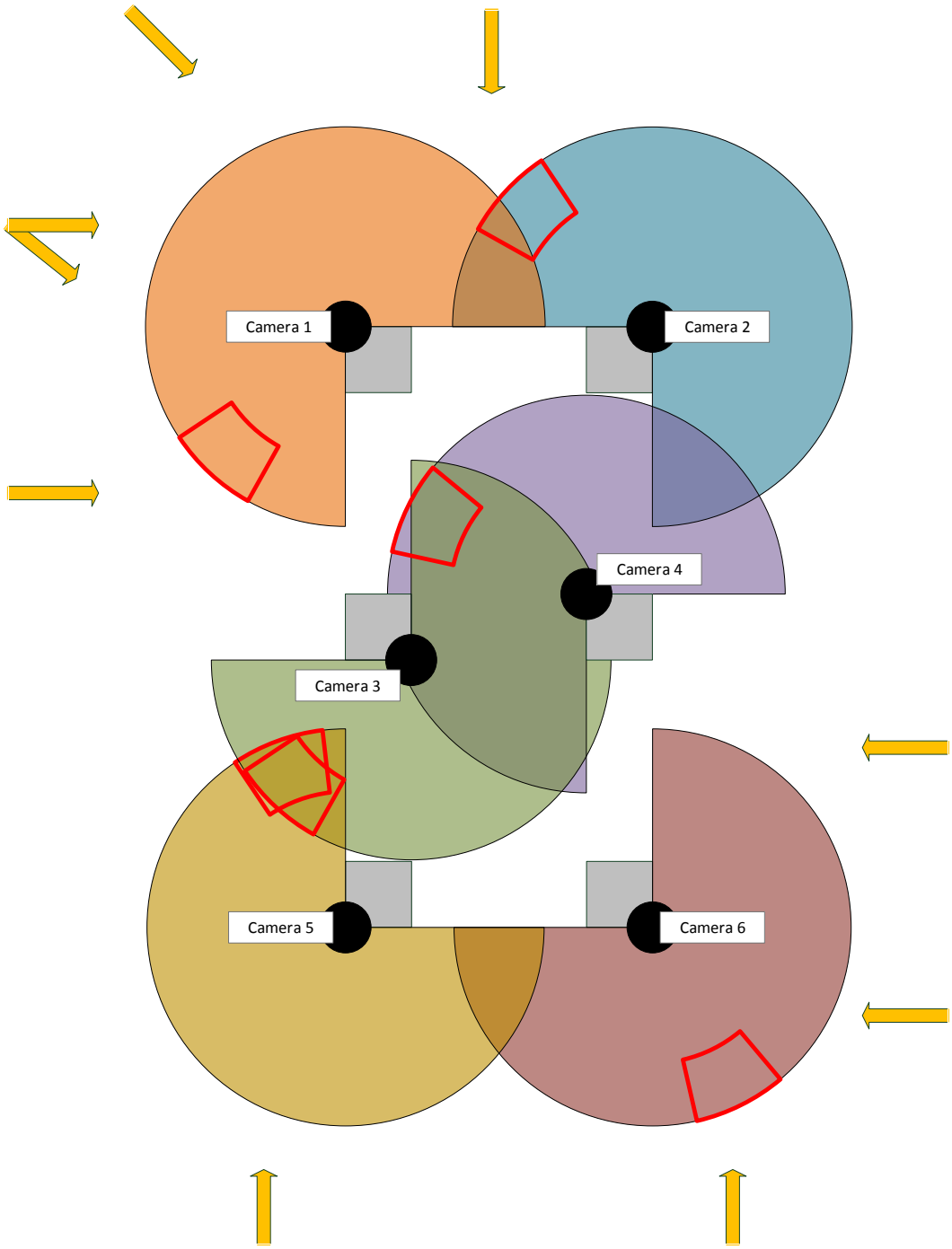


Figure 8.6: *SCN 3*. A top-down view on a larger smart camera network with 6 cameras. The position of the cameras are depicted by the black dots. Each of them has a colored circle around it marking the area that can be potentially observed by it and a red shape that shows an exemplary observed area. The gray squares show the position of pillars limiting the potentially observable areas. The yellow arrows mark entry points of observation targets and there movement direction.

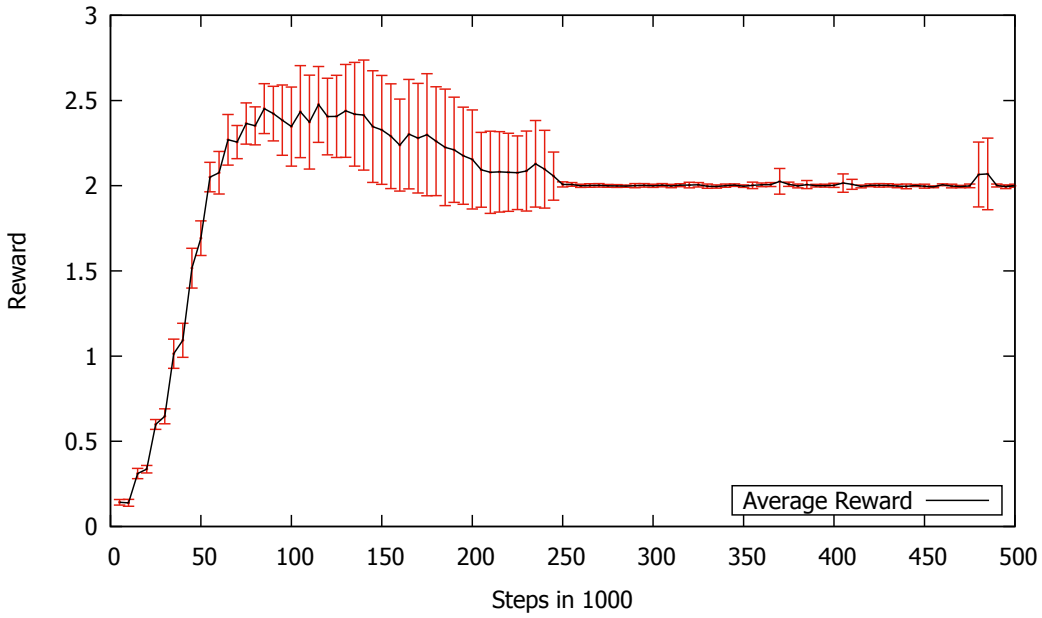
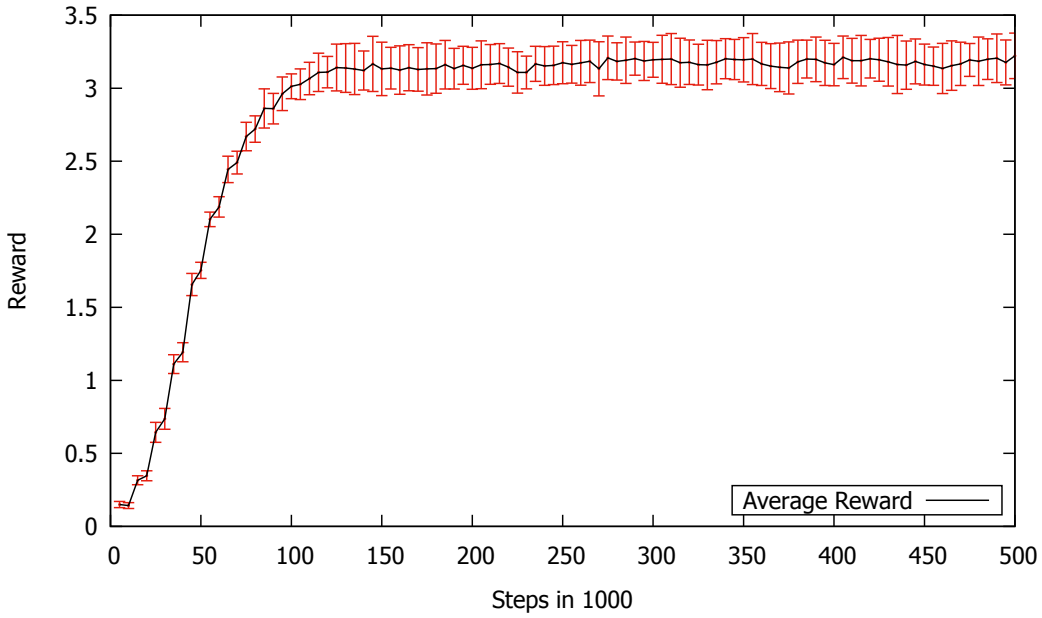
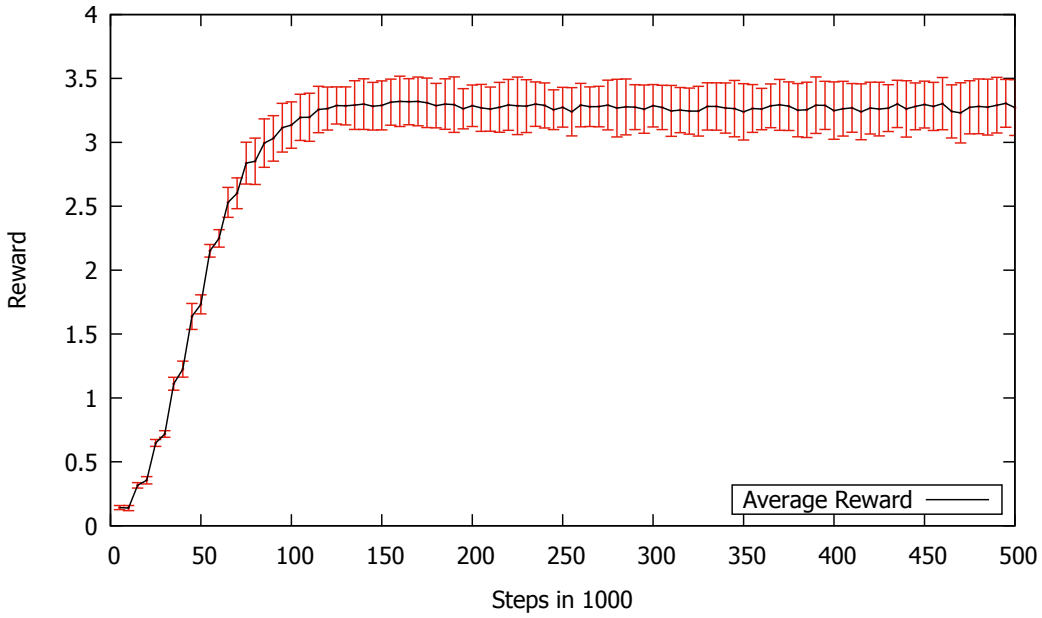


Figure 8.7: The figure shows the average reward over 10 runs in the scenario SCN 3. The cameras are each controlled by a Q-learning algorithm with a ϵ-greedy action selection, where the ϵ decreases. It is restricted to single agent learning.



(a) The results if $3k$ samples are used to determine the candidates for an integration.



(b) The results if $10k$ samples are used to determine the candidates for an integration.

Figure 8.8: The figure shows the average reward over 10 runs in the scenario SCN 3. The cameras are each controlled by a Q-learning algorithm with a ϵ-greedy action selection, where the ϵ decreases. It uses the integration of states at runtime.

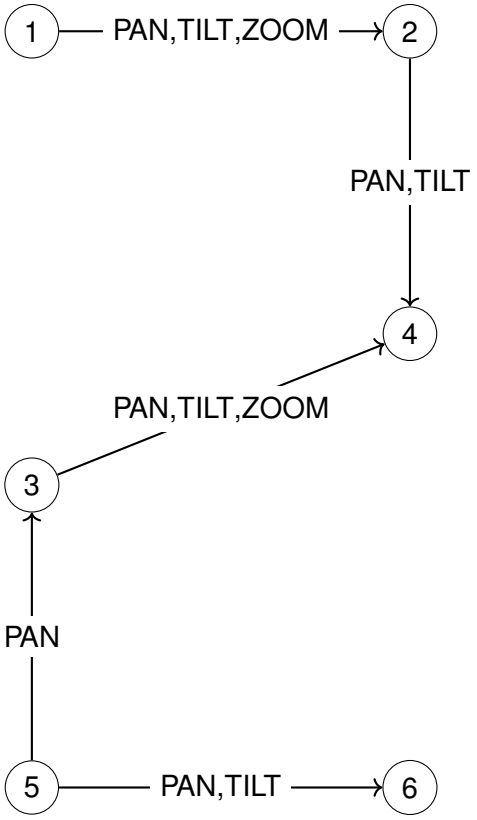


Figure 8.9: A snapshot of the structure of the learning network in an exemplary run on SCN 3 after 200000 steps

9 | Practical Considerations

In the previous chapters, we have seen several techniques to find influences among subsystems: the general approach, the consideration of other configuration components, the handling of delayed influences, the detection at runtime, as well as an approach to self-adapt to influences.

In this chapter, we outline a practical aspects that will allow for a more efficient and effective usage of the influence detection in real-world systems. This includes borderline cases where the detection can potentially be expensive regarding the communication or computational requirements where advice are given on how to avoid such situations as good as possible.

9.1 Choosing appropriate Dependency Measures

In this thesis, we introduced several dependency measures for random variables and compared their usefulness in different scenarios. For the final experiments with the runtime adaption, we used the MIC but, as stated before, the results would have been equally with other measures. In general, however, the choice of the measure can be crucial to a successful detection. There are two main criteria. The first is the type of the configuration component: nominal, ordinal, or infinite real-valued. For nominal it can be useful to avoid measures that rely on the distance between the categories. The only measure guaranteeing this is the discrete mutual information, but this does not mean that the other measures are useless. For the ordinal values, we have seen a wide range of useful measures. The big exception here was the continuous mutual information that cannot handle the discrete values while other measures for continuous values, such as the maximal information coefficient, did not show weaknesses in this area. Finally, for the infinite real-valued configurations, we can use any measure except the discrete mutual information.

The second criteria is what we previously called the strength. The set of measures that can find stochastic dependencies, i.e., all dependencies that can appear, are the discrete and continuous mutual information, the maximal information coefficient, and the distance correlation. Two measures are able to find only monotone dependencies: the Kendal rank correlation and the Spearman rank correlation. And finally, the Pearson correlation can measure only linear correlations reliably.

If used in a productive system it can be useful to consider the usage of multiple measures. This can basically be achieved by two means: either the measures can be run in parallel and afterwards combined by a applying a threshold which can reduce malicious detections, or they can be applied in stages, i.e., first a computational light-weight algorithm is used and in case it cannot find any associations a more powerful measure can be applied. An advantage of the second approach is the potentially reduced resource requirement.

9.2 Sequential Analysis of Systems

In the conducted experiments, each system measures the influence of each other system on itself. This can lead to a high system load and possibly to a backlog in the detection of influences, especially in large-scale systems.

To avoid such situations, it is possible to analyze each other system in a sequential order instead of simultaneously (possibly with multiple measures), i.e., a system creates an order and starts analyzing the influence from the first system in the sequence. As soon as it is done analyzing the first system it can switch to the next and so on. This order can be randomly or based on the physical or virtual distance, for instance. This leads to less computational load.

9.3 Reduction of Network Load

In the previous experiments, we assumed that the influence origins from an arbitrary node in the overall system, i.e., each of the systems distributes its configuration via network to the other systems. Depending on the communication capabilities and the origin of influences in the systems this can lead to a high network load affecting the transmissions necessary to operate the systems, e.g., the results in a wireless sensor network cannot be forwarded to the sink.

Similar to the reduction of computational resources this can be avoided by

analyzing the systems sequential. Another effective way that can be used for some system types is to send the reward to the other systems instead of the configuration. This switches the places where the influences are calculated, i.e., instead of system A receiving the configurations of system B and calculating the influence of system B on itself, it sends the rewards of itself to system B that in turn calculates the influence. Since the configuration will most often create more traffic then just the reward this can reduce the computational overhead but this approach is limited to systems which are trusted and are willing to share their resources.

9.4 Employ more Accurate Conditioning

In Chapter 5, we have seen how the other configurations can be incorporated in the influence detection. The method relies on conditioning the dependency measure by them, but a very accurate computation will make the sample size for each of the parts small. Therefore, in the experiments, only two parts have been used to keep the maximal sample size. However, it is possible that two parts do not reflect the actual influences or it takes very long to find them. In this case it can be reasonable to use more parts. If conditioned by a discrete configuration, it would be possible to use several parts up to the number of configurations in this component. For continuous configuration components the number is theoretically unlimited but the sample size should not be too small. In case no influence has been detected for two parts the number of parts can be increased gradually but each part should hold an adequate number of samples. For the smart camera networks, we saw that each part can hold as few as around 100 samples to ensure useful results. This number is deducted from the last experiment where 3000 samples have been used with 32 parts meaning on average every part holds $3000/32 = 93.75$ samples. However, the actual number is depended on the application.

9.5 Handling Temporal Influences

In this thesis, the experiments are conducted under the assumption of static influences, i.e., the influences are there from the start of the system and do not change or vanish. In real-world systems other time behaviors are possible. Regarding the smart cameras, this could be a partial failure on one camera, which consequently cannot use the whole pan angle potential and does not longer share an overlap with another camera. In terms of machine learning one could talk about concept drift or

concept shift. To catch such changes, it is necessary to continuously reanalyze the influences using a sliding window of samples. For the experiments in Chapter 8, we already used such a technique and compared the results for a sliding window of size 3000 and 10000 and have seen that the results do not differ much. Starting with a small size for the window and increasing it gradually in case no results are found can ensure to find an optimal window size.

9.6 Handling Influences from Multiple Systems

In the conducted experiments, we focused on influences that have their origin at one system and influences another. This has been extended to include the configuration of the influenced system itself in Section 5. In real-world applications, it can be necessary to find influences that only appear if two (or more) systems configure in a specific way, i.e., joint influences, and cannot be detected by the basic method. This happens due to the same reasons we considered the remaining configuration components of the influencing system and the configuration components of the influenced system. Hence, this can be addressed by conditioning the measurement by the configuration components of the second system.

10 | Conclusion

In this last chapter, the thesis is summarized and discussed. Additionally, an outlook on future research directions is given.

10.1 Summary and Discussion

In this thesis, we started out by reviewing several current and looming trends in the domain of OC and related fields with a focus on the mutual influences that appear within such systems. Based on this examples we derived the problem statement and gave an outline for the thesis.

This was followed by a clarification of terms and a definition of the target systems for this thesis. Furthermore, the system model was put into the context of the observer/controller architecture and reinforcement learning. Additionally, a brief introduction to the applications used in this thesis was given.

Afterwards, related work regarding the detection of mutual influences is explored. The approaches are mainly from the MARL domain and alternative statistical methods. It was followed by the introduction of the approach proposed in this thesis, i.e., the general workflow and the technical application of mutual influence detection was elaborated. This included the dependency measures used in this thesis as well as examples and experiments on elementary use cases and the SCN domain.

The methodology has then then been extended to consider multiple configuration components by using an approximation of the conditioned dependency values. Again, an example was given and the approach was evaluated on elementary use cases and the SCN application.

Afterwards, a further extension was introduced to enable the detection of delayed influences, i.e., influences that will only show their effect after a period of time. This was achieved by including the past configuration in the detection. Since such

delayed influences do not appear in the examined SCN application, the approach has been evaluated on the smart factory application.

Next, the behavior of the approach has been examined under runtime conditions. This led to disturbances of the measurement in the form of correlations that occur by chance, and a course of action to circumvent this was shown. The evaluation has been conducted on the SCN application.

Finally, a methodology for the adaption to influences was given by adapting the basic RL model to handle them by means of an integration of the influencing configuration components into in the state space of the influenced system. The usefulness of the approach has been demonstrated in SCN scenarios.

After the introduction of the method and its extensions, the approach as a whole was discussed by giving practical advice on how to detect influences in an efficient way.

In the following, we revisit the potential areas of application introduced in Section 1.1 and discuss exemplary how the influence detection can contribute to these areas.

- **Cloud computing** often relies on several servers that have to be coordinated and possibly fulfill different tasks. In such an environment it is very likely that the servers influence each other on a level that is not not obvious to the user. For instance a server can rely on the well-function of another (cf. Amazon S3 in Section 1.1). Such influences can be detected by methods proposed in this thesis in order to allow the servers to adapt to each others behavior.

- **Vehicular traffic control** can be improved by employing so called progressive signal systems, which time the green phases of traffic lights of neighboring intersections to reduce the waiting time. Today, the identification of potential paths of such a coordination is typically done manually by a traffic engineer and therefore time consuming and mostly static over the life time of the system. Following the methodology introduced in this thesis, such influences can be detected automatically during runtime and the systems can self-adapt to the influences. Furthermore, it can be detected if the influences change during runtime, e.g., in case of road works between intersections, and the system can adapt to this circumstances, accordingly.

- **Industry 4.0** contains the emerging field of smart factories. The potential

benefits of the influence detection in this area have been shown throughout this thesis. For instance, this is the detection of influences between work stations regarding their configuration of tools which can lead to a improved behavior of the factory if considered accordingly.

- **Internet of things** relies on smart camera networks for several applications, e.g., the identification of objects, persons, or their behavior. The potential benefits of the influence detection for an optimal configuration of the smart cameras have been shown throughout this thesis. For instance, this is the distinction between influencing and non-influencing cameras for the purpose of observation from multiple angles. This distinction can then be used to learn a cooperative behavior.

10.2 Outlook

In the following two directions for future work are outlined. They aim mainly to further automate the detection process by an automatic combination of the dependency measures and to extend the applicability to systems that are not covered by the current system model.

Automated Weighting of Dependency Measures In Section 9.2, we introduced the idea to use multiple dependency measures for the detection. This is due to the different detection rates for the different classes of systems. The recommendation relies on simple techniques such as majority voting, but a useful approach could be to consider ensemble learning techniques [114]. Especially methods, where the combination of the individual learners is done by determining the weights of the individual learners on a case by case basis regarding the structure of the system, could further automate the process of influence detection. The main challenge here lies in the engineering of a feedback signal or a training set that allows the combinator to find optimal weights for each situation.

Detecting Influences in Systems without a Local Reward Signal In this thesis, the system model relies on a local reward that is a feedback signal to the subsystems regarding their decisions. If their is no obvious choice for such a reward signal, it can be useful to have an influence detection mechanism that is able to work without such. Therefore, an interesting approach for future work is to change the system model to a type where only the configuration and a current state of the subsystems is necessary.

The dependency can then be measured between one system's configuration and another system's situation. However, this introduces the problem that the situation is very likely not a one dimensional variable but has several dimensions, e.g., the position is composed of a x- and y-coordinate. Therefore, it will be necessary to use dimension reduction techniques on the situation of a system or to measure the dependency between two groups of random variables, e.g., with a canonical correlation analysis [115].

Besides these two concrete directions it is promising to apply the influence detection to real-world systems, especially with different learning algorithms and influence structures.

Bibliography

The entries [1]-[26] are own publications and can be found on page xiii.

[27] M. Weiser, “The computer for the 21st century,” *Scientific American*, vol. 265, no. 3, pp. 66–75, Jan. 1991.

[28] Bitkom e.V., *Zukunft der consumer technology – 2017: Marktentwicklung, Trends, Mediennutzung, Technologien, Geschäftsmodelle*, 2017.

[29] Kantar Worldpanel, *An incredible decade for the smartphone: What’s next? The future of mobile is in combining devices, content, and services*, Feb. 24, 2017.

[30] M. T. Koroglu and K. M. Passino, “Illumination balancing algorithm for smart lights,” *IEEE Transactions on Control Systems Technology*, vol. 22, no. 2, pp. 557–567, Mar. 2014.

[31] A. Radziwon, A. Bilberg, M. Bogers, and E. S. Madsen, “The smart factory: Exploring adaptive and flexible manufacturing solutions,” *Procedia Engineering*, vol. 69, no. Supplement C, pp. 1184–1190, 2014, 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013.

[32] J. Lutin, A. Kornhauser, and E. Lerner-Lam, “The revolutionary development of self-driving vehicles and implications for the transportation engineering profession,” *ITE Journal*, pp. 21–26, Jul. 2013.

[33] M. Broy (Ed.), *Cyber-Physical Systems: Innovation durch softwareintensive eingebettete Systeme*, ser. acatech Diskutiert. Berlin: Springer, 2010.

[34] C. Low, Y. Chen, and M. Wu, “Understanding the determinants of cloud computing adoption,” *Industrial Management & Data Systems*, vol. 111, no. 7, pp. 1006–1023, 2011.

[35] C. Müller-Schloer and S. Tomforde, *Organic Computing – Technical Systems for Survival in the Real World*. Springer International Publishing, 2018.

[36] C. Müller-Schloer, H. Schmeck, and T. Ungerer, *Organic Computing - A Paradigm Shift for Complex Systems*, 1st. Berlin, Heidelberg: Springer-Verlag, 2011.

[37] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[38] D. Tennenhouse, “Proactive computing,” *Commun. ACM*, vol. 43, no. 5, pp. 43–50, May 2000.

[39] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[40] S. Kernbach, T. Schmickl, and J. Timmis, “Collective adaptive systems: Challenges beyond evolvability,” *CoRR*, vol. abs/1108.5643, 2011.

[41] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[42] C. O’Flaherty, *Transport planning and traffic engineering*. Elsevier, 1997.

[43] H. Prothmann, S. Tomforde, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, “Organic traffic control,” in *Organic Computing — A Paradigm Shift for Complex Systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Basel: Springer Basel, 2011, pp. 431–446.

[44] S. Tomforde, H. Prothmann, F. Rochner, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, “Decentralised progressive signal systems for organic traffic control,” in *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Oct. 2008, pp. 413–422.

[45] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, “How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective,” *J. Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, vol. 8, no. 1, pp. 37–44, 2014.

[46] K. Bellman, A. Herkersdorf, and M. G. Hinchey, “Organic computing - design of self-organizing systems (dagstuhl seminar 11181),” in *Dagstuhl Reports*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, vol. 1, 2011.

[47] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[48] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013, Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[49] S. P. Mohanty, U. Choppali, and E. Kougianos, “Everything you wanted to know about smart cities: The internet of things is the backbone,” *IEEE Consumer Electronics Magazine*, vol. 5, no. 3, pp. 60–70, Jul. 2016.

[50] M. Yun and B. Yuxin, “Research on the architecture and key technology of internet of things (iot) applied on smart grid,” in *2010 International Conference on Advances in Energy Engineering*, Jun. 2010, pp. 69–72.

[51] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. S. Kwak, “The internet of things for health care: A comprehensive survey,” *IEEE Access*, vol. 3, pp. 678–708, 2015.

[52] S. Tomforde, J. Hähner, H. Seebach, W. Reif, B. Sick, A. Wacker, and I. Scholtes, “Engineering and Mastering Interwoven Systems,” in *ARCS 2014 - 27th International Conference on Architecture of Computing Systems, Workshop Proceedings, February 25-28, 2014, Luebeck, Germany, University of Luebeck, Institute of Computer Engineering*, 2014, pp. 1–8.

[53] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, “Possibilities and limitations of decentralised traffic control systems,” in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2010, pp. 1–9.

[54] M. Quaritsch, M. Kreuzthaler, B. Rinner, H. Bischof, and B. Strobl, “Autonomous multicamera tracking on embedded smart cameras,” *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 092 827, Jan. 2007.

[55] K. L. Bellman, S. Tomforde, and R. P. Würtz, “Interwoven systems: Self-improving systems integration,” in *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014*, 2014, pp. 123–127.

[56] S. Tomforde and C. Müller-Schloer, “Incremental design of adaptive systems,” *JAISE*, vol. 6, no. 2, pp. 179–198, 2014.

[57] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st. Cambridge, MA, USA: MIT Press, 1998.

[58] A. van Lamsweerde, “Goal-oriented requirements engineering: A guided tour,” in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[59] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, 1st. Wiley Publishing, 2009.

[60] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, “Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty,” in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014, Hyderabad, India: ACM, 2014, pp. 17–26.

[61] L. Busoniu, R. Babuska, and B. De Schutter, “A comprehensive survey of multiagent reinforcement learning,” *Trans. Sys. Man Cyber Part C*, vol. 38, no. 2, pp. 156–172, Mar. 2008.

[62] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Springer Publishing Company, Incorporated, 2014.

[63] C. J. C. H. Watkins and P. Dayan, “Technical note q-learning.,” *Machine Learning*, vol. 8, pp. 279–292, 1992.

[64] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck, “Observation and Control of Organic Systems,” in *Organic Computing - A Paradigm Shift for Complex Systems*, ser. Autonomic Systems, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., Birkhäuser Verlag, 2011, pp. 325–338.

[65] C. Gruhl, B. Sick, A. Wacker, S. Tomforde, and J. Hähner, “A building block for awareness in technical systems: Online novelty detection and reaction with an application in intrusion detection,” in *2015 IEEE 7th International Conference on Awareness Science and Technology (iCAST)*, Sep. 2015, pp. 194–200.

[66] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, “Adaptivity and Self-organization in Organic Computing Systems,” *ACM Trans. on Autonomous and Adaptive Systems*, vol. 5, no. 3, pp. 1–32, 2010.

[67] IBM, “An architectural blueprint for autonomic computing,” IBM, Tech. Rep., Jun. 2005.

[68] S. Tomforde, *An Architectural Framework for Self-configuration and Self-improvement at Runtime*. 2011.

[69] S. Soro and W. Heinzelman, “A survey of visual sensor networks,” *Advances in multimedia*, vol. 2009, 2009.

[70] Beckhoff Automation GmbH. (2014). “Industrie 4.0 Forum: pc-based control concepts as core technology for the smart factory,” [Online]. Available: `http://ftp.beckhoff.com/download/document/catalog/Flyer_Industry_40_forum_2014.pdf` (visited on 03/29/2016).

[71] R. Sanchez, M. Smith, B. Person, and P. B. Hole, “C-137 universal developments,” *Meseeks Journal*, vol. 29, no. 2, pp. 42–69, 2013.

[72] D. Keil and D. Q. Goldin, “Modeling indirect interaction in open computational systems,” in *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, 9-11 June 2003, Linz, Austria*, 2003, pp. 371–376.

[73] R. Logie, J. G. Hall, and K. G. Waugh, “Towards mining for influence in a multi agent environment.,” in *IADIS European Conf. Data Mining*, A. Abraham, Ed., IADIS, 2008, pp. 97–101.

[74] R. Logie, J. G. Hall, and K. G. Waugh, “Investigating agent influence and nested other-agent behaviour,” *International Journal on Advances in Intelligent Systems Volume 2, Number 4, 2009*, 2010.

[75] J. M. Broersen, “CTL.STIT: enhancing ATL to express important multi-agent system verification properties,” in *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, 2010, pp. 683–690.

[76] P. Stone and M. Veloso, “Multiagent systems: A survey from a machine learning perspective,” English, *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[77] N. Prasad, V. R. Lesser, and S. E. Lander, “Learning organizational roles in a heterogeneous multi-agent system,” in *Proceedings of the Second International Conference on Multi-Agent Systems*, California: AAAI Press, Jan. 1996, pp. 291–298.

[78] J. R. Kok, M. T. J. Spaan, and N. Vlassis, “Multi-robot decision making using coordination graphs,” in *Proceedings of the International Conference on Advanced Robotics (ICAR)*, A. T. de Almeida and U. Nunes, Eds., Coimbra, Portugal, Jun. 2003, pp. 1124–1129.

[79] J. R. Kok, P. J. ’. Hoen, B. Bakker, and N. Vlassis, “Utile coordination: Learning interdependencies among cooperative agents,” in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, Colchester, United Kingdom, Apr. 2005, pp. 29–36.

[80] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, “Learning what to observe in multi-agent systems,” in *Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence*, 2009, pp. 83–90.

[81] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, “Learning multi-agent state space representations,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 715–722.

[82] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, “Solving delayed coordination problems in mas,” in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, ser. AAMAS ’11, Taipei, Taiwan: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 1115–1116.

[83] P. Vrancx, Y.-M. De Hauwere, and A. Nowé, “Transfer learning for multi-agent coordination,” in *Proceedings of the 3th International Conference on Agents and Artificial Intelligence*, Rome, Italy, 2011, pp. 263–272.

[84] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, “A unified game-theoretic approach to multiagent reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4190–4203.

[85] H. Peng, F. Long, and C. Ding, “Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.

[86] S. Lange and M. Riedmiller, “Deep auto-encoder neural networks in reinforcement learning,” in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2010, pp. 1–8.

[87] A. M. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor, “Regularized policy iteration,” in *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, 2008, pp. 441–448.

[88] J. Z. Kolter and A. Y. Ng, “Regularization and feature selection in least-squares temporal difference learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML ’09, Montreal, Quebec, Canada: ACM, 2009, pp. 521–528.

[89] D.-R. Liu, H.-L. Li, and D. Wang, “Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey,” *International Journal of Automation and Computing*, vol. 12, no. 3, pp. 229–242, 2015.

[90] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman, “An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning,” in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML ’08, Helsinki, Finland: ACM, 2008, pp. 752–759.

[91] J. Bishop and R. Miikkulainen, “Evolutionary feature evaluation for online reinforcement learning,” in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, Aug. 2013, pp. 1–8.

[92] H. Hachiya and M. Sugiyama, “Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information.,” in *ECML/PKDD (1)*, ser. Lecture Notes in Computer Science, vol. 6321, Springer, 2010, pp. 474–489.

[93] T. Nguyen, Z. Li, T. Silander, and T. Y. Leong, “Online feature selection for model-based reinforcement learning,” in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, S. Dasgupta and D. Mcallester, Eds., vol. 28, JMLR Workshop and Conference Proceedings, 2013, pp. 498–506.

[94] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*, ser. Wiley Series in Probability and Statistics. Wiley, 2008.

[95] L. Ljung, “Approaches to identification of nonlinear systems,” in *Control Conference (CCC), 2010 29th Chinese*, IEEE, 2010, pp. 1–5.

[96] T. Suzuki, M. Sugiyama, J. Sese, and T. Kanamori, “A least-squares approach to mutual information estimation with application in variable selection,” in *Proceedings of the 3rd workshop on new challenges for feature selection in data mining and knowledge discovery (FSDM 2008), Antwerp, Belgium*, 2008.

[97] S. Geisser and W. O. Johnson, *Modes of parametric statistical inference*. John Wiley & Sons, 2006, vol. 529.

[98] G. V. Dallal, *The little handbook of statistical practice*. Gerard V. Dallal, 1999.

[99] D. Fisch, M. Jänicke, E. Kalkowski, and B. Sick, “Learning from others: Exchange of classification rules in intelligent distributed systems,” *Artificial Intelligence*, vol. 187, pp. 90–114, 2012.

[100] D. Fisch, M. Jänicke, E. Kalkowski, and B. Sick, “Techniques for knowledge acquisition in dynamically changing environments,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 1, 16:1–16:25, May 2012.

[101] J. Boes and F. Migeon, “Self-organizing multi-agent systems for the control of complex systems,” *Journal of Systems and Software*, vol. 134, pp. 12–28, 2017.

[102] K. Pearson, “Note on regression and inheritance in the case of two parents,” *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, 1895.

[103] M. G. Kendall, “A new measure of rank correlation,” *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[104] G. J. Székely, M. L. Rizzo, and N. K. Bakirov, “Measuring and testing dependence by correlation of distances,” *The Annals of Statistics*, vol. 35, no. 6, pp. 2769–2794, Dec. 2007.

[105] C. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[106] A. Kraskov, H. Stögbauer, and P. Grassberger, “Estimating mutual information,” *Phys. Rev. E*, vol. 69, p. 066 138, 6 Jun. 2004.

[107] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti, “Detecting novel associations in large data sets,” *Science*, vol. 334, no. 6062, pp. 1518–1524, 2011.

[108] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, “Mason: A multiagent simulation environment,” *Simulation: Transactions of the society for Modeling and Simulation International*, vol. 82, no. 7, pp. 517–527, 2005.

[109] I. Dusparic and V. Cahill, “Distributed w-learning: Multi-policy optimization in self-organizing systems.,” in *2009 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009)*, IEEE, Dec. 21, 2009, pp. 20–29.

[110] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[111] S. W. Wilson, “Classifier Fitness Based on Accuracy,” *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[112] S. W. Wilson, “Get real! xcs with continuous-valued inputs,” in *Learning Classifier Systems*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 209–219.

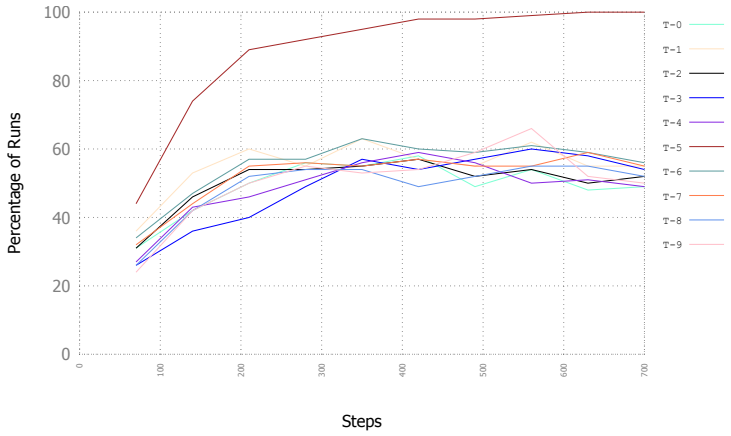
[113] X. Li and G. Yang, “Transferable xcs,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO ’16, Denver, Colorado, USA: ACM, 2016, pp. 453–460.

[114] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, “Ensemble learning for data stream analysis: A survey,” *Information Fusion*, vol. 37, pp. 132–156, 2017.

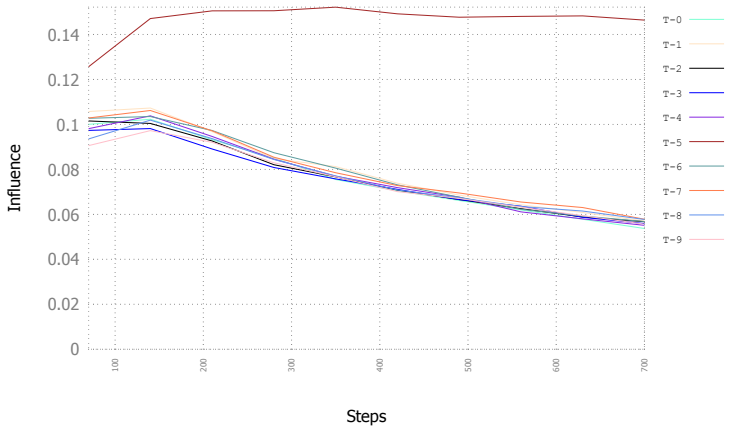
[115] W. Härdle and L. Simar, *Applied multivariate statistical analysis*. Springer, 2007, vol. 22007.

Appendices

A | Delayed Influences

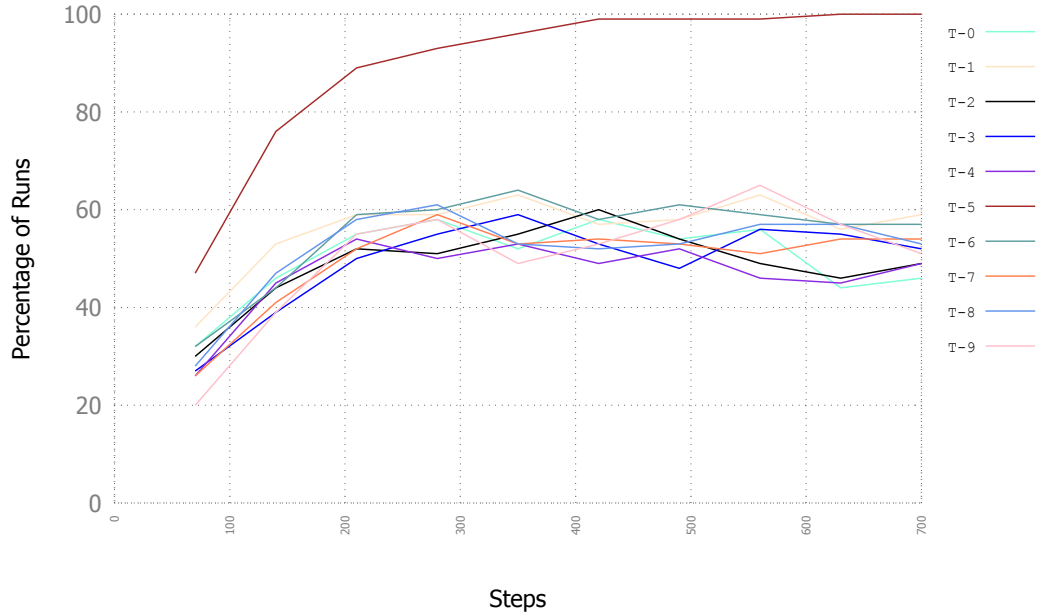


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

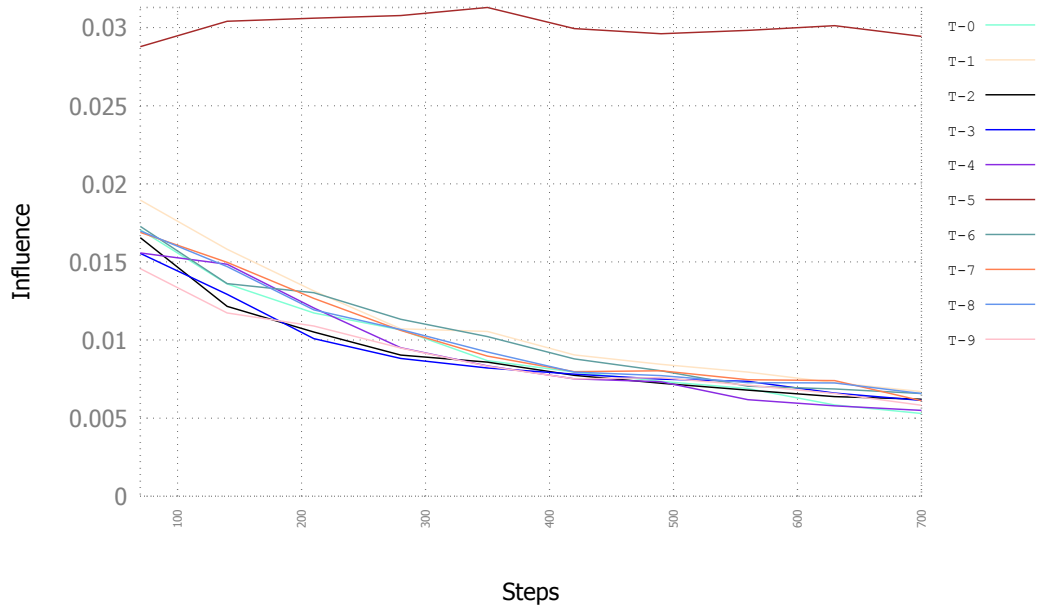


(b) The averaged influence values.

Figure A.1: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **distance correlation**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

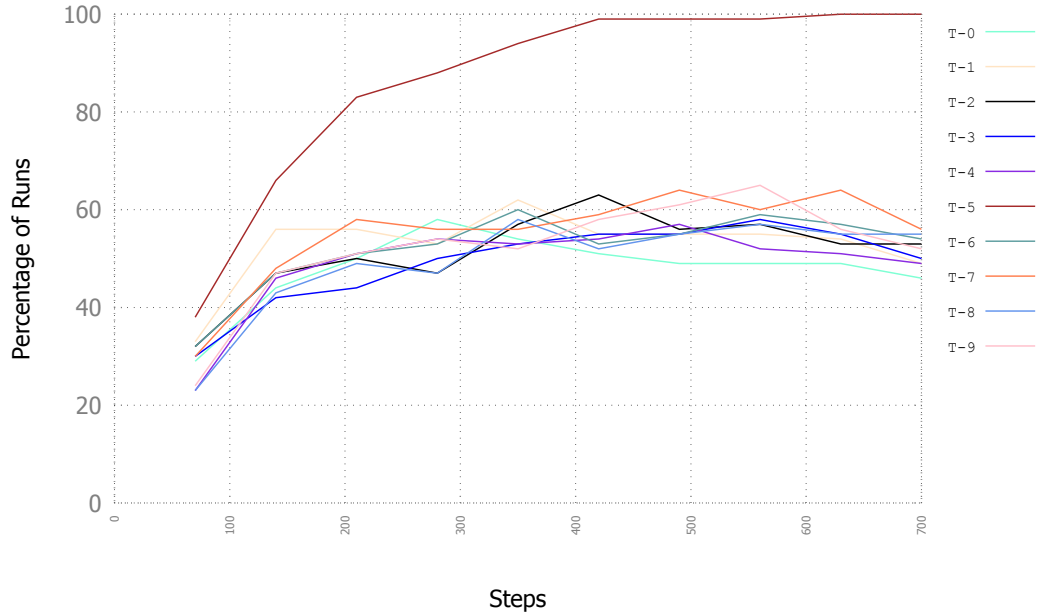


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

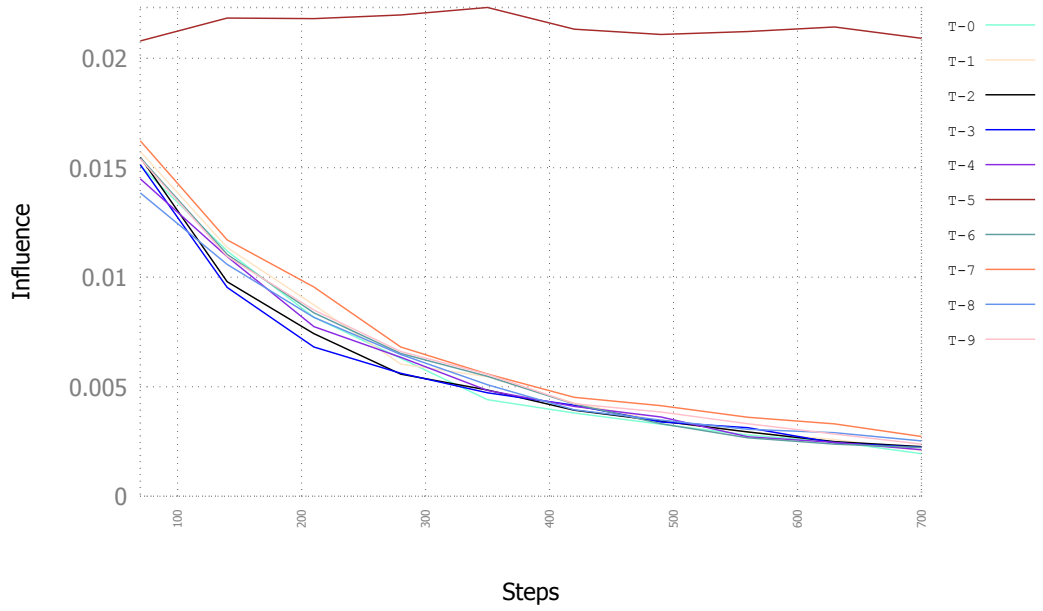


(b) The averaged influence values.

Figure A.2: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Kendall rank correlation**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

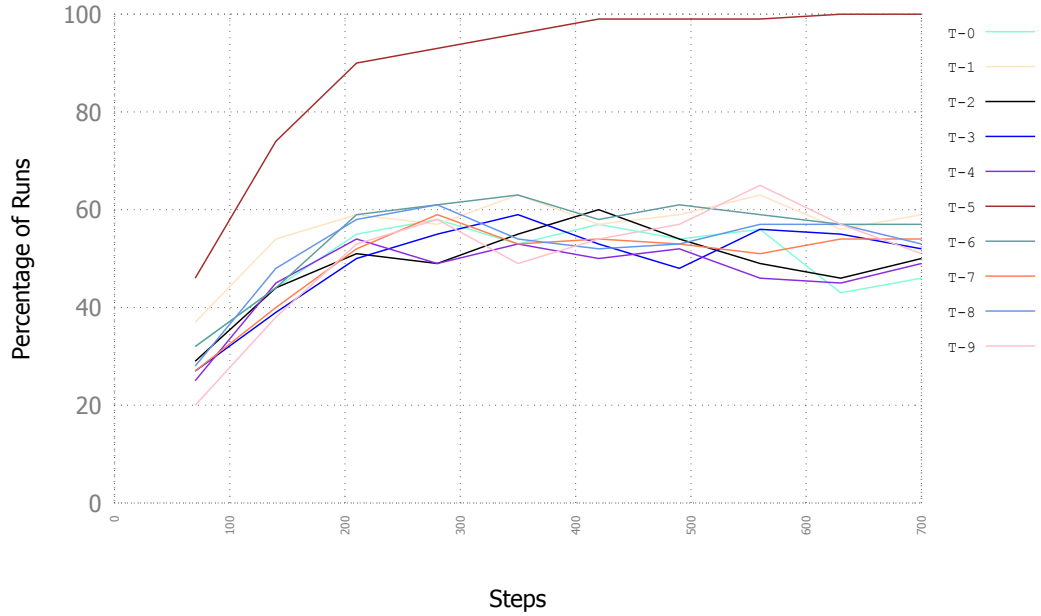


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

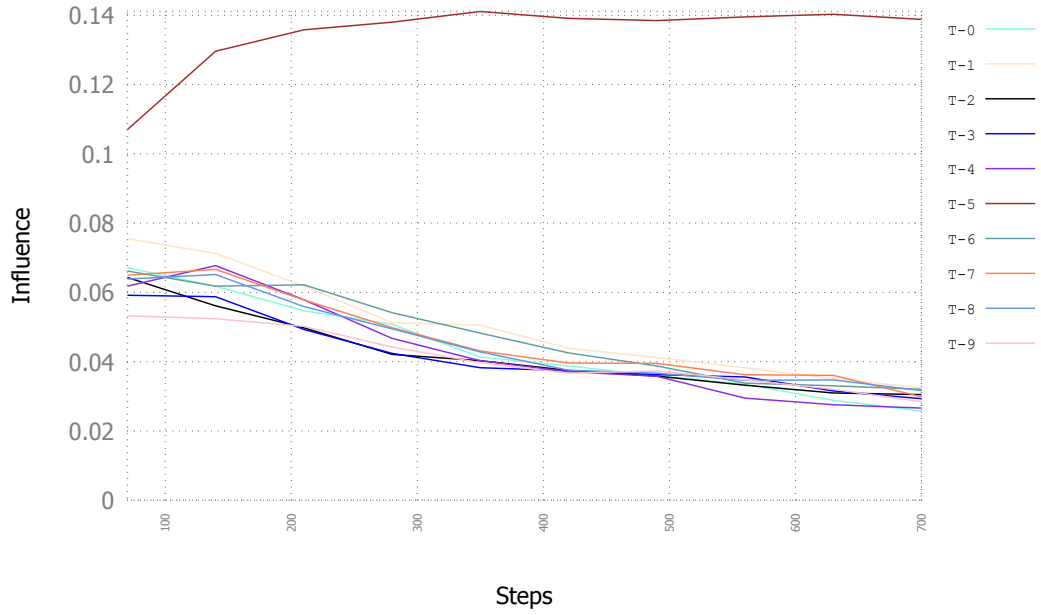


(b) The averaged influence values.

Figure A.3: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Discrete Mutual Information**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

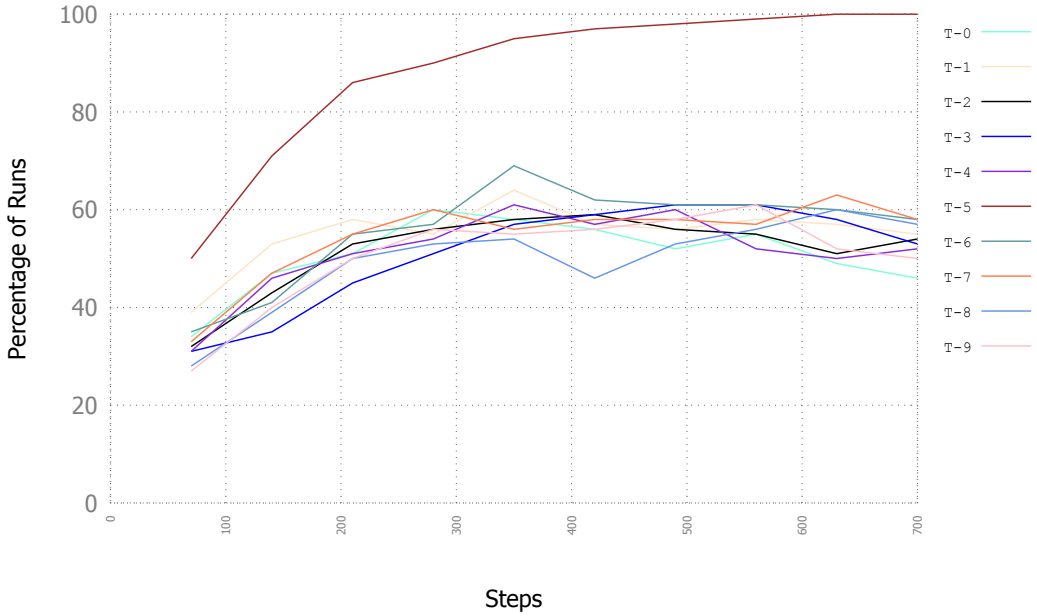


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

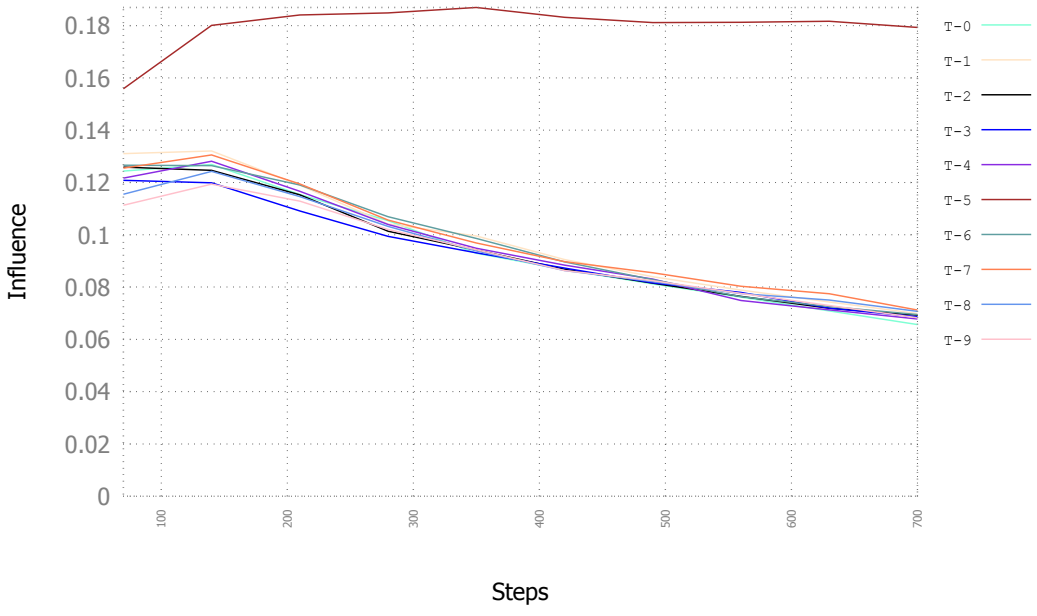


(b) The averaged influence values.

Figure A.4: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Spearman rank correlation**. For this measurement only a **single estimator** has been used, i.e., the configurations of workbench B_{RU} have not been considered.

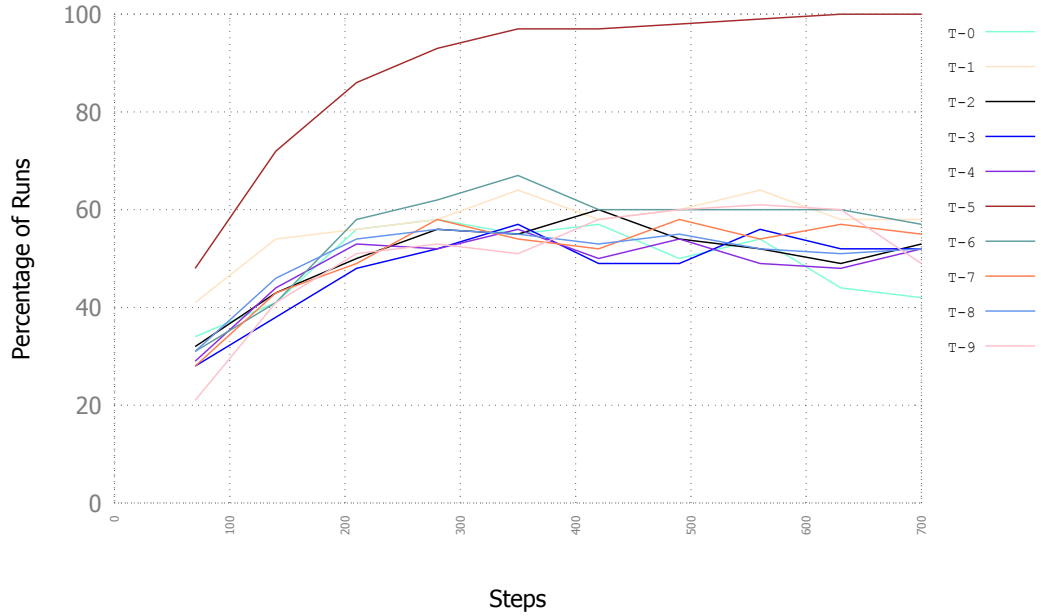


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

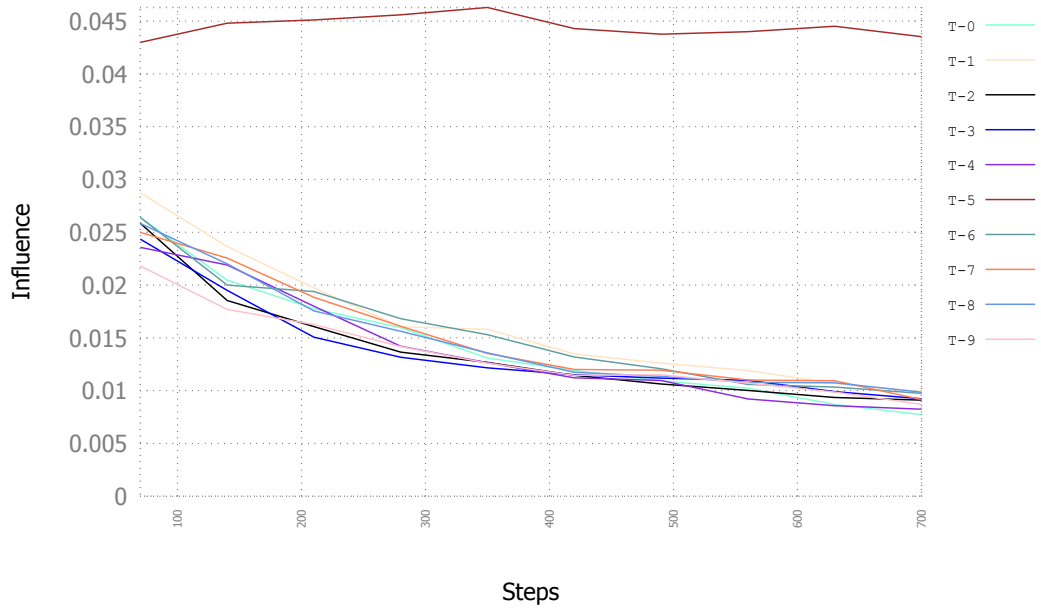


(b) The averaged influence values.

Figure A.5: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **distance correlation**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.

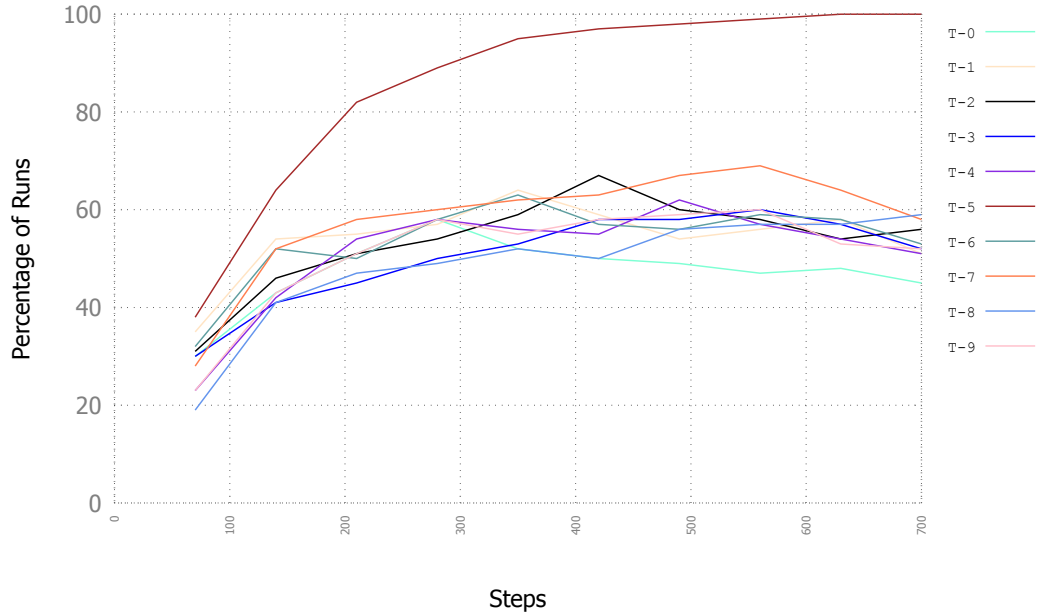


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

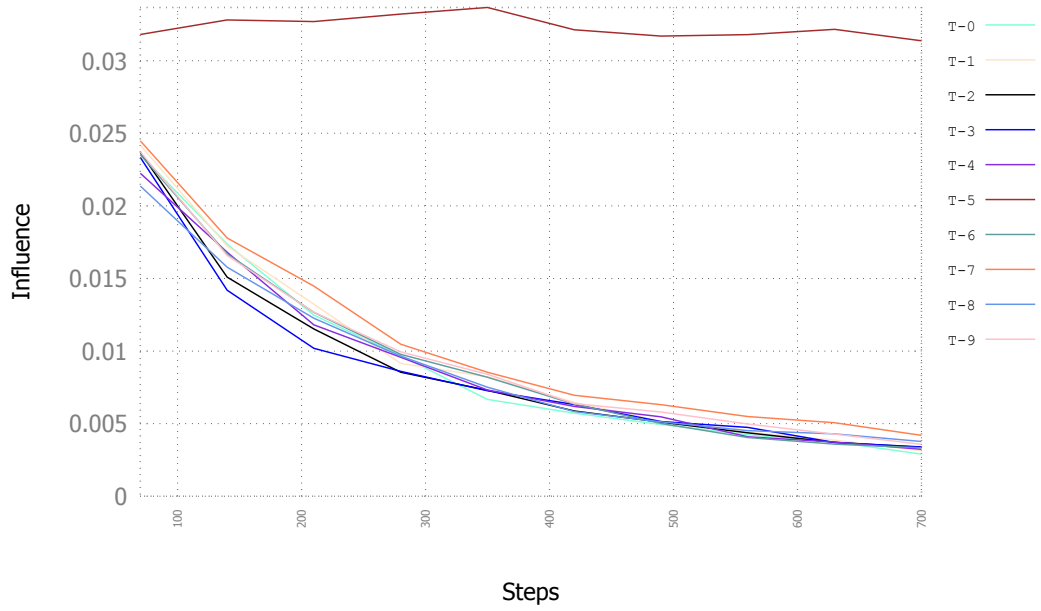


(b) The averaged influence values.

Figure A.6: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Kendall rank correlation**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.

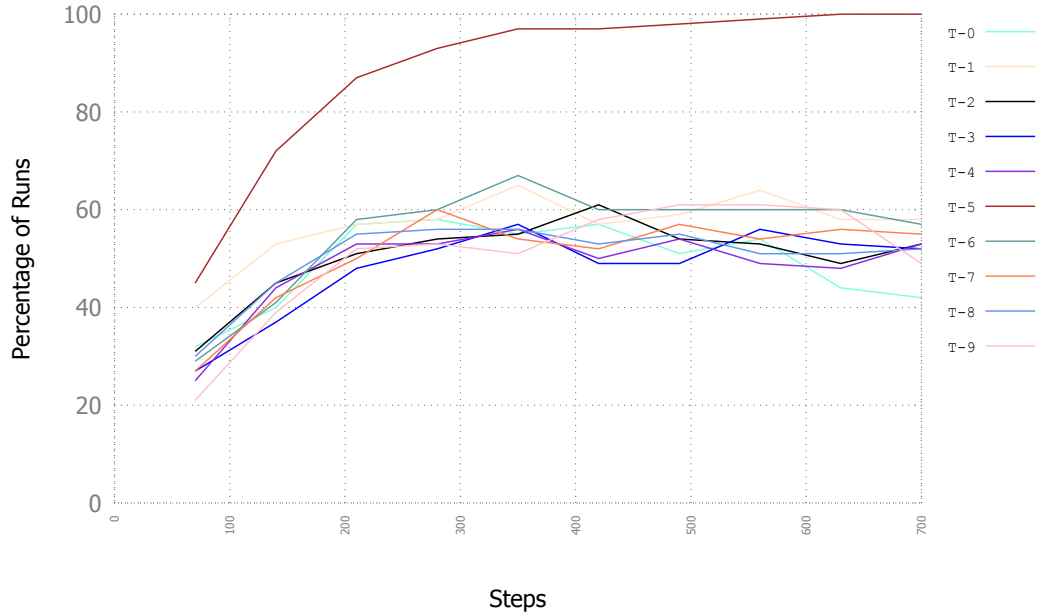


(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.

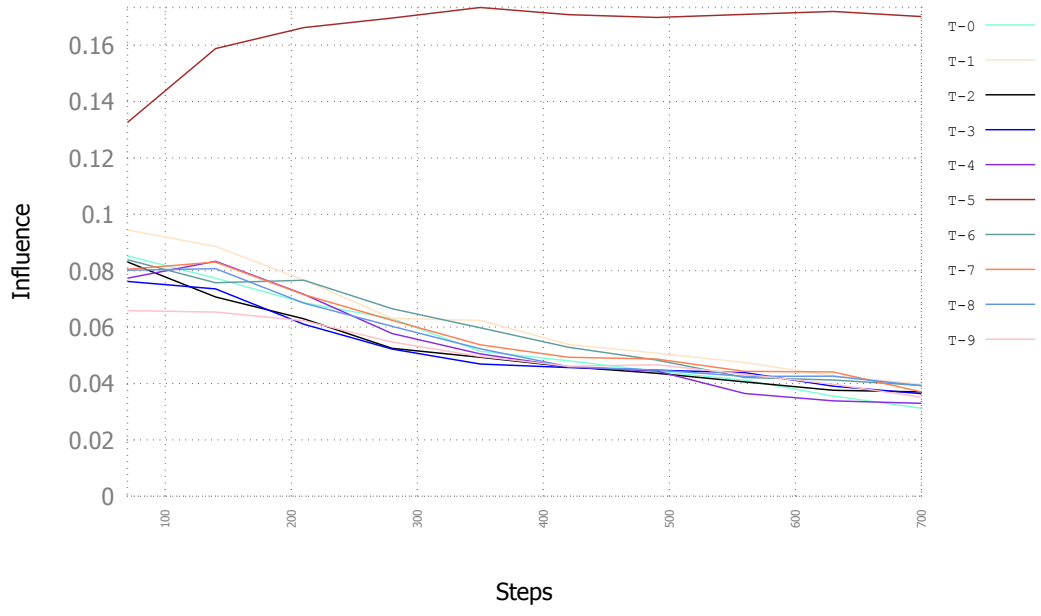


(b) The averaged influence values.

Figure A.7: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Discrete Mutual Information**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.



(a) The fraction of 100 runs in which the influence has been detected as higher than a notional independent workbench for each delay in time steps.



(b) The averaged influence values.

Figure A.8: The results for the smart factory application. The graphs show the results for the influence of the upper workbench at the left station B_{LU} on the upper workbench on the right station B_{RU} measured with the **Spearman rank correlation**. For this measurement **two estimators** have been used, i.e., the configurations of workbench B_{RU} have been taken into account.