Industrial-Scale Environments With Bounded Uncertainty: A Productivity Maximisation Challenge

Daniel Sykes
Ocado Technology
daniel.sykes@ocado.com

Gavin Keighren
Ocado Technology
gavin.keighren@ocado.com

ABSTRACT

We present an outline of the operating domain for control software in Ocado warehouses, and provide results which suggest that in this well-understood and highly controlled environment, there are limits to the uncertainty which planning and control systems need to consider. More specifically, that planning approaches can be biased towards *rapid recovery* when something goes wrong, rather than trying to deal with all possible eventualities up-front. Since academic interest has generally focused on complex and highly fault-tolerant up-front planning, we believe this domain and planning approach is fertile ground for further investigation.

CCS CONCEPTS

• **Software and its engineering** → **Software fault tolerance**; *Real-time systems software*; • **Applied computing** → *Online shopping*;

KEYWORDS

Planning, Bounded Uncertainty, Redundancy, Resilience

ACM Reference Format:
Daniel Sykes and Gavin Keighren. 2018. Industrial-Scale Environments With Bounded Uncertainty: A Productivity Maximisation Challenge. In *RoSE'18: RoSE'18:IEEE/ACM 1st International Workshop on Robotics Software Engineering , May 28-June 28 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196558.3196563

1 INTRODUCTION

Autonomous, self-adaptive, and robotic systems are deployed in many environments which exhibit unavoidable unpredictability of events and outcomes. Consequently, much research effort has been expended on approaches that tolerate and adapt to uncertainty [5, 9]. However, the associated uncertainty differs by environment and therefore the ideal planning and control strategies will differ. Environments that are largely unexplored, where the cost of failure is very high (such as the surface of Mars), demand elaborate, open-ended and highly fault-tolerant planning and machine learning that come at significant computational and financial cost [2, 3]. In contrast, environments that are well understood and highly controlled (such as a logistics warehouse) allow for approaches which can operate at scale while still being cost-effective [11].

Ocado operates a number of automated warehouses in which robots, conveyors and lifts operate 24/7 to fulfil grocery orders, subject to hard shipping deadlines, product shelf-life requirements, and storage and handling constraints [8, 15]. These warehouses (also known as Customer Fulfilment Centres — CFCs) constitute a controlled artificial environment in which sources of uncertainty are generally bounded and limited to a set of *known unknowns* (in the Rumsfeld ontology [17]). We therefore focus our effort on minimising the impact of these known failures, rather than undertaking complex deliberative processes that attempt to deal with every possible eventuality.

One of the ways the impact of uncertainty is minimised in the automated warehouse is by making the hardware components interchangeable (i.e., redundant). The components present in an Ocado warehouse include mobile robots, conveyors, shuttles,[1] and lifts (which we collectively term *actuators* hereafter). Redundancy is used to ensure there is no single point of failure. Each warehouse is divided into three temperature regimes—ambient, chill and freezer—according to the kind of stock stored there.

The sources of uncertainty in the warehouse include (but are not limited to):

- failures due to mechanical wear and tear,
- performance variations due to manufacturing inconsistencies,
- performance variations due to operational capabilities of the hardware,
- variation in the performance of human operators,
- variation in radio communication performance,
- failures due to hardware manufacturing defects,
- failures due to dirt and contamination, and
- human errors.

To account for slight performance differences between individual instances of a specific type of actuator, suitable bounds can be specified for these variations. This *performance envelope* allows the control system to more easily account for and tolerate individual performance deviations. The failure of an individual actuator is usually tolerable as the work can be allocated to another of the same type. The challenge is to ensure the fault is adequately quarantined and the work is reallocated as quickly as possible and without human intervention.

Allocation of work is done by the warehouse control systems, which select strategies and decide upon the sequence of actions to be performed. Since the control systems exploit bounded uncertainty in assuming reliable hardware operating within a performance

[1] https://www.tgw-group.com/en/Products/Storage-Solution/Shuttle-Systems/Shuttle-Systems

RoSE'18, May 28-June 28 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5760-9/18/05…$15.00
https://doi.org/10.1145/3196558.3196563

envelope, any failure invalidates at least one local task, and, in the worst case, every task. The control systems must then take remedial action to quarantine or correct the fault and reallocate work to ensure minimal loss of throughput (which can be measured as the rate of completing customer orders).

In the remainder of the paper, Section 2 outlines our general approach in dealing with bounded uncertainty, Section 3 presents some results of the approach from a live Ocado warehouse, and Sections 4 and 5 relate our approach to other work in the field and give our view on what challenges remain.

2 APPROACH

The primary actions available to the warehouse control system are those related to movement of storage containers in the X, Y, or Z direction. For example, a conveyor may move containers in the X direction, while a mobile robot or a lift may move containers in multiple axes. The sensors available to each hardware component may indicate a fault or reduced performance, either during an action or before an action is scheduled to be performed.

Sequences of actions are selected to allocate work to particular actuators, and coordinate their movement (for instance, to avoid collisions). Action sequences are composed into local plans for each actuator, and (notionally) a global plan for the warehouse as a whole. The large number of actuators in the warehouse (which is in the thousands) means that this plan generation task is computationally challenging, even without considering the possibility of failures. For example, if 500 storage containers are in motion at once, then at least 500 concurrent actions must have been selected and scheduled by the control system.

Among the wide range of approaches for planning, we have therefore chosen to apply computationally efficient algorithms to minimise this up-front cost, and defer most decisions pertaining to failures until the failures actually occur. This is in contrast to approaches such as [1, 4–6, 16], which variously attempt to precalculate strategies for failure handling in more or less exhaustive ways. Those approaches increase the cost of generating the initial plan (and any subsequent plans that have to be generated as a result of *unknown unknowns*) in the hope of never needing to regenerate a plan. In the warehouse, we can exploit the large-scale redundancy of components to localise the impact of failures, and perform local recalculations instead of global ones. Furthermore, the use of performance envelopes ensures that actual failure cases are sufficiently infrequent.

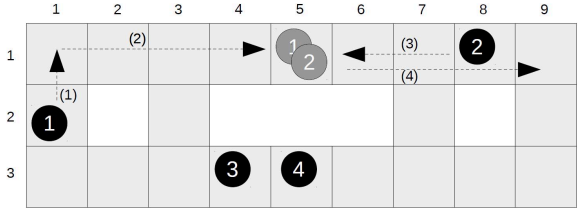

Figure 1: Black circles show actuators capable of moving in X or Y. Arrows indicate planned actions.

Let us consider the case of a storage container being moved from an (x, y) storage location at $(1, 1)$ to $(9, 1)$ as in Figure 1. Suppose there is a constraint in the physical environment that means two actuators must be co-ordinated to complete this movement. The control system may generate a partially ordered sequence of actions such as

```
1 (move, _, actuator1, (_, _), (1, 1))
2 (move, container1, actuator1, (1, 1), (5, 1))
3 (move, _, actuator2, (_, _), (5, 1))
4 (move, container1, actuator2, (5, 1), (9, 1))
```

In other words, *actuator1* must move into position $(1, 1)$, where it will collect *container1*; *actuator1* must move (in the X direction) from $(1, 1)$ to $(5, 1)$; *actuator2* must move into position $(5, 1)$, where *container1* will be handed over from *actuator1*; and *actuator2* must move from $(5, 1)$ to $(9, 1)$. Note that although the actions are presented in a total order, step 3 is free to happen earlier or in parallel with steps 1 and 2.

The two actuators may invalidate this plan in various known ways. For example, *actuator1* may perform slower than expected and miss its synchronisation point with *actuator2*. Alternatively, *actuator2* may experience a hardware fault on its way to synchronise with *actuator1*. If either of these situations actually occurs, there is little reason for it to have immediate impact on the work of *actuator3* and *actuator4* (which may be handing over a different container), or the thousands of other actuators operating simultaneously. This locality of failure is the key characteristic of Ocado's problem domain that makes our approach advantageous.

Once the control system is apprised of the situation, the local plan is invalidated and remedial actions are applied. The steps to recovery are as follows:

(1) **Plan invalidation** - detecting which tasks are no longer valid.
(2) **Fault quarantine** - e.g., disabling faulty hardware as necessary and ensuring no other actuators attempt to interact with it.
(3) **Fault escalation** - e.g., alerting hardware engineers or operations personnel.
(4) **Performance book-keeping** - e.g., altering preference functions.
(5) **Plan recalculation** - including physical remedial steps such as "undoing" previous actions, correcting mistakes arising from the fault, and actions that resolve any ambiguity in the physical state of the system (e.g., scanning the barcode on a container to check its identity).

In the case of slow performance of *actuator1* (which is outside the expected performance envelope), it may be the case that no specific remedial action (other than recalculating a plan) is necessary. If the actuator is frequently performing poorly, the control system may decide to escalate the issue to an engineer and prefer not to allocate work to the actuator in the meantime. The recalculated plan may continue to use *actuator2*, although the time of synchronisation between the actuators will be different. There may be a knock-on effect on any future work involving either actuator, which must be recalculated. For example, if *actuator2* was meant to hand over the container to *actuator4*, the plan for *actuator4* will need to be recalculated.

In the case of a hardware fault on *actuator2*, the remedial action will depend on the type of problem and history of the actuator. As with performance issues, if the actuator is frequently exhibiting the fault, the control system may escalate the issue to an engineer. In any case, the fault is immediately quarantined and the rest of the warehouse continues operating. Since *actuator2* can no longer be used for this work, a different actuator (e.g. *actuator4*) must be selected to take over steps 3 and 4 of the task.

Human errors (such as a container being moved unexpectedly) are, generally speaking, detected indirectly through the sensing capabilities of each hardware component. Since the control system must tolerate sensor values changing unexpectedly (due to hardware faults), the recovery process is often identical, even when the resulting system state is ambiguous. For example, the physical state of the system can be determined by scanning the barcodes on the affected containers.

3 RESULTS

To demonstrate the utility of our approach, we provide empirical results from a live Ocado warehouse showing the impact of situations that invalidate the global plan in comparison to situations with local impact. Figure 2 shows how one measure of system productivity — the number of in-progress tasks — is affected when the global plan is invalidated. The downtime duration is dependent on how long it takes for the actuators to be returned to a state in which they can be allocated work (steps 1-4 of the recovery process described above) or in which they can safely be left unused — a process which may require manual intervention (step 3 of the recovery process). In this particular case, the downtime is less than 1 minute and the ramp-up period following the downtime takes approximately 10-15 seconds.

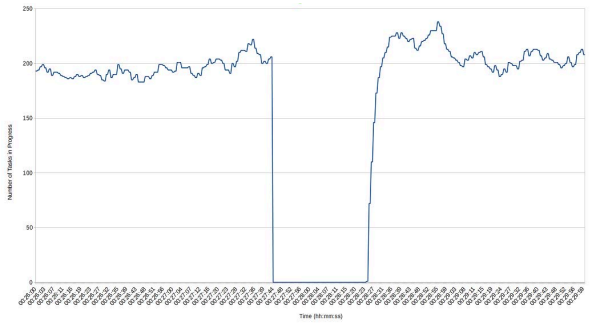

Figure 2: Impact of clearing the global plan

Figure 3, in contrast, shows the impact of a local fault, with respect to the same measure of system productivity as used in Figure 2. At point B on the graph, 11 tasks were invalidated, while at point A, 18 tasks naturally completed within approximately 100ms of each other. Given that the subsequent recovery of system productivity is very similar in both cases (taking 1s or less), this shows that the impact of local plan invalidation is indistinguishable from the natural variation in work coming into the system.

Figure 4 shows how often some number of tasks are invalidated as a result of local plans being recalculated, with over 95% of cases

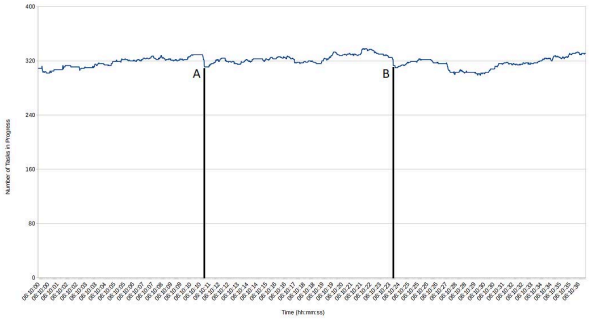

Figure 3: Impact of clearing local plans

invalidating fewer than 5 tasks.[2] This is orders of magnitude fewer tasks compared to when the global plan has to be recalculated.

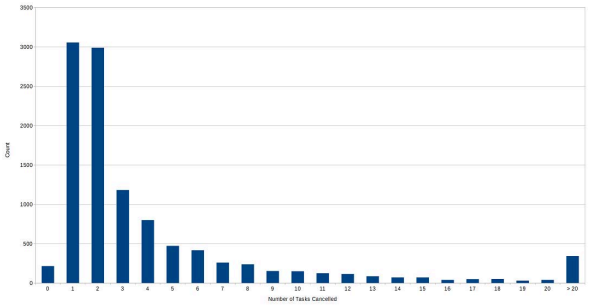

Figure 4: Frequency of task invalidation counts due to the clearing of local plans

4 RELATED WORK

Our approach is aligned with both the MAPE-K reference model [12, 13] and the conceptual architecture of the three-layer model of Kramer and Magee [10, 14]. The latter provides a hierarchical distinction between functionality such that real-time concerns are dealt with as close to the hardware as possible, while slower deliberative tasks take place in the higher layers. In contrast, MAPE-K explicitly separates the stages of monitoring, analysing, planning and executing. In the warehouse, monitoring of performance and failures provides input for plan recalculation and eventual execution by the actuators. We have focused on making the recalculation efficient so that loss of productivity is minimised.

As indicated above, our approach runs somewhat counter to the many and varied approaches that attempt to deal with uncertainty up-front [1, 4–6, 16, 18]. Our experience, with a highly redundant industrial-scale system (with high availability requirements), has been that it is better to amortise the cost of the bounded uncertainty present in the warehouses over the long run. FUSION is a notable example of another approach that attempts to reduce these up-front costs [7].

[2]The Y axis values are somewhat arbitrary as they depend on the time period covered by the data.

5 CONCLUSIONS

In this paper we have outlined our pragmatic approach to dealing with bounded uncertainty in a highly redundant industrial-scale system. We have described how we exploit the bounded impact of actuator failures and performance variations by performing local plan invalidation, which lowers the time taken to recover from a failure, thus minimising its impact on overall productivity. This is supported by empirical results from a live Ocado warehouse, where local recalculation is indistinguishable from variations in work the system is asked to do.

We believe this insight could be a fruitful basis for future research into the different categories of uncertainty and the approaches that are best suited to each of them.

ACKNOWLEDGEMENTS

Our thanks go to many of our colleagues at Ocado Technology who helped to improve this paper, and who work tirelessly on the systems we have mentioned.

REFERENCES

[1] Ronen I Brafman and Guy Shani. 2012. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research* 45 (2012), 565–600.

[2] Joseph Carsten, Arturo Rankin, Dave Ferguson, and Anthony Stentz. 2007. Global path planning on board the mars exploration rovers. In *Aerospace Conference, 2007 IEEE*. IEEE, 1–11.

[3] Amedeo Cesta, Gabriella Cortellessa, Michel Denis, Alessandro Donati, Simone Fratini, Angelo Oddi, Nicola Policella, Erhard Rabenau, and Jonathan Schulster. 2007. Mexar2: AI solves mission planner problems. *IEEE Intelligent Systems* 22, 4 (2007).

[4] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147, 1-2 (2003), 35–84.

[5] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. 2014. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 688–699.

[6] Nicolás D'Ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. 2011. Synthesis of live behaviour models for fallible domains. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 211–220.

[7] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. 2010. FUSION: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 7–16.

[8] Ocado Engineering. 2018. Ocado Smart Platform CFCs. (Jan 2018). https://ocadoengineering.com/warehouses/#osp_cfcs

[9] Naeem Esfahani and Sam Malek. 2013. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*. Springer, 214–238.

[10] Erann Gat and R Peter Bonnasso. 1998. On three-layer architectures. *Artificial intelligence and mobile robots* 195 (1998), 210.

[11] Christian Hütter. 2016. More shuttles, less cost: energy efficient planning for scalable high-density warehouse environments. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*.

[12] IBM. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 31 (2006), 1–6.

[13] Didac Gil De La Iglesia and Danny Weyns. 2015. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10, 3 (2015), 15.

[14] Jeff Kramer and Jeff Magee. 2007. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering*. IEEE Computer Society, 259–268.

[15] Lars Sverker Ture Lindbo, Robert Rolf Stadie, Matthew Robert Whelan, and Christopher Richard James Brett. 2016. Apparatus for retrieving units from a storage system. (July 7 2016). US Patent App. 14/910,858.

[16] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 1–12.

[17] Wikipedia.org. 2002. There are Known Knowns. (Feb 2002). https://en.wikipedia.org/wiki/There_are_known_knowns

[18] Shuo Yang, Xinjun Mao, Sen Yang, and Zhe Liu. 2017. Towards a hybrid software architecture and multi-agent approach for autonomous robot software. *International Journal of Advanced Robotic Systems* 14, 4 (2017), 1729881417716088.