WASP: Software Engineering Assignment

Matti Vahs

August 2025

Since I could not attend the lectures in person, I added an extra section at the end summarizing the discussions held during the lectures as a make-up assignment.

1 Introduction

My research focuses on risk-aware motion planning and control of robotic systems. Whenever we deploy robots in the real world, uncertainties in various forms are present which have to be considered in the design process of robot control algorithms to ensure that our robots operate as intended. Specifically, I am interested in partially observable systems in which the state of a robot (i.e. its position, joint angles etc.) are not known directly but can be observed through noise corrupted sensors. As a consequence, the exact state of the robot is unknown but we can obtain a belief, i.e. a probability distribution over possible states.

I am concerned with the question of how to design formal synthesis to provide rigorous safety guarantees under imprecise state knowledge. To achieve that, I move the design process of robot autonomy to the belief space instead of the state space. Reasoning about beliefs instead of states allows us to exploit two main properties.

1. **Risk-awareness**: Measures of risk reason about specification violation in the presence of uncertainty. Using risk measures in belief space enables robust planning and control, as risk measures quantify the severity of violation and can capture undesired tail events such as the probability of colliding with obstacles.

2. **Information Gathering**: planning in belief space enables active information gathering behavior where a path can be planned to e.g. reduce the uncertainty about a robots' position.

However, treating problems in belief spaces is challenging for several reasons: 1) the curse of dimensionality in belief spaces complicates the design of computationally efficient solutions, 2) the true underlying distribution over states is unknown as the standard Bayes filter is intractable for continuous nonlinear systems, and 3) hard safety constraints on the state cannot be ensured due to unbounded probability distributions. Especially the latter point requires us to take a risk-aware perspective on safety meaning that small violations are allowed such as, e.g. allowing a small probability of collision. In my research, I strive to tackle these challenges, aiming to achieve my **overarching vision**: benefit from the mathematical rigor of formal methods-based approaches while ensuring their practical applicability in real-world robotic systems.

2 Lecture Principles

2.1 Verification and Validation

One principle covered in the lectures is the distinction between verification (*are we building the product right?*) and validation (*are we building the right product?*).

When developing control and planning algorithms, verification corresponds to ensuring that the implemented algorithm follows its mathematical specification, e.g., that the probability of colliding with an obstacle is bounded to be less than 1 %. I would argue that this is the main purpose of my research, namely ensuring that we find the right mathematical model to verify certain desirable conditions of our robots.

However, validation is at least similarly important. It addresses whether these algorithms actually produce robot behaviors that meet the real-world safety requirements. For instance, a planner might be formally correct but still unsafe if its model fails to capture other environment dynamics such as sensor degradation or rare but critical failure modes. This distinction highlights a key challenge in safety related robotics. A system can be verified at the algorithmic level yet invalidated in practice when exposed to noisy sensors or unmodeled dynamics. In my research, although focusing more on the verification side, I am trying to validate my algorithms on robotics hardware to show that the initial design choice of picking a certain model has its validity in the real world.

2.2 Iterative Development and Continuous Testing

A further principle is that the development of robot control algorithms cannot be treated as a one-shot design effort but must instead follow an iterative cycle of development and testing. Robotic systems operate in environments that are inherently uncertain, and thus an algorithm that performs well in simulation may behave unexpectedly once deployed on hardware. Iteration provides a mechanism to close this gap over time.

In practice, this means that a control algorithm is first developed and evaluated in simulation where assumptions and design choices can be explored efficiently. Once the results are satisfying, the algorithm is deployed on physical robots under rather controlled conditions. Here, continuous testing is crucial since unexpected hardware effects such as sensor noise and environmental disturbances have often not been considered in simulation.

By repeating this cycle, i.e. simulate, deploy, test, refine, the control/planning algorithm improves in both robustness and often some assumptions have to be refined. Moreover, continuous testing enables evaluation across a wide range of scenarios rather than relying on a small set of benchmarks. This iterative process ensures that safety and performance cannot be guaranteed by design alone but must be demonstrated through hardware evaluation.

3 Guest-Lecture Principles

3.1 Levels of Abstraction in Requirements

One principle from the guest lecture was the importance of distinguishing between different levels of abstraction in requirements. These can be summarized as the context layer (*why*), the requirements layer (*what*) and the system layer (*how*). This separation is essential in order to avoid vague requirements and to ensure that design decisions can be traced back to the initial motivation.

In the context of robot motion planning, this principle can be applied as follows. At the context level the motivation is clear. Robots must operate safely and effectively in uncertain environments and, e.g., under no circumstances should do harm to a human. At the requirements level, this translates into capabilities such as collision avoidance under partial observability (due to limited sensing), or maintaining safe distances with high probability. Finally, at the system level, these requirements drive concrete design choices, such as the use of specific controller classes that allow us to handle the imposed constraints such as, e.g., *model predictive control* to name one. Maintaining this layered structure ensures that implementation details remain aligned with the underlying safety objectives and it provides a framework to reason about whether the control algorithms are addressing the intended problem.

3.2 Goal Modeling

Another principle that was covered is goal modeling. Goals capture desired properties of a system at different levels including usage goals (relevant to end users), system goals (relating to technical capabilities) and business goals (strategic). Goals can also be refined or shown to be in conflict which enables handling of trade-offs in the design process.

For my research, goal modeling helps me in reasoning about competing objectives. For example, a usage goal might be "the robot shall navigate safely among humans" while a system goal could be "the robot shall maintain collision probability below a specified threshold." These may in turn conflict with other goals common in robot navigation such as minimizing travel time or energy consumption. Goal modeling thus allows us to balance safety against efficiency. This could be for example achieved by picking a lower safety level that still results in navigation times that are acceptable for the user. Importantly, it also supports refinement of goals. High-level safety goals can be decomposed into measurable subgoals, such as limiting the variance of state estimates or reducing uncertainty through active sensing.

4 Data Scientists vs. Software Engineers

I mostly agree with the distinction between data scientists and software engineers in the book chapters. Data scientists are usually concerned with models and, as the name suggests, data while software engineers emphasize reliability and long-term maintenance of a code base. Both perspectives are essential, but they often lead to different priorities. In robotics, this divide is very visible. A model that achieves good accuracy in simulation on some benchmarks may still fail catastrophically in deployment because of unmodeled effects or simply because the benchmark does not capture the reality well enough. Oftentimes the engineer needs to fine tune models for their specific use case while the data scientist tries to keep things as general as possible.

I do not expect the roles to stay entirely separate. The book discusses T-shaped professionals, who are deeply skilled in one area but have enough breadth to collaborate across domains. I think this will become increasingly important. In my own research, this means again to ensure that the algorithm/model you develop does not only perform well on benchmarks but is also suitable for long-term deployment on a robot.

Overall, the difference between the two roles is real, but in practice everyone needs to have knowledge in both domains. Especially in robotics, success requires interdisciplinary teams where both mindsets are present to develop a product that actually works.

5 Paper Discussion

5.1 MLOps: Five Steps to Guide its Effective Implementation [2]

This paper introduces a five-step framework for adopting MLOps, namely (1) changing organizational culture, (2) preparing model development, (3) choosing appropriate tools, (4) automating pipelines and (5) continuously monitoring deployed models. The key contribution is to provide a structured way of managing the full ML lifecycle. This tries to bridge the gap between research prototypes and actual production systems. It is useful for software engineering of AI systems because it improves reliability and ensures that models remain adaptive under evolving data distributions (due to possible distribution shifts).

My research connects to these issues raised in the paper. Robot planners and controllers, especially those involving learned models, must cope with dynamic, uncertain environments (that are changing over time). Without lifecycle support, their performance can quickly degrade if some environment variable changes. The practices described in this paper can be directly used to ensure that planning/control algorithms remain safe and effective in long-term deployments.

A concrete example can be seen in a smart warehouse robotic system, where fleets of autonomous robots rely on ML models for navigation, and obstacle avoidance. MLOps practices would help ensure that these models are deployed consistently across many robots, monitored for performance shifts, and retrained when conditions change, for example when new layouts of the warehouse are introduced. Next, I argue how the proposed steps can be directly used to improve my algorithms and their deployment in the large scale project.

To align my work more closely with the five steps proposed in the paper, I would adapt my research as follows. First, regarding changing organizational culture, I would design my work to integrate easily with other subsystems, encouraging collaboration between planning, perception, and other teams. Second, for model preparation, I would emphasize data collection from robot trajectories and failures, creating datasets that directly support the training other models. Third, in choosing tools, I would package my planning algorithms within containerized environments and use version control so that results are reproducible across multiple robots. Fourth, in automating pipelines, I would link planning modules to software in the loop (SITL) workflows, ensuring that new versions of planners are automatically tested in simulations before reaching physical robots. Finally, for continuous monitoring, I would design planners that generate performance and safety metrics during operation, allowing early detection of distribution shifts and thus supporting automated retraining when needed.

5.2 Bringing Machine Learning Models Beyond the Experimental Stage with Explainable AI [3]

This paper examines how explainable AI (XAI) can help move ML models from experimental prototypes into real-world production systems. The authors present a case study at a Danish bank, where a trading model had been validated in research but never deployed. By introducing explanation techniques that made model predictions more transparent, domain experts (in finance) were able to understand and retrain models themselves without relying on data scientists. This enabled the successful integration of the model into a decision support tool. For software engineering, the key insight is that explainability can make ML systems transparent enough to be validated and improved by non-specialists.

In robotics, planning and control algorithms are often highly technical and mathematically complex, making it difficult for operators or non-experts to understand how decisions are made. This lack of transparency can limit the algorithms applicability, especially in safety-critical settings. Just as financial traders in the study used XAI to monitor model behavior, robotic operators could use explainability tools to better understand and supervise planning algorithms. For example, recent works in robot control explore the use of large language models to turn a non-experts language command into an objective function that is used in an optimization based planning framework [1]. Coming up with the right mathematical objective is often tedious and difficult for users.

In the aforementioned large scale warehouse system, ML models may be used for perception, while my research provides the planning and control layer. XAI techniques could be employed to make both perception and planning decisions more transparent. For exmaple, explanations could highlight which sensor inputs most influenced a navigation decision, or why the planner considered one route less safe than another. This would allow the non-expert staff to monitor robot decisions in real time and intervene when necessary. Together, explainability and risk-aware planning can form a system in which safety is strengthened by having a human in the loop that can understand the robot's decisions.

Adapting my research to align with this paper would mean going beyond providing only formal guarantees of safety. I could design planners that not only enforce risk constraints but also generate human-understandable explanations for their choices. For instance, alongside a planned path, LLMs could be leveraged to provide a summary of the risk trade-offs that led to that decision (e.g. "this path minimizes collision probability at the cost of a slightly longer travel time"). Such explanations would bridge the gap between mathematical rigor and practical usability. In the long run, coupling explainability with risk-aware planning could make robotic autonomy more transparent and thus easier to understand for non-expert operators. I believe that modules should be supported by language descriptions as well as visualizations to make the whole robotics stack more explainable.

6 Research Ethics & Synthesis Reflection

To select the two CAIN papers, I browsed the proceedings and read several options before choosing those that best connected to robotics, safety, and deployment. Titles were generally not misleading, so I simply skimmed some papers and checked if it makes sense to discuss them in the context of my research. Since my research does not directly include the training of ML models, I chose two that can be discussed in a broader context, namely general MLOps as well as explainability. To ensure originality, I tried to take the steps described in the papers and directly apply them to my research. I definitely found it easier to apply explainability to my research than the five steps of MLOps in [2] since explainability plays a major role in my phd.

7 Make-up Assignment: Group Discussions

7.1 Lecture 1

Q1: Do you agree with the general view that there are many other activities in AI/ML than the core algorithms and models/architectures?

I agree that AI/ML development involves much more than just algorithms. In my own research, I spend a lot of time thinking about planning methods, but deploying these on actual robots requires far more work than just the algorithm itself. Getting something to work in the real world is always an interdisciplinary project that involves people working on various subjects also outside of AI/ML. At the same time, I can see why research papers often focus narrowly on algorithms. It makes benchmarking cleaner and highlights fundamental contributions. But if we want robotics to be safe and reliable in practice, the involved engineering activities cannot be ignored.

Q2: The paper is now rather old, is the view it presents now more understood in the field or is there still too much focus on the core algorithms and models?

I think the issues described in the paper are widely acknowledged now. Industry has embraced pipelines and lifecycle management as core parts of ML engineering. In academia, though, there is still a strong emphasis on algorithmic novelty and improvements in accuracy on benchmark datasets are often what gets published. My own research leans more toward the algorithmic side, but I try to also consider the broader system context. For example, it is not enough to design an elegant planner, it must also be useful for downstream tasks.

Q3: Discuss some of the “ML system anti-patterns” identified in the paper.

Some of the anti-patterns from the paper are familiar to me. In robotics, ”pipeline jungles” often appear when combining multiple modules (perception, control, planning, etc.), each with its own scripts and configurations and potentially different naming conventions. ”Dead experimental codepaths” also arise sometimes when testing out many algorithms variations with different parameter settings without proper cleanup. They often accumulate and become a burden when features need to be removed or analyzed in a later stage of a project.

7.2 Lecture 2

Q1: Would you say that typical AI/ML projects are good at testing and quality assurance?

In think most AI/ML projects are not particularly strong at testing or quality assurance. In research especially, the emphasis tends to be on proving concepts or publishing benchmark results, often with code that is ad hoc and not maintained beyond the experiment. Evaluation is usually done against static test sets, which rarely capture the variability and edge cases of real-world deployments. In robotics, this gap is especially critical because models and controllers that appear reliable in simulation may fail once deployed on hardware.

Q2: In your view, is testing in AI/ML important or is it secondary to other activities/tasks?

For robotics and especially safety-critical domains, I would argue testing is not secondary at all but one of the most important activities. A model or planner that performs well on average can still produce catastrophic outcomes if rare failures are not caught. For example, a navigation system that fails only in unusual lighting conditions could still cause accidents. Testing helps to reveal such edge cases. While paper deadlines can push QA aside, in long-term deployments we should definitely put enough efforts into testing.

Q3: Do you think there are good ideas from testing of "traditional" software that AI/ML can learn from or is it very different for AI/ML?

Many testing principles carry over to AI/ML, such as regression tests and system-level checks. However, the probabilistic nature of models means verification must be adapted to the stochastic setting. Still, large parts of an ML pipeline are traditional software and can be tested with established methods.

8 Lecture 3

Q1: Why do you think teams neglect these trustworthiness practices?

Teams often neglect practices like fairness or privacy because they are difficult to measure performance metrics. In both academia and industry, accuracy gains are easier to measure while trust-related outcomes are less visible. In my own research, algorithmic improvements usually take priority while documenting risks or finding edge cases is harder to justify.

Q2: How would you address these barriers in your own team or company?

I would say reproducibility matters a lot. Robotics experiments are hard to replicate, since hardware and environments (for example lighting conditions) might differ across labs. Without reproducibility, it is difficult to know whether improvements are real or just artifacts of specific conditions. Practices such as versioning code or keeping transparent logs should therefore be adopted first.

Q3: Given that the study is now a few years old and essentially "pre-LLM era" what are the most glaring problems and missing practices or effects you can think of?

LLMs introduce some risks, especially when they generate code. In any field, relying on such outputs without careful review could lead to bugs or unsafe behaviors without actually knowing why. Modern practice should therefore include explicit auditing of AI-generated code and clear indication when LLMs have been used in projects, so that additional code reviews can be done.

9 Lecture 4

Q1: Is there a need for AI Engineering which relates to core/fundamental/present-day AI/ML as Software Engineering relates to Computer Science?

I believe there is a need for AI engineering, similar to how software engineering relates to computer science. AI/ML systems in practice can be messy as for example stitching together multiple learned modules in a robotics stack. Without systematic engineering discipline, these systems can fail in subtle but costly ways if not properly maintained. For robotics in particular, safe deployment requires much more than novel algorithms. It requires processes for testing, monitoring and adapting models over time.

Q2: If we assume there is some need for AI Engineering, does it need to be its own thing or can it mainly be based on Software Engineering and existing knowledge there (and in related fields)?

AI Engineering does not have to start from scratch. Many principles from Software Engineering remain the same. But also AI systems bring unique challenges that go beyond traditional software like stochasticity, continuous learning or ethical considerations such as fairness. Therefore, I see AI Engineering as an extension of Software Engineering but with its own focus on data and trustworthiness.

Q3: Are there some unique AI/ML characteristics which makes it likely AI Engineering will need to be somewhat different from Software Engineering?

What makes AI different is that correctness cannot always be defined as in traditional software. Models are sometimes probabilistic and sometimes even unverifiable because they are treated like a blackbox. I think, instead of rigorous verification we will move to something similar as in medicine where empirical studies are conducted in order to estimate the effectiveness of AI/ML modules. In robotics, this could mean that robots are deployed in a test environment over a longer time and, if they act as intended in let's say 99 % of the time, they can be deployed commercially. Of course the risk level heavily depends on the application domain.

References

[1] Diego Martinez-Baselga, Oscar de Groot, Luzia Knoedler, Javier Alonso-Mora, Luis Riazuelo, and Luis Montano. Hey robot! personalizing robot

navigation through model predictive control with a large language model. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2025.

[2] Beatriz MA Matsui and Denise H Goya. Mlops: five steps to guide its effective implementation. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pages 33–34, 2022.

[3] Niels With Mikkelsen, Lasse Steen Pedersen, Mansoor Hussain, Victor Foged, and Ekkart Kindler. Bringing machine learning models beyond the experimental stage with explainable ai. In *4th International Conference on AI Engineering–Software Engineering for AI (CAIN)*, pages 119–129. IEEE, 2025.