An Education Model of Reasonable and Good-Faith Effort for Autonomous Systems

Cindy M. Grimm[1], **Woodrow Hartzog**[2], **and William D. Smart**[1]
[1]Oregon State University
Corvallis, OR 97331
United States
[2]Northeastern University
Boston, MA 02115
United States
Email: {cindy.grimm,bill.smart}@oregonstate.edu, w.hartzog@northeastern.edu

Abstract

In this paper we propose a framework for conceptualizing and demonstrating a good-faith effort when developing autonomous systems. The framework addresses two fundamental problems facing autonomous systems: (1) the disconnect between human-mental models and machine-based sensors and algorithms; and (2) unpredictability in complex systems. We address these problems using a mix of *education* — explicitly delineating the mapping between human concepts and their machine equivalents in a structured manner — and *data sampling* with expected ranges as a testing mechanism.

Introduction

In this paper we argue that the difficulty in demonstrating reasonable good-faith effort to ensure correct behavior when developing autonomous systems is a *human* problem, not a problem with the underlying technology. In particular, it is the inherent impreciseness in human language and mental models — and a mis-match between these and the underlying technology — that is at the core of the challenge in developing reliable autonomous systems. We believe that we cannot even begin to define what constitutes an "ethical" system without first addressing this problem.

We propose a semi-formal language framework — based on demonstrating reasonable effort through an educational model — that begins to address this problem from the *human* side. Specifically, our framework aims to reduce the impreciseness in human language by simultaneously making human-level specifications more structured (and precise) and by directly tying them to the underlying statistics, sensors and algorithms (the "education" component).

Mapping a complex autonomous system to a semi-formal language understandable by humans all in one go would be daunting. For this reason our framework starts from the bottom (simple nouns and verbs) and works up to more complex statements. Our levels are chosen to make it easier to map human terms to technical descriptions at the same level.

We assume here that the developer of the autonomous system is ethical and wants to to design a system that is safe, reliable, and does what it is supposed to do, to the best of their abilities. We look at ethics through the lens of the legal concept of fault (Smart, Grimm, and Hartzog 2017), drawing on the law's notion of reasonable effort to establish protocols and guidelines for ensuring techniques are not used blindly. Our goal is to allow developers to explicitly demonstrate their efforts — what their assumptions are, efforts they went to to ensure safety, and their expected behavior of the system.

Our discussion will be agnostic to the specific AI/Machine Learning algorithms and sensor technology used. We largely treat these elements as black boxes with inputs and outputs that can be measured, sampled, and statistically characterized. We note, however, that algorithms that have transparency and explainability would make it much easier to establish the mapping from human terms to technology, and might be particularly well-suited for novice developers.

Related work

There is a rapidly growing interest in explainable AI (XAI) (Langley et al. 2017; Biran and Cotton 2017) fueled, in part, by the recent successes to techniques like deep neural networks and other "black box" AI techniques. XAI aims to make AI techniques that are currently opaque to humans more transparent, but providing human-understandable explanations of the decisions that the AI system made. While this work is very much in the same spirit as the work presented in this paper, it is complementary to it. Explainable AI systems will make the work described in this paper easier, but it will not supplant it. We still need to do the work of educating the humans involved in the design and deployment of autonomous systems about their capabilities and responses, and help them make sure nothing is lost in translation from sloppy human descriptions to precise computer software.

The work in this paper draws from the idea of formal specification in software engineering (Sommerville 2015, chapter 27), which attempts to remove the ambiguity in software specifications with a similar intent to our attempts to remove it from autonomous systems. Although there has been a considerable amount of work in using formal languages in robotics (Winkler et al. 2011; Simmons and Apfelbaum 1998; Klavins 2003; Coste-Maniere and Turro 1997; Risler and von Stryk 2008, for example), the vast majority of this has been to ensure correct behavior. Our goal is different: we want to provide evidence that the humans involved

in the design, implementation, and deployment of the system did their very best to get things right.[1]

We also draw inspiration from the field of software testing (Ammann and Offutt 2016; Crispin and Gregory 2009) and, in particular, from the idea of unit tests (Osherove 2013), integration tests, and regression tests. The main idea that we take from this prior work is that it is not enough to write software (to control an autonomous system); we must also write formal tests to verify that the software does what we expect it to (and causes the behaviors that we expect). Not only do we need to explicitly specify the tests that we will perform, we must also explicitly declare the input data (sensor readings and environments, simulated or real) that the software will be tested on. Unit tests validate the behavior of single components in the system (such as a face detector). Integration tests validate the behavior of the whole system (which is a collection of the individual components). Regression tests ensure that the overall system behavior does not change when we make changes to the individual components, or how they are composed. The data sets define the operating conditions under which we have tested the system, and under which we expect it to be used when it is deployed.

Finally, we draw lessons from legal regimes that attribute fault and, ultimately, financial liability, to actors that unreasonably fail to protect against forseeable risks of harm when designing complext technologies (Vladeck 2014). Tort law provides a private cause of action against those who negligently cause harm to another. In assessing culpability, courts look to forseeable risk and how burdensome it is to mitigate or protect against those risks (American Law Institute 2010). If the risk is greater than the burden of precation against those risks, then failure to take those precautions will be seen as negligent. In other words, courts demand reasonable behavior, a corollary to "good faith efforts," from people who build products and release them into the world. This means anticipating reasonable misuse, misunderstandings, and harm likely caused by other people while using a product (Hogan 1994). One way to act reasonably is to remedy information asymmetries of other relevant parties through education. Education efforts play an important role in law and policy and could also form a bedrock ethic and rule for parties that create, deploy, and use autonomous systems.

Framework

In this section we broadly outline a framework for shared communication and education for autonomous systems. We first define the *stakeholders* involved in a deployed autonomous system, and what their roles are. Next, we define a *language framework* for communication between these stakeholders, using the language framework to establish what *types* of information need to be communicated between the different stakeholders. We frame *Good Faith Effort* as the development of "unit tests" for each of the elements in the language framework. Finally, we break failures in autonomous systems into four types that can be explained within this overall framework.

[1]Of course, we also care about correct behavior in our robot systems. However that is not the focus of this paper.

Stakeholders

Our stakeholder descriptions are meant to capture the three different roles that arise when developing an end-to-end application. These are broad categories defined by goals and skill sets.

1. **End-users**: These are the people *using* a specific application developed based on the technology. They are experts in their domain area, but are not expected to have any deep understanding of the underlying technology. They may be required to undergo training to use the technology effectively depending on the difficulty of using the tool/technology. Their goal is to accomplish their domain-specific task.
2. **Procures or merchants**: These are people who put together the application-specific use of the underlying, low-level technology. They understand the end-use case and underlying low-level technology well enough to write the application, but are not necessarily domain experts or low-level technology experts. Their role is to bridge the gap between end-user needs and abilities and the available low-level technologies that can be used to meet those needs. Their goal is to define end-user needs than turn that into design specs that can be realized using existing technology.
3. **Developers**: These are the people developing the low-level technology components (the actual algorithms and sensors). They focus on improving the algorithms and sensors, but are (largely) agnostic as to the use they will be put to. They might use one (or more) specific application domains as test cases to drive development, but are not experts on application-specific needs. Their goal is to improve both the efficiency of their algorithms or sensors and their applicability.

Obviously, these are not always clearly-defined categories, and an individual (or group) may straddle more than one category, but for our purposes the point is to emphasize the different goals for each category. In our framework end-users work with procurers, and procurers work with developers, but end-users do not directly interact with developers. We focus here on the second case — procurers working with developers — because techniques for defining the end-user to procurer relationship have been well-studied (Sedlmair, Meyer, and Munzner 2012).

Language framework

The point of our language framework is to make explicit the gap between human representations and machine ones. We are not proposing a formal predicate language here, but a framework that lies between that and open-ended human speech. The goal is to *educate* the developer on the specific needs of the procurer.

We first define the elements of the framework and then the details of what this means for the stakeholder relationship. The procurers initially specify their goals to the developers with the following four elements:

1. **Syntax (nouns, verbs, modifiers):** Nouns are the list of objects or items in the real world that are necessary for the

stakeholder to accomplish their task. These are items that must be recognized or manipulated (or both). Verbs are actions that operate on the nouns, again, either actions that must be recognized and/or actions that need to be taken. Modifiers are attributes of nouns or verbs that matter to the stakeholder.

2. **Semantics (recognition, action):** These are explicit expressions of relationships — built out of the nouns, verbs, and modifiers — that the stakeholder requires. They can consist of recognition relationships (e.g., is `noun + modifier-value + verb` happening?) and action relationships (e.g., `action-verb apply-to-noun`).
3. **Assumptions (modifiers, nouns, likelihoods of semantic events):** This is a bracketing of the expected values for the above elements. For modifiers, this is expected values the modifiers take on (e.g., hot maps to $35+$ degrees Celsius). For nouns, this can be a list of exemplars for that noun or brackets (e.g., boxes are expected to be of a certain size and made of cardboard). For semantic events this is an estimate of how likely a recognition event is, and preferences for action relationships.
4. **Scenarios:** These are end-to-end exemplars of the expected behavior of the system (use-cases in traditional human-computer interaction). They include a description of the environment (what nouns are where and with what modifiers/verbs) and a sequence of recognition and action events.

Armed with this semi-formal problem description, the developers are responsible for *educating* the procurers on how the underlying sensors and algorithms will be mapped to elements one and two. Specifically, for each noun and recognition verb, the developer must clearly disclose what sensors are being used — and how — to recognize that noun/activity. Similarly, the developers must specify how action verbs are mapped to actuators and sensors.

Good-faith effort as unit tests

At this point, good-faith effort can be demonstrated by developing data and tests for each of the syntactic terms given by the procurer (we call these unit tests[2]. Note that each term now has both a human-level and an algorithm/sensor level description; this helps with defining tests that cover both aspects.

Building on the unit tests, the procurer then provides specific semantic-level tests, with variation drawn from the given assumptions. For these tests, the developer can separate "syntax" correctness from semantic, focusing on the logic.

The assumptions and scenarios, together, document the expected use of the overall system. Provided the procurer has adequately captured the end-user's needs and expected use, this serves as documentation of the conditions the system were tested under.

[2]This is an abuse of the formal definition of unit test, but the intent is similar.

Failure cases

Using the above framework, we can break autonomous system failures into four categories. We include here failures at both the end-user to procurer interface and the procurer to developer.

1. **Syntactic failures** occur when there is a mismatch between the precision of artificial sensors and the robustness of human senses. A syntactic failure indicates the need for an additional "unit test".
2. **Semantic failures** occur when the human-articulated goals and intentions for autonomous systems are not translated correctly into logic/algorithms. These can arise either because the procurer failed to include that semantic relationship in the original specification, or because of an algorithmic/logic failure. This indicates that either the specification needs to be updated, there is an inconsistency in the specification, or the inference/logic algorithm is incorrect.
3. **Testing failures** occur when a necessary assumption or scenario test is simply missing from the test set. This is largely the developer's fault, but could also be the fault of the procurer for not fully articulating all of the desired use cases. Testing failures can also occur when the necessary syntactic or semantic tests are not conducted appropriately or are otherwise invalid.
4. **Warning failures** occur when users are not appropriately made aware of avoidable problems caused by the unpredictability of systems (the developer to the procurer or the procurer to the end-user). Warning failures are, in many ways, the inverse of semantic failures. Warning failures flow from developer to procurer to end-user, while semantic failures flow from procurer to developer.

Example

In this section we walk through an example of a security guard robot patrolling a factory. The robot patrols the inside of the factory and raises an alert if it finds a person (or persons) in the factory. The exception to this is the security guard (who has a pass).

The first step is for the procurer to define the syntax of the system (left hand of Table 1). For each element, the developer must specify how this will be implemented. At this point, the developer and procurer can work together to refine the definitions and assumptions. Take equipment: The procurer would be responsible for listing types of equipment that are expected to be found in the warehouse (boxes, tables/benches, fork lifts). The developer would be responsible for listing known failures for lasers (which are used to identify equipment): E.g., Tall, skinny objects and bright daylight. The procurer would then either modify their assumptions list (all tall skinny objects must be in containers, all tables must have drapes/backing, no full-sunlight lit areas) or the developer might adjust their equipment detection, for example, adding a vertical laser scanner.

The developer can then develop unit tests, including tests designed to capture potential failure tests (eg, flagging if the overall detected light levels indicate the laser will fail). Note

Noun/verb/mod	Implementation
Person	Line of sight infra red camera
	Person-shaped blob
Equipment	Knee-height laser-scan detects obstacle
	Not wall
Space	Defined by blue-print
	Doors, windows marked
Security Pass	RFID tag
	Detectable within ten feet
Patrol	Visit all physically accessible places
	Rate of n sq meters covered per hour
	Minimal back-tracking
Confront	Move within 3 feet of detected person
	Issue request for pass
Raise alarm	Send wireless signal
Detect	Laser/camera registers new object
	Persistent (seen from multiple locations)
Verify pass	Known RFID tag read by scanner
Person pose	Vertical, prone, sitting
Lighting	Dim, lights on

Table 1: Nouns and verbs for a robot patrol application.

Recognition	Frequency/likelihood
Detect Equipment Change	Always/low likelihood
Detect Person	Always/medium likelihood
Action	Condition
Patrol Space	Always
Confront Person	After detect person
Verify Pass	After Confront Person
Raise Alarm	After Confront Person and Verify Pass failed

Table 2: Semantic statements and their likelihoods.

that the unit tests do not have to be all or nothing; for example, a people detector may only need to have a false positive and a false negative rate within given bound.

The next step is for the procurer to specify the semantics (Table 2. Each of these statements require a simple unit test to verify that the appropriate sequence of behaviors is correctly implemented, as well as a more exhaustive set of tests to determine if actions happen that shouldn't.

At this point a failure in the semantics becomes apparent, because the robot will continuously detect and confront a person in the space. The procurer would have to add another verb — track — to address this problem. A similar semantic issue arises with *two* detected people; this would require an addition to the semantics and an additional modifier for number of people.

The next step is to add in assumptions, such as (modifier) that a person might be trying to crouch down to hide, or sitting in a chair. This assumption might be addressed by adding in more unit tests. Other assumptions might include: The security guard only patrols once an hour and always turns the lights on. These might induce additional semantic-level statements and corresponding tests, such as explicitly defining what the robot should do if the security guard patrols more often or doesn't turn the light on. These assumptions might also contain environmental expectations, such as that the robot can view every part of the space. Combined with a bound on equipment size, this can lead to a negative test, where the robot raises an alarm if there is a sufficiently large portion of the space it can't view.

The final step in this framework is example end-to-end scenarios (described in the language framework above), complete with sample environments and expected variations on modifiers. For example, a patrol use-case might be informally described as: Define what it means for a robot to successfully **patrol** a **space**, taking into account the warehouse **space**, size, and **equipment** occupancy. The sample environments make this concrete by defining: Bounds on the size of the overall space, number of corners allowed in the walls, ranges for number and type of equipment and their spacing, how often equipment is added/deleted/moved, and minimum/maximum spacing between equipment. The measure of success is defined by: The robot must be able to visit every part of the space within a specified amount of time and detect new equipment with a given probability, and trigger no more than one false alarm for every n patrols.

The procurer might use the same environmental definition as above for a detect-intruder use-case. In this case, the procurer would define the use-case as: Generate an **alarm** when a single intruder (**person**) enters the space, attempts to steal an object, then leaves. The procurer must specify bounds on where the person enters and leaves the building, how fast they travel within the space, and where in the space the item is located. Success is defined as an 80% detection rate perhaps scaled by overall warehouse size.

By using the given environmental variations — and the false positive/false negative rates of the unit tests — the developer can largely run these tests in simulation. While simulation is not ideal, testing with uniform coverage of expected conditions is better than nothing.

Test and training data sets

In traditional manufacturing and design (particularly where safety is an issue) the industry develops standards — often based on measurable physical properties — for managing liability and risk. To some extent these standards are a codified history of what *not* to do as well as a list of good practices. Cribs and car seats are an extreme example of this — the standards are updated very frequently, usually with edge cases that just haven't arisen in the past.

For physical systems it is relatively easy to develop measurable standards (eg, the tensile strength of rebar should be x in a wall of height y). These standards rely on the predictability of intermediate values in those physical systems — this makes it easier to simply bracket desirable and undesirable values. Unfortunately, autonomous systems lack this predictability; giving it example a and c, with b lying somewhere in between, usually doesn't guarantee that the response to b also lies somewhere in between the responses for a and c.

We argue here that test and training data sets built in the proposed manner could serve as a standards mechanism for autonomous systems — provided they are documented and

shared openly, and updated with new cases as they arise. Labeling the test and training sets with the language framework in Section would identify the intent of the tests. As failure edge cases are discovered they would be added to the test set. Ensuring that the components (particularly at the syntactic level) passed the tests would document a good-faith effort at meeting those standards.

Bias

Bias in training and test sets is notoriously difficult to detect — and algorithms will learn them if they are present in the data set. One approach to dealing with biases in, for example, hiring or evaluation practices, is to make a list of desirable attributes/skills, describe how they are required for the job, then explicitly map evaluation criteria to the attributes/skills (in Science and Engineering 2012). Our approach is similar to this: Write down the task, how the sensor will be used to do the task, and demonstrate how the sensors will be used to accomplish the task. In parallel, define the data set attributes that will be used to verify the system is performing as expected.

Discussion

Nothing we have outlined here is necessarily deep or surprising — it is essentially lifting best-practices from software engineering and adapting it to a situation where ground truth must be modeled with statistical variation and sampling (the unit tests).

What is novel is the insistence that every human statement be paired with its corresponding computational/sensor equivalent, at the appropriate level (nouns/verbs to unit tests, assumptions to mid-level tests, environment/use-cases to explicit variations/statistical expectations). This is the *education* component of our argument. Procurers must educate developers on the behavior they want and nail down the expected operating conditions. Developers must communicate how the sensors and algorithms are implementing the desired behaviors.

The proposed framework will not magically "fix" autonomous systems. Any reasonably complex system operating in the real world is going to behave unpredictably at some point. The goal with this framework is to make it easier to identify at what level the failure occurred, then introduce additional tests/modify the framework to address that failure. Over time, these tests can be developed into standards for common low-level behaviors in autonomous systems — just like we currently have standards for tensile strength in steel.

References

American Law Institute. 2010. Restatement (third) of torts: Physical & emotional harm, §3.

Ammann, P., and Offutt, J. 2016. *Introduction to Software Testing*. Cambridge, UK: Cambridge University Press, second edition.

Biran, O., and Cotton, C. 2017. Explanation and justification in machine learning: A survey. In *Proceedings of the IJCAI Workshop on Explainable Artificial Intelligence (XAI)*.

Coste-Maniere, E., and Turro, N. 1997. The Maestro language and its environment: Specification, validation and control of robotic missions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, 836–841.

Crispin, L., and Gregory, J. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional.

Hogan, III, R. B. 1994. The crashworthiness doctrine. *American Journal of Trial Advocacy* 18:37.

in Science, W., and Engineering. 2012. Reviewing applicants: Research on bias and assumptions.

Klavins, E. 2003. A formal model of a multi-robot control and communication task. In *Proceedings of the Conference on Decision and Control (CDC)*.

Langley, P.; Meadows, B.; Sridharan, M.; and Choi, D. 2017. Explainable agency for intelligent autonomous systems. In *Proceedings of the Twenty-Ninth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI)*, 4762–4764.

Osherove, R. 2013. *The Art of Unit Testing*. Manning Publications, second edition.

Risler, M., and von Stryk, O. 2008. Formal behavior specification of multi-robot systems using hierarchical state machines in XABSL. In *Proceedings of the AAMAS Workshop on Formal Models and Methods for Multi-Robot Systems*.

Sedlmair, M.; Meyer, M.; and Munzner, T. 2012. Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics* TBD.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, 1931–1937.

Smart, W. D.; Grimm, C. M.; and Hartzog, W. 2017. An education theory of fault for autonomous systems. In *Proceedings of We Robot*.

Sommerville, I. 2015. *Software Engineering*. Pearson, 9th edition.

Vladeck, D. C. 2014. Machines without principals: Liability rules and artificial *Washington Law Review* 89:117–150.

Winkler, L.; Kettler, A.; Szymanski, M.; and Wörn, H. 2011. The Robot Formation Language: A formal description of formations or collective robots. In *Proceedings of the IEEE Symposium on Swarm Intelligence (SIS)*.