Explainability and Knowledge Representation in Robotics: The Green Button Challenge

Freek Stulp*[0000−0001−9555−9517], Adrian S. Bauer*[0000−0002−1171−4709], Samuel Bustamante*[0000−0002−7923−8307], Florian S. Lay*[0000−0002−5706−3278], Peter Schmaus*[0000−0002−6639−0967], DLR-RMC Green Button Teams*, and Daniel Leidner*[0000−0001−5091−7122]

*German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC), Münchner Str. 20, 82234 Weßling, Germany
{freek.stulp, firstname.lastname}@dlr.de
http://www.dlr.de/rmc

1 Introduction

Deep learning has been one of the driving factors behind the current interest in explainability. This is because the knowledge encoded in a trained deep network is, due to its distributed nature, only available implicitly. Several methods exist for the post-hoc interpretation of such networks in terms of explicit rules or examples [11]. These rules can then be interpreted and understood by humans. Such approaches may be considered *Freudian*, as implicit (*subconcious*) knowledge in the model (*the patient*) is not accessible to the model itself, and must therefore be elucidated post-hoc by an outside observer (*the psychoanalist*, e.g. Freud).

Whereas many low-level controllers and almost all low-level perception modules in robotics rely on data-driven methods, many higher-level modules such as semantic scene understanding, task planning, and human-robot interaction require explicit representations of knowledge. *How* to recognize a spoon may be done most effectively implemented with deep learning. But we believe knowing *what* a spoon is, and especially *what* its purpose is, should be represented explicitly, i.e. symbolically. In this case, explainabilty corresponds to the mapping of these internal but explicit symbols to a language that humans can interpret and understand [3]. This task, which one could denote *Wittgensteinian Explainability* is, in principle, easier than post-hoc explanations of implicit knowledge, as language itself consists of symbols[1].

We believe the importance of this form of explainability has been underestimated, and is of particular importance to robotics. This holds especially for robots that act in human-made environments, where most objects are made with a specific purpose in mind which the robot must know about.

The main aim of this paper is not to present state-of-the-art in automated reasoning or description logics. Rather this papers presents a proof-of-concept for explainability in robotics, based on translating internal symbolic representations of plans and actions to natural language. The experiences gathered in achieving

[1] Proposition 3.343 of Wittgenstein's *Tractatus Logico-Philosophicus* (1921): "Definitions are rules for the translation of one language into another. Every correct symbolism must be translatable into every other according to such rules."

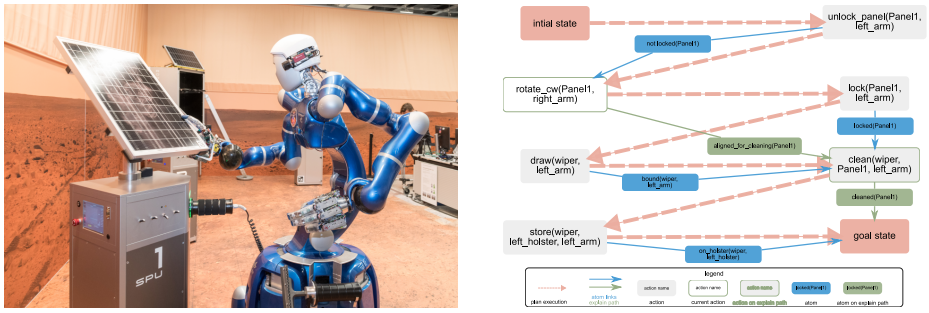

Fig. 1. Left: Justin aligning the solar panel prior to cleaning, during teleoperation from the International Space Station (ISS). Right: Example of an explanatory graph, enabling Justin to explain what it is doing and why.

this during our "Green Button Challenge" (to be explained in the next section) lead us to pose the following theses as a basis of discussion for XLoKR20:

• In robotics, post-hoc Freudian Explainability may be necessary to make implicit knowledge in learned (deep) models explicit. But Wittgensteinian Explainability should always be preferred, as the internal explicit representation corresponds more closely to the structure used for explanation (natural language). • We argue that robots thus require explicit (symbolic) knowledge representations, and stronger ties between the Knowledge Representation (KR) and Robotics communities should be formed. We hope that explainability in robotics can serve as a broad source of inspiration for the KR community. • What is known (by humans), should not be learned (by robots). Rather, methods must be developed which enable humans to transfer their knowledge to robots. • Robots should always be able to explain what they are doing and why, in natural language.

Next, we describe the Green Button Challenge followed by one concrete implementation of explainability on the robot Rollin' Justin.

2 The Green Button Challenge

To avoid robots from inflicting unintentional harm, they are equipped with an emergency stop button. This is a red button that, when pushed, stops the robot. The Green Button Challenge is, in a nutshell: "Provide your robot with a green button. When pressed, the robot explains, in spoken natural language, what it is doing. When pressed again, the robot also explains why." [14].

While the physical green button itself and speaking robots are not essential from a scientific point of view, they symbolize something important. And it is that robotic behavior should be explainable to humans at all times (i.e. at the press of a button), and that not only robotics experts should be able to understand these explanations (i.e. natural language, not Planning Domain Definition Language (PDDL) or first-order logic). We believe that this is a key element in human-robot interaction. For instance, it has been shown that Wittgensteinian-like Explainability fosters human trust in robots, more than other explainability modalities [4]. Moreover, the button enables explanations in real time, which is

also a factor in human trust [4]. Needless to say, system trust is a cornerstone of device acceptance.

There are many domains where the robotics community could benefit from well-grounded explainable systems, some of which we explored in the Green Button Challenge:

• **Space assistance**: as astronauts are exposed to dangerous environments where precision is a must [10], they will want to control what the robot is doing and what it will do next. • **Future manufacturing**: autonomous industrial robots are becoming more flexible (see [12]). This creates new scenarios of human collaboration in manufacturing. Like with a human colleague, communication and trust should be built to ensure assertive collaboration. • **Household and care robotics**: large parts of the general public remain skeptical towards robots. In elderly care facilities, there has been reported a need for *"meaningful communication abilities as well as cues that enhance the predictability of [the robot's] behavior"* [6].

We are currently initiating actions to make the Green Button Challenge a global public outreach initiative. But in this paper, the challenge refers to an internal competition at our institute, with the intention of determining how amenable different robot programming approaches are for explainability. In the next section we describe the approach of the humanoid space assistant Rollin' Justin, winner of the competition held in January 2020.

3 PDDL-Explainability on the Robot Rollin' Justin

The implementation of a green button on Rollin' Justin [2] is strongly tied to the planning system in use on Justin. Thus, we first provide some insights into how action plans are generated on the robot before we present how we enable the robot to use its knowledge about actions to generate explanations of their role in action plans.

Task and Motion Planning on Rollin' Justin. Rollin' Justin employs an integrated task and motion planning approach that is presented in detail in [9]. Core to all planning is the description of actions in form of Action Templates (ATs). Their role is to store and provide information about actions both on a symbolic and a geometric level. In an object-centric manner, they are attached to objects or object types and are inherited by deriving objects.

Parameters, preconditions, and effects of an action are stored in the *symbolic header* of ATs. We employ PDDL [5] for describing effects and preconditions and use the fast downward planner [7] for planning in action space. As we consider a deterministic environment, both preconditions and effects are lists of conjuncted, potentially negated, atoms. Goals are defined in the same way, e.g. `activated SPU1` for Smart Payload Unit 1 (SPU1) being activated.

The *geometric body* of ATs translates the abstract description from the header to robot movements. It is, thus, not relevant for PDDL-explainability.

Chaining Action Sequences Through Atoms. For the following consider that the robot created a plan $\mathcal{P} = (a_1, \dots a_n)$ to achieve a given set of goal atoms $\mathcal{G}$. All actions a_i are fully specified, i.e. their parameters are resolved. While the

robot executes action $a_j \in \mathcal{P}$, the user presses the green button and the robot has to explain what it is doing and why.

The answer to the *what* question is straightforward as the robot knows the action name of a_j and reports it together with its parameters. Reporting *why* this action is part of the plan is more advanced, and our implementation builds on previous work on causal link explanations [8,15,13,1]. As mentioned above, the preconditions pre_{a_i} and effects eff_{a_i} of action a_i consist each of a set of atoms p_k. Intuitively action a_j is part of $\mathcal{P}$ because either a) one of its effects is a precondition of a later action:

$$\exists p_x \in \text{eff}_{a_j} \mid p_x \in \text{pre}_{a_l}, j < l \leq n \tag{1}$$

or b) one of its effects is part of the goal state $\mathcal{G}$:

$$\exists p_x \in \text{eff}_{a_j} \mid p_x \in \mathcal{G} \tag{2}$$

Using this observation, we create an ordered graph for every plan. The nodes in this graph are formed by actions of the plan, links are formed by atoms that fulfill eq. (1) or eq. (2), and the order is determined by the order of actions in the plan. However one would be ill-advised to simply create a link from an action to all later actions that share an atom in this way. An atom can potentially be *consumed* and then re-created by another action in between. Instead, we designed our algorithm to proceed its way backwards through the plan. This is, we start at the goal state and create a link for each atom to the latest action that produces this atom. Next, we proceed with the preconditions of the last action a_n and work our way back to a_1 as displayed in algorithm 1. The advantage of advancing backwards is that every precondition of an action is linked to the latest previous action that created it. Thus, the effect of an action can be linked to a precondition of one or more other actions, but each precondition links to at most one effect. If a precondition or goal atom does not link to an action effect, it was already fulfilled initially.

The robot answers *why* questions based on this graph. On the next button press, after having answered *what* for a_j, the robot selects a link from the outgoing links of a_j via atom p to action a_k and replies that it executes a_j in order to achieve atom p. Next, it reports that it aims to achieve p to be able to execute a_k. From there on the procedure repeats. This procedure is guaranteed to reach the goal state because a) it approaches the goal state with every step due to the order of the graph and b) there are no dead ends since an action would not be part of the plan if it did not contribute to it in any way.

But which outgoing link to select? We decided to select the link connected to the first effect because we typically consider this to be the desired main effect of the action. The following effects are seen as side-effects. Other choices are to also select the nearest or furthest reaching link, thus going through the plan in many small or fewer big steps.

Exemplary Results in the SOLEX environment. We demonstrate the result of our implementation on the SOLEX proving ground [10], an environment created for evaluating shared control concepts together with astronauts on board the International Space Station. The environment resembles a solar farm on Mars and contains, among other items, three Smart Payload Units (SPUs) that are

equipped with solar panels or an antenna. In our example Justin has to clean a solar panel. To do so, it must unlock the panel, rotate it (see fig. 1, left), lock it again, grasp the wiper, and clean the SPU.

The output created can be explained based on the graph on the right in fig. 1. Assume the robot is executing action `rotate_cw Panel1 right_arm` and the button is pressed. The robot answers that it executes `rotate_cw Panel1 right arm` (*what*). On the next button press, the users wants to know why Justin is executing this action, and Justin answers that it tries to achieve atom `aligned_for_cleaning Panel1`, then execute action `clean wiper Panel1 left_arm`, and finally achieve the atom `cleaned Panel1`. If now the user clicks again, Justin's reasoning reached its end and it bluntly admits that with the words: "I don't know. Nobody told me".

References

1. Bercher, P., Biundo, S., Geier, T., Hoernle, T., Nothdurft, F., Richter, F., , Schattenberg., B.: Plan, repair, execute, explain – how planning helps to assemble your home theater. In: ICAPS (2014)
2. Borst, C. et al.: Rollin' Justin - Mobile platform with variable base. In: ICRA Proc. IEEE Int. Conf. Robotics and Automation (ICRA). (2009).
3. Chakraborti, T., Sreedharan, S., Kambhampati, S.: The emerging landscape of explainable AI planning and decision making. In: IJCAI (2020)
4. Edmonds, M., Gao, F., Liu, H., Xie, X., Qi, S., Rothrock, B., Zhu, Y., Wu, Y.N., Lu, H., Zhu, S.C.: A tale of two explanations: Enhancing human trust by explaining robot behavior. Science Robotics **4**(37) (2019).
5. Ghallab, M., Howe, A., Christianson, D., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - The Planning Domain Definition Language. AIPS98 Plan. Comm. **78**(4), 1–27 (Aug 1998)
6. Hebesberger, D., Koertner, T., Gisinger, C., Pripfl, J.: A Long-Term Autonomous Robot at a Care Hospital: A Mixed Methods Study on Social Acceptance and Experiences of Staff and Older Adults. IJSR **9**(3), (2017).
7. Helmert, M.: The Fast Downward Planning System. JAIR **26**, 191–246 (Jul 2006).
8. Kambhampati, S.: Mapping and retrieval during plan reuse: A validation structure based approach. In: AAAI (1990)
9. Leidner, D.: Cognitive reasoning for compliant robot manipulation. Springer (2019)
10. Lii, N.Y. et al: Simulating an Extraterrestrial Environment for Robotic Space Exploration: The METERON SUPVIS-JUSTIN Telerobotic Experiment and the SOLEX Proving Ground. In: (ASTRA). (2015)
11. Lipton, Z.C.: The mythos of model interpretability. CoRR **abs/1606.03490** (2016), http://arxiv.org/abs/1606.03490
12. Rodriguez, I. et al.: Iteratively Refined Feasibility Checks in Robotic Assembly Sequence Planning. IEEE Robotics and Automation Letters **4**(2), (2019).
13. Seegebarth, B., Muller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In: ICAPS (2012)
14. Stulp, F.: What? How? Why? Explainable artificial intelligence in robotics (Feb 2020), https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3755/17612_read-63005/
15. Veloso, M.M.: Learning by Analogical Reasoning in General Problem Solving. Ph.D. thesis, Carnegie Mellon University (1992)

Appendix

DLR-RMC Green Button Teams

We acknowledge all contributors to the Green Button Challenge, namely Mathilde Connan, Maximilian Denninger, Thomas Eiband, Giuseppe Gillini, Katharina Hagmann, Maged Iskandar, Sebastian Jung, Ulrike Leipscher, Korbinian Nottensteiner, Gabriel Quere, Antonin Raffin, Hanna Riesch, Ismael Rodriguez Brena, Frederick Sauer, Stefan Schneyer, Franz Steinmetz, Martin Sundermayer, and Jörn Vogel.

Algorithm for creating links from plan

Algorithm 1 Creating Links from Plan

```
1: procedure CreateLinks(P, G)
2:     n ← length(P) − 1
3:     steps ← [G, P_n ... P_0]
4:     while length(steps) > 0 do
5:         step ← steps.pop()
6:         if step == G then
7:             atoms = G
8:         else
9:             atoms = step.precondition
10:        end if
11:        for p ∈ atoms do
12:            a_l ← GetLinkedAction(steps, p)
13:            if a_l ≠ None then                                   ▷ see line 25
14:                a_l.links.append((p, step))                      ▷ append link to list of links
15:            end if
16:        end for
17:    end while
18: end procedure
19: procedure GetLinkedAction(steps, p)
20:    for step ∈ steps do                ▷ states are already in reverse order, see line 3
21:        if p ∈ step.effects then
22:            return step
23:        end if
24:    end for
25:    return None          ▷ no action produced p ⇒ it was already present initially
26: end procedure
```