The execution rules allow to: (1) define execution variables (different from planning variables) associated to action executions; (2) check their values at run-time in order to execute local recovery procedures when the values of such execution variables affect the success of the execution of actions; (3) define conditions that allow repetitions of parts of the plan.

This idea has been implemented and experimented with different formalisms, including the transformation of an MDP policy to a conditional plan [Iocchi et al. ICAPS 2016] and the use of the conditional planner Contingent-FF integrated in ROSPlan (http://kclplanning.github.io/ROSPlan).

While the robust plan is represented using the Petri Net Plans (PNP) formalism (http://pnp.dis.uniroma1.it) and executed by the PNP engine.

The method has been tested with a real robot interacting with many users in public environments, within the COACHES project (https://coaches.greyc.fr/)

Although, the system is still sensible to wrong perceptions (i.e., it is still possible that wrong perceptions determine wrong executions of the plans), this risk is minimized, since perceptions are performed a minimum number of times, that is only when it is strictly required to proceed with the plan.

Discussion will include how to further improve the system, by adding more principled solutions of the above mentioned problems. Moreover, generation and execution of robust plans is an orthogonal feature that is useful for every robot planning application domain.

3.16 Rethinking Computational Investments in Planning and Execution

Gal A. Kaminka (Bar-Ilan University – Ramat Gan, IL)

Planning when to plan, and when not to. I work on multi-robot plan execution; in particular, on executing plans for teams of robots. I have been working on plan execution systems for teams of robots, for more than 15 years. A cornerstone of executing team plans is to recognize the points in which robots will make a decision, e.g., by voting or other social-choice mechanism. By recognizing these points, the systems I have designed are able to guarantee good teamwork of the robot, in the sense of carrying out their tasks in an agreed-upon manner.

There is really no feasible way to plan the voting process in advance, in the sense of planning out which robot will vote for what option. This is inherently an execution-time process. However, planning to hold a vote should be possible, just as it should ideally be possible to plan whether to hold a vote of a specific type (e.g., plurality vs. Borda vs. dictatorial). The lesson is that some executions you cannot plan, but you can plan to execute.

More generally ... Back in 1997, I was taking a graduate course in artificial intelligence planning, taught by Craig Knoblock and Yolanda Gil. As a final project, we were asked to write a paper that would tackle an open question, provide the literature survey and recommend directions for further research. My paper addressed the computational effort in planning and execution. I contrasted the approaches of the planning community (graphplan and satplan were the newest planners), and the robotics community (subsumption was being pushed out in favor of behavior-based control). The two communities were seemingly at odds, scientifically. The mainstream planning community focused, as it does today, on building

planners that extensively relied on simulating the effects of actions. To do this, they needed models of how the world behaves. The mainstream robotics community had just completed its embrace of reactivity, subsumption and behavior-based robotics, which emphasized throwing away models (and therefore planning), or at the very least limiting the role of planning significantly. Just a few years before, Matt Ginsburg wrote that "Universal Planning is an almost universally bad idea" in an issue of AI Magazine, and Agre and Chapman, having successfully built Pengo without a planner, were leaving both communities with the impression that there is a justified gap between robotic acting in dynamic environments, and planning for static environments. Sure, there were also researchers working on integrated planning and execution, but they were mostly working on softbots or learning policies via reinforcement learning.

I took the position that the planning community and the robotics community were in fact in complete agreement: they were both advocating the use of incredibly dumb executors. The planning community were expecting an executor which will blindly execute a series of actions, given as grounded operators. The most that would be expected from such an executor would be to check whether preconditions and effects hold as predicted. The robotics community, having given up hope on planning, was instead building very dumb mechanisms as well. Execution of policies, from this respect, is not very different: follow the policy and myopically respond as dictated. The various behavior selection and fusion mechanisms which are often discussed in this community are as myopic as the sequential selection of grounded operators for execution. In short, both communities were assuming essentially all computation is carried out in planning time. By comparison, decision-making during execution is computationally trivial, because it is myopic.

I would like to see us shifting the computational burden, to do more computation during execution (possibly, invested in projections of future state, but not only). Some HTN planners, and BDI agent architectures, come somewhat close to this, in the sense that they both allow on-line refinements for plan recipes. But their refinement is an all-or-nothing deal: recipes given in advance are either refined or are not; they are not usually modified during execution. The philosophical shift in focus of the autonomous agents community, from the agent as a planner, to the agent as a plan selector is not enough in this regard. It emphasizes the importance of integrating planning and execution, and it highlights the very real challenges involved in everything beyond generating plans. But plans are still thought of as rigid objects, generated by planners, to be handed off for execution after some filtering and selection.

I would like to have a planner that knows when to plan, and when to leave off planning to a later point in time; a planner that plans for later planning. I believe the way to achieve this goes through rethinking of the planning process, as a process that spans execution. No more interleaving of calls to a planner with calls to step execution. Rather, a single computational process that plans when it can, and automatically stops filling in details when it cannot.

3.17 Cognitive Robotics on the Factory Floor

Erez Karpas (Technion – Haifa, IL)

Industry 4.0 is one of the most common buzzwords heard today. The term is a reference to the industrial revolutions of the past: The (first) industrial revolution, in the 18th century, came about with the advent of the first steam- powered machines. The second industrial