EPiC Series in Computing

Volume 50, 2017, Pages 134–147

GCAI 2017. 3rd Global Conference on Artificial Intelligence

A Modular Approach of Decision-Making in the Context of Robot Navigation in Domestic Environments

Alexandra Kirsch*

Department of Computer Science
Eberhard Karls Universität Tübingen
Tübingen, Germany
alexandra.kirsch@uni-tuebingen.de

Abstract

Decision processes in Artificial Intelligence are often organized hierarchically. One example is robot navigation with a global path planner and a local executor. This paper examines whether a shift from optimizing the two typical modules in navigation towards a dynamic interaction of more, not necessarily hierarchically linked, modules leads to robust navigation behavior. We empirically evaluate different organizations of modules for navigation in a simulated household with a simulated PR2 robot.

1 Motivation

Robot navigation in household environments is still a challenge. Navigation in general is difficult because of the high variety of situations. In addition, households are more narrow and less structured than classical test environments such as museums or office spaces, and the movement should be legible[1] to people. Kirsch [15] introduces a heuristics-based local navigation method that moves a robot safely, efficiently and legibly inside single rooms, being able to circumvent obstacles without the need of a map or a planner.

However, moving into other rooms or in very cluttered rooms requires planning. The standard approach is to calculate a global trajectory (represented by a list of intermediate points) with a global path planner and then to trace it with a local planner (also called controller) by moving the robot from one trajectory point to the next. In human-aware robot navigation, the global planner has received more research effort than the local planner [17], but the interaction of global and local planner faces some fundamental challenges: 1) When the environment changes (possibly because humans or pets are moving), a new global plan is calculated. The recalculation can be done efficiently, but the impression of goal-directed behavior is lost, the legibility of the trajectory suffers [20, 16]. 2) Global planners abstract the state space and thus ignore some details. As an example, the reachability of points may be estimated by the line of sight. With a simple local planner, the planned trajectory may not be executable without

*This research is funded by Deutsche Forschungsgemeinschaft and the Bavarian Academy of Sciences and Humanities.

[1] According to Lichtenthäler et al. [20] "Robot behavior is legible, if a human can infer the next actions, goals and intentions of the robot with high accuracy and confidence."

```
function decide-next (dm)
  produced-alternatives ← map(λ(p): produce(p, dm.goal), dm.proposers)
  all-alternatives ← union(dm.alternatives, produced-alternatives)
  evaluated-alternatives ← map(λ(e): evaluate(e, dm.goal), dm.evaluators)
  aggregated-evaluations ← map(λ(alt): aggregate-evaluations(alt, dm.weights), evaluated-alternatives)
  sorted-alternatives ← sort-descending-by(aggregated-alternatives, aggregated-evaluations)
  return sorted-alternatives

procedure run-decision-module (dm)
  loop
    iteration ← 0
    loop
      decision-alternatives ← decide-next(dm)
      best ← first(decision-alternatives)
      if acceptable?(best)
        dm.decision ← best
        return
      iteration ← iteration + 1
      until iteration ≥ dm.max-iterations
```

Figure 1: Decision procedure in a decision module. A decision module (dm) contains all of its configuration such as proposers, evaluators and the function *acceptable?*. Alternatives and goals can be set in a decision module by external processes (variables dm.alternatives and dm.goal). The decision of a decision module (variable dm.decision) is accessible from other processes and can be used to configure other decision modules or can be executed as a command.

collisions. A typical remedy is to increase safety distances between the trajectory and obstacles. In a narrow environment with a reasonably sized robot this will often result in the inability to find a plan at all. Recent planners take into account parameters of the execution, but with the cost of the computational overhead and complexity of the algorithm. 3) Tracing a given trajectory requires several parameters to be adjusted. When is an intermediate point accepted as being reached? What should the robot's orientation be at the intermediate points?

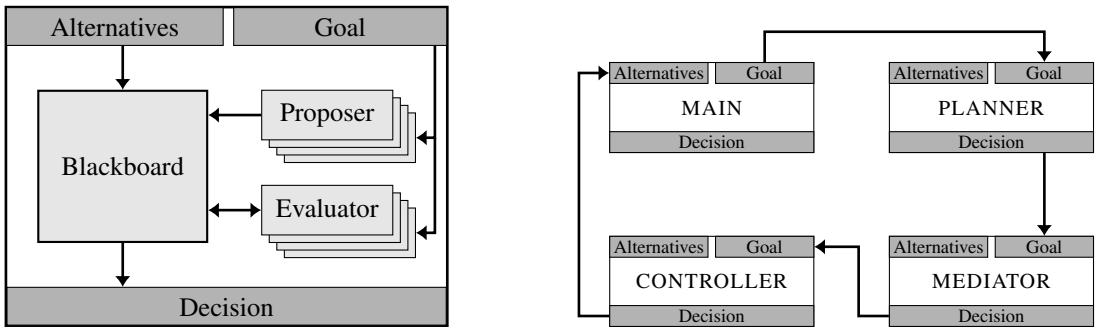
These challenges are not unique to robot navigation, indeed they are rather fundamental in Artificial Intelligence, as in the interplay between planning and plan execution or in hierarchical search methods. Therefore, the proposed approach is not tailored to navigation, but offers a general framework for any type of decision-making. We deliberately use rather simple algorithms in the modules to make sure that we assess the general usefulness of loosely coupled modules rather than a set of algorithms that have been specifically optimized for navigation over the last decades.

This paper examines whether the challenges above can be overcome by a stronger emphasis on the interaction of modules rather than on single modules. We propose two basic mechanisms: 1) the use of additional modules to mediate between the planner and the controller, 2) the renunciation of a strict hierarchy, treating modules more as equal interaction partners than commanders and executors.

We evaluated the approach in the MORSE simulator [19] using a PR2 robot in a fully furnished apartment.

2 Approach

We first explain the basic mechanism of all modules, followed by the specifics of each module and their interactions.



(a) Decision Module. (b) Interaction of the decision modules.

Figure 2: Heuristic problem solving process with decision modules.

2.1 Heuristic Problem Solving

Each module uses the heuristic problem solving approach from [15] (Figure 2(a) and 1), which runs in continual decision cycles. At each decision cycle a set of proposer processes compute solution alternatives. Alternatives may also be provided from outside the decision module. The proposed alternatives are then evaluated by several evaluator processes, each providing an evaluation between 0 (= alternative is considered inadequate) and 1 (= alternative is perfect), or null to indicate that the alternative should not be considered further (e.g., when it would lead to a collision). The individual evaluations are aggregated in a weighted sum to provide an overall evaluation for each proposed command. The alternative with the highest total evaluation is returned as the result of the decision cycle. If several alternatives receive an equal evaluation or are very close (configurable by the function *acceptable?* in Figure 1), another call to *decide-next* is invoked. This only makes sense if some of the producers contain randomness. In the case of navigation, exactly equal outcomes are very rare, therefore we always choose the (unique) best alternative.

In [15], the result was executed as a control command. Here — as we have several modules — the result can either serve as input to another module or as a control command.

Modules differ in the type of alternatives (and thus output) and the experts (i.e., proposers and evaluators).

Heuristic problem solving can emulate existing approaches for planning or control by using only one proposer that implements the existing algorithm and a dummy evaluator that accepts the one proposed alternative. This way we were able to use a standard approach to path planning and to turn off modules to emulate the standard two-level hierarchical approach for the evaluation (Section 3).

2.2 Modules

Figure 2(b) shows how the modules interact. The interface of a module is defined by three variables: goals and alternatives can be set, the decision of a module can be read by other processes. Technically these variables are Clojure[2] atoms with callbacks that make sure that if some variable is changed other modules are immediately informed[3]. For example, a new decision in PLANNER instantaneously changes the goal in MEDIATOR.

[2] https://clojure.org/

[3] Clojure offers the function `add-watch` to invoke some functionality whenever the value of the atom is changed. This mechanism is similar to the `ObservableValue` class and its method `addListener` in Java.

Table 1: Configurations of decision modules. The numbers denote the weights of the evaluators used in the experiments in Section 3. Parameters of producers are shown in parentheses.

(a) Configurations of MEDIATOR.

MEDIATOR *classical*	
Proposers:	
work off plan	
Evaluators:	
accept sole alternative	1.0

MEDIATOR *modular*	
Proposers:	
all points in plan	
goalpoint	
Evaluators:	
remove unreachable	0.5
prefer point near goal	0.5
prefer late point in plan	0.8

(b) Configuration of CONTROLLER

CONTROLLER	
Proposers:	
random sampling (quantity=31)	
repeat last	
motion primitives	
turn (quantity=10)	
Evaluators:	
safety laser	0.19
safety tables	0.21
dwa align	0.25
dwa velocity	0.47
look at goal	1.00
look at path	0.32
goal distance	0.77
towards goal	0.92

(c) Configuration of MAIN.

MAIN	
Evaluators:	
dwa align	0.57
dwa velocity	1.00
look at goal	0.40
look at path	0.84
goal distance	0.81

Module PLANNER computes a global navigation plan from a topological map (Figure 3: the points were defined manually, the connections were computed by the line of sight). It determines the closest reachable point on the topological map from the robot's current position and computes a path through the topological map using A* search to the closest reachable point from the goal point. This strategy of integrating the situation and goal at hand with the static topological map is rather simple. But instead of trying to refine this strategy manually, the idea is that even suboptimal decisions in one module will be compensated for by the interaction of several modules. The decision of this module is a navigation plan, i.e., a list of points.

Section 2.1 mentioned that existing algorithms can be used in the Heuristic Problem Solver by implementing them as a proposer, together with a dummy evaluator that just accepts the computed alternative. The PLANNER module uses exactly this trick to perform a manually implemented A* search. The module runs continually like all the others, so whenever the goal or the world change (including the robot's own position), a new plan is computed.

Module MEDIATOR selects an intermediate goal from the plan that has been produced by PLANNER. We use MEDIATOR in two modes.

In the *classical* mode, it emulates a hierarchical coupling of PLANNER and CONTROLLER, returning the next unvisited point in the plan. A point is considered as reached when the robot has been closer than 0.3 m to it.

In the *modular* mode, the MEDIATOR has a more active role in selecting the next subgoal. Instead of blindly following the plan, it considers all the points in the plan and the global goal point as possible candidates, eliminating those that are unreachable in a direct line of sight and preferring those that are close to the goal and rather at the end of the plan (preferring only the goal can get the robot stuck in dead ends). The experts, with the weights of the evaluators, are listed in Table 1(a).

Module CONTROLLER computes control commands to reach a given intermediate point. The intermediate points include no orientation, whereas the goal pose does have an orientation.

The instantiation of this module is a generalization of the Dynamic Window Approach [7, 2]. The experts of CONTROLLER are listed in Table 1(b), a more detailed account of the computed functions is given in [15]. The proposers comprise random sampling of commands (with only 31 commands, in the Dynamic Window Approach usual configurations use in the order of hundreds of points); the repetition of the last command; a set of "motion primitives", i.e., commands that move the robot in only one of the possible dimensions (for-/back-/sideward, turn) by a random amount; and additionally 10 commands that turn the robot on the spot.

Some of the evaluators rely on a prediction of the robot's pose for the command to be evaluated. Working in simulation, motion equations as in [7] provide sufficiently good estimates.

Two evaluators check for the safety of a command: one using simulated laser data, another using information from a map. If a command would lead to collision, those evaluators remove it. For safe commands, these evaluators prefer commands that keep the robot away from obstacles. Two evaluators are taken from the Dynamic Window Approach: the align evaluator that prefers movement directions towards the (intermediate) goal, and the dwa velocity evaluator that prefers high velocities. Two evaluators consider the rotation of the robot body: one prefers the robot to look at the goal point, the other to look into the direction of the movement. The latter are good examples of evaluators directly contradicting each other. By considering both, the algorithm can find a compromise for maximum legibility of the movement. Finally, two evaluators consider the robot's approach towards the goal: One considers the absolute distance of the predicted pose to the goal, the other the relative approach with that command.

Module MAIN receives the global navigation goal and has the final decision on the command. As in a classical top-down approach, the goal is passed on to the PLANNER, leading to an intermediate goal for CONTROLLER. But instead of just executing the best command found by CONTROLLER, MAIN has access to all the alternatives on the blackboard of CONTROLLER and thus can reconsider what the best command is. For the navigation task as we present it here, this means basically that the best alternatives considered by CONTROLLER can be re-evaluated with a different set of evaluators, possibly adding some advantage in velocity or legibility.

This approach is similar to the subsumption architecture by Brooks [3], where the decision of a "lower-level" module can be overridden by a more complex module.

Module MAIN has no proposers of its own, but considers only the top n (in the experiments in Section 3, $n = 10$) alternatives of CONTROLLER. The parameters are listed in Table 1(c).

3 Evaluation

The basic idea of the proposed approach is to examine whether modules without a strict hierarchy and with unoptimized proposers and evaluators can produce flexible and robust behavior by the interaction of modules and explicit mediation between them. We also include the basic question of whether any abstraction is necessary at all (in the case of navigation whether we need path planning).

The evaluation is done in the context of robot navigation and it is to test the following hypotheses:

1. A global planner is necessary to fulfill arbitrary navigation tasks.

2. Mediating between the global and local planner leads to reaching target points faster and more legibly (measured by jerk and side- or backward movements).

3. The "subsumption" tactic of module MAIN will make trajectories slightly faster and more legible.

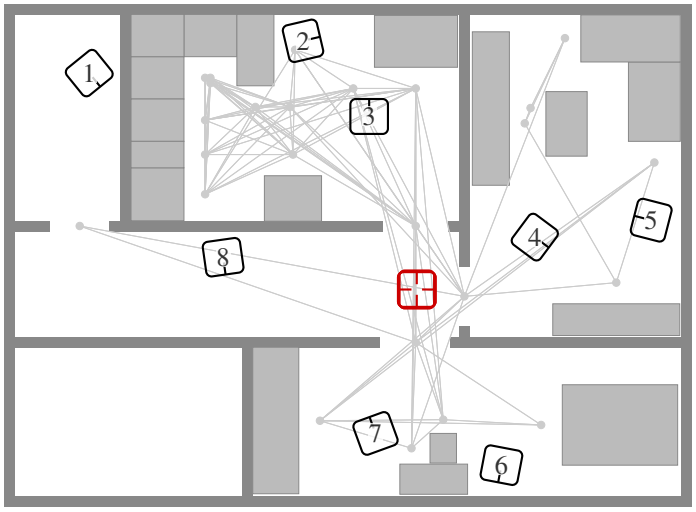


Figure 3: Evaluation setup: Eight randomly generated goal points, the starting point is marked in red and shows the four starting orientations. The figure also shows in light gray the topological map used by the PLANNER module.

All experiments were run on a desktop computer (Intel Core i5-2520M (2.50GHz), 8 GB RAM, Ubuntu 16.04) using the robot simulator MORSE[4], version 1.4. All modules ran continually at 10 Hz.

3.1 Setup

Figure 3 depicts the navigation tasks. The robot started from the point drawn in red, in the four cardinal directions. Eight goal poses were chosen by first generating 30 poses with rejection sampling (poses were rejected when the robot would be in collision with an obstacle), and then randomly selecting one pose in the bathroom and corridor, and two in every other room. Thus, we have 4 starting poses and 8 goal poses, resulting in 32 navigation tasks. Each task was repeated 10 times for each configuration.

3.2 Configurations

The modules presented in Section 2.2 are combined in four configurations. *local* uses only module CONTROLLER, i.e., only a local planner; *classical* uses modules CONTROLLER and PLANNER. MEDIATOR is configured to simulate a classical top-down hierarchy (mode *classical*). MAIN is also active, but just passes on the highest ranked alternative from CONTROLLER. Configuration *hps* uses modules PLANNER, MEDIATOR, CONTROLLER. MAIN is active, but just passes on the highest ranked alternative from CONTROLLER. *hps-sub* is the same as *hps*, but with a subsumption-like configuration of module MAIN (Table 1(c)).

The parameters for the heuristics in module CONTROLLER were optimized in the context of configuration *hps* and then used in all configurations. For the optimization, another set of eight points was selected in the same way as the points in Figure 3. A genetic algorithm optimized the weights of the experts, the number of randomly generated alternatives and the inclusion or exclusion of proposers in 30 iterations. The resulting configuration is given in Table 1. For *hps-sub*, the parameters in MAIN were optimized in the same way, keeping the parameters of CONTROLLER as in *hps*.

[4] https://www.openrobots.org/wiki/morse/

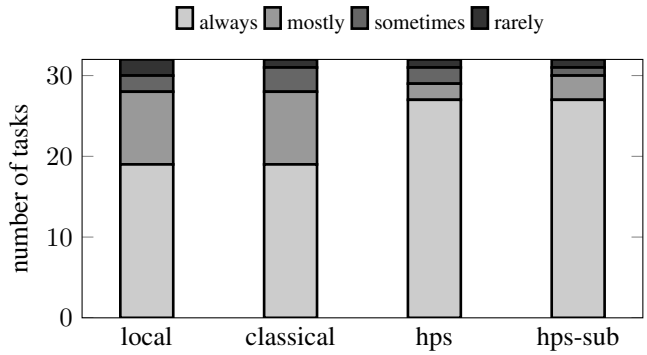


Figure 4: Success of fulfilling the 32 tasks. “Always” means a task was achieved in all 10 trials, “mostly” in 7–9, “sometimes” in 4–6, “rarely” in 1–3 trials. The case “never” did not occur.

3.3 Data

The following data were recorded:

Success A goal was defined as being reached if the robot was within 0.1 m. If a goal pose was not reached within 30 seconds, the trial was aborted. We expected configuration *local* to fail in many trials, and all others to succeed most of the time (hypothesis 1).

Time As the time depends on the specific task, the reported navigation times are relative to the average duration per task, considering only successful runs. Hypothesis 2 predicts that configuration *hps* be faster than *classical*, hypothesis 3 that *hps-sub* be slightly faster than *hps*. Configuration *local* should in general be comparable to *hps*, but in some runs could take more time when it has to trial-and-error its way around obstacles or doorways.

Jerk The absolute jerk was calculated from the robot positions, which were recorded with a frequency of 5 Hz. The jerk is often used to indicate the smoothness of movement [1]. We expect only small differences between the strategies, with *hps-sub* showing slightly smoother movement than *hps* (hypothesis 3). *local* may also lead to jerkier movement as it needs more trial-and-error.

Side- and backward movement In [15], side-/ backward movement serves as a proxy for legible, smooth navigation. In this experiment we expect little difference, possibly more such movements in *classical* as it may try to follow sharp curves, and slightly fewer for *hps-sub* (hypothesis 3).

Collisions All configurations use the same collision avoidance strategy, which should prohibit commands leading to collision. In the narrowness of the apartment, it still happens that the robot streaks an obstacle, but this should happen rarely.

3.4 Results

Figure 4 shows the success of each configuration by the number of times each of the 32 tasks was achieved out of the ten trials. Table 2 lists the most difficult tasks, where at least one configuration failed four or more times.

Table 2: Difficult navigation tasks.

goal	start dir.	*local*	*classical*	*hps*	*hps-sub*
1	→	2	4	0	4
2	↓	9	8	1	0
2	←	5	1	0	0
2	→	1	6	0	0
4	←	5	0	6	1
5	←	9	5	9	8
6	↑	1	0	4	1

The table lists the number of failed runs (out of 10) for the instances in which at least one configuration had 4 or more failed runs. The starting direction shows the robot orientation in relation to the map in Figure 3.

Surprisingly, the main difference was not between configuration *local* and the others, but all four configurations were rather successful, with *local* and *classical* being slightly worse than *hps* and *hps-sub*. *local* being so successful shows that local navigation can be rather powerful, even without a planner. But in all tasks the robot had to cross at most one door. A follow-up experiment (Section 3.5) shows how more complex tasks change the success rate of *local*.

Another surprise was that all configurations had serious problems in reaching Goal 5 when starting with the back to that room. As some heuristics try to avoid moving backward and the area around the starting point is rather narrow, the robot decides not to move at all. When the robot starts from a slightly different angle, the point is reached reliably by all configurations. *hps-sub* was supposed to remedy such situations by being able to pick a different command than CONTROLLER using different heuristics. In the case of getting to goal 4 from the same difficult starting pose, this actually happens: *hps* fulfills the task in only 4 trials, whereas *hps-sub* succeeds in 9 trials. A follow-up experiment (Section 3.5) examines whether a different configuration of module MAIN can improve such situations.

The difference between *classical* and *hps* is slight. Following the planned path point by point, *classical* needs to take sharper turns. For example, it sometimes ends up in the bedroom (the room with goal 6 and 7) on the way to goal 1. Similarly, going to goal 2 from orientation ↓ all configurations had the same problem of not wanting to move backward. But *classical* had the additional problem that it moves along the planned trajectory, visiting each point exactly once. When trying to turn around, it comes close enough to the topological map point in the door to be considered visited, so the next intermediate goal is the final destination. But the robot is still busy trying to pass the door and thus gets into the same situation as *simple*, having to reach the goal without any helpful plan. Of course, this could be remedied by changing the strategy of the proposer work off plan and allowing it to visit plan points several times. But this leads to a plethora of new problems, causing the robot to oscillate around plan points in other tasks. This example nicely illustrates the fundamental problem of robot navigation with two hierarchical layers: it is impossible to find the perfect strategy in each of the layers for arbitrary goals.

However, the flexibility of *hps* leads to problems when moving to goal 4 from direction ← and to goal 6 from direction ↑. *hps* skips the plan point in the living room door, trying to reach the goal point directly and thus moving with the same strategy as *local*. This problem is not caused by the interaction, but by the CONTROLLER module preventing backward movements. Adding functionality to MAIN in *hps-sub* helps to overcome this problem as discussed above. When going to goal 6, the data looks as if *local* did better than *hps*, but both try to reach the goal point directly. The different values in the table are due to the nondeterminism of the simulation and the similarity of aggregated evaluations for the top

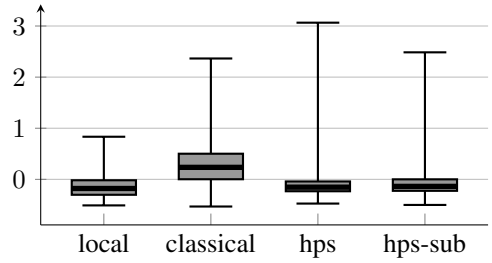
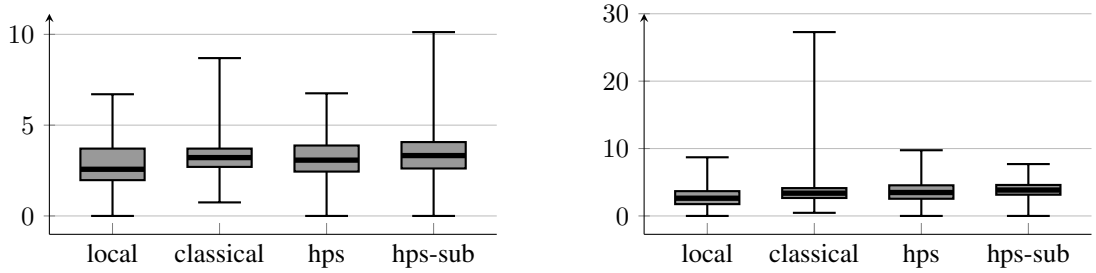


Figure 5: Navigation time of successful runs (in s), relative to average time per task. The thick line is the median, the box contains the lower and upper quartile, the whiskers show the minimum and maximum.



(a) Translational jerk (m/s^3)

(b) Rotational jerk (rad/s^3)

Figure 6: Absolute jerk values.

alternatives. Thus, small variations in the situation can lead to one command being selected over another in different runs, leading to different outcomes.

Giving power to MAIN can lead to worse performance than relying on CONTROLLER alone to choose the command, as can be seen in the first line of Table 2. By being able to override the decision of CONTROLLER, MAIN can choose wrongly. But obviously CONTROLLER is not perfect, so even if in this one case, the performance degrades slightly by the interaction, still the performance over all tasks can rise a lot with MAIN (see also Section 3.5 and Figure 9).

Figure 5 shows the relative navigation times, supporting hypothesis 2: *classical* is generally slower than the other configurations. For *local* we had expected higher times due to the lack of a plan. Even if the method manages to reach the goal, we had expected more trial-and-error, which would result in a less direct path. However, the time needed by *local* is on the same level as *hps* and *hps-sub*.

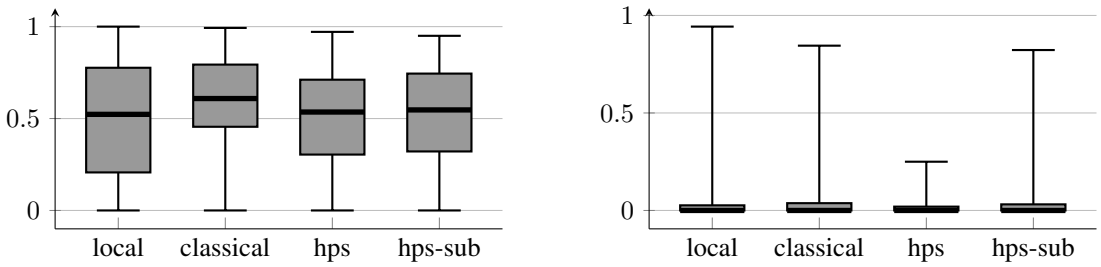
Figure 6 shows the absolute jerk of translational and rotational movement. All configurations show similar values, the supposed gain in legibility between *hps* and *hps-sub* (hypothesis 3) did not occur.

Similarly with side- and backward movement and collisions: all configurations perform on a similar level (Figure 7).

3.5 Follow-up Experiments

The experimental results have opened additional questions, which are examined in two additional experiments.

More complex tasks A surprise was the low failure rate of configuration *local* (contradicting hypothesis 1). A possible reason is that the tasks were too simple to show the difference. In a second experiment



(a) Rate of moving side- or backward.

(b) Collisions (in percent of observed states).

Figure 7: Legibility and collision results.

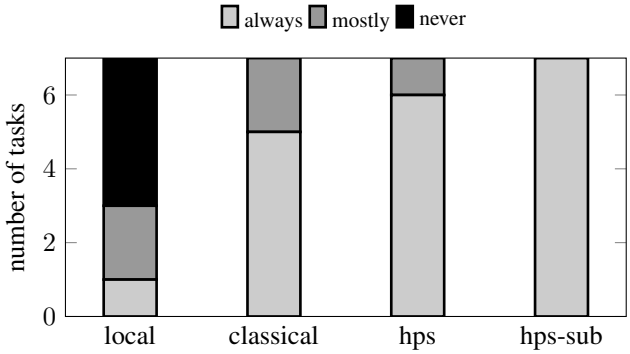


Figure 8: Success of reaching goal poses when starting at point 1. "Always" means a task was achieved in all 10 trials, "mostly" in 7–9, "never" if the pose was not reached at all. Other cases did not occur in this experiment.

pose 1 (Figure 3) served as the starting pose and the tasks were to reach the seven other points (again 10 trials per task). Figure 8 shows that *local* is unable to reach four of the seven points, whereas the other configurations stay at the level of robustness from the previous experiment.

The other configurations even appear to perform better compared to the results in Figure 4. This is due to the favourable orientation of point 1, pointing roughly into the direction of the door and thereby in the direction of travel. The problems in Figure 4 and Table 2 are mostly due to the necessity of a sharp turn at the start of the trajectory.

Parameters of MAIN To test whether the parameters of MAIN can be chosen in a way that this module breaks ties when the reaching of a point is hindered by other considerations such as legibility, we tried three other, more extreme configurations with fewer evaluators: using 1) only look at goal , 2) only towards goal (which was only used by CONTROLLER in the previous experiment, but which seems promising for breaking the ties described above), 3) look at goal and towards goal with equal weights. The results in Figure 9 show that by only using towards goal the success rate increases. But still goal 5 starting in direction ← was reached only in 5 out of 10 trials.

Figure 10 shows that this specific optimization hardly affected the efficiency, the navigation times of *towards* are almost identical to *hps-sub*, but with a slightly smaller spread of values. However, the performance in the legibility parameters diminishes slightly. This explains why *hps-sub* is less

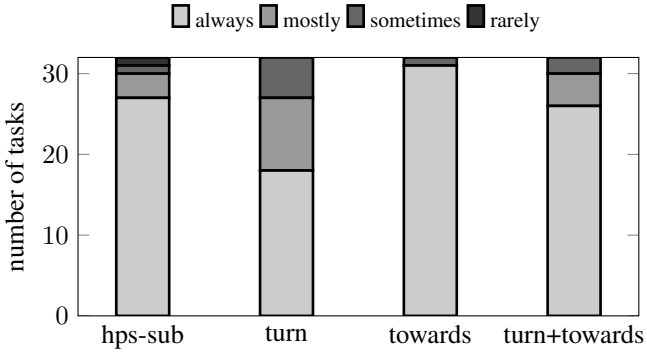
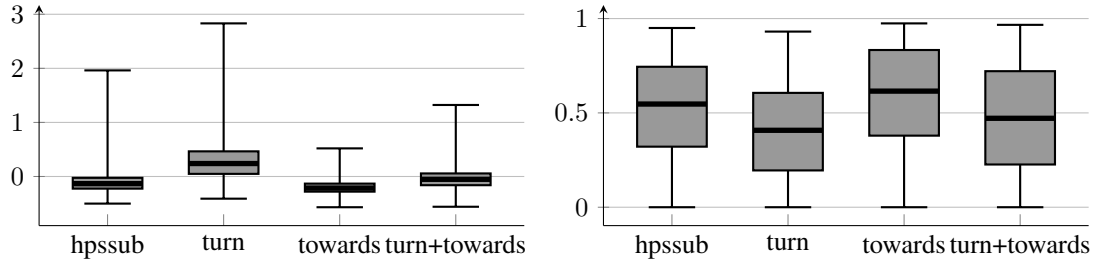


Figure 9: Success counts for modified MAIN parameters."Always" means a task was achieved in all 10 trials, "mostly" in 7–9, "sometimes" in 4–6, "rarely" in 1–3 trials. The case "never" did not occur.



(a) Navigation time of successful runs (in s), relative to average time per task.

(b) Rate of moving side- or backward.

Figure 10: Time and legibility measures for modified MAIN parameters.

successful: it was optimized to fulfill all objectives. Also, this ex post optimization was applied to the test data, whereas *hps-sub* was trained on a different set of tasks.

The main finding in this additional experiment is that the subsumption-like interaction of modules can compensate for bad decisions in single modules, but that this depends largely on a well-chosen parameterization.

4 Related Work

Human-aware navigation has been considered as a fundamental necessity of robots interacting with humans [17]. Research on legibility of robot motion has focused on the global planner [23, 12, 24]. Local planning is usually considered as pure execution of the plan with responsibility to ensure safety [8]. In [15], we have shown that the local planner contributes much to the legibility of the movement, and this paper adds the observation that it can in itself be rather powerful in avoiding obstacles. Therefore the path planner may not need as much sophistication as was previously assumed.

The architectural principle of each module is based on blackboard architectures [4, 11] and heuristic decision making [5, 13]. Haber and Samut [10] propose the use of several blackboards for a cognitive architecture. The interaction of modules proposed here follows the tradition of plan execution schemes such as Reactive Action Packages [6] and Teleo-Reactive Programs [22] (or later developments thereof like the T-REX architecture [21]). While these architectures promote the use of loosely coupled mod-

ules, to our knowledge its benefit has never been assessed empirically.

Hierarchical abstraction is a basic principle in Artificial Intelligence. A well-described and often used architecture for AI planning and execution is the three-layer architecture [9]. Robot navigation seems to be considered a simpler task and usually gets by with two layers [18, 2]. In this paper we used a third module as MEDIATOR, showing that an explicit deliberation between plan and execution improves performance. This is not to show that three layers are better than two, but rather that the interaction of modules should be explicit and not strictly hierarchical. Brooks [3] challenges the hierarchical approach with his subsumption architecture, which differentiates modules not by abstraction, but by behavior. Complex behavior modules can use or override decisions from models producing less complex behavior. The MEDIATOR can in principle (when configured with other experts) determine any point as next intermediate point, it does not even have to be one from the plan computed by PLANNER. But it uses the outcome of PLANNER as a hint or heuristic to determine a good intermediate point. By the dynamics of the system, where each module constantly recomputes the best choice for the moment, bad decisions are compensated either by other modules directly or in the near future by re-computation in a slightly changed situation. This idea has also proved useful in another context of route finding [14].

This view of modules instead of hierarchies is supported by findings from cognitive science. For human route planning Wiener and Mallot [25] suggested a 'fine-to-coarse' model that takes into account region knowledge in path planning, but unlike 'coarse-to-fine' strategies that promote a strict top-down approach, they suggest that in working memory a simplified copy of the hierarchical organization of the world is held and modified during the path finding process. Similarly, knowledge of a global navigation plan can be used for navigation, but in a simplified and modifiable form.

5 Conclusion

We proposed a shift from optimizing single modules towards an interaction of equitable modules for a more robust, efficient, and legible navigation in domestic environments. The experiments show that this approach leads to more efficient and similar or even more robust behavior than a classical (not particularly optimized) approach. We are sure it would be possible to reach a similar level of navigation performance by tweaking the global and local planner in a classical approach. But this tweaking is specific for navigation, whereas the modular approach is so general that it can be applied to other control and decision-making tasks. Other than expected, the modular approach did not make the behavior more legible.

At first glance, the modular approach increases the number of parameters that need to be optimized: apart from the parameters of the single modules, it adds parameters of the interaction (for example, the number of alternatives of module CONTROLLER that are taken over to MAIN). Figure 9 indicates that the parameterization does play some role, but the level of performance is affected only slightly. Still, for future research, a standard optimization method should be part of the architecture, so that all parameters (those inside the modules and those controlling the interaction, such as the number of alternatives of module CONTROLLER that are taken over to MAIN) can be found automatically or even adapted for individual tasks.

Another aspect to consider in future work is the computational efficiency. Adding more modules and decision steps adds computational load. But as the single modules are not required to deliver perfect solutions, they can be configured parsimoniously, like CONTROLLER only generating 31 random samples, which is far less than in a usual configuration of the Dynamic Window Approach. The frequency of 10 Hz used in this work is rather low and still led to good results. Future work will have to show how the behavior is affected when some modules run less frequently than others.

References

[1] G. Arechavaleta, J.P. Laumond, H. Hicheur, and A. Berthoz. An optimality principle governing human walking. *Robotics, IEEE Transactions on*, 24(1):5–14, 2008.

[2] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *IEEE International Conference on Robotics and Automation*, pages 341–346, 1999.

[3] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, April 1986.

[4] Robert Engelmore and Tony Morgan, editors. *Blackboard Systems*. Addison-Wesley Publishing Company, 1988.

[5] Susan L. Epstein. Pragmatic navigation: Reactivity, heuristics, and search. *Artificial Intelligence*, 100:275–322, 1998.

[6] J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, WA, 1987.

[7] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1):23–33, 1997.

[8] Thierry Fraichard. A short paper about motion safety. In *IEEE Int. Conf. on Robotics and Automation*, 2007.

[9] E. Gat. On three-layer architectures. In P. Bonasso, D. Kortenkamp, and R. Murphy, editors, *Artificial intelligence and mobile robots*, pages 195–210. MIT Press, Cambridge, MA, 1998.

[10] Adam Haber and Claude Sammut. A cognitive architecture for autonomous robots. *Advances in Cognitive Systems*, 2:257–275, 2013.

[11] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[12] Rachel Kirby, Reid Simmons, and Jodi Forlizzi. COMPANION: A constraint optimizing method for person-acceptable navigation. In *IEEE International Symposium in Robot and Human Interactive Communication (Ro-Man)*, 2009.

[13] Alexandra Kirsch. Humanlike problem solving in the context of the traveling salesperson problem. In *AAAI Fall Symposium on Advances in Cognitive Systems*, 2011.

[14] Alexandra Kirsch. Hierarchical knowledge for heuristic problem solving — a case study on the traveling salesperson problem. In *First Annual Conference on Advances in Cognitive Systems*, 2012.

[15] Alexandra Kirsch. Heuristic decision-making for human-aware navigation in domestic environments. In *2nd Global Conference on Artificial Intelligence (GCAI)*, 2016.

[16] Thibault Kruse, Patrizia Basili, Stefan Glasauer, and Alexandra Kirsch. Legible robot navigation in the proximity of moving humans. In *Advanced Robotics and its Social Impacts (ARSO), 2012 IEEE Workshop on*, pages 83–88. IEEE, 2012.

[17] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[18] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[19] S. Lemaignan, Echeverria G., M. Karg, M. Mainprice, A. Kirsch, and R. Alami. Human-robot interaction in the MORSE simulator. In *Proceedings of the 2012 Human-Robot Interaction Conference (late breaking report)*, 2012.

[20] Christina Lichtenthäler, Tamara Lorenz, Michael Karg, and Alexandra Kirsch. Increasing perceived value between human and robots — measuring legibility in human aware navigation. In *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 89–94, 2012.

[21] Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. T-rex: A model-based architecture for auv control. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, 2007.

[22] N. J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[23] Emrah Akin Sisbot, Luis F. Marin-Urias, Rachid Alami, and Thierry Simeon. A human aware mobile robot

motion planner. *IEEE Transactions on Robotics*, 23:874–883, 2007.

[24] Mikael Svenstrup, Thomas Bak, and Hans Jørgen Andersen. Trajectory planning for robots in dynamic human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.

[25] Jan M. Wiener and Hanspeter A. Mallot. 'Fine-to-Coarse' route planning and navigation in regionalized environments. *Spatial Cognition and Computation*, 3(4):331–358, 2003.