Keep It Simple, Hybrid! — A Case Study in Autonomous Office Courier Robot Control

Adriana Arghir, Joachim Hertzberg[⋆]

GMD – German National Research Center for Information Technology
Institute for Autonomous intelligent Systems (AiS)
Schloss Birlinghoven
53754 Sankt Augustin, Germany

Abstract. We describe the control architecture of an autonomous mobile robot, using courier tasks in office buildings as a demonstrator domain. The robot robustly performs state-of-the-art tasks such as autonomous navigation through the building, dynamically avoiding obstacles, entering rooms, passing automatic doors, and calling and riding the elevator. We give performance statistics of robot tours of about 8 km length in our three-storied institute building. The interesting feature of the control software is its simplicity. This is achieved by combining pieces of fuzzy control, basic geometrical reasoning, target-point planning, and map-based route planning in a hybrid levelled control architecture. The control parameters proposed by low-level behaviors are fused and executed by a complex behavior, which gets selected according to the recently executed operator in a high-level mission plan. The paper sketches these pieces individually and their interplay, with emphasis on the low-level control.

1 Background and overview

Autonomous robots clutter todays' research lab corridors. Realizing the basic skills of an autonomous office courier robot, such as obstacle avoidance and localization within a building, has become much of routine—at least within a research context. This is the time then to experiment with "softer" aspects of robot control systems, such as architectures and representation formalisms that help reduce the effort for developing and maintaining the control software, or the integration of a robot into an existing information infrastructure and facility management. So, the issue is design, and a clue to good design is simplicity.

In this paper, we describe a study in robot control system design, whose main purpose was to achieve simplicity in several respects, rather than a new robot functionality. The demo application scenario is autonomous office courier services of the following kind: The robot has a map of the building (which needs not be metrically accurate), and it has a wireless link to the local ethernet and to a software interface to the elevator control and the control of all automatic doors at the ends of corridors, which are normally kept shut for fire protection. The robot receives courier jobs from human agents by e-mail, consisting of start and end points of transportation tasks within the whole institute building, which may involve entering rooms and using the elevator. Job execution has to be planned and scheduled automatically, and the robot has to travel autonomously under normal security requirements, such as not running into walls or over humans. This application scenario is clearly similar to others in the literature, such as for MOPS [TGVS99] or RHINO [Bee99].

The robot used for the practical experiments in our study (Fig. 1a) is a sturdy (200 kg) industry-standard driver-less transportation platform with an added control module to allow for autonomous control, consisting mainly of a PC/104 computer under Linux, which supports WAVE-Lan communication. The robot has a differential drive plus four caster wheels. It has two types of sensors: A simple odometry (dead reckoning from wheel encoder signals) and two SICK laser scanners at its front and back. [Arg00b] describes all technical details. The arena is our three-storied institute building, of which Fig. 1b shows a map of the first floor. The three storeys are connected by a staircase, which the robot must avoid, and by an elevator, which it operates over its Ethernet link. A typical transportation task is to go from office 102 on the first floor to 227 on the second floor. The robot platform and the infrastructure was available for us from prior projects [PFPB99], and we have used it for our study without changes in the hardware and low-level software.

[⋆] Email: [arghir | hertzberg]@gmd.de

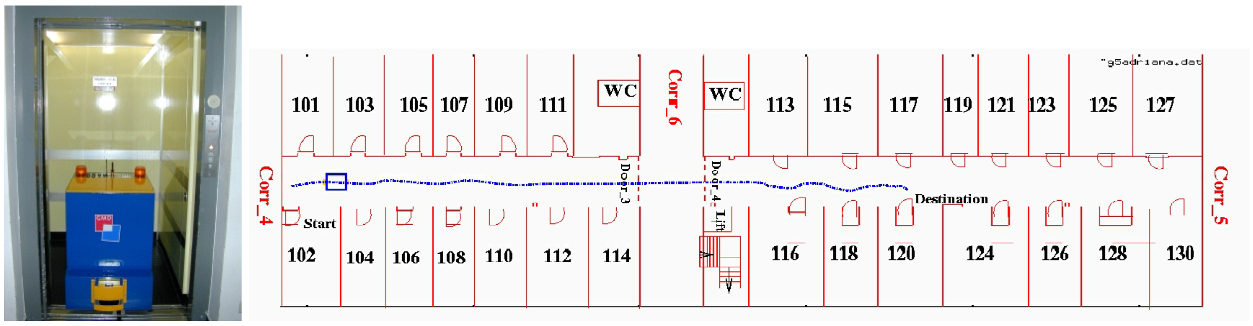

Fig. 1. (a) the robot exiting from the elevator; (b) map of the first floor with a performed concrete trajectory inserted. (The insertion is done manually, as the robot in the study has no access to ground truth.) Door_3 and Door_4 are automatic doors; the elevator is between the staircase and Door_4.

The rest of this paper describes, first, the general control architecture including the planning layer, which is not very sophisticated in our study. Then, we continue bottom-up, describing the basic behaviors (Sec. 3) and the context-dependent combination of their outputs into the complex robot behavior (Sec. 4). After that, we discuss the performance of the robot and of its control system design (Sec. 5). Section 6 concludes.

2 The control architecture

An important issue in simplifying the robot control system is an architecture that allows those problems to be handled off-board the robot that logically need not be done on-board. A chief problem of that kind is to handle the information about the building. This may be a relatively simple topological and metrical building model plus some basic information of direct importance to the robot control, such as whether office doors would open into the corridor or into the office; we have used such a relatively simple model for our study. Alternatively, the information may come from a full-fledged facility information system, whose purpose far exceeds the direct needs of supporting a mobile robot. Note that such a software design-oriented separation of functionality blurs the boundary of the robot, whose control then partly resides inside a computer network that is physically disconnected from the “proper” mobile robot body. This distinction is unimportant and may be invisible for a human user.

To enhance reliability of the wireless communication to the mobile robot and to reduce traffic over the wireless link, the communication with the robot should be on some abstract, task-oriented level. This idea has been used before, for example, in high-level teleoperation, such as in the ROTEX space station manipulation robot [BLSH95], or in plan-based mission control as in the Deep Space 1/Remote Agent experiment [MNPW98, Rem00]. The resulting control architecture is levelled, which is obviously not a new design principle, as, e.g., its extensive coverage in a textbook like [Ark98, Ch. 6] shows.

In our study, a typical user request is for the robot to move from one room to another room, e.g., *Go(102,120)*. The *abstract* trajectory, based on abstract regions and objects like corridors and doors, is computed using the map of the building like in Fig. 1b. (Fig. 1b also shows a *concrete* trajectory as performed by the robot in executing the abstract one.) The computation is done in the *Planning layer* of the architecture, cf. Fig. 2, based on the *building model,* and it is simple: depending on the building structure, it consists of a graph search or even a tree search. In the example, the abstract trajectory would be to move from *102* into and through *Corr_4*, through *Door_3* into and across *Corr_6*, through *Door_4*, and so on. The planning layer is also the place in the architecture for handling sets of user requests or for breaking down more complex tasks to a sequence of elementary motion tasks as just described.

A sequence of motion tasks is not yet the *action list* sent to the physical robot for execution. The building model contains more specific information for the low-level robot control. For example, driving along a corridor like *Corr_5* where office doors open into the corridor requires more care than along *Corr_4*, which can be expected to be comparatively open (but of course with the permanent chance of humans crossing or temporary obstacles blocking). *Corr_6* again requires different low-level control as its floor covering is different (tiles vs. carpet) and proximity to the staircase must be avoided with guarantee at any time, so the position control must be strict. The *action list* sent to the robot (cf. Fig. 2) respects and encodes that type of available information by translating the abstract trajectory context-dependently into a sequence of *mid-level actions,* which are fine-grained enough to capture these differences. In our study, we have used 26 different such actions, which include, for reasons of uniformity, some trivially executable ones like waiting or sending a command to the automatic doors or to the elevator. The interesting ones are those for motion: different varieties for turning, driving, and correcting the position estimation using the laser scanners in defined areas like corridor ends.

Space does not permit to describe the planning layer and its components in more detail, to which we refer to [Arg00b]. We will come back to describing and discussing some mid-level actions below, when it comes to executing them.

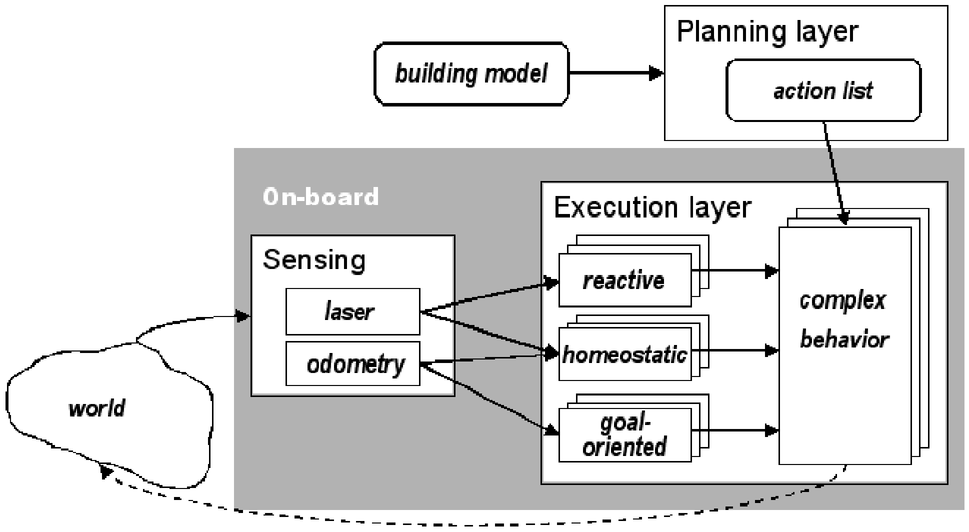

Fig. 2. Simplified sketch of the overall control architecture for the study. The full implementation as in [Arg00b] has structured the planning layer further, which is omitted here for brevity. Solid arrows depict data flow; the dashed arrow depicts physical action by the robot. See text for more explanations.

3 Basic behaviors

We now turn to describing the *execution layer* of the architecture as in Fig. 2. We start bottom-up with the basic behaviors (left part of the "execution layer" box); the process of fusing the outputs of the individual basic behaviors is the topic in Sec. 4.

We are using three classes of basic behaviors: reactive, goal-oriented (in the sense of reaching some target location) and homeostatic[2]. The rationale for this distinction is the need to link sensing and control differently in different basic behaviors: Some (the reactive behaviors) rely on sensory input from the laser scanners, some (goal-oriented behaviors) require odometry data, and some (homeostatic behaviors) require both laser and odometry data, as depicted in Fig. 2. Of course, this idea can be generalized to sensor configurations that are richer than the one in our case study.

All these basic behaviors are permanently active in the sense that they run conceptually in parallel in shortest possible cycles. The control parameters that are computed in our study are exactly two for each behavior, namely, desired velocity variation v and desired angle α. The most recently available outputs are then fused, as described later. Depending on the current goal and on the environmental contingencies, the result of the fusion is typically that the output of two basic behaviors would actually influence the motor control commands sent to the robot hardware.

We have used two main mathematical frameworks for formulating behaviors, namely, fuzzy logic and basic geometrical reasoning. Basic behaviors are uniform in that each behavior yields as output α and v,

[2] We are using the term *homeostatic* in the sense of [Ark98, Sec. 10.2.1].

given as scalar values. Internally, they may use whatever framework is most appropriate (and most simple) for their intended aspect of control.

Reactive behaviors use fuzzy logic [YZ92] based on the laser scanner data. For the study, we have implemented some varieties of obstacle avoidance, whose fundamental principle is that any goal position or direction attracts the robot, while any obstacle repels it. Here is an example of how the basic behavior `forward-obstacle-avoidance` works.

The laser scan area of 180 deg is divided into four sectors with the linguistic variable names FL, FFL, FFR and FR, like *front left, forefront left,* etc., see Fig. 3. These linguistic variables can take terms like *close, medium, far*, representing the minimal measured distance value per scan area, sketched by the black arrows in Fig. 3. (Precisely, FL, FR can take 2 terms, FFL, FFR 4 terms.) Gaussian membership functions are used to represent the possibility distribution of the input variables; details can be found in [Arg00b]. The output values of every reactive behavior for the speed v and the angle α is computed in two stages. First, fuzzy rules are used to compute terms for these two variables. For α, e.g., the possible terms are *hard left, medium left, straight, medium right, hard right*, whose membership functions, again, are Gaussians. To give an example of the fuzzy rules, here is one for determing α:

if FR is *close* **and** FFR is *medium* **and** FL is *far* **then** α is *medium left*

Note that the rule does not mention FFL. Omitting variables is possible in our rule format, which is due to the usage of the Gaussian membership functions that never vanish completely to 0. This considerably reduces the number of necessary rules: Overall, there are 5 such rules for determining the speed and 7 rules for the angle. They are all hand-crafted.

As behaviors are required to uniformly return scalar values rather than fuzzy sets, there is a second stage of computation, namely, defuzzification. We use the standard center-of-gravity (COG) method [YZ92].

Our rule set is remarkably small both in the number and the length of rules compared to other fuzzy robot control systems reported in the literature. For example, Tunstel and Jamshidi [TJ94] describe a fuzzy control system for following a wall (in a non-variable distance) consisting of 54 rules; Xu *et al.* [XTF97] present a simple fuzzy robot control system for basic indoor navigation using 40 rules for 3 input and 2 output variables (velocity and angle).

The simplicity of our rule set is of obvious advantage for checking the correctness of individual rules as well as the consistency of their interplay. The main reason for this simplicity is that we use fuzzy logic only in few and limited parts of the overall control, which means that the context in which it is applied is very well-determined. This is an argument for hybridness not only between deliberative and behavior-based control layers within a robot control architecture, but also for different formalisms within the behavior layer.

Within the overall control, the reactive behaviors are the most important ones. In particular, they get used for avoiding obstacles and for crossing narrow passages like corridors cluttered with open doors and other objects. The obstacle avoidance is robust; it is able to handle even concave obstacles of a depth until about 1.5 times the robot length.

Goal-oriented behaviors are different varieties of the type `go-to`(X, Y), where (X, Y) designates a destinated position in the arena, and the recent v and α control parameters have to be determined with respect to the current estimated pose (position plus orientation) in the basic geometrical environment model on-board the robot. They are predominantly used for short distances where the position control must be strict.

Velocity is a function of the distance to the goal position, which we compute using fuzzy logic, based on the internal odometry data. The principle is this: if the distance is large, then the velocity is high; if the distance is small, then the velocity is low. The angle α is determined using basic geometry for determining strictly the heading towards the goal position. That means, on the other hand, that a goal-oriented behavior will fail if there is no straight passage from the estimated current to the goal position. This feature is foreseen and must be tolerated or handled by other parts of the control.

Homeostatic behaviors are continually trying to approach or keep stable over time some internal or external parameter. The two main instances of this behavior class in our study are `wall-following` (which keeps stable the measured distance to the wall while driving along some corridor over some given distance) and `correction` of the estimated robot pose on corridor ends using the laser scanner (which approaches zero difference between estimated pose and laser data).

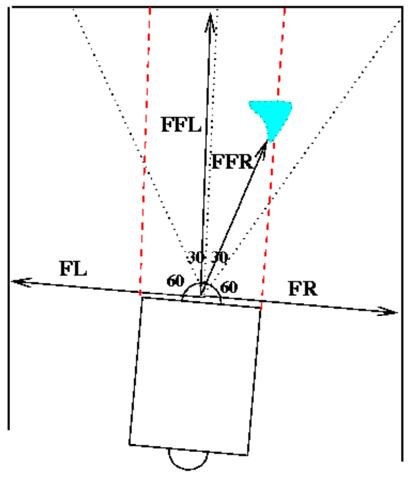

Fig. 3. The four linguistic variables for forward obstacle avoidance and their respective scan areas (bordered by the dotted lines). Black arrows show the minimal measured distance for every variable in the given situation. FFL and FFR use only the intersection between their respective scan areas and the "virtual tunnel" given by the striped lines, which is a little wider than the robot itself and has proved a helpful concept for navigating through corridors and narrow passages.

For `wall-following`, we are using angle histograms [WvP94] to determine the robot orientation θ with respect to the vanishing line of the corridor. Second, we compute an angle δ to the wall using the data from both front and back laser scanners and determining the wall distances using the median functions. θ and δ are then merged depending on the situation, e.g., θ is used exclusively if the δ value is blurred by standing besides an open door.

Homeostatic behaviors are predominantly used in clear corridors or corridor parts, as they have foreseeable problems in densely populated corridors, where the angle histograms may be severely disturbed. Such problems can be handled partly within the behaviors by dampening α changes and reducing v. More frequent occurrences of that type of problems need be handled by other parts of the control, i.e., by not using homeostatic behaviors in the respective critical environment conditions.

Let us summarize the philosophy behind the design of the basic behaviors. Rather than trying to employ an exclusive, uniform principle or framework for formulating them, such as fuzzy control or the theory of dynamical systems, we have used a hybrid approach, using small instances of different frameworks and methods in different behaviors or behavior classes. The individual basic behaviors are very simple and comprehensible. Obviously, the approach scales up to enlarging the amount of basic behaviors, as they are independent of each other. (Computation time is no issue here, as the computations involved are typically very basic, and computationally more demanding behaviors could run in parallel.)

The three basic behavior classes use different types of sensor information and different methods. A consequence that our study has shown is that it is possible to combine the basic behavior effects easily to achieve a coherent and robust overall control. Technically, this combination is achieved by using complex behaviors, which are the topic of the subsequent section.

4 Complex behaviors: Combining behavior effects and mid-level actions

This section—and for that matter, the complex behaviors in our control architecture, which this section describes—deals with two issues. First, the issue how the actions from the action list coming from the planner (Fig.2) influence the physical robot performance; the answer, briefly, is: By selecting an appropriate complex behavior for the recent action. Second, the possibly conflicting control values proposed by the basic behaviors have to be "brought together" in some sense. The section deals with these two issues in turn.

4.1 Selecting complex behaviors for execution

It is typical for layered control architectures [Ark98, Ch. 6], that the success or failure of the resulting robot control systems depends on how the abstract action plans are mapped into control code. We have used the following principle here:

> The robot is effectively controlled *not* by the basic behaviors, but by one *complex behavior* at any time, which is selected according to the recent mid-level action from the action list. This complex behavior is modulated in its functioning by the v and α outputs of the basic behaviors, which the complex behavior receives as inputs and fuses them as appropriate for its functionality (see Sec. 4.2).

This is different from typical layered robot control architectures [Ark98, Ch. 6], which would translate high-level plan actions into mid-level execution routines into low-level control code. In that sense, the term "behavior" may be uncommon for control units like our complex behaviors. We are using it in the spirit of [Saf97, Sec.3.3] to emphasize that a complex behavior has a very similar syntactic form to a basic behavior; in particular, its output includes a pair of scalar values v, α, which are fed into the robot's motion control. (Additional outputs are possible for complex behaviors, like commands sent to the doors and elevator.)

Selecting a complex behavior, given the recent mid-level action from the action list, is not a problem, because there is a 1-1 mapping between the two, which is hand-coded as part of the domain modeling. This principle requires that the domain modeler/robot programmer finds a common level of granularity for mid-level actions (as in the action list) and for executable control routines (as in the inventory of complex behaviors). It is easy to experiment with different implementations of a mid-level action in terms of different variants of complex behaviors; Sec. 5 will give an example for that.

In consequence, for each and every of the 26 mid-level actions mentioned in Sec. 2 there is exactly one corresponding complex behavior. Examples are

- turn (left/right) by a given relative/absolute angle;
- drive forward/backward a given distance with/without obstacle avoidance;
- drive to a relative/absolute position with/without obstacle avoidance;
- follow the corridor walls left/right at some clearance over some distance;
- call/ride the elevator to a given storey;
- open/close a given automatic door;
- correct estimated position at the corridor end using the front/back laser scanner;
- wait until a relative/absolute time has passed.

One possible *action list/complex behavior sequence* for generating the first part of the robot trajectory given in Fig. 1b is given next [*with some comments*]. Assume the robot has its initial estimated pose by the corridor end close to office 102, facing the corridor.

A1 correct the initial estimated position at the corridor end with the back laser scanner;
A2 turn left to an absolute angle of 0 deg [*which is by definition the orientation of the corridor*];
A3 follow the corridor walls left with clearance 100 cm over [*estimated!*] 1908 cm [*which is the distance between the estimated initial position and the corridor end according to the map, minus the front clearance necessary in front of the automatic door to await its opening*];
A4 correct the estimated position at the corridor end [*automatic door*] using the front laser scanner;
A5 open Door_3;
A6 drive with obstacle avoidance [*i.e., stopping before any obstacle*] to the [*estimated!*] absolute position (2381,0) [*which, according to the map, is the center of the crossing between corridors 4 and 6*];
A7 close Door_3;
A8 turn to 0 deg absolute [*i.e., correct the orientation*];
A9 open Door_4;
A10 ...

4.2 Fusing the basic behavior outputs

While the basic behaviors are active all the time, a complex behavior is activated only if the corresponding action currently under execution so dictates. Complex behaviors rely on the inputs of more than one basic behavior; none of the complex behaviors relies on the inputs of all basic ones. v, α values of an irrelevant basic behavior are simply ignored by a respective complex one. For all more or less relevant v, α values, there is the well-known (e.g., [Saf97],[Ark98, Ch.3.4]) question of how to bring them together.

Again, we are using no uniform formula here. However, the pattern in which this is done is uniform, and very simple, in all complex behaviors. We present it by example of a complex behavior to drive to an absolute position (X, Y) with obstacle avoidance, call it `drive-oa`(X, Y).

Mostly, one basic behavior dominates the v output and one (not necessarily the same) the α output of the complex behavior, and for both parameters, there are very few (typically 1) basic behaviors that may modify the dominating output. In the case of complex driving behaviors involving obstacle avoidance, a basic behavior like `forward-obstacle-avoidance` (cf. Sec.3) would dominate iff some obstacle is close, and a goal-oriented basic behavior `go-to`(X, Y) would dominate else.

```
if       $v_o$ is high then $v := v_g$
else if  $v_o$ is low then $v := v_o$
else     $v := \min(v_g, v_o)$
```

```
if    $\mathrm{sign}(\alpha_g) \neq \mathrm{sign}(\alpha_o)$ then $\alpha := \alpha_o$
else  $\alpha := \max(\alpha_o, \alpha_g) - d \cdot |\alpha_o - \alpha_g|$
```

Fig. 4. Fusion procedures for v and α values in the complex behavior `drive-oa`(X, Y). $v_g, \alpha_g, v_o, \alpha_o$ are the respective outputs of the basic behaviors `go-to`(X, Y) and `forward-obstacle-avoidance`, respectively. For the damping factor d, we require $0 \leq d \ll 1$. Further explanations in the text.

Fig. 4 shows how the v and α values of `drive-oa`(X, Y) are computed from the outputs of the basic behaviors `go-to`(X, Y) and `forward-obstacle-avoidance`. The value for the velocity is, roughly, the lower of the two. Here, "high" and "low" are no linguistic terms in the sense of fuzzy logic, but stand for threshold constants that have been determined by experimentation. The **else** part of the α determination obviously uses a ramp function dampening the larger value by some percentage of the smaller one. Again, determining a suitable damping factor d is a matter of experimentation.

5 Robot performance and "architecture performance"

The reader may get an impression of the robot performance from the Quicktime videos under [Arg00a]. Two features are worth mentioning as they challenge the robot control. First, the elevator cabin is only slightly broader and deeper than the robot frame (see also Fig. 1a), so entering it autonomously requires considerable fine-tuning in motion. Second, if we mention above corridors with doors opening into the free space and cluttered with objects, we really mean it. The robot was able to reliably pass areas barely wide enough for it to fit through. Again, that is proof that the basic behaviors, in particular, the different variants of obstacle avoidance, do their jobs.

Let us mention further that the physical robot used in this study was not ideal for the type of task. So if the robot has performed well, then this cannot be attributed to superior hardware or to a close harmony between the robot accessories and its working arena. To repeat, the purpose of our study was to examine a particular robot *control architecture*, not to set new office courier robot benchmarks.

Table 1 lists some of the empirical data of our study; the complete set is given in [Arg00b]. The table describes different control modes and their outcomes in driving the tour shown in Fig. 1b. The columns labelled **Corr_i** correspond to the corridors in Fig. 1b and state the type of complex behavior used while driving through the respective corridor in terms of the *class of the dominating basic behavior*. For example, the complex behavior A3 in the example list in Sec. 4.1, which is the complex behavior used in *Corr_4,* is of the type "homeostatic+goal-oriented" as in line 1 of the table; the complex behavior listed as A6 is the one used in *Corr_6.*

As the table tells, it is advisable to drive reactively in *Corr_5.* This is expected, as the respective doors open into the corridor (cf. Fig. 1b), often blocking the robot's path and disturbing the laser histograms—the drop of performance in line 2 shows the effect. *Corr_4,* with its door opening into the offices, is more benign, as comparing lines 1 and 3 shows. In *Corr_6,* only goal-oriented complex behaviors are used, because the way is short (i.e., the accumulated odometry error is small), and it is important to reach the target position quite exactly to be out of the automatic doors' way.

Version	**Corr_4**	**Corr_6**	**Corr_5**	#Trials	#Correct	
1	homeostatic + goal-oriented	goal-oriented	reactive	20	17	85%
2	reactive	goal-oriented	homeostatic	20	8	40%
3	reactive	goal-oriented	reactive	140	119	85%

Table 1. Statistics about robot runs between start and destination points on corridors Corr_4 and Corr_5, resp., from Fig .1b using different versions of complex behaviors. "Correct" means that the destination position is reached within a given tolerance. See text for further explanation and discussion.

The best robot performance in the table is 85%, which is not ideal. Our failure statistics for line 3 tells that 12 of the 21 failures were due to data communication problems with the laser scanners, so 9 of 140 trials (6.4%) failed for control software problems. Most of these failures could be solved by a more sophisticated localization. The data recorded for the complete statistics for this tour in [Arg00b] correspond to robot tours of about 8 km length. There were considerably more tours between other offices (e.g., between storeys), which qualitatively reproduced the mentioned correctness rates. All test tours were performed during office hours, i.e., humans were crossing the robot's path.

For the case study reported here, the performance of the robot, e.g., robustness and speed of task fulfillment, is in fact of lesser interest than the "performance" of the hybrid representation and of the control architecture as a whole. Here, relevant criteria are on the level of software engineering, such as ease of developing, maintaining and changing the control system—which is hard to quantify.

Our point about that type of performance must be read *between* the lines of Table 1. We are able to change the robot control program on the mid-level in no time, if the point is experimenting with existing, alternative mid-level actions—that is exactly what leads from line to line in the table. Writing a new basic or complex behavior for implementing a new or an existing mid-level action takes time and meticulosity as usual; but integrating the new behavior into an existing control is relatively easy because the restrictive structure of the architecture keeps behavior effects (both of basic and complex behaviors) strictly local in the contexts in which they are explicitly applied.

The hybridness of formalisms on the basic behavior level allows basic behaviors to be implemented as the programmer feels it is most natural. This comes at a price: First, the output is standardized to being v, α pairs. This would scale up to including more parameters, but it remains a restriction. Second, there is no way to support *in the architecture* the merging of different behavior results that is possible in behavior-based control using a uniform formalism. In pure fuzzy control systems [Saf97], e.g., basic behavior outputs can be fused without prior defuzzification, i.e., on the basis of their fuzzy sets. Suitable parts of programming environments for other behavior-based frameworks could be integrated in a design tool for a hybrid basic behavior design tool.

So, the hybridness comes at the obvious price of giving up the comfort that uniform frameworks provide. We found that acceptable for the relative naturalness and simplicity that the hybrid basic behaviors buy us.

6 Conclusion

Let us summarize the main contributions of the study reported in this paper. They are all on the level of robot control program design, not about presenting an unprecedented robot functionality or performance.

Hybridness among layers and within the behavior layer. We have presented a hybrid, layered robot control architecture, whose behavior-level is in itself hybrid in the sense that it allows behaviors to be formulated in different formalisms or frameworks. In this study, we combine pieces of fuzzy control, basic geometrical reasoning, target-point planning, and map-based route planning. This freedom allows individual behaviors to be formulated quite simply. The restriction necessary to integrate their outputs is to require a common output format in terms of a tuple of control variable values.

Behaviors are simple to change and add. Due to the locality of (basic as well as complex) behavior influences, modifying or adding a behavior in the overall control system is simple in the sense that the ramifications of such a change are limited and their range is forseeable. The design scales up with adding more basic or complex behaviors within reasonable limits of computation load; running behaviors asynchronously in parallel is possible.

Complex behaviors fuse basic behaviors and abstract actions. The control architecture is such that the complex behaviors control the physical robot action. A complex behavior fuses all relevant basic behavior output values. Each and every complex behavior implements exactly one mid-level action. The domain modeler/robot programmer models and implements abstract actions, complex and basic behaviors and their mapping; the control architecture makes it easy to change, experiment with and optimize that mapping. The design scales up with adding actions and complex behaviors to a domain representation.

We have implemented a robot control system prototype that has allowed to examine these items to the point of running extensive experiments on a robot in a rich office environment. It remains a topic of future work to develop a reusable robot control architecture software that would support to be ported to different robot platforms.

Acknowledgements

Much of the background of the work reported here has been laid by the former ROB team of the AiS institute. We gratefully acknowledge this contribtion by Klaus Beck, Marion Finke, Michael Pauly, and Liliane Peters.

References

[Arg00a] A. Arghir. Aktionen des Roboters. http://ais.gmd.de/∼arghir/, 2000.

[Arg00b] A. Arghir. Planbasierte Navigation eines autonomen mobilen Roboters in einer aktiven, strukturierten Umgebung. Master's thesis, Univ. Bonn., Computer Sci. Dept., September 2000.

[Ark98] R. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, MA, 1998.

[Bee99] M. Beetz. Structured reactive controllers—a computational model of everyday activity. In *Proc. Third Intl. Conf. Autonomous Agents. (AA99)*, pages 228–235, 1999.

[BLSH95] B. Brunner, K. Landzettel, B.-M. Steinmetz, and G. Hirzinger. TeleSensorProgramming – a task-directed programming approach for sensor-based space robots. In *Proc. Int. Conf. on Advanced Robotics (ICAR-95)*, September 1995.

[MNPW98] N. Muscettola, P. Nayak, B. Pell, and B.C. Williams. Remote Agent: to boldly go where no AI system has gone before. *J. Artificial Intelligence*, 103:5–47, 1998.

[PFPB99] M. Pauly, M. Finke, L. Peters, and K. Beck. Control and service structure of a robot team. In *IEEE Intl. Conf. Intelligent Robots and Systems (IROS-99), Kyungju, South Korea, October 17-21, 1999*, pages 1069–1074, 1999.

[Rem00] Remote Agent Project. Remote agent experiment validation. http://rax.arc.nasa.gov/DS1-Tech-report.pdf, February 2000.

[Saf97] A. Saffiotti. Using fuzzy logic in autonomous robotics. *Soft Computing*, 1(4):180–197, 1997.

[TGVS99] N. Tschichold-Gürmann, S. J. Vestli, and G. Schweizer. Operating Experiences with the Service Robot MOPS. In *Proc. Third Europ. Workshop Advanced Mobile Robots (EUROBOT '99)*, pages 227–234. IEEE Press, 1999.

[TJ94] E. Tunstel and M. Jamshidi. Embedded fuzzy logic-based wall-following behavior for mobile robot navigation. In *1st Intl. Joint Conf. of NAFIPS/IFIS/NASA '94*, pages 329–330, San Antonio, TX., 1994.

[WvP94] G. Weiß. C. Wetzler and E. von Puttkamer. Positions- und Orientierungsbestimmung von bewegten Systemen in Gebäuden durch Korrelation von Laserradardaten. In P. Levi and T. Bräunl, editors, *Autonome Mobile Systeme 1994*, pages 55–64. Springer, Berlin, Heidelberg, 1994.

[XTF97] W.L. Xu, S.K. Tso, and Y.H. Fung. Sensor-based reactive navigation of a mobile robot. In *Intl. Conf. on Robotics and Automation (ICAR-97)*, pages 361–366, Monterey, CA, 1997.

[YZ92] R.R. Yager and L.A. Zadeh. *An introduction to fuzzy logic applications in intelligent systems.* Kluwer, Boston, 1992.