*Fundamenta Informaticae XX (2003) 1–22*
*IOS Press*


# Intelligent Execution Monitoring in Dynamic Environments

**Matthias Fichtner**

**Axel Großmann**

**Michael Thielscher**

*Department of Computer Science*
*Technische Universität Dresden*
*Dresden, Germany*

**Abstract.** We present a robot control system for known structured environments that integrates robust reactive control with reasoning-based execution monitoring. It provides a robot with a powerful method for dealing with situations that were caused by the interaction with humans or that are due to unexpected changes in the operating environment. On the reactive level, the robot is controlled using a hierarchy of low-level behaviours. On the high level, a logical representation of the world enables the robot to plan action sequences and to reason about the state of the world. If the execution of an action does not have the expected effect, high-level reasoning allows the robot to infer possible explanations and, if necessary, to recover from the failure situation. For the robot to act optimally, the discrepancies between the internal world model and the real world have to be detected and corrected. The proposed system obtains new information about the world by executing sensing actions (active perception) and by sensory interpretation during the robot's operation. It also takes into account temporal information about changes in the environment. All updates of the world model are performed in a way that the changes are consistent with an underlying action theory. Having implemented the proposed system on a common mobile robot platform, we demonstrate the value of intelligent execution monitoring by means of two realistic office delivery scenarios.

**Keywords:** Artificial Intelligence, Knowledge Representation

## 1. Introduction

In recent years, robotics has been subject to promising advances in sensor and actuator hardware, sensory processing techniques, and low-level control methods. Yet, if we want a mobile robot to perform complex tasks in real-world environments, we still face a number of problems. The information the robot has about

its operating environment might be out of date, incomplete, and uncertain. The execution of actions might fail due to a multitude of reasons. Ideally, we want the robot to reason about unexpected situations and to infer possible explanations in order to recover from the failure situation.

The capabilities described above require the robot to maintain information on its own state, usually obtained by processing the sensory data, as well as knowledge about the operating environment and the task at hand, commonly referred to as *world model*. To deal with the uncertainty in the robot's observations, it is common practice to represent state information such as the robot's location in the environment or the position of objects of interest using state enumeration and probabilities. Popular approaches include position probability grids [5] and particle sets [29]. On the other hand, as the robotic tasks become more complex, we would like to use reasoning and planning techniques. These require a symbolic representation instead. In fact, there is a large variety of symbolic reasoning and action planning methods for mobile robots. However, they are very difficult to integrate with the commonly used mechanisms for perception and reactive control due to the incompatibility of the underlying representations.

In order to make efficient use of high-level control for task planning and error recovery in dynamic real-time environments, temporal information about changes in the environment need to be incorporated into the world model. Therefore, we decided to realise the following two kinds of data acquisition. The robot may obtain information about the environment using its sensory processing system concurrently while executing a sequence of actions, or it may choose to execute specific sensory actions.

The aim of our work is to enable a mobile robot to perform action planning and reasoning-based execution monitoring in known structured environments, allowing for action failures due to the interaction with humans and dynamic changes in the environment. We present a hierarchical, modular control architecture that integrates low-level behaviours with a high-level controller, thus combining the robustness of reactive control with the power of intelligent reasoning. We have developed a layered scheme of execution monitoring. It allows the robot to find explanations for action failure using the logic-based world model and the history of the task execution. The capabilities of this novel kind of execution monitoring are evaluated using realistic scenarios for an office delivery robot.

The paper is organised as follows. In Section 2, we discuss related work on logic-based representations of dynamic information and on execution monitoring. In Section 3, we describe the main components of the proposed architecture and the representations and techniques used at the reactive and the abstract level. In Section 4, we describe the concepts and techniques for representing dynamic information, the acquisition of information while executing a plan, and active sensing. In Section 5, we demonstrate the functionality of the system using example scenarios. We conclude in Section 6.

## 2. Related Work

There is a vast literature on the integration of planning and reactivity in autonomous mobile robots, e.g., on the traditional sense-plan-act architectures as well as behaviour-based control and variations thereof [6, 17]. Several of the issues related to that are discussed in the following sections. However, a detailed discussion of the previous work on this topic goes beyond the scope of this paper. In this section, we therefore focus on literature on the realisation of reasoning-based execution monitoring and on logic-based representations of dynamic worlds.

## 2.1. Execution Monitoring

The problem of execution monitoring can be addressed in various ways. The individual solutions usually depend on the control architectures and the tasks they are used with. For example, there are efficient methods for handling errors in navigation tasks, such as the D*-algorithm [23]. Moreover, there is a large body of related work in the fields of fault detection and isolation (FDI) and industrial control. Since we want our solution to be applicable to complex delivery tasks, we address the problem of execution monitoring in a logical framework, going far beyond the requirements of pure navigation tasks.

There is no generally accepted definition of execution monitoring. De Giacomo and colleagues [3] define *execution monitoring* as "the robot's process of observing the world for discrepancies between the actual world and the robot's internal representation of it, and recovering from such discrepancies". In this work, we extend this notion in the way that the robot should come up with *explanations* for the detected discrepancies as well. Consequently, we can divide the overall process of execution monitoring into three steps: detecting discrepancies, explaining the situation, and launching a recovery procedure. They are discussed in the remainder of this section.

### Detecting Discrepancies

In general, the robot maintains a representation of its current state, such as the position in the environment, the distance to obstacles, and the state of the gripper. In addition, the execution of complex actions requires an internal model of the robot's interaction with the environment and a description of the task to be solved. By comparing all this information, it should be possible to detect erroneous situations. However, we do not expect such a comparison to be straightforward as the models and representations are likely to be complex and incompatible. In general, the representation of the robot's state will be layered and distributed. The control architectures usually include specialised modules for the interpretation of sensory data. At the low levels of control, probabilistic representations are commonly used to deal with incomplete and erroneous sensory information. At the highest level of control, in contrast, symbolic, logic-based representations are preferred.

Suppose the robot is to execute a sequence of actions. At the beginning, the robot generates an expectation as to the result of this sequence, i.e., the change of state caused by each action. These expectations are going to be layered and distributed as well. The anticipated effect of a go-to action at the low level, for example, is a change of the robot's position within a certain amount of time while maintaining a minimum distance to obstacles. At the highest level of control, expectations can be inferred using a knowledge base and an underlying action theory.

### Providing Explanations

Once an action turned out to have a different effect than expected, we would like to know the reason, i.e., find an explanation for the encountered discrepancy between state information and expectation. In general, this will require reasoning. This goes beyond the capabilities of reactive control and has to be performed at the highest level of control.

The robot can generally only observe symptoms of the current situation. For instance, a robot getting stuck could have been caused by a variety of reasons: a localisation failure, a visible object such as a person, an invisible obstacle such as a bump in the floor, an unexpected change of the environment such as a door being closed, and so on. The more relevant features of the environment are included in the

state information and the smaller the granularity of the world model and task description, the easier it is to make conjectures about the possible reasons of failure. On the other hand, to keep all this information up to date and consistent is challenging.

Decomposition planning meets the requirements mentioned above [30]. If, for example, a complex navigation task is broken down into smaller actions like door passings, corridor and room traverses, then failures can be substantiated with higher reliability. Default logic [14] provides means for the planning and reasoning module to abstract away from the sheer non-exhausting, but increasingly unlikely, set of preconditions, thus solving the qualification problem [12]. In case of unexpected situations though, these default assumptions must be checked according to an ordered preference list [11], thus providing the most likely explanation for action failure.

#### Recovering

Once an explanation of the current situation is found and the state information and world model are corrected, some recovery strategy is expected to remedy the failure. As suggested by Fernández and Simmons [4], this can be done by launching a predefined recovery plan. For example, when the robot notices that it ran into a dead end, it computes a path that brings it back on the original track and continues with the initial plan. Given a strong planning tool, instead of just correcting the mistake, we are able to find an optimal plan for the current situation, provided that the world model is correct. Suppose the robot did not run into a dead end, but found a shortcut. Now, the planner can provide a better solution if returning to the old track proves to be more costly.

### 2.2. Logic-based Representations of Dynamic Worlds

Previous logic-based representations of dynamic worlds are rooted in the general framework developed by Sandewall [18], in which dynamic information is modelled by autonomous processes that run in parallel and that may eventually trigger further changes in the environment. This technique has been integrated, for example, into situation calculus [15], event calculus [21], or fluent calculus [26]. Implementing these approaches, the agent programming and planning languages ConGolog [2] or FLUX [10] support the specification of concurrent processes and their effects.

A disadvantage of these methods is that all changes need to be explicitly inferred as effects of endogenous events, whose occurrence in turn needs to be derived from the ongoing processes. Robots which follow this approach in highly dynamic environments would be overwhelmed with constantly calculating all changes that happen around them. In order to avoid these frequent updates, we represent temporal knowledge about a dynamic property by attaching the time of its observation to the corresponding property. Decay of information will then be simulated by automatically forgetting about outdated fluents after a certain amount of time so that only recent knowledge is considered when devising plans.

## 3. Building the System

To put the functionality described above into practice, we added a high-level planning and reasoning component to a fairly standard hierarchical robot control system. In the following, we describe the parts of the system that are relevant specifically to execution monitoring.

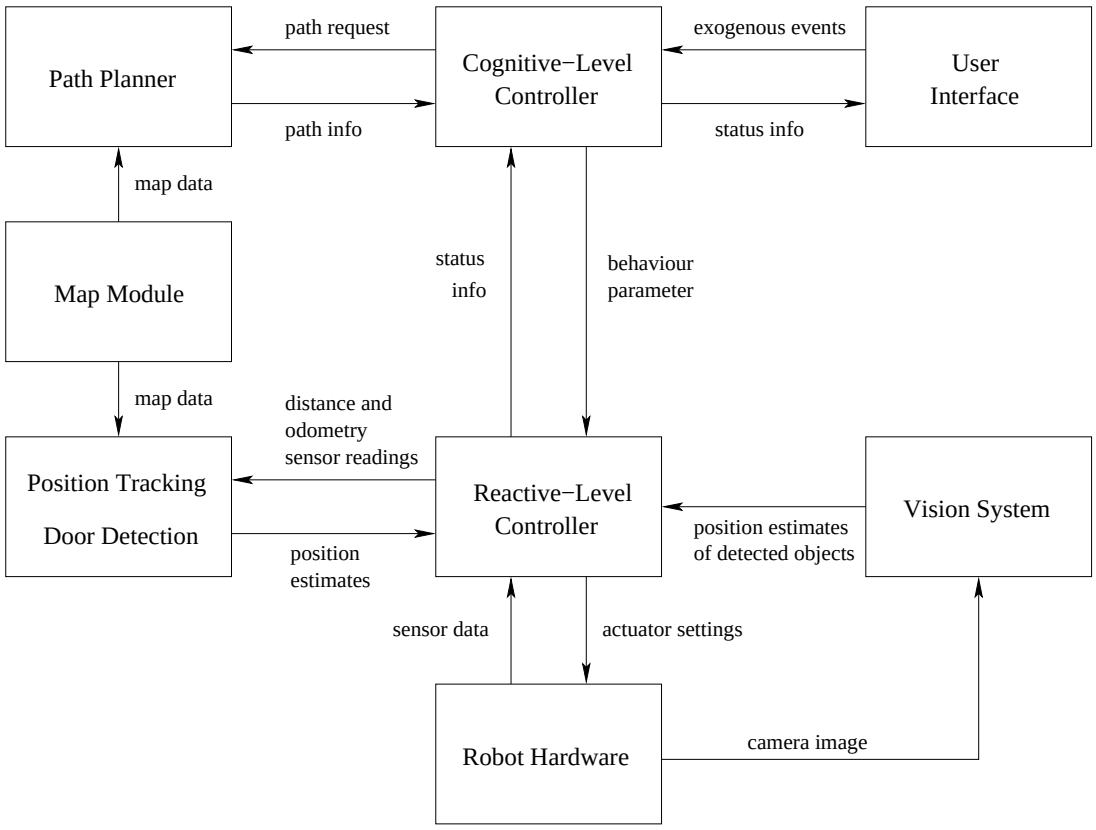


Figure 1. Control architecture of the robot.

## 3.1. System Architecture

The control architecture of the robot, as depicted in Figure 1, consists of several modules. The hardware controller talking directly to the robot's sensors (odometry, sonars, laser) and actuators (drive motors, gripper) is considered the lowest level of control.

The basic perceptual and behavioural functions of the robot are implemented by the reactive-level controller. Aiming at reactivity and robustness, we have chosen a behaviour-based approach. That is, the reactive controller includes several interacting, task-specific programs that are referred to as low-level behaviours. Each behaviour program takes the current sensor readings and the state information and computes target values of the robot's actuators. Individual behaviours can overwrite the output of other behaviours. The overall behaviour of the robot is controlled by activating or disabling individual low-level behaviours and by setting behaviour parameters such as target coordinates. Obstacle avoidance and local navigation (navigation inside a single room or corridor) are implemented in this way.

The robot's goal-oriented behaviour is directed by the cognitive-level controller. For example, the high-level controller allows for global navigation (navigation between rooms) by providing target coordinates to the reactive-level controller such as the location of doors and persons. In return, the low-level controller provides status information to the cognitive-level controller, notifying the planning and rea-

soning system about the success or failure of individual actions or the encounter of unexpected situations.

There are specialised sensor-processing modules for visual object detection, laser-based position tracking, and door detection. These components maintain a probabilistic representation of the detected objects, robot poses, and door angles, respectively. The reactive-level and the cognitive-level controllers depend on the information from the sensory interpretation modules. However, both control programs are unable to process complete, possibly multi-modal, probability distributions. Therefore, the probabilistic representations of the sensory modules are simplified by computing an approximation as a mixture of Gaussians. Only their mean and variance parameters are passed on to the controllers.

## 3.2. At the Lower Levels

Independently of the task to be performed, the safety of the robot and the environment has to be maintained at all times. Therefore, the reactive controller includes a set of low-level safety behaviours, e.g., for obstacle avoidance and velocity control, that cannot be switched off by higher levels of control. The other low-level behaviours are designed to achieve specific (parameterised) goals such as to travel to a target position or to pick up an object.

Suppose the robot is to execute a sequence of high-level actions. Then for each action, there is a designated process that supervises the execution of that action. This execution monitoring process invokes the appropriate low-level behaviours. The monitoring processes are implemented as finite state machines. Some states are common to all actions, others are specific to the task. For each monitoring process, there is a predefined set of exceptions, represented by status information. There are exceptions that are passed on by the low-level behaviours and there are exceptions that were detected by the sensor processing systems.

In the following, we illustrate this concept for the action of travelling to a given office. The corresponding low-level behaviour *GoToPos* consists of the following states:

| | |
|---|---|
| *Initialise* | Define target parameters |
| *ExecutePlan* | Initiate plan execution |
| *InProgress* | Execution in progress |
| *Success* | Execution terminated successfully |
| *FailureStalled* | Drive motors stalled |
| *FailureObstacle* | Path blocked by obstacle |
| *FailureDoor* | Path blocked by door |
| *Timeout* | Execution timeout |
| *Interrupt* | Execution interrupted |

We would like to stress that execution monitoring happens at either level of control. The individual low-level behaviours might be able to deal with the situation on their own. For example, if a small obstacle is detected by the distance sensors, the obstacle avoidance behaviour might be able to navigate around it. If the low-level behaviour terminates unsuccessfully, e.g., after bumping into an obstacle, the cognitive level is notified and has to take over.

### 3.3. At the Highest Level

The high-level controller maintains a symbolic world model. Reasoning about actions is used at this level to plan complex tasks and to generate expectations as to the effects of actions. When a discrepancy arises between the expectations and the robot's actual perception, the high-level controller uses its reasoning facilities to come up with suitable explanations and a recovery plan. As the underlying action theory we use the fluent calculus with its solution to the classical frame, ramification, and qualification problems. Our system builds on the inference engine FLUX for the fluent calculus [28].

#### Symbolic State Representations

A many-sorted predicate logic language, the fluent calculus extends the classical situation calculus by the concept of a state. Its signature includes the standard sorts FLUENT and STATE. The intuition is that a state is characterised by the fluents that hold in it. Formally, every fluent, i.e., term of sort FLUENT, is also of sort STATE. The signature of the fluent calculus moreover contains the constant $\emptyset : \text{STATE}$ (denoting the empty state) and the binary function symbol $\circ : \text{STATE} \times \text{STATE} \mapsto \text{STATE}$. The latter is usually written in infix notation and denotes the union of two states. If, for example, $\mathit{Carries}(\mathit{Book123}, \mathit{Sandra})$ is a FLUENT and $z$ a variable of sort STATE, then $\mathit{Carries}(\mathit{Book123}, \mathit{Sandra}) \circ z$ is also a state. A *finite* state is a term of the form $f_1 \circ \ldots \circ f_n$ where each sub-term $f_i$ is of sort FLUENT.

In order to capture the intuition of identifying states with the fluents that hold, the special connection function " $\circ$ " of fluent calculus obeys certain properties which closely resemble those of the union operation for sets.

**Definition 3.1.** Let $\mathit{Holds}(f, z)$ be an abbreviation for the equational formula $(\exists z')\, z = f \circ z'$. The *foundational axioms* $\mathcal{F}_{\mathit{state}}$ of the fluent calculus are:[1]

1. Associativity and commutativity,
$$(z_1 \circ z_2) \circ z_3 = z_1 \circ (z_2 \circ z_3)$$
$$z_1 \circ z_2 = z_2 \circ z_1$$
2. Empty state axiom,
$$\neg \mathit{Holds}(f, \emptyset)$$
3. Irreducibility,
$$\mathit{Holds}(f, g) \supset f = g$$
4. Decomposition,
$$\mathit{Holds}(f, z_1 \circ z_2) \supset \mathit{Holds}(f, z_1) \vee \mathit{Holds}(f, z_2)$$
5. State equality,
$$(\forall f)\, (\mathit{Holds}(f, z_1) \equiv \mathit{Holds}(f, z_2)) \supset z_1 = z_2$$
6. State existence,
$$(\forall P)(\exists z)(\forall f)\, (\mathit{Holds}(f, z) \equiv P(f))$$

[1] Below, $f, g$ are FLUENT variables while $z_1, z_2, z_3$ are STATE variables. Free variables are universally quantified.

The very last axiom stipulates the existence of a state for any set of fluents, whereby $P$ is a unary, second-order predicate variable of sort FLUENT.

For formalising the effects of actions, and ultimately for solving the classical frame problem [13], the fluent calculus uses a purely axiomatic characterisation of removal and addition of fluents to states. Let $z_1, z_2$ be states and $f$ a fluent, then the expression $z_1 - f = z_2$ (denoting removal of $f$ from $z_1$) is defined as an abbreviation for the formula

$$[z_2 = z_1 \vee z_2 \circ f = z_1] \wedge \neg \mathit{Holds}(f, z_2)$$

Let $\vartheta^- = f_1 \circ \ldots \circ f_n$ $(n \geq 1)$, then an inductive extension of this macro defines $z_1 - (\vartheta^- \circ f) = z_2$ as $(\exists z)\,(z_1 - \vartheta^- = z \wedge z - f = z_2)$. The addition of fluents, written $z_1 + f_1 \circ \ldots \circ f_n = z_2$, is defined as an abbreviation for the formula $z_1 \circ f_1 \circ \ldots \circ f_n = z_2$. Finally, the *update equation* $z_2 = (z_1 - \vartheta^-) + \vartheta^+$, which lays the foundation for an axiomatic solution to the frame problem, is defined as

$$(\exists z)\,(z_1 - \vartheta^- = z \wedge z + \vartheta^+ = z_2)$$

**Specifying the Effects of Actions**

The sorts ACTION and SIT (for situations as sequences of actions) are inherited from the situation calculus [16] along with the standard functions $S_0$ : SIT (denoting the initial situation) and *Do* : ACTION $\times$ SIT $\mapsto$ SIT (denoting the successor situation of performing an action). To this the fluent calculus adds the special function *State* : SIT $\mapsto$ STATE which denotes the state of the world in a situation. With this, a fluent $f$ is defined to hold in a situation $s$ thus:

$$\mathit{Holds}(f, s) \stackrel{\text{def}}{=} \mathit{Holds}(f, \mathit{State}(s))$$

It is important to note the intuition for states and situations. Being the fundamental representation, a state represents a (timeless) state of the world by means of a set of fluents denoting properties that hold in it. In contrast, a is like a "point in time" and is characterised by the agent's sequence of actions, rooted in the initial situation $S_0$. The basic expression $\mathit{State}(s)$ refers $s$ to a state $z$.

The fluent calculus provides a solution to the fundamental frame problem in classical logic. For every action, a so-called *state update axiom* uses the concept of fluent removal and addition to specify the effects of an action. For example, the action $\mathit{Receive}(o, p)$ of receiving object $o$ from person $p$ is specified by:

$$\begin{aligned}&\mathit{Poss}(\mathit{Receive}(o, p), s) \supset \\ &\quad (\exists p')\,(\mathit{Holds}(\mathit{Request}(p, o, p'), s) \wedge \\ &\qquad \mathit{State}(Do(\mathit{Receive}(o, p), s)) = (\mathit{State}(s) - \mathit{Request}(p, o, p')) + \mathit{Carries}(o, p')\,)\end{aligned}$$

Here, the standard predicate $\mathit{Poss}(a, s)$ denotes that action $a$ is possible in situation $s$. The update axiom describes the subsequent state, $\mathit{State}(Do(\mathit{Receive}(o, p), s))$, in terms of an update of the current state, $\mathit{State}(s)$, by the negative effect $\mathit{Request}(p, o, p')$ and the positive effect $\mathit{Carries}(o, p')$. That is, upon receiving $o$ from $p$ addressed to person $p'$, the robot carries this object for $p'$ and the corresponding delivery request is cancelled.

The general form of a state update axiom supports the specification of conditional as well as non-deterministic effects.

**Definition 3.2.** A *state update axiom* is a formula

$$\begin{aligned} \mathit{Poss}(A(\vec{x}), s) \supset\ & (\exists \vec{y}_1)\,(\Delta_1(\vec{x}, \vec{y}_1, s) \wedge \mathit{State}(\mathit{Do}(A(\vec{x}), s)) = \mathit{State}(s) - \vartheta_1^- + \vartheta_1^+) \\ & \vee \ldots \vee \\ & (\exists \vec{y}_k)\,(\Delta_k(\vec{x}, \vec{y}_k, s) \wedge \mathit{State}(\mathit{Do}(A(\vec{x}), s)) = \mathit{State}(s) - \vartheta_k^- + \vartheta_k^+) \\ & \vee \ldots \vee \\ & (\exists \vec{y}_n)\,(\Delta_n(\vec{x}, \vec{y}_n, s) \wedge \mathit{State}(\mathit{Do}(A(\vec{x}), s)) = \mathit{State}(s) - \vartheta_n^- + \vartheta_n^+) \end{aligned} \tag{1}$$

where $n \geq 1$ and, for each $1 \leq i \leq n$, $\Delta_i(\vec{x}, \vec{y}_i, s)$ is a first-order formula specifying additional conditions with free variables among $\vec{x}, \vec{y}_i, s$, and $\vartheta_i^+, \vartheta_i^-$ are finite states with variables among $\vec{x}, \vec{y}_i$.

The main theorem of the basic fluent calculus says that state update axioms solve the frame problem [24]:

**Theorem 3.1.** Let $\mathcal{F}_{\mathit{state}}$ be the foundational axioms of the fluent calculus, and consider a state update axiom of the form (1), then for any $1 \leq k \leq n$, the formula $\mathit{Poss}(A(\vec{x}), s) \wedge \Delta_k(\vec{x}, \vec{y}_k, s) \wedge \bigwedge_{i \neq k} (\forall \vec{y}_i)\, \neg\Delta_i(\vec{x}, \vec{y}_i, s)$ entails

$$\begin{aligned} \mathit{Holds}(f, \mathit{Do}(A(\vec{x}), s)) \equiv\ & \mathit{Holds}(f, \vartheta_k^+) \\ & \vee \mathit{Holds}(f, s) \wedge \neg \mathit{Holds}(f, \vartheta_k^-) \end{aligned}$$

**Specifying Action Preconditions**

The cognitive controller requires precondition and effect specifications of each high-level action. To account for unexpected action failure, we make the distinction between normal and abnormal preconditions. The former need to be ascertained before an action can be planned while the latter are assumed away by default but serve as possible explanations in case the action surprisingly fails. For example, the following axiom specifies the preconditions of the action $\mathit{Deliver}(o, p)$ of delivering object $o$ to person $p$:

$$\begin{aligned} \mathit{Poss}(\mathit{Deliver}(o, p), s) \equiv\ & \mathit{Holds}(\mathit{Carries}(o, p), s) \wedge \\ & (\exists r)\,(\mathit{Holds}(\mathit{InRoom}(r), s) \wedge \mathit{Office}(r, p)) \wedge \\ & \neg \mathit{Holds}(\mathrm{Ab}(\mathit{Traceable}(p)), s) \wedge \neg \mathit{Holds}(\mathrm{Ab}(\mathit{NotLost}(o)), s) \end{aligned} \tag{2}$$

The fluent $\mathrm{Ab}(x)$ indicates the presence of abnormal condition $x$. Hence, the precondition axiom says that normally a delivery is possible if the robot carries the object in question, $o$, and happens to be in the office $r$ of the recipient $p$. However, the action fails under the unusual circumstances that the respective person is not traceable or the object has been lost.

Actions sometimes fail to produce the intended effect. For example, in exceptional cases a delivery

may leave the recipient with the wrong item:

$$\begin{aligned}
&\mathit{Poss}(\mathit{Deliver}(o,p),s) \supset \\
&\quad \neg(\exists o')\, \mathit{Holds}(\mathrm{Ab}(\mathit{Deliver}(o',p)),s) \wedge \\
&\qquad \mathit{State}(\mathit{Do}(\mathit{Deliver}(o,p),s)) = \mathit{State}(s) - \mathit{Carries}(o,p) \\
&\quad \vee \\
&\quad (\exists o',p')\ \big(\ \mathit{Holds}(\mathrm{Ab}(\mathit{Deliver}(o',p)),s) \wedge \\
&\qquad \mathit{Holds}(\mathit{Carries}(o',p'),s) \wedge o \neq o' \wedge \\
&\qquad \mathit{State}(\mathit{Do}(\mathit{Deliver}(o,p),s)) = \mathit{State}(s) - \mathit{Carries}(o',p')\ \big)
\end{aligned} \tag{3}$$

The condition $\mathit{Holds}(\mathrm{Ab}(\mathit{Deliver}(o',p)),s)$ represents the abnormal case of delivering the wrong item $o'$ to person $p$ in situation $s$. Abnormal conditions in state update axioms, too, are assumed away by default but may serve as explanations for observed discrepancies between the expected and the actual outcome.

Possible indirect effects of actions are specified by causal relationships. This solves the ramification problem [7]. For example, suppose the robot searches for an object $o$ among the group of people in some room $r$. Whenever $\mathit{Has}(p',o)$ becomes true, stating that person $p'$ is in possession of the object, then all other previously considered possibilities of people having $o$ are ruled out:

$$\mathit{Holds}(\mathit{MightHave}(p,o,r),s) \supset \mathit{Causes}(\mathit{Has}(p',o), \neg\mathit{MightHave}(p,o,r),s)$$

Here, the standard macro definition $\mathit{Causes}(e,r,s)$ means that effect $e$ causes indirect effect $r$ in situation $s$.

**Explaining Action Failures**

Action failure is explained on the basis of the various abnormalities that have been specified for each action. Following the solution to the qualification problem developed in [27], abnormal conditions $\mathrm{Ab}(x)$ are assumed away by default unless there is evidence to the contrary. We use a non-monotonic default theory to this end. Whenever the observations suggest a discrepancy between the default expectations and the actual world, the default theory entails that one or more default assumptions no longer hold.

Suppose, for example, a delivery action cannot be performed although the robot believes that it carries the right object and is in the right office. Precondition axiom (2) then offers two explanations by means of the respective positive instances of $\mathrm{Ab}$. In addition, if the failed action occurs after other deliveries, then update axiom (3) offers a further explanation, namely, that the object was previously accidentally delivered to the wrong person. If so, the update axiom implies that the regular precondition $\mathit{Holds}(\mathit{Carries}(o,p),s)$ in (2) does actually not hold. In this way, the high-level controller uses its reasoning facilities to generate suitable explanations for the encountered failure [11]. By appealing to prioritised default logic [1], one can specify qualitative knowledge of the relative likelihood of the various explanations for abnormal qualifications. The accompanying concept of preferred extensions then helps selecting the most likely explanations. In cases where it is impossible to provide an exhaustive specification of the reasons for a particular action to fail, a special $\mathrm{Ab}$ instance can be added to the precondition axiom indicating an inexplicable failure. If this abnormality is specified as being least preferred, then the controller falls back upon it only if all other explanations fail.

```
loop(S,Z):-
  ( /* Stationary */
    notify_reach(Z, URL)    -> As = URL;
    call_help(P,Z)          -> As = [call_help(P)];
    search_fail(O,P,Z)      -> As = [email(O,P)];
    delivery(O, P, Z)       -> As = [deliver(O,P)];
    receipt(O, P, Z)        -> As = [receive(O,P)];
    /* Knowledge Acquisition */
    search(SearchDialog,Z)  -> As = SearchDialog;
    /* Navigation */
    continue(GoAct, Z)      -> As = [walk_on|GoAct];
    /* Otherwise */            As = [idle]
   ),
   execute(As, S, Z).
```

Figure 2. Implementation of the main loop.

**Planning**

A controlling mechanism that monitors action execution inevitably requires a high amount of reactivity. On the low level, a set of interacting behaviours seem to meet this requirement. In an analogous manner, an abstract task planner should be able to react on events that might affect the current agenda. Action failures and general world changes require re-planning for both successful accomplishment and efficiency reasons.

The maintenance of a state and a set of abstract state evaluation functions consulted during every action-execution cycle are the base for the planning loop. Once a critical world change is realised, the current agenda is dropped and the planner is invoked again.

The implementation of the main loop, predicate `loop/2`, is given in Figure 2. The planner investigates the current state according to various criteria of decreasing priority. The predicates on the left-hand side of the arrows can be understood as diagnostic functions of the current state. Predicate *NotifyReach* holds if people became out of reach recently. The second argument *URL* is a sequence of notification actions about the unreachable persons. Predicate *CallHelp* succeeds if it is necessary to call for help in the current state. *SearchFail* notifies the originator of the search request by email if none of the possible candidates has the desired item. If delivery or receipt is possible, the according actions are performed. If no stationary action is launched, that is, if all the previous state diagnostic predicates failed, then predicate *Continue* is invoked. Depending on the current state (e.g., the position), a rather complex path planning procedure is invoked and returns, if possible, a navigation action sequence (consisting of *Goto*, or several *Enter* and *GotoDoor* actions). If all else fails, the robot goes *Idle*. If stationary actions are possible, then these are executed. The plan can either be a single action or an action sequence.

In the examples below, we illustrate the use of temporal information by means of knowledge about changing doors. Keeping track of multiple requests, our delivery robot has to serve a number of people in general. For finding an optimal plan, one usually has to consider a number of aspects. While there are general aspects like high efficiency and low risk, domain dependent criteria like guarantee of service

also constrain planning. Regarding navigation, we impose a ranking among the destination locations.

Predicate *Continue* computes an optimal path plan given the current tasks and situation. The planning strategy behind it exploits temporal knowledge about changing door states and blocked doors in the following way: First, for all possible routes to a destination, the planner prefers a path that involves doors currently known to be open and reachable. Alternatively, a valid route is found if it does not contain doors that are known to be closed or blocked. In the remaining case, no door is excluded from the route plan. Of course, a sensing action executed in front of each door will provide the knowledge of the state of any door at execution time just before entering the door. In other words, we prefer routes the robot recently found to work out. Please note, that no option is ruled out, but rather the best one is scheduled first.

# 4. Extensions for Modelling a Dynamic World

To consider also the dynamic properties of the world in action planning and reasoning, this information must be part of the robot's state description. Consequently, we have to distinguish between recent and outdated information, because otherwise invalid knowledge may mislead the robot or prevent it from finding a plan at all. In the following, we show how temporal knowledge about dynamic properties of the world can be represented in FLUX, together with a mechanism for its maintenance.

## 4.1. Representing Temporal Information

From the perspective of a robot that has incomplete knowledge of its environment, the world can be in any of several possible states. When a robot uses its sensors to gain more knowledge of its environment, it can reduce the set of possible states to those that satisfy the new information. This, in a nutshell, is the approach to representing and reasoning about knowledge and sensing in the fluent calculus presented in [25], rooted in the situation calculus-based approach [19, 20]. Formally, the fluent calculus has been extended by the predicate $\mathit{KState} : \text{SIT} \times \text{STATE}$. An instance $\mathit{KState}(s, z)$ means that, according to the knowledge of the agent, $z$ is a *possible* state in situation $s$. For instance, the axiom

$$(\forall z)\,(\mathit{KState}(S_0, z) \supset \mathit{Holds}(\mathit{Carries}(\mathit{Book123}, \mathit{Sandra}), z)) \tag{4}$$

says that the robot carries *Book123* for *Sandra* in all states that are possible in the initial situation. Hence, the robot can be said to *know* this fact in $S_0$. Generally, a fluent is known to hold in a situation (not to hold) just in case it is true (false, respectively) in all possible states:

$$\mathit{Knows}(f, s) \stackrel{\text{def}}{=} (\forall z)\,(\mathit{KState}(s, z) \supset \mathit{Holds}(f, z))$$

This macro can be inductively generalised to the knowledge of logical combinations of state properties:

$$\mathit{Knows}(\varphi, s) \stackrel{\text{def}}{=} (\forall z)\,(\mathit{KState}(s, z) \supset \mathit{HOLDS}(\varphi, z))$$

$$\begin{aligned}
\mathit{HOLDS}(f, z) &\stackrel{\text{def}}{=} \mathit{Holds}(f, z) \quad \text{if } f \text{ fluent} \\
\mathit{HOLDS}(\neg\varphi, z) &\stackrel{\text{def}}{=} \neg\mathit{HOLDS}(\varphi, z) \\
\mathit{HOLDS}(\varphi_1 \vee \varphi_2, z) &\stackrel{\text{def}}{=} \mathit{HOLDS}(\varphi_1, z) \vee \mathit{HOLDS}(\varphi_2, z) \\
\mathit{HOLDS}((\exists x)\,\varphi, z) &\stackrel{\text{def}}{=} (\exists x)\,\mathit{HOLDS}(\varphi, z)
\end{aligned}$$

The representation of incomplete knowledge by possible states provides a means to reason about actions from the subjective perspective using *knowledge update axioms* (KUAs) [25]. Like a state update axiom, a KUA is used to specify the effects of an action. However, rather than describing the update of the actual state of the world, $\mathit{State}(s)$, a KUA defines an update of the totality of possible states. In this way, both actions that change the state as well as pure sensing actions can be specified.

While the approach of [25] facilitates the representation of incomplete state knowledge, it rests on the assumption that knowledge, once established, does not change unless an action is performed which effects a change. Consequently, if at some point the robot senses, e.g., that a particular door is open, then it assumes that this door is still open in any later situation, unless this situation includes an action which affects the state of this door. Hence, this representation is unsuitable for dynamic worlds, in which properties may undergo frequent changes which a robot is usually not aware of.

We have developed a new way of dealing with uncertain information, in which knowledge of a dynamic property of the world is represented by attaching the time of its observation. Formally, for each fluent $f(\vec{x})$ which we want to treat as dynamic, we introduce the two fluents $f(\vec{x},t)$ and $\overline{f}(\vec{x},t)$. If $f(\vec{x},t)$ is true in all possible states of a situation $s$, then this indicates that $f(\vec{x})$ has been observed to be true at time $t$. Likewise, if $\overline{f}(\vec{x},t)$ is true in all possible states of a situation $s$, then $f(\vec{x})$ has been observed to be false at time $t$. For each such pair of fluents, we introduce the following three domain axioms, which stipulate that the time-point of an observation is always unique and that not both $f$ and $\overline{f}$ can be true in a possible state:

$$\begin{aligned}
&\mathit{Knows}(f(\vec{x},t_1) \wedge f(\vec{x},t_2)\,,s) \supset t_1 = t_2\\
&\mathit{Knows}(\overline{f}(\vec{x},t_1) \wedge \overline{f}(\vec{x},t_2)\,,s) \supset t_1 = t_2\\
&\neg\mathit{Knows}(f(\vec{x},t_1) \wedge \overline{f}(\vec{x},t_2),s)
\end{aligned}$$

When a robot senses information about a dynamic property, it records the observation by updating the set of possible states accordingly. The following generic KUA defines the effect of sensing whether a fluent $f(\vec{x})$ holds. It uses the function $\mathit{Time}(\mathit{Do}(a,s))$ which denotes the time at which action $a$ is performed in situation $s$:

$$\begin{aligned}
&\mathit{Poss}(\mathit{Sense_}f(\vec{x}),s) \supset\\
&\quad\Big(\ \mathit{KState}(\mathit{Do}(\mathit{Sense_}f(\vec{x}),s),z') \equiv\\
&\qquad(\exists t,z)\ \Big(\mathit{KState}(s,z)\ \wedge\ t = \mathit{Time}(\mathit{Do}(\mathit{Sense_}f(\vec{x}),s)) \wedge\\
&\qquad\qquad(\exists z'')\ \big((\forall t_1,t_2)\, z'' = z - f(\vec{x},t_1) \circ \overline{f}(\vec{x},t_2) \wedge\\
&\qquad\qquad\qquad\big[\ z' = z'' + f(\vec{x},t)\ \wedge\ \mathit{Holds}(f(\vec{x}),s) \vee\\
&\qquad\qquad\qquad\ \ z' = z'' + \overline{f}(\vec{x},t)\ \wedge\ \neg\mathit{Holds}(f(\vec{x}),s)\ \big]\big)\Big)\Big)
\end{aligned} \tag{5}$$

By this KUA, previous knowledge about fluent $f$ is positively discarded, and $f$ is respectively true or false in all possible states $z'$, depending on whether the fluent actually holds in situation $s$. The correctness of this axiom relies on the aforementioned domain axiom which says that $f(\vec{x},t_1)$ or $\overline{f}(\vec{x},t_2)$ can be true for at most one time-point. The universal quantification over $t_1,t_2$ then ensures that the respective instance no longer holds in state $z''$.

The parameter $t$ can be taken as a measure for the reliability of the knowledge of a dynamic fluent. Decay of information is simulated by forgetting about outdated fluents after a certain amount of time. By

regularly retracting fluents considered to be out of date, we make sure that the current state description only contains recent knowledge. Hence, no additional checks for recency of fluents are required for reasoning. Nevertheless, any observation made in the past is crucial for explaining action failures as they directly influence the planner's computation, and must be kept in the situation history.

In general, the sensory processing modules operate in two modes: autonomous broadcasting of relevant information (concurrently to the execution of actions) and a direct query of the current belief – both will be described next.

## 4.2. Gathering Information while Executing a Plan

Suppose a dynamic world in which doors might be open or closed and unknown obstacles might block the way. Following the schematic formula (5), a possible knowledge update axiom for such a domain is:

$$
\begin{array}{l}
\mathit{Poss}(\mathit{SenseDoor}(d), s) \supset \\
\quad \Big( \mathit{KState}(\mathit{Do}(\mathit{SenseDoor}(d), s), z') \equiv \\
\qquad (\exists t, z)\ \Big( \mathit{KState}(s, z)\ \wedge\ t = \mathit{Time}(\mathit{Do}(\mathit{SenseDoor}(d), s)) \wedge \\
\qquad\qquad (\exists z'')\ \big( (\forall t_1, t_2, t_3)\, z'' = z - \mathit{Op}(d, t_1) \circ \mathit{Cl}(d, t_2) \wedge \\
\qquad\qquad\qquad \big[\ z' = z'' + \mathit{Op}(d, t)\ \wedge\ \mathit{Holds}(\mathit{Op}(d), s) \vee \\
\qquad\qquad\qquad\ \ z' = z'' + \mathit{Cl}(d, t)\ \wedge\ \neg \mathit{Holds}(\mathit{Op}(d), s)\ \big] \big) \Big) \Big)
\end{array}
$$

where *Op* and *Cl* abbreviate *DoorOpen* and *DoorClosed*, respectively.

Once a sensory processing module has gained sufficient confidence about a certain property of the environment by integrating observations over time, the new belief is announced to all listening modules including the high-level controller. FLUX in turn interrupts the current action and incorporates the new observations in the manner of the KUA above. Since new observations may render the current plan obsolete or suboptimal, a new plan is computed to accommodate to the new situation. Hence, the robot always follows an optimal plan according to its goals and current observations.

## 4.3. Active Sensing

Besides the concurrent, autonomous broadcast of perception results, the sensory processing modules provide information on demand. The high-level controller may request the execution of a sensing action. Subsequently, the corresponding sensing module would then be queried about its current belief.

Revisiting the example above, the action *SenseDoor* asks the door sensing module about the current state of the door at hand. The result is then incorporated into the world model by means of the KUA above for changing doors. Before the robot can pass through a door, it has to obtain the current state of this door for safety reasons. While $\mathit{DoorOpen}(d, t)$ is part of the preconditions for the enter-door action, the KUA exhibits a non-deterministic behaviour with unknown result at the time of planning, but is resolved by *SenseDoor* at the time of execution. Only if the queried property turns out to hold, the preconditions of subsequent actions are fulfilled and the robot can continue; otherwise FLUX tries to find an alternative plan based on the revised situation.

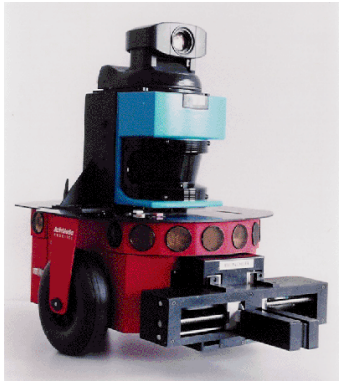

Figure 3. Pioneer 2 mobile robot equipped with a colour camera, a laser range finder, a ring of sonar sensors, and a gripper.

## 5. Experiments

A detailed evaluation of the presented robot control system and its execution monitoring techniques renders very difficult. In many cases it is only possible to analyse specific aspects of the system or particular techniques separately, as a layered control system bears complex interactions between various aspects and functionalities at the same time. Moreover, a single functionality is usually distributed among several modules of different layers. Consequently, it is difficult to compare the system directly with implementations of other control architectures. Nevertheless, distinctions regarding specific features will be summarised in Section 6.

The aim of the experiments reported here is to evaluate the control system with respect to its ability to deal with dynamic changes of the world and the use of intelligent execution monitoring for complex delivery tasks. The choice of the possible test scenarios is usually restricted by the perceptual capabilities of the robot platform. In our case, the robot control system was used on a Pioneer 2 mobile robot, as shown in Figure 3. Due to the constraints imposed by the robot hardware, we focused on office-delivery scenarios. Here, a mobile robot receives requests from users and is supposed to take items from one place to another and to search for items among persons in the offices.

Unfortunately, no system for on-line tracking of persons was available to us at the time of the experiments. Therefore, we modelled the dynamic properties of the world using unknown obstacles blocking the way of the robot, as it may be caused by people or moved pieces of furniture, using doors that open and close during the robot's operation, and using recipients that change their location unexpectedly. Despite the simple nature of the environment dynamics, our experimental setup shares important features and requirements with other real-world robotic applications:

- The robot has little prior knowledge about dynamic properties and objects of the world. Such information is gathered during the operation.
- The robot's world model will be incomplete because real environments are not fully observable and perceptual capabilities are limited in range, modalities and quality.

As a consequence, the robot experiences unexpected changes of the world according to its world model

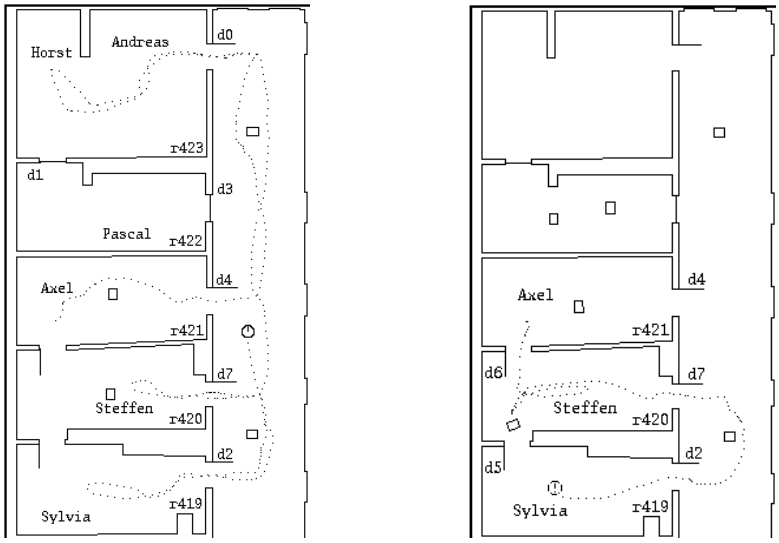


Figure 4. Office environment used in the experiments consisting of five rooms and a hallway.

as well as action failures. It is our opinion that this demands for intelligent execution monitoring.

The presented control system is portable to other mobile robot platforms. Solely the reactive-level controller contains programs that were specifically optimised for the robot platform at hand. The proposed techniques for intelligent execution monitoring are independent of the robot platform. Specifically, the kernel of the FLUX controller comprises the foundational functionality without being domain dependent – the task and domain are axiomatised separately.

We proceed with the description of two example scenarios highlighting different features of the robot control system. Thereby, we illustrate the need for intelligent execution monitoring and the utilisation of temporal knowledge about dynamic properties of the environment.

## 5.1. Dealing with Temporal Knowledge

In the following, we exemplify our statement that maintaining knowledge about dynamic properties of the operating environment is of advantage in practice. The example also demonstrates how an agent can exploit the aspect of recency of information.

Consider the situation depicted on the left-hand side of Figure 4. In this example, the robot is situated in room `r421`, at Axel's place, initially and it knows that door `d2` is closed. Upon receiving a request to bring a cup of coffee from Horst to Sylvia, the high-level planner is invoked. The lack of (recent) knowledge on door states prevents finding a valid path considering only doors that are known to be open. Following the second heuristic, the planner computes a valid plan excluding doors that are known to be closed:

```
State: [request(horst,coffee,sylvia), door_open(d2,24), in_room(r421), at(axel)|Zp]
Agenda: [walk_on, gotodoor(d4), sense_door(d4), enter(corr), gotodoor(d0), sense_door(d0), enter(r423),
         goto(horst)]
```

Hereof, the second argument of predicate `door_open` denotes the time of the observation in seconds (cf. Section 4.1). As usual in FLUX, the tail variable `Zp` of the state indicates that the robot has incomplete

knowledge. The agenda shows the plan that FLUX determined, a path plan entering two doors. For a mobile robot in real-world environments, the complexity of the action to enter a door necessitates to split the action into distinct parts executed subsequently. This is due to noisy sensor data and imprecise localisation. Therefore, entering a room comprises the high-level actions to position the robot in front of the door, to check the current state of the door, and, in case the door is open, to actually steer the robot through. While the robot travels, the low-level behaviours bypass obstacles on its way autonomously. The interaction within the system architecture is described in more detail in the second example.

**Active sensing.** Having arrived at door `d4`, a sensing action explicitly determines the state of this door by querying the door sensing module. If it gained enough confidence in the door state, this observation is returned to the cognitive level, which in turn updates its internal world model, $Z$. Since there was no previous knowledge about the state of the door at hand, $Z$ is enriched by this observation with a time stamp attached, `door_open(d4,79)`. Now knowing that the door ahead is open, the preconditions for action `enter(corr)` are satisfied and the robot continues executing the current plan.

**Concurrent sensing.** At door `d3`, the door sensing module autonomously signals that `d3` is open upon which FLUX computes a new plan since any observation may influence the current plan (cf. Section 5.3). Here, the new plan equals the continuation of the previous one due to independence of the state of door `d3`, yielding:

```
State: [door_open(d3,106), door_open(d4,79), in_room(corr), request(horst,coffee,sylvia),
        door_open(d2,24)|Zp]
Agenda: [walk_on, gotodoor(d0), sense_door(d0), enter(r423), goto(horst)]
```

**Planning with multiple requests.** Suddenly, Steffen requires a book which should be on Andreas' desk. Since both scheduled providers, Horst and Andreas, can be reached (by default) according to the agent's knowledge, the planner chooses Andreas, being the nearest person, as its first target while keeping track of remaining requests. Next, sensing the state of door `d0` provides the prerequisites to enter room `r423`. Having arrived at Andreas' desk and subsequently received the book for Steffen, the robot derives a new plan and determines to pick up the coffee from Horst next. On the way to Horst, the door sensing module signals that door `d1` appears to be closed. While the new observation does not interfere with the current plan, the robot could derive that it cannot enter room `r422` given its current knowledge on door states. After the robot successfully received the coffee from Horst, two deliveries are due, from which the planner selects Steffen's request according to the strategy of serving the nearest recipient that is reachable first. While passing by door `d1` on its way back, the door sensing module detects `d1` to still be closed. By means of the knowledge update axiom for `sense_door` (cf. Section 4.2), previous knowledge about `d1` is discarded and the new observation `door_closed(d1,250)` is asserted. On its way to Steffen, the robot observes that `d3` is now closed. The corresponding world model and agenda are:

```
State: [door_closed(d3,330), door_open(d0,288), in_room(corr), door_closed(d1,250),
      carries(horst,coffee,sylvia), carries(andreas,book,steffen), door_open(d4,79), door_open(d2,24)|Zp]
Agenda: [walk_on, gotodoor(d7), sense_door(d7), enter(r420), goto(steffen)]
```

**Exploiting temporal knowledge.** As Sylvia is still waiting for her coffee, the robot successfully delivers the book to Steffen. The FLUX planner computes a path to Sylvia and succeeds considering only such doors that are known to be open since it recently entered through `d7`, while `d2` is still known to be open from the initial situation. Although directly entering Sylvia's room via door `d5` would result in the shortest path, the planner's strategy prefers paths through doors that were open recently and proceeds to Sylvia via the corridor.

After our diligent robot had delivered the coffee to Sylvia, Pascal requested a copy of a journal from Axel. Again, the planner selects a path through the corridor instead of risking the encounter of a closed door in the shortcut via Steffen's office. The left-hand side of Figure 4 illustrates the current situation. Meanwhile, FLUX forgets about the open door `d0` and also about the closed doors `d1` and `d3`. This immediately enables the robot to start the delivery of the journal to Pascal as the state description shows:

```
State: [carries(axel,journal,pascal), at(axel), door_open(d4,699), in_room(r421), door_open(d2,637),
        door_open(d7,468)|Zp]
Agenda: [walk_on, gotodoor(d4), sense_door(d4), enter(corr), gotodoor(d3), sense_door(d3), enter(r422),
         goto(pascal)]
```

Please note that without a mechanism for forgetting dynamic properties of the environment which are represented in the world model, outdated knowledge can prevent the high-level controller to find a plan, such that the robot would wait forever in this example. In the given solution for changing doors, the reachability of rooms is checked again after some time. In the current implementation, the lifetime of temporal information is an interval of a fixed size which is sufficient for the robot to travel to the remote end of the hallway and back.

## 5.2. Explaining Unexpected Situations

In the following, we want to exemplify the application of the reasoning capabilities of FLUX together with the layered scheme of execution monitoring to finding possible explanations for an action failure and an appropriate recovery strategy thereafter.

Starting at Axel's place and knowing nothing about dynamic properties of the world, suppose the robot receives requests from Axel to take `book1` to Steffen and `book2` to Sylvia. The FLUX planner chooses to visit Steffen first taking the shortcut through door `d6` according to the built-in strategy explained above. Steffen being delighted about the book, the robot plans to proceed to Sylvia carrying the remaining book by taking the shortcut through door `d5`.

**Computing an explanation.** Since an unexpected obstacle completely blocks the way to door `d5`, the obstacle avoidance behaviour detects a failure to reach the desired door node of the underlying topological graph and interrupts the navigation behaviours. Upon this crucial incident, the reactive controller informs FLUX about the failure together with lists of nodes that have been reached so far and remaining nodes of the topological path to the destination. FLUX in turn determines the potential actions that might have caused the failure and recursively searches for an explanation by inserting uninstantiated abnormality fluents. An abnormality that unifies with the state update axiom of one of the potential actions provides a possible explanation for the failure and is registered. Re-planning then determines a recovery strategy based on the new situation and computes an alternative path plan via the corridor.

The doors `d7` and `d2` turn out to be open so that our busy robot can reach Sylvia. This situation is depicted in the right-hand side of Figure 4 and comprises:

```
State: [at(sylvia), door_open(d2,208), in_room(r419), door_open(d7,152), door_open(d6,112),
        carries(axel,book2,sylvia), ab(reachable(d5,r420),51)|Zp]
Agenda: [deliver(book2,sylvia)]
```

Please note that abnormality fluents are considered as dynamic properties of the world that may change, which is why a time stamp is attached in order to invalidate them later.

**Reasoning about action failure.** When the robot arrived at Sylvia's desk, she does not accept the book unexpectedly. The analysis of the robot's action history reveals two possible explanations: The

item might have been lost on the way, or some person took the wrong book during a previous delivery, both of which the robot cannot sense. FLUX assumes that the latter happened since both `book1` and `book2` were in the tray when the delivery to Steffen took place, which is the preferred explanation if applicable. The robot will discover which explanation actually holds later. The computed recovery plan comprises to return to Steffen in order to pick up `book2` for Sylvia and to deliver `book1` that seems to be in the tray at that time. The corresponding state description is:

```
State: [request(steffen,book2,sylvia), at(sylvia), door_open(d2,208), in_room(r419), door_open(d7,152),
        ab(reachable(d5,r420),51), door_open(d6,112), carries(axel,book1,steffen)|Zp]
Agenda: [walk_on, gotodoor(d2), sense_door(d2), enter(corr), gotodoor(d7), sense_door(d7), enter(r420),
         goto(steffen)]
```

Avoiding the blocked path and using the doors that were recently open, the robot carries `book1` to Steffen which he regrets to have mixed up. By that, the explanation proves to be correct. Otherwise, `book2` being lost would serve as an alternative explanation. Later, `book2` has been successfully delivered to Sylvia, eventually.

We would like to point out that only for the purpose of understanding we have demonstrated the two aspects of dealing with temporal information and explaining action failures in separate examples. In practice they are employed simultaneously.

### 5.3. Conclusions

The performance of the robot control system was evaluated in several runs using the conditions of the example scenarios mentioned above. The robot was employed in the office environment of our institute comprising a corridor and several adjacent rooms. The layout is given in Figure 4. The performance of the robot in these runs exhibited no significant differences to the description above – it successfully accomplished the requests and gracefully reacted to unexpected situations. The experiments demonstrated the benefits of intelligent execution monitoring being robust action execution, recognition of action failures and optimal response to failure situations according to the world model. We were also able to show that representing information about dynamic properties of the world can be of advantage and is necessary for acting efficiently. The practical application of the proposed control system and techniques helped us to identify the following issues.

The possible scenarios a robot is applicable to are mostly constrained by the robot's limitations regarding perception and actuators. For example, the Pioneer 2 we used in the experiments requires to move on an even surface. Because door sills cannot be recognised, such obstacles either have to be known in advance in order to avoid them, or can be learnt from experience, i.e., once the action to enter a particular door bearing an invisible door sill failed, this knowledge is represented in the world model and is taken into account in future planning.

As our robot cannot sense the state of its tray, it is unable to recognise whether a person took the wrong item, for example. By means of intelligent execution monitoring, we can infer certain information given its history of actions and observations and the domain axiomatisation; for details, see Section 5.2.

In practical application, the perceptual limitations directly constrain the types of dynamic objects a robot can recognise and handle. We want to develop further specialised object recognition modules as well as more generic means to recognise objects of regular shape.

When a change of a dynamic property of the world is recognised, the high-level controller can either react immediately, or defer its processing until the next plan is to be computed. On the one hand, optimal

behaviour is gained with respect to always choosing the best action according to the world model in the first case. On the other hand, this kind of reactivity of the high-level controller can become intractable in highly dynamic environments. Deferring to take new relevant observations into account will result in inefficient performance due to the increased risk of impending action failures.

Handling objects of dynamic occurrence and properties demands for a solution of the so-called anchoring problem, that is to identify and maintain the correspondence between percepts of objects and its symbolic representation at the cognitive level consistently. We want to develop our current, rigid approach toward a flexible and generic solution.

## 6. Summary and Future Work

We have presented a layered scheme of execution monitoring for mobile robots operating in known structured environments, allowing for action failures due to the interaction with humans and dynamic changes in the environment. It allows the robot to find explanations for action failures using a logic-based world model and the history of the task execution. In the experiments, we were able to illustrate the advantages of reasoning-based execution monitoring over reactive control methods. The implementation is based on the fluent calculus along with its augmentations.

Our approach is similar to the systems by Shanahan [22], Haigh and Veloso [9], and Hähnel and colleagues [8] in the sense that all use high-level planning on real robots. Shanahan's Khepera robot based on the event calculus makes heavy use of abductive planning and explanations of failures. However, these failures are mainly due to the sensor limitations of this fairly simple robot. In contrast to our work, the high-level planner is not embedded in a complex modular architecture with clearly defined low-level behaviours such as obstacle avoidance and sophisticated localisation techniques.

The control architecture used in our work is more comparable to the ones proposed by Haigh and Veloso [9] and Hähnel *et al.* [8]. These, in turn, provide only hand-coded recovery procedures for failures of the current action. Undetected failures of earlier actions are not considered. Furthermore, the use of the fluent calculus allows us to model side effects of actions.

Preference lists for action failures involve a great deal of speculation and need to be specified in advance. An alternative method would be a form of hypothesis testing: being left with a set of possible explanations for some action failure, each of them could be double checked by additional sensing or state verification. This topic is closely related to central questions in the field of active perception, for example, active vision. The application of a cognitive planner for sensing actions in the fashion outlined above could help in minimising action effort and maximising knowledge gain. Thus, it seems promising to formalise active perception domains within the fluent calculus. Furthermore, the static nature of the preference list could be overcome by learning from experiences.

## References

[1] Brewka, G.: Adding priorities and specificity to default logic, *Proceedings of the European Workshop on Logics in AI (JELIA-94)*, 838, Springer, 1994.

[2] De Giacomo, G., Levesque, H.: ConGolog, a concurrent programming language based on the situation calculus, *Artificial Intelligence*, **121**(1–2), 2000, 109–169.

[3] De Giacomo, G., Reiter, R., Soutchanski, M.: Execution monitoring of high-level robot programs, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, 1998.

[4] Fernández, J. L., Simmons, R. G.: Robust execution monitoring for navigation plans, *Proceedings of the Conference on Intelligent Robots and Systems (IROS-98)*, 1998.

[5] Fox, D., Burgard, W., Thrun, S.: Markov localization for mobile robots in dynamic environments, *Artificial Intelligence Research*, **11**, 1999, 391–427.

[6] Gat, E.: Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots, *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, MIT Press, 1992.

[7] Ginsberg, M. L., Smith, D. E.: Reasoning about action I: A possible worlds approach, *Artificial Intelligence*, **35**, 1988, 165–195.

[8] Hähnel, D., Burgard, W., Lakemeyer, G.: GOLEX - Bridging the gap between logic (GOLOG) and a real robot, *Proceedings of the 22nd German Conference on Artificial Intelligence (KI-98)*, 1998.

[9] Haigh, K. Z., Veloso, M. M.: High-level planning and low-level execution: Towards a complete robotic agent, *Proceedings of the First International Conference on Autonomous Agents*, ACM Press, 1997.

[10] Martin, Y.: The concurrent, continuous FLUX, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-2003)*, Morgan Kaufmann, 2003.

[11] Martin, Y., Thielscher, M.: Addressing the qualification problem in FLUX, *Proceedings of the German Annual Conference on Artificial Intelligence (KI-2001)*, 2174, Springer, 2001.

[12] McCarthy, J.: Epistemological problems of artificial intelligence, MIT Press, Cambridge, MA, 1977.

[13] McCarthy, J., Hayes, P. J.: Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence*, **4**, 1969, 463–502.

[14] Reiter, R.: A logic for default reasoning, *Artificial Intelligence*, **13**(1–2), 1980, 81–132.

[15] Reiter, R.: Natural actions, concurrency and continuous time in the situation calculus, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, Morgan Kaufmann, 1996.

[16] Reiter, R.: *Logic in Action*, MIT Press, 2001.

[17] Saffiotti, A.: Some notes on the integration of planning and reactivity in autonomous mobile robots, *Proceedings of the AAAI Spring Symposium on Foundations of Planning*, 1993.

[18] Sandewall, E.: Combining logic and differential equations for describing real-world systems, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, Morgan Kaufmann, 1989.

[19] Scherl, R., Levesque, H.: The frame problem and knowledge-producing actions, *Proceedings of the AAAI National Conference on Artificial Intelligence*, Washington, DC, 1993.

[20] Scherl, R., Levesque, H.: Knowledge, action, and the frame problem, *Artificial Intelligence*, **144**(1), 2003, 1–39.

[21] Shanahan, M. P.: Representing continuous change in the event calculus, *Proceedings of the European Conference on Artificial Intelligence (ECAI-90)*, 1990.

[22] Shanahan, M. P.: Robotics and the common sense informatics situation, *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, AAAI Press, 1996.

[23] Stentz, A.: The focussed D* algorithm for real-time replanning, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Morgan Kaufmann, 1995.

[24] Thielscher, M.: From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem, *Artificial Intelligence*, **111**(1–2), 1999, 277–299.

[25] Thielscher, M.: Representing the knowledge of a robot, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)* (A. Cohn, F. Giunchiglia, B. Selman, Eds.), Morgan Kaufmann, Breckenridge, CO, 2000.

[26] Thielscher, M.: The concurrent, continuous fluent calculus, *Studia Logica*, **67**(3), 2001, 315–331.

[27] Thielscher, M.: The qualification problem: A solution to the problem of anomalous models, *Artificial Intelligence*, **131**(1–2), 2001, 1–37.

[28] Thielscher, M.: FLUX: A logic programming method for reasoning agents, *Theory And Practice of Logic Programming*, 2004, To be published.

[29] Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust Monte Carlo localization for mobile robots, *Artificial Intelligence*, **128**(1–2), 2000, 99–141.

[30] Yang, Q.: *Intelligent Planning: A Decomposition and Abstraction Based Approach*, Springer, Berlin, 1997.