Planning with Noisy Actions (Preliminary Report)

Michael Thielscher

Dresden University of Technology
mit@inf.tu-dresden.de

Abstract. Ignoring the noise of physical sensors and effectors has always been a crucial barrier towards the application of high-level, cognitive robotics to real robots. We present a method of solving planning problems with noisy actions. The approach builds on the Fluent Calculus as a standard first-order solution to the Frame Problem. To model noise, a formal notion of uncertainty is incorporated into the axiomatization of state update and knowledge update. The formalism provides the theoretical underpinnings of an extension of the action programming language FLUX. Using constraints on real-valued intervals to encode noise, our system allows to solve planning problems for noisy sensors and effectors.

1 Introduction

Research into Cognitive Robotics aims at explaining and modeling intelligent acting in a dynamic world. Whenever intelligent behavior is understood as resulting from correct reasoning on correct representations, the classical Frame Problem [12] is a fundamental theoretical challenge: Given a representation of the effects of the available actions, how can one formally capture a crucial regularity of the real world, namely, that an action usually does not have arbitrary other effects? Explicitly specifying for each single potential effect that it is actually not an effect of a particular action, is obviously unsatisfactory both as a representation technique and as regards efficient inferencing [3]. The predicate calculus formalism of the Fluent Calculus [15], which roots in the logic programming approach of [7], provides a basic solution to both the representational and the inferential aspect of the Frame Problem. This solution also forms the theoretical underpinnings of the action programming language FLUX (the *Fluent Calculus Executor*) [16], which is based on constraint logic programming and allows to specify and reason about actions with incomplete states, and thus to solve planning problems under incomplete information.

In order to make it possible for a robot to reason about the correct use of its sensors, the Fluent Calculus has been extended by an axiomatization of how sensing affects the robot's knowledge of the environment [17]. A corresponding extension of FLUX has been developed in [18], which allows to solve planning problems with sensing actions. However, the method shares a common assumption of high-level approaches to sensing, namely, that sensors are ideal. This ignoring the noise of physical sensors and effectors has always been a crucial

barrier towards the application of cognitive robotics to real robots, because noisy sensors and effectors may take influence on the correctness of plans.

In this paper, we extend the existing model for acting and sensing in the Fluent Calculus to the representation of noise in both sensors and effectors. Actions with noise result only in limited certainty about the values of the affected state variables. As part of the approach we define a notion of executable plans in which the robot may condition its further actions on previously obtained sensor readings. In the second part of the paper, we present an extension of the action programming language and system FLUX which allows to solve planning problems with noisy actions, using constraints on real-valued intervals to encode noise. Prior to presenting these results, we give a brief introduction to the basic Fluent Calculus and FLUX.

2 FLUX

The action programming language FLUX [16] is a recent implementation of the Fluent Calculus using constraint logic programming. The Fluent Calculus combines, in classical logic, elements of the Situation Calculus [9] with a STRIPS-like solution to the Frame Problem [15]. The standard sorts ACTION and SIT (i.e., situations) are inherited from the Situation Calculus along with the standard functions S_0 : SIT and Do : ACTION $\times$ SIT $\mapsto$ SIT denoting, resp., the initial situation and the successor situation after performing an action; furthermore, the standard predicate $Poss$: ACTION $\times$ SIT denotes whether an action is possible in a situation. To this the Fluent Calculus adds the sort STATE with sub-sort FLUENT. The Fluent Calculus also uses the pre-defined functions $\emptyset$: STATE; $\circ$: STATE $\times$ STATE $\mapsto$ STATE; and $State$: SIT $\mapsto$ STATE; denoting, resp., the empty state, the union of two states, and the state of the world in a situation. As an example, let the function $Dist$: $\mathbb{R} \mapsto$ FLUENT denote the current distance of a robot to a wall. If z is a variable of sort STATE, then the following incomplete state specification says that initially the robot is somewhere between `4.8m` and `5.1m` away from the wall:[1]

$$(\exists x, z)\,(\,State(S_0) = Dist(x) \circ z \wedge \texttt{4.8m} \leq x \leq \texttt{5.1m}\,) \tag{1}$$

That is, the state in the initial situation is composed of the fluent $Dist(x)$ and sub-state z representing arbitrary other fluents that may also hold.

Based on the general signature, the Fluent Calculus provides a rigorously logical account of the concept of a state being characterized by the set of fluents that are true in the state. This is achieved by a suitable subset of the Zermelo-Fraenkel axioms, stipulating that function $\circ$ behaves like set union with $\emptyset$ as the empty set (for details see, e.g., [18]). Furthermore, the macro $Holds$ is used to specify that a fluent is contained in a state:

$$Holds(f, z) \stackrel{\text{def}}{=} (\exists z')\, z = f \circ z' \tag{2}$$

[1] Free variables in formulas are assumed universally quantified. Variables of sorts ACTION, SIT, FLUENT, and STATE shall be denoted by the letters a, s, f, and z, resp. The function $\circ$ is written in infix notation.

A second macro, which reduces to (2), is used for fluents holding in situations:

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

As an example, consider the following so-called state constraint, which stipulates that the distance to the wall be unique in every situation:

$$(\forall s)\,(\exists!x)\,Holds(Dist(x), s)$$

The Frame Problem is solved in the Fluent Calculus using so-called state update axioms, which specify the difference between the states before and after an action. The axiomatic characterization of negative effects, i.e., facts that become false, is given by an inductive abbreviation which generalizes STRIPS-style update to incomplete states:

$$z' = z - f \stackrel{\text{def}}{=} [z' \circ f = z \vee z' = z] \wedge \neg Holds(f, z')$$

$$z' = z - (f_1 \circ \ldots \circ f_n \circ f_{n+1}) \stackrel{\text{def}}{=} (\exists z'')\,(z'' = z - (f_1 \circ \ldots \circ f_n) \wedge z' = z'' - f_{n+1})$$

This is the general form of a state update axiom for a (possibly nondeterministic) action $A(\vec{x})$ with a bounded number of (possibly conditional) effects:

$$\begin{aligned} &Poss(A(\vec{x}), s) \supset \\ &\quad (\exists \vec{y}_1)\,(\Delta_1(\vec{x}, \vec{y}_i, State(s)) \wedge State(Do(A(\vec{x}), s)) = (State(s) - \vartheta_1^-) \circ \vartheta_1^+) \\ &\quad \vee \ldots \vee \\ &\quad (\exists \vec{y}_n)\,(\Delta_n(\vec{x}, \vec{y}_n, State(s)) \wedge State(Do(A(\vec{x}), s)) = (State(s) - \vartheta_n^-) \circ \vartheta_n^+) \end{aligned}$$

where the sub-formulas $\Delta_i(\vec{x}, \vec{y}_i, State(s))$ specify the conditions on $State(s)$ under which $A(\vec{x})$ has the positive and negative effects ϑ_i^+ and ϑ_i^-, resp. Both ϑ_i^+ and ϑ_i^- are STATE terms composed of fluents with variables among $\vec{x}, \vec{y}_i$. If $n = 1$ and $\Delta_1 \equiv True$, then action $A(\vec{x})$ does not have conditional effects. If $n > 1$ and the conditions Δ_i are not mutually exclusive, then the action is nondeterministic.

Consider, as an example, the function $MoveFwd : \mathbb{R}^+ \mapsto$ ACTION denoting the action of the robot moving a certain (positive) distance towards the wall. Under the assumption that the effectors are ideal, the effect of this axiom can be axiomatized by the following state update axiom:

$$\begin{aligned} &Poss(MoveFwd(d), s) \supset \\ &\quad (\exists x, y)\,(Holds(Dist(x), s) \wedge y = x - d \wedge \\ &\qquad State(Do(MoveFwd(d), s)) = (State(s) - Dist(x)) \circ Dist(y)) \end{aligned} \tag{3}$$

Put in words, moving the distance d towards the wall has the effect that the robot is no longer x units away from the wall and will end up at $x - d$. Recall, for example, formula (1) and suppose for the sake of argument that $Poss(MoveFwd(\mathtt{2m}), S_0)$. After combining the inequations, our state update axiom and the foundational axioms imply

$$\begin{aligned} (\exists y, z)\,(&State(Do(MoveFwd(\mathtt{2m}), S_0)) = z \circ Dist(y) \\ &\wedge\ \mathtt{2.8m} \leq y \leq \mathtt{3.1m}) \end{aligned} \tag{4}$$

A crucial property of this new state equation is that sub-state z has been carried over from (1). Thus any additional constraint on z in (1) equally applies to the successor state $State(Do(MoveFwd(2\mathtt{m}), S_0))$. This is how the Frame Problem is solved in the Fluent Calculus.

Based on the theory of the Fluent Calculus, the distinguishing feature of the action programming language FLUX is to support incomplete states, which are modeled by *open* lists of the form

```
Z0 = [F1,...,Fm | Z]
```

(encoding the state description $\mathtt{Z0} = \mathtt{F1} \circ \ldots \circ \mathtt{Fm} \circ \mathtt{Z}$), along with constraints

```
not_holds(F, Z)
not_holds_all([X1,...,Xk], F, Z)
```

encoding, resp., the negative statements $(\exists \vec{y})\, \neg Holds(\mathtt{F}, \mathtt{Z})$ (where $\vec{y}$ are the variables occurring in F) and $(\exists \vec{y})(\forall \mathtt{X1}, \ldots, \mathtt{Xk})\, \neg Holds(\mathtt{F}, \mathtt{Z})$ (where $\vec{y}$ are the variables occurring in F except $\mathtt{X1}, \ldots, \mathtt{Xk}$). These two constraints are used to bypass the problem of 'negation-as-failure' for incomplete states. In order to process these constraints, so-called declarative Constraint Handling Rules [4] have been defined and proved correct under the foundational axioms of the Fluent Calculus. In addition, the core of FLUX contains definitions for `holds(F,Z)`, by which is encoded macro (2), and `update(Z1,ThetaP,ThetaN,Z2)`, which encodes the state equation $\mathtt{Z2} = (\mathtt{Z1} - \mathtt{ThetaN}) \circ \mathtt{ThetaP}$.

As an example, the following is the FLUX encoding of our state update axioms (3) (ignoring preconditions) and the initial specification (1):[2]

```
state_update(Z1, move_forward(D), Z2) :-
  holds(dist(X), Z1), Y = X - D,
  update(Z1, [dist(Y)], [dist(X)], Z2).

init(Z0) :- X :: 4.8..5.1, holds(dist(X), Z0),
            duplicate_free(Z0).
```

where the constraint `duplicate_free(Z)` means that list `Z` does not contain multiple occurrences. The following sample query computes the conclusion made in (4):

```
[eclipse 1]: init(Z0), state_update(Z0, move_forward(2), Z1).

Z0 = [dist(X{4.8..5.1}) | _Z]
Z1 = [dist(Y{2.8..3.1}) | _Z]
```

[2] Throughout the paper we use ECLIPSE-Prolog notation. The interval expression `X::L..R` is taken from the library RIA, the constraint solver for interval arithmetic (see Section 4 below).

3 Update for Noisy Actions

The Fluent Calculus provides a simple and elegant means to axiomatize noisy effectors. Uncertainty regarding the values of affected fluents can be represented in a state update axiom as existentially quantified and constrained variables. For example, suppose that the effectors of our robot are noisy in that the actual position after moving towards the wall may differ from the ideal one by the factor 0.05. The following is a suitable state update axiom for this noisy action:

$$\begin{aligned} &Poss(MoveFwd(d), s) \supset \\ &\quad (\exists x, y)\,(\,Holds(Dist(x), s) \wedge \\ &\qquad State(Do(MoveFwd(d), s)) = (State(s) - Dist(x)) \circ Dist(y) \wedge \\ &\qquad |y - (x - d)| \leq d \cdot 0.05\,) \end{aligned} \tag{5}$$

Moving a distance d thus has the effect that the robot will end up at some distance y which is at most $d \cdot 0.05$ units away from the goal position $x - d$.

To represent knowledge in the Fluent Calculus and to reason about sensing actions, the predicate $KState : \text{SIT} \times \text{STATE}$ has been introduced in [17]. An instance $KState(s, z)$ means that according to the knowledge of the planning robot, z is a possible state in situation s. On this basis, the fact that some property of a situation is known to the robot is specified using the macro $Knows$, which is defined as follows:

$$Knows(\varphi, s) \stackrel{\text{def}}{=} (\forall z)\,(KState(s, z) \supset HOLDS(\varphi, z)) \tag{6}$$

where

$$\begin{aligned} HOLDS(\alpha, z) &\stackrel{\text{def}}{=} \alpha \qquad (\alpha \text{ arithmetic constraint}) \\ HOLDS(f, z) &\stackrel{\text{def}}{=} Holds(f, z) \\ HOLDS(\neg\varphi, z) &\stackrel{\text{def}}{=} \neg HOLDS(\varphi, z) \\ HOLDS(\varphi \wedge \psi, z) &\stackrel{\text{def}}{=} HOLDS(\varphi, z) \wedge HOLDS(\psi, z) \\ HOLDS((\forall x)\,\varphi, z) &\stackrel{\text{def}}{=} (\forall x)\,HOLDS(\varphi, z) \end{aligned}$$

This model of knowledge uses pure first-order logic. As an example, the precondition for the action $MoveFwd$ can be specified in such a way that the robot always keeps a safety distance to the wall. This of course requires to take into account the uncertainty of the effectors:

$$Poss(MoveFwd(d), s) \equiv Knows(\,(\forall x)(Dist(x) \supset x - 1.05 \cdot d \geq 0.1\mathtt{m}), s)$$

Hence, moving forward is possible only if the robot *knows* that it will end up at least $0.1\mathtt{m}$ away from the wall. Suppose given that the robot knows that its initial position is somewhere between $4.8\mathtt{m}$ and $5.1\mathtt{m}$ away from the wall, that is,

$$KState(S_0, z) \supset (\forall x)\,(Holds(Dist(x), z) \supset 4.8\mathtt{m} \leq x \leq 5.1\mathtt{m}\,) \tag{7}$$

With the help of macro (6) it follows that $Poss(MoveFwd(2\mathtt{m}), S_0)$.

The Frame Problem for knowledge is solved by axioms that determine the relation between the possible states before and after an action. More formally,

the effect of an action $A(\vec{x})$, be it sensing or not, on the knowledge is specified by a *knowledge update axiom* of the form

$$\begin{array}{l} Poss(A(\vec{x}), s) \supset \\ \quad [\,(\forall z)\,(KState(Do(A(\vec{x}), s), z) \equiv (\exists z')(KState(s, z') \wedge \Psi(z, z', s))\,)\,] \end{array} \tag{8}$$

In case of non-sensing actions, formula Ψ defines what the robot knows of the effects of the action. E.g., state update axiom (5) for moving with unreliable effectors corresponds to the following knowledge update axioms:

$$\begin{array}{l} Poss(MoveFwd(d), s) \supset \\ \quad [\,(\forall z)\,(KState(Do(MoveFwd(d), s), z) \equiv \\ \qquad (\exists x, y, z')\,(\,KState(s, z') \wedge Holds(Dist(x), z') \wedge \\ \qquad\qquad z = (z' - Dist(x)) \circ Dist(y) \wedge \\ \qquad\qquad |y - (x - d)| \leq d \cdot 0.05\,)\,)\,] \end{array} \tag{9}$$

The generic ACTION term $Sense(f)$ has been introduced in [17] to denote the action of sensing whether a fluent f holds. The corresponding knowledge update axiom,

$$\begin{array}{l} Poss(Sense(f), s) \supset \\ \quad [\,KState(Do(Sense(f), s), z) \equiv \\ \qquad KState(s, z) \wedge [Holds(f, z) \equiv Holds(f, s)]\,] \end{array} \tag{10}$$

says that among the states possible in s only those are still possible after sensing which agree with the actual state of the world as far as the sensed fluent is concerned. An important implication is that after sensing f either the fluent or its negation is known to hold [17]. Thus axiom (10) can be viewed as modeling an ideal sensor.

In order to model sensing with noise, axiom schema (8) needs to be used in a different manner, where formula Ψ restricts the possible states to those where the value of the sensed property may deviate from the actual value within a certain range. To this end, we introduce the generic ACTION function $Sense_F$ where F can be any domain function of type $\mathbb{R} \mapsto$ FLUENT.[3] For later purpose, we assume that for any fluent which can thus be sensed there is an additional fluent $SensorReading_F(x)$ denoting the last sensor reading. Let ϱ_F denote the maximal deviation of the noisy sensor reading from the actual value. The effect of noisy $Sense_F$ is then specified by this knowledge update axiom:

$$\begin{array}{l} Poss(Sense_F, s) \supset \\ \quad [\,(\exists r)\,(\forall z)\,(KState(Do(Sense_F, s), z) \equiv \\ \qquad (\exists r', x, y, z')\,(\,KState(s, z') \wedge \\ \qquad\qquad Holds(F(x), s) \wedge \\ \qquad\qquad Holds(F(y), z') \wedge Holds(SensorReading_F(r'), z') \wedge \\ \qquad\qquad z = (z' - SensorReading_F(r')) \circ SensorReading_F(r) \wedge \\ \qquad\qquad |x - r| \leq \varrho_F \wedge |y - r| \leq \varrho_F\,)\,] \end{array} \tag{11}$$

[3] For the sake of simplicity, we assume that each sensor delivers just a unary value. The generalization to perceivable fluents with multiple arguments, such as the position in a two-dimensional space, is straightforward.

Put in words, after sensing, a sensor reading r is obtained which differs from the actual value x by at most ϱ_F, and only those states are still considered possible where the value y of the sensed fluent deviates from r by at most ϱ_F and where the old sensor reading r' has been updated to r. The accompanying state update axiom says that a new sensor reading is obtained within the allowed range:

$$\begin{aligned} &Poss(Sense_F, s) \supset \\ &\quad (\exists r, r', x)\,(Holds(F(x), s) \wedge Holds(SensorReading_F(r'), s) \wedge \\ &\qquad State(Do(Sense_F, s)) = (State(s) - SensorReading_F(r')) \\ &\qquad\qquad \circ SensorReading_F(r) \\ &\qquad \wedge\ |x - r| \leq \varrho_F\,) \end{aligned} \tag{12}$$

While precise knowledge of the sensed property is thus no longer guaranteed, an important consequence of the generic knowledge update axiom (11) is that sensing never cause loss of possibly more precise initial knowledge wrt. the sensed fluent.

Proposition 1. *Let α be an arithmetic constraint with free variable x, then (11) and the foundational axioms entail*

$$Knows((\exists x)\,(F(x) \wedge \alpha), s) \supset Knows((\exists x)\,(F(x) \wedge \alpha), Do(Sense_F, s))$$

A second crucial consequence is that the sensor reading itself will be known by the robot.

Proposition 2. *(11) and the foundational axioms entail*

$$(\exists x)\, Knows(SensorReading_F(x), Do(Sense_F, s))$$

Suppose, for example, that the noise of the robot's sensor for measuring the distance to the wall is given by $\varrho_{Dist} = \mathtt{0.1m}$, and recall the specification of initial knowledge given in (7). Then after moving $\mathtt{2m}$ towards the wall and sensing the distance, the robot will know the distance to the wall within a range of $\mathtt{0.1m}$. Moreover, this distance must be between $\mathtt{2.7m}$ and $\mathtt{3.2m}$ (as above) while the possible sensor readings are between $\mathtt{2.6m}$ and $\mathtt{3.3m}$ due to the noise of the sensor. Formally, let $S_2 = Do(Sense_{Dist}, Do(MoveFwd(\mathtt{2m}), S_0))$, then the state and knowledge update axioms for $MoveFwd$ and $Sense_{Dist}$ entail

$$\begin{aligned} &(\exists x)\,(\mathtt{2.7m} \leq x \leq \mathtt{3.2m} \wedge Knows((\exists y)\,(Dist(y) \wedge |y - x| \leq \mathtt{0.1m}), S_2)\,) \\ &\wedge\ (\exists r)\,(\mathtt{2.6m} \leq r \leq \mathtt{3.3m} \wedge Knows(SensorReading_{Dist}(r), S_2)\,) \end{aligned}$$

Solutions to planning problems with noisy sensing actions may require to condition an action on the outcome of sensing. Suppose, for example, the goal of our robot is to be at a point where it is between $\mathtt{2.8m}$ and $\mathtt{3.2m}$ away from the wall. Given the initial knowledge state (7), this problem is solvable only by a plan which includes reference to previous sensing readings. A possible solution along this line would be for the robot to advance $\mathtt{2m}$, measure the distance, and if necessary adjust its position according to the obtained reading r. A suitable

choice of the argument for the second *MoveFwd* action is $r - \texttt{3m}$: As we have seen above, the robot knows that it will be between `2.7m` and `3.2m` away from the wall after the initial move. Moreover, the sensor reading r will measure the actual position x within the range $x \pm \texttt{0.1m}$. Thus, even with the added uncertainty caused by the new movement, the robot can be sure without further sensing that it will end up at a distance which is between `2.885m` and `3.12m`.

In order to allow for the use of sensor readings as parameters for actions, we need to make precise when formal actions such as $MoveFwd(r - \texttt{3m})$ can be considered executable by the robot. To this end, we introduce the macro $Kref(\tau, s)$ (inspired by [14]) with the intended meaning that the arithmetic expression τ can be evaluated by the robot on the basis of its knowledge in situation s. The macro is inductively defined as follows:

$$\begin{array}{rll} Kref(c,s) & \stackrel{\text{def}}{=} & True \qquad (c \text{ constant}) \\ Kref(F,s) & \stackrel{\text{def}}{=} & (\exists x)\, Knows(F(x),s) \qquad (F \text{ value fluent}) \\ Kref(op(\tau_1,\tau_2),s) & \stackrel{\text{def}}{=} & Kref(\tau_1,s) \wedge Kref(\tau_2,s) \qquad (op \in \{+,-,\cdot,\ldots\}) \end{array} \tag{13}$$

The executability of a plan $Do(a_n, \ldots, Do(a_1, S_0)\ldots)$ is then defined as the macro *EXEC* as follows:

$$\begin{array}{rcl} EXEC(S_0) & \stackrel{\text{def}}{=} & True \\ EXEC(Do(A(\tau_1,\ldots,\tau_k),s) & \stackrel{\text{def}}{=} & EXEC(s) \wedge Poss(A(\tau_1,\ldots,\tau_k),s) \\ & & \wedge\, Kref(\tau_1,s) \wedge \ldots \wedge Kref(\tau_k,s) \end{array}$$

4 Planning with Noise in Flux

Encoding state update axioms for noisy actions requires to state arithmetic constraints. A constraint solver is then needed to deal with these constraints. A suitable choice for FLUX is the standard ECLIPSE constraint system RIA (for real number interval arithmetic). Incorporating this constraint module, the following is a suitable encoding of state update axiom (5), specifying an action with unreliable effectors:

```
:- lib(ria).

state_update(Z1, move_forward(D), Z2) :-
  holds(dist(X), Z1),
  abs(Y-(X-D)) *=< 0.05*D,
  update(Z1, [dist(Y)], [dist(X)], Z2).
```

In comparison with the computed answer shown at the end of Section 2, the new state update axiom causes a higher degree of uncertainty wrt. the resulting position:

```
init(Z0) :- X :: 4.8..5.1, holds(dist(X), Z0),
            duplicate_free(Z0).
```

```
[eclipse 1]: init(Z0), state_update(Z0, move_forward(2), Z1).

Z0 = [dist(X{4.8..5.1}) | _Z]
Z1 = [dist(Y{2.7..3.2}) | _Z]
```

While the explicit notion of possible states leads to an extensive framework for reasoning about knowledge and noisy sensing, automated deduction becomes considerably more intricate by the introduction of the modality-like *KState* predicate. As a consequence, in [18] we have developed an inference method which avoids separate update of knowledge and states. In what follows, we extend this result to noisy sensors and effectors and show how knowledge updates are implicitly obtained by progressing an incomplete state through state update axioms.

Our approach rests on two assumptions. First, the planning robot needs to know the given initial specification $\Phi(State(S_0))$, and this is all it knows of S_0, that is, $KState(S_0, z) \equiv \Phi(z)$. Second, the robot must have accurate knowledge of its own actions. That is, formally, the possible states after a non-sensing action are those which would be the result of actually performing the action in one of the previously possible states:

Definition 3. [**17**] A set of axioms Σ *represents accurate effect knowledge* if for each non-sensing ACTION function A, Σ contains a unique state update axiom

$$Poss(A(\vec{x}), s) \supset \Gamma_A\{z/State(Do(A(\vec{x}), s)), z'/State(s))\} \tag{14}$$

(where $\Gamma_A(\vec{x}, z, z')$ is a first-order formula with free variables among $\vec{x}, z, z'$ and without a sub-term of sort SIT) and a unique knowledge update axiom which is equivalent to

$$\begin{aligned} Poss(A(\vec{x}), s) \supset\ [\, &(\forall z)\,(KState(Do(A(\vec{x}), s), z) \equiv \\ &(\exists z')(KState(s, z') \wedge \Gamma_A(\vec{x}, z, z'))\,)\,] \end{aligned} \tag{15}$$

Accurate knowledge of effects suffices to ensure that the possible states after a non-sensing action can be obtained by progressing a given state specification through the state update axiom for that action. The effect of sensing, on the other hand, cannot be obtained in the same fashion. To see why, let S be a situation and consider the knowledge specification

$$\begin{aligned} &KState(S, z) \equiv \\ &\quad (\exists x, y)\,(z = Dist(x) \circ SensorReading_{Dist}(y) \wedge \texttt{4.8m} \leq x \leq \texttt{5.1m}) \end{aligned} \tag{16}$$

Suppose that $Poss(Sense_{Dist}, S)$, then knowledge update axiom (11) yields different models reflecting the possible sensing result r, which run from `4.7m` to `5.2m`. However, in each model the distance is known up to an error of just $\varrho_{Dist} =$ `0.1m`! Hence, while we cannot predict the sensing outcome, it is clear that the sensed value will be known within the precision of the sensor. This knowledge is not expressible by a specification of the form $KState(Do(Sense_{Dist}, S), z) \equiv \Phi(z)$ entailed by (16) and (11). Hence, the effect of a sensing action cannot be obtained by straightforward progression.

In order to account for different models for *KState* caused by sensing, we introduce the notion of a *sensing history* ς as a finite, possibly empty list of real numbers. A history is meant to describe the outcome of each sensing action in a sequence of actions. For the sake of simplicity, we assume that the only sensing action is the generic $Sense_F$ with knowledge update axiom (11) and state update axiom (12).

For the formal definition of progression we also need the notion of an *action sequence* σ as a finite, possibly empty list of ground ACTION terms. An action sequence corresponds naturally to a situation, which we denote by S_σ:

$$S_{[\,]} \stackrel{\text{def}}{=} S_0 \quad \text{and} \quad S_{[A(\vec{t})\,|\,\sigma]} \stackrel{\text{def}}{=} Do(A(\vec{t}), S_\sigma)$$

We are now in a position to define, inductively, a *progression operator* $\mathcal{P}(\sigma, \varsigma, z)$ along the line of [18], by which an initial state specification $\Phi(State(S_0))$ is progressed through an action sequence σ wrt. a sensing history ς, resulting in a formula specifying z:

$$\mathcal{P}([\,], \varsigma, z) \stackrel{\text{def}}{=} \Phi(z) \quad \text{if } \varsigma = [\,] \tag{17}$$

$$\mathcal{P}([A(\vec{t})\,|\,\sigma], \varsigma, z) \stackrel{\text{def}}{=} (\exists z')\,(\mathcal{P}(\sigma, \varsigma, z') \wedge \Gamma_A(\vec{t}, z, z')) \quad \text{if } A \text{ non-sensing with state update (14)} \tag{18}$$

$$\begin{array}{l} \mathcal{P}([Sense_F\,|\,\sigma], \varsigma, z) \stackrel{\text{def}}{=} \\ \quad (\exists r', x, y, z')\,(\mathcal{P}(\sigma, \varsigma', z') \wedge \\ \qquad\qquad Holds(F(x), s) \wedge \\ \qquad\qquad Holds(F(y), z') \wedge Holds(SensorReading_F(r'), z') \wedge \\ \qquad\qquad z = (z' - SensorReading_F(r')) \circ SensorReading_F(r) \wedge \\ \qquad\qquad |x - r| \leq \varrho_F \wedge |y - r| \leq \varrho_F\,) \\ \qquad\qquad\qquad\qquad \text{where } \varsigma = [r\,|\,\varsigma'] \end{array} \tag{19}$$

In case the length of the history ς does not equal the number of sensing actions in σ, we define $\mathcal{P}(\sigma, \varsigma, z)$ as *False*. Progression provides a provably correct inference method for knowledge update.

Theorem 4. *Consider the initial state and knowledge* $\Sigma_0 = \{\Phi(State(S_0)), KState(S_0, z) \equiv \Phi(z)\}$ *and let* Σ *be the foundational axioms plus a set of domain axioms representing accurate effect knowledge. Let* σ *be an action sequence such that* $\Sigma \cup \Sigma_0 \models EXEC(S_\sigma)$. *Then for any model* $\mathcal{M}$ *of* $\Sigma_0 \cup \Sigma$ *and any valuation* ν,

$$\mathcal{M}, \nu \models KState(S_\sigma, z) \quad \textit{iff} \quad \mathcal{M}, \nu \models \mathcal{P}(\sigma, \varsigma, z) \ \textit{for some } \varsigma$$

The proof is by simple induction on σ.

This theorem serves as the formal justification for the FLUX encoding of knowledge and sensing. The sensing action $Sense_{Dist}$, for example, is encoded by a state update axiom which carries as additional argument the result of sensing, that is, the sensor reading:

```
state_update(Z1, sense_dist, Z2, SV) :-
   holds(dist(X), Z1), holds(reading(Y), Z1),
   abs(X-SV) *=< 0.1,
   update(Z1, [reading(SV)], [reading(Y)], Z2).
```

The definition of progression is a direct encoding of (17)–(19):

```
p([], [], Z) :- init(Z).
p([A|S], H2, Z2) :- p(S, H1, Z1),
                    ( state_update(Z1, A, Z2), H2=H1 ;
                      state_update(Z1, A, Z2, SV), H2=[SV|H1] ).
```

In principle, the FLUX clauses we arrived at can readily be used by a simple forward-chaining search algorithm. Enumerating the set of plans, including all possible sensing actions, a solution will eventually be found if only the problem is solvable. However, planning with incomplete states usually involves a considerable search space, and the possibility to generate conditional plans only enlarges it. The concept of nondeterministic robot programs has been introduced in GOLOG as a powerful heuristics for planning, where only those plans are searched which match a given skeleton [10]. This avoids considering obviously useless actions such as ineffectual sensing. In [18] we have shown how this concept can be adopted in FLUX on the basis of a progression operator, in order to make planning with sensing more efficient. These heuristics can be directly applied to planning with noisy actions in FLUX.

5 Summary and Discussion

We have presented an approach to planning with noisy actions by appealing to the Fluent Calculus as a basic solution to the Frame Problem. The axiomatization has be shown to exhibit reasonable properties. Moreover, we have extended the action programming language FLUX to obtain a system for solving planning problems that involve noisy actions.

Both the axiomatic approach as well as the realization in FLUX are an extension of the solution to the Frame Problem for knowledge [17, 18]. A distinguishing feature of this approach is its expressiveness in comparison to most existing approaches to planning with knowledge and sensing. Unlike other systems, FLUX is not tailored to restricted classes of planning problems (as opposed to, e.g., [6, 5, 2, 11, 8]) and allows to search for suitable sensing actions during planning (as opposed to [13]).

Closest to our work is [1], where an extension of the Situation Calculus is presented that allows to axiomatize noisy actions. The crucial difference to our approach is the indirect way of modeling a noisy action as a non-deterministic selection among actions with determined effects. To this end, the approach uses the non-deterministic programming constructs of GOLOG for modeling noise. Consequently, these programs can no longer be used as planning heuristics, and therefore the theory cannot be straightforwardly integrated into GOLOG to provide a planning system that deals with noise. On the other hand, the approach

of [1] includes a notion of probability distribution for noisy effects. The extension of our approach along this line is an important goal for future work.

References

1. F. Bacchus, J. Halpern, and H. Levesque. Reasoning about noisy sensors and effectors in the situation calculus. *Artif. Intell.*, 111(1–2):171–208, 1999.
2. C. Baral and T. C. Son. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. of ILPS*, pp. 387–401, 1997. MIT Press.
3. W. Bibel. Let's plan it deductively! *Artif. Intell.*, 103(1–2):183–208, 1998.
4. T. Frühwirth. Theory and practice of constraint handling rules. *J. of Logic Programming*, 37(1–3):95–138, 1998.
5. G. De Giacomo, L. Iocchi, D. Nardi, and R. Rosati. Planning with sensing for a mobile robot. In *Proc. of ECP*, vol. 1348 of *LNAI*, pages 158–170. Springer, 1997.
6. K. Golden and D. Weld. Representing sensing actions: The middle ground revisited. In *Proc. of KR*, pp. 174–185, 1996. Morgan Kaufmann.
7. S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.
8. G. Lakemeyer. On sensing and off-line interpreting GOLOG. In *Logical Foundations for Cognitive Agents*, pages 173–189. Springer, 1999.
9. H. Levesque, F. Pirri, and R. Reiter. Foundations for a calculus of situations. *Electronic Transactions on Artif. Intell.*, 3(1–2):159–178, 1998. URL: `http://www.ep.liu.se/ea/cis/1998/018/`.
10. H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31(1–3):59–83, 1997.
11. J. Lobo. COPLAS: A conditional planner with sensing actions. In *Cognitive Robotics*, vol. FS–98–02 of *AAAI Fall Symposia*, pp. 109–116. AAAI Press 1998.
12. J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intell.*, 4:463–502, 1969.
13. R. Reiter. On knowledge-based programming with sensing in the situation calculus. In *Cognitive Robotics Workshop at ECAI*, pp. 55–61, Berlin, Germany, August 2000.
14. R. Scherl and H. Levesque. The frame problem and knowledge-producing actions. In *Proc. of AAAI*, pp. 689–695, 1993.
15. M. Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artif. Intell.*, 111(1–2):277–299, 1999.
16. M. Thielscher. The Fluent Calculus: A Specification Language for Robots with Sensors in Nondeterministic, Concurrent, and Ramifying Environments. Technical Report CL-2000-01, Dept. of Computer Science, Dresden Univ. of Tech., 2000. URL: `http://www.cl.inf.tu-dresden.de/~mit/publications/reports/CL-2000-01.pdf`.
17. M. Thielscher. Representing the knowledge of a robot. In *Proc. of KR*, pp. 109–120, 2000. Morgan Kaufmann.
18. M. Thielscher. Inferring implicit state knowledge and plans with sensing actions. In *Proc. of KI*, vol. 2174 of *LNAI*, pp. 366–380, 2001. Springer.