Automatic Design of Robot Behaviors through Constraint Networks Acquisition

Mathias Paulin, Christian Bessiere, Jean Sallantin

Automatic Design of Robot Behaviors Through Constraint Network Acquisition

Mathias Paulin Christian Bessiere* Jean Sallantin

LIRMM (CNRS/University of Montpellier)
161 rue Ada, 34392 Montpellier, France
{paulin,bessiere,sallantin}@lirmm.fr

Abstract

Control architectures, such as the LAAS *architecture [1],* CLARATY *[12] and* HARPIC *[9], have been developped to provide autonomy to robots. To achieve a robot's task, these control architectures plan sequences of* sensorimotor behaviors*. Currently carried out by roboticians, the design of sensorimotor behaviors is a truly complex task that can require many hours of hard work and intensive computations. In this paper, we propose a Constraint Programming-based framework to interact with roboticians during the sensorimotor behaviors design. A constraint network acquisition platform and a CSP-Based planner are used to* automatically *design sensorimotor behaviors. Moreover, our architecture exploits the propagation properties of the acquired CSPs to supervise the execution of a given sensorimotor behavior. Some experimental results are presented to validate our approach.*

1 Introduction

Robotics has massively invested the industrial world in the second half of the last century. The robots production have gradually replaced humans in repetitive, arduous or hazardous tasks (soldering, painting, assembling of large components, *etc.*). The robots production have been designed to automatically perform pre-defined and repetitive tasks. These robots are incredibly effective but are not very flexible nor adaptive. For several decades, research in robotics has expanded to the design of *autonomous* robots; *i.e.*, multipurpose machines capable of adapting to different environments in order to carry out complex tasks. On the other hand, the spectrum of current robotics covers a large number of areas and technologies, such as mechanical design, control theory, data acquisition, signal processing, human-machine interactions, artificial intelligence techniques (planning, machine learning), *etc*. The autonomy challenge is then no longer in the production of automatons, but rather in the design of *intelligent* machines from heterogenous hardware and software technologies, that are able to perform complex tasks.

Different control architectures, such as the LAAS architecture [1], CLARATY [12] or HARPIC [9], have been proposed in order to tackle the autonomy challenge. These control architectures are used successfully for the supervision of mobile robots (LAAS, HARPIC) and for space applications (CLARATY). These control architectures generally use several supervision levels. For instance, the LAAS architecture is composed of three levels: a functional level, an executive level and a decision level. The functional level (which is the lowest one) offers a panel of *sensorimotor behaviors*, processing functions or actuators control loops, that can be executed automatically. In order to achieve a given task, sensorimotor behaviors are activated by the executive level. The executive level controls and coordinates the execution of the sensorimotor behaviors according to the plans computed at the decision level. The decision level is the highest level of the LAAS architecture. It produces both sequences of sensorimotor behaviors and reacts to events by adjusting those sequences in order to achieve the general mission of the robot.

In these control architectures, sensorimotor behaviors are considered as *basic* robot action and perception capacities. They are executed as automaton modules. However, these sensorimotor behaviors are not really basic as they are the result of the combination of elementary actions. (An elementary action corresponds to a small set of commands to the actuators leading to an explicit modification of the state of robot.) Elementary actions and sensorimotor behaviors are designed by roboticians. Each time they have to supervise a new robot with a control architecture, roboticians have to design elementary actions for the robot and to combine them in order to design a panel of sensorimotor behaviors. However, this design is a hard task. Indeed, each elementary action of the robot is modelled by sets of

*Supported by the ANR project ANR-06-BLAN-0383-02.

mathematical equations which involve really complex physical laws, such as mechanical energy conservation law, kinetic momentum, partial derivative equations... Building and combining elementary actions in order to define a sensorimotor behavior is thus difficult and can require a lot of hard-work hours.

In this paper, we propose to interact with roboticians during the process of designing sensorimotor behaviors. Our approach automatically encodes each elementary action as a CSP by Machine Learning. To perform a given sensorimotor behavior, we sequence these multiple constraint networks using a CSP-based planner. The CSP modelling we use has the twofold advantage of abstracting a declaratory model from the elementary actions and of having strong propagation properties, which is a capital aspect for planning. With this approach, we wish to contribute to the automatic design of sensorimotor behaviors.

This paper is organized as follows. Section 2 describes the general architecture we propose. In Section 3, we discuss the motivation for our CSP-based approach. Section 4 presents our approach. Some experimental results are described in Section 5. They validate our approach for automatic acquisition of elementary actions and design of sensorimotor behaviors. Finally, we conclude and discuss some future improvements in Section 6.

2 General Architecture

The approach we propose in this paper consists in modelling the elementary actions of a robot with constraint networks, which inherit the properties of Constraint Propagation of CSPs, and then combining those CSPs in order to plan and supervise the execution of sensorimotor behaviors.

We use the constraint network acquisition platform CONACQ [5, 4] to *automatically* model by Machine Learning each elementary action of the robot. For each elementary action a_i, we have to supply CONACQ with a set of valid instances of a_i and with a set of instances which do not correspond to a_i. CONACQ then automatically models a_i with a constraint network which describes the conditions in which a_i can be executed (*i.e.* its preconditions), its effects, and how the different actuators of the robot must react in order to perform a_i. The acquired CSPs are then combined by planning to constitute sensorimotor behaviors.

Given a sensorimotor behavior, our architecture encodes this behavior as a task T. A CSP-based planner combines some of the acquired CSPs in order to determine a sequence $\mathcal{S}$ of elementary actions which must be executed to carry out T.

Once computed, the sequence $\mathcal{S}$ is transmitted to the control module, which then supervises its execution by the robot. The execution of $\mathcal{S}$ is performed step-by-step, one action after another. After the execution of each elementary action, the control module compares the real descriptors values with the predicted ones (*i.e.* given by the constraint modelling). If the current state of the robot corresponds perfectly to predictions, the control module continues the execution of the sequence as defined originally. If the current state is not equal to predictions,[1] the control module determines by propagation if it is possible to adjust next elementary actions in order to achieve T without redefining the sequence $\mathcal{S}$. If such kind of adjustements exist, the control module imposes thoses *minor* corrections to the robot in order to rectify the execution of the sequence. On the other hand, if it is impossible to adjust the current sequence whithout redefining it (because of a too big gap between the current state and predictions), the control module calls again the planner to compute a new sequence $\mathcal{S}'$ of elementary actions to achieve the task T from the current state. The control module supervises then the execution of $\mathcal{S}'$ step-by-step, by checking and rectifying possible differences between the execution reality and predictions.

Even if our approach deals with planning and uses a control and correction operating cycle, it could be noted that our approach is limited to the design and the supervision of sensorimotor behaviors. The previously cited control architectures, which notably deal with temporal planning and can manage multiple goals (according to the global mission), would be used in order to achieve high-level tasks and missions. Our architecture would be used then by those control architectures for performing sensorimotor behaviors, replacing the behaviors which are currently hand-designed by roboticians.

3 Why Constraint Programming?

Constraint Programming has been successfully used for solving robotics problems such as trajectory verification of parallel robots [10], control of reconfigurable robots [13] or localization of underwater robot [6]. In addition, many works focused on the use of Constraint Programming for planning [11, 3]. In these approaches, Constraint Programming is used as computation support to solve effectively large combinatorial problems. Like these approaches, our architecture will exploit the solving power of Constraint Programming.

In addition, our supervision architecture encodes elementary actions as discrete CSPs. The use of discrete CSPs may seem like a bold choice to tackle problems involving

[1] Because one of the actuator did not react properly, because an external pertubation has disturbed the execution, or because the CSP which modelled the previous action was not correct.

low-level commands. However, experimental results presented in Section 5 demonstrate that our approach can plan efficiently non-elementary behaviors. On the other hand, the discrete constraint networks combined by our architecture inherit the properties of *Constraint Propagation* of CSPs, which are fully exploited during the execution of a sensorimotor behavior in order to adjust it to the operational reality of the robot.

Finally, we choose to use Machine Learning in order to automatize the elementary actions modelling process. By this way, we wish to contribute to relieve the roboticians task during the sensorimotor behaviors design process, which constitutes a really hard step and can require a lot of time.

4 How Constraint Programming?

This section defines the supervision process of our architecture. For this, we consider a poly-articulated robot R constituted by p descriptors, which describe various dimensions of R and its environment, and by q actuators (*i.e.* motors). Let be $\Delta = \{\delta_1, \ldots, \delta_p\}$ the set of the descriptors of R, $\alpha = \{\alpha_1, \ldots, \alpha_q\}$ the set of its actuators, and $\mathcal{A} = \{a_1, \ldots, a_m\}$ the set of elementary actions that the robot can execute.

4.1 Modelling elementary actions with constraint networks

As briefly described in Section 2, our approach consists in modelling each elementary action $a_i \in \mathcal{A}$ with a constraint network. For this, we introduce the functions $varI$, $varA$ and $varF$, defined on $\mathcal{A}$ as follows: $varI(a_i)$ returns the set $\{\delta_1^I, \ldots, \delta_p^I\}$ of variables which model the descriptors of R **before** the execution of a_i, $varA(a_i)$ returns the set $\{\alpha_1, \ldots, \alpha_q\}$ of variables which model the actuators R **during** the execution of a_i, and $varF(a_i)$ returns the set $\{\delta_1^F, \ldots, \delta_p^F\}$ of variables which model the descriptors of R **after** the execution of a_i.[2]

Each elementary action $a_i \in \mathcal{A}$ is modelled as follows.

Definition 1 (CSP Model) *An elementary action $a_i \in \mathcal{A}$ is modelled by the constraint network $\mathcal{P}_i = (X_i, D, C_i)$ such that:*

$$\begin{cases} X_i = varI(a_i) \cup varA(a_i) \cup varF(a_i), \\ D = \{D(x_j) \mid x_j \in X_i\}, \\ C_i = Pre(a_i) \cup Actuators(a_i) \cup Post(a_i), \end{cases}$$

where $D(x_j)$ is the finite domain of possible values for $x_j \in X_i$, $Pre(a_i)$ models the conditions in which a_i can be executed, $Actuators(a_i)$ models how a_i is executed, and $Post(a_i)$ models the effects of a_i on the robot and its environment. $Pre(a_i)$ is defined on $varI(a_i)$, $Post(a_i)$ is defined on $varI(a_i)$ and $varF(a_i)$, and $Actuators(a_i)$ is defined on $varI(a_i)$, $varA(a_i)$ and $varF(a_i)$.

It could be noted that Definition 1 allows to acquire, for each elementary action $a_i \in \mathcal{A}$, a model which abstracts a set of possible instances of a_i. That is, our approach models a space of possible solutions for the execution of a_i. This makes easy the combination of several elementary actions as a conjunction of their respective CSPs. As a result, we can adjust the execution of a sensorimotor behavior after one of its elementary actions a_i has been performed, simply by re-solving the conjunction of CSPs with updated values on the variables representing the output of a_i. For instance, let us imagine an elementary action which can be performed to move a robot for a distance d, such that d is between 1 cm and 40 cm. The corresponding CSP will model the effective distance d covered by the robot depending on the actuators values, and will model that $1 \leq d \leq 40$. Let us consider now we want the robot to move for 55 cm. The robot can achieve this sensorimotor behavior by moving (for instance) for 30 cm at a first step, and for 25 cm at a second step. Instead of the predicted 30 cm, let us imagine that the robot has moved only for 29 cm at the first step (for instance because of an external perturbation). The sensorimotor behavior may still be achieved. Indeed, propagating the fact that the real covered distance is 29 cm (instead of 30) allows to determine that the robot has to move for 26 cm (instead of the predicted 25 cm) at the second step in order to cover the 55 cm. This kind of *minor corrections* will be fully exploited by the operating cycle of our architecture in order to adjust the execution of a sensorimotor behavior (see Section 4.5).

In our approach, elementary actions are seen as elementary *building blocks* which are combined in order to generate and supervise non-elementary behaviors. However, our architecture does not tackle high-level planning problems.[3] We choose consequently to limit our approach to classical planning. The formalism we use to encode elementary actions is an extension of the STRIPS langage, for which we introduce the component *Actuators*, which is used to encode the actuator behaviors.

4.2 Automatic CSPs building by constraint acquisition

In order to automate the modelling process, we have chosen to use the constraint network acquisition platform

[2] Indeed, $varI$ corresponds to the Initial state of R, $varF$ to the Final state, and $varA$ to its Actuators.

[3] As said previously, high-level missions will be supervised by control architectures such as the LAAS architecture, CLARATY or HARPIC.

CONACQ [5, 4]. In this subsection, we first present the constraint network acquisition problem. Then, we present how to use CONACQ to automatically model each elementary action with a CSP.

4.2.1 Constraint network acquisition

The goal of constraint network acquisition consists in automatically acquiring a constraint network from solutions and non solutions of the problem we want to encode as a CSP [5, 7]. The set $\mathcal{X}$ of the variables and their values domains $\mathcal{D}$ are known. The input data consist of E^+, a subset of solutions of the studied problem, and of E^- a set of non solutions. The goal of the acquisition process is to model the problem in a constraint solver. In the general use of contraint network acquisition, the learning bias $\mathcal{B}$ is then a constraint library from this solver.

A set of constraints belongs to the learning bias if and only if this set contains only constraints of the bias. Given a set of variables $\mathcal{X}$, their domains $\mathcal{D}$, two sets E^+ and E^- of instances on $\mathcal{X}$, and a learning bias $\mathcal{B}$, the constraint acquisition problem consists in finding a set of constraints $\mathcal{C}$ such that $\mathcal{C} \subseteq \mathcal{B}$, $\forall e^- \in E^-$, e^- is a non solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, and, $\forall e^+ \in E^+$, e^+ is a solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

4.2.2 Automatic acquisition of elementary actions with CONACQ

For the acquisition process of one elementary action $a_i \in \mathcal{A}$, we need a constraint library $\mathcal{L}$ that constitutes the learning bias, and two disjoint sets E^+ and E^- which respectively contain positive instances of a_i and negative ones.

To establish these training data, we can either carry out simulations with a 3D Computer Aided Design software, or directly execute actions with the real robot. We then distinguish executions that match with the elementary action a_i, which constitute the set E^+, and those which do not, and constitute E^-. Each training instance is a labelled (positive or negative) tuple $(\delta_1^I, \ldots, \delta_p^I, \alpha_1, \ldots, \alpha_q, \delta_1^F, \ldots, \delta_p^F)$ where $\delta_1^I, \ldots, \delta_p^I$ and $\delta_1^F, \ldots, \delta_p^F$ correspond respectively to the descriptor's state at the beginning and at the end of each action, and where $\alpha_1, \ldots, \alpha_q$ refer to the activator's behavior during the execution.

From this training data, the CONACQ platform will automatically build a constraint network which matches Definition 1.

The expressiveness of the learning bias is a capital aspect for the success of the acquisition process. The constraint library $\mathcal{L}$ must fulfil two partially opposite requirements: *simplicity*, in order to garantee good computational properties, and *modelling capability*, in order to efficiently describe each elementary action. The learning bias will have to be chosen consequently in close interaction with roboticians.

Experiments we present in Section 5 have demonstrated that CONACQ is able to acquire an accurate model for each elementary action. However, for complex robots, an accurate learning bias would perhaps be hard to determine. In this case, CONACQ would be used as a pre-processing tool for acquiring an initial model of each elementary action. The acquired CSPs would have to be then improved and reformulated in cooperation with roboticians in order to obtain accurate models.

4.3 Determining sensorimotor behaviors by planning

Given a sensorimotor behavior, our approach encodes the behavior as a task T, expressed with an initial state S_I and a goal S_F. Determining a plan for performing T consists in finding a sequence $\mathcal{S}$ of elementary actions from the set $\mathcal{A}$, such that if all these actions are performed from S_I, they allow to reach a state satisfying S_F.

To limit the solving time of this planning problem, we restrict the model of each elementary action a_i to its preconditions and its effects (like in the STRIPS formalism), given by the sets of constraints $Pre(a_i)$ and $Post(a_i)$, and we use a CSP-based planner inspired by CSP-PLAN [8]. As defined in this algorithm, we impose a bound k on the size of the plan. The problem is now to determine whether there is a plan of size k to achieve the goal of T from the initial state.

4.3.1 Encoding the planning problem as a CSP

Given a size k, the k-sized planning problem is encoded as follows:

1. The variables - To model the k-sized planning problem, we define $k+1$ sets $\Delta^t = \{\delta_1^t, \ldots, \delta_p^t\}$ of *descriptor variables*, where Δ^t is the descriptors set Δ at the step t, $0 \leq t \leq k$. Hence, $\forall t \in \{0 \ldots k\}$, $\forall i \in \{1 \ldots p\}$, δ_i^t models the descriptor δ_i of the robot at the step t. We define also k sets $\mathcal{A}^t = \{a_1^t, \ldots, a_m^t\}$ of *action variables* where, $\forall j \in \{1 \ldots m\}$, a_j^t is a Boolean variable such that a_j^t is true iff the action a_j is performed from the step t to the step $(t+1)$, $0 \leq t \leq k-1$.

2. The constraints - To obtain a correct model of the planning problem, we define the following sets of constraints:

1. *Initial state and goal.* The values of the descriptor variables δ_i^0 at step 0 and the values of descriptor variables δ_i^k at step k must respectively be compatible with the initial state and the goal of T. We thus define the corresponding constraints on variables δ_i^0 and δ_i^k, $1 \leq i \leq p$. For instance, we define the constraint

($v_{min} \leq \delta_i^k \leq v_{max}$) if the descriptor δ_i of the robot must take its value in the domain $[v_{min}, v_{max}]$ at the end of the plan,

2. *Preconditions verification.* An action $a_j \in \mathcal{A}$ can be performed from the step t to the step $t+1$ only when *all* of its preconditions are verified. Hence, for each step $t \in \{0 \dots k-1\}$ and for each action variable a_j^t, we add the constraint $a_j^t \rightarrow Pre(a_j)^t$, where $Pre(a_j)^t$ is the a_j's preconditions relative to the step t (*i.e.* defined on Δ^t),

3. *Action effects.* Similarly, $\forall t \in \{0 \dots k-1\}$, $\forall j \in \{1 \dots m\}$, the constraint $a_j^t \rightarrow Post(a_j)^t$ is added, where $Post(a_j)^t$ is defined on Δ^t and Δ^{t+1} and models the effects of a_j performed from the step t to the step $t+1$.

As such, our model is similar to the BASE-CSP built by CSP-PLAN, extended to non-Boolean variables. However in this paper, we limit our study to sequential plans (*i.e.* only one action performed at each step). For each step $t \in \{0..k-1\}$, we add two additional constraints: $atleast(\{a_1^t, .., a_m^t\}, 1)$ imposes that at least one action is executed from step t to step $t+1$, and $atmost(\{a_1^t, .., a_m^t\}, 1)$ imposes that at most one action is executed from step t to step $t+1$.

4.3.2 Solving the planning problem

To determine a plan for performing the given task T, according to CSP-PLAN, we invoke our planner by varying the length k (beginning with $k = 1$). For a given size k, we model the corresponding k-sized planning problem and send it to a CSP solver which determines if there exists solutions or not. If the solver reports no solution, we increase the size of the plan to $k+1$. This cycle continues until a plan has been found or until a upper bound on k has been reached. By this process, the first solution provided by the planner is an optimal plan (*i.e.* minimum number of steps).

If there exists a plan for performing T, the CSP-based planner will output a sequence $\mathcal{S}$ of elementary actions from $\mathcal{A}$ that the robot must execute to achieve T.

4.4 Modelling a sequence of elementary actions with a CSP

In this section, we present how the control module of our architecture determines how the successive actions of a sequence $\mathcal{S}$ (planned as previously described) must be executed by the robot in order to perform T. The control module encodes $\mathcal{S}$ as a CSP, noted *global-CSP*. This global-CSP is quite similar to the CSP previously defined for the planning process. However, we replace *action* variables by *actuator* variables in order to model how the actuators must react during the execution of the sequence $\mathcal{S}$.

To encode a given sequence $\mathcal{S}$ of k elementary actions as a CSP, we define $k+1$ sets $\Delta^t = \{\delta_1^t, \dots, \delta_p^t\}$ of *descriptor variables* and k sets $\alpha^t = \{\alpha_1^t, \dots, \alpha_q^t\}$ of *actuator variables*. Like in the model of the planning problem, $\forall t \in \{0 \dots k\}, \forall i \in \{1 \dots p\}$ δ_i^t models the descriptor δ_i at step t. $\forall j \in \{1 \dots q\}$ α_j^t models the actuator α_j from the step t to the step $t+1$, $0 \leq t \leq k-1$. Notice that α_j^t is no longer a Boolean (as opposed to Section 4.3.1).

The constraints of the global-CSP are then given by the successive actions of $\mathcal{S}$. The first elementary action of $\mathcal{S}$ must be executed from step 0 to step 1, the second elementary action of $\mathcal{S}$ must be executed from step 1 to step 2, and so on, the t-th elementary action of $\mathcal{S}$ must be executed from step $t-1$ to step t. Let $a_j \in \mathcal{A}$ be the elementary action to be performed from the step t to the step $t+1$, $0 \leq t \leq k-1$. To define correctly the execution of a_j, we add to the global-CSP the following sets of constraints. The set of constraints $Pre(a_j)^t$ which is the a_j's preconditions relative to step t is defined on Δ^t and added to the model. $Post(a_j)^t$, which models the effects of a_j, is defined on Δ^t and Δ^{t+1} and added, exactly like in the model of the planning problem. Finally, $Actuators(a_j)^t$, which models how the actuators must react from the step t to the step $t+1$, is defined on Δ^t, α^t and Δ^{t+1}, and added to the model.

We encode the initial state of the robot and the goal of the task T the same way as in Section 4.3.1. We add the following constraints to the global-CSP. For each descriptor variable δ_i^0, we add the constraint $(\delta_i^0 = v)$ where v is the value of the descriptor δ_i in the initial state. Similarly, if the descriptor δ_j must take its value in $[v_{min}, v_{max}]$ in the goal of the task T, we add the constraint $(v_{min} \leq \delta_j^k \leq v_{max})$.

Defined as above, the global-CSP correctly encodes the sequence $\mathcal{S}$ and the task T. If there exists a solution, the values of the descriptor variables δ_i^t indicate the successive values of the descriptor δ_i during the execution of $\mathcal{S}$. The values of the actuator variables α_i^t indicate how the actuator α_i must successively react in order to perform T.

4.5 Operating cycle for supervising the execution of a sequence of elementary actions

If a solution of the global-CSP is provided by the solver, the control module supervises step-by-step the execution of the computed sequence $\mathcal{S}$ and adjusts it to the operational reality.

After the execution of the elementary action performed from step t to step $t+1$, $0 \leq t \leq k-1$, the control module retrieves the current state of the robot, given by the set $\Delta =$

$\{\delta_1, \ldots, \delta_p\}$. This current state is compared to the predicted state, given by the values of the $\{\delta_1^{t+1}, \ldots, \delta_p^{t+1}\}$ descriptor variables of the global-CSP. The execution of the sequence is then adjusted differently depending on the gap between the current state and predictions:

- **Execution identical to predictions -** If the current state corresponds perfectly to predictions, the control module continues the execution of the sequence as defined originally in the global-CSP. The next elementary action is performed with the values of the actuator variables $\alpha_1^{t+1}, \ldots, \alpha_q^{t+1}$ for the robot's actuators.
- **Minor correction -** If the current state is not equal to predictions, the global-CSP is reduced to the remaining elementary actions of $\mathcal{S}$ and we impose the current state to the descriptor variables δ_i^{t+1} $\forall i \in \{1..p\}$. The solver determines[4] if it is possible to adjust[5] remaining elementary actions of $\mathcal{S}$ in order to achieve T from the current state without redefining $\mathcal{S}$. If such adjustements exist, the control module rectifies the execution of $\mathcal{S}$ by imposing those minor corrections to the next actions to be performed.
- **Replanning -** If it is impossible to achieve T from the current state with the remaining actions of $\mathcal{S}$, the control module calls again the planner to determine a new sequence $\mathcal{S}'$ of elementary actions to carry out T from the current state. The control module supervises then the execution of $\mathcal{S}'$ using the same operating cycle, until T is completed, or until the planner determines that T cannot be achieved from the current state.

5 Experiment

For empirical studies, we have used the CONACQ platform [4] as mentioned previously, and CHOCO[6] as constraint solver. Moreover, a video of the experiment described in this section is available at: `http://www.lirmm.fr/~paulin/Tribot/`.

5.1 The Tribot robot

In this section, we present the empirical studies we carried out on the TRIBOT robot. Illustrated by Figure 1, TRIBOT is a 3-wheeled mobile robot proposed in the LEGO MINDSTORMS NXT kit.[7]

TRIBOT is constituted by 3 servo motors. Two of them power the robot's driving base, while the third allows to open and close the TRIBOT's claw. TRIBOT is constitued by 4 descriptors. The *ultrasonic* sensor detects obstacle in front of TRIBOT and determines the separating distance (up to $255cm$). The *light* sensor, located under the robot, determines the color of the floor. The *touch* sensor detects when it is being pressed by something and when it is released again, and the *sound* sensor detects decibels (*i.e.* sound pressure). Finally, those servo motors and descriptors are linked to the NXT, a 32-bit microcontroller. Our supervision architecture uses the URBI language [2] to control servo motors and access sensors.

Figure 1. The TRIBOT robot.

In this experiment, the sensorimotor behavior that TRIBOT had to achieve was to grab (with the claw) a mug located in the surrounding area, using the supervision architecture we described.

5.2 Acquisition of elementary actions for Tribot

For the TRIBOT experiment, we choose to model five elementary actions: careful move, standard move, find target, open claw, and close claw. We use the following descriptors: the ultrasonic sensor provides the distance d between TRIBOT and the mug. We use the touch sensor as a bumper in order to determine if the robot is contact with the mug ($b = 1$ in this case, $b = 0$ else), and c is a Boolean descriptor such that c is true iff the TRIBOT's claw is open. For clarity reasons, the light sensor and the sound sensor have not been taken in account in our experiment. The three actuators of TRIBOT are given as follows: α_R (*resp.* α_L) refers to the servo motor of the Right wheel (*resp.* Left), and α_C is the servo motor of the TRIBOT's claw.

For this experiment, the training data sets E^+ and E^- are the result of real executions on TRIBOT. Each training instance given to CONACQ was a labelled (positive or negative) tuple $(d^I, b^I, c^I, \alpha_R, \alpha_L, \alpha_C, d^F, b^F, c^F)$ where the exponents I and F indicate respectively the state of the descriptors at the beginning and at the end of each action.

[4] The resolution of the remaining global-CSP is restarted from scratch.

[5] As we mentioned in Section 4.1 for the correction of the execution of elementary actions.

[6] Available at: `http://choco.sourceforge.net`.

[7] More details on `http://mindstorms.lego.com/`.

5.2.1 Acquisition of the *Careful move* elementary action a_1

The first elementary action we studied was the *Careful move* elementary action a_1, which must be executed when we want TRIBOT to move slowly for a precise distance. We gave thirteen positive and negative instances of a_1 (see Table 1) to CONACQ and the platform automatically built the following CSP:

$$a_1\textbf{: Careful move}$$
$$\begin{cases} Pre(a_1) = \{(d^I < 100), (b^I = 0), (c^I = 1)\} \\ Actuators(a_1) = \{(\alpha_R = \alpha_L = 24 \times (d^I - d^F)), (\alpha_C = 0)\} \\ Post(a_1) = \{(d^I - 10 \leq d^F \leq d^I - 1), (d^F = 0 \Rightarrow b^F = 1), \\ \qquad\qquad (c^F = c^I)\} \end{cases}$$

The acquired CSP models accurately the *Careful move* elementary action a_1. We highlight here its notable features: a_1 can be executed iff the distance between TRIBOT and the mug is less than one meter (*i.e.* $(d^I < 100)$) and iff the claw is open (*i.e.* $(c^I = 1)$). The constraint $(\alpha_C = 0)$ models the fact that the claw's actuator is not used during the execution of *Careful move*, and $(c^F = c^I)$ means that the claw remains open during the execution of this elementary action. The actuators α_R and α_L are controlled in *position* mode.[8] The constraint $(\alpha_R = \alpha_L = 24 \times (d^I - d^F))$ models the fact that α_R and α_L must be moved to a position equal to $24 \times d$ in order to move TRIBOT for a distance d. Moreover, the constraints of the $Post(a_1)$ component which involve d^F, *i.e.* $(d^I - 10 \leq d^F \leq d^I - 1)$, model the fact that the *Careful move* must be executed to move the robot for a distance between 1 and 10 cm. Finally, the constraint $(d^F = 0 \Rightarrow b^F = 1)$ encodes accurately the fact that TRIBOT is in contact with the mug at the end of a_1 if the final value of the ultrasonic sensor is equal to 0.

	d^I	b^I	c^I	α_R	α_L	α_C	d^F	b^F	c^F
e_1^+	80	0	1	240	240	0	70	0	1
e_2^-	120	0	1	240	240	0	110	0	1
e_3^+	50	0	1	24	24	0	49	0	1
e_4^-	60	0	1	24	24	10	59	0	1
e_5^+	15	0	1	48	48	0	13	0	1
e_6^-	70	0	0	120	120	0	65	0	1
e_7^+	20	0	1	72	72	0	17	0	1
e_8^-	50	0	1	200	200	0	40	0	1
e_9^+	35	0	1	120	120	0	30	0	1
e_{10}^-	80	0	1	24	24	0	79	0	0
e_{11}^-	101	0	1	240	240	0	91	0	1
e_{12}^+	8	0	1	24	24	0	7	0	1
e_{13}^-	0	0	1	0	0	0	0	0	1

Table 1. Training data for the acquisition of a_1.

[8] The position mode is used to move the motor to a given position (in degrees).

5.2.2 Acquisition of the other elementary actions

Using the same protocol, the four remaining elementary actions of TRIBOT have been automatically encoded by the CONACQ platform as the following constraint networks:

$$a_2\textbf{: Standard move}$$
$$\begin{cases} Pre(a_2) = \{(10 < d^I < 100), (b^I = 0)\} \\ Actuators(a_2) = \left\{(\alpha_R = \left\lceil (d^I - d^F)/0.9 \right\rceil), \right. \\ \qquad\qquad \left. (\alpha_L = \alpha_R), (\alpha_C = 0)\right\} \\ Post(a_2) = \{(d^I - 36 \leq d^F \leq d^I - 27), (d^F > 10), \\ \qquad\qquad (b^F = 0), (c^F = c^I)\} \end{cases}$$

$$a_3\textbf{: Find target}$$
$$\begin{cases} Pre(a_3) = \{(d^I > 100), (b^I = 0)\} \\ Actuators(a_3) = \{(\alpha_R = 5), (\alpha_L = -5), (\alpha_C = 0)\} \\ Post(a_3) = \{(d^F \leq 100), (c^F = c^I)\} \end{cases}$$

$$a_4\textbf{: Open claw}$$
$$\begin{cases} Pre(a_4) = \{(d^I > 10), (b^I = 0), (c^I = 0)\} \\ Actuators(a_4) = \{(\alpha_R = \alpha_L = 0), (\alpha_C = 10)\} \\ Post(a_4) = \{(d^F = d^I), (b^F = b^I), (c^F = 1)\} \end{cases}$$

$$a_5\textbf{: Close claw}$$
$$\begin{cases} Pre(a_5) = \{(d^I < 100), (c^I = 1)\} \\ Actuators(a_5) = \{(\alpha_R = \alpha_L = 0), (\alpha_C = -50)\} \\ Post(a_5) = \{(d^F = d^I), (d^F = 0 \Rightarrow b^F = 1), (c^F = 0)\} \end{cases}$$

5.3 Supervision of sensorimotor behaviors

As we mentioned before, the sensorimotor behavior that TRIBOT had to achieve was to grab a mug located in a surrounding area.

In the first experiment, the mug was located in front of TRIBOT, and the claw of the robot was closed. The sequences planned by our architecture had the following scheme: *Open claw*, a number[9] of *Standard Move*, *Careful move* (*i.e.* in order to be in contact with the mug) and finally *Close claw*. During this first experiment, the *grabbing* sensorimotor behavior was achieved only by adjusting the planned sequences to the reality of their executions. In the second experiment, perturbations occured. Initial conditions were the same and TRIBOT planned consequently the same kind of sequences than during the first experiment. However, the mug was moved during the execution of the planned sequence, which has forced a replanning. According to the TRIBOT's state, the CSP-based planner determined a new sequence of elementary actions, which begun with the *Find target* elementary action. The execution of this new sequence was then supervised by the control module according to the operating cycle of our approach.

[9] According to the distance between TRIBOT and the mug.

As shown on the video, TRIBOT has achieved the *grabbing* sensorimotor behavior. This experiment validates the capability of our architecture to design and supervise the execution of sensorimotor behaviours, by combining elementary actions modelled by CSPs automatically acquired by constraint acquisition.

6 Conclusion and Future Work

In this paper, we have proposed an architecture to interact with roboticians during the design of sensorimotor behaviors. Our architecture uses constraint network acquisition to *automatically* acquire CSPs which model the elementary actions of a robot. In a second time, a CSP-based planner combines the acquired CSPs to define a sequence of elementary actions which must be executed in order to achieve a given sensorimotor behavior. The execution of this sequence is then supervised and adjusted by a control module. Empirical studies have validated our approach for automatic design and supervision of sensorimotor behaviors.

In the future, our approach could be used by control architectures, such as the LAAS architecture, HARPIC or CLARATY, for designing and supervising sensorimotor behaviors, replacing the behaviors which are currently hand-designed by roboticians. The performance of our architecture is currently sufficient for the experiments we launched. However, improvements could be considered on two distinct aspects: the computation efficiency and the expressiveness. First, on robots with higher mechanical complexity than TRIBOT, we could face bad computation times. To increase the computational performance of our architecture, we could improve our planner and our control module. The current version of our planner, which is an extension of the BASE-CSP built by CSP-PLAN, could be improved by adding the set of the transformations that can be used by CSP-PLAN to increase its performances (see [8] for more details). Our control module could be improved in such a way that the corrections of an execution sequence would be computed incrementally, whereas they are currently computed with a restart *from scratch* of the CSP solver. Second, to increase the expressiveness of the CSPs we handle, *hybrid* constraint networks, mixing discrete and continous constraints, could be used to obtain a better model (according to the command and control theories) of each elementary action. This would garantee more accurate sensorimotor behaviors. However, the performance of the current generation of hybrid CSP-solvers are prohibitive for our kind of application.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[2] J.-C. Baillie. Urbi: Towards a universal robotic low-level programming language. In *IROS'05 (International Conference on Intelligent Robots and Systems)*, pages 820–825, August 2005.

[3] G. Beaumet, G. Verfaillie, and M. Charmeau. Estimation of the Minimal Duration of an Attitude Change for an Autonomous Agile Earth-Observing Satellite. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, pages 3–17, Providence, USA, September 2007.

[4] C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. A SAT-Based Version Space Algorithm for Acquiring Constraint Satisfaction Problems. In *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, pages 747–751, Porto, Portugal, October 2005.

[5] C. Bessiere, R. Coletta, B. O'Sullivan, E. Freuder, S. O'Connell, and J. Quinqueton. Semi-Automatic Modeling by Constraint Acquisition. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)*, pages 812–816, Springer Kinsale, Cork, Ireland, September 2003.

[6] L. Jaulin. Localization of an underwater robot using interval constraints propagation. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06)*, pages 244–255, Nantes, France, September 2006.

[7] A. Lallouet and A. Legtchenko. Two Contributions of Constraint Programming to Machine Learning. In *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, pages 617–624, Porto, Portugal, October 2005.

[8] A. Lopez and F. Bacchus. Generalizing GraphPlan by Formulating Planning as a CSP. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 954–960, Acapulco, Mexico, August 2003.

[9] D. Luzeaux and A. Dalgalarrondo. Hybrid architecture for autonomous robots, based on representation, perception and intelligent control. *Recent advances in intelligent paradigms and applications*, 113:37–56, 2003.

[10] J.-P. Merlet. A Generic Trajectory Verifier for the Motion Planning of Parallel Robots. *Journal of Mechanical Design*, 123:510–515, 2001.

[11] A. Nareyek, E. Freuder, R. Fourer, E. Giunchiglia, R. Goldman, H. Kautz, J. Rintanen, and A. Tate. Constraints and AI Planning. *IEEE Intelligent Systems*, 20(2):62–72, 2005.

[12] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The claraty architecture for robotic autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*, pages 121–132, Big Sky, Montana (USA), March 2001.

[13] Y. Zhang, M. Fromherz, L. Crawford, and Y. Shang. A general constraint-based control framework with examples in modular self-reconfigurable robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, pages 2163–2168, Lausanne, Switzerland., October 2002.