Proceedings of the Tenth Annual Conference on Advances in Cognitive Systems Submitted 9/2022; published 11/2022

# Motion Planning and Continuous Control in a Unified Cognitive Architecture

**Pat Langley** LANGLEY@STANFORD.EDU
Center for Design Research, Stanford University, Stanford, CA 94305 USA

**Edward P. Katz** EDPKATZ@STANFORD.EDU
Stanford Intelligent Systems Laboratory, Stanford University, Stanford, CA 94305 USA

## Abstract

This paper presents a theory that unifies symbolic reasoning and continuous control at a deeper level than previously attempted. The approach builds on the earlier PUG architecture, but it also incorporates some important new ideas: symbolic concepts can match to different degrees; symbolic skills include control equations that are modulated by mismatches of target concepts; intentions can operate in parallel and processes can predict future states to support mental simulation; and the resulting trajectories, annotated with utilities, guide search for effective motion plans. We also describe PUG/C, an implemented version of this theory and demonstrate its behavior in a simulated robot environment. We conclude by discussing links to earlier research in the area.

## 1. Introduction and Motivation

Early research in artificial intelligence focused on high-level tasks such as problem solving and multi-step reasoning, and the cognitive systems paradigm has carried on this tradition. However, humans also operate in a physical environment that requires them to exert fine-grained control over their effectors to achieve concrete goals. In addition, early efforts emphasized the role of symbolic mental structures, including the use of domain knowledge to guide behavior, which the cognitive systems movement has adopted as a core principle. Yet physical environments have a continuous character that is difficult to handle with discrete representations alone.

In contrast, the cybernetics and control communities (Bennett, 1996) have directly addressed fine-grained application of effectors in the external world, supporting this ability with continuous state encodings and quantitative control equations. But the control framework offers no obvious pathway to high-level cognitive processes, including the planning needed to achieve many physical goals, and it ignores the clear benefits of relational representations for situations and expertise. Clearly, neither paradigm provides a complete account of embodied intelligence, which poses an intellectual challenge for the broader research community.

In this paper, we present an alternative account of embodied intelligence that offers a deep unification of the two views, showing that they are compatible. We start by reviewing PUG, a cognitive architecture that took some steps in this direction but that also had important limitations. Next we present a revised theory that addresses these shortcomings and we report PUG/C, an augmented

architecture that uses the new assumptions to support fluid reactive control, flexible mental simulation, and continuous motion planning. After this, we report empirical demonstrations of these abilities in a simulated robotics environment. Finally, we reiterate our framework's key ideas and examine their connections to previous research on symbolic AI and continuous control.

## 2. A Review of the PUG Architecture

We desire a computational theory that supports the embodied character of human cognition while retaining insights about high-level symbol processing. In previous research (Langley et al., 2016), we have described PUG, a cognitive architecture for agents in continuous physical domains that:

- Grounds symbolic relations in quantitative descriptions of physical situations;
- Associates numeric utilities with symbolic goals that can reflect tradeoffs;
- Combines discrete actions with continuous parameters to specify fine-grained activities;
- Supports adaptive control that takes into account both the agent's objectives and its situation;
- Uses numeric simulation to generate motion trajectories that guide high-level planning.

PUG is a hybrid architecture that combines symbolic with numeric processing, borrowing ideas from ICARUS (Choi & Langley, 2018) but moving beyond them. In this section, we review this framework, examining commitments about representation and claims about cognitive processing.

### 2.1 Representation in the PUG Architecture

Like other cognitive architectures, PUG incorporates theoretical assumptions about the nature of mental content. The framework identifies three distinct types of long-term knowledge structures that remain stable over time. These include:

- *Concepts*, which encode generic symbolic relations among entities that are grounded in numeric attributes associated with those entities;
- *Motives*, which specify the conditions on when relations are desired or undesired, including their utility as a function of numeric attributes; and
- *Skills*, which encode generic discrete activities and their relational outcomes, but also include continuous equations for control attributes.

Concepts state how to decide what relations are true in the environment, motives indicate how to determine their value for the agent, and skills specify how to achieve them. Both unary categories (e.g., *tower*, *disk*) and relational concepts (e.g., *facing*, *between*) are defined in terms of these features. This is directly related to claims that cognition is *embodied*.

For example, an agent might include concepts for when it is *at* an object or *facing* it. These would be grounded in the agent's distances and angles relative to other entities in the environment. Its knowledge base might also contain motives that declare the desirability of being near a target or far from an obstacle. These can refer to defined concepts, as well as to values of their numeric attributes. Moreover, the agent may have skills for approaching a target object or turning to face it, referring to defined concepts in its conditions or effects. These would include equations for calculating the rate of forward motion or linear force and for angular motion or rotational force.

In addition, PUG postulates that these long-term structures have short-term, dynamic analogs, each of which is an instance of the corresponding generic element. These include:

- *Beliefs*, which are concrete instances of concepts that specify symbolic relations and have associated numeric attributes;
- *Goals*, which are desired or undesired beliefs with associated numeric utilities that can change as a function of the situation; and
- *Intentions*, which are instances of skills that refer to specific entities and that predict beliefs their execution will produce.

A collection of beliefs that the agent holds at the same time is a *state*, whereas a set of goals denotes a desired state that may or may not hold. Intentions are organized into sequential *plans*, which are further embedded in *search trees* that led to their construction.

### 2.2 Processing in the PUG Architecture

The PUG architecture also makes commitments about the mechanisms that operate over these mental structures. As in most computational treatments of extended cognition, these involve the repeated use of long-term content to update dynamic elements. The framework assumes three distinct processing levels, each with its own cognitive cycle:

- *Belief and state processing*, which matches concepts against percepts to infer beliefs, uses motives to compute goal utilities, and carries out a single step of skill execution.
- *Mental simulation*, which invokes state-level processing repeatedly to generate a trajectory of belief states, goals, and associated utilities.
- *Problem solving*, which carries out heuristic search through a space of task plans, each of which comprises a sequence of simulated intentions and associated utilities.

Each level builds on structures produced by the previous one, with state processing providing the raw material for mental simulation, which in turn offers the alternatives for problem solving and the evaluation scores to guide heuristic search.

State processing is related to the operation of production system architectures (Neches, Langley, & Klahr, 1987), as it relies on pattern matching and rule application to produce a sequence of mental states. Within this level, conceptual inference generates beliefs that lets the motivational module assign utilities to goals and that lets skill execution carry out agent intentions. However, it comes closer to Nilsson's (2001) design for a *tower architecture* and to ICARUS (Choi & Langley, 2018), which also separate inference from execution. Problem solving also mimics that in found ICARUS, in that it searches for sequences of durative actions that achieve goals, but PUG relies on mental simulation of skills rather than on their execution in the environment.

### 2.3 PUG's Successes and Limitations

Langley et al. (2016) demonstrated PUG's abilities on ten mission scenarios that involved a robot delivering sensors to target sites in the presence of danger areas. They provided the system with skills for turning left or right to face an object, moving toward an object, refueling at depots, de-

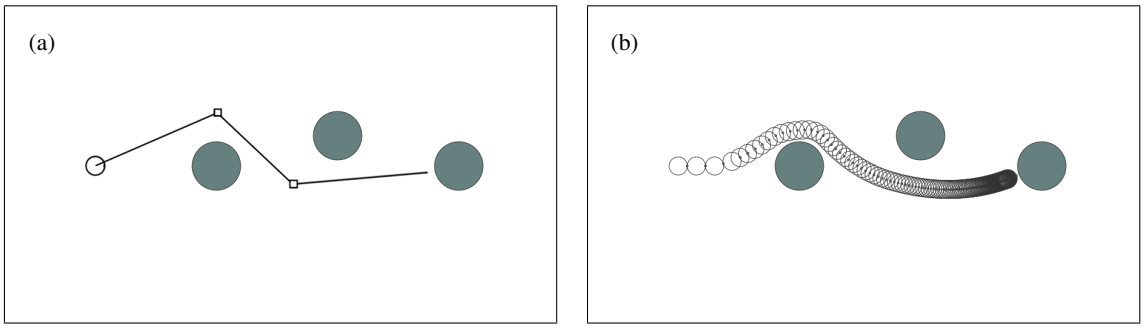


*Figure 1.* (a) A plan that the original PUG architecture would generate for moving the robot on the left to the target object on the right when waypoints are provided; (b) a more efficient and natural trajectory with no waypoints that the system could not produce. In the first plan, turning and moving forward must happen sequentially; in the second plan, the two activities take place in parallel.

positing sensors at targets, and collecting samples. These runs showed the architecture's ability to ground symbolic concepts in physical descriptions, combine discrete actions with continous parameters, and invoke mental simulation to envision and evaluate trajectories, which it used to guide search for extended plans that reasoned about tradeoffs and inconsistent goals. In some cases, the system decided to abandon goals because side effects would offset the benefits of achieving them.

However, they also revealed important limitations, most of them at the level of continuous control. In particular, PUG could not carry out more than one skill at the same time, such as moving and turning toward an object in parallel, or modulate a skill's behavior to reflect other factors, such as encountering obstacles on the way to a target. For instance, it could generate the inefficient plan in Figure 1 (a), but only if provided waypoints around the obstacles; it could not produce the smoother, more natural trajectory in Figure 1 (b). Neither could the system consider the effects of natural processes, such as friction or headwinds, when predicting future states. An improved version of the architecture would address these issues while retaining strengths of its predecessor.

## 3. A Theory of Motion Planning and Control

In response to these limitations, we have developed a revised theory of embodied agency that addresses them. In this section, we present tenets of the new framework that differ from the earlier one. As before, we first discuss postulates about representation and then turn to processes that operate over them. Readers should assume that unmentioned features of the architecture remain unchanged.

### 3.1 New Representational Postulates

The new framework retains the original distinction among concepts, motives, and skills, but it alters them in important ways to support more flexible and fluid control. One crucial change to the architectural theory is that it now posits:

- Concepts are *graded* in that they include functions to specify the degree to which they match. These refer to numeric attributes associated with entities or relations among them.

For instance, the match function for the concept *robot-at* might vary the robot's distance to an object, whereas the one for *robot-facing* might refer to its angle. The idea that some entities or situations fit a concept description better than others is akin to the notion of *typicality* in categorization (Rosch & Mervis, 1975). This is not the same as incorporating probability distributions in the concept description, although it does not exclude them.

Another revision concerns skills, which encode knowledge about physical activities in which the agent can take part. In particular, the new framework postulates that:

- Skills have associated *target concepts* they aim to achieve, as well as *control equations* whose effects are proportional to the degree these concepts are mismatched.

For example, a robotic agent's skill for moving to an object might include a *robot-at* relation it aims to achieve, whereas one for turning to an object might have *robot-facing* as its target. Because such concepts match situations to varying degrees, they can serve as error signals, much like the difference between the state and a 'set point' in classic feedback control. We can also view such target concepts as sources for potential fields that attract or repel the agent. At the same time, they are analogous to symbolic effects that appear in operators for symbolic planning.

One drawback of the previous theory was that knowledge for both calculation of actions and their effects was embedded in skills, only one of which could be active at a time. This meant that the agent could not pursue multiple actions, such as moving forward and turning, in parallel, and it could not reason about external forces. In response, the new framework also claims that:

- *Processes* specify how the attributes of objects in the environment change in response to their current values and the values of control attributes.

This revision factors knowledge about dynamics into skills, which determine the values of control attributes, and processes, which predict changes to the environment. It also lets the agent model the influence of mechanisms like wind and gravity over which it has no control.

Naturally, each of these assumptions about long-term cognitive structures also have corresponding ones about shorter-term elements. These include postulates that:

- Beliefs specify the *veracity* to which their concepts match and their *utility* obtained from motives;
- Intentions specify their *activation* based on mismatch of their target concept; and
- Events predict *changes* to attributes based on the current environment and control values.

As the agent or objects move in the environment, these quantities may change, causing these numbers to vary while the basic relations remains the same. In AI planning research, a state is simply a conjunction of relations; in our framework, a state as a point in an N-dimensional space, where each dimension refers to a belief's veracity or utility or to an intention's activation level. The new theory also lets the agent pursue multiple skills in parallel, so it includes a new structure that specifies a *set* of intentions.[1] We will see that, when combined with other information, these imply a physical trajectory over time, so we will refer to such sets as *motion plans*. They play the same role in higher-level task plans as did single intentions in the earlier framework.

1. Clearly, some intentions are mutually exclusive, such as avoiding an obstacle on the left and the right, which suggests the need for another form of knowledge – *constraints* – that we will not address here.

### 3.2 New Processing Postulates

Of course, these extended representations can have no effect on agent behavior without augmented mechanisms to interpret them, which the new framework also incorporates. One important change along these lines is that:

- Conceptual inference computes the *veracity* of each belief by evaluating the match function for the concept it instantiates.

This numeric score ranges from zero (no match) to one (a perfect match), with it varying over time in response to changes in the attributes used to compute it. For instance, the veracity of *(robot-at R1 O1)* will vary with the robot's distance from the object. The theory also posits a more adaptive mechanism for skill execution that builds on the results of conceptual inference:

- Skill execution treats the mismatch of each intention's target concept as its *activation*, which it combines with control equations to compute the values for control attributes.

This is a variation on feedback control that uses one minus the target concept's veracity as the error signal. Combined with the previous postulate, this lets the theory retain the symbolic character of skills while gaining the flexible character of continuous control.

Another change to the theory is that the agent can carry out multiple intentions in parallel, which raises issues about how to handle their interactions. Here the theory claims that:

- To determine the values for control attributes on a given cycle, skill execution sums the values computed by different intentions.

For instance, an intention for approaching object *O1* may cause a turn to the right, while another for avoiding object *O2* may cause a turn to the left. In such cases, the influences on control attributes are added, much as in the calculation of vector sums in potential field approaches to control (Khatib, 1985). In addition, the framework includes mechanisms for using processes to make predictions:

- Process interpretation computes expected changes to attribute values as a function of the current state and control values.

As with skill execution, if multiple processes affect the same attribute, then their influences are combined. For example, the robot's forward motion resulting from control settings will be offset if it encounters a headwind, with the resultant movement being their vector sum.

Finally, the possibility of parallel skill execution introduces a new level of choice for agents, since they must decide which set of intensions to pursue. Each such set, if executed, will produce a distinct trajectory over time and, combined with process-driven prediction, the agent can simulate these trajectories to select among them. We will refer to search through the space of intention sets as *motion planning*, a cognitive activity that occupies an intermediate level between state processing and task planning that did not occur in the previous framework.

## 4. The PUG/C Architecture

We have incorporated these theoretical ideas into PUG/C, an extension of the earlier PUG architecture (Langley et al., 2016). This earlier system also relied on grounded concepts and embedded continuous behavior within symbolic skills, but it did not support graded categories, feedback con-

*Table 1.* (a) Two PUG/C concepts for the robot domain, each of which includes a relational head, observed entities, and a veracity function based on the entities' attributes. The function $(linear\ obs\ max\ min)$ returns 1.0 if the observed value $obs = max$, 0.0 if $abs(obs) \geq min$, and $abs(obs/(max - min))$ within that range. (b) Two PUG/C motives for the same domain, which specify utility functions for beliefs in their heads and indicate whether they address achievement or maintenance goals.

```
(a) ((robot-at ^id (?r ?o) ^distance ?d)
      :elements   ((robot ^id ?r ^radius ?rr)
                   (object ^id ?o ^distance ?d ^radius ?or))
      :veracity   ((linear ?d (+ ?rr ?or) 10.0)))
    ((robot-facing ^id (?r ?o) ^angle ?a)
      :elements   ((robot ^id ?r) (object ^id ?o ^angle ?a))
      :veracity   ((linear ?a 0.0 45.0)))
```

```
(b) ((robot-at ^id (?r ?o))
      :conditions ((robot ^id ?r ^radius ?rr)
                   (object ^id ?o ^type target ^distance ?d ^radius ?or))
      :function   (cond ((< ?d (+ ?rr ?or 0.25)) 10.0) (t 0.0))
      :type       achievement)
    ((approaching ^id (?r ?o))
      :conditions ((robot ^id ?r ^radius ?rr)
                   (object ^id ?o ^type obstacle ^distance ?d ^radius ?or))
      :function   (cond ((< ?d (+ ?rr ?or)) -20.0) (t 0.0))
      :type       maintenance)
```

trol, parallel uses of intentions, process-guided prediction, or motion planning. In this section we describe the new architecture, focusing first on its representation of cognitive structures and then on how it interprets them to achieve an agent's objectives in an environment.

### 4.1 PUG/C Representation and Syntax

PUG/C provides a programming language that supports the construction of embodied intelligent agents. The syntax covers long-term knowledge structures that are stable over time and short-term elements that change during processing. This extends the earlier PUG notation that reflect its new theoretical commitments. Table 1 shows two *conceptual rules* for the two-dimensional robotic domain depicted in Figure 1. These define the relations *robot-at* and *robot-facing*. Each rule has a head that specifies a predicate and a set of attribute values, including an identifier for the relation. In addition, an *:elements* field describes a set of typed objects, each with an identifier and numeric attribute values, and a *:tests* field with Boolean tests that must be satisfied for the concept to match. The *:veracity* field specifies how to compute the concept's degree of match as a function of bound variables; this supports the notion of graded category membership.

Table 2 presents examples of *beliefs* that describe the scenario in Figure 1 from the robot's egocentric perspective. The four percepts – primitive beliefs that come directly from the environment or prediction – each specify an object type, an object identifier, and attribute values that describe it.

*Table 2.* Four percepts for the robot domain that describe objects the agent perceives for the scenario in Figure 1 (a) and five beliefs that describe its relations to these objects. Each structure includes a predicate, an identifier (which may be a list), attribute values, and a veracity score (in brackets) between zero and one. Beliefs that have a score below 0.5, such as *(robot-at ^id (R1 O3)*, do not appear in memory.

```
(robot ^id R1 ^radius 0.15 ^move-rate 0.0 ^turn-rate 0.0) [1.0]
(object ^id O1 ^distance 2.0 ^angle 0.0 ^radius 0.4) [1.0]
(object ^id O2 ^distance 4.123 ^angle 14.032 ^radius 0.4) [1.0]
(object ^id O3 ^distance 6.0 ^angle 0.0 ^radius 0.4) [1.0]
(robot-at ^id (R1 O1) ^distance 2.0) [0.847]
(robot-at ^id (R1 O2) ^distance 4.123) [0.622]
(robot-facing ^id (R1 O1) ^angle 0.0) [1.0]
(robot-facing ^id (R1 O2) ^angle 14.032) [0.688]
(robot-facing ^id (R1 O3) ^angle 0.0) [1.0]
```

The latter include the object's distance and angle from the robot in agent-centered polar coordinates, along with their radii, since they have circular cross sections. The table also contains five inferred beliefs (instances of defined concepts) about the robot's relations – *robot-at* and *robot-facing* – to the perceived objects *O1*, *O2*, and *O3*. These contain more than a symbolic predicate; they include attribute values derived from their constituent entities. Moreover, each belief includes a *:veracity* score that reflects the degree to which it matches its respective concept. For instance, *(robot-facing R1 O2)* has the score 0.688 because the instantiated veracity expression *(linear 14.032 0.0 45.0) = 0.688*. This score, along with the belief's derived attribute values, can change over time while its symbolic aspects remain stable.

PUG/C also provides a syntax for skills that describe how to achieve the agent's objectives. Table 3 (a) presents two examples from the robot domain, one for moving toward a given object and another for turning to face an object. Each skill has a head that specifies a name and set of arguments, along with an *:elements* field that describes the arguments' types and values for a subset of their attributes. As in conceptual rules, a *:tests* field includes Boolean tests that must be satisfied for the skill to apply. Most important, a skill specifies a *:target* relation that it aims to achieve and a *:control* field with expressions for computing values for control attributes as a function of the degree to which the target concept is *mismatched*. [2]

The architecture also incorporates a notation for processes, with two examples shown in Table 3 (b). This includes an *:elements* field that describes arguments and types, along with an optional *:tests* field with Boolean expressions. The key differences from skills are that they specify no control equations and they lack a target concept, as they are not teleological in character. Instead, they incorporate a *:changes* field that specifies how values for attributes of one or more entities will change as a function of variables matched in the *:elements* field. Thus, they encode causal knowledge about the effects of actions or environmental forces.

Just as beliefs in PUG/C are instances of general concepts, so *intentions* are instances of skills and *events* are instances of processes. Each intention provides a skill name and arguments, along

2. The mismatch score is simply one minus the match score reflected by the target belief's *:veracity* field.

*Table 3.* (a) Two skills for the robot domain, each of which includes a relational head, a set of observed entities, arithmetic tests, control equation, and a target concept. The *move-to* skill influences the control attribute *move-rate*, while *turn-to* affects *turn-rate*. (b) Two processes that describe the effects of these control attributes on the robot's *distance* and *angle* to an object. The symbols **da* and **da* denote trigonometric functions for computing linear and angular change.

```
(a)((move-to ?r ?o)
     :elements ((robot ^id ?r ^turn-rate ?t) (object ^id ?o ^angle ?a))
     :tests    ((> ?a -90) (< ?a 90))
     :control  ((robot ^id ?r ^move-rate (* 0.3 $MISMATCH)))
     :target   ((robot-at ^id (?r ?o))))
   ((turn-to ?r ?o)
     :elements ((robot ^id ?r)
                (object ^id ?o ^angle ?a ^distance ?d))
     :control  ((robot ^id ?r ^turn-rate (* 5.0 (sign ?a) $MISMATCH)))
     :target   ((robot-facing ^id (?r ?o))))
```

```
(b)((move-relative ?r ?o)
     :elements ((robot ^id ?r ^move-rate ?m)
                (object ^id ?o ^distance ?d ^angle ?a))
     :changes  ((object ^id ?o ^distance (*dd ?d ?a ?m) ^angle (*da ?d ?a ?m))))
   ((turn-relative ?r ?o)
     :elements ((robot ^id ?r ^turn-rate ?t)
                (object ^id ?o ^angle ?a))
     :changes  ((object ^id ?o ^angle (* -1.0 ?t))))
```

with a mismatch score for the target belief and values for the associated control attributes. The symbolic aspects remain unchanged over time, while the numeric values vary as the robot or its environment changes. For instance, for the scenario in Figure 1 (b), the agent might have the intention set *((move-to R1 O3) (turn-to R1 O3) (avoid-on-left R1 O1) (avoid-on-right R1 O2))*. The target mismatch and control values for each intention would vary over time, while its symbolic relation would remain the same. Events encode only the process name, its arguments, and the derivatives they predict for the current situation.

Finally, PUG/C incorporates the theory's notion of encoding *motion plans* as sets of intentions. The continuous trajectories associated with these structures are generated as needed, during mental simulation, rather than stored in memory. However, the intention sets are retained either as basic steps in higher-level task plans, which are themselves embedded in search trees that contain alternative courses of action considered by the agent during problem solving.

### 4.2 Belief and State Processing

Like other cognitive architectures, PUG/C operates in discrete cycles, but this occurs at five cascaded levels of processing, with results from each one providing raw material for the next. The two lowest levels focus on belief and state processing, with the first addressing conceptual inference. On each inference cycle, the module finds all conceptual rules with elements whose types match

against current percepts. For each match, the system computes derived attribute values, calculates the *:veracity* score, and, if this exceeds a threshold, adds an inferred belief to a state memory. Next the module finds rules with elements whose types match a subset of the percepts and these inferred beliefs, leading to new beliefs that are added to the state description if their scores are high enough. This procedure continues until no more conceptual matches occur, giving a set of beliefs and percepts that encode the current state. For instance, given the scenario in Figure 1, the concepts in Table 1 and the percepts in Table 2 would produce the beliefs in Table 2.

At the second level, PUG/C's state-processing module examines the current set of intentions. In each case, it checks to see whether the elements and tests are satisfied by the current belief state. If so, then the module accesses each intention $I$'s target belief and its associated *:veracity* score. The interpreter substitutes one minus this score for the symbol *$MISMATCH* in $I$'s control equations, along with the values for any bound variables. Next the system evaluates the instantiated expression to compute the value for each control attribute. For example, given the intention *(move-to R1 O1)* and the *:veracity* score 0.518 for target belief *(robot-at ^id (R1 O1))*, the mismatch for would be $1 - 0.518 = 0.482$ and the control attribute *move-rate* would be $0.3 \times 0.482 = 0.145$. When more than one intention applies on a cycle, the module computes target mismatches and control values for each one. Different intentions may affect the same control attribute, in which case PUG/C takes the vector sum of their various outputs.

Once the architecture has computed the summed values for its control attributes, it uses matched instances of processes (events) to predict changes to the environment. Some processes calculate the effects of agent actions, whereas others describe only external influences. Each event predicts a change to one or more environmental attributes, with their individual effects being summed to determine overall results. For instance, given the value 0.145 for the control attribute *move-rate* from above, the current *distance* of 5.105 and *angle* of 33.189 to *O1*, the event *(move-relative R1 O1)* predicts a chance of $-0.121$ for *distance* and 0.913 for *angle*. Different instances of the *move-relative* process also predict changes to the robot's *distance* and *angle* to other objects.

The state-processing module also matches all motives against the current beliefs. For each matched instance, it substitutes bound variables into the associated utility function, computes the result, and deposits it in the *:utility* field of the corresponding belief. If multiple motives have the same head, PUG/C assigns the sum of their utilities to this belief. This mechanism is identical to that in the original architecture except it creates no separate goal structures. Instead, it stores utility with beliefs themselves, even when their *:veracity* scores are too low to include them otherwise.

### 4.3 Mental Simulation and Motion Planning

The next two layers are responsible for mental simulation and motion planning. The first relies on a simple iterative mechanism. Recall that, combined with the calculation of control attributes by skill execution, processes let the architecture predict the next physical state in terms of what the agent can expect to perceive. This lets it apply conceptual inference to create a new belief state, which in turn lets it invoke skill execution, process prediction, and motivational processing. When applied repeatedly, this mechanism of mental simulation produces a trajectory of belief states over time. Each simulated trajectory follows deterministically from an initial situation and a set of intentions.

PUG/C lets its users provide such a motion plan manually, but it can also find such structures using heuristic search, using mental simulation and utility calculation to guide it. Motion planning starts with a set of desired relations that it wants to achieve. To this end, PUG/C retrieves candidate intentions for each relation by matching them against target concepts of skills. If this produces multiple options for the desired relations, PUG/C considers all combinations, as there will seldom be more than a few. Each set of intentions serves as an initial motion plan from which to initiate search. For every such initial plan, the architecture uses mental simulation to envision the trajectory that would result if it were executed in the environment. On each time step, it uses applicable motives to compute the utility of each belief and stores a running total of values associated with those beliefs. Next PUG/C retrieves intentions that would reduce or eliminate beliefs that are sources of negative utility, favoring ones that occurred earlier in the trajectory based on stored timing statistics. The system adds each of these to the current motion plan, simulates the alternatives, and selects the best. However, the associated trajectory may introduce new sources of negative utility (e.g., obstacles) that require additional intentions to address. This process continues until it cannot find any way to improve on the best candidate found so far, at which point it halts.

In summary, PUG/C carries out greedy search through a space of motion plans, starting with initial intentions that address target beliefs and elaborating it to improve average utility. Mental simulation produces a trajectory for each node in the search tree, but every candidate comprises a discrete set of intentions. This approach differs radically from classic methods, which either introduce waypoints that guide continuous control or carry out search through a fine-grained discretized space. In contrast, PUG/C combines conceptual inference, feedback control, and process-driven prediction to generate deterministic trajectories, then uses motives to assign utilities and to retrieve intentions that improve on them. The architecture passes the selected motion plans to its problem solver, which uses them as elements in higher-level search through a space of task plans. This facet has not changed from the earlier version of PUG, so we will not discuss it here.

## 5. Empirical Demonstrations

To demonstrate that PUG/C supports the target abilities described earlier, we developed a two-dimensional environment in which a mobile robot can sense its surroundings and control its motion. The agent perceives other nearby objects, including attributes like their radii, distances, and angles. The robot's absolute position and orientation is known to the simulator but not to the agent itself, which senses relations to other entities in egocentric polar coordinates. The environment obeys a form of Aristotelian physics, in that objects move only in response to explicit actions or forces. The agent can set values for two control attributes – *move-rate* (distance per cycle) and *turn-rate* (angles per cycle). We can create different scenarios by specifying the robot's initial pose and the number, type, location, and attributes of other objects. We can also include natural processes that affect the robot or other objects, leading to nonvolitional changes.

### 5.1 Combining Intentions with Processes to Simulate Trajectories

Our first demonstrations show that PUG/C can carry out multiple intentions in parallel and use processes to generate a simulated trajectory of environmental behavior. We provided the system with two concepts *robot-at* and *robot-facing* from Table 1, along with the skills *move-to* and *turn-*

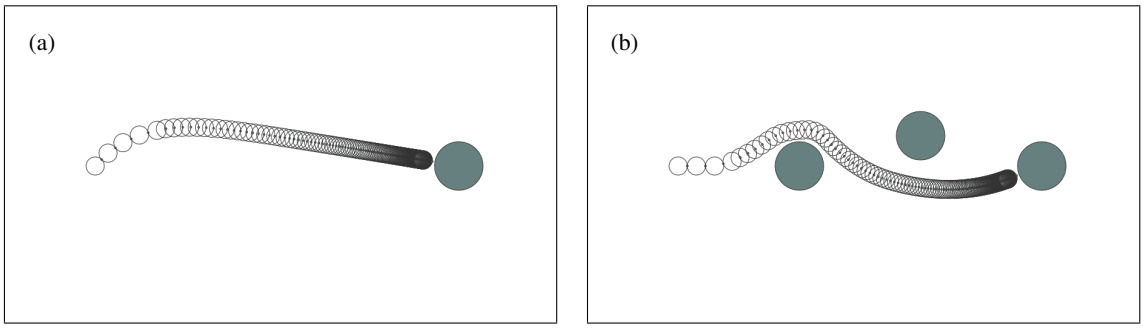


*Figure 2.* Two scenarios in which a robot must approach a nearby object: (a) when the robot is facing away from the target, it turns and moves forward simultaneously; (b) when two obstacles fall on the robot's path to the target, it veers left to avoid the first and right to evade the second. Traces show the robot's pose at equal time intervals, illustrating that its position and orientation change more slowly as it approaches the objective.

*to* from Table 3. We also gave it two intentions (instances of these skills) – *(move-to R1 O1)* and *(turn-to R1 O1)* – with target beliefs *(robot-at R1 O1)* and *(robot-facing R1 O1)*, respectively. Figure 2 (a) shows the behavior produced by these knowledge structures when *O1* is initially 6.0 units away from the robot and 45 degrees to its right. Conceptual inference shows that *(robot-facing R1 O1)* has an imperfect match score (initially 0.0), so the system invokes the intention *(turn-right-to R1 O1)*, which sets a positive value the control attribute *turn-rate* (initially $-10.0$). The intention *(move-to R1 O1)* also has a poorly matching target belief, *(robot-at R1 O1)*, so it sets the value for the control attribute *move-rate*. These lead the robot's position and orientation to change in parallel, with both slowing down over time as the *:veracity* score for each target belief approaches unity. The run continues for 145 iterations, but *(turn-right-to R1 O1)* becomes inactive after cycle 137, as the robot is facing *O1* at this point and it needs only to move forward.

We also tested PUG/C on scenarios that require the robot to avoid obstacles it encounters on the way to a target object. Here we provided the system with the same concepts and skills as in the previous run. However, we included a new concept, *approaching*, that holds when the robot is approaching an object, with the veracity depending on how much it will clear the obstacle given its current course. In addition, we gave it four intentions – *(move-to R1 O3)*, *(turn-to R1 O3)*, *(avoid-on-left R1 O1)*, and *(avoid-on-right R1 O2)*. The latter referred to two skills for avoiding objects on the left and on the right, both with *approaching* as their target concept. These do not affect the *move-rate* parameter; they only alter the robot's turn rate, which is necessary to swerve around the obstruction. This influence conflicts directly with the agent's intention to face the target object, but its effect becomes stronger as the robot gets closer to the obstacle and halts after it has passed. In the meantime, the intention for moving to the target continues to draw the robot forward.

Figure 2 (b) shows the resulting behavior when the agent must avoid two obstacles on its way to the target. In this run, the intention *(avoid-on-left R1 O1)* only starts to affect behavior on cycle 3, when the robot first approaches object *O1*. At this point, its effect on *turn-rate* becomes dominant and the robot starts to veer around the obstacle. However, once it has passed *O1*, this influence disappears and the robot turns back to *O3* until it comes close to *O2*, the second obstacle. Starting

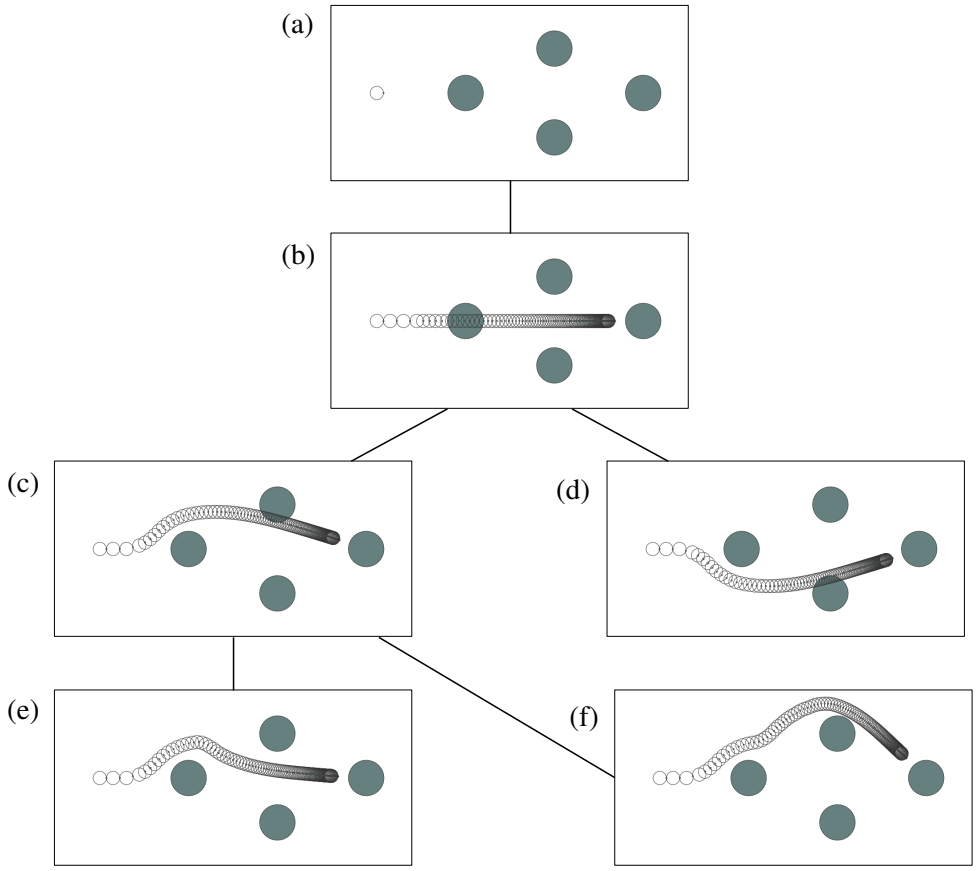


*Figure 3.* The tree explored during heuristic search for a motion plan that will take the robot from its initial position on the far left to the target object on the far right. At each level of search, PUG/C adds candidate intentions that should improve utility, generates a simulated trajectory, and selects the best alternative based of scores. In this run, the system settles on the motion plan depicted in (e).

on cycle 14, *(avoid-on-left R1 O2)* causes the robot to turn in the other direction, but its influence ends by cycle 18, when a collision is no longer imminent. Note that *(turn-to R1 O3)* is active during most of the run, but the avoidance intentions have greater impact. These traces provide evidence that PUG/C combines intentions in an effective manner, reproducing the behavior of potential fields in classic control systems. They also show the architecture can combine skill execution with process prediction to mentally simulate motion plans that involve continuous behavior.

### 5.2 Searching for Motion Plans

We also wanted to demonstrate PUG/C's ability to generate motion plans using heuristic search. We presented the system with the scenario in Figure 3 (a), in which the robot on the left must reach the target object on the right while avoiding three obstacles. The architecture initially retrieves two

intentions, *(move-to R1 O3)* and *(turn-to R1 O3)*, that should achieve the two target beliefs we provided, *(robot-at R1 O3)* and *(robot-facing R1 O3)*. This produces the trajectory in Figure 3 (b), which achieves these objectives but has the average utility $-1.137$ because it runs through object *O1*. The belief *(approaching R1 O1)* is the only source of negative utility, so PUG/C retrieves two intentions, *(avoid-on-left R1 O1)* and *(avoid-on-left R1 O1)*. The system adds each to the original intentions and simulates both plans, producing the trajectories in Figure 3 (c) and (d).

Both motion plans have average utilities of $-0.652$ because, although they bypass *O1*, their trajectories intersect another obstacle, one on the left and another on the right. Because they have equal scores, PUG/C selects one of them at random, say the option in (c), and examines the source of the negative utility, in this case *(approaching R1 O2)*. As before, the system retrieves two intentions that could eliminate this relation: *(avoid-on-left R1 O2)* and *(avoid-on-left R1 O2)*. The architecture adds each option to a new plan and simulates their behavior to generate the trajectories in Figure 3 (e) and (f). The first alternative, in which the robot turns right, has an average utility of $0.097$, whereas the second, in which it turns left another time, has $0.090$ as its utility. For this reason, PUG/C selects the former and, since this motion plan involves no other sources of negative value, returns this motion plan as the final result. We have also tested the system on a more complex scenario in which the robot must run a gauntlet of six obstacles, which produced similar results.

## 6. Related Research

Our extended theory incorporates ideas not only the original PUG architecture but from the broader literature, yet it also moves beyond the latter in important ways. There has been considerable work on combining symbolic and numeric processing for sequential decision making, but none has attempted our deep unification. For instance, 'hybrid' control systems (Goebel, Sanfelice, & Teel, 2009) embed continuous control within discrete finite-state networks, with different equations for each mode, but do not join the two paradigms at the control level. Similarly, extensions to the Soar (Laird, 2012) and ACT-R (Salvucci, 2006) architectures use feedback control in physical settings, but discrete processing calls on continuous computation as a subroutine. Planning formalisms like PDDL+ (Cashmore et al., 2020; Fox & Long, 2006) augment symbolic operators with difference equations that apply across cycles, but they do not support adaptive control. Like the original PUG, ICARUS (Choi & Langley, 2018) supports durative skills with continuous effects, but it has similar limitations. In contrast, our framework offers a true unification of symbolic and continuous control.

On another front, there have been many efforts to combine inference with control. However, nearly all have focused on estimating the probabilities for simple state encodings from noisy or incomplete sensor readings, rather than creating rich relational descriptions. Cognitive architectures like Soar and ICARUS provide explicit mechanisms for elaboration of sensed percepts to form higher-level beliefs, but they do not easily support graded category membership. Choi's (2010) extension to ICARUS incorporated this notion, but his variant did not use the degree of conceptual match as an error signal to produce adaptive behavior. Fuzzy controllers (Katz, 1997; Zadeh, 1968) employ membership functions in much the same way as PUG/C but, again, they encode only low-level state descriptions. Instead, our framework augments symbolic inference of relational descriptions with a veracity score to provide continuous feedback.

A third key idea is that sequential action is guided by differences between the current and desired state. Of course, continuous control methods rely crucially on error signals to calculate values for control attributes, but the notion of reducing differences was also central to operation of the General Problem Solver (Newell et al., 1960), arguably the first symbolic planner. Nilsson's (1994, 2001) teleoreactive framework uses symbolic control rules to bring embodied agents closer to desired ends in continuous environments, but it did not rely on numeric error. ICARUS uses means-ends analysis to direct choices about which actions to carry out, but only at the symbolic level. Our framework is distinctive in that it uses degree of match for symbolic goals (target concepts) to drive continuous control within the context of discrete skills. The target concepts associated with skills also serve as the source of potential fields that are combined to balance the agent's competing objectives.

Still, these comparisons are not entirely justified. Recall that our extensions were motivated largely by limitations of the previous PUG architecture. The earlier framework incorporated durative skills that produced continuous effects, with their application conditioned on conceptual inferences that were grounded in percepts. However, it did not support either feedback control or degrees of conceptual match, much less use the latter to guide the former. Neither could it could execute only one skill at a time, use processes to predict their effects, or search for alternative motion plans. The revised theory and its implementation offer a more flexible replacement for the lower levels of that architecture, not an entirely new account of embodied, goal-directed agency.

## 7. Closing Remarks

In this paper, we presented an extended framework for physical agents that unifies ideas from symbolic problem solving and continuous control. We described the theory's postulates about mental structures and the cognitive mechanisms that operate on them. These assume that inference matches relational concepts to different degrees, skill execution calculates control values from how well their target concepts match, processes support mental simulation of multiple intentions, and heuristic search finds high-utility motion plans. Thus, it enriches ideas from earlier traditions in ways that go beyond any of them in isolation. In addition, we described PUG/C, a cognitive architecture that provides a syntax for encoding concepts, motives, skills, and processes, along with an interpreter that supports conceptual inference, continuous control, mental simulation, and motion planning. We demonstrated these abilities in a simulated robotics environment, including scenarios that required parallel skill application, fluid obstacle avoidance, and generation of motion plans.

The initial evidence suggests that the extended theory has substantial promise, but clearly considerable work remains. We should demonstrate the integration of motion planning with task planning on multi-goal scenarios that require extended operations. We should also interleave planning with monitored execution, as supported by an earlier PUG extension (Langley et al., 2017). Another important elaboration would address the representation and use of 'virtual objects' that are defined in relation to visible landmarks, which PUG/C could then use to encode waypoints and place concepts. Finally, we should test the architecture on richer domains such as urban driving, which can introduce factors like social norms that require reasoning about tradeoffs among competing objectives during planning and execution (Langley, 2019). Taken together, progress on these fronts will produce an even stronger computational theory of embodied intelligent agents.

## References

Bennett, S. (1996). A brief history of automatic control. *IEEE Control Systems Magazine*, *16*, 17–25.

Cashmore, M., Magazzeni, D., & Zehtabi, P. (2020). Planning for hybrid systems via Satisfiability Modulo Theories. *Journal of Artificial Intelligence Research*, *67*, 235–228

Choi, D. (2010). *Coordinated execution and goal management in a reactive cognitive architecture*. Doctoral dissertation, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA.

Choi, D., & Langley, P. (2018). Evolution of the ICARUS cognitive architecture. *Cognitive Systems Research*, *48*, 25–38.

Fox, M., & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, *27*, 235–297.

Goebel, R., Sanfelice, R. G., & Teel, A. R. (2009). Hybrid dynamical systems. *IEEE Control Systems Magazine*, *29*, 28–93.

Katz, E. P. (1997). Extending the teleo-reactive paradigm for robotic agent task control using Zadehan (fuzzy) logic. *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation* (pp. 282–286). Monterey, CA.

Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. *Proceedings of the 1985 IEEE International Conference on Robotics and Automation* (pp. 500–550). St. Louis.

Laird, J. E., Kinkade, K. R., Mohan, S., & Xu, J. Z. (2012). Cognitive robotics using the Soar cognitive architecture. *Proceedings of the AAAI-2012 Cognitive Robotics Workshop*. Toronto.

Langley, P. (2019). Explainable, normative, and justified agency. *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence* (pp. 9775–9779). Honolulu, HI: AAAI Press.

Langley, P., Choi, D., Barley, M., Meadows, B., & Katz, E. P. (2017). Generating, executing, and monitoring plans with goal-based utilities in continuous domains. *Proceedings of the Fifth Annual Conference on Cognitive Systems*. Troy, NY.

Neches, R., Langley, P., & Klahr, D. (1987). Learning, development, and production systems. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*. Cambridge, MA: MIT Press.

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264). UNESCO House, France: UNESCO.

Nilsson, N. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Nilsson, N. (2001). Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, *5*, 99–110.

Rosch, E., & Mervis, C. B. (1975). Family resemblance studies in the internal structure of categories. *Cognitive Psychology*, *7*, 573–605.

Salvucci, D. D. (2006). Modeling driver behavior in a cognitive architecture. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *48*, 362–380.

Zadeh, L. A. (1968). Fuzzy algorithms. *Information and Control*, *12*, 94–102.