Advances in Cognitive Systems X (20XX) 1-6 Submitted X/20XX; published X/20XX

Portals and Spaces: An Egocentric Knowledge Representation for Reasoning About Actions and its Implementation

Christopher Geib CGEIB@SIFT.NET
Jeffrey Rye RYE@SIFT.NET
Vasanth Sarathy VSARATHARY@SIFT.NET
SIFT, 319 1st Ave. North, Suite 400, Minneapolis, MN 55401, USA

Abstract

While there is significant research in the psychology literature that human's use of egocentric representations of space, almost all work in AI plan recognition and planning has assumed access to universal coordinate frames, or allocentric representations of space. Reasoning with such allocentric representations actually makes some kinds of inference more difficult, and seems to conflict with current models of embodied agents informing robotics. This paper presents first steps toward formulating a useable egocentric representation of domains for AI plan recognition and planning based on Portals and Spaces. This representation will allow for the integration of high level AI reasoning with low level continuous control and reasoning systems like those found in modern robotics and virtual environments. We will show how portals and spaces have been added to the ASISTANT system developed on the DARPA ASIST project.

2. Introduction

As long ago as Shakey the robot, AI researchers have been integrating low-level continuous sensor and control systems for robots and virtual agents with high-level reasoners. Each such system is forced to make a number of, frequently undocumented, decisions about what data is shared between the levels and in what form, and where and how low-level control and sensing ends and high-level planning and reasoning takes over. In the pursuit of working systems such interfaces have usually been ad-hoc and driven by the exigencies of the specific systems being integrated. Thus, the answers for the "how and why" of system building result from an number of engineering considerations including: the maturity or features of the sensors or control systems or high level planers and reasoners, the specific domain of application, desired limitations of specific integrated components, desired infrastructure architecture and even the chosen software and integration toolsets. Design decisions made for any of these reasons can only tell us anything about how such systems should be built if we do post analysis of what worked well in the system and what didn't (a level of system analysis rarely performed). As such, while they may be effective engineering solutions, they tell us nothing about the science behind repeatedly engineering such systems. They tell us nothing formal, principled, or well grounded about how interfaces between such systems should be built.

While some work has attempted to claim that either the high or low level is not actually necessary, we will not address this reductionist position. Instead, in this paper we propose to begin the process of formalizing and building a theory of how differing levels of reasoning and control should interact. We would argue any solid theory formalizing the interfaces between high and low level reasoning systems should at minimum enable:

- abstraction and encapsulation of reasoning, enabling the levels to use specialized representations tailored to their specific inferential tasks,
- bi-directional transfer of information between high and low level reasoners necessary for their functioning, and
- grounding of high-level reasoning and concepts in low-level, continuous control signals (ie. implement the embodied cognition hypothesis).

This fundamentally requires producing a set of identifiers for actions, objects, and state conditions, that can be used by and passed between the high and low levels of reasoning. In this work we will argue that such identifiers must be *egocentric*.

There is significant evidence from psychology Piaget & Inhelder (1956) and biology Menzel et al. (2005); Wehner R (1996) that people and animals often use *egocentric* knowledge representations. That is, they represent their knowledge of the world relative to themselves and their perception of it rather than using *allocentric* representations based on external universal coordinate frames. For example, capturing the location of a robot's gripper as a distance relative to the object to be grasped and a pose within the robots joint space would be an egocentric representation, while using an x,y,z coordinate for the tip of the end effector relative to the enclosing space would be an allocentric.

Allocentric representations are common in both low-level sensor and control research and high-level reasoning research. They are attractive precisely because they are external to the systems, are often available in engineered or virtual environments, and can reduce ambiguity between multiple systems working in the same domain. However, for real systems deployed in the real world such representations are often not available. Given this, one of the core questions that inspired this research is "Should we assume access to such systems?" Self reflection tells us they are not required to solve the larger problem, however the existence of another approach is not a sufficient reason to forgo working alocentric approaches. The deeper question is does reliance on alocentric reference frames create avoidable problems for system building or API development? In the following, we argue that it does. For example, consider using GPS coordinates with a half meter resolution to plan a trans-continental trip for a robot. If all reasoning has to be done at the level of a half a meter such a plan will be very long and potentially unnecessarily detailed.

This paper will formalize the ideas of *Portals and Spaces* to formalize the representation of problem domains and use them to capture actions, objects, and states for use and integration between both high-level reasoners and low-level, continuous sensing and control systems. As an example, this paper will then discuss representing physical and conceptual spaces within a Minecraft domain on the DARPA ASIST project. We will then discuss the relationship of this approach to prior work including that on Object Action Complexes (OACs) and the idea of affordances from psychology.

3. Background

The issues we will discuss here are common to both systems that generate plans and actions for an agent as well as those that recognize the actions and plans of other agents. As such, most of our discussion will be agnostic to the kind of reasoning that is being performed by the system. With this in mind, for brevity, and to avoid committing to specific technologies that implement such systems, we introduce two pieces of terminology. First, we will use the term *Low Level Reasoner* (LLR) to denote the low-level sensor or controller of some system that is directly responsible for controlling the actuators and sensors of a virtual or embodied system. Such systems frequently must address issues of continuous control and continuous sensor inputs. Second, we will use *High Level Reasoner* (HLR) to denote component of the system that is responsible for high-level inference for the same system. Such systems have often been the purview of artificial intelligence reasoners based on propositional logics, first order logics and other discrete reasoning methods.

We believe our claims about LLR and HLR systems are general enough that they apply to all or almost all such systems. We are not ruling out HLRs that are implemented probabilistically or by neurally inspired methods. Neither are we ruling out LLRs implemented using first order or other logical frameworks. Instead, our objective in this paper is to provide a well formalized and founded method for grounding the inference of HLRs and the outputs of LLRs such that they can easily be connected and we can thereby leverage the strengths of each of them to perform more general and flexible reasoning about actions. We will to accomplish this in a modular manner that avoids the ad-hoc design choices of prior work.

3.1 What About "Robot Operating Systems"?

This work is NOT addressing the same issues as a robot control architecture like ROSStanford Artificial Intelligence Laboratory et al. (2018) or other "robot programming" APIs. Such systems are frequently little more than scripting languages for invoking behaviors in sequence. These systems often simply extend the robot engineering project upward using the same representations used at the lowest levels. They do not actually abstract reasoning in continuous spaces to those of discrete spaces. Instead they allow for the scripting of frequently occurring low level control sequences, and as such, they do not address the central question of this paper.

3.2 What About Neural Network Architectures?

There are works using neural networks and other distributed, sub-symbolic representations to control robots and other systems that might seem to address or obviate our claims about the LLR and HLR integration question. However, such an approach effectively begs the entire question. It is well known that various parts of the human nervous system and even the brain have specialized functions. As such, even if all of the components were implemented using neural or sub-symbolic representations and reasoning processes, the functional units would still require integration, leading us back to the central questions of this paper. Alternatively, work that treats the whole low level sensor to high level cognition and back to actuator loop as a single network computing a single function seems to raise more questions than it addresses. Why would such a system seem to provide access to multiple other levels of abstraction if they are not part of the actual computation? A complete refutation of such an approach is outside the scope of this paper.

3.3 Assumptions

We make a number of assumptions about the LLR and the HLR to help define the interface problem.

Assumption 3.1 *The LLR has a set of* ***Motor Programs*** *(MPs) that are able to drive the system's sensors, and end effectors. Invocations of these MPs are the implementation of any physical or sensing actions a resulting system might be capable of performing in the world .*

We think of MPs as control programs that change the position and orientation of the system or its available end effectors. That is, an MP only specifies how to move the "body" of an agent from one configuration to another. Their invocation is not limited to a particular setting or situation. Nor are they required to have a particular effect on the state of the world without further qualification. For example the same motor program might be used to turn a key to unlock a door, start a car, or even launch a rocket. However, which effect eventuates depends completely on the context of its execution and the objects it acts on. This means that while the invocation of MP are the core of any interface between the HLR and the LLR, and we must have a way for both the HLR and LLR to refer to them, they alone they are incomplete. They require parameterization, and any parameters and further input arguments must be part of our interface.

Note, this paper will not be concerned with how MPs are leaned or constructed; as this is itself a large and independent area of research in robotics and control theory. We also note that for any reasonable system our interface should provide a method for adding new MPs to both the LLR and the HLR thereby extending the interface. However, this work will not discuss this, and instead work only with fixed sets of MPs.

Assumption 3.2 *MPs have the potential to move the agent or an end effector to a large number of possible positions, defined relative to a specific object or object type. Thus we will discuss interface level invocations of an MP's as having two parameters, first an object instance or object type parameter and second a set of possible final successful positions within the agents internal joint and sensor space.*

For example, in our key turning example, the motor program would be parameterized by first, the class of objects that are keys, and second, the state as having turned the key, maintained contact with it and keeping the lock and key close to the agent's body. Note that simply executing the control program on the object might result in a number of other states (e.g. loosing touch with the key) that in some other condition might be acceptable. This also means that if either the agent or the objects are able to move within the environment, that the MPs must be defined relative to the agent, the object, and possibly other parts of the world model, NOT a universal coordinate system. [1]

1. We can imagine "performative" MPs that are not defined relative to an external object or object type. However, these are rare since the relation to the object can be oblique. If they exist, such MPs are not within the scope of this paper.

Assumption 3.3 *The LLR has a unique identifier for each MP.Further, given such an identifier (and its associated arguments) a LLR is able to perform any necessary reasoning and to invoke the system's control system to execute that MP.*

We can think of UIDs as the names for the control programs the LLR can run. As such they are an integral part of the interface between the LLR and HLR. We expect the HLR will send a UID to cause the LLR execution of a specific MP. As a result, any high level plans the HLR builds must be expressed as a sequence of MPs with associated arguments.

4. Definitions

Under these assumptions we can define an interface based on *portals and spaces*.

Def: 1 *We will denote by $\mathcal{S}_{physical}$ the complete space of possible configurations the system can be placed in by the LLR. With the following properties:*

- *it uses an egocentric representation in its space,*
- *it can use either object types or system specific instance identifiers in capturing knowledge about and relations to objects, and*
- *it captures all possible static and dynamic configurations that LLR can achieve,*

We can think of $\mathcal{S}_{physical}$ as capturing the sensory motor manifold of the system controlled by the LLR. Since $\mathcal{S}_{physical}$ uses an egocentric representation it captures knowledge relative to its internal sensors and feedback from its external sensors. Further we will assume that through the use of typed arguments or internal unique identifiers, $\mathcal{S}_{physical}$ goes beyond just the internal proprioceptive state of the system to encode continuous, egocentric information about the system's position relative to any objects necessary for motor programs. (e.g. Is the system holding the key in the correct orientation?) Note it is also intended to represent dynamic relations between the system and the external world. (e.g. Is the key positioned for insertion into the lock?)

Def: 2 *We will denote by $\mathcal{S}_{mental}$ the discrete mental world model of the system used by the HLR that can be senses by the LLR.*

We can think of $\mathcal{S}_{mental}$ as a subset of the elements of a first order model used by a planning or high-level plan recognition system to reason about the effects and efficacy of actions. We specifically limit it to include only those predicates, types, and objects the LLR can sense. We do this to guarantee that it can be used for information exchange between the levels. It is more than possible, and is in fact expected, that the HLR's model of the world might contain entities that are not directly sensible by the LLR and therefore would not be included.

Def: 3 *We define $\mathcal{S} = \mathcal{S}_{physical} \cup \mathcal{S}_{mental}$*

Note, a point or single "state" in $\mathcal{S}$ can include both continuous conditions within the agent's internal control space as well as discrete properties normally associated within higher level models. In the abstract, we can think of the LLR working in $\mathcal{S}$, and the HLR as working with a model that is a superset of $\mathcal{S}_{mental}$. To enable more detailed discussion of MP functions,

Def: 4 *We define a* ***condition****, denoted as s_i, as a set of points within the space of all possible states, $s_i \subseteq \mathcal{S}$ and an individual* ***state****, as a single point within the space, denoted $\mathbf{s}_i \in s_i$ or $\mathbf{s}_i \in \mathcal{S}$.*

With these definitions in hand, we can think of each MP as a function (parametrized by an object and a condition capturing the desired destination states) over $\mathcal{S}$.

$$\mathbf{mp}_i[o_j][s_{dest}] \rightarrow \mathcal{S} \times \mathcal{S}$$

where $\mathbf{mp}_i$ is the motor program identifier, o_j can be either a specific object instance or a typed object variable that will be bound to an instance for execution, and $s_{dest} \subseteq \mathcal{S}$ set of acceptable final states. We will refer to a motor program with its parameters as a motor program execution specification or just an **execution specification** for brevity. Note that given the physical limits of such systems, any given execution specification may not be defined for every state in $\mathcal{S}$. Therefore, we will denote the domain of such a function as $s_{dom[i,j,dest]}$ and the range as $s_{rng[i,j,dest]}$ or as just s_{dom} and s_{rng} when the details are not necessary. Therefore:

$$\mathbf{mp}_i[o_j][s_{dest}] \rightarrow s_{dom[i,j,dest]} \times s_{rng[i,j,dest]},$$

and we denote invoking an execution specification from a state, $\mathbf{s}_0$ resulting in the state $\mathbf{s}_1$ as:

$$\mathbf{mp}_i[o_j][s_{dest}](\mathbf{s}_0) = \mathbf{s}_1$$

where $\mathbf{s}_0 \in s_{dom[i,j,dest]} \subseteq \mathcal{S}$ and $\mathbf{s}_1 \in s_{rng[i,j,dest]} \subseteq \mathcal{S}$.

Remember, our objective has been to specify principled interfaces between HLRs and LLRs. We claimed, that HLR invoking LLR control programs to move in and sense the environment is core to this, and further that egocentric representations of the domain are critical to this. We have defined execution specifications using such representations. However, to use them, the HLR and the LLR must agree on identifiers for: $\mathbf{mp}_i$, o_j, s_{dest}, and $\mathbf{s}_0$. We argue that s_{dest} is the most challenging.

If the system wants to execute the action in the current state of the world then $\mathbf{s}_0$ does not actually need a shared specification between the two levels because it is the current state of the world as represented by each reasoner. Further, in the case of a sequence of invocations for the HLR we can imagine a recursive specification of the state for execution, that is the state that eventuates from the execution of the previous action in the chain. Thus, we will treat $\mathbf{s}_(0)$ as a not a significant problem. We have assumed LLRs have unique identifiers for their $\mathbf{mp}_i$s that can be shared. Further, since objects are central constructs in any LLR, we would not be surprised if they have unique identifiers that could be shared with and used by the HLR . However, specifying an arbitrary collection of points that make up $s_{dest} \subseteq \mathcal{S}$ is more problematic.

4.1 Portals

One natural way to specify such a condition would be to use propositional or first order logic to enumerate the properties shared by the states within the condition. However this presents a problem for building LLR to HLR interfaces. Using a logic within the space of the HLR to express this condition would require a logic able to express all of the information in $\mathcal{S}_{mental}$. Which, up until this point, has only been captured within the LLR. Representing it within the HLR would defeat the point of having two different reasoners with abstraction and data-encapsulation between them. If all the same knowledge is represented at both levels, are sure that we need separate reasoners?

Being unable to use a logic to specify these conditions, suggests that, like the other elements of the execution specification used by the HLR, these conditions should be well known within the domain specification. That is, rather than being arbitrary conditions they should be relevant for the LLR's processing. We can then imagine the LLR possesses a unique identifiers for them that could be shared with the HLR without sharing the details of their representation or semantics for the LLR.

One preexisting set of conditions that have this property are that of the s_{dom} of well known execution specifications. This has the desirable property of tying the successful performance of one execution specification as ending in a state that makes some other execution specification possible. We argue that this is a productive approach and will use it to define both portals and spaces.

Def: 5 *Given a problem domain with set of execution specifications,* $\mathbf{mp}_i[o_j][s_{dest}] \rightarrow s_{dom[i,j,dest]} \times s_{rng[i,j,dest]}$ *we define the set of* ***portals****,* $\mathcal{P}$*, as the set of function domains for the execution specifications, and denote each of them* $P_{[i,j,dest]} = s_{dom[i,j,dest]}$.

In the case where the details of a portal's execution specification are not critical we will denote a portal as P_i. Thus, portals are the starting conditions for known successful execution specifications captured in the agent's own egocentric representation of the world. Because these conditions are well known to the LLR we can reasonably assume the system has a unique identifier for each portal that can be shared between the LLR and HLR.

The intuition behind this definition is that a motor program just calls for the driving of the effector in particular ways, at particular speeds, and the like. As such, it could be invoked from any point in $\mathcal{S}$. However, it will only have specific outcomes if it is executed in particular locations and with the correct relation to a specific objects or objects of a particular type. For example, imagine while typing an email you were to raise you hands six inches above the keyboard. While you could move your fingers in exactly the same manner, your actions would not result in an email.

Portals then capture the starting conditions for common, repeatable patterns of use of the motor program in the agent's own LLR's representation. For example while the same motor program might be used to turn a key to unlock a door, start a car, or even launch a rocket, their respective objects, and s_{dest}s would be very different and therefore they would have very different portals. It is also worth noting that because execution specifications are defined by conditions some portals are not required to specify the values of some elements of $\mathcal{S}$. For example, it may not be necessary to know anything about the left hand while turning a key with the right. As such the execution specification for this action would not restrict the left hands position or orientation.

Note that portals are similar to STRIPS Fikes & Nilsson (1971) style precondition for operators in planning systems. The s_{dom} roughly capturing the preconditions of the motor program having the effects described in the s_{rng}. However, they differ from preconditions in four critical ways. First, portals capture semantically relevant points within the LLR's egocentric space that can still be used by the HLR. That is they are required to be egocentric which STRIPS preconditions are not required to be. Thus the spaces they are defined over as completely different.

Second, unlike traditional preconditions, portals can't be arbitrarily chosen by the system designer. They are defined within the LLR's representational space, relative to a specific set of success criteria for the control program. As such, portals have a stricter definition and play an explicit role in the interface to HLRs. There are no such restrictions or requirements on STRIPS preconditions. This has lead to a great deal of work in AI planning defining different kinds of preconditions that have different types of effects or implications for the results of the systems.

Third, portals allow the LLR to explicitly capture knowledge that is specific to an individual object or relative to types of objects. Thus it allows an embodied system to represent detailed knowledge about a specific object. For example, the door to my house sticks and requires a slightly different MP to open it than other doors. Such knowledge is difficult to encode in traditional planning systems without equality conditions and constants.

Fourth, and finally, portals force our representations and the interface of HLR and LLRs to be built on principled foundations rather than arbitrary engineering choices that make the work of the HLR easy or control its search more effectively. One frequent engineering choice in building systems is to add preconditions to actions that either prevent the system from engaging in some search or resulting the binding of parameters to particular values.

4.2 Spaces

We will organize portals into collections we define as *spaces*.

Def: 6 *Given a condition* $\boldsymbol{s}_0 \subseteq \mathcal{S}$*, we define a* ***space****,* S_{s_0}*, as the set of all portals* P_0*, such that, there exists an execution specification,*

$$\mathbf{mp}_i[o_j][s_{dest}] \rightarrow s_{dom[i,j,dest]} \times s_{rng[i,j,dest]}$$

with $\boldsymbol{s}_0 \subseteq s_{dom[i,j,dest]}$ *and* $P_0 \subseteq s_{rng[i,j,dest]}$*.*

Intuitively then, a space, S_{s_0}, is just the set of portals reachable from a given set of states by and agent using a single execution specification. Choosing a portal from our domain to define the first space, and then reifying over the set of all portals and execution specifications, this definition provides an accessibility relation defining the set of all reachable portals and the spaces that contain them. As such, it defines locations and collections of locations that define an egocentric representation of the domain and are:

1. usable by both the HLR and the LLR ,
2. *definitionally* necessary for the successful execution of actions by the LLR, and therefore
3. semantically significant for both the LLR the HLR.

With these definitions for portals and spaces we have completed the definitions that we need for our interface. We have discussed how to create egocentric represenations for $\mathbf{mp}_i$, o_j, s_{dest}, and $\mathbf{s}_0$ to be shared between a LLR and a HLR. In the next section we will discuss an example implementation using these definitions within the ASISTANT system as part of the DARPA ASIST program.

5. Application in a Virtual Environment

As part of the DARPA ASIST project, we have developed the ASISTANT system that uses a portals and spaces representation for its HLR domain specification and the interface between its LLR and HLR. The central task for the ASISTANT system is to provide advice to a team of humans performing a simulated search and rescue mission within Minecraft. A team of players must navigate a Minecraft environment and triage victims they find. The ASISTANT system must recognize the player's movement and victim saving plans and provide advice on how the team can perform better. A stream of events characterizing the player's actions from the testbed provide the system's input. Note however, that Minecraft testbed and hence the event stream uses an alocentric representation. This is most noticeable in its description of a player's movement within the domain. Therefore, the system must first translate the observed event stream into an egocentric representation as defined above. In the ASISTANT system we have made the LLR responsible for this translation resulting in a portals and spaces based model of the domain.

While our definition of portals and spaces can capture even purely mental spaces, for this application, physical location is the main determiner of what an agent is able to do. In the case of actions like picking up an object or putting it down, the observation of a successful execution of the action (by definition) means the player must have been at the correct portal otherwise they couldn't have succeeded at this action. Therefore when such events are reported by the Minecraft simulator, the LLR(can infer the portal the player was at, and it can be added to the domain, and players current space. This makes the addition of portals for manipulation tasks relatively straight forward.

In contrast, converting the players location and movement in the domain from allocentric to egocentric is much harder. The point of using portals and spaces is to abstract the lowest level space away from raw sensor reports. Therefore we don't want to define every physical movement taken by the player as being from one portal to another. For navigation, especially in large spaces, defining portals spaces is much less clear. The rest of this section will focus on how our system translates alocentric **position-update** events from Minecraft into a reduced sequence of egocentric **locToA**, **transitPortalA**, and **transitDoorA** events. The **locToA** events that indicate the player has moved from one portal to another within a space, while the **transitPortalA** and **transitDoorA** events indicate the player has moved between spaces.

5.1 "Real" Spaces and Adding Events to the Observation Stream

Mirroring the definition of spaces, we assume that for an agent to navigate between portals within a single space the HLR does not need to be invoked. That is, given an execution specification, the LLR can be invoked to carry the agent between any pair of portals in the same space. For portals that are in different spaces the HLR must be invoked and a sequence of execution specifications generated (or recognized) to carry a player between them. This means, that the events produced by the LLR navigate between portals within a space, and from the perspective of the HLR, the player can always be thought of as being at some portal. We have made the LLR responsible for determining the portal the HLR believes the player is at, and all the events produced by the LLR as input for the HLR begin and end at a portal (possibly the same one).

This helps to more clearly define the roles and remits of LLRs and HLRs, however it also ties portals and spaces much more closely to reasoning about acting in the world than they are to the physically contiguous locations that we might naturally think of as spaces (e.g. rooms, hallways, buildings, etc...). For example, imagine a large oblong room with a fireplace and chairs at one end and a couch and TV at the other. Because moving between them requires effort (invoking the HLR) it is completely possible that this single room would be represented as at least two distinct spaces that have to be navigated between: one for conversation and one for watching TV.

To implement this idea our system pre-processes the ASIST Minecraft event stream treating any significant change in the direction traveled by the player as potentially marking a move by the player from one space to another. For each of these, the system may add a **change-direction** event to the existing event stream that can cause the LLR to recognize this spot as a portal between spaces.

The ASIST Minecraft testbed event stream contains **position-update** events to record changes in the player's location. To detect changes in direction, our LLR maintains a two second, sliding window of these events for each player. After each update of the sliding window, it performs a t-test to determine if the mean direction of travel differs between the first and second halves of the window by more than plus or minus twenty two point five degrees (forty five degree total window).

If the test is significant (p-value = < 0.01) the system inserts a **change-direction** event in the middle of the window.[2] Note that this relative difference in direction traveled is an egocentric metric and could have been provided by the testbed directly or recovered from an internal sensor on an embodied system. The addition of these events produces an event stream the ASIST_{ANT} LLR can translate into a portals and spaces based representation of the domain.

5.2 Translating the Event Stream to Portals and Spaces

After pre-processeing, the observed event stream is made up of **position-update** events, **change-direction** events, and events that record actions performed by a single player on a domain object (e.g., **equip-item**, **triage-started**, and others.). Translation of this stream of events has two major components: first removing the **position-update** and **change-direction** events and replacing them with a smaller number of **locToA** and **transitPortalA** and **transitDoorA** events. Second, the system needs to add the required portals and compute the spaces that will be used in the domain. As we have already argued, converting events that capture interactions with domain objects into our portals and spaces representation it is relatively straightforward bookkeeping. New objects and the required portals for the events can be added to the domain for each observation of a successful use of an object. However, to convert the player's **position-update** and **change-direction** events into portal based locomotion though spaces in the world requires constructing the spaces as the input stream is processed. This means the system needs to:

2. The length of the window, angular change, and p value for the t-test were all determined by empirical studies and are configurable parameters to our implementation.

1. Maintain each player's Current Location Portal (CLP).
2. Add any new objects and portals to the domain,
3. Create spaces and assign objects and portals to them, and
4. Assign **position-update** and **change-direction** events to spaces. For all other events, the referenced object determines its space.
5. Convert all **position-update** and **change-direction** events into a reduced set of **locToA**, **transitPortalA**, and **transitDoorA** events.

To do this translation, the system splits event stream into chunks (10-15 seconds in length) containing the original and added events. For each observed event in the chunk the LLR does the following:

1. Add any new objects and required portals to the model.
2. Add the event and its objects and portals to the *state assignment qeue*.
3. If the event is a **change-direction** event, call the UPDATE function (described below).
4. If the event is a **position-update** event:
    (a) Update the player's CLP:
        i. player is within 1.0m of a door, CLP = door's portal,
        ii. player is within 3.0m of a **change-direction** event, CLP = the event's portal.
        iii. otherwise, CLP = none.
    (b) If the player's CLP is changed, add a new **transitPortalA** or **transitDoorA** event capturing the player's entering a new space.
5. If the number of events in the *state assignment queue* exceeds a threshold, call the UPDATE function. This prevents the production of multiple spaces around closely grouped portals.

After a chunk is processed, UPDATE is called to assign spaces for any remaining events, objects, or portals. The UPDATE function assign each of the elements in the *state assignment queue* to a state:

1. Assigns **position-update** events to the closest space. If this changes the actor's current space, the LLR adds an appropriate **transitPortalA** or **transitDoorA** event to the event stream.
2. Adds any events, portals and objects to the current space.
3. Resets the data structures.

5.3 Space Construction Details

ASISTANT's LLR computes the domain model's spaces:

1. Compute the bounds for the pending moves.
2. Add all known portals within bounds of the pending moves to the pending moves list.
3. Compute the bounds with the portal locations included.

4. Find the existing space with the greatest overlap with these bounds.
5. If the the overlap is greater than 60%, leave the current space unchanged.
6. Otherwise:
    (a) If the reason for the last update was the end of a chunk, leave the current space unchanged (just keep extending the old space).
    (b) Otherwise initialize a new empty space and make this the current one for the actor.
7. Add the new points to the actor's current space.
8. Recompute the bounds for the space.
9. Ensure that the pending portals are in the space.
10. If any portals were added to the space, recompute the bounds to include the portal locations.

Throughout this process the physical bounds of a proposed space are computed by finding an alpha shape containing the hypothesized portals.

The whole process runs in realtime, allowing the LLR to build the spaces and portals used by the HLR during the 17 minute trials. However, since the number of spaces in the domain increases with time, its runtime does increases over time. We believe that future work can address this problem by:

- Periodically resample the points in a space to limit the time required to compute its bounds.
- Use quad trees or other space partition methods to reduce the spaces considered for overlaps.
- Find methods for efficient merging or repartition spaces to reduce their number.

Note that currently portals and spaces are incrementally added to the domain, but not removed or merged. Enabling this kind of model revision is an area of active future work. It is also worth noting that while this algorithm makes use of the allocentric space representation provided by Minecraft we do not believe it is necessary for the computation.

In the end, the process produces a domain model based on portals and spaces. This includes identifying which portals are in which spaces as identified by the player's navigation and locomotion events. ASISTANT's LLR does this all online and requires no prior training. It is also possible to initialize the portals and spaces domain model, from a given map when it is available a priori. Finally, to date, the ASISTANT LLR computes a single set of spaces and portals for all actors, though it reasons about each actor's movement independently. That said, the approach can be used with no change when computing separate spaces and portals for each actor.

5.4 Example Use in the HLR

A simple example may help to clarify this approach and give us an example to discuss. The following sequence of observations was produced by the ASISTANT LLR for player id **p000185** successfully triaging **victim1231**. It has been lightly edited for clarity and brevity, but shows the major points of how the portals and spaces representation is used in our application.

```
1  locToA (p000185, functionDoor39),
2  openDoorA (p000185, functionDoor39, transitDoor39),
3  playerSawA (p000185, transitDoor39),
4  locToA (p000185, transitDoor39),
5  transitDoorA (p000185, space497, transitDoor39, space500),
6  playerSawA (p000185, transitDoor39),
7  playerSawA (p000185, functionVictim121),
8  playerSawA (p000185, transitDirchange532),
9  locToA (p000185, functionVictim121),
10 triageStartedA (p000185, victim121, functionVictim121),
11 triageSucceededA (p000185, victim121, functionVictim121),
```

The events on lines 1 to 5 capture the player entering the room the victim is in. It does this by:

- line 1: Moving to the portal that enables opening door39 (**functionDoor39**),
- line 2: Opening the door,
- line 4: Repositioning to the portal that enables going though door39 (**transitDoor39**),
- line 5: Transiting the door.

The **playerSawA** action on line 3 captures the player's new percept of the open state of Door39. Likewise the **playerSawA** actions on lines 6-8 capture the players new percepts transiting from **space497** to **space500** where the victim is located. Notice specifically, that on line 8 the player observes portal **transitDirchange532**. Portals with names of this form are the result of the **change-direction** events we inserted into the event stream (See Section 5.1) and can be transited without opening them (unlike doors). Lines 9 - 11 capture the players moving to the victim and triaging them. Using this portals and spaces representation, the HLR in the ASIST$_{\text{ANT}}$ system (built on the ELEXIR reasoning system Geib (2016); Geib & Goldman (2011)) is able to recognize this sequence of events as a successful triaging of a victim in this domain. Further our system as a whole, using this approach to incrementally build and use a portals and spaces domain model is able to successfully recognize the multistep individual and team plans of three player teams over twenty seven, seventeen minute trials without falling behind the players. We view this as an unqualified success for this approach to domain construction and interface building. With this success in mind, the next section will discuss some of the implications of this approach.

6. Discussion

First, it is worth noting that these successes on the ASIST program suggest that not only is a portals and spaces based interface theoretically well founded, but it appears to be computationally practical as well. Our implementation not only achieves our integration objectives but it also enables the whole system to scale to real world sized problems. While the ELEXIR plan recognition technology behind our implementation of the HLR is known be able to process and recognize plans given thousands of observations, it was far from clear that we would be able to use a LLR to construct the domain from the input stream and make the appropriate calls to the HLR and get results back in real time. It is the abstraction of large sequences of **position-update** and **change-direction** events into a small number of **locToA**, **transitPortalA**, and **transitDoorA** events that allows the HLR to work at scale. The full Minecraft event stream for three players produces tens of thousands of observed events over the course of each of the seventeen minute trials. Our LLR implementation builds a domain model and abstracts tens of thousands of **position-update** and **change-direction** events down to only thousands of **locToA**, **transitPortalA**, and **transitDoorA** events for the HLR.

To achieve this, it was critical that the process and the resulting domain model removed any need for the HLR to do numerical computations (as opposed to logical computations). As we have already argued, portals act as names or placeholders for specific locations. Using a portals and spaces domain in the LLR to HLR interface means the HLR no long needs to be able to perform any computations regarding the planning of paths or other control using numeric values. The HLR is not even responsible for maintaining the players current <X,Y,Z> values of an allocentric representation. This information is hidden within the abstraction of each players current portal as maintained by the LLR. This is an important change for HLRs. Like many other technologies that have been used to implement HLRs, ELEXIR doe not natively support any arithmetic operations. Its reasoning is logic based rather than numeric. For example, it is not possible for it to recognize multiple incremental changes in say the x-coordinate of a player as approaching a victim. Instead, it is tailored to recognize complex, multi-step, hierarchical plan structures common to human activity. Thus it is appropriate for reasoning about such trajectories to be captured using other specilaized algorithms in the LLR and abstracted away from use or even representation in the HLR

The reduction in the size of the input also made it possible for the HLR to run in real time. While the ELEXIR system does scale quite well to large input sets, without this reduction we believe the system would have been unable to keep up with the actions of the players. Thus the move to portals and spaces based interface between the LLR and the HLR not only provided an appropriate representational abstraction between the reasoners but the abstraction itself also dramatically reduced the input size while still allowing the systems to function at scale.

As such, our work on the ASISTANT system and its ability to recognize the simultaneous activities of multiple players in real time based on thousands of input observations provides solid evidence that a portals and spaces representation is at least as viable as any prior ad hoc method for designing domains and interfaces for such systems. The obvious question is does it meet the criteria we laid out in the beginning of this paper for a formal theory for such interfaces? Does it provide something more than ad hoc design methods?

Our use of portals and spaces to ground the design of the ASISTANT system provides solid evidence that it meets all three of our inital criteria. The HLR and LLR use qualitatively different reasoning methods. Most of the LLR's computations are based on numerical, windowed, stream-based aggregation methods. The HLR is first-order, formal grammar based system that uses weighted model counting to compute probabilities. Their representations and algorithms could hardly be more different. While our implementation has not yet demonstrated the bi-directionality of this representation, the flow of information from the LLR to the HLR has been demonstrated and the ELEXIR system on while the HLR is built has demonstrated both the ability to recognize and build plans *using a single domain for both tasks*. Thus we have good reason to believe our future work will demonstrate this second feature. And finally we believe that in addition to the grounding of the HLR's conclusions in the outputs of the LLR, the abstraction of the locomotion and navigation primitives of the Minecraft simulation system clearly demonstrate the ability of this approach to allow high level reasoning about abstract actions (like entering a room or triaging a victim) to be grounded in low-level signals.

The ASISTANT system supports the addition of portals and spaces constructed as described in the previous section to its domain model. This allows it to have an incrementally built model of the physical space and objects that it will reason about. It also provides a level of abstraction and flexibility not found in prior work. Usually an HLR can only reference objects that can be converted by the LLR to HLR interface to specific 3D coordinate points. This is necessary so that when an execution specification is chosen by the HLR that it can be instantiated by the LLR. This requires the complete set of possible objects and how they can be interacted with be designed for BEFORE the system is able to interact in any way. This is brittle and requires the system designer to pick these points out and encode them for both the LLR and the HLR at system design time. Effectively the system designer must not only design the interactions between the components of the system but also design and enumerate (at least implicitly) ALL of the specific instances of inference and reasoning that the system should be capable of down to the lowest level of the alocentric representation used by the LLR.

While we know of no prior work that has attempted to formalize the interface of LLRs and HLRs using an egocentric representations of space and actions, there are still relations between this work and other work in robotics and psychology. For example, the idea of a portal shares a great deal with the psychological concept of an affordanceGibson (1979). They also seem very related to the idea of Action Object Complexes (OACs) Geib et al. (2006); Krüger et al. (2011). Much like an affordance or an OAC, a portal associates an action with an object to produces a specific result. However, since a portals is defined in terms of a specific egocentric relationship between the agent and the object it goes beyond either of these concepts in the formal requirements for its implementation. Further, prior work on affordances and OACs have not explicitly linked them to communication between different levels of reasoning within systems. As a result, while it shares many of the same intuitions, we believe the portals and spaces representation is very different and goes well beyond either affordances or OACs in attempting to place concrete requirements on the interfaces between LLRs and HLRs and help to build and formalize large multi-component AI systems.

7. Conclusions

This paper has argued that the way in which most current large scale AI systems are built relegates them to one-off engineering exercises that all but prevents the identification of more general principles and knowledge about how such systems should be built. It has argued that to ease the engineering of these systems, such systems makes a fundamental mistake in using allocentric domain representations. It has then argued for an egocentric method of domain representation in the form of portals and spaces. It has both formalized this representation and discussed how it is used in one such system, ASISTANT. This paper should not be seen as claiming portals and spaces are the only such egocentric representation or that we cannot learn lessons about the general construction of AI systems without its use. Instead we would claim that use of such representations focus research on questions that have not previously be answered about how to build and integrate AI systems and may open the door to much greater insights about principled large scale AI system construction.

References

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Geib, C., ao, K. M., Petrick, R., Pugeault, N., Steedman, M., Krueger, N., & Wörgötter, F. (2006). Object action complexes as an interface for planning and robot control. *Proceedings of the HUMANOIDS-06 Workshop Toward Cognitive Humanoid Robots*.

Geib, C., & Goldman, R. (2011). Recognizing plans with loops represented in a lexicalized grammar. *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)* (pp. 958–963).

Geib, C. W. (2016). Lexicalized reasoning about actions. *Advances in Cognitive Systems*, *Volume 4*, 187–206.

Gibson, J. J. (1979). The theory of affordances. In *The ecological approach to visual perception*. Boston: Houghton MIffline.

Krüger, N., et al. (2011). Object–action complexes: Grounded abstractions of sensory–motor processes. *Robotics and Autonomous Systems*, *59*, 740–757. From `https://www.sciencedirect.com/science/article/pii/S0921889011000935`.

Menzel, R., et al. (2005). Honey bees navigate according to a map-like spatial memory. *Proceedings of the National Academy of Sciences*, *102*, 3040–3045. From `https://www.pnas.org/doi/abs/10.1073/pnas.0408550102`.

Piaget, J., & Inhelder, B. (1956). *The child's conception of space*. Routledge.

Stanford Artificial Intelligence Laboratory et al. (2018). Robotic operating system. From `https://www.ros.org`.

Wehner R, Michel B, A. P. (1996). Visual navigation in insects: coupling of egocentric and geocentric information. *Jouranl of Experimental Biology*.