Advances in Cognitive Systems 7 (2019) 1-16 Submitted 5/2019; published 8/2019

Using Domain Knowledge to Correct Anchoring Errors in a Cognitive Architecture

Aaron Mininger MININGER@UMICH.EDU
John E. Laird LAIRD@UMICH.EDU
Computing Science and Engineering, University of Michigan, Ann Arbor, MI 48109 USA

Abstract

A cognitive robotic agent needs to maintain associations between perceptual data and symbolic object representations over time, a process called *anchoring*. Often anchoring is done externally to the cognitive component, but this leads to two potential limitations: the agent may possess knowledge that could be useful for anchoring but is inaccessible; and the agent is less able to handle anchoring errors that do occur. We develop a taxonomy of anchoring errors, then show how an agent in a symbolic cognitive architecture endowed with a continuous spatial memory can take an active role in anchoring by deliberately reasoning over perceptual updates and unifying them with an internal self-maintained set of anchors. During this unification, the agent uses information from disparate sources including action and task knowledge, environmental regularities, and human interaction. Through experiments in a real-world robotic domain, we show how the agent is robust to perceptual noise and errors while tracking objects as it interacts with them, and how more available knowledge leads to better performance.

1. Introduction

In order for a robotic agent to support cognitive capabilities, such as high-level reasoning and planning, it must construct and maintain an accurate and useful representation of the world. In robotic agents with a symbolic reasoning component, this typically involves an object-based representation with object positions, attributes (color, shape, etc.), and other semantic information. Constructing and maintaining this representation is challenging, as it needs to be *accurate* – the knowledge about the world needs to be correct, *robust* – the knowledge about the world is free of noise and errors and stable over time, and *comprehensive* – it contains all the information that the agent needs to accomplish its tasks. This is difficult because real-world environments are complex and dynamic, and sensory data is incomplete, noisy, and high-dimensional.

Bridging the divide between low-level non-symbolic information and more abstract symbolic representations requires *anchoring*: "creating and maintaining the correspondence between symbols and sensor data that refer to the same physical object" (Coradeschi & Saffiotti, 2003), a special case of the symbol grounding problem. Anchoring can be both bottom up, where new percepts create and update anchors, and top down, where the system ties an object description to its corresponding percepts. In both cases, the goal is to generate a set of *anchors*: data structures that contain both symbolic and perceptual information about objects in the world.

Bottom-up anchoring combines the problems of *data association*, updating the appropriate anchor with new measurements, and *tracking*, maintaining the anchoring correspondences over time as objects move (Elfring et al., 2013). These two problems are often tackled together through *multitarget tracking* or *MTT* (Vo et al., 2015) – locating multiple objects or targets in a video feed and tracking them over time. This is challenging because real-world robots often encounter objects that move, change in appearance, look identical, or occlude one another. In addition, perception often involves noise and errors. We have developed a taxonomy of anchoring errors that include five distinct types:

- **E1 False anchor:** An anchor is created for a non-existent object, usually due to sensor noise.
- **E2 Missing anchor:** An anchor does not exist for a known object (e.g., from occlusion).
- **E3 Misidentified anchor:** A object percept is not mapped to the correct anchor but is used to create a new one (e.g., due to a tracking error).
- **E4 Merged anchor:** One anchor exists for multiple objects (e.g., due to segmentation errors).
- **E5 Fragmented anchor:** Several anchors exist for one object (e.g., due to segmentation errors).

A common system organization in cognitive robotics is to have a component that does perceptual processing, another that does bottom-up anchoring using the new percepts, and a cognitive component that handles the high-level abilities such as reasoning and planning. In this approach, the anchoring component is opaque to the cognitive component, which only receives the symbolic portion of the anchors as the end result. The advantages of a separate, external anchoring component are that it can be fast, continuously active, and not interrupt the agent's deliberate reasoning. However, this separation can lead to two major issues.

First, the cognitive component can have high-level knowledge that would be useful during anchoring. This can include knowledge of the agent's current actions and goals, object affordances and properties, and knowledge of environmental regularities. For example, consider a game of cups and balls where someone places a ball under a cup and then moves the cup. The agent cannot rely on direct perceptual information to track the ball, but it can use domain knowledge about containers. Or consider a robot that moves a block with its arm and during movement the block is occluded from the camera by the arm. The agent has high-level knowledge about its goal of placing that block at a given region that could be used to reacquire the track when the object reappears at the intended destination. A third example concerns agent using a knife to cut an apple. If the agent has appropriate actions models, it can predict that one complete object will become two halves. These examples suggest that there needs to be a channel for knowledge to flow back into the anchoring component. However, many approaches to anchoring do not have mechanisms for incorporating knowledge in this way.

The second issue with this hard separation between the anchoring and cognition is that the agent lacks the information needed to detect and handle the anchoring errors described above. For example, suppose there is a misidentified anchor, where an object is assigned a new id. If the agent only has access to the symbolic portion of the anchors, it is difficult to determine if an error actually occurred or if one object appeared at the same time another disappeared. Or consider if one block is placed upon another and occludes the bottom one, causing the anchor to become missing. If that

occluded block is necessary for the agent to detect a goal as being satisfied, it will fail to detect it. In cognitive systems research, the possibility of these errors is sometimes overlooked. However, if a real-world robotic agent is relying on consistent, long-term object identity in order to complete its objectives, encountering anchoring errors can significantly degrade its performance. Improving the accuracy of the bottom-up anchoring can help, but no purely bottom-up approach is likely to be perfect and, as described above, the agent can have access to knowledge that can correct some of these errors that occur when low-level assumptions about the world are violated.

This paper both explores and demonstrates how an agent in a symbolic cognitive architecture with access to a spatial short-term memory can use spatial reasoning and domain knowledge to participate in the anchoring process and to detect and correct bottom-up anchoring errors in its input. A key insight of our approach is that it maintains two sets of anchors. As is common, an anchoring component performs standard bottom-up anchoring to maintain a *perceptual anchor set* that is provided as input to the cognitive component. However, this input not only contains the symbolic information stored in working memory, but also metric information in spatial memory. Crucially, the cognitive component maintains a second *belief anchor set* that it uses during planning and reasoning. The cognitive component compares these two sets of anchors to identify discrepancies between them and then attempts to reconcile them. This process can update the belief anchors with new information or detect an anchoring error and attempt to reconcile it. Error handling can be internal (e.g., just ignore the error) or external (e.g., telling the anchoring component to merge two anchors). The agent uses knowledge about the world, object affordances, and its current tasks and actions to aid in this process.

By using domain knowledge and participating in the anchoring process, the agent maintains this belief anchor set as its world representation. This representation is more *accurate* (avoids certain errors), *robust* (ignores noise), and *comprehensive* (maintains information not currently perceived) than that given by perception. This approach has been used across several real-world and simulated robotic domains to enable high-level task reasoning, planning, and learning even when the perceptual systems are noisy and error prone and when they have only a limited view of the world. In the following sections we first describe related work and then go into details of this approach, which includes extensions to support temporally extended tasks in large scale, multi-room environments. After this, we present an experiment that evaluates the approach's performance on a complex task in the presence of significant occlusion, tracking, and segmentation errors.

2. Related Work

Previous research on detecting and recovering from anchoring errors (Bouguerra et al., 2006; Broxvall et al., 2005) has focused on top-down anchoring, where an ambiguous object description matches multiple percepts. Most of the work related to anchoring focuses on how to avoid producing errors, as opposed to correcting for errors that do occur. This can be done through better object persistence, object tracking, or object/motion models. The work of Loutfi et al. (2005) reduces missing anchor errors (**E2**) by adding object persistence to maintain anchors for objects that are out of view. The system of Blodow et al. (2010) also maintains beliefs about objects it cannot currently observe and then uses probabilistic anchoring to identify them when they come back

into view, even after they have moved, to guard against misidentified anchors (**E3**). The work of Elfring et al. (2013) represents a major step in integrating a sophisticated tracking algorithm with anchoring, using a Multiple Hypothesis Tracker algorithm to improve tracking reliability during occlusion and detect clutter (false positives). This reduces the occurrence of anchoring errors **E1**, **E2**, and **E3**. Heyer & Graser (2012) use relative anchors to represent relations between two reference anchors. These help in noisy environments where detecting one object improves the chance of detecting the other. They also incorporate knowledge about actions to update the anchors of objects being moved. Persson et al. (2019) described a probabilistic reasoning system that infers the state of occluded objects and continue tracking through occluded movements. Nitti et al. (2014) implemented a relational particle filter that draws on commonsense world knowledge during tracking, such as when one object is inside another. These approaches to anchoring usually assume perfect perceptual segmentation (none of the percepts are merged or fragmented), so they are susceptible to errors **E4** and **E5**.

There has been a large body of work in multiple target tracking apart from anchoring (Vo et al., 2015; von Hoyningen-Huene & Beetz, 2009; Khan et al., 2006). These approaches can handle complex tracking scenarios with merged measurements and occlusions. Our approach is complimentary in that it aims to provide knowledge in situations where the model assumptions are violated (e.g., an object is fully occluded while the robot moves it) or where a tracking error occurs.

3. System Overview

Figure 1 presents an overview of the system architecture, which consists of three major components: perceptual, anchoring, and cognitive. The perceptual component takes raw sensor data and generates a set of object percepts, measurements, and classifications. The anchoring component then takes that data and updates a set of anchors while tracking the objects. The cognitive component is implemented in a symbolic cognitive architecture and does planning and reasoning for the agent. It receives updates about the perceptual anchors, compares them to its own set of anchors, and uses reasoning to keep them consistent. It can also send commands to the arm controller and communicate with a person through a chat interface.

3.1 Perceptual Component

The perceptual component takes incoming sensory data and segments it into a set of percepts, where each percept is a collection of raw measurements assumed to originate from the same object. It also extracts features for a set of attributes from the percepts. For example, for the color attribute, it computes the average RGB values of the pixel data for that object, and for the position attribute, it estimates XYZ coordinate of the object's center position. Our approach requires that the perceptual processing produces features that include position, rotation, and scale transforms for a bounding box volume of the object in a globally consistent reference frame. It must also produce classifications that are unary predicates about visual properties of the object, such as color (`red`), shape (`sphere`), size (`large`), and object categories (`table`), along with confidence values from zero to one that allow multiple hypotheses per property. The agent assumes each property has one valid value (e.g., objects have a single color) and adopts the hypothesis with the highest confidence.

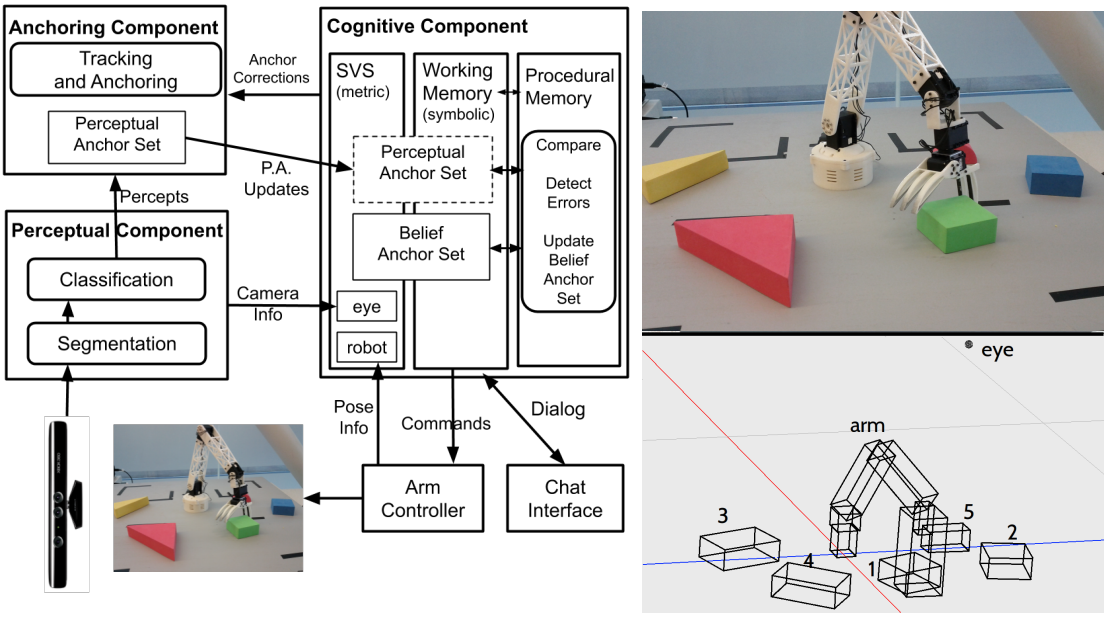

Figure 1. Left: An overview of the agent architecture. Right: An example scene in the tabletop domain and the corresponding scene graph representation in SVS (the agent's spatial memory).

Our approach does not assume any specific perceptual algorithms, and it has been used in several environments that involve different perceptual processing. In the tabletop domain, a Kinect camera senses the scene and generates a 3D point cloud. That point cloud is segmented into object percepts that consist of a collection of 3D points. For each one, the perceptual component classifies properties such as color, shape, and size and calculates a minimal bounding box.

3.2 Anchoring Component

The anchoring component is responsible for bottom-up anchoring by incorporating new percepts in the current set of anchors (each identified with a unique symbol x_i). Its two main responsibilities are *acquiring* new anchors for newly perceived objects and *tracking* existing anchors by updating them with new perceptual information. The end result is a set of anchors that represent the current world state, which we designate the *perceptual anchor set*. These anchors are then sent to the cognitive component, but without the percept data. We require that the multiple target tracking (MTT) algorithm must support three operations to correct anchoring errors:

- **move-anchor**(x_i, pos)**:** Changes the position of the anchor with the given id x_i.
- **change-anchor-id**(x_i, x_j)**:** Changes an anchor id from x_i to x_j.
- **merge-anchors**($x_i, x_j, ...$)**:** Merge the anchors with the given ids together (keep id x_i).

Table 1. The SVS filters used to extract spatial knowledge about objects.

Distance(x, y)	The distance between the object centroids.
Volume(x)	The volume of x's bounding volume
Intersects(x, y)	True if the bounding volumes of x and y intersect
Overlap(x, y)	The percentage of x's bounding volume that intersects y's
Occlusion(x, p)	The approximate percentage of object x visible from point p

In our system, the MTT algorithm used is akin to a simple nearest neighbor approach, where new object percepts are matched against the previous perceptual anchors with a one-to-one mapping, trying to minimize differences in both position and volume. This tracking approach is not probabilistic and only uses the previous step's information. However, our method does not require that a particular MTT algorithm be used. Thus, it is compatible with probabilistic or other multiple hypothesis algorithms that support the operations above.

3.3 Cognitive Component

The cognitive component is implemented in the Soar cognitive architecture (Laird, 2012), which has multiple memories and processes that support anchoring. Soar's memories can contain both the symbolic and spatial information from the anchors. When an update is received from the anchoring component, the symbolic portion of the anchors (the identifying symbol and set of predicates) is placed into an input buffer in working memory and the metric portion (bounding box information) is put into Soar's Spatial Visual System (SVS). This is a short-term spatial memory that provides a scene graph representation of objects and their bounding volumes (see Figure 1). In addition to the object bounding boxes, SVS contains a rough 3D model of the robot that is updated with new pose information. For example, it contains a volume for each segment of the arm for the tabletop robot. It also includes the position of the camera and a frustum that represents the 3D volume of the camera's target region. This is used to determine whether the robot should be able to see an object in a partially observable environment, as we describe later.

SVS provides the necessary representations and computational infrastructure to support complex reasoning over metric data, such as computing intersections or occlusions. This reasoning would be very difficult to do using procedural rules alone. It uses a scene graph representation of objects together with a set of filter queries that extract symbolic, relational, and metric information from the scene graph representation and serve as an abstraction over it. Table 1 lists the filters used in this paper. The agent also calculates an `in-view(x)` predicate. In a fully observable environment (such as a tabletop arm), it is always true. In a partially observable environment, it is computed using `intersects(x,` o_{view}`)`, with the view volume provided by the perceptual system.

Soar also has several long-term memory stores, including a rule-based procedural memory that contains knowledge about how to reason over its perception and thus implements the approach described in this paper. In addition, the architecture includes a declarative episodic memory that automatically captures a historical trace of the agent's working memory from which it can later retrieve information about past events.

Table 2. The discrepancies that can be detected between the perceptual and belief anchor sets.

New-Object(x)	$x \in A_P$ and $x \notin A_B$
Missing-Object(x)	$x \notin A_P$ and $x \in A_B$
Moved-Object(x)	$distance(pos(x, A_P), pos(x, A_B)) > c_{move}$
Grown-Object(x)	$vol(x, A_P)/vol(x, A_B) > c_{grown}$
Shrunken-Object(x)	$vol(x, A_P)/vol(x, A_B) < c_{shrunk}$
New-Predicate(x, p)	$p(x) \in A_P$ and $p(x) \notin A_B$
Missing-Predicate(x, p)	$p(x) \notin A_P$ and $p(x) \in A_B$
Different-Location(x, p)	$cur\_loc(A_P) \neq cur\_loc(A_B)$

4. Detecting and Handling Anchoring Errors

In order to detect and handle anchoring errors, the agent maintains two separate sets of anchors. The *belief anchor set* (A_B) is the agent's internal representation of the current state of the world stored in Soar's working memory. In contrast, the *perceptual anchor set* (A_P) is maintained by the anchoring component and then is sent to the cognitive component. Since there are two sets of anchors, SVS may contain two different bounding volumes for each object, one from A_P and one from A_B. The agent can select which set to use with SVS queries.

The belief anchor set represents the agent's current knowledge about the environment and is used for reasoning and planning. Having a world representation distinct from perception means that it is only updated through the agent's deliberate reasoning. This supports accuracy and robustness by allowing the agent to correct certain perceptual errors, ignore some perceptual noise, and only include changes it believes to be correct. It is more comprehensive since it can contain information not currently provided by perception, such as objects that are not currently in view. However, because the belief anchor set is not automatically updated from perception, the agent must keep it up to date so that it does not deviate from reality.

The cognitive component detects errors in the output of the anchoring component by comparing the perceptual anchor set against the belief anchor set. When it detects a discrepancy, it determines why the discrepancy arose and how to resolve it. Discrepancies are not necessarily due to anchoring errors. They can arise because the environment has changed and the belief anchor set is no longer accurate. By doing this within the cognitive component, the agent can use other knowledge it possesses to influence this process.

4.1 Discrepancy Detection

The agent has access to both the perceptual anchor set A_P and belief anchor set A_B in working memory and SVS. To detect differences between these two sets, the agent has procedural rules that continually compare the two world representations and detect discrepancies. These rules fire in parallel with other procedural knowledge and do not interrupt or interfere with the agent's deliberate reasoning. Only when a discrepancy is detected will the agent perform deliberate reasoning to determine its cause and resolve it. Table 2 shows a list of these discrepancies.

The constants c_{move}, c_{grown}, and c_{shrunk} are environment dependent parameters and determine which differences are considered significant. Due to perceptual noise, the position and volume values constantly change, even for a stationary object. Setting a low threshold causes the agent to spend more time reasoning about unimportant differences and decrease efficiency, but a high threshold may cause the agent to miss important changes and be less accurate. We use hand-tuned thresholds and reserve learning them as an area for future research. In the tabletop domain, c_{move} = 2cm, c_{grown} = 1.2, and c_{shrunk} = 0.8.

4.2 Discrepancy Resolution

Once a detector is triggered, the agent does deliberate reasoning (implemented as operators in Soar) about the detector. First, it uses SVS filters to gather more information and identify *why* the detector was triggered – whether it represents an actual change in the environment or some perceptual/anchoring error. Second, it takes the appropriate actions to resolve the error by using knowledge about physical properties of objects to guide reasoning. Implicit in this reasoning are assumptions that two objects cannot occupy the same physical space, objects occlude one another, objects maintain spatial and temporal continuity (i.e., do not teleport or vanish), and objects do not merge, split apart, or change in size. Obviously, these assumptions are sometimes incorrect (cutting an apple will split the object) or appear to be violated due to incomplete sensing (an object can appear to teleport when moved by a person). However, we have found that for our domains, they are sufficient for most cases. Below, we describe how the agent handles each of the five anchoring errors.

A false anchor error (E1) occurs when the perceptual system creates an anchor for an object that is not actually there and usually results from perceptual noise. To reduce the chance of accepting false anchors, when the cognitive component sees a `new-object(x)` detector, it notes the time it appears and waits some time c_{new}. If after that time the new object is still present, it is added to the belief set. Having a longer time makes the agent less susceptible to false positives, as they would have to span multiple perceptual updates, but less reactive to the appearance of new objects. We set the constant c_{new} to be one second, which is a good compromise for our environments.

A missing anchor error (E2) occurs when the perceptual anchor set does not contain an anchor for an existing belief object. In this case, the `missing-object(x)` detector will be triggered and there will not be any corresponding `new-object` detectors. The agent must determine if the object is actually no longer present or simply not currently visible. To do this, it checks whether the object is being occluded (`occlusion(x, eye)` > $c_{occlusion}$) or no longer `in-view(x)`. If either of these conditions is true, the agent concludes the object must still be present, so it keeps the belief anchor. In this way, agent deals with partial observability by maintaining anchors for objects that are not current visible. Here we use $c_{occlusion}$ = 0.2.

A misidentified anchor error (E3) occurs when a new anchor is incorrectly created from a percept instead of the appropriate existing anchor being updated. In this case, the `missing-object(x)` and `new-object(y)` detectors are triggered. The agent checks to see if the bounding volume of the new anchor overlaps the missing one. If `overlap(y, x)` > $c_{overlap}$ (and there are no conflicting properties), then it merges the anchors by sending a `change-anchor-id(y, x)` command to the perceptual system to correct the tracking error. Here we use $c_{overlap}$ = 0.5.

A merged anchor error (E4) occurs when an anchor incorrectly contains percepts that correspond to multiple objects. This can happen if the perceptual component fails to segment two or more objects into distinct percepts. The cognitive component can correct for this if at one point the anchors were not merged. When a `missing-object(x)` detector triggers, the agent checks for overlap between the missing object and the current bounding volume information for perceptual objects. If there exists a perceptual object y where `overlap(x, y)` > $c_{overlap}$, the missing object's bounding volume is contained inside another's volume, and the agent deduces that the percepts for object y belong to both x and y. Thus the object is no longer labeled as missing.

A fragmented anchor error (E5) occurs when multiple anchors represent a single object. This can happen if the perceptual component over-segments an object into multiple fragments. The cognitive component can correct for this if at one point the anchor was not fragmented. One or more `new-object(x)` detectors will be triggered, and the agent checks whether there is some belief object y whose bounding volume contains the new object. If `overlap(x, y)` > $c_{overlap}$, then it will send the command `merge-anchors(x, y)` to the perceptual component.

If the agent detects none of the above errors, then it updates its belief anchor set with the new perceptual information. In the case of `new-object(x)` or `missing-object(x)`, the anchor is either added or removed from A_B. In the other cases where the position, volume, or predicates differ between the two anchors, the cognitive component checks for occlusion (`occlusion(x, eye)` > $c_{occlusion}$). If the object is not occluded, it updates the belief anchor with the new information and otherwise ignores the discrepancy. This means the agent only updates a belief about an object if it has an unobstructed view.

Resolving discrepancies requires deliberate reasoning, which competes with the agent's other tasks. How much this happens depends on the environment and perceptual system. If the environment is relatively static and the perceptual system is reliable, then the agent only occasionally needs to update its belief. In most cases, resolving a discrepancy only takes a few decision cycles ($<$ 10ms) and there are hundreds between perceptual updates. However, the agent can become overwhelmed if there are hundreds of changes per second. In such cases, it can choose to delay the perceptual processing and give priority to other cognitive activities, at the cost of decreased reactivity.

4.3 Domain Knowledge and Top-Down Reasoning

A major advantage of allowing the cognitive component to participate in the anchoring process is that it can use domain knowledge and top-down reasoning to improve overall performance. In our research, we have examined three types of domain knowledge that are useful for anchoring and how each of them is applied during perceptual reasoning.

First, the agent has knowledge about objects and their affordances. One assumption is that two objects cannot occupy the same space, but this is not true of containers. For example, the bounding volume of a milk carton can be inside the refrigerator. Normally this will result in the agent determining that there are two percepts (milk and refrigerator) due to a fragmented anchor and merge their anchors. However, if it knows that an object is a container (e.g., a refrigerator, a cup, a trash can), then it rejects the fragmented anchor hypothesis and keeps them as separate objects. Although not currently implemented, there could be other ways of using object knowledge to aid in anchoring, especially in cases where the above assumptions are violated. For example,

objects may sometimes split apart, such as when taking the lid off a jar, or combine, when putting the lid back on. Some objects are transparent and this knowledge could be used when reasoning about occlusion.

Second, the agent also has knowledge about its current actions and how they might affect its perception, such as when it is moving objects with a robotic arm. In our experience, as the arm moves and interacts with the scene, it causes occlusions and perceptual noise, while also making it difficult to track the object in the gripper. To counteract this, the agent does not update its belief anchors while the arm is moving. In addition, it will not remove an anchor for an object that is in the gripper and will update its position in SVS to that of the gripper. When it puts the object down, it moves its belief volume to where it expects the object to appear (e.g., on top of another block) and sends a `move-object(x, pos)` command to the anchoring component. When that object becomes visible, the anchoring component is more likely to track it once it appears at the anticipated position. This can correct tracking errors for moved objects that are difficult to overcome otherwise.

Another important application of action knowledge occurs when the robot arm drops an object that it is trying to move. The motor system provides a signal indicating that the grab action failed. The agent responds to this by looking for any new objects in the environment that match the object it was moving. For example, if it is moving a red cube and detects that it dropped the object, it looks everywhere on the table for a new red cube to resolve the missing anchor. Furthermore, if the agent still cannot find the object (e.g., if it fell off the table), it asks the human for assistance in locating the missing object. In our implementation, the person can place the object on the table and say *'Here is the cube.'* This instruction lets the agent anchor the new percepts to the belief object and try to pick up the object again.

A third type of domain knowledge concerns the current task. The task may involve objects that cannot be immediately anchored, so the agent creates a new belief anchor without any spatial or metric information. For example, suppose the current task is to throw away a soda can into a trash can, but there is no trash can currently in the belief anchor set. The agent adds the trash can to its belief state and then can use it to plan how to complete the task, even without knowing the object's location. Part of this planning might be a subtask of finding the trash can, and once one is visible, it merges the new perception object with this belief object. In our system, such belief objects also come from human interaction. For example, if the instructor gives an instruction such as *'Fetch me a stapler,'* the agent adds a stapler object to the belief anchor set. This lets it reason and plan about objects that it cannot currently perceive.

4.4 Handling Multiple Locations

In a tabletop environment, the agent can keep track of all the objects on the table. However, with a mobile robot operating across multiple rooms in real-world environments, the capacity to track everything is quickly exceeded. To address this limitation, we assume that it is reasonable to only keep track of the objects in the robot's immediate surroundings or those related to the its current goals. Our approach involves having a known map of the environment, which is divided into convex regions called *locations*, usually a room or a part of a hallway. The robot's current location is included in perceptual updates. When the agent moves to a different location, it sees this as a

context switch and clears its belief anchor set, except for those that are involved in the current task. For example, if the task is trying to fetch a stapler from a different room, the stapler is not deleted.

This approach reduces the number of objects being tracked at any one time and allows the agent to work within multiple rooms and with their associated objects. However, this comes at the cost of deleting information that might be needed later. For example, suppose the agent stores a certain mug inside a cupboard and then leaves the room. If later it is back in that room and needs to find the mug, it would have to search the entire environment. Soar has a long-term episodic memory that automatically records the history of working memory and includes the symbolic information in the belief anchor set. However, it does not include the spatial information; in Soar there is no long-term spatial memory, a shortcoming that we plan to address in the future.

To overcome this limitation within the current architecture, when the agent leaves a location and before it removes the belief anchor set from its memory, it performs queries to SVS to extract the metric information (position, rotation, and scale) from each object and adds it to working memory. This snapshot is then automatically recorded in episodic memory. When the agent enters a new location, it performs an episodic memory retrieval cued on the last time it was in that location. This retrieves an episode that includes this snapshot of spatial memory and provides enough information to reconstruct the previous belief anchor set for that location. The room might have changed since it was last visited, but any incorrect beliefs can be fixed as the robot drives around.

5. Empirical Evaluation

We have used this approach to support agents that can learn tasks from instruction in several domains, including the tabletop arm, a mobile robot that operates across multiple rooms (Mininger & Laird, 2016), a simulated indoor kitchen environment, and a toy forklift robot. Although these environments differ in their embodiments and perceptual processing, our method provides a stable and reliable representation of the world across these different domains. To evaluate this approach, we focused on the tabletop environment.

5.1 Experimental Design

We evaluated performance on an end-to-end task in the tabletop environment with an arm that can manipulate foam blocks. A suitable task must require that having an accurate belief anchor set is necessary to perform the task well and that the agent will be exposed to common perceptual challenges likely to induce various anchoring errors. We chose the task of measuring two nonvisual properties of objects (temperature and weight), finding the object with the highest or lowest value of that property, and placing it onto a goal region. This is referred to as a *superlative request* (e.g., find the heaviest block). These properties are simulated and can only be attained through a measurement action involving placing the block on a certain region on the table (e.g., the scale). This task is difficult because the agent must know the values for all the objects before being able to achieve the goal. Every time it fails to track an anchor, it must remeasure the object by moving it back onto the relevant measurement region.

Our evaluation metric is the number of movements it takes to satisfy each superlative request. The more accurately the cognitive component can maintain its set of anchors, the fewer remeasure-

ments are needed. This evaluation does not directly record how many anchoring errors the system makes, but the performance on this task is directly tied to how well the cognitive component can detect and recover from errors. We evaluate performance in this way because ground truth is difficult to obtain and we want to examine the influence of errors on end behavior. If the agent does manage to measure and track all six objects, it will always identify the correct block. For some agent and domain combinations this task is extremely difficult, so we consider the run a failure once the agent has done thirty moves without finding the correct block.

The blocks are placed onto a tabletop with the robotic arm at the center. Different areas of the table are designated as named regions where the arm can put objects. The *goal* region is where the agent places the requested block, whereas the *scale* and *thermometer* regions are used to gather weight and temperature information. Figure 2 labels these regions as G, S, and T, respectively. There are also several *bin* regions where the agent must place blocks it is not using. If there are multiple blocks on a bin, the agent must stack them.

We evaluated four versions of the agent across four versions of the domain. Each agent includes all the capabilities of the previous versions plus an additional one (Figure 3). This gives a qualitative comparison of how adding error handling capabilities improves the overall performance. We tested each agent version across four variations of the domain, which vary in complexity and perceptual difficulty (Figure 4). In each one, the task of finding objects is the same, but the objects and configuration of the table differ.

We ran each combination of agent and domain using the same script of find requests. We gave the agent two requests for the same type of superlative, with five random moves in between. For example, it would find the lightest block, randomly move five blocks, then find the lightest block again. These intermediate moves provided more opportunities for anchoring errors to occur. We report the average number of moves required to satisfy the second superlative request. The better the agent is at maintaining an accurate anchor set, the fewer remeasurements the agent will make, since it can *immediately* satisfy the request if all the anchors were successfully maintained.

Table 3. Different agent variations, each one includes the capabilities of those previous.

A1. No Error Handling	The agent relies exclusively on the perceptual anchors for planning
A2. +Action Knowledge	The agent notifies the anchoring component when it moves a block
A3. +Object Permanence	The agent can reason about occlusions and maintain anchors to handle errors related to **E1**, **E2**, **E3**.
A4. +Segmentation Reasoning	The agent does not assume one anchor per object, and can handle errors relating to **E4** and **E5**.

Table 4. Different versions of the domain that vary in difficulty.

D1. No Occlusion 6 bins and 6 blocks	This has one bin per block, so no stacking or occlusions, making segmentation easy. There should be minimal anchoring errors.
D2. Partial Occlusion 3 bins and 6 blocks	The agent must stack blocks, and so it has to deal with partial occlusion of objects lower on the stack (Figure 2). Stacking greatly increases the chances of fragmented segmentations (**E5**), tracking errors (**E3**), and noise (**E1**).
D3. Total Occlusion 3 bins and 6 blocks Only 1 bin visible	In this version, we artificially restrict perception so that the agent can only view objects in one bin at a time. The agent must deliberately select which bin it can observe. Thus the perceptual anchor set will be missing anchors for occluded objects (**E2**).
D4. Merged Percepts 3 bins and 6 blocks 2 block colors	Because there are only 2 colors, stacked blocks of the same color are often segmented as one object. The anchoring component will produce anchors that correspond to multiple merged objects (**E4**).

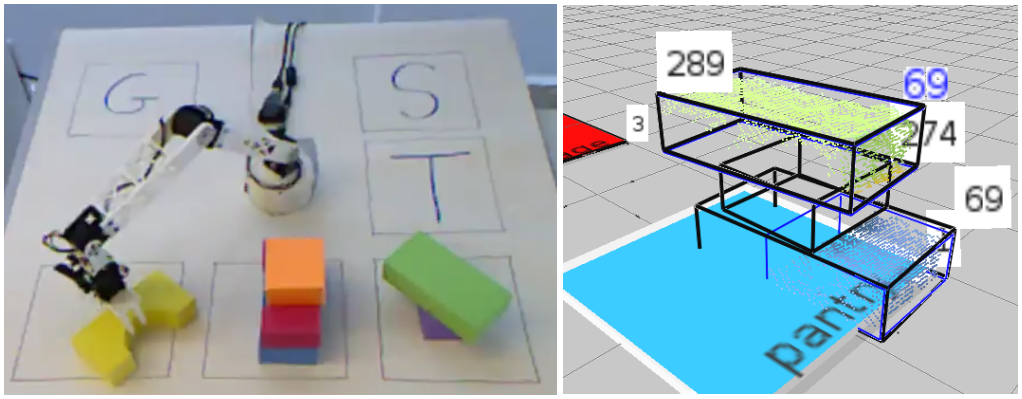

Figure 2. Left: The second domain variation D2 with three bins to stack objects and goal, scale, and thermometer regions (labeled G, S, and T). Right: The point cloud data from the Kinect camera during stacking showing total occlusion of the middle block and partial occlusion of the bottom one.

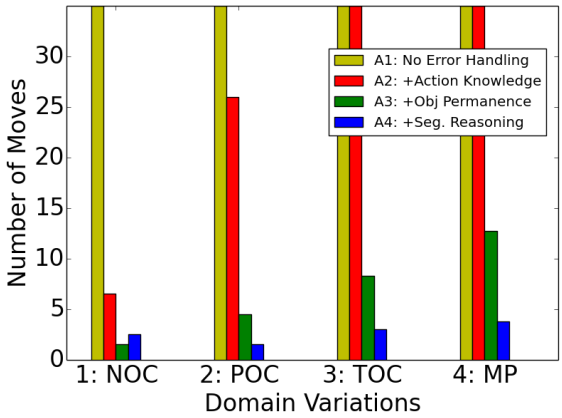

Figure 3. The number of moves to satisfy each request for a previously seen superlative averaged across four runs. The best possible score is 1. Bars above 30 indicate failure.

5.2 Experimental Results

Figure 3 shows the performance of each agent type in the different domain variations averaged across four runs. These results are summarized below.

- Agent **A1**, which uses only the perceptual anchor set to make decisions, always fails, even the easiest variation. This is because the anchoring component makes assumptions during tracking that are broken when the arm moves an object (due to arm occlusion, the block disappears and reappears somewhere else). When this occurs, the agent loses the nonvisual information needed for the task.
- Agent **A2**, which uses knowledge about its actions to inform the anchoring component when it moves an object, can maintain correct anchors through movement, but cannot recover from anchoring errors. Thus, it succeeds in variation **D1** but has a very difficult time on the harder variations where there are many more anchoring errors.
- Agent **A3**, which does have the knowledge of object permanence, can deal with errors due to partial and total occlusions in the harder variations. It successfully completes the task across all variations. However, in **D3** and **D4**, there are more errors due to anchors being merged or fragmented that causes it more difficulty.
- Agent **A4**, which has the complete set of knowledge (including how to handle errors relating to merged and fragmented anchors), performs well in all of the domain variations, with only a small decrease in performance on the harder ones.

These results show that the error handling capabilities of the cognitive component makes the system more *robust*. Without those capabilities, the number of anchoring errors make the task impossible for the agent to perform, as demonstrated by **A1**. As error handling capabilities are added, the overall performance of the system improves. Another desired characteristic is that the perceptual reasoning is *efficient*. It is important that the extra processing required to handle the

errors and maintain the separate belief anchor set does not significantly slow processing. The agent must remain reactive to new updates and changes in the environment. To evaluate the overhead of our approach, we measured the average time per arm movement that the cognitive component spent processing the perceptual anchors and maintaining the cognitive state. The basic agent **A1** spent 410 ms per arm movement, while the full agent **A4** spent 331 ms per movement. Qualitatively, this shows that the error-handling capabilities does not slow the agent, and may sometimes make it faster by letting it ignore perceptual noise. Since each arm movement takes around fifteen seconds, the cognitive component spends less than three percent of its time handling perceptual updates, most of which are done while the arm is moving when the rest of the cognitive component is idle.

6. Conclusion

In this paper, we have demonstrated a method for allowing an agent in a symbolic cognitive architecture to detect and correct bottom-up anchoring errors in its input. The key feature of this approach is that the agent maintains two sets of anchors, one updated by perception and one representing the agent's beliefs, which it compares to detect discrepancies and resolve them. This allows the agent to correct anchoring errors and incorporate domain knowledge into the anchoring process. Consequently, the representation of the world used for planning and reasoning, which comes from the belief anchor set, is more accurate, robust, and comprehensive.

There are some limitations to this approach. First, if the system cannot process perceptual updates fast enough, it will become overwhelmed. This can happen if the environment is very noisy or if it violates the agent's assumptions. It can also occur if there are too many objects, as some operations grow quadratically in the number of objects. Consequently, performance is negatively impacted with more than ten to fifteen objects. Tracking objects only in the current room reduces the likelihood of this problem arising, but this could be further aided by the additional of an attentional mechanism that restricts the agent to only focusing on parts of the world that are important to the current task.

Another limitation is that the cognitive component assumes objects are mostly stationary, and thus anchoring errors caused by moving objects will not be handled property. In addition, it cannot it detect when the anchoring component has swapped the identifiers for two objects. Another possible source of error is from the approximation of the volume as a bounding box. Objects that do not fill their bounding boxes will negatively affect the error handling procedures, as the overlap and occlusion calculations will be inaccurate. SVS supports geometry that is more complex than simple bounding boxes, so that if a perceptual system could provide more accurate geometric information, it would likely improve performance further. We have tried to make the cognitive component independent of the specific anchoring algorithm used. However, the system uses simple perception and tracking algorithms, which makes errors more likely. Adding a more sophisticated MTT method would lead to fewer errors and improve performance. An interesting area of future research would be determining how the corrections made by the cognitive component could be integrated with a multiple hypothesis tracking system, probabilistic or otherwise. While such approaches could reduce the noise and errors in the world representation and make some of our anchoring strategies

unnecessary, there will still be occasions in which the assumptions inherent in the bottom-up approaches are violated and top-down reasoning could help correct.

Acknowledgements

The work described here was supported by the Air Force Office of Scientific Research under Grant Number FA9550-15-1-0157 and the Office of Naval Research under Grant N00014-18-1-2337. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of AFOSR, ONR, or the U.S. Government.

References

Blodow, N., Jain, D., Marton, Z.-C., & Beetz, M. (2010). Perception and probabilistic anchoring for dynamic world state logging. *Proceedings of the Tenth IEEE-RAS International Conference on Humanoid Robots* (pp. 160–166). Nashville, TN: IEEE.

Bouguerra, A., Karlsson, L., & Saffiotti, A. (2006). Situation assessment for sensor-based recovery planning. *Frontiers in Artificial Intelligence and Applications*, *141*, 673–677.

Broxvall, M., Coradeschi, S., Karlsson, L., & Saffiotti, A. (2005). Recovery planning for ambiguous cases in perceptual anchoring. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 1254–1260). Menlo Park, CA: AAAI Press.

Coradeschi, S., & Saffiotti, A. (2003). An introduction to the anchoring problem. *Robotics and Autonomous Systems*, *43*, 85–96.

Elfring, J., Van Den Dries, S., Van De Molengraft, M., & Steinbuch, M. (2013). Semantic world modeling using probabilistic multiple hypothesis anchoring. *Robotics and Autonomous Systems*, *61*, 95–105.

Heyer, T., & Graser, A. (2012). Intelligent object anchoring using relative anchors. *Proceedings of the Thirteenth International Conference on Optimization of Electrical and Electronic Equipment* (pp. 1444–1451). Brasov, Romania: IEEE.

von Hoyningen-Huene, N., & Beetz, M. (2009). Robust real-time multiple target tracking. *Proceedings of the Ninth Asian Conference on Computer Vision* (pp. 247–256). Xi'an, China.

Khan, Z., Balch, T., & Dellaert, F. (2006). Mcmc data association and sparse factorization updating for real time multitarget tracking with merged and multiple measurements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*, 1960–1972.

Laird, J. (2012). *The soar cognitive architecture*. Cambridge, MA: MIT Press.

Loutfi, A., Coradeschi, S., & Saffiotti, A. (2005). Maintaining coherent perceptual information using anchoring. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* (pp. 1477–1482). Edinburgh, Scotland.

Mininger, A., & Laird, J. E. (2016). Interactively learning strategies for handling references to unseen or unknown objects. *Proceedings of the Fourth Annual Conference on Advances in Cognitive*

Systems (pp. 1–16). Evanston, IL.

Nitti, D., De Laet, T., & De Raedt, L. (2014). Relational object tracking and learning. *Proceedings of the 2014 IEEE International Conference on Robotics and Automation* (pp. 935–942). Hong Kong, China: IEEE.

Persson, A., Martires, P. Z. D., Loutfi, A., & De Raedt, L. (2019). Semantic relational object tracking. *arXiv preprint arXiv:1902.09937*.

Vo, B.-N., Mallick, M., bar shalom, Y., Coraluppi, S., Osborne III, R., Mahler, R., & Vo, B.-T. (2015). Multitarget tracking. *Wiley encyclopedia of electrical and electronics engineering*, (pp. 1–25).