Chapter 4  

Applying the Methodology  

(Bruce)

The first agent designed by the design methodology laid out in chapter 3 was Bruce [82][83]. Bruce’s task is to play hide-and-seek in our laboratory against a human controlled opponent. Figure 6a shows Bruce and figure 6b shows the vehicles that he can play against (though only one at a time). The human hides the vehicle somewhere within the game area (see figure 7). Bruce will search the game area for the opponent and attempt to touch, i.e. tag, it upon sight. Once Bruce has spotted the opponent, the human may drive it away from Bruce. If Bruce looses his opponent during the chase, he should examine nearby objects to see if the opponent is hiding behind them. If Bruce does tag his opponent, he wins and the game is over.

Bruce was built at UVA and is based on the RugWarrior board [38]. He possess two in-
dependent drive wheels with fairly coarse shaft encoders, a single color camera on a pan/tilt platform and a “bump skirt” capable of detecting impact on the front, left and right sides. Onboard processing is done by a pair of Motorola MC68HC11s. Bruce has a radio modem to communicate with his host workstation, a Sun Sparc 10, where most of the processing is done. Bruce also has a video transmitter that broadcasts the images from his camera to a Datacube MV200 for image processing.

In this section I examine how Bruce’s software architecture was developed using the methodology presented in this thesis. Figure 7 shows a plan to search the game environment that Bruce must execute. The numbers indicate steps in the plan with the solid line representing the path to follow. The dashed lines indicate where and in which direction Bruce should turn his camera to look for his opponent. Bruce and his field of view are shown near the completion of step 3. The other shaded shapes denote objects in the game zone. By following the methodology’s steps and answering its questions, a design to follow this plan and play hide-and-seek emerges.

![Figure 7. The Search Plan](image)

4.1. Task Decomposition

*What are the agent’s primitive skills?* The designer must now see what capabilities can be combined into “black box” skills that can be used in the bottom-up portion of the de-
composition process. Bruce’s basic skills are navigation to a location and control of the camera’s pan/tilt mount. The agent navigates to a location by combining motor and perceptual capabilities. By servo-ing toward the position of location-specific perceptual characteristics detected in its camera, Bruce can move throughout his environment. By varying these characteristics, the agent can move to a variety of static objects or chase the opponent.

Which tasks can be decomposed into sequential subtasks? Which can be decomposed into parallel subtasks? As mentioned previously, playing hide-and-seek consists of searching for the opponent, chasing the opponent (to tag it) and possibly performing another search behind nearby objects if Bruce looses the opponent during a chase. These three tasks are shown as subtasks of the play-hide-and-seek task in figure 8 and the control flow between them, i.e. the order in which they execute, is shown in figure 9a. Note that in figure 4, arrows denote flow control with dashed arrows indicating control flow between parent and child tasks.

In order to search, the agent needs to go between landmarks looking for the opponent and avoiding obstacles. In order to view the entire game area, Bruce will need to point his camera at certain areas of the environment, but he need not navigate to them. So, the search task

![Figure 8. Hide-and-Seek Task Decomposition](image-url)
consists of a navigation task (move-toward-landmark), an obstacle detection task (detect-obstacles), a camera control task (look-at-region) and a task that monitors for the opponent (watch-for-opponent), all operating in parallel. These are shown as subtasks of the search-for-opponent task in figure 8. In figure 9b, these tasks are shown as stacked ovals, meaning
they operate in parallel. The control flow arrows from the parent task show that they all begin executing at the same time, but only move-toward-landmark and watch-for-opponent can transfer control back to the parent (and thus end the execution of all four tasks). This occurs when Bruce either reaches the landmark he is moving toward, or detects the opponent.

When the opponent is detected, Bruce should begin chasing it. In this situation, control flows from the search-for-opponent task to the chase-opponent task (figure 9a) and from the chase-opponent task to the tag-opponent task (figure 9d). The tag-opponent task simply visual servos Bruce toward the opponent until Bruce’s “bump skirt” detects an impact (and the opponent is close enough that Bruce believes he has impacted the opponent and not something else).

If the opponent escapes while being chased, Bruce should look for it behind any object in the vicinity of where Bruce lost visual contact. This process is handled by the look-behind-occluder task and its sequential subtasks, shown in the right of figure 8. The control flow of figure 9c shows that several sets of parallel subtasks are used to check if the opponent is hiding nearby. The watch-for-opponent task runs in parallel with every other task so that Bruce switches to the chase-opponent task if the opponent is detected at any point. When looking behind objects, Bruce must first select an object, near the opponent’s last known position, that is possibly occluding his view of the opponent, i.e. the opponent is hiding behind it. This is handled by the choose-occluder task. Then Bruce must drive “behind” it (relative to his starting location). Due to the mechanics of Bruce’s drive system, going behind the object involves moving until the rear (drive) wheels are past the object and then turning back to face it. This sequence is performed by the drive-past-occluder and
turn-back-to-occluder tasks. Note that in figure 9c, the watch-for-opponent subtasks only transfer control back to the look-behind-occluder tasks while the other subtasks transfer control to the next subtask in the sequence. This is because the other tasks are sequential steps in “looking behind” an object, while watch-for-occluder indicates that a different branch of the decomposition (chase-opponent) should take over.

4.2. Identify Task Roles

*What are the task roles?* The search-for-opponent task has three roles, **opponent**, **landmark** and **region**, to represent the aspects of the environment important to searching the game area. Its subtasks must deal with the complexity of navigation. So the move-toward-landmark task has **landmark** and **obstacle** roles since where it directs the agent is influenced by the position of the goal landmark and any obstacles. The detect-obstacles task has an **obstacle** role. Watch-for-opponent has an **opponent** role to associate with the vehicle Bruce must tag, and look-at-region has a **region** role corresponding to the area of the game environment at which Bruce must look.

The chase-opponent and tag-opponent tasks each have the **opponent** role since these tasks are concerned only with driving toward the opponent. The various subtasks that make up look-behind-occluder (choose-occluder, drive-past-occluder and turn-back-to-occluder) all have the **occluder** role. The drive-past-occluder task also has a role called **intermediate-target**, which is not mentioned in the task name. This role is associated with an object that Bruce can use to estimate his position relative to the occluding object (see figure 11b). Figure 10 shows the same task decomposition diagram as figure 8 annotated with the task roles. The roles are shown in bold when they occur in the task name and in callout boxes otherwise.
What entities can fulfill those roles? The four roles in the search portion of the hide-and-seek game are landmark, opponent, region and obstacle. The landmark role will be played, at various times, by the various objects along the agent’s search route (see figure 7). The opponent role can only be played by Bruce’s opponent, which is either the dump truck or the racecar from figure 6b, but not both in the same game. The same is true for the opponent role in chase-opponent and its subtask. The region role can be filled by any portion of the game area at which Bruce needs to look. In fact, this role can be better described as a neck angle than as specific features of the environment. Finally, the obstacle role is filled by any object that is along the direct path between Bruce and his current navigation goal.

In the look-behind-occluder portion of the game, the occluder and intermediate-target roles can be filled by any object that is in the correct position. That is, the occluder can be any object that has approximately the same azimuth and is closer than the last known position of the opponent while the intermediate-target can be associated with any object that is off to the side of the occluder and further away than the occluder plus one Bruce-length. This allows the same motor control system used in the move-toward-landmark task to be
re-used to servo Bruce toward the intermediate-target. However, this time, he must stop when he has passed the occluding object and not when he reaches the intermediate-target.

4.3. Representation of Task Roles

*What role bindings are shared between tasks?* For Bruce’s task, this is simple. The *landmark*, *region* and *opponent* roles are shared between search-for-opponent, and its child tasks move-toward-landmark, look-at-region and watch-for-opponent respectively. The *obstacle* role of the detect-obstacles task is shared with the move-toward-landmark task since that task calculates how to move the agent’s wheels based on the *landmark* and *obstacle* roles. The *opponent* role is also shared between the watch-for-opponent, chase-opponent, tag-opponent) and look-behind-occluder tasks. From figure 9 we can see that both the watch-for-opponent tasks in the decomposition hierarchy transition to their parent task which transitions to the chase-opponent task. The *opponent* binding is shared so that Bruce chases the detected vehicle. Finally, the *occluder* role is shared between the subtasks of the look-behind-occluder task (choose-occluder, drive-past-occluder and turn-back-to-occluder).

*What information about the entity bound to a task role is needed for the task?* In this domain, the agent must navigate through the environment searching for the opponent. The tasks of move-toward-landmark, detect-obstacles, look-at-region, tag-opponent, drive-past-occluder and turn-back-to-occluder all need the positions of the entities associated with their roles to control the agent’s effectors. Watch-for-opponent and choose-occluder both determine initial positions for the objects associated with their roles and pass this data on to other tasks. So, the important information about the entities associated with task roles in this domain is position. I chose to store positions in ego-centric, polar coordinates be-
cause the agent’s capabilities make position estimates relative to its current location much more accurate than position estimates relative to some external frame. In other words, any position computed from information gathered by sensors mounted on the agent can be turned into an agent-centric position by a simple transform (such as shifting by the current camera pan angle or translating from the sensor’s position to the center of the agent’s body). Although such positions can be converted to other coordinate frames, maintaining positions in those frames requires either special sensors or knowledge of the positions of multiple other landmarks to triangulate [21]. Since Bruce will not often be able to see many landmarks (and his encoders are error prone), he is much more successful with an ego-centric system.

For what roles would it be useful to develop an explicit representation? The landmark, opponent and occluder roles can all benefit from explicit representation because Bruce may have to take action with respect to them when they are outside his field of view. The move-toward-landmark task may begin with the next landmark outside the current field of view and a representation for that role will allow Bruce to begin moving toward it. The tag-opponent task benefits from role representation because the opponent may drive outside of Bruce’s field of view during the chase and Bruce may be able to reacquire the opponent by turning in the direction where it was last seen. The object bound to the occluder role will necessarily go outside Bruce’s view during the look-behind-occluder task, specifically during drive-behind-occluder. The occluder role needs representation so that Bruce can execute the turn-back-to-occluder task because the occluding object will not be visible after completing the drive-past-occluder task (see figure 11b).

The intermediate-target role would seem to not need representation because, as shown
in figure 11b, it is played by an object that is always within sight of the agent. However, obstacle avoidance en-route could cause the intermediate-target to fall outside Bruce’s field-of-view and so a representation for this role could be useful.

The **region** role is interesting because there is no object in the environment that the agent should bind to this role, rather it is described by a head angle to pan the camera in search of the opponent. As such, the head angle could just be a parameter for the “look-at” action and the role needs no representation. However, I have opted to represent this role in order to have a uniform communication structure between parent tasks and their children (see Section 4.7 for a discussion on this communication interface).

The representation of the **obstacle** role is an interesting issue that causes iteration between the representation and perception steps of this methodology. Bruce’s vision system often mis-identifies obstacles (false positives). If the **obstacle** role has a representation, it may take longer for the agent to realize that the binding is incorrect than it would take in a stateless system. This is the classic argument against representation put forward by Brooks...
[15]. Since a stateless system cannot store information, it must re-evaluate the environment whenever it wishes to act. This means it can forget previous erroneous percepts in the minimum possible time. A system with state or representation may continue to make decisions based on previously determined, incorrect information. This argues that obstacles should not have a representation since they are often misidentified and this may cause Bruce to avoid empty space. However, due to the limited ability of the camera to pan, obstacles go outside Bruce’s field of view, as shown in figure 11a, step 2. Without an obstacle representation, Bruce would invariably hit obstacles along his flank when his camera is past them, but his drive wheels are not. This argues for a representation of local-space [14]. The designer can choose to have a representation for the \textbf{obstacle} role and try to improve the perception or choose no role representation and try to improve the navigation control algorithm to avoid obstacles outside the field of view. For Bruce, I use a representation compromise. Obstacles that are detected at more than a certain distance away are not bound to the \textbf{obstacle} role, although Bruce’s navigation system will take their position into account on that cycle of his PA loop. This allows Bruce to be only somewhat fooled by far away, false obstacles. When the obstacles are too close to Bruce, they are bound to the \textbf{obstacle} role so that they are represented should they pass outside the camera’s field of view. Clearly, there should be multiple \textbf{obstacle} roles, since Bruce can be avoiding one obstacle that he can’t see when another obstacle blocks his new path. However, Bruce’s simple visual avoidance algorithm cannot avoid two obstacles and try to move toward the goal\textsuperscript{1}.

\textbf{1.} Part of the thinking in designing Bruce’s obstacle avoidance routine was that a stateless system was almost capable of handling the avoidance. Representation was only needed to remember obstacles that had passed outside the field of view. Since this thesis is about the design of representation systems, significant effort was not spent in designing a routine to avoid multiple obstacles. Once the \textbf{obstacle} role is bound, Bruce will not avoid other obstacles until this role is unbound.
How often should the task role information be verified? Since the positions of the entities associated with the roles are used to control Bruce’s effectors (wheels and camera platform), they should be verified at the same rate as the effector control loop. The role maintenance scheme here is simple. The agent needs to know about the position of either a landmark or occluder (depending on the current task), plus any obstacle in order to navigate. He also must attempt to detect the opponent at all times. Therefore, for each pass through the active task’s effector control loop, the agent must verify the position of the entities associated with the current task’s roles and attempt to bind the opponent role.

4.4. Perception

What information can be extracted from the environment to recognize the entities that should be bound to the current task’s roles? Bruce’s perception system consists of a single color camera whose signal is transmitted to an off-board receiver and processed by a combination of a Datacube MV200 image processor and a Sun Sparc 10. This hardware supports two primitive visual operations at speeds high enough to support Bruce’s task, edge detection and color histogramming. Bruce’s visual identification routines are straightforward given the availability of these operations. First, Bruce finds the ground/non-ground boundary, or groundline [34] as shown in figure 12. Large vertical discontinuities in this line may represent the boundaries of objects (note the Sprite can). Object identification is accomplished by comparing the color histogram of the pixels between pairs of discontinuities with stored histograms for various objects [72]. So, by comparing the truck histogram with the histogram of the pixels within discontinuity pair 1 and within discontinuity pair 2, Bruce can determine that pair 2 delineates the truck, while pair 1 does not.

I designed Bruce with the necessary histograms to recognize the landmarks that appear
along his search path, as well as the opponent he must chase. Obstacles can be detected as any objects (i.e. pair of discontinuities) that are closer than Bruce’s current goal, and on a direct path to the goal (see Appendix A, section A.1.1 for more on how the navigation system works).

Bruce must be able to determine the positions of the entities associated with the task roles. For this, the “ground-plane constraint” [34], which states that the ground is a flat horizontal plane and all objects in the world rest on the ground, was used. This constraint allows Bruce to calculate the distance to an object based on the smallest distance between the bottom of the image and any of the points along the groundline between the object’s discontinuity boundaries. That is, using known camera geometry, we can relate distance from the bottom of the image to depth. Bruce computes azimuth from an object’s position within the image and the camera’s known angular field of view and pan angle. These polar coordinates (azimuth and depth) are ego-centric, meaning with the agent at the coordinate sys-
tem’s origin. Since Bruce’s tasks consist of moving himself toward objects in the world (either to search or to chase) this works fine for the leaf tasks. However, the main search task must convert from the landmark coordinates stored in Bruce’s game area map to ego-centric coordinates for navigation and vice versa. I discuss this in Appendix A.

How does the duration of various role/entity bindings effect the perception system? First, the binding duration of the roles in Bruce’s tasks must be determined. In particular, the designer is interested in roles that have “long” binding durations. In other words, roles that stay associated with the same object over long periods of time, i.e. are bound to different objects infrequently relative to their task’s execution time. There is no single definition of how much time is “a long time”, it is relative to the agent’s perceptual capabilities (how often the binding could be changed) and the execution time of the agent’s tasks.

In general, there are two ways for a role to have a “long” binding duration. The first is for a single task to have a role remain bound to the same entity for the duration of its execution (which is “a long time”). Bruce does this with the landmark, region and intermediate-target roles of the move-toward-landmark, look-at-region and drive-past-occluder tasks respectively. Each of these roles should remain bound to a single object for the duration of the task that uses them. The second means of having a long binding duration is to have many sequential tasks share a role that they expect to be bound to the same object. Both the opponent and occluder roles are shared by multiple, sequential tasks and all those tasks expect the role to be bound to the same entity. The only role in this domain with a “short” binding duration, i.e. one that is not long, is the obstacle role. On each cycle of Bruce’s perception/action loop, the detect-obstacles task will bind this role to any object that is an obstacle to Bruce’s current destination, even if that object is the same object that
was an obstacle last time. The catch is that there could be a long binding duration when an
obstacle goes outside the field of view of the front mounted camera, but can still be hit by
Bruce’s body with its rear mounted wheels. If the obstacle is close enough to the edge of
the field of view, the role must remain bound to it (so that Bruce remembers its location
until he is “past” it).

Now we can use the binding duration information to structure the perception system.
Bruce’s visual processing can be divided into two classes, location and tracking. Location
is the process by which a role is initially bound to an entity in the environment. Tracking is
the processes by which the information about the entity bound to a task role is maintained.
Most roles remain bound to the same entity for the duration of their task and (once bound)
the opponent and occluder roles remain bound across multiple tasks. This means that
Bruce can save a significant amount of visual processing by foveating the regions of space
occupied by entities bound to task roles. In this case, foveating means limiting visual pro-
cessing to a particular region of the image.

In principle the location and tracking processes could use completely different mecha-
nisms for identifying objects in the image(s). Presumably, the location process would be
slower but more accurate than the tracking process, while the tracking process could use
the context (previous position(s)) for assistance. Bruce, however, uses the exact same visu-
al operations (edge detection and color histogramming) for both location and tracking.
Tracking is faster because any discontinuity boundary pairs that are too far from the last
known position of any object are not matched to that object’s stored histogram. This means
fewer objects need to be analyzed and the computation can proceed faster.

One difficulty with this two process scheme is what to do if the associated object moves
beyond the tracking fovea. In other words, what happens if the associated object moves faster than expected (or the agent moves slower) or goes out of the field of view? Bruce needs a means of identifying when the tracking process is failing or has failed, so that he can activate the location process again. Simple thresholding on the “amount of match” between the stored histogram and the histograms of any objects detected in the groundline suffices for this purpose. The histogram matching process is a modified version of Swain and Ballard [72] and Terzopoulos and Rabie [73]. It produces a percentage match value equal to the number of pixels of matching color between the two histograms divided by the total number of histogrammed pixels (see section A.1.2 in Appendix A for more details).

When this score becomes too low, Bruce can disable the tracking process and either begin the location process again, or start the look-behind-occluder task.

What level-of-detail (or resolution) is required in the information of the representation?

This question must be answered for each of the primary subtasks of play-hide-and-seek. For the search-for-opponent task, the information being used is the position of various landmarks, obstacles, regions and, at some point, the opponent. Since this is being used for effector control, we want it to be as accurate as possible. This means that Bruce should be estimating (dead-reckoning) such positions as little as possible, implying that he should be pointing his visual resource at these objects as often as possible. During obstacle avoidance, it is often impossible to keep the obstacle, the destination and the agent’s body’s current heading in view at the same time. Bruce must be looking at the objects effecting his navigation as often as possible, understanding that some trade-offs must be made based on how long it has been since an important object was last seen (see the discussion on confidence in Section 4.5).
The chase-opponent task is similar in that the opponent’s position is used for effector control and so must be as accurate as possible. This means Bruce should keep his camera trained on the opponent as much as possible.

The look-behind-occluder task has different needs for the occluder and intermediate-target roles. The intermediate-target is an object selected for its spatial relationship to the occluding object. Bruce’s physical layout (front camera, rear wheels) means he is more successful navigating to a point that is fairly straight in front of him than to a point behind an object he must maneuver around. This is because, at some point when moving around an object, that object can typically not be seen by the camera and so the estimated position of the object becomes inaccurate. So, Bruce selects an intermediate-target such that, if he navigates toward it for a certain distance, he will be in the correct position relative to the occluder to execute the turn-back-to-occluder subtask. Bruce can afford to reduce the accuracy of his estimate of the occluder’s position because his present concern is the intermediate-destination, an object that has been specifically chosen for the purpose of getting Bruce to a certain position relative to the occluder. He only needs to accurate enough knowledge of the occluder’s position that he can turn around and view the area behind it (remember he’s not really interested in finding the occluder, but the opponent).

4.5. Communication

What information is important in inter-task relationships? The play-hide-and-seek task is divided into three groups of communicating subtasks. By examining each group, we can discover the role information communicated between tasks. The search-for-opponent task moves Bruce throughout the game environment attempting to find the opponent. This task has opponent, landmark and region roles about which it communicates with the watch-
for-opponent, move-toward-landmark and look-at-region subtasks respectively. The search-for-opponent task gives the move-toward-landmark task a perceptual description of a landmark to which it can bind its **landmark** role and then maneuver the agent toward the associated object. It also gives the watch-for-opponent subtask a description of the opponent so that the **opponent** role in that task can be bound when it is detected (and trigger the switch to the chase-opponent task). The look-at-region subtask receives a description (the position) of the region to look at in hopes that the watch-for-opponent task will bind the **opponent** role while the agent is looking there.

The search-for-opponent task must communicate this perceptual data to its subtasks because there are various landmarks and regions that Bruce needs to be concerned with throughout the task. In other words, the move-toward-landmark task should not include hard-coded perceptual information because the agent’s route plan will dictate what landmarks the agent should move toward. By having the search-for-opponent task pass this role information to its subtasks, the subtasks can be reused to execute each leg of the search route (instead of having a different leaf task for each landmark or region). In fact, since the agent can play the hide-and-seek game against different opponents (see figure 6b), the watch-for-opponent task can be parameterized by opponent perceptual description. The parent task will know what opponent to search for and communicate this to the subtask.

Another important kind of data that is communicated between search-for-opponent and its subtasks is action input and output. Input means parameters that the task needs to perform its action(s) and output means the results of the actions(s). Move-toward-landmark moves the agent to within a certain tolerance of the landmark. This tolerance is determined by search-for-opponent to insure Bruce stops at the correct position (to perform a look-at-
region for example). Move-toward-landmark reports when it has reached the specified landmark and supplies the landmark’s current ego-centric position. This allows search-for-opponent to determine the agent’s location in its map of the environment and decide on the ego-centric position of the next landmark to be visited. Note that the landmark’s current position is already computed as the “information” that the task needs to know about the entity associated with the role (as per section 4.3) and so the action result can be just a boolean value (action completed successfully or action failed) if the position is communicated as well. Both the watch-for-opponent and look-at-region tasks receive no configuration information about how to perform their actions on their task role, and each communicates success or failure to its parent task (watch-for-opponent signals “success” or “failure” by binding or not binding the opponent role).

Note that detect-obstacles does not communicate with the parent task. This is because it is a part of the search task that is abstracted away from search-for-opponent task. However, it does communicate with the move-toward-landmark task by binding the obstacle role. This role and the landmark determine the effector commands sent to the wheels by move-toward-landmark.

Next, I examine the inter-task communication in the chase-opponent task. The chase-opponent task consists of a single subtask, tag-opponent. These tasks communicate the perceptual description of the opponent and whether or not the opponent has been “tagged”, i.e. the result of the “chase” action. In addition, the tag-opponent task must communicate some information to decide if control should transfer to the look-behind-occluder task. I call this information “confidence” and it represents the agent’s belief that the stored position of the opponent is correct. As long as the binding between the opponent and the op-
ponent is being adequately maintained, i.e. the tracking function is finding the opponent in the camera images, the confidence stored in the opponent representation will remain high. However, if the opponent slips outside the field of view or behind an occluding object, the opponent confidence will decrease over time, until the chase-opponent task transfers control to look-behind-occluder.

Finally, the look-behind-occluder task and its subtasks communicate similar perception and action information. The choose-occluder subtask’s job is to select an object that is potentially occluding the opponent, based on the opponent’s last known position. When the occluder role is bound, i.e. the choose-occluder task successfully completes, the perceptual description of the bound object is passed up to the parent look-behind-occluder task so that is can be distributed to that task’s other children. The drive-past-occluder and turn-back-to-occluder inform their parent task when they have completed their respective missions (getting within tolerance of the object associated with the intermediate-target and facing the occluder). The fact that these tasks communicate this information to their parent task instead of to the next task in the sequence is really a decision that results from implementation control flow (see section 4.6 for an explanation).

4.6. Architecture

How should the agent’s tasks be laid out in its architecture? The first issue the designer must tackle is determining which tasks have rapid perception/action cycles, bind their roles to PRs (see section 3.6) and require no inference. Such tasks can be executed by the agent’s PA layer. Typically, these processes control some sensor(s) and/or effector(s). Table 4.1 shows the tasks and sensor processing or effector they control.
Most of the leaf tasks in the hide-and-seek decomposition control effectors, as shown by the number of wheel and camera mount controlling tasks in table 4.1. Watch-for-opponent and detect-obstacles control sensors, but also need to be PA layer tasks because their representations are used by other PA layer tasks. **Opponent** and **obstacle** are both PRs and the tasks that share these roles expect them to be maintained at effector control rates (so watch-for-opponent and detect-obstacles need to be in the PA layer).

**Table 4.1: Task Controlling Capabilities**

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Sensor/Effect Controlled</th>
</tr>
</thead>
<tbody>
<tr>
<td>move-toward-landmark</td>
<td>wheels</td>
</tr>
<tr>
<td>drive-past-occluder</td>
<td>wheels</td>
</tr>
<tr>
<td>turn-back-to-occluder</td>
<td>wheels</td>
</tr>
<tr>
<td>tag-opponent</td>
<td>wheels</td>
</tr>
<tr>
<td>look-at-region</td>
<td>camera mount</td>
</tr>
<tr>
<td>choose-occluder</td>
<td>camera mount</td>
</tr>
<tr>
<td>watch-for-opponent</td>
<td>opponent detection</td>
</tr>
<tr>
<td>detect-obstacles</td>
<td>obstacle detection</td>
</tr>
</tbody>
</table>

The other tasks in the decomposition tree belong in other layers of the architecture. The search-for-opponent task uses a map of the game environment to direct its (now PA layer) subtasks. The map is in a global coordinate system that is different from the egocentric system used to store positions in the PA layer tasks. Therefore, search-for-opponent uses a different representation. This representation does not need to be updated and so I place the search-for-opponent task in another layer of the architecture that I call the task executor layer, or TE. I also place the look-behind-occluder task in this layer so that it can sequence its subtasks to drive the agent behind an occluder. This is not the same control flow shown in figure 9. Control flows to the parent between each subtask. I choose to do this so the TE is in charge of activating and deactivating processes in the PA layer (via markers). This reduces the amount of control flow with which the PA layer need be concerned. Similarly, the chase-opponent task exists mainly to allow the switch between
phases of the hide-and-seek task (searching, chasing, etc.) to take place in the TE. As shown in figure 9, search-for-opponent transitions to chase-opponent, and chase-opponent activates the tag-opponent PA layer task. Watch-for-opponent could directly transfer control to tag-opponent, but I prefer control to flow through the TE.

Bruce does not need a planner to play hide-and-seek in this domain because he is given a search plan to execute\(^2\). Thus, the play-hide-and-seek task (the root of the decomposition tree) is not a separate task in the implementation. However, a more complex version of the game can be imagined in which play-hide-and-seek has an actual implementation and it is in a third layer of the architecture. Suppose that Bruce is chasing the opponent and he can no longer locate it. At this point, Bruce can continue his current search pattern, but if he had a route planner, he could compute a new search route based on his current location. Play-hide-and-seek could be a route planning task that passed its search plan to search-for-opponent. When a new plan was needed, control would transfer back to play-hide-and-seek and a new plan would be generated. This task, then, should be placed in a new layer of architecture because it requires more inference than the TE layer tasks. In fact, this task requires a full inference (planning) engine, whereas the TE layer tasks merely set the context for the PA layer to execute pre-defined plans. The TE tasks need no inference capabilities and so to preserve their cycle time, the route planning task should be in a different layer of the architecture.

4.7. Bruce Implementation

This section describes the result of the design process carried out in the previous sections. This section details how the answers to the methodology’s questions were combined to

\(^2\) I generated a search route for Bruce to follow for the game layout shown in figure 7.
form a working autonomous agent. The remainder of this section is organized as follows. First, I describe the agent’s PA layer representation and the operation of the PA layer’s main loop (including marker maintenance). I then describe the actual PA layer behaviors and how they achieve the agent’s goals. Lastly, I discuss how representation is used for inter-layer task communication.

4.7.1 Bruce’s Representation

First I will describe the structures that are used to represent task roles in the PA layer. These representations are called markers after the work of Agre and Chapman [2]. Based on the information communicated between tasks (section 4.5), each marker has components for all the communicated information outlined in section 3.7.3, except “progress”. It also has a “dependency list”. Table 4.2 summarizes what is contained in the various marker components and the dependency list. These summaries will be explained in greater detail as I discuss marker maintenance and the general operation of the PA layer’s main loop.

Table 4.2: Components of Bruce’s Markers

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The role name used in combination with Action (see below) to select the current action from the PA layer’s action table.</td>
</tr>
<tr>
<td>Property</td>
<td>Ego-centric, polar (r, θ) coordinates of associated object</td>
</tr>
<tr>
<td>Identify</td>
<td>Divided into Locate and Track subcomponents. Each has a color histogram of the object that this role can be bound to an “amount of match” threshold to declare a match successful. Although the histograms are the same, the match thresholds are typically different and the region searched for an appropriate object is different (see below).</td>
</tr>
</tbody>
</table>
Bruce’s PA layer operates in a tight perception/action loop. However, since the PA layer uses representation, it must have representation maintenance as part of that loop. In order to understand the maintenance algorithm and the PA layer’s loop, I must first introduce the concept of instantiated vs. uninstantiated markers. Simply, an instantiated marker is one for which the PA layer has selected an entity in the environment to associate with the marker, based on the marker’s Identify component. An uninstantiated marker then, is a marker that has not yet been associated with an entity. An uninstantiated marker often reflects a higher layer’s expectations about the world and the PA layer can confirm those expectations by instantiating (binding) the marker to some object. For example, when Bruce has completed step 3 of his search plan (see figure 7), object b3 is not within his field of view. Since b3 is the landmark for the next invocation of the move-toward-landmark task (step 4), the TE

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Used along with Index to determine an action to take from an action table. Also contains an action result component to store data for parent task about the outcome of a particular action. Note that this component can be null, indicating no action to take on this marker. This happens when a marker represents an object that is part of an action indicated by the Action component of another marker (see Dependency List).</td>
</tr>
<tr>
<td>Confidence</td>
<td>Contains a “found” flag that is set whenever the associated object is detected in the current image and a counter for how many images have passed without finding the associated object. Also contains the “instantiated” flag indicating whether or not this marker is currently associated with an object in the world.</td>
</tr>
<tr>
<td>Dependency List</td>
<td>A list of other markers in the PA layer that are needed to complete the action specified in the Action component. This is how tasks with multiple roles are executed. A marker’s Action component contains the action that completes the task and that task’s roles are represented by the marker and the markers in its dependency list. For example, obstacle markers are placed in the Dependency List of the landmark marker for the move-toward-landmark task.</td>
</tr>
</tbody>
</table>
layer search-for-opponent task will create an uninstantiated **landmark** marker with the appropriate Identify component for b3. Although the move-toward-landmark task cannot immediately detect object b3, the **landmark** marker contains b3’s ego-centric position. Bruce can begin to move toward the expected position of b3 and when b3 comes into view, he can start running the marker’s Locate routine to try and detect the object. When the Locate routine finds an appropriate object, it is associated with the marker and the marker is said to be instantiated.

### 4.7.2 The PA Layer Main Loop

The purpose of the PA layer’s perception/action loop is to select appropriate actions for the agent. Other layers influence the PA layer’s decision, but ultimately, it is the PA layer that controls the agent’s effectors. For the PA layer, actions will be closely coupled to current and recent perceptions. Since Bruce’s PA layer has representation, the action selected is a function of both the current perceptions and the information stored in the markers.

The main loop proceeds as follows. First, it executes its maintenance algorithm on the instantiated markers and then it tries to instantiate any markers that remain uninstantiated. Finally, the PA layer examines the Action component of each marker and executes the specified action. I will explain each of the steps in detail.

#### 4.7.2.1 PA Loop Step 1: Update Markers

This step verifies that the positions stored in the Property\(^3\) component of the markers are consistent with the positions of their associated objects in the world. Positions are updated by the following hypothesize-and-test algorithm. First, a new groundline is computed and

---

3. Recall that in section 3.7.3, I decided to call this component Property instead of Position (which would be appropriate here) because different agents may store different data in their representations.
“object regions” are segmented from it as in section 4.4. Next, new positions for all objects associated with markers are hypothesized by transforming the positions stored in the markers based on the wheel encoder values read since the last update. For each instantiated marker, if its position falls within the agent’s current field of view, the marker’s Track routine is used to analyze the objects detected in the groundline. The results of the Track routine and the distance between each object and the marker’s stored position are used to create an ordered list of the detected objects by likelihood that they are associated with the marker. After all instantiated markers have run their Track routines, each object creates an ordered list of markers by likelihood that it is “the” object associated with the marker, i.e. by differences between hypothesized marker position and detected object position. These ordered lists of markers to objects and objects to markers are passed to a stable marriage algorithm [40] that determines marker-object correspondences. If an object is selected as a marker’s correspondent, then the position of the object is stored in marker’s Property field and its found flag (as mention in the Confidence component of table 4.2) is set. If no object is selected as the marker’s correspondent, then the hypothesized position becomes the new stored position.

4.7.2.2 PA Loop Step 2: Instantiate Markers

After the instantiated markers have been updated, the uninstantiated markers can be matched to objects not associated with instantiated markers. Using the transformed positions from step 1, the agent proceeds through a similar matching process, using the results of the markers’ Locate functions instead of Track functions. This system is biased toward markers that are already instantiated because an object will only be allowed to correspond to one marker. In other words, because the instantiated markers get matched to the objects

130
first, an instantiated marker could be matched to an object that is “supposed to” match an
uninstantiated marker. This bias is allowed because an object associated with an instanti- ed marker has presumably been tracked over time, while an object associated with an un- instantiated marker is based on expectations of the higher layers. So, if the foveal image region searched by the Track function contains a matching object, it is more likely that that object corresponds to the instantiated marker than an uninstantiated marker. This is because perceptual context, i.e. previously executions of the PA loop, has lead the Track function to consider that image region, but the uninstantiated marker has yet to be grounded in per- ception. Bruce values perception over memory or expectations.

### 4.7.2.3 PA Loop Step 3: Select/Execute Action

After updating the markers, the PA layer looks through its list of markers to see if any have Action components with actions. It searches the list, in order, and executes the first non-null Action component it finds. This Action component specifies some action to be done on (or with respect to) the object associated with the marker. The PA layer has access to an action database indexed by the Action component and the Index component. Each of the “actions” in this database consists of one or more tasks executed in parallel (though the implementation is pseudo-parallel).

### 4.7.3 Task Implementation

The leaf tasks in the decomposition of figure 8 are implemented as behaviors or skills, called PA processes. These PA processes use the information in the marker containing the action and possibly information in other markers in that marker’s dependency list. Several tasks can be started by the Action component of a single marker. Table 4.3 shows the map-
ping between leaf tasks and marker Action/Index components.

### Table 4.3: Mapping of Tasks to Activating Marker Components

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Action and Index Component of Activating Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>move-toward-landmark</td>
<td>GoTo on <strong>landmark</strong> marker</td>
</tr>
<tr>
<td>detect-obstacles</td>
<td>GoTo on marker with any Index</td>
</tr>
<tr>
<td>look-at-region</td>
<td>LookAt on <strong>region</strong> marker</td>
</tr>
<tr>
<td>watch-for-opponent</td>
<td>(uninstantiated <strong>opponent</strong> marker)</td>
</tr>
<tr>
<td>tag-opponent</td>
<td>Chase on <strong>opponent</strong> marker</td>
</tr>
<tr>
<td>choose-occluder</td>
<td>(uninstantiated <strong>occluder</strong> marker)</td>
</tr>
<tr>
<td>turn-back-to-occluder</td>
<td>AlignWith on <strong>occluder</strong> marker</td>
</tr>
<tr>
<td>drive-past-occluder</td>
<td>GoTo on <strong>intermediate-destination</strong> marker</td>
</tr>
</tbody>
</table>

This section discusses how the representation activates PA processes and how they use the representation. The GoTo action on a **landmark** marker begins the move-toward-landmark and detect-obstacles processes. These processes share the **landmark** marker and any **obstacle** marker that is created in order to move Bruce toward a specific landmark. Actually, Bruce’s navigation system is more complex than just these two processes and is described fully in Appendix A, but those details are unimportant to the discussion here.

The look-at-region, chase-opponent and turn-back-to-occluder are each implemented by single PA process that are activated by the various marker Actions indicated in table 4.3. The implementation of each of these processes is unimportant as their effects have been amply described in previous sections.

Drive-past-occluder is activated by a GoTo action on an **intermediate-destination** marker. This marker comes from the TE (see next section) and its position and GoTo Action parameter are based on the occluder’s position. The parameter to the GoTo action (re-
call that parameters are also stored in a marker’s Action component) indicate how close Bruce needs to get to his destination. This parameter is based on the position in the occluder marker and should be close enough to the intermediate-destination to make sure Bruce can turn around and see the area behind the occluder. The GoTo action can be used again because the PA processes that implement GoTo can navigate Bruce to the position stored in any marker. In general, there need not be separate PA processes for each task because, for example, move-toward-landmark and drive-past-occluder can be implemented by the same ones.

The watch-for-opponent and choose-occluder tasks do not need implementing PA processes because they’re handled by step 2 of the PA layer’s main loop. The occluder marker has a special Locate routine that allows it to be bound to any object near the position stored in the opponent and the opponent Locate routine has a histogram for the truck or car opponent. The parent tasks in the TE monitor for these markers to be instantiated.

4.7.4 Inter-layer Communication

This section describes how the markers are used to communicate with the tasks in the TE layer. When a PA layer action completes, the PA processes associated with that action are deactivated and a reference to the marker whose Action component initiated the processes is passed to the TE. I say “a reference” merely to indicate that the marker is not deleted from the PA layer, and the TE is alerted that it should look at the marker’s contents. The TE’s response to information it reads in markers is usually to create new markers and pass them to the PA layer. The Action components of these markers will cause the PA layer to begin new actions. Now the TE layer may choose to delete the old marker from the PA layer, based on the completed task. For example, a landmark marker whose GoTo action has
been completed will be deleted, but an **opponent** marker that has been instantiated will not.

When a GoTo or LookAt action completes (meaning a landmark has been reached or a region has been looked at), the marker’s Action result component is set to COMPLETE and the marker is passed to the search-for-opponent task at the TE layer. Based on the **landmark** or **region**’s position relative to the agent and the current step in the search plan, this TE process generates the next **landmark** or **region** marker and passes it to the PA layer.

If the **opponent** marker is instantiated, and the agent is executing the tag-opponent PA process, the opponent marker’s confidence is assessed based on the count of the number of frames since the correspondent was found in the image. When this number grows too large, the tag-opponent PA process exits, placing the TARGET-LOST result code in the Action result slot of the marker and passes the marker to the TE chase-opponent task. This task then transfers control to the look-behind-occluder task.

The look-behind-occluder task generates an **occluder** marker that is passed to the PA. When this marker is instantiated, the TE creates an **intermediate-destination** marker with a GoTo action. This begins the drive-behind-occluder task. When that task completes, the TE places an AlignWith action on the **occluder** marker. The **occluder** marker is not deleted from the PA layer after instantiation and so it maintained during drive-behind-occluder. Turn-back-to-occluder uses the occluder position to turn back and face the object. If the opponent marker is instantiated at any point while executing this task, the same transfer of control to the chase-opponent task as before.