References

[Bar-Shalom and Fortmann, 1988] Y. Bar-Shalom and T. E. Fortmann, *Tracking and Data Association*, Academic Press, 1988.

[Dean and Kirman, 1992] T. Dean and J. Kirman, "Representation Issues in Bayesian Decision Theory for Planning and Active Perception," In *Proceedings: DARPA Image Understanding Workshop*, pages 763–768, 1992.

[Duda and Hart, 1973] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, 1973.

[Levitt *et al.*, 1989] T. Levitt, T. Binford, G. Ettinger, and P. Gelband, "Probability-based Control for Computer Vision," In *Proceedings: DARPA Image Understanding Workshop*, pages 355–369, 1989.

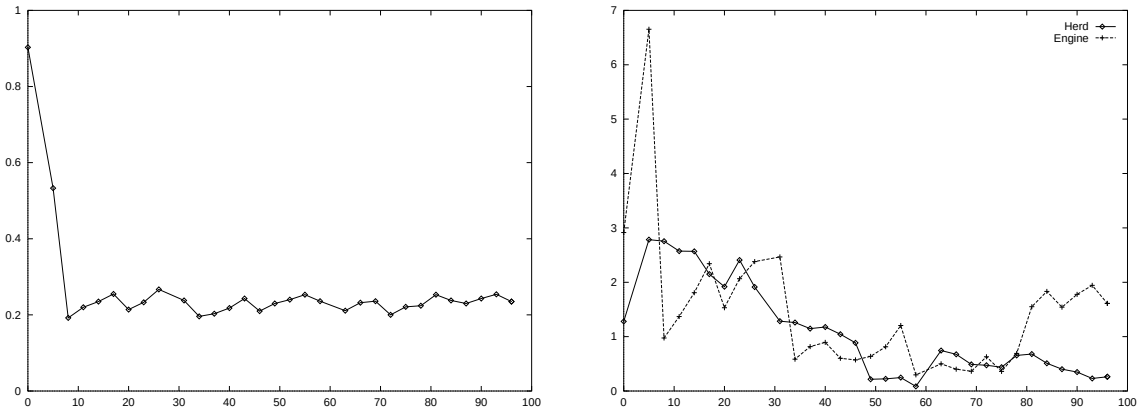
[Pearl, 1988] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufman, 1988.

[Rimey, 1993] R. D. Rimey, *Control of Selective Perception Using Bayes Nets and Decision Theory*, PhD thesis, Department of Computer Science, University of Rochester, 1993, (Also available as Technical Report 468, Department of Computer Science, University of Rochester, 1993).

[Rimey and Brown, 1992] R. D. Rimey and C. M. Brown, "Task-Oriented Vision with Multiple Bayes Nets," In A. Blake and A. Yuille, editors, *Active Vision*, pages 217–236. MIT Press, 1992.

[Rimey and Brown, 1993] R. D. Rimey and C. M. Brown, "Control of Selective Perception Using Bayes Nets and Decision Theory," *International Journal of Computer Vision*, 1993, To appear (Special Issue on Active Vision).

[von Kaenel, 1992] P. A. von Kaenel, "Mongoose: A Dynamic Domain Simulator to Support Purposive Sensory Motor Systems," Internal software documentation, 1992.



(a) (b)

Figure 20: (a) The average belief in the goal on a scene with high plant noise using the utility function V. (b) The average error on a scene with high plant noise using utility function V.

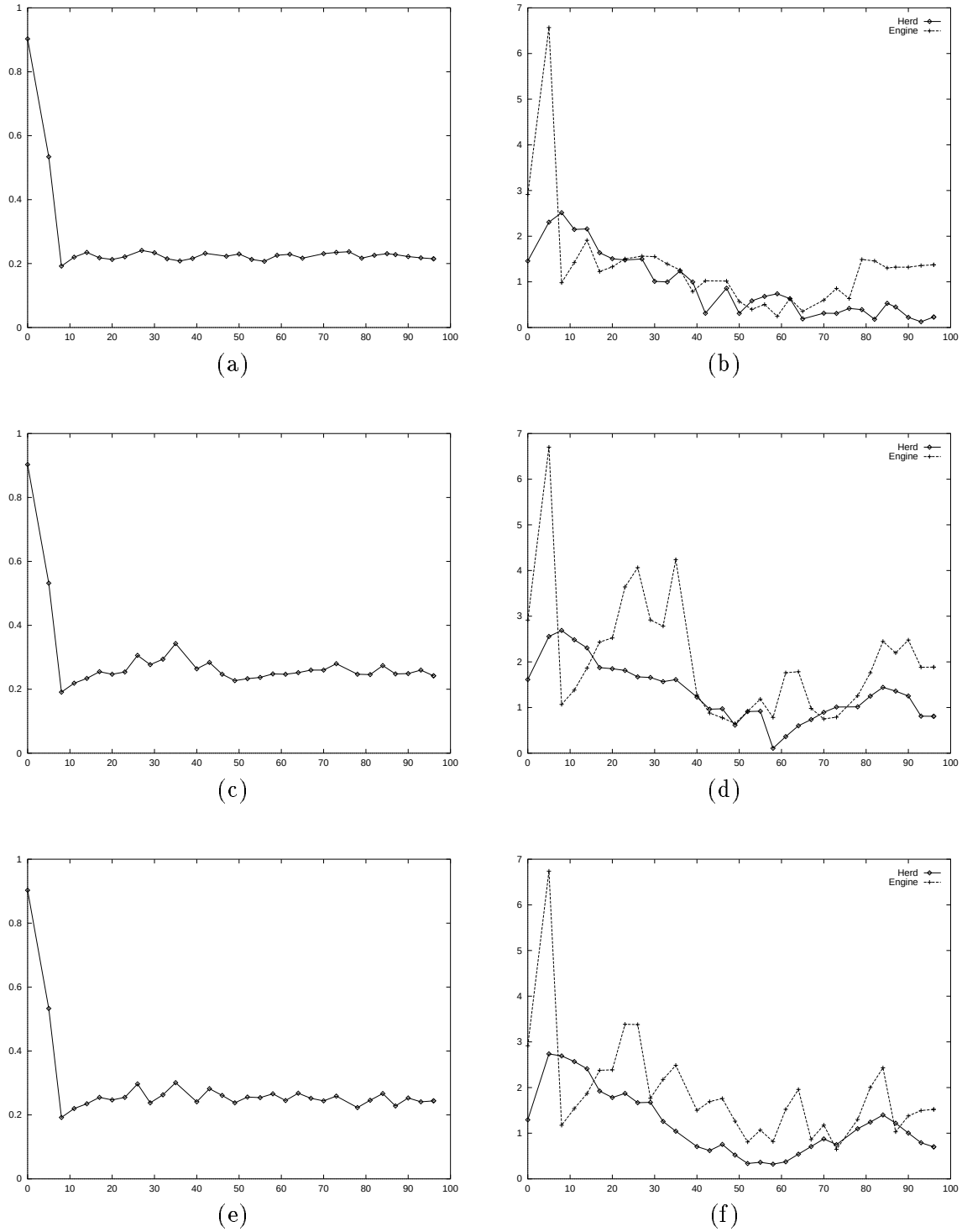


Figure 19: (a) The average belief in the goal on a scene with high plant noise using the utility function, $K(r) \times \frac{V}{rC}$. (b) The average error on a scene with high plant noise using the utility function, $K(r) \times \frac{V}{rC}$. (c) The average belief in the goal on a scene with high plant noise using the utility function $\frac{V}{rC}$. (d) The average error on a scene with high plant noise using the utility function $\frac{V}{rC}$. (e) The average belief in the goal on a scene with high plant noise using the utility function $\frac{V}{C}$. (f) The average error on a scenewith high plant noise using the utility function $\frac{V}{C}$.

Knowing how well dTEA-1 *believes* it did is a lot different than how well it really did. Fig. 19(b) shows the error for both the engine and the herd. The error is not much more than when there was less plant noise as seen in Fig. 17(b).

Hence, dTEA-1 performs well when using $K(r) \times \frac{V}{rC}$ as the utility function even in a scene with a lot of plant noise, because dTEA-1 searches for the engine and herd before their expected areas grow to large: *i.e* the function does not emphasize the expected area of the herd (which is larger) over the engine's.

5.4.2 Added Plant Noise using Utility Function $\frac{V}{rC}$

The second experiment uses the same scene and level of plant noise as in the previous experiment only it uses $\frac{V}{rC}$ as the utility function. The belief in the goal can be seen in Fig. 19(c). Notice how the belief jumps around as opposed to the belief in Fig. 19(a). There were several times when dTEA-1 lost the engine for a while, and that loss is reflected in the belief values. The error, shown in Fig. 19(d), reflects the occasional loss of the engine, especially during time 20 through 40.

5.4.3 Added Plant Noise using Utility Function $\frac{V}{C}$

The third experiment uses $\frac{V}{C}$ as the utility function. The belief in the goal can be seen in Fig. 19(e). The belief is not as stable as that in Fig. 19(a), and the error (Fig. 19(f)) is also worse.

5.4.4 Added Plant Noise using Utility Function V

The final experiment uses the simplest utility function, V. Since it does not care about the cost of an action, it tends to search for the herd more than necessary. The belief in the goal (Fig. 20(a)) is almost as good as when using $K(r) \times \frac{V}{rC}$, except that it has more variation. The error in Fig. 20(b) is worse than in Fig. 19(b), but overall, the utility function V performs better than $\frac{V}{rC}$ and $\frac{V}{C}$.

6 Conclusion

There have been many changes and additions made in dTEA-1 that allow it to perform tasks on scenes with moving objects. The system can be set up to work on scenes that including both constrained and unconstrained moving objects along with static objects. A train scene was used to test dTEA-1's ability to perform tasks on dynamic scenes since it includes constrained, unconstrained, and static objects. The experiments run on the scene demonstrate the power of the chosen utility function, the expected area update technique, and the error-handling capability which is in the form of an Alpha-Beta filter and well defined confidence regions. From the experiments it has been determined that the current configuration is good at tracking objects even when there is a lot of plant noise.

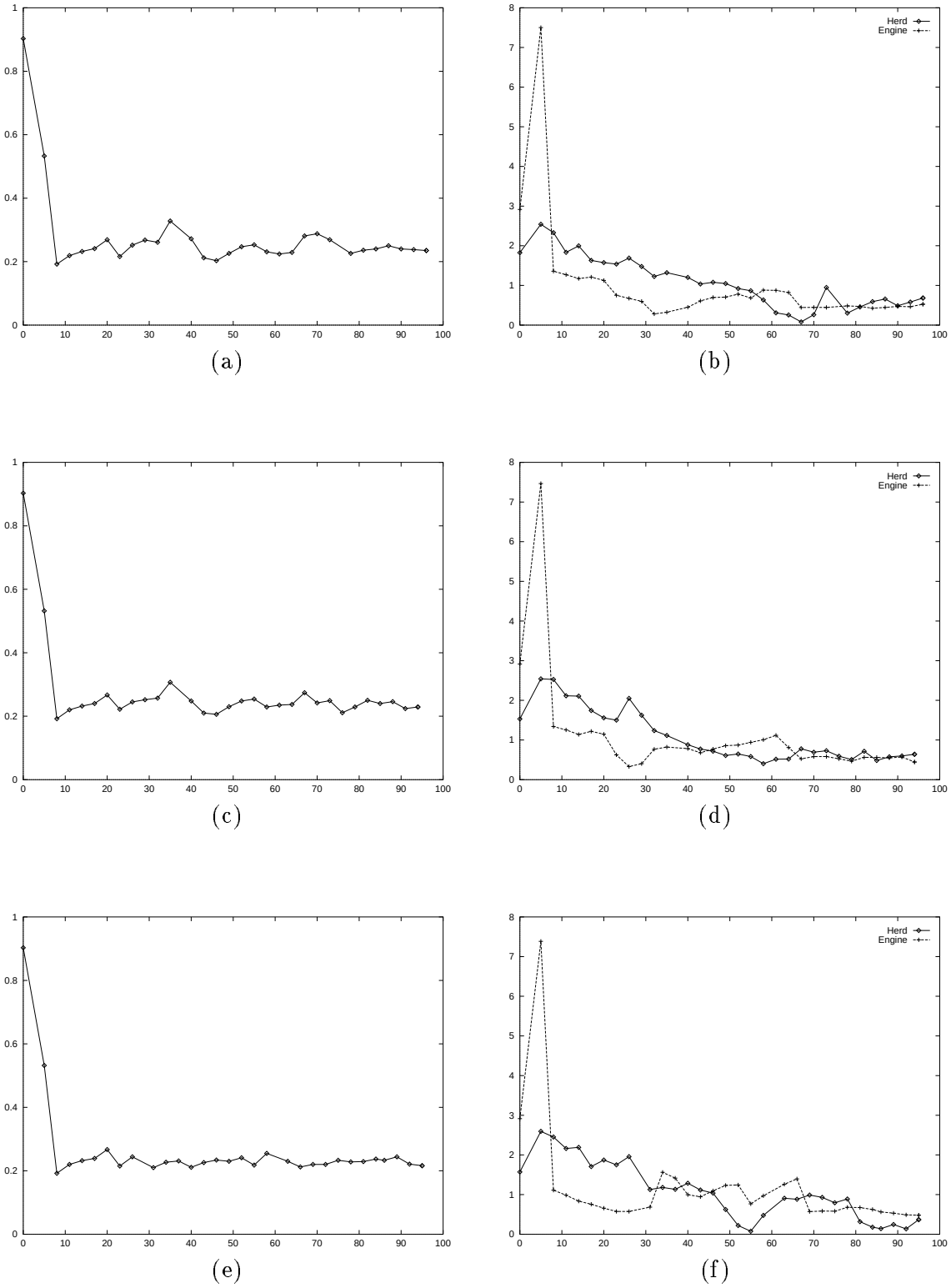


Figure 18: (a) The belief in the goal for the utility function $\frac{V}{rC}$. (b) The error for the engine and herd with a utility function of $\frac{V}{rC}$. (c) The belief in the goal for the utility function $\frac{V}{C}$. (d) The error for the engine and herd with a utility function of $\frac{V}{C}$. (e) The belief in the goal for the utility function V. (f) The error for the engine and herd with a utility function of V.

5.3.2 Set the Utility Function to V/C

The utility function for the second experiment is $\frac{V}{C}$; this utility function ignores the amount of space that must be searched in the proposed field of view. In actuality, this utility function is not all that different from $K(r) \times \frac{V}{rC}$ since K(r) and r are so similar (except for a constant multiple whose effect is nullified by changing the utility value of no_action). The difference between this utility function and $K(r) \times \frac{V}{rC}$ is the variance in values in the belief in the goal as seen in Fig. 18(c). Hence this utility function suffers from the same problem as $\frac{V}{rC}$.

As with $\frac{V}{rC}$, there is not much error in where *d*TEA-1 thinks the engine and herd are located (see Fig. 18(d)). Again, this is due to the effectiveness of the Alpha-Beta filter and confidence regions.

For this experiment the utility value of no_action was changed to 0.00195.

5.3.3 Set the Utility Function to V

This last utility function to be tried is also the most simple: V. This utility function not only ignores the size of the search space, it is unable to take notice of the cost of running the action, and focuses totally on the value the action would have on the system. The belief in the goal for this experiment, seen in Fig. 18(e), again has more variation then the utility function $K(r) \times \frac{V}{rC}$.

The error, as seen in Fig. 18(f), is good due to the Alpha-Beta filter and the well defined confidence regions.

For this experiment the utility value of no_action was changed to 0.003.

5.4 Adding more Plant Noise

In the last three experiments, only *d*TEA-1's belief varied. It was mentioned that a scene with less constrained objects would demonstrate how this slight difference would affect the data more. The following four experiments are run on the same scene, only now there is much more plant noise: *i.e.* the engine and the herd have a greater speed variance. During all of the experiments neither the herd nor the engine was completely lost; on several occasions the engine was lost for a short period of time, but it was always reacquired. Such error handling shows how well the expected area updating works.

The main problem with the utility functions that do not use the $K(r)$ term is how they allow variance in the belief. The high variance is due to the utility function recommending an inappropriate action. The utility function $K(r) \times \frac{V}{rC}$ keeps the variance in belief fairly low.

The results from the following experiments are the average of ten trial runs of thirty actions each.

5.4.1 Added Plant Noise using Utility Function $K(r) \times \frac{V}{rC}$

The first experiment, run on a high plant noise scene (three times as much plant noise for the engine and twice as much for the herd), uses the utility function that includes the $K(r)$ term. Fig. 19(a) shows the belief *d*TEA-1 has that the objects are not isolated. Notice how the variance is as low as the less noisy case shown in Fig. 17(a). There is low variance in the belief since $K(r) \times \frac{V}{rC}$ recommends executing the proper action.

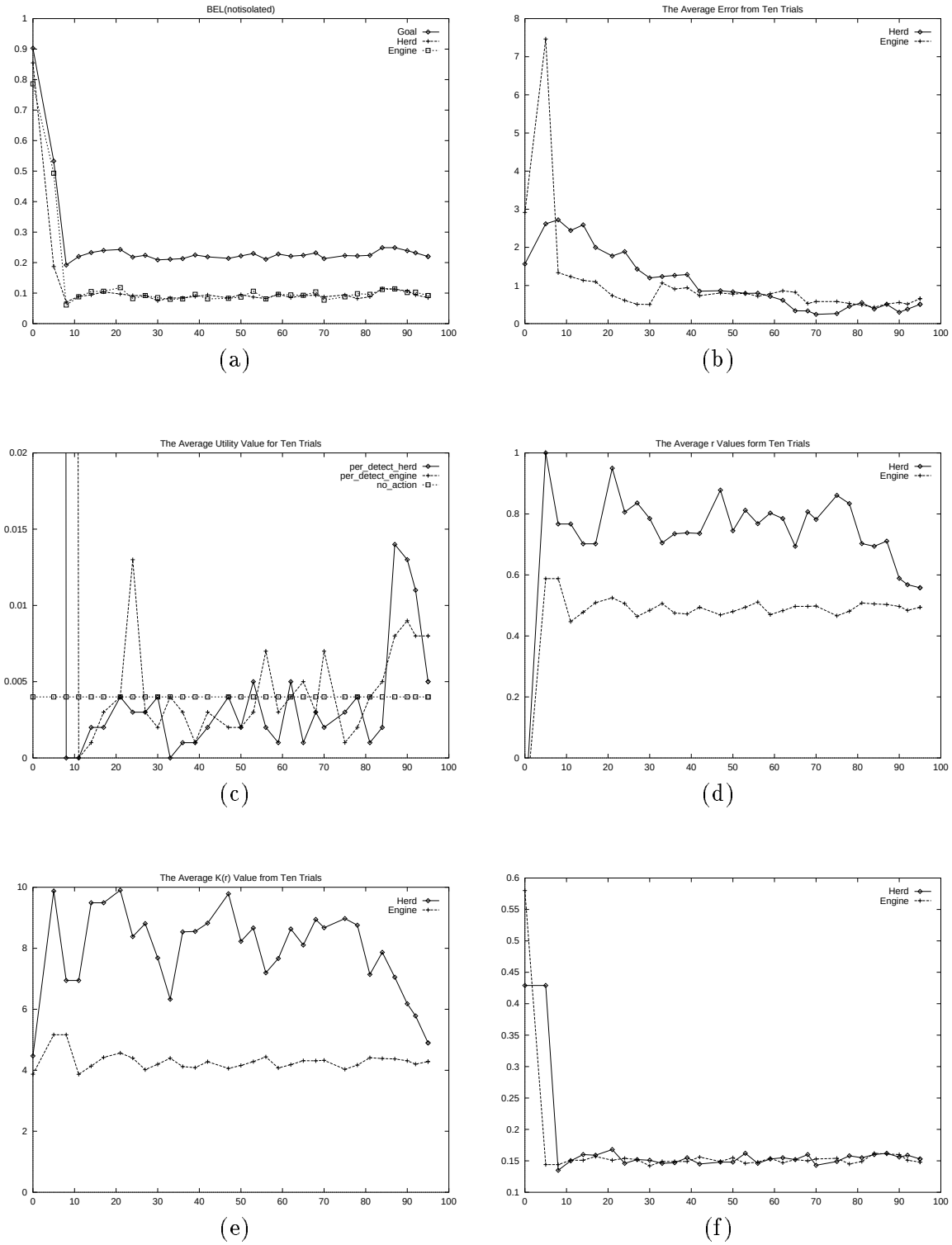


Figure 17: Average Results: (a) *d*TEA-1's average belief in the goal and in the positions of the engine and herd. (b) The average error in *d*TEA-1's estimate of the locations of the engine and the herd. (c) The average utility values for the three actions. (d) The average of the r values for the engine and the herd. (e) The average of the K(r) values for the engine and herd. (f) The average of the expected area package values for the engine and herd.

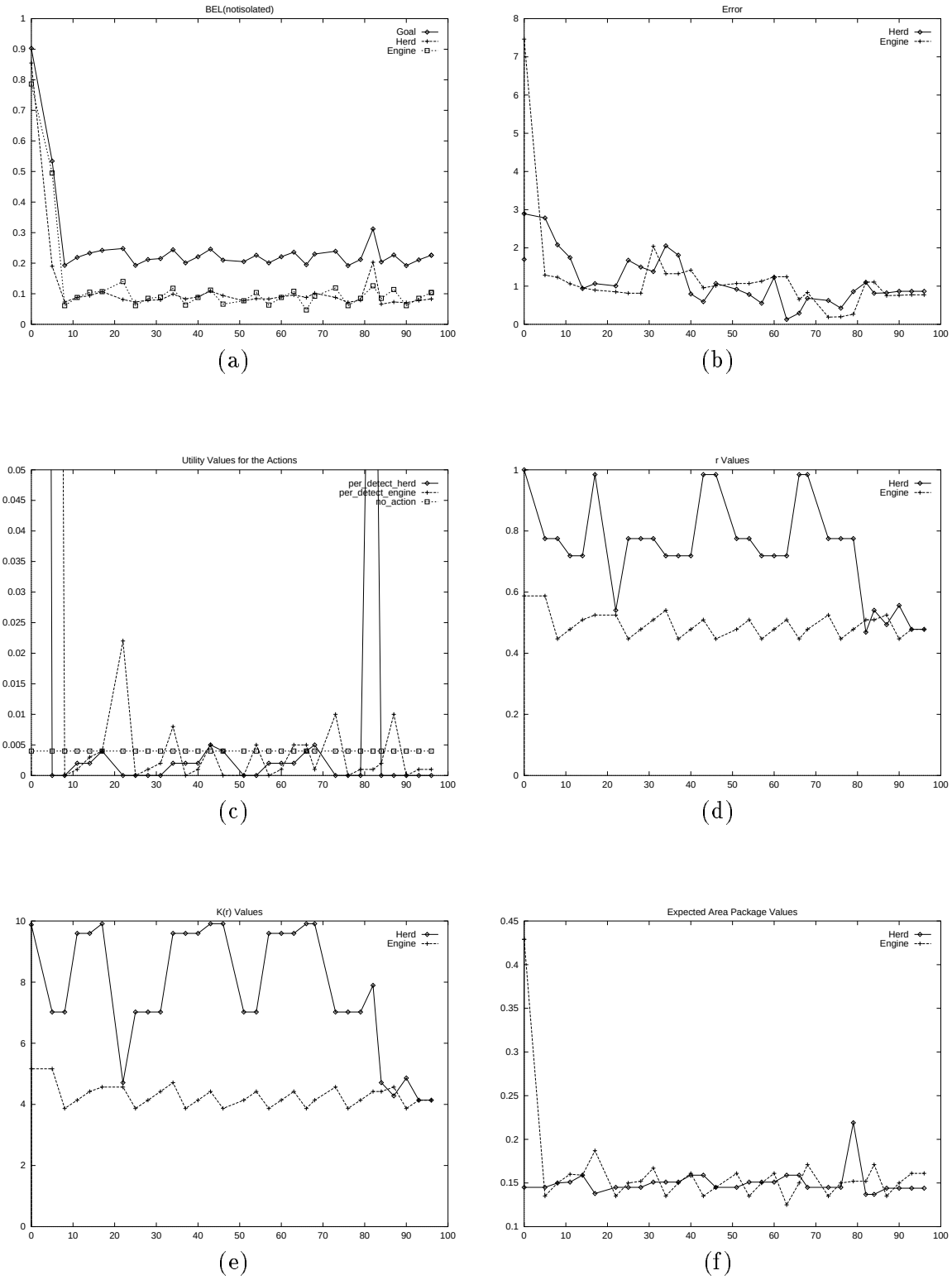


Figure 16: Good Case: (a) *d*TEA-1's belief in the the goal and in the positions of the engine and herd. (b) The distance from the actual to expected locations. (c) The utility values of the three actions. (d) The r values for the engine and herd. (e) The K(r) values for the engine and herd. (f) The expected area package values for the engine and herd.

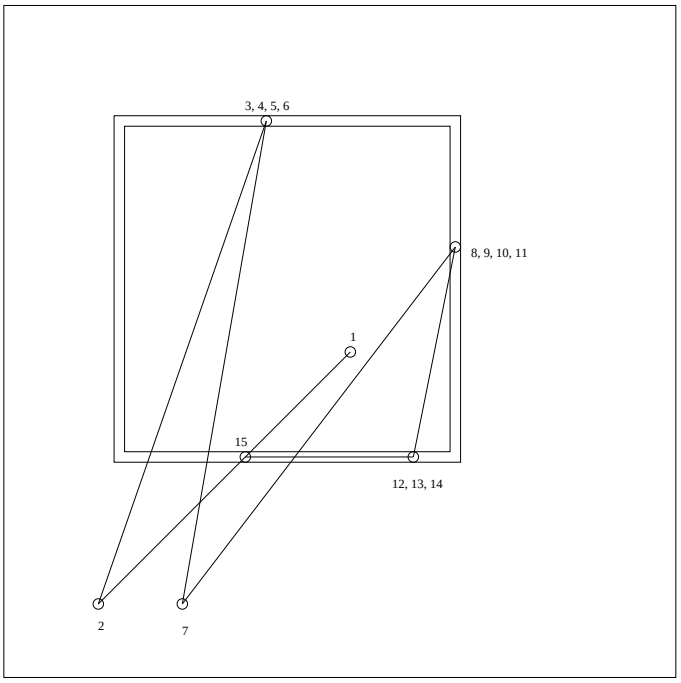


Figure 15: Good Case: location of the field of view during the first fifteen actions.

5.3 Changing the Way It Runs

This set of experiments focuses on changing an aspect of the way *d*TEA-1 gathers data. The four experiments involve using slightly different utility functions that do not use all the properties of $K(r) \times \frac{V}{rC}$, which have already been discussed. The results from the following four experiments are data averaged from ten trial runs of thirty actions each.

5.3.1 Set the Utility Function to V/rC

In this first experiment the utility function is set to $\frac{V}{rC}$; the only change is the absence of the K(r) function. The theoretical change, due to the lack of the K(r) factor, is that the engine will be over-emphasized (searched for more often than necessary) and the herd is de-emphasized (not searched for often enough). Fig. 18(a) shows the belief in the goal for this experiment. Even though the values stay fairly low, the high variance of the values would adversely effect the end results in a more difficult task including objects whose motion was not so structured.

The error in *d*TEA-1's location estimation (Fig. 18(b)) is as good as the utility value including $K(r)$, but this is due to the Alpha-Beta filter and confidence regions. Since this utility value allows *d*TEA-1 to lose belief in its knowledge of the objects' locations, objects with more randomness in motion would quickly be lost.

There was one other change to the system that had to be made to make this a fair experiment: the utility value of no_action was changed to 0.00275. This change made *d*TEA-1 look for the engine and herd with about the same frequency as the utility function $K(r) \times \frac{V}{rC}$ when the utility value for no_action was 0.004.

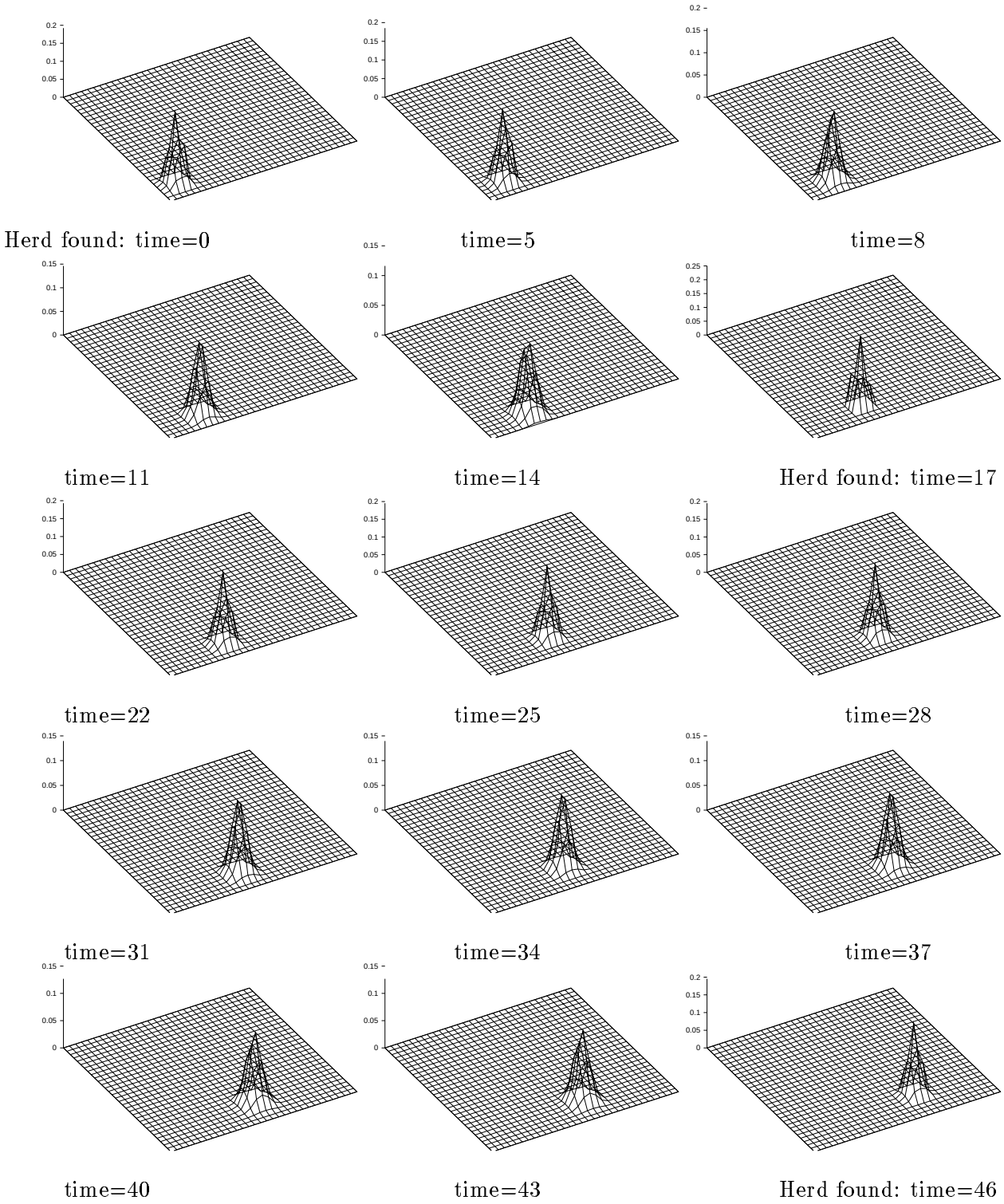


Figure 14: Good Case: sequence of expected area maps of the herd.

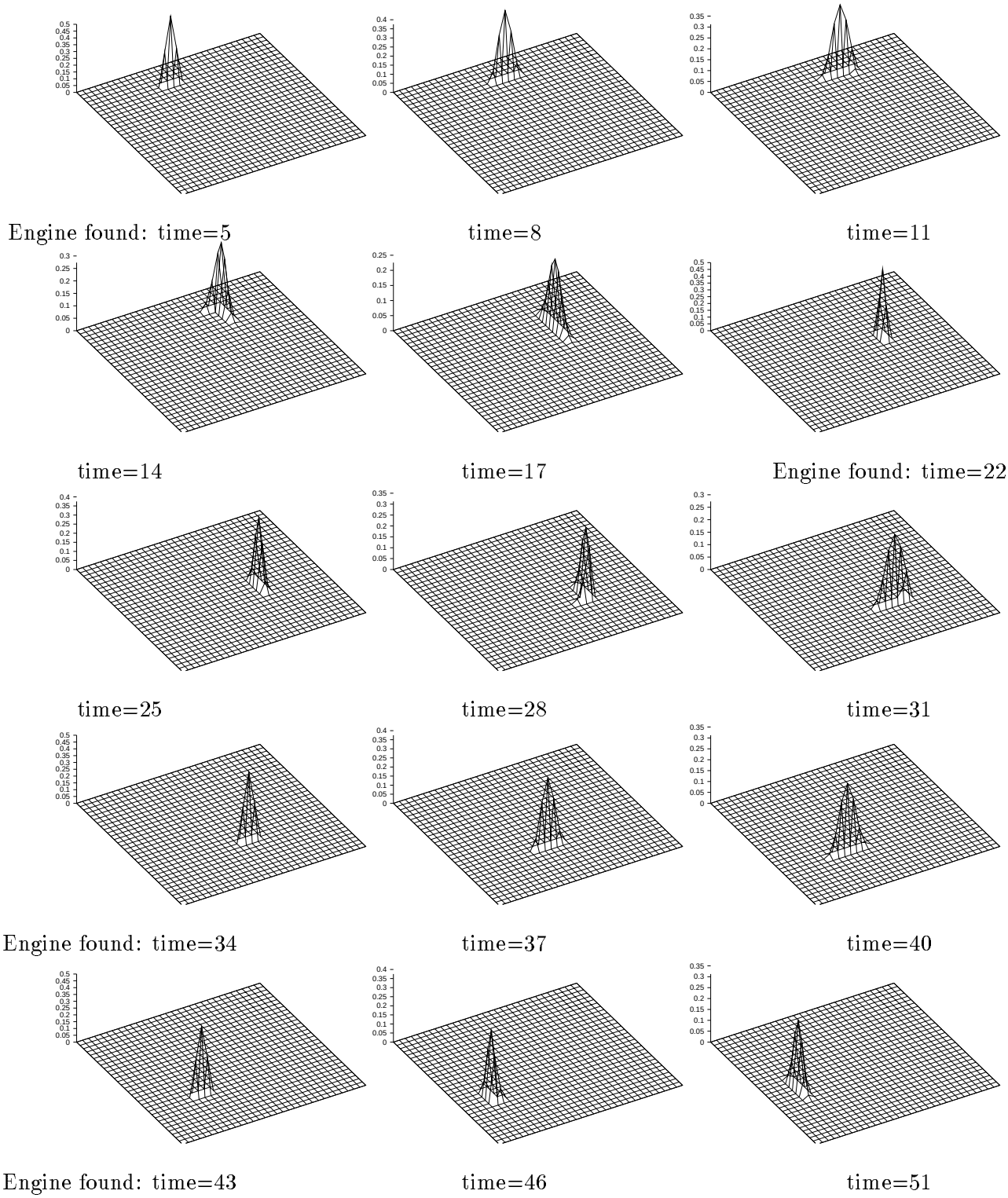


Figure 13: Good Case: sequence of expected area maps of the engine.

Figs. 13 and 14 show the expected area maps of the engine from time 8 through time 52 and of the herd from time 5 through time 49, respectively. Fig. 15 shows what part of the scene was searched during the first fifteen actions. The location marked 1 is the initial location of the field of view. From the initial field of view location, *d*TEA-1 decided to look for the herd, the engine, did nothing for three actions, *etc.*

After seeing when actions were run, how long they ran for, and where *d*TEA-1 actually looked, the next step is to see how well *d*TEA-1 believes it did. Fig. 16(a) is a graph of the belief that the system does not know the locations of the engine and herd. Notice how initially *d*TEA-1 has little knowledge of the scene, but after only a few time steps, it has a good idea of their locations and does not lose them. The belief in the goal is not the only interesting belief value. Fig. 16(a) also contains the belief values for both the engine and the herd, the sum of which yields the belief in the goal.

How well the system thinks it is doing is important, but when considering that data it is vital that some idea is given of how well it really is doing. Fig. 16(b) represents the distance from the object to *d*TEA-1's estimate of its position (*d*TEA-1's error). The error is in expected area coordinates which are the discrete points in the expected area array that represent locations in the scene. Since the length of the train track is 64 blocks, an error of one or two is on the order of 5%.

Fig. 16(c) shows the utility values for the actions at every time step. Initially, the utility values of per_detect_engine and per_detect_herd are very high since their expected areas include all locations in which they can ever exist. The largest valued action at each time step is the action that is run. Fig. 16(c) gives a better idea of when no_action is run in comparison with the other two actions.

Some other values of interest are the r values (Fig. 16(d)) which represent the percent of the field of view the expected area fills, K(r) values (Fig. 16(e)) which were created to counterbalance the effects of a growing expected area, and the expected area package values (Fig. 16(f)) which help *d*TEA-1 determine when to search for objects depending on the area and length of the expected area.

5.2 Average Results

This section show the average results from ten trial runs using all of the same parameters as in the last section. Fig. 17(a) is the average of the belief in the goal and is just as good as the perfect run in the last section only smoother. Fig. 17(a) also contains the average of the belief for both the engine and the herd. Notice how, on average, *d*TEA-1 strongly believes it knows the locations of both the engine and the herd.

The error in object location can be seen in Fig. 17(b). The error in *d*TEA-1 starts out to be a little more than in the good case, but as more data is filtered into the nets, the accuracy of *d*TEA-1's estimation greatly increases.

Fig. 17(c) shows the average utility values for the three actions. Toward the end of the trials it is interesting to notice how the utility of searching for the engine and the herd markedly increases, which can be attributed to the incorporation of noisy data from a few late trials in which the engine's location was not correctly determined.

Other values of interest are the average of the r value for the herd and engine (Fig. 17(d)), the average of the K(r) values for the herd and engine (Fig. 17(e)), and the average of the expected area package values for the herd and engine (Fig. 17(f)).

Time	Action
0	per_detect_herd
5	per_detect_engine
8	no_action
11	no_action
14	no_action
17	per_detect_herd
22	per_detect_engine
25	no_action
28	no_action
31	no_action
34	per_detect_engine
37	no_action
40	no_action
43	per_detect_engine
46	per_detect_herd
51	no_action

Table 1: Good Case: the first fifteen actions and when they were run.

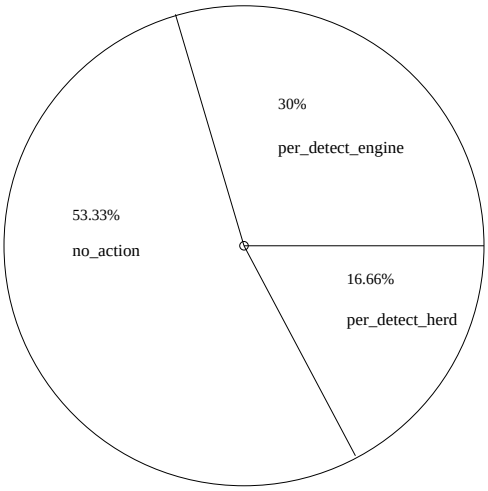


Figure 12: Good Case: percent each action was run for thirty actions.

the middle right; it also contains an engine moving around a square track in the clockwise direction. An example of the scene at time zero can be seen in Fig. 11.

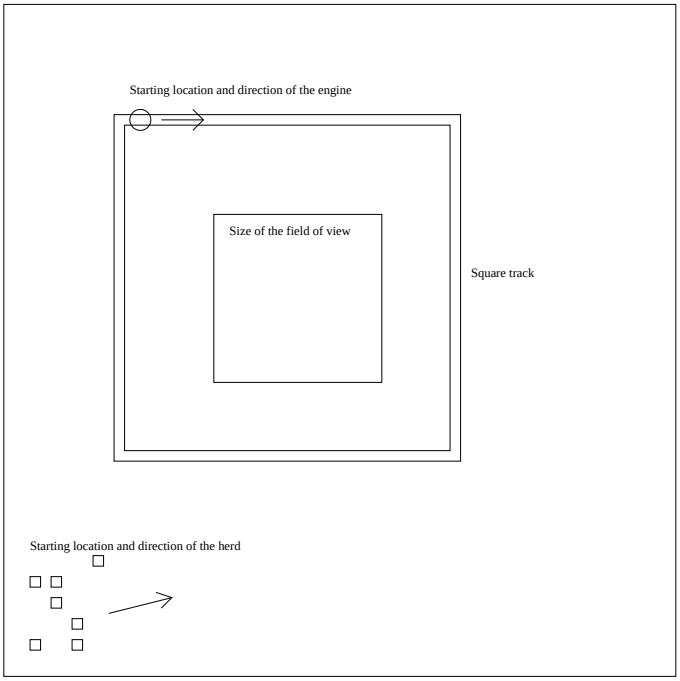


Figure 11: A sample scene at time zero.

The Task net shown in Fig. 9 calculates *d*TEA-1's belief in object locations. For the following experiments when a belief value is referred to, a high value implies the object's location is not isolated, while a low value implies *d*TEA-1 is more certain of the object's location.

A sequence of thirty actions was executed in each of the following trials. The actions include per_detect_engine, per_detect_herd, and no_action where *per* indicates a peripheral action (this task does not require foveal actions). Each action takes a variable amount of time to run according to how much of the field of view must be searched (except for no_action which takes a constant amount of time).

5.1 A Good Sample Run

The first experiment, known as *Good Case*, is an example of *d*TEA-1 monitoring the locations of both the engine and the herd without losing their positions. This trial consists of thirty actions. To get a feel for what *d*TEA-1 is doing the first fifteen actions and the time at which they are run is shown in Table 1.

Due to the frequent use of no_action, it is clear that the information in the system is not outdated to quickly. It too is important to notice how much time per_detect_herd takes in relation to per_detect_engine: per_detect_herd usually takes five time ticks while per_detect_engine takes three. The action execution time is dependent on the percent of the field of view that must be searched. Hence, the herd detection action takes more time than the engine detection action, because the expected area of the herd fills more of the field of view. The percentage of each action run for all thirty actions can be seen in Fig. 12.

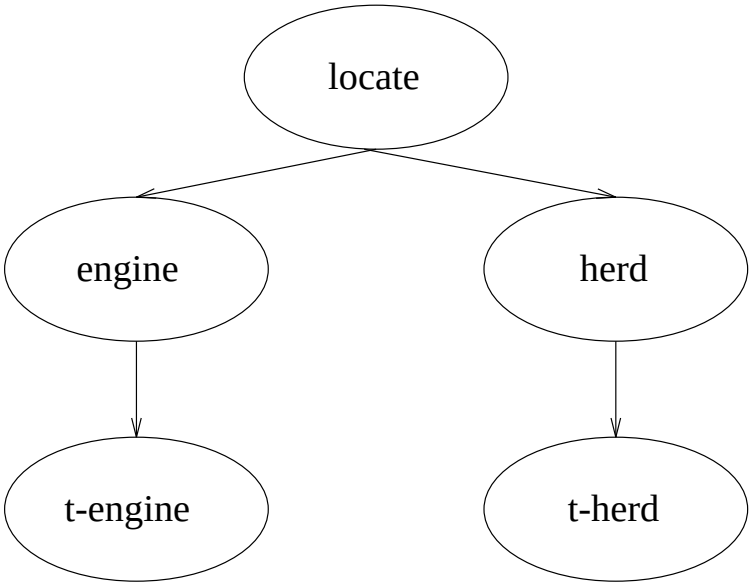


Figure 9: The Task net used by *d*TEA-1 when monitoring the locations of the engine and herd of cows.

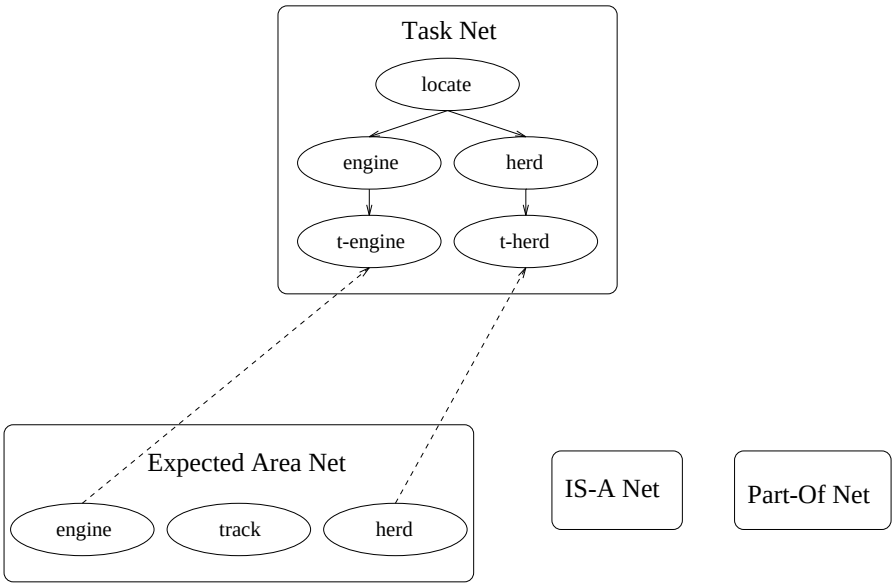


Figure 10: The package values connecting the Task net with the information stored in the other relevant networks.

4.4 Belief in Detection Action Results

Since dynamic parameters are incorporated into past data using an Alpha-Beta filter, we need initial values for detection action results to be filtered into. We assume the first run of a moving object's detection action returns error free results, which allows *d*TEA-1 to accept the data instead of filtering it. If the first run had error, then filtering later data into it would cause *d*TEA-1 to arrive at poor estimates. Also, if the object is incorrectly identified, these results could make finding the object impossible in the future. It should be clear that the initial error free detection action results are only possible in a simulated scene: if the scene were real and all data was acquired through visual sensors, a claim to such certainty would be impossible.

After the initial run of detection actions, detector noise is added to the results and there is a possibility that the object is incorrectly identified: *e.g.* the train is identified where no train exists. These kinds of results are what can be expected from real vision routines run on image data. Since both the train and the herd of cows have structured motion (the train must be on the track and maintains a nearly constant speed; the herd moves in one general direction and has little variation in speed) it is possible to distinguish a positive detection from a negative. *d*TEA-1 takes all detection action results and compares them to what the system believes the result should be. If the results fall within specified confidence regions, then *d*TEA-1 accepts the results and filters them into past data. If the results fall outside of the confidence regions, then *d*TEA-1 ignores the results and runs another action.

More specifically, when the herd detection action returns what is believed to be a positive match, the speed, direction, and location must be within a certain percent of what is expected, according to past detection. Similar to the herd, the speed and location of the train must be within a specified variance of the belief. Testing direction, however, is different. Since the location of the track is known, the detected direction of the train can be compared to the track's at any location. If the directions are off by a number of degrees above a specified threshold, then the results are rejected.

5 A Train Domain Task

*d*TEA-1 was given the task of maintaining the estimated positions and velocities of a train engine and herd of cows. The following subsections detail several experiments involving this task and demonstrate how *d*TEA-1 performed.

We assume *d*TEA-1 knows the starting locations of both the herd and the engine. Without this information *d*TEA-1 could spend a lot of time searching the large initial expected areas: the entire track for the engine and the field for the herd. This task emphasizes *d*TEA-1's ability to 1) differentiate between true and false detections, 2) filter out plant and measurement noise, 3) decide which action to run next according to current knowledge, and 4) track multiple targets.

This is a very simple task and has the Task net shown in Fig. 9. Fig. 10 gives an idea how the different networks communicate. The dotted lines in the graph are the package values that are sent to the Task net. In the case of this task, only information from the Expected-Area net is needed, but more difficult tasks could have similar connections between the Task net on one side and the rest of the nets on the other.

Following are the results from several experiments all running the object location task on the same scene. The scene includes a herd of cows that moves from the lower left corner of the scene to

be shown.

4.2 Uncertainty in Engine Location

Section 3.1.2 described how the one-dimensional loci array (representing the track) of a constrained object is convolved with the following error pdf.

0.25	0.5	0.25

This model of uncertainty has a maximum at its mean value, like the Gaussian, instead of uniformly distributing the uncertainty. This pdf was chosen since it emphasizes previous non-zero expected area array values.

The more uncertain *d*TEA-1 is about the location of the train, the more often the one-dimensional loci array is convolved with the error pdf. So, when there is more uncertainty, the expected area is essentially convolved with a larger pdf.

*d*TEA-1 has a simple way of determining how uncertain it is in detection action results: the faster an object moves, the more frequently the one-dimensional loci array is convolved with the error pdf. For example, the engine's one-dimensional loci array is usually convolved with its error pdf once every two time tics while the herd's expected area array is usually convolved with its error pdf once every five tics.

4.3 Uncertainty in Herd Location

The plant noise of the herd is directionally dependent, so the error pdf is broken down into four basis functions which are weighted and summed according to the direction of the herd. The four pdf's represent the directions 0°, 90°, 180°, and 270°. For example, if the herd has direction 56°, then the pdf's for 0° and 90° are combined using $(\sin 56°)pdf(90°) + (\cos 56°)pdf(0°)$ to make an error pdf that will be convolved with the two-dimensional expected area array of the herd. The four pdf's used are as follows:

0°

0.0574	0.01149	0.0
0.2873	0.5747	0.0
0.0574	0.01149	0.0

90°

0.0	0.0	0.0
0.01149	0.5747	0.01149
0.0574	0.2873	0.0574

180°

0.0	0.01149	0.0574
0.0	0.5747	0.2873
0.0	0.01149	0.0574

270°

0.0574	0.2873	0.0574
0.01149	0.5747	0.01149
0.0	0.0	0.0

As in the case of the train, the frequency of convolution with the error pdf is dependent on the speed of the herd.

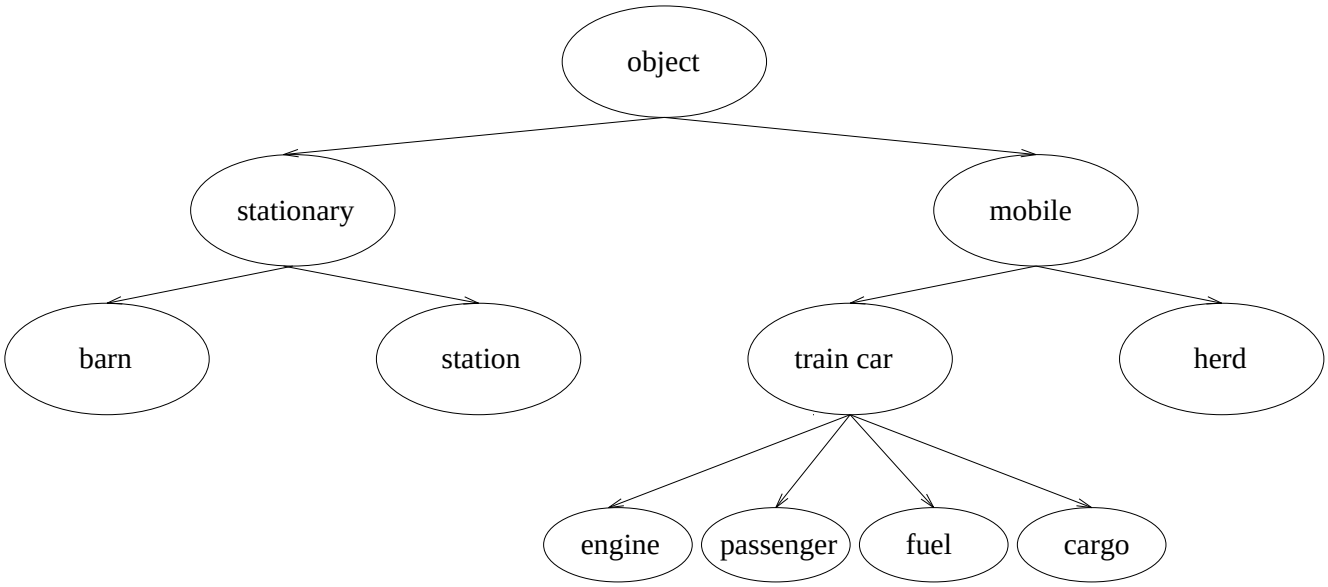


Figure 7: The Is-A net used by *d*TEA-1 in the train domain.

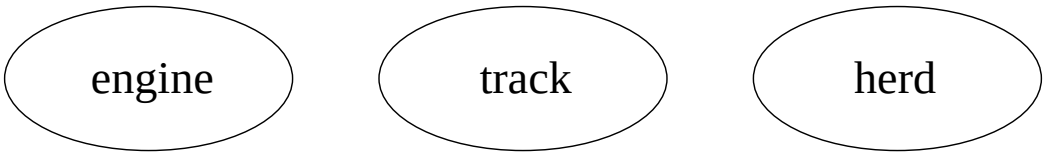


Figure 8: The Expected-Area net used by *d*TEA-1 in the train domain.

4.1 Bayesian Networks Used for the Train Domain

The set of Bayes nets used in the dynamic train domain is similar to the set used on static dinner table scenes. There is a Part-Of net, Is-A net, Expected-Area net, and Task net. The Part-Of net is used to keep track of all the parts of the scene (*e.g.* the train engine, boxcars, herd of cows, etc.) and the Is-A net is used to distinguish between different object classifications (*e.g.* the different kinds of boxcars that are pulled by the engine). A graph of the Part-Of net can be seen in Fig. 6 and the Is-A net in Fig. 7. The Part-Of net stipulates that there can be one train with at most five boxcars, one herd, at most five barns, and one station. These upper bounds can be expanded by adding nodes to the net. The Is-A net classifies train components as either engine, passenger car, fuel car, or cargo car. Again, more can be added and there can even be different classifications of barn simply by adding to the net.

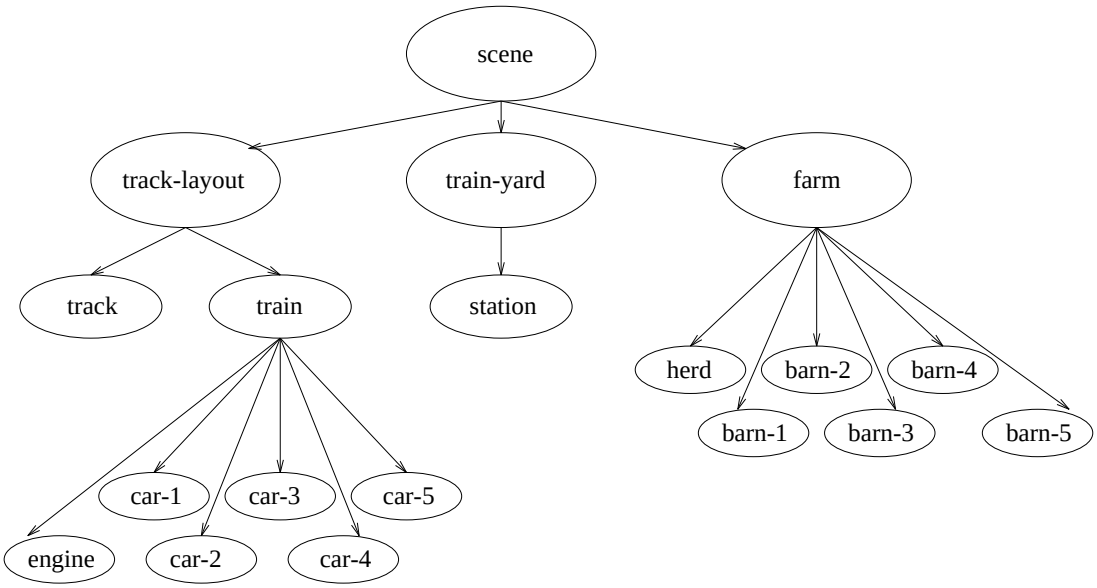


Figure 6: The Part-Of net used by *d*TEA-1 in the train domain.

The Expected-Area net is also used by TEA-1 in static scenes, but major changes were made to accommodate time. The process of expected area evolution has already been explained in detail. The Expected-Area net used can be seen in Fig. 8. It is a very simple network that only contains the objects that are in the scene used for the task (which will be mentioned later on), hence reducing the time of many calculations. The nodes are not joined so that the position of one object does not effect the position of the others; also, the task that is performed does not require any spatial relationships.

While the design of the Task net is essentially the same as the one used by static domain TEA-1, changes were made to allow for questions whose answers vary with time and hence might never be finally answered. For example, the place setting tasks are questions that will be answered with more certainty as more information about the scene is discovered. A task in the train domain could be as simple as keeping track of the locations of all the moving objects; notice how this task involves the degree of certainty in the location of objects at a certain time. Hence, *d*TEA-1 can never complete this task, it can only strive to obtain the most up-to-date information regarding object speed, direction, and location. After the sample task has been presented, the task net will

where V and C are the same as above. The new r value now represents the percent of the field of view that is filled by the expected area of the object (instead of the percent of the entire scene). This change was made since the action will only be searching the field of view and not the entire scene. $K(r)$ is a function of r that increases the utility value of an object with an expected area that is increasing in size. The $K(r)$ function can be seen in Fig. 5. An interpretation of $K(r)$ is as follows: as the size of the expected area of an object increases (as r grows), $K(r)$ simulates dTEA-1's desire to find that object. Hence, $K(r)$ does not allow dTEA-1 to fixate on an object it knows the location of (small r value), and it influences dTEA-1 to search for objects whose locations are not well known (large r value). The $K(r)$ value also ensures that objects with small expected area, and therefore small costs of searching, do not get precedence over objects that have not been located for a long time. This nonlinear utility is specialized to objects whose locations are to be monitored: other $K(r)$ functions could implement other systematic biases.

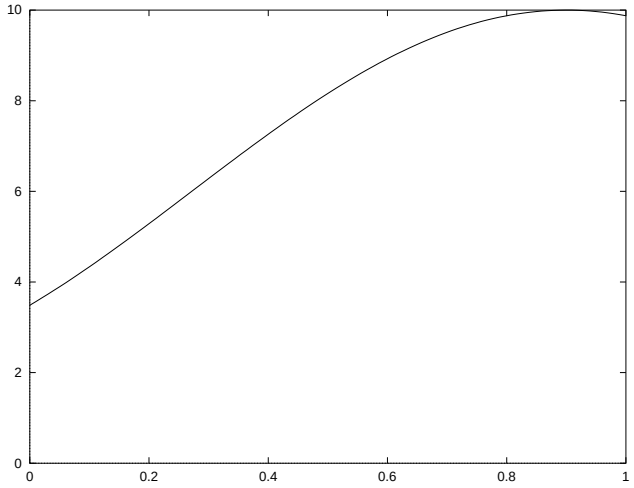


Figure 5: A graph of the $K(r)$ function; r is the x axis and K(r) is the y axis.

4 A Dynamic Domain: The Train Scene

So far major changes have been made in dTEA-1, but these changes are meaningless without a domain to work in and a task to complete in that domain. A train scene was chosen since it can have any number of moving and stationary objects that can add to the complexity of the scene: a train on a track, a train station, herds of cows, barns. Also, the train and the herd of cows capture the essence of constrained and unconstrained motion: *e.g.* a train must stay on the track and cows can wander, but only in a field. Such constraints are given to dTEA-1 so that spatial relations can be formed and a smaller search space envisioned. Thus a train scene contains representative forms of several general types of domain scene components.

The following subsections describe the domain specific changes to the system that allow dTEA-1 to solve train scene tasks. The domain changes made to dTEA-1 are task independent: hence, they apply to any task in any train scene.

Where $\bar{x}^T_{(i,j)} = (x_i - x_c, y_j - y_c)$, for x_c and y_c the x-y coordinates of the center of the surface in the expected area array, $BEL_{(i,j)}$ is the belief value at (x_i, y_j), and N is the number of rows or columns.

Once the 2×2 matrix has been calculated, the eigenvalues, λ_1 and λ_2, of the matrix are computed. Then the approximate lengths of the axes of the surface, l_1 and l_2, are calculated as follows:

$$l_1 = 2 \times \sqrt{\lambda_1}$$

$$l_2 = 2 \times \sqrt{\lambda_2}$$

The larger of l_1 and l_2 (the principal axis) is then used to find the percent of the length of the scene that is filled by the length of the expected area of of the object:

$$percent\_length = \frac{l_i}{N}$$

Where l_i is the larger of the two lengths and assuming the expected area map is a square array.

Since the expected area of unconstrained objects is generally wide and constrained objects is linear, the *Percent Area* method is usually used for constrained objects, and the *Percent Length* method for unconstrained objects.

3.4 New Action: No Operation

In some dynamic applications there is the possibility that the most cost-effective action is to do nothing for a time, therefore a No-op action was added. No-op is useful when tracking the locations of one or more objects. If their locations are well known, then No-op is called instead of running an unnecessary location detection action. *d*TEA-1 is now extended to contain such an action whose utility and time-duration is fixed. *d*TEA-1 implements No-op by allowing Mongoose to increment time and update the simulated scene by that amount of time.

3.5 A Slightly New Utility Value

A utility function is used by *d*TEA-1 to decide which action should be run next according to all current information about the objects. The utility function used by *d*TEA-1 has been changed to work with a dynamic domain. The original function used was $\frac{V}{rC}$ where V is the value of running the action, r is the percent of the scene the expected area of the object fills, and C is the cost of running the action on the entire field of view: basically, the utility function was value divided by cost. The problem with this utility function is that as an object moves, *d*TEA-1 is less sure of its location since the size of its expected area increases. With the original utility function, the larger the expected area, the lower the utility value; this is logical for static objects since finding some other object first might reduce the expected area of other objects. For moving objects, however, a growing expected area is a signal that *d*TEA-1 is losing precision in its knowledge about the object's location.

The new utility function includes a new factor, one that compensates for the shrinking expected area problem, and a slightly different r value. The new function looks like the following: $K(r)\frac{V}{rC}$

An important part of completing tasks on a dynamic scene is to keep track of the locations of the moving objects. A new kind of package was required, from the Expected-Area net to the Task net, to effectively monitor object positions. This new *relocate* package transmits what percent of the scene the object's expected area fills. The percent of the scene the expected area fills relates to how much of the expected area could fit inside *d*TEA-1's field of view: *i.e.*, if a large percent of the scene is filled by an object's expected area, then the entire expected area of the object will not fit into *d*TEA-1's field of view.

The size of the expected area of moving objects in relation to the size of the field of view is needed to determine whether the location of the object is well known or not. The relocate package delivers the system's knowledge about the size of the expected area of the moving objects to the Task net. If the location of the object has become poorly defined, due to uncertainty, and the task requires precise knowledge about the object's location, then *d*TEA-1 will look for the object in hopes of re-acquiring it and updating the system's model of the object. The relocate package allows *d*TEA-1 to search for a moving object before the search space becomes larger than a field of view. For example, suppose *d*TEA-1 is performing a task on a scene with several moving objects, one of them being a mobile robot. Suppose *d*TEA-1 has just located the robot and starts to search for another object in the scene; as time goes on, *d*TEA-1 becomes less sure of the location of the robot since it might have changed direction or speed, or due to the compounding of plant and measurement noise of the previous robot detection action; hence, the expected area of the robot grows to reflect the uncertainty of *d*TEA-1. After a while the expected area of the robot fills the size of *d*TEA-1's field of view. If reliable information about the robot's location is needed, then *d*TEA-1 should search for the robot to verify its location, speed, and direction before the search space becomes large.

Although percent of the scene the expected area fills is one measure of the uncertainty in an object's position, another aspect is the number of fields of view necessary to cover the area. Hence, the length of an object's expected area is needed. Two methods are used to calculate the size of the expected area in relation to the size of the field of view and the worse of the two results is used as the relocate package value. Following are descriptions of both methods.

3.3.1 Percent Area

The first technique is called the *Percent Area* method, and is the simpler of the two. The *Percent Area* method involves finding the number of elements in the expected area array that contain a non-zero value and dividing by the total number of elements in the array. This value is the percent of the scene that might contain the object.

3.3.2 Percent Length

The second technique is the *Percent Length* method which draws on the best eigenvector fit of a set of data points as described in [Duda and Hart, 1973]. The expected area array can be considered a surface, where the values in the array are the z-axis values. The *Percent Length* method computes the principal axes of the distribution represented by the expected area. The first step is to create a 2×2 matrix using the following equation.

$$\sum_{i=0}^{N} \sum_{j=0}^{N} (\bar{x}_{(i,j)} \bar{x}_{(i,j)}^{T} BEL_{(i,j)})$$

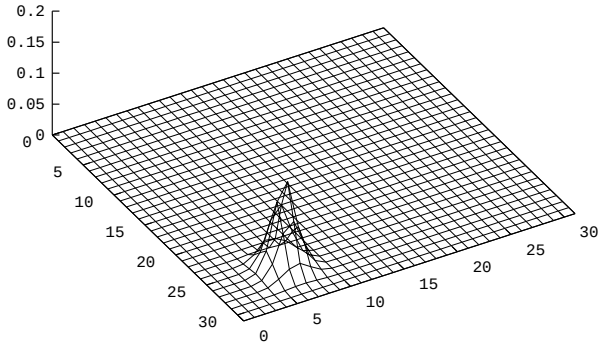


Figure 4: The expected area of an aggregation.

object is unconstrained with random direction and speed, it is almost impossible to discount action evidence since any object motion is possible.

Once data has been accepted as reliable, should it replace the data that was in the system? This would be a bad idea since there is both plant and measurement noise: variations in and slightly misdetected location, speed, direction, *etc.* In order to keep some of the past knowledge in the system, an Alpha-Beta filter is used to average the new data in with the old.

The main equation for the Alpha-Beta filter is

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k|k) + \begin{bmatrix} \alpha \\ \beta/T \end{bmatrix} [\mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k)]$$

where k is time and where $\hat{\mathbf{x}}$, $\mathbf{z}$, and $\hat{\mathbf{z}}$ are column vectors containing position and velocity, and represent filtered data, observed results, and expected results, respectively. The parameter $(k+1|k)$ means at time $k+1$ given what is known at time k. The time since the last detection, T, is important since speed is acquired from the current and past location values. Finally, $\alpha \in [0,1]$ and $\beta \in [0,1]$ are combinations of the ratio of plant and measurement noise. Effectively, past results are emphasized if there is a lot of measurement noise, and the current observed results are emphasized if there is a lot of plant noise. For a more through analysis of confidence regions and the Alpha-Beta filter see [Bar-Shalom and Fortmann, 1988].

3.3 Enhanced Packages

Since *d*TEA-1 implements multiple Bayes nets, there is a need for communication of information between networks. Packages are blocks of information that are collected from one or more networks and sent to another: generally communication is between the Task net on one end and the rest of the nets on the other [Rimey, 1993; Rimey and Brown, 1993].

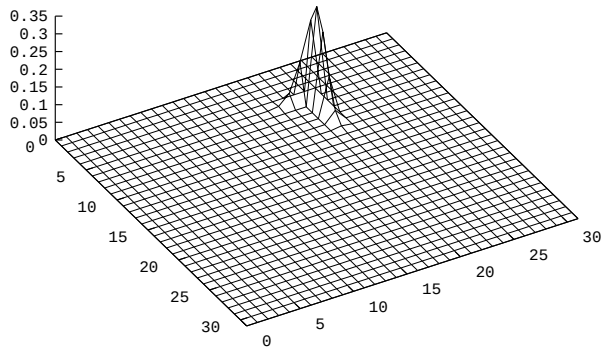


Figure 3: The expected area of a constrained object.

object reaches the edge of the scene, it continues trying to "walk through a wall". This behavior at boundaries can easily be modified if object motion allows departure from the scene.

After the expected area values of the aggregation have been translated according to speed, direction, and time, the expected area array is convolved with the error pdf to account for plant and measurement noise. The pdf can be anything including a Gaussian or even the weighted sum of several different pdf's.

An example of the expected area of an aggregation can be seen in Fig. 4.

3.2 Using Detection Action Results

*d*TEA-1's ability to detect groups of objects, such as cows, stems from the use of a simulated scene instead of a real scene. The detector for an aggregation of objects is easier to simulate than to provide in real life; *d*TEA-1's simulated detector returns the number of objects in the field of view, their center of gravity, average height, width, speed, and direction, all with measurement noise.

The information returned by an action that detects an object or a group of objects, either will be used to estimate the current dynamic state of the object or will be disregarded as a false match. Before new data are used to propagate estimates into the net, there are several tests that they must pass; if the action results fall outside of confidence regions (or gates) determined by past detection data, then the results are rejected by the system. By testing new data it is possible not only to reject an incorrect match, but to reject an error-infected positive match.

The confidence regions are determined statistically. The mean value in the confidence region is set equal to the detection action's expected result. The size of the confidence region depends on the amount of variation in the expected returned value. It should be noted that the confidence regions are not determined at run time, they are calculated by other means and given to *d*TEA-1. Consider the following example: a constrained object that has little or no speed variations can have confidence regions that will allow *d*TEA-1 to disregard most incorrect detections. If, however, the

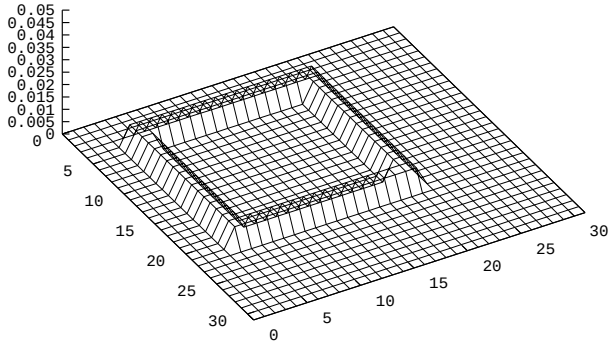


Figure 2: The expected area of a one-dimensional locus.

the one-dimensional array holds a locus location and a probability of the constrained object being in that location: the array holds the probability density function of the location of the object. Updating the probability density function through time shifts the probabilities in the array by the distance the object is expected to have moved. The velocity of such a one-dimensional system is expressed in terms of a preferred direction along the locus. Once the probabilities in the array have been translated, a convolution of the array with the error pdf is performed; the error pdf is the sum of the plant noise and measurement noise pdf's. Next, the values are copied to the object's expected area array. A 3-dimensional plot of the expected area of an object constrained to the one-dimensional locus in Fig. 2 can be seen in Fig. 3.

Constrained one-dimensional objects can be isolated or be part of a group of objects composed of a leader and several dependent objects that precede or follow it on the locus. For instance, a convoy or a train with rolling stock is modeled as one leader object and a set of related objects whose position on the locus is fixed relative to the leader object.

3.1.3 Updating the Expected Area of an Unconstrained Group of Objects

Updating the expected area of single two-dimensional dynamic objects or associated aggregations of individuals, such as a flock of birds, herd of cows, or cooperating group of vehicles off the road, is similar to updating the expected area of an object constrained to a one-dimensional locus. The difference is that all expected area translation and convolution is done in a two-dimensional rather than a one-dimensional representation.

The first step is to translate all of the non-zero expected area probabilities of the aggregation to a location derived from its most recently detected speed and direction. Boundary conditions must be enforced: when an object hits the boundary of the scene, it is no longer able to move. For this reason, when the expected area values *should* be moved out of the scene, they are simply added to the values that are already at the edge of the scene. This gives the effect of a pile up: as an

of an object has been detected (with a certain amount of error), then that expected location should be translated according to its direction and speed as would the object's actual location. An object's expected location is updated in a two-dimensional expected area array M, where every element of M is a location in the scene and the value of each element is the probability that object O is at that location (the expected area arrays of all objects reside in the Expected-Area net). Due to plant and measurement noise, a group of adjacent elements in the array will have non-zero values. Consider the expected area array of a moving object O at time t, $M_O(t)$. Once the speed and direction of O along with its location have been detected, a simple translation of every non-zero element in $M_O(t)$ can be made to calculate the array $M_O(t+1)$ at time $t+1$. Predicting objects' locations eases the future search for those objects when their exact locations are needed.

Dynamic objects exhibit "plant noise": their velocities or accelerations may be subject to random variations described by the plant noise parameters. For instance, a "constant-velocity" object may experience a random acceleration at each simulated update, drawn from a zero-mean Gaussian distribution. Likewise the sensors for position and velocity are subject to "measurement noise" that perturbs their outputs.

Whenever an object's location, speed, and direction have been visually acquired, the values should be considered estimates since there is plant and measurement noise. Thus, if an object O has been detected at time t, there is no guarantee that the speed and direction of O will be the same at time $t+r$ for $r > 0$. Hence, the expected area array $M_O(t+r)$ should reflect the effects of plant and measurement noise. The cumulative effects of these errors is represented by enlarging the possible location area of the object within the array M, by convolving it at each time step with the error probability density function (pdf) for plant noise. When an object is located, expected area is likewise limited by the measurement noise pdf. While convolution increases the uncertainty of the exact location of the object, it also increases the system's knowledge of when to perform a new search, which will re-establish contact with an object whose speed and direction may have changed.

3.1.1 Locating All One-Dimensional Loci

Before dTEA-1 starts performing actions to complete a task, it is given the locations of all one-dimensional loci in the scene: thus the loci are part of domain knowledge. Buy giving dTEA-1 information about all one-dimensional loci, the expected areas of constrained objects are more reliable since there is no error in detected loci locations. The area which is covered by a one-dimensional locus is potentially a large area to search, but it is much smaller than the entire scene. A surface plot of a one-dimensional locus can be seen in Fig. 2.

In general, when an object's motion is constrained to certain directions, there are often "switch point" locations where the object can move in one of a number of different directions: *e.g.*, a highway intersection, a fork in a path, a switch on a train track. Future versions of Mongoose and dTEA-1 will support this feature.

3.1.2 Updating the Expected Area of Objects Constrained to One-Dimensional Loci

Any moving object that is constrained to a one-dimensional locus has its location probabilities manipulated in a one-dimensional array representing the one-dimensional locus before those values are placed in the two-dimensional expected area array representing the entire scene. Each element in

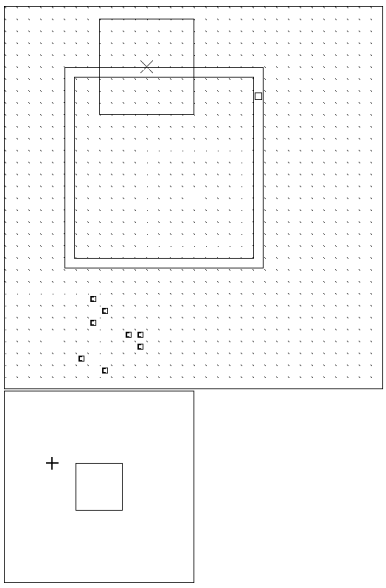


Figure 1: An example of a dynamic scene produced by Mongoose.

and unconstrained individuals and aggregations.

3.1 Updating the Expected Area Net through Time

The Expected-Area net is one of the Bayes nets used to constrain or estimate objects' locations according to their spatial relationships to other objects [Rimey and Brown, 1992]. The scenes viewed by *d*TEA-1 are much larger than the field of view and expected area constraints make vision practical. An example stems from the original task of TEA-1: answering questions about dinner tables. If the location of a plate has been visually verified, then according to the supplied spatial relationships the location of a fork in the scene is more highly constrained. Another example can be derived from a scene including moving objects: if the location of a one-dimensional locus, such as a street, is known, the search space for an automobile is greatly reduced. Spatial relationships and prior knowledge of objects' expected locations reduces future search times.

Three general types of objects have been added to *d*TEA-1 to make it a time-dependent system. The first type of object does not move, but is needed by some moving objects: a one-dimensional locus with switching points (switching points are not implemented in this version of *d*TEA-1). For moving objects that are constrained to paths, roads, tracks, or known trajectories, this one-dimensional locus implements the constraints, and provides an initial guess about the objects' locations. The second type of object is constrained to a one-dimensional locus: *e.g.* a train, a truck, or a riverboat. The third type of object is an unconstrained dynamic object that is able to wander in any direction within the two-dimensional scene. A dynamic object is characterized by all the object features of static objects (shape, *etc.*) but also by one- or two-dimensional velocity and acceleration information and plant noise parameters. The one-dimensional velocities are with respect to the one-dimensional loci (if the object is constrained), and thus translate to two-dimensional velocities for a given location on the locus.

Moving objects require repetitively updated expected areas; for static objects, the expected area is unchanged when only time has been updated. For example if the location, speed, and direction

1 Introduction

TEA-1 is a selective vision system that uses Bayes nets [Pearl, 1988] for representation and benefit-cost analysis for control of visual and nonvisual actions. TEA-1 solves T-world problems, a class of problems involving static two-dimensional scenes. For example, TEA-1 has been demonstrated answering questions about scenes of dinner tables. For details about TEA-1 see [Rimey and Brown, 1993; Rimey, 1993; Rimey and Brown, 1992].

This paper presents *d*TEA-1, an extension to TEA-1 that allow tasks to be performed on dynamic scenes. The main contributions of this work are management and use of information about dynamic parameters of objects and uncertainties associated with them. The domain used to test *d*TEA-1 is a train domain which includes trains, train tracks, train stations, barns, and herds of cows. Results from the task, "keep track of the locations of the train and herd of cows," are presented.

One of the first Bayes net systems used with computer vision was reported in [Levitt *et al.*, 1989]. Also, a Bayes net system was created by Dean to help a mobile robot plan how to locate a static object with several constraints such as having to find it in a certain amount of time [Dean and Kirman, 1992]. These systems demonstrate the usefulness of Bayes nets in different applications.

The *d*TEA-1 system uses multiple Bayes nets to store the information discovered in the scene. Some of the nets used by *d*TEA-1 include a Task net to calculate the belief in the task, an Expected-Area net used to keep track of object positions in the scene, a Part-Of net to monitor the objects in the scene, and an Is-A net to classify the objects into different categories. These four nets are used together to answer questions about or perform tasks on the scene given to *d*TEA-1.

2 Simulated Scenes using Mongoose

Mongoose [von Kaenel, 1992] is a simulator for static and dynamic two-dimensional scenes, which are projected onto a two-dimensional simulated image. Configuration files describe object properties in general and the properties of object instances in a particular simulated scene. Mongoose has facilities for creating sets of objects with random property values (number of objects, placement, etc.). Using X Windows, Mongoose provides facilities for a graphical display of the scene including moving objects whose positions are discretely updated through time. The simulator operates in stand-alone mode, or it can provide input to *d*TEA-1. *d*TEA-1's interface to Mongoose is identical to the interface to the vision laboratory hardware, and is accomplished with identical subroutine calls.

The top window in Fig. 1 is an example of a train scene simulated by Mongoose. The double line square represents a train track and the small square on it represents a train engine (without trailing boxcars). The group of squares at the bottom left is a herd of cows. The large square with central x represents the camera's field of view. The window at the bottom left specifies where in the field of view an object was located by *d*TEA-1 (if a detection action was run).

3 Additions in *d*TEA-1 to Accommodate a Dynamic Domain

The following subsections describe the new features in *d*TEA-1 that enable it to operate in dynamic domains which include moving objects such as those constrained to 1-dimensional loci

Goal-Oriented Dynamic Vision

Peter A. von Kaenel, Christopher M. Brown, and Raymond D. Rimey

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 466

August 1993

Abstract

TEA-1 is a selective vision system that uses Bayes nets for representation and benefit-cost analysis for control of visual and nonvisual actions. TEA-1 solves T-world problems, a class of problems involving static two-dimensional scenes. For example, TEA-1 has been demonstrated to answer questions about scenes of dinner tables.

This paper presents *d*TEA-1, an extension to TEA-1 that allow tasks to be performed on dynamic scenes. Currently, *d*TEA-1 successfully performs a task on a simulated train scene. The objects in the scene include a train on a track and a herd of cows, but the domain may be extended to include static objects and other classifications of moving objects. The task is to keep track of the locations of objects, and the system intelligently allocates its effort to keep uncertainty and cost to a minimum.

This material is based upon work supported by the National Science Foundation under the REU program (Grant numbers CDA-8822724 and IRI-8920771) and DARPA Contract number MDA972-92-J-1012. The Government has certain rights in this material.