# Computational Perceptual Attention

**Micheal Hewett**

**Dissertation**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Doctor of Philosophy**

**The University of Texas at Austin**

**October 20, 2000**

# Computational Perceptual Attention

**Approved by**
**Dissertation Committee:**

Benjamin J. Kuipers, Supervisor

Barbara Hayes-Roth

Raymond Mooney

Gordon Novak, Jr.

Bruce Porter

# Abstract

This dissertation describes CPA, a general-purpose mechanism for expressing and implementing attention policies that control the allocation of resources among sensory processing tasks in a robot or other advanced intelligent system. A wide variety of attention policies can be expressed in this mechanism, which also supports soft real-time constraints on perceptual processing. Intelligent systems can become inundated with data, resulting in perceptual overload and a consequent inability to formulate a timely or appropriate response.

Perceptual overload is often modulated by a perceptual attention mechanism that filters and prioritizes incoming data. Most existing attention mechanisms are tailored to the specific task the system is performing. A general-purpose attention mechanism must have a task-independent interface for controlling attention; support a heterogeneous set of sensors; support heterogeneous methods for processing sensor data; and support real-time throughput constraints. The CPA is a general-purpose attention mechanism that supports multimodal perceptual attention. Using it, an intelligent system can enact and control a variety of attention policies for any type or combination of sensor or sensor data.

An intelligent system dynamically creates multiple heterogeneous perception tasks in accord with behavioral goals and installs them in the CPA. The CPA supports two general categories of perception tasks: detectors, which do not retain information between perception cycles; and trackers, which do. Perception tasks are prioritized using an attention policy and are executed using a priority-based scheduler. A wide range of attention policies can be expressed in this mechanism, including policies that dynamically modify perception priorities, policies in which emergency input overrides normal perception processing, and policies that dynamically change the level of resistance to perceptual distractions.

Results show that perception intervals as short as 100 milliseconds can be achieved with a five-sensor robot under a variety of attention policies. Analysis of the system's performance under perceptual load shows that qualitatively different attention policies can be realized in the attention mechanism. We show that intelligent systems can use the CPA to implement the four primary characteristics of human perceptual attention: selective attention, sustained attention, divided attention, and top-down control.

# Chapter 1: Introduction

This dissertation describes a general-purpose mechanism for expressing and implementing attention policies that control the allocation of resources among sensory processing tasks in a robot or other advanced intelligent system. A wide variety of attention policies can be expressed in this mechanism, which also supports soft real-time constraints on perceptual processing.

Advances in sensor technology have enabled the construction of robots and other intelligent systems that utilize a large number of sensors. These sensors produce a wide variety of data about the physical world, often at very high data rates. Some of the systems need to respond to inputs within a very short period of time. However, they can become inundated with data, resulting in perceptual overload and a consequent inability to formulate a timely or appropriate response.

In many systems, perceptual overload is modulated by a perceptual attention mechanism that filters and prioritizes incoming data. The most common mechanism is one that spatially filters visual data by removing portions of the input located outside a spatial focus of attention. This can greatly reduce the amount of processing needed to produce a response, but introduces the possibility of ignoring relevant input. A few systems use mechanisms based on visual features such as color and shape, or non-visual features such as numeric readings. Most existing attention mechanisms are tailored to the specific task the system is performing.

A general-purpose attention mechanism has four major requirements: it must have a task-independent interface for controlling attention; it must support a heterogeneous set of sensors; it must support heterogeneous methods for processing sensor data; and it must support real-time throughput constraints. This dissertation introduces the CPA, a general-purpose attention mechanism that supports multimodal perceptual attention. Using it, an intelligent system can enact and control a variety of attention policies for any type or combination of sensor or sensor data.

An intelligent system using the CPA creates multiple heterogeneous perception tasks in response to behavioral goals and installs them in the CPA. The set of perception tasks can dynamically change as behavioral goals change. The CPA supports two general categories of perception tasks: detectors, which do not retain information between perception cycles; and trackers, which do. Each attention policy controls the length of the perception interval, the allocation of perception time between the sets of detectors and trackers, and the prioritization of sensors, detectors and trackers.

Perception tasks are prioritized using parameters in the attention policy and are then executed using a priority-based scheduler. The scheduler executes as many perception tasks as it can before the perception interval expires. This behavior can result in starvation of low priority tasks, in contrast to operating system schedulers which typically provide guarantees of fairness and liveness.

A wide range of attention policies can be expressed in this mechanism, including policies that dynamically modify perception priorities, policies in which emergency input overrides normal perception processing, and policies that dynamically change the level of resistance to perceptual distractions.

We evaluate the attention mechanism in a robot navigation system that requires high-speed perceptual throughput in order to effectively control its motion. The robot has five input streams that provide a mixture of numeric and symbolic data. The quality of perception is measured using a perception quality metric that is a function of both the quantity and quality of the input. Results show that perception intervals as short as 100 milliseconds can be achieved under a variety of attention policies. Analysis of the system's performance under perceptual load shows that qualitatively different attention policies can be realized in the attention mechanism.

We show that intelligent systems can use the attention mechanism to implement the four primary characteristics of human perceptual attention: selective attention, sustained attention, divided attention, and top-down control.

Future work will formalize constraints on the behavior of attention policies, incorporate hard real-time scheduling methods for more precise time management, define classes of domain-specific attention policies and explore methods for automatically learning effective attention policies.

## 1.1 ATTENTION IN PEOPLE AND ROBOTS

In people, attention provides two main functions. First, it prevents perceptual overload. The human senses take in far more information than the human brain can process in time for it to be useful. Attempting to attend to multiple inputs at the same time causes interference and performance degradation [Posner and DiGirolamo, 1998]. Attention helps to select the most important information to process. In a robot, an attention mechanism can help prevent perceptual overload.

Even if a person could process all of the incoming information, he or she would then be subject to response incoherence. The amount of input might generate so many potential responses that a person would be overwhelmed trying to decide what to do and in what order. Attention helps to select the most important information to respond to. A robot could use an attention mechanism to help prevent response incoherence.

If these problems arise in a robot and we want to use an attention mechanism, what form should it take? Is a human-style attention mechanism useful in a robot? There are fundamental differences between computers and people, which indicates that a computational perceptual attention mechanism may differ from a human attention mechanism. A computer's fundamental processing speed is currently about six orders of magnitude faster than the human brain and is increasing; it can retain information longer and more accurately; and it can use different, more powerful, sensors which can easily be added or removed. Additionally, most computers use a single sequential processor, while the human brain performs much of its processing in parallel. Furthermore, if it chooses to, a computer can completely ignore certain inputs, or even entire streams of data, while people have difficulty ignoring unwanted input. Given these differences, are the properties of a human attention mechanism useful in a robot?

In summary, the questions this dissertation addresses are:

1. *Do robots need an attention mechanism?*
   In our robot navigation work, we have found that robot controllers require a perceptual latency (the time delay between when sensor data is collected and when it is available for high-level processing) of less than 200 milliseconds, and preferably closer to 100 milliseconds. Some computer vision systems have even more stringent requirements, with maximum latencies of approximately 30 milliseconds. Processing a single object typically takes anywhere from 1 to 10 milliseconds in our system. So in the worst case, a minimum of 10 objects in the visual field can cause the system to exceed the 100-millisecond latency required for good performance. Robot sensors on current robots are capable of sensing dozens, if not hundreds of distinct objects at a time. Robots can use an attention mechanism to reduce perceptual overload in several ways: by filtering input to reduce the number of objects processed; by prioritizing input so the most important input is processed first; and by ensuring that the latency requirements are met even when the perceptual system is under a heavy load.

2. *What are the useful characteristics of human attention?*
   Studies of human perception and attention have identified several characteristics that are useful in certain situations. These characteristics include: *early selection*, where sensor input is heavily filtered or blocked before it reaches the cognitive processor; *late selection*, where sensor input is prioritized according to perceptual goals related to high-level tasks; *habituation*, where repeated, identical inputs are eventually reduced in priority or even ignored; *priming*, where recently processed concepts influence the percepts selected for processing; *decay of activation*, where the "interestingness" of a percept decays exponentially over a relatively short period of time; and *inhibition of return*, where objects in the field of view which have been subject to recent scrutiny acquire a reduced perception priority. It is not difficult to envision scenarios in which a robot can utilize some or all of these characteristics to improve its performance.

3. *Which characteristics of human attention are not useful?*
   The answer to this question is somewhat task-dependent, but extremes of human attentional behavior include *autism*, of which one of the primary symptoms is an inability to shift the focus of attention to novel items. An attentional disorder at the other extreme is *attention deficit*, which is the inability to

maintain a focus of attention for an appropriate amount of time. There are also other, more specific, disorders related to brain pathologies. However, except for extreme cases of these disorders, they can not be viewed as strictly negative characteristics because they can be useful in certain situations. For example, an attention deficit may be of benefit to a security guard who must be alert to anything unusual. And difficulty with shifting the focus of attention may be useful to a chess player, who needs to concentrate for long periods of time on a relatively small and static field of view. Clearly, the attention mechanism must be adaptable to different situations and exhibit a wide range of behavior.

4. *Given the differences between human and computer computation and perception, what form should a computational attention mechanism take*?
   This dissertation takes the view that computational attention should exhibit the flexibility and characteristics of human attention, without attempting to duplicate the underlying architecture. The single-processor, multitasking computers that are prevalent today are so different from the highly parallel human brain that a computational attention mechanism requires a significantly different design. Chapters 3 and 4 discuss the design tradeoffs and the architecture of the resulting system in detail.

## 1.2 OVERVIEW OF THE DISSERTATION

Chapter 2 discusses characteristics of human attention in greater detail. Chapter 3 discusses previous work in robot perception and attention, analyzes differences in human and robot attention, and sketches a design for a general-purpose multimodal perceptual attention system. Chapter 4 discusses the design and implementation of CPA (Computational Perceptual Attention), a system for robot perceptual attention. Chapter 5 evaluates the CPA in different perception scenarios and describes how to use the CPA for robot navigation. Chapter 6 discusses the Q-measure, which measures perception quality and provides an indirect measure of the performance of the attention mechanism. A learning system can use the performance measure to tune the attention mechanism. Chapter 7 discusses the relationship between the CPA and other mechanisms for attention, task scheduling and object tracking. Chapter 8 summarizes the dissertation and discusses future research based on this work.

# Chapter 2: Human Perception and Attention

Attention is involved in the process of perception, somehow enhancing or reducing the "interestingness" of various sensory inputs so that some are attended to and others are not. Attention is not completely understood, but it appears to consist of a set of mechanisms that exhibit different, sometimes opposing, effects. Much of the research in attention has focused on the visual sense (or visual *mode*, as psychologists would say), both because it is easy to test and because it is complex enough to be interesting. Most of the remaining research has investigated attention in auditory perception. There has been some research on cross-modal attention, usually involving combined visual and auditory perception.

In the early days of attention research, many investigators performed experiments by subjecting themselves to various stimuli and reporting their perceptions and thoughts about what had happened [LaBerge, 1995, chapter 2]. In the last half of the twentieth century the focus shifted to testing volunteer subjects, often using standard double-blind tests to achieve more objective results. A typical test measures the reaction time to a stimulus and the effect of preceding or coincident stimuli on the reaction times.

During the last half of the 1990s, the use of positron emission tomography (PET) scans and other forms of brain observation have allowed researchers to observe which parts of the brain are activated when different attention mechanisms are activated. The results show that attention is more widespread and non-localized than some had believed [Luck and Girelli, 1998].

In developing a theory of perceptual attention for computer systems, we are interested in attention mechanisms that improve perceptual processing (so we can use them), and in attention disorders that hinder perception (so we can avoid them). We are also interested in how attention mechanisms are activated, how they are controlled, and how such mechanisms might differ if implemented on a fast, sequential computer than on the slow, highly parallel human brain.

## 2.1 THE STRUCTURE OF HUMAN PERCEPTION

The human body contains multiple sensors that send information through a network of nerves to the brain, where it is processed. The brain contains areas devoted to specific processing functions and specific sensor input, although several of these areas may activate during any single act of perception. The sensory system is highly parallel in both input and processing. It is possible, for example, to monitor visual input while listening to separate, unrelated conversations in each ear [Posner and DiGirolamo, 1998]. Performance at such a task is not very good, but that it can be done at all is a testament to the flexibility and power of the human brain.

Most research has focused on vision due to its complexity and the ease of performing tests on subjects. The visual pathways in the brain illustrate the complexity of processing. The figure below is based on one in [Goodale and Humphrey, 1999]. It is a simplified representation of visual information flow in the macaque brain. Similar pathways have been found in other primates and it is generally believed that this representation applies to the human brain, too. The darker lines represent greater information flow.

The diagram shows that processing is separated by functionality (object recognition vs. spatial perception) and that it occurs in stages. We do not yet know enough details about the types of processing that occur in the different brain regions to implement the same mechanism in computing devices.

It is estimated that the human brain contains $10^{11}$ neurons, and $10^{14}$ pathways, implying a high degree of interconnectivity among the neurons. Electrical impulses, triggered by chemical reactions, transfer information from neuron to neuron. The speed of these impulses is much slower than the speed of electrical impulses in a computer. For example, when presented with a set of visual stimuli, the brain can shift its attention from one object to another in 30 to 50 milliseconds [Niebur and Koch, 1998]. This is a mental shift, rather than a physical shift of visual attention; an eye saccade requires about 200 milliseconds. Perception can be directed to certain inputs by visual or auditory cues. The effect of these cues reaches a peak in about 200ms and decays in a few hundred more milliseconds [Niebur and Koch,

1998]. These times are illustrative of the fundamental perceptual processing speeds of the human brain, which is obviously much slower than a computer of today.

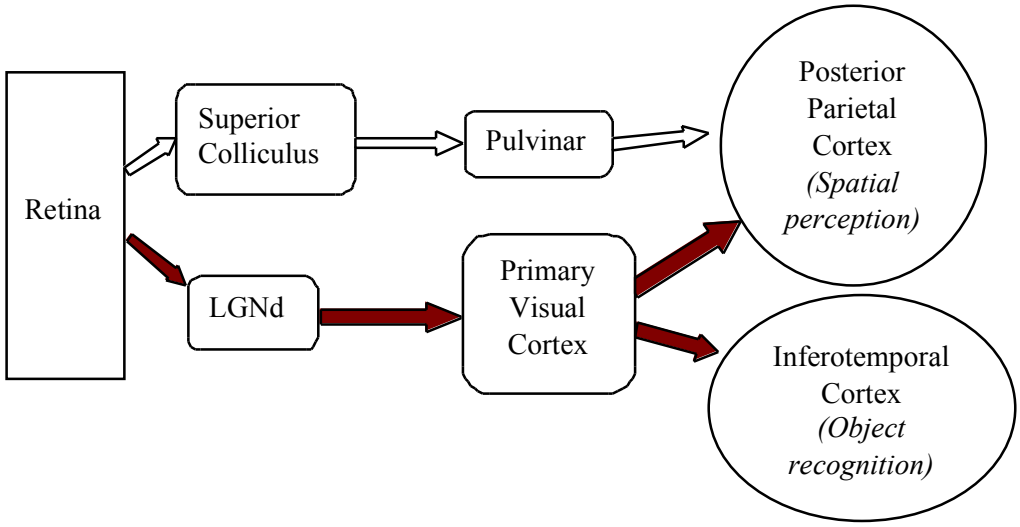


**Figure 1.** Visual information flow in the human brain.

## 2.2 A BRIEF HISTORY OF ATTENTION RESEARCH

Aristotle, Lucretius, Descartes and Leibniz, among others, speculated about the mechanisms of attention [Hatfield, 1998]. The modern era of attention research began with William James, who was the first to outline a theory of human attention [James, 1890]. His primary hypothesis was that thought processes produce expectations of what we *should* perceive, which influences what we *do* perceive. Other researchers, including Gibson, Von Helmholtz, Wundt and Tichenor, hypothesized additional mechanisms including spatial attention (attending to certain locations), and intentions to act. (Much of this historical background is from [Van der Heijden, 1992, Chapter 2]).

The field of attention research experienced a revival in the 1950s. Broadbent proposed his *filter theory of attention* in an attempt to explain many of the existing experimental results [Broadbent, 1958]. This theory postulates a low-level filter that allows only a limited number of percepts to reach the brain at any time. In this theory, the importance of conscious, directed attention is minimized. The part of attention involving low-level filtering is now called *early selection*.

A few years later, the *response selection theory of attention* [Deutsch and Deutsch, 1963] described an attention process in which nearly all of the sensory input reaches a fairly high-level part of the brain, where it is then processed if its activation is above a threshold that changes according to the person's needs. The part of attention involving high-level processing is now called *late selection*.

Starting in 1960, Treisman proposed a series of models that combined early and late selection into a model known as Feature Integration Theory (FIT) [Treisman, 1988]. In FIT, features (colors, shapes, and contrasts) are recognized in parallel and stored in feature maps. A three-dimensional location map independently stores pointers to locations of all features. Objects are then recognized as conjunctions of features. Currently, Treisman and others believe that early selection (sensor and feature attenuation) is most active when the perceptual load is high, whereas late selection (object-based and location-based) is used when perceptual load is low [Treisman, 1998].

Currently there is a variety of research using brain scans to trace brain processing during perception [Luck and Girelli, 1998]. There have also been advances in neurophysiological models of attention [Crick and Koch, 1990; [Olshausen et al., 1993; Desimone and Duncan, 1995; Niebur and Koch, 1998]. An excellent collection of papers covering current research is [Parasuraman, 1998a].

## 2.3 BASIC ATTENTION MECHANISMS

Attention consists of a set of mechanisms for improving perception in relation to the person's current high-level goals. Inside the attention process there is a fundamental tension between the ability to notice certain stimuli and the equally valuable ability to ignore stimuli. Many of the mechanisms have an opposing mechanism that counterbalances their effect. What we think of as attention is a function of the push and pull of a number of mechanisms, including *selection*, *vigilance*, *control*, and the measurable effects of *spatial focus*, *priming*, *inhibition of return* and *decay* [Parasuraman, 1998b].

*Selection* is the ability to select the most important percepts from the large set of incoming percepts. This includes selection across modalities and selection within a modality. The discussion earlier in this chapter mentioned the distinction between early selection (filtering percepts before they reach conscious thought) and late selection (conscious selection of percepts from a set of percepts). In a computational model of attention, the reasoning process should be able to control the entire attention process, even if some of the mechanisms being controlled correspond to automatic processes in the human brain. An enhancement of this is *divided attention*, in which percepts related to different goals are selected at the same time.

*Vigilance*, or sustained attention, ensures that perceptual goals are maintained as long as necessary. Ongoing perceptual goals can be interrupted or impaired by unexpected high-priority inputs or by an increase in sensory data rates. An opposite mechanism is *habituation*, in which repeated similar percepts are gradually reduced in importance.

*Attentional control* is the ability to interrupt and resume perceptual activities as required. In a complex system it is no small task to interrupt and then correctly resume a process. The other active goals may have changed, meaning that the priority of the resumed process may need to be adjusted. This is almost surely a duty of the high-level reasoning process.

*Spatial attention* is a commonly-used visual filtering mechanism. The sensory input in the *spatial focus*, commonly called the *spotlight*, has priority for processing. This is a very effective method of reducing the amount of input to be processed. Computer vision systems often use spatial attention as their primary, and sometimes only, attention mechanism. An opposing effect is *inhibition of return*, in which recently-scanned spatial locations have a lower priority than normal.

*Priming* occurs when a spatial or perceptual cue facilitates perceiving a stimulus that follows. The facilitation typically results in a reduced time to recognize or categorize a percept. Its opposite is *inhibition*, in which a cue impairs perceiving another stimulus.

The effects of priming and inhibition *decay* over a relatively short period of time. The brain rarely perceives a constant set of inputs. The world changes rapidly and one of the effects of decay is to ensure that cues do not outlast the world in which they were generated.

## 2.4 ATTENTION DISORDERS AND ERRORS

As might be expected from the number of times one is told to "*pay attention*", perceptual attention does not always work correctly. There are several brain disorders and injuries that can affect attention. One of the most prevalent is *attention deficit disorder (ADD)* in which the ability to focus attention is impaired. Some of the recognized symptoms of ADD [Swanson et al., 1998] include:

1. Inability to pay attention to details,
2. Difficulty sustaining attention,
3. Difficulty organizing tasks,
4. Easily distracted by extraneous stimuli.

As mentioned in Chapter 1, some of these characteristics are, at times, beneficial.

At the other end of the attention spectrum, one of the characteristics of *autism* is the inability to change the focus of attention [Rodier, 2000]. Again, this characteristic can be beneficial in some tasks. It appears that a computational mechanism should be tunable to exhibit a wide spectrum of behavior as deemed necessary.

There are situations where incorrect attentional behavior has had disastrous consequences. Everyone today is familiar with accidents caused by people who are fiddling with their car stereo or talking on

cellular phones. The most well-documented large-scale case of inattention is probably that of Eastern Airlines flight 401, which crashed on December 29, 1972. In that incident, the crew forgot to monitor the plane's flight while they attempted to visually determine whether its nose gear was properly extended. While they were distracted, the plane gradually descended and crashed into the Everglades, killing over one hundred people. The flight crew received an alert from an automatic warning system 13 seconds before the crash, but were so focused on the nose gear that they could not react in time to prevent the crash [NTIS, 1973].

## 2.5 SUMMARY OF HUMAN PERCEPTUAL ATTENTION

Human perception and perceptual processing is highly parallel, yet relatively slow compared to the speed of today's computers. Attention is not completely understood, but it appears to consist of a set of mechanisms that exhibit different, sometimes opposing, effects. Attention provides a wide range of behaviors, from highly-focused to easily distractible. A computer-based perception process may need to exhibit equally flexible behavior in order to accommodate a wide variety of tasks.

## 2.6 TOWARD COMPUTATIONAL PERCEPTUAL ATTENTION

Human perceptual attention exhibits a variety of mechanisms that allow it to adapt to different tasks. General-purpose perceptual attention for computers should exhibit the same flexibility. The following properties of the human attention mechanism are key design goals for our computational attention system.

1. **Sustained attention:** the mechanism should allow the computer or robot to set perceptual goals and maintain them over a period of time. The termination of a perceptual goal might be condition-based, time-based or directed by a higher-level system. *Detection* and *tracking* are two important classes of perceptual tasks that should be sustainable.
2. **Selective attention:** the mechanism should support selectivity in forwarding sensor input to the reasoning component of the system. Selectivity should be guided by features such as color, size, shape and spatial location.
3. **Divided attention**: the ability to pursue more than one perceptual goal at a time is very important. Otherwise the performance of a robot will be very limited—for example, navigation and object recognition could not be performed at the same time, requiring the robot to stop whenever it needed to detect objects.
4. **Controllable:** top-down control is important, since perceptual goals are related to other goals such as navigation. Two key methods of control are the ability to set perceptual goals and the ability to dynamically prioritize perceptual goals. The ability to be interrupted by unexpected input should be adjustable since different tasks require different strengths of perceptual focus.

# Chapter 3: Perception and Attention in Computers

The human brain and current digital computers differ significantly in structure, speed and capabilities. The design of perception systems is greatly affected by these differences.

## 3.1 SPEED, SEQUENTIAL PROCESSING AND SENSOR VARIETY

The most important difference between the human brain and computers is that most computers are single-processor machines, which process information sequentially, while the human brain is a vastly parallel system. With the speed and capabilities of today's multi-tasking operating systems, it is easy to overlook this fundamental point. Because of the sequential nature of machine processing, the time to perform perception tasks usually scales linearly with the number of objects detected. As sensors become more powerful, more objects will be detected, making it unlikely that sensor processing will ever keep up with sensor acquisition.

The second most important difference is that computers have a fundamental speed that is orders of magnitude faster than that of the human brain. The speed factor compensates somewhat for the sequential processing, but not as much as one might hope, possibly because useful operations like face and object recognition still take a relatively long time and are not always accurate.

The third most important difference is the variety of sensors available to computers. Humans have a fixed number of sensors, but robots have access to an ever-growing catalogue of sensors, whose capabilities in many cases equal or exceed human sensors. Other sensors provide input not available to people, such as infrared vision, GPS locations and compass directions. A robot can receive a huge quantity of data at a high data rate. Furthermore, these sensors are more controllable than in people. Robot sensors can be turned off or completely ignored as desired, something that is difficult with human senses [Posner and DiGirolamo, 1998].

All of these differences point to time management as a critical component of computer perception. This affects the design of the entire perceptual system, as described in Chapter 4.

## 3.2 GENERALITY VS. SPEED REQUIREMENTS

The design process is complicated by a tension between generality and the need for fast low-level sensor processing. This research aims to produce a general-purpose, multimodal, controllable perception system. One can design a simple multimodal perception system as shown in the figure below. This design is pleasingly simple but it has several areas of potential inefficiency.

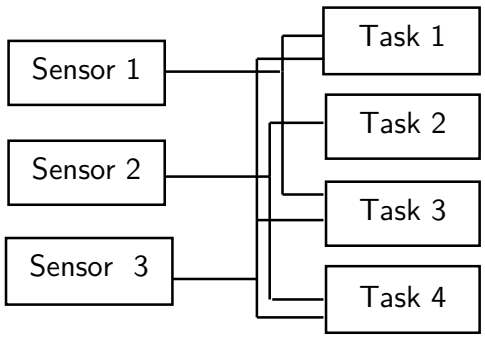


**Figure 2.** A simple, but potentially inefficient, perception system.

### 3.2.1 Data transmission times

A hidden cost of perception is the time required to transmit data from the sensor to the task. A robot whose components are distributed across multiple systems may encounter significant transmission times. On one of our robots, we found that a rangefinder scan, which consists of 270 polar coordinates, takes

several hundred milliseconds to transmit from one machine to another using a distributed object system. Since robot control works best when the perceptual latency is less than 100ms, we could not control the robot very well when transmitting raw sensor scans. In the CPA system described in this dissertation, the solution was to move sensor post-processing from the task side to the sensor side of the connection so that raw data sets need not be transmitted.

### 3.2.2 Redundant processing

More than one task may be performing the same post-processing on the raw data. For example, in our robot the wall-following task and the view-matching task both utilize line segments derived from rangefinder data. Line segment extraction takes 15-25 milliseconds, a non-negligible amount of time. Most or all post-processing of raw sensor data should be performed before the task level and tasks should share the results.

### 3.2.3 No limits on processing time

The diagram above implicitly relies on an unspecified external program to govern the amount of time a task can use. There is no guarantee that an important task will receive any time at all. The CPA system contains a scheduler that allocates time to perception tasks in order of importance.

### 3.2.4 Solutions

As described above, a simple perception architecture can be the source of numerous inefficiencies. How do we solve the above problems? As shown in the next chapters, we:

1. Add postprocessors to the sensor streams so tasks can share processed, as well as raw, data.
2. Move all perceptual tasks that take a significant amount of time, such as object recognition, into the perception module where they can be controlled and where they will be under the control of a real-time task scheduler.
3. Implement a centralized attention manager that allocates time to perception tasks according to their priority.

The resulting system is illustrated by the figure below.

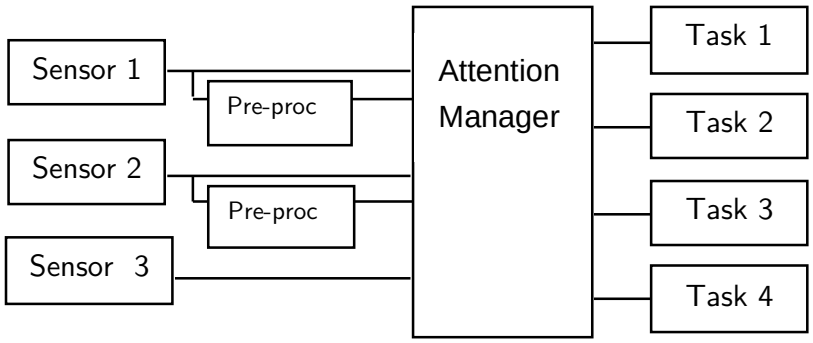


**Figure 3.** A better design for a perception system.

## 3.3 PREVIOUS PERCEPTUAL ATTENTION SYSTEMS

VISIT [Ahmad, 1991] processes 256x256 pixel images by moving a spatial attention "spotlight" around the image. A control system, consisting of a connectionist network, controls the spotlight. VISIT implements inhibition of return (see Chapter 2) to aid in scanning the entire image. The network can be trained to move the spotlight based on different characteristics of the image, including center of mass of the portion of the image inside the current spotlight, pixel clusters in the entire image, and object features

such as color and shape. The attention mechanisms it uses are spatial attention, attention control and inhibition of return.

VISIT is not multimodal and uses only spatial attention. The control mechanism must be trained separately for each task and can perform only one task at a time. The size of the control network is proportional to the size of the image being processed, so changing the input size requires creating and training a new control network.

There are numerous other visual attention systems that also manage spatial attention. These are popular because until recently computers were not fast enough to process an entire camera frame in the time allowed, even for relatively simple processing. A useful optimization is to select only a portion of the input to process. Some of these systems are:

- A system that integrates bottom-up feature maps with top-down model-based object knowledge using non-linear relaxation based on energy minimization [Milanese et al., 1992].
- A connectionist system for constructing a saliency map that emphasizes and de-emphasizes regions of the visual input for the ALVINN autonomous navigation system. The system learns to handle difficult visual scenes that the general-purpose algorithm in ALVINN can not handle [Baluja and Pomerleau, 1996].
- The SCAN (Signal Channeling Attentional Network) model, which is a sparsely-connected, scalable neural network that incorporates an expectation-generating classifier network. The classifier network allows SCAN to be expectation-driven. Like the other systems, SCAN is used for directing spatial attention [Postma et al., 1997].
- An extension of ACT-R to incorporate visual attention and pattern recognition [Anderson et al., 1995]. This system also uses spatial attention, and does not appear to provide real-time guarantees.
- A spatial attention system for a social robot that incorporates top-down motives and bottom-up feature maps [Breazeal and Scassellati, 1999].

One of the few multimodal systems is QuickSet, which manages voice and pen input for interacting with a map [Cohen et al., 1997]. The user can, for example, circle a location on the map and say, "What is this?" to retrieve a description of the object at that location. However, this is a relatively easy perception problem, compared to managing perception on a robot, because the inputs are relatively sparse and the maximum response time limits are relatively long.

A system with broader goals and multimodal capabilities is a perception planner for mobile robots [Xu and Vandorpe, 1994]. The system uses sensor fusion to integrate readings from multiple sensors. The perception planner adjusts sensor parameters, uses utility functions to determine which perception tasks have a high priority, and selects fusion techniques. The planner also incorporates uncertainties in data or internal states to adjust priorities of perception tasks.

In summary, most previous attention systems were unimodal and focused on spatial attention. Most are special-purpose mechanisms suited only for the application they were designed for. The CPA system described in the next chapter provides a general-purpose multimodal solution to perception processing.

## 3.4 ATTENTION-RELATED PROBLEMS IN CURRENT SYSTEMS

Chapter 2 mentioned the crash of Eastern Airlines flight 401, in which the flight crew was so focused on another task that the plane flew into the ground while they were distracted. It is of interest to note that a nearly identical situation has occurred in a computational system.

ModSAF is a complex battlefield simulation that includes simulated helicopters flown by simulated helicopter pilots [Hill et al., 1997]. The pilot agents must fly their helicopters while visually scouting for enemy tanks. If tanks are found, the helicopter scouts return to the main group and report their findings. In one scenario, three helicopters came over a hill and suddenly encountered a field containing ninety tanks. Unable to divide their attention correctly, the simulated pilots concentrated on processing visual information about the tanks. Meanwhile the helicopters drifted out of position because the helicopter gauges and controls were ignored. Consequently, one of the helicopters crashed into a hillside while the other two helicopters crashed into each other. Input from the helicopter controls should not have been

ignored while processing the visual input. Instead, perceptual tasks for the two operations (scanning and flying) should have been interleaved. Assuming the tasks were prioritized correctly, the CPA system discussed in the Chapter 4 of this dissertation would handle this correctly.

A second, less destructive, example is MIT's Intelligent Room project [Coen, 1997]. This room contains a speech recognition system and a vision system that tracks people and recognizes gestures. It is a prototype of a room where people can point at displays containing maps and ask questions of the room's computer. A recent video of the system shows a person gesturing at a map, waiting a few seconds, and then asking a question. The system is apparently unable to process visual and audio input at the same time. To be fully functional, the room needs to be able to perceive multiple sources of input at the same time.

These examples illustrate the need for a perceptual attention process in modern intelligent agents. The rest of this dissertation discusses the design and implementation of a general-purpose attention process that will be useful in multimodal systems like these.

# Chapter 4: Computational Perceptual Attention

This chapter discusses the design goals, operational characteristics and implementation of a system, CPA, for multimodal computational perceptual attention. An example that uses the CPA in a robot navigation task is discussed in Chapter 5. Chapter 6 discusses ways to evaluate the performance of the CPA.

## 4.1 COMPUTATIONAL ENVIRONMENT

CPA is designed for a single-processor, multi-tasking computational environment. It is designed to meet real time bounds, but since it does not assume a real time operating system it is subject to processor contention from unrelated processes, and can meet only soft real time bounds. It is compatible with a distributed system environment in which various components run on separate machines.

The overall structure of the CPA is shown in the figure below. The various components are discussed in more detail later in this chapter.

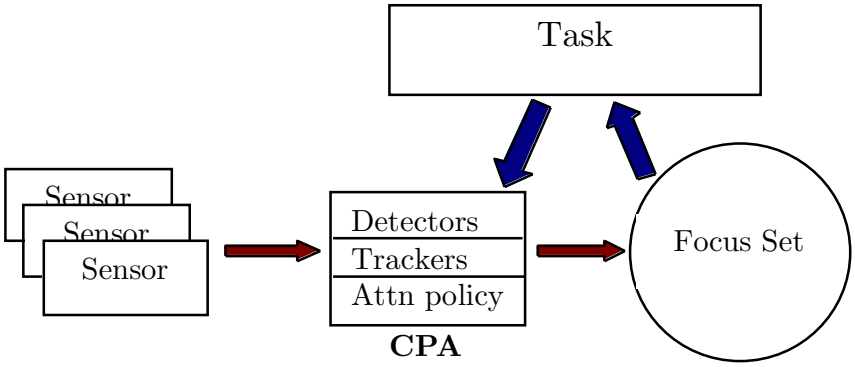


**Figure 4.** Structure of the CPA.

A high-level system, such as a robot navigation system, directs the CPA to monitor sensor input for types of percepts that are relevant to its current task. The CPA delivers percepts to the focus set at regular time intervals, the percepts having been filtered and prioritized according to the task's needs. The task periodically retrieves percepts from the focus set for processing. The sensor buffers (one for each sensor) and the focus set act as transfer points for percepts as they pass between the asynchronous sensor, CPA and task processes. As the high-level system performs different tasks, it modifies the CPA to specialize it for the task at hand.

## 4.2 DESIGN GOALS

The primary purpose of the attention mechanism is to maintain a *focus set* of important percepts by filtering and prioritizing perceptual input in response to the dynamically changing goals of the current task or tasks. The design assumes the existence of sensors to provide input and a cognitive mechanism to control the attention process by adding, removing and prioritizing detectors as necessary. The task does this by collecting the perceptual components of high-level goals, from which perceptual goals can be extracted. The perceptual goals are then embodied in detectors and trackers. For example, in the task "Enter the door", the component "door" has perceptual features. A knowledge base can hold the perceptual features and associated parameters needed to detect a door using cameras, rangefinders or other sensors. This information is used to construct an appropriate routine to detect the door and track its position relative to the robot as the robot moves. The detectors and trackers are activated and deactivated by the high-level task as necessary.

To be useful, the input must become available to the cognitive process fairly quickly. For example, in a robot navigation controller the correct speed and heading of the robot are a function of the current perceptions, speed and heading. We have found that the quality of navigation control deteriorates rapidly when the *perceptual latency*, the delay between the time of sensor input and the time the information is available to the high-level system, is greater than 200 milliseconds. Thus, the design includes a real time bound that the CPA must adhere to.

### 4.2.1 Design tradeoffs

The first factor complicating the design process is the tension between generality and the need for fast low-level processing. The CPA is intended to be a general-purpose attention mechanism for multimodal systems. To that end, it is useful to abstract most operations to as high a level as possible. However, low-level operations such as visual object recognition are complex tasks that often need to occur 10-30 times per second. This tradeoff was resolved by creating a class of perceptual tasks called Detectors, which directly encapsulate low-level operations such as object recognition. A Detector feeds percepts directly to the focus set, reducing delays in making results available to the high-level system. A second class of perceptual tasks, Trackers, represent tasks that retain memory across perception intervals. Trackers are more efficient than Detectors at dealing with objects that have already been detected once.

The second factor complicating the design process is the obvious opportunity for parallelism in multimodal, multi-tasking systems, which is offset by the reality that single-processor computers are by far the most prevalent type of system in use today. The CPA incorporates a priority scheduler based on dual queues for Detectors and Trackers. The scheduler assumes that only one processor is available. It provides soft real-time perceptual guarantees for the high-level system as long as each perceptual tasks cooperates in meeting its deadline, which is prescribed by the scheduler. An implementation based on multiple processes controlled by a UNIX-class scheduler does not work as well (see Chapter 7). Alternatively, the system could use a true real-time scheduler and operating system that would provide hard real-time guarantees for perceptual latencies.

A third design problem is the need for flexibility in sensor input. One task may require unprocessed frames from cameras, while another may need a list of the doorways found while processing those camera frames. One task may require entire newsgroup articles, while another would like only the nouns used in the article. This problem was resolved by allowing a "sensor stream" to consist of processed percepts as well as raw percepts (although usually not both at the same time). Following is a discussion of each design goal.

### 4.2.2 Maintain a focus set

The primary purpose of the CPA mechanism is to maintain a set of recent perceptions in an attentional focus[1]. The percepts are ordered by their activation level, which decays over time. The decay function usually has a very short half-life. In order to reduce computation, the activation level of a perception is computed only when the members of the focus set need to be sorted, i.e. when the focus set is retrieved.

When psychologists measure decay of activation in people, the decay curve is exponential [Peterson and Peterson, 1959] as shown in Figure 5.

$$activation(t) = ae^{-kt}$$

**Figure 5.** An exponential decay equation for activation level.

The variable $a$ is the initial activation level of the percept. The time $t$ is in seconds. The value $k=0.554$ produces a curve in which the initial activation level decays by two-thirds after two seconds,

[1] The attentional focus may or may not correspond to *short-term memory*. We have purposely made no attempt to map one onto the other, believing that if they are related the operational characteristics of a limited short-term memory may emerge as a by-product of normal processing.

which we have found heuristically to produce a useful decay curve. Higher values of *k* cause the activation level to decay faster.

The percepts in the attentional focus are incoming percepts that have been annotated by the detector or tracker that processed them during the attention phase. Typical annotations include a type indicator and some computed values such as the distance to or size of the object.

Incoming percepts that correspond to items already in the focus set should refresh the activation level of the pre-existing focus item, rather than create an entirely new entry. The appropriate detector, tracker and/or sensor should contain the information needed to determine whether or not a percept is new. When a high-level task retrieves the contents of the focus set, the focus set is emptied in preparation for the next set of percepts.

### 4.2.3 Real-time delivery of percepts

The CPA has a time bound that specifies the maximum amount of time allowed to update the focus set. The time bound will typically be on the order of 100 milliseconds for many robot-related tasks. The importance of sensors, detectors and trackers (discussed below) should determine the amount of time allocated to each data item. The CPA's performance (see Chapter 6) should degrade gracefully when the CPA's requirements meet or exceed the allotted time. The overhead of computing the time allocation for various detectors and trackers should not exceed 1% of the time bound.

In order to provide an appropriate amount of attention to detectors and trackers, the available time is divided into two pools: detection and tracking. This provides a coarse method of limiting the amount of time used for either task.

Under conditions of heavy perceptual load, some trackers will not be called on a given attention cycle. There are two cases when a tracker may not be called during an attention cycle. One is when there is no sensor data for that tracker. In this case, the attention mechanism calls the tracker anyway with a NIL percept. The tracker may wish to provide an estimate of the object's current position or other characteristics it is tracking. The second case is when the attention mechanism runs out of time. In this case, the tracker will be notified that it is being skipped.

### 4.2.4 Multimodal perception

The CPA must support multiple asynchronous sensory input streams, each of which may supply a different type of data. The importance of each sensor will often change while the CPA is executing. Changes in sensor importance should be reflected in the amount of time the CPA allocates to processing inputs from each sensor and/or in the priority assigned to data from each sensor. Each sensor stream should be buffered and the buffer should have a fairly small capacity: three percepts, for example. This ensures that data in the buffer are current and provides a coarse method of handling perceptual overload.

A sensor stream may contain *raw* data directly from the sensor, or it may contain *processed data*, which is derived from the raw data. The CPA treats both types of sensor streams equally.

### 4.2.5 Adaptable attention policy

The attention policy consists of several adjustable values and functions that are used to determine the allocation of time to the active perceptual processing components. The adjustable values include:

1. **Attention time bound**. This parameter specifies the maximum amount of time allowed to acquire perceptual input and update the focus set.
2. **Focus strength**. The focus strength determines the activation level assigned to percepts that were not accepted by any of the active detectors or trackers. They receive an activation of (1 – focus strength). For some tasks, these unexpected inputs may be important, while other tasks may wish to ignore these inputs. A high focus strength means that the system will be less easily distracted from its perceptual goals.
3. **Decay rate**. The decay rate affects how quickly the activation level of percepts in the focus set decays. This is the constant *k* in the decay equation presented earlier in this chapter.
4. **Percentage of time allocated to object detection vs. object tracking**. As discussed in Section 4.2.3, this is a coarse method of adjusting the time allocated to detection and tracking. For example,

if most of the time is allocated to object tracking, it is possible that "new" objects will not be noticed. This is desirable in some tasks, but undesirable in others.

5. **Priority of sensors**. A sensor's priority affects the order in which raw sensor percepts are processed. A sensor's priority may change as the cognitive task changes.
6. **Priority of detectors and trackers**. As described below, trackers are created by detectors to track newly-detected objects. Each detector has a priority that specifies how important it is to the current cognitive task. This priority may change as the current task changes. A tracker's priority is some combination of the priority it inherited from the detector and the priority of the object it is tracking. This priority may change as the cognitive task changes.
7. **Prioritization functions**. Each detector and tracker needs to be prioritized according to some value, but several values are present: its sensor priority, its own priority, and its uncertainty in its tracked parameters (for trackers). The task can select the method of integrating these values into one overall priority value.

An *attention policy* is an instantiation of the above parameters. The attention policy affects the contents of the focus set and, thus, the quality of perception. Chapter 6 discusses the Q-measure, which is used to measure the quality of perception and, indirectly, the quality of the attention policy.

## 4.3 COMPONENTS OF THE ATTENTION MECHANISM

The previous section mentions several components such as sensors, detectors and trackers. This section describes each component in detail.

### 4.3.1 Sensor streams

In human attention, a sensor stream provides a steady and mostly automatic stream of data. In robots and machines, the sensory stream is much more controllable and its contents can vary widely. In the CPA implementation, the Sensor class is derived from a Generator class, since a sensor can be thought of as a generator of data (although in actuality it may just be a conduit for the data).

The code for a very simple sensor is shown below. This sensor produces odd numbers starting from 1, every 300ms.

```
int num=1;
loop:
  produce num;
  num = num + 2;
  sleep(0.3);
end loop;
```

**Figure 6.** A simple sensor.

A more typical sensor might collect a robot's sonar readings every 100ms and place them in the sensor stream:

```
loop:
  produce(robot7.getSonarData());
  sleep(0.1);
end loop;
```

**Figure 7.** A sensor for sonar data.

Usually, each sensor runs in a separate thread, asynchronous with the CPA and with each other. It stores percepts in a sensor buffer, which is typically a limited-capacity FIFO buffer. The percepts in each sensor buffer are retrieved periodically by the attention mechanism. The sensor buffer acts as a coupling mechanism between the sensor and the attention mechanism, which run in parallel, but asynchronously.

The details of the actual Sensor and Buffer classes, as well as all other classes used in the CPA implementation are discussed at the end of this chapter, and are used in the robot navigation system described in the next chapter.

### 4.3.2 Percepts

All objects in the sensor buffers and the focus set are Percepts. The sensors produce *raw percepts* while the focus set may contain either raw percepts or *processed percepts*. A processed percept typically contains the raw percept as well as some information derived from the raw percept. For example, the raw percept may be a set of laser rangefinder readings, while the processed percept might be a percept representing an open door whose existence has been determined by examining the set of rangefinder readings. Another example is a raw percept representing one article from a `jobs` newsgroup, with a processed percept representing a summary of a Java programming job extracted from the article.

### 4.3.3 Detectors

A detector encapsulates an object recognition method for a high-level percept. The term "object recognition" is used here in its most abstract sense, for detectors can be created for visual objects, audio objects, text objects, numeric values outside a certain range, prime numbers and so on. A task should create a detector for every object type that it finds interesting. A detector receives raw or processed percepts from one or more sensor streams. If it detects the object it is looking for, it forwards the percept to the focus set or creates one or more processed percepts and forwards them to the focus set. A detector may optionally instantiate a tracker to monitor the object over time.

Normally, the CPA contains a *default detector* that is always active. The default detector accepts any percept that was not accepted by any other active detector or tracker. It places the percept in the focus set with an activation level of `(1 – focusStrength)`. This makes unexpected input available to the ongoing task at an appropriate activation level.

### 4.3.4 Matchers

Matchers contain functions used by detectors and trackers to identify percepts in sensory streams. The matcher is extracted as a separate object so it can be used in different detectors. A matcher is task– and sensor–independent, and can be parameterized to deal with different types of data. This allows, for example, a wall matcher to be used with different sensor streams and to be specialized for different size walls. Part of the specification of a detector is a specification of its matcher and a set of arguments that specialize the matcher for a specific task.

### 4.3.5 Trackers

Trackers differ from detectors in that they retain knowledge of object state between calls. Their processing time is also allocated from a different pool than that of the detectors. They may use their state knowledge to locate an object faster or to compute information (such as object speed) that requires more than one raw percept to compute. To prevent detectors from generating trackers for objects that are already being tracked, a tracker installs *masks* on detectors to prevent them from detecting certain objects. A common use of masks is to block the detector from seeing an object in a certain spatial location. Like detectors, a tracker either forwards a raw percept to the focus set or instantiates a processed percept for the focus set.

A tracker is called every attention cycle, if time allows. Sometimes, there may be time to call the tracker, but no sensor data for it on that cycle. If so, the tracker will be called with a NIL percept. When this happens, the tracker can still add a percept to the focus set. The percept usually contains an estimate

of the object's characteristics, based on the amount of time that has passed since the last real percept was received[2].

If no time is available to call a tracker, its `skipped` method is called to let it know that time has passed, but it wasn't called. Since time has run out, the tracker should not perform any computationally-intensive tasks inside the `skipped` method.

### 4.3.6 Focus Set

The focus set contains the list of percepts that have been accepted by a detector or tracker. Percepts arrive in the focus set with an initial activation level that has been set by their detector or tracker. When the focus set is retrieved, the current activation level of each percept is calculated using the decay equation described in Section 4.2.1. Thus, the task receives prioritized percepts that are relevant to the task and are sorted according to the task's priorities as embodied in the set of active detectors and trackers. When a task retrieves the contents of the focus set, the default behavior is to clear the focus set. However, if multiple tasks are accessing the focus set without coordinating the use of perceptual input, it may be desirable to retain the contents after retrieval.

### 4.3.7 CPA: the attention manager

The attention manager component, which has the same name as the system, manages all the sensors, detectors, and trackers as well as the focus set. In order to detect a class of percepts, a higher-level task creates detectors and adds them to the CPA, removing them when they are no longer relevant. Similarly, the task adds and removes sensors. The task can pause the CPA, which in turn pauses each of the sensors, detectors and trackers. The CPA also manages the focus set, as detectors and trackers add percepts to it. The main loop of the CPA activates detectors and trackers and monitors time used so that the loop can finish within the time bound. The CPA attention loop is shown in the figure below.

In Steps 3B and 4B, percepts are processed in order of sensor priority, and trackers/detectors are processed in order of tracker/detector priority. The nested loops are required to maintain the order of processing.

## 4.4 CPA Implementation details

Appendix 1 gives details of the classes used to implement the CPA: `CPA`, `Sensor`, `Buffer`, `Detector`, `Matcher`, `Tracker` and `Percept`. Chapter 5 provides a full example of a CPA-based system, and Chapter 6 provides several examples that illustrate various facets of its behavior.

[2] This is somewhat equivalent to what occurs when a person moves through a room after turning out the lights. The relative position of objects is estimated based on the last known position.

**1. Sensing.**

A. Sort the sensors using the sensor prioritization function.

B. Read a percept from each sensor buffer that contains data. The percept order should reflect the relative sensor priorities.

**2. Prioritization.**

A. Sort the active trackers and detectors using the prioritization functions.

**3. Tracking.**

A. Calculate the total time available for tracking, based on the time bound and the percentage of time allocated to tracking.

B. For each percept `p` from 1B,
  For each tracker `t`,
    If tracking time has expired,
      End Step B
    Else if `sensor(p)` is in `sensors(t)`,
      `NewPercepts ← (trackingFunction(t))(p)`
  FocusSet.add(`NewPercepts`).

C. while tracking time is available,
  For each tracker `t` not called in Step B,
    `NewPercepts ← (trackingFunction(t))(NIL)`
  FocusSet.add(`NewPercepts`).

D. For each tracker `t` not called in Steps B or C,
    `skipped(t)`.

**4. Detection.**

A. Calculate the total time available for detection. There may be extra time available if the trackers did not use all of their tracking time.

B. For each percept `p` from 1B,
  For each detector `d`,
    If detection time has expired,
      End Step B
    Else if `sensor(p)` is in `sensors(d)`,
      `NewPercepts ← (matchFunction(d))(p)`
  FocusSet.add(`NewPercepts`).

C. If there is time left, call the default detector on any percept that was not accepted by any detector or tracker.

The purpose of this procedure is to add new percepts to the focus set, under conditions that the contents of the focus set are both (a) current and (b) concurrent. Any changes to the above procedure should satisfy these two conditions.

# Chapter 5: An example: Robot Navigation

This chapter presents a detailed example that uses the CPA in a robot navigation problem. At the end it presents in brief form a second, completely different example that uses the CPA to scan newsgroup articles. The specific procedures used here for robot navigation and object recognition are instances of existing methods, e.g. [Kortenkamp, et al., 1998]. The intent is to illustrate perception management using the CPA and not necessarily to describe the best possible methods for navigation or object recognition.

Robot navigation is a difficult problem for many reasons. Physical control of the robot's speed and direction is governed by mathematical models that must work within a system that does not have continuous access to sensor data. Sensing the environment with sufficient detail and precision is an even more difficult problem. The position of a wall or door as detected by sensors contains uncertainty due to sensor error and limitations in the sensor data processing. This is complicated by relatively large errors in the robot's odometry information. The robot is never exactly sure where it is or where the external objects are. Furthermore, the consequences of making a navigation mistake can be disastrous. Running an expensive robot into a wall is neither good for the robot nor good for the wall.

This chapter discusses the perceptual problems of robot navigation in the context of a specific example and how navigation can use a general-purpose attention mechanism, the CPA described in Chapter 4, to manage multimodal perception. This chapter illustrates how the problem can be addressed in our general-purpose framework without losing the efficiency of handcrafted, low-level solutions.

## 5.1 A ROBOT NAVIGATION PROBLEM

In this example, a robot navigation system is directing a robot down a hallway, intending to stop at a certain door. It navigates by locating the hallway walls and following a path between the two walls. Additionally, it must avoid any obstacles in its path. In this example, we are using the `Flat` robot simulator [Flat, 2000], although the implementation would be nearly identical for a real robot with a similar interface[3]. `Flat` simulates a robot in a 2-dimensional world. It simulates not only the physical movement of the robot, but several sensors, including laser rangefinders, sonars, and an absolute positioning system. The configuration used for this example assumes two laser rangefinders, one on each side of the robot mounted at a 45° angle outward from the center of the robot. This provides approximately a 270° scan around the robot at 1° resolution for a distance of 25 meters.

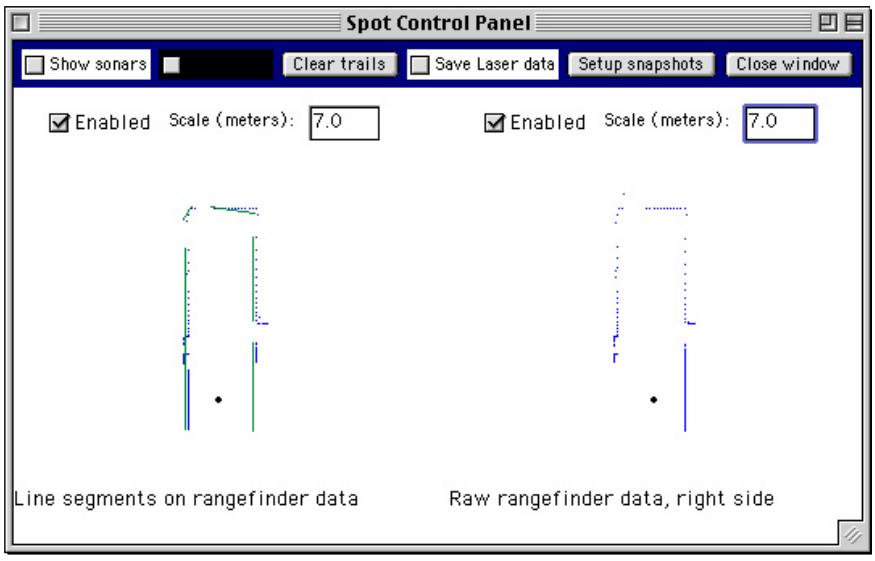


**Figure 8.** A rangefinder scan showing line segments (left) and raw data (right).

[3] To ease the process of transferring navigation routines from the simulator to a real robot, the software in our lab utilizes an interface layer that provides identical functionality in both the simulator and the real robots.

Figure 8 above shows a single scan from a robot with dual rangefinders. The scan is an overhead view; the robot is the black dot in the lower middle of each display and the robot's current direction is always toward the top of the window.

Line segments extracted from the data are shown on the left side, while the raw scan from the right rangefinder is shown on the right side. (The two scans have a 90° overlap, so one scan is sufficient to view most of the environment.) To the human eye, the presence of a hallway is fairly obvious in the set of line segments, as is the doorway on the right side of the hallway.

## 5.2 TOP-DOWN FORMULATION OF PERCEPTION TASKS

Perception provides the robot with information about the external world. The robot uses that information along with its internal behavioral goals to initiate, alter or terminate actions. In addition to behavioral goals, the robot will have perceptual goals. For example, in order to go to a specific room in a building, the robot has to detect and identify the room. On its path to the room, it will need to recognize other objects such as corridors and elevators that help it to determine its position. If the robot were to scan for all known objects at all times, the set of perception tasks would be overwhelmingly large. Thus, the perceptual goals are typically determined at a high level and passed down to the perception mechanism as needed. This is known as top-down attention [Posner and DiGirolamo, 1998]. The robot may have a few important perception goals that are active all the time, such as general obstacle detection, but most perception goals are initiated from the cognitive part of the system.

This section discusses methods that a system can use to instantiate detectors and trackers from perceptual goals. These have been implemented in a prototype system but were not used in the detailed examples of the next chapter.

### 5.2.1 Using spreading activation to find relevant objects

A general-purpose spreading activation process can collect a set of relevant concepts from memory. This section describes such a process that collects objects into a structure called the SACK (Set of Activated Concept Knowledge). Perceptual features of the objects in the SACK are used to create detectors and trackers.

A simple form of spreading activation [Anderson, 1984] can be used to activate relevant concepts based on a set of *primary concepts* contained in the current set of high-level goals (strategies). For example, the goal `Go through the door` contains the noun `door` which has perceptual characteristics, so it would be placed in the SACK as a primary concept. Once the primary concepts have been collected in this manner, *secondary concepts* are then activated from the primary concepts using spreading activation. For a `door`, related secondary concepts might be `doorknob`, `doorframe` and `wall`. The activated concepts provide perceptual schemas or templates that are used to instantiate detectors or trackers.

As the agent's plan evolves, concepts can be added and removed from the `SACK` according to the algorithm below. Let `G` be the current set of behavioral goals, and `P` be the objects in the goals that have perceptual characteristics. Each element of the `SACK` has two components: an object and its activation, which we will represent as `(o, a)`, where `0 <= a <= 1`. Let the activation function *f* be a function such that $f(x) < x$.

1. Update the set of active behavioral goals `G`, and the related set `P`.

2. Initialize the `SACK` using the set `P`, with the activation of each object set to `1`: `SACK = {(`$p_i$`, 1) |` $p_i$ `in P}`.

3. For each concept `c` in the `SACK`,
   Activate(`c.object`).

The `Activate` function computes a transitive closure of relevant objects over the set of perceptually relevant relations. The algorithm terminates when the activation level of an object drops below a defined minimum, MIN_ACTIVATION.

```
Activate(c):
For each relevant relation r on c.object,
   Let w = r(c.object)
   If w is already in the SACK,

     Activation(w.object) =
            max(activation(w.object), activation(c.object))
   Else
     a = f(Activation(c.object))
     If Activation(w.object) > MIN_ACTIVATION,
       SACK = SACK + (w, a)
       Activate(w).   ;; Recur on the new object.
```

The set of relevant relations in Step 3 includes spatially relevant relations such as `near` and `part-of`, that are used to find related perceptual objects for spreading activation. This requires a rich set of knowledge about perceptually observable objects. The spreading process ends at each object whose activation is below a threshold level. The activation must decrease at each successive recurrence in order to ensure termination of Step 3. Step 3 takes $O(k^d)$ time, where $k$ is the average branching factor of the spread, and $d$ is the depth of the spread. The branching factor $k$ increases as the number of perceptually-relevant relations increases. The depth $d$ is related to the activation function $f$ and the value of MIN_ACTIVATION.

Once the SACK is constructed, the perceptual features of the objects in the SACK are retrieved and used to direct perception. For example, if `office-door` is in the SACK, its features `size`, `color` and `location` can be used to construct trackers.

In summary, the SACK is a network of concept knowledge whose roots are in the current plan and whose contents change as the behavioral goals of the robot change. This allows the SACK to focus the attention of the low-level perceptual system on perceptual tasks related to current goals.

### 5.2.2 Perception templates and schemas

A simpler, but less flexible, method of collecting perception tasks uses *perception templates* in order to recognize a specific object. For example, many doors look alike and are differentiated by minor perceptual details. A `door` template specifies the typical size, shape and color of a door. When a specific door needs to be recognized, its perceptual details are retrieved from a knowledge base and combined with the door template to produce a description that a detector can use to recognize a door. Since the robot has many sensors, different templates will be instantiated depending on which sensor(s) are utilized for the perceptual task. Although the detectors used in the examples of this dissertation use one specific sensor, a sophisticated system with active sensors will need a *sensor planning* module to allocate sensors to perceptual tasks. Active sensors can be directed in response to perceptual goals. For example, if a camera is being used to track an obstacle on the left side of the robot, it can not be used to detect a doorway on the right side of the robot.

Individual perceptual templates are useful when perceptual knowledge is relatively sparse. But some knowledge bases may have much more perceptual information. For example, a hallway is a visually complex scene consisting of doors, walls, lights, floors, nameplates, branching corridors, trash cans, pedestrians, light switches and many other things. A naïve perceptual template for a hallway would generate too many perception tasks for the system to process. In this case it would be better to use a *perception schema* to specify a subset of the perceptual characteristics that are to be used to detect the object. While a template specifies important features of objects, a schema specifies a set of objects and the templates to use for those objects. The use of schemas should significantly reduce the perceptual cost for complex objects. A potential drawback of both templates and schemas is the possibility that the set of templates and schemas is incomplete.

### 5.2.3 Retrieval using viewpoints

In a situation where schemas or templates are not available or feasible, a knowledge base retrieval method based on *viewpoints* [Acker and Porter, 1994] could be used to collect a set of relevant objects based on dynamic goal-related criteria. Using a *perceptual viewpoint* to perform a memory retrieval will retrieve only (or mostly) objects relevant to perception. A viewpoint is similar to the schemas discussed above, but can be more abstract and more dynamic, and thus applicable to more tasks. Constructing the viewpoint templates is apparently the most difficult part of the process. It is possible that these could be dynamically generated by a "viewpoint expert". Further research in this area will determine how applicable this approach is to the attention problem.

### 5.2.4 Relevance to high-level attention goals

Section 2.6 lists four characteristics of human attention that are important for high-quality attention. They are: *sustained attention*, the ability to set perceptual goals and maintain them over time; *selective attention*, the ability to selectively filter input based on features such as color, size and shape; *divided attention*, the ability to pursue multiple perceptual goals in parallel; and *top-down control*, the ability to control perception based on high-level goals. The mechanisms described in this section, especially the SACK-based mechanism of Section 5.2.3, support all four of these characteristics.

In the SACK algorithm described in the previous section, perceptual goals are in effect as long as their corresponding behavioral goals are active, which is sustained attention. The templates and their associated detectors provide selective attention. The ability to activate multiple detectors and trackers provides divided attention. And the template, schema and spreading activation methods of producing perceptual goals all provide top-down control. I conclude that the CPA, in conjunction with top-down generation of perceptual goals, provides all of the key characteristics of human attention, and thus a solid foundation for general-purpose computational perceptual attention.

## 5.3 SENSORS

The `Flat` robot simulator provides several simulated sensors. In this example we will use absolute positioning and laser rangefinder information. In addition, we will use several streams of processed rangefinder data, including jumps (discontinuities in the data), line segments, blobs (clumps of rangefinder readings) and the robot's linear and angular velocities as determined from the readings of the absolute positioning sensor. The latter are provided by a set of utilities collectively known as the `dead-reckoning`.

### 5.3.1 Absolute positioning

The Flat simulator provides the absolute position and orientation of the robot in the simulated world[4]. The difference between successive readings can be used to determine the robot's linear and angular velocities. The system needs velocity information in order to predict the future position of objects based on their last observed position and the robot and object velocities. Flat is able to send position information at most every 100 milliseconds. In this example, let's assume that the navigator needs updated position information every 200 milliseconds (0.2 seconds).

First we'll define a class (see Figure 9) to represent the speed and position information calculated from the position information.

Next, we'll define the sensor that collects the position information from Flat and creates the incoming percepts. First, we create the position sensor (see Figure 10).

Next is the function to acquire position readings and create a position percept (see Figure 11). This function is used in the position sensor to acquire data and convert it to a percept.

[4] A real robot usually provides *odometry readings*, which are much less precise than absolute position information. However, at this time `Flat` does not provide standard odometry data.

```
T-position-percept
  Pos                ;; position (x y)
  Orientation        ;; angle (degrees)
  Linear-velocity    ;; meters/sec
  Rot-velocity       ;; degrees/sec
```

**Figure 9.** Contents of a position percept.

```
(make-instance 'T-sensor
  :name "Position sensor"
  :cpa    cpa
  :priority priority
  :generator-fn   #position-perception
  :generator-args  nil))
```

**Figure 10.** Creating a position sensor.

```
(defun position-perception (sensor &OPTIONAL
                      (interval 0.2))
  "Produces a list of (position linear-velocity rot-velocity).
Converts to meters."

  (let ((position  nil)
        (stream    (output-stream sensor))
        (data      nil))

    (loop
      ;;   "Returns (time x y z theta)  sec,mm,mm,mm,degrees"
      (setq data (cdr (get-robot-reckoning-position "Spot")))

     ;; We reflect across the X-axis to convert
      ;; the readings into a standard egocentric coordinate system.
      (when data
          (setq position
                (make-instance 'T-position-percept
                    :pos    (make-instance 'T-point-2d
                              :x (/ (first data) 1000.0)
                              :y (/ (- (second data)) 1000.0))
                    :orientation     (fourth data)
                    :linear-velocity  (/ (second
                                          (get-robot-reckoning-v
                                          "Spot"))   1000.0)
                    :rot-velocity    (- (/ (second
                                          (get-robot-reckoning-w
                                          "Spot"))   1000.0))))
      (store stream
             (make-instance 'T-percept
               :cpa         (cpa sensor)
               :sensor       sensor
               :timestamp   (get-time-milliseconds)
               :value        position
               :tag         :POSITION)))
      (sleep interval))))
```

**Figure 11.** Creating position sensor percepts.

Finally, we create a simple match function for the position detector. This function matches any position input so that it can be placed in the focus set and made available to other system components.

```
(defmethod position-match-fn
   ((matcher T-matcher) (percept T-percept) (detector T-detector)
    max-detection-time)

  (declare (ignore max-detection-time))

  ;; Always matches the position input.
  (instantiate percept detector *high-priority* :POSITION))
```

**Figure 12.** A match function for odometry.

### 5.3.2 Line segments from laser rangefinder data

The laser rangefinders in our robots (both simulated and real) provide both raw and processed data. One of the types of processed data consists of line segments extracted from the raw data. To the human eye, it is obvious which parts of the data form line segments, but it is non-trivial to compute them from the raw data [Guil et al., 1995]. In this example, we treat line segments as a separate sensor stream. The simulator will provide segments every 0.3 seconds.

As in the odometry sensor above, we'll first create a percept type to hold the line segments. The segments-percept contains the line segments as computed by the simulator, the same set of line segments converted to the robot's coordinate system, and the robot position at the time the segments were produced.

```
T-segments-percept
  Raw-segments
  Converted-segments
  Robot-pos
```

**Figure 13.** Contents of a line-segments percept.

```
(defmethod segment-match-fn-with-display
   ((matcher T-matcher) (percept T-percept) (detector T-detector)
    max-detection-time)

  (declare (ignore max-detection-time))

  ;; Always matches the set of segments input.
  (instantiate percept template *high-priority* :SEGMENTS)

  ;; Send the line segments to a display.
  (display-segments percept)
  )
```

**Figure 14.** A match function that also interfaces with a display.

The simple match function in Figure 14 is essentially identical to the position-match-fn shown above. In addition, the match function can be used to direct segments to a display as part of the user interface. Since all segments pass through the match function, it is an ideal place to add a call to a display function.

The function to retrieve laser segments has many parameters that control how line segments are extracted from the data. These parameters are passed into the rangefinder-segment-perception function, which collects the returned line segments and creates a percept.

```
(defun rangefinder-segment-perception
    (sensor min_theta max_theta k_max_distance
     k_discontinuity_thresh k_squid_window_size
     k_squid_window_std_dev k_colinear_range_thresh
     k_colinear_theta_thresh k_colinear_contig_thresh
     k_min_segment_length k_use_filter k_filter_window_size)

  (let ((segments      nil)  ;; from the sensor
        (raw-segments  nil)  ;; same data, but in object form
        (stream        (output-stream sensor))
        )

    (loop
      (setq segments
        (get-laser-segments min_theta max_theta k_max_distance
               k_discontinuity_thresh k_squid_window_size
               k_squid_window_std_dev k_colinear_range_thresh
               k_colinear_theta_thresh k_colinear_contig_thresh
               k_min_segment_length k_use_filter
               k_filter_window_size))

      (when segments
        (setq raw-segments (make-segment-objects segments))
        (store stream
             (make-instance 'T-percept
               :cpa       (cpa sensor)
               :sensor    sensor
               :timestamp  (get-time-milliseconds)
               :tag       :SEGMENTS
               :value     (make-instance 'T-segments-percept
                          :raw-segments raw-segments
                          :converted-segments
                              (fix-flat-segments-2d raw-segments)
                          :robot-pos
                          (fix-position
                           (cdr (get-robot-reckoning-position
                                (flat-data-robot-name *robot-state*))))
                          ))))

      (sleep 0.4)
      )))
```

**Figure 15.** Perceiving laser rangefinder segments.

### 5.3.3 Blobs from laser rangefinder data

A blob is defined as a set of points that are near each other but collectively far from other data points. Parameters control the formation of blobs by setting minimum and maximum values for size, maximum distance between the blob's points and minimum distance of the blob from other readings. The blob perception is almost identical to line segment perception, the major difference being a call to `get-laser-blobs` instead of `get-laser-segments`. Some objects that form blobs in the laser data include table and chair legs, human legs, lamp stands and smallish objects like robots and computers when seen from certain angles.

### 5.3.4 Doorways from laser rangefinder data

The laser rangefinder module provides a quick, but error-prone, method for detecting the possible presence of a doorway in the rangefinder field of view. It is based on the presence of a discontinuity, or *jump*, in the rangefinder data. At a doorway, the rangefinder readings will suddenly increase as the rangefinder beam sees through the doorway into the room or corridor beyond. This quick detection method has a high false positive rate, but a negligible false negative rate. We will use this test as a quick doorway detector, but the associated tracker will use a more accurate test involving line segments. The sensor to retrieve a list of possible doorways is nearly identical to the blob and segment sensors. The low-level function it calls is `get-laser-doorways`.

The figure below shows the sensor streams and object detectors in the system. The object detectors are described in the next section.

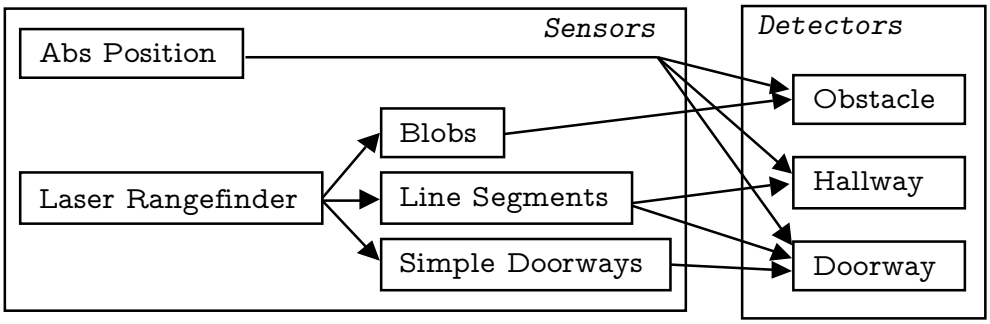


**Figure 16.** Sensor streams and object detectors in the system.

## 5.4 OBJECT DETECTORS

For this example, we need several detectors: a wall detector, a doorway detector, a box detector and a blob detector. First, we will create a couple of simple detectors that cause line segments and blobs to be displayed on the user interface as they are processed. These are shown in Figure 17 below.

As an example of a more complex detector and tracker, the door detector is presented below. It is one of the most complex detectors developed so far. The other detectors are similar in that they detect and track objects based on characteristics of one or more line segments or blobs.

### 5.4.1 A doorway detector and tracker

To detect doorways, we'll use two different methods. The laser rangefinder module provides a method for detecting doorways that is quick but also produces many false positives. We will use this method in the detector, but the tracker, which must be more accurate, will use a slower but more accurate method that examines line segments. First, we need a matcher:

```
(setq segment-matcher
        (add-matcher *my-cpa*
                  (make-instance   'T-matcher
                    :name          "segment-display-matcher"
                    :cpa           *my-cpa*
                    :match-fn      #'segment-match-fn-with-display)))

       ;; This matcher gets all the blobs and displays them
       ;; on the Flat display.
(setq blob-matcher
        (add-matcher *my-cpa*
                  (make-instance   'T-matcher
                    :name          "blob-display-matcher"
                    :cpa           *my-cpa*
                    :match-fn      #'blob-match-fn-with-display)))

(setq segment-detector
        ;; This detector triggers on any segment input
        ;; and sends it to the display.
        (make-instance   'T-detector
          :name         "Segment display"
          :cpa          *my-cpa*
          :sensor        segment-sensor
          :priority      1.0
          :matcher      segment-matcher))

(setq blob-detector
        ;; This detector triggers on any blob input
        ;; and sends it to the display.
        (make-instance   'T-detector
          :name          "Blob display"
          :cpa           *my-cpa*
          :sensor        blob-sensor
          :priority      1.0
          :matcher        blob-matcher))
```

**Figure 17.** Simple detectors that display blobs and segments.

Usually, the robot is looking for a specific doorway and knows whether it is on the right side of the hallway or the left side. A design decision is whether to distinguish between the right-side and left-side doorways in the matching function or whether to make the distinction at the detector level. If distinguished during matching, the two categories of doorways will be available to all detectors, possibly eliminating duplicate code in detectors, so we will use the matcher to do a quick check that determines which side of the hallway the doorway is on. The figure below shows a left-side doorway match function. It uses a simple doorway detector based on large jumps in the rangefinder data, as described earlier. The percept contains the angle of the jump and the distance to the jump.

```
;; Uses a tracker
(defmethod simple-left-side-doorway-match-fn
   ((matcher T-matcher) (percept T-percept) (detector T-detector))

  (let ((data (value percept)))
    (when (and (>= -25.0 (angle data) -110.0)
               (not (angle-masked-p detector (angle data))))

       (track percept detector *high-priority* :LEFT-SIDE-DOORWAY)
       )
     )
   )
```

**Figure 18.** A matcher function for left side doorways.

Note that this match function activates a tracker by using the `track` method rather than the `instantiate` method used in previous examples. Also, since it uses a tracker it checks to see whether a mask has been set on the doorway position before creating a tracker. This prevents a pathological situation where a detector repeatedly instantiates trackers for the same object. In this case, the mask is an egocentric angular range that covers the expected location of the doorway. It is adjusted each time the tracker is called.

The track method creates a tracker, which is a subclass of detector. In this case, it will create an instance of `doorway-tracker`, shown below. A doorway tracker is a subclass of `tracker` with two additional doorway-specific fields.

```
T-doorway-tracker
  Last-doorway-position
  Last-update-time
```

**Figure 19.** The `doorway-tracker` class.

After the tracker has been created, the CPA will call an `initialize-tracker` method, at which time the tracker can initialize itself. Now that we have some of the tracker foundation laid, let's look at the definition of the doorway detector:

```
(make-instance  'T-detector
         :name         "Left side doorways"
         :cpa           *my-cpa*
         :sensor         simple-laser-doorway-sensor
         :priority       0.85
         :matcher        left-side-matcher
         :tracking-fn    #'doorway-tracking-fn
         :tracker-type   'T-doorway-tracker
         :mask-fn        #'create-mask-for-doorway
         :sensor-fn      #'(lambda () segment-sensor)
         ))
```

**Figure 20.** A doorway detector.

This detector uses several fields not used in previous examples. The `tracker-type` provides the type of tracker to create for an object. The `tracking-fn` performs a function similar to the `matcher`, but in the context of a tracker. The CPA calls the `mask-fn` of a tracker once every attention cycle. The `sensor-fn`

indicates which sensor stream(s) the newly-instantiated tracker should be linked to. In this case, the tracker's sensor stream (segment-sensor) is different from the detector's sensor (simple-laser-doorway-sensor) because the detector uses the simple doorway detector while the tracker uses a more precise detector based on collinear line segments. The track method creates a new tracker object, calls instantiate-tracker and returns the new tracker, which is then added to the CPA.

Finally, let's look at the tracking function. As described in Chapter 4, once a tracker is created, its tracking function is called every attention cycle whether or not there is sensor data available. If data is available, the data is checked to see whether the doorway is still visible. If so, its position and the uncertainty in the position are updated.

If data is not available, or if the doorway is not visible, its current position is estimated based on the last known position and the robot's speed and direction. Appendix 2 discusses how to estimate the position of objects when the robot and the object are moving. In this case, of course, the doorway is assumed to be stationary. The doorway estimator contains code to perform the position estimation and to determine the uncertainty in the estimate.

```
(defmethod initialize-tracker ((this T-doorway-tracker)
                               (percept T-percept))
  "Initialize the tracker after creating it."

  ;; Percept has (theta jump-size) for a doorway
  ;; theta is in degrees.
  (setf (last-update-time      this) (timestamp percept))
  (setf (last-doorway-position this) nil)
  (setf (sensor this)                *segment-sensor*)
  (setf (uncertainty this)           0.0)

  ;; Set a mask.
  (let ((angle (- (car (value percept)))))
    (add-mask (detector this) this (list (- angle 15.0)
                (+ angle 15.0))))
  (resume *segment-sensor*))
```

**Figure 21.** The initialize-tracker method for the doorway tracker.

```
(defmethod doorway-tracking-fn
          ((this T-doorway-tracker) (percept NULL) id time)
  "Uses the percept to update the location, or creates a
virtual percept if none is provided.  Returns the instantiated percept
so it can be placed in the Focus Set.
Time is in ms."

  ;; if the uncertainty is too high, we may have lost
  ;; track of the data.  If so, deactivate the tracker.
  (when (> (uncertainty this) *uncertainty-limit*)
    (remove-tracker (cpa this) this)
    (return-from DOORWAY-TRACKING-FN nil))
  ;; otherwise, estimate the doorway's position at this time
  (estimate-doorway-position this id time))
```

**Figure 22.** The doorway-tracking-fn method for a NIL percept.

```
(defmethod doorway-tracking-fn ((this T-doorway-tracker)
                                (percept T-Percept) id time)
  "Uses the percept to update the location, or creates a
virtual percept if none is provided.  Returns the instantiated percept
so it can be placed in the Focus Set."

  ;; calculate the doorway position based on data.
  (setq doorway (find-doorway-in-segments this percept id time))

  (cond (doorway
          (instantiate
            (make-instance 'T-percept
              :cpa    (cpa this)
              :sensor (sensor percept)
              :timestamp  (get-time-milliseconds)
              :value  (list doorway-angle (/ doorway-dist 1000.0)))
            this *high-priority* :TRACKED-DOORWAY)))

        (T    ;; If we get here, we didn't find the doorway.
          ;; if we don't have a last-doorway-position, then
          ;; the possible doorway was a false alarm.
          ;; If so, delete this tracker.
          (cond ((null (last-doorway-position this))
                 (remove-tracker (cpa this) this)
                 )
                (T    ;; Otherwise, call the estimator
                  (doorway-tracking-fn this nil id time))))))
```

**Figure 23.** The doorway-tracking-fn method for real percepts.

## 5.5 NAVIGATION

The previous sections have described the sensors, detectors and trackers needed for robot navigation. To put it into action, we create an instance of the CPA as described at the end of Chapter 4 and set its timebound to 100 milliseconds. The navigation system can then retrieve an updated focus set every 100 milliseconds and use the contents in order to control the robot's motion. It uses the wall percepts to guide the robot's path down the hallway and the doorway percepts to determine a stopping point. As it detects obstacles, it must maneuver around them, and it can stop at fuel stations to recharge its batteries.

As discussed in Chapter 1, high-quality robot control requires very short perceptual latencies. In this example, we set a time bound of 100 milliseconds in order to produce high-quality control. However, the sensors do not provide input that fast, so the trackers need to have high-quality position estimators to support the controller.

## 5.6 SETTING THE ATTENTION POLICY

Chapter 6 discusses how to measure the performance of perception, which is affected by the attention policy or policies in effect during a task. Normally, a task will make several adjustments to the attention policy while it is being performed. The parameters of the attention policy were discussed in Section 4.2.5. The table below provides appropriate values for the attention parameters of this problem and briefly discusses the reasons behind each choice. Section 6.6 discusses ways to tune the attention policy if perception-related problems arise.

| | | |
|---|---|---|
| **Time bound** | 100 milliseconds | Must be short; perceptual latency affects robot control. |
| **Focus strength** | 0.6 | Too high a value risks ignoring unexpected input. |
| **Detection %** | 20% | Detectors are fast in this example; don't need much time. |
| **Sensor priority** | 1.0 | All sensors are equally important. |
| **Detector priority** | Obstacles: 0.95<br>Walls: 0.9<br>Doorways: 0.8<br>Boxes: 0.4 | Can't afford to hit anything, but can always back up to find a door. |
| **Prioritization functions** | `priority(x)`<br>`× uncertainty(x)` | For detectors, just use the detector priority. For trackers, use this equation. As an object's uncertainty rises, its tracker acquires a higher priority. |

**Table 1.** Suggested values for the attention policy parameters.

### 5.7 A SECOND EXAMPLE: A NEWSGROUP SCANNER

As another example, and to illustrate the flexibility of the attention mechanism, this section includes a completely different kind of application. The application searches newsgroups for job postings that mention Java or LISP. The sensor in this application reads messages from several `*.jobs` newsgroups. A percept is an entire message, stored as a list of strings, one line per string. Two detectors search for messages containing keywords such as "Java" and "LISP". The task is to find and print the articles containing relevant jobs. The first figure presents the match function from the matcher. The second figure presents the perception routine, which relies on a small set of functions to access articles via the NNTP protocol. The third figure presents the top-level application that uses the focus set to list relevant articles.

The match function below finds articles containing the word "LISP". Matchers for other keywords or sets of keywords would be nearly identical to this one.

```
;;; The article is a list of strings.
(defmethod lisp-match-fn
   ((matcher T-matcher) (percept T-percept)
    (detector T-detector))

  (let* ((data    (value percept))
         (article  (article data))   ;; a list of strings
         )

    (when (member-if #'(lambda (line)
                         (search "LISP" (string-upcase line)))
                     article)
      (instantiate percept detector *high-priority* :LISP)
      )))
```

**Figure 24.** The match function for messages containing "lisp".

```
(defun newsgroup-article-perception (sensor newsgroup-token)
  "TOKEN is a string which will be used to find
newsgroups.  Examples are \"jobs\" or \"mac\"."

  ;; Reads an article every 0.25 seconds.
  (let ((newsgroups   nil)
        (max-article  0)
        (min-article  0)
        (group        nil)
        (article      nil)
        (stream       (output-stream sensor))
        )

    (open-news "newshost.cc.utexas.edu")
    (setq newsgroups (filter-newsgroups (news-list) newsgroup-token))
    (close-news)

    (dolist (newsgroup newsgroups)
      (multiple-value-setq (group min-article max-article)
        (parse-newsgroup-info newsgroup))

      (when (> max-article min-article)
        (format *standard-output* "~2%Opening ~a, ~d articles"
                group (1+ (- max-article min-article)))
        (open-news)
        (get-article-range group)
        (loop
          (setq article (get-article))
          (store stream
                 (make-instance 'T-percept
                   :cpa       (cpa sensor)
                   :sensor    sensor
                   :timestamp  (get-time-milliseconds)
                   :value     (make-instance 'news-article-percept
                                :group    group
                                :article  article)))
          (unless (next-article)
            (return))
          (sleep *newsgroup-article-sleep-time*)
          )
        (close-news)
        )))

  (format *standard-output* "~2%Done with newsgroups"))
```

**Figure 25.** The perception function for the newsgroup scanner.

```
(defun job-scanner ()

 (let ((articles      nil)
       (lisp-articles  nil)
       (java-articles  nil)
       )

  (loop
    (setq lisp-articles  nil)
    (setq java-articles  nil)
    (setq articles (focus *my-cpa*))  ;; retrieve the focus set

    (setq java-articles
      (remove-if-not #'(lambda (article)
                         (eq (tag article) :JAVA))
                     articles))

    (setq lisp-articles
      (remove-if-not #'(lambda (article)
                         (eq (tag article) :LISP))
                     articles))


    (when lisp-articles
      (format *standard-output* "~%LISP: ~{ ~a~}" lisp-articles))

    (when java-articles
      (format *standard-output* "~%JAVA: ~{ ~a~}" java-articles))

    (when (or lisp-articles java-articles)
      (terpri *standard-output*))

    (sleep 0.3)
    )
  )
 )
```

**Figure 26.** The job scanner application.

# Chapter 6: Measuring Perception Performance

Chapters 4 and 5 describe and discuss various facets of the attention mechanism, including attention policies and how they can affect perception. This chapter discusses a method of measuring the quality of an attention policy while a task is in progress and methods for using the measure to learn or fine-tune an attention policy. This measure, called the Q-measure, evaluates the performance of perception for a given task. Since perception is managed by attention, the Q-measure is a useful measure of the quality of the attention mechanism.

Attention, as defined in this thesis, involves receiving sensor data, detecting and tracking objects that are contained in or computed from the sensor data, and filtering and prioritizing the resulting percepts. The best measure of the quality of attention would be one that compares the resulting prioritized list of percepts with a correctly prioritized list. However, the correctly prioritized list is difficult to define, and if we could define it and compute it efficiently we wouldn't even need an attention mechanism.

Another good measure is one that compares the set of objects that are detected or tracked against the desired set of objects. In a completely known world this provides an accurate quality measure. We can set up experiments in which the world is completely known, and use this measure to get an accurate evaluation of the quality of different attention policies. Even if the world is not completely known, this measure can be approximated. The Q-measure defined in this chapter is based on the ratio of detected objects to desired objects.

## 6.1 THE Q-MEASURE

The Q-measure defined in this section is a measure of the perceptual system's performance on a given task at a given time. For simplicity, we will drop the parameters and refer to it as simply $Q$. The table below lists the variables used in the formulas of this chapter. $N_d$ is the number of objects to detect. In a controlled environment, this can easily be determined. In a real-world task, this can be determined either from the number of active detectors or estimated from a model of the environment and the task. Similarly, $N_t$ is the number of objects to track. The variables $d$ and $t$ represent the number of objects actually detected and tracked, respectively.

| | |
|---|---|
| $N_d$ | Number of objects to detect. |
| $N_t$ | Number of objects to track. |
| d | Number of objects detected. |
| t | Number of objects tracked. |

**Table 2.** Values used when computing the Q-measure.

A simple form of the Q-measure is the ratio:

$$Q = \frac{d + t}{N_d + N_t} \qquad (1)$$

but that formula is limited in that it measures the quantity of perception, but not the quality. Each tracker, and in some cases each detector, measures the uncertainty in the characteristic (location, color, etc.) that it tracks. An object being tracked with a high uncertainty is of lower quality than an object being tracked with a low uncertainty. The uncertainty is a strictly task-specific value. For example, tracking a moving object requires the robot's speed and direction of travel, the object's speed and travel, as well as its location relative to the robot. If each of these has associated uncertainties, as is typical, the predicted position of the object will be an elliptical region. The area of the region is a measure of the uncertainty in the prediction, and thus in the ability to track the object.

We'll define *uncertainty* as a number from 0 to 1, where 0.0 represents absolute certainty, and *certainty* as the value one minus the uncertainty. We define the *certainty sums* $C_d$ and $C_t$ as the sum of the certainties of all objects detected or tracked, respectively.

$$C_t = \sum_{i=1}^{t} (1 - u(i)), \text{ where } u(i) \text{ is the uncertainty in object } i.$$

The Q-measure now becomes:

$$Q = \frac{C_d + C_t}{N_d + N_t} \qquad (2)$$

When all objects are detected and tracked with absolute certainty, this equation is identical to the first one.

A final modification of the Q-measure takes into account the value of an object. Valuable objects tracked with low uncertainty are much more important to a task than low-value or high-uncertainty objects. We define the value $v$ of an object to be a number from 0 to 1. It is typically a function of one or more of the priority of the sensor(s) that perceived it, the priority of the detector or tracker that is handling it, or other task-dependent characteristics. The sums $V_d$ and $V_t$ are computed using a combination of the objects' values and certainties using the following formula:

$$V_t = \sum_{i=1}^{t} v(i), \text{ where } v(i) \text{ is the value of the object.}$$

$$T_t = \sum_{i=1}^{t} v(i) * (1 - u(i)), \text{ where } v(i) \text{ is the value of the object.}$$

The final version of the Q-measure is now:

$$Q = \frac{T_d + T_t}{V_d + V_t} \qquad (3)$$

The value of an object will undoubtedly vary in relation to changing goals, so the Q-measure is not necessarily constant over time for a fixed set of objects and system load.

The following sections contain several experiments that use attention policies to handle different tasks and use the Q-measure to measure performance.

### 6.2 EXAMPLE 1: COMPARING THREE ATTENTION POLICIES

As described above, Q-measure can be used to measure the performance of an attention policy. Assuming that an application has good object detectors and trackers, the primary factor affecting perception quality is the amount of time available for perception. In the experiment described below, a robot moving into a room must detect and track twenty-two objects. In three runs described below, the robot uses three different attention policies while the Q-measure is recorded. The first policy is a default policy, where perceptual tasks are executed in the order received. This is unlikely to be a good policy for most applications. The second policy prioritizes detectors and trackers using their overall importance, and for trackers combines this with their uncertainty in the location of objects they are tracking. This results in much better quality perception under load. The third policy processes the detectors and trackers in a generic round robin fashion. The perception quality using this method is between that of the other two policies.

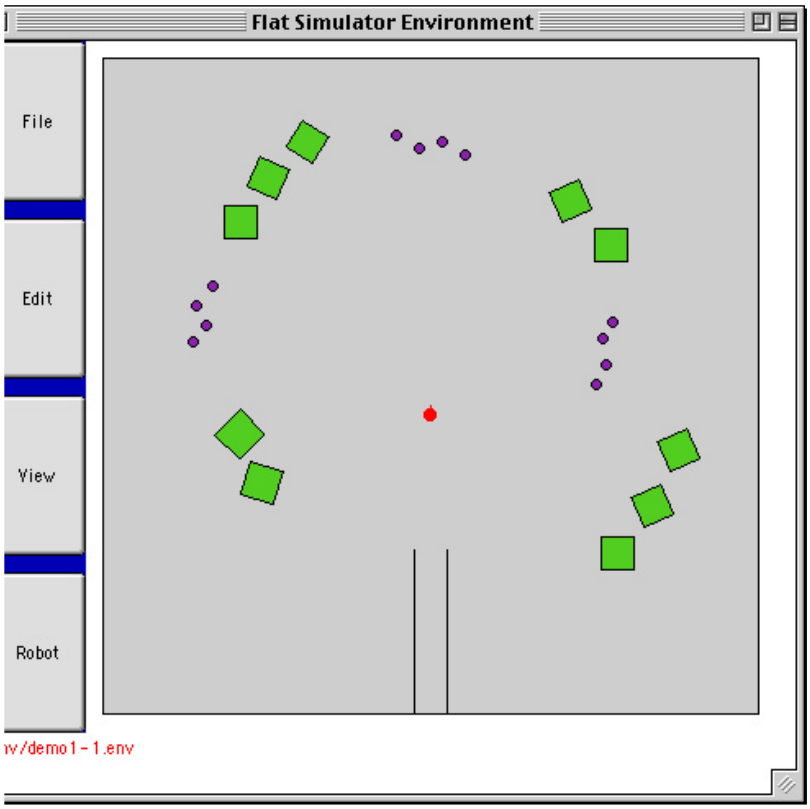


**Figure 27.** Test environment containing twenty-two objects.

In the experiment, the robot is placed at the lower edge of the room shown in Figure 27 above. It moves upward into the room, gradually emerging from the narrow corridor so that all of the objects are visible. There are two types of objects in the room, boxes and balls. In this experiment, both types of objects have the same priority. The simulation is running in the Flat robot simulator: the room is 20 meters on each side, the boxes are 1 meter square, the balls are about the same diameter as a human leg, and the robot is about half a meter in diameter.

In this example, detectors have no uncertainty and trackers have an uncertainty that increases linearly on each attention cycle in which sensors provide no data about the tracked object. When data arrives and the tracker can "see" the object again, the uncertainty is reset to zero.

As the robot moves out of the corridor and into the center of the room, it gradually detects and tracks each of the objects in the room. The Q-measure begins near 0, then moves toward 1.0 as the robot gradually perceives all of the objects. The experiment then artificially increases the time required to perform perception operations in order to simulate increased perceptual load. Perception loads of 2, 4, 6, 8 and 10 times normal are simulated, with each phase lasting 8 seconds. The perception load then returns to normal. The only difference in the two runs is the attention policy.

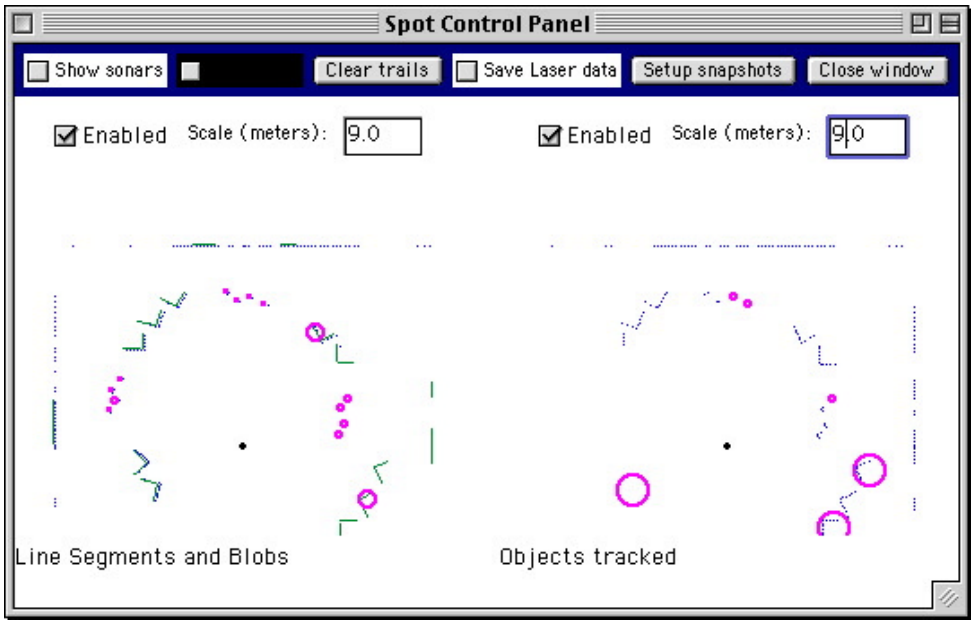


**Figure 28.** Tracking under heavy perceptual load.

Figure 28 shows the tracking display for this application as it appears when the system is under 8 times normal load. The left side shows the line segments and blobs that have been detected, superimposed on the rangefinder data. The front edges of the boxes are clearly visible and the circles indicate blobs. Note that two of the box edges, denoted by the larger circles, are also blobs. These are not confused with the balls in the room because they are larger than the known size of a ball. The right side of Figure 28 shows the location of tracked objects, denoted by circles, superimposed on the rangefinder data. It shows that although the sensors are detecting all 22 objects, the system only has time to track 6 objects with a 100ms timebound.

### 6.2.1 Run 1: a simple attention policy – no prioritization

The simplest attention policy maintains a list of detectors and trackers, and allocates time to them in the order they happen to appear on the list, until time is exhausted. The graph in Figure 29 shows the Q-measure over time for this attention policy. The Q-measure initially reaches a value near 1.0 around cycle 80. It then decreases in stair steps as the perceptual load increases, returning to a high value at the end of the experiment when perceptual load returns to normal. The Q-measure is not constant within each interval because the sensors are slower than the attention cycle, causing the tracking uncertainty to vary.

As perceptual load increases, the application doesn't have time to track all the objects and eventually loses track of all but a handful of the twenty-two objects in the room. Predictably, the performance decreases drastically as perception load increases. However, the next section discusses a better attention policy that maintains a higher quality of perception.

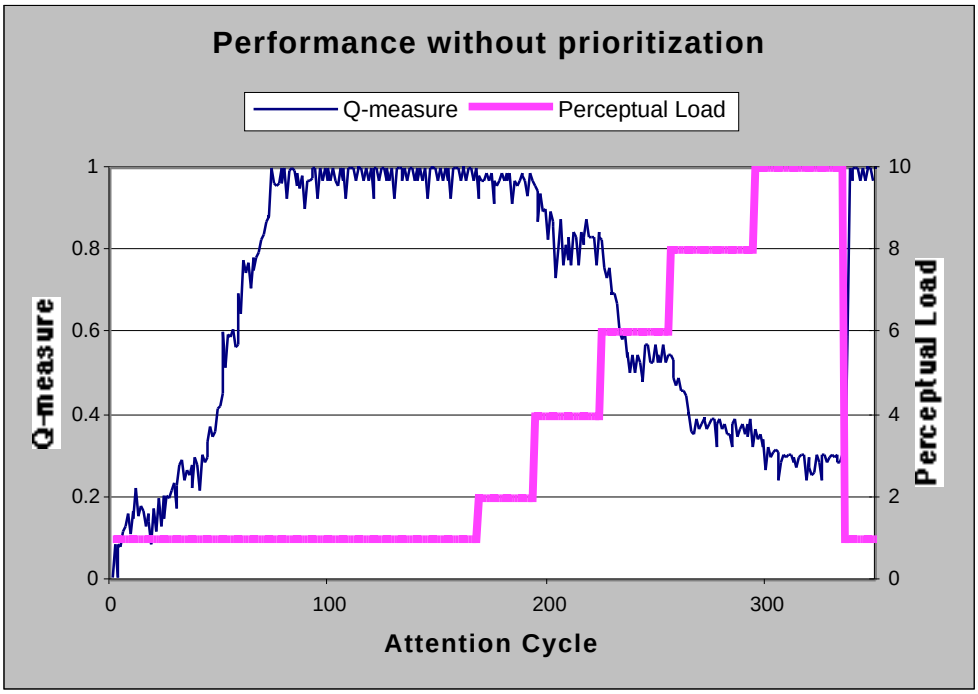


**Figure 29.** Perception performance without prioritization.

### 6.2.2 Run 2: prioritization using uncertainties

As described in Chapters 4 and 5, the attention mechanism allows an application to prioritize detectors and trackers using prioritization functions. The test described below uses an attention policy whose prioritization function takes into account the tracking uncertainty. The sorting order of a detector or tracker is determined by the product of its priority and its uncertainty. Under heavy perceptual load, only a few trackers can be called each attention cycle. Over time the uncertainty rises in trackers that are not called. As the uncertainty rises higher, those trackers rise in the sorted list of trackers until they are near the top. When they are called, their uncertainty drops to zero. This process results in a type of round robin scheduling where every tracker is called often enough so that all objects can be tracked at a moderate level of uncertainty. (A built-in round-robin scheduler would be as effective in this example, but would not be as effective in general usage because it would not take into account dynamically changing priorities. Also, moving objects have a higher uncertainty than stationary objects, meaning the uncertainties would change at different rates, something a round-robin scheduler is not prepared to deal with.) The perception quality rises due to better time allocation within the set of trackers. The figure below shows the Q-measure over the course of this run.

The stair step effect is reduced, although the variance at each level increases due to the fluctuating uncertainties. As Section 6.2.3 shows, the mean Q-measure in this run is significantly higher than that of the first run when the system is under load.

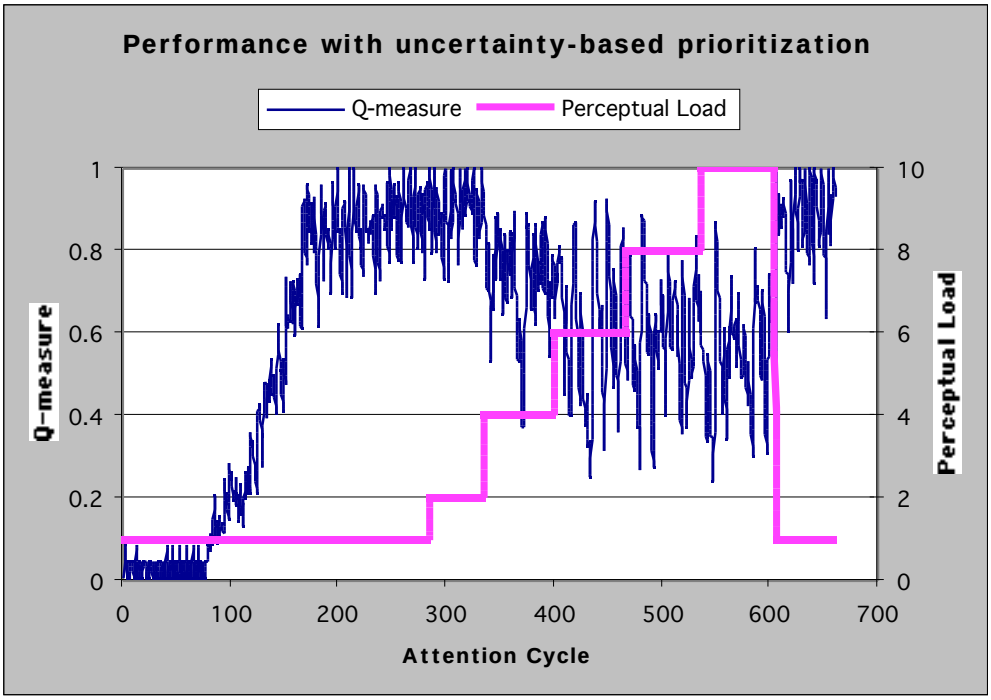


**Figure 30.** Perception performance with uncertainty-based prioritization.

These two examples illustrate several points. First, the quality of perception can become quite low under heavy time pressure. Second, the quality can be improved significantly by adjusting the attention policy. Third, the adjustable parameters of the attention mechanism are sufficient to allow scheduling rules such as round robin scheduling to emerge as one instance of an attention policy.

Table 3 provides the attention policy settings for this example.

| | |
|---|---|
| **Time bound** | 100ms (default) |
| **Focus strength** | 0.8 (default) |
| **Decay rate** | 0.554 (default) |
| **Detection %** | 20% (default) |
| **Default detector** | Default detector |
| **Sensor priorities** | HIGH: 0.8, LOW: 0.6 |
| **Priority of detectors and trackers** | HIGH: 0.8, LOW: 0.6 |
| **Prioritization functions** | Run 1: none<br>Run 2: p = Mean of the sensor and detector/tracker priorities.<br>Priority = p × uncertainty |

**Table 3.** The attention policy for Example 1.

### 6.2.3 Comparing the two runs

An additional test was performed in which both versions of the experiment were run 40 times. The changes in perceptual load were synchronized so that the runs could be overlaid for comparison purposes. In order to get comparable runs, care was taken to perform a full garbage collection between each run. This ensured that, although garbage collection was active during each run, it did not consume a significant amount of processing time. Also, both the LISP session and the Flat robot simulator were restarted at least every 4 runs. The mean Q-measure for each attention cycle, plus the standard deviation and a 99% confidence interval on the mean were calculated for each of the attention cycles across the 40 runs. The

results are shown below.

Figure 31 shows the mean Q-measure for the 40 runs of the default attention policy experiment described in Section 6.2.1. The averaged curve is a bit smoother than that presented in Figure 29 for one run, but it has the same general shape.

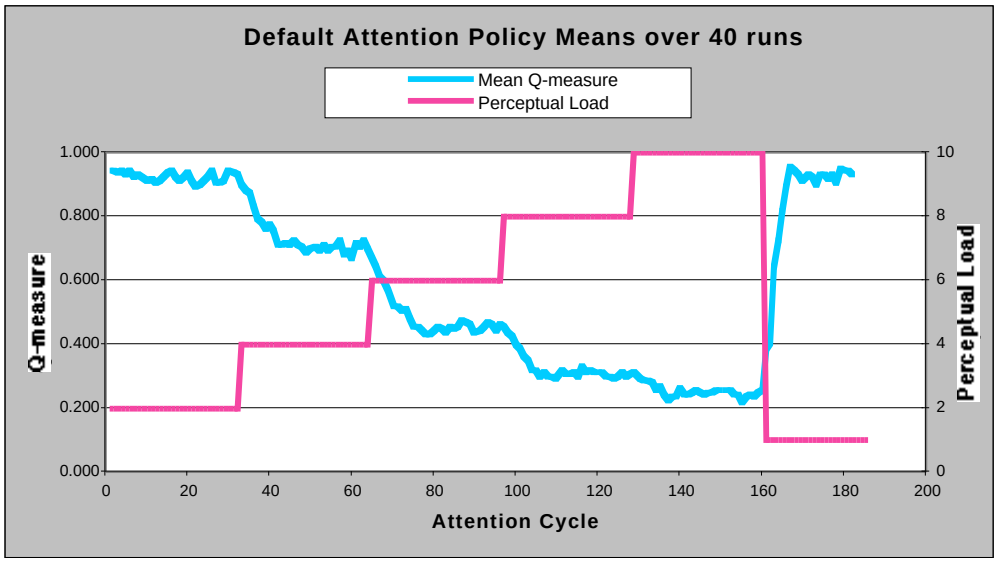


**Figure 31.** Mean Q-measure across 40 runs using the default attention policy.

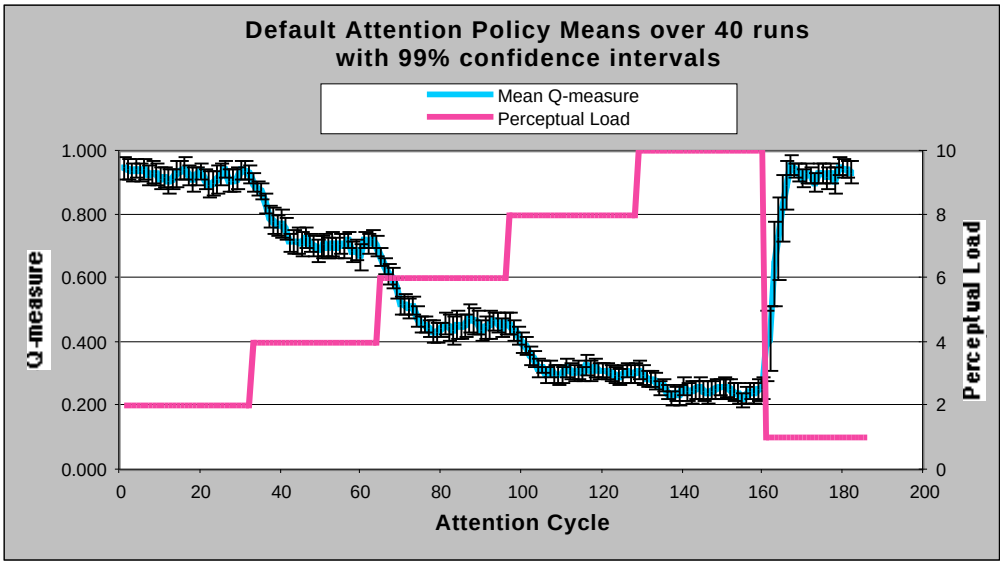


**Figure 32.** Mean Q-measure with confidence intervals across 40 runs.

Figure 32 presents the same data with error bars indicating 99% confidence intervals on each mean. As the graph indicates, the confidence intervals are fairly small.

Figures 33 and 34 show the same data for the experiment of Section 6.2.2, which uses a more sophisticated attention policy. Again, the mean data reflects the data from the original run shown in Figure 30. The mean of the Q-measure values under heavy perceptual load is above 0.4, as opposed to the mean from the runs using the default attention policy, which was just above 0.2. The confidence intervals for the second attention policy are noticeably wider than the confidence intervals for the default attention policy. This reflects the greater variability in the Q-measure using that attention policy.

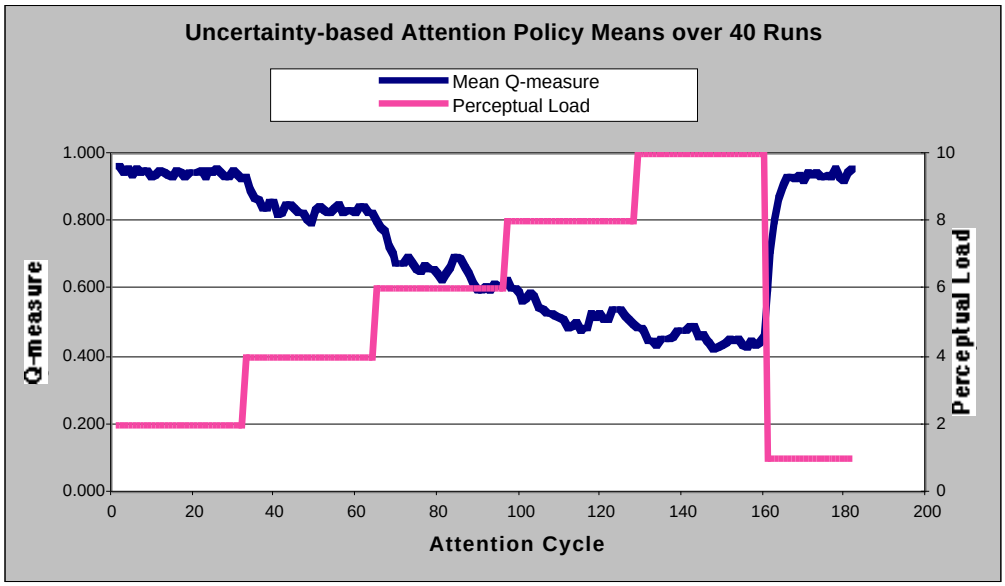


**Figure 33.** Mean Q-measure across 40 runs using the second attention policy.

Figures 35 and 36 present a combined view of both graphs, for comparison purposes. In Figure 36, the 99% confidence intervals are non-overlapping when the perceptual load is 4x or higher. This shows that the CPA can be used to implement attention policies that exhibit significantly different performance under load. The attention policies in these experiments are just two of the many different types of attention policies that can be implemented in the CPA.

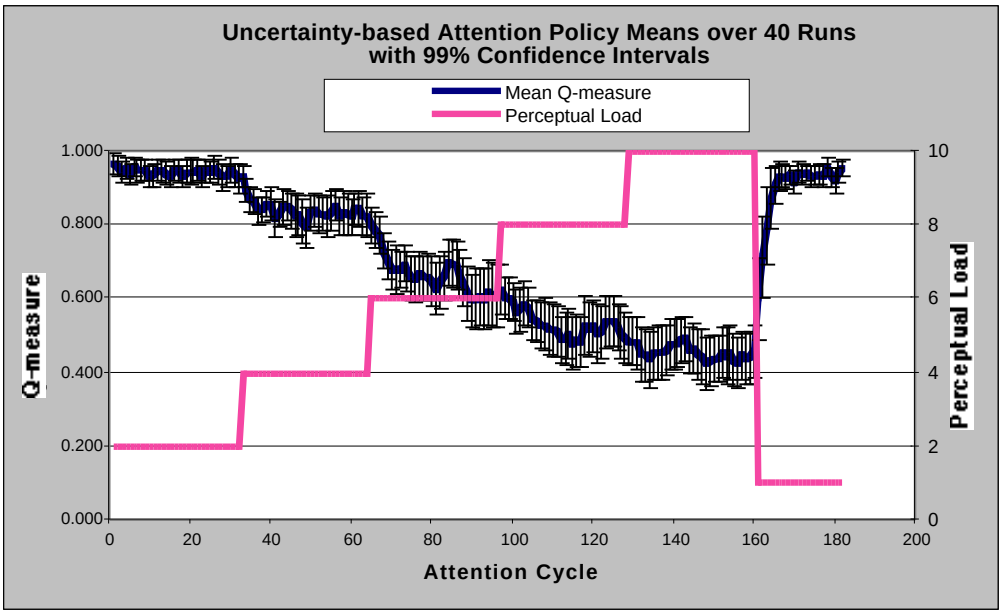


**Figure 34.** Mean Q-measure across 40 runs with confidence intervals.

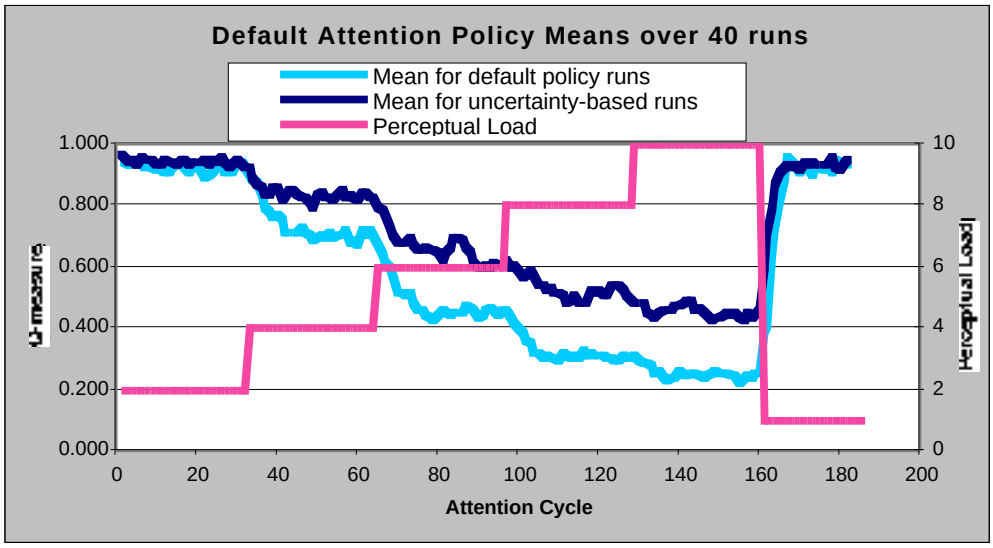


**Figure 35.** Combined graph of mean Q-measure for both experiments.

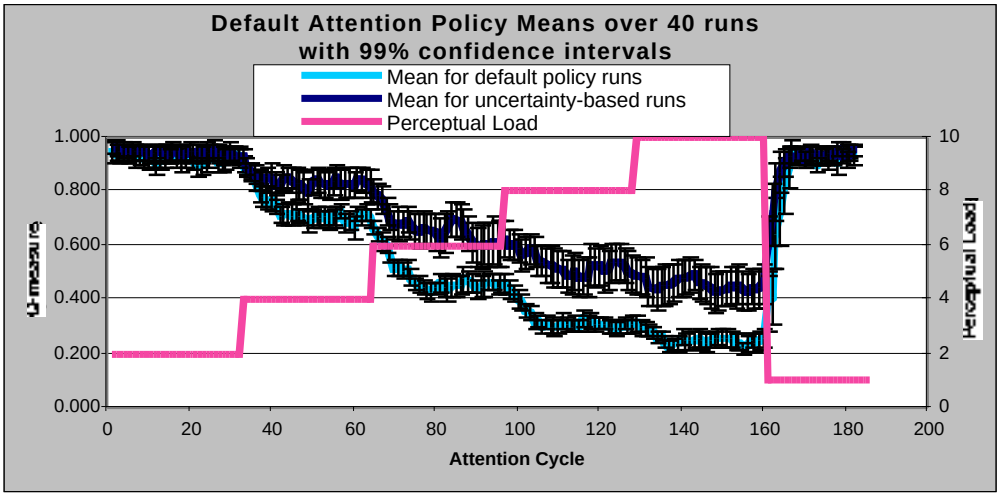


**Figure 36.** Combined graph with confidence intervals for both experiments.

The next three sections contain simpler examples that illustrate specific behaviors of the CPA.

### 6.2.4 Using a generic round robin policy

The experiment described above was also performed using a generic round robin attention policy. Under this policy, if the full set of detectors or trackers was not executed on one attention cycle, execution on the next cycle resumed at the point it left off. These results were combined with the results above to produce the graph in Figure 37. The corresponding graph with error bars is not presented because the error bars for the three lines form a nearly continuous region of black bars, rendering the graph unreadable.

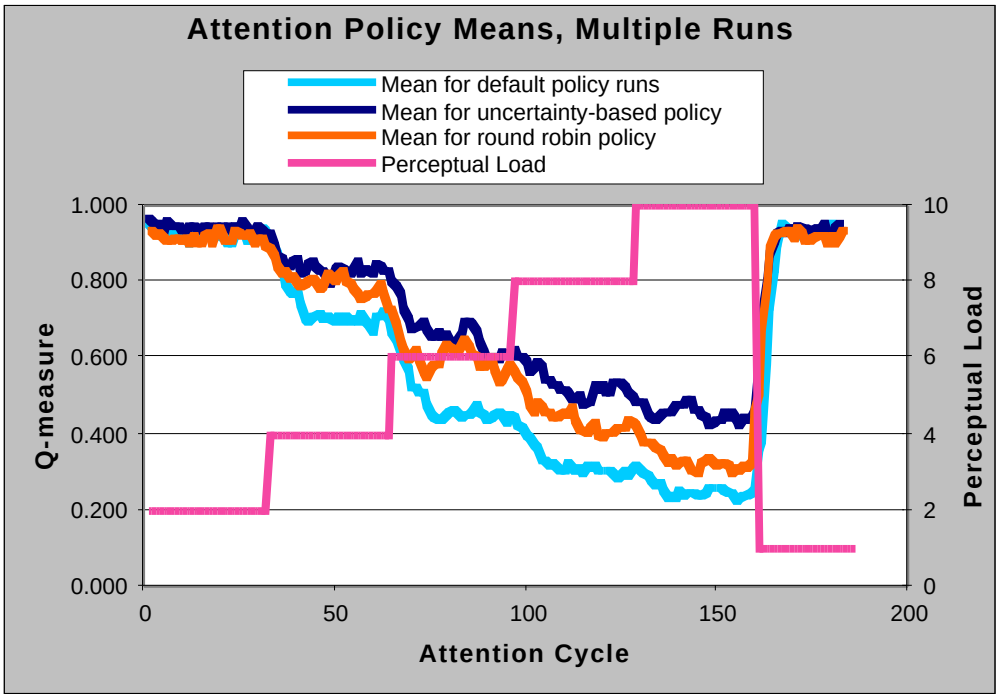


**Figure 37.** Combined graph for all three experiments.

A z-test shows that the round robin policy and the uncertainty-based policy are statistically different at 142 of the 181 data points (78.5%) and at every point in the interval of highest load.

### 6.3 EXAMPLE 2: FOCUS ON PRIORITY OBJECTS UNDER LOAD

This example illustrates the capability to prioritize perceptual input and the effect of that prioritization on the results of perception under heavy perceptual load. In this example there are two sensor streams. The objects in one are given a high priority (0.8) by their detector. The objects in the other have a lower priority (0.6). At the start of the run, the system is running at normal load (1x) and an equal number of objects of each type are present in the focus set (the output of the attention process).

As the run continues, the load increases to 2x, 4x, 6x, 8x and 10x as the system continues to process objects. Each phase lasts for 8 seconds, which is 80 attention cycles. As the table and chart below show, at a load of 2x, the attention mechanism is able to maintain the same level of performance (7 objects of each type). At 4x, however, it can not perceive all of the objects, so it continues to perceive 7 of the high-priority objects, but the number of low-priority objects drops to 3. Note that the relative object counts are a by-product of the prioritization and filtering process prescribed by the attention policy rather than by a deliberate decision to reduce the number of objects of a certain type.

At a perceptual load of 6x normal, the low-priority objects are rarely perceived. As the table and graph below show, the system can occasionally allocate time to perceive one low-priority item. The number of objects given in the accompanying table is an average over time, and therefore is fractional. At 8x and 10x the number of high-priority objects is decreased. When the load returns to normal, the system's performance almost immediately returns to the level achieved at the beginning of the run. The system is quite responsive to change.

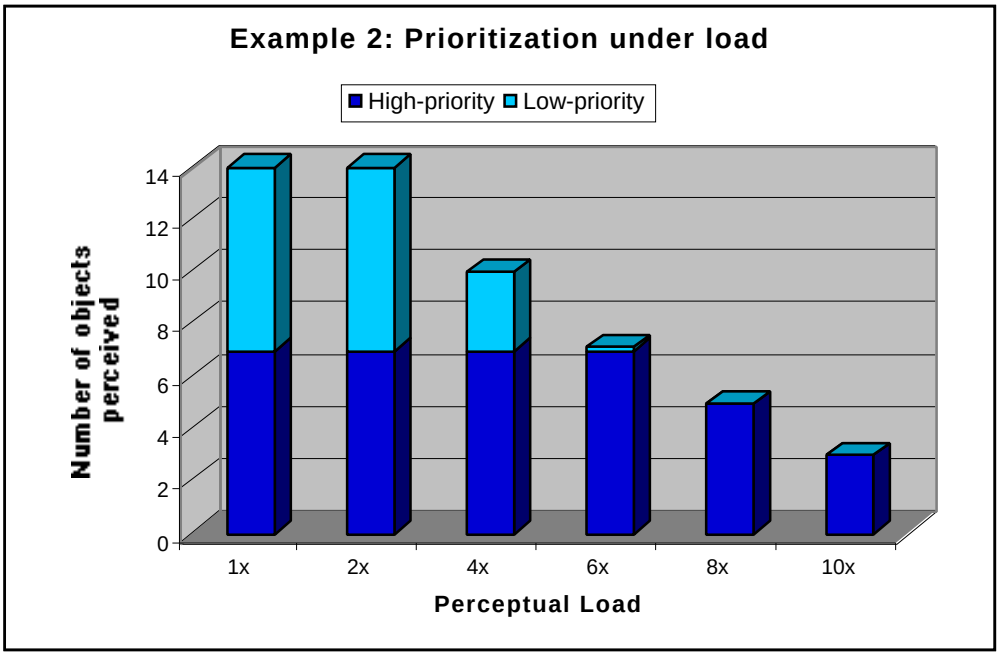


**Figure 38.** Graph of perceived objects for Example 2.

| *Perceptual Load* | *Number of high-priority objects* | *Number of low-priority objects* |
|---|---|---|
| ***1x*** | 7 | 7 |
| ***2x*** | 7 | 7 |
| ***4x*** | 7 | 3 |
| ***6x*** | 7 | 0.114 |
| ***8x*** | 5 | 0 |
| ***10x*** | 3 | 0 |

**Table 4.** Number of perceived objects for Example 2.

The attention policy settings for Example 2 are in the Table 5.

| | |
|---|---|
| **Time bound** | 100ms (default) |
| **Focus strength** | 0.8 (default) |
| **Decay rate** | 0.554 (default) |
| **Detection %** | 5% |
| **Default detector** | None |
| **Sensor priorities** | HIGH: 0.8, LOW: 0.6 |
| **Priority of detectors and trackers** | HIGH: 0.8, LOW: 0.6 |
| **Prioritization functions** | priority = Mean of the sensor and detector/tracker priorities. |

**Table 5.** The attention policy for Example 2.

### 6.4 EXAMPLE 3: HANDLING EMERGENCY INPUT UNDER LOAD

This example illustrates the ability to perceive high-priority sensor input that is only occasionally received. This example is based on the same system used in Example 2: one sensor stream containing high-priority (0.8) data and a second, lower priority (0.6), sensor stream. A third sensor stream contains

emergency priority (1.0) data, but only occasionally contains data.

During the example run, the system starts at a 1x load, and 7 objects are perceived from each of the two standard streams (HIGH and LOW in the chart of Figure 39). This is the phase labeled "Normal" in the chart. After a short time, the system load increases to 4x, which causes the number of low-priority objects to drop to 3, just as in Example 2. This is to illustrate responsiveness under load in the next phase. In the third, "Emergency", stage the EMERGENCY stream starts sending output, whereupon the system immediately perceives 7 objects from that stream (the maximum possible) as well as 3 objects from the HIGH stream. The LOW stream count drops to zero since there is no time left to perceive objects from it. After the emergency input is finished, the system immediately returns to the previous state, and then to the original state as the load drops back to 1x. The chart below shows the behavior of the system over time. The attention policy settings for this example are in Table 6.

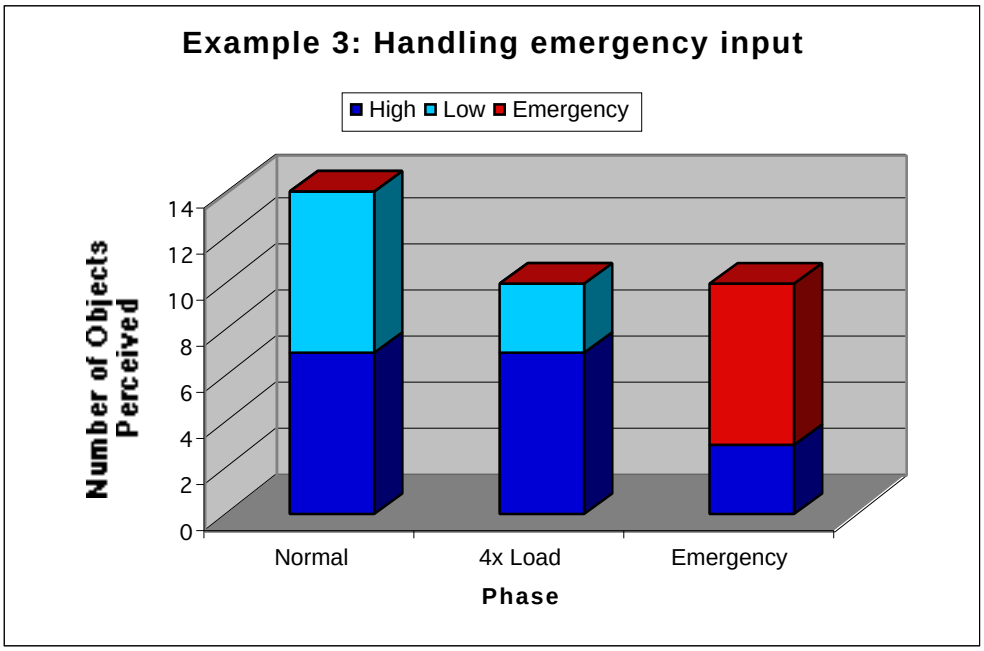


**Figure 39.** Graph of perceived objects for Example 3.

| | |
|---|---|
| **Time bound** | 100ms (default) |
| **Focus strength** | 0.8 (default) |
| **Decay rate** | 0.554 (default) |
| **Detection %** | 5% |
| **Default detector** | None |
| **Sensor priorities** | HIGH: 0.8, LOW: 0.6, EMERGENCY: 1.0 |
| **Priority of detectors and trackers** | HIGH: 0.8, LOW: 0.6, EMERGENCY: 1.0 |
| **Prioritization functions** | priority = Mean of the sensor and detector/tracker priorities. |

**Table 6.** The attention policy for Example 3.

## 6.5 EXAMPLE 4: ADJUSTING THE FOCUS STRENGTH

This example illustrates use of the `focus strength` parameter to adjust the activation level of extraneous input. As discussed in Chapters 2 and 3, people have varying abilities to ignore input that is not relevant to the current task(s). An extreme inability to ignore extraneous input can interfere with task performance and is called attention deficit disorder. At the other end of the spectrum is extreme focus, which is characteristic of autism. Each of these extremes can be useful at times, but is often a disability to "normal" processing. The CPA provides the `focus strength` parameter to allow extraneous input to be enhanced or suppressed as required by the current task.

The percepts in the focus set are sorted by activation level so that the cognitive component of the system can address the most important percepts first without expending resources to determine which are the most important. This example shows that by varying the `focus strength` parameter, the system can alter the order of percepts in the focus set in order to adjust the system's sensitivity to unexpected input.

Let's suppose a multi-sensor robot is moving down a hallway looking for a certain doorway. It's cameras are looking for the doorway, its rangefinder is detecting potential obstacles, its audio input system is listening for commands and its GPS, sonar and compass systems are producing readings at regular intervals. Extraneous input for this task would be anything not related to doorways or obstacles. If the activation of the extraneous input is too high, the robot might not have time to process doorway input, and thus miss the doorway. On the other hand, if it focuses too much on obstacles as detected by the rangefinder, a close reading by the sonar might be ignored, causing the robot to hit something. This example shows how varying the `focus strength` parameter affects the activation level of extraneous input

As in the last example, this example uses three input streams, in this case called HIGH, LOW and EXTRA. The system contains detectors for the HIGH and LOW streams, but none for the EXTRA stream, which is handled by the default detector and tracker. When the contents of the focus set are sorted, the EXTRA percepts in the focus set will be ordered according to their activation. The activation assigned to percepts by the default detector is `(1 – focusStrength)`, so the system can set the importance of extraneous input by adjusting the `focus strength` parameter.

In this example, the focus strength starts out at 0.5, which places the activation of the EXTRA percepts between that of the HIGH and LOW percepts. The focus strength is then set to 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 for five seconds each before returning to the 0.5 setting. In this system, each stream generates the same number of percepts.

The graph in Figure 34 shows the results of the test. This graph, unlike the previous ones, shows the mean activation of each type of percept in the focus set. For example, if at some time the focus set contains three high-priority percepts, the data point for that time is the average of the activations of each of those percepts.

After an initial phase with default settings, the system is first put under a 2x load to place some load on the perceptual system. The mean activation varies little in this phase.. After five seconds, `focus strength` changes to 0.0. In this phase, the activation of the EXTRA percepts is about 0.7, which is higher than the percepts from the other two streams. During the remaining phases the `focus strength` increases in steps. The activation of the EXTRA percepts drops in corresponding steps while the activation of the other percepts remains roughly the same. By the end of the run, the activation of the EXTRA percepts is much lower than the other percepts.

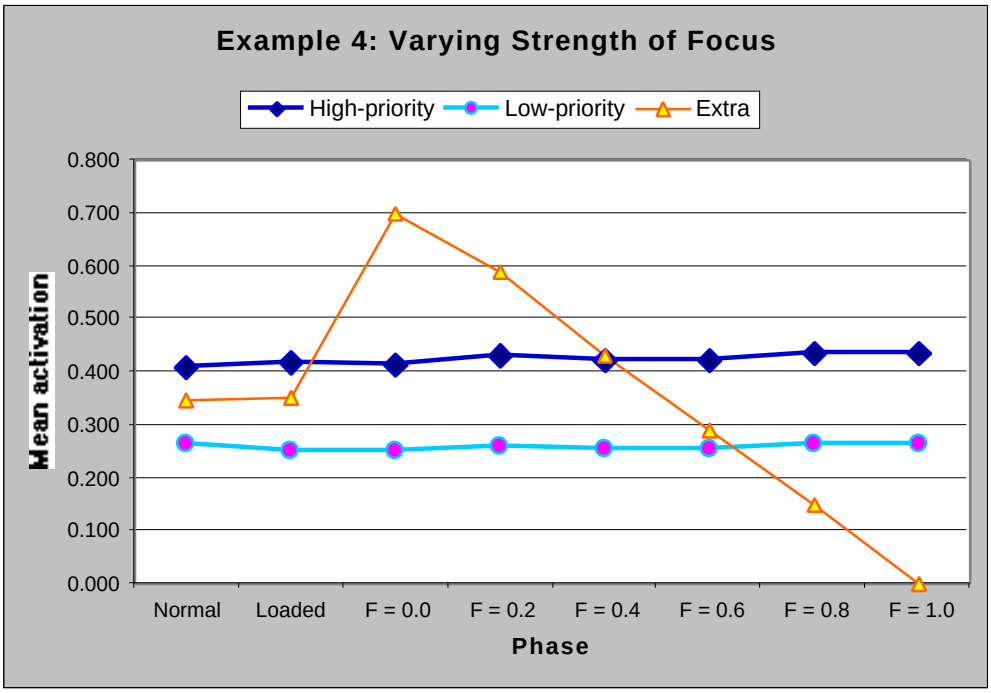


**Figure 40.** Activation of extraneous percepts varies with focus strength.

The attention policy for this example is in the table below.

| | |
|---|---|
| **Time bound** | 100ms (default) |
| **Focus strength** | 0.0, 0.2, 0.4, 0.6, 0.8, 1.0 |
| **Decay rate** | 0.554 (default) |
| **Detection %** | 5% |
| **Default detector** | Default detector/tracker |
| **Sensor priorities** | HIGH: 0.6, LOW: 0.4, EXTRA: 1.0 |
| **Priority of detectors and trackers** | HIGH: 0.6, LOW: 0.64, EXTRA: (1 – `focusStrength`) |
| **Prioritization functions** | priority = Mean of the sensor and detector/tracker priorities. |

**Table 7.** The attention policy for Example 4.

## 6.6 TUNING AN ATTENTION POLICY

This section discusses how to use the available attention parameters to improve an attention policy. Section 6.1 describes the parameters that affect the Q-measure, which are repeated in the table below. Section 5.6 lists the attention parameters that can be adjusted.

The attention policy may be modified in order to pursue a task-specific policy such as prioritizing input from one sensor, or in order to react to a problem such as perceptual overload. There are a variety of ways to modify the attention policy or alter a system in some other way to manage the problem.

Several common problems are: mis-prioritization of percepts in the focus set, trackers not called often enough, too many or too few trackers generated, and too many percepts in the focus set. There are many potential remedies within the CPA, including: adjust sensor priorities, adjust detector/tracker priorities, adjust the detection %, adjust the focus strength, change the prioritization functions and turn off sensors and detectors. Outside of the CPA, the system can also adjust by reducing its dependency on high data rates (e.g. the robot can slow down) or by adjusting sensor data rates. Below is a discussion of each of the problems and possible solutions.

1. **High-level tasks are not receiving enough percepts.**
   A task may perform poorly due to starvation: not receiving enough percepts. The required percepts may be in the focus set, but their activation is so low relative to the other percepts that the task is not always able to process them. Percept priority can be affected by sensor priority, detector/tracker priority, time allocated to tracking and the prioritization functions. If all percepts from one sensor are mis-prioritized, the sensor's priority can be adjusted. Similarly, if all percepts from one detector are mis-prioritized, the detector's priority should be adjusted. If the tracker prioritization function takes into account its uncertainty, the prioritization function can be changed or the uncertainty calculation may be incorrect. If it does not include uncertainty calculations, perhaps it needs to. Trackers that are called with no data will usually increase their uncertainty, which may negatively affect the resulting priority of their percepts. If the default detector/tracker is generating percepts, the CPA's focus strength may need to be adjusted or turned off. Another cause might be that the required percepts are completely missing from the focus set. In this case, the sensor may be malfunctioning, the detector/tracker may be malfunctioning, or the priority of the detector/tracker is so low that it is never called.
2. **Trackers are not performing well.**
   A good tracker requires a good perceptual model and a good motion model of the object it is tracking. If the model is not good enough, the tracker's uncertainty will rise, or it may lose track of the object completely. If the tracker is continually losing track of its object, perhaps the model needs to be improved. Even if its model is good, if a tracker is not called often enough with real data, the tracker may not have a high enough priority, the detection % may be too high, the sensor data rate may be too low or the timebound may be too short. The CPA does not provide a way to adjust sensor data rates, but the other problems can be addressed within the CPA. The first problem with inadequate tracking is that poor tracking leads to poor task performance; the robot can collide with a wall or can miss its goal. The second problem is that when trackers are not called their uncertainty rises to the point where they deactivate themselves, thus losing track of the object and requiring it to be detected again. In the system described in Chapter 5, detection is more efficient, but less accurate, than tracking. So if the trackers are not allocated enough perception time, the detectors will take over much of the perception task. They will detect many objects, most of which could be ignored with more analysis. The robot must slow or stop until the overload situation eases.
3. **Too many or too few trackers being generated.**
   Too many redundant trackers (tracking the same object) indicates that the masking function on the tracker is not correct. If a tracker is tracking an object, it should set a mask on its detector so the detector won't generate a duplicate tracker. Too many non-redundant trackers indicates perceptual overload. The system may need to reduce the data rate of one or more sensors, or shut them down completely. In addition, the tracking function can be adjusted to be more selective about what it tracks.
4. **The high-level program is unable to process all of the focus set.** Percepts are generated from detectors and trackers out of sensor data, so either there are too many detectors/trackers, too much sensor data, or the task is too slow. In any case, the system is not performing up to specification and perhaps should be scaled back by reducing the number of tracked objects, increasing the timebound or reducing the sensor input.

| | |
|---|---|
| $N_d$ | Number of objects to detect. |
| $N_t$ | Number of objects to track. |
| d | Number of objects detected. |
| t | Number of objects tracked. |
| A(i) | Activation of an object |
| C(i) | Certainty of a tracker or detector |

**Table 8.** Values used when computing the Q-measure.

The Q-measure components can also be used to adjust some CPA parameters. If the value $\frac{d}{N_d}$ is high, but $\frac{t}{N_t}$ is low, the system may need to allocate more time to tracking by decreasing the "Detection

%" parameter. If the reverse is true, the "Detection %" parameter can be increased. If both of those measures are low, the system is under severe time pressure. The "time bound" parameter can be increased, or the system may need to adopt less ambitious goals by decreasing the number of active detectors and trackers. The table below gives several other situations and how the CPA can be adjusted to compensate. This table does not discuss various object recognition difficulties, such as viewing angle and occlusion, which can also affect perception quality.

| *Indicators* | *Action* |
|---|---|
| $\frac{d}{N_d}$ is high, $\frac{t}{N_t}$ is low | Decrease *Detection %*. |
| $\frac{d}{N_d}$ is low, $\frac{t}{N_t}$ is high | Increase *Detection %*. |
| Both $\frac{d}{N_d}$ and $\frac{t}{N_t}$ are low | Increase *time bound*, if possible, or increase the sensor rates. |
| One class of object is ignored | Increase the priority of its detector or sensor. |
| An object's activation level is not correct | Have the tracker assign an activation that varies according to the object's characteristics, and use it for its own priority. |
| Activation levels are decaying too fast | Increase the decay rate constant or retrieve the contents of the focus set more often. |

**Table 9.** Hints for tuning the attention parameter

# Chapter 7: Related Work

There are a variety of intelligent agents that have real-time constraints and can or do utilize multimodal perception and perceptual attention, including agents for autonomous navigation [Pomerleau, 1993], medical diagnosis [Larsson et al., 1996], acoustic signal understanding [Lesser et al., 1995] and socially-aware robots [Breazeal and Scassellati, 1999].

Such systems include perception, cognition and action components; the overall architecture and the details of each component are discussed in [Hayes-Roth, 1990]. This dissertation focuses on the interface between the perception and cognition components, with the understanding that both perceptual input and the perception-cognition interface are affected by past, current and future actions.

A typical agent has the characteristics shown in the figure below. Multimodal sensors send data into the perception subsystem, where they are filtered and prioritized using filters supplied by a cognition module. In order to meet real-time constraints, a scheduler controls the execution of filters. The policies of the scheduler are dictated by the attention policy in effect. The cognition subsystem then receives the percepts and acts upon them. In a complex dynamic system, some of the actions will alter the attention policy or the set of active filters.

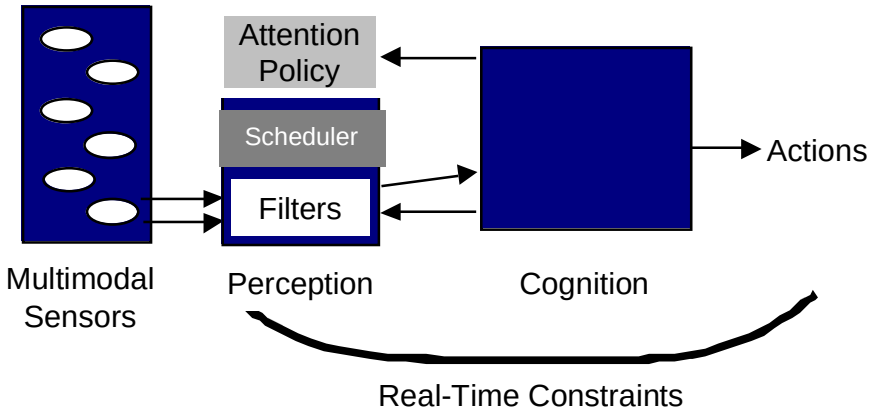


**Figure 41.** A typical real-time intelligent agent.

Whether an agent is *multimodal* is ambiguous at times. In human perception, the term is used when input is aggregated across completely different sensors, such as the eyes and ears. However, a robot has a much wider variety of sensors available. When it uses two different types of sensors, for example visual and acoustic, it is clearly multimodal. However, systems such as radar-based trackers sometimes use very similar radars operating at different frequencies. These are termed multimodal systems in the literature because sensor fusion techniques are applied to the two inputs, much as they would be to inputs from two completely different sensors. In general, if two sensors require substantially different filtering and processing techniques or require complex fusion techniques, they can be treated as separate modes.

Agents also have many different types of *filtering*. For a few simple problems, an agent may require no filtering at all. Those agents that do require input filtering to prevent perceptual overload can use a static set of filters or can dynamically create sets of filters in order to satisfy high-level goals. In addition, filters may be memory-less or memory-based, depending on whether they require knowledge of past inputs.

If a set of filters is required, the agent will use an *attention policy* to prioritize them in relation to high-level goals. For simple problems the attention policy can be static, but more complex problems need dynamic attention policies.

The primary component of the *real-time constraints* is the response interval, which determines how much time an agent can take to formulate a response to an input. Part of the response interval is allocated to perception, and the rest to cognition. An agent's performance typically deteriorates, sometimes disastrously, if the response interval constraint is not met.

The CPA mechanism described in this dissertation does not constrain the sources of input, so it is useful with any type of input, including multimodal input, and has architectural features for efficient

distribution of input data to filters. It has been demonstrated in applications that use numeric, symbolic and text input.

The CPA allows dynamic filter sets and includes both memory-less filters (detectors) and memory-based filters (trackers). External actions alter the active set of filters.

The CPA allows a wide variety of dynamic attention policies for prioritizing filters. Once the filters have been prioritized, it provides an efficient mechanism for scheduling and executing the filters under soft real-time constraints.

The real-time constraints allowed by the CPA include an overall time bound on perception processing, as well as individual time bounds for the sets of memory-less and memory-based filters. The CPA scheduler is non-preemptive, so it can not enforce hard real-time bounds, but it provides each perceptual task with the maximum time it can execute before it must return an answer. This form of "cooperative" real time requires individual tasks to be well-behaved.

## 7.1 EXEMPLAR PERCEPTIVE INTELLIGENT AGENTS

Section 3.2 discussed several perceptual attention mechanisms that have been designed for specific applications. This section discusses several complex agents that use or could use a perceptual attention mechanism.

### 7.1.1 Guardian

Guardian is a real time agent that accepts data from numerous monitors attached to a patient in an intensive-care unit (ICU) and formulates recommendations for treating conditions that develop [Larsson et al., 1996]. Guardian is one of the most sophisticated real time perceptive agents in existence. In terms of the agent description at the beginning of this chapter, the perceptual components of Guardian are described in the table below.

| *Component* | *Type* |
|---|---|
| **Sensors** | Numerous multimodal sensors producing numeric data. Interaction with the nurse/physician is handled by a different mechanism. |
| **Filters** | Memory-based to compute numeric trends (e.g. rising, falling). |
| **Attention Policies** | Dynamic, in response to diagnostic goals and system load. Size of input buffers can be varied in order to limit processing of old data. |
| **Real-time constraints** | The system must complete the diagnosis in time to treat the patient before he/she dies (typically a few minutes). Perception during individual diagnostic steps must be timely. |

**Table 10.** Perception characteristics of Guardian.

Guardian has time to process less than 10% of its input. Consequently, it relies heavily on intelligent filtering of the numeric input data. Its `Focus` module uses a process called *dynamic thresholding* to filter inputs unless they exceed a threshold value, which is adjusted in response to input load and behavioral goals [Washington, 1990]. The filters calculate short-term and long-term trends (e.g. rising, falling) which are attached to the incoming percepts. In Guardian, the output of the filters is integrated into the system as a set of *perceptual events* that are processed with the other, internally-occurring, blackboard events [Hewett and Hayes-Roth, 1989]. In clinical tests, Guardian performed extremely well when tested against nurses and doctors in a real ICU.

Guardian's `Focus` module contains nearly all of the functionality of a complex perceptual attention mechanism like the CPA. Perhaps its only real limitation is the separation of its numeric input from textual dialog input. The former is handled by the attention mechanism, while the latter is not. The CPA

would offer an integrated mechanism for both types of input as well as time constraints on individual perceptual tasks.

### 7.1.2 IPUS

IPUS (Integrated Processing and Understanding of Signals) is a complex system for understanding acoustic signals of all kinds [Lesser, et al., 1995]. It accepts a block of input representing an acoustic signal over a period of time. The length of a block is unspecified, but the waveform examples in the cited paper are typically 1-4 seconds long. The goal is to recognize different sounds in the waveform, such as telephone ringers, alarm buzzers, car horns and sounds of glass clinking. The table below lists the perceptual attention characteristics of IPUS.

| *Component* | *Type* |
|---|---|
| **Sensors** | Anything that produces waveforms. Typical input is an acoustic signal over a period of time. |
| **Filters** | Complex, memory-based, parameterized signal processing modules that can be activated, tuned and deactivated. |
| **Attention Policies** | Dynamic, in response to goals. Consists of the parameters to the filters. |
| **Real-time constraints** | No formal real-time constraints. |

**Table 11.** Perception characteristics of IPUS.

IPUS is designed to detect and correct errors produced by mis-tuned or inappropriate front-end signal processing modules. Consequently, a great deal of its time and effort is spent adjusting the set of filters and the attention policy.

Like many other systems, IPUS contains precisely the attention mechanism it needs, but no more. Integrating the CPA into IPUS would add extra functionality, especially multimodal input and real-time constraints, that might be useful in future applications. However, IPUS does not appear to need the extra functionality at this time.

## 7.2 BLACKBOARD-BASED CONTROL METHODS

The literature on blackboard systems contains a fairly large body of work that discusses "focus of attention", which refers to methods of selecting the most relevant action to perform [Hayes-Roth, 1993; Xu and Van Brussel, 1997]. I categorize this as *behavioral control*, not perceptual attention. Some of these methods are applicable to real-time systems [Garvey, 1993]. The question arises whether behavioral control methods can be used for perceptual attention.

Garvey's paper divides real-time behavioral control methods into two groups: those that have multiple methods (fast and slow) available, and those based on anytime algorithms, in which processing can be interrupted at any time and ordered to produce the best result available.

Any filter in a perceptual attention mechanism could use multiple methods based on available time. In fact, this method is used in several of the example systems in this dissertation. Standard anytime algorithms might be useful in memory-based filters. Typical blackboard-based control algorithms provides the following perceptual features:

| *Component* | *Type* |
|---|---|
| **Sensors** | Data are instances of a fixed set of actions. Actions may differ, but have the same interface. |
| **Filters** | Preconditions filter actions and control heuristics prioritize them. Mostly memory-less, although a few are memory-based. |
| **Attention Policies** | Dynamic, in response to goals. Consists of the set of control heuristics. |
| **Real-time constraints** | Some systems support response interval constraints on action selection. |

**Table 12.** Perception characteristics of blackboard control architectures.

When viewed in this light, the behavioral control mechanisms offer many of the same features as perceptual attention mechanisms. The main differences are in the input, which has less variety and is usually not considered to be multimodal; and in the real-time constraints, which typically consist of a global response time that is not subdivided further.

The other possible limitation is that few, if any, blackboard control mechanisms have been used in situations that require response intervals of 100 ms, as is required for robot navigation. They are more typically applied to large-scale control problems where their performance and behavior are a more appropriate fit. I do not know of any blackboard control implementations that operate at very short time scales.

### 7.3 DISTRIBUTED PERCEPTION IN EXPERT SYSTEMS

Of interest is the ABE/RT toolkit for building real-time expert systems, which takes a different approach to perception [Lark, et al., 1990]. This toolkit assists in constructing and testing hard real-time systems such as pilot assistants. It does not contain a centralized mechanism for perception and attention, except for a component that facilitates sharing input among different processes. Instead, it schedules high-level processes and counts on each of them to know its own perceptual time constraints and to adhere to them.

This approach, while less complicated for the system architecture, might lead to an unwieldy distribution of knowledge about the importance of various percepts. It is unclear how well this design turned out since the cited paper contains no evaluation and the software is no longer sold.

### 7.4 MULTITARGET MULTISENSOR TRACKING

Applications such as air traffic control, battlefield surveillance and air defense all require identification and tracking of multiple targets [Mazor et al., 1998]. Many of the applications combine data from multiple, usually identical, sensors that are scanning contiguous or slightly overlapping spatial areas. A few use non-identical sensors, such as radars that scan at different frequencies [Zhang et al., 1997]. Sensor data arrive very slowly, often have a low signal-to-noise ratio, and are sometimes incomplete [Stone et al., 1999]. Because the noise level is high, the data can be interpreted in many ways, and it is hard to tell which interpretation is correct. Consequently, Multiple Hypothesis Tracking (MHT) [Reid, 1979] is the most common approach to the problem of tracking multiple targets, while Kalman filters are used to handle noisy or missing data [Ramachandra, 1990]. In MHT, multiple possible tracks for each target are maintained concurrently, with each track having an associated probability. MHT is computationally complex and one of the primary goals of an efficient algorithm is to prune unnecessary tracks [Kurien, 1990].

Due to the complexity of the problem, one might suppose that multitarget tracking (MTT) methods would use perceptual attention techniques. However, the commonly-used algorithms assume that all of the data from each sensor sweep is of equal value and that all of it must be processed. Consequently, the focus of research in this area has been on faster, non-optimal algorithms, e.g. [Blair and Bar-Shalom,

1996; Bethel and Paras, 1998; Saha and Chang, 1998]. The perceptual characteristics are summarized below.

| *Component* | *Type* |
|---|---|
| **Sensors** | Various types of radar and sonar producing very noisy data. |
| **Filters** | Spatial filters only. All spatially relevant data are analyzed. |
| **Attention Policies** | Typically static for each problem. |
| **Real-time constraints** | Typically 2-10 seconds, the interval between radar scans. |

**Table 13.** Perception characteristics of multitarget tracking.

Track pruning techniques do contain some methods that fall into the category of perceptual attention. Kurien describes four general methods for eliminating unlikely hypotheses, some of which are related to perceptual attention:

1. **Gating**. This corresponds to spatial attention. The path of the target is projected forward in time. Tracks that do not fall within the projected path are pruned.
2. **Clustering**. Readings are clustered into small groups and the most likely track is selected from the small cluster rather than from all of the tracks. This is solely for computational efficiency. If the clustering is spatial, this has some resemblance to spatial attention.
3. **Classification**. Pruning tracks with a low probability. Since tracks are the result of processing during target tracking, as opposed to being directly perceived, this is difficult to classify as perceptual attention, although it is somewhat analogous to sorting detectors by their priority.
4. **N-scan approximation**. Basing a track on the last *n* sensor inputs rather than on all the scans, for computational efficiency. This is not related to attention.

These are the only methods in multitarget tracking that contain a hint of perceptual attention. One reason is that standard radars scan an area only once every 10 seconds. At that data rate, it is important to use as much data as possible to track objects. Even at that speed, MHT methods are barely fast enough to cope with the data. However, if faster radars are developed, the processing methods might need to use perceptual attention techniques to help cope with the higher data rates.

## 7.5 PROCESS SCHEDULING

One component of the CPA is the perceptual task scheduler. It is useful to compare this scheduler to other existing schedulers.

### 7.5.1 The CPA task scheduler

The CPA's scheduler was selected based on three primary criteria: only one task will be active at any time (since the CPA is designed for a single-processor system, this efficiently utilizes the processor); it is more important to execute high-priority tasks than to execute all the tasks; and task priorities should be dynamically adjustable. These criteria led to the development of a scheduler based on dual priority queues [Sedgewick, 1999], one for the set of detectors and one for the set of trackers. Operating systems typically use several queues for keeping track of active jobs [Buttazzo, 1997]. One component of the attention policy tells the scheduler how to divide perception time between the two queues. The scheduler executes tasks from the tracker queue first, stopping when time expires or when it reaches the end of the queue. It then uses the remaining time to execute tasks from the detector queue, again stopping when time expires or it runs out of tasks. At the beginning of each cycle, the scheduler adjusts the priority of active tasks according to the active attention policy.

As shown in Chapter 6, the dynamic attention policy allows a wide variety of scheduling behaviors. However, it is not apparent from the experiments presented whether the scheduler itself is an optimal

design. The sections below compare the CPA's scheduler to alternative schedulers and to other types of systems that perform various types of dynamic-priority scheduling with real-time constraints, including real-time operating systems [Liu, 2000].

### 7.5.2 The UNIX operating system scheduler

The UNIX operating system provides for multiple simultaneous processes and contains an efficient scheduler. Priorities of processes are dynamically adjustable and each process receives time quanta in proportion to its relative priority [Srinivasan, 1998]. Indeed, it has been suggested that the CPA should simply use the UNIX scheduler instead of its own. However, I have found that the UNIX scheduler does not meet the design criteria specified in Section 7.5.1 above and performs much worse on the perception examples than the CPA scheduler. I found that the user can not directly control actual process priorities, and that the UNIX scheduler does not allow us to guarantee that high-priority tasks will receive as much CPU time as possible.

While a UNIX user (or an attention policy) can alter a process's '`nice`' parameter in order to adjust the amount of CPU time it receives, UNIX maintains an internal priority for each process that is inaccessible to the user. The '`nice`' setting is just one parameter of the internal priority calculation; the other parameters include the amount of CPU time the process has used recently (a higher amount decreases the priority) and the number of times the process has been passed over for scheduling (a higher amount increases the priority). Experiments with a pair of processes, one high-priority and one low-priority show that the internal priority differs significantly from the '`nice`' setting and in fact will occasionally invert, with the low-priority task receiving an internal priority that is higher than that of the high-priority task. The conclusion is that the user (or attention policy, in this case) can not directly set process priorities in UNIX.

However, the other problem is worse: the user can not turn off round robin scheduling in UNIX, and the CPU usage of high-priority tasks is not preserved when additional low-priority tasks are activated. The consequence of this is that high-priority tasks receive less and less CPU time as load increases. A simulation of a simplified version of the robot experiments from Section 6 showed that under normal load the highest-priority task received only 33% of the CPU time, even when all the other tasks were set at the lowest possible priority. As a consequence, the highest-priority tasks takes three times as long to complete as they should. Under higher than normal load, CPU time would decrease farther and it is possible that the task would never complete within the time bound.

This violates the second design criteria, which says that it is more important to execute high-priority tasks than to execute all of the tasks. Under load, the UNIX scheduler will allocate time to tasks that will never finish, thus wasting perception time. Round robin scheduling can not be turned off in standard UNIX systems.

### 7.5.3 UNIX-compatible and non-UNIX real-time OSes

The problems noted above, as well as several other serious problems, have also been noticed by others [Yodaiken, 2001]. There are now packages available that replace the UNIX scheduler, memory manager and other system modules in order to support hard real-time scheduling. A prime example is RTLinux [Yodaiken, 2001]. There are also non-UNIX real-time operating systems available, notably OS-9 [Microware, 2000], Lynx [LynuxWorks, 2000] and OSE [Paul, 2001]. There are far too many of both types of systems to catalog here, but useful overview lists are available at [Chandana, 2001] and [Yodaiken, 2001].

An attention mechanism running on a real-time operating system could use the operating system scheduler to control the tasks. This would provide hard real-time guarantees with the added overhead of dealing with unschedulable sets of tasks.

However, real-time operating systems use a different kind of scheduler. The scheduler first determines whether a set of tasks is *schedulable*: that is, whether the set of tasks can be completed while meeting real-time deadlines. Most production real-time systems deal with a fixed set of static priority tasks that must be run at regular intervals. The set of tasks and constraints can be analyzed and scheduled before the tasks begin. As long as there are no problems in the system, the schedule will be followed. There are numerous algorithms that produce schedules under various combinations of task constraints and processor configurations [Liu, 1991; Ecker, 1991].

Dynamic-priority tasks and dynamic task-set scheduling is more difficult because schedules have to

be recomputed when the set of tasks changes or when any task's priority changes. Many algorithms specifically tie a task's priority to its execution time, with the shortest length task having the highest priority [Liu, 2000, Ch. 6]. The scheduler can not determine *a priori* whether a set of tasks is schedulable unless it knows enough about the task set to enumerate every possible combination of task and priority. The scheduling problem is somewhat easier when tasks are *preemptable*, meaning that they can be temporarily interrupted in order to let another task meet a deadline.

Some dynamic-priority systems use a method called *tick scheduling*, where the scheduler runs every *n* time units, rather than whenever the task list or task priorities change [Liu, 2000, Ch. 5]. At each scheduling interval, the scheduler produces a new schedule; optionally, it may first perform a schedulability analysis on the active tasks. Tick scheduling of dynamic-priority tasks of dynamic-priority jobs is considered to be one of the very hardest scheduling problems [Liu, 2000][5].

In real-time scheduling terms, the CPA is performing tick scheduling of dynamic-priority, non-preemptable tasks on a single processor. However, its scheduling problem is greatly simplified from the general problem in that it does not have to run all of the tasks. That is, the fairness criterion is waived. After applying the current scheduling rules, the CPA executes the tasks in prioritized order, keeping track of available time. If there are still tasks to execute when time expires, it notifies the rest of the tasks that they did not have time to run, and it then proceeds to the next scheduling cycle. In such a situation the set of tasks is unschedulable and a typical real-time scheduler would signal a fault. But the CPA is designed to perform as best it can under heavy perceptual load. Under such conditions, perfect performance is simply not possible, but abandoning the scheduling problem is not a good response.

However, this may lead to situations where perception quality is insufficient under heavy load. For example, the robot may judge an input to be low-priority, when in fact it indicates an obstacle. If the input is not processed the robot may hit the obstacle, and one might say that perception has failed. Failure-proof perceptual systems (or any system, for that matter) are difficult if not impossible to build and require techniques not in everyday use [Leveson, 1995]. AI systems must be able to detect and recover from perceptual failure. The cognitive system must take that into account that perceptual limitations will arise when the system is under perceptual load.

[5] A search at Google.com turns up only six documents containing the phrase "tick scheduling". Two are in German, but appear to be about fixed-priority systems. The others are references to the Linux kernel).

# Chapter 8: Summary and Future Work

Intelligent robots need high-speed, multimodal perception with limits on maximum perceptual latencies. This dissertation introduces CPA, a general-purpose, multimodal perception management system. CPA contains domain-independent methods for focusing attention on high-priority input while satisfying soft real-time requirements.

Characteristics of human perception and attention were incorporated into the CPA design, including: sustained attention, especially for detecting and tracking objects; selective attention, for filtering and prioritizing input; divided attention, in order to pursue more than one perceptual goal at a time; and control, so the cognitive component of the system can set and modify perceptual goals. However, the CPA implementation is very different because the human brain and today's computers process information in fundamentally different ways.

Fundamental questions addressed in the CPA design include generality vs. speed, parallel design vs. sequential implementation and the need for different types of sensor input, including input processed from the raw input. The resulting system manages sensors, detectors and trackers in order to produce a focus set containing prioritized percepts.

## 8.1 FUTURE WORK

The CPA, which is already very useful, can be extended in several ways. First, when multiprocessor systems become more common, the CPA can be modified to utilize multiple processors. The main loop allocates time to detectors and trackers in several places. The allocation sections can be modified to manage a pool of processors or, if the operating system provides sufficient support, to simply start as many detectors or trackers as there are available processors.

A more declarative version of the CPA could probably be developed. The code for most of the sensor streams is identical. A great deal of the detector code is the same for each detector. Only the trackers differ considerably, although they have some similarities, too.

I have not implemented many trackers that use multiple sensor streams. The CPA does not necessarily deliver a full set of sensor percepts to a tracker at the same time. It should not be hard to set up a generic framework that stores sensor data for a tracker until all of it has arrived. It is unclear what to do if only part of the necessary set of data arrives on one attention cycle.

There are opportunities to implement more sophisticated attention policies. The ones I have implemented utilize some of the available parameters, but more complex strategies could be developed.

The dissertation outlines a method for creating perceptual tasks at runtime from perceptual goals such as *Go to room-37*. The high-level concepts have associated perceptual information which can be used to derive a perceptual task. One question is whether perceptual tasks are generated specifically for a perceptual goal or whether they are generic tasks that can be reused with other goals. I suspect they are generic but parameterized, and some knowledge will be required to correctly parameterize them for a specific goal.

The theory of real-time operating system schedulers provides several mechanisms for describing and analyzing real-time scheduling processes. It would be useful to use these formal methods to describe and analyze scheduling as performed in the CPA. The analysis techniques might provide ways to develop alternate schedulers for different problems, or for multiprocessor machines. In addition, an implementation based on a true real-time operating system might be more generally useful.

# Appendix 1: CPA Implementation Details

This section lists all of the classes in the CPA system along with their fields and methods. The next section provides a small example that uses some of the classes. The more complex example in the next chapter utilizes more of the features of the system.

In each of the classes below, the fields have accessor methods that are not listed below. The existing implementation of the CPA is written in Common LISP using the CLOS object system.

## A1.1 CPA

As described above, the CPA class manages the attention process by keeping track of active sensors, detectors and trackers; by allocating time to perceptual processes; and by updating the focus set.

| | |
|---|---|
| Decay-rate | Constant used in the decay equation. |
| Default-detector | The default detector (if any). |
| Default-matcher | The matcher used by the default detector. |
| Detection-time | Portion of perception time allocated to detection. Default is 0.2. |
| Detectors | List of active detectors. |
| Detector-order-fn | The function used to sort detectors. |
| Focus | Contains the focus set. |
| Focus-strength | Resistance to being distracted by unanticipated inputs. Used when setting activation level of percepts matched by the default detector. |
| init-time | The (wall clock) time this run started. |
| Lock | A thread lock variable for the CPA. This prevents multiple processes from accessing the focus set at the same time. |
| Matchers | List of active matchers. |
| Name | The name of this CPA. |
| Sensors | List of active sensors. |
| Sensor-buffers | List of active sensor buffers. |
| Sensor-order-fn | The function used to sort sensors. Part of the attention policy. |
| State | :initialized, :running, :paused or :killed |
| Thread | Contains a pointer to the thread in which the CPA is running. |
| Timebound | The focus set update interval (default 100ms). |
| Trackers | List of active trackers. |
| Tracker-order-fn | The function used to sort trackers. |
| Update-counter | Counts the number of times through the main loop, in order to provide statistics on timing. |
| Update-time | Used when computing timing statistics. |

**Table 14.** Fields of the CPA class.

| | |
|---|---|
| add-to-focus | Adds a percept to the focus set. |
| add-detector | Adds an active detector. |
| add-matcher | Adds an active matcher. |
| add-sensor | Adds an active sensor. |
| add-tracker | Adds an active tracker. |
| c-avg | Computes the average certainty the active trackers have in the characteristics of the objects they are tracking. |
| cpa-load | Returns the average "load" of the CPA, which is the time it takes to perform one attention cycle. |
| cpa-main-loop | The loop that performs the attention cycle. |
| Current-time | Returns the current wall clock time relative to the start of the run. |
| Decayed-activation | Using the decay equation, it computes the current activation of a percept in the focus. |
| focus | Returns the current focus set, prioritized. |
| initialize-instance | After creating an instance of the CPA, this method adds a default detector and matcher. |
| kill | Kills all the active matchers, sensors, detectors and trackers, then terminates the CPA's thread. |
| pause | Pauses all the active matchers, sensors, detectors and trackers, then terminates the CPA's thread. |
| q-measure-real | Calculates the current quality of perception. The quality can also be estimated, hence the "-real" suffix on this method. |
| remove-detector | Removes a detector from the CPA. |
| remove-matcher | Removes a matcher from the CPA. |
| remove-sensor | Removes a sensor from the CPA. |
| remove-sensor-buffer | Removes a sensor buffer from the CPA. |
| remove-tracker | Removes a tracker from the CPA. |
| remove-all-detectors | Removes all detectors from the CPA. |
| remove-all-trackers | Removes all trackers from the CPA. |
| resume | Resumes the CPA after a pause. |
| set-sensor-priority | Sets the priority of a sensor. |
| start | Starts the CPA after it is created. |

**Table 15.** Methods of the CPA class.

## A1.2 SENSOR

Sensors store percepts in sensor buffers. A sensor inherits most of its methods from a Generator class.

| | |
|---|---|
| Cpa | A pointer to the CPA object which manages it. |
| Generator-args | Arguments passed to the generator function. |
| Generator-fn | The function which reads or generates data and produces percepts. |
| Input-stream | A place to store an input stream, if needed. |
| Name | The name of this sensor. |
| Output-stream | A place to store an output stream, if needed. |
| Priority | The priority (importance) of this sensor. |
| Thread | The thread in which this sensor is running. |

**Table 16.** Fields of the Sensor class.

| | |
|---|---|
| close-input-stream | Closes the input stream. |
| close-output-stream | Closes the output stream. |
| initialize-instance | After a generator object is created, this method creates a thread to run the generator-fn. |
| kill | Closes the input and output streams and kills the thread in which this Sensor runs. |
| pause | Pauses this sensor's thread. |
| resume | Resumes the thread after it has been paused. |

**Table 17.** Methods of the Sensor class.

## A1.3 BUFFER

Buffers are a general-purpose class associated with the Generator class. They are used unchanged as sensor buffers. Buffer inherits from an io-stream class in order to use some of its methods. The low-level data buffer that it manages is a FIFO buffer, typically about three elements long. It keeps track of number of items read from the buffer, written to the buffer, or dropped from the buffer due to buffer overruns.

| | |
|---|---|
| bound | The maximum number of elements in the buffer. |
| buffer | The actual vector that stores elements. |
| cpa | A pointer to the CPA object which manages it. |
| drop-count | Tallies the number of percepts dropped due to buffer overruns. |
| lock | The synchronization lock which controls access to the buffer. |

| | |
|---|---|
| name | The name of this buffer. |
| read-count | Tallies the number of items read from the buffer. |
| read-pointer | A pointer into the low-level data buffer. |
| sensor | The sensor to which this buffer belongs. |
| size | The current size of (number of items in) the buffer. |
| state | :OPEN, :CLOSED, :PAUSED |
| write-count | The priority (importance) of this sensor. |
| write-pointer | A pointer into the low-level data buffer. |

**Table 18.** Fields of the Buffer class.

| | |
|---|---|
| close | Closes this buffer. When closed it will not accept input. |
| flush | Clears the buffer. |
| initialize-instance | After a buffer object is created, this method initializes the data buffer and creates a semaphore used to synchronize buffer access. |
| increment-pointer | Increments or decrements the read and write pointers of the circular buffer. |
| pause | Sets the state to :PAUSED. |
| resume | Sets the state to :OPEN. |
| retrieve | Returns the next item in the data buffer. |
| store | Stores an item in the data buffer. |

**Table 19.** Methods of the Buffer class.

### A1.4 MATCHER

A matcher contains a function used by detectors or trackers.

| | |
|---|---|
| cpa | A pointer to the CPA object which manages it. |
| match-fn | The function used in percept matching. |
| name | The name of this matcher. |

**Table 20.** Fields of the Matcher class.

| | |
|---|---|
| match-data | Receives a percept, a detector and the maximum amount of time it is allowed to run. It returns one or more percepts. |

**Table 21.** Methods of the Matcher class.

### A1.5 DETECTOR

A detector uses a match function to match raw percepts. If it finds a match, it either forwards the percept to the focus set or creates a processed percept to encapsulate the raw percept and forwards it to the focus set.

| | |
|---|---|
| cpa | A pointer to the CPA object which manages it. |
| masks | A list of masks that trackers have placed on this detector. The contents are task-dependent. |
| mask-fn | A function which will create a mask. |
| matcher | The matcher used with this detector. |
| name | The name of this detector |
| priority | The priority (importance) of this detector. |
| sensor | The sensor(s) which this detector monitors. |
| sensor-fn | If a tracker is generated, this function is called to get the sensor list for the tracker. The tracker's sensor(s) need not be the same as the detector's. |
| tracker-type | The class to instantiate to create a tracker from this detector. |
| tracking-fn | The equivalent of the match-fn for a tracker generated from this detector. |
| user-info | The user can store anything here. |

**Table 22.** Fields of the Detector class.

| | |
|---|---|
| add-mask | A tracker calls this method to add a mask to the detector. |
| clear-masks | Removes all the masks from the detector. |
| create-tracker | Creates a tracker from the detector. |
| match | Calls the match-data method of this detector's matcher. |
| remove-mask | Removes all masks associated with a given tracker. |

**Table 23.** Methods of the Detector class.

### A1.6 TRACKER

A tracker is a subclass of Detector. It works much like a detector except that it can add masks to the Detector that spawned it. The Detector fields and methods are not listed here, although they do apply to trackers. See the previous section for a list of those elements. The tracker has no methods of its own.

| | |
|---|---|
| certainty | The tracker's certainty in the object it is tracking. For example, it may be a measure of certainty of the object's position. |
| detector | The detector that spawned this tracker. |
| update-id | A tag supplied by the CPA that is unique for every attention cycle. The CPA uses this to determine whether the tracker was called during the current attention cycle. |

**Table 24.** Fields of the Tracker class.

## A1.7 PERCEPT

A percept represents either a raw sensor data value or else something processed from the raw data. For example, the raw percept might be an array of numbers representing a frame of camera input, but the processed percept might be a description of an animal detected in the video frame.

| | |
|---|---|
| `claimed` | Signals whether a detector or tracker accepted the percept during this attention cycle. |
| `counter` | Counts the instantiated. The number is used to generate a unique identifier for each percept. |
| `cpa` | A pointer to the CPA object which manages it. |
| `current-activation` | The current activation of this percept. |
| `initial-activation` | The initial activation of this percept. |
| `ID` | A unique tag generated by the attention cycle. |
| `match-info` | Stores the detector and its priority. |
| `sensor` | The sensor this percept came from. |
| `sensor-priority` | The priority of that sensor when the percept was created. |
| `tag` | A user-defined tag that usually identifies the type of a raw or processed percept. |
| `timestamp` | The time the percept was generated. |
| `tracked` | Whether this percept is involved with a tracker. |
| `value` | The raw data item from the sensor. |

**Table 25.** Fields of the Percept class.

| | |
|---|---|
| `Activation` | Returns the current activation of the percept, after applying the decay equation. |
| `Instantiate` | Sets the timestamp, ID and tracked fields of the percept to indicate that a detector has accepted it. |
| `instantiated-p` | Returns T if the ID field is not null. |
| `Track` | In addition to what (`instantiate`) does, this method also creates a new tracker (by calling `create-tracker` on the detector) and adds the new tracker to the CPA. |

**Table 26.** Methods of the Percept class

# Appendix 2: Object tracking

In order to track stationary or moving objects while the robot is moving, it is helpful to predict the future position of an object with respect to the robot. The robot can improve its perceptual efficiency by limiting search to areas around the predicted position. This appendix derives equations that can be used to predict the position of stationary or moving objects in the environment in which the robot is traveling.

Difficulties with object tracking are not addressed here, but some of the common ones are: inability to recognize an object due to missing sensor data or limitations in the object recognition algorithm; occlusion; confusion with similar nearby objects; and incorrect knowledge of the robot's motion vector or the object's motion vector.

## A2.1 COORDINATE SYSTEM AND UNITS

The robot's egocentric coordinate system, shown in Figure A1, is the only coordinate system used here. It is a standard Cartesian coordinate system that is rotated 90 degrees from the usual perspective, so that the *x*-axis points directly ahead of the robot. The origin of the coordinate system is at the center of the robot.

In this coordinate system, angles increase in the standard counter-clockwise direction. Coordinates with $x > 0$ are ahead of the robot's position and coordinates with $y > 0$ are to the left of the robot.

Any units may be used for distance measurements in the equations below, but angular measurements must be in radians.

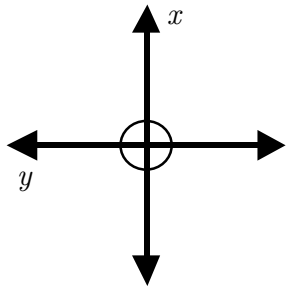


**Figure A1.** Robot egocentric coordinate system.

## A2.2 PROBLEM DESCRIPTION

Figure A2 below illustrates the problem. At time $t_1$ the robot is at position $R_1$, which is the origin of its coordinate system. The robot is moving with linear velocity $v$ and angular velocity $\omega$ so that at time $t_2$ it is at position $R_2$. Meanwhile, in the same time interval, an object moves from position $O_1$ to position $O_2$. Given the starting positions of the robot and the object, the linear and angular velocities of the robot, and the linear velocity of the object[6], what is the position of the object relative to the robot at time $t_2$?

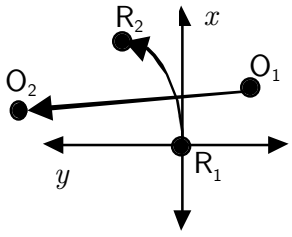


**Figure A2.** Illustration of the object prediction problem.

## A2.3 PREDICTING THE ROBOT'S POSITION

If $\omega = 0$, the robot's new position after $\Delta t = t_2 - t_1$ is:

[6] The equations would be more accurate if they included the angular velocity of the object, but the angular velocity is difficult to determine and the linear velocity is sufficiently accurate for short time intervals.

$$R_2 = \vec{v} \quad \Delta$$

If $\omega \neq 0$, the robot's path for constant $v$ and $\omega$ is along the perimeter of a circle. The $x$-axis of the robot's coordinate system always lies tangent to the circle. The distance along the arc of the robot's path from $R_1$ to $R_2$ must be $v*\Delta t$. The angle of the robot at $R_2$, in the $R_1$ coordinate system is $\omega*\Delta t$. Figure A3 below shows the positions and angles in the system.

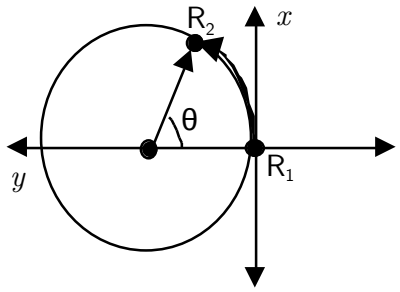


**Figure A3.** Geometry of the robot's motion.

Since the $x$-axis of the robot's coordinate system at $R_2$ is tangent to the circle, we know that the angle $\theta$ is equal to $\omega*\Delta t$. Since the length of the arc along the perimeter of the circle is $r\theta$, we conclude that

$$r\theta = v \quad \Delta, \quad \text{so } r = \frac{v \quad \Delta}{\omega \quad \Delta} = \frac{v}{\omega}.$$

Now we can calculate the new position of the robot using trigonometry based on the robot's circular path:

$$R_2.x = r\sin\theta = \frac{v}{\omega}\sin(\omega \quad \Delta)$$

$$R_2.y = r - r\cos\theta = \frac{v}{\omega}(1-\cos(\omega \quad \Delta))$$

### A2.4 PREDICTING THE OBJECT'S POSITION

The object is assumed to have linear motion, so computing its position $O_{2,R1}$ at time $t_2$ in the robot's original coordinate system is easy:

$$\overrightarrow{O_{2,R1}} = O_1 + \vec{v}_o \quad \Delta$$

This position needs to be translated and rotated into the robot's new coordinate system. The translation is and the rotation is $-\omega*\Delta t$. Performing the translation first and then rotating, we arrive at the final equation:

$$O_2 = (\overrightarrow{O_{2,R1}} - \overrightarrow{R_1R_2}) \cdot \begin{matrix} & -\overrightarrow{R_1R_2} \\ \text{é}\cos(\omega \quad \Delta) & -\sin(\omega \quad \Delta)\text{ù} \\ \text{ê}\sin(\omega \quad \Delta) & \cos(\omega \quad \Delta)\text{ú} \end{matrix}$$

### A2.5 Example: a pedestrian approaching the robot

The situation is illustrated in Figure A4 and described by the following parameters: $O_1$ = *(6.7, -1.2),*

$\boldsymbol{v_o}$ = *[-0.4 0.15]*, $v = 1.6$, $\omega = 0.2$. Relative to the robot's new position, where is the tracked object after 1 second ($\Delta t = 1.0$)?

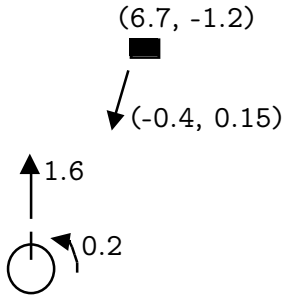


**Figure A4.** A pedestrian approaching the robot.

$$\omega \Delta t = 0.2$$

$$\frac{v}{w} = \frac{1.6}{0.2} = 8.0$$

$$R_2 = (8.0\sin(0.2),\ 8.0(1-\cos(0.2)))$$

$$= (1.589,\ 0.159)$$

$$\overrightarrow{O_{2,R1}} = (6.7,\ -1.2) + (-0.4,\ 0.15)$$

$$= (6.3,\ -1.05)$$

$$O_2 = (\overrightarrow{O_{2,R1}} - \overrightarrow{R_1 R_2}) \cdot \begin{bmatrix} \cos(\omega \Delta t) & -\sin(\omega \Delta t) \\ \sin(\omega \Delta t) & \cos(\omega \Delta t) \end{bmatrix}$$

$$= [6.3 - 1.589 \quad -1.05 - 0.159] \cdot \begin{bmatrix} \cos(0.2) & -\sin(0.2) \\ \sin(0.2) & \cos(0.2) \end{bmatrix}$$

$$= [4.711 \quad -1.209] \cdot \begin{bmatrix} \cos(0.2) & -\sin(0.2) \\ \sin(0.2) & \cos(0.2) \end{bmatrix}$$

$$= [4.377 \quad -2.121]$$

After 1 second, the pedestrian will be 4.377 meters ahead of the robot and 2.121 meters to the right of the robot.

## Bibliography


Acker, L. and B. Porter (1994). Extracting Viewpoints from Knowledge Bases, in: *Proceedings of the National Conference on Artificial Intelligence* (AAAI-94).

Ahmad, S. (1991). *VISIT: An Efficient Computational Model of Human Visual Attention*. Ph.D. Thesis, University of Illinois at Urbana-Champaign. Also available as *Technical Report TR-91-049*, International Computer Science Institute, Berkeley, CA, 1991.

Anderson, J. R., M. Matessa and S. Douglass (1995). The ACT-R theory and visual attention, in: *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pp. 61-65. Hillsdale, NJ: Lawrence Erlbaum Associates.

Baluja, S. and D. A. Pomerleau (1997). Expectation-Based Selective Attention for Visual Monitoring and Control of a Robot Vehicle, *Robotics and Autonomous Systems*, **22**(3-4):329-344.

Bethel, R. E. and G. J. Paras (1998). A PDF Multisensor Multitarget Tracker, *IEEE Transactions on Aerospace and Electronic Systems*. **34**(1):153-168.

Blair, W. D. and Y. Bar-Shalom (1996). Tracking Maneuvering Targets With Multiple Sensors: Does More Data Always Mean Better Estimates?, *IEEE Transactions on Aerospace and Electronic Systems*. **32**(1):450-456.

Boddy, M. and T. L. Dean (1989). Solving time-dependent planning problems, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (IJCAI-89), pp. 979-984, Detroit, Michigan.

Breazeal, C. and B. Scassellati (1999). A Context-dependent attention system for a social robot, in: *Proceedings of IJCAI-99*, AAAI Press.

Bridges, P. (2001). *Real-Time Operating Systems*, WWW URL: http://www.cs.arizona.edu/people/bridges/os/realtime.html.

Broadbent, D. E. (1958). *Perception and communication*. Pergamon Press, Oxford.

Buttazzo, G. C. (1997). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Boston.

Chandana, C. (2001). *Embedded and Real-Time Systems*, WWW URL: http://www.ifi.unizh.ch/groups/ailab/links/embedded.html

Coen, M. H. (1997). Building Brains for Rooms: Designing Distributed Software Agents. In: *Proceedings of the AAAI-97*, Providence, Rhode Island.

Cohen, P. R., M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith and J. Clow (1997). QuickSet: Multimodal interaction for distributed applications, in: *Proceedings of the Fifth International Multimedia Conference (Multimedia '97)*, ACM Press, pp. 31-40.

Crick, F. and C. Koch (1990). Some Reflections on Visual Awareness, in: *Proceedings of the Cold Spring Harbor Symposia on Quantitative Biology*, Volume LV. Cold Spring Harbor Laboratory Press.

Decker, K., A. Garvey, M. Humphrey and V. Lesser (1992). *Real-Time Control of Approximate Processing*, Technical Report UM-CS-1991-050, Computer and Information Science Department, University of Massachusetts.

Desimone, R. and J. Duncan (1995). Neural mechanisms of selective visual attention. *Annual Review of Neuroscience* **18**:193-222.

Deutsch, J. and D. Deutsch (1963). Attention: Some theoretical considerations. *Psychological Review*, **70**:80-90.

Ecker, K. (1991). Algorithmic Methods for Real-Time Scheduling, in *Real Time Computing*, W. A. Halang and A. D. Stoyenko, eds., pp. 9-29. Springer-Verlag, Prentice-Hall, New Jersey.

Flat (200). [WWW document] URL: http://www.cs.utexas.edu/users/qr/robotics/flat/

Garvey, A. and V. Lesser (1993). *A Survey of Research in Deliberative Real-Time Artificial Intelligence*, Technical Report UM-CS-1993-084, Computer and Information Science Department, University of Massachusetts.

Goodale, M. A. and G. K. Humphrey (1999). The Objects of Action and Perception, in *Object Recognition in Man, Monkey and Machine*, M. J. Tarr and H. H. Bulthoff, eds. MIT Press, Cambridge, Massachusetts.

Guil, N., J. Villalba and E. L. Zapata (1995). A Fast Hough Transform for Segment Detection, *IEEE Transactions on Image Processing* **4**(11):1541-1548.

Hatfield, G. (1998). Attention in Early Scientific Psychology, in: *Visual Attention*, R. D. Wright, ed. Oxford University Press, New York.

Hayes-Roth, B. (1990). Architectural Foundations for Real-Time Performance in Intelligent Agents, *Real-Time Systems* **2**:99-125.

Hayes-Roth, B. (1993). Opportunistic Control of Action in Intelligent Agents, *IEEE Transactions on Systems, Man and Cybernetics* **23**(6):1575-1587.

Hewett, M. and B. Hayes-Roth (1989). Real-Time I/O in Knowledge-Based Systems, in *Blackboard Architectures and Applications*, Jagannathan, V., R. Dodhiawala and L. S. Baum (eds.), pp. 269-284, Academic Press, Boston.

Hill, R. W. Jr., J. Chen, J. Gratch, P. Rosenbloom and M. Tambe (1997). Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft. In: *Proceedings of the AAAI-97*, Providence, Rhode Island.

Intille, S. S., J. W. Davis and A. F. Bobick (1997). Real-Time Closed-World Tracking, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Also available as MIT Media Laboratory Perceptual Computing Section Technical Report No. 403.

James, W. (1890). *The Principles of Psychology*. See also [WWW document] URL: http://www.yorku.ca/dept/psych/classics/James/Principles/index.htm

Kurien, T. (1990). Issues in the Design of Practical Multitarget Tracking Algorithms, in *Multitarget-Multisensor Tracking: Applications*, Y. Bar-Shalom, ed., pp. 43-83, Artech House, Norwood, MA.

Kortenkamp, D., R. P. Bonasso and R. Murphy (1998). *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, AAAI Press, Menlo Park, California.

LaBerge, D. (1995). *Attentional Processing: The Brain's Art of Mindfulness*, Harvard University Press, Cambridge, Massachusetts.

Lark, J. S., L. D. Erman, S. Forrest, K. P. Gostelow, F. Hayes-Roth and D. M. Smith (1990). Concepts, Methods, and Languages for Building Timely Intelligent Systems, *Real-Time Systems* **2**:127-148.

Larsson, J. E., B. Hayes-Roth and D. Gaba (1996). *Guardian: Final evaluation*. Technical Report KSL-96-25, Knowledge Systems Laboratory, Stanford University, August 1996.

Lesser, V. R., S. H. Nawab and F. I Klassner (1995). IPUS: An architecture for the Integrated Processing and Understanding of Signals, *Artificial Intelligence* **77**(1), August/September 1995.

Leveson, N. G. (1995). *Safeware: System Safety and Computers*, Addison-Wesley.

Liu, C. L. (1991). Fundamentals of Real-Time Scheduling, in *Real Time Computing*, W. A. Halang and A. D. Stoyenko, eds., pp. 1-7. Springer-Verlag, Prentice-Hall, New Jersey.

Liu, J. W. S. (2000). *Real-time Systems*, Prentice-Hall, New Jersey.

Luck, S. J. and M. Girelli (1998). Electrophysiological Approaches to the Study of Selective Attention in the Human Brain, in: *The Attentive Brain*, R. Parasuraman, Ed., MIT Press, Cambridge, Massachusetts.

LynuxWorks (2000). *LynuxWorks Patented Technology Speeds Handling of Hardware Events*, http://www.lynx.com/products/whitepaper_patent.html

Mazor, E., A. Averbuch, Y. Bar-Shalom and J. Dayan (1998). Interacting Multiple Model Methods in Target Tracking, *IEEE Transactions on Aerospace and Electronic Systems* **34**(1):103-123.

Microware (2000). *OS-9®: Real-Time Operating System*, WWW URL: http://www.microware.com/Products/Rsrcs/WhitePapers/OS9_wp.pdf

R. Milanese, J.-M. Bost and T. Pun (1992). A Bottom--up Attention System for Active Vision, in: *Proceedings of the 10th European Conference on Artificial Intelligence*, Vienna, August 3--7, John Wiley and Sons, 1992, pp. 808-810.

Niebur, E. and C. Koch (1998). Computational Architectures for Attention, in: *The Attentive Brain*, R. Parasuraman, ed. MIT Press, Cambridge, Massachusetts.

NTIS (1973). Eastern Air Lines, Inc., L-1011, N310EA, Miami, Florida, December 29, 1972. NTIS Report Number 19B-222359/2 and NTSB Report Number NTSB-AAR-73-14. See also [WWW document] URL: http://www.d-n-a.net/users/dnetGOjg/291272.txt

Olshausen, B. A., C. H. Anderson and D. C. Van Essen (1993). A Neurobiological Model of Visual Attention and Invariant Pattern Recognition Based on Dynamic Routing of Information, *Journal of Neuroscience* **13**(11):4700-4719.

Parasuraman, R., ed. (1998a). *The Attentive Brain*, MIT Press, Cambridge, Massachusetts.

Parasuraman, R., ed. (1998b). The Attentive Brain: Issues and Prospects, in: *The Attentive Brain*, MIT Press, Cambridge, Massachusetts.

Paul, C. (2001). *OSE™ Realtime Kernel*, WWW URL: http://www.enea.com/PDF/rtk.pdf

Peterson, L. R. and M. J. Peterson (1959). Short-term retention of individual verbal items, *Journal of Experimental Psychology* **58**:193-198.

Pomerleau, D. A. (1993). *Neural Network Perception for Mobile Robot Guidance*, Kluwer Academic Publishing.

Posner, M. I. and G. J. DiGirolamo (1998). Executive Attention: Conflict, Target Detection, and Cognitive Control, in: *The Attentive Brain*, R. Parasuraman, Ed., MIT Press, Cambridge, Massachusetts.

Postma, E. O., H. J. van den Herik and P. T. W. Hudson (1997). SCAN: A Scalable Model of Attentional Selection, *Neural Networks* **10**(6):993-1015.

Ramachandra, K. V. (1990). *Kalman Filtering Techniques for Radar Tracking*, Marcel Dekker, Inc., New York.

Reid, D. B. (1979). An Algorithm for Tracking Multiple Targets, *IEEE Transactions on Automatic Control* **AC-24**(6):843-854.

Rodier, P. M. (2000). The Early Origins of Autism, *Scientific American* **282**(2):56-63.

Saha, R. K. and K. C. Chang (1998). An Efficient Algorithm for Multisensor Track Fusion, *IEEE Transactions on Aerospace and Electronic Systems* **34**(1):200-210.

Sedgewick, R. (1998). *Algorithms in C Parts 1-4: Fundamentals, Sorting, Searching, and Strings, 3rd edition*, Addison-Wesley,

Srinivasan, B. (1998). *A Firm Real-Time System Implementation using Commercial Off-The-Shelf Hardware and Free Software*, Masters Thesis, Department of Electrical Engineering and Computer Sciences, University of Kansas.

Stone, L. D., C. A. Barlow and T. L. Corwin. (1999). *Bayesian Multiple Target Tracking*, Artech House, Boston.

Treisman, A. (1998). Perception of Features and Objects, in: *Visual Attention*, Oxford University Press, New York.

Treisman, A. and S. Gormican (1988). Feature analysis in early vision: evidence from search asymmetries, *Psychology Review* **95**:15-48.

Van der Heijden, A. H. C. (1992). *Selective Attention in Vision*, Routledge, London.

Washington, R., L. Boureau and B. Hayes-Roth (1990). *Using Knowledge for Real-Time Input Data Management*, Technical Report KSL-90-14, Knowledge Systems Laboratory, Stanford University.

Xu, H. and H. Van Brussel (1997). A Behaviour-based Blackboard Architecture for Reactive and Efficient Task Execution of an Autonomous Robot, *Robotics and Autonomous Systems* **22**:115-132.

Xu, H. and J. Vandorpe (1994). Perception Planning in Mobile Robot Navigation, in: *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI '94)*, Las Vegas, NV.

Yodaiken, V. (2001). The RTLinux Manifesto, WWW URL: http://www.rtlinux.org/documents/papers/rtmanifesto.pdf.

Zhang, Y., H. Leung, M. Blanchette, T. Lo and J. Litva (1997). An Efficient Decentralized Multiradar Multitarget Tracker for Air Surveillance, *IEEE Transactions on Aerospace and Electronic Systems* **33**(4):1357-1363.