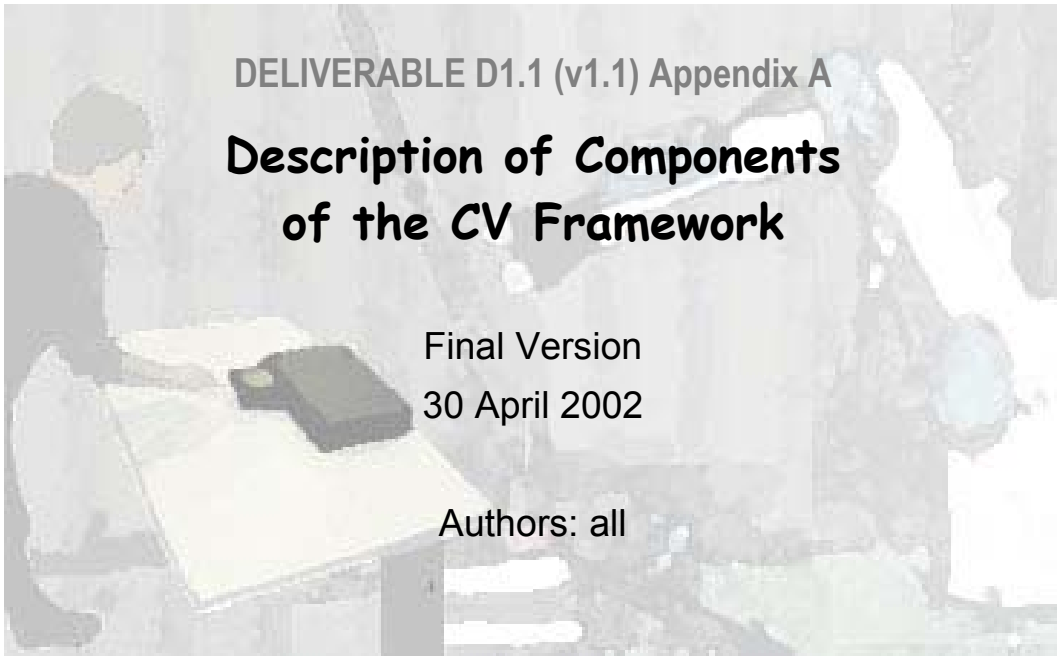


DELIVERABLE D1.1 (v1.1) Appendix A

Description of Components of the CV Framework

Final Version

30 April 2002

Authors: all

Project acronym: **ACTIPRET**

Project full title: **Interpreting and Understanding Activities of Expert Operators for Teaching and Education**

Action Line IV.2.1: **Real Time Distributed Systems (Cognitive Vision)**

Contract Number: **IST-2001-32184**

Contents

Appendix A: Description of components

This Appendix provides a fuller description of all components envisaged within the ActIPret Demonstrator (AD) than that provided in the main D1.1 document (see Figures 4 and 5 in the main document for an overview of system topology).

Techniques, methods and the relation to the state of the art of some of the components are described in separate deliverables:

- object relations in D3.1,
- object recognition in D4.1,
- attentive behaviours in D6.1

The following are provided for each component:

- detailed description of functionality
- relation to Cognitive Vision (CV): learning, memory, control, reasoning etc.
- what will be done (minimum) to fulfil ActIPret and what would be desirable
- relation to other components, other services requested

A.1 Service list (v1 ACIN)

A.1.1 Description

The Service List is a database where components register their abilities and the properties of these abilities. These abilities are called ***services*** and form an abstract interface description

of the components. All components and their services depend on the resources available.

Resources

Resources are restricted. There is always a limited amount of CPU power, only a limited number of cameras and the free working space of the robots used to get good viewpoints is limited. Therefore the access to resources must be managed to solve conflicting resource needs.

In order to optimise the resource management, it is necessary to manage the components, which request resources. If resources are not managed a component would be activated, which has no or very bad probability to get the requested resources. To start a component on a PC with nearly zero available CPU time makes no sense. Hence every component (in fact its abstract representation, the service) has to be managed by the system, because of the restricted and conflicting resources it requires.

Abstract service description

The following three values should be used for an abstract service description:

- **Quality of Service (QoS):** describes the performance (confidence, accuracy, speed, etc.) a service can actually deliver.
- **Costs:** are the abstract descriptions of the effort (resource requirement) of the establishment of a service. This is used to quantise the narrowness of a resource and the conflicts (incompatibility) of resource requests. E.g., the effort to transfer a running component from one PC to another PC (e.g., to get more CPU time on this PC) is embodied by migration costs.

- **Priority:** is set by the resource-requesting component to give a hint on how to solve conflicts of resource requests.

A.1.2 Principal output services/data

The Service List entries can be updated from the service-providing component.

The Service List delivers all components providing the requested services that fit the requested properties.

A.1.3 Start condition (must be initialised with)

Starts as the *first* component because all other components register in the Service List.

A.1.4 Single-shot or continuous running behaviour

It runs in 'Single-shot', because it only responds direct to service requests or service entry updates.

A.1.5 Requirements during run-time

A.1.5.1 Computation

The required CPU load is low as well as the storage capacity. Just a fast connectivity to the other components is recommended to increase the response time of requests.

A.1.5.2 Quality of service

There is no reasonable QoS value for the Service List itself because it delivers a fixed functionality.

A.1.6 Stop condition

- The Service List is the *first* component that starts and the *last* that stops.
- A restart of the Service List (in case of fall outs) requires a special methodology that is not intended to be realised during the project.

A.1.7 Stopping action

A stopped Service List is turned off.

A.1.8 Requirements that are likely to be unfulfilled

No such requirements.

A.1.9 Example scenarios

No scenarios necessary.

A.1.10 General comments

We propose using the CORBA trading service or a solution as close as possible to guarantee compatibility with other projects.

A.2 User HMI (v1 by Jon Howell @ COGS)

A.2.1 Description

This module has overall control of the ActIPret system, and switches it between the major phases of operation:

- **learning phase:** individual modules are trained on specific databases of generic information.
- **expert phase:** the entire system is run in a 'record-only' mode analysing the activities of an expert - demonstrating some specific activity and providing explicit start and end signals. The activity plans extracted from each demonstration of the scenario are combined into a composite representation of the scenario.
- **tutor phase:** the system either 'explains' how a scenario is carried out to a learner via some VR representation or 'watches' the learner attempt to carry out the scenario, either giving a binary 'correct/false' interpretation, or attempting to give guidance to the user when errors or confusion occur.

A.2.2 Principal output services of the functionality

The output for this module is the current status of the ActIPret system:

- learning phase: success of the training of individual modules
- expert phase: extracting example activity plans, maintaining/optimising a database of example activity plans
- tutor phase: success/failure of user to complete specific scenario, or interactive guidance as user undertakes individual steps. This could be accompanied or preceded by a VR demonstration of the task.

A.2.3 Start condition

The start condition for this module would be a command to start the ActIPret system.

A.2.4 Single-shot or continuous running behaviour

This function would run continuously (once started), until some command to close the ActIPret system down was encountered.

A.2.5 Requirements during run-time

Not applicable.

A.2.6 Stop condition

The stop condition for this module would be a command to stop the ActIPret system.

A.2.7 Stopping action

Not applicable.

A.2.8 Requirements that are likely to be unfulfilled

Not applicable.

A.2.9 Example scenarios

Not applicable.

A.3 Activity planner (v2.2 by Kingsley Sage @ COGS)

A.3.1 Description

General

In expert mode, the activity planner is concerned with the synthesis of an activity plan in a static database for the scenario as demonstrated by the expert. The scenario is demonstrated to ActIPret as exemplars. These exemplars may have end-to-end significance or the expert may choose to structure them in a hierarchical fashion. Where the expert structures the exemplars in a hierarchical fashion, the result is a hierarchical activity plan with embedded learned sequence models. Exemplars of the plan are represented in the conceptual language. This conceptual language will be defined formally in another document.

The conceptual language in general consists of:

- sequences of plan concept functions (atomic plan primitives) organised as (possibly) a mixture of exemplars with end-to-end significance or organised into a hierarchical structure;
- specific behaviour models for activities, actions or events observed in the exemplars and derived from abstract models defined within the reasoning engine (see the paper on learning models for more details); and
- (possibly) a limited amount of domain specific data, outside the scope of generalised models, that might be contained in the reasoning engine necessary for the correct interpretation of the scenario. Example of such domain knowledge might include state variables to represent very specific aspects of the state of functions on a CD player.

The conceptual language can be visualised in a number of useful ways. This document uses both finite state automata and hierarchical graph structure representations to illustrate various arguments.

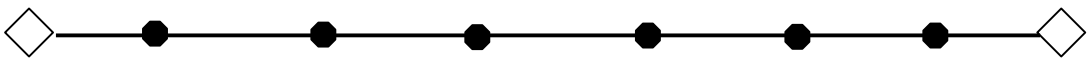
In tutor mode, VR reconstruction of the scenario using the activity plan will provide initial instruction for the trainee. The activity planner is then used to generate a new temporary workspace activity plan for stepwise comparison with the static activity plan produced in expert mode. Comparison of the static and temporary plans will be used to determine whether the trainee has made any errors. VR reconstruction will then be used to explain any discrepancies to the trainee.

Visualisation of activity plan

We can visualise a single scenario exemplar with end-to-end significance as a linear finite state automaton:

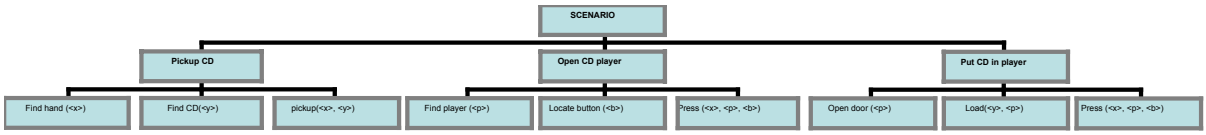
Key:

Triangles represent plan start and end points
Black circles represent plan atomic primitives



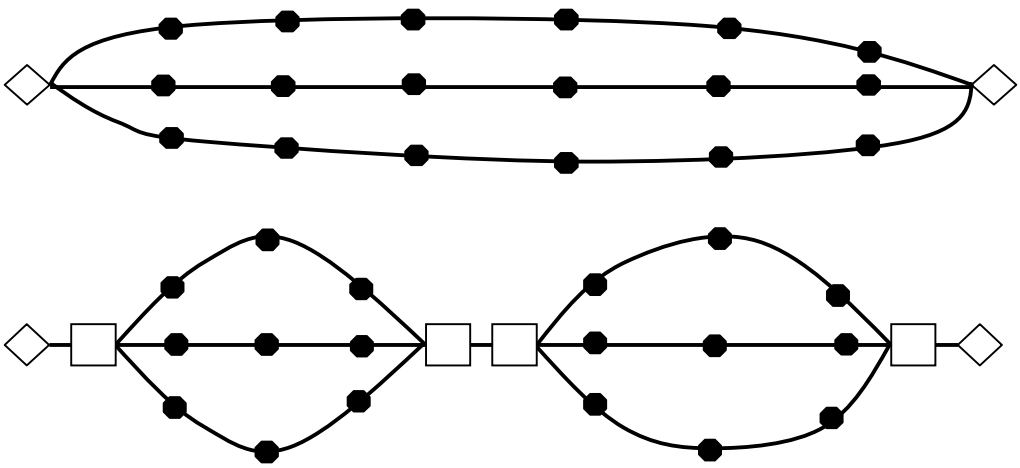
Linear planning: Activity plan consisting of 1 scenario exemplar
(1 execution path only)

We can visualise a conceptual hierarchy with intermediate plan concepts formed from ordered sequences of atomic plan primitives as a graph. Acquisition of the scenario exemplars could now occur in smaller sections as defined by the intermediate plan concepts:



The expert demonstrates each intermediate plan concept independently. This could be achieved either as part of the learning phase (the notion of a learned sequence) or through more complex flow control within the expert mode. Finally, the expert demonstrates/specifies the intermediate concept sequencing to complete the plan. Now in tutor mode it is possible to recognise an end-to-end exemplar that is composed of any combination of the sub-structure ordered sequences. The hierarchical approach can be represented graphically as finite state automata:

Key:
Triangles represent plan start and end points
Black circles represent atomic plan primitives
Square represent intermediate concept start and end points



Hierarchical planning: Activity plan consisting of 2 intermediate plan concepts each with 3 exemplars.

A.3.2 Principal output services/data

The output from the activity planner is an activity plan. A complete activity plan consists of:

- sequences of plan concept functions;
- specific behaviour models for activities, actions or events observed in the exemplars and derived from abstract models defined within the reasoning engine; and

- (possibly) a limited amount of domain specific data outside the scope of generalised models necessary for the correct interpretation of the scenario.

A very simple activity plan might be represented as follows (using C++ derived notation – the conceptual language is not yet developed fully):

```
// ---------------------------------------------------------------------
// Using C++ notation (refer also to the paper on learning models …)

linear_exemplar(1) {
       button_press(button0,cdplayer0);
       button_press(button1,cdplayer0);
       pick_up(cd0,nondef);
       put_down(cd0,cdplayer0);
       button_press1(button1,cdplayer0);
       button_press2(button2,cdplayer0);
       }

button_press(button1, cdplayer0) : public button_press(b,object) {
       // button_press(b,object) would be defined within the reasoning engine
       minimum_distance = 10.0;
       persistence_time = 5;
       }
// ---------------------------------------------------------------------
```

A.3.3 Start condition (must be initialised with)

Flow control of activity plan synthesis relies on conditions that fall outside of the semantics of the conceptual language (i.e. external events or hard coded models). The flow control simply determines the way in which the plan is built. At the most abstract level, activity plan synthesis is initialised by a service request from the User HMI component.

Flow control for the direct synthesis of hierarchical planning could be complex, as it may be difficult to ensure consistency between distinct intermediate plan concepts exemplars. We envisage that we will restrict ourselves to linear plan synthesis initially. Any extension to hierarchical planning would be likely achieved by getting the expert to manually partition linear plan sequences separately from the expert mode (plan editing).

In service call terms all flow control signals will come from the User HMI component. The range of services that the User HMI component will be able to request from the activity planner can be summarised as:

- Capture_exemplar

And to enable maintenance of the activity plan:

- Maintain_activity_plan, which might decompose further into:
  - ⇒ Display_entire_current_activity_plan
  - ⇒ Display_exemplar_from_activity_plan(exemplar_id)
  - ⇒ Delete_entire_current_activity_plan
  - ⇒ Delete_examplar_from_activity_plan(exemplar_id)

And possibly further diagnostic requests.

And possibly to support expert editing of plans and grouping of atomic plan primitives into intermediate plane concepts:

- Edit_activity_plan.

A.3.4 Single-shot or continuous running behaviour

In general, the service level semantics for activity plan generation are single shot (i.e. once the service is requested by the User HMI component, they execute once and then return control to the calling component. But the activity plan is a static database that remains in existence even if there is no current service call to the activity planner. The database remains in existence until modified or deleted by service call requests from the User HMI component.

An alternative way of looking at the service semantics is to separate the management of the activity plan database from the functions (services) that operate on it. In that case, the database manager that mediates access to the database is a continuous service and the functions that operate upon it have proper single shot semantics.

A.3.5 Requirements during run-time

In general:

- service requests from the User HMI component to initiate functions;
- ability to make service requests of the reasoning engine; and
- static storage space for the activity plan.

A.3.5.1 Inputs (functionalities/services/data)

The activity planner will need the following inputs from the activity reasoning engine:

- quantified plan concept functions that will form the atomic plan primitives; and
- details of any specific (parametric) refinements of generic models defined within the reasoning engine.

A.3.5.2 Computation

The computational load on the activity planner will be likely light in comparison to the vision modules.

A.3.5.3 Quality of service

The activity planner only communicates with the User HMI component and the reasoning engine. As there is no option for selection of alternative components to provide services and data, there is no useful meaning parametric quantification of quality of service. It is reasonable to suggest that any breakdown of the service request semantics from the User HMI component will result in a nonsense activity plan. Any failings in the quality of data provided by the reasoning engine will manifest itself as a plan exemplar that is neither repeatable nor useful later (e.g. in the tutor mode). It is intended that diagnostic tools will be built into the activity planner so that the user/expert can review the plan/exemplars that have been captured and remove any that do not make sense.

A.3.6 Stop condition

The plan database is static and remains in existence whilst the ActIPret demonstrator is active. The plan database will be used by other component/modules (e.g. for the tutor mode).

A.3.7 Stopping **action**

The activity planner does not pre-determine in any way what the scenario exemplars consist of (any view to the contrary would undermine the generality of the approach) and so has no

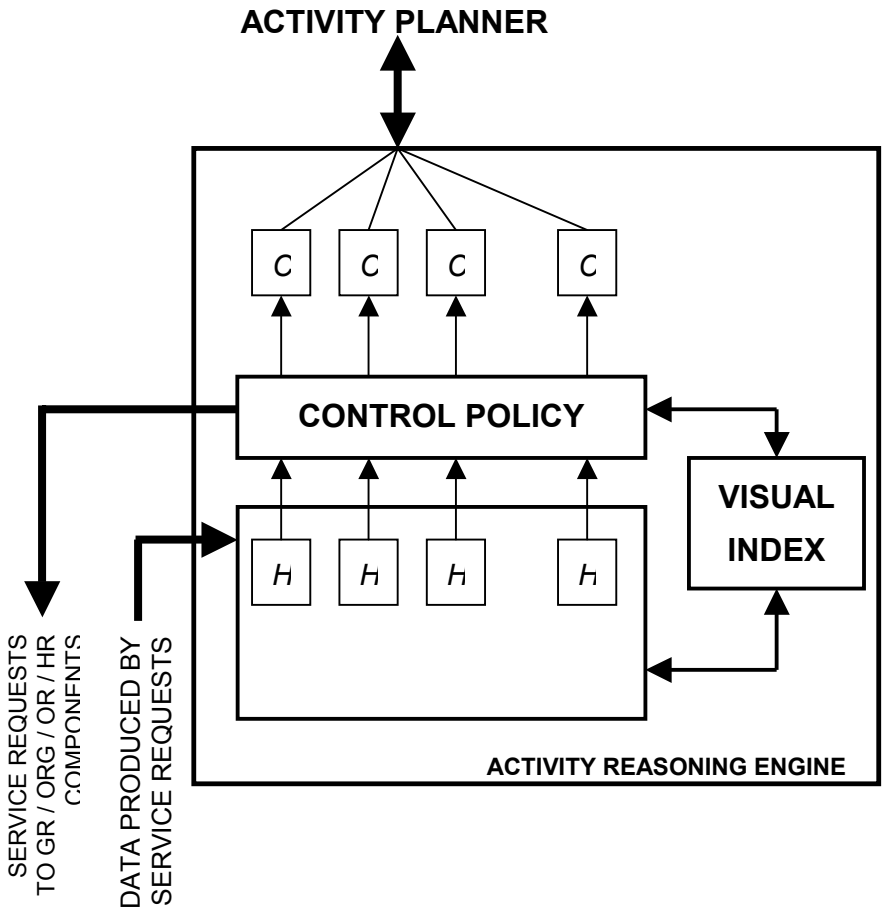
way of knowing when an exemplar is complete, or a intermediate concept is complete. This interferes with the pure single shot semantics of service requests to synthesise plans because the activity planner cannot define a stop condition. This stop condition needs to come from the User HMI component.

We might further include a time-out function or a specific expert activated stop function so that we can abort acquisition of a particular sequence if necessary (for example, the expert has made an error or decided that the exemplar is a poor case).

A.4 Activity reasoning engine (v2.1 by Kingsley Sage @ COGS)

A.4.1 Description

The reasoning engine can be represented at a high level in the figure below:



The reasoning engine contains four separate elements:

Control policy

The control policy determines at the most abstract level the task control strategy for the whole ActIPret system. When ActIPret is first started, this policy will be generic e.g. "find possible hand objects". This policy will evolve as task relevant vision data is produced (such as requests for visual verification tasks issued to the attentive visual processes). For example, if a hypothesis has emerged that a candidate hand object is moving with a consistent trajectory, then the control policy might determine that the object relation generator should locate candidate objects along that trajectory that the hand might pick up.

The control policy embodies two different types of rules:

- those that inform the selection of plan hypotheses in a manner necessary to initiate useful task relevant processing throughout the system. These rules increase the chances of creating the desired task based plan hypotheses; and
- those that consider whether it is worth trying to improve the probabilistic yields of ***H*** through the application of attentive visual processes in order to properly instantiate the relevant plan concept function. Application strategies for these rules include 'by relevancy' and 'by belief value with respect to cost of service'.

It may, or may not, be possible to make sense of these rules. If the policy is unable to inform the selection of task relevant objects, then the penalty is that we must continue to attend to all possible such objects with a correspondingly heavier computational load on the pre-attentive visual phase. If we have already been able to obtain relatively high probability yields from ***H*** (i.e. we are confident that the coarse classifications are correct and do not require further information to fully instantiate our plan concept functions) then we do not need to apply further attentive visual processing. If we have only been able to obtain relatively low probability yields from ***H***, then we can engage further attentive visual processing to get better data about any of the objects within the scope of the proposed plan hypothesis function.

For example, we might have a plan hypothesis h_k = *grasped-by(<object1>, <object2>)* where the pre-attentive phase has assigned a high probability that <*object1*> has a coarse classification of "hand" but no coarse classification at all for <*object2*> as the viewing geometry during the pre-attentive phase was inappropriate. In that case, in order to instantiate the relevant plan concept function we need to obtain a classification for <*object2*>. The attentive phase can exercise purposive control of its camera to achieve this.

The control policy determines thresholds for the probability yields obtained from instances of the member functions of ***H***. When these thresholds are exceeded, an instance of the relevant member functions of ***C*** is instantiated for input to the activity plan generator. These thresholds are described more fully in deliverable 1.1 under 'belief values'.

Visual index

The visual index is a database of task relevant objects. The definition of task relevance will depend on the context set by the control policy.

Hypothesis functions ***H***

The set of plan hypothesis functions ***H*** provide posterior probability estimates for a set of plan hypotheses. For example, using the previous example, the probability h_i is given by h_i = *grasped-by(<object1>, <object2>*). There are two distinct types of plan hypothesis function: those that map object tuples to deictic relationships (such as the example above) that may lead to instantiated plan concept functions and those that map individual objects that are required for informing the control policy. The latter category might include, for example, h_j = *found-hand(<object1>).*

The sub-set of functions ***H*** that map tuples actually serve two functions. First they provide a means to select the set of plan concept functions ***C*** that are relevant at any stage of the processing (i.e. those whose posterior probabilities exceed some pre-determined belief threshold) and are thus indicative of a particular concept. Secondly they serve at potential triggers for the attentive visual processes (subject to belief revision rules embodied within the control policy).

Hypothesis manager

The hypothesis manager has 5 functions:

- causing plan hypothesis functions to update when evidence on which they are based is updated as a result of service requests made to lower system components (the revision of belief values);
- it is the functional home for abstract behavioural models. Specific behavioural models are derived from these abstract models and form one class of plan hypothesis function. The derivation of the specific models is a function of the learning phase;
- deleting plan hypothesis functions where their belief values value fall beneath a viability threshold;
- generating parameterised service requests to the object relation generator (as dictated by the requirements of the control policy); and
- creating new plan hypothesis functions in response to data arising from service requests to the object relation generator, as well as registering such new objects in the visual index.

The hypothesis manager may include a temporal erosion of belief values.

Concept functions ***C***

The set of plan Concept functions ***C*** represent instantiated instances of activities, actions or events.

A.4.2 Principal output services/data

The output from the reasoning engine is a real time stream of plan concept functions that will make us the atomic plan primitives for the activity planner. A complete activity plan may also require specific behaviour models for activities, actions or events observed in the scenario exemplars and derived from abstract models embedded within the reasoning engine. A mechanism will be required for this parametric data to be sent to the activity planner.

A.4.3 Start-condition (must be initialised with)

In service call terms, all requests for services from the reasoning engine will come from the activity planner. The range of services that can be requested can be summarised as:

- *Produce_plan_concept_functions*
- *Set_null_hypotheses* (also deletes all entries in the visual index)
- *Specify_behavioural_model_parameters(model_name)*

And possibly further diagnostic requests.

A.4.4 Single-shot or continuous running behaviour

Continuous.

A.4.5 Requirements during run-time

In general:

- service requests from the activity planner to initiate functions;
- ability to make service requests of lower level components;
- storage space for the visual index; and
- control policy rules.

A.4.5.1 Inputs (functionalities/services/data)

The reasoning engine will need the following input from the object relation generator:

- data concerning candidate objects that are in relationships with objects specified in the original service request. The returned data will be used by the hypothesis manager to generate relevant plan hypotheses and entries in the visual index.

The reasoning engine will need the following input from lower level system component (gesture recogniser / object recognition / hand recognition):

- data about the objects specified in the originals service request. The returned data will be used by the hypothesis manager to revise the belief in the current set of plan hypotheses functions.

A.4.5.2 Computation

Very difficult to estimate. The computation effort required for the control policy will be low, so the computational load will be roughly proportional to the number of hypotheses.

A.4.5.3 Quality of service

The quality of service that the reasoning engine provides to the activity planner can be defined in terms of the number of relevant atomic plan primitives that are generated for any given scenario exemplar. If no primitives are generated, then no activity plan synthesis will occur. Too many primitives suggests that erroneous plan concepts are being generated. This aspect of quality of service can be regulated by adjusting the belief value for transition from hypothesis to concept.

The effect of varying quality of service from requests made to the object relation generator and other lower components is on the extent to which it will be possible to revise hypothesis belief values.

In either case, it should be possible to define a mechanism for setting the belief thresholds adaptively.

A.4.6 Stop condition

The stopping condition would be defined by a request from the activity planner.

A.4.7 Stopping action

- Cancel all outstanding service requests to object relation generator and other lower level components.
- Reset control policy to nominal “rule 1” condition.
- Delete all current plan hypotheses.
- Delete all entries in the visual index.

A.4.8 Requirements that are likely to be unfulfilled

Not applicable.

A.4.9 Example scenarios

Control policy

Rule1	**IF** no functions in H exceed activation threshold **AND** a hand recogniser service is available **THEN** make a service request to a hand recogniser to find hand candidates
Rule2	**IF** a hand object is found **AND** a gesture recogniser service is available **THEN** make a service request to the gesture recogniser to see if it moves with a consistent trajectory.
Rule3	**IF** (a hand object moves with a consistent trajectory) **AND** an object relation service is available **THEN** make a service request to the object relation generator to mark non-hand objects along that trajectory **AND** use attentive processing to establish classifications for objects where none has been provided by the pre-attentive phase.
Rule4	**IF** (a hand object moves with a non-consistent trajectory) **THEN** mark all non-hand objects in the scene.

Table: Example of control policy rules.

Sequence of events

1. User HMI is started, so the reasoning engine makes a service request to the activity planer to *capture_exemplar*. The activity planner makes a service request to the reasoning engine to *produce_plan_concept_functions*. No functions in ***H*** exceed activation threshold so control policy rule 1 is asserted.
2. After some time, a hand recogniser finds a candidate hand. A unary plan hypothesis find-hand*(<marker1>*) is created in the reasoning engine and *<marker1>* is entered in the visual index. The creation of the unary plan hypothesis causes control policy rule 1 to be revoked and rule 2 is asserted.
3. The pre-attentive phase tracks the hand which randomly moves around the picture. This continues for some time. The hand trajectory is non-consistent. The gesture recogniser cannot determine purposeful motion. Control policy rule 4 is asserted and appropriate service requests are made to lower level detection components.
4. The pre-attentive phase identifies 3 objects using *find_non_hand_object(<x>)* and marks them as <*marker2*>), <*marker3*> and <*marker4*> in the visual index. No specific coarse classification is readily achieved. Tracking of all 4 objects continues.
5. After some further time, the gesture recogniser determines that the hand is moving with purposeful motion. In response, control policy rule 4 is revoked and rule 3 is asserted. The objects marked as <*marker2*> and <*marker3*> are on that trajectory, but object <*marker4*> is not. Object <*marker4*> is deleted from the visual index. Service requests are made to the attentive phase is engaged to obtain classifications for objects <*marker2*> and <*marker3*>).
6. After some (yet to be defined) processing, the plan hypothesis function *grasped_by(<marker1>, <marker3>)* exceeds a minimum threshold and is input to the control policy. Object <*marker3*> has been determined to be a CD by the post-attentive

phase. There is not a very high level of belief in the plan hypothesis function grasped-by(*<marker1>, <marker3>*) and it has been fully instantiated so a plan concept function is generated *grasped_by(hand-1, cd-1)* and is sent to the activity plan generator. Object *<marker2>* is deleted from the visual index along with any remnant plan hypotheses functions that make reference to it.

7. The plan hypothesis *is_holding(<marker1>, <marker3>)* has also reached a high degree of belief and a corresponding plan concept function *is_holding(hand-1, cd-1)* is sent to the activity plan generator. In temporal terms it has been sent after *grasped_by(hand-1, cd-1)* which defines its temporal relationship to is-holding in this case.

At the end of this sequence, we have only 2 active markers *<marker1>* and *<marker3>.* What happens next is determined by further control policy rules which we have not defined yet.

A.5 Gesture recogniser (v3 by Jon Howell @ COGS)

A.5.1 Description

The Gesture Recogniser (GR) module is contained in the view-independent area of the ActIPret system and will use a Time-Delay Radial Basis Function (TDRBF) network to categorise characteristic purposive 3-D hand trajectories (gestures) over time. During the learning phase, the network learns the appearance of the general categories of these hand trajectories from a database of example data.

The tasks talked about in the project proposal DoW concern manipulation of objects by hands in 3-D space. Therefore the major purposive hand gestures (or sub-activities) that need to be recognised by this module are:

- A hand moving away from the torso
- A hand moving towards the torso

On initialisation by a service request from the Activity Reasoning Engine (ARE), these can be recognised by the GR via a TDRBF network using either 3-D hand trajectory information (relative to the torso centroid) or 6-D hand/pose trajectory information. This information is passed back to the ARE in order that more complex activities (such as the hand grasping an object) could be recognised via additional attentive queries, such as `Is the hand empty or holding something?'

Where more than one hand is present in the scene, several parallel service requests can be made by the ARE, one for each hand.

A.5.2 Principal output services of the functionality

The output for this module is a set of probabilities for task-significant hand gestures (for one specific hand marker) in an immediately-previous time period.

When the belief that a task-significant hand gesture has been observed has gone above some threshold ('looking promising'), this information would be attached to the hand marker for higher-level processing.

A.5.3 Start condition

The start condition for this module would be a service request for gesture information from the activity reasoning engine (ARE) once it was sufficiently confident that a hand object had

been found in the scene. Due to the nature of the time delay representation, some delay may occur before useful information is output from the module.

The module will send a service request to the hand tracking module to provide continuously-updated information about the marker until further notice. Time-stamps would need to be attached to this information, so that the module can normalise time-steps through its history (this allows for asynchronous connection between this module and all others).

Where more than one hand is present in the scene, keeping track of which hand is which needs to be a function of the ARE so that reasoning about occlusion and appearance/disappearance of hands can be handled in a proper manner. Since the GR and the hand tracker have to work in a task-relevant way, they should be initiated separately for each instance of hand found via a specific service request from the ARE.

A.5.4 Single-shot or continuous running behaviour

This function would run continuously (once started), using continually queued hand marker information from the hand tracker, in order that time series of trajectory information for the hand are maintained.

A.5.5 Requirements during run-time

The only requirement for this module is that the hand tracking module is able provide continuously-updated hand information as input for the network. The module will reflect hand tracking/detection confidence levels in the confidence levels of its output, i.e. where hand tracking becomes uncertain, the gesture confidences will be similarly low.

Some assumptions which need to be made about the user include:

- torso aligned to objects picked up/put down or inserting other objects into.
- the right hand is used for all manipulation (needed?)
- if a CD needs to be picked up, it is already out of its storage case (removes current need to track/monitor gestures for both hands).

A.5.6 Stop condition

Two possible conditions are:

- Termination from reasoning engine.
- Hand tracker loses hand or confidence in object being a hand goes below some threshold.

A.5.7 Stopping action

Once stopped, the module would inform the hand tracking module to stop sending trajectory information.

A.5.8 Requirements that are likely to be unfulfilled

The hand trajectory information needs to be provided in a coordinate system relative to the user's torso, but it is not anticipated that this would be technically difficult.

A.5.9 Example scenarios

For the training of the network component, a representative set of examples (say, at least 10 sequences) of each task-significant hand gesture will be required. This information will need to be in a form similar to that provided by the hand markers, i.e. giving torso-relative trajectories with timestamps and tracking confidence.

A.5.10 General comments

See deliverable 3.1 for more details.

A.6 Object relationship generator (v1 ACIN)

A.6.1 Description

The component Object Relationship Generator (ORG) controls the generation and maintenance of spatio-temporal relationships between two objects. It determines relationships between the objects (hand, CD player, button) in 3-D at one instance or over several cycles. The relationships can be quantitative (frame relative to frame) or qualitative (proximity, above, left, moves along, etc.).

Specific functions

1. To confirm/create specific deictic relationship queries. This uses specific references to objects for answering a specific (single shot) query, for example, *proximity<Obj1, Obj2>*, where *Obj1/2* is a reference to a given object)
2. To create deictic relationships. This uses other components to obtain potential object candidates to establish the relationship. The relationship determines levels of task-related interest of combinations, for example, proximity. The evaluation of the relationship may be recreated each cycle or updated (see below). Priorities for this would be determined by the Control Policy, e.g., *'find objects on hand trajectory'* (with priority *p*).
3. To update current deictic relationships. This newly evaluates a previously created relationship each cycle by using the updates from other components, such as tracking.

Note that the ORG might send a request to another component to deliver the objects needed for a functions, such as item 2. In particular, this request for an object is specified with a region of interest (ROI). For example, the request to (continuously) monitor a region sent to a pre-attentive process (detector). This allows the reasoning engine to work in an abstract, task-related domain, as the ORG calculates the specific coordinates for regions of interest. It would also be responsible for the later terminating of this monitoring.

Specific relationships

The specific relationships derived by the Object Relationship Generator are simple, task-related operators and are in two forms (see deliverable 3.1 for more details): *Basic relationships* are generally behaviours of a single object or simple relations between two objects. *Aggregate relationships* are relations between two (or more) objects and in the spatio-temporal domain.

These types of relationships can be calculated using Dynamic Decision Networks (DDNs) (see Hilary's paper 'Task-based (Cognitive) Control for ActIPret Project' for more details) or using a Kalman filter approach. Of particular interest for the ActIPret system will be:

- **Purposive behaviour trajectory:** a single object trajectory is of interest for the synthesis process, if the object (hand) makes a purposive behaviour trajectory, that is a trajectory that is task relevant. An example is a hand moving towards an object. The

goal is to detect the purposive trajectory as early as possible to be able to focus processing to the area indicated by the direction of the trajectory. An example is grasping, where processing should be focused on the objects at the estimated grasp location. Methods to detect a purposive behaviour trajectory can be based on Kalman filters (as proposed by ACIN) or a time-delay Radial Basis Function (RBF) network (as proposed by COGS).

- **Distance between two objects (mutual proximity):** the exact distance between two objects depends on the representation of the objects used. The simplest representation uses the reference coordinate frames. In this case the distance is the Euclidian distance between the origins of the two object coordinate frames. If the objects are represented with a hull, the distance can be defined as the closest point between the two hulls.
- **Find objects near to each other**: this service provides a pre-attentive predictive cue, based on the assumption that the closer two objects are to each other, the more likely they are to have some task-relatedness.
- **Object near trajectory of object:** this service provides a pre-attentive predictive cue, based on the assumption that the closer an object is to a hand's trajectory, the more likely it is to be manipulated by that hand.

.

Typical service

"*Get related object candidates*"

- input: type of relationship (proximity, near object trajectory, etc.) from service requester, reference to object (using its pose and/or trajectory) from anywhere;
- output: hypothesis of relations (value/qualitatively, with confidence measure) and related object reference

A.6.2 Principal output services/data

Principal:

- 3-D relationships such as relative pose (expressed as pose values or as above/below/left/right/etc. classification), shortest distance to object (proximity), etc.

Additional options:

- A specific list of relationships can be selected as needed.
- Uncertainty (standard deviation) of pose relationship.
- 2-D relationships such as vicinity (in pixel or "sloppy" terms), relative orientation, relative motion vector (getting closer, etc.), etc.

A.6.3 Start condition (must be initialised with)

- no initialisation required (or needed at the moment).
- Start: takes input from Activity Reasoning Engine and requests input, for example, of tracking and recognition functions.

Re-start conditions (was running and only suspended shortly):

- Old relationships, possibly old region of interest (ROI = tracking window) could be reused.

What information would make your function more robust:

- Depends mainly on tracking/detection/recognition results.
- Prediction from last values possible.
- Recognised actions and knowledge about hypothesis.

A.6.4 Single-shot or continuous running behaviour

- Single shot is possible.
- In general: continuous as soon as a request is placed. Uses tracking to obtain continuous values.
- Trigger: works on service request.

A.6.5 Requirements during run-time

A.6.5.1 Camera and view

- View: send 3-D pose relationship to tracking/recognition component to enable these components to derive a potential occlusion and obtain a better view, e.g. from the side. Does not request views by itself.
- Both objects in image, should support good pose estimation of objects, since this is the basis for the relationships.
- (2-D relationships: tell little/nothing about better views).

A.6.5.2 Inputs (functionalities/services/data)

- Pose of object (3-D = 6DOF), object trajectory (3-D + time).
- Pose of recognition (3-D).
- Confidence values of features and objects.
- Additional: 2-D features; context knowledge (e.g. present activity hypothesis).

A.6.5.3 Computation

- not known yet, minor (Kalman filter) to medium (DDN).

A.6.5.4 Quality of service

- Measure: confidence measure of relationship.
- Format of the expected quality: confidence = [0-1] (=0% - 100%).

Expected improvement if more information (relations, context) becomes available (see A.6.3):

- potential prediction from last values improves results;
- impairment of quality of service if stopped/restarted: negligible, see A.6.3; and
- quality:
    - $\Rightarrow$ robustness: Confidence measures aggregated from tracking results; and
    - $\Rightarrow$ accuracy: dimensions of pose estimation or qualitatively.

A.6.6 Stop condition

- Can be stopped by requesting component with/without finishing calculation at present cycle.
- Stop/suspend execution: last relationships can be reported, last state values can be retained.
- What is necessary to restart a stopped Function?

A.6.7 Stopping action

- If stopped and allowed to finish: all relationships are available, otherwise the relationships that have been already computed are available.
- Available from a stopped functionality: last input and calculated values.

A.6.8 Requirements that are likely to be unfulfilled

- To compute all possible combinations of relations (combinatorial explosion); hence it works only when triggered by another component to enable pruning by relevance (vicinity, importance/confidence values of objects, etc.).

A.6.9 Example scenarios

Scenario 1: grasping (hand + object).

Scenario 2: transferring object (hand + objects (CD + CD-Player)).

Scenario 3: releasing object (hand + objects (CD + CD-Player)).

Scenario 4: pressing button (vicinity/touch hand button).

For each scenario:

- situations: different users/hands, different lightings, different finger grips, different CDs, different views;
- the more sequences the better (min 10, best is a min of 3 for each situation); and
- mono/stereo: both.

A.6.10 General comments

"High" level input (context) must be considered in more detail.

A.7 Service list with view controller (v1 by Gerald Umgeher @ Profactor)

A.7.1 Description

The service list with view controller is similar to the normal Service List specified by ACIN. This process provides additional functionality to assign a specific robot to a requested service.

This component contains the view controllers, which are responsible to operate the view requests of the services. It contains functionality to merge the requested viewpoints and initiates the movement of the robots.

A.7.2 Principal output services/data

- assigns a robot to the requested service
- finds a component for the requested service
- moves the robot arms/cameras to desired positions
- alignment of cameras to obtain desired viewpoints

A.7.3 Start condition (must be initialised with)

Home position of the robots.

A.7.4 Single-shot or continuous running behaviour

Service: Continuous operation, quasi-trigger: new pose/viewpoint.

A.7.5 Requirements during run-time

A.7.5.1 Camera and view

Controls the viewpoint.

A.7.5.2 Inputs (functionalities/services/data)

- desired fixation point (tracked object) and orientation + distance + zoom-value
- from where to fixate it (including allowed deviations/constraints)
- steady camera / moving camera / maximum motion velocity (to prevent smearing features)
- Collision Avoidance functionality.

A.7.5.3 Computation

Not known yet. (The actual robot-joint-controller is running on a reserved PC)

A.7.5.4 Quality of service

Accuracy of the robot.

A.7.6 Stop condition

Never stops.

A.7.7 Stopping action

Never stops.

A.7.8 Requirements that are likely to be unfulfilled

None.

A.7.9 Example scenarios

None.

A.8 Hand detector and tracker (v1 by Antonis Argyros @ FORTH)

A.8.1 Description

The functionality of this service is to detect and track human hands present in the scene viewed by the ActiPret stereoscopic vision system. This functionality will be based on:

1. **Colour information:** Skin colour models will be used as a cue indicating the presence of hands in the viewed scene. Several such models are currently under review and preliminary yet promising results are already available for the first example sequence provided by Profactor.
2. **Motion information:** Since hands participate in activities and actions, a skin-coloured region in the image becomes more salient if it moves. Motion is measured with respect to a world-centred coordinate system, which actually coincides with the coordinate system in which camera positions are measured and reported. Because both the hand and the observer move concurrently, the solution of the problem relies on knowledge of the depth of the scene. Provided that depth is known (stereoscopic processing) and camera egomotion is also known (encoder information), a 3-D motion field for rigid objects (i.e. non hands) can be predicted. The actual flow is then compared to the hypothesized flow and significant deviations between these two flows are attributed to independently moving objects. To compute motion information the hand tracker invokes the *motion detection service*.
3. **Depth information:** Depth information is necessary for reporting 3-D position of hands in the image. The "locality of reference principle" (in simpler words, the continuity of the 3-D locations of a hand as a function of time) will be used to resolve ambiguities. Depth information can (and most probably will) be used in one more way: It is unlikely that a hand will appear on the floor or on the ceiling of the scene. Therefore the saliency of hand hypotheses should also be influenced by the 3-D position of a hand in the 3-D volume of the workspace. Current efforts towards depth estimation are towards reviewing techniques for computing dense correspondence maps between two stereoscopic views.
4. **Tracking mechanisms**: The previous modules provide visual cues to tracking mechanisms that will track the various hand hypotheses in the scene. Kalman filtering is a classical tracking mechanism. Additionally, current investigations include particle filters and more specifically, tracking based on the CONDENSATION algorithm.

The techniques above are accompanied with other methods to compute 3-D camera positions beneficial for observing the scene and tracking the hands present in the scene. For the moment, two criteria are considered:

- the most salient hand hypothesis should remain in the field of view of both cameras; and
- The viewpoint should be such that the most salient hand moves in an almost front-parallel plane (appears to be moving laterally).

A.8.2 Principal output services/data

- A list of 3-D coordinates, each of which represents the location of a "hand hypothesis" in the scene. The 3-D location refers to the centre of mass (centroid) of the hand. The coordinates are given with respect to a world-centred (non task-centred) origin that coincides to the coordinate system in which camera positions are reported.
- Confidence value of 3-D coordinates for each hand hypothesis. This quantifies the certainty about the reported 3-D position of each hand hypothesis.
- The list of 3-D coordinates is then ranked according to these confidence values.

The quality of the above results highly depends on the accuracy of the provided intrinsic camera parameters and camera position information.

A.8.3 Start condition (must be initialised with)

Initial position of hand(s) in image: This start condition was reported in the initial function description. Effort will be put to avoid this start condition so that hand hypotheses can be formulated and further tracked by the system, even if a hand is not present in the initial scene, but appears later in time.

A.8.4 Single-shot or continuous running behaviour

Continuous by definition.

A.8.5 Requirements during run-time

- The system will be able to track hands from the time they appear in the field of view, until they disappear. If a hand disappears from the scene and then appears later, then a new hypothesis will be formed which will have no relation to the previous one.
- Minimal occlusion of hand(s) by other objects or self-occlusion.

A.8.5.1 Inputs (functionalities/services/data)

- Synchronized image stereo pair.
- Accurate knowledge of camera/stereo parameters.

A.8.5.2 Computation

- Accurate figures not available.
- It is almost sure that the hand tracker will be further decomposed into a number of constituent elementary modules/services that will run as separate processes in different workstations. However, this decomposition will be transparent to the rest of the system/framework..

A.8.5.3 Quality of service

Measure: confidence value of 3-D coordinates.

A.8.6 Stop condition

Requesting component stops tracking (will it ever happen?)

A.8.7 Stopping action

Last return value (available from a stopped functionality): the set of outputs for the last processed input before stopping (see section A.8.2 above).

A.8.8 Requirements that are likely to be unfulfilled

Hand tracking under:

- severe occlusion;
- too distant observations (hands cover small area in the field of view); and
- variable lighting conditions.

A.8.9 Example scenarios

Important in practically all ActiPret scenarios.

A.8.10 General comments

Required accuracy for 3-D trajectory computation needs to be quantified and matched to request demands.

A.9 Object detector and tracker (ACIN)

A.9.1 Description

The functionality of this component is model-based detection and tracking of objects. Tracking means to regularly determine the object location (and orientation) in 3-D in soft real-time (latency not longer than several frame cycles). 3-D pose is extracted from one or more 2-D camera views. The model is either a wire-frame representation using vertices, lines, ellipses and regions (for man-made objects like a CD player) or just colour, colour histograms or texture (e.g. for regions). Detection is one possible initialisation of tracking and can be based on colour, colour histograms or texture derived from the model. Using data of extern recognition is another way for initialisation.

A.9.2 Principal output services/data of the functionality

Principal

- 3-D pose (position and orientation) relative to world coordinate frame at each time step (20/40 ms).
- Uncertainty value of pose (standard deviation).

Additional

- Confidence value of object tracking.
- Reprojection of object into images, image location of (found/projected) features.
- Location and confidence values of individual features in images in 2-D (e.g. to inform other trackers about possible occlusions)
- Transformation to other coordinate systems (e.g. camera, head).

A.9.3 Start-condition (must be initialised with)

- 3-D Space of interest (SOI).
- camera poses.
- object model (at least some known).
- 3-D object start pose (if known).

Re-start conditions (was running and only suspended shortly):

- Old SOI could be reused.
- Set new SOI when motion has been followed with other sensors (e.g., only egomotion of head or robot).
- Camera poses.

What information would make your function more robust?:

- Start pose known.
- To know about occlusions (e.g., regions determined as hand, arm, another object in other functions).
- Preference to initialise an object with a specific pose (on the table in one or two possible poses).
- Other constraints on pose (like limited change of pose from frame to frame).
- Other pose estimates (from relationships, recognised actions and knowledge).
- To know about egomotion of cameras.
- Rich models (i.e. containing colour, texture, distinguished regions, etc.).
- Lighting conditions.

A.9.4 Single-shot or continuous running behaviour

- Detection is single shot.
- Tracking is a continuous operation at field/frame rate (20/40 ms).
- Trigger: new image signal from camera or frame grabber (initialisation: see A.9.3.)

A.9.5 Requirements during run-time

A.9.5.1 Cameras and views

- Requested object in at least one field of view: full object in view, object fills 75% of view (calibrated camera parameters).
- Preferable views exist for every object: e.g., CD: viewed from top or bottom, angle to surface normal < 80 degrees (better: surface size in image c. > 20 pixels); CD-Player, front plate with buttons: same constraint of surface normal.
- Constraints on zoom: camera calibration needed to correct distortion, hence better no zoom.

A.9.5.2 Inputs (functionalities/services/data)

- Image acquisition: storing image in memory, image number and new image signal.

- Camera poses.

A.9.5.3 Computation

- Detection: no idea yet.
- Required average processing time on a Pentium III 1000MHz (without image grabbing): 1-2 ms per line feature, 3-5 ms per ellipse.
- several ms for pose calculation.

A.9.5.4 Quality of service

- Measure (no change of view): confidence value of pose (possible: Confidence value of tracking).
- Measure (change of view as requested): improvement factor (to be implemented).
- Format of the expected quality: confidence = [0-1] (=0% - 100%).
- Expected improvement if more information (relations, context) becomes available (see A.9.3).
- Occlusions known: very high improvement.
- Pose restrictions: significant improvement for detection.
- Impairment of Quality of service if stopped/restarted: relatively high, see A.9.3.
- Quality.
- Robustness: confidence measure of tracking the right object presently i.e. number of features tracked successfully.
- Accuracy: pixel location in image and pose plus uncertainty measure of pose (see A.9.2).

A.9.6 Stop condition

- Requesting component stops tracking.
- Tracking and/or pose calculation fails.
- Stop/Suspend execution: return value is last pose.
- What is necessary to restart a stopped function?

A.9.7 Stopping action

- Last return value: pose
- Available from a stopped functionality: last SOI, last pose, last feature values.

A.9.8 Requirements that are likely to be unfulfilled

- image motion not too fast to have smearing features (but we will not be this fast yet).
- no occlusions, perfect lighting, uncluttered background etc.

A.9.9 Example scenarios

Scenario 1: take CD, transfer and place CD:

- situations: different users/hands, different lightings, different finger grips, different CDs, different views;
- the more sequences the better (min 10, best is a min of 3 for each situation); and
- mono/stereo: both (=stereo, we can treat it as mono).

A.9.10 General comments

Meta data such as whether objects may be grasped will not determined *a priori*.

A.10 Object recogniser (v1 by Jiri Matas @ CMP)

A.10.1 Description

This component uses an example based object representation to learn and recognise objects in a scenario.

A.10.2 Principal output services/data

- Provides information about identity of objects in the field of view.
- Provides region in the image where the object is present.

A.10.3 Start condition (must be initialised with)

- A database of object representations.
- Image (optionally region of interest).
- Some representation of priorities/expectations of objects present.
- Pose restrictions may input resulting in a speed-up.

A.10.4 Single-shot or continuous running behaviour

Single-shot.

A.10.5 Requirements during run-time

A.10.5.1 Camera and view

None. (I envisage cooperation between the object recognition and detection module and the camera controller, but at this moment I am not sure what will the structure of control be). By camera controller we mean the image server (frame grabber) as well as the lens/zoom, robot arm and active head.

A.10.5.2 Inputs (functionalities/services/data)

- Image.
- Image + region of interest (recognition of tracked objects).

- Image + label (verification of an expected object).

A.10.5.3 Computation

Heavy.

A.10.6 Stop condition

- Recognition/localisation of a single object of interest.
- Recognition/localisation of *k* objects.
- Time limit reached.

A.10.7 Stopping action

None.

A.10.8 Requirements that are likely to be unfulfilled

It is anticipated that moderate occlusion will not be a problem. Performance will deteriorate gracefully with the level of occlusion.

A.10.9 Example scenarios

- Training views of the objects. Preferably:
  - ⇒ resolution as high as in the test views;
  - ⇒ one object per image (for a start); and
  - ⇒ no occlusion.
- Test views, which should be images from a 'normal' tutor session.

A.10.10 General comments

The above refers to recognition in the expert and tutor phases only. Object learning is performed as part of the system learning phase, possibly using standard databases.

A.11 Hand recogniser (FORTH)

To be defined as required.

A.12 Motion detector (v1 by Antonis Argyros @ FORTH)

A.12.1 Description

The Motion Detector module finds moving objects in the scene, while the observer is also moving. Motion is measured with respect to a world-centred coordinate system, which actually coincides with the coordinate system in which camera positions are measured and reported. Since the observer also moves, the solution of the problem relies on knowledge of the depth of the scene. Provided that depth is known (stereoscopic processing) and camera egomotion is also known, a motion field can be predicted, assuming that the 3-D world

remains rigid. The actual flow is then compared to the predicted flow and significant deviations between these two flows are attributed to independently moving objects.

A.12.2 Principal output services/data

A binary image: points belonging to independently moving objects have a “one” label and points belonging to static (with respect to the world coordinate system) points have a “zero” label.

A.12.3 Start condition (must be initialised with)

Depth estimation of the scene (possibly provided by another service, yet to be defined).

A.12.4 Single-shot or continuous running behaviour

Continuous by definition.

A.12.5 Requirements during run-time

- Accurate depth estimation.
- Accurate egomotion parameters.

A.12.5.1 Inputs (functionalities/services/data)

- Image.
- System/camera/motion parameters.
- Depth estimation computed from stereo.

A.12.5.2 Computation

Accurate figures not available, estimated figures are rather reasonable for almost real time implementation.

A.12.5.3 Quality of service

Confidence value of motion detection.

A.12.6 Stop condition

All requesting components are stopped

A.12.7 Stopping action

Last return value, available from a stopped functionality: the set of outputs for the last processed input before stopping (see section A12.2 above).

A.12.8 Requirements that are likely to be unfulfilled

Accurate motion detection for very small motions.

A.12.9 Example scenarios

Always useful, not linked to particular scenarios.

A.13 Ellipse detector (CMP+ACIN)

A.13.1 Principal Description of Functionality

Detects ellipses in an intensity (colour) image. Ellipses can be partially occluded. Ellipses are found by grouping of convex edge strings.

A.13.2 What is/are the principal output services/data of the Functionality?

- Number of ellipses found in the image.
- 5 parameters for each ellipse found.
- Confidence for each ellipse found.

A.13.3 Start-Condition (must be initialized with)

- region of interest (ROI) in the image
- terminate after 'number' of ellipses found
- reference to the image to search

Initial guesses or constraints for the ellipse parameters could make the process more robust. Also the knowledge of inside colour will make the process faster and more robust.

A.13.4 Single-shot or continuous running behavior

Single shot, triggered by another module.

A.13.5 Requirements during Run-Time

Camera+View:

- none

Inputs (Functinalities/Services/Data):

- Image server must be running

Computation:

- rather high, but very data dependent (many ellipses, noisy image -> higher)
- estimated: faster than 1 second

A.13.6 Stop Condition

Stops when all ellipses found.
But can also be terminated anytime. Then the ellipses found so far are returned.

A.13.7 Stopping action

none.

A.13.8 In General: Requirements that will very likely be unfulfilled:

No occlusion. Performance will, hopefully, deteriorate gracefully with the level of occlusion.

A.13.9 Example Scenarios

Same as for object recognition.

A.14 Image server (camera) (v1 by Gerald Umgeher @ Profactor)

A.14.1 Description

The Image Server is responsible to provide the images of cameras, which are mounted on a robot. In order to avoid sending whole images with TCP/IP we will use shared memory to transmit the data.

A.14.2 Principal output services/data

The service provides the key and the size of the shared memory.

It also sends the timestamp of the image acquisition.

The format of the image is also part of this information (in ActIPret “YUV422” is used).

The shared memory key and the image is stored in a ring-buffer.

Components that request this service must be co-located in order to use shared memory.

A.14.3 Start condition (must be initialised with)

No initialisation of the service is needed. The Service List is necessary to register the service.

The service must be started at the computer where the FireWire card is placed.

A.14.4 Single-shot or continuous running behaviour

The service should be started in continuous mode at a fixed frame rate.

A.14.5 Requirements during run-time

A.14.5.1 Camera and view

Provides the images of the requested view.

A.14.5.2 Computation

The computation time of the service is not known exactly but will depend on the frequency of image acquisition. Tests show that it is not very high.

A.14.5.3 Quality of service

It is not possible to define a quality for this service.

A.14.6 Stop condition

The service never stops because it is a continuous running component. It stops when you shutdown the whole system.

A.14.7 Stopping action

Remove the service entry form the Service List.

A.14.8 Requirements that are likely to be unfulfilled

None.

A.14.9 Example scenarios

None.

A.15 Pose server (v1 by Gerald Umgeher @ Profactor)

A.15.1 Description

This service responsible to provides the current position and orientation of the robot-TCP (Tool Centre Point). The service also provides the current information of the camera angles, which are mounted at the robot.

A.15.2 Principal output services/data

- Position and Orientation of the TCP in world coordinates.
- Angles of the camera-head and baseline as well as.
  - ⇒ position and orientation of the cameras in world coordinates; and
  - ⇒ position/status of a gripper (not really necessary for ActIPret).

A.15.3 Start condition (must be initialised with)

No initialisation of the service is needed. The Service List is necessary to register the service.

A.15.4 Single-shot or continuous running behaviour

It is running in Single-Step mode and provides the information when the service is requested.

A.15.5 Requirements during run-time

A.15.5.1 Camera and view

The service provides the current information about the view. Each robot needs his own Pose Server.

A.15.5.2 Computation

The computation time of this service is very low.

A.15.5.3 Quality of service

Accuracy of the robot.

A.15.6 Stop condition

There is no stop condition.

A.15.7 Stopping action

Remove the service entry form the Service List.

A.15.8 Requirements that are likely to be unfulfilled

None.

A.15.9 Example scenarios

None.

A.16 CPU controller (v1 by Gerald Umgeher @ Profactor)

A.16.1 Description

The CPU Controller is a service, which determines the available CPU-Time of a specific computer. The Information is used to find a computer with the lowest CPU usage for a requested service.

A.16.2 Principal output services/data

Provides the percent of the unused (idle) CPU time of the computer where the CPU controller service is executed. It also provides the performance (bogomips) of the computer. By combining this two information you can find a computer with the highest available CPU power.

A.16.3 Start condition (must be initialised with)

No initialisation of the service is needed. The Service List is necessary to register the service. The service must be started on each computer and provides only the information of this CPU load.

A.16.4 Single-shot or continuous running behaviour

It is running in single-step mode and provides the information when the service is requested. (It is also possible to run the service continuously and update the information in the Service List)

A.16.5 Requirements during run-time

A.16.5.1 Computation

The computation time of this service is very low.

A.16.5.2 Quality of service

It is not possible to define a quality for this service.

A.16.6 Stop condition

There is no stop condition.

A.16.7 Stopping action

Remove the service entry form the Service List.

A.16.8 Requirements that are likely to be unfulfilled

None.

A.16.9 Example scenarios

None.

A.17 Scene modelling for visualisation (CMP)

A.17.1 Description

The three-dimensional scene will be described using the VRML standard and presented using third-party VRML browser/viewer. The objects can be parameterised (scale, colour, surface texture, etc.). Humanoid model should be H-Anim 1.1 compliant and it can be parameterised, too. Data describing motion can be converted to the VRML interpolator format and used in VRML scene.

For the task of presenting human activity using the VR tools, an approach of VRML based scene possibly connected to the application via EAI interface has been used. The following text presents basic components for such task.

VRML International Standard

ISO/IEC 14772, the Virtual Reality Modeling Language (VRML), defines a file format that integrates 3D graphics and multimedia. Conceptually, each VRML file is a 3D time-based space that contains graphic and aural objects that can be dynamically modified through a variety of mechanisms. This VRML language defines a primary set of objects and mechanisms that encourage composition, encapsulation, and extension.

Java EAI Interface

External Authoring Interface (EAI) is an interface between VRML scene and application. It provides one of the mechanisms of dynamical modification of the VRML object. The EAI specification is the second part of the VRML ISO standard.

H-Anim 1.1

A specification of the standard virtual humanoid body model based on VRML is suitable for presenting human activities in a virtual environment. It defines the virtual human body structure, building elements and provides anthropometrical data of the human body. The specification can be found at www.hanim.org.

A.17.2 Demo scene

Scene with the CD and the player will present an animation of human inserting the CD into the player in three-dimensional virtual environment. It is aimed to highlight possibilities of chosen approach.

WP 7 “Scene modelling and representation” starts only in months 7, hence more details will be available at the year one Deliverable.