A Knowledge-Based Planner for Processing Unconstrained Underwater Videos

Citation for published version:
Nadarajan, G, Chen-Burger, J & Fisher, B 2009, A Knowledge-Based Planner for Processing Unconstrained Underwater Videos. in *Proceedings IJCAI-09 Workshop on Learning Structural Knowledge From Observations.* pp. 37-44. <http://www.inf.ed.ac.uk/publications/online/1384.pdf>

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Early version, also known as pre-print

Published In:
Proceedings IJCAI-09 Workshop on Learning Structural Knowledge From Observations

Nadarajan, G., Chen-Burger, J., & Fisher, B. (2009). A Knowledge-Based Planner for Processing Unconstrained Underwater Videos. In Proceedings IJCAI-09 Workshop on Learning Structural Knowledge From Observations. (pp. 37-44).

A Knowledge-Based Planner for Processing Unconstrained Underwater Videos

Gayathri Nadarajan, Yun-Heh Chen-Burger, Robert B. Fisher
CISA, School of Informatics, University of Edinburgh, U.K.
gaya.n@ed.ac.uk, {jessicac,rbf}@inf.ed.ac.uk

Abstract

Video data collected continuously are pervasive today but analyzing them in an efficient manner has proven to be a challenge. This is because raw data is unlabelled and prone to noise, causing difficulty in extracting knowledge. With the aid of user-provided domain knowledge and heuristics used by image processing experts, an automated solution is implemented. It makes use of formalisms for goal-directed behavior in the form of hierarchical task networks (HTNs). These are incorporated within a novel workflow composition framework that aims to assist naive users conduct complex video processing tasks automatically. An example is illustrated for video classification, fish detection and fish counting in unconstrained underwater videos.

1 Introduction

The Taiwanese Ecogrid project [2006] offers a unique opportunity for long term ecological monitoring and planning via the integration of geographically distributed sensors, computing power and storage resources into a uniform and secure platform for continuous information gathering. Wireless sensor nets have been installed and managed in several national parks in Taiwan and the information collected is stored in and made available through Ecogrid for access. These include surveillance videos in the entire area of Fu-Shan National Park for observing natural lives and protecting them from potential poachers, audio recording of rare frog species, undersea coral reef and marine life observation stations and more. Due to the continuous and non-intrusive methods deployed, such monitoring and recording efforts have already made ecological discoveries of significant importance that traditional methods otherwise could not have made.

However, there is a great challenge as how this data may be transformed into useable information for the ecologists in a timely fashion. For instance, a one minute video clip typically takes 1500+ frames and is stored in 3–4 MB. This translates into 200+ MB per minute, 5+ GB per day and 2 TB per year for one operational camera, and due to the unpredictability of nature, one may not easily skip frames as they may contain vital information. It is estimated that one minute's clip will cost 15 minutes of manual processing time on average. This means that one year's recording of a camera would cost human experts 15 years' effort just to perform basic analyzing and classifying tasks. This is exacerbated with the presence of three underwater cameras in operation currently. This is clearly an impractical situation and appropriate *automation* methods must be introduced.

In this context, the challenge lies in the fact that there is a lack of effort in conceptualizing the tasks involved in the process of video analysis, as the majority of vision developers focus on improving low-level techniques and algorithms that perform with extreme accuracy. The traditional approach used by image processing experts of providing highly specialized solutions for specific tasks would contribute little to the problem at hand. Moreover, the goals and constraints are not restricted, making it more difficult for image processing experts to come up with generalized solutions.

This work aims to tackle this problem by providing a planning- and semantics-based workflow system. First, the system overview is outlined in section 3. Next, the derivation and usage of HTN plans are described in sections 4 and 5 along with some sample results. Section 6 finally concludes.

2 Related Work

Attempts to solve automatically image processing problems were conducted within knowledge-based systems such as OCAPI [Clément and Thonnat, 1993] , MVP [Chien and Mortensen, 1996] and BORG [Clouard *et al.*, 1999]. However these systems remain limited to a list of restricted and well known goals. Therefore *a priori* knowledge about the application context (domain-specific concepts such as sensor type, noise, lighting, etc) and about the goal to achieve were implicitly encoded in the knowledge base. This implicit knowledge restricts the range of application domains for these systems and it is one of the reasons for their failure.

Within the workflow community, major systems such as Pegasus [Deelman *et al.*, 2004], Taverna [Oinn *et al.*, 2004] and Kepler [Ludäscher *et al.*, 2005] are composed manually. Thus the user, who is usually a domain expert (e.g. bioinformaticians using Taverna), is responsible for constructing the workflow based on their goals. Only Pegasus has the additional capability of automatic workflow composition in the form of mapping abstract non-executable workflows to their concrete executable forms. Pegasus utilizes deferred planning to generate partial executable workflows based on al-

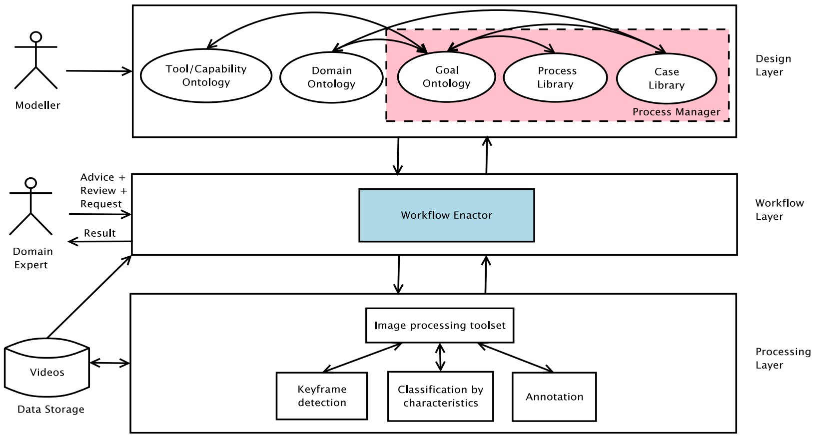

Figure 1: Hybrid Workflow Composition Framework for Video Processing.

ready executed tasks and the currently available resources by a partitioner. This allows for dynamic scheduling that would prevent workflows from failing to execute should any of the resources fail. Although this is a step towards performance optimization and reliability, Pegasus is still limited in that it does not support looping which is essential for the modeling of iterative processes such as image processing.

Currently there is no easy way to provide image processing solutions when there is no clear understanding of the application domain or the object(s) within the images that need to be manipulated with. To address this issue, we use a knowledge-based approach to provide a rich and flexible mechanism that would allow automatic process selection and dynamic workflow composition that can provide suitable image processing solutions in a problem domain that is not well-understood. This is an ambitious aim, as a starting point we limit our scope to uncontrolled underwater marine life observations by using the domain knowledge to address the inherent ambiguities and to derive useful structure. In the past only image processing experts had access to the software libraries and could improve the quality of the solutions by trial and error cycles based on past experiences. Now, by understanding the background and domain knowledge, non-image processing experts can also have access to the tools and can conduct experiments themselves. Planning, combined with semantics-based technologies such as ontologies could prove useful in generating automatic solutions for pervasive problem domains such as video processing. A framework that incorporates these features is outlined next.

3 Overview of Design and Implementation

In order to tackle multiple user objectives and to capitalize on the strengths of various image processing tools and the capability of planning systems, a robust architecture is required to derive good enough answers for users. This can be achieved by incorporating all these components within an integrated framework. Figure 1 illustrates the proposed framework that aims to provide automation for the video processing application domain [Nadarajan *et al.*, 2006]. It distinguishes three levels of abstraction through the design, workflow and processing layers, that capture knowledge of varying structural complexity.

3.1 Design Layer

At the top-most layer, high level concepts are provided by the user which are used for the goal formulation. The design layer contains components that describe the domain, goals, capabilities and processes to be carried out in the system. These are represented by three ontologies and two libraries. The goal ontology contains the classes of tasks (e.g. *Detection, Classification, Segmentation*) with the constraint qualifiers for the goal (e.g. *Performance Criteria, Accuracy, Occurrence, Quality Criteria*). The domain ontology describes the videos whereby qualitative concepts such as “blur” and “clear” for image clearness and “low”, “medium” and “high” for brightness level, among others, are included. The capability ontology contains the classes of video and image processing tools according to their functions. It also contains the performance level of the tools according to domain description and/or constraints based on experimental findings by image processing experts. A modeler is able to manipulate the components of the design layer, for example populate the libraries and modify the ontologies.

3.2 Workflow Layer

The workflow layer acts as the main interface between the design and processing layers. This layer ensures the smooth interaction between the components, access to various resources such as raw data, image and video processing tool set, interface to user, as well as the provision of the final output. This is provided by a workflow enactor, that acts as the interpreter of the events that occur within the system. Section 3.4 describes the workings of this layer in more detail.

3.3 Processing Layer

The processing layer consists of a set of image and video processing tools that act on the data. The functions of these tools are represented in the capability ontology in the design layer. Once a workflow is composed, these tools may work on the videos directly. It should be noted that for each capability, there could be more than one tool available. Depending on the quality of the video and the task at hand, each tool may perform with a different level of accuracy. Thus, having the user provide feedback on the performance of a particular combination of tools would be beneficial to the system's future improvement. The central idea of this architecture is that users who do not possess image processing expertise can conduct complex video processing tasks with the help of an automated system and their domain expertise. This is realized via a planning- and ontology-based workflow enactor that acts as the main interface between the high-level user requests and the low-level application components.

3.4 User-System Interaction

The workflow engine is implemented based on the framework proposed in Figure 1. Using a declarative approach, the workflow engine and planner are developed using SICStus Prolog 3.12.5. The user-system interaction is given by Figure 2 and further explained below.

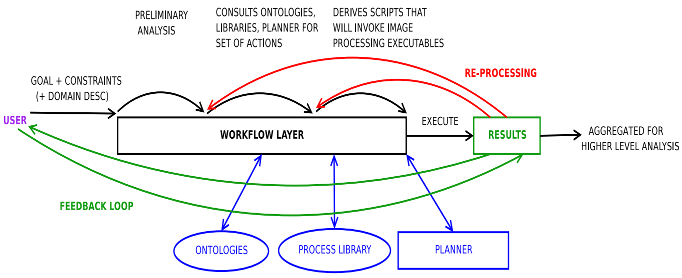

Figure 2: Overview of interaction between the user, workflow and other components in the system.

The system first prompts the user for the goal (leftmost in Figure 2). Then, the constraints and initial domain description are obtained via a *Preliminary Analysis* stage. These are provided by the user, otherwise, generated automatically. Once the goal, constraints and initial domain information are formulated, they are checked against the goal and domain ontologies for consistency. The workflow then consults the planner, process library and capability ontology for the set of actions required to achieve the goal. The result of that is the invocation of the image processing executables corresponding to these actions on the input video. The result of the image processing task is displayed to the user for evaluation. Based on the user's assessment, the system will re-process (or replan) parts where necessary. In this way, the overall performance of the system can be improved incrementally.

Example Task: Classify Video, Detect Fish, Count Fish

Suppose the user wants a video clip to be classified based on brightness (luminosity), clearness (smoothness), level of green tone (to detect the presence of algae on the camera lens), occurrence of fish and count of fish. Thus, given an input video clip, the output is a new clip identical to the original one with annotated values for the criteria given above. Figure 3 provides a few example results for this task [Spampinato *et al.*, 2008]. The user specifies the goal, constraints and domain description based on valid inputs presented by the system. Otherwise the constraints and domain description will be generated automatically using default values recommended by image processing experts. The terms representing the goal and constraints given by the user are checked with the concepts in the goal ontology.

Figure 3: Three sample annotated video captures for the task video classification, fish detection and fish counting.

Each term in the list of goals provided by the user should match with an instance in the goal ontology and the class of this goal instance should be a subclass of the class "Goal". The constraints provided by the user are checked with the instances of the classes "Performance Criteria", "Quality Criteria", "Accuracy" and "Occurrence" respectively. Due to space limitation, a partial representation of the goal ontology with relevant instances in bold and their immediate classes highlighted is given by Figure 4. Nadarajan and Renouf [2007] describe the three ontologies in further detail and illustrate their usage in the image processing problem formulation. These are then formulated as Prolog predicates and checked against the goal and domain ontologies, represented in a first order logic-based data language, FBPML-DL [Chen-Burger and Stader, 2003]. This formalism was chosen because a logical language for planning, process modeling and workflow composition in one integrated system was required. The recommended ontology language, OWL [McGuinness and van Harmelen, 2004] does not provide direct reasoning support for programming languages, hence it was not selected. A sample set of values for the goal and constraints is given below:

Goal: [classify_video, detect_presence_fish, count_fish]
Constraints: [Occurrences = all, Performance Criteria = processing_time, Quality Criteria = best_compromise, Accuracy = prefer_miss_than_false_alarm]

The Prolog representation to describe the goal (including constraints) is given by the caption and predicate below:

```
% goal(Goal_list, Ordering, Constraints, Solution).
goal([classify_video(Filename), detect_presence_fish(all), count_fish(all)], [], [processing_time, best_compromise, prefer_miss_than_false_alarm, all], Solution).
```

Filename refers to the input video in MPEG format. Ordering is explicitly stated if one of the tasks must be executed before another. Solution is the list of all the steps returned by the planner eventually. A preliminary analysis is then conducted to obtain the initial domain description. This can be done manually (given by the user) or automatically. The initial domain description is also checked for consistency with the domain ontology as with the goal and constraints. A sample domain description is as follows:

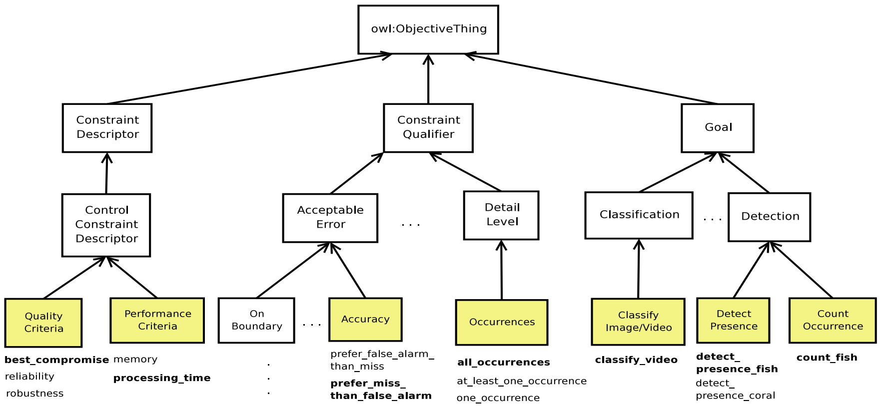

Figure 4: Goal ontology with relevant classes and instances for the task "classify video, detect fish and count fish".

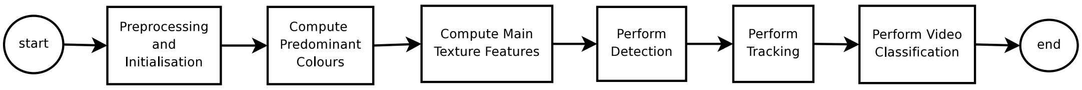

Figure 5: Top-level plan for goal "Classify Video, Detect Fish and Count Fish".

Initial Domain: [Brightness = medium, Smoothness = clear, Percentage Background Object = low]

The Prolog predicate to represent this initial description is asserted into the knowledge base.

start_state([brightness(medium), smoothness(clear), percentage_bg_object(low)]).

Other prominent domain information that are not shown above but will be discussed in the next section include initial speed of object movement, texture features and type of noise. The goal, constraints and initial domain description are fed into the planner that selects a sequence of steps using hierarchical decomposition. This will be described further in the following section.

4 Deriving HTN Plans from Domain Experts and Image Processing Heuristics

Generating solutions for video processing tasks can be defined as a planning problem where the goals are high-level user requirements, such as "Detection", "Classification", "Segmentation" and so on. Nadarajan [2007] provided groundwork for our planning component which has since been used to generate plans for more complex tasks. The constraints provided by the user or generated automatically will be used as preconditions for operator selection. The operators are the image processing tools (or capabilities) that are involved in achieving these goals. Close examination of image processing programs has unveiled how the developers go about solving video processing problems and the heuristics that they use for the selection of actions. Essentially a high level task is broken down into a sequence of one or more subtasks until primitive processes are encountered. This can be demonstrated better with an example. For the goal "Classify Video", "Detect Fish" and "Count Fish" described in the previous section, the high level process breakdown is given by Figure 5. Using the conventions provided by Nau, Ghallab, & Traverso [2004], this could be described using HTN method as follows:

top-level(*file, frame_no, curr_frame_img, bg_model, img_with_blobs, texture_features*)
task: classify_video-detect_fish-count_fish(*file, frame_no, curr_frame_img, bg_model, img_with_blobs, texture_features*)
subtasks:
u_1 = preprocessing_initialization(*file*),
u_2 = compute_predominant_colours(*frame_no, curr_frame_img*),
u_3 = compute_main_texture_features(*frame_no, curr_frame_img*),
u_4 = perform_detection(*frame_no, curr_frame_img, bg_model*),
u_5 = perform_tracking(*frame_no, curr_frame_img, img_with_blobs*),
u_6 = perform_video_classification(*file, texture_features*)
constr: $u_1 \prec u_2, u_2 \prec u_3, u_3 \prec u_4, u_4 \prec u_5, u_5 \prec u_6$

Each of the subtasks is further decomposable to other primitive and non-primitive subtasks. Initially, using a top-down approach, the operators were represented by the primitive tasks in an image processing program. A primitive task is one that is not further decomposable, for instance a function call within a vision library, or an arithmetic, logical or assignment operation. This was done based on the fact that each function call within the image processing library could represent a suitable executable component that requires precondition(s) and results in postcondition(s). Each primitive task may take in one or more input values and return one or more output values. Experimentation has shown that a typical program written using a standard vision library such as OpenCV[1] for a task of average complexity contains approximately 1000 lines of code that correspond to

[1] Available as http://sourceforge.net/projects/opencvlibrary/

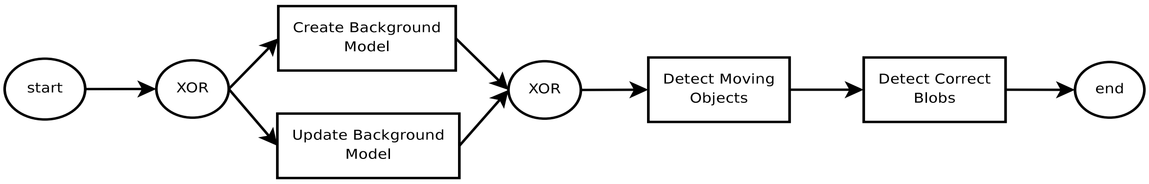

Figure 6: Process Model for "Perform Detection", a subtask within the goal "Classify Video, Detect Fish, Count Fish".

Background Model	Domain		Constraint	
	Criteria	Value	Criteria	Value
Gaussian	Clearness Speed of Movement % Background Object Noise Type	High Low Low Gaussian		
Gaussian Mixture	Texture Features Speed of Movement % Background Object Noise Type	Variance, 4th moment High Medium Gaussian	Accuracy	Prefer miss than false alarm
Moving Average	Speed of Movement (for computing alpha)		Accuracy Performance	Prefer false alarm than miss Processing time
Intra-Frame	% Background Object Noise Type	High Gaussian		
W4	Texture Features % Background Object Noise Type	Mean, 3rd moment High Salt & Pepper		
Poisson	Texture Features	Mean, Variance		
Adaptive Poisson	Texture Features	Histogram, Mean Variance		

Table 1: Description of Domain Information and Constraints in Background Model Creation

thousands of operator invocations over multiple frames in a video. For example, a sample run for the video classification, fish detection and counting task using this level of granularity has yielded 85 unique operators (function calls) and a total of 11511 steps in 1.16 seconds for a video of 50 frames. However, this would result in scalability issues when the video size is larger. A three-minute video clip containing 5400 frames would cause a tenfold increase in the number of execution steps and processing time, thus a coarser level of granularity would be more appropriate.

Adopting a bottom-up approach next, sets of primitive tasks were grouped or "chunked" under meaningful headings as executables. Repeated discussions with image processing experts resulted in the refinement of the processes and the modification of the process library and ontologies with respect to these changes. The advantage of this bottom-up refinement approach has allowed us to identify modules that could be reused for most video processing tasks. A combined systematic approach of top-down and bottom-up refinements has resulted in the specification of 31 executables that represent the *independent components* or operators in the process library (see Appendix A). This setting is not just easier to manage, the independent components also represent meaningful subtasks that could be reused for various other image processing tasks. The derived executables are defined according to their input, output, pre-conditions and post-conditions in agreement with image processing experts.

This provided a more efficient way of generating image processing solutions by reusing existing executables than having an application-specific image processing program developed from scratch each time a video processing task needs to be done. A re-run of the workflow engine using this new level of granularity on the same 50-framed video for the same task has resulted in 1456 steps in 0.44 seconds, almost eight times fewer steps and 62% faster than the initial outcome. Methods are in place to reduce this number of steps further, for instance further refinement of the executables and elimination of loops with known number of iterations from within the planner, where possible. An example for background model creation will be illustrated in the next section.

5 Example: Background Model Creation

An important component of the video processing task described in the previous section is "Perform Detection". The process model for this subtask is given by Figure 6. As can be seen a decision is made whether to create a background model or update a background model (depicted by the `XOR` construct), depending on the current frame number. The background model is created if the current frame is the first frame, otherwise it is updated in accordance to the existing background model. The creation of the background model is a suitable example to illustrate this as it provides a spectrum of cases where different background models could be created based on these features. Seven types of background model have been identified and of these only one will be selected for a particular video depending on domain description and/or constraints. Table 1 outlines the domain and constraint descriptions for all the background model types. To demonstrate a few variations, the HTN methods for creating the Gaussian mixture model and the moving average model are given below.

bg-model2(*curr_frame_img, dir, frame_no, high, medium, gaussian, prefer_miss_than_false_alarm*)

task:	create-gaussian-mixture-model(*curr_frame_img, frame_no, dir, high, medium, gaussian, prefer_miss_than_false_alarm*)
subtasks:	u_1 = create_gmm_model(*curr_frame_img, dir*)
constr:	before($\{u_1\}$, frame_no(1), init_speed(*high*), percentage_bg_obj(*medium*), noise_type(*gaussian*), accuracy(*prefer_miss_than_false_alarm*))

bg-model3(*curr_frame_img, dir, frame_no, learning_speed, prefer_false_alarm_than_miss, processing_time*)

task:	create-moving-average-model(*curr_frame_img, dir, frame_no, learning_speed, prefer_false_alarm_than_miss, processing_time*)
subtasks:	u_1 = create_ma_model(*curr_frame_img, dir)*
constr:	before($\{u_1\}$, frame_no(1), init_speed(*high*), accuracy(*prefer_false_alarm_than_miss*), performance(*processing_time*))

In the first method, for the task `create-gaussian-mixture-model`, the preconditions that must be met are the initial speed of movement is high, the percentage of the background object is medium the noise type is Gaussian and the accuracy value is prefer_miss_than_false_alarm. For the second method, the moving average model is the most preferred model when the initial speed of movement is high, the accuracy value is prefer_false_alarm_than_miss and the performance criterion is processing_time. Note also that the parameter learning speed that is passed into this method is determined using the initial speed of movement. Some of this information is provided by the user at the initial stage after specifying the goal. In the capability ontology these conditions are incorporated as criteria with performance measures that are tied to the operators that may perform the respective background model creation tasks. Thus the best tool to perform a task may be determined using the information derived from the capability ontology.

Based on the experimental findings and domain understanding provided by the video observations, planning using HTNs is a suitable approach for automatically generating solutions for video processing tasks. This is because there are many ways to achieve a vision task, depending on the quality of the video, the tools available to perform the task, and the constraints on the goal. HTN planning would allow different methods for solving the same task to be incorporated into the inference engine. Thus, by capturing some of the best-practices used by image processing experts to solve vision problems, we have gained some invaluable insight into the processes involved in solving such complex tasks.

6 Conclusion

Observations acquired from unconstrained underwater videos have assisted in the modeling of plans through the understanding of the application domain and the relevant processes involved to solve some of the known problems pertaining to the application. This has resulted in the derivation of formalisms for goal-directed behavior in the form of ontologies and HTNs. These enabled the rapid generation of a prototype of an approximate image processing system that produced sub-optimal but sufficiently accurate answers for naive users.

Acknowledgments

The authors would like to acknowledge Concetto Spampinato from University of Catania, Italy for collaboration on the implementation of the image processing executables. Access to research image data was sponsored by the Royal Society of Edinburgh, U.K., National Center for High Performance Computing (NCHC) and Academia Sinica, Taiwan.

A List of Independent Components

A.1 Pre-processing and Initialization

1. **View Video**
   This component takes a video file, determines its frame rate, number of frames, compression algorithm and finally displays the video to the screen. A conservative filtering is also applied to remove noise. This component will create a directory and store candidate background images in it.
2. **Preliminary Analysis**
   This module is responsible for capturing the initial domain description of the video. Taking in the video file, it retrieves the initial brightness, clearness, speed of movement, date and initial texture features. These features are contained in the domain ontology. A small motion detection system is also performed to identify background movement, e.g. plants.
3. **Grab Frame Image**
   This component takes in a video file and a frame number, retrieves the image for that frame and stores it for further processing. This component is necessary as each frame of the video is processed using its image representation.

A.2 Compute Pre-dominant Colours

4. **Extract RGB Colours**
   This component extracts the red, green, blue and yellow channels of a frame.

A.3 Compute Main Texture Features

5. **Compute Histogram**
   This component takes in a frame image, computes its histogram value and image representation. User may select to view the histogram representation.

6. **Compute Main Statistical Moments**
This module determines main statistical moments such as mean, variance, third and fourth moment, entropy, uniformity and smoothness given a set of histogram values (e.g. obtained from component 5).

7. **Compute Gabor Filter**
This component applies a Gabor filter onto a complex image made up of a real and an imaginary part. The absolute value of the complex image is computed, followed by the mean and variance, which are texture features. For 4 angles and 3 scales, this will yield 12 mean-variance values, so for each image 24 values will be extracted. These features could be used for the detection of coral reef, for instance.

A.4 Perform Detection

8. **Create Gaussian Background Model**
Given a frame image, this module creates a background model represented by 2 images; foreground and background. It stores the background model in a directory given as input. This background model works well for videos where movement of background objects vary slightly and **not** suitable for videos with *waving trees*.

9. **Create Gaussian Mixture Model**
Similar to component 8 above, the background model created by this component is represented by 2 images and stored in a specified directory. In addition, this module overcomes the limitation of component 8 and is suitable for videos with *waving trees*.

10. **Create Moving Average Model**
This module takes in a frame image, a directory to store the resulting background model and a *learning speed (alpha)* and creates an image that represents the background model. The advantage of this algorithm is that it does not require a learning phase. Alpha is dependent on the speed of movement, which could be obtained from preliminary analysis (component 2).

11. **Create Intra-Frame Difference Model**
This module only requires a directory where the background model needs to be stored. The background model is represented by 2 matrices. This algorithm overcomes the limitations of Gaussian background model (component 8) but is problematic for videos with impulsive noise such as salt and pepper noise.

12. **Create W4 Background Model**
This module also only requires a directory where the background model is to be stored. The background model is represented by 3 matrices. This algorithm overcomes the limitations of components 8 and 11 and is particularly suitable for videos with low changes in luminosity (e.g. indoor applications).

13. **Create Poisson Model**
As with 11 and 12 this module takes in a directory where the background model is to be stored. 2 matrices are created to represent the background model. This algorithm is particularly suitable for videos with uniform background colour (e.g. blue water, tar road).

14. **Create Adaptive Poisson Model**
Similar to the previous 3, a directory name is required to store the background model. The background model is represented by 2 images; foreground and background. This algorithm is similar to component 13, additionally it can manage colour variation with light changes.

15. **Update Moving Average Model**
This component takes in a frame image and an exisiting moving average background model to create a new background model (1 image). It utilizes absolute subtraction between pixels.

16. **Update Background Model**
This component works on all background models *except* for moving average model. It takes in a frame image and an existing background model (either images or matrices) and creates a new background model accordingly. The algorithm checks that a pixel in an image is in a range of values. It can cater for input of different number of images, hence it could be used for different background models.

17. **Detect Moving Objects**
This module creates an image with identified blobs from a frame image and a background model (which could be 1, 2 or 3 images). The result is a binary image. The algorithm works by removing occlusion and small objects. It also utilizes statistical or derivative algorithm based on the type of the background model.

18. **Perform Morphological Operation**
This operation is only applicable to some background models. Given an image with identified blobs (e.g. from component 17), noisy detected pixels are removed.

A.5 Perform Tracking

19. **Extract HSV values**
This component extracts the images for the hue, saturation and value channels for a given frame image. A preprocessing step (e.g. histogram equalization) could be included to exclude green colour if algae is present.

20. **Compute Backprojection**
This module depends on component 19 as it takes in a hue channel to create a histogram of the hue channel, which is then used to create an image which represents the backprojection of the hue plane. The hue plane is used because it gives the most useful colour information for an image.

21. **Compute Connected Components**
This module takes in an image with potential blobs (e.g. from component 18) and returns an image with the correct blobs and the total number of blobs in that image.

22. **Compute Ratio Area Convex Hull over Area Blob**
This component calculates the ratio between the area of the convex hull of a blob and the area of the blob itself.

It is executed over all blobs in an image (e.g. image from component 21).

23. **Compute Camshift**
This component predicts what a blob will look like in the next frame. Given the backprojection of a hue plane, an image with identified blobs and the current blob number, this algorithm draws a bounding box of the blob and returns the center, orientation and size of the blob. It is executed over all blobs in the image.

24. **Compute Closest Blob**
This component is reponsible for finding the minimum Euclidian distance between two blobs. Thus, given the centers, orientations and sizes of two blobs, this distance is computed. It is executed in a double loop to compare a blob with all the blobs in a segment of consecutive frames (e.g. component 10) to find the *closest* blob.

25. **Count and Write Number of Fish in Frame**
This module takes in an array of mininum distances (computed from component 24 for example), the ratio between the blob area and frame area and writes the number of fish (blobs) onto an output frame. It also sets the blob as being counted.

26. **Count and Write Number of Fish in Video**
This component takes in an array of boolean values to represent whether a blob has been counted or not (component 25) and writes the number of fish in the video onto an output frame.

27. **Determine Presence of Fish Blocking Screen**
Given the area of the blob and the area of the frame, this module will return a true or false value to inform if a blob (fish) is blocking the screen. The minimum threshold for this condition to hold is set to 70%.

A.6 Perform Classification

28. **Compute and Write Average Luminosity**
This module is responsible for determining if the brightness level of a video is “bright”, “medium” or “dark” based on its average luminosity value, calculated using the mean and 3rd moment. The value is written onto the output frame.

29. **Compute and Write Average Clearness**
This module is responsible for determining if the smoothness level of a video is “clear” or “blur” based on its average clearness value, calculated using the variance, 4th moment, entropy and uniformity. The value is written onto the output frame.

30. **Compute and Write Presence of Fish**
This module is responsible for determining if the video has “fishes” or “no fishes” based on the number of fish in the video. The value is written onto the output frame.

31. **Compute and Write Presence of Algae**
This module is responsible for determining if algae is present or not in the video, categorized as “green” or “not green” based on its green channel value. The value is written onto the output frame.

References

[Chen-Burger and Stader, 2003] Y.-H. Chen-Burger and J. Stader. Formal Support for Adaptive Workflow Systems in a Distributed Environment. In Layna Fisher, editor, *Workflow Handbook*. Future Strategies Inc., 2003.

[Chien and Mortensen, 1996] S.A. Chien and H.B. Mortensen. Automating Image Processing for Scientific Data Analysis of a Large Image Database. *IEEE PAMI*, 18(8):854–859, August 1996.

[Clément and Thonnat, 1993] V. Clément and M. Thonnat. A Knowledge-Based Approach to Integration of Image Procedures Processing. *CVGIP: Image Understanding*, 57(2):166–184, Mar 1993.

[Clouard *et al.*, 1999] R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg : A Knowledge-Based System for Automatic Generation of Image Processing Programs. *IEEE PAMI*, 21(2):128–144, 1999.

[Deelman *et al.*, 2004] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G.Mehta, S. Patil, M.H. Su, K. Vahi, and M.Livny. Pegasus: Mapping Scientific Workflows onto the Grid. In *Across Grids Conference*, 2004.

[Ecogrid, 2006] Ecogrid. National Center for High Performance Computing (NCHC), Taiwan, 2006. URL: http://ecogrid.nchc.org.tw/.

[Ludäscher *et al.*, 2005] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 2005.

[McGuinness and van Harmelen, 2004] D. McGuinness and F. van Harmelen. *OWL Web Ontology Language*. World Wide Web Consortium (W3C), 2004.

[Nadarajan and Renouf, 2007] G. Nadarajan and A. Renouf. A Modular Approach for Automating Video Processing. In *CAIP*, 2007.

[Nadarajan *et al.*, 2006] G. Nadarajan, Y.-H. Chen-Burger, and J. Malone. Semantic-Based Workflow Composition for Video Processing in the Grid. In *IEEE/ACM Web Intelligence*, 2006. pp. 161-165.

[Nadarajan, 2007] G. Nadarajan. Planning for Video Processing using Ontology-Based Workflow. In *ICAPS*, 2007.

[Nau *et al.*, 2004] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[Oinn *et al.*, 2004] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P.Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 2004. 20(17):3045–3054.

[Spampinato *et al.*, 2008] C. Spampinato, Y.-H. Chen-Burger, G. Nadarajan, and R. B. Fisher. Detecting, Tracking and Counting Fish in Low Quality Unconstrained Underwater Videos. In *VISAPP*, 2008.