Bridging the Sense-Reasoning Gap using the Knowledge Processing Middleware DyKnow

Fredrik Heintz, Piotr Rudol, and Patrick Doherty

Department of Computer and Information Science
Linköpings universitet, Sweden
{frehe, pioru, patdo}@ida.liu.se

Abstract. To achieve sophisticated missions an autonomous UAV operating in a complex and dynamic environments must create and maintain situational awareness. It is achieved by continually gathering information from many sources, selecting the relevant information for the current task, and deriving models about the environment and the UAV itself. Often models close to the sensor data, suitable for traditional control, are not sufficient for deliberative services. More abstract models are required to bridge the sense-reasoning gap. This paper presents how DyKnow, a knowledge processing middleware, can bridge the gap in a concrete UAV traffic monitoring application. In the presented example sequences of color and thermal images are used to construct and maintain qualitative object structures modeling the parts of the environment necessary to recognize the traffic behavior of the tracked vehicles in realtime. The system has been implemented and tested both in simulation and on data collected during test flights. [1]

1 Introduction

Unmanned aerial vehicles (UAVs) are becoming more and more commonly used in both civil and military applications. Especially missions which are considered dull, dirty and dangerous could benefit from being more or less automated. One important application domain for UAVs is different types of surveillance missions. Such missions may involve flying over unknown areas to build terrain models, to quickly get an overview of a disaster area including helping the rescue services to find injured people and deliver medical supplies, or to help law enforcement agencies to monitor areas or people for ongoing or potential criminal activity. To achieve these complex missions an autonomous UAV must continually gather information from many different sources, including sensors, databases, other UAVs, and human operators, select the relevant information for the current task, and derive higher-level knowledge about the environment and the UAV itself in order to understand what is happening and to make the appropriate decisions. In other words, the UAV must create and maintain situational awareness.

To become situationally aware the UAV needs to build models of the environment and use them to reason about what is going on. These models should be constructed from the information gathered from the distributed sources as close as possible to realtime in order to capture the latest developments and be up to date. Since there are

[1] This work is supported in part by the National Aeronautics Research Program NFFP04 S4203 and the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII.

infinitely many models that could be built and the UAV has limited resources it is important that the appropriate models are constructed for the particular task at hand. When the task changes the models should reflect this change as well. What is an appropriate model will depend on what properties of the world which are relevant, what reasoning is needed to make decisions to achieve the task, and the context within which the reasoning is made. One important problem is therefore to construct models with different properties from information collected by various sensors that can be used to reason about the environment and the UAV in realtime. Traditionally this problem has been separated in two different approaches, the "sensor" approach constructing quantitative models based on sensor signals and the "reasoning" approach constructing qualitative models using formal languages based on common sense reasoning. Not only have the models been different, but also the purpose of the models have been radically different. For example, while the "reasoning" approach has mainly focused on building general and global domain models the "sensor" approach has worked on building local models based on specific sensors and assumptions. The problem with the "reasoning" approach is that the models are very difficult to implement in uncertain and dynamic environments, while the "sensor" approaches are designed to handle these cases but fail to reason about events on a more global scale and to build overview models using encoded background knowledge. The gap between these two approaches is called the sense-reasoning gap. The goal is to close this gap by constructing high-level models of the environment using the information collected by sensors. The purpose of this paper is to present an approach which tries to bridge this gap in order to provide a solution which integrates the large amount of work done in both communities.

The application is implemented and has been tested both in simulation and on data collected during test flights.

2 Traffic Monitoring

Imagine a human operator trying to maintain situational awareness about the traffic situation in an urban area using UAVs looking for accidents, reckless driving, or other relevant activities. One approach would be for one or more UAVs to relay videos and other data to the operator for human inspection. Another, more scalable approach, would be for the UAVs to monitor the traffic situations which arise and only report back the high level events observed, such as cars turning in intersections and doing overtakes, to reduce the amount of information and help the operator focus her attention. This paper describes such a traffic monitoring application where cars are tracked by a UAV platform and streams of observations are fused with a model of the road system in order to draw conclusions about the behavior of the cars in the environment. The input consists of images taken by the color and thermal cameras on the UAV which are fused and geolocated to a single world position. This stream of positions is then correlated with a geographical information system (GIS) in order to know where in a road system the object is located. Based on this information high level behaviors such as turning in intersections and overtaking are recognized in realtime as they develop using a chronicle recognition system.

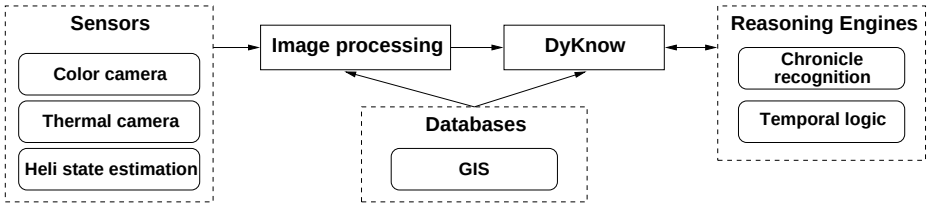

Fig. 1. Overview of the components of the traffic monitoring application.

An overview of the components of the traffic monitoring application is shown in Fig. 1. The three sensors used, the two cameras and the heli state estimation (which is fused from INS and GPS data), are shown to the left. These provide the primitive information about the environment and the UAV. The next component is the image processing system which tracks objects seen by the cameras. When an object is being tracked the images from the two cameras are fused to provide an estimation of the position in the world of the tracked object. Each time a new frame is analysed a new position estimate is produced. From this stream of position estimations, the system recognizes high level events such as turning in intersections, overtaking, and so on.

To describe these events a formal representation called *chronicles* is used [1]. A chronicle defines a class of events by a simple temporal network (STN) [2] where the nodes are primitive events and the edges are temporal constraints between the occurrences of the primitive events. If a stream of primitive events contains a set of event occurrences which satisfies all the constraints then an instance of the chronicle is recognized. The chronicles used in this application contain primitive events which capture the structure of the road network, qualitative information about cars such as which road segment they are on, and qualitative spatial relations between cars such as beside and behind. If all occurrences of the primitive events are in the stream used by the chronicle recognition engine then the recognition is complete, meaning that all chronicle instances in the stream will be recognized. Creating this stream of events, accurately representing the environment of the UAV, based on sensor data is a concrete instance of bridging the sense reasoning gap. This is done by a knowledge processing middleware called DyKnow (see Section 5) [3, 4].

DyKnow takes the stream of position observations provided by the image processing system (see Section 4) and derives an event stream representation of cars and qualitative spatial relations between cars (see Section 7.1). DyKnow also derives an event stream representation of the road network from the information stored in the GIS. One issue that must be handled is how to anchor the car symbols used in the chronicles to objects being tracked. Since the image processing system may lose track of cars or start tracking other objects than cars DyKnow has to dynamically estimate and continually monitor the type and identity of objects being tracked. To do this, the normative behavior of different objects and the conditions for assuming that two objects have the same identity are described using temporal logic (see Section 6). When a tracked object is found which satisfies the normative behavior for e.g. a car a new car representation is created and the tracked object is *linked* to the new car representation. From this moment

the car representation will be updated each time the tracked object is updated. Since links only represent hypotheses, they are always subject to becoming invalid given additional observations, and therefore the UAV continually has to verify the validity of the links. This is done by monitoring that the normative behavior of the assumed object type is not violated. For example, an object assumed to be a car must not violate the normative constraints on cars, e.g. leaving the road. If it does violate the corresponding link is removed, in other words the object is no longer assumed to be a car. To evaluate temporal logical formulas DyKnow has to derive temporal models representing the value of the variables used in the formulas. Since these models are derived from the sensor data it is another concrete example of how DyKnow can be used to bridge the sense reasoning gap.

3 The Hardware Platform

The WITAS UAV platform [5] is a slightly modified Yamaha RMAX helicopter (Fig. 2), equipped with a more efficient power generator and a taller landing gear. Its total length is 3.6 m, including the main rotor, and it is powered by a 21 horse power two-stroke engine. The maximum takeoff weight of the platform is 95 kg and it can stay in the air up to one hour. The onboard avionics system is enclosed in an easily detachable box mounted on the side of the UAV.

Fig. 2. The WITAS RMAX autonomous helicopter.

Fig. 3. The onboard color and thermal cameras mounted on a pan-tilt unit.

The onboard computer system contains three PC104 embedded computers. The primary flight control (PFC) system runs on a Pentium-III (700MHz) and is responsible for sensor fusion required for basic helicopter control and for sequentializing control modes (i.e. takeoff, hover, 3D path following, landing, vehicle following etc.). PFC interfaces with the RMAX helicopter through YAS (Yamaha Attitude Sensor) and YACS (Yamaha Attitude Control System) and receives data from the GPS receiver and a barometric altitude sensor. The deliberative/reactive control (DRC) system runs on a Pentium-M (1.4GHz) and executes deliberative functionalities such as planning, execution monitoring, and scenario recognition.

The image processing (IPC) system runs on a Pentium-III (700MHz) embedded computer. The camera platform suspended under the UAV fuselage is vibration isolated by a system of springs. The platform consists of a Sony color CCD block camera FCB-780P and a ThermalEye-3600AS miniature infrared camera mounted rigidly on a pan-tilt unit (Fig. 3). Both cameras delivers analogue PAL signals with the frame size 768x576 pixels at a rate of 25Hz.

Network communication between the onboard computers is physically realized with serial lines (RS232C point-to-point realtime communication) and Ethernet (non-realtime communication). Finally, the onboard system contains two miniDV video recorders controlled by software through a Control-L (LANC) interface. The live video is also sent to the ground and presented to the UAV operator. The recorded video is synchronized with the log data (i.e. complete UAV state) allowing off-line processing.

4 Image Processing

The task of image processing in this work is to calculate world coordinates of vehicles tracked in video sequences. First, an object tracker is used to find pixel coordinates of the vehicle of interest based on color and thermal input images. Second, the geographical location of the object is calculated and expressed as world coordinates. The object tracker developed for the purpose of this work can be initialized automatically or manually. The automatic mode chooses the warmest object on a road segment (description fetched from the GIS database) within the thermal camera view and within a certain distance from the UAV (the process of calculating the distance to a tracked object is explained below). The area around the initial point is checked for homogeneity in thermal and color images. The object is used to initialize the tracker if its area is consistent with the size of a car signature. This initialization method works with satisfactory results for distances up to around 50m from the tracked object.

If the tracker is initialized incorrectly the user can choose the object of interest manually by clicking on a frame of the color or thermal video. The corresponding pixel position (for color and thermal images) is calculated knowing the parameters of the cameras, the UAV's position and attitude and the model of the ground elevation. After initialization, tracking of an object is performed independently in color and thermal video streams. Tracking in the thermal image is achieved by finding the extreme value, the warmest or coldest spot, within a small window, i.e. 5 percent of the image size, around the previous result. Object tracking in color video sequences is also performed within such a small window and is done by finding the center of mass of the color blob in the HSI color space. The thresholding parameters are updated to compensate for illumination changes. Tracking in both images is performed at full frame rate (i.e. 25Hz) which allows for compensating for moderate illumination changes and moderate speeds of relative motion between the UAV and the tracked object. The problem of automatic reinitialization in case of loss of tracking is not addressed in this work. The result from the thermal image tracking is preferred if the trackers do not agree on the tracking solution.

In order to find the distance to the tracked object as well as corresponding regions in both images, the cameras have been calibrated to find their intrinsic and extrinsic

parameters. The color camera has been calibrated using the Matlab Camera Calibration Toolkit [6]. The same toolkit could not be used for finding optical parameters of the thermal camera because it was infeasible to obtain sharp images of the chessboard calibration pattern. To find focal length, principal point and the lens distortion parameters, a custom calibration pattern and the calibration toolkit by Wengert et. al. has been used [7]. The extrinsic parameters of the cameras were found by minimizing the error between the calculated corresponding pixel positions for several video sequences.

Finding pixel correspondences between the two cameras can not be achieved by feature matching commonly used in stereo vision algorithms since objects generally appear different in color and thermal images. Because of this, the distance to an object whose projection lies in a given pixel must be determined. Knowing the camera intrinsic and extrinsic parameters, helicopter attitude and position, and the ground elevation model, the distance to an object can easily be calculated as the distance from the camera center to the intersection between the ground plane and the ray going through the pixel belonging to the object of interest. For the environment in which the flight tests were performed the error introduced by the flat world assumption is neglectable. Finally, calculating pixel correspondences between the two cameras can be achieved by performing pixel geolocalization using intrinsic and extrinsic parameters of one of the cameras followed by applying inverse procedure (i.e. projection of geographical location) using the other camera parameters.

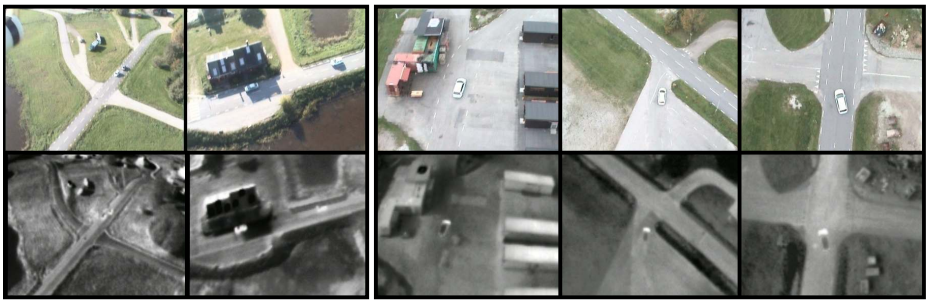

Fig. 4. A. Two frames from video sequence with the UAV hovering close to a road segment observing two cars performing overtaking maneuver. **B**. Three frames from video sequence with the UAV following a driving car passing road crossings. Top row contains color images and bottom row contains corresponding thermal images.

Using the described object tracker several data series of world coordinates of tracked vehicles were generated. Two kinds of video sequences were used as data sources. In the first kind (Fig. 4A) the UAV is stationary at altitudes of 50 and 60 meters and observes two vehicles as they drive on a nearby road. In the second kind (Fig. 4B) both the car and the UAV are moving. The ground vehicle drives several hundreds meters on the road system passing through two crossings and the UAV follows the car at altitudes

between 25 and 50 meters. For sequences containing two cars, the tracker was executed twice to track both vehicles independently.

A precise measure of the error of the computed world location of the tracked object is not known because the true location of the cars was not registered during the flight tests. The accuracy of the computation is influenced by several factors, such as error in the UAV position and the springs in the camera platform suspension, but the tracker in general delivers world coordinates with enough accuracy to determine which side of the road a car is driving on. Thus the maximum error can be estimated to be below 4-5 meters for distances to the object of around 80 meters.

5 DyKnow

To facilitate the development of general traffic scenario recognition application a knowledge processing middleware called DyKnow is used [3, 4]. The main purpose of DyKnow is to provide generic and well-structured software support for the processes involved in generating state, object, and event abstractions about the environments of complex systems. The generation is done at many levels of abstraction beginning with low level quantitative sensor data and resulting in qualitative data structures which are grounded in the world and can be interpreted as knowledge by the system. To produce these structures the system supports operations on data and event streams at many different levels of abstraction. For the result to be useful, the processing must be done in a timely manner so that the UAV can react in time to changes in the environment. The resulting structures are used by various functionalities in a deliberative/reactive architecture for control, situation awareness and assessment, monitoring, and planning to achieve mission goals. In the current application it is used to derive high level information about the objects being tracked. DyKnow provides a declarative language for specifying the structures needed by the different subsystems. Based on this specification it creates representations of the external world and the internal state of a UAV based on observations and a priori knowledge, such as facts stored in databases.

Conceptually, a knowledge processing middleware processes streams generated by different components in a distributed system. These streams may be viewed as representations of time-series data and may start as continuous streams from sensors or sequences of queries to databases. Eventually, they will contribute to definitions of more refined, composite, knowledge structures. Knowledge producing processes combine such streams by computing, synchronizing, filtering and approximating to derive higher level abstractions. A knowledge producing process has different quality of service properties such as maximum delay, trade-off between quality and delay, how to calculate missing values and so on, which together define the semantics of the knowledge derived by the process. It is important to realize that knowledge is not static, but is a continually evolving collection of structures which are updated as new information becomes available from sensors and other sources. Therefore, the emphasis is on the continuous and ongoing knowledge derivation process, which can be monitored and influenced at runtime. The same streams of data may be processed differently by different parts of the architecture by tailoring the knowledge processes relative to the needs and constraints associated with the tasks at hand. This allows DyKnow to support

easy integration of existing sensors, databases, reasoning engines and other knowledge producing services.

5.1 Fluent Streams

For modelling purposes, the environment of the UAV is viewed as consisting of physical and non-physical *objects*, *properties* associated with these objects, and *relations* between these objects. The properties and relations associated with objects will be called *features*, which may be static or dynamic. Due to the potentially dynamic nature of a feature, that is, its ability to change values through time, a total function from time to value called a *fluent* is associated with each feature. It is this fluent, representing the value over time of a feature, which is being modelled.

A *fluent stream* is a partial representation of a fluent, where a stream of *samples* of the value of the feature at specific time-points is seen as an approximation of the fluent. A sample can either come from an observation of the feature or a computation which results in an estimation of the value at the particular time-point, called the *valid time*. If the samples are ordered by the time they become available to the fluent stream, then the result is a stream of samples representing the value of the feature over time, that is, an approximation of its fluent. The time-point when a sample is made available or added to a fluent stream is called the *add time*. To be able to totally order the samples in a fluent stream a requirement that the add time is unique is imposed, i.e. at most one sample may be added to a fluent stream at any given time-point. A fluent stream has certain properties such as start and end time, maximum size, i.e. number of samples in the stream, sample rate and maximum delay. These properties are specified by a declarative *policy* which describes the constraints placed on the fluent stream.

For example, the position of a car would be an example of a feature. The true position of the car at each time-point during its existence would be its fluent, and a particular sequence of observations of its position would be a fluent stream. There can be many fluent streams all approximating the same fluent.

5.2 Computational Units

A *computational unit* encapsulates a computation on one or more fluent streams. A computational unit takes a number of fluent streams as input and computes a new fluent stream as output. The encapsulated function can do anything, including calling external services.

Examples of computational units are filters, such as Kalman filters, and other sensor processing and fusion algorithms. Several other examples of computational units will be presented later.

6 Object Linkage Structures

One problem that has to be dealt with is recognizing that the object being tracked by the UAV platform is actually a car, that is, to anchoring the symbolic representation of a car with the stream of positions extracted by the image processing system. This is called

the anchoring problem [8]. Our approach is based on using temporal logic to describe the normative behavior of different types of entities and based on the behavior of an observed entity hypothesize its type. For example, in the traffic monitoring domain the object being tracked is assumed to be a physical object in the world, called a *world object*. Then, if the position of this world object is consistently on the road system then it is hypothesized to be a *road object*, i.e. an object moving within the road system. By further monitoring the behavior and other characteristics such as speed and size of the road object it could be hypothesized whether it is a car, a truck, a motorcycle, or another type of vehicle.

An object or an aspect of an object is represented by an *entity structure*. The term *entity* is used to indicate that it is more general than an object. An entity structure consists of a type, a name and a set of features representing the properties of the entity. The name is supposed to be unique and is used to identify the entity. All entity structures with the same type are assumed to have the same features. An *entity frame* is a data structure representing a snapshot of an entity. It consists of the type and name of an entity structure and a value for each of the features of the entity. An entity structure is implemented in DyKnow as a fluent stream where the values are entity frames. Each entity frame represents the state of an entity at a particular time-point and the fluent stream represents the evolution of the entity over time.

Each object is represented by an entity structure and relations between the entities are represented by *links*. The description of a class of links from entities of type A to entities of type B consists of three constraints, the *establish*, *reestablish*, and *maintain* constraints, and a computational unit for computing B entity structures from A entity structures.

The establish constraint describes when a new instance of type B should be created and linked to. For example, if the position of a world object is on the road for more than 30 seconds then a road object is created together with a link between them. A road object could contain more abstract and qualitative attributes such as which road segment it is on, which makes it possible to reason qualitatively about its position in the world relative to the road, other vehicles on the road, and building structures in the vicinity of the road. At this point, streams of data are being generated and computed for the features in the linked object structures at many levels of abstraction as the UAV tracks the road objects.

The reestablish constraint describes when two existing entities of the appropriate types which are not already linked should be linked. This is used when the tracking of a road object is lost and the tracker finds a new world object which may or may not be the same object as before. If the reestablish constraint is satisfied then it is hypothesized that the new world object is in fact the same road object as was previously tracked. Since links only represent hypotheses, they are always subject to becoming invalid given additional data, so the UAV continually has to verify the validity of the links. This is done by monitoring that the maintenance constraint is not violated.

A maintenance constraint could compare the behavior of the new entity, which is the combination of the two representations, with the normative behavior of this type of entity and, if available, the predicted behavior of the previous entity. In the road object example the condition is that the world object is continually on the road maybe with

shorter periods when it is outside. If this condition is violated then the link is removed and the road object is no longer updated since the hypothesis can not be maintained.

One purpose of the object linkage structures is to maintain an explicit representation of all the levels of abstraction used to derive the representation of an object. It is believed that this will make the anchoring problem easier since sensor data, such as images, does not have to be directly connected to a car representation but can be anchored and transformed in many small steps. Another benefit is that if the tracking is lost only the link between the world object and the road object is lost, if the road object is linked to a car object then this link can still persists and the car will be updated once the road object has been linked to a new world object. Another usage of the linkage structures could be to replace the world object which is lost with a simulated world object which simulates or predicts the development of the previous world object based on its history.

7 Scenario Recognition

In many applications it is crucial to describe and recognize complex events and scenarios. Chronicles [1] is one formalism used to represent complex occurrences of activities described in terms of temporally constrained events. In this context, an event is defined as a change in the value of a feature. For example, in the traffic monitoring application, a UAV might fly to an intersection and try to identify how many vehicles turn left, right or drive straight through a specific intersection. In another scenario, the UAV may be interested in identifying vehicles overtaking. Each of these complex activities can be defined in terms of one or more chronicles. In our project, we use the C.R.S. chronicle recognition system developed by France Telecom [9].

A chronicle is a description of a generic scenario whose instances should be recognized. A chronicle is represented by a set of events and a set of temporal constraints between these events with respect to a context [1]. An event represents a change in the value of a feature. The online recognition algorithm takes a stream of time-stamped event instances and finds all matching chronicle instances. To do this C.R.S. keeps track of all possible developments in a tractable and efficient manner by using simple temporal networks [2]. A chronicle instance is matched if all the events in the chronicle model are present in the stream and the time-stamps of the event instances satisfies the temporal constraints. Recognized instances of a chronicle can be used as events in another chronicle, thereby enabling recursive chronicles.

In order to use chronicle recognition to recognize event occurrences the event must be expressed in the chronicle formalism and a stream of primitive events must be available. The only requirement on the stream is that the primitive events arrive ordered by occurrence time. The reason is that all the information for a specific time-point has to be available before the temporal network can be updated with the new information. This means that whenever a new event arrives with the occurrence time t all partial chronicle instances are updated up to the time-point $t - 1$ and then the new information is added. If an event arrives out of order it will be ignored. The integration using DyKnow is done in two steps, integration of chronicles and integration of events.

The first step is when a chronicle is registered for recognition. To integrate a new chronicle DyKnow goes through each of the features in the chronicle and subscribes to

the corresponding fluent stream. The name of each feature is required to be a reference to a fluent stream containing discrete values. To make sure that the chronicle recognition engine is aware of all the changes in the fluent stream a policy is constructed which subscribes to all changes in the fluent stream and makes sure that the changes are ordered by valid time by use of a monotone order constraint. When subscriptions for all fluent streams are set up the recognition engine is ready to recognize chronicles.

The second step is when a sample arrives to the chronicle recognition engine. To integrate a sample it must be transformed into an event, i.e. a change in the feature represented by the fluent stream. To do this the recognition engine keeps track of the last value for each of the features and creates an event if the feature changed values compared to the stored value. The first value is a special case where the value changes from an unknown value to the new value. When the chronicle recognition engine is started events representing the initial state must be added. Since it is assumed that the events arrive in order the recognition engine updates the internal clock to the time-point before the valid time of the new sample. Thus the chronicle engine can prune all partial chronicles which are inconsistent with the current knowledge of what events have taken place. When a chronicle instance is recognized this is announced by an event.

7.1 Recognizing Overtakes

One instance of the traffic monitoring application involves the UAV observing a road segment and collecting information about the behavior of the vehicles passing by. The example used is recognizing overtakes, but the approach is general.

An overtake is considered to have occurred when two cars are driving on the same road and when one of the cars has caught up with and passed the other car. This is the same as saying that an overtake occurs if a car is first behind another car, and then later in front of the same car while both cars are on the same road. A requirement that the cars are beside each other at some time-point in between could be added to strengthen the definition. The overtake event can be formalized by the following overtake chronicle:

```
chronicle overtake[?car1, ?car2] {
 event(same_road[?car1, ?car2]:(?, true), t0)
 event(behind[?car1, ?car2]:(?, true), t1)
 event(behind[?car1, ?car2]:(true, false), t2)
 event(in_front[?car1, ?car2]:(false, true), t3)

 noevent(behind[?car1, ?car2]:(true, false), (t1, t2-1))
 noevent(behind[?car1, ?car2]:(false, true), (t2, t3-1))
 noevent(in_front[?car1, ?car2]:(false, true), (t1, t3-1))
 noevent(same_road[?car1, ?car2]:(true, false), (t0, t3-1))

 t0 < t2
 t1 < t2
 t1 < t3 }
```

An `event(A[?x]:(v1, v2), t)` statement says that the feature `A` of the object assigned to the variable `?x` changed values from `v1` to `v2` at time-point t. The value can be `?` which works as a wildcard and represents that the actual value is not important. Each `?` represents a new variable. A `noevent(A:(v1, v2), (t1, t2))`

statement says that an event, where the feature A changes values from v1 to v2, should not occur in the interval $[t_1, t_2]$. The temporal constraints at the end of the chronicle can be any metric constraint on pairs of time-points. The constraint t0 $<$ t2 could also be written t2-t0 in [1, oo]. The noevent statement is used to make sure that a value does not change within an interval. For example, to say that A should be true over the interval $[b, e]$ the following statements could be used:

```
event(A:(?, true), a)
noevent(A:(true, false), (b, e))
a <= b
```

The recognition has been tested both on simulated cars driving in a road system and on real data captured during flight tests. One example of the latter is shown in Fig. 5.

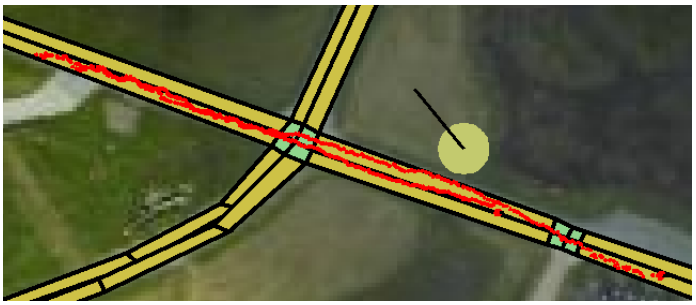

Fig. 5. An example overtake situation recorded during a test flight.

To recognize overtakes using the above chronicle a stream of qualitative spatial relations between pairs of cars, such as behind and beside, must be computed. This might sound like a very simple task, but does in fact require a number of steps. First, the set of cars that are actually being tracked must be extracted from the stream of car observations using the object linkage structures described in the previous section. Second, the set of pairs of active cars can be computed from the set of cars. Third, for each pair of car names a stream of synchronized pairs of car entity structures have to be created. Since they are synchronized both car entities in the pair are valid at the same time-point, which is required to compute the relation between the two cars. Fourth, from this stream of car pairs the qualitative spatial relations must be computed. Finally, this stream of car relations can be used to detect overtakes and other driving patterns using the chronicle recognition engine. All these functions are implemented as computational units. The complete setup is shown in Fig. 6. A more detailed description follows.

To extract the active cars a computational unit is created which keeps track of the names of all cars that have been updated the last minute. This means that if no observation of a car has been made in more than 60 seconds it will be removed from the set of active cars. For example, assuming the stream of car observations looks like: $\langle\langle \mathsf{car1}, \ldots\rangle, \ldots \langle \mathsf{car2}, \ldots\rangle, \ldots \langle \mathsf{car3}, \ldots\rangle, \ldots\rangle$ then the stream of sets of active cars would be $\langle\{\mathsf{car1}\}, \{\mathsf{car1}, \mathsf{car2}\}, \{\mathsf{car1}, \mathsf{car2}, \mathsf{car3}\}\rangle$. Since the qualitative spatial relations are computed on pairs of cars where the cars are different and the order of the two

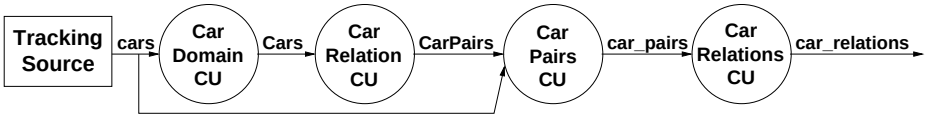

Fig. 6. The DyKnow setup used in the overtake monitoring application.

cars are not important the computational unit that extracts pairs of car names only computes one pair for each combination of distinct car names. To continue the example, the stream of set of pairs would be $\langle\{\}, \{\langle \mathsf{car1}, \mathsf{car2}\rangle\}, \{\langle \mathsf{car1}, \mathsf{car2}\rangle, \langle \mathsf{car1}, \mathsf{car3}\rangle, \langle \mathsf{car2}, \mathsf{car3}\rangle\}\rangle$. The stream of sets of pairs is called CarPairs and is updated when a car is added or removed from the set of active car names, called Cars. This stream of car name pairs is then used as input together with the stream of car entities to a state extraction computational unit which for each pair synchronizes the streams of car entities as shown in Fig. 7. The figure shows that for each of pair of car names in the CarPairs stream a new state synchronization unit is created where only updates to those car entities with the specified names are sent. Each time one of the state synchronization units derives a new state it is added to the fluent stream defined by the computational unit.

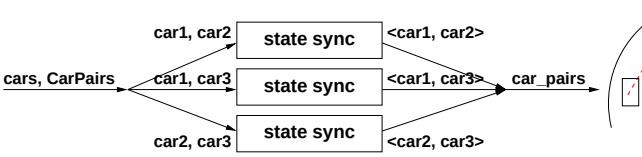

Fig. 7. The synchronization of car pairs.

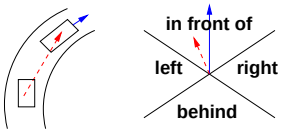

Fig. 8. The qualitative spatial relations used.

Finally the car pair entities are used as input in the car relation computational unit which computes the qualitative spatial relation between the two cars by comparing the forward direction of `car1` with the direction from `car2` to `car1`, as shown in Fig. 8. The forward direction of the car is assumed to be either along the current road segment or against it. To compute which, the current driving direction of the car is compared to the forward direction of the road segment. Since the estimated driving direction of the car is very noisy this provides a much better estimate.

Each time the car relations are computed the chronicle recognition engine is notified, since it subscribes to that fluent stream, and the chronicles instance are updated with the new information. In the next section some experimental results are presented.

8 Experimental Results

The traffic monitoring application has been test both in simulation and on images collected from flight tests. The only difference between the two cases is who creates the world objects. One of the trajectories collected at a flight test is shown in Fig. 5.

The robustness to noise in the position estimation was tested in simulation by adding random error to the true position of the cars. The error has a uniform distribution with a known maximum value and is added independently to the x and y coordinates. If the maximum error is 2 meters, then the observed position is within a 2x2 meter square centered on the true position. Two variables were varied, the speed of the car and the sample period of the position. For each combination 10 simulations were run were a car overtook another. If the overtake was recognized the run was considered successful. The results are shown in Table 1.

Speed	**Sample period**	**0m error**	**1m error**	**2m error**	**3m error**	**4m error**	**5m error**	**7m error**
15 m/s	200 ms	100%	100%	70%	60%	50%	40%	20%
20 m/s	200 ms	100%	90%	60%	70%	50%	50%	20%
25 m/s	200 ms	100%	100%	100%	50%	40%	10%	0%
15 m/s	100 ms	-	-	60%	90%	60%	90%	0%
20 m/s	100 ms	-	-	90%	90%	90%	80%	20%
25 m/s	100 ms	-	-	90%	80%	70%	80%	0%

Table 1. The results from varying the speed of the cars and the sample period.

The conclusions from these experiments are that the speed of the car is not significant but the sample period is. The more noise in the position the more samples are needed in order to detect the overtake. The reason for the very poor performance when the error is 7 meters is due to the fact that most observations are outside the road and those observations are filtered out. Since the estimated error from the image processing is 4-5 meters the system should reliably detect overtakes.

9 Related Work

There is a great amount of related work which is relevant for each of the components, but the focus will be on integrated systems. There are a number of systems for monitoring traffic by interpreting video sequences, for example [10–15]. Of these, almost all work on sequences collected by static surveillance cameras. The exception is [12] which analyses sequences collected by a Predator UAV. Of these none combine the input from both color and thermal images. Another major difference is how the scenarios are described and recognized. The approaches used include fuzzy metric-temporal logic [11], state transition networks [13], belief networks [15], and Petri-nets [10, 14] neither of which have the same expressivety when it comes to temporal constraints as the chronicle recognition approach. Another difference is the ad-hoc nature of how the components of the system and the data flows are connected. In our solution the basis is a declarative description of the properties of the different data streams which is then implemented by the DyKnow knowledge processing middleware. This makes it very easy to change the application to e.g. add new features or to change the parameters. The declarative specification could also be used to reason about the system itself.

10 Conclusions

A traffic monitoring application which is an instance of a general approach to creating high-level situation awareness applications is presented. The implemented system takes as input sequences of color and thermal images used to construct and maintain qualitative object structures and recognize the traffic behavior of the tracked vehicles in realtime. The system is tested both in simulation and on data collected during test flights. It is believed that this type of system where streams of data are generated at many levels of abstraction using both top-down and bottom-up reasoning handles many of the issues related to closing the sense-reasoning gap. A reason is that the information derived at each level is available for inspection and use. This means that the subsystems have access to the appropriate abstraction while it is being continually updated with new information and used to derived even more abstract structures. High-level information, such as the type of vehicle, can then be used to constrain and refine the processing of lower level information. The result is a very powerful and flexible system capable of achieving and maintaining high-level situation awareness.

References

1. Ghallab, M.: On chronicles: Representation, on-line recognition and learning. In: Proceedings of the Fifth Intl Conf on Principles of Knowledge Representation and Reasoning. (1996)
2. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. AIJ **49** (1991)
3. Heintz, F., Doherty, P.: DyKnow: An approach to middleware for knowledge processing. Journal of Intelligent and Fuzzy Systems **15**(1) (2004) 3–13
4. Heintz, F., Doherty, P.: A knowledge processing middleware framework and its relation to the JDL data fusion model. Journal of Intelligent and Fuzzy Systems **17**(4) (2006) 335–351
5. Doherty, P., Haslum, P., Heintz, F., Merz, T., Nyblom, P., Persson, T., Wingman, B.: A distributed architecture for autonomous unmanned aerial vehicle experimentation. In: Proceedings of the 7th Intl Symposium on Distributed Autonomous Robotic Systems. (2004)
6. Bouguet, J.: Matlab camera calibration toolbox (2000)
7. Wengert, C., Reeff, M., Cattin, P.C., Székely, G.: Fully automatic endoscope calibration for intraoperative use. In: Bildverarbeitung für die Medizin, Springer-Verlag (2006) 419–23
8. Coradeschi, S., Saffiotti, A.: An introduction to the anchoring problem. Robotics and Autonomous Systems **43**(2-3) (2003) 85–96
9. Telecom, F.: C.R.S. website (2007) Retrieved March 22, 2007, from http://crs.elibel.tm.fr/en.
10. Ghanem, N., DeMenthon, D., Doermann, D., Davis, L.: Representation and recognition of events in surveillance video using petri nets. In: Proceedings of Conference on Computer Vision and Pattern Recognition Workshops CVPRW. (2004)
11. Nagel, H., Gerber, R., Schreiber, H.: Deriving textual descriptions of road traffic queues from video sequences. In: Proc. 15th European Conference on Artificial Intelligence. (2002)
12. Medioni, G., Cohen, I., Bremond, F., Hongeng, S., Nevatia, R.: Event detection and analysis from video streams. IEEE Trans. Pattern Anal. Mach. Intell. **23**(8) (2001) 873–889
13. Fernyhough, J., Cohn, A., Hogg, D.: Building qualitative event models automatically from visual input. In: Proceedings of the International Conference on Computer Vision. (1998)
14. Chaudron, L., Cossart, C., Maille, N., Tessier, C.: A purely symbolic model for dynamic scene interpretation. International Journal on Artificial Intelligence Tools **6**(4) (1997)
15. Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S., Weber, J.: Automatic symbolic traffic scene analysis using belief networks. In: AAAI'94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 2). (1994) 966–972