From Images to Traffic Behavior - A UAV Tracking and Monitoring Application

Fredrik Heintz, Piotr Rudol and Patrick Doherty
Department of Computer and Information Science
Linköpings universitet, Sweden
{frehe, pioru, patdo}@ida.liu.se

Abstract— **An implemented system for achieving high level situation awareness about traffic situations in an urban area is described. It takes as input sequences of color and thermal images which are used to construct and maintain qualitative object structures and to recognize the traffic behavior of the tracked vehicles in real time. The system is tested both in simulation and on data collected during test flights. To facilitate the signal to symbol transformation and the easy integration of the streams of data from the sensors with the GIS and the chronicle recognition system, DyKnow, a stream-based knowledge processing middleware, is used. It handles the processing of streams, including the temporal aspects of merging and synchronizing streams, and provides suitable abstractions to allow high level reasoning and narrow the sense reasoning gap.**

Keywords: Level 2/3 fusion, tracking, knowledge representation, sensor and symbol integration.

I. Introduction

Imagine a human operator trying to maintain situational awareness about a traffic situation in an urban area using UAVs. One approach would be for the 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. This paper describes such a traffic monitoring application where cars are tracked by a UAV platform and the stream of observations is fused with a model of the road system in order to draw conclusions about the behavior of these cars in the environment. The inputs are 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 real time as they develop using a chronicle recognition system.

To facilitate the easy integration of the streams of data from the sensors with the GIS and the chronicle recognition system our stream-based knowledge processing middleware DyKnow is used [1], [2]. It takes care of the temporal aspects of merging and synchronizing streams and provides suitable abstractions to describe and implement these types of applications and close the sense reasoning gap. The approach is then extended with a method to dynamically estimate and continually monitor the type of objects being tracked.

The approach is based on using temporal logic formulas to describe the normative behavior of different objects and to use formula progression to evaluate when formulas are satisfied or violated. Based on the results, the objects being tracked can be hypothesized to be of different types. The application is implemented and has been tested both in simulation and on data collected during test flights.

A. Structure of the Paper

The paper starts with Section II and Section III describing the UAV platform and the image processing system which is used to track and geolocate objects in the world. Section IV describes the DyKnow knowledge processing middleware which is used to integrate streams of data from the sensors with the GIS and the chronicle recognition system. Section V and Section VI describe the intersection and road segment monitoring scenarios which have been implemented. Section VII describes an extension to the representation of objects which makes it possible to make hypotheses about the type of object being tracked. The paper concludes with related work in Section IX and a summary in Section X.

II. The Hardware Platform

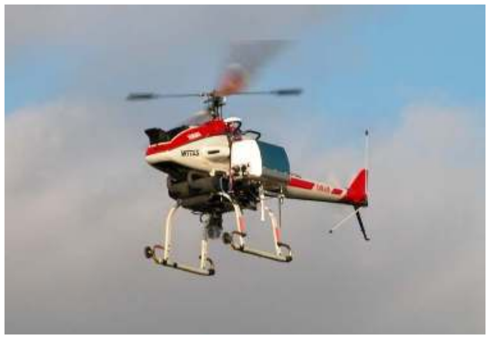

Fig. 1. The WITAS RMAX autonomous helicopter

The WITAS UAV platform [3] is a slightly modified Yamaha RMAX helicopter (Fig. 1), equipped with a more efficient power generator and a higher landing gear. Its total length is 3.6 m (including the main rotor) and 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.

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 the sensor fusion required for basic helicopter control and for sequentializing control modes (i.e. takeoff, hover, 3D path following, landing, vehicle following etc.). The PFC interfaces with the RMAX helicopter through the 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 (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. 2). Both cameras deliver analogue PAL signals with the frame size 768x576 pixels at 25Hz rate.

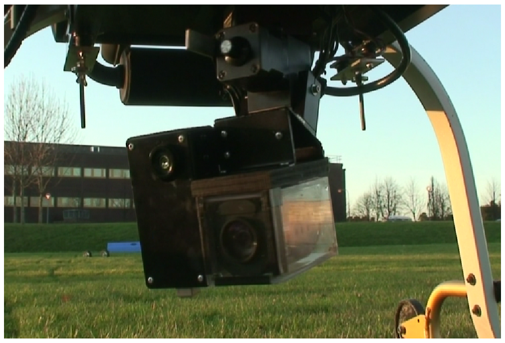

Fig. 2. Onboard camera system consisting of color and thermal cameras mounted on a pan-tilt unit.

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 for off-line processing.

III. 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 method of initialization 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 based on 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 the color and thermal video streams. Tracking in the thermal image is achieved by finding the extreme value (warmest or coldest spots) within a small (5 percent of the image size) window 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 a 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, as well as more sophisticated interplay between both trackers, 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 [4]. The thermal camera has been calibrated using a custom calibration pattern and a different calibration method [5] because it was infeasible to obtain sharp images of the standard chessboard calibration pattern. The extrinsic parameters of the cameras were found by minimizing the error between 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 stereovision algorithms since objects generally appear different in color and infrared images. Because of this fact, the distance to an object whose projection lies in a given pixel must be determined. Given the camera parameters, helicopter pose, and the ground elevation model the distance to an object can be calculated. It is the distance from the camera center to the intersection between the ground model 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 (i.e. ground elevation model simplified to a plane) is negligible. Finally, calculating pixel correspondences between the two cameras can be achieved by performing pixel geolocalisation using intrinsic and extrinsic parameters of one of the cameras followed by applying a inverse procedure (i.e. projection of geographical location) using the other camera parameters.

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. 3A) the UAV is stationary at altitudes of 50 and 60 meters and observes two vehicles as they drive on a nearby road. In the other kind (Fig. 3B) 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 from 25 to 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. For example results of car tracking see Fig. 5 and Fig. 8.

IV. DyKnow

To facilitate the development of the traffic scenario recognition applications a knowledge processing middleware framework called DyKnow is used [1], [2]. 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, the 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.

A. 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*. 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 constraints 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.

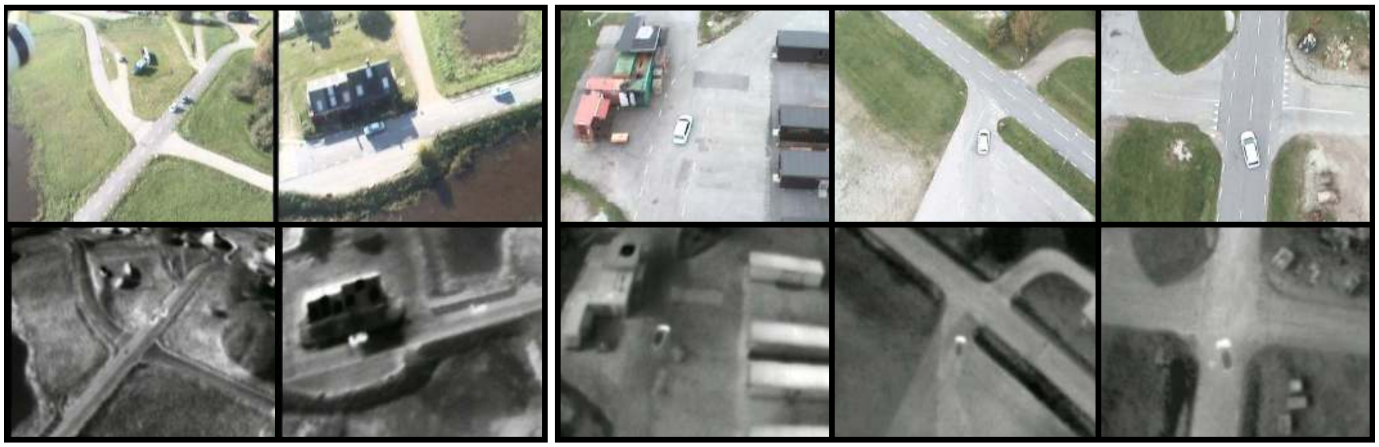

Fig. 3. A. Two frames from video sequence with the UAV hovering close to a road segment observing two cars performing overtaking maneuver. B. Three frames from video sequence with the UAV following a driving car passing road crossings. Top row contains color images and bottom row contains corresponding thermal images.

B. 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 examples of computational units will be presented later.

C. Entity Structures

An object is represented by an *entity structure* consisting of a type, a name and a set of attributes 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 attributes. 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 attributes 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.

D. Chronicle Recognition

In many applications it is crucial to describe and recognize complex events and scenarios. In this work, the chronicle formalism [6] is 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 a 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 vehicle 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 [7].

A chronicle is a description of a generic scenario whose instances we would like to recognize. The chronicle is represented as a set of events and a set of temporal constraints between these events with respect to a context [6]. 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 temporal constraint networks [8]. 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 suitable stream of primitive events must be derived. The only requirement on the stream is that the primitive events arrive ordered by valid 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. So, whenever a new sample arrives with the valid time t the network is propagated to the time-point $t-1$ and then the new information is added. If a sample arrives out of order it will be ignored. The integration 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 attributes in the chronicle and subscribes to the corresponding fluent stream. Each attribute is considered a reference to a fluent stream containing discrete values. To make sure that the chronicle recognition engine gets all the intended 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 the fluent streams are setup the recognition engine is ready to start recognizing chronicle instances.

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 an attribute. To do this the recognition engine keeps track of the last value for each of the attributes and creates an event if the attribute changed values. The first value is a special case where the value changes from an unknown value to the new value. 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. In this manner the chronicle engine can update the constraints and prune all partial chronicles which can no longer be recognized.

V. Intersection monitoring

The first part of the traffic monitoring application is to monitor activities in an intersection. In this case the UAV stays close by an intersection and monitors the cars going through. Each car should be tracked and it should be recorded how it travelled in the intersection to create a stream of observations such as car *c* came from road *a* to crossing *x* and turned left onto road *b*. The cars are tracked by the vision system on the UAV and the information about the road system comes from a GIS. This section describes how this information is fused and how it is used to recognize the behavior of the cars in real-time as the situation develops.

The road system is represented in the GIS as a number of *areas* which cover the road system. Each area is classified as either being a crossing or a road (in Fig. 4 the green areas are the crossings and the yellow are roads). There are different areas representing the different lanes of a road. To represent the road connecting two crossings an abstraction called a *link* is introduced. All road areas between two crossings are part of the link. The separation of areas and links are made in order to be able to reason both about the geometry and other low level properties of the roads and higher level road network properties. The geometry is for example needed in order to simulate cars driving on roads and to find the road segment given a position. The network structure is for example needed when reasoning about possible routes to a destination.

To represent the possible turns that can be made from a link an entity structure called Link is used. The entity structure has four attributes, left, right, straight, and uturn. Since it is possible to turn e.g. left in many different ways it is necessary to represent sets of triples $\langle \texttt{link1}, \texttt{crossing}, \texttt{link2} \rangle$ where each triple represents that a car going from `link1` through `crossing` to `link2` made a left turn. The link entities are made available in a fluent stream called links.

The tracking functionality of the UAV platform (see Section III) provides a stream of world coordinates which represents the best estimation of the current position of the tracked object. Based on this position it is possible to derive further information about the object which for now is assumed to be a car. Cars are represented by the following entity structure:

```
Car {
  Position pos
  string link
  string crossing
  bool drive_along_road
}
```

The attribute pos is the position provided by the tracker. The link attribute is the name of the link the car is on according to the GIS. If the position is not on a link then the value is “no_link”. The crossing attribute is similar to the link attribute but has a value if the area is a crossing, otherwise the value is “no_crossing”. This means that the car is not on the road system if the link attribute is “no_link” and the crossing attribute is “no_crossing” at the same time. The drive along road attribute will be explained in the next section. The car entities are made available in a fluent stream called cars.

The information about the world is thus provided as two fluent streams, one containing information about links and one about cars. In order to detect the intersection behavior these streams must be further analyzed. In this application chronicle recognition as described in Section IV-D is used to describe and recognize behaviors. Part of the chronicle for detecting left turns is shown below.

```
chronicle turn_left_in_crossing[?c,?l1,?x,?l2]
{
 occurs(1,1,cars.link[?c]:(?l1,no_link),(t2,t3))
 occurs(1,1,cars.crossing[?c]:
           (no_crossing,?x),(t2,t3))

 event(cars.crossing[?c]:(?x,no_crossing),t4)
 event(cars.link[?c]:(no_link,?l2),t5)

 event(links.left[?l1,?x,?l2]:(?,true),t1)

 t1 < t2
 t3-t2 in [-1000, 1000] //timeunit milliseconds
 t4-t3 in [0, 10000]
 t5-t4 in [-1000, 1000]
}
```

The chronicle says that a car makes a left turn if it is on link `?l1`, enters crossing `?x`, leaves on link `?l2`, and the triple $\langle$ `?l1, ?x, ?l2` $\rangle$ constitutes a left turn according to the GIS. The name `cars.link[car1]` refers to the link attribute of a Car entity with the name car1 found in the cars fluent stream. The temporal constraints at the end assert that the car should be observed to be in the crossing within 1 second before or after it has been observed not to be on any link and that the turn should not take more than 10 seconds to make. The chronicle also contains a number of noevent statements which are not shown to make sure that no changes happen between the entering of the crossing and the leaving of the crossing.

Since the link attribute is quite coarse it is possible to manage the uncertainty in the position of the car which causes it to be on the right link but not on the correct lane. It is also possible to define a chronicle to detect turns which are made from the correct lane, but this will often fail due to noise. For example, see Fig. 5, where the trajectory of a tracked car is shown as it drives through an intersection.

The chronicle will fail if no observation is made of the car in the crossing, which can happen when the speed of the car is too high or the time between observations is too long. To predict that a car actually turned in the crossing even though it was only observed on a link before the crossing and on a link after the crossing the following chronicle is used:

```
chronicle
 turn_left_in_crossing_predicted[?c,?l1,?x,?l2]
{
 event(links.left[?l1, ?x, ?l2]:(?, true), t1)
 event(cars.link[?c]:(?, ?l1), t2)
 event(cars.link[?c]:(?l1, ?l2), t3)
 t1 < t2
 t2 < t3
}
```

This chronicle is much simpler than the one before since it only checks that the car passed from one link to another and that according to the GIS this transition indicates that the car must have passed a crossing and actually made a left turn. This is an example of where qualitative information about the road system can be used to deduce that the car must have passed through the crossing even though this was never observed.

There is still one issue which is illustrated by Fig. 4 where the noise in the position of the tracked car makes it look like it enters the crossing, leaves the crossing, and then comes back. This type of oscillating attributes are very common in the transition between two values of an attribute. A solution is to introduce a filter which only changes the value of an attribute if it has been stable for a fixed amount of time, in our case 500 milliseconds. Since the attribute which was filtered was the link attribute of the Car entity it is reasonable to say that the value must be stable for half a second, since it is not expected to change very often (a link is usually several hundred meters even though shorter links may exist in urban areas). One possible problem is if the car is not in a crossing for more than 500 milliseconds, but this case will be detected by the predicted turn chronicle so the turn will be detected in any case.

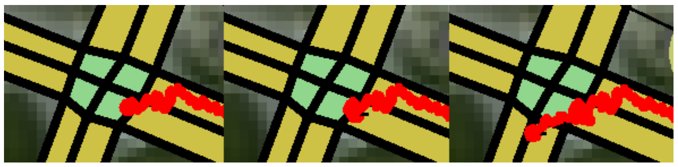

Fig. 4. An example where noise makes it look like a car enters a crossing twice. Each red dot is an observed car position.

Using the setup described and the chronicles described above it is possible to detect all the turns made by one or more cars driving in an urban area using either simulated cars or cars tracked by our UAV platform during test flights. One particular trajectory from a test flight where two left turns are recognized is shown in Fig. 5. In the simulation the cars passed through many different crossings and turned in all possible directions. To simulate the coordinates produced by the tracker white noise was added to the simulated position of the cars.

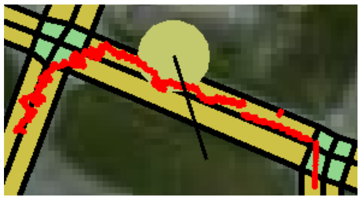

Fig. 5. An example intersection situation recorded during a test flight.

VI. Road Segment Monitoring

The second monitoring task involves the UAV observing a road segment and collecting information about the behavior of the vehicles passing by. In this paper the focus is on recognizing overtakes, but this is just an example, other behaviors could be detected in the same way. To recognize overtakes a stream of qualitative spatial relations between pairs of cars, such as behind and beside, are computed and used as input by the chronicle recognition system. 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 and based on this the set of pairs of active cars can be computed. Second, 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 two cars. Third, 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.

To extract the active cars a computational unit is created which keeps track of all car names which 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, assume the stream of car observations look 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 relations that are computed are symmetric and irreflexive 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 sets 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 to a state extraction computational unit which for each pair synchronizes the corresponding streams of car entities as shown in Fig. 6.

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`.

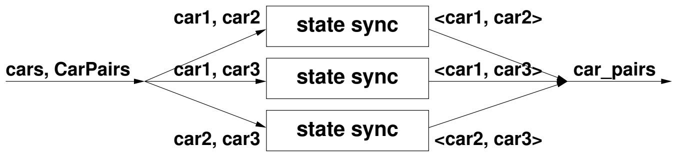

Fig. 6. The synchronization of car pairs.

The forward direction of the car is assumed to be either along the current road segment or against it. To compute which, the current direction of the car as estimated by derivating the position of the car, is compared to the forward direction of the road segment. Since the estimated velocity of the car is very noisy this provides a much better estimate.

The chronicle that is used to detect overtakes is quite simple and is therefore left out, it detects that car1 is first behind car2 and then it is in front of car2. A requirement that they are beside each other could be added to strengthen the definition.

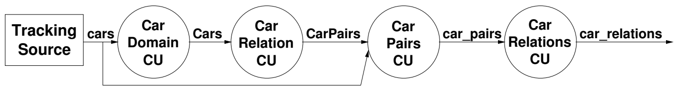

Fig. 7. The DyKnow setup used in the overtake monitoring application.

The complete setup is shown in Fig. 7. The recognition has been tested on both simulated cars driving in a road system and on real data captured during flight tests. One example of the latter is shown in Fig. 8.

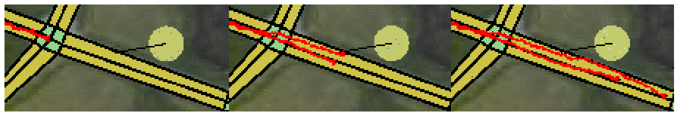

Fig. 8. An example overtake situation recorded during a test flight.

VII. 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 anchor the symbolic representation of the car with the stream of sensor data collected by the UAV. This is called the anchoring problem [9]. 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 real object in the world, represented by a *world object*. Then, if the position of this world object is consistently on the road system then it is assumed that it is actually a *road object*, i.e. an object moving along 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, truck, motorcycle, or other type of vehicle. Here it is assumed that background knowledge about vehicle types exists and can be put to use in determining vehicle type.

Each object is represented by an entity structure and relations between the entities are represented by *links*. A link 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 attributes 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 a 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 with shorter periods off the road. 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. This makes the anchoring problem easier since senor data, such as images, do 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 persist 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 a lost world object with a simulated world object which takes the previous world object and predicts the development of the object based on its history.

VIII. Experimental Results

The traffic monitoring application has been tested both in simulation and on images collected during flight tests, an

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

TABLE I

THE RESULTS WHEN VARYING THE CAR SPEED AND THE SAMPLE PERIOD.

example of which was shown in Fig. 8. The only difference between the two cases is who creates the world objects.

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 e and is added independently to the x and y coordinates, i.e. the observed position is within a exe 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 where a car overtook another. If the overtake was recognized the run was considered successful. The results are shown in Table I.

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. Since the estimated error from the image processing is at most 4-5 meters the system should reliably detect overtakes when using a 100ms sample period.

IX. Related Work

There is a great amount of related work which is relevant for each of the components, but in the spirit of the paper 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 operate 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], none of which have the same expressivity when it comes to temporal constraints as the chronicle recognition approach we use. Another difference is the ad-hoc nature of how the components of the system and the data flow 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 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 and even modify it at run-time.

X. Summary

An instance of a general approach to creating high level situation awareness applications is presented. The system implemented 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 real time. 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 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.

a) Acknowledgements: This work is supported in part by the National Aeronautics Research Program NFFP04 S4203 and the Strategic Research Center MOVIII, funded by the Swedish Foundation for Strategic Research, SSF.

References

[1] F. Heintz and P. Doherty, "DyKnow: An approach to middleware for knowledge processing," *Journal of Intelligent and Fuzzy Systems*, vol. 15, no. 1, pp. 3–13, nov 2004.
[2] ——, "A knowledge processing middleware framework and its relation to the JDL data fusion model," *Journal of Intelligent and Fuzzy Systems*, vol. 17, no. 4, pp. 335–351, 2006.
[3] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman, "A distributed architecture for autonomous unmanned aerial vehicle experimentation," in *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.
[4] J. Bouguet, "Matlab camera calibration toolbox," 2000.
[5] C. Wengert, M. Reeff, P. C. Cattin, and G. Székely, "Fully automatic endoscope calibration for intraoperative use," in *Bildverarbeitung für die Medizin*. Springer-Verlag, March 2006, pp. 419–23, http://www.vision.ee.ethz.ch/∼cwengert/calibration_toolbox.php.
[6] M. Ghallab, "On chronicles: Representation, on-line recognition and learning," in *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 1996.
[7] F. Telecom, "C.R.S. website," 2007, retrieved May 7, 2007, from http://crs.elibel.tm.fr.
[8] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.
[9] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem," *Robotics and Autonomous Systems*, vol. 43, no. 2-3, 2003.
[10] N. Ghanem, D. DeMenthon, D. Doermann, and L. Davis, "Representation and recognition of events in surveillance video using petri nets," in *Proceedings of Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'04)*, vol. 7, 2004.
[11] H. Nagel, R. Gerber, and H. Schreiber, "Deriving textual descriptions of road traffic queues from video sequences," in *Proc. 15th European Conference on Artificial Intelligence (ECAI-2002)*, 2002.
[12] G. Medioni, I. Cohen, F. Bremond, S. Hongeng, and R. Nevatia, "Event detection and analysis from video streams," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 8, pp. 873–889, 2001.
[13] J. Fernyhough, A. Cohn, and D. Hogg, "Building qualitative event models automatically from visual input," in *Proceedings of the International Conference on Computer Vision ICCV98*, IEEE. Narosa, 1998.
[14] L. Chaudron, C. Cossart, N. Maille, and C. Tessier, "A purely symbolic model for dynamic scene interpretation," *International Journal on Artificial Intelligence Tools*, vol. 6, no. 4, pp. 635–664, 1997.
[15] T. Huang, D. Koller, J. Malik, G. Ogasawara, B. Rao, S. Russell, and J. Weber, "Automatic symbolic traffic scene analysis using belief networks," in *Proceedings of the 12th National Conference on Artificial intelligence*, 1994.