A FLEXIBLE RUNTIME SYSTEM FOR IMAGE PROCESSING IN A DISTRIBUTED COMPUTATIONAL ENVIRONMENT FOR AN UNMANNED AERIAL VEHICLE

KLAS NORDBERG, PER-ERIK FORSSÉN, JOHAN WIKLUND,
PATRICK DOHERTY[⋆], PER ANDERSSON[†]

Department of Electrical Engineering, Linköping University, Sweden
[⋆]*Department of Computer Science, Linköping University, Sweden*
[†] *Department of Computer Science, Lund University, Sweden*

A runtime system for implementation of image processing operations is presented. It is designed for working in a flexible and distributed environment related to the software architecture of a newly developed UAV system. The software architecture is characterized at a coarse scale as a three level system, with a deliberative layer at the top, a reactive layer in the middle, and a processing layer at the bottom. At a finer scale each of the three levels is decomposed into sets of modules which communicate using CORBA, allowing system development and deployment on the UAV to be made in a highly flexible way. The image processing takes place in a dedicated module located in the process layer, and is the main focus of the paper. This module has been designed as a runtime system for data flow graphs, allowing various processing operations to be created on-line and on demand by the higher levels of the system. The runtime system is implemented in Java, which allows development and deployment to be made on a range of hardware/software configurations. Optimizations for particular hardware platforms have been made using Java's native interface.

1. Introduction

The WITAS[a] Unmanned Aerial Vehicle Project[b] [2] is an ambitious, long-term basic research project whose main objectives are the development of an integrated hardware/software VTOL platform for fully-autonomous missions and its deployment in applications such as traffic monitoring and surveillance, emergency services assistance, photogrammetry, and surveying. Basic and applied research in the project includes a wide range of topics such as the following: development of a deliberative/reactive software architecture, a helicopter control system with flight modes for stable hovering, trajectory following and tracking, trajectory-

[a] WITAS (pronounced *vee-tas*) is an acronym for the Wallenberg Information Technology and Autonomous Systems Laboratory at Linköping University, Sweden.
[b] This work and the project is funded by a grant from the Wallenberg Foundation.

and task-based planning systems, a chronicle recognition system for identifying complex vehicular patterns on the ground, geographical information and knowledge databases for on-board use, multi-modal interfaces (including dialogue) for ground operator/UAV communication, and an on-board image processing system (IPM) which will be the focus of this paper. Before describing the IPM, a brief overview of both the hardware and software platforms will be provided to set the context for the image processing system.

The VTOL platform used in the project is a slightly modified Yamaha RMAX helicopter, shown in figure 1. It is approximately 2×1 meters, and equipped with various sensors including INS, DGPS, and a color video camera mounted on a stabilized gimbal with a pan/tilt interface which attenuates vibrations.

In order to deal with the complexity of a deliberative/reactive (D/R) software architecture with as much functionality as we require for our applications and to insure clean and flexible interfacing to the hardware platform, we have chosen to design and implement a loosely-coupled distributed architecture using Real-Time CORBA [3] [c]. In short, CORBA (Common Object Resource Broker Architecture) is middleware that establishes client/server relationships between objects or components. Objects can make requests to and receive replies from other objects located locally in the same process, in different processes, or in different processors on the same or separate machines.

Many of the functionalities which are part of the architecture can be viewed as clients or servers where the communication infra-structure is provided by an ORB (Object Resource Broker) and other services such as real-time event channels. This architectural choice provides us with an ideal development environment and versatile run-time system with built-in scalability, modularity, software relocatability on various hardware configurations, performance (real-time event channels and schedulers), and support for plug-and-play software modules. Figure 2 depicts a (incomplete) high-level schematic of the WITAS software architecture where each of the components may be viewed as a CORBA server/client providing or requesting services from each other and receiving data and events through both real-time and standard event channels. Some data flow and control links pertaining to the image processing capabilities are also depicted. Each of these functionalities have been implemented and are being used and developed in our applications.

Two different hardware configurations used for executing the deliberative/reactive components and image-processing system are being developed and experimented with in the lab. The first alternative uses two additional PC104*s* with

[c] TAO (The Ace Orb) is an open source implementation of CORBA 2.3.

Linux, one for the D/R components and the other for the image-processing components. The second alternative uses an additional embedded Pentium for the D/R components and a G4 PowerPC processor for the image processing components. Due to the flexibility of the CORBA-based infrastructure, it is relatively easy to move the various components onto different processors during development.

In the rest of the paper, focus will be placed on the image processing module and its integration with deliberative and reactive components in the software architecture. In section 2 we begin by describing particular scenarios in some of the application areas and the requirements they place on an on-board image-processing module (IPM) and its integration with the rest of the system. In section 3, we describe the image processing module in some detail. In section 4, we consider some specific operations involving the IPM. We then conclude with a summary and current status of the work.

2. Some Scenarios and Requirements

In the project, we have focused initially on traffic surveillance and emergency services assistance where sensing and understanding traffic situations is an important functionality. A typical scenario would be to find, identify and track a vehicle based on information about its signature (color, size, type, etc), when it was last seen and where. This type of scenario involves signal-to-symbol conversions from video streams with real-time constraints and continuous fusion of this information with other qualitative knowledge about the scenario. Such knowledge may consist of normative behavior of vehicles stored in the knowledge repository together with geographic and road data stored in the geographic data repository. It could also be acquired during the mission via communication with a ground operator.

It is important to emphasize that tight integration is required between the low-level image processing capability and the deliberative or reasoning capabilities of the UAV which use qualitative models. In addition, there are generally combinations of hard and soft real-time constraints involved in any of these scenarios where dynamic behavior is being tracked or observed. This implies the need for runtime modification of the image processing algorithms being used based on the current context and task at hand and the ability of the software architecture to support the monitoring of the success of the task and the quality of the sensory data being generated while achieving the task.

Due to the constraints associated with scenarios of the type described, an important operational requirement of the IPM is that it is highly flexible and reconfigurable. The idea is that the UAV should be able to switch between different modes of operations, where each mode may require a different configuration of

the IPM. For example, in one mode the UAV may hover and observe a particular road section. Triggered by some event, e.g., it observes a particular vehicle, the UAV switches to a tracking mode where the observed vehicle is being followed both by moving the helicopter and the camera. Additional flexibility is required in terms of using different implementations of the same type of operation for different purposes. For example, a relatively simple and fast method may be used for detecting moving ground objects. However, to estimate their true velocity, a more complex and time consuming operation has to be used if this feature is vital in achieving a particular task.

Examples of operations used in the current applications are "find vehicle" and "track vehicle". Finding a vehicle can be done in many different ways, using simple, fast and unreliable methods such as detecting colored blobs on roads, or complex, computationally demanding and robust methods based on, e.g., estimation of motion and shape. Also the tracking can be done using more or less sophisticated methods. Depending on the time available for the processing and the requirements for robustness, the deliberative component in the architecture can choose different algorithms dynamically during run-time or actually modify existing algorithms by using different image-processing operators in existing algorithms.

3. Image Processing Module

It was argued that it was necessary to develop an image processing system which could be accessed during runtime and have its services modified dynamically. In this case the IP modules services are IP algorithms used for achieving tasks specified at the mission level. Internally, these IP algorithms are represented using a data flow graph based model. Nodes in the graphs represent operations and the arcs represent data on which the operations are performed. In general, the graphs may be cyclic, e.g., feed-back loops are required for certain types of operations.

We call this computational model the Image Processing Data Flow Graph model (IP-DFG model) [1]. An IP-DFG is based on a *hierarchy* of boolean DFGs and new rules for token consumption. This variant of DFGs allows a high-level specification of complex image processing algorithms which can include iteration or tail-recursion and is suitable for the type of runtime modification required in our architecture. In addition, one can automatically generate information about timing constraints between nodes and memory requirements for internal data buffers used by operations in the graph. This form of analysis is very useful for optimization purposes and distributing execution of nodes onto different processors if required.

Conceptually, the Image Processing Module (IPM) consists of two parts,

- IPAPI - An Image Processing Application Program Interface – This is

the declarative interface that other components in the D/R architecture can use to create, manipulate, configure and execute various image processing algorithms.
- IPAPI-Runtime - This is the runtime component that manages the configuration and execution of image processing algorithms, dynamically allocates memory for buffers needed during execution, interfaces to the image controller and other components in the architecture, and manages an existing library of pre-defined image processing algorithms.

IPAPI-Runtime is implemented in the Java programming language. The use of Java offers a number of advantages; rapid prototyping, support on a number of different hardware/software configurations, access to an ever growing number of APIs for various purposes, e.g., both CORBA support and a powerful toolkit for building graphical applications is available. Thanks to the recent JIT technology for Java, it is also reasonable to implement the actual data processing in Java, i.e., the execution inside the nodes of the graphs. Only certain nodes for which the execution time is critical, e.g., convolution, have also been implemented at native level using JNI. In these cases the optimization has been targeted to use processor specific instruction sets for acceleration of floating point operations, e.g., Altivec for PowerPC or SSE for Pentium. A result of this work is a growing library of nodes which implement various image processing operations, some of them in Java, and some optimized at native level.

In IPAPI, graphs can be defined as new node classes and instantiated as nodes in other graphs. This implies that relatively complex graphs, containing a large number of nodes, can be implemented without too much work. Furthermore, memory management, scheduling of execution and data flow can be customized for various purposes. This flexibility in scheduling is lacking in existing systems of similar type such as AVS, Khoros, etc.

The IPM is responsible for all low-level image processing, much of which has to be done in real time. This real-time requirement implies both that the response times from this module have to be short enough to allow it to be included in control loops, e.g., for tracking of ground vehicles, and that sequences at normal video rate have to be managed. It should be emphasized that the system is not designed to manage the processing of a continuous video sequence, but instead the processing of short image bursts at varying rates, e.g., for motion estimation.

The two modules which the IPM communicates most closely with are the CONTAP Executor Module (CEM) and the Dynamic Object Repository (DOR). CEM is part of the reactive layer of the system, responsible for setting up and monitoring the execution of various low-level sub-tasks, e.g., the flight of the he-

licopter, control of the camera, and the processing of images. The DOR is a soft real-time database which keeps records of information about various objects, both static and dynamic, that the system needs to know about when achieving various tasks. This may include sighted ground vehicles during a traffic surveillance scenario, but may also include information about the helicopter itself and the camera. Information in the DOR is normally subject to variation over time, and one of the main tasks of the IPM is to keep the information about various sensed objects up to date, at least for the objects rated as interesting by the higher levels of the system. The interaction between the CEM and the IPM may be viewed as a form of higher-level active vision where the context which determines image processing policy is represented implicitly in the DOR and interpreted by the CEM via the use of other deliberative services.

The CORBA based solution makes the system very flexible regarding choice of hardware/software platforms for various parts of the system, only at the expense of a relatively small overhead in communication latency. For example, for the development of the software architecture and even use during runtime, each of the three modules mentioned here can be run on the same or separate hardware platforms, using CORBA to manage the interface issues. In the case of hard real-time constraints, the IPM or DOR could be executed on a processor with a real-time OS using CORBA real-time channels to ensure quality of service relative to event arrival and scheduling. Even for communicating images, e.g., from the camera to the image processing module, the time delays introduced by CORBA are acceptable for most of the planned applications. However, this relies on using advanced implementations of CORBA, e.g., supporting real-time functionality and shared memory.

4. Examples of operations

As mentioned above, IPAPI has a CORBA interface which means that any other module of the system potentially can act as a client, creating and executing data flow graphs corresponding to various image processing operations. In practice, however, it is the CEM who is the client using these services.

Recall two of the high-level tasks mentioned in section 2, “find vehicle” and “track vehicle”. Depending on the time available for the processing and the requirements for robustness, the CEM can choose an appropriate algorithm and implement it in the IPM using the library of predefined nodes existing in IPAPI or pre-defined DFGs specialized for common tasks.

A typical result of the “find vehicle” operation is that a set of potential vehicles are identified. The terminal node of the corresponding data flow graph then

exports the list of objects, acting as a CORBA client, to the DOR which is implemented as a CORBA server. At this point, the reasoning layers of the system examine the "vision objects", e.g., check if their velocities are consistent with the direction of the road, or if their positions are on a road, or are consistent with predictions from earlier positions. If this is the case, a hypothesis can be made that this is an "on-road object" or a "car object" after additional reasoning. The chronicle recognition service can then be called to identify various patterns of interest, i.e., simple sequences of events such as changing lane, stopping, turning, vehicle overtaking, etc. One DFG used for identifying a vehicle is shown in figure 3[d].

Based on this type of processing, the CEM can determine to start tracking a vehicle. This means that the IPM is reconfigured with a new processing graph implementing a suitable tracking method. With sufficient memory and processing power, multiple vehicles can be tracked simultaneously. The tracking operation continuously updates the object record in the DOR corresponding to the vehicle being tracked, while the CEM monitors the tracking for possible tracking failure. One DFG used for tracking a vehicle is shown in figure 3.

5. Summary

A runtime system for implementation and execution of image processing operations has been presented. The system is intended to be used as part of an on-board image processing system on a UAV flying over complex operational environments. The paper has emphasized both the requirements on an image processing system for the intended applications and a software architecture intended to meet those requirements. An experimental version of the architecture is implemented and parts have been tested in the on-board system. The full architecture with the image processing system has been tested in simulation and using video streams captured during autonomous flight. Full integration and experimentation with the IP system on-board is currently one of the major focuses of the project at the systems level.

References

1. P. Andersson et. al. Integrating a computational model and a run time system for image processing on a uav. In *Proceedings of the Euro-Micro Conference*, 2002.
2. P. Doherty et. al. The WITAS unmanned aerial vehicle project. In *Proc. of the 14th European Conf. of Artificial Intelligence*, 2000.
   Project URL: http://www.liu.se/ext/witas.
3. Object Computing, Inc. *TAO Developer's Guide, Version 1.1a*, 2000.

[d] *GetROI* stands for *get Region Of Interest*. The ROI is chosen dynamically by sending requests to the Image Controller.

Figure 1. The WITAS UAV Platform

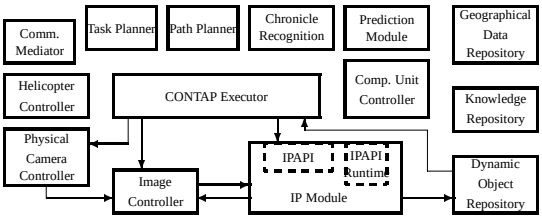

Figure 2. The D/R Architecture

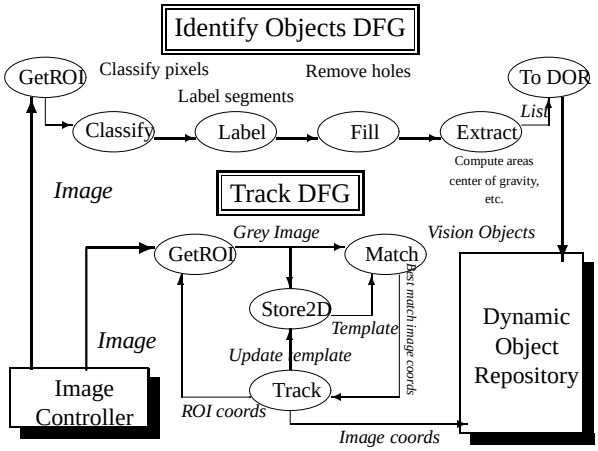

Figure 3. Identify and Track DFGs