Machine Vision and Autonomous Integration Into an Unmanned Aircraft System

Chris Van Horne[3]
Graduate Advisor: James Dianics[4]
Faculty Advisor: Hermann F. Fasel[1]**, Michael W. Marcellin**[2]
[1]**Department of Aerospace and Mechanical Engineering,** [2]**Department of Electrical and Computer Engineering,** [3]**Department of Computer Science,** [4] **Department of Systems and Industrial Engineering**

University of Arizona, Tucson, AZ, 85721

ABSTRACT

The University of Arizona's Aerial Robotics Club (ARC) sponsors the development of an unmanned aerial vehicle (UAV) able to compete in the annual Association for Unmanned Vehicle Systems International (AUVSI) Seafarer Chapter Student Unmanned Aerial Systems competition. Modern programming frameworks are utilized to develop a robust distributed imagery and telemetry pipeline as a backend for a mission operator user interface. This paper discusses the design changes made for the 2013 AUVSI competition including integrating low-latency first-person view, updates to the distributed task backend, and incremental and asynchronous updates the operator's user interface for real-time data analysis.

Keywords: Unmanned Aerial Vehicle, UAV, Pattern Analysis, Computer Vision, Distributed Systems, Convex Optimization, Queueing Theory

1. INTRODUCTION

A capable imagery system has been well documented with the 2011 and 2012 updates. We have previously shown two systems that capture high-resolution 10MP images synchronously and send them to an image server on the ground that runs machine vision and pattern recognition functions displaying the results to a user. Previously, this was always thought of as a serial imagery pipeline and assumed image processing functions would execute quickly to provide an interactive feel for the human operator. This failed in 2011 as the processing functions took minutes to run on each image resulting in large image queue build-up and negating any real-time responsiveness. In 2012 the system was revamped using an optimized machine vision library to reduce runtime which gave the user real-time imagery with real-time results. The improved system required high-end consumer computing hardware and required manual tuning of functions within the pipeline to meet the low latency requirements. In this paper we propose a distributed framework with asynchronous functions, running on the network, to process a generalized real-time image stream with telemetry for a user interface.

Further, the proposed system has a natural flexibility in the amount of computing power that is applied to each function. This is accomplished with a global optimizer that observes processing times and proportions computing resources appropriately. As defined by the user, results from different functional pathways are more important and effect the system at each optimization iteration. Different optimization approaches are formalized and then examined to allow the system to meet user requested benchmarks.

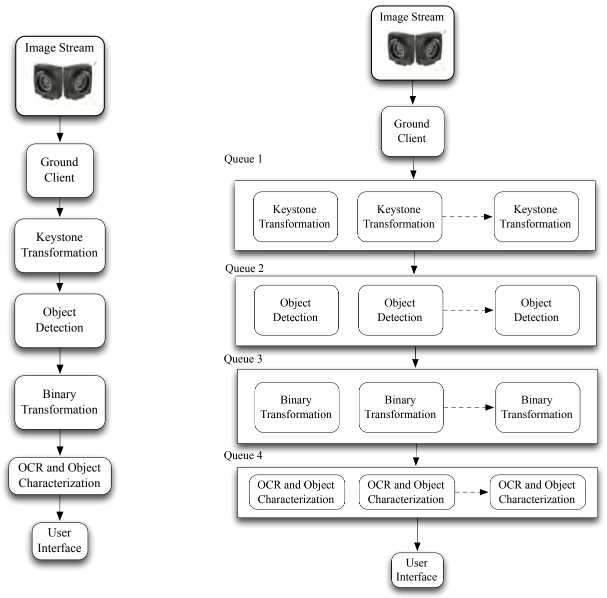


Figure 1. Changes from a serial pipeline (left) to a parallel queued system (right).

1.1. LAARK Implementation

<u>L</u>ow <u>A</u>ltitude <u>A</u>erial <u>R</u>econnaissance <u>K</u>it (LAARK) is composed of the avionics carried in the JUPITER airframe. The CloudCap Piccolo 2 autopilot is used for air vehicle stabilization and waypoint navigation as well as a telemetry source for the onboard FitPC2 computer. The onboard computer uses the telemetry to control the gimbal camera stabilization system and for image metadata. Images are snapped at a rate that guarantees full ground coverage 120° from direction of travel at 35 knots. Images are sent over standard 2.4GHz WiFi to a ground client residing on the LAARK network. The analog video stream from a GoPro Hero 3 is transmitted over 5.8GHz and provided to the LAARK network and is then displayed through the Oculus Rift VR headset. Imagery is guaranteed at a rate of 10Mbps within a 2-

mile radius with the use of omnidirectional and sector antennas. Range of video transmission has not been tested.

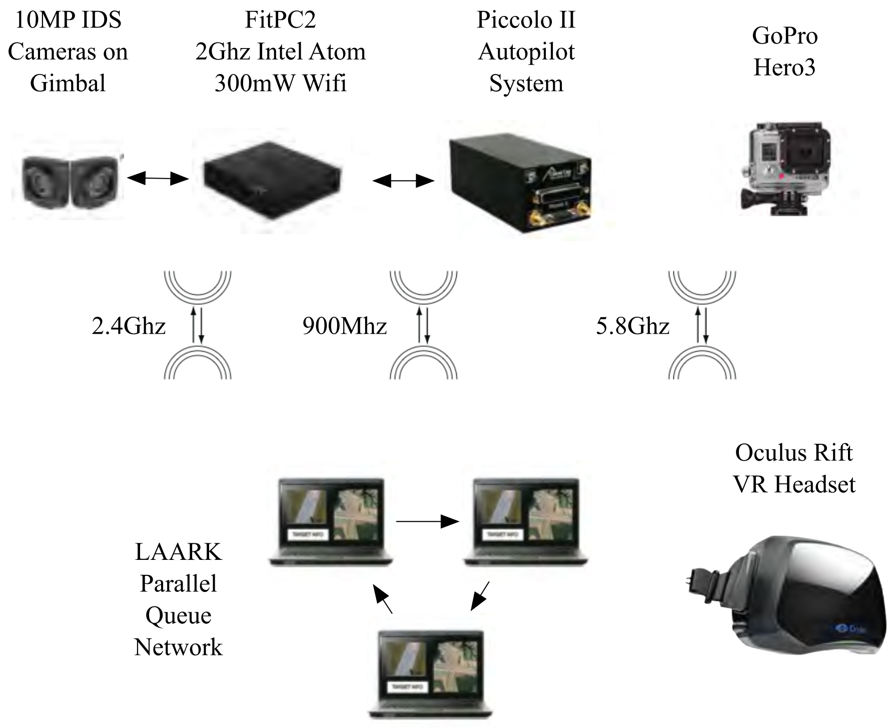

10MP IDS Cameras on Gimbal
FitPC2 2Ghz Intel Atom 300mW Wifi
Piccolo II Autopilot System
GoPro Hero3
2.4Ghz
900Mhz
5.8Ghz
Oculus Rift VR Headset
LAARK Parallel Queue Network

Figure 2. LAARK components diagram; in airframe (top) and ground (bottom). Imagery utilizes 2.4GHz, telemetry and autopilot 900MHz, video 5.8GHz.

LAARK software has been developed to detect small 4x4 foot targets for the Seafarer Chapter AUVSI competition. The system performs filtered object recognition for triage and then reports shape, alphanumeric character, color and location. The results are reported to a user interface in the browser. LAARK has been extended to include real-time video to an Oculus Rift VR headset using a USB digitizer to capture analog video and do transformations in OpenCV. The parallel queue system is utilized to display results at varying rates. A serial system would result in a video display at the rate of the slowest updating component, with parallel queues the system displays frames at max frequency (30Hz NTSC standard 480i) while updating other features like airspeed or detected targets at slower rates. The air computer ran Ubuntu 12.10, computers on the LAARK network ran OS X or Linux and the CloudCap groundstation software ran in Windows XP.

2. FILTERING AND CLASSIFYING OBJECTS

Automatic Target Recognition, or ATR, can be described as a series of algorithms or devices that actively detect objects from a collection of one or more sensors. ATR does not necessarily constitute object detection in imagery, but instead is generalized to any sensor. It is often

- Numpy — Scientific Computing
- ZeroMQ – Message Queue Implementation
- Redis – Data Structure Server
- OpenCV – Image Transformations and Object Detection
- py-rq — Queue Implementation
- CVXOPT — Convex Optimization
- somagic — Unix USB Analog Capture Drivers

Figure 3. List of utilized open source packages

the case that a multitude of sensors, such as GPS, acoustic, and imagery, will all be used in conjunction to contribute to the success of an accurate target recognition algorithm.

It is often necessary to filter results from a classifier to determine characteristics of the targets. In this section a method of filtering targets is detailed in a case-analysis as described in the previous section.

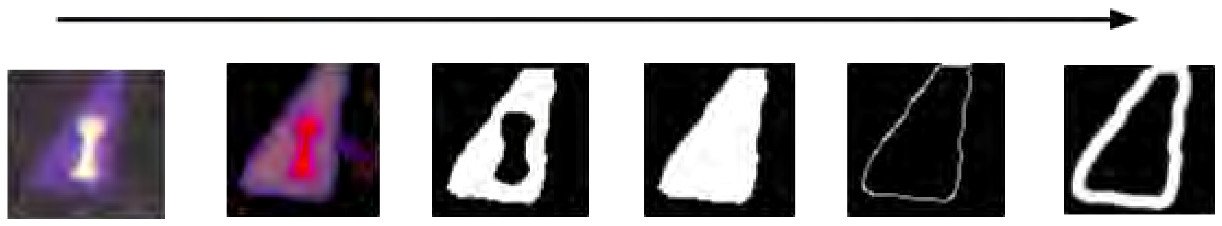


Figure 4. Six stages of progressive chip analysis.

Object recognition can be characterized by the output from the object classifier. At this part of the chip analysis pipeline, it is assumed that an image chip is available for processing. This is represented by the leftmost image in Figure 4. Next, filtering on the background is used to remove noise and provide a clear representation of the target. A simple mean and standard deviation of chip corners filters out pixels which fit within the constraint. Stages three and four then perform a binarization technique to remove remaining interior colors for object detection. Colors are binned by HSV which then allows easy filling, masking, and extraction. The final stage, shown as the rightmost chip in Figure 4, is generated by analysis on contours via decision trees and/or approximation.

Target classification is provided by the Viola and Jones algorithm for rapid object detection using a boosted cascade of simple features. This approach is built into OpenCV's [1] objdetect module. In addition, the existing implementation was improved with extended Haar features from Lienhart and Maydt [2] [3].

3. LIMITATIONS OF A SERIAL IMAGE PROCESSING ARCHITECTURE

During competition years 2011 and 2012, an easy-to-understand serial pipeline architecture had been implemented for telemetry processing. Year 2012 saw a previously monolithic image processing portion of the serial pipeline refined into separate classification stages detailed in section 2. Chip analysis (Figure 4) details the processing function separation into stages which then were modeled into a serial pipeline shown in left Figure 1. Due to architecture decisions of not allowing more than one ingress image into the pipeline during a full processing of the queue, no parallel gains from a distributed architecture could be realized. Abstraction of network facilities through a message queue layer provided the level of indirection necessary for future work to fully exploit a parallel workload within the image processing queue.

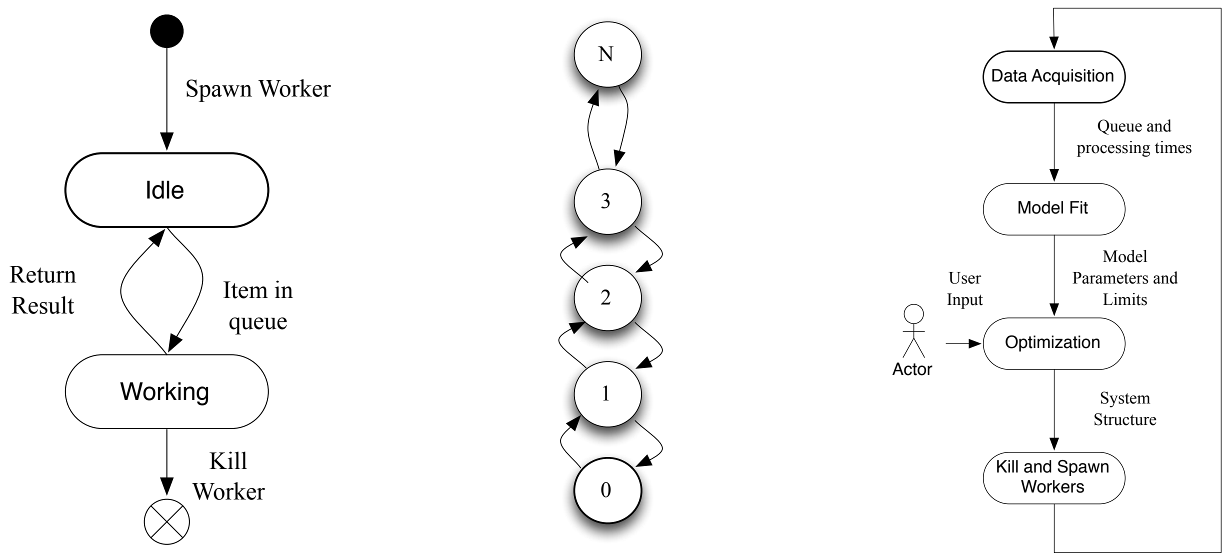
4. IMPLEMENTATION FOR A PARALLEL PROCESSING ARCHITECTURE

The system is composed of an image and telemetry source, workers for each function and an optimizer. Workers are assigned to queues by the optimizer and each queue is separated by function. A Redis [4] key-value datastore is used in conjunction with the RQ [5] Python queue implementation for communication. The system starts with the initialization of the optimizer and the local worker pool as other machines come online they post their participation to the Redis database and are used in the next optimization loop. A base configuration is established that assigns at least one worker per queue. As imagery is received jobs get added to the queue by a ground station client that interfaces with the camera system via a ZeroMQ [6] socket.

5. CONVEX OPTIMIZATION FOR AN OPTIMAL PARALLEL PROCESSING CONFIGURATION

Providing the necessary parallel-capable architecture changes, attention was then turned to optimally CPU-loading compute nodes amongst the various processing queues. While finding a performant layout for a small network of queues and/or machines can be found with a quick trial-and-error method, it was soon realized that a more general solution was needed to scale for future system growth.

We begin with the formal extension of convex optimization over *M/M/1* queues to minimize service delay for a Markovian queueing system of N queues. The proof that such a construct is convex and efficiently solvable is detailed in *Chiang, Sutivong, and Boyd* [7]. Specifically, we seek to minimize the sum arrival rate λ of images and sum service rate μ of classified chips. We place lax constraints on the average queue delay W, total delay before service D,



(a) Worker (b) Queue (c) Optimization

Figure 5. State diagrams of a worker, queue, and optimizer. Queues can be aligned in series or parallel and have unlimited number of workers. The system is observed by a signal optimizer.

and strict restraint on queue occupancy Q. Each queue is supported by constraints W_i, D_i, Q_i, and optimizes over λ_i and μ_i. Effectively, we seek to manipulate the depth Q_i, or more concretely the number of processing nodes needed in the i'th queue, given ingress λ_i work to be completed relative to egress μ_i work completed for a particular queue. Formally, the optimization is programed as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} \alpha_i \frac{\mu_i}{\lambda_i} \\ \text{subject to} \quad & W_i \leq W_{i,max} \\ & D_i \leq D_{i,max} \\ & Q_i \leq Q_{i,max} \\ & \lambda_i \geq \lambda_{i,min} \\ & \sum_{i=1}^{N} \mu_i \leq \mu_{max} \end{aligned}$$

where optimization occurs on the arrival rate λ_i and service rate μ_i. The weight α_i gives the weight, or priority in our context, of the i'th queue. The process of minimization over the weighted sum of service load ratios gives a per queue query that tells if processing nodes should be added or removed. For each queue we consider $\frac{\lambda_i}{\mu_i}$ to be the ingress/egress ratio ϕ_i and perform an action on the occupancy of queue Q_i based on the value of ϕ_i.

The manipulation of queue occupancy for each queue is then a simple case-based inspection of the respective ϕ. When $\phi_i < 1$, the optimizer is stating that Q_i can receive more ingress traffic as its egress traffic is greater. If $\phi_i > 1$, there is a larger amount of ingress traffic feeding into Q_i than what is being completed and egressed. Specifically, in the image processing pipeline each of these scenarios is stating when a queue is over-provisioned and processing nodes could be removed or under-provisioned and processing nodes must be added. The process of removal within a finite resource pool is implemented as a swap operation between the under-provisioned queue Q_i and an over-provisioned queue Q_j where $i \neq j$. When $\phi_i = 1$, ingress and egress are balanced and no node manipulation is performed on the occupancy of Q_i .

6. FRUITS OF PARALLEL POWER: AUGMENTED REALITY HEADSETS

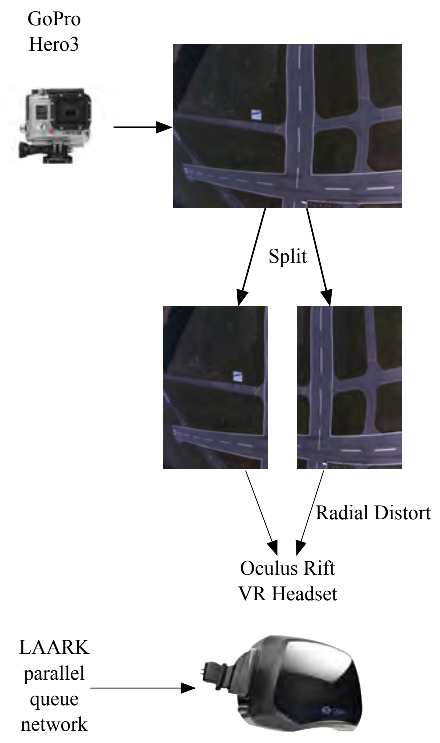


Figure 6.

Since the image source has been generalized for the system we can now use it to easily run object detection from any source. By taking advantage of the optimized parallel nature changing resolutions and image frequency affects the system in such a way that does not delay displaying frames to the user. Added functionality gets ran in the background and reports to the user at a readily available time. With a serial pipeline one would need to tweak and tune the functions to ensure the through rate hit the 30 frames per second max.

The split image view is required to drive the oculus rift as a video display device. Imagery is captured into the system using an off the shelf analog video digitizer. Once captured, a simple function is written to split the image with a slight overlap. Radial distortion is then applied to each image to mock the natural distortion of the eye. On projection the user regains some natural feeling of space from a 3D environment. This, along with LAARK intelligent object detection, adds a new dimension to UAV human interfaces.

7. CONCLUSIONS

The previous uses of a network abstraction layer via messaging queuing provided a stable platform for modification which led to the early ideas of true parallel processing. The use of computer vision algorithms which had feature detection as a partitionable operation naturally fit into the idea of a parallel processing queues. Extended to the Markovian queue, we were able to provide optimal configuration of processing nodes within each queue for minimizing varying constraints. This ability to distribute workloads optimally provided the increased available resources to perform more interesting tasks such as augmented reality through HMD setups.

8. ACKNOWLEDGEMENTS

AVATAR/LAARK has been supported by Dr. Hermann F. Fasel of Department of Aerospace and Mechanical Engineering, University of Arizona, Tucson, for which we owe our thanks. Special thanks for technical assistance goes to: Mark Ballinger and Anton Kochevar, Raytheon Missile Systems; Robert Perry, SAIC; Justin Armer, Latitude Engineering; faculty mentor Dr. Ricardo Sanfelice, Department of Aerospace and Mechanical Engineering, University of Arizona, Tucson; David Henz, Edmund Optics; faculty mentor Dr. Michael W. Marcellin, Department of Electrical and Computer Engineering, University of Arizona, Tucson.

REFERENCES

1. OpenCV (Open Computer Vision Library) 2013. `http://opencv.itseez.com/`.
2. Viola, P. and Jones, M., "Rapid object detection using a boosted cascade of simple features," Computer Vision and Pattern Recognition, IEEE Computer Society Conference on **1**, 511 2001.
3. Lienhart, R. and Maydt, J., "An extended set of haar-like features for rapid object detection," in IEEE ICIP 2002, 900–903 2002.
4. Redis 2013. `http://redis.io/`.
5. RQ (Redis Queue) 2013. `http://python-rq.org/`.
6. ZeroMQ: The Intelligent Transport Layer 2013. `http://www.zeromq.org/`.
7. Chiang, M., Sutivong, A., and Boyd, S., "Efcient nonlinear optimizations of queuing systems," in IEEE Global Telecommunications Conference (GLOBECOMM), 3:2425-2429, November 2002., 2002.