Machine Vision and Autonomus Integration Into an Unmanned Aircraft System

Undergraduate Students: Josh Alexander[1], Sam Blake[1], Brendan Clasby[1], Anshul Jatin Shah[2], Chris Van Horne[3], and Justin Van Horne[3]
Graduate Advisor: James Dianics[4]
Faculty Advisor: Hermann F. Fasel[1]
[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, Tucson, AZ, USA

ABSTRACT

The University of Arizona's Aerial Robotics Club (ARC) sponsored two senior design teams to compete in the 2011 AUVSI Student Unmanned Aerial Systems (SUAS) competition. These teams successfully designed and built a UAV platform in-house that was capable of autonomous flight, capturing aerial imagery, and filtering for target recognition but required excessive computational hardware and software bugs that limited the systems capability. A new multi-discipline team of undergrads was recruited to completely redesign and optimize the system in an attempt to reach true autonomous real-time target recognition with reasonable COTS hardware.

Keywords: Unmanned Aerial Vehicle (UAV), Haar-like features, ZeroMQ networking, CUDA$^{\text{TM}}$ programming, OpenCV.

1. INTRODUCTION

The Seafarer's chapter of AUVSI defines a simulated military rescue scenario that teams were assigned to performed. The full brief of the rules and scenario can be found at `http://www.auvsi-seafarer.org/`. In short the basic requirements are

- Autonomous flight including take-off and landing
- Autonomous target recognition
  - Alphanumeric character and color
  - Target shape and color
  - GPS location and orientation

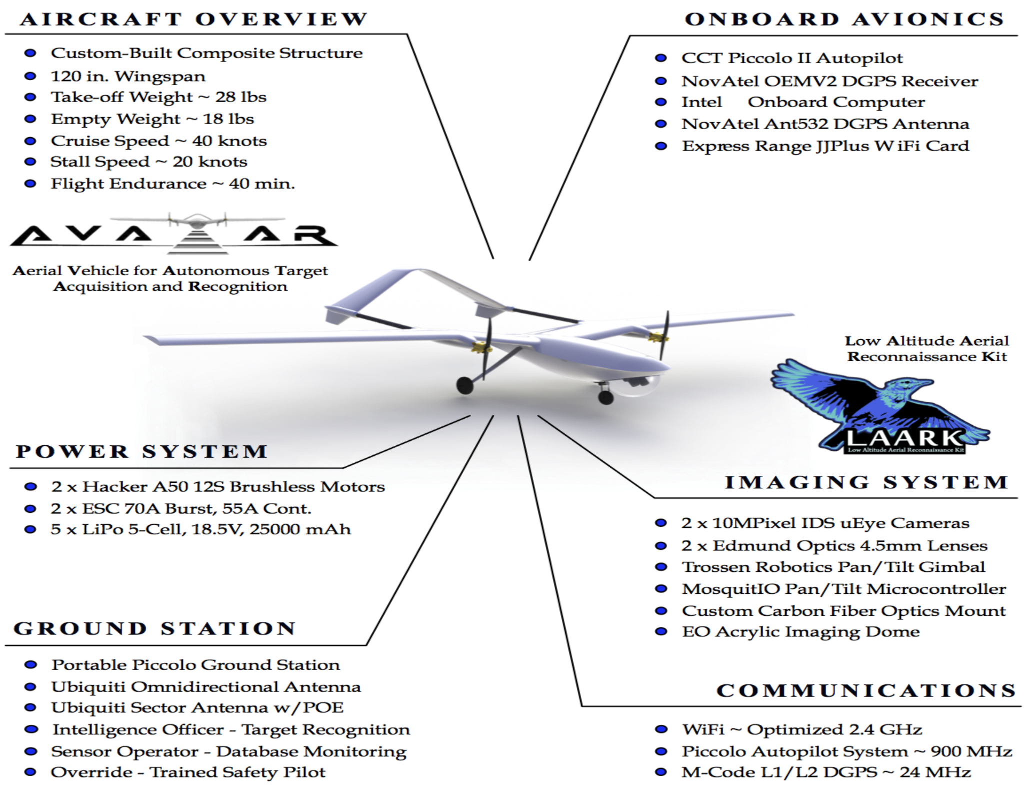
AIRCRAFT OVERVIEW
Custom-Built Composite Structure
120 in. Wingspan
Take-off Weight ~ 28 lbs
Empty Weight ~ 18 lbs
Cruise Speed ~ 40 knots
Stall Speed ~ 20 knots
Flight Endurance ~ 40 min.
AVATAR
Aerial Vehicle for Autonomous Target Acquisition and Recognition
ONBOARD AVIONICS
CCT Piccolo II Autopilot
NovAtel OEMV2 DGPS Receiver
Intel Onboard Computer
NovAtel Ant532 DGPS Antenna
Express Range JJPlus WiFi Card
Low Altitude Aerial Reconnaissance Kit
LAARK
Low Altitude Aerial Reconnaissance Kit
POWER SYSTEM
2 x Hacker A50 12S Brushless Motors
2 x ESC 70A Burst, 55A Cont.
5 x LiPo 5-Cell, 18.5V, 25000 mAh
IMAGING SYSTEM
2 x 10MPixel IDS uEye Cameras
2 x Edmund Optics 4.5mm Lenses
Trossen Robotics Pan/Tilt Gimbal
MosquitIO Pan/Tilt Microcontroller
Custom Carbon Fiber Optics Mount
EO Acrylic Imaging Dome
GROUND STATION
Portable Piccolo Ground Station
Ubiquiti Omnidirectional Antenna
Ubiquiti Sector Antenna w/POE
Intelligence Officer - Target Recognition
Sensor Operator - Database Monitoring
Override - Trained Safety Pilot
COMMUNICATIONS
WiFi ~ Optimized 2.4 GHz
Piccolo Autopilot System ~ 900 MHz
M-Code L1/L2 DGPS ~ 24 MHz

Figure 1. LAARK AVATAR System Component List

2. AIRFRAME BACKGROUND

A very specific platform was necessarily designed and manipulated for implementation of an onboard reconnaissance kit. The Aerial Vehicle for Autonomous Target Acquisition and Recognition, further to be recognized as the AVATAR, was meticulously designed and constructed for aerial performance with respect to the onboard vision recognition system. In order to provide the most effective operational floor for reconnaissance, the AVATAR airframe is aerodynamically oriented for high stability in numerous flight regimes. Inherent flight stability is achieved through the use of specific features such as an inverted v-tail, a dihedral wing, and a dual prop design rather than a single prop design.

Stable
Autonomous Flight
Auto
Takeoff
Auto
Landing

To increase the stability and performance of the aircraft beyond a mechanical standpoint, the Piccolo II autopilot was incorporated making this an entirely unmanned aerial system. Establishing the autopilot control allowed for a much larger operational range, by changing the limiting factor from human sight to radio frequency range, and provided computer assistance for performance and stability. The autopilot system allowed for tuning of the control surfaces to reduce oscillations during flight, and introduced greater stability by performing in time calculations of wind speed and direction and adjusting its controls accordingly. These both contributed to a much steadier flight providing a best case scenario for image acquisition. Besides increased stability, the autopilot maintained steady flight speeds and altitudes adjusting for aerodynamic inconsistencies in the air. These constraints were refined through trigonometric calculation and field testing giving an operational flight altitude of 250 ft AGL and an air speed of 32.5 kts. These characteristics are idealized for the imagery being acquired. At such conditions the AVATAR can operate autonomously for nearly 35 minutes and can fly steadily in crosswinds of up to 18 kts. This has been tested proven in real flight missions, thus the AVATAR is an ideal aerial solution providing a stable, efficient platform for aerial reconnaissance.

3. TRIVIAL AIR-GROUND TRANSPORT LAYER WITH ZEROMQ

3.1. INTRODUCTION TO ZEROMQ

The ZeroMQ[1] (”Zero-Em-Queue”) network transport layer grew out of Wall Street efforts to create a low-latency message queue framework which simplified traditional BSD socket programming. Instead of creating a complex, all-encompassing, or simplistic and narrow API, ZeroMQ provides the concept of transport ”patterns.” By logically extending existing socket naming conventions (`send(2)`, `recv(2)`), the ZeroMQ API is easy for unfamiliar programmers to pick-up and quickly begin thinking in terms of high-level network topology and communication patterns.

The request-reply pattern, Figure 2.a, represents the most simple and obvious network communication pattern. Sockets operating in the REQ-REP context perform their operations in lockstep. Typically in a loop, the server and client communicate by sending messages back-and-forth via send and receive API calls. If a client or server attempts to send or receive before the other side has acknowledged, deadlock occurs and an exception is thrown on the offending side.

Fan-out concurrency models are useful in worker-consumer patterns. ZeroMQ provides the PUSH-PULL communication pattern to provide this network topology. An arbitrary number of workers (PUSH) may spin-up on demand and push their completed work to a centralized consumer (PULL.) As shown below, code to demonstrate the worker-consumer pattern is short and readable. Secondly, the code differences between server and client is minimal and largely abstracted behind the conceptual model:

```
context = zmq.Context()                  context = zmq.Context()
socket = context.socket(zmq.PUSH)        socket = context.socket(zmq.PULL)
socket.connect('tcp://127.0.0.1:5555')   socket.bind('tcp://*:5555')

while True:                              while True:
    # Do expensive function..                # Receive completed work...
    message = do_expensive_op()              message = socket.recv()
    # Send results to consumer.              # Alert user of completed work
    socket.send(message)                     print 'completed', i
```

Traditionally, a BSD socket approach would require complicated polling and possibly locking when operating in an N:1 context. ZeroMQ goes further by allowing more complex M:N worker-consumer strategies with no code changes; simply run more PULL services.

3.2. AIR-GROUND MESSAGING

The LAARK subsystem uses three distinct ZeroMQ socket patterns: REQ-REP for imagery synchronization; PUSH-PULL for data transfer to the ground station(s); and a variation

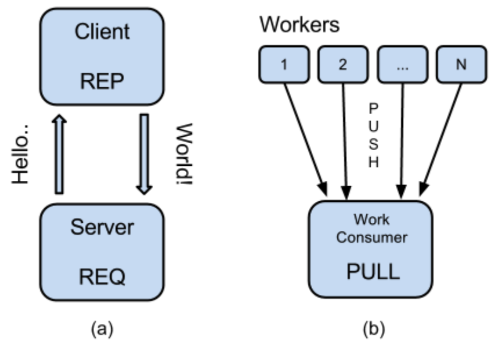

Figure 2. (a) lockstep REQ-REP and (b) PUSH-PULL concurrency with fan-out.

of publish-subscribe called PUB-PULL for telemetry streaming. Isolating components with particular communication patterns also simplified the conceptual model of how the entire system fit together. Knowing that a particular component communicated as request-reply inferred knowledge of how to add or manipulate the surrounding code. Using the high-level language of network architecture provided by ZeroMQ allowed the LAARK team to quickly and remove code on the fly without having to reason locking semantics of private resources.

4. AUTOMATIC TARGET RECOGNITION WITH OPENCV

4.1. INTRODUCTION TO AUTOMATIC TARGET RECOGNITION

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 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.

This paper discusses the approaches of target recognition in imagery using the Viola-Jones algorithm. Viola and Jones describe an algorithm for rapid object detection using a boosted cascade of simple features. This approach was built into OpenCV's[2] `objdetect` module and is most often attributed to the success of facial recognition. In addition, the existing implementation was improved with extended Haar features from Lienhart and Maydt[3] [4].

4.2. TRAINING A SUCCESSFUL CLASSIFIER

Traditionally a classifier is constructed by describing features on a large collection of imagery in a supervised learning setting. Labeled examples are described with corresponding feature vectors. In any case it is not unusual to train against thousands of images to create a reliable classifier, such as the case of facial recognition.

Unlike many supervised learning algorithms, Viola-Jones is a unsupervised object detection algorithm, that is it deduces features from unlabeled training data. This means that quality of a classifier is dependent on the quality of the training data. The disadvantage to this approach is that it is often susceptible to over-fitting. However, by providing training data of natural and computed distortions of targets a complex model classifier can be created with an optimum interference-estimation calibration.

The success of an ATR algorithm is based on ratio of positives, false-positives, and negatives of detected targets. These metrics are found by testing a trained classifier on non-trained, but validated data. This is often a laborious task, however it can be automated by generating samples and automatically truthing the data.

Unlike facial recognition, aerial imagery is often very consistent in its background; in its most basic form the imagery can be initially classified by its terrain, such as desert, forest, rural, grass, and urban. This general feature is important to providing an initial filter on the data to mitigate the interference of background noise. An example filter on a grassy field could be computing the standard deviation of the corner neighbors in an image and applying an adaptive threshold within a certain percentile of the standard deviation to remove most background noise. In isolated regions, such as neighborhoods, plains, or forests enough negative samples can be supplied to the classifier that the interference is negligible. This type of classifier can be achieved by maintaining a detection-false -alarm rate of 1:2.

OpenCV provides a means of creating such a classifier with the `opencv_haartraining` application. This tool is strictly for computed a classifier based on the Viola-Jones/Lienhart-Maydt algorithm. The parameters specified allow the user to specify the number of stages, branches, hit-rate, and false-alarm rate.

4.3. CASE-USE CLASSIFIER FOR OBJECT DETECTION IN A GRASS FIELD

To facilitate the target recognition of common shapes in a grassy field, a series of training tools were constructed to automate the task of generating sample data. This sample data included the automation of skewing, rotating, and applying perspective transforms to aid in the prevention of over-fitting the data.

The output of these tools create a positive and negative sample list that can be used as input to `opencv_createsamples` application. OpenCV expects a specialized vector format which is used in the processing of the `opencv_haartraining` tool. An example of the input formats are shown below. Each line in the `positive.txt` file describe an image where the objects reside, the number of objects, followed by the position and dimensions of each object. The `negative.txt` lists each image in which no object is found. The specialized vector format can then be constructed:

```
opencv_createsamples -info positive.txt -vec positive.dat -num 131 -width 20 -height
20
```

For this application the initial classifier was constructed with 150 positive samples and

```
image-0001.jpeg 1 2650 330 47 33
image-0002.jpeg 1 2710 1580 40 37
image-0003.jpeg 3 2797 940 53 37 866 967 37 30 966 1394 27 23
image-0004.jpeg 2 736 276 40 37 480 1000 23 20
```

Figure 3. positive.txt

```
image-0005.jpeg
image-0006.jpeg
image-0007.jpeg
image-0008.jpeg
```

Figure 4. negative.txt

300 negative samples. OpenCV's `opencv_haartraining` was invoked with the following parameters:

```
opencv_haartraining -data haartraining -vec positive.dat -nstages 20 -nsplits
2 -minhitrate 0.999 -maxfalsealarm 0.5 -npos 150 -w 20 -h 20 -mem 512 -mode ALL
-bg negative.txt -nneg 300
```

These parameters construct a 20-stage cascade classifier with a minimum hit-rate of 0.999, a false-alarm rate of 0.5, and a minimum target size of 20x20. This procedure is computationally expensive and may take upwards of days to compute the final stage in the classifier.

The end-result classifier can be tested against the `opencv_performance` tool which runs the generated classifier against a positive sample file. This provides a means of computing metrics such as the hit, miss, and false-positives of the classifier. An example output is shown below:

```
$ opencv_performance -data haar -info positive.txt
```

File Name	Hits	Misses	False
image-0010.jpeg	1	0	6
image-0011.jpeg	1	0	8
image-0012.jpeg	1	2	0
image-0013.jpeg	1	0	2
Total	4	2	16

```
Number of stages: 15
Number of weak classifiers: 48
Total time: 9.000000
```

4.4. FILTERING AND IDENTIFYING OBJECTS

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.

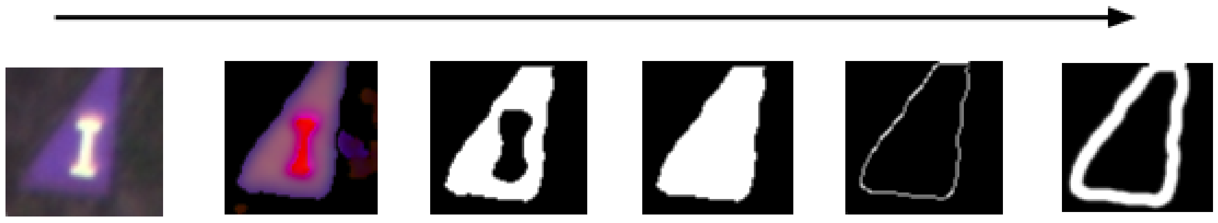

Figure 5. Six stages of progressive chip analysis.

Object recognition can be characterized by the output from the object classifier. At this part of the chip analysis pipeline, it was assumed that an image chip was available for processing. This is represented by the leftmost image in Figure 5. Next, filtering on the background was used to remove noise and provide a clear representation of the target. A simple mean and standard deviation of chip corners filtered out pixels which fit within the constraint. Stages three and four then performed a binarization technique to remove remaining interior colors for object detection. Colors were binned by HSV which then allowed easy filling, masking and extraction. The final stage, shown as the rightmost chip in Figure 5, was generated by analysis on contours via decision trees and/or approximation.

5. CUDA–ACCELERATED TANGENTIAL AND RADIAL DISTORTION CORRECTION

The wide-angle lens of the LAARK imagery system, coupled with non-orthogonal mounting angles to provide 120° field-of-view, caused a great deal of tangential (”keystone”) and radial (”barrel”) distortions in imagery. Correcting the aberrations requires applying the following transformations[5]:

$$\begin{aligned} x_i^{(t)'} &= 2\,p_1\,x_i\,y_i + p_2\left(r_i^2 + 2\,x_i^2\right) \\ y_i^{(t)'} &= p_1\left(r_i^2 + 2\,y_i^2\right) + 2\,p_2\,x_i\,y_i \end{aligned} \tag{1}$$

$$\begin{aligned} x_i^{(r)'} &= x_i\left(k_1\,r_i^2 + k_2\,r_i^4 + k_3\,r_i^6 + ...\right) \\ y_i^{(r)'} &= y_i\left(k_1\,r_i^2 + k_2\,r_i^4 + k_3\,r_i^6 + ...\right) \end{aligned} \tag{2}$$

where old pixel coordinates (x_i, y_i) are mapped to (x_i', y_i') through the respective transform pair, p_1 and p_2 are tangential distortion coefficients, and k_1, k_2 and k_3 are radial distortion

	CPU	GPU
3GHz Core 2 Duo / GT 430	2078 ms	348 ms
3.4GHz i5 / GTX 590	1012 ms	112 ms

Table 1. CPU vs. GPU OpenCV `remap` performance on 3264 x 2448 dimension image.

coefficients. These parameters are assembled into a five-element vector. From here, OpenCV allows us to then apply the `remap` function which can correct an image.

The process of applying the remapping projection to images of 3000 $\times$ 2000 pixels and larger presented a problem for real-time UAV imagery. Each image remap would take longer than the transmission time of a single image and thus caused a backlogging of work to be completed on the ground side. As a cluster of computers was not a practical solution, the OpenCV CUDA$^{\text{TM}}$-assisted API was investigated.

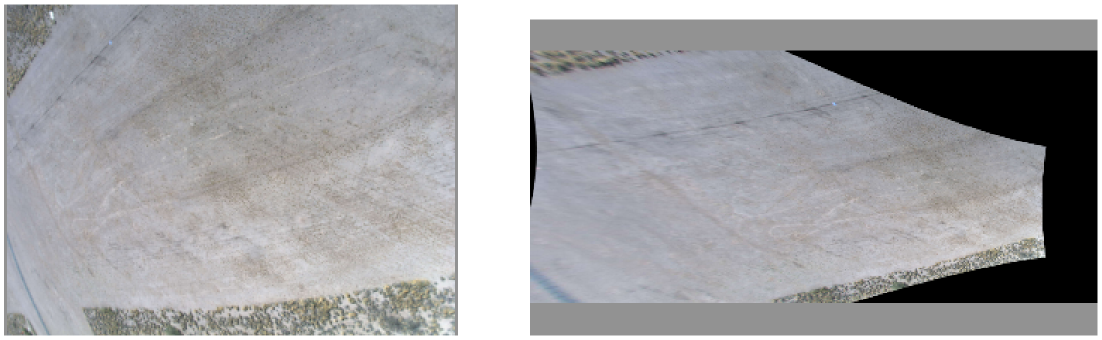

Figure 6. Left: distorted "raw" imagery. Right: tangential-radially corrected image.

6. CONCLUSIONS

The use of ZeroMQ for networking allowed for a rapid development process and ease in debugging. Often times, simply drawing out a flowchart for a desired or buggy operation allowed other team members to quickly spot deficiencies and aid in general understanding by assigning formal names to different socket patterns. Results from Haar-like features provided us with an ability to truth and test against various backdrops, from grassy-green Maryland to desert-dust Arizona, without making any changes to our underlying detection framework. Coupled with OpenCV's traditional performance, we were able to make low-latency, high-throughput image analysis pipelines with minimal hardware. Migration of image processing bottlenecks, namely re-projection, to the much faster GPU via OpenCV CUDA backends, allowed us to further reduce our system hardware profile. These improvements led to the successful deployment of the 2012 AUVSI AVATAR/LAARK system on a reduced hardware profile while improving extensibility, readability, and performance.

7. 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. ZeroMQ: The Intelligent Transport Layer 2012. `http://www.zeromq.org/`.
2. OpenCV (Open Computer Vision Library) 2012. `http://opencv.itseez.com/`.
3. 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.
4. Lienhart, R. and Maydt, J., "An extended set of haar-like features for rapid object detection," in IEEE ICIP 2002, 900–903 2002.
5. Heikkila, J. and Silven, O., "A four-step camera calibration procedure with implicit image correction," in Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), CVPR '97, 1106–, IEEE Computer Society, Washington, DC, USA 1997.