The most trusted source of analysis, advice, and engineering for embedded processing technology and applications

Implementing Vision Capabilities in Embedded Systems

EE LIVE! Featuring ESC

Shehrzad Qureshi

Berkeley Design Technology, Inc.
Walnut Creek, California USA
+1 (925) 954-1411

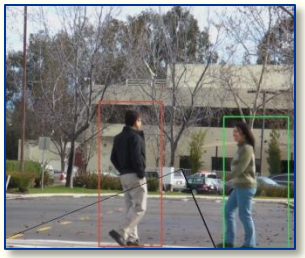
info@BDTI.com
http://www.BDTI.com

INTRODUCTION

What is BDTI?

BDTI is a 20 year-old company dedicated to helping engineers use digital signal processing technology

BDTI performs hands-on, independent benchmarking and evaluation of chips, tools, algorithms, and other technologies



BDTI helps system designers implement their products through specialized engineering services, including embedded vision system design

BDTI offers a wealth of free information:

- Benchmark results
- White papers
- Technology insights

What is Embedded Vision?

- "Embedded vision" refers to embedded systems that extract meaning from visual inputs
  - Embedded vision is distinct from multimedia
- Emerging high-volume embedded vision markets include automotive safety, surveillance, and gaming
  - The Xbox Kinect is the fastest-selling CE device to date: 10 million units in 4 months

$130 including game

$920 installed

$300 + $6/month

Why is Embedded Vision Proliferating Now?

1. It has the potential to create huge value
   - Applications in consumer, medical, automotive, entertainment, retail, industrial, aerospace, …
2. It's now possible
   - Sufficiently powerful, low-cost, energy-efficient processors are now emerging
3. Increasingly, it will be expected
   - As embedded vision becomes common in gaming, consumer electronics, and automotive equipment, consumers will expect it

Implementing Embedded Vision is Challenging

- It's a whole-system problem
- There is limited experience in building practical solutions
- Embedded systems are often highly constrained in cost, size, and power consumption
- It's very computationally demanding
  - E.g., a 720p optical flow algorithm, optimized for a modern VLIW DSP architecture, consumed about 200 MHz/frame/second ➔ 5 fps @ 1 GHz
  - Many vision functions will require highly parallel or specialized hardware
  - Algorithms are diverse and dynamic, so fixed-function compute engines are less attractive

Objective/Scope of This Presentation

- Introduction to embedded vision
- Example embedded vision applications
- Example embedded vision algorithms
- Processor types for embedded vision
- A few thoughts on tools and techniques for embedded vision
- Resources for further exploration

APPLICATIONS

Applications: Introduction

- Applications of embedded vision are numerous and diverse
- They span almost every major electronic equipment market, including consumer, entertainment, automotive, industrial, security, medical, and aerospace
- In this section we'll briefly look at a few representative applications
- It can be useful to consider the functionality required as distinct from the system and the market
  - Similar functionality may be useful in a variety of systems targeting different markets
  - E.g., gesture-based user interfaces can be useful in smartphones, point-of-sale terminals, industrial equipment, medical devices

Application: Human Machine Interface

- The expectations of how we interact with technology is changing at a fast pace
- The Xbox Kinect has introduced a whole generation to the concept of gesture recognition
- Laptops and smartphones use face recognition to unlock
- The latest smartphones use gaze tracking to determine when the user is looking at the phone

Application: Surveillance

- In the U.S., retail theft alone amounts to ~$40 billion per year
- With growing concerns about safety and security, the use of surveillance cameras has exploded in the past 10 years
- The U.K. has led this trend, and has ~1.85 million cameras installed
  - Approximately one camera for every 35 people
  - ~1.85 million cameras generate ~2.5×10^9 minutes of video daily
- It's impossible to manually monitor all of this video
- Studies in the U.K. generally show no significant reduction in crime where cameras are installed
- "Smart" surveillance cameras use vision techniques to look for specific kinds of events
- Intelligence can be in the camera, in a local server, or in the cloud
- Key challenge: accuracy with diverse environments and requirements

Cernium Archerfish Solo

Application: Automotive Safety

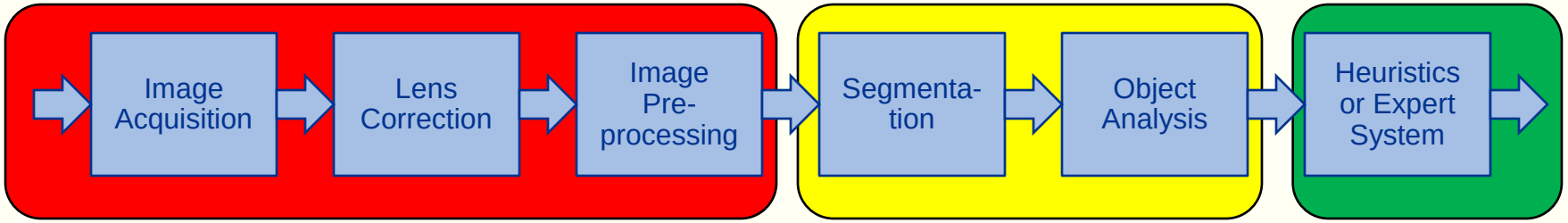
- ~1.2 million people are killed in vehicle accidents annually
- ~80 million new light vehicles are produced annually
- Vision-based safety systems aim to reduce accidents by:
  - Warning when closing in too fast on vehicle ahead
  - Warning of a pedestrian or cyclist in path of vehicle
  - Warning of unintentional lane departure
  - Preventing spoofing of drunk-driving prevention systems
  - Alerting driver when drowsiness impacts attention
  - Automatically dimming high-beams
- Most systems are passive: alert the driver
  - A few apply the brakes
- Some systems augment vision with radar
- Key challenge: accuracy across diverse situations (weather, glare, groups of people, …)

Mobileye C2-270

ALGORITHMS

How Does Embedded Vision Work?

A typical embedded vision pipeline:



Ultra-high data rates; low algorithm complexity

High to medium data rates; medium algorithm complexity

Low data rates; high algorithm complexity

Typical total compute load: ~10-100 billion operations/second

Loads can vary dramatically with pixel rate and algorithm complexity

Lens Distortion Correction—The Problem

- Lenses (especially inexpensive ones) tend to distort images

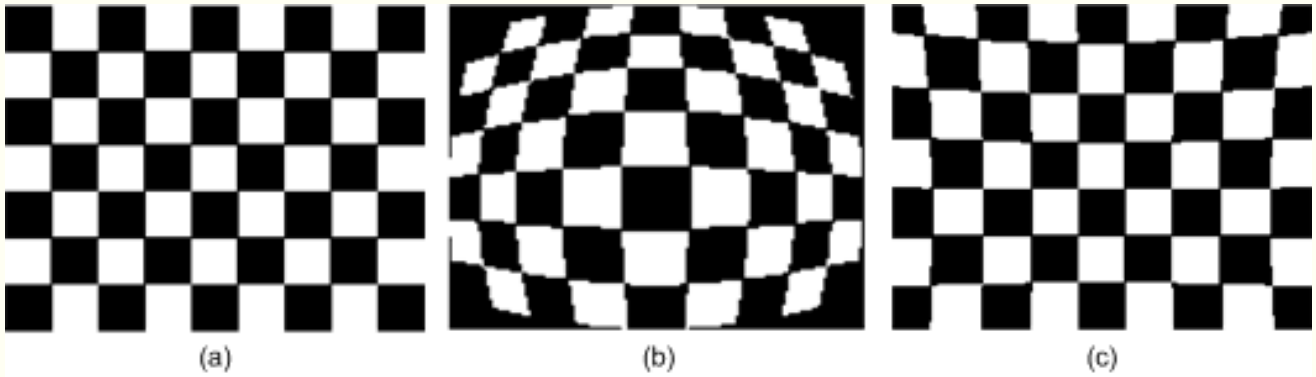


Figure 2. Examples of different types of lens distortion. (a) original (b) barrel distortion (c) pincushion distortion

- Straight lines become curves
- Distorted images tend to thwart vision algorithms

Image courtesy of and © Luis Alvarez

Section based on “Lens Distortion Correction” by Shehrzad Qureshi; used with permission.

Lens Distortion Correction—A Solution

- A typical solution is to use a known test pattern to quantify the lens distortion and generate a set of warping coefficients that enable the distortion to be (approximately) reversed

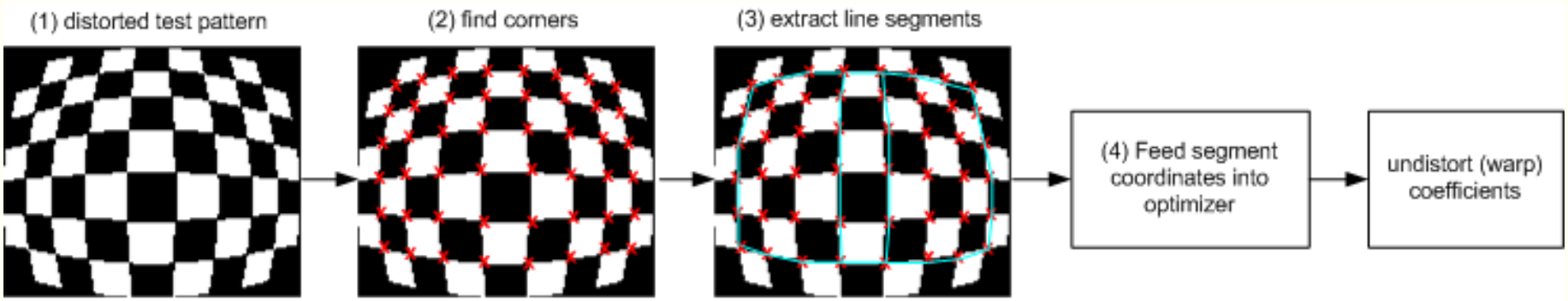


Figure 3. Camera calibration procedure

- The good news: the calibration procedure is performed once
- The bad news: the resulting coefficients then must be used to "undistort" (warp) each frame before further processing
- Warping requires interpolating between pixels

Lane-Departure Warning – The Problem

Detect lane markings on the road and warn when car veers out of the lane

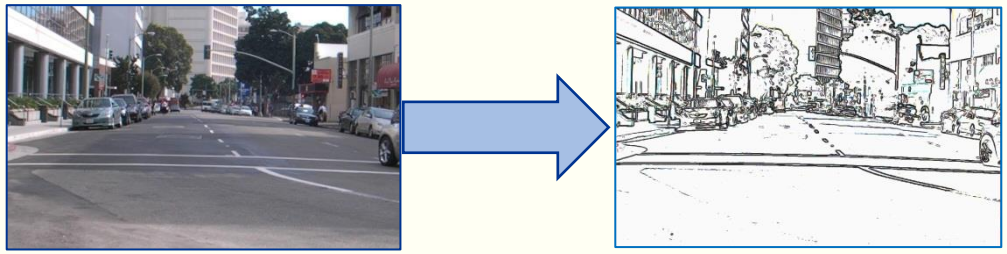
The "textbook" solution:

- Acquire road images from front-facing camera (often with fish-eye lens)
- Apply pre-processing (primarily lens correction)
- Perform edge detection
- Detect lines in the image with Hough transform
- Determine which lines are lane markings
- Track lane markings and estimate positions in the next frame
- Assess car's trajectory with respect to lane and warn driver in case of lane departure

Lane-Departure Warning: Edge Detection

In a lane-departure warning system, edge detection is the first step in detecting lines (which may correspond to lane markings)

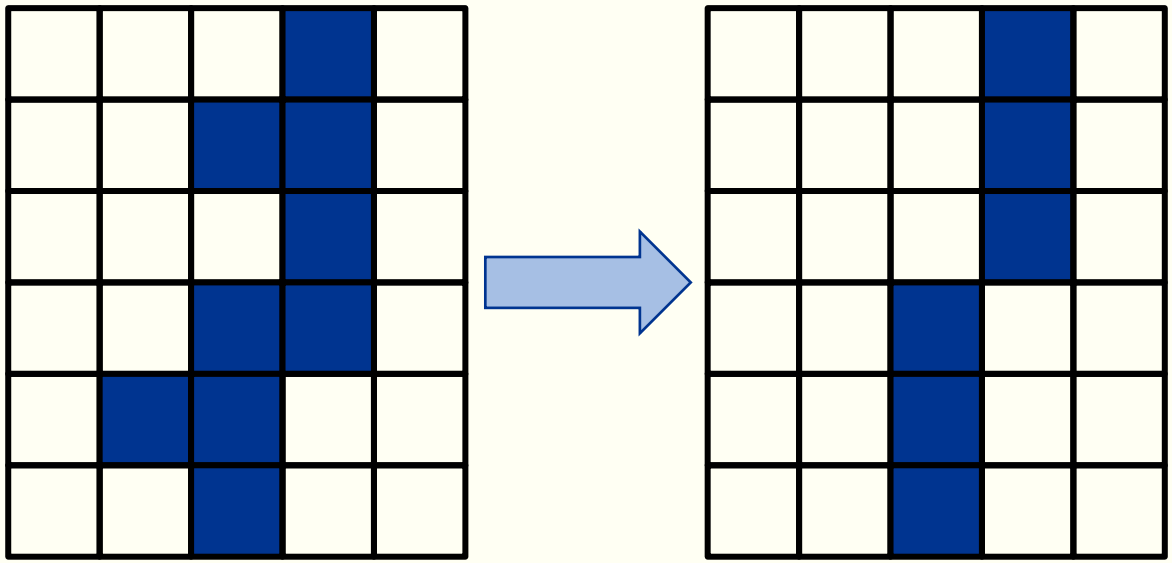
- Edge detection is a well-understood technique
- Primarily comprises 2D FIR filtering
- Computationally-intensive pixel processing
- Many algorithms are available (Canny, Sobel, etc.)
  - Some algorithms are highly parallel and generally good candidates for FPGA implementation
  - Others (e.g. Canny) include steps such as edge-tracing that are a poor match for FPGA implementation



Lane-Departure Warning: Edge Thinning

Edge thinning removes unwanted spurious edge pixels

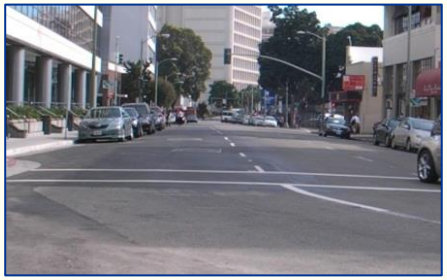
- Improves output of Hough transform
- Often performed in multiple passes over the frame
- Also useful in other applications



Lane-Departure Warning: Hough Transform

- Hough transform examines the edge pixels found in the image, and detects predefined shapes (typically lines or circles).
- In a lane-departure warning system, Hough transform is used to detect lines, which may correspond to lane markings on the road.

Original image

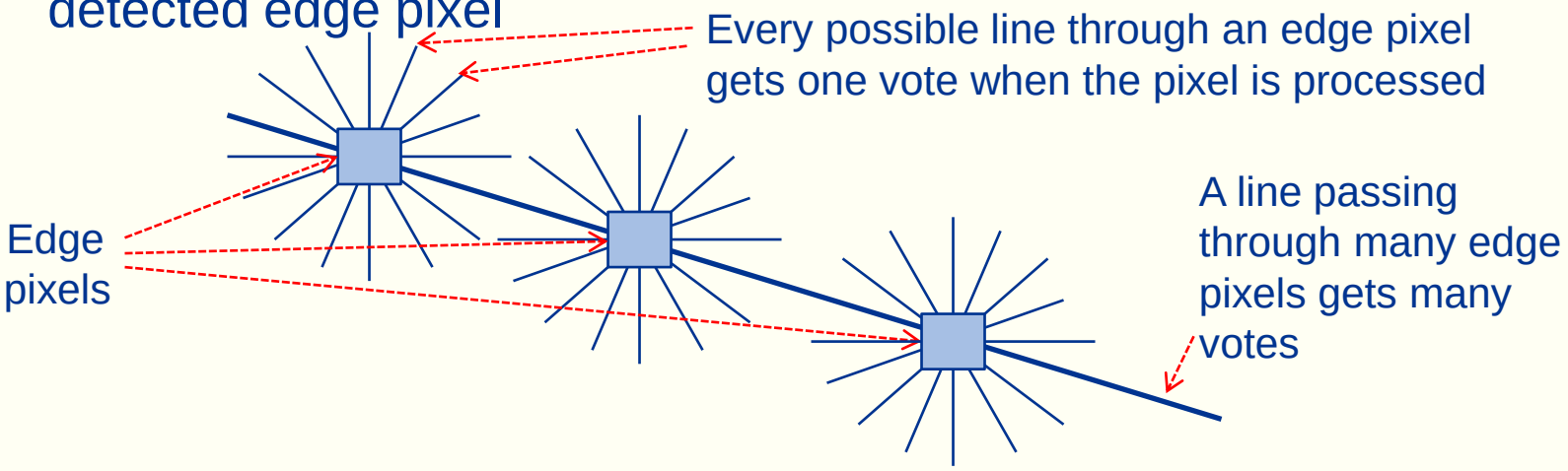


Edges detected

Lines detected

Lane-Departure Warning: Hough Transform

- Similar to a histogram
  - Each detected edge pixel is a “vote” for all of the lines that pass through the pixel’s position in the frame
  - Lines with the most “votes” are detected in the image
  - Uses a quantized line-parameter space (e.g. angle and distance from origin)
  - Must compute all possible line-parameter values for each detected edge pixel

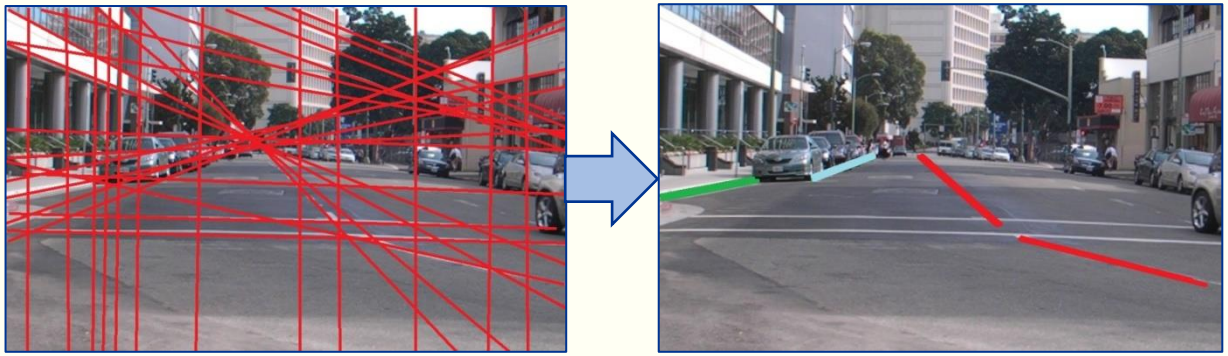


Lane-Departure Warning: Detecting Lane Markings

Filter the detected lines to discard lines that are not likely to be lane markings

- Find start and end points of line segments
- Filter by length, position, and angle
- Filter by line color and background color
- Additional heuristics may apply (e.g. dashed or solid lines are likely to be lane markings, but lines with uneven gaps are not)

Possibly classify the lines as lane markings or other lane indication (e.g. curb)



Lane-Departure Warning: Tracking Lane Markings

Tracking lane markers from each frame to the next

- Helps eliminate spurious errors
- Provides a measure of the car's trajectory with respect to the lane

Typically done using a predictive filter:

- Predict new positions of lane markings in the current frame
- Match the lane markings to the predicted positions and compute the prediction error
- Update the predictor for future frames

Kalman filters are often used for prediction in vision applications

- Theoretically these are the fastest-converging filters
- Found in OpenCV
- Simpler filters are often sufficient

Very low computational demand due to low data rates

- E.g. 2 lane marking positions × 30 fps = 60 samples per second

Lane-Departure Warning: Challenges

- The basic algorithm presented is not robust. May need significant enhancements for real-world conditions
  - Must work properly on curved roads
  - Must handle diverse conditions (e.g. glare on wet road at night)
- Integration with other automotive safety functions

PROCESSORS FOR EMBEDDED VISION

High-performance Embedded CPUs

Though challenged with respect to performance and efficiency, unaided high-performance embedded CPUs are attractive for some vision applications

- Vision algorithms are initially developed on PCs with general-purpose CPUs
- CPUs are easiest to use: tools, operating systems, middleware, etc.
- Most systems need a CPU for other tasks

However:

- Performance and/or efficiency is often inadequate
- Memory bandwidth is a common bottleneck

Example: Intel Atom (used in NI 1772C Smart Camera)

Best for: Applications with modest performance needs

Application-Specific Standard Product + CPU

Application-specific standard products (ASSPs) are specialized, highly integrated chips tailored for specific applications or application sets

- ASSPs may incorporate a CPU, or use a separate CPU chip
- By virtue of specialization, they tend to deliver superior cost- and energy-efficiency
- They usually include strong application-specific software development infrastructure and/or application software

However:

- The specialization may not be right for your particular application
- They may come from small suppliers, which can mean more risk
- They use unique architectures, which can make programming them, and migration to other solutions, more difficult
- Some are not user-programmable

Example: PrimeSense PS1080-A2 (used in Kinect)

Best for: Ultra-high-volume, low-cost applications

Graphics Processing Unit (GPU) + CPU

GPUs, mainly used for 3-d graphics, are increasingly capable of being used for other functions

- Referred to as “general-purpose GPU” or “GPGPU”
  - 👍 Often used for vision algorithm development
  - 👍 Widely available; easy to get started with parallel programming
  - 👍 Well-integrated with CPU (sometimes on one chip)
  - 👎 Typically cannot be purchased as a chip, only as a board, with limited selection of CPUs
  - 👎 Low-cost, low-power GPUs (designed for smart phones, tablets) are not GPGPUs

Example: NVIDIA GT240 (used in GE NP240 rugged single-board computer)
Best for: Performance-hungry apps with generous size/power/cost budgets

DSP Processor + Co-processors + CPU

Digital signal processors ("DSP processors" or "DSPs") are processors specialized for signal processing algorithms

- This makes them more efficient than CPUs for the kinds of signal processing tasks that are at the heart of vision applications
- DSPs are relatively mature and easy to use compared to other kinds of parallel processors

However:

- DSPs often lack sufficient performance, and aren't as easy to use as CPUs
- Hence, DSPs are often augmented with specialized co-processors and a CPU on the same chip

Example: Texas Instruments DaVinci (used in Archerfish Solo consumer smart surveillance camera)

Best for: Apps with moderate performance needs and moderate size/power/cost budgets

Mobile “Application Processor”

A mobile “application processor” is a highly integrated system-on-chip, typically designed primarily for smart phones but also used for other applications

- Typically comprise a high-performance CPU core and a constellation of specialized co-processors: GPU, VPU, 2-d graphics, image acquisition, etc.
- 👍 Energy efficient
- 👍 Often have strong development support, including low-cost development boards, Linux/Android ports, etc.

However:

- 👎 Specialized co-processors are usually not user-programmable

Example: Qualcomm QSD8650 (used in HTC Incredible)

Best for: Apps with moderate performance needs, wireless connectivity, and tight size/power/cost budgets

FPGA + CPU

FPGA flexibility is very valuable for embedded vision applications

- Enables custom specialization and enormous parallelism
- Enables selection of I/O interfaces and on-chip peripherals

However:

- FPGA design is hardware design, typically done at a low level (register transfer level)
- Ease of use improving due to:
  - Platforms
  - IP block libraries
  - Emerging high-level synthesis tools
- Low-performance CPUs can be implemented in the FPGA; high-performance integrated CPUs on the horizon

Example: Xilinx Spartan-3 XC3S4000 (used in Eutecus Bi-i V301HD intelligent camera)

Best for: High performance needs with tight size/power/cost budgets

DEVELOPMENT AND TOOLS

Embedded Vision System Development Challenges

Developing embedded vision systems is challenging

- Vision is a system-level problem: success depends on numerous elements working together, besides the vision algorithms themselves:
  - Lighting, optics, image sensors, image pre-processing, etc.
  - Getting these elements working together requires multi-disciplinary expertise
- Many computer vision experts know little about embedded systems, and many embedded system designers know little about vision
  - Many projects die in the chasm between these groups
- There are numerous algorithms available, but picking the best and ensuring that they meet application requirements can be very difficult
- Vision often uses complex, computationally demanding algorithms; implementing these under severe cost, size, and energy constraints requires selecting the right processor for the job
  - Expect to optimize algorithm implementations for the processor

The PC is Your Friend

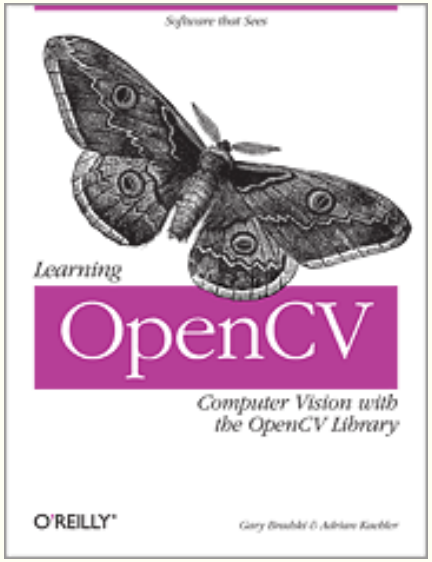
- Most embedded vision systems—and virtually all vision algorithms—begin life on a personal computer
- The PC is a fabulous platform for research and prototyping:
  - Ubiquitous
  - Inexpensive
  - Outstanding development infrastructure:
    - Generic software tools, libraries
    - Vision-specific libraries
    - Domain-specific design, simulation tools
    - Example applications
  - Easy to integrate cameras, displays, networks, and other I/O
- One can begin implementing vision applications within a day of unpacking a new PC and webcam
- Parallel acceleration of vision algorithms can be done using GPGPUs

The PC is Your Foe

- The PC is not an ideal platform for implementing most embedded vision systems
- Although some applications can embed a PC, many cannot, due to cost, size, and power considerations
- PCs lack sufficient performance for many real-time vision applications
  - GPGPUs don't yet address embedded applications
- Many of the same tools and libraries that make it easy to develop vision algorithms and applications on the PC also make it difficult to create efficient embedded implementations
  - Algorithm expert: "Here's my algorithm. It has 99% accuracy."
  - Embedded developer: "How is it coded? How does it perform?"
  - Algorithm expert: "It uses 85 MATLAB functions, 27 OpenCV functions, and double-precision floating-point. It runs at 1/20th of real-time on a 3 GHz quad-core workstation."
  - Embedded developer: "Just shoot me!"

OpenCV

- OpenCV is a free, open source computer vision software component library comprising over two thousand algorithms
  - Originally developed by Intel, now led by Willow Garage
- The OpenCV library, used along with Bradski and Kahler's book, is a great way to quickly begin experimenting with computer vision
- However:
  - Some OpenCV functions work better than others
  - OpenCV is a library, not a standard
  - OpenCV is not ideally suited to embedded implementation
- Ports to non-PC platforms have been made, and more are underway, but there's little coherence to these efforts



CONCLUSIONS

Conclusions

To date, embedded computer vision has largely been limited to low-profile applications like surveillance and industrial inspection

Thanks to the emergence of high-performance, low-cost, energy efficient programmable processors, this is changing

In the coming years, embedded vision will change our industry

Embedded vision technology will rapidly proliferate into many markets, creating opportunities for chip, equipment, algorithm, and services companies

But implementing embedded vision applications is challenging, and there is limited know-how in industry

Conclusions (cont'd)

Don't go it alone! Re-use what you can:

- Algorithms
- Cameras
- Software libraries
- Application platforms
- Lessons learned

Be realistic!

- Recognize that vision is a system-level problem
- Accept that many vision problems are hard; if the application requires perfect vision performance, it may never succeed
- Expect challenges in implementing vision within embedded cost, size, and power budgets

RESOURCES

Embedded Vision Summit West
Santa Clara Convention Center—Thursday, May 29th

- Practical knowledge in the design of systems and applications that use embedded vision
  - Focus on recognition and autonomy, with presentations on:
    - Algorithms
    - Applications
    - Implementation challenges and techniques
  - Keynotes by industry luminaries from Google and Facebook
  - Demos of vision technology and opportunities to meet with leading technology suppliers
- Hands-on workshops held on Wednesday, May 28th
- Co-located with Augmented World Expo at the Santa Clara Convention Center
- For more info, visit www.Embedded-Vision.com

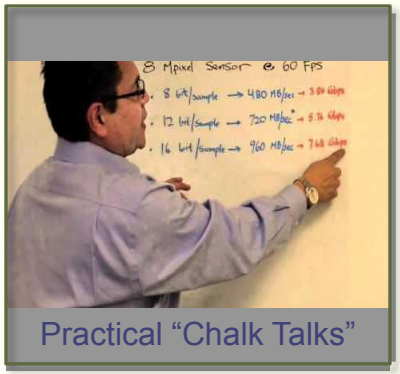
Selected Resources: The Embedded Vision Alliance

The Embedded Vision Alliance web site, at www.Embedded-Vision.com, provides free, high-quality technical educational resources for engineers

Register on the Alliance web site for free access to:

- The Embedded Vision Academy—in-depth tutorial articles, video "chalk talks," code examples, and discussion forums.
- Embedded Vision Insights—bimonthly newsletter with industry news and updates on new resources available on the Alliance website.


Luminary Interviews


Practical "Chalk Talks"


Hands-On Workshops

Selected Resources

- OpenCV:
  - http://opencv.willowgarage.com/wiki/
  - Bradski and Kaehler, “Learning OpenCV: Computer Vision with the OpenCV Library”, O’Reilly, 2008
- MATLAB/Octave:
  - “Machine Vision Toolbox”, P.I. Corke, IEEE Robotics and Automation Magazine, 12(4), pp. 16-25, November 2005. http://petercorke.com/Machine_Vision_Toolbox.html
  - P. D. Kovesi. “MATLAB and Octave Functions for Computer Vision and Image Processing.” Centre for Exploration Targeting, School of Earth and Environment, The University of Western Australia. http://www.csse.uwa.edu.au/~pk/research/matlabfns.
- Visym (beta): http://beta.visym.com/overview

Selected Resources

- "Predator" self-learning object tracking algorithm:
  - Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures," International Conference on Pattern Recognition, 2010, pp. 23-26
  - http://info.ee.surrey.ac.uk/Personal/Z.Kalal/
- Vision on GPUs: GPU4vision project, TU Graz: http://gpu4vision.icg.tugraz.at
- Lens distortion correction:
  - Luis Alvarez, Luis Gomez and J. Rafael Sendra. "Algebraic Lens Distortion Model Estimation." Image Processing On Line, 2010. DOI:10.5201/ipol.2010.ags-alde:

    http://www.ipol.im/pub/algo/ags_algebraic_lens_distortion_estimation/

Additional Resources

BDTI's web site, www.BDTI.com, provides a variety of free information on processors used in vision applications.

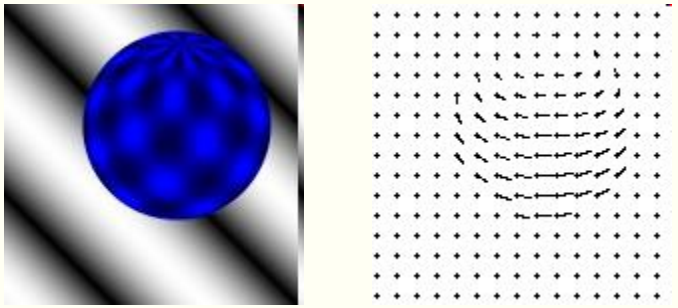
BDTI's free "InsideDSP" email newsletter covers tools, chips, and other technologies for embedded vision and other DSP applications. Sign up at www.BDTI.com.

The "Embedded Vision Insights" newsletter showcases tutorials, interviews, and other videos, along with technical articles, industry analysis reports, news write-ups, and forum discussions that have recently appeared on the Embedded Vision Alliance's website. Sign up at http://www.embedded-vision.com/user/register.

BACK-UP MATERIAL

Dense Optical Flow—The Problem

- Estimate the pattern of apparent motion of objects, surfaces, and edges in a visual scene
- Typically results in a motion vector for each pixel position in a video frame



Rotating sphere and corresponding optical flow field
(Images from http://of-eval.sourceforge.net/)

Used in vision applications

- To estimate observer and object positions and motion in 3D space
- To estimate image registration for super-resolution and noise reduction algorithms

Dense Optical Flow—Challenges and Trade-offs

- Optical flow can't be computed without making some assumptions about the video content (this is known as the *aperture problem*)
- Different algorithms make different assumptions
  - E.g. constant illumination, smooth motion
- Many algorithms exist, roughly divided into the following classes:
  - **Block-based methods** (similar to motion estimation in video compression codecs)
  - **Differential methods** (Lucas-Kanade, Horn-Schunck, Buxton-Buxton, and variations)
  - **Other methods** (Discrete optimization, phase correlation)
- Aliasing can occur
  - E.g. when an object in the scene has a repeating texture pattern, or when motion exceeds algorithmic constraints
- Some algorithms are sensitive to camera noise
- Most algorithms are computationally intensive

Dense Optical Flow—A Solution

Lucas-Kanade method with image pyramid is a popular solution

- Lucas-Kanade method is a differential method of estimating optical flow; it is simple but has significant limitations
  - Assumes constant illumination and constant motion in a small neighborhood around the pixel-position of interest
  - Limited to very small velocity vectors (less than one pixel per frame)
- Image pyramids extend Lucas-Kanade to support greater motion
  - Original frames are sub-sampled to create several pyramid levels
  - Lucas-Kanade method is used at the top level (lowest resolution) yielding a coarse estimate, but supporting greater motion
  - Lucas-Kanade is used again at lower levels (higher resolution) to refine the optical flow estimate

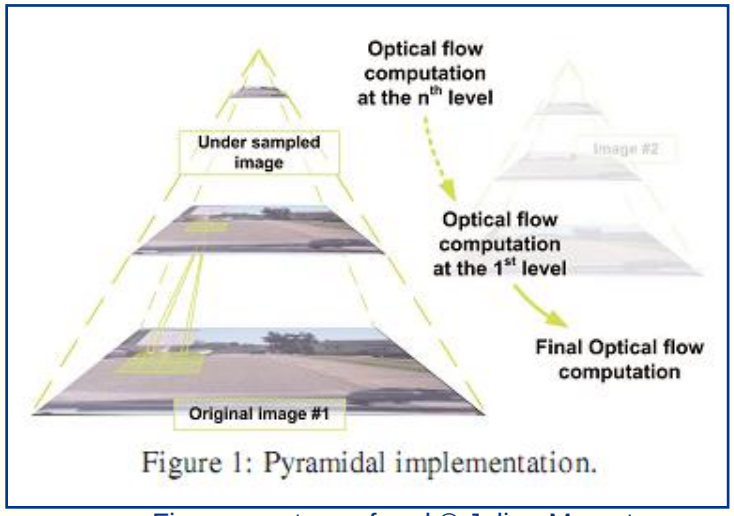


Figure courtesy of and © Julien Marzat

Pedestrian Detection—The Problem

- Surveillance/monitoring applications
  - Unauthorized access to restricted areas
  - People tracking in a given area
- Automotive safety applications
  - Pedestrian detection in the path of the vehicle
  - Must distinguish between pedestrians and other moving objects
- High accuracy is required
  - To avoid missed alarms (under-sensitive detector)
  - To avoid false alarms (over-sensitive detector)
- Real-time processing
  - Low latency is required (quick response is desired)

Pedestrian Detection—Example

- Detecting pedestrians at a stop sign

Pedestrian Detection—Challenges and Trade-offs (1)

- Challenges:
  - Detecting the relative motion of pedestrians against a moving background
  - Pedestrians come in many sizes, shapes, and costumes
  - Pedestrians sometimes move in erratic ways
  - Pedestrians frequently travel in groups
  - The camera's view of a pedestrian may become occluded
  - Computationally intensive: can reach hundreds of GOPS

Pedestrian Detection—Challenges and Trade-offs (2)

- Trade-offs:
  - Fixed camera view; limiting application scope for less computation
  - Detection based on motion of vertical edges rather than colors; limiting identification and tracking capabilities for less computation
  - Object classification based on aspect ratio; identification of individuals rather than groups, thus filtering out non-pedestrian size/shape moving objects
  - Tracking based on confidence level; improved tracking of occluded objects at the expense of detection latency. This also compensates for some of the erratic human behavior such as sudden stops

Pedestrian Detection—A Solution