Low-Cost Wireless Image Sensor Networks for Visual Surveillance and Intrusion Detection

IEEE ICNSC’15
Taipei, Taiwan,
April 10th, 2015
Time for presentation

is about 20 minutes
VISUAL DATA FOR SITUATION-AWARENESS

Collect data to improve the responsiveness of rescue operations
Visual/image sensors

- Cyclops
- SeedEyes
- CMUcam4 & CMUcam5 (PIXY)
- iMote2/IMB400...

- Mostly based on ad-hoc development of the visual part (dedicated camera circuit or dedicated μC for image acquisition/processing)
- Image encoding mechanism rarely adapted to low-resource platform (memory, radio,...)
- Can hardly run out-of-the-box for surveillance applications
Motivations & Objectives

- Offer an off-the-shelf solution so that anybody can reproduce the hardware and software components
  - Arduino-based solution for maximum flexibility and simplicity in programming and design;
  - Simple, affordable external camera to get raw image data
- Integrate and apply fast and efficient compression scheme with the host μC (no additional nor dedicated μC)
  - Small size image
  - Packet loss-tolerant bit stream
Arduino + uCamII
128x128 8bpp RAW IMAGES

Arduino Due board, AT91SAM3X8E at 84MHz, 96KB SRAM, 512KB flash
Our image sensor

Red led, indicates that uCam is taking picture and encoding is undergoing

Green led, indicates that uCam is ready

XBee 802.15.4 module connected at 125000 bauds

uCamII camera, connected 115200 bauds

uCamII configured for RAW 128x128 8-bit/pixel gray scale

Arduino Due board, AT91SAM3X8E at 84MHz, 96KB SRAM, 512KB flash

Arduino MEGA2560 board, ATmega2560 at 16MHz, 8KB SRAM, 256KB flash
ADJUSTABLE IMAGE QUALITY FACTOR Q

raw 16384b  Q=100; 9768b (1.67)  Q=80; 3729b (4.4)
PSNR=51.344  PSNR=29.414  PSNR=28.866
Q=90; 5125b (3.2)

Q=70; 2957b (5.5)  Q=60; 2552b (6.4)  Q=40; 2024b (8.1)
PSNR=28.477  PSNR=28.024  PSNR=27.423
Q=50; 2265b (7.2)

PSNR=26.038  PSNR=25.283  PSNR=23.507
Q=40; 2024b (8.1)

Q=30; 1735b (9.5)  Q=20; 1366b (12)
PSNR=26.933  PSNR=26.038
Q=10; 911b (18)

PSNR=25.283  PSNR=23.507
Q=5; 576b (28.44)
PACKET LOSS-TOLERANT BIT STREAM, ANY RECEPTION ORDER

Q=50; 10% pkt losses  Q=50; 20% pkt losses  Q=50; 30% pkt losses

Q=50; 40% pkt losses  Q=50; 50% pkt losses  Q=50; 60% pkt losses

Scientific cooperation with V. Lecuire from CRAN laboratory for the optimized image encoding algorithm
IMAGE CHANGE DETECTION FOR INTRUSION DETECTION

Sends image to gateway on intrusion detection

Real-time synchronization with your smartphone through cloud applications, e.g. Dropbox

Very lightweight "simple-differencing" method, takes into account modification in image luminosity
Very low-memory platforms

Arduino MEGA2560 at 16MHz, 8KB SRAM

Only 2KB SRAM available at runtime

Modified encoding algorithm to avoid having all the raw image in SRAM: encoding, packetization and transmission in a row per image packet

Reference image and current raw stored in an SD card

Encoding and packetization will read image blocks from SD card

Arduino MEGA2560 board, ATmega2560 at 16MHz, 8KB SRAM, 256KB flash
Multi-camera system
LOW-COST OMNIDIRECTIONAL VISUAL SENSING
## Performance Measures

**Arduino Due**

<table>
<thead>
<tr>
<th>Quality Factor Q</th>
<th>size in bytes (MSS=90)</th>
<th>Number of packets</th>
<th>time to read data from ucam (measured)</th>
<th>global encode + pkt time (measured)</th>
<th>global transmit time (computed)</th>
<th>transmit time/pkt (computed)</th>
<th>global encode + pkt + transmit time (measured)</th>
<th>encode+transmit time/pkt (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>9768</td>
<td>158</td>
<td>1.512</td>
<td>1.027</td>
<td>1.064</td>
<td>0.0067</td>
<td>2.091</td>
<td>13.2342</td>
</tr>
<tr>
<td>90</td>
<td>5125</td>
<td>70</td>
<td>1.512</td>
<td>0.782</td>
<td>0.539</td>
<td>0.0077</td>
<td>1.321</td>
<td>18.8714</td>
</tr>
<tr>
<td>80</td>
<td>3729</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>2957</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>2552</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>2265</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2024</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1735</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1366</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>911</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>576</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Graph:**

- Blue: global transmit time (computed)
- Red: global encode + pkt time (measured)

**Quality factor range:** 100 to 5
Detailed measures

\[
F \\
G \\
H = F + G \times N \\
I = H + C \times N \\
J \\
K = (C + G) \times N
\]

Comparison of encode + pkt
- \(\text{encode + pkt time (computed)}\)
- \(\text{global encode + pkt time (measured)}\)

Comparison of encode + pkt + transmit
- \(\text{encode + pkt + transmit time (computed)}\)
- \(\text{global encode + pkt + transmit time (measured)}\)

Comparison of RCV time & 1-hop latency
- \(\text{RCV time (computed)}\)
- \(\text{RCV time (measured)}\)
- \(\text{1-hop latency (snapshot-display)}\)

\[
R + D + J = 1.512 + 0.879 + 0.349 = 2.74s
\]
Out-of-the-box surveillance!
Using realistic parameters in simulation

- 128x128 8bpp encoded image
- Quality Factor of 50: encoded image of 2265 bytes in 28 packets
- We need 200ms to configure the camera. Time before image data can be processed is 1.512s
- So the maximum frame capture rate is 0.58fps
- Camera angle of view could be 56°, 76° or 116°
- Depth of view of 25m
- Packet overhead at the image source is 11ms (8ms for transmission and 3ms for packetization)
- Packet relaying time is 16ms (based on measures of MicaZ platform)
The simulation model is used to study performance of large-scale intrusion detection system.

Using real measures for image processing tasks and packet transmission overheads produces very accurate simulation results that are consistent to what have been found in real multi-hop experimentations.

First intrusion seen by node 60, image packets are sent and relayed by node 93, then received and displayed by node 3 (sink) at time 86.8. 28/28 pkts, received latency is 0.42s and image was sent 0.46s earlier.
Conclusions

- Low-cost image sensor from off-the-shelves components with fast and packet loss-tolerant encoding
- Can run out-of-the box to perform surveillance tasks based on image change detection
- The image latency can be less than 2.3s using medium quality image
- At 1-hop, the receive & display latency can be less than 2.8s using medium quality image
- Detailed performance measures are used to produce more accurate large-scale simulation models
An image sensor board based on Arduino Due/MEGA and uCamII camera

last update March 30th, 2015.

Introduction

There are a number of image sensor boards available or proposed by the very active research community on image and visual sensors: Cyclops, MedEyez, Click, WiCam, SeeEyez, Panoptic, CMUcam1, CMUcam4, CMUcam2, CMUcam3, PVEye, Allsee3D, etc. All these platforms and products are very good and our motivations in building our own image sensor platform for research on image surveillance applications are:

1. to have an off-the-shelf solution so that anybody can reproduce the hardware and software
2. to use an Arduino-based solution for maximum flexibility and simplicity in programming and design
3. to use a compact and affordable external camera to get RAW image data, as JPEG
4. to use a fast and efficient compression scheme with the best microcontroller to produce robust and very small image data suitable for large scale surveillance or search/detection/surveillance awareness applications
5. to use a simple and flexible strategy to implement our criticality-based image sensor scheduling proposition.

2. the integration with our vision for enabling data-intensive communication with low-resource smart platforms (audio and image)
4. fully compatible with our vision for enabling data-intensive communication with low-resource smart platforms (audio and image)

Architecture and components

We use both Arduino Due and MEGA2560. The Arduino Due board has enough SRAM memory (64K) to store an 1280x128 8-bit/pixel RAW image (16144 bytes). On the MEGA2560, which has only 8K of SRAM memory, we store the captured image on an SD card (see right figure below for an example) and then perform the encoding process by incrementally reading small portions of the image file.

For the camera, we use the uCamII from 3D systems. You can download the reference manual from 3D systems web site. The uCamII can deliver 1280x128 raw image data. JPEG compression can be realized by the embedded microcontroller but this feature is not used as JPEG compression is not suitable at all for lowy environments. We instead apply a fast and efficient compression scheme with the best microcontroller to produce robust and very small image data.