SANDIA REPORT
SAND2008-5776
Unlimited Release
Printed August 2008

OpenCV and TYZX: Video Surveillance for Tracking

Eric Chu, Andrew Spencer, and Jim He

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online order:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2008-5776
Unlimited Release
Printed August 2008

OpenCV and TYZX: Video Surveillance for Tracking

Eric Chu
Sandia National Laboratories
P.O. Box 969
Livermore, CA. 94551-0969

Andrew Spencer
Sandia National Laboratories
P.O. Box 969
Livermore, CA. 94551-0969

Jim He
Sandia National Laboratories
P.O. Box 969
Livermore, CA. 94551-0969

Abstract

As part of the National Security Engineering Institute (NSEI) project, several sensors were developed in conjunction with an assessment algorithm. A camera system was developed in-house to track the locations of personnel within a secure room. In addition, a commercial, off-the-shelf (COTS) tracking system developed by TYZX was examined. TYZX is a Bay Area start-up that has developed its own tracking hardware and software which we use as COTS support for robust tracking. This report discusses the pros and cons of each camera system, how they work, a proposed data fusion method, and some visual results. Distributed, embedded image processing solutions show the most promise in their ability to track multiple targets in complex environments and in real-time. Future work on the camera system may include three-dimensional volumetric tracking by using multiple simple cameras, Kalman or particle filtering, automated camera calibration and registration, and gesture or path recognition.

Acknowledgements

Thanks to Davd Burnett, Donald Dowdle, Ann Lehman Harren, Harvey Ho, and Marisa Ruffolo for their advice, inspiration, and teamwork during the early phase of this project.

Acknowledgements go to Karen McWilliams for her publication support, and Doug Gehmlich for his support of the project.

Contents

Figures

Preface

During the summer of 2007, video cameras were employed to perform basic tracking and recognition tasks in a National Security Engineering Institute (NSEI) project. The surveillance system implemented then detected the presence of a person via facial recognition and tracked a color-coded asset. The computer vision algorithms were revisited and updated during 2008 to perform more sophisticated target tracking, where the targets were personnel operating in a cluttered room (i.e., a room with static occlusion and obstructions). An exploratory algorithm was simulated in late 2007 for tracking multiple people in an empty room, which was found to fail for cluttered scenes [9]. A second, brute-force approach developed in early 2008 yielded good track results, but was unable to maintain identity. Finally, a COTS solution was obtained in the spring of 2008. It was developed by TYZX, Inc. of Menlo Park, CA. It was tested and found to provide the most stable and robust solution to the tracking problem, giving real-time coordinate readings of people working in a room. Its embedded, distributed approach to machine vision provided better performance and accuracy in solving the tracking problem.

Summary

Video surveillance units are usually the first element of a security system. While they are the most intuitive to understand and can be programmed for many tasks, they also have many vulnerabilities, such as sensitivity to light levels and large computational requirements. In this document, we explore the simple task of tracking multiple persons in a room using two different methods: one, a centralized, software-based solution developed at Sandia by using an open-source computer vision library (OpenCV); and the other, a distributed, hardware-based solution through commercial hardware (TYZX).

In our analysis, we have found TYZX stereo-vision cameras to be most capable of basic object tracking in large areas. Its hardware-based solutions to machine vision problems provide real-time results and accuracy that can not be rivaled by smart software. Hardware-based approaches also provide scalable and distributed solutions to the tracking problem, distributing the computational power among many parallel processes. As of this report's time of writing, TYZX's solution gives track results in two dimensions: the x-, y-coordinates of a person from an overhead perspective. Future work may include three-dimensional (3D) volumetric tracking by obtaining stereo-camera development kits from TYZX or through multicamera registration [6]. While tracking a single person in an empty room is trivial for both the OpenCV and TYZX solution, further development may be necessary to maintain track of multiple people in more complex environments, including those with static occlusions (tables, chairs, bookshelves, etc.). Three-dimensional information will also facilitate gesture recognition for high-level task analysis. In conclusion, Sandia would benefit greatly by taking advantage of COTS hardware and open-source software, adapting them for use to solve national security problems.

Nomenclature

NSEI	National Security Engineering Institute, a summer institute designed to expose students to national security issues and solutions through engineering
OpenCV	an open-source computer vision library initially developed by Intel
TYZX	a stereo-vision based start-up in Menlo Park, CA; pronounced to sound like “physics”
Point Grey Research (PGR)	a Canadian video camera company that builds high-speed, high-resolution cameras for imaging
AVATAR	a machine learning algorithm employing decision trees developed by Philip Kegelmeyer of Sandia, CA
Fps	an acronym for “frames per second”
FOV	an acronym for “field of view,” representing the angles at which a camera can detect light
LOS	an acronym for “line of sight,” describing if an object can be seen by a camera

This page intentionally left blank.

1 Introduction

Video cameras are widely employed in security systems because they closely resemble the human sense of sight. However, video frames must be processed by a computer or custom hardware to provide information that rivals human vision. In designing a system for tracking personnel, the main concern is real-time processing (about 15 fps) and producing accurate track results relative to a ground truth. Because video cameras provide much information, a measured in megabits, image processing and computer vision algorithms often suffer from bloat and inelegance, resulting in slow computation. To provide real-time surveillance, efficient algorithms must be implemented and used (as in OpenCV) or a custom hardware solution must be pursued (as in TYZX). Section 2 describes one possible software implementation by using Point Grey Research (PGR) cameras and OpenCV. Section 3 roughly describes the TYZX implementation and its hardware.

In this document, we examine the two computer vision systems to track personnel as a case study in video surveillance. Our main criterion for evaluating the systems are how well the system tracks a single person, how well the system performs with multiple people, and, finally, how well it scales to larger environments. The first system is a Sandia design utilizing basic geometry and computer vision methods implemented in OpenCV to extract an estimate of location. The second system is a COTS design by TYZX using stereo vision and automatic camera calibration and registration to accomplish the task. The TYZX design is proprietary, but what is known about it is briefly discussed in Section 3.

Furthermore, estimates of position and velocity of tracked people provide valuable insight into their behavior and intent. These estimates can be analyzed to determine a person's trajectory and intent. Currently, AVATAR, a machine learning program developed by Philip Kegelmeyer of Sandia, is being used to learn and distinguish authorized from unauthorized behavior for various tasks [1].

2 Point Grey Research Cameras

The Sandia, OpenCV system is built upon eight Point Grey Research (PGR) cameras. Point Grey Research is a Canadian company, spun out of UBC (University of British Columbia), specializing in advanced digital camera technology for computer vision research. We use Point Grey's Dragonfly cameras for our system. Technical documents about the PGR Dragonflies can be found at PGR's website, although PGR is replacing the Dragonfly line with Dragonfly2's.

All eight cameras are linked via IEEE 1394a Firewire cables (maximum data speed of 400Mbps) to a single, Pentium dual-core Xeon machine with 2GB of RAM. We use the OpenCV library to process the frames. OpenCV is an open-source computer vision library written by Intel in C/C++ and released to the general public for computer vision programming. PGR's FlyCapture SDK—the software development kit provided by Point Grey—is used to interface with the cameras. PGR's MultiSync is used to synchronize the eight cameras. The firmware is updated to the latest version (at the time of writing)—v2.1 beta 22 (2.1.1.22)—using the UpdatorGUI, which can be found after logging into Point Grey's support section. The frames are captured by Point Grey's FlyCaptureSDK and then converted into a format useable by OpenCV for further processing. Figure 1 shows the hierarchy of hardware and software used to build the tracking system.

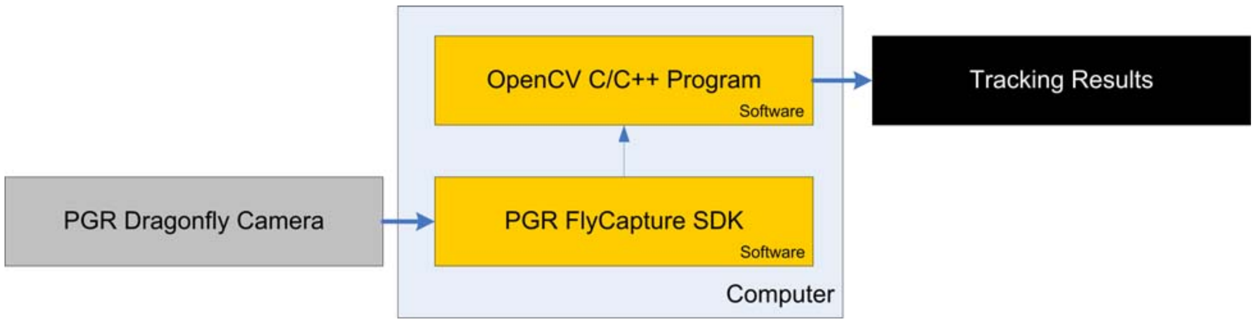


Figure 1. Software and hardware hierarchy to accomplish tracking task.

In addition, the Intel IPP (Integrated Performance Primitives) are used with PGR's FlyCapture SDK to speed up image acquisition. Intel's IPP is a low-level library designed specifically for Intel processors to perform many signal and image processing operations in special hardware rather than through the CPU, thus optimizing the execution of common linear algebra operations. There is also an option to use the IPP with OpenCV, though that has not been explored or implemented yet. The XVID codec was used for video compression and viewing.

Finally, the eight cameras are set up inside a demonstration room and connected to an 80/20 Industrial Erector Set for easy reconfiguration. They are pointed to the floor and positioned parallel to the ground. They are placed by hand to roughly maximize coverage around the server racks (see Figure 2).

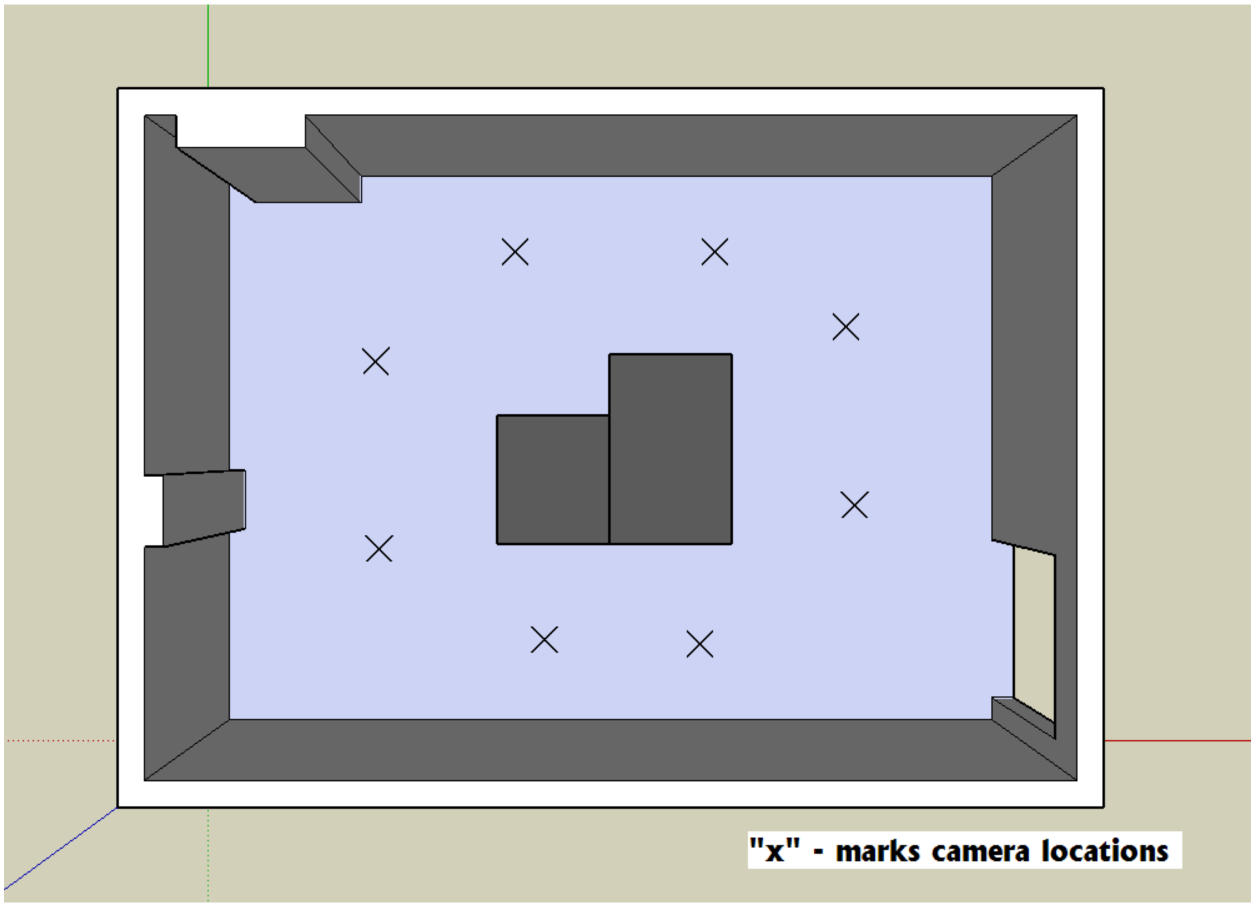


Figure 2. Room and camera layout.

2.1 Tracking Pipeline

Figure 3 shows the data flow from the camera frames to the resultant track. To acquire an accurate track, we first perform background subtraction on the individual videos. Then, we crudely stitch the foregrounds together, and track the foreground blobs. Blobs are a group of pixels in the foreground mask that represent the location of foreground objects. The blobs are tracked through a pre-existing OpenCV blob-tracking algorithm [2].

The following bullets describe the various components of the tracking pipeline shown in Figure 3.

- The “frames” are acquired from the video feed off one of the eight cameras.
- The “FG/BG Detector” separates foreground objects from background objects and produces a foreground mask for objects that are not part of the static background. See 2.2.
- The “Frame Stitching” algorithm is a crude geometric overlay of the images based on the physical location of the cameras. See 2.3.

- The “Blob Detection” algorithm detects when a new blob has appeared on the scene. See 2.4.
- The “Blob Tracking” algorithm tracks the blobs around the scene. See 2.4.
- The “Trajectory Post-Processing” algorithm is used to smooth out the tracking result, using a Kalman filter. See 2.4.
- “Blob Location and Velocity” represents the results of the current blob state for further analysis in the machine learning algorithm.

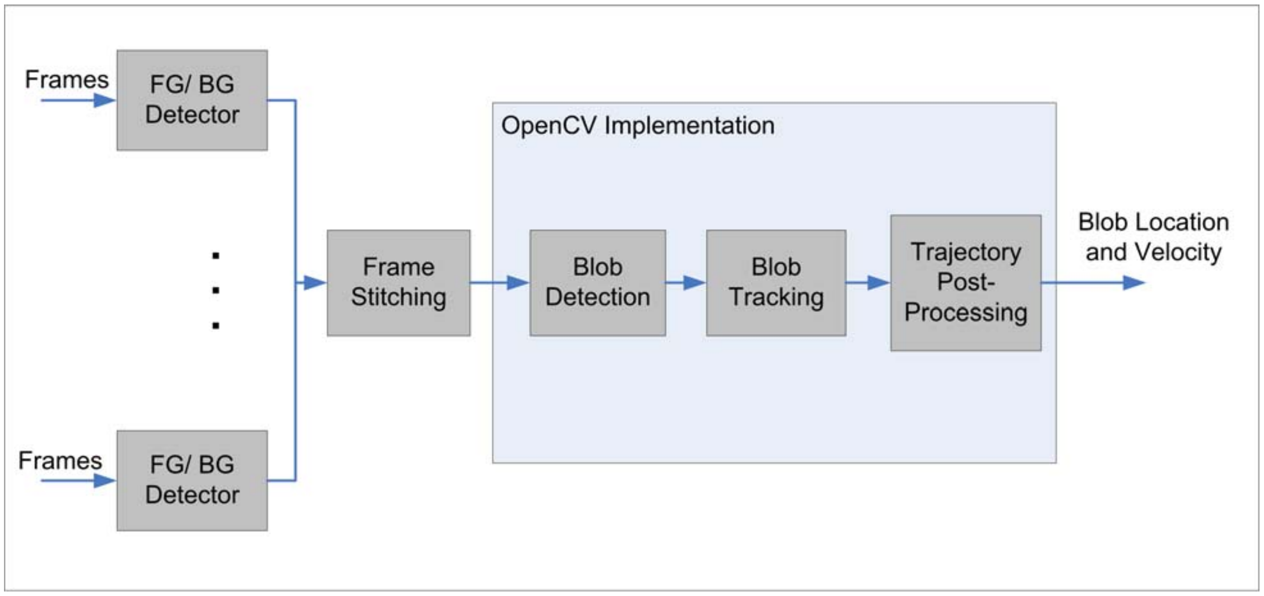


Figure 3. Flowchart of tracking algorithm.

2.2 Background-Foreground Segmentation

The background subtraction algorithm is described by Li [5]. It is CPU intensive to run the algorithm separately on all eight camera frames, so we start by tiling the eight videos into a 4x2 image array to make a single call to the background-foreground segmentation algorithm. The choice to tile them into a 4x2 array is arbitrary, but was done so we can construct a single frame from which to call the segmentation algorithm. (See Figure 4.) We use custom parameters that differ from Li’s implementation for our segmentation.

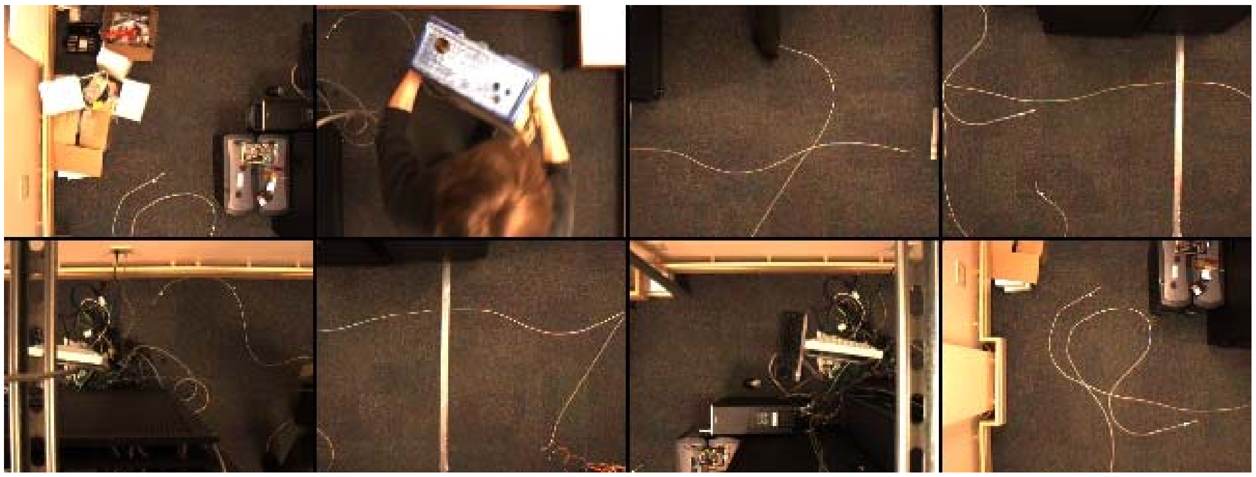


Figure 4. Tiled images shows how eight video views were arranged before background segmentation.

2.3 Foreground Stitching

After the foreground has been extracted, the foregrounds are re-stitched to resemble the shape of the room, and thus the blob's center of mass gives an approximation of personnel location. The stitching is done by manually measuring the physical locations of the camera relative to an origin and overlaying the video frames on top of each other. The foreground masks are OR-ed together to produce the desired result. Figure 5 shows the resulting stitch. The lower-left corner represents the origin at (0, 0). The stitching is hand-tweaked to produce good, visual results. A better stitching algorithm would be computationally more expensive, but provide more accurate means of tracking, since tracking is done on the OR-ed foreground masks. OpenCV's pre-packaged video surveillance algorithm [2] is used to track the blobs; it is described in Section 2.4. The tracked blob is marked with a green circle. Note the false track in Figure 5, which is a result of our personnel having moved the ladder from its original position in the background.

This algorithm is unable to distinguish between people and objects. It makes the naive assumption that all moving objects are human and that all static objects are background. Hence, if a background object has been moved, it will be thought of as a "person." The segmentation algorithm "learns" what pixels belong in the static background and compares the change in pixel intensities between frames. When the ladder is moved from its original location, the pixel values change dramatically. The carpet, then, is marked as a foreground object because of the change in pixel intensity. The tracking algorithm assumes this cluster of pixels is now a new "person" who has simply appeared in the scene and is standing still. After some time, the segmentation algorithm adapts the carpet into the background, and the track disappears.

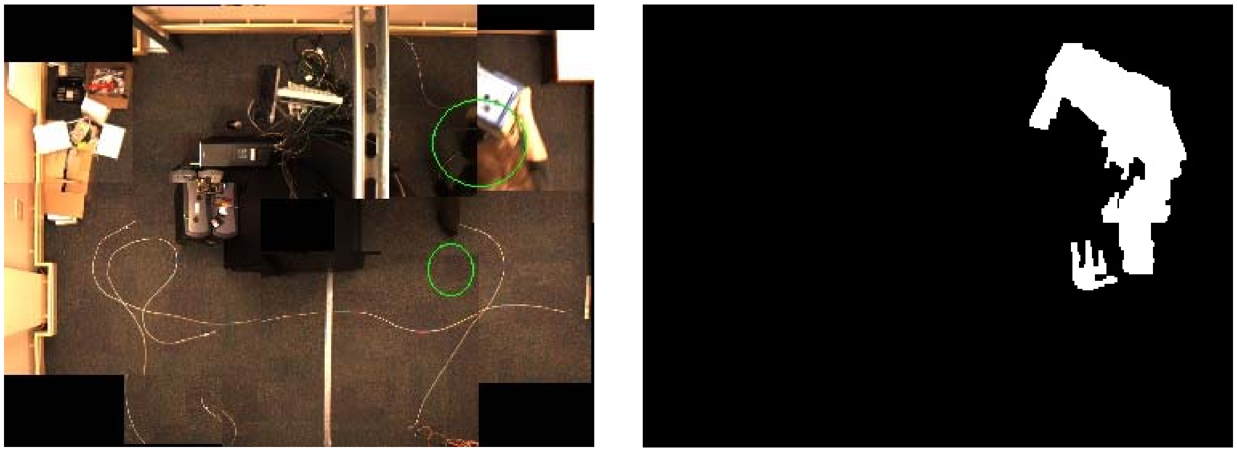


Figure 5. The image on the left shows tracking results. Note the false positive.The image on the right shows the results of stitching together the foreground using an OR operator on overlaps.

2.4 OpenCV Tracking Algorithm

The actual tracking algorithm is implemented in OpenCV [2]. Various methods are implemented to detect new blobs, and even more methods are used to track the blobs. More details about these methods can be found in the references listed in [2]. The code for tracking can be found in the blobtrack.cpp file under ...\OpenCV\samples\c. Details about the implementation can be found in the folder ...\OpenCV\cvaux\src\vs.

The current pipeline uses a simple method to detect new blobs in the foreground image, keeping track of the history of blob appearances. There is no collision detection done on the blobs (for tracking multiple people), since collision detection ends up slowing down the system. Tracking is done by finding connected components in sequential frames. The final path is Kalman filtered to produce the result (see [7] for information on Kalman filters). This pipeline is the simplest and produces fast results in real-time. To achieve better results, we would require more computational power.

Figure 6 shows the results of the overhead, PGR tracking system and gives a visual sense of the amount of error in the system. This system runs at 7.5 fps to give enough time to process the data between frames. The dots in Figure 6 show the sampled locations (at 7.5 times a second) of a single user walking along the ground truth path (the solid blue line). The person was instructed to walk around the room a couple of times, so the data covers multiple paths. Near the entryway, the error is large because the door is often tracked as a foreground object. Along the upper wall, the shadow cast by the user is tracked as the foreground object and the stitching is non-ideal, thus the track deviates greatly from the ground truth. The accuracy can be improved by utilizing a better stitching algorithm, though it may prove more computationally expensive.

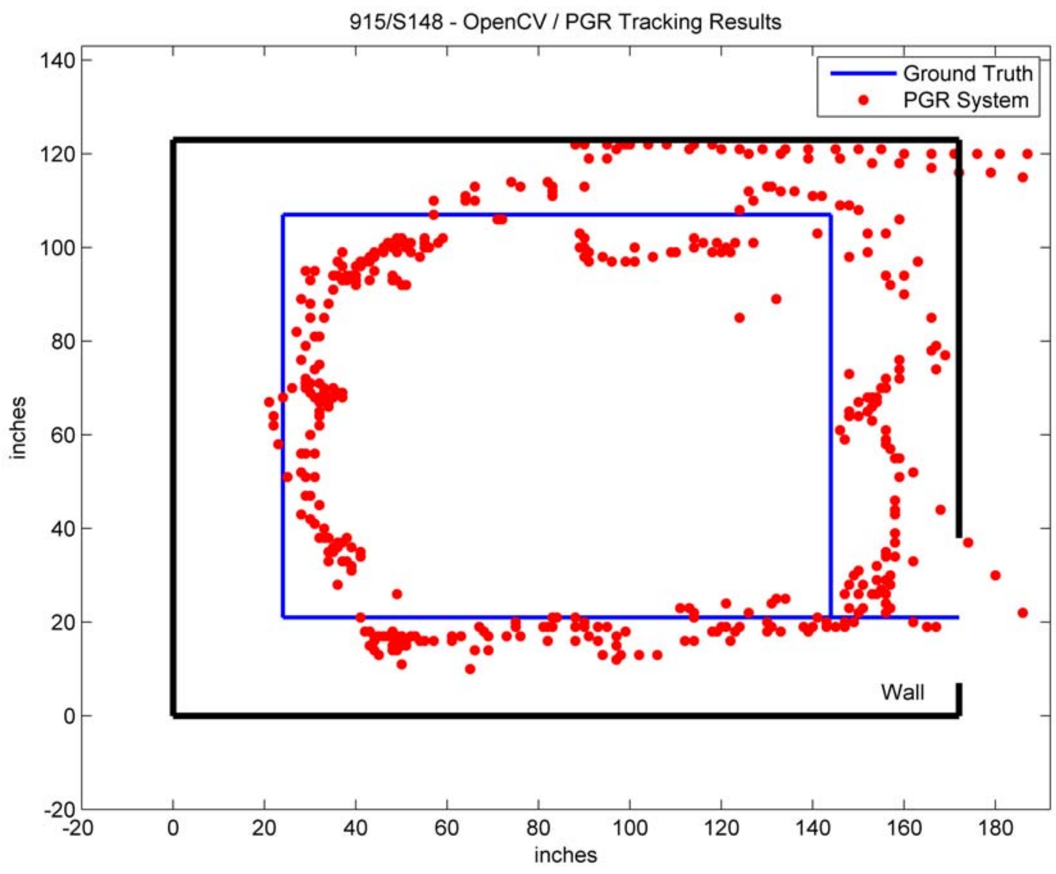


Figure 6. The results of tracking a single person walking along the ground truth path shown with a solid blue line. The dots mark the detected locations.

2.5 Known Issues

The OpenCV implementation is not without its limitations. Because the background segmentation algorithm is adaptive and relies on pixel values, colors similar to the carpet are not properly segmented. Skin tones are correctly separated from the ground, but dark-colored clothing can blend in to the background. Because of these missed positives in the segmentation algorithm, a single person may be split into separate tracks. Furthermore, because the segmentation algorithm is adaptive, a person that stands very still may become adapted into the background, and the track on him or her is lost.

Stitching the camera frames together as described in Section 2.3 introduces errors from the different fields of view. These errors are a function of the locations of the cameras and the distances between the object and each camera. The error can be reduced by camera calibration and registration, a step we highlight in Section 6, Future Work. These errors can also cause a single person to split along the seams of two images.

With multiple people in the scene, multiple blobs are tracked. When people brush by each other, their blobs merge. Since there is no collision detection and avoidance (see Section 2.4), multiple people may be tracked as one.

As with all camera systems, lighting changes drastically affect performance. Decreasing light levels decrease the contrast between foreground and background objects. Increasing the light levels in the room may cause spurious foreground objects to appear, since the pixel values suddenly appear brighter. This system is only acceptable for indoor tracking where indoor lighting is usually at a constant level, as opposed to outdoors where lighting levels vary with time of day.

Furthermore, the cameras are networked via Firewire directly into a single computer, passing megabytes of video data to the Firewire bus. To cover a larger physical space would require more cameras and more data bandwidth, which may not be possible with a Firewire bus. This design may work well for small rooms, but will not scale easily.

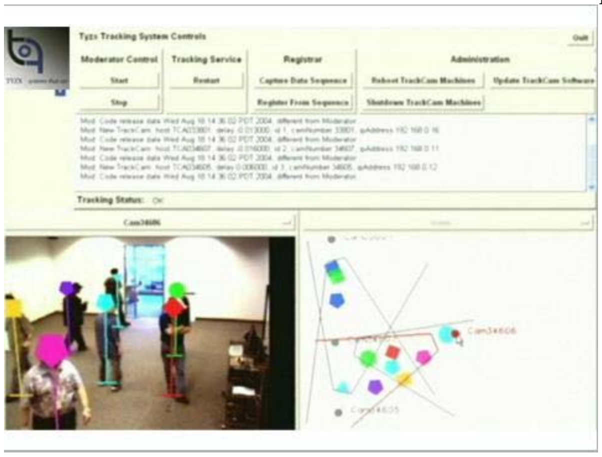
2.6 Summary

In this section, we described how to build a tracking system out of eight overhead cameras and demonstrated that it can produce a rudimentary track of personnel within a secure room. We also discussed limitations of the system. There is still room for improvement, as described in Future Work (Section 6). While this system is functional and provides a bare-bones implementation of a tracking algorithm for further research, the commercial TYZX system described in Section 3 provides better tracking results with fewer long-term costs.

3 TYZX

TYZX is a proprietary system developed by a start-up company of the same name in Menlo Park, CA. It is a hardware-based, embedded image processing / computer vision system. Using stereo vision (two cameras placed slightly apart from each other), it calculates the distance of a foreground object from the camera. It maps that data to an overhead perspective of the camera's field-of-view. With multiple cameras, such information can be combined to differentiate and track individuals working around the room. Figure 7 shows the robustness of the TYZX tracking system, capable of tracking a large group of individuals in a crowded environment. Figure 7 also shows TYZX tracking two people, as if tracking a two-person security team.

From these images, we can see that TYZX provides a very accurate method to track multiple people in the face of occlusion for complicated environments. [3] and [8] provide the technical details about the TYZX cameras and various applications for 3D tracking.



```
Msd: Code release date Wed Aug 18 14:36:02 PDT 2004, different from Moderator
Msd: New TrackCam: host TCA033801, delay -0.013000, id 1, camNumber 33801, ipAddress 192.168.0.16
Msd: Code release date Wed Aug 18 14:36:02 PDT 2004, different from Moderator
Msd: New TrackCam: host TCA034607, delay -0.016000, id 2, camNumber 34607, ipAddress 192.168.0.11
Msd: Code release date Wed Aug 18 14:36:02 PDT 2004, different from Moderator
Msd: New TrackCam: host TCA034606, delay 0.006000, id 3, camNumber 34606, ipAddress 192.168.0.12
Msd: Code release date Wed Aug 18 14:36:02 PDT 2004, different from Moderator
```

Tracking Status: OK

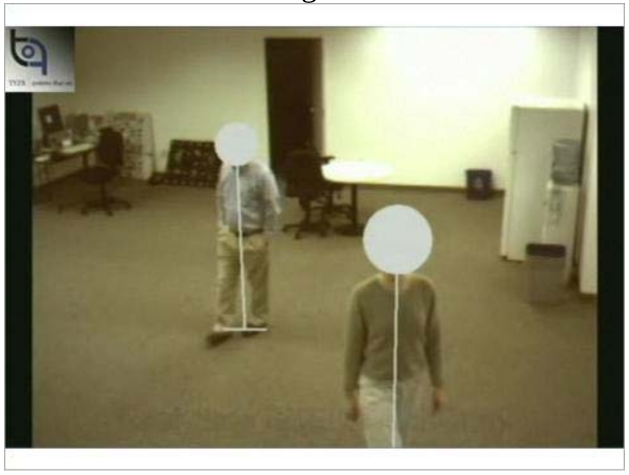


Figure 7. A screen capture from a TYZX video surveillance demonstration. On left, TYZX is capable of tracking multiple people in a busy room. On right, we see TYZX maintain track of a two-person team.

3.1 How TYZX Works

The TYZX tracking system relies fundamentally on stereo vision, emulating the physical architecture of human vision. With two cameras placed a known distance apart from each other, there is considerable overlap in the two fields of view. Because the cameras see the same scene from a slightly different perspective, certain objects in the scene will appear to be shifted: the shift is proportional to how far away the object is from the cameras. Using two cameras, then, one can accomplish depth perception without relying on pixel values, as shown in Figure 8.

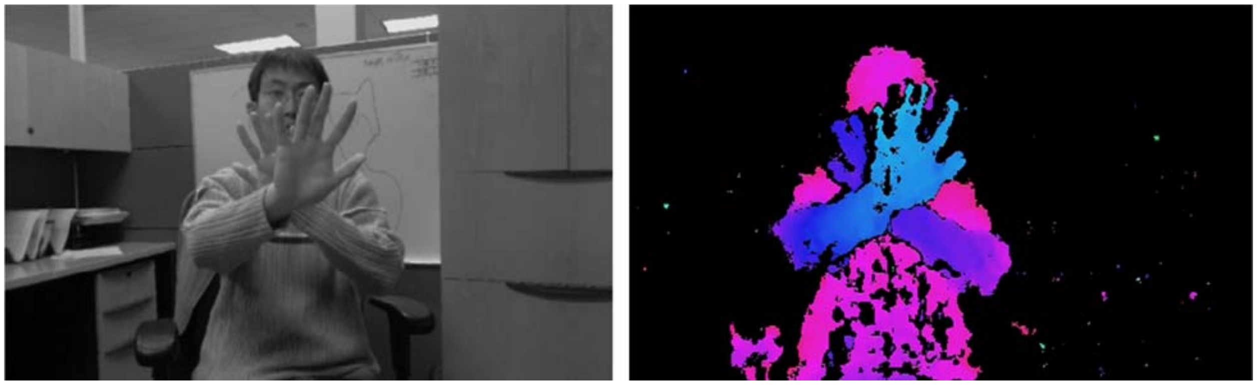
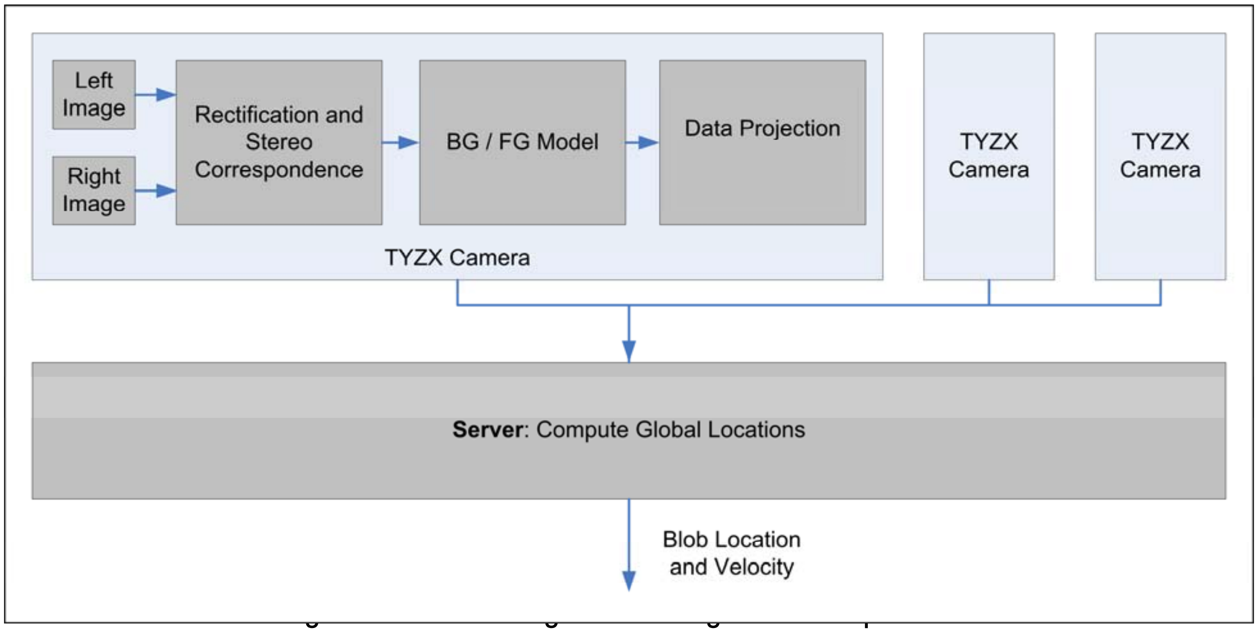


Figure 8. On the left is a scene and the right shows the depth map. Images taken from [3].

The depth values can then be used to track targets in three dimensions (3D). Height thresholds can be applied to exclude objects that are too short, and distance thresholds can be used to exclude walls or static barriers.

TYZX's image processing pipeline is shown in Figure 9; compare it to Figure 3. Note that in the PGR / OpenCV implementation, only the background / foreground segmentation algorithm was theoretically offloaded to the camera. A dedicated server polled all eight frames and computed the proper track through the OpenCV software. In TYZX's implementation, computation is distributed among the cameras. Background models are built by learning how far away each pixel is from the camera (a depth map) instead of the pixel's color value. Foreground objects are segmented by marking pixels that differ in depth from the background model. Each camera finds these foreground objects (in 3D), projects them into the world (room) coordinate system, and sends the height coordinate of the head back to the server. The server provides a central clock to synchronize data among the distributed cameras. It also collates and fuses the data from the cameras to produce a single reading. Since the hard work is distributed among the cameras, TYZX is more efficient at tracking people and more scalable (i.e., network traffic is smaller). Details about the TYZX design can be culled from [8].


Left Image
Right Image
Rectification and Stereo Correspondence
BG / FG Model
Data Projection
TYZX Camera
TYZX Camera
TYZX Camera
Server: Compute Global Locations
Blob Location and Velocity

the person-tracking task. Compare to Figure 3.

3.2 Tracking System

To set up TYZX, we place the cameras so that they have some amount of overlap in their fields of view and can provide maximal coverage for the demonstration room. The cameras are networked via Ethernet to a central switch and fed to a central server for further processing. Because of the room's layout and the number of cameras we had access to, we settled on the layout in Figure 10, which leaves an open, blind spot in the room (as shown in Figure 11). The blind spot is a result of a tall server rack placed in the center of the room, simulating static occlusions for the person-tracking problem. The rack blocks the line-of-sight of the cameras such that the area behind it cannot be tracked. Effectively, the blind spot can be thought of as a "shadow" cast by the server rack in the room. It can be eliminated by purchasing a separate camera to cover that space.

Once the cameras are positioned in the room, they are automatically calibrated by providing a track target (usually a person) that traverses the space in the track area. From the collected track information, the server is able to compute the correspondence in track locations between the cameras and back-calculate the orientation and location matrix (six degrees of freedom: the x, y, and z positions and the x, y, and z rotations) of each camera. This geometric transform can then be used to transform any detected targets into arbitrary coordinates through basic linear algebra.

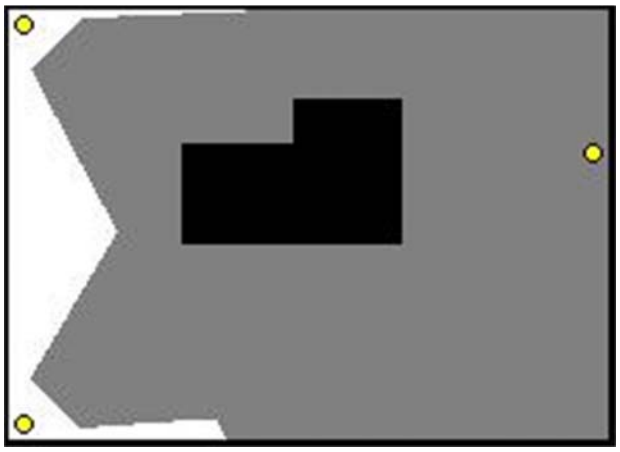


Figure 10. The yellow dots show the location of the cameras. The black boxes are occlusions, and the grey area is the effective coverage.

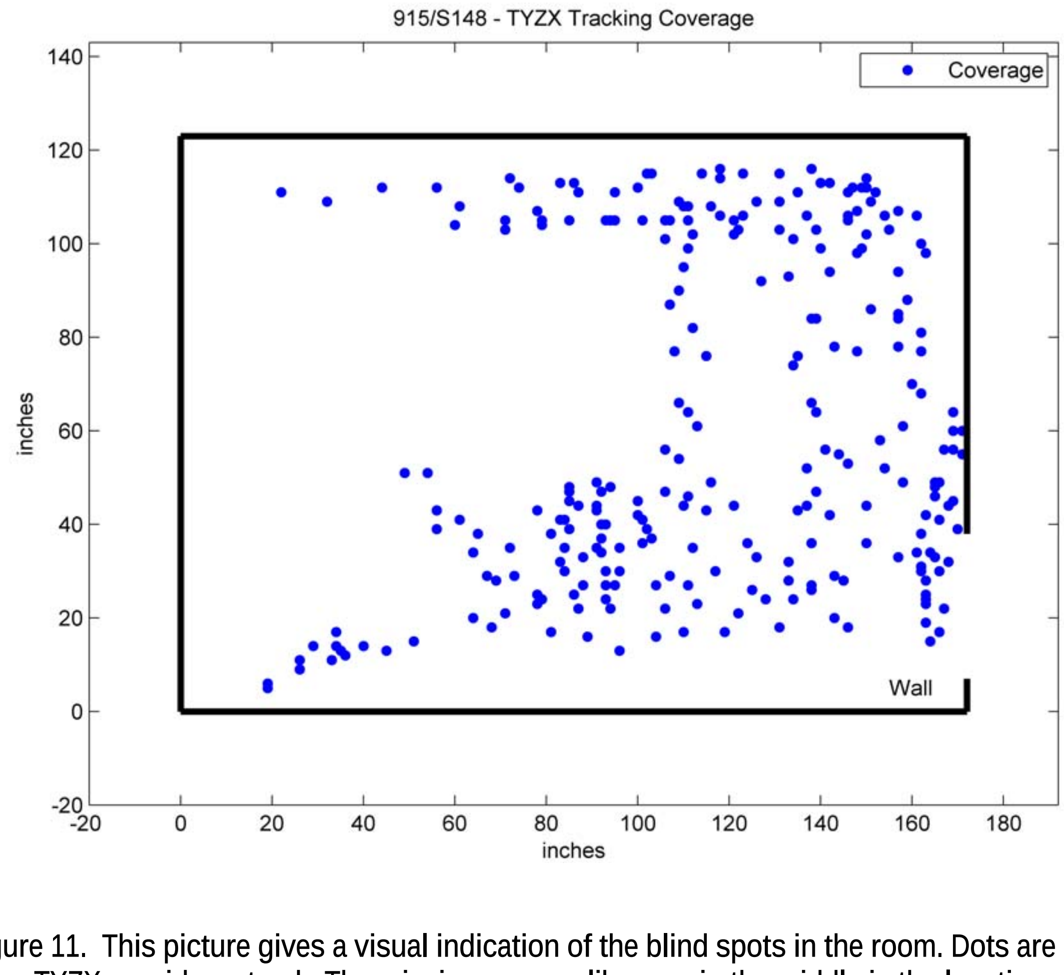
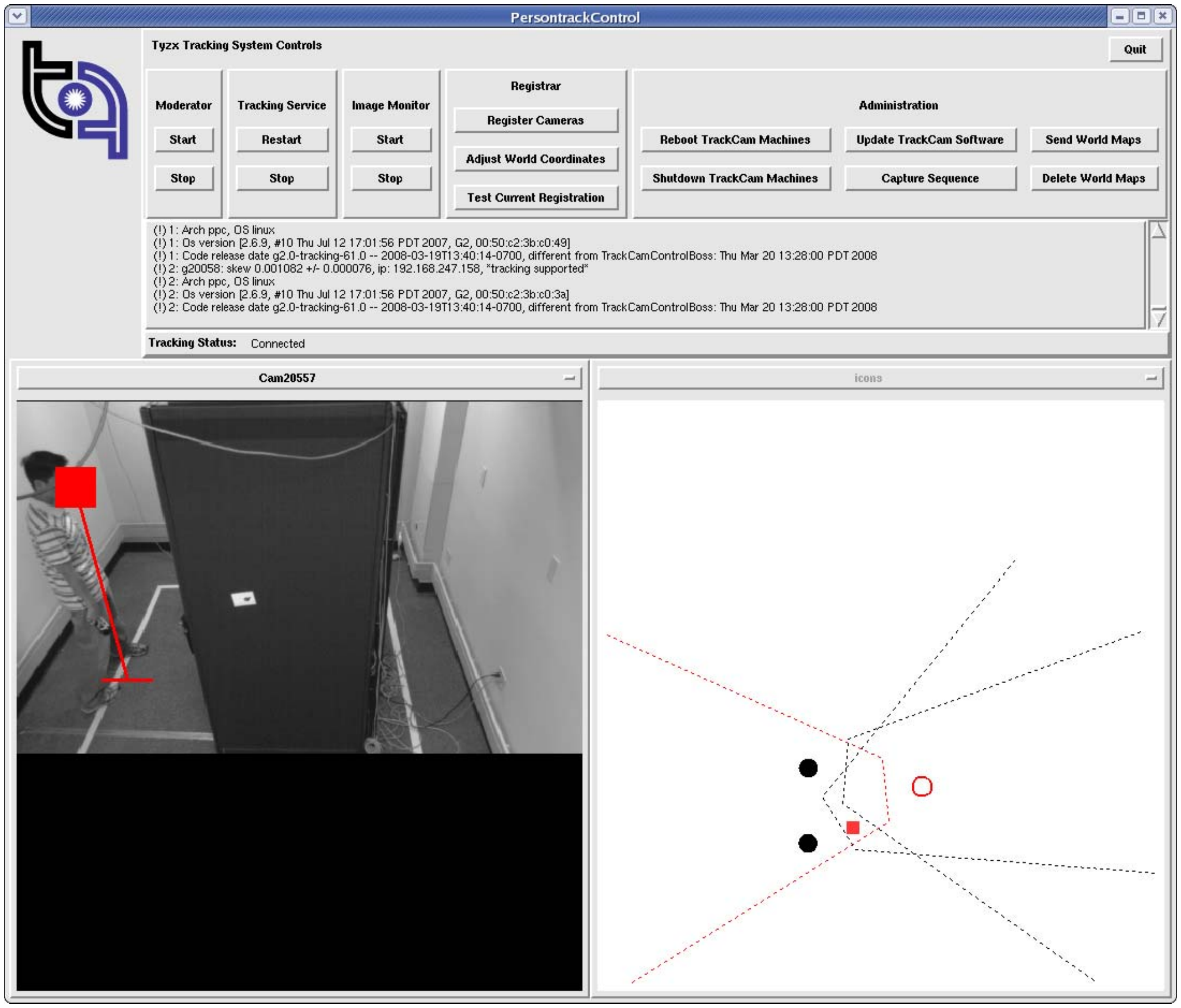


Figure 11. This picture gives a visual indication of the blind spots in the room. Dots are drawn where TYZX provides a track. The missing, square-like area in the middle is the location of static occlusions, such as the server rack. The other missing track areas are blind spots.

Figure 12 shows a screenshot of the TYZX PersonTrackControl program which comes with the TYZX install CD. This program serves as the control gateway to visualize the networked cameras and their track results. Note that the GUI displays the location of the cameras (the circles) and their fields of view. It also displays the image as viewed from a selected camera, with a marker drawn on the tracked target.

Multiple targets are easily tracked when different cameras see the same group of people from different angles (refer to Figure 7 for a screenshot). Depth information allows us to differentiate whether one object is in front of the other, and information from a separate angle may serve to corroborate the presence of a second person occluded by the first.



```
(!) 1: Arch ppc, OS linux
(!) 1: Os version [2.6.9, #10 Thu Jul 12 17:01:56 PDT 2007, G2, 00:50:c2:3b:c0:49]
(!) 1: Code release date g2.0-tracking-61.0 -- 2008-03-19T13:40:14-0700, different from TrackCamControlBoss: Thu Mar 20 13:28:00 PDT 2008
(!) 2: g20058: skew 0.001082 +/- 0.000076, ip: 192.168.247.158, "tracking supported"
(!) 2: Arch ppc, OS linux
(!) 2: Os version [2.6.9, #10 Thu Jul 12 17:01:56 PDT 2007, G2, 00:50:c2:3b:c0:3a]
(!) 2: Code release date g2.0-tracking-61.0 -- 2008-03-19T13:40:14-0700, different from TrackCamControlBoss: Thu Mar 20 13:28:00 PDT 2008
```

Tracking Status: Connected

Figure 12. A screen capture from the TYZX PersonTrackControl program. The right panel shows the locations of the cameras and their lines of sight. The red box represents a tracked person. The left panel shows video from one of the three cameras, with the tracked target effectively marked with the red box.

3.3 Known Issues

As with the OpenCV software tracking, there are limitations to the TYZX system. The first issue is inherent to all camera systems: that which they cannot see, they cannot track. TYZX is unable to maintain track in the blind spots.

Second, because TYZX provides 3D measurements of targets—their x, y position from a bird's-eye-view and their height—TYZX thresholds the height to be above a default minimum, so that small foreground objects are not tracked as people. However, this default threshold setting also means that if people were to crawl or drop to the floor, TYZX would immediately lose track. This threshold can be controlled by the PersonTrack API.

3.4 Summary

In this section we have described how the TYZX camera system works (depth perception), showcased its capabilities and the particular test set-up, and provided visual results regarding the coverage of the system. Note that it gives accurate tracking in real-time by offloading most computation on the hardware, leaving the server to do simple tasks such as collating the data for display. The tracking results for TYZX are discussed in comparison with the overhead, PGR system in Section 4.

4 Comparing the Two Camera Systems

The TYZX system addresses many of the limitations of the PGR system: unreliable background-foreground segmentation, camera registration errors, inability to track multiple targets, and scalability issues.

Because TYZX's segmentation algorithm is based on the distance of objects from the camera instead of the color values of the pixels, the color of objects does not affect the segmentation nor do slight lighting changes. Instead, it is changes in depth perception that designate an object as foreground or background. Pixel values are completely irrelevant when detecting foreground objects to track. TYZX, then, is able to work around the unreliability of color-based background-foreground segmentation algorithm of an OpenCV approach.

In the process of setting up TYZX cameras, an automated camera calibration stage is performed that computes the six-degrees-of-freedom (position and orientation) for each camera. This step helps the server accurately compute the global coordinates of objects tracked relative to a known basis. The manually calibrated and configured PGR cameras lack this step and this level of flexibility. The OpenCV software solution is unable to deal with multiple targets in real-time. Rather, a change in the setup is required to provide enough information to distinguish multiple targets moving close together. With TYZX, depth information allows the cameras to distinguish people occluding each other, and multiple cameras provide different perspectives of the same scene, providing another view where the people are not blocking each other. Using this physical setup, there is no need for collision detection and identity maintenance in the TYZX software.

The TYZX cameras transmit 3D coordinates about the location of targets back to the server. This information is small and takes little bandwidth over Ethernet, which allows us to scale to larger numbers of cameras to cover a larger area than is possible when cameras are transmitting full frames to the server. The distributed pre-processing also removes much of the computational burden required of the central server, thus freeing up CPU time to do real-time calculations and tracking.

Thus, the TYZX system is far superior to the homegrown PGR system. OpenCV is a useful tool to quickly build machine vision applications, but distributed hardware-based image processing offloads much of the complexity from the CPU and also allows scalability and flexibility in building a generic, person-tracking system. TYZX's hardware-based and distributed model easily outperforms the centralized, software-based image processing system (embodied by the PGR system). Figure 13 shows the comparison between the TYZX tracking system and the PGR system. Excluding the blind spot, TYZX results are closer to the ground truth path than the PGR points.

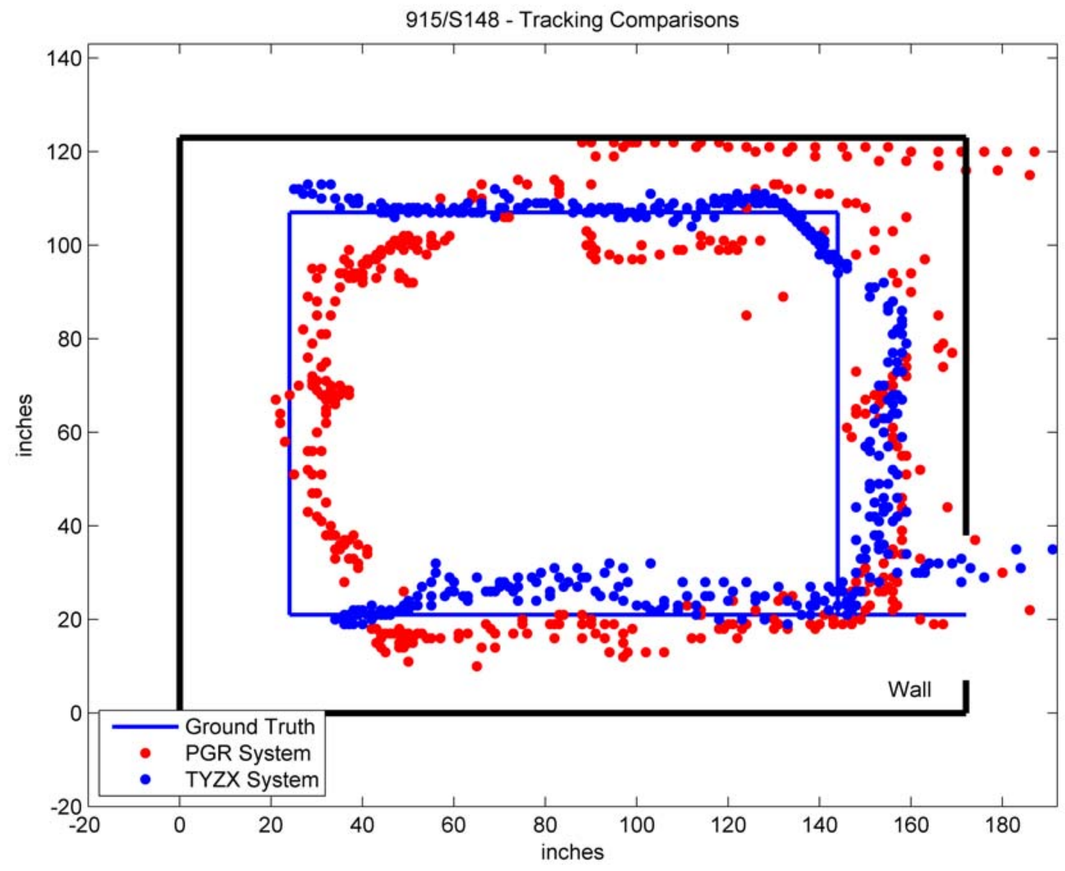


Figure 13. A comparison of the tracking results between the PGR overhead and TYZX camera systems. Note the accuracy of TYZX along the ground truth when the target is not in a blind spot.

5 Integrating the Two Camera Systems

Experimental results led us to believe that the most complete tracking system would incorporate data from each of the available systems. The blind spot that appears in the TYZX coverage is a function of our small demonstration room (approximately 170”x123”) with the server rack blocking a large area relative to the size of the room. While buying a fourth TYZX camera to cover the blind spot would solve our problem, procuring a fourth camera was not possible due to budget constraints: we used the opportunity to develop a simple data fusion technique for tracking people.

Each system maintains its own pool of tracked objects—a TYZX pool and a PGR pool--which then feed in to a global pool. Our system uses a “just-in-time” correlation scheme. Matching between an object in the TYZX pool and an object in the PGR pool is only done when a switch must be performed in the global pool (e.g. when an object enters the TYZX blind spot). When a target enters a TYZX blind spot, we find the closest track in the PGR pool and continue to follow that object until it reappears in the TYZX pool. This algorithm helps us to maintain the identity of the object as it moves around the room. To find the closest match, given an object, *q*, contained in either pool, we linearly search for the object *r* in the opposite pool which minimizes the heuristic in Equation 1.

$$h = c_1 \cdot t(q,r) + c_2 \cdot d(q,r) \qquad (1)$$

Note that c_1 and c_2 are constants tuned to the application, *t* is a function of two objects that returns the difference of the objects’ times of entry into their respective pools, and *d* is a function of two objects that returns the Euclidean distance (defined as $\sqrt{(q_x - r_x)^2 + (q_y - r_y)^2}$) between them.

The importance of the difference between the entry times term in our application was determined to be negligible because of the high number of false entry and exit events issued by the TYZX system.

An alternative to the “just-in-time” method would be a “soon-as-possible” approach. In such a case we would constantly maintain mappings between objects of the two pools and modify these mappings whenever a new object enters either view. This approach does more computation without actually improving our system. Figure 14 shows the resultant tracking capabilities of our “just-in-time” approach.

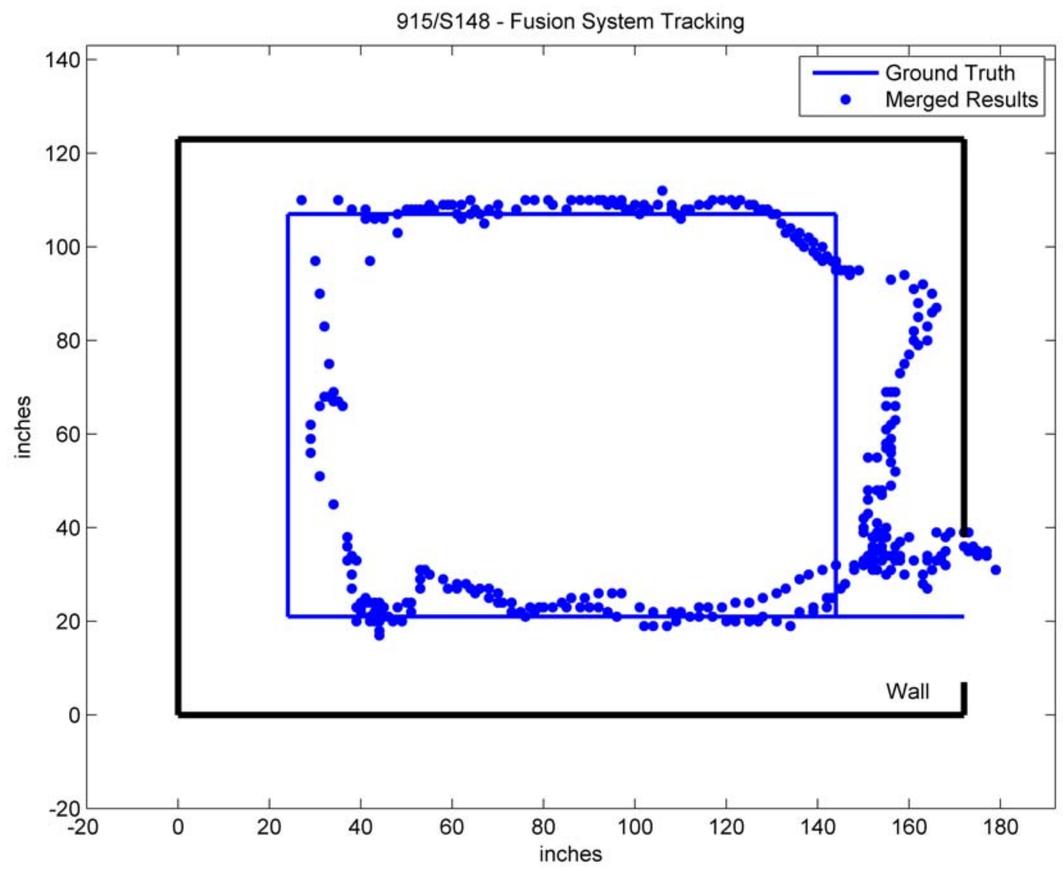


Figure 14. Results of merged outputs. Compare to individual outputs in Figure 13.

To extend the tracking, we can apply Kalman filtering and use the prediction data to smooth out the estimates and provide minimal estimation when a track is completely lost. This step is further discussed in Section 6.

The main downside to this design is the necessity of data synchronization. We must ensure that data from both camera systems represent track data at exactly the same moment in time. Any time lag will introduce a large amount of error into our tracking estimates. As it stands now, however, because both systems operate in real-time, the perceived error is small so no data synchronization is yet implemented. Note that future systems should implement time synchronization between distributed systems in order to produce accurate results.

6 Future Work

We use the tracking problem to explore a centralized, software-based solution (PGR cameras and OpenCV) and a distributed, hardware-based implementation. While the human tracking problem is essentially solved through TYZX and Sandia's in-house computer vision algorithm, further development may be necessary to continue tracking in the event of occlusions or more complex environments. More work can be done to build other functionality with the cameras or to better model it for computer simulations or serious gaming applications.

First, a part of the future work may be to build a model of how the system measures location and velocity. In essence, we wish to model a system $y = Ax$, where y is the measured position and velocity vector and x is the actual position and velocity vector. The matrix A describes how our "tracking algorithm" measures the real position and velocity, x. With such a matrix, we could easily abstract the tracking system to a computer program or simulation. Essentially, we hope to solve an inverse problem to allow us to port the model—along with its errors—to a simulation or serious game.

Second, since the locations of the Sandia cameras are hard-coded, it might also be useful to have the cameras automatically determine their locations relative to each other based on the features in their field of view. Such an algorithm would be immune to slight deviations in camera placement or measurement errors made by humans. It will also provide the most accurate coordinates to stitch the images together and perform tracking. This algorithm could be extremely similar to the automatic calibration algorithm that TYZX performs and could also be implemented with work done by Tomas Svoboda [6].

Third, Kalman filters and particle filters can be implemented on top of the tracking results to provide tracking in more complicated environments, and possibly to maintain tracking in blind spots (see [4]).

Finally, instead of using stereo cameras as TYZX does, we can rearrange the PGR cameras to provide multiple camera views of a central location. Using these multiple camera views, we can build an automated calibration routine as in [6]. From the multiple camera data, we can build simple 3D volumetric models for use in gesture recognition or behavioral pattern recognition. 3D volumetric information gives more information than only the location of people. We may be able to take the algorithm and embed custom hardware to build a generic camera security system that is scalable and easily deployable.

7 Summary

We have given a general overview of Sandia’s tracking system used for the NSEI project. It employs eight Point Grey Research Dragonfly cameras, placed at various locations in the room, to provide a rudimentary track of personnel. The video feed is processed by software provided by Point Grey (PGR FlyCapture SDK) and Intel (OpenCV) to give tracking results. We also describe the proprietary TYZX system. We compare the TYZX system to the PGR-based system and conclude that hardware-based solutions to machine vision problems provide scalable, networkable, distributed, and embedded results that give better performance than a software-based solution. We also propose a simple data fusion method using Euclidean distance to fuse track data from two independent tracking systems. Finally, we propose some avenues for future work with the tracking system and a potential new area of work for video surveillance.

References

[1] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning Ensembles from Bites: A Scalable and Accurate Approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

[2] Trista P. Chen, Horst Haussecker, Alexander Bovyrin, Roman Belenov, Konstantin Rodyushkin, Alexander Kuranov, and Victor Eruhimov. Computer Vision Workload Analysis: Case Study of Video Surveillance Systems. *Intel Technology Journal*, 09:109–118, May 2005.

[3] Gaile Gordon, Xiangrong Chen, and Ron Buck. Person and Gesture Tracking with Smart Stereo Cameras. In Brian D. Corner, Masaaki Mochimaru, and Robert Sitnik, editors, *Three-Dimensional Image Capture and Applications 2008*, olume 6805, page 68050T. SPIE, 2008.

[4] Michael Isard and Andrew Blake. Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision*, 29:5–28, 1998. http://www.robots.ox.ac.uk/misard/condensation.html.

[5] L. Li, W. Huang, I.Y.H Gu, and Q. Tian. Foreground Object Detection from Videos Containing Complex Background. *ACM Multimedia*, 2003.

[6] Tomas Svoboda. A Software for Complete Calibration of Multicamera Systems. Talk given at MIT CSAIL, January 2005. http://cmp.felk.cvut.cz/svoboda/SelfCal/index.html.

[7] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report TR 95–041, Department of Computer Science, University of North Carolina at Chapel Hill, July 2006. http://www.cs.unc.edu/welch/media/pdf/kalman intro.pdf.

[8] John Iselin Woodfill, Gaile Gordon, Dave Jurasek, Terrance Brown, and Ron Buck. The TYZX DeepSea G2 Vision System, a Taskable Embedded Stereo Camera. *Computer Vision and Pattern Recognition Workshop, 2006 Conference on*, pages 126–132, June 2006.

[9] D. Yang, H. Gonzalez-Banos, and L. Guibas. Counting People in Crowds with a Real-time Network of Image Sensors. In *Proc. IEEE ICCV*, 2003.

Appendix: Programming Resources

This appendix section assumes the reader of the article has basic programming experience with Linux (make, MAKEFILEs, etc.) or Microsoft Visual Studio. It is intended to point the reader to programming resources used in this project.

A.1 Setting Up OpenCV

To set up the OpenCV library for use, follow the instructions found under the "Installing OpenCV" and "Getting Started" sections on the OpenCV wiki found at http://opencvlibrary.sourceforge.net/.

A.2 Setting Up Point Grey's SDK

The Point Grey SDK is set up similarly to the OpenCV library. It should be a fairly straightforward operation in Linux, but since the camera computers in this project were Windows machines, it was installed for use in Microsoft Visual Studio .NET. Follow the same instructions as in OpenCV but include the paths to the FlyCapture SDK's "...\lib", "...\src", and "...\include" directories. In addition, the "...\bin" directory should be included under "executable files." Finally, include the static "flycapture.lib" file in the linker under project properties.

A.3 Setting Up TYZX

Setting up TYZX requires networking the TYZX cameras to a central computer through a hub or a switch. The server's IP must be 192.168.247.2 and it must have ntp running. The ntp server must be configured properly to serve time (whether on an open network or a closed network); TYZX tech support is extremely helpful (e-mail: info@tyzx.com) in helping to set up the cameras and set up the time server. Furthermore, TYZX provides excellent documentation for their cameras which come with their install CD and can also be obtained directly from them.

Once the central server is set up and networked to the cameras and has properly synchronized time with the cameras (running ntptrace should yield a stratum of less than or equal to 4), then PersonTrackControl can be run. The TYZX person-tracking GUI (as shown in Figure 12) then appears on screen and provides tracking results.

Distribution

1	Andrew Spencer 307 SE Onyx Court Lee’s Summit, MO, 64063	
1	MS 9106	Doug Gehmlich, 8226
1	MS 9106	Ann Harren, 8226
1	MS 9106	Harvey Ho, 8226
1	MS 9106	Bryn Miyahara, 8226
1	MS 9106	Marisa Ruffolo, 8226
1	MS 9106	Eric Chu, 8226
1	MS 9102	Jim He, 8227
1	MS 9102	Donald Dowdle, 8229
1	MS 9102	David Burnett, 8229
1	MS 0899	Technical Library, 8944 (electronic)
2	MS 9018	Central Technical Files