Sequence Recognition Algorithm for Lightweight Single Person Tracking Using Pyroelectric Infrared Boundary Detectors

John Burchett, Mohan Shankar, Nikos Pitsianis, B. D. Guenther, and David J. Brady

ABSTRACT

We introduce a tracking system for a single human using pyroelectric infrared sensors. In contrast with conventional tracking systems, which rely on high-resolution imaging for human tracking, we track a human source using a distributed collection of pyroelectric detectors. We introduce a spatial decomposition algorithm to relate the sensor’s events to spatial boundary crossings, combined with a sequence processing algorithm to relate these events to paths within the space. We demonstrate successful tracking of a single person within a studio space using 14 distributed sensors.

1. INTRODUCTION

Person tracking is an active field of research, for uses varying from security and commerce to adaptive interfaces and control systems. Most available tracking systems are video or imaging based, relying on intensive analysis to locate persons and categorize their movements. Other methods, including the use of acoustic localization, floor-based pressure sensors and infrared tracking sensors are attractive alternatives principally because of their lower computational requirements. Our group recently constructed an infrared tracking system using pyroelectric detectors and relying on overlaps between the viewing space of individual volume sensors to perform triangulation. While such a system is easy to construct, its efficiency depends on the relative abundance of overlapping sensory regions. By relying on the overlap of detectors rather than treating detectors independently, we implemented a mapping function where \( N \) detectors can represent \( 2^N \) distinct position states. Sensors that detect two dimensional boundary crossings add a degree of flexibility at the expense of the number of representable states but are significantly easier to distribute and monitor. We develop a theoretical event-driven framework for the association of position states with sequences of sensor events. After addressing the characterization of our sensor modules, we develop a spatial decomposition algorithm that facilitates tracking using our framework.

1.1. State Definition

In a detection system, the state of a monitored space is assigned a value based on the collective measurement of the sensor array. We model people by an intensity function \( \psi(\vec{r}, t) \) defined within the physical space. For our purposes, we limit \( \psi \) to reflect the infrared heat signature of body heat as viewed from above, primarily characterized by vertical emissions from the head. Given a visibility function \( V_i(\vec{r}) \) for each sensor, we define each sensors’ output to be

\[
M_i(t) = \int \psi(\vec{r}, t)V_i(\vec{r})d\vec{r}.
\]

Imaging arrays are perhaps the most common tracking devices, providing both high resolution and a simple one-to-one mapping function that allows for visual verification. Individual pixels are mapped to spherical cones of space, and tracking is accomplished by linearly searching the array. After determining which pixels see the person, we define the state of the image to be the collection of these pixels. Since pixel order is irrelevant, a pixel array of length \( N \) can describe \( 2^N \) unique states. In a tracking context, we define a person’s state as a centroid position determined from the collection of active pixels. However, humans occupy a contiguous region on the image plane, greatly undersampling the number of representable states. This waste increases as a person occupies a larger number of pixels, reducing the accuracy of the centroid approximation. Conversely, resolution becomes better as fewer pixels are occupied; the best resolution is achieved when the target occupies a single pixel. Even in the best case, we can only resolve \( N \) possible position states.
1.2. Static Multiplexed Representation

The state assignment fostered by imaging solutions resembles a technique known as “one-hot” state encoding. One-hot encoding is used to simplify the decoding of signal information, trading ease of interpretation for significant waste of state encoding elements (sensors) as compared to the number of quantifiable states (positions). This is advantageous for human-monitored systems because we interpret information this way naturally, but it is inefficient in storage requirements. A compact encoding defines each bit of the state vector to represent a different set of separate states; an example of this would be a binary state encoding. Given that only one state can be occupied at any given time, it is an optimal encoding scheme; \( N \) bits can represent \( 2^N \) possible states. A multiplexed sensor performs a similar function in space, breaking the space into many discrete spatial regions and assigning each sensor to view a combination of possibly disjoint regions. In the example shown in Figure 1, 4 sensors view a subset of the 16 possible target locations. Each location is coded with a 4-bit binary number, where a ‘1’ in the \( i^{th} \) bit means that location is visible to the \( i^{th} \) sensor. The potential of this method has been explored in its theoretical aspects and demonstrated using a fiber floor mat designed to track two people on an 8x8 grid.

Multiplexed sensing represents an ideal coding solution, but it has limitations when applied to person tracking. Humans are not point sources; their limbs move, their radiation pattern is highly non-uniform, and while their motions are continuous, they are self-occluding. Interpreting multiplexed sensor data becomes difficult when multiple states may be realized simultaneously, e.g. multiple persons present in the space. This case can potentially be solved by using more sensors and a coding derived by a genetic algorithm. However, a person may also overlap multiple visible regions of a given sensor, which would be indistinguishable from only being present in one visible region. Neighborhood state assignment can address this problem. Neighborhood codes ensure that if two adjacent regions each trigger their own code, then the resulting overlapped code word will be equal to one of the two codes. Further, multiplex coding requires a certain flexibility in assigning codes; the utilization of the code space depends directly on how much of the sensor space is visible to each sensor. Each sensor must be able to see the total spatial region being observed for every code word to be realizable. Tracking is typically performed at significant distance from the target, and sensors are distributed through the space, making construction of an efficient mapping very difficult. Spatial mappings must be evolutionarily determined at significant computational expense; they are also highly dependent on the precise locations and visibilities of the sensors, making rapid deployment of an efficient code impractical. While multiplexed coding is a good idea, it is difficult to apply to a long-ranged, distributed tracking process.

1.3. Sequence-based State Machine

Person tracking is a study of humans in motion; multiplexed sensing is primarily applicable to memoryless staring sensor systems. Multiplex coding can be efficient in its use of sensors but works best only in cases where the visibility of sensors
can be very precisely known. This is true for cameras and other high-resolution devices but not for lightweight systems using individual detectors. A system of pressure-sensor fibers was proposed by Zheng, implementing both a multiplexed code layout and an event-driven model where fibers act as boundary sensors. The utility of a multiplexed code is proven, but sensor deployment is difficult; a genetic algorithm is required to construct an efficient fiber layout. The event-driven model treats each fiber as a tripwire whose optical properties change when a person steps on it. Using the fibers as boundary sensor, Zheng developed a state machine model that treats human motion as a series of independent, though correlated, events.

Applied to a pyroelectric motion system, each sensor is responsible for monitoring a plane in space projected from its location, triggering when motion is detected. As people move around a space distributed with sensors, they trigger sensors in a sequence that is directly determined by how their path intersects with each sensor’s visibility region. Given an understanding of the spatial relationship between detectors, we perform a decomposition similar to that of a multiplexed system. Rather than assigning a fixed code to each spatial region, we associate a position with the set of possible walking paths ending at that location. A person walking each path in this set will generate a unique sequence of sensor events. We model the event-driven sensor network as a neighborhood graph, where each vertex corresponds to a spatial resolution element, and each connecting edge represents the sensor sequence that will occur if the person moves from one region to another. After creating the graph by projecting sensor boundaries in a common space, we evaluate each possible sensor sequence in a spatial context. Tracking is accomplished by pattern matching the most recent sensor triggers with the each stored sequence to evaluate possible target locations.

We have developed a method for modeling the response of sensors to human motion, culminating with an event-driven tracking model that maps sensor sequences to a spatial connection graph. After a brief description of our detectors and a demonstration of their usefulness as boundary monitors, we introduce an algorithm for the construction of the spatial connection graph. We also survey several deployment strategies, describing bounds on the resolution of the sensor deployments. We apply our decomposition to a sensor network deployed to track a person around a studio space with distributed clusters of pyroelectric detectors, showing results from the tracking of a single person. Extension to multiple targets will be left to a future publication.

2. SENSOR MODEL

Human bodies radiate heat to their environment as predicted by the Stefan-Boltzmann Law. An average human frame an approximate surface area of two square meters and has an approximate internal temperature of 37°C. That average person would radiate outward about 1026 watts of power, meaning about 500W/m². Allowing for an inflow from a 20°C environment of about 819 watts, this gives a net out-flow of about 200 watts and a net out-flow density of 100W/m². Radiated heat is also a strong function of the assumed emissivity; for bare skin it is 0.98, but when clothing intervenes it is reduced to approximately 0.5. If the body is approximated by a black-body source with an internal temperature of 37°C, Planck’s equation predicts a wavelength-varying spectral radiance with a peak value near 9μm. The heat is radiated non-uniformly because of both the orientation of the skin and various modulating factors such as clothing, etc.

2.1. Detector Platform

To detect this radiation, we employ the dual element pyroelectric detector PIR325 from Glolab Corporation. To improve both our collection efficiency and spatial resolution, we use a Fresnel lens array to create a discontinuous visibility pattern. By segmenting the detectors’ visibility, the detectors produce a time-varying signal that can be used to both track and classify objects moving in the field of view. A detailed description of the detector and our platform has previously been reported. We chose a lens array with 11 lenslet elements, which produces the signal shown in Figure 2 when a person walks from left to right and then returns.

Each lens in the array has an angular coverage of 7°, and the overall coverage of the lens array is 88°. The sensor modules are assembled using eight pyroelectric detectors, each of whose fields of view is modulated by a Fresnel lens array. The total sensor module package is shown in Figure 3.
2.2. Boundary Sensors

Our detector setup is responsive to thermal motion within its visibility area; it projects a spherical section outward from its location. In our event-driven model, we use ceiling mounted devices that project this section to a rectangular region on the floor. The detectors’ angular visibility results in a rectangle measuring approximately 5m by 1m when mounted from a 5m ceiling. As a person moves across the longer axis, multiple trigger events will be detected as shown in Figure 2. As the motion vector moves off-axis, the number of peaks diminishes as fewer lenslets are crossed, finally resulting in a single trigger being detected if a person moves along the minor axis. At that point, the person is crossing the visibilities of each of the dual elements at the same time, so the output will be independent of the direction of motion.

We model this visibility as a line segment, where we associate a sensor trigger with a person crossing the visibility region along the minor axis. There are two principle difficulties with this approximation, since the minor axis would ideally be as narrow as possible. First, as a person crosses parallel with the minor axis, a three-part pulse is detected by the dual element detector in the form $[+,-,+]$, similar to Figure 2. The pulse’s form is primarily a function of the impulse response of the detector and the non-uniformity of the human body; we apply a Kalman filter remove this dependence, reducing this pulse to a single detection. Secondly, the person crossing the sensor’s visibility may not be walking directly along the minor axis. The multiple triggers that result will be interpreted as the person crossing the line back and forth with a frequency that depends on both their velocity and their angle with respect to the major axis. In the general case, this is acceptable because even with an ideal (thin) line detector, walking along the line effectively involves this recrossing due to slight weight shifts as the person walks. It may cause errors if the recognized crossings place the person on the wrong side of the line after he leaves the visibility region, but this will be dealt with in the detection algorithm as discussed in subsequent sections.
Figure 4. Sensor “Line” approximation

2.3. Sensor Projection

Modeling of our boundary sensors in a common space requires the projection of their fields of view in common context. Given each sensor’s location and orientation, its field of view can be determined mathematically using the characterization of the detector and its lens. Each sensor cluster’s location is measured with respect to a common reference point, and its orientation with respect to reference angles. Let \( P_i \) and \( O_i \) be the position and orientation of the sensor cluster, respectively, defined as

\[
P_i = (x_i, y_i, z_i)
\]

\[
O_i = \Theta_i, \Phi_i
\]

Each sensor’s field of view is characterized as a spherical cone, with its center line defined by the point-vector pair \( P_i \) and \( O_i \). Its field of view is defined by an angular range of \( \pm 44^\circ \) along the major axis of the lens and \( \pm 6^\circ \) along the minor axis. Let \( r \) be the characterized range of the detectors’ ability to detect human motion; in practice this is at least 30 feet but will also be limited by the physical dimensions of the sensor space. To model the boundary line created by a sensor, we construct four line vectors defining the edges of the field of view. We project the lines to determine their intersection point with the nearest spatial boundary, either the walls, floor, or ceiling of the sensor space. These four points define four corners of a boundary stripe, which if crossed will result in a sensor trigger event. An example of this stripe is shown in Figure 4; a boundary line is approximated by averaging each pair of end points.

A line segment \( L_i \) is defined by connecting these two averaged points. Each line \( L_i \) defines the boundary line monitored by the \( i \)th sensor in a universal coordinate system. This approximation is reasonable for moderate range, with the size of the rectangular region being 5x1 meters when projected from the ceiling to the floor at a range of 5 meters. These boundary sensors lines are distributed within a sensor space, creating a spatial decomposition context to facilitate the tracking of a moving person.

3. DECOMPOSITION ALGORITHM

Interpreting the significance of sensor events requires an encompassing spatial model that makes the spatial connections between sensor events clear. Zheng has developed a more general model, using the free placement of optical fibers to decompose a floor space into regions. Using a boundary sensor model makes this task simpler because the segments are linear and their intersection points with other segments are well defined. We demonstrate an algorithm to simulate...
the spatial decomposition imposed by a collection of sensor lines, considering several cases and corrections required to maintain an accurate model.

3.1. Algorithm Review

Our spatial decomposition algorithm is similar to a fundamental problem in computational geometry, commonly referred to as the Maintainance of Line Arrangements. A variety of algorithms exists for the evaluation of the intersection of a collection of line segments, including plane-sweep methods\textsuperscript{12,13} and graph based approaches\textsuperscript{14}. Optimal methods in both time and space requirements have been developed,\textsuperscript{15} but are also complex and rarely used. While our algorithm was developed independently, it is essentially similar to those developed for the LEDA toolkit\textsuperscript{16,17}. Line intersection algorithms are commonly generalized to work with a collection of hyperplanes in arbitrary dimension, creating an arrangement of polygon volumes by intersecting the planes. Arrangements of polygons created by this decomposition are used to test for degeneracy conditions and to optimize the solving of certain linear constraint problems. It has been proven that the number of polygons resulting from the decomposition of $N$ sensor lines will result in $O(N^2)$ spatial regions and will take $O(N^2)$ running time.\textsuperscript{11}

3.2. Algorithm Details

Each sensor line defines a broad range of possible transitions in the position of a person present in the space. While each sensor line has spatial context, a sensor trigger provides poor localization given the length of each line and ambiguity in the direction of motion. Sensor events are also correlated because of their spatial layout. We model this correlation by constructing a neighborhood graph that models the relationship between sequential sensor events. Each node in the neighborhood graph is a spatial polygon that represents a possible position state of a person within the sensor space. Sensor lines form the edges of this polygon, connecting it to adjacent nodes in the graph. Each edge represents a sensor that will trigger if a person moves from that node to an adjacent node. By definition, no sensor lines penetrate a node; they represent minimum resolution elements. Each edge is stored with two reference values: a number identifying the sensor whose boundary line created this edge and a pointer referencing the polygon node lying on the other side. Most polygon edges correspond to a boundary line of a sensor; boundaries of the space are initialized as “dead” edges since they do not represent a sensor triggering event.

Beginning with a polygon representing the total sensor space, we iteratively intersect each sensor line with every polygon in the space. If the line intersects with the edges of a polygon, we create a new polygon by splitting the polygon in two and dividing its vertices. The newly created edge, bounded by the two intersection points of the sensor line, is associated with the sensor that split the polygon. Reference pointers of the two polygons and their neighbors are then updated to reflect the split. We repeat this process, intersecting each sensor line with all the existing polygons one by one. While the sensors’ fields of view are limited, we use a line vector for this intersection process to limit the number of special cases. Maintaining the accuracy of our model requires that we make some corrections; one is performed during each intersection step, and the other is performed post hoc for reasons that will be seen below. For clarity in discussing the intersection step, the polygon being split is referred to as Poly and the new polygon being split from it as NewPoly, as shown in Figure 5. Tables 1 and 2 contain a pseudocode layout of the decomposition algorithms.

3.3. Two corrections

A correction step is required while splitting each polygon to maintain the integrity of the graph, specifically to maintain the neighborhood reference pointers of Poly, NewPoly, and their neighbors. As we iterate the splitting action of a sensor line over the set of polygons, we often use the same line to split adjacent polygons. Since we are splitting each polygon one by one, the two adjacent polygons that also intersect with the sensor line may or may not have been split. An example of this case is shown in Figure 5; the shaded polygon is currently being split by the indicated sensor line. The adjacent polygon above has already been split, but the polygon below has not. In the former case, our neighbor is already in its final state (at least as regards to this sensor line), and its relevant neighbor pointers need to be directed to Poly and NewPoly. If it has not been split, we can safely leave the redirection to the subsequent intersection of the sensor line with that polygon.

To determine whether an adjacent polygon has been split, we first reference it using the neighbor pointer of each edge that intersected with the sensor line (the upper and lower edges of Poly/NewPoly in the figure). Searching through the vertices of the adjacent polygon, we look for an edge that is marked with the identification of the sensor line. If such an edge exists, we know that the adjacent polygon has been split, and its neighbor pointers must be updated. Further, we
int DoTheyIntersect( LineVector A, LineSegment B )
  
  * Check that $A, B$ are not co-linear ($A \cdot B - |A||B| \neq 0$).
  * Define a vector $H$ pointing from the head of $A$ to the head of $B$.
  * Define the intersection point along $\vec{A}$ as $I = A_{head} + \mu_A \frac{\vec{A}}{|A|}$.
  * We know from geometry that
    \[
    \mu_A = \frac{(B \cdot H)(A \cdot B) - (A \cdot H)(B \cdot B)}{(A \cdot A)(B \cdot B) - (A \cdot B)(A \cdot B)}
    \] (4)
  * Similarly define $I$ in terms of $B$ to get an expression for $\mu_B$; we find that
    \[
    \mu_B = \frac{(B \cdot H) + \mu_A (A \cdot B)}{B \cdot B}
    \] (5)

  * **Note:** The two points, $(A_{head} + \mu_A \frac{\vec{A}}{|A|})$ and $(B_{head} + \mu_B \frac{\vec{B}}{|B|})$, describe the closest pair of points on the respective lines. Since our lines are in 2 dimensions, these will be equal and identify the same point.
  * If $\mu_A$ or $\mu_B$ is between 0 and 1, then the intersection point is within the specified line segment.
  * We treat $A$ as a line vector and $B$ as a segment, so we return TRUE if $\mu_b$ is between 0 and 1; otherwise FALSE.

int IntersectLineWithPoly( LineVector L, PolygonType Poly, int SensorID )
  
  * Sanity Check: see if any of the polygon’s edges are co-linear with $L$. If so, simply re-label that edge with a new SensorID and quit.
  * Loop through polygon’s edges, intersecting each with $L$ using DoTheyIntersect( L, edge ); if we find an intersection ($0 \leq \mu_b \leq 1$), remember that edge as edge1.
  * Keep looping until another intersection is found (since $L$ is treated as a vector, there must be a second intersection). Remember this second intersecting edge as edge2.
  * If the line vector $L$ does not intersect with any of the polygon’s edge segments, return FALSE.
  * Truncate the two split edges by inserting the two new intersection points.
  * Copy the edges from edge1 to edge2 into a new polygon, NewPoly, while removing them from Poly.
  * Update the SensorID for the new edge in both Poly and NewPoly to reflect the caller’s SensorID.

  * **Correction:** if $\mu_B$ is within machine epsilon of 0 or 1, then the above will result in a similarly small edge near the intersection points. If this occurs, we delete this spurious edge to preserve numerical accuracy.
  * **Correction:** Check to see if the polygon bordering each of the two split edges has already been split by the same line $L$. If so, correct the respective pointers in Poly and NewPoly to point to either of the now-split halves of this bordering polygon.
  * **Correction:** Loop through the edges of NewPoly, correcting the neighbor pointers for bordering polygons (since they previously pointed to Poly)
  * Return TRUE.

| Table 1. Pseudocode for Polygon Split Algorithm |
must also updated the neighbor pointers of the polygon referenced by that edge. For clarity, we refer to these as \textbf{poly1} and \textbf{poly2}. The edge pointers of each of the four polygons (Poly, NewPoly, poly1, poly2) that refer to the edge split by the sensor line must be corrected to point to one another. In our example this assignment is clear; Poly is connected to poly1, and NewPoly and poly2 are adjacent. However, depending on the order of each polygon’s edges (clockwise or counter-clockwise), the adjacencies between these 4 may become reversed. We can establish these adjacencies analytically because the membership of the two endpoints of the edge split by the sensor line (darkened in the figure) are known. We determine adjacency by searching the vertex list of poly1 and poly2 for these two points.

A second correction step is required to maintain the accuracy of our model, given the finite range of sensor visibility as implied by our use of line vectors. The sensors’ visibility lines are in fact line segments with extent given by the projection used earlier to determine them. However, if we use this knowledge from the start, we run into a problem. In most cases, sensor line \textit{segments} do not span the sensor space; they will not intersect any two edges of the initial room-sized polygon if intersected as strictly line segments. This will result in no polygons being split; each consecutive line that is intersected with this same polygon would produce no decomposition. We earlier treated these lines as vectors for the purpose of intersection, resulting in a decomposition that, while guaranteed to produce a set of polygon regions, is inaccurate in one respect– there are edges created by parts of the sensor line that lie beyond the endpoints of that sensor’s visibility. We mark edges that extend beyond these endpoints as “dead” edges, similar to the initial edges around the sensor space. After decomposition is complete, the edges that correspond to a sensor line have been cut multiple times by other perpendicular sensor lines; this correction maintains the accuracy of the model while avoiding the lack of decomposition that would result from applying this condition earlier. Performing this \textit{post hoc} preserves the integrity of the graph model while preserving the edges that fall within each sensor’s visibility.

\section{3.4. Decomposition Results}

Our decomposition algorithm was applied to a set of 14 sensors, dividing our sensor space into a total of 82 polygons. This example decomposition is shown in Figure 6. The center polygon is highlighted for demonstration purposes, with a white circle drawn at its centroid and shaded circles at each of its 5 vertices. We represent the connecting edges of the neighborhood graph by lines connecting its centroid to its neighbors’ centroids.
In summary, our decomposition algorithm is successful in representing a sensor space as a connected network of possible position states of a person within the space. Each position element is connected to neighboring elements by sensor transitions; when a person crosses from one to another, the corresponding sensor will trigger as he crosses. Our spatial model provides a complete understanding of the space as it relates to the sensors positioned within it and will facilitate localization and tracking of a single person.

4. SEQUENCE-BASED TRACKING

Using the polygon decomposition derived earlier, we have quantified our understanding of the relationship between sensor triggers and movement of a person around the space. There are two complementary goals for single person tracking: localization of the person in $N$ measurements and tracking over time. Each is concerned with the connections between position states and the different sequences of sensor triggers that lead to given position states. After discussing a sequence processing algorithm, we analyze the goals and constraints of sensor deployment, deriving a bound on the spatial resolution. Several potential sources of error are evaluated, followed by results from the tracking of a single person.

4.1. Sequence Processing Algorithm

One important performance metric of person tracking is the number of unique measurements that are required to locate a target given no prior information. Imaging systems track by spatially oversampling, using pixel resolutions much more dense required given the size of the human body. Since new information is only relevant along the extremity of the body, many of the measurements performed by the pixels are uninformative. The boundary tracking model takes measurements that reflect motion across a boundary in space, effectively considering groups of pixels together as a single measurement. Tracking is accomplished by noting the sequence of sensors that detect movement, using the decomposition algorithm developed earlier to interpret these sequences and indicate possible target locations.

The decomposition graph developed earlier clarifies the spatial meaning of sensor events in relation to one another. By discretizing space with minimum resolution spatial polygons, tracking of a person requires only the association of a detected sequence of sensor events with the corresponding path through the space. Using a simple grid decomposition, Figure 7 illustrates several different paths through the space that correspond to the same destination but involve different
resultant sequences of events. Given a decomposition graph and limiting ourselves to paths with no cycles or oscillations in location, we can quantify every set of sequences that correspond to paths between any two points. While shorter sequences may correspond to several different paths, as the sequence lengthens it will describe a unique path. Sequence length requirements for a unique path have been described by Zheng\textsuperscript{7} in terms of the number of sensors and the geometry of the sensor deployment.

An analytical evaluation of a sensor sequence begins with no prior information, only the most recent sensor event. Given our decomposition, this reduces our consideration to those regions that border that sensor’s line; we determine these by direct searching of all available polygons, though this information can be stored in a lookup table. The list of polygons that match this condition are taken to be possible locations of the moving target. When another sensor event happens, the subject is assumed to have moved from one of these regions and crossed the line corresponding to that sensor. Each former estimated location is evaluated to see if it borders the line that was just triggered. If it does not, then the target could not have been in that location previously, and it is discarded. If it does border the active sensor, then the target may have crossed from that region to the corresponding adjacent region; that adjacent region replaces the former location as a new possible position of the target. This evolutionary step updates the possible locations of the target with each sensor trigger; as the sequence lengthens, incorrect locations propagated from the original “somewhere along this line” assumption will be eliminated, and the sequence will describe a path to a unique destination. Once the target is localized to a single region, the tracking is exact, and minimal processing is required to determine the evolving target location. Table 2 contains a pseudocode representation of the algorithm.

Qualitatively, consider a single sensor event as a sequence of length one. The location of the person given this one-element sequence can be determined by considering which of the tabularized sequences begin with the one-element sequence. In this case, searching the table will produce all the spatial regions that border the one sensor that fired. Extending to an $M$-element sequence, possible target locations are all those regions for which the first $M$ elements of one of its tabularized sequences matches the detected sequence. Each matched sequence defines both a possible target location given the detected sequence and a possible starting location of the person before the motion occurred. As the detected sequence increases in length, we eliminate non-matching paths, resulting eventually in a single path corresponding to the actual path taken by the person.

In the presence of error, the above algorithm may eliminate all possible target locations. When the target is lost, we
int BuildDecomposition()

- Initialize a list of polygons with a single polygon representing the dimensions of the sensor space; all its edges are marked "dead," since they do not represent a sensor event.
- Loop over each sensor line
  - Loop over each existing polygon
    - Call IntersectLineWithPoly( L, Poly ) to intersect Poly and the sensor line L.
    - If a new polygon is created (true is returned), place the NewPoly in a queue.
  After the loop completes, add queued polygons to the polygon list.
- Correction: Loop over every edge of every polygon; if the endpoints of an edge extend beyond the sensor line segment with whose SensorID it is tagged, mark if as a 'dead' edge to preserve model accuracy.

int PreviousLocations[MAX - POLYGONS];
int NewLocations[MAX - POLYGONS];
int SequenceHistory[];
int SequenceProcess( int ID )

- PreviousLocations[i] is set if the i\textsuperscript{th} polygon was determined to be a probable location of the target after the last sensor event. SequenceHistory[] is the list of sensor IDs that most recently triggered.
- Initialize NewLocations array to zero.
- Loop through PreviousLocations[]; if PreviousLocations[i] is non-zero
  - Evaluate edges of the i\textsuperscript{th} polygon; noting if any of them is tagged as corresponding to ID.
  - If an edge is marked with ID, record as poly the polygon that is across that edge border.
  - If no such edge is found, ignore it [since the target could not have been in that polygon region, because the target triggered the sensor corresponding to ID].
  - If poly was defined above, set NewLocations[poly] to 1, marking it as a new probable location of the target.
- Add ID to the end of SequenceHistory[], maintaining a list of previous sensor event.
- If no new locations are marked above, we have encountered an error in the sensor sequence and must redetect the target.
  - The sequence represented by SequenceHistory[] projected to no valid target locations; so remove the oldest sensor event and reproject possible locations [similar to the loop used above].
  - If the resulting projection still eliminates all locations, remove the next oldest sensor event; repeat until target location are reacquired.
- If the target is narrowed to no more than 3 polygons, determine a single locations by computing a centroid of the polygons.
- Verification: We can check localization by using a pan-tilt-zoom camera, aiming the camera at the target location.

Table 2. Pseudocode for Decomposition and Sequence Processing algorithms
rely on the recorded history of sensor events to recover the target location. Given that the received sequence results in no locations, we iteratively eliminate the oldest sensor event and reproject target locations based on the remaining sequence elements. In the worst case, we will eliminate all but the most recent event, making our new list of locations equivalent to the original one-event estimation. If the received sequence is only partially corrupt, this iteration will cease when the list of projected locations is not the empty set; ideally this will happen once the corrupt sequence data has been excised. Sources of error that may lead to this condition are reviewed in a following section.

4.2. Sensor Deployment

The resolution of our boundary system is determined by the size and distribution of spatial resolution elements created by the sensor boundaries. The decomposition algorithm uses geometry to predict the segmentation that is produced by a given arrangement of sensor elements, but determining an optimal arrangement that produces an even segmentation with minimal sensor requirements is an open problem. The segmentation is principally determined by the intersection points of the lines, since these are responsible for the splitting of polygons; our analysis will focus on different spatial distribution of sensors and the effects of layout on the expected numbers of intersections.

Our first design used a sensor layout for each platform that diversifies the orientation of the 8 sensors mounted on each platform. We orient 7 of the detectors at 7 degrees rotation from each other and tilted each at an angle of 12 degrees from the mounting plane. The eighth sensor is mounted at an angle of 90 degrees but not tilted from the plane. The tilt is employed to address the multiple intersection problem cited above while increasing the area over which sensor lines fall. Projected to the floor at a distance of 5 meters, we get the single-platform visibility regions shown in Figure 8. This design is not optimal for polygon placement but guarantees a diverse collection of sensor boundaries, even if platforms are mounted in a regular arrangement.

The spatial resolution of a boundary tracking system is governed by the number of polygons resulting from the decomposition of the “root” polygon representing the boundaries of the sensor space. New regions are created when each sensor line intersects with existing polygon edges and splits them in two. The number of new regions created is therefore directly proportional to the number of intersections between sensor lines, since already processed sensor lines will form the edges of the polygons with which later sensor lines are intersected. To guarantee evenly sized polygons, we employ a layout where each platform creates a square grid of sensor lines, similar in structure to that shown in Figure 7. The number of polygons is easily seen to be \( O(MN) \), where \( M \) and \( N \) are the number of horizontal and vertical edges, respectively.

More generally, the number of polygons that are split in two by a given sensor line depends directly on the degree to which other sensor lines run perpendicular to it. In general, this bound is proportional to one minus the dot product between lines, since lines running roughly perpendicular are expected to intersect and create a new spatial resolution element. A polygon is created when a line intersects another line; the number of polygons created is directly proportional to these intersections. When multiple lines intersect at or near a common point, the resulting polygon region will be unusably small (smaller than a person’s projection on the floor) and therefore wasted. By reducing the number of these multiple intersections, we increase the number of usable polygons.

Our grid layout is also easily extensible; each platform is used to monitor a small grid, and a larger grid is created by aggregating the grids of neighboring platforms. It guarantees an evenly sized collection of polygons while minimizing the occurrence of points that intersect more than 2 sensor lines. One downside of this layout is that it imposes a strict limit on the intersections that are possible between sensor lines on a given set of platforms. Interaction is limited between platforms; sensors on neighboring platforms are either redundant (they are co-linear) or else they do not intersect at all. This limits the number of polygons created, since they scale with the number of sensor intersections. While the number of realizable sensor sequences is reduced, the evenness of the decomposition reduces the several forms of error discussed in the following section.

4.3. Error Correction

Error in a tracking system is generally defined as the uncertainty present in a location estimation. In our boundary sensor model, ambiguity is possible both because of the minimum resolution elements imposed by the decomposition and due to several possible sources of error in the recognition of sensor sequences. In the best case, the target is localized to a single resolution element, and our uncertainty is determined by the average size of this region. The regions are currently uneven because of the construction of our sensor head, but future publications will demonstrate a more even decomposition. We are limited by the approximate nature of our line sensor, on the order of a square meter, but future publications. This
uncertainty is static and determined by the deployment of sensors, as seen in the previous section. In the latter case, the ambiguity depends on the number of possible target locations to which the received sensor sequence could correspond. Error in acquisition and interpretation of this sequence can be caused by several sources; each is addressed in turn.

**Sensor Error** is caused by the analog nature in which our sensors detect motion. As reviewed recently, our sensors are subject to the potential instability of a high-gain amplifier circuit. We apply a Kalman filter to normalize the analog stream and account for the known impulse response of the detector. The filter results in a smoothed positive and negative peak pattern, which we threshold to obtain a mean-adjusted signal level, and peak determination is performed by thresholding the second derivative. Filtration means that we will more often err on the side of missed events rather than false detection. Depending on the noise present in the sensor output (when no human source is present) and the profile and motion of the source as compared to the gain setting, a person may not generate a pulse of significant magnitude and shape to be recognized. By missing such peaks or falsely detecting peaks, errors may be introduced into the sensor sequence. With proper tuning of our filter, we achieve a satisfactory digital response with an acceptably small error. When errors do occur, we handle them similar to the **Sequence Ordering** condition discussed later.

**Sequence Error** is caused by the reality that a human source is not a point source; the subject has arms and legs that may trigger sensors multiple times when the body crosses the line or, similarly, trigger adjacent lines. It is normally assumed for tracking algorithms that the subject follows a path through the space rather than oscillating between regions or walking in cycles because the sensory data resultant from such motion is often inconclusive. To avoid sequence error, we impose a temporal threshold on the sensors output; it must trigger and remain active for a quarter second (measured with respect to the current sampling rate) to register as a sequence event. We apply this limit empirically, attempting to remove triggers caused by incidental motion rather than by the actual movement of the torso. We expect this threshold to be unnecessary if we confine the sensors’ fields of view (making the idealized line a more accurate model), but it helps to remove transient motion errors from our current sensor setup.

**Sequence Ordering** becomes an issue when multiple sensor lines intersect at or near a common point. While this may be accentuated by the non-point nature of a person, the latency inherent in the processing and collection of sensor signals may also cause sensor events to be retrieved in the wrong order. It is impossible to identify and correct error in this case; the sensor layout design should attempt to minimize this condition since it also impacts the resultant spatial resolution, as was seen earlier.

With a good choice of sensor deployment and proper accommodation for the above cases, we reduce our long term error to the average size of a resolution element. As the lengthening sequence corresponds to an increasingly smaller set
of spatial elements, error is minimized. In case of sequence errors, the target may be temporarily lost (maximum error) but can be easily recovered as new measurements are received.

### 4.4. Testing
We deployed multiple platforms around the ceiling of our studio space to test the accuracy of our sequence-based approach. Using our decomposition algorithm as a guide, we choose platform positions and orientations to produce the decomposition shown in Figure 9. After building the neighborhood graph and determining the polygons and their connecting sensor edges, real-time sensory data is gathered while the person walks a prescribed path. By processing the sensor events, we determine the sequence elements. We analyze the sequence in real-time, rather than looking up received sequences in what would be a very large table. The computed neighborhood graph yields a list of possible sensor triggers that may happen for each possible target location along with the destinations to which the target is assumed to move. Given a previously known set of possible locations for a single target, we use this graph to evaluate possible new locations of the target based on the new sequence element. Regions are eliminated for which a present target could not generate the received sensor sequence, reducing ambiguity over time. Ideally, a single location will be maintained over time as the target moves around the sensor space. To evaluate the accuracy of the tracking, we plot the segmentation overlapped with both the path taken by a target and the sequence of estimated locations. For ease of viewing, estimated locations are only displayed when the target is localized to 3 or fewer locations, and the centroid of these locations is displayed as a data point. Figure 9 shows the actual and estimated path.

To verify tracking was being reliably estimated, we used a Pan-Tilt-Zoom camera platform controlled by a serial interface. After computing a centroid location for the target, we determine the required angles to aim the camera towards the target at approximate head level. Since the locations are determined in real-time, tracking was verified by noting the consistent centering of the head in the camera’s image. As the person moved about the space, we occasionally temporarily lose the target but reacquire after several sensors were subsequently triggered.

### 5. SUMMARY
Conventional imaging tracking systems offer high resolution detection of human subjects but at the cost of high data and computational requirements. We turn to lightweight pyroelectric detectors to detect human motion, demonstrating their
utility as boundary sensors. We introduced a geometric decomposition algorithm to place these sensors in a collective space, modeling motion through the space as a sequence of boundary crossing events. We then developed a sequence algorithm that connects these sequences with possible paths of a single person through the space. Different forms of error are discussed, concluding with a demonstrated tracking result of a walk through the space. A future publication will extend this research to the tracking of multiple persons using similar algorithms.

Acknowledgements The authors would like to acknowledge the Army Research Office for supporting this work through the grant DAAD19-03-1-0352.

REFERENCES