Finding Field of View Overlap by Motion Analysis

(A Master Thesis at Axis Communications AB)

Fredrik Sydvart
dat12fsy@student.lu.se

Hampus Altvall
tpi11hal@student.lu.se

September 28, 2017

Master's thesis work carried out at Axis Communications AB.

Supervisors: Fredrik Andersson, fredrik.y.andersson@axis.com
Cristian Sminchisescu, cristian.sminchisescu@math.lth.se

Examiner: Kalle Åstrom, kalle@maths.lth.se

Abstract

Network cameras has in the recent years become more powerful. Each camera is independent and has its own surveillance task. It is reasonable that network cameras in the future should cooperate together to increase surveillance effectiveness. There is a need to find cameras sharing the same field of view in order for an operator to switch perspective. This thesis investigates how multiple network cameras can cooperate by finding the shared field of view between cameras. With the shared field of view, we implement an additional knowledge above system of network cameras and new use cases arises. Our method consist of applying a grid of cells on each camera's video stream and study movement detection. We gather contradicting proof of connectedness between each cell in the whole network of cameras. Our method avoids problems with feature detection such as different perspectives or image quality. We found that our method works with promising results and we can find shared field of view between cameras. There is a limitation in memory of storing all cells and we can only find overlap in regions with movement. This field has not been researched so much, making evaluation hard, as many approaches focuses on feature detection.

Keywords: surveillance network, network cameras, shared field of view, camera overlap, motion detection

Acknowledgements

We would like to thank our supervisor Fredrik Andersson for giving us the help and encouragement we needed. Especially for his drive to push the project forward and help us open up our minds to tackle the problem. We would also like to thank any employees on Axis we happened to come across and received help from. Lastly thanks to Axis for being an awesome, helping and welcoming company. Fredrik would like to especially thank his family for giving him the encouragement to continue working on this thesis. Hampus would like to thank his family and especially his dad for all the feedback.

Contents

Chapter 1
Introduction

This chapter describes the background, introduces problem definition and lastly brings up related work to see what other researches have found.

1.1 Background

Video surveillance is used by companies and governments for monitoring behavior and activities with the purpose of protecting peoples and assets. In 2014 there were 245 million professionally active and operational video surveillance cameras globally [1]. Over 20 percent are estimated to have been network cameras and the number is increasing. A network camera is defined as a static or dome camera with its own IP address and built-in wired or wireless network capability. Network cameras is interesting because of its agile capabilities. Each network camera enables software to be uploaded remotely and its monitoring task to be altered easily.

Even for a smaller network of cameras human operators monitoring ongoing or past activity require assistance from software to filter out relevant data. Adding new cameras to a network does not necessarily increase effectiveness in monitoring since it puts more load in filtering out relevant data for human operators.

To put into perspective the transportation market has, according to IHS data, an average of 12 surveillance cameras per building. This is twice as many as the government vertical market and four times as many as in education and retail [1]. However, regional differences in legislation and regulation in security requirements for government and corporation buildings have an impact in this installation data. Many times a corporation, government or school have multiple buildings which increases the size of the surveillance network. Active surveillance might also be outsourced to security companies where a few human operator will be responsible of multiple surveillance camera networks.

There is a need for extending camera networks to allow for better effectiveness by utilizing each network camera's capability. From here on in the report, if not explicitly

written, when we mention camera we mean network camera.

1.2 Problem Definition

Since cameras usually reside on the same network and monitor objects from different perspectives, this property can be exploited to further extend utilization of current camera network systems. The problem we have identified is:

> "Find the *shared vision topology* of a surveillance network. The shared vision topology is a graph describing network camera's field of views as edges linking camera nodes, also called *camera overlap*."

Let us consider the example setup over a street in figure 1.1 in order to understand the topology we want to find above. The setup consists of three different cameras *PTZ*, *Pan* and *S* with its own respective field of view and capabilities. It is possible to derive the shared vision topology and represent it as a graph where each camera is a node. Edges represent that there is some subset in the field of view which is shared between the two nodes. The example setup can be visualized as the following graph:

```
PTZ: [Pan]
Pan: [PTZ, S]
S: [Pan]
```

The topology must be up-to-date, easy to calibrate and accurate.

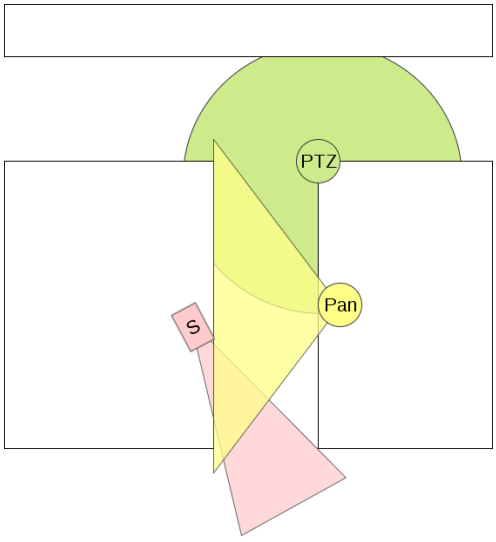

Figure 1.1: Example of a spatial camera setup. Each camera has different capabilities and tasks. Pan is a static panorama camera with wide field of view while S is a static camera with only zoom capability.

1.3 Motivation

This section aims to answer the question: *I know the shared vision topology of a camera network, what can I do with this information?*

There are multiple uses of having a shared vision topology over a known camera network. One example is together with tracking. When an operator follows a moving person, the operator might want to see the person in another perspective to identify the face. A shared vision topology gives a suggestion over an area in the current field of view that is also present in another camera.

Another example usage could be optimizing an installation. Each surveillance camera cost money to manufacture, install, maintain and use. Having too many cameras monitoring the same area is a waste of resources and can thus be optimized by reinstalling cameras with known overlapping field of view.

Another usage of the shared vision topology information could involve multiple cameras working together for one purpose. Suppose there is a panorama camera monitoring, through a wide field of view, a street together with multiple PTZ or static cameras nearby. An operator could only monitor the panorama camera. If any interesting object is found, the operator can get a list of cameras nearby to zoom in on the object.

1.4 Requirements

When solving the problem defined in section 1.2 we have some requirements:

1. The data gathering method for deriving the topology should not disturb with current surveillance or at least have minimum impact.
2. It should be possible to derive the topology in a network that contains cameras with different capabilities. One such capability is PTZ or panorama cameras.
3. It should be possible to find out which parts of a camera's field of view that is overlapping.
4. The solution should be able to handle large surveillance networks.
5. The complete solution should work with existing installed camera systems with little or no modification.
6. The complete solution should handle each camera malfunctioning, going offline or being replaced.

1.5 Related work

During our research we found different approaches for our problem definition, i.e. finding shared vision topology in a system of network cameras.

One way is to utilize feature-detection for determining shared vision graph [2]. This method requires SIFT or alike. The features can be passed around to other cameras and compared to determine shared vision. However, when camera orientation or perspective is significantly different from each other then SIFT fails. Feature detection can be tough since light condition and exposure can change features. There is also a problem in which similar looking objects can be interpreted as being in the same field of view as another camera, e.g. street light.

Another method by Detmold et al. (2007) relies on dividing all camera streams into a grid of cells, then assume all cells share the same field of view and find contradicting proof of connectivity between cells until there is not enough contradicting data compared by an empirically determined threshold [3]. Research by Detmold et al. (2007) currently does not support PTZ. Their solution assumes everything is connected between cells and new data creates exclusion. Their solution assumes area with no movement is connected to other cameras unless disproved. Detmold et al. (2007) will, if there is no movement, generate a lot of false positives since their solution assumes areas with no movement is connected to all cells unless contradicting proof is presented.

Chapter 2
Method

This chapter aims to explain our method to find overlap between multiple camera streams field of view. In order to do this, each camera's field of view is divided into a grid of cells and overlap is found between cells. We also implement our method as a prototype and discuss any assumptions needed.

2.1 Coordinate systems

There are two ways of dividing a camera's field of view into a grid of cells, depending on the type of camera. If the camera is static, the cells boundaries is defined in image pixel coordinates. If the camera has PTZ capabilities, the cells boundaries is instead defined in spherical coordinates as pan and tilt values as degrees. These pan and tilt values is limited by mechanical capabilities in the PTZ camera.

2.1.1 Image pixel coordinates

Image pixel coordinates has its y-axis oriented downwards and its origin in the upper left corner as seen in the right image in figure 2.1. Normalized image coordinates has its origin in the center of the image, and the perpendicular distance from the image edge to the origin is 1. To convert normalized image coordinates to image pixel coordinates we use the following equation:

$$\begin{aligned} x &= \frac{1 + x_n}{2} \cdot I_w, \\ y &= \frac{1 - y_n}{2} \cdot I_h \end{aligned} \tag{2.1}$$

where x and y is image pixel coordinates, x_n and y_n is normalized image coordinates and I_w and I_h is width and height of the image in pixels. Figure 2.1 illustrate usage of equation 2.1 for an arbitrary image.

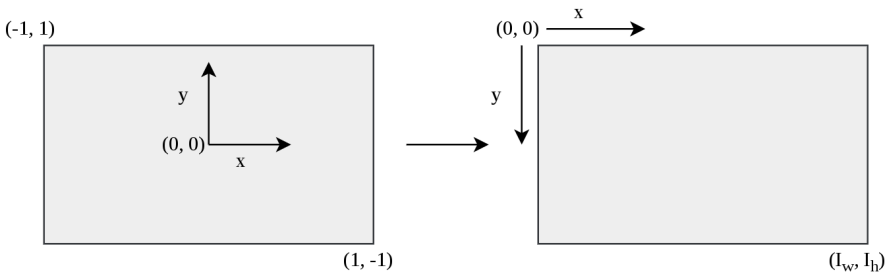

Figure 2.1: To the left is an image with normalized image coordinates. This can be converted into image pixel coordinates as seen to the right. Image pixel coordinates is limited by image width and height in pixels.

2.1.2 Spherical coordinates

We need a way to transform image pixel coordinates to spherical coordinates. A PTZ camera moves around by changing mechanical pan and tilt values. Panning is a horizontal movement of the lens while tilting is a vertical movement. The camera can be visualized as being inside a unit sphere. Every pixel in the image corresponds to an angle in relation to the camera.

We use a spherical coordinate system (r, θ, φ) where r is the radial distance, θ is polar angle and φ is azimuth angle. The coordinate system, in our case, is limited by:

$$\begin{aligned} r &= 1, \\ 0^\circ \leq &\theta \leq 180^\circ, \\ 0^\circ \leq &\varphi \leq 360^\circ. \end{aligned}$$

However, other ranges is sometimes also used in this report depending on capability of PTZ camera. Common ranges can be $(-180^\circ, 180^\circ]$ for φ or $(-90^\circ, 90^\circ]$ for θ.

2.2 Cell generation

Cell generation is the most important step in our method. The algorithm to find overlap between cells is independent on camera type and therefore grid cell structure. As long as each cell have a unique id they are distinguishable. Cell generation can be arbitrary generated, as long as the cell generation is consistent. We will cover how we generate cells on static and PTZ cameras.

2.2.1 Static camera

When defining where each cell's boundaries are in a static camera, we use image pixel coordinates. The image width and height is used from a camera stream to generate cells with unique id. In the prototype we specify how many cells fit in x- and y-axis. An example of a generated grid of cells can be seen in figure 2.2. Generated cells for one static camera from our prototype can be seen in figure 2.3. Depending on size of moving objects in a

real-world environment, cells in each camera can be bigger or smaller to increase overlap computation accuracy.

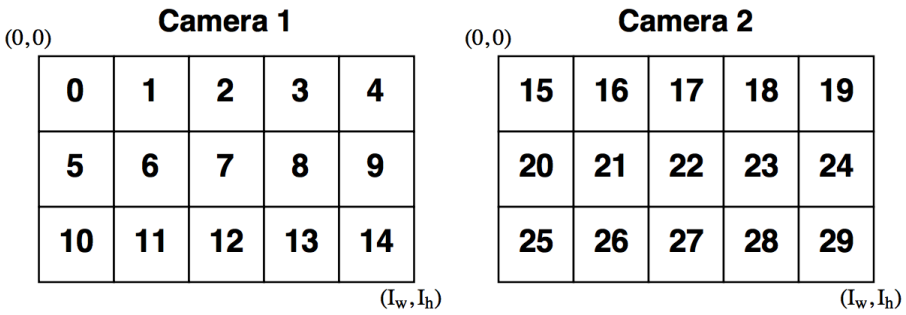

Figure 2.2: Generated cells with unique ids for two static cameras. Here 5 cells was used in x-axis and 3 in y-axis.

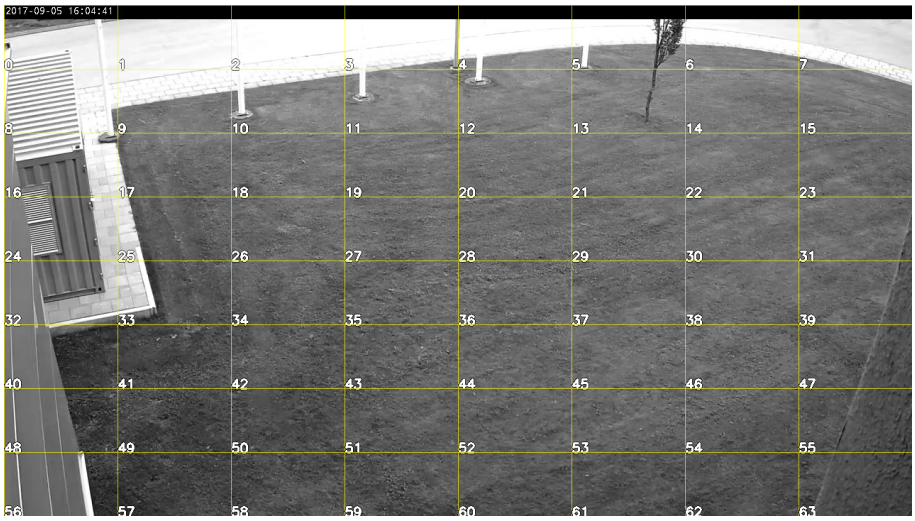

Figure 2.3: Cell generation for static camera from our implementation shown inside camera stream.

2.2.2 PTZ camera

Mechanical pan and tilt limits in degrees for a PTZ camera is used to generate cells with an unique id. A PTZ camera can be thought as being inside a unit sphere. Pan and tilt values can be defined as a 2-dimensional surface. Thus we generate uniform cells on that 2-dimensional surface as seen in figure 2.4. The concept is the same for static camera but now it generates cells depending on supported mechanical pan and tilt values in degrees. This idea can be visualized with an example PTZ camera that has 360° pan and 180° tilt capabilities with 20 cells generated in each pan and tilt direction as seen in figure 2.5. Note that cell generation is not optimal as cell area near equator is bigger than near the poles. Generated cells for one PTZ camera from our prototype can be seen in figure 2.6.

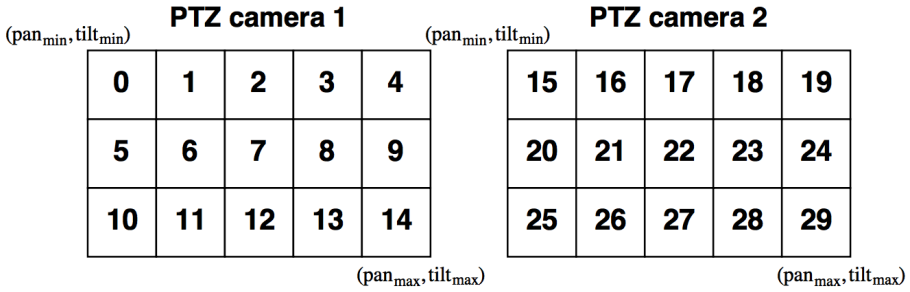

Figure 2.4: Example of generated cells with unique ids for two PTZ cameras. Here 5 cells was used in x-axis and 3 in y-axis. Cell generation is dependent on mechanical limit for pan and tilt values.

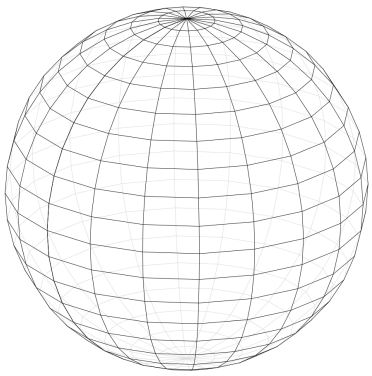

Figure 2.5: Illustration of a sphere divided into a grid of cells. A PTZ camera is assumed to be in the middle of the sphere with radius $r = 1$. When the PTZ camera moves, mechanical pan and tilt values are updated and visible cells from current field of view is derived. This sphere have 20 cells generated in each pan and tilt direction yielding a total of 400 cells.

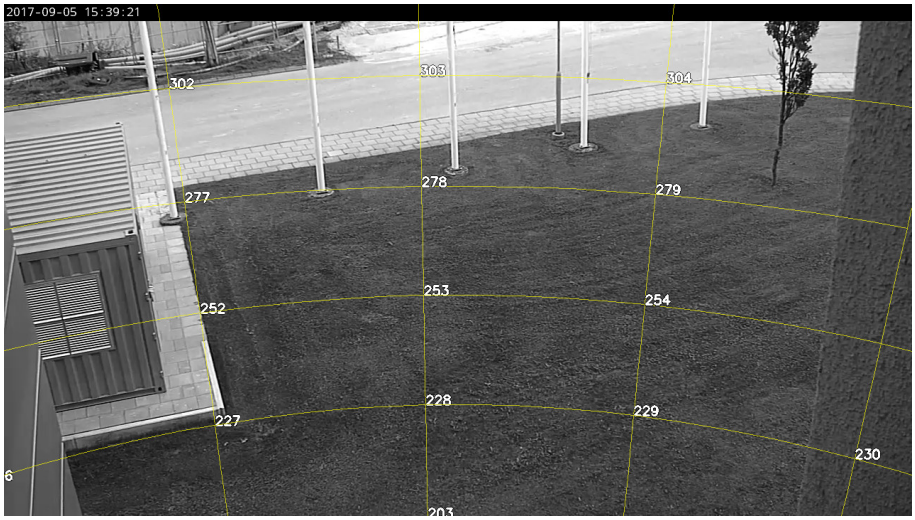

Figure 2.6: Cell generation for PTZ camera from our implementation shown inside camera stream. Mechanical limitation is $-180° \leq$ pan $\leq 180°$ and $-90° \leq$ tilt $\leq 0°$. 25 cells was generated in pan direction and 13 in tilt direction yielding a total of 325 cells. Only a subset of all cells is visible.

2.3 Overlap computation

Time is divided into discrete time frames. Any event happening is converted to the closest time frame. Frame length can be changed if needed. Events of interest is movement on cells. If there is movement in one cell at a specific time frame, the cell is considered as *occupied* at that time frame. A cell can either be in *occupied* or *unoccupied* state. If events of movement have been collected during T frames, each cell will have a binary vector of length T that stores all the states. These vectors are referred to as *occupancy vectors*. Overlap is found by comparing the cells occupancy vectors. The occupancy vector is be described with the following notation:

$$\mathbf{u}_i = \{u_{i0}, u_{i1}, u_{i2}, ..., u_{iT}\} \tag{2.2}$$

where u can be either 1 or 0 if occupied or not, i denotes the cell index and T is the number of total frames. A cell can either be present or not in each frame due to different reasons. One reason appears when a PTZ pan or tilts to a new position and some visible cells disappear which can cause contradicting data. Another reason could be information lost in the network. To fix this problem we define an availability vector $\mathbf{v}$ for each cell that is set to 1 if the cell was present in the frame or 0 if not. A cell is present if the cell's state is either occupied or unoccupied. Length of the availability vector is the same as occupancy vector. If the cell was not present in the frame assign 0. The availability vector acts as a filter, and is used to filter out cells that have not been active during the test. Reasons for being non-active is typically camera downtime or if not being visible in a PTZ camera. We use the following notation for availability vector:

$$\mathbf{v}_i = \{v_{i0}, v_{i1}, v_{i2}, \cdots, v_{iT}\}$$

The filtered occupancy vector can be described with:

$$\mathbf{o}_i = \mathbf{u}_i \wedge \mathbf{v}_i$$

The core of our method is exclusion. We use an unidirectional **exclusive or** when comparing between occupancy vectors:

$$o_{it} \ominus o_{jt} = \begin{cases} 1, & \text{if } o_{it} = 1, o_{jt} = 0 \\ 0, & \text{otherwise} \end{cases}$$

The unidirectional **exclusive or** is used on all the values throughout the occupancy vectors and then the result is summed up:

$$E_{ij} = \sum_{t=0}^{T} o_{it} \ominus o_{jt,} \qquad E_{ji} = \sum_{t=0}^{T} o_{jt} \ominus o_{it} \tag{2.3}$$

E_{ij} can be seen as the number of times cell i was *occupied* at the same time cell j was *unoccupied*. In order to retrieve a certainty measure C_{ij} and C_{ji}, regarding if cell i and j is overlapping, we first must calculate the number of times cell i and j was occupied.

$$S_i = \sum_{t=0}^{T} o_{it}, \qquad S_j = \sum_{t=0}^{T} o_{jt} \tag{2.4}$$

The next step is to calculate a measure of excluding data between cell i and j:

$$C_{ij} = \frac{S_i - E_{ij}}{S_i}, \qquad C_{ji} = \frac{S_j - E_{ji}}{S_j} \tag{2.5}$$

If cell i and j do not overlap, then $C_{ij} \to 0$ when $E_{ij} \to S_i$. Since the calculation of E_{ij} and E_{ji} values uses an unidirectional operator, E_{ij} and E_{ji} is not symmetric. Therefore we have two directed certainty values for each cell pairs. In order to strengthen the possibility of an overlap we check both directed certainty values. We decide if two cells overlap with the help of an empirically predefined threshold C^* and the following boolean function:

$$\text{Cell } i \text{ and } j \text{ overlap } = C_{ij} > C^* \wedge C_{ji} > C^* \tag{2.6}$$

Overlap between cells in the same camera is ignored. Change in frame length impact cell overlap.

2.4 Implementation

Our implementation, also called prototype, will use a centralized solution. Meaning each camera will stream its video stream and motion detection event data to a central computer for cell overlap calculation. See figure 2.7 for a brief prototype overview. We will refer one run of the prototype as a test. The test consist of several stages;

1. Data gathering.

2. Occupancy vector building (defined in equation 2.2).

3. Overlap estimation calculation.

Each stage requires specific input. The data gathering stage only needs a list of IP-addresses to the included cameras in the test. It then outputs a list of motion detection events for each camera. The occupancy building stage takes these lists as input and then produces occupancy vectors for each cell. The occupancy vectors are then used in the overlap estimation calculation stage.

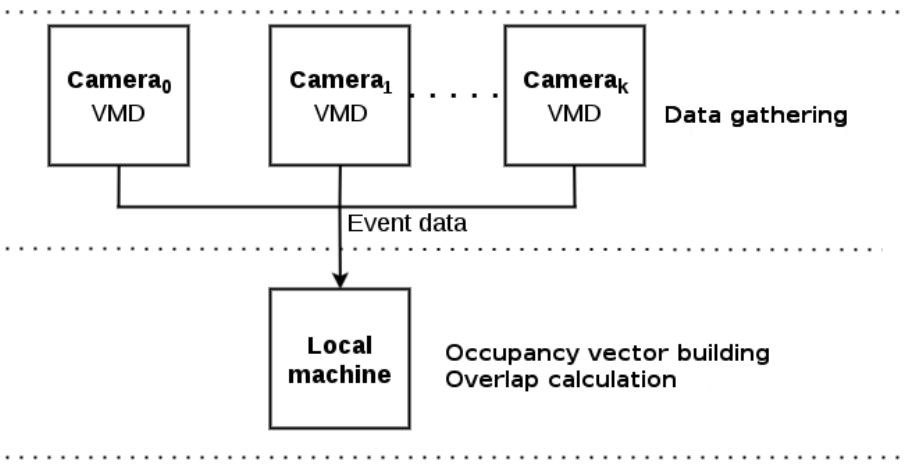

Figure 2.7: Brief prototype system overview.

2.4.1 Data gathering stage

The prototype utilizes Axis Video Motion Detection (VMD). VMD is an application installed on each camera and every time a motion is detected, an event will be sent from the camera to the computer that runs the prototype. The event contains a timestamp and normalized image coordinates for polygons to the bounding boxes surrounding the objects that triggered the event. This method works well with our requirements defined in 1.4 since VMD can run on existing camera network with little modification and does not disturb current surveillance.

When running the prototype, there will be a thread running for each camera included in the test. Each thread is responsible for collecting event data sent from its corresponding camera. The data gathering method is centralized and only data sent from the camera to the local machine are motion detection events:

```
{
  "UtcTime": "2017-05-18T12:03:05.443660Z",
  "polygons": "#00FF00
    0.0290534,-0.9780274
    0.1708984,-0.9780274
    0.1708984,-0.6521004
    0.0290534,-0.6521004
```

```
    0.0290534,-0.9780274;"
}
```

The event contains an UTC timestamp and a `polygons` string. The `polygons` string consists of a color in hex format and a list of five floating point number pairs representing a polygon's bounding box in normalized image coordinates. Note that the `polygons` string can contain multiple colors and floating point number pairs and is separated with a semi-colon, thus multiple bounding boxes can be received. VMD can be configured to minimize false alarms by adding multiple filters and ignore areas in the video stream. The color in the event data represent if the polygon was an alarm (red) or acknowledge and ignored (green) by VMD.

The polygon bounding box is always a rectangle, and we will hereafter refer to it as *movement rectangle*. The event data is specific to the current field of view for the camera. Depending on the type of camera, the movement rectangle coordinates must be converted to different coordinate systems in order to derive which cells is included in the movement rectangle during time frame.

For static cameras, movement rectangles must be converted to image pixel coordinates to know which cell was included in each movement rectangle. In static cameras each cell is always visible. The process to derive which cell is included from a movement rectangle follow this conversion in coordinates:

$$\text{normalized image} \Rightarrow \text{image pixel} \Rightarrow \text{list of included cells}$$

For PTZ cameras, movement rectangles must be converted in a slightly different way since cells is represented by mechanical pan and tilt values instead of image pixel coordinates. The process follow instead this pattern for conversion in coordinates:

$$\text{normalized image} \Rightarrow \text{spherical} \Rightarrow \text{list of included cells}$$

However, for PTZ cameras only a small number of cells are visible at a time. Depending on the current field of view for a PTZ camera certain cells may be visible or not as seen in figure 2.8. It is necessary to know which cells that were visible for each frame during the test. Without knowing which cells were visible, overlap could be over-written since incoming data could be considered contradicting data by our method. Also, we want to find overlap for cells that is only visible a fraction of the test.

2.4.2 Occupancy vector building stage

Useful data from events is timestamps and movement rectangle bounding box coordinates. For each gathered rectangle, we calculate which cells in the generated grid it covers. We have two methods of defining how a rectangle covers cells in the cell grid:

1. **Centroid method**: Calculate the centroid point of the rectangle. Find which cell this centroid point belongs to.

2. **Area method**: Find cells that is covered by the rectangles area above a certain coverage threshold.

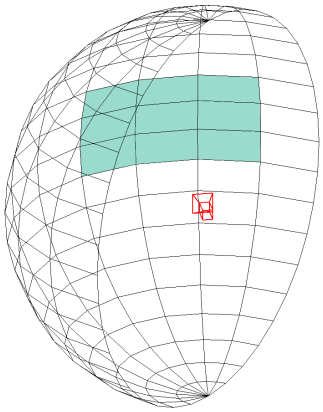

Figure 2.8: Illustration of a PTZ camera inside a half sphere. 9 cells is visible from the current field of view for the camera. Movement rectangles can only be inside any of the 9 cells. Mechanical limitation is $0° \leq \text{pan} \leq 180°$ and $0° \leq \text{tilt} \leq 180°$

Example of these two methods can be seen in figure 2.9. In the figure, a rectangle from an event has been drawn over a cell grid. The rectangle touches cell index 3, 4, 8, 9, 13, 14, but only 8, 9, 13 and 14 are regarded as occupied since 3 and 4 doesn't meet up to the minimum required area coverage threshold of 49%. If a cell is covered, we say that the cell is occupied at that time frame, specified by current timestamp. We will evaluate both methods later.

The goal is to create an occupancy vector for each cell. The element at each index in the occupancy vector represents the cell's state at that current time frame. The length of the occupancy vectors is calculated as:

$$\frac{\text{start} - \text{end}}{\text{frame length}} \tag{2.7}$$

Where *start* is the start time of the test, *end* is the end time of the test and *frame length* is the length of a frame, measured in seconds, in the test. We chose frame length 1 second, but this can be changed depending on real-world camera network environment and how well each camera's time is synced. The occupancy vectors is initialized as a long vector of zeros, and then filled with ones at indexes that have a corresponding timestamp. The mapping between an individual timestamp and occupancy vector index can be done in a similar way as equation 2.7:

$$\frac{\text{start} - \text{timestamp}}{\text{frame length}}$$

The timestamp can not be taken after the end of the test, or else it will be mapped to an index that exceeds the length of the occupancy vector.

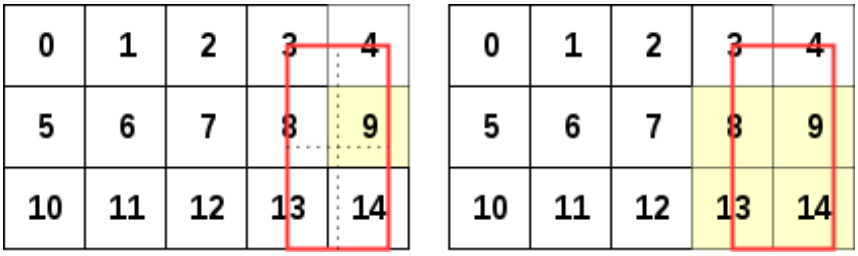

Figure 2.9: Cell coverage example. On the left is method 1, as described above, and to the right is method 2. Method 2 uses area coverage threshold 49%, meaning at least 49% of the cell area must be covered by the movement rectangle.

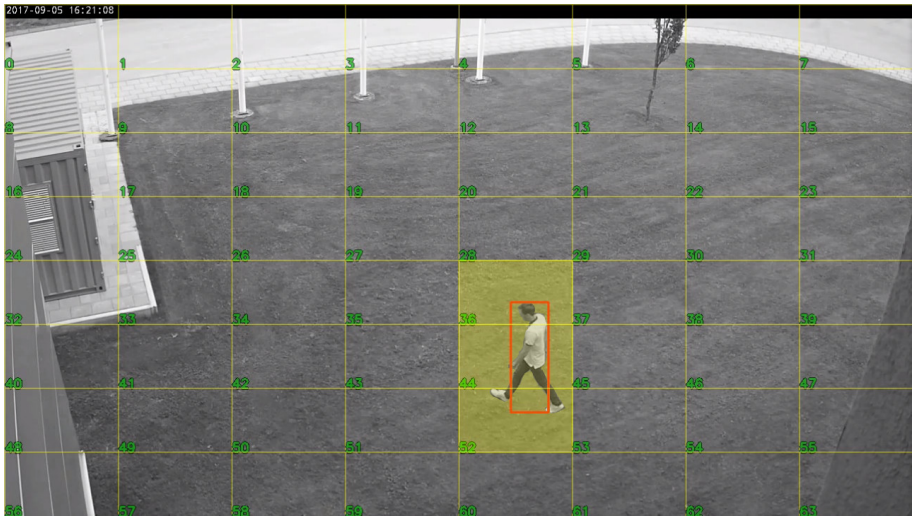

Figure 2.10: Example of a moving object with area coverage method threshold 5%. Cell 36, 44 and 52 is considered occupied at this time frame.

2.4.3 Overlap calculation stage

During this stage all occupancy vectors are compared with each other using equations 2.3, 2.4, 2.5 and 2.6 described in section 2.3. Result is a list of overlapping cells that the prototype has found. For example, consider the overlap found in figure 2.11, the prototype would output:

```
14: 15
15: 14
```

Note that from the output above there is also additional information not seen, i.e. all other cells not present in the list does not overlap with any cell. This information must be considered when evaluating our prototype. The output can directly be used to check against a predefined ground truth for evaluation.

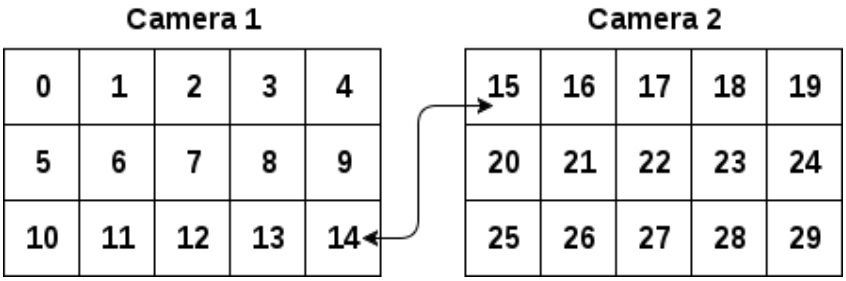

Figure 2.11: Example cell grid setup and overlap for two cameras.

2.5 Assumption

Our approach can only find shared field of view by movement inside the field of view. Thus it can not find shared field of view when e.g. observing walls or buildings. It is a reasonably assumption since surveillance usually is important in places where there is movement. However, movement in new places in a camera stream requires additional training and basically gives a new ground truth. For evaluation we can not assume new ground truth is generated after time.

Currently our approach is centralized for cell and occupancy generation but can easily be decentralized by allowing each camera to be responsible for it's own cell generation and handling of motion detection. Each camera then only sends out occupied cell id together with corresponding timestamp.

Our prototype assumes the surveillance network is online using configured motion detection for the current scene to minimize detections that generates contradicting data (e.g. flag swaying in one camera while a person is walking in another camera which could generate shared field of view in our algorithm).

Our prototype also assumes all cameras in the network have clock synchronized. Any clock drift can be handle in the algorithm by padding additional frames in time or change frame length, however it could generate false positives. It is a reasonably assumption to make that every camera is synchronized.

Chapter 3
Evaluation

In this chapter we evaluate our prototype of finding overlap. We start by introducing F1-score, together with precision and recall, as a benchmark tool. Then we explain our experimental setup. We also explain some evaluation problems and lastly we evaluate the prototype.

3.1 F1-score

F1-score measures a binary classification test's accuracy. With F1-score one can compare different binary classification methods. F1-score is the harmonic mean of precision and recall. Precision is a fraction of relevant items that has been selected over the total amount of relevant items, while recall is a fraction of how many relevant items are selected from the total amount of relevant items. The possible outcomes from a binary classification, compared with a ground truth, is:

- *True positive (tp)*: Test predicted true. Ground truth is true.
- *False positive (fp)*: Test predicted true. Ground truth is false.
- *False negative (fn)*: Test predicted false. Ground truth is true.
- *True negative (tn)*: Test predicted false. Ground truth is false.

From this formulate precision and recall as:

$$\text{Precision} = \frac{\sum \text{tp}}{\sum \text{tp} + \sum \text{fp}}$$
$$\text{Recall} = \frac{\sum \text{tp}}{\sum \text{tp} + \sum \text{fn}}$$

Lastly we can compute F1-score as:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F1-score work great in our case since cells can either be overlapping or not. With a predefined ground truth, precision and recall can be calculated. Thus we can compare different ways to assign occupancy for a cell at a timestamp, or change the algorithm and still obtain an accuracy score.

F1-score is not entirely optimal since true negatives is ignored in precision and recall. True negatives is interesting because our algorithm can predict no overlap, while the ground truth state no overlap. This is also relevant information. To solve this we can use accuracy as:

$$\text{Accuracy} = \frac{\sum \text{tp} + \sum \text{tn}}{\sum \text{total population}}$$

where total population is the total number of relevant items for the test. In our case total population is the number of overlapping and non-overlapping cells in ground truth. Now with F1-score, together with accuracy, we can compare any method for finding overlap efficiently.

3.2 Experimental setup

It is possible to derive different results depending on which thresholds and variables are chosen. The list of thresholds and variables that can be changed are:

- r_i = Number of cell rows in camera i.
- c_i = Number of cell columns in camera i.
- a = Cell area coverage threshold (see figure 2.9).
- C^* = Overlap certainty threshold (see equation 2.6).
- f = Frame length.

There are a number of problems to take into account when selecting these values. If r_i or c_i are changed, then the area of the cells changes, which will affect the efficiency of threshold a. The cameras spatial properties should also be taken into consideration before running the test. If the camera is placed far away from moving objects, then the objects will appear smaller in the camera's field of view, and if r, c or a are too high, then the cell might not ever register movement as the cell is too big and require significant area coverage from movement rectangle to be considered occupied. We want to test which values are the best for different setups. The initial thought is to select r_i and c_i so that most moving objects have roughly the same size as a cell.

The frame length f should be as short as possible since the theory builds upon registered movement that happens at the same time. However, if the value of f is too small, the time difference between the internal clocks in the cameras will affect which frame events

will be part of to a much greater degree. That is not a wanted behavior. Also, the smaller value of f, the longer the length of the occupancy vectors will get, which will affect memory usage and the computational time negatively.

When testing, there is a problem of evaluating the result since it is hard to know the ground truth for the current setup of cameras. It is possible to place the cameras in such a way that you know a certain set of cells will overlap, but it requires very accurate camera placement.

Tests will be carried out multiple times but with different values of r_i, c_i, a, C^*, f. There is a problem of achieving the exact same spatial properties between the test. This would require that movement happens at the exact same and time during every test and the cameras cannot move. We consider this an almost impossible requirement to achieve consequentially. It is also time consuming to have to wait another full test length when you just want to change one variable. This problem is solved by saving all the event data during the *data gathering stag*e (see 2.4.1), and then run the *occupancy vector building stage* and *overlap calculation stage* of this data. This way, a test that took several hours to collect all the event data, can be run multiple times, taking only a few seconds.

There are multiple ways of placing two cameras so that their field of view overlap. We do not have time to test every interesting possibility of camera placement. Therefore we will focus on one test scenario, where it is fairly simple to generate a ground truth. This test scenario will be referred to as *full overlap*, meaning that both cameras shares 100% of its view with the other camera.

3.3 Problem with evaluating result

When evaluating our prototype there is a problem in having a valid ground truth. Cell width and height in a cell determines which other cell it might overlap with, thus it exists many ground truth for only two cameras if there is a known overlap in their field of view. There does not exist any public database with camera streams, together with motion data events, and polygons indicating which part of each camera's stream share same field of view with other cameras. We could either build said database ourselves, or find ways to position cameras with a known ground truth beforehand. There is different kinds of ground truth:

- Specific angle (e.g. half of the camera's field of view is overlapped with another camera).
- Spatial setup for cameras with no overlap.

We will focus on using specific spatial angles to get a known ground truth between two cameras. We should only evaluate with ground truth for overlap, not for when or how overlap was found. Lastly if there was no movement in a cell during the test, we should assign no overlap in the ground truth to reduce false-negatives lowering our recall for F1-score. We will be clear and consistent when we do this.

We have to assume a fixed ground truth. In a real world application there could be regions in a camera's stream in which there is no movement during training (e.g. locked doors in a corridor). These regions could change and suddenly involve movement after

training. Thus the ground truth has been changed. During evaluation we must assume a fixed ground truth to compare methods or configurations to our algorithm.

There is also a subjective interpretation of cell overlap when generating a grid of cells over camera streams. As seen in figure 3.1, camera one can have zero or four cells that overlap depending on visual ground truth (e.g. is some part of ground truth covered enough to consider sharing same field of view or must the whole ground truth be visible in the cell).

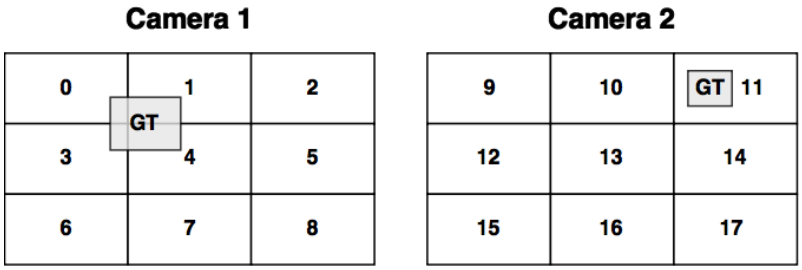

Figure 3.1: Illustration of ground truth when generating a grid of cells over two cameras. GT is ground truth, i.e. the GT region in camera 1 is sharing the same field of view in GT region in camera 2. One can argue which cell actually overlap with ground truth depending on visual interpretation of the camera scene.

3.4 Validate prototype theory

This section aims to validate that the underlaying theory for the prototype works as a method for overlap calculation. We do this in an environment where there is no problems with ground truth. This environment is simulated by only using one camera, and then duplicate its output test data and assign it to a fictional camera, as if the fictional camera also recorded exactly the same events. This ensures a full overlap and the ground truth is thus known.

The prototype should output a graph almost identical to the ground truth. When using some thresholds (a, C^*), it was discovered that the precision could be as good as 1.0, but the recall was correlated to how many cells that has registered movement. This is mentioned in the limitations, we can only find overlap between cells that have registered some movement. The process of finding good thresholds is fairly simple, since we are able to run the exact same test multiple times, we can just change these threshold between each iteration and then see which threshold governs the best F1-score.

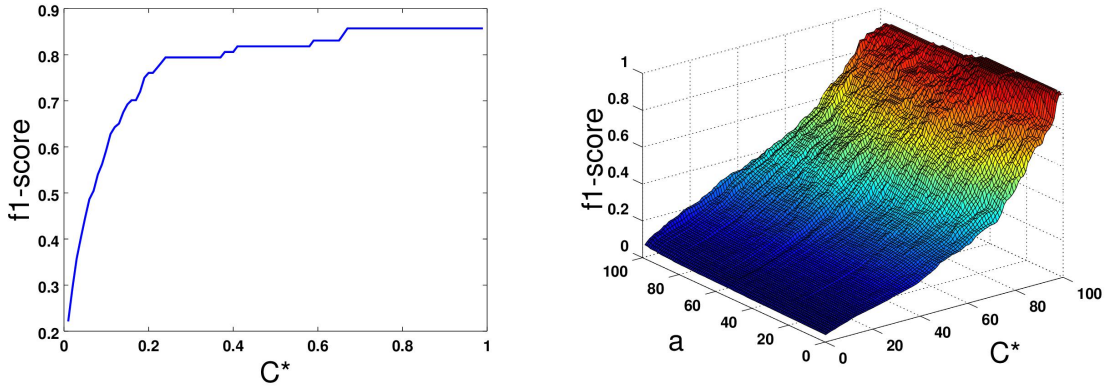

Figure 3.2: Investigating different thresholds across two different cell occupation methods. To the left centroid method is shown and to the right area method from section 2.4.2 respectively. C^* and a are parameters defined in section 3.2. Notice how in the area method, the search for the optimal parameter is more complex.

The results presented in figure 3.2 are both produced from the same test data and with the same parameters, but with different methods for calculating which cells are covered by movement rectangles. The fixed parameters for both tests are; the number of cameras, number of cells, frame length, and test length. The number of cameras are 2 (one of them is a fictional duplicate), and the total number of cells for these two cameras are $r_0 * c_0 + r_1 + c_1 = 12 * 6 + 12 * 6 = 144$. The frame length is $0.5s$ and the test length is $1h$. During the one hour the test was running, all areas of the field of view has experienced some kind of movement.

When using the centroid method, we discovered that the optimal thresholds are $C^* = 0.67..0.99$, all producing an F1-score of 0.86. When using the area method, we have to find the optimal area threshold a and overlap threshold C^* at the same time. In this case the optimal values gained a perfect F1-score of 1.0 throughout many values of a at higher thresholds $C^* > 0.9$.

There are a few reasons the area method wins over the centroid method in this particular test. It has a slightly better recall, without any loss of precision. The reason for the better recall is that it is better at covering more cells. In this test, one movement rectangle is most likely to be able to cover multiple cells with the area method, whereas the centroid method is defined to only be able to activate one cell per movement rectangle. If we would discard all cells that never gets covered by movement rectangles with the centroid method, we would get a better result, as presented in figure 3.3. There we get an F1-score of 0.99. This new result cannot directly be compared with the result presented in 3.2 since the tests included cells and ground truth has been altered, but it shows that the centroid method is also viable when enough cells gets covered.

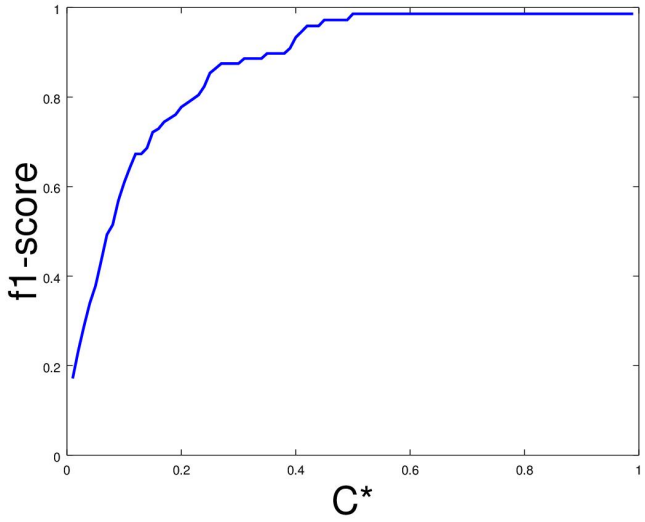

Figure 3.3: Finding threshold with the centroid method where cells that never has been covered by movement rectangles has been discarded from the test.

The results presented in this section has proven that our theory and implementation of prototype works as intended. The next step in the evaluation process is to test the prototype in a more realistic environment.

3.5 Full overlap between two cameras

This test will show how both methods performs when we do not have a perfect environment, i.e. we do not duplicate the test data to a fictional camera. The requirement for time synchronization and equal configuration between the two cameras are now much greater. We do also have to assume that both installations of VMD behave the same on both cameras. In order to achieve that we use two cameras of the same model and two installations of VMD with the same version number. Both cameras start out with a freshly installation of VMD, and the test starts when both cameras has a newly started VMD application running. Both cameras captures the same scene, and should thus have full overlap. It is much harder to ensure that there is full overlap between the two cameras since we have to ensure that they have the exact spatial properties. We can of course not place both cameras in the exact same location, so we have to place them next to each other and then capture movement rectangles from objects moving from a long distance from the cameras so that they have roughly the same pixel coordinates in both cameras.

3.5.1 Centroid and area method

In figure 3.4 we can see the best threshold for the centroid method regarding this scenario. There is two lines, the green one (lower) belongs to the test where no cells has been discarded, and the blue one (upper) belongs to the test where we have discarded cells that has never been covered by movement rectangles. The offset between the two lines depends

on the proportion of cells that has never been discarded. The optimal threshold is a bit

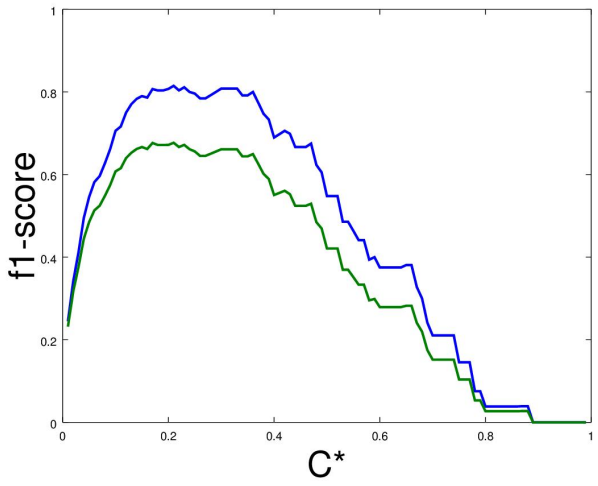

Figure 3.4: Finding threshold with the centroid method. Lower line has not any discarded cells. Upper line has discarded cells that never has been covered by movement rectangles.

lower than expected, this is due to the noisy environment the cameras was capturing. In the previous chapter 3.4, there was no problems with noise, and the optimal threshold was thus much higher. In figure 3.5 we can see the best values for the thresholds a and C^* when using the area method. The left plot shows the F1-score for the test where no cells are discarded. The best F1-score is 0.34 at $a = 5\%$, $C^* = 82\%$. If we discard cells, we get a better result, as seen in the right plot. There we get the best F1-score of 0.5 at $a = 76\%$, $C^* = 62\%$. In this test scenario, the centroid method was a clear winner. The reason for

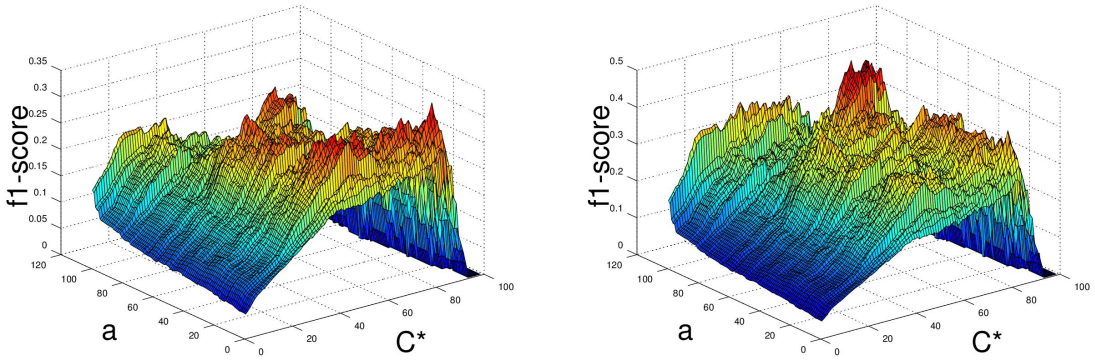

Figure 3.5: Left plot shows F1-score at different thresholds using the area method. Right plot shows F1-score at different thresholds using the area method and with disposal of cells that never gets covered by movement rectangles.

this is that the area method has problems with false-positives. With the area method we

find a lot of overlapping cells that is neighbouring cells with true overlap. This is all due to the way that the area method is able to cover multiple cells per movement rectangle. We thought it was possible to choose an area threshold a so that this problem wouldn't occur, and that is possible to some degree. But after trying a range of 100 area thresholds, it still doesn't beat the centroid method.

3.5.2 Test length

There is yet to explore how our prototype learns overlap through time. It is however, hard to present the result in a graph where time is used as a dimension on one of the axis, since we have no control over how many events are produced during one time instant. Therefore we will show how our prototype learns overlap through how many events it has registered. The scenario used in this test consists of 26×10^3 events, produced during one hour. By using the centroid method we have calculated the F1-score after each gathered 10^3 events and it is presented in figure 3.6.

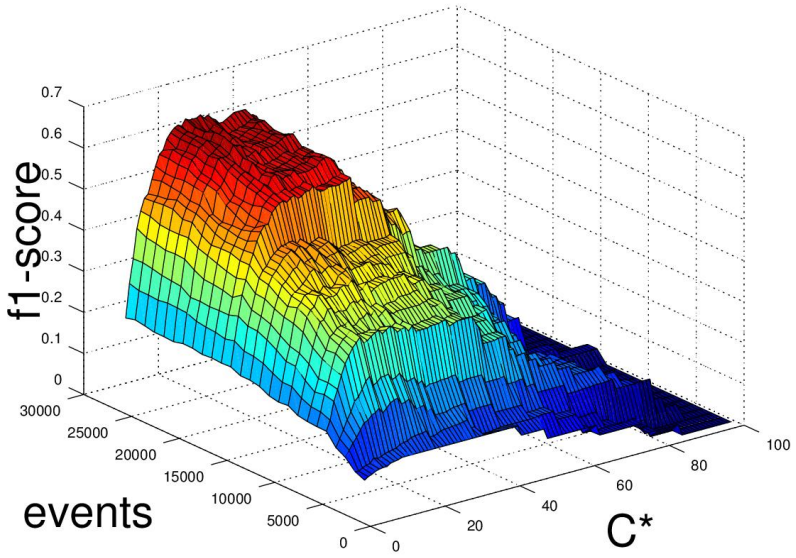

Figure 3.6: Finding the best threshold after each gathered 10^3 events.

We can see that the F1-score increases with the number of sampled events. But the increase stagnates after after about 20×10^3 events. This result is gathered without disposal of cells since the set of non-covered cells differ between throughout the plot.

3.5.3 Frame length

Another variable that can affect the F1-score is the frame length. Testing has until now utilized a frame length $f = 0.5$. We will use the centroid method for this test again since it has the best performance in this scenario. Figure 3.7 shows two plots, the left one has frame lengths from $f \leq 1s$ and the right plot has frame lengths $f \geq 1s$. We can see that frame

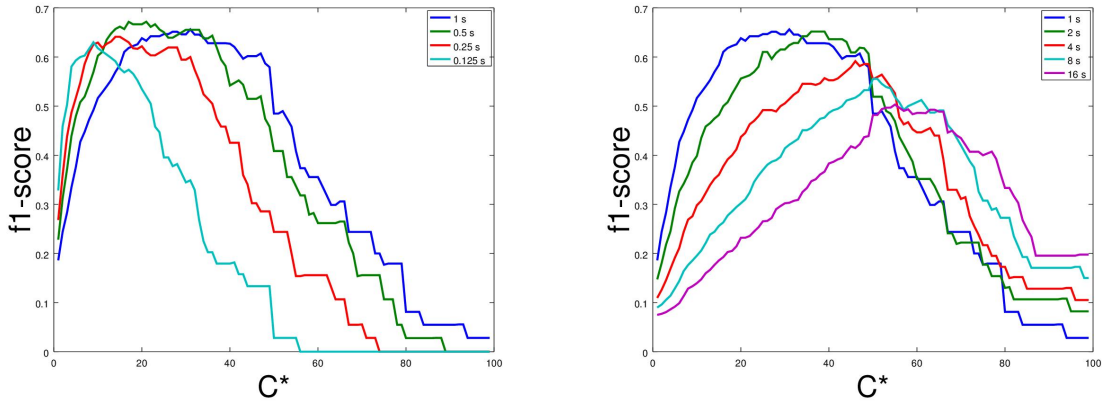

Figure 3.7: Finding the best threshold with the centroid method and different frame length.

lengths below 1s offsets to the left while still remaining roughly the same peak F1-score. If the frame length is shortened, we must also use a lower value of C^*. We recommend to not use a frame length lower than 0.5s since we do not get any better F1-score and shorter frame length has some drawbacks like more memory usage and worse tolerance of C^*. The tolerance for bad values of C^* is very low when looking at the graph for frame length 0.125s. Any threshold over 0.5 will result in a F1-score of 0. If we instead take a look at the right plot we can see that if we increase the frame length, bad guesses of C^* is more acceptable, but for the price of an overall lower F1-score. By remembering the occupancy vectors defined in equation 2.2, we can explained the behavior of different C^* tolerance for different frame length. Lower frame length results in higher difference between the cells occupancy vectors, and higher frame length results in the opposite. An extreme case would be if the test length and frame length would be the same, the test would only have one frame, and all cells that have registered movement would be equal. By taking both plots into consideration, we can conclude that a frame length between 0.5 and 2 is the best choice.

3.5.4 Cell size

It is hard to consistently test how the cell size affects the F1-score since an increase or decrease also affects the total number of cells in the test. In general, if fewer cells are included in the test, the F1-score gets better because of the simple ground truth. We have tested this, and it is true for both the centroid and area method. But if we would like to include more cells in the test, which method would perform the best. Until now, the tests have utilized a cell size relatively near the average area of movement rectangles, and the centroid method had the best performance. The area method should, at some point, have better performance than the centroid method if we would decrease the cell size. If we consider the smallest possible cell size, the pixel perfect cell size, the centroid method might not work at all whereas the area method can still perform. This threshold between when area method is better than centroid method could possibly be found, but we omit it in the report because of the complexity and limited time.

Chapter 4

Conclusions

4.1 Discussion

In this thesis we purposed an unsupervised learning algorithm to find overlap between multiple cameras field of view. The algorithm can also be seen as either online or offline depending on implementation but we chose evaluating an offline version. Our method can also be seen as incremental learning with streamed data. A trained model can be loaded and continued trained on.

Given the problem of evaluating our results and prototype, we still managed to show that our method works and solves the problem stated in problem definition. While evaluation only was for two cameras, it can support many cameras in a network. Only limitation is memory which increases by the total number of cells in the network.

In a real-world solution each camera would probably be consistently in training mode since all movement regions in all cameras would probably not be covered during training. Also in a real-world environment new cameras could be installed in the network and our method has to find any new overlap by being continued trained on. Our method can not handle massive crowd. One way to solve this is by disable training when there is multiple events in one time frame, to suppress noise in computations.

We see potential in using our method to add an additional knowledge on top of a system of network cameras. New use cases arises and effectiveness in surveillance is increased. One common use case is using a panorama camera and PTZ to collaborate. An operator can monitor a panorama camera and if an interesting object is found, the operator can click on a cell and a nearby PTZ will zoom in to that cell in its own field of view. In short the operator will be given a new perspective.

4.2 Limitation

How we generate cells is a crucial step of our method. The way we generate cells for PTZ camera is not optimal. In figure 2.5 we can see that cells near the poles has significantly lower area than cells near the equator. Thus we waste memory since these cells must still be present in our matrices for computation. A better approach would be to generate cells with uniform area. However, it would change the geometry of the cells and thus maybe not be optimal for each scene in a real-world environment. A numerical method for generating cells with roughly uniform area could also work.

In a worst case scenario a camera network consist only of PTZ cameras. In order to find cameras sharing the same field of view, all cameras must pan and tilt to cover all combinations. To cover all combinations is a brute-force search. The best way to tackle this problem is that PTZ cameras must consistently be in online training mode with a "good enough" mentality for real-world surveillance application.

Two or more cameras with non-overlapping field of view should not have cells that are occupied at the same time frequently. If that is the case, false positives is generated.

4.3 Future work

The generated cells for PTZ cameras is not optimal since cells closer to the equator has bigger area than cells near the poles. Each cell requires memory and put an unnecessary load to our prototype as well as worse performance since cells near poles is smaller. Future work would look into generate cells with approximately uniform area when projected on a sphere. One way is to use inversed cylindrical equal-area projection since we have our 2-dimensional plane of limited mechanical pan and tilt values. Different cylindrical equal-area projections exists [4].

Future work would also investigate on using adaptive cell sizes. E.g. instead of generating a fixed number of cells, specific to each camera type, we would generate cells depending on movement. Each camera is responsible to cover as much as is possible with a fixed number of cells. In a camera's field of view, some part has no movement at all and some part has a lot of movement. Areas with no movement should have as few cells as possible and the opposite of areas with high movement. With adaptive cell sizes, we would also solve the problem of having cells with too little area near the poles for PTZ cameras. The adaptive cell size method could be also used on both PTZ cameras as well as on static cameras instead of having two different solutions which our prototype uses.

Our prototype uses a centralized solution. Each camera streams to a centralized computer to generate occupancy vector and compute overlap. In a real world solution with many cameras in a network, a centralized solution may be impossible due to the amount of bandwidth and memory required.

In a distributed solution each camera would get a fixed number of cells to generate unique cells with. Then each camera sends only which cell was occupied at a specific frame to a localized computer. The localized computer would then compute overlap between cells. This approach would save a lot of bandwidth and abstract out cell generation. Each camera is responsible for handling, depending on its own capabilities, how a cell is represented.

In a decentralized solution, combining with a distributed solution, a few cameras would act as a server node which receives local cell id occupancy and generate a local overlap matrix. Then the local overlap matrix is passed to another server camera which compares with its own matrix. In this way less memory is required per server node than a centralized solution. Operation is also more stable as server nodes can crash without crashing the whole program. However it increases complexity and could increase bandwidth as potentially redundancy information is passed around.

If we had more time, we would build a database of saved video streams for each camera and manually annotated which region overlap with another camera with polygons. Then when generating a grid, one could check if a cell is inside ground truth polygon and if so, which cell it should overlap with. This way we could generate a ground truth database for easy evaluation. As this field is not well investigated, there exist no public ground truth database. Meanwhile there exists ground truth databases for computer vision, machine learning, etc. but not for this specific problem. This part could easily be a thesis itself.

Bibliography

[1] Niall Jenkins. Video Surveillance Camera Installed Base Report - Industrial , Security and Medical | Video Surveillance. *IHS*, pages 12–13, 2015.

[2] Zhaolin Cheng, Dhanya Devarajan, and Richard J Radke. Determining Vision Graphs for Distributed Camera Networks Using Feature Digests. 2007, 2007.

[3] Henry Detmold, Anton Van Den Hengel, Anthony Dick, Alex Cichowski, Rhys Hill, Ekim Kocadag, Katrina Falkner, and David S Munro. Topology Estimation For Thousand-Camera Surveillance Networks. 2007.

[4] John P. Snyder. Map Projections: a Working Manual. Number no. 1395 in U.S. Geological Survey professional paper, pages 76–85. U.S. Government Printing Office, 1987.

Appendices

Appendix A
List of Abbreviations and Expressions

Network camera
A static or dome camera with its own IP address and built-in wired or wireless network capability.

PTZ
Pan-Tilt-Zoom. Mechanical camera movement capability.

VMD
Axis Video Motion Detection. Plug-in program to Axis network cameras to allow for motion detection and detect bounding box of moving objects.

Movement rectangle
Output from VMD. When the program register movement, it outputs the image coordinates to the smallest rectangle that surrounds the found movement of an object.

Timestamp
Output from VMD. Specifies at what time in UTC format a movement rectangle was recorded.

Test data
All movement rectangles and timestamps outputted from VMD running on an camera during one test.

MASTER THESIS Finding Field of View Overlap by Motion Analysis
STUDENT Fredrik Sydvart, Hampus Altvall
SUPERVISOR Cristian Sminchisescu (LTH), Fredrik Andersson (Axis Communications AB)
EXAMINER Kalle Åström (LTH)

Picture analysis free approach for finding shared field of view between network cameras

POPULAR SCIENCE SUMMARY **Fredrik Sydvart, Hampus Altvall**

Network cameras has significantly increased in the recent years. Each camera is independent and its monitoring task can easily be remotely altered. This work focuses on increasing effectiveness of a system of cameras by adding additional knowledge on top of the system and by utilizing multiple cameras when a shared field of view has been found.

With the rise of network cameras over the years, human operators responsible for monitoring is flooded with cameras. In order to filter out relevant information, intelligent software analyzing video streams becomes more important.

If a terrorist is moving in one camera, and the terrorist's face is not shown in the camera, an operator might desperately look if there is another camera nearby monitoring the same object. This relies on that the operator knows the camera environment. For instance, the city. We present an approach where cameras automatically figures out which camera shares the same field of view, that is, what a camera is seeing, with another camera. In this way, the operator can get a list of cameras monitoring the terrorist in multiple perspectives and select the one required for the job.

Our approach is based on movement. Imagine you have many cameras in a closed network. In each camera you divide the field of view into a grid of cells. When movement is detected, a bounding box surrounding the object is generated. This bounding box is used to derive which cell is considered occupied at the current time. Our approach uses contradicting proof to find connectivity between cells. If one cell is occupied at a time and another cell is not occupied at the same time, then those two cells can surely not observe the same thing. If this is done over a significant time and there is enough movement across all cameras in the system, we can find cameras sharing the same field of view.

There is multiple uses cases our solution solves. One powerful use case is switching perspectives, another one is optimizing camera installation as each camera cost money to manufacture, install and maintain. If too many cameras observe the same thing, this is not optimal and waste resources. By also implementing tracking, that means following an object's path across multiple cameras, an operator can also follow that same object in other cameras field of view and thus maybe identify a terrorists face to alert the public.

Our prototype showed very promising results and shared field of view could be found very quickly as long as there is movement in each camera stream.