Visibility Planning: Predicting Continuous Period of Unobstructed Views

Ser-Nam Lim, Larry S. Davis, and Yung-Chun (Justin) Wan

Computer Vision Laboratory, UMIACS, University of Maryland, College Park
{sernam,lsd,ycwan}@umiacs.umd.edu

Abstract. To perform surveillance tasks effectively, unobstructed views of objects are required e.g. unobstructed video of objects are often needed for gait recognition. As a result, we need to determine intervals for video collection during which a desired object is visible w.r.t. a given sensor. In addition, these intervals are in the future so that the system can effectively plan and schedule sensors for collecting these videos. We describe an approach to determine these visibility intervals. A Kalman filter is first used to predict the trajectories of the objects. The trajectories are converted to polar coordinate representations w.r.t. a given sensor. Trajectories with the same angular displacement w.r.t. the sensor over time can be found by determining intersection points of functions representing these trajectories. Intervals between these intersection points are suitable for video collection. We also address the efficiency issue of finding these intersection points. An obvious brute force approach of $O(N^2)$ exists, where N is the number of objects. This approach suffices when N is small. When N is large, we introduce an optimal segment intersection algorithm of $O(N \log^2 N + I)$, I being the number of intersection points. Finally, we model the prediction errors associated with the Kalman filter using a circular object representation. Experimental results that compare the performance of the brute force and the optimal segment intersection algorithms are shown.

Keywords: Sensor planning, visibility planning, scheduling, filtering, surveillance

1 Introduction

We describe asymptotically efficient algorithms for controlling a collection of surveillance cameras to acquire video sequences of moving objects (people, vehicles), subject to visibility, ground resolution and positional constraints. The problem is motivated by the following surveillance scenario:

We are given a collection of calibrated surveillance cameras, each of which can be independently panned, tilted and zoomed. They must be controlled to acquire surveillance video over a large surveillance site, which can most simply be modelled as a large patch of ground plane, possibly annotated with the locations of specific regions of interest (e.g., regions near the entrances to buildings, receptacles such as trash cans, or regions defined dynamically as vehicles enter and stop in the site).

Each camera has a field of regard, which is the subset of the surveillance site that it can image by controlling its pan and tilt. A field of view of a camera is the image obtained at a specific pan/tilt/zoom setting and is generally much smaller than its field of regard.

As people and vehicles move into and through the surveillance site, the cameras are to be controlled to acquire sets of videos that satisfy temporal and positional constraints that define generic surveillance tasks; these surveillance tasks could be represented in a suitable temporal logic, although we do not consider their representation in this paper. Examples of typical surveillance tasks are:

1. Collect k seconds of unobstructed video from as close to a side angle as possible for any person who enters the surveillance site. The video must be collected at some minimal ground resolution. This task might be defined to support gait recognition, or the acquisition of an appearance model that could be used to subsequently identify the person when seen by a different camera.
2. Collect unobstructed video of any person while that person is within k meters of region A. This might be used to determine if a person deposits an object into or takes an object out of region A.

One could imagine other surveillance tasks that would be defined to support face recognition, loading and unloading of vehicles, etc. Additionally, one would expect that there would be tasks related to system state maintenance - for example, tasks to image a person or vehicle to obtain current position data to update a track predictor like a Kalman filter; or tasks to intermittently monitor regions in which people and vehicles can enter the surveillance site. We would like to efficiently schedule as many of these surveillance tasks as possible, possibly subject to additional constraints on priority of the tasks. Finally, one would need some mechanisms to verify that the tasks have been successfully completed (i.e., that in fact we have obtained unobstructed video of some given person) so that it an be determined if the task has to be rescheduled (which, of course, will not always be possible).

The information that would be needed by any such scheduling algorithm would include the future time intervals during which each object being tracked through the surveillance site would be (probabilistically) visible from each of the surveillance cameras - i.e., would be in its field of regard and not occluded by another moving object or a fixed object in the site. The prediction of these visibility intervals must take into account the positional uncertainty of the moving people and vehicles, which generally increases as time progresses. We refer to this problem as the visibility planning problem. In our implementation, we use a simple Kalman filter to predict linear trajectories and their uncertainties. The scheduling algorithm must also take into account latencies associated with camera control. Practically, predictions need to be done over a time interval that is at least some multiple of the camera latencies, and is bounded above by increasing positional uncertainties of the objects in the site. Typically we would perform predictions over a time interval of 2-5 seconds in the future, so that the visibility analysis, task selection and camera control must be accomplished in some fraction of that time.

There is an obvious brute force solution to the visibility planning algorithm, which would have computational complexity $O(N^2)$, where N is the sum of the number of moving and fixed objects in the site. For a small number of moving objects, this brute force algorithm would be efficient enough, but as the number of moving objects increases, it could lead to a system bottleneck. We describe, then, a more efficient $O(N \log^2 N + I)$ algorithm based on an optimal segment intersection approach to solve the visibility planning problem. Here I is the number of reported intersections.

We start by providing a very brief review of research on visibility planning; there is an extensive literature on visibility analysis and planning in the graphics, computational geometry and robotics literature, and our review is not comprehensive. We then describe an efficient visibility planning algorithm for point objects, which does not take into account positional uncertainty. Then, we extend the representation so that we can model time-varying positional uncertainty or the physical extent of an object. Finally, we present the results of some simulation experiments that are designed to determine the problem size at which our approach becomes more efficient than the brute force algorithm.

The paper is organized as follow: Section 2 will briefly describe the background for the analysis in this paper, Section 3 will describe how line of sight visibility planning can be performed efficiently, Section 4 describes how the approach can be extended to circular representation of objects. Finally, Section 5 and 6 will give results and discussions.

2 Background

Some notable previous work on visibility planning can be found in [1], [2] and [3]. They used similar approaches in maintaining data structures of boundary surfaces, 3D free space that is not occupied by objects and tangential rays.

Updating these data structures are triggered by certain events, for example [1] defined "peek events" when an object just becomes visible as a result of sensor motion. Their approach is closely related to the filtering method that we use. Instead of waiting for events to trigger updates, we predict events using technique such as Kalman filtering described in [4, 5]. Updates are triggered, in our case, when the prediction errors become large. Space carving approach to create visibility space is also described in [6]. Other work includes [7], where visibility graphs with complexity $O(N^2 \log N)$ are described. A survey of sensor planning can also be found in [8].

3 Visibility Model

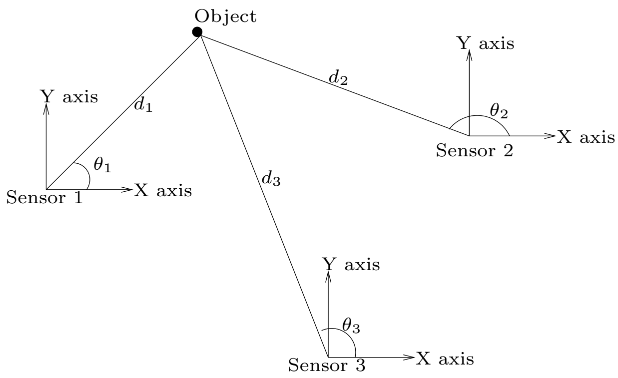


Fig. 1. The coordinate systems used in the visibility model

In this section, we describe the visibility model used to determine whether the moving line of sight from a sensor to some object intersects any other objects during the time interval t_0 to t_1. This visibility model is used to determine subintervals between t_0 and t_1 that an object is unobstructed from a given sensor's view. The visibility model represents the objects as 2D points on the ground plane. In Section 4, we will generalize the object representation to circles to model object size and limited uncertainty. Let N be the number of objects in the scene and M be the number of sensors, located on the ground plane, also referred to as the plan view. M polar coordinate systems are used to represent the location of N objects w.r.t. each of the M sensors. Line segments connecting each sensor to each object are created during initialization. Each line is represented in polar coordinate (d, θ), where d is the distance of the object from the sensor and θ, between $-\pi$ and π, is the angular displacement the line makes with the sensor and which is available from the plan view. Given a set of objects with the same θ w.r.t. a sensor at some time, only the object nearest to the sensor is unobstructed[1]. The coordinate systems are illustrated in Figure 1. In the figure, $M = 3$ and the coordinates of the object are (d_1, θ_1) w.r.t sensor 1, (d_2, θ_2) w.r.t sensor 2 and (d_3, θ_3) w.r.t sensor 3. It is clear that at any time instant, we can compute the coordinates of an object w.r.t. a sensor given some prediction model for the motion of that object.

For each object O_i, a time varying function $f_{si}(t) \rightarrow \theta$ that describes its trajectory w.r.t sensor s is computed. We assume that each $f_{si}(t)$ is represented by a straight-line trajectory[2]. Given some time instant t, $f_{si}(t)$ returns the angle, θ, that O_i makes with s. Consequently, a set of functions $F = \{f_{si}(t) \mid s \in [1, M], i \in [1, N]\}$ can be created. The general form of $f_{si}(t)$ is

$$f_{si}(t) = \arctan2((1 - \hat{t})y_1 + \hat{t}y_2, (1 - \hat{t})x_1 + \hat{t}x_2) \tag{1}$$

where (x_1, y_1) and (x_2, y_2) are the current and ending positions of the predicted trajectory of object O_i, $\hat{t} = \frac{t_1 - t_0}{\Delta t_i}$, t_0 is the time instant when i is at (x_1, y_1), t_1 is the predicted time instant when i is at (x_2, y_2), $\Delta t_i = t_1 - t_0$, and $\arctan2(y, x)$ is the four-quadrant inverse tangent function over the range $-\pi$ to π. Note that for mathematical convenience, we transformed the coordinates to the plan view so that the sensor location in the plan view becomes the $(0, 0)$ position. We can rewrite Equation 1 as

$$f_{si}(t) = \arctan2((t - t_0)y_1 + (t_1 - t)y_2, (t - t_0)x_1 + (t_1 - t)x_2). \tag{2}$$

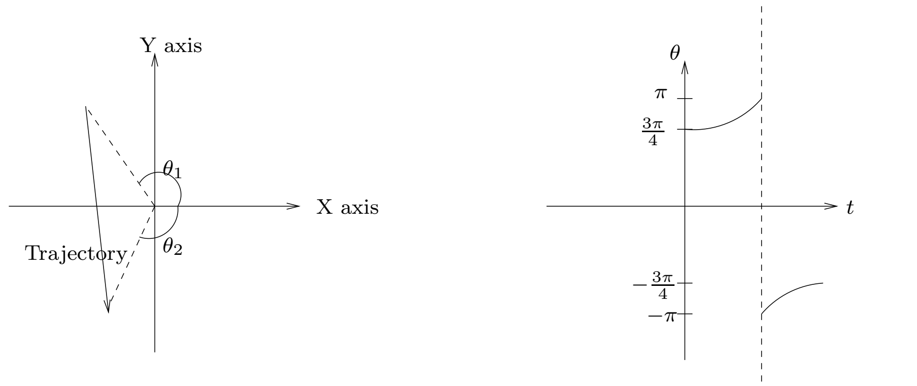
Here $\theta = f_{si}(t)$ for $t_0 \le t \le t_1$ defines a t-monotone line (usually a curve) segment on the θ-t plane. Special care has to be taken for degenerate cases where the segment is not continuous. First, if the object passes through the sensor in the ground plane, $f_{si}(t)$ is changed by $\pm\pi$ at the instant the object is at the origin. Second, when the object passes through the negative portion of the x-axis in the x-y plane, θ wraps around between $-\pi$ and π, as illustrated in Figure 2. Both degenerate cases can be handled by splitting the curve segment into segments[3].

Using the set of continuous curve segments, we can find intersection points between all $f_{si}(t)$. This is performed in the θ-t plane as shown in Figure 3. Each intersection point in the plane represents the situation when objects involved have the same θ w.r.t. the sensor. For example, in Figure 3, the sensor has unobstructed view of object 1 between t_0 and t_2. Object 2 is unobstructed in the whole period analyzed because it is the nearest object to the sensor at every intersection its segment makes with other segments. We can use the visibility information in these intersection points to schedule sensors for future time instants.

[1] The model can easily be changed to place the sensors above and looking down at the ground plane which would then shorten the "occlusion shadow" cast by each object. This would also involve assigning a height to each object.

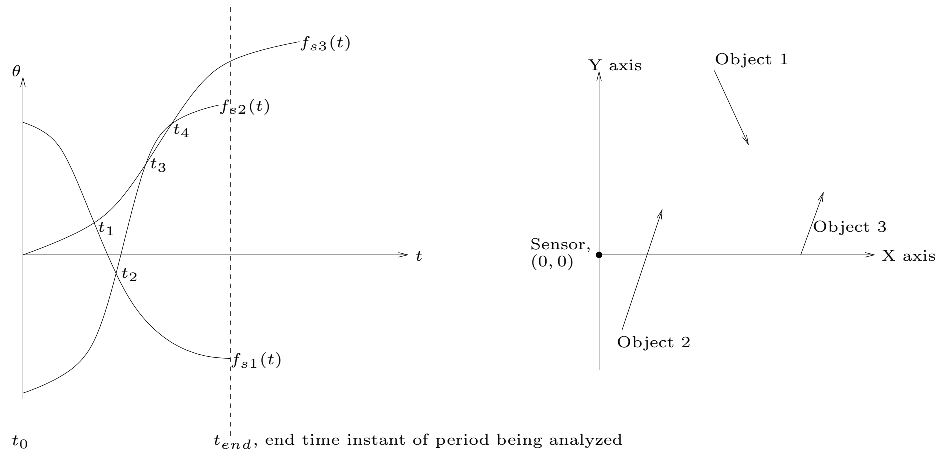
[2] If the trajectory is not a straight line, we can model it as a piecewise-linear function.

[3] Continuous curve segments are not absolutely necessary, but it eases the implementation of performing line segment intersections described in Section 3.1.



Use $\theta_1 = \frac{3\pi}{4}, \theta_2 = -\frac{3\pi}{4}$. Corresponding θ-t curve is split.

Fig. 2. Handling wrap around segments



Objects	Unobstructed Period
1	$[t_0, t_2), (t_2, t_{end}]$
2	$[t_0, t_{end}]$
3	$[t_0, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_{end}]$

Fig. 3. Intersection on the θ-t plane indicates objects in the same line of sight at the same time

3.1 Finding Intersection Points

In this section, we will show how to efficiently use the visibility model described in the previous section to find the intersection points. It is obvious from Equation 2 that visibility intervals occur between the intersections of $f_{si}(t)$ corresponding to different objects. Solving for the intersections between two object trajectories $f_{si}(t)$ and $f_{sj}(t)$, we obtain

$$at^2 + bt + c = 0 \tag{3}$$

where

$$\begin{aligned}
a &= (x_{j1} - x_{j2})(y_{i1} - y_{i2}) - (x_{i1} - x_{i2})(y_{j1} - y_{j2}) \\
b &= (t_{i0} + t_{j0})(x_{i1}y_{j1} - x_{j1}y_{i1}) + (t_{i1} + t_{j0}) \\
&\quad (x_{j1}y_{i2} - x_{i2}y_{j1}) + (t_{j1} + t_{i0})(x_{j2}y_{i1} - x_{i1}y_{j2}) + \\
&\quad (t_{i1} + t_{j1})(x_{i2}y_{j2} - x_{j2}y_{i2}) \\
c &= (t_{i0} + t_{j0})(x_{j1}y_{i1} - x_{i1}y_{j1}) + (t_{i1} + t_{j0}) \\
&\quad (x_{i2}y_{j1} - x_{j1}y_{i2}) + (t_{i0} + t_{j1})(x_{i1}y_{j2} - x_{j2}y_{i1}) + \\
&\quad (t_{i1} + t_{j1})(x_{j2}y_{i2} - x_{i2}y_{j2})
\end{aligned}$$

and (x_{i1}, y_{i1}) and (x_{i2}, y_{i2}) are the starting and ending positions of object i at time instants t_{i0} and t_{i1} respectively while (x_{j1}, y_{j1}) and (x_{ij2}, y_{j2}) are the starting and ending positions of object j at time instants t_{j0} and t_{j1} respectively. Obviously, using Equation 3 in a brute force manner will incur $O(N^2)$ running time. Note that Equation 3 also shows that two objects can lie on the same line of sight of a given sensor at most twice.

The brute force approach can be used to determine intersections when N is small. When N is large, several efficient segment intersections algorithms can be used. We will also show in Section 4 that when objects are represented as circles rather than points, the number of segments at least doubles; therefore a more efficient segment intersection algorithm is necessary. These include the plane sweep algorithm due to Bentley and Ottmann [7] (pp 22-29) and an optimal algorithm first introduced by Balaban [9], both of which give similar complexity. Our implementation of the Balaban's algorithm is based on [10] because it is more efficient than other implementations we tried. Intersections of all $f_{si}(t)$ on this plane are computed within a time period up to $t_0 + \max(\Delta t_i)$ where Δt_i is the predicted elapsed time before object i exits the scene.

Balaban's Algorithm Here we briefly outline Balaban's algorithm; readers should refer to [9] for details. Let (b, e) denote the vertical strip $b \le t \le e$ and let l and r be the x coordinates of the endpoints of a segment s $(l < r)$. The segment s is said to be spanning the strip (b, e) if $l \le b < e \le r$. Given segments s_1 and s_2, $s_1 <_a s_2$ if s_1 and s_2 intersect the vertical line $t = a$ and the intersection point with s_1 lies under the intersection point with s_2. A staircase, D, is the pair $(Q, (b, e))$, where segment set Q possesses the following properties:

– $\forall s \in Q$, s spans the strip (b, e).
– $Int_{b,e}(Q) = \emptyset$, where $Int_{b,e}(Q)$ denotes the segments pairs intersecting within the strip (b, e).
– Q is ordered by $<_b$.

Intersections of segments in Q with (b, e) are called stairs of D. The staircase D is called *complete relative* to a set S if each segment of S either does not span the strip (b, e) or intersects one of the stairs of D. The basic idea of Balaban's algorithm is to split a set of segments L into the set Q and L' so that the staircase $D = (Q, (b, e))$ is complete relative to L'. The first phase is then to find all the intersections D with L' and the second phase to find all the intersections in L'. Algorithm 1 shows how L can be split efficiently while Algorithm 2 shows how intersections between D and L' can be determined effectively. For the algorithms to work, L has to be a set of segments spanning the strip (b, e) and ordered by $<_b$. R represents the reordering of L by $<_e$. For a set of segments, Balaban's algorithm sorts the endpoints based on the x coordinates, generating a sorted list of $2N$ endpoints with the smallest and largest values used as an initial strip. This strip is then recursively split into halves during which intersections in the resulting strips are also checked for. This is done until each strip has resulting width of 1, at which time Algorithm 2 is applied to each of these strips.

Algorithm 1 $\mathrm{Split}_{b,e}(L,\, Q,\, L')$

{Let $L = (s_1, ..., s_k)$, $s_i <_b s_{i+1}$}.
$L' = \emptyset$.
$Q = \emptyset$.
for $j = 1, ..., k$ **do**
  if the segment s_j doesn't intersect the last segment of Q within (b, e) and spans this strip **then**
    add s_j to the end of Q
  else
    add s_j to the end of L'.
  end if
end for

The complexity of Balaban's algorithm is $O(N \log^2 N + I)$, where I is the number of intersection points. The algorithm is output-sensitive, since its running time depends on the number of intersections I output. Therefore the algorithm is more useful when the number of intersections is small. Note that Balaban also presented an asymptotically faster algorithm which runs in $O(N \log N + I)$. We use the former algorithm due to its relative simplicity. Readers should refer to [9] for details.

4 Modelling Objects as Circles

When the physical extent of the objects become significant, a simple point representation is insufficient. Additionally, positional uncertainty requires a different

Algorithm 2 SearchInStrip$_{b,e}$(L,R)

Split(L,Q,L').
if $L' == \emptyset$ **then**
  $R = Q$.
  Exit.
end if
Find $Int_{b,e}(Q, L')$.
SearchInStrip$_{b,e}$(L',R').
$R = \text{Merge}_e(Q, R')$.
{Merge$_x(S_1, S_2)$ is a procedure of merging sets S_1 and S_2 ordered by $<_x$.}

model of object geometry. Here we show how the method of Section 3 can be generalized to represent objects as circles on the ground plane (cylinders in 3D). This provides a fast, admittedly limited, approach to modelling object size and positional uncertainty. Figure 4 shows a circle representation with radius, r, of an object with respect to the sensor, s. Each circle has two tangents that define its extent. The visibility model can be modified to use its two tangents for finding intersection points. Let θ be the angular displacement of the circle center w.r.t. s. The angular displacement of the upper and lower tangents can then be expressed as $\theta + \alpha$ and $\theta - \alpha$ respectively, where $\alpha = \arcsin \frac{r}{d}$.

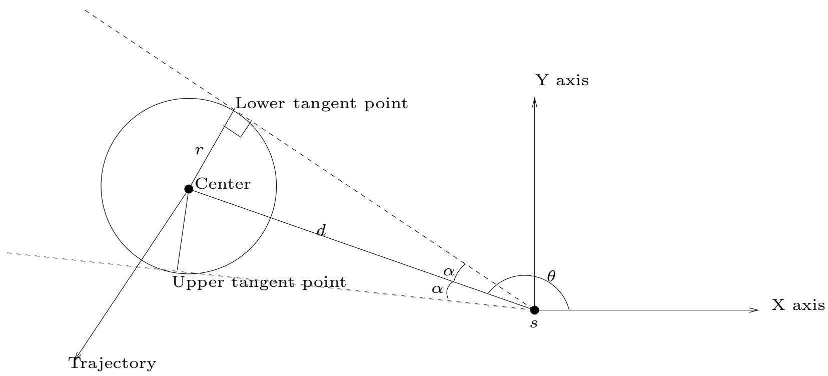


Fig. 4. Representing objects as circles ($\alpha = \arcsin \frac{r}{d}$)

Section 3.1 shows that given two point objects moving linearly, solving a quadratic equation in t allows us to exactly find the time instants when the two objects have the same θ. For the circle representation, the trajectory of the object, $f_{si}(t)$, can be expressed as the trajectories of its tangent points

$$f_{si}(t) = \theta(t) \pm \arcsin \frac{r(t)}{d(t)} \tag{4}$$

where $d(t)$ and $\theta(t)$ are the distance and the angular displacement respectively, of the center of object O_i at time instant t from sensor s; $\theta(t)$ is the same as in Equation 1 while $d(t) = \sqrt{(x_0 + v_x t)^2 + (y_0 + v_y t)^2}$ where (x_0, y_0) is the initial position of the object, v_x is the horizontal velocity and v_y the vertical velocity of the object. Although we assume that the center of the object is moving linearly, note that since the positions of these two tangents depend on the position of the object, the tangent points are actually moving non-linearly. Moreover, the radius of the circle for an object increases as the variance of the prediction increases over time. To find the potential time instants when two objects O_i and O_j occlude one another, we need to solve

$$f_{si}(t) = f_{sj}(t) \tag{5}$$

Unlike in Section 3.1, it is now clearly non-trivial to solve Equation 5 (a polynomial in t of degree 6). To resolve this problem, we split the trajectories of each tangent point into piecewise linear trajectories using Algorithm 3, and apply Equation 3 to solve for the resulting linear trajectories. Due to the splitting, the number of resulting trajectories can be large and Balaban's algorithm becomes helpful for finding intersection points efficiently.

Algorithm 3 SplitTangent(t_0,t_1)

Let p_{t_0} be the position of the tangent point in the x-y plane at t_0 using Equation 4.
Let p_{t_1} be the position of the tangent point in the x-y plane at t_1 using Equation 4.
Interpolate for the midpoint, m, of p_{t_0} and p_{t_1} in the x-y plane, i.e., $m = \frac{p_{t_1} + p_{t_0}}{2}$.
Compute the actual position, m', in the x-y plane of the tangent point at time $\frac{t_0+t_1}{2}$ using Equation 4.
if the difference between the angular displacement of m and m' is small **then**
    Assume the trajectory of the tangent point is linear from time t_0 to t_1, and return this trajectory.
else
    SplitTangent(t_0, $\frac{t_1-t_0}{2}$).
    SplitTangent($\frac{t_1-t_0}{2}$, t_1).
    Return the trajectories found in the above two SplitTangent() calls.
end if

5 Results

A Java based simulator has been implemented on top of available code from [10]. The simulations are run on an 800 MHz Pentium III Linux machine with 384 MB of RAM. Simulations are run to compare the performances between the visibility planning algorithm using the brute force algorithm and Balaban's algorithm respectively. We use a scene of size 50m×50m, with only one sensor located at the middle of the left border. We assume the field of regard of the sensor covers the whole scene. In each experiment we generate N objects where their initial positions are uniformly distributed. Their walking speeds are also uniformly distributed at 0.6 to 1 m/s, and their walking directions are randomly chosen. Since the Java Hotspot Virtual Machine's adaptive compiler optimizes Java programs as it runs them, we repeat the experiments with the same input for 30 times in the same Virtual Machine and record the average time taken for the last 10.

5.1 Experiments with Point Representation

Here we model objects as points, and we investigate which algorithm runs faster as N varies. Figure 5(a)(b)(c) shows that visibility planning using Balaban's algorithm does become faster than the brute force when N is sufficiently large. The prediction time refers to the interval between the current and some future time instants that the current trajectory prediction of an object is valid. A few observations can be made from the figure:

1. As the prediction time increases, the break-even point of N increases.
2. As the number of objects doubles, the running time of the brute force algorithm quadruples, while the running time of the Balaban's algorithm only triples.
3. The processing time is much smaller than the prediction time. This is important for showing that visibility planning into the future is viable.
4. The number of object pairs intersected (i.e., I) are much fewer than $O(N^2)$, even when the prediction time was 30 seconds (typical prediction time is 2 - 5 seconds). This shows that using an output-sensitive algorithm like Balaban's algorithm is more favorable than a brute force $O(N^2)$ algorithm. This is shown in Figure 5(d).
5. For a typical prediction time of 2 secs, we see that the break-even point is $N \approx 100$.

5.2 Experiments with Circular Representation

We model the object as circle to represent the physical extent and positional uncertainty of a given object. A fixed radius is initialized for the physical extent while the positional uncertainty is modelled by increasing that radius over the prediction time so that the confidence interval remains at 90%. The radius is increased based on the variances, σ^2_{pred1} and σ^2_{pred2}, in the prediction of the trajectory's velocity magnitude and direction returned by the Kalman filter. For realistic simulations, we monitor the trajectories of objects near an entrance of a real scene. These trajectories are then used to derive probable values for σ^2_{pred1} and σ^2_{pred2} over time. The corresponding standard deviations are determined as 0.02165 meter and 0.048 radian respectively, which are used to set the radius of the circle. The variances of the white noise for both the magnitude and direction are set as 0.5 meter and 0.01 radian respectively while the linear coefficient in the Kalman filter relating the measurements and predicted values is set as 0.8. The speed of using Balaban's algorithm and the brute force algorithm is compared in Figure 5(e)(f)(g) for prediction time of 2, 5 and 10 secs respectively. We can see that for a typical prediction time of 2 secs, the breakeven point is $N \approx 40$. This

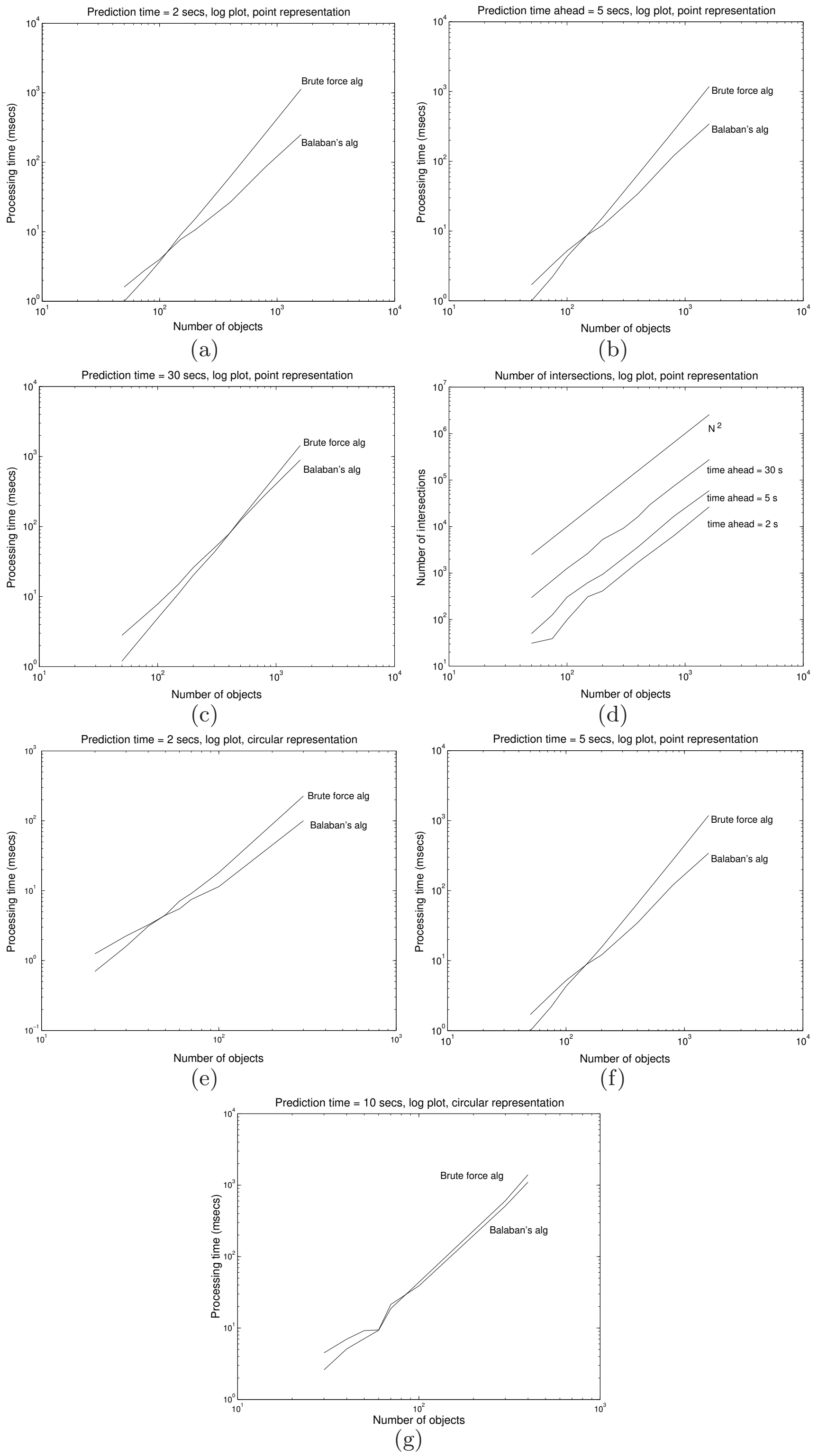


Fig. 5. Representing objects as points: (a)(b)(c) Speed comparisons for prediction time of 2, 5 and 30 secs. The breakeven point for a typical 2 secs prediction time is $N \approx 100$. (d) The number of object pairs that were checked for intersections is much lesser than the brute force approach which always checks all object pairs ($O(N^2)$). **Representing objects as circles**: (e)(f)(g) The breakeven point for a typical prediction time of 2 secs is now $N \approx 40$. This is expected since each object has 2 tangent points, giving $2N + c$ segments, where c is the number of segments that were split from the tangent trajectories

is expected since we know that circular representation will create $2N$ segments for N objects, giving a total of $2N + c$ segments including segments that were split from the tangent trajectories.

6 Discussions

We have described a novel approach to acquire unobstructed surveillance video in intervals between intersection points on the θ-t plane. We addressed the efficiency issue and showed with results that brute force approach is only viable when the number of segments created is small and Balaban's algorithm is used when the number of segments is large, particularly for circular object representation. Lastly, we have shown how prediction errors can be modelled with circular object representation. Future work on extending to a more accurate ellipsoidal object representation and using the visibility algorithm described for scheduling sensors are desired.

References

1. Olaf A. Hall-Holt, *Kinetic Visibility*, Stanford University, Computer Science Department, PhD Thesis, 2002.
2. Amy J. Briggs and Bruce Randall Donald, "Visibility-based planning of sensor control strategies," *Algorithmica*, vol. 26, no. 3-4, pp. 364–388, 2000.
3. I. Stamos and P. Allen, "Interactive sensor planning," in *Computer Vision and Pattern Recognition Conference*, Jun 1998, pp. 489–494.
4. Ismail Haritaoglu, David Harwood, and Larry S. Davis, "W4:who? when? where? what? a real time system for detecting and tracking people," in *International Conference on Face and Gesture Recognition*, 1998.
5. Kalman, Rudolph, and Emil, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
6. Sing Bing Kang, Steven M. Seitz, and Peter-Pike Sloan, "Visual tunnel analysis fpr visibility prediction and camera planning," in *Computer Vision and Pattern Recognition*, Jun 2000, p. 2195.
7. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*, Springer, 1997.
8. K.A. Tarabanis, P.K. Allen, and R.Y. Tsai, "A survey of sensor planning in computer vision," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 1, pp. 86–104, 1995.
9. Ivan J. Balaban, "An optimal algorithm for finding segments intersections," in *Proceedings of the eleventh annual symposium on Computational geometry, Vancouver, British Columbia, Canada*, 1995, pp. Pages: 211–219.
10. Radu Florian, "Computational geometry," *http://bigram.cs.jhu.edu/~rflorian/CompGeom1.html*.