Tracking Objects in Videos

Mostafa Twifiq 6672

Mohamed Zayton 6670

Introduction

Image alignment consists of moving, and possibly deforming, a template to minimize the difference between the template and an image. Since the first use of image alignment in the Lucas-Kanade optical flow algorithm, image alignment has become one of the most widely used techniques in computer vision. Besides optical flow, some of its other applications include tracking, parametric and layered motion estimation, mosaic construction, medical image registration, and face coding.

Lucas-Kanade:

The original image alignment algorithm was the Lucas-Kanade algorithm [13]. The goal of Lucas-Kanade is to align a template image $T(\mathbf{x})$ to an input image $I(\mathbf{x})$, where $\mathbf{x} = (x, y)^{\mathrm{T}}$ is a column vector containing the pixel coordinates. If the Lucas-Kanade algorithm is being used to compute *optical flow* or to *track* an image patch from time $t = 1$ to time $t = 2$, the template $T(\mathbf{x})$ is an extracted sub-region (a 5×5 window, maybe) of the image at $t = 1$ and $I(\mathbf{x})$ is the image at $t = 2$.

Let $\mathbf{W}(\mathbf{x}; \mathbf{p})$ denote the parameterized set of allowed warps, where $\mathbf{p} = (p_1, \ldots p_n)^{\mathrm{T}}$ is a vector of parameters. The warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ takes the pixel $\mathbf{x}$ in the coordinate frame of the template T and maps it to the sub-pixel location $\mathbf{W}(\mathbf{x}; \mathbf{p})$ in the coordinate frame of the image I. If we are computing optical flow, for example, the warps $\mathbf{W}(\mathbf{x}; \mathbf{p})$ might be the translations:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \begin{pmatrix} x + p_1 \\ y + p_2 \end{pmatrix} \tag{1}$$

where the vector of parameters $\mathbf{p} = (p_1, p_2)^{\mathrm{T}}$ is then the optical flow. If we are tracking a larger image patch moving in 3D we may instead consider the set of affine warps:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \begin{pmatrix} (1 + p_1) \cdot x & + & p_3 \cdot y & + & p_5 \\ p_2 \cdot x & + & (1 + p_4) \cdot y & + & p_6 \end{pmatrix} = \begin{pmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2}$$

where there are 6 parameters $\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6)^{\mathrm{T}}$ as, for example, was done in . (There are other ways to parameterize affine warps. Later in this framework we will investigate what is the best way.) In general, the number of parameters n may be arbitrarily large and $\mathbf{W}(\mathbf{x}; \mathbf{p})$ can be arbitrarily complex. One example of a complex warp is the set of piecewise affine warps used in Active Appearance Models and Active Blobs

The goal of the Lucas-Kanade Algorithm

The goal of the Lucas-Kanade algorithm is to minimize the sum of squared error between two images, the template T and the image I warped back onto the coordinate frame of the template:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \,]^2 \,. \tag{3}$$

Warping I back to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ requires interpolating the image I at the sub-pixel locations $\mathbf{W}(\mathbf{x};\mathbf{p})$. The minimization of the expression in Equation (3) is performed with respect to $\mathbf{p}$ and the sum is performed over all of the pixels $\mathbf{x}$ in the template image $T(\mathbf{x})$. Minimizing

the expression in Equation (1) is a non-linear optimization task even if $\mathbf{W}(\mathbf{x};\mathbf{p})$ is linear in $\mathbf{p}$ because the pixel values $I(\mathbf{x})$ are, in general, non-linear in $\mathbf{x}$. In fact, the pixel values $I(\mathbf{x})$ are essentially un-related to the pixel coordinates $\mathbf{x}$. To optimize the expression in Equation (3), the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known and then iteratively solves for increments to the parameters $\Delta\mathbf{p}$; i.e. the following expression is (approximately) minimized:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x}) \,]^2 \tag{4}$$

with respect to $\Delta\mathbf{p}$, and then the parameters are updated:

$$\mathbf{p} \;\leftarrow\; \mathbf{p} + \Delta\mathbf{p}. \tag{5}$$

These two steps are iterated until the estimates of the parameters $\mathbf{p}$ converge. Typically the test for convergence is whether some norm of the vector $\Delta\mathbf{p}$ is below a threshold ϵ; i.e. $\|\Delta\mathbf{p}\| \leq \epsilon$.

Derivation of the Lucas-Kanade Algorithm

The Lucas-Kanade algorithm (which is a Gauss-Newton gradient descent non-linear optimization algorithm) is then derived as follows. The non-linear expression in Equation (4) is linearized by performing a first order Taylor expansion on $I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))$ to give:

$$\sum_{\mathbf{x}}\left[I(\mathbf{W}(\mathbf{x};\mathbf{p}))+\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p}-T(\mathbf{x})\right]^2. \tag{6}$$

In this expression, $\nabla I=(\frac{\partial I}{\partial x},\frac{\partial I}{\partial y})$ is the *gradient* of image I evaluated at $\mathbf{W}(\mathbf{x};\mathbf{p})$; i.e. ∇I is computed in the coordinate frame of I and then warped back onto the coordinate frame of T using the current estimate of the warp $\mathbf{W}(\mathbf{x};\mathbf{p})$. The term $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ is the *Jacobian* of the warp. If $\mathbf{W}(\mathbf{x};\mathbf{p})=(W_x(\mathbf{x};\mathbf{p}),W_y(\mathbf{x};\mathbf{p}))^{\mathrm{T}}$ then:

$$\frac{\partial\mathbf{W}}{\partial\mathbf{p}}=\begin{pmatrix}\frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n}\\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n}\end{pmatrix}. \tag{7}$$

We follow the notational convention that the partial derivatives with respect to a column vector are laid out as a row vector. This convention has the advantage that the chain rule results in a matrix multiplication, as in the expression in Equation (6). For example, the affine warp in Equation (2) has the Jacobian:

$$\frac{\partial\mathbf{W}}{\partial\mathbf{p}}=\begin{pmatrix}x & 0 & y & 0 & 1 & 0\\ 0 & x & 0 & y & 0 & 1\end{pmatrix}. \tag{8}$$

The Lucas-Kanade Algorithm

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Compute the error image $T(\mathbf{x})-I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(3) Warp the gradient ∇I with $\mathbf{W}(\mathbf{x};\mathbf{p})$
(4) Evaluate the Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ at $(\mathbf{x};\mathbf{p})$
(5) Compute the steepest descent images $\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$
(6) Compute the Hessian matrix using Equation (11)
(7) Compute $\sum_{\mathbf{x}}[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}]^{\mathrm{T}}[T(\mathbf{x})-I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
(8) Compute $\Delta\mathbf{p}$ using Equation (10)
(9) Update the parameters $\mathbf{p}\leftarrow\mathbf{p}+\Delta\mathbf{p}$

until $\|\Delta\mathbf{p}\|\leq\epsilon$

Minimizing the expression in Equation (6) is a least squares problem and has a closed from solution which can be derived as follows. The partial derivative of the expression in Equation (6) with respect to $\Delta \mathbf{p}$ is:

$$2 \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right] \tag{9}$$

where we refer to $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ as the *steepest descent* images. (See Section 4.3 for why.) Setting this expression to equal zero and solving gives the closed form solution for the minimum of the expression in Equation (6) as:

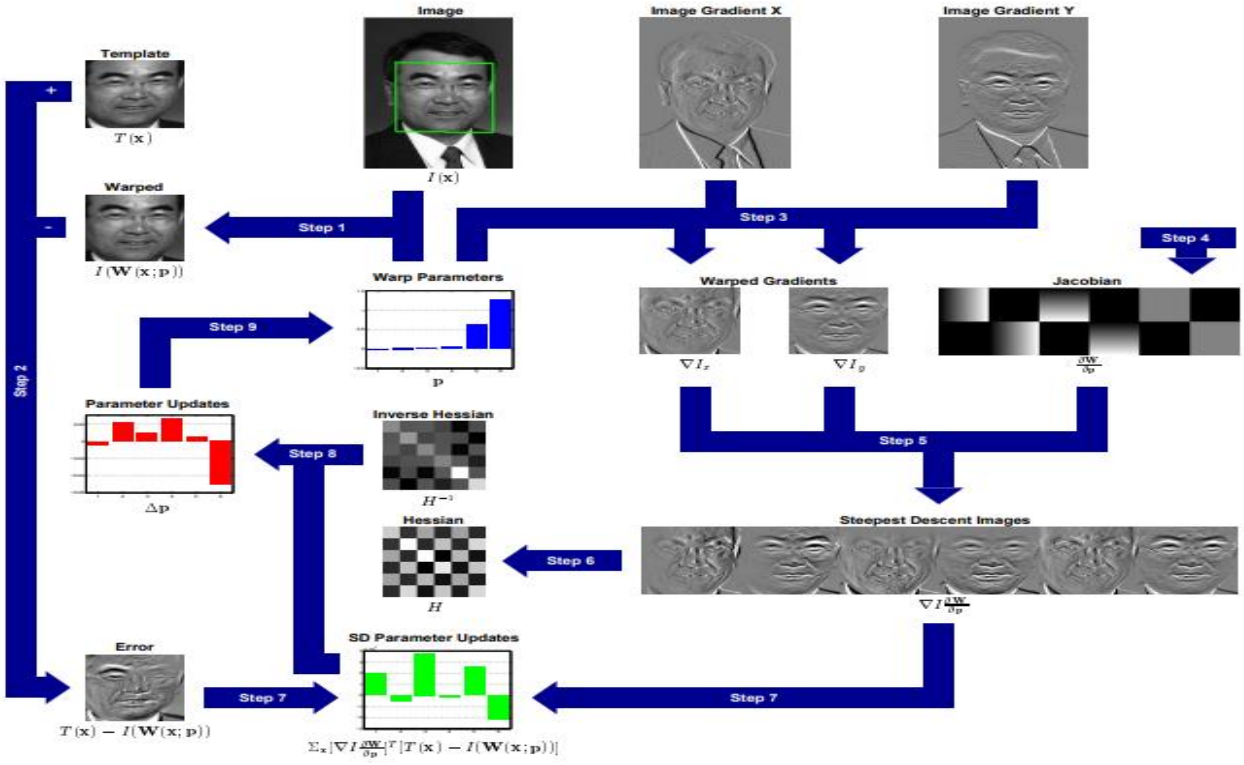
$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))] \tag{10}$$

where H is the $n \times n$ (Gauss-Newton approximation to the) *Hessian* matrix:

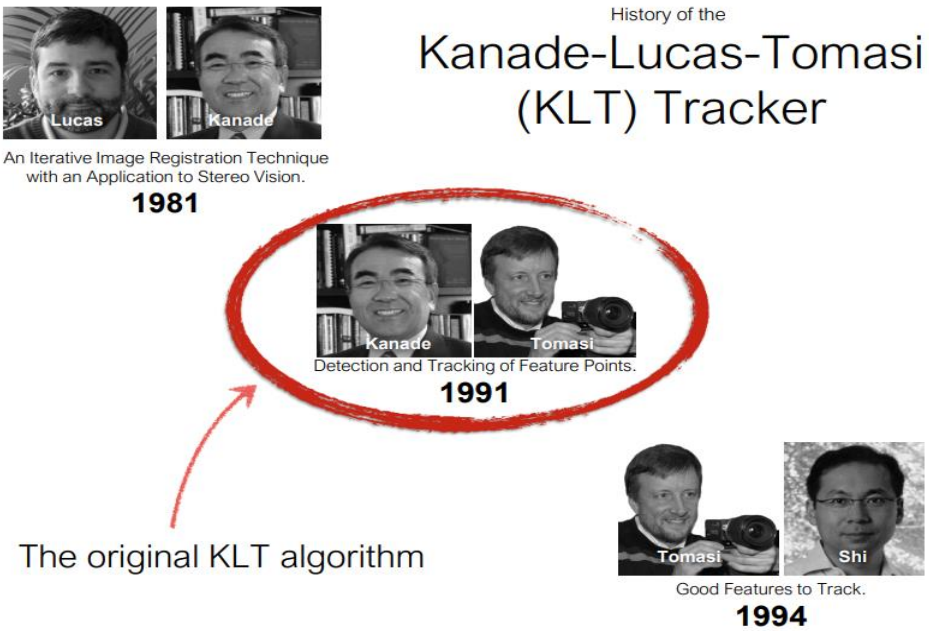
$$H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] . \tag{11}$$

For reasons that will become clear later we refer to $\sum_{\mathbf{x}} [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$ as the *steepest descent parameter updates*. Equation (10) then expresses the fact that the parameter updates $\Delta \mathbf{p}$ are the steepest descent parameter updates multiplied by the inverse of the Hessian matrix. The Lucas-Kanade algorithm [13] then consists of iteratively applying Equations (10) and (5).

A schematic overview of the Lucas-Kanade algorithm



History of KLT

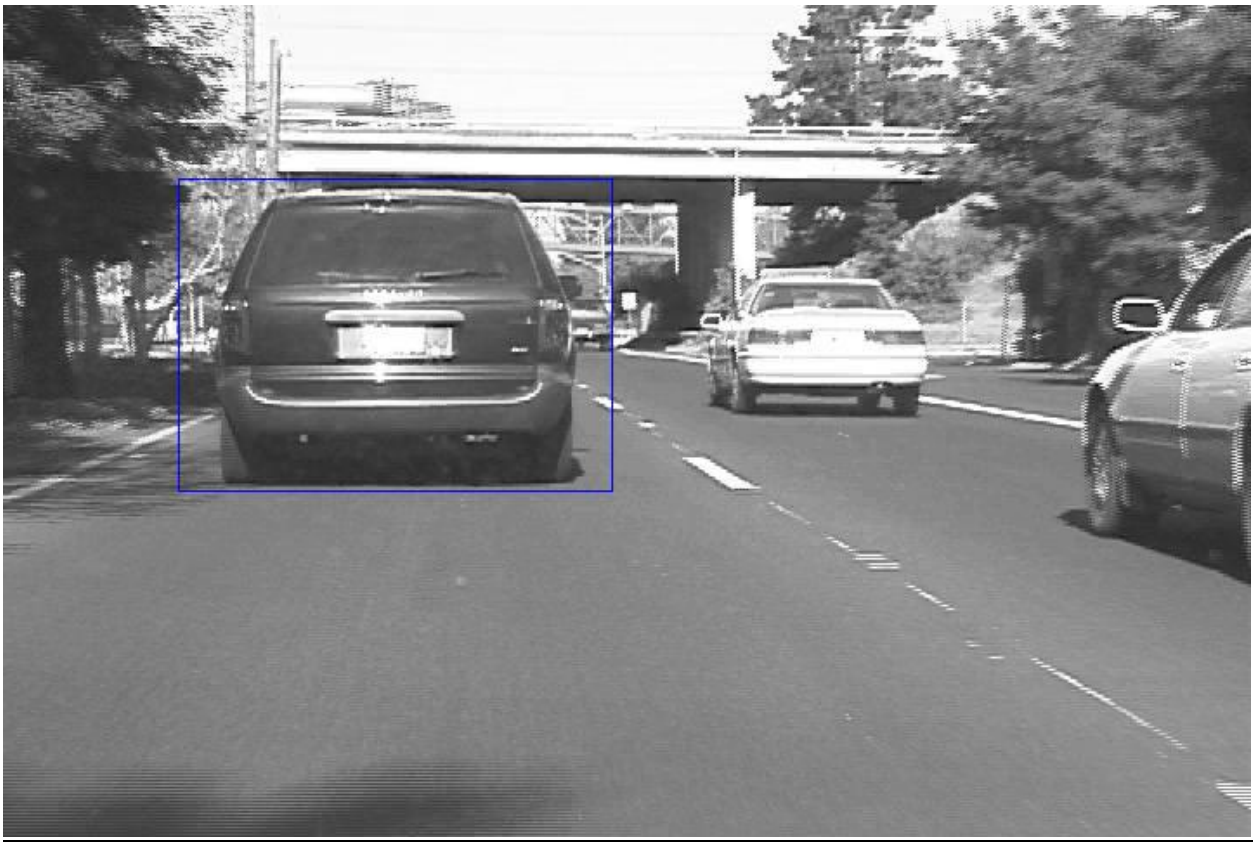


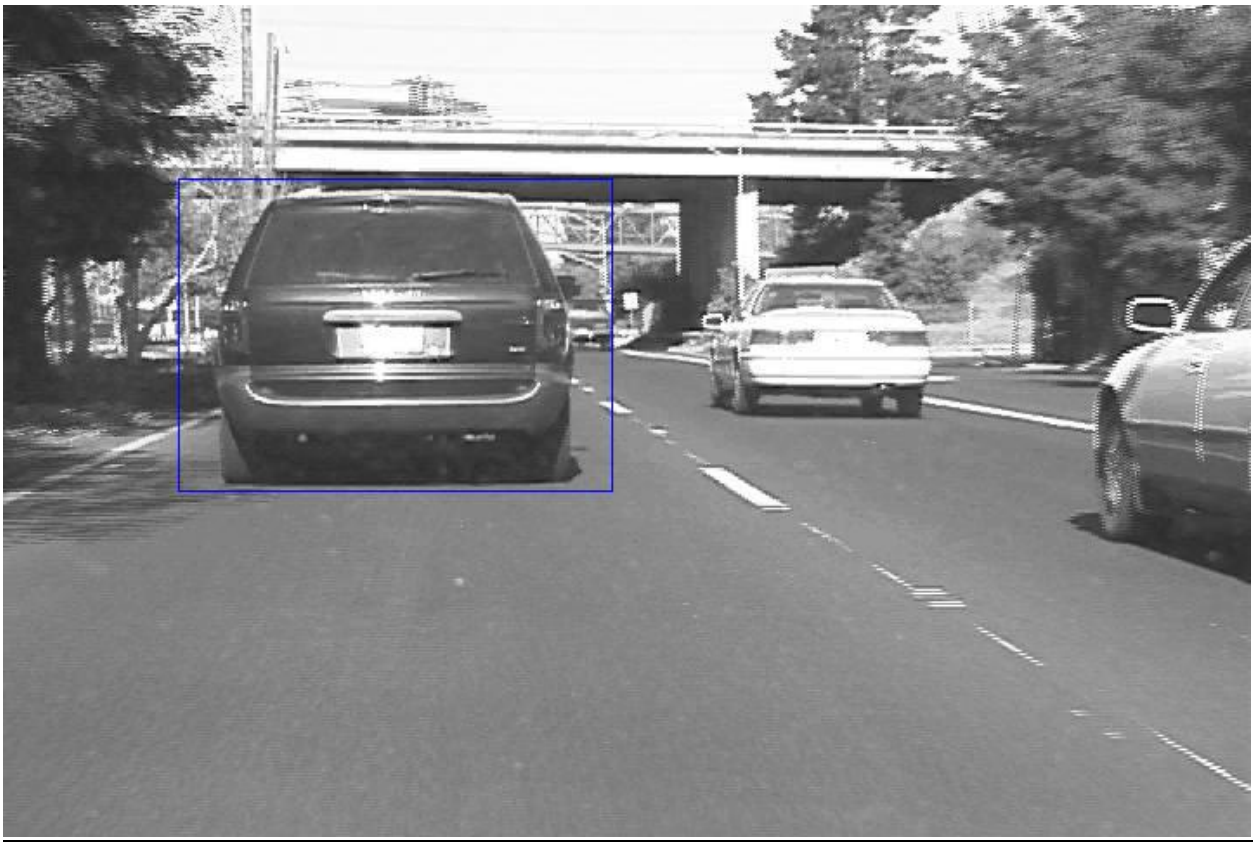
KLT algorithm

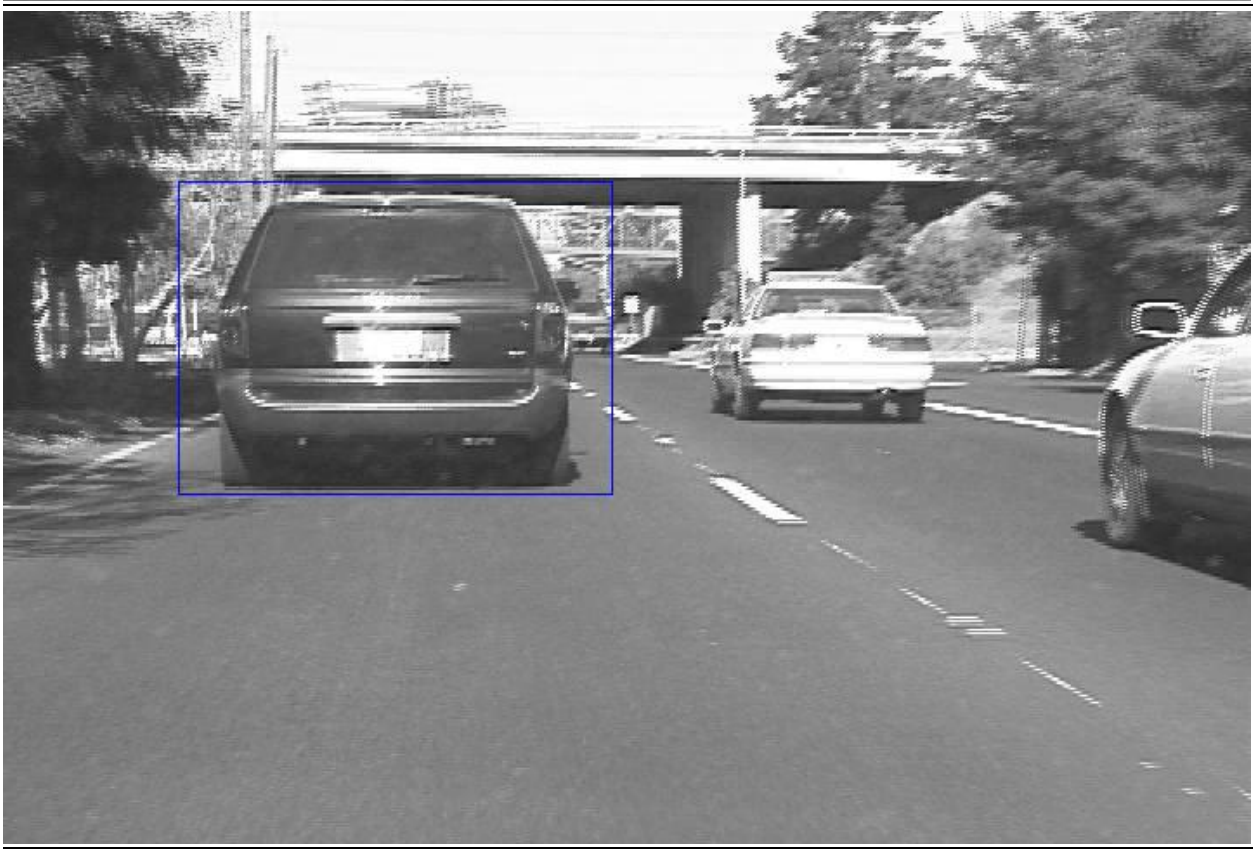
1. Find corners satisfying
2. For each corner compute displacement to next frame using the Lucas-Kanade method
3. Store displacement of each corner, update corner position
4. (optional) Add more corner points every M frames using 1
5. Repeat 2 to 3 (4)
6. Returns long trajectories for each corner point

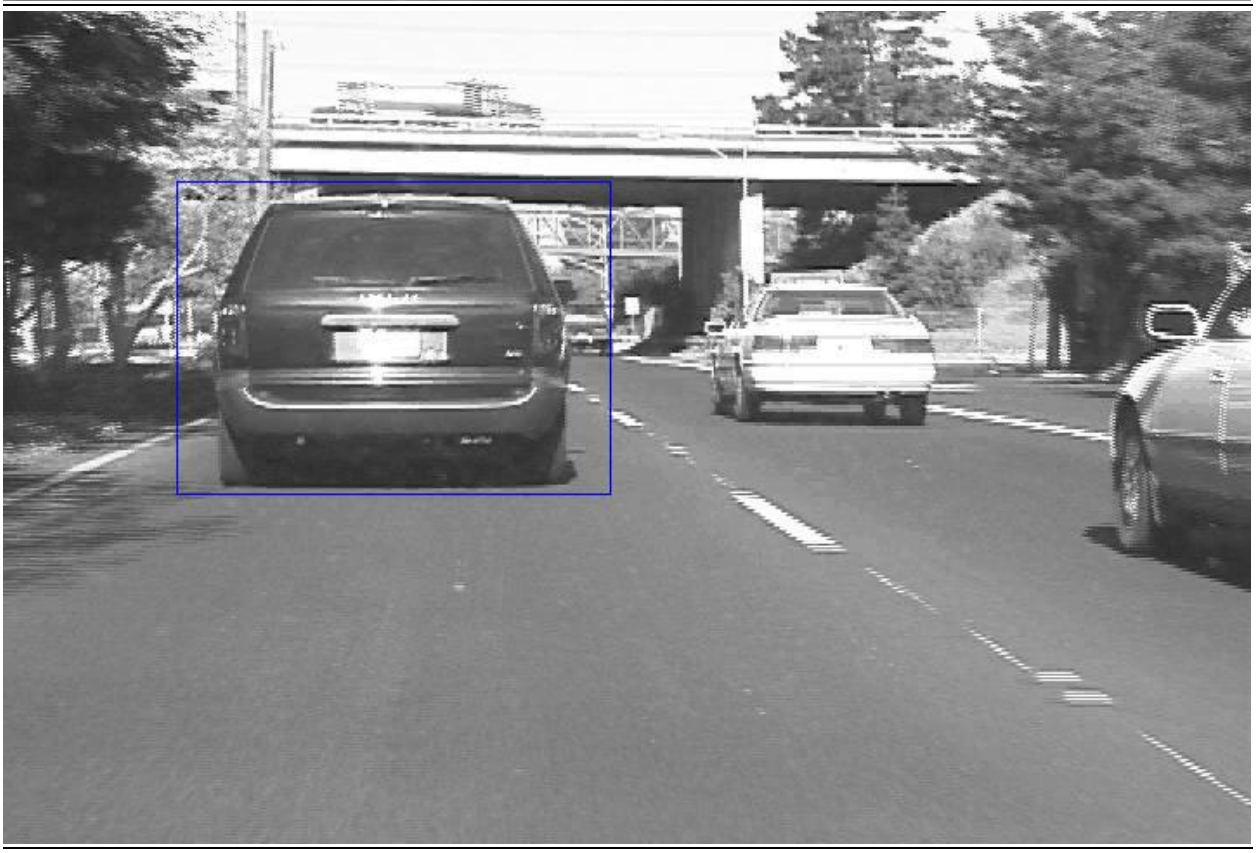
Sample from the result

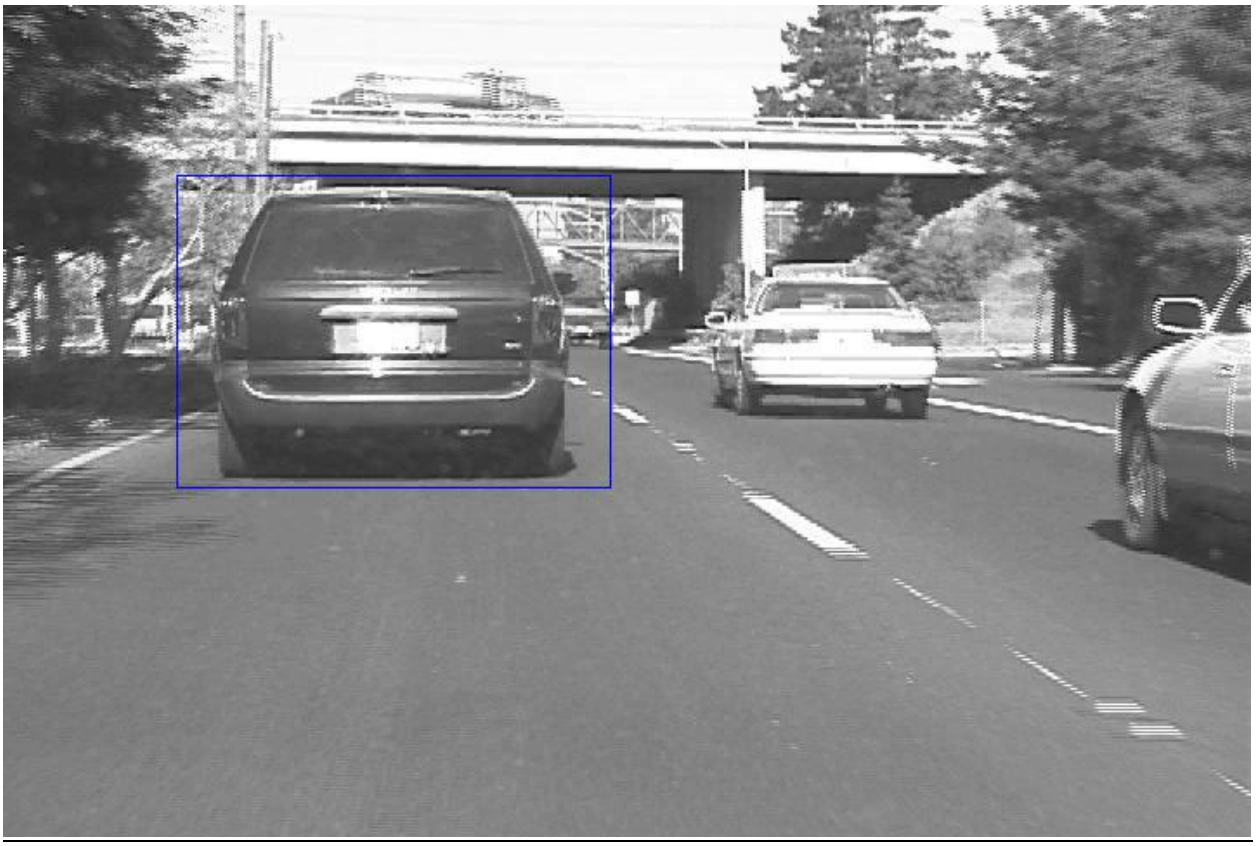
Car1

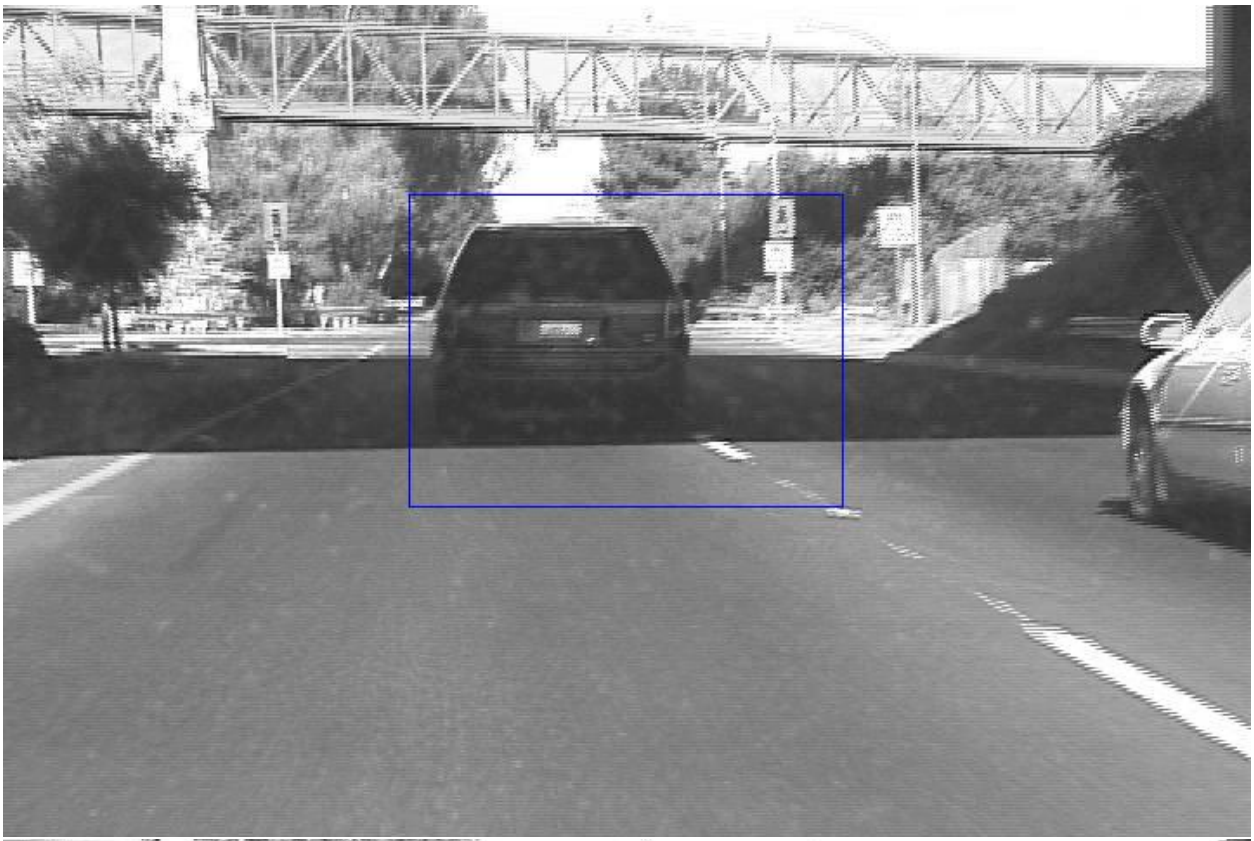


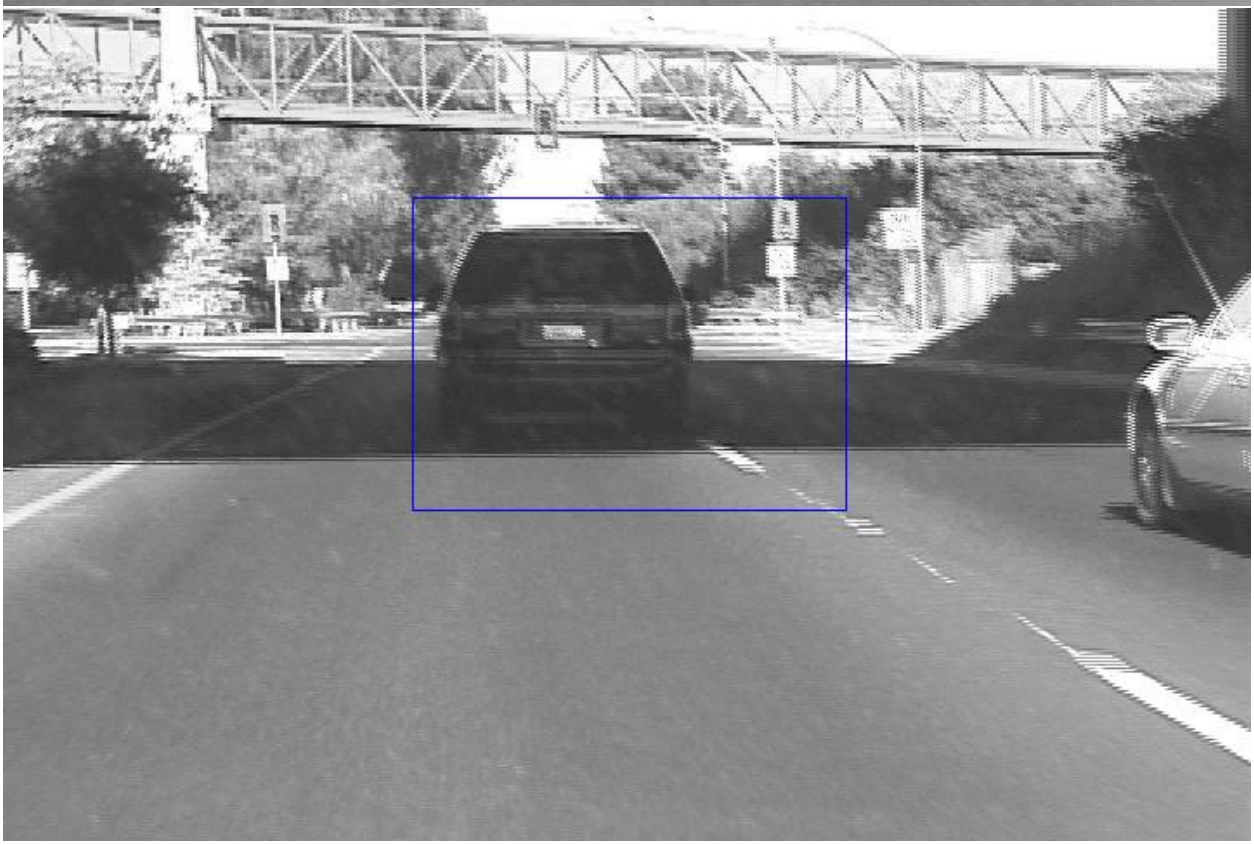


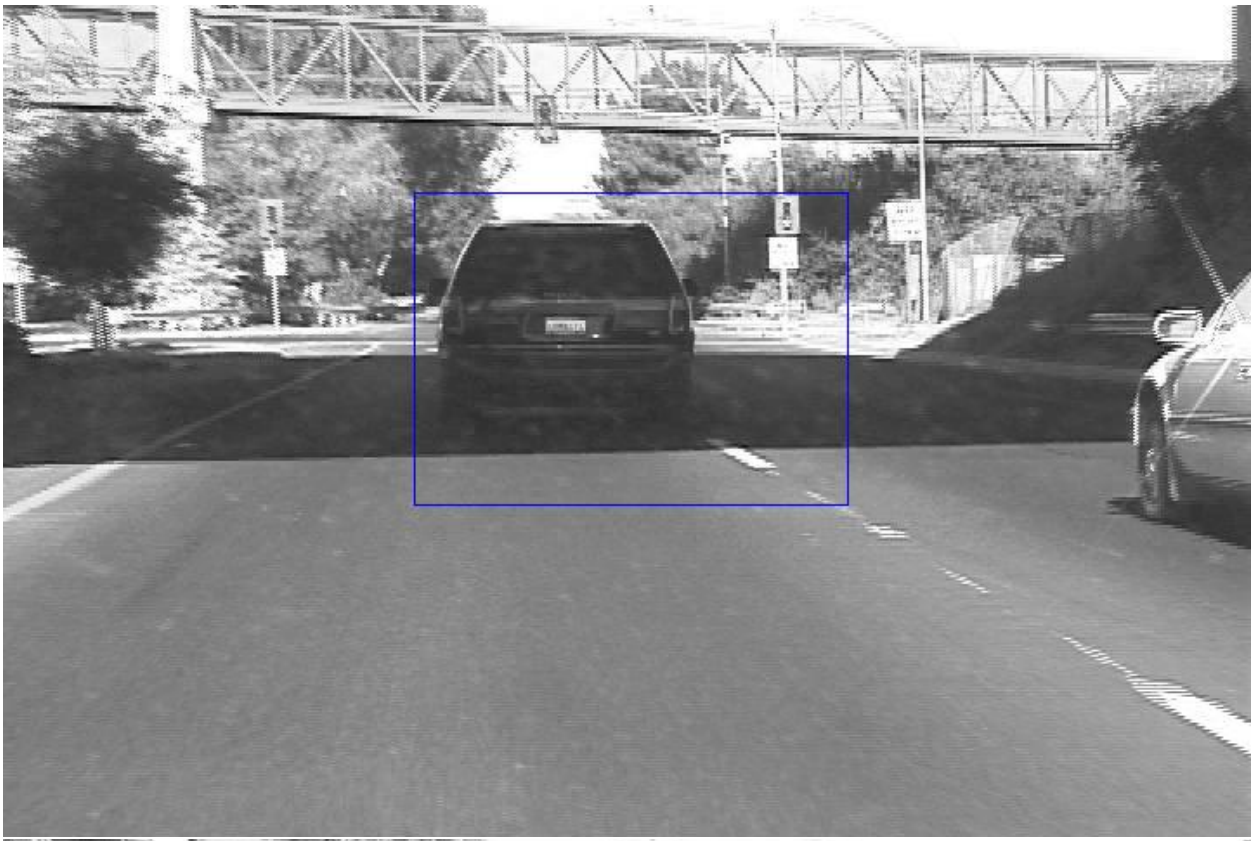


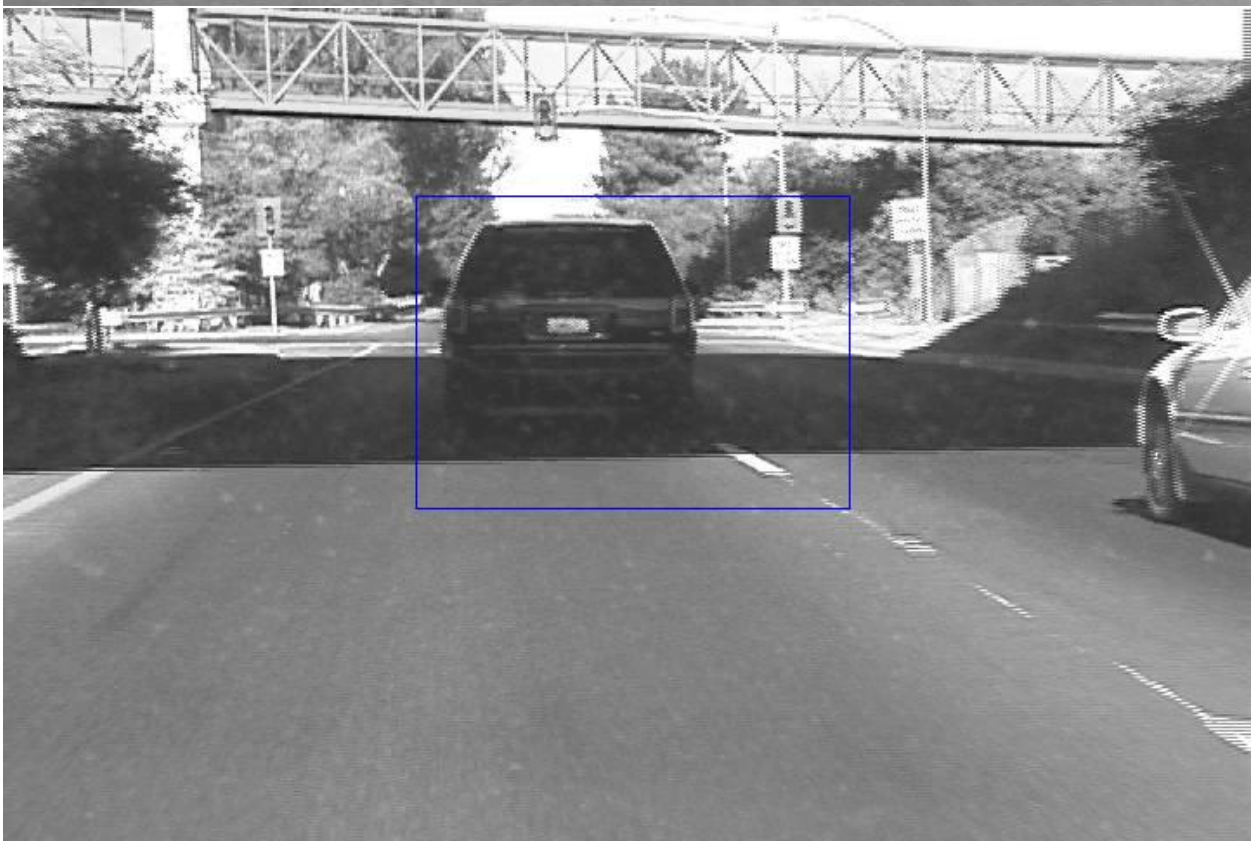


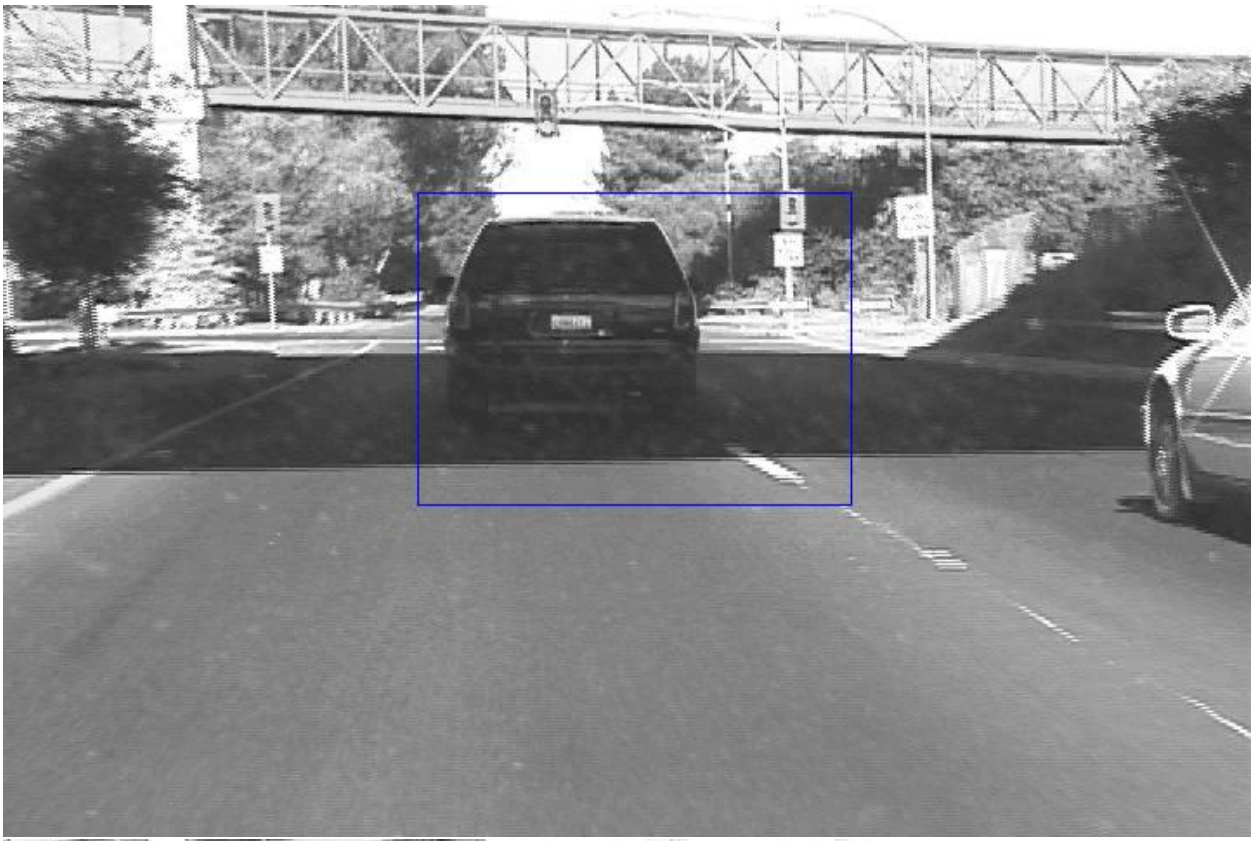


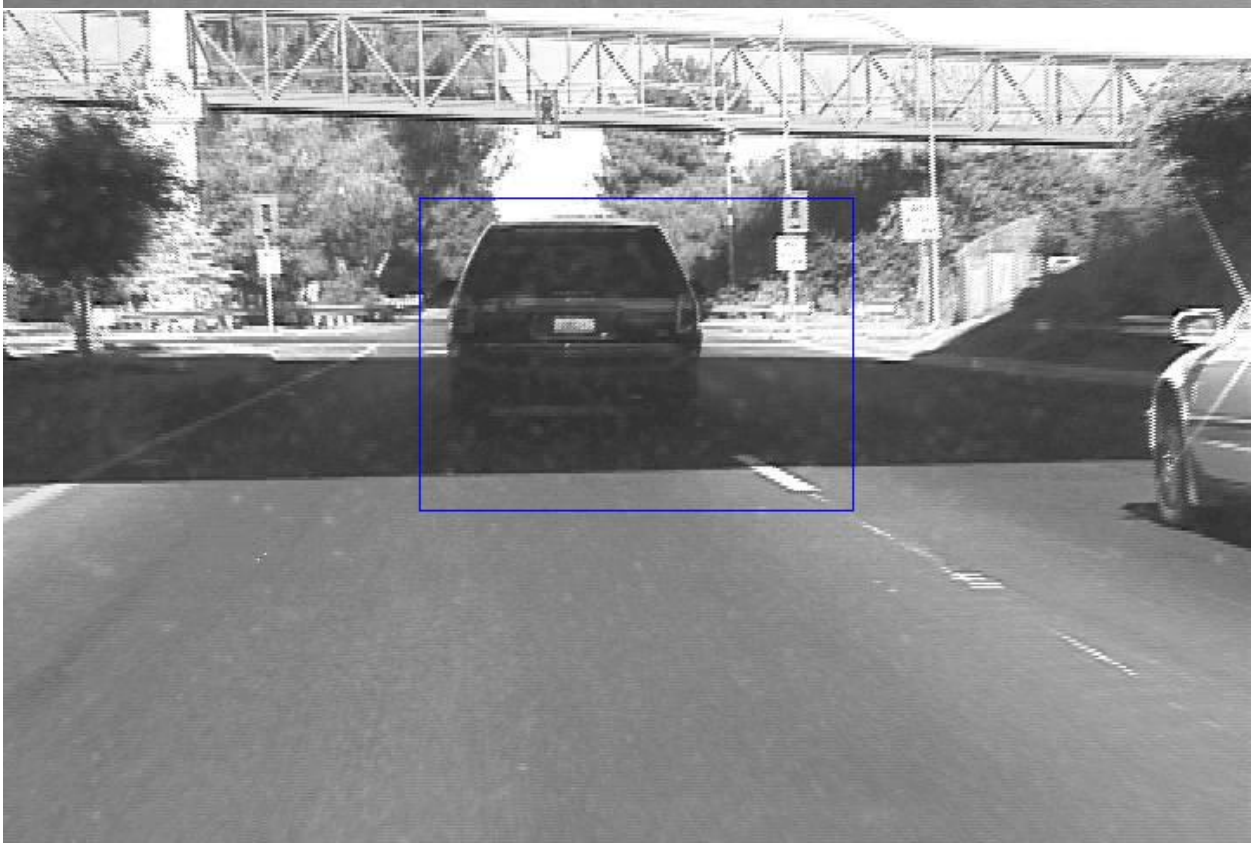


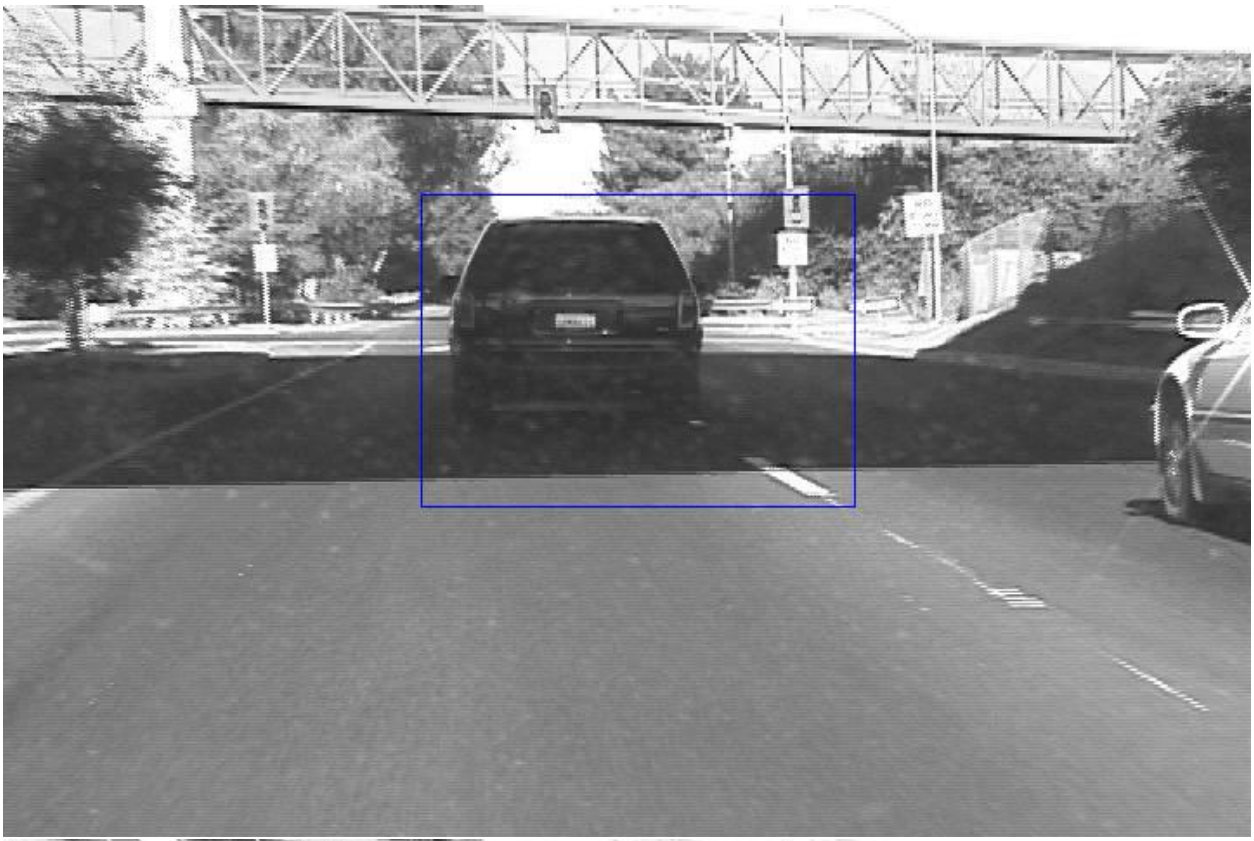


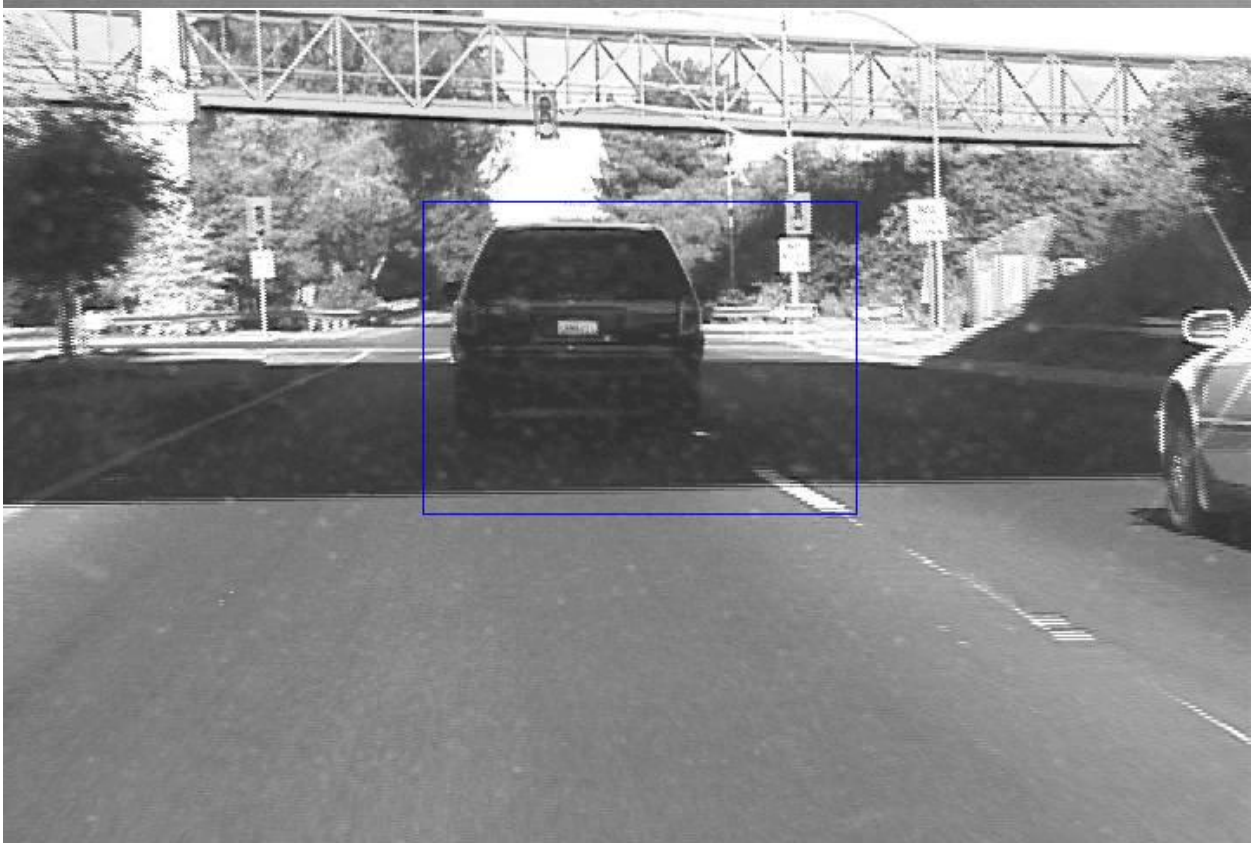


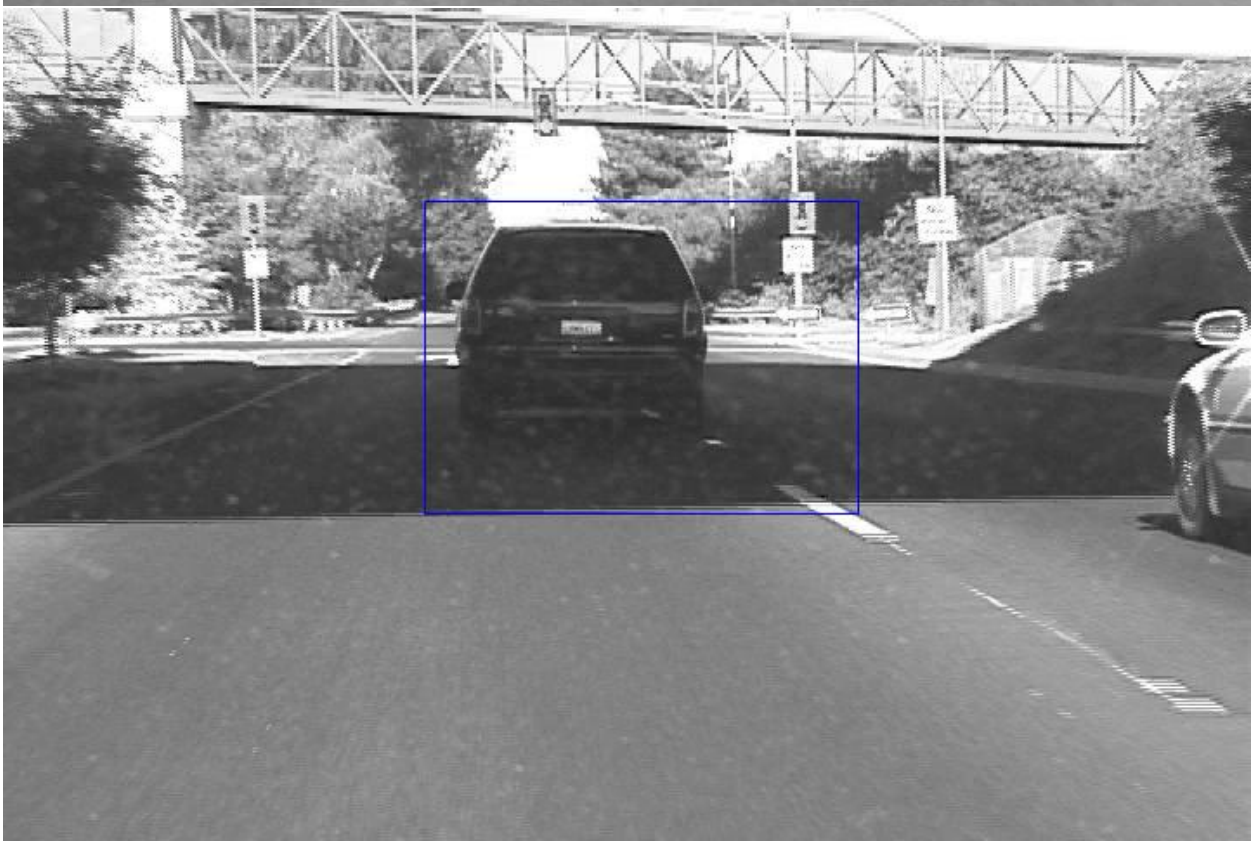


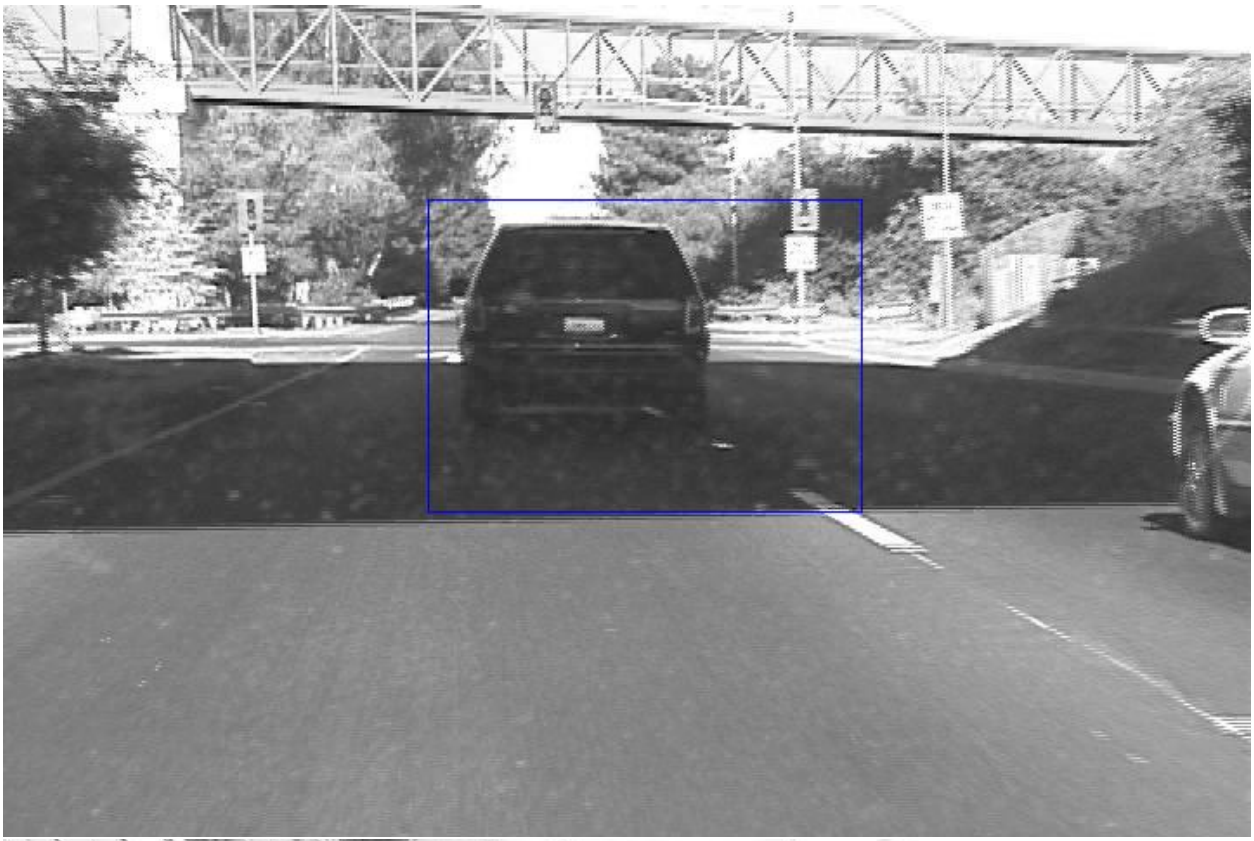


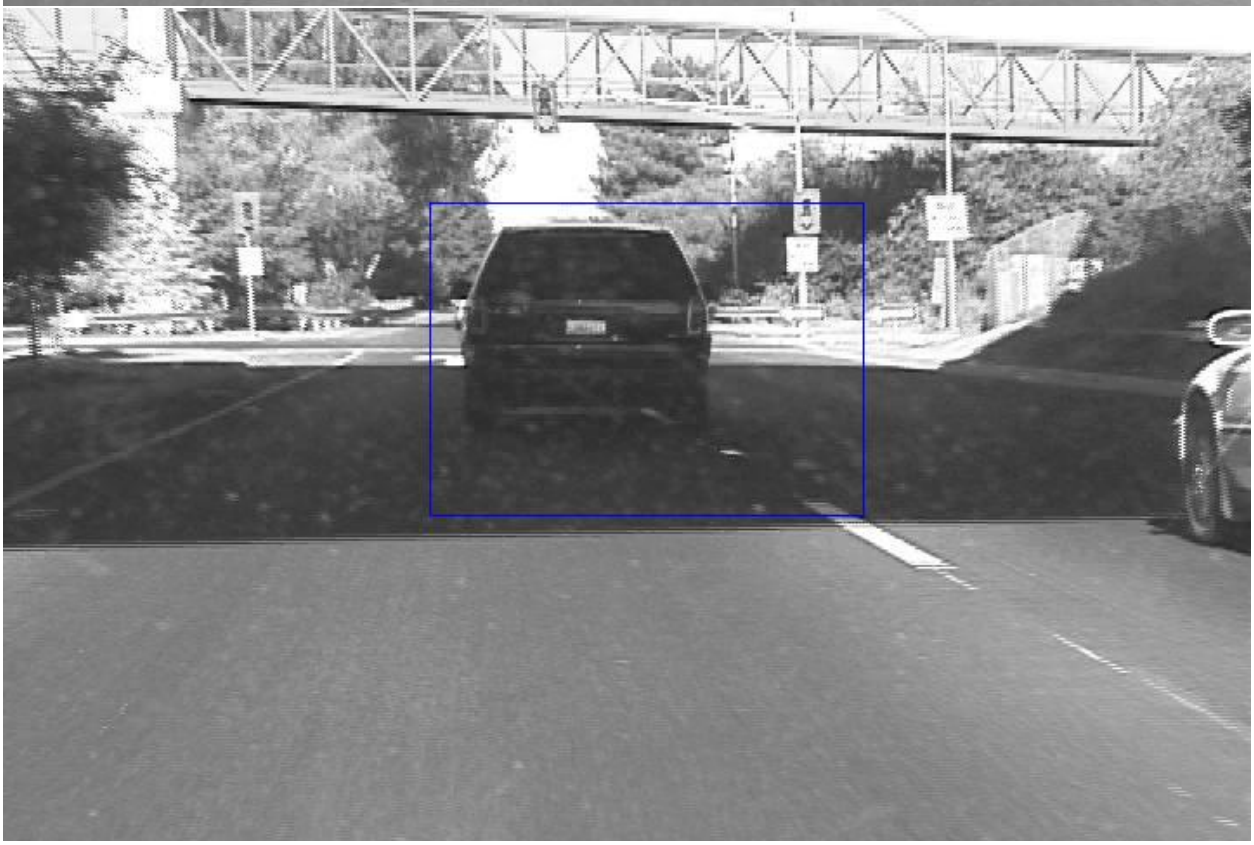




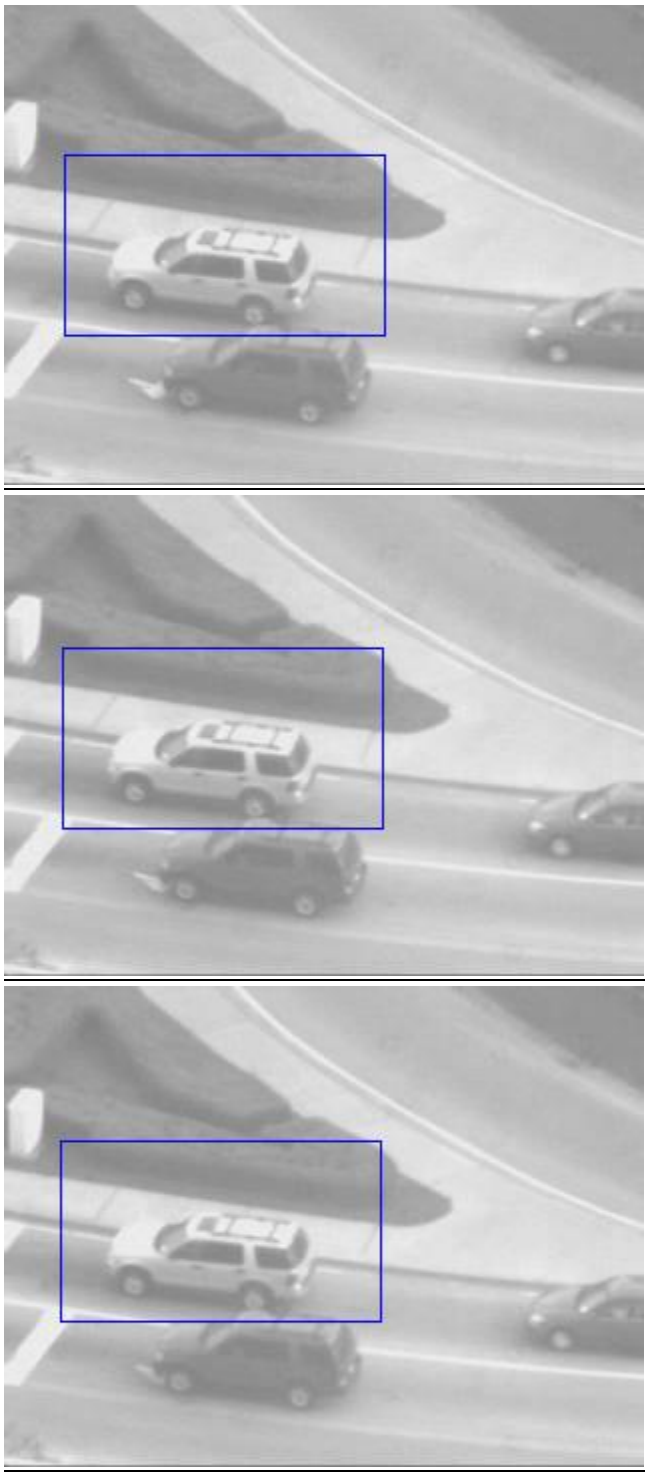


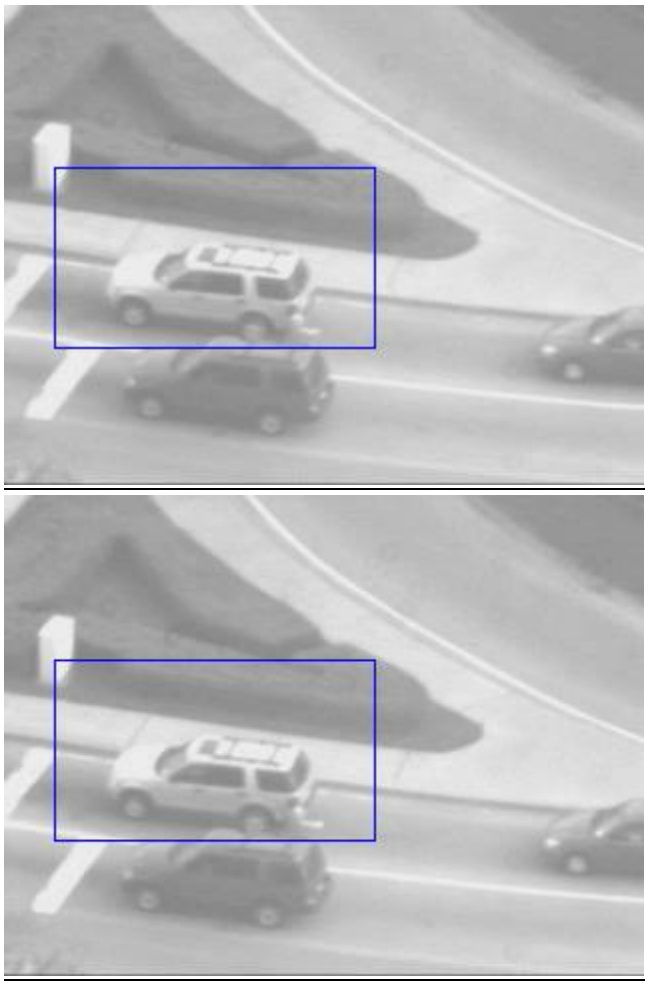


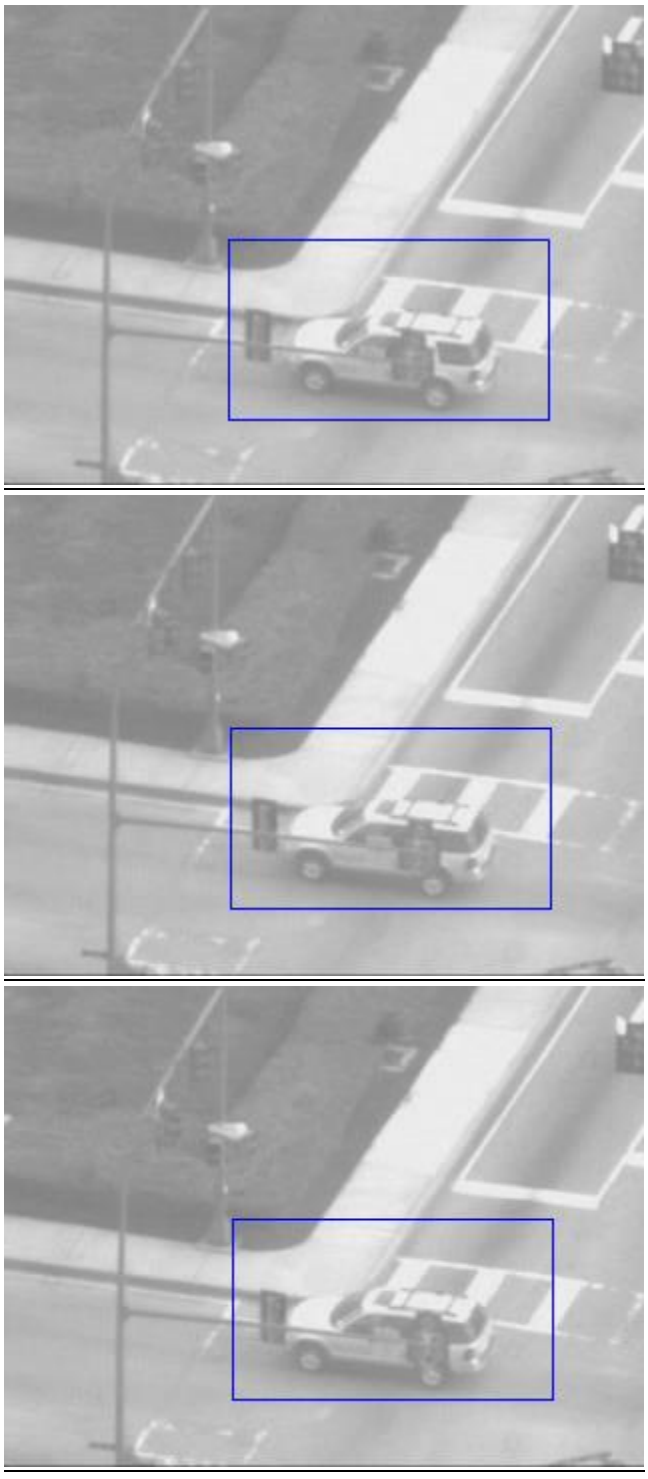


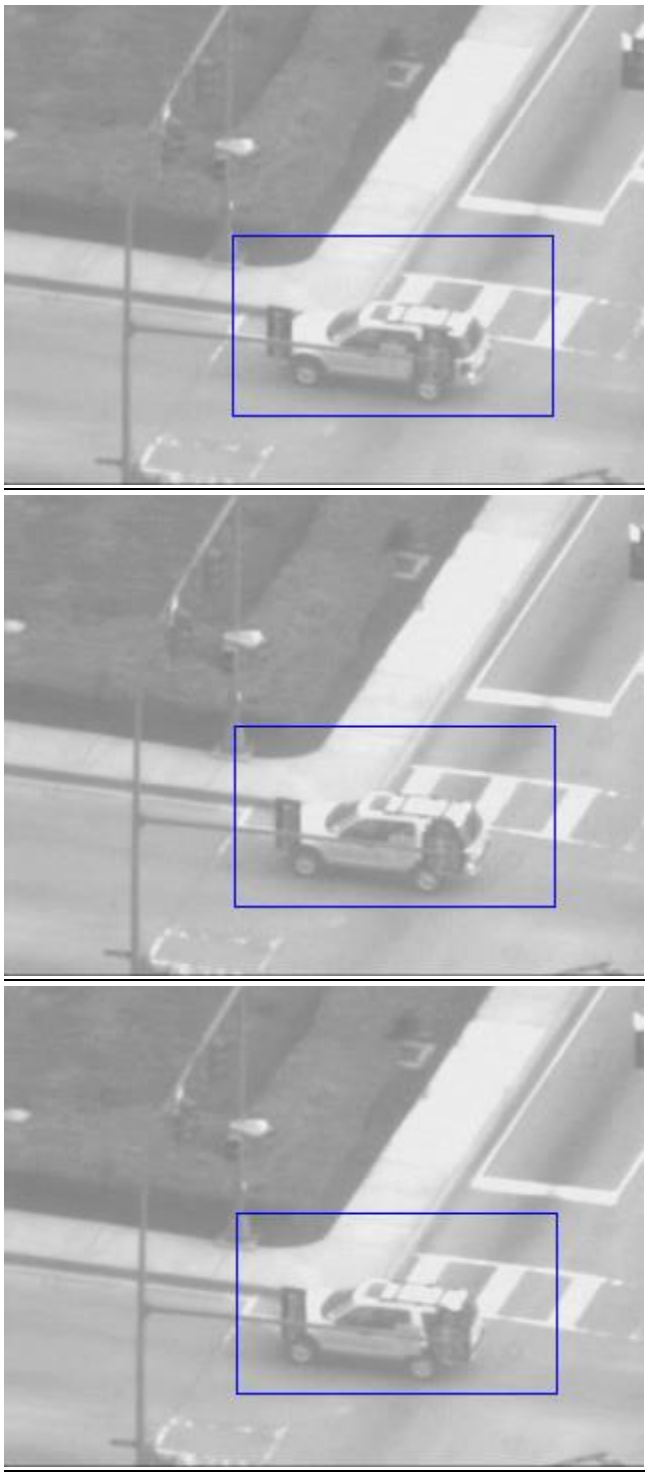


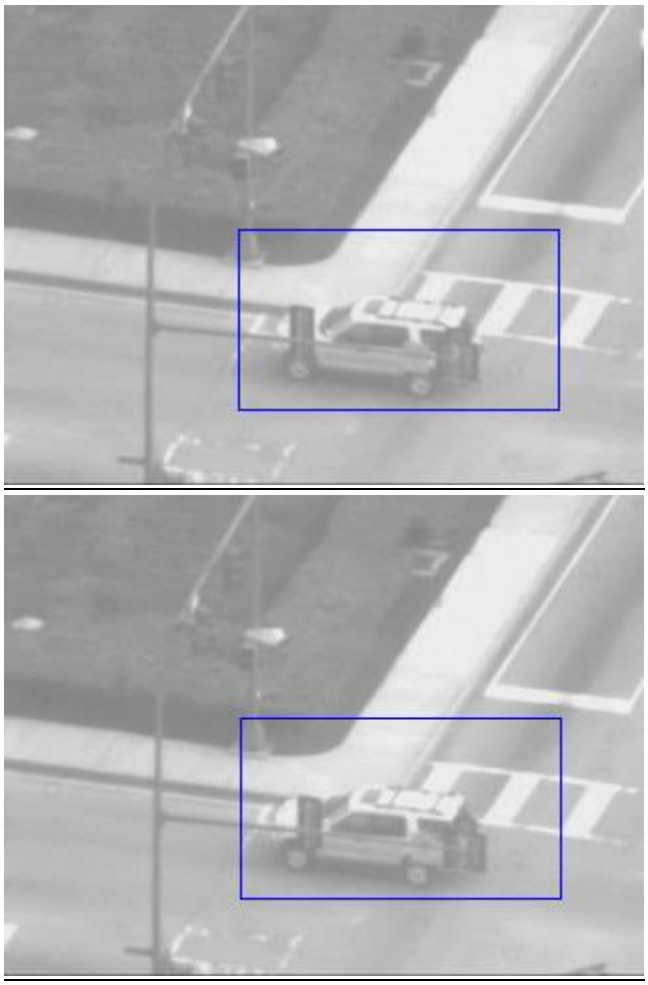
Car2

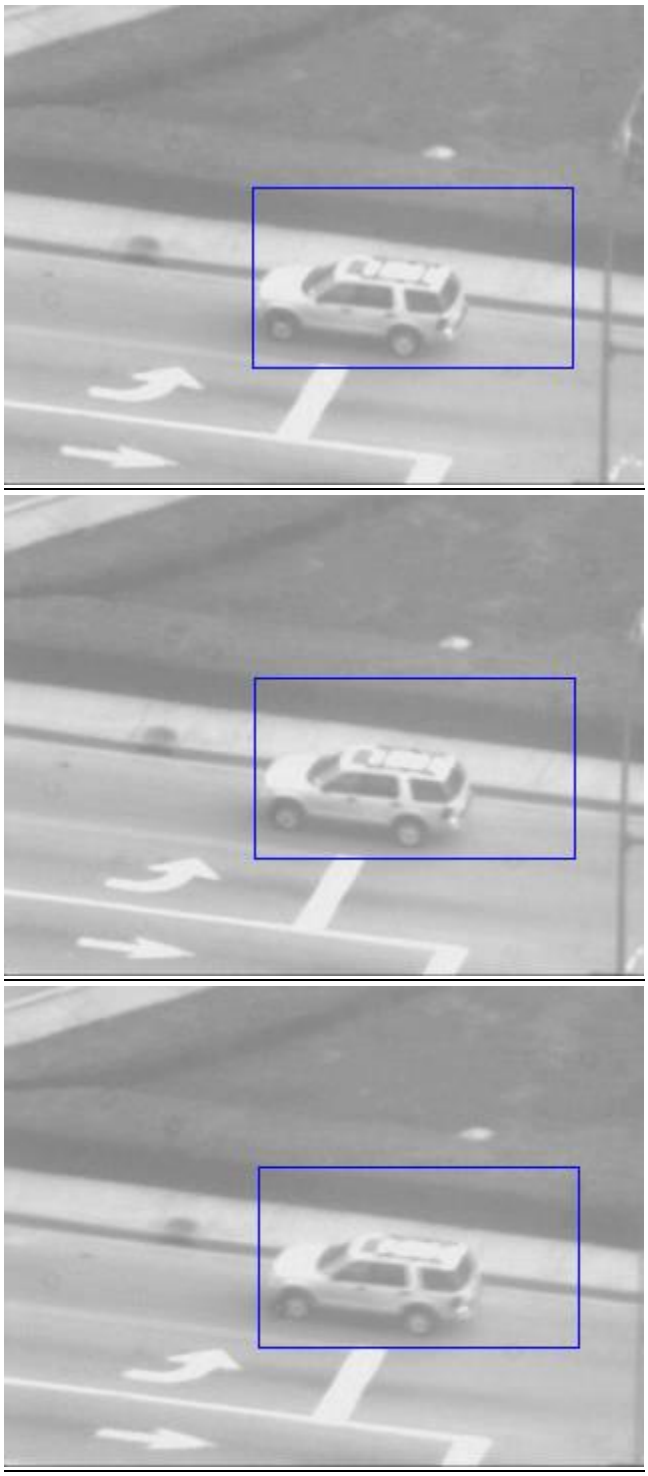


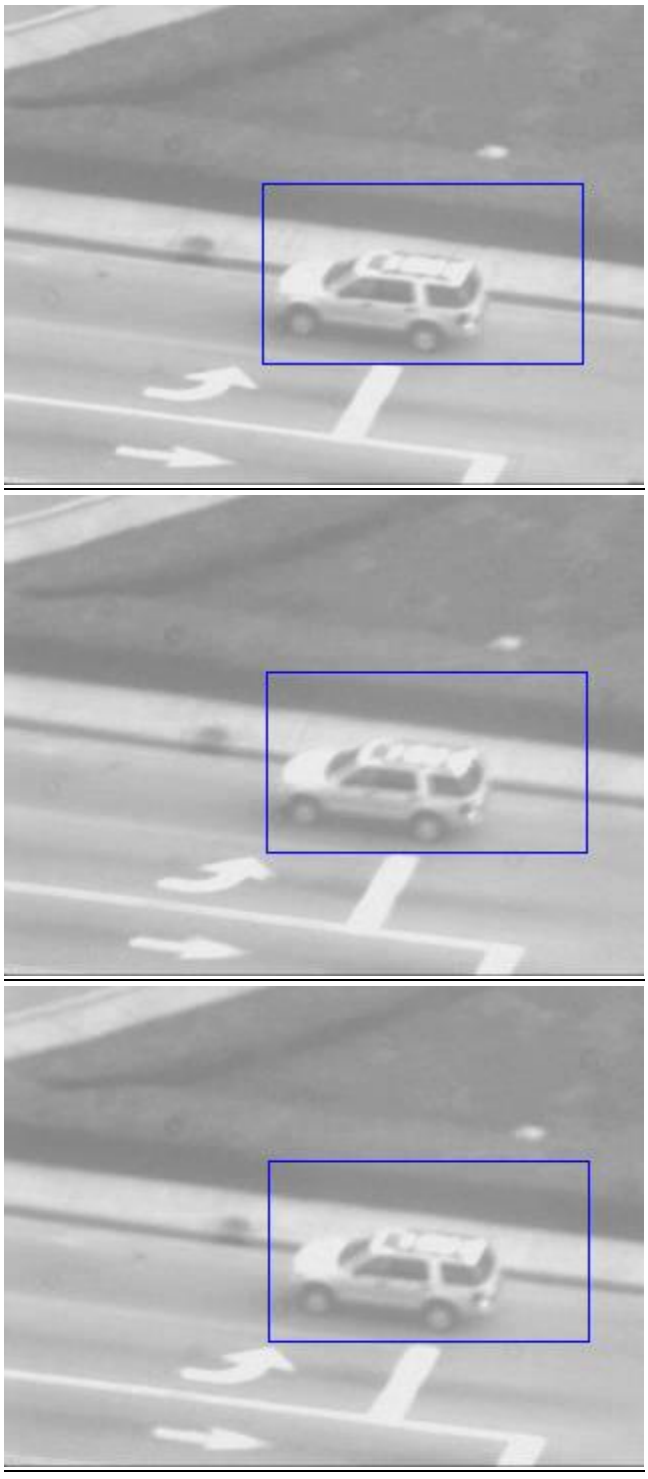


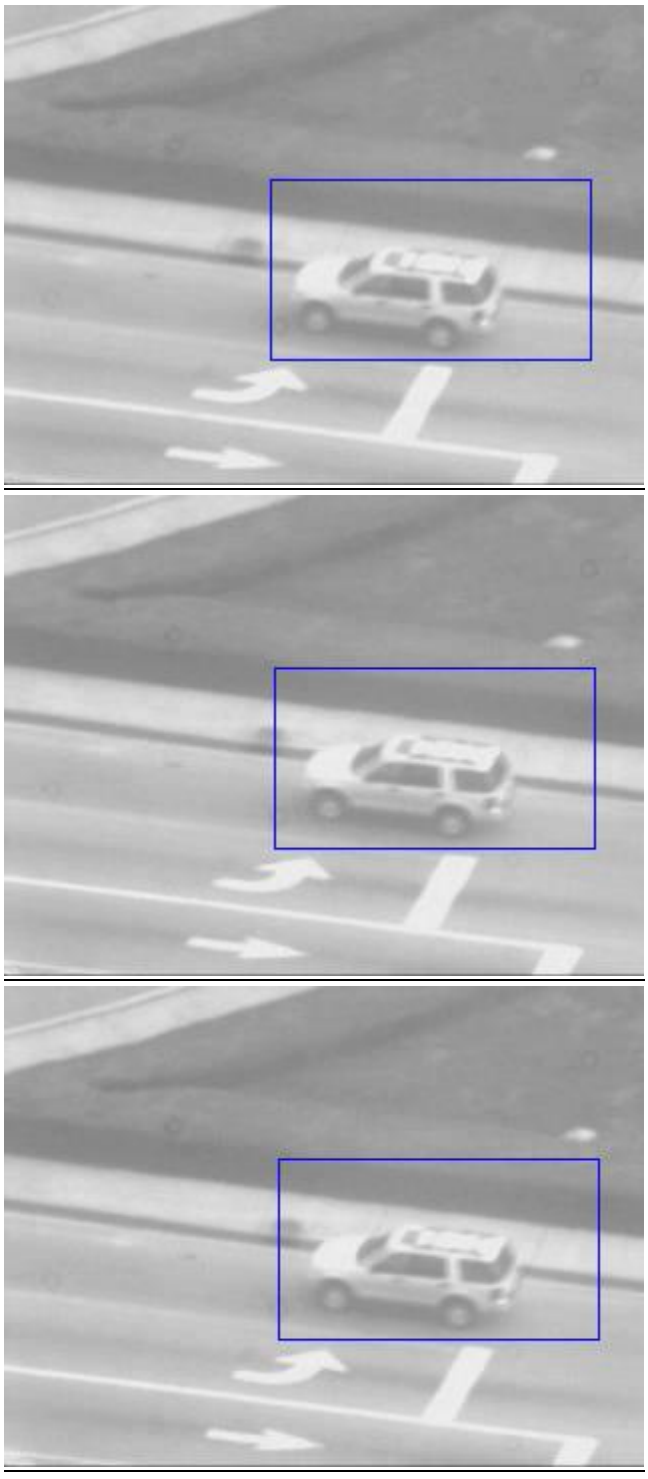


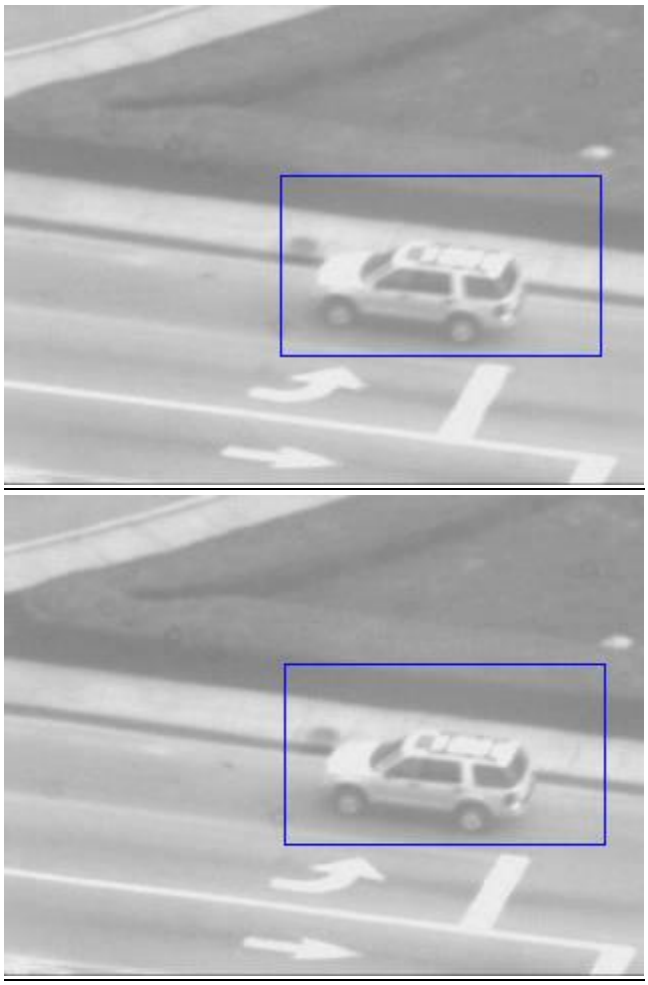












Landing


3 5


3 5


35


35


35

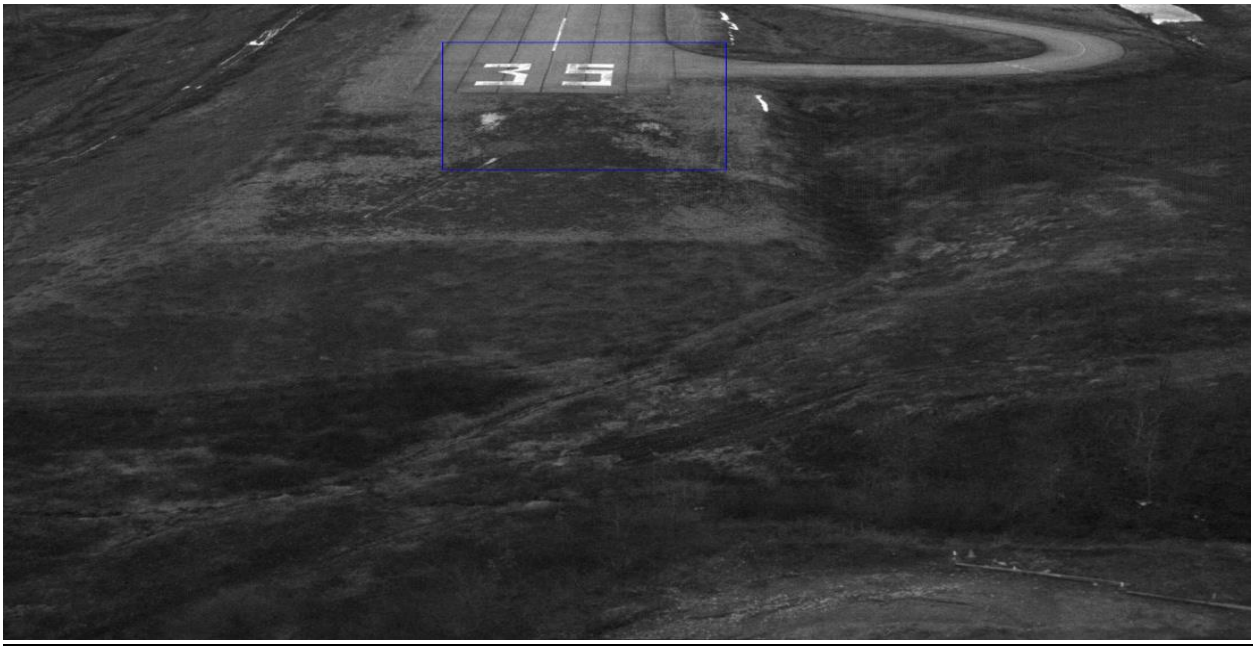

35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35


35