“[ENPM 673] HOMEWORK - 4”

ENPM 673 – PERCEPTION FOR AUTONOMOUS ROBOTS

HW4 REPORT

ADITYA VAISHAMPAYAN -116077354

ISHAN PATEL – 116169103

NAKUL PATEL – 116334877

Instructor: Dr. Cornelia Fermuller

CONTENTS

INTRODUCTION 2
WHAT IS OPTICAL FLOW? 2
LUCAS-KANADE METHOD 3
LUCAS KANADE ALGORITHM 3
TEMPLATE, WARPED, ERROR, AND GRADIENT IMAGES 4
FLOWCHART OF LUCAS KANADE ALGORITHM 8
LUCAS KANADE ALGORITHM VARIABLE SHAPES 9
BIG O NOTATION OF LUCAS KANADE ALGORITHM: 9
OUTPUT IMAGES 10
INTUTION BEHIND THE PROCESS: 11
CALCULATING THE WARP PERSPECTIVE 12
REFERENCES: 13

INTRODUCTION

One incredibly important aspect of human and animal vision is the ability to follow objects and people in our view. Whether it is a tiger chasing its prey, or you trying to catch a basketball, tracking is so integral to our everyday lives that we forget how much we rely on it.

To initialize the tracker, we need to define a template by drawing a bounding box around the object to be tracked in the first frame of the video. For each of the subsequent frames the tracker will update an affine transform that warps the current frame so that the template in the first frame is aligned with the warped current frame.

WHAT IS OPTICAL FLOW?

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second

Optical flow works on several assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighboring pixels have similar motion.

Consider a pixel $I(x, y, t)$ in first frame. It moves by distance (dx, dy) in next frame taken after **dt** time. So, since those pixels are the same and intensity does not change, we can say,

$$f_x u + f_y v + f_t = 0$$

$$f_x = \frac{\partial f}{\partial x};\; f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{dx}{dt};\; v = \frac{dy}{dt}$$

Above equation is called Optical Flow equation. In it, we can find f_x and f_y, they are image gradients. Similarly, f_t is the gradient along time. But (u, v) is unknown. We cannot solve this one equation with two unknown variables. So, several methods are provided to solve this problem and one of them is Lucas-Kanade.

LUCAS-KANADE METHOD

We have seen an assumption before, that all the neighboring pixels will have similar motion. Lucas-Kanade method takes a 3x3 patch around the point. So, all the 9 points have the same motion. We can find (f_x, f_y, f_t) for these 9 points. So now our problem becomes solving 9 equations with two unknown variables which is over-determined. A better solution is obtained with least square fit method. Below is the final solution which is two equation-two unknown problem and solve to get the solution.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

LUCAS KANADE ALGORITHM

Step1: Warp I with W (x; p) to compute I (W (x; p))

Step 2: Computing the error images T(x) – I (W (x; p))

Step 3: Warping the gradient ∇I with W (x; p)

Step 4: Evaluate the Jacobian $\frac{\partial W}{\partial p}$ at (x; p)

Step 5: Compute the steepest descent images $\nabla I \frac{\partial W}{\partial p}$

Step 6: Compute hessian matrix

$$\text{Hessian matrix} = \text{H} = \sum_x [\nabla I \frac{\partial W}{\partial p}]^T [\nabla I \frac{\partial W}{\partial p}]$$

Step 7: Compute $\sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[T(x) - I\big(W(x;p)\big)\right]$

Step 8: Compute Δp using the equation below:

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

Step 9: Update parameters p → p + Δp

Iterate above until $||\Delta p|| \leq \epsilon$

Also, there is a huge computational cost in calculating the hessian in every iteration. So, we can keep it constant and keep reusing it. Then each iteration will just consist Step 1,2,7,8,9 and the other steps will be pre-computed. Unfortunately, the Hessian is a function of in both formulations. Although various approximate solutions can be used (such as only updating the Hessian every few iterations and approximating the Hessian by assuming it is approximately constant across the image these approximations are inelegant and it is often hard to say how good approximations they are. It would be far better if the problem could be reformulated in an equivalent way, but with a constant Hessian.

TEMPLATE, WARPED, ERROR, AND GRADIENT IMAGES

Fig: template image

Fig: warped image

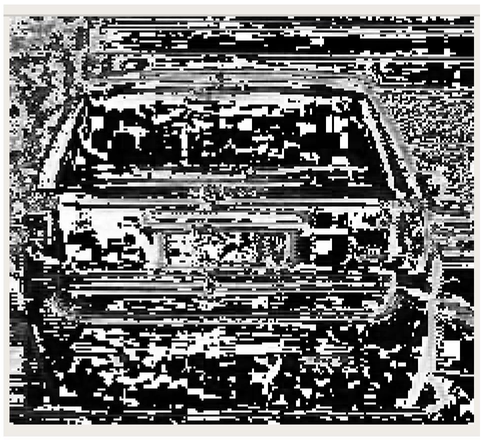


Fig: Error image

Fig: gradient in X direction

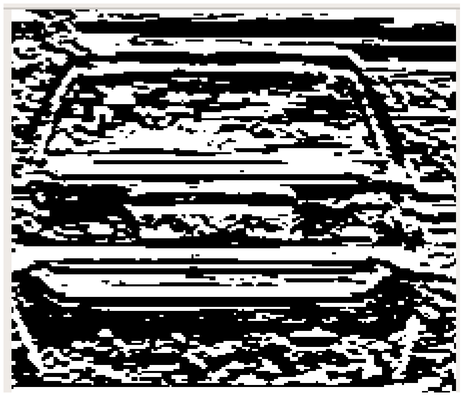


Fig: Gradient along the Y axis

Fig: steepest descent 1 image

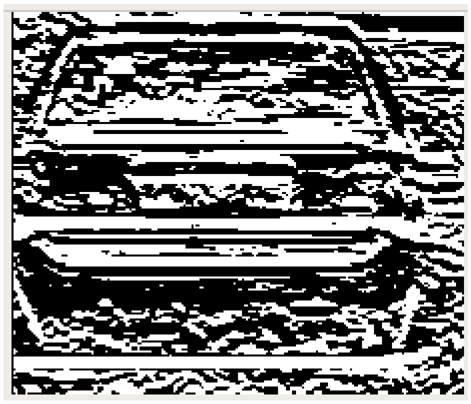


Fig: steepest descent 2 image

Fig: steepest descent 3 image

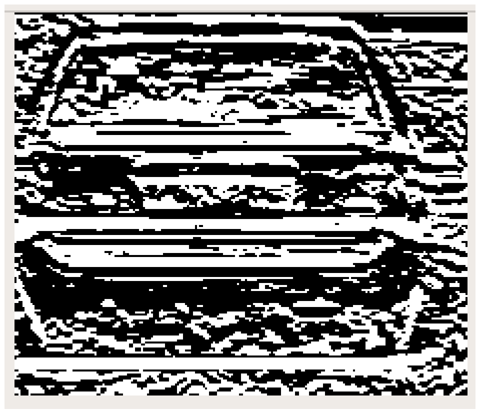


Fig: steepest descent 4 image

Fig: steepest descent 5 image

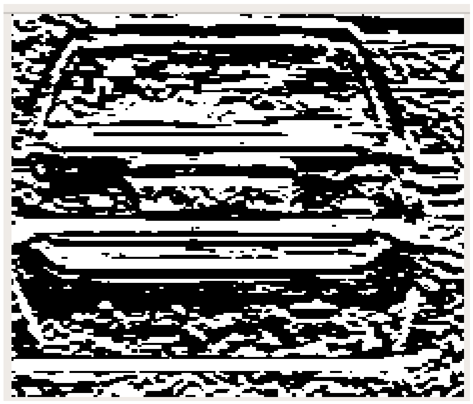


Fig: steepest descent 6 image

FLOWCHART OF LUCAS KANADE ALGORITHM

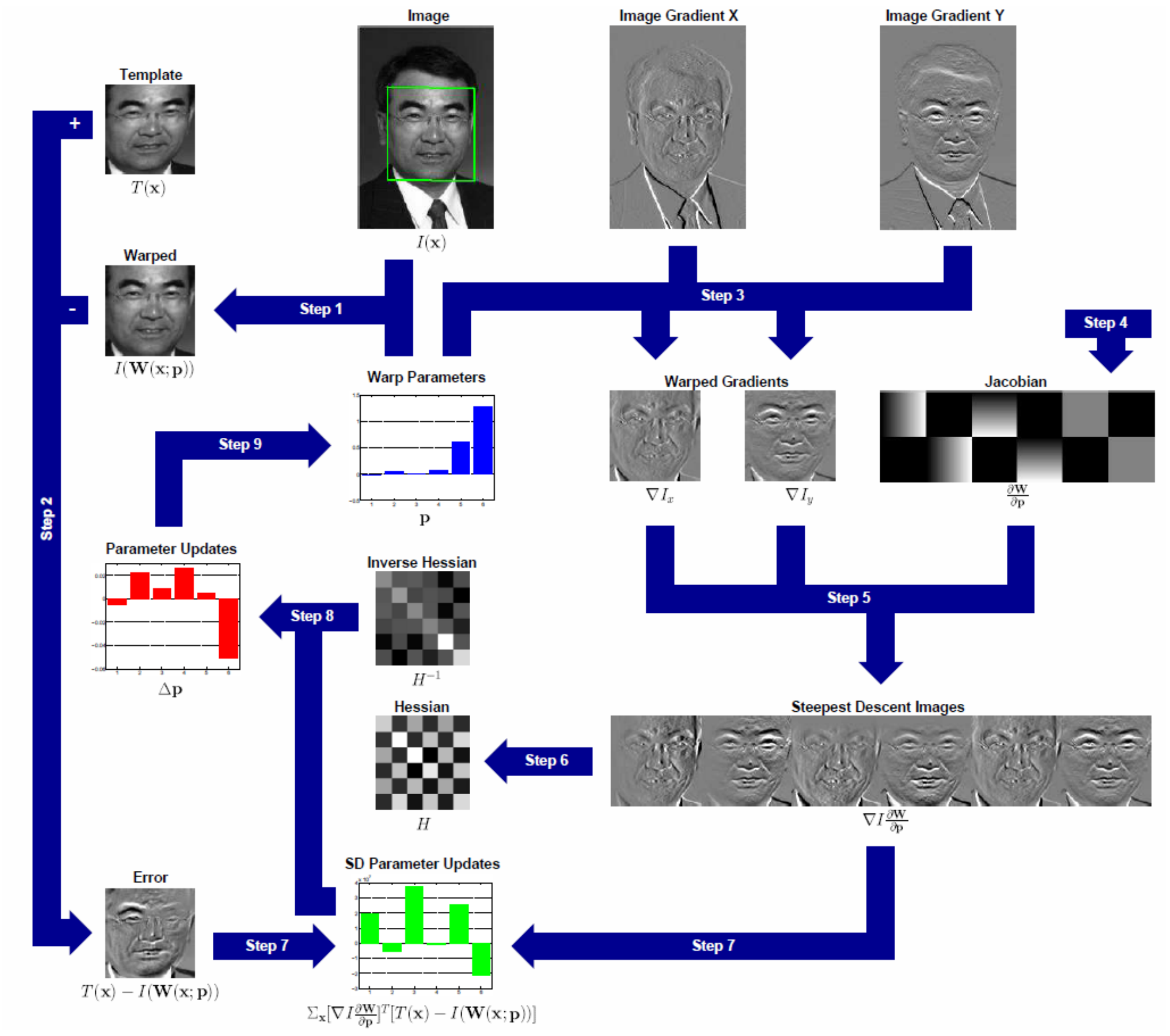
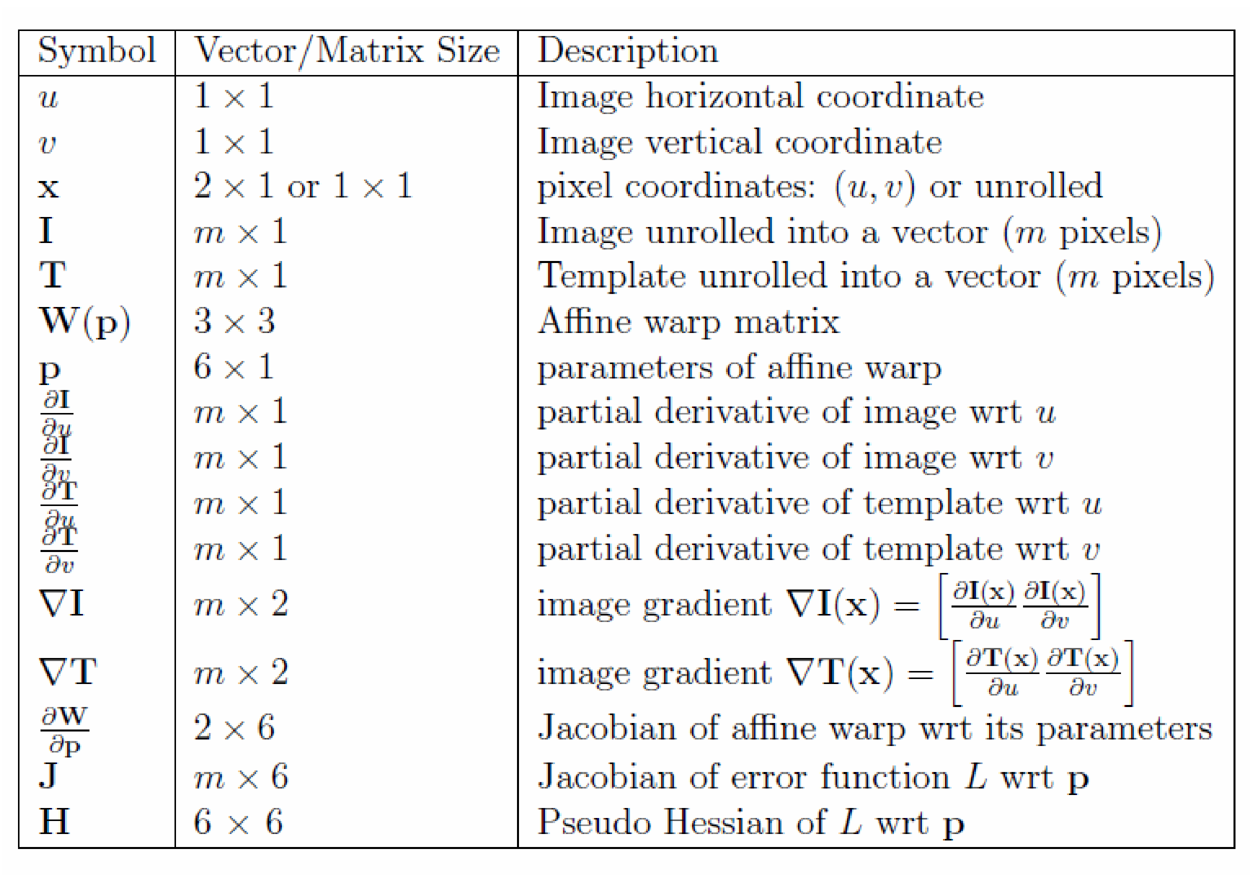


Fig: Lucas Kanade Algorithm

A schematic overview of the Lucas-Kanade algorithm. The image **I** is warped with the current estimate of the warp in Step 1 and the result subtracted from the template in Step 2 to yield the error image. The gradient of **I** is warped in Step 3, the Jacobian is computed in Step 4, and the two combined in Step 5 to give the steepest descent images. In Step 6 the Hessian is computed from the steepest descent images. In Step 7 the steepest descent parameter updates are computed by dot taking the dot product of the error image with the steepest descent images. In Step 8 the Hessian is inverted and multiplied by the steepest descent parameter updates to get the final parameter updates Δp which are then added to the parameters p in Step 9.

LUCAS KANADE ALGORITHM VARIABLE SHAPES

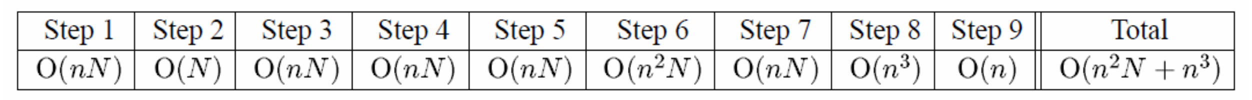


Symbol	Vector/Matrix Size	Description
u	1×1	Image horizontal coordinate
v	1×1	Image vertical coordinate
$\mathbf{x}$	2×1 or 1×1	pixel coordinates: (u, v) or unrolled
$\mathbf{I}$	$m \times 1$	Image unrolled into a vector (m pixels)
$\mathbf{T}$	$m \times 1$	Template unrolled into a vector (m pixels)
$\mathbf{W}(\mathbf{p})$	3×3	Affine warp matrix
$\mathbf{p}$	6×1	parameters of affine warp
$\frac{\partial \mathbf{I}}{\partial u}$	$m \times 1$	partial derivative of image wrt u
$\frac{\partial \mathbf{I}}{\partial v}$	$m \times 1$	partial derivative of image wrt v
$\frac{\partial \mathbf{T}}{\partial u}$	$m \times 1$	partial derivative of template wrt u
$\frac{\partial \mathbf{T}}{\partial v}$	$m \times 1$	partial derivative of template wrt v
$\nabla \mathbf{I}$	$m \times 2$	image gradient $\nabla \mathbf{I}(\mathbf{x}) = \left[\frac{\partial \mathbf{I}(\mathbf{x})}{\partial u} \; \frac{\partial \mathbf{I}(\mathbf{x})}{\partial v}\right]$
$\nabla \mathbf{T}$	$m \times 2$	image gradient $\nabla \mathbf{T}(\mathbf{x}) = \left[\frac{\partial \mathbf{T}(\mathbf{x})}{\partial u} \; \frac{\partial \mathbf{T}(\mathbf{x})}{\partial v}\right]$
$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$	2×6	Jacobian of affine warp wrt its parameters
$\mathbf{J}$	$m \times 6$	Jacobian of error function L wrt $\mathbf{p}$
$\mathbf{H}$	6×6	Pseudo Hessian of L wrt $\mathbf{p}$

Table: The shapes of all the variables used in Lucas Kanade Algorithm

BIG O NOTATION OF LUCAS KANADE ALGORITHM:

Assume that the number of warp parameters is and the number of pixels in is Step 1 of the Lucas-Kanade algorithm usually takes time O (n N). For each pixel x in T we compute W(x; p) and then sample I at that location. The computational cost of computing W(x; p) depends on W but for most warps the cost is O (n) per pixel. Step 2 takes time O(N). Step 3 takes the same time as Step 1, usually O (n N). Computing the Jacobian in Step 4 also depends on W but for most warps the cost is O(n) per pixel. The total cost of Step 4 is therefore O (n N). Step 5 takes time, O(n), Step 6 takes time $O(n^2 N)$, and Step 7 takes time O (n N). Step 8 takes time $O(n^3)$ to invert the Hessian matrix and time O(n^2) to multiply the result by the steepest descent parameter updated computed in Step 7. Step 9 just takes time O(n) to increment the parameters by the updates. The total computational cost of each iteration is therefore, O (n^2N + n^3) the most expensive step being Step 6.



Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Total
$O(nN)$	$O(N)$	$O(nN)$	$O(nN)$	$O(nN)$	$O(n^2N)$	$O(nN)$	$O(n^3)$	$O(n)$	$O(n^2N + n^3)$

OUTPUT IMAGES

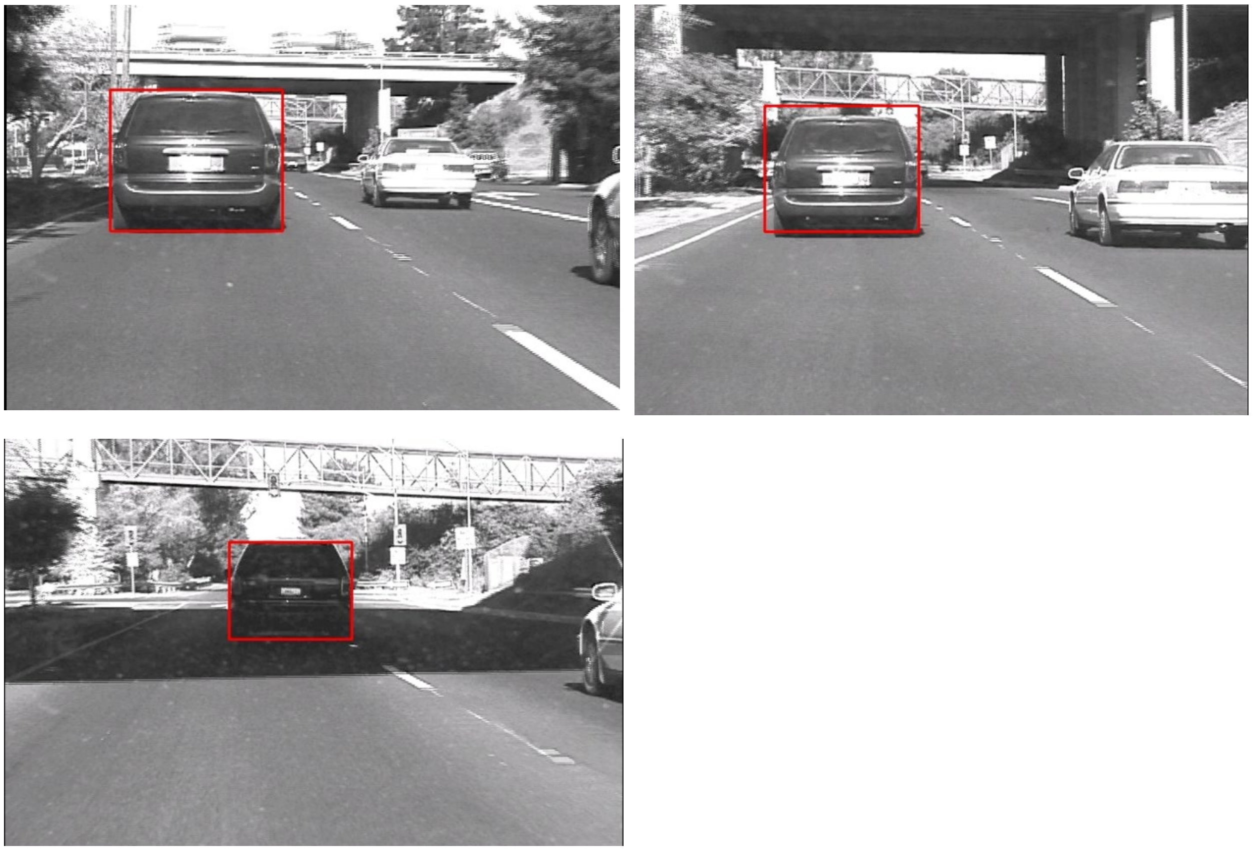


Fig: Car tracker

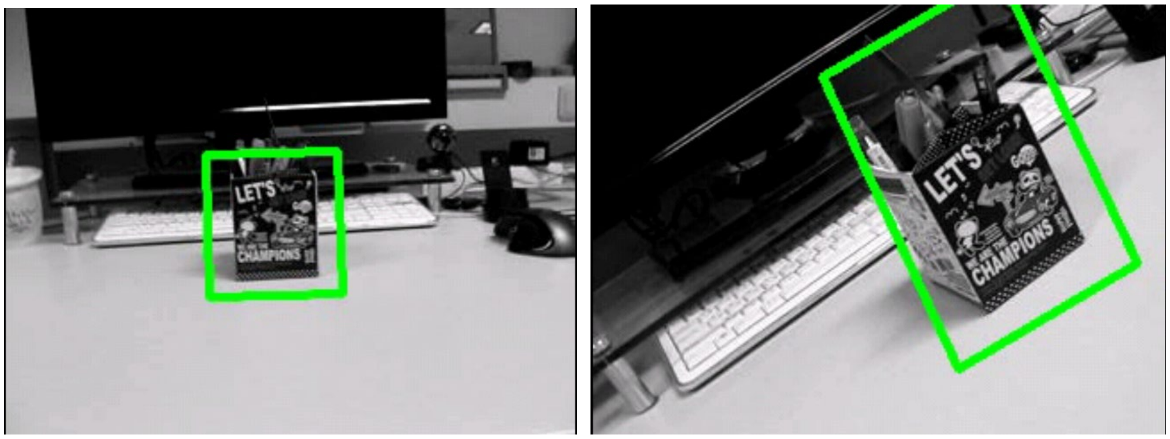


Fig: Vase tracker

INTUTION BEHIND THE PROCESS:

Basically, we try to minimize $min_p \sum_x \left[I\left(W(x;p)\right) - T(x)\right]^2$. This is a non- linear quadratic function of a non-parametric function. Function I is non parametric. So, we make a good initial guess of p so the equation can be written as

$$\sum_x \left[I\left(W(x;p + \Delta p)\right) - T(x)\right]^2$$

because we Assume known approximate solution and solve for increment. However, this is still a non-linear quadratic function, so we need to linearize it using Taylor series expansion.

The Jacobian is written as:

$$\frac{\partial \mathbf{W}}{\partial \boldsymbol{p}} = \begin{bmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_N} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_N} \end{bmatrix}$$

Now we re-write the previous equations as follows:

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

- hence we get this equation where
- x : pixel coordinate(2x1)
- I : the image intensity (scalar)
- W: warp function (2x1)
- P : warp paramters (6 for affine)

- ∇I : image gradient (1x2)
- $\frac{\partial W}{\partial p}$: Partial derivatives of warp function (2x6)
- ∇p : incremental warp (6x1)
- T(x) : template image intensity (scalar)

To solve the above equations we can write it as

$$min_p \sum_x \left[\nabla I \frac{\partial W}{\partial p} \Delta p - \{T(x) - I\big(W(x;\, p)\big)\}\right]^2$$

The equation looks like Ax – b so it can be solved using least squares approximation as follows:

$$\hat{x} = arg \min_x \|Ax - b\|^2$$

$$\text{and } x = (A^T A)^{-1} A^T b$$

Thus we can see that,

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

is similar to $x = (A^T A)^{-1} A^T b$

CALCULATING THE WARP PERSPECTIVE

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

where P is $(p_1, p_2, p_3, p_4, p_5, p_6)^T$

The key to efficiency is switching the role of the image and the template, as in, where a change of variables is made to switch or *invert* the roles of the template and the image.

Inverse affine warp is used as a part of inverse compositional algorithm,

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

Inverse of affine warp:

$$\frac{1}{(1+p_1)*(1+p_4)-p_2*p_3}\begin{pmatrix}-p_1-p_1*p_4+p_2*p_3\\ -p_2\\ -p_3\\ -p_4-p_1*p_4+p_2*p_3\\ -p_5-p_4*p_5+p_3*p_6\\ -p_6-p_1*p_6+p_2*p_5\end{pmatrix}$$

Equation for parameter updates:

$$\begin{pmatrix} p_1+\Delta p_1+p_1\cdot\Delta p_1+p_3\cdot\Delta p_2\\ p_2+\Delta p_2+p_2\cdot\Delta p_1+p_4\cdot\Delta p_2\\ p_3+\Delta p_3+p_1\cdot\Delta p_3+p_3\cdot\Delta p_4\\ p_4+\Delta p_4+p_2\cdot\Delta p_3+p_4\cdot\Delta p_4\\ p_5+\Delta p_5+p_1\cdot\Delta p_5+p_3\cdot\Delta p_6\\ p_6+\Delta p_6+p_2\cdot\Delta p_5+p_4\cdot\Delta p_6 \end{pmatrix}$$

REFERENCES:

- Simon Baker, et al. Lucas-Kanade 20 Years On: A Unifying Framework: Part 1, CMU- RI-TR-02-16, Robotics Institute, Carnegie Mellon University, 2002
- Simon Baker, et al. Lucas-Kanade 20 Years On: A Unifying Framework: Part 2, CMU- RI-TR-03-35, Robotics Institute, Carnegie Mellon University, 2003
- Bouguet, Jean-Yves. Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the algorithm, Intel Corporation, 2001