# ENPM 673: Perception for Autonomous Robots
# Project 4

Shreya Gummadi
Sneha Ganesh Nayak
Ghanem Jamal Eddine

19 April 2019

## 1 Lucas Kanade Implementation

In this project, we implement the Lucas Kanade algorithm. It aligns two images by minimizing the difference between the template and the image. It does this by implementing gradient descent. The gradient descent calculates the increment in the affine warp. We use the Lucas Kanade for the application of tracking three different objects: a car, a human and a vase.

## 2 Initialization of Object Coordinates

The first step in the algorithm is to generate a template to compare the rest of the frames to. This is done manually. For this part, we defined a function that uses the event handling property of matplotlib to get the coordinates of a bounding rectangle around the object. We choose the points and saved them to use for the further steps.

## 3 Warping and Error Estimation

The template and the image from each frame, is warped back onto the coordinate frame of the template. The minimization of error is performed with respect to p and the sum is performed over all of the pixels in the template image. We assume that a current estimate of p is known and then iteratively solve for increments to the parameters and hence the error is minimized and the parameters are updated. These two steps are iterated until the estimates of the parameters converge. Then we test for convergence, by checking if $\Delta p$ is below a particular set threshold.

## 4 Hessian Matrix Computation and Steepest Descent Estimation

$\Delta p$ from the previous step, gets us the steepest descent parameters. We then had to compute the Hessian matrix and invert it. The steepest descent parameters are then multiplied by the inverse of the Hessian matrix over each iteration. We can compute the Hessian times any vector with just two computations of the gradient. All steps of the Lucas Kanade algorithm must be repeated in every iteration because the estimates of the parameters p vary from iteration to iteration to iteration. We repeat iterations and break when we pass a certain threshold.

# 5 Evaluation of the Tracker:

## 5.1 Results

- **Car**

Tracker gets jumpy in a few frames due to changes in illumination.

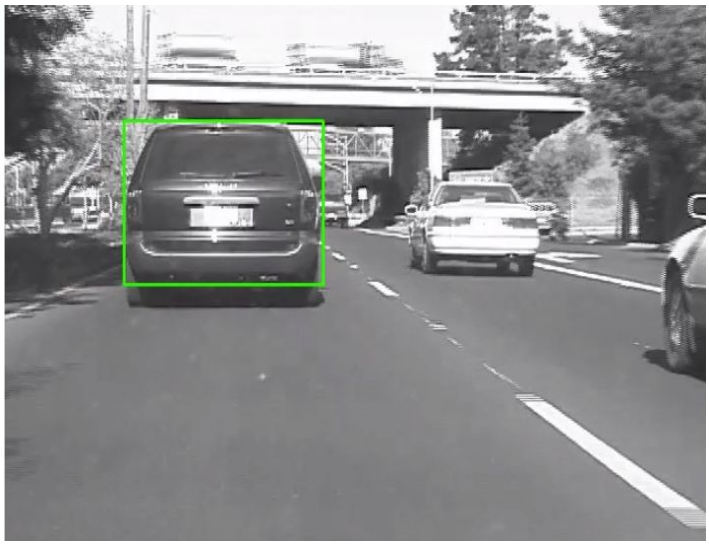

Fig 1. Car tracking

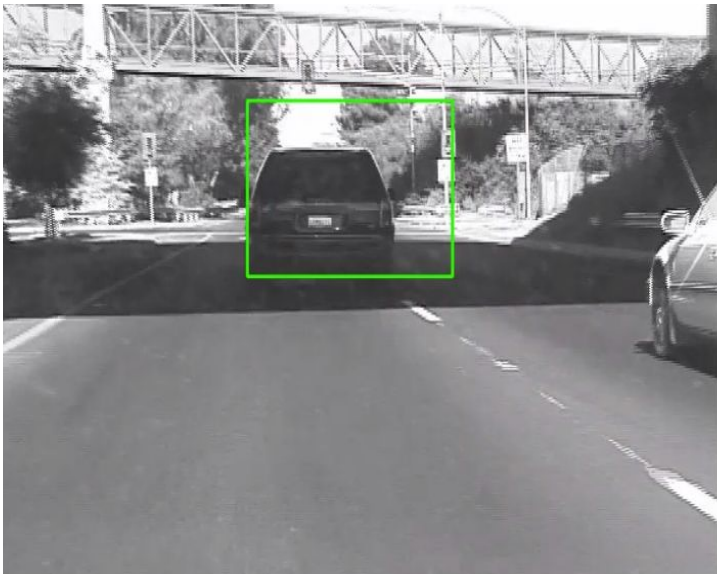

Fig 2. Car tracking

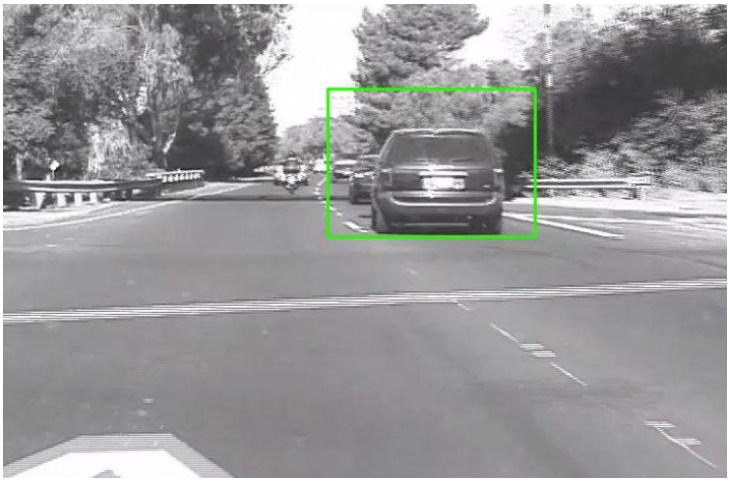

Fig 3. Car tracking

- **<u>Human</u>**
  Initially it is able to track the human accurately. However, it breaks at a certain point.

Fig 4. Human tracking

Fig 5. Human tracking breaks

Fig 6. Human tracking

- **<u>Vase</u>**
  For the vase, the 6 parameter affine should take into account the rotation. Even then, the tracker didn't work very efficiently.


Fig 7. Vase tracking


Fig 8. Vase tracking

### 5.2 Evaluation

**Templates that work well:**

- Templates with uniform color distribution, like the car's rear window
- Templates with only the object and no background pixels in them work better.

**When does the tracker break?**

- Tracker breaks when there is rotation.
- Tracker breaks down when there is a change in pixel distribution or illumination.

## 6 Robustness to Illumination

### 6.1 Scaling Brightness and Normalizing

Since the car sequence has a change in illumination the algorithm breaks. In order to overcome this, we increased the brightness of the template and all the frames. We also normalized the data by taking

subtracting the mean and dividing by its standard deviation. In doing so we got better results for the same data than before.

Fig 9. Car  tracking by increasing brightness

Fig 10. Car  tracking by increasing brightness