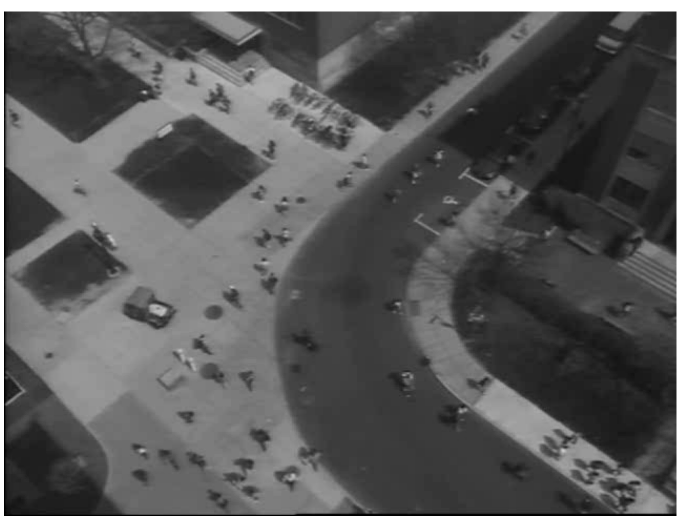
Computer Vision for HCI

Kanade-Lucas-Tomasi (KLT) Tracker

1

Motivation



2

Feature-based Tracking

- With small motion, a window/patch can be tracked by optimizing some matching criterion
  - But uniform texture are bad for tracking
- **Sparse** Optic Flow! (do only at select pixels)
- **How select a window/patch?**
  - Find "good" feature locations with enough texturedness/cornerness
- **How to track those windows/patches?**
  - Optical flow propagation

3

3

Overview

- Tracking framework
- How to select good features
- Examples

4

4

Overview

- **Tracking framework**
- How to select good features
- Examples

5

5

Tracking Features

Lucas-Kanade Optical Flow Algorithm

- Solve for optical flow vector **d** for a patch

Recall the "aggregate optical flow" from earlier!

$$\frac{dx}{dt} = u \qquad E = \sum_i (f_{xi}u + f_{yi}v + f_{ti})^2$$

$$\frac{dy}{dt} = v \qquad \frac{\partial E}{\partial u} = \sum_i (f_{xi}^2 u + f_{xi}f_{yi}v + f_{xi}f_{ti}) = 0$$

Uses *optimized* and *iterative* formulation along with pyramids

$$\frac{\partial E}{\partial v} = \sum_i (f_{yi}f_{xi}u + f_{yi}^2 v + f_{yi}f_{ti}) = 0$$

$$\underbrace{\begin{bmatrix} \sum_i f_{xi}^2 & \sum_i f_{xi}f_{yi} \\ \sum_i f_{yi}f_{xi} & \sum_i f_{yi}^2 \end{bmatrix}}_{G} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum_i f_{xi}f_{ti} \\ \sum_i f_{yi}f_{ti} \end{bmatrix}$$

Perform least squares with G^{-1} to solve

6

Tracking/Connecting Features

Over Long Sequences of Images

- Works well for tracking between "adjacent frames"
  - Motion is described well by simple translation model
- Individual frame-to-frame tracking over time unfortunately makes it possible for **drift** to occur
  - Since reference patch is constantly varying
  - Tracking can **"fall off" target object**
- Track frame-to-frame using translation model and verify that appearance of current patch is similar to that of the original patch
  - Check for large appearance changes
  - But added computation (see paper)

7

Overview

- Tracking framework
- **How to select good features**
- Examples

8

8

Selecting "Good" Features

- Success of tracking algorithm depends on pixel/patch quality of locations being tracked
- Define a good feature as a location that can be tracked well
  - Uniform? Edge? Texture?
- Recall Lucas-Kanade optical flow equation
- Feature can be tracked well if G
  - is above the image noise (eigenvalues of G are large)
  - well-conditioned (eigenvalues cannot differ by several orders of magnitude)

9

Selecting "Good" Features

Let $\lambda_1 \geq \lambda_2$ be the characteristic values of G
(Eigenvalues)

$\lambda_1 < \varepsilon \Rightarrow$ intensity is nearly constant over patch

$\lambda_1 >> \lambda_2 \Rightarrow$ edge was found

$\lambda_2 > \tau \Rightarrow$ corner or textured pattern was found

Good features to track satisfy $\lambda_2 > \tau$

10

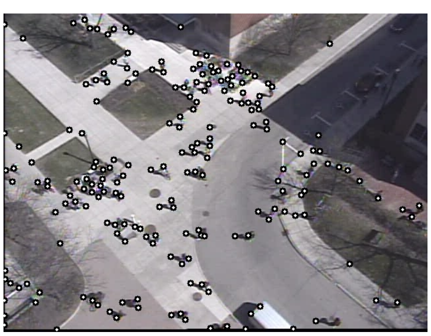
Overview

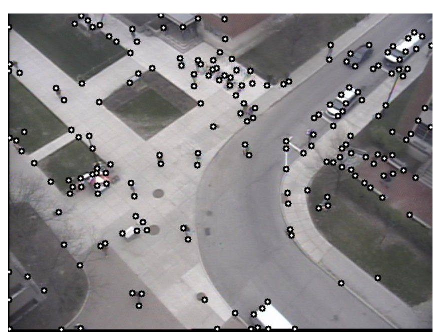
- Tracking framework
- How to select good features
- **Examples**

11

11

"Good" Features





Multiple good features
(at least 10 pixels apart between features)

12

Tracking "Good" Features

https://www.youtube.com/watch?v=6B_PNDCWtz4

http://www.youtube.com/watch?v=pmKtNQphq1E

13

Implementation

Open Source

- Real-time operation!
- Open source implementations available
  - OpenCV, etc.

14

Summary

- Feature-based tracking
- Select “good” features based on patch texturedness (using Eigenvalues)
- Track selected patches frame-to-frame using iterative Lucas-Kanade flow
- Check for dissimilarity from initial patch
- Open Source implementations available

15

15