Image-based Tracking
Image-based Tracking

- Many devices have cameras!
- Pixel accuracy ➔ computer vision

Marker Types for mobile phone tracking
- Templated
  - Restrictions on visual info in marker
  - Max. ~40 markers in practice
- ID-based
  - Simple or BCH binary encoding
  - 4096 markers, ≤ 4 errors
- DM
  - Data matrix code, ECC ≤ 60%
  - Virtually unlimited marker set
- Frame
  - Data encoded in marker border
  - Arbitrary visual info in marker
- Split
  - Data encoded in 2 disjoint marker parts
  - Arbitrary visual info in marker
- Grid
  - Track regular textured planar objects (e.g. maps)
  - Requires adding dots

local vs. global

Marker
- Natural Features
- Image-based Localization
- Visual Search
Image-based Tracking

• Many devices have cameras!
• Pixel accuracy $\Rightarrow$ computer vision
• Model-based tracking
  – Natural features
  – Computer graphic models
• Requires Initialization
• Not robust
  – Sensor fusion
  – Recovery methods
Starting Simple: Marker Tracking

- Has been done for more than 10 years
  - Some mobile phones today are faster than computers of that time
- Several open source solutions exist
- Fairly simple to implement
  - Standard computer vision methods
- A rectangular marker provides 4 corner points → enough for pose estimation!

(see last lectures)
Many, many designs

- ARToolKit
- ARSTudio
- ARTag
- ARToolKitPlus
- SCR marker
- IGD marker
- HOM marker
- CyberCode
- Visual Code from ETHZ
- USC’s multi-ring marker
- Intersense IS-1200 marker
- Shotcode

QR Codes
Marker Tracking Pipeline Overview

Goal: Do all this in less than 20 milliseconds on a mobile phone...
Marker Tracking – Overview

1. Camera Image
2. Fiducial Detection
3. Contours
4. Rectangle Fitting
5. Identified Markers
6. Pattern Checking
7. Rectangles
8. Lens Undistortion
9. Undistorted Corners
10. Pose Estimation
11. Estimated Poses
Marker Tracking – Fiducial Detection

• Threshold the whole image
• Search scanline per scanline for edges (white to black steps)
• Follow edge until either
  – Back to starting pixel
  – Image border
• Check for size
  – Reject fiducials early that are too small
Marker Tracking – Rectangle Fitting

– Start with an arbitrary point “x”
– The point with maximum distance must be a corner c0
– Create a diagonal through the center
– Find points c1 & c2 with maximum distance left and right of diagonal
– New diagonal from c1 to c2
– Find point c3 right of diagonal with maximum distance
– Repeat to check if no more corners exist
Marker Tracking – Pattern checking

• Calculate homography using the 4 corner points
  – “Direct Linear Transform” algorithm
  – Maps normalized coordinates to marker coordinates
    (simple perspective projection, no camera model)

• Extract pattern by sampling

• Check pattern
  – Id (implicit encoding)
  – Template (normalized cross correlation)

• 4 2D-3D correspondences ~ pose estimation
Marker Tracking – Corner refinement

• Refine corner coordinates
  – Critical for high quality tracking
  – Remember: 4 points is the bare minimum!
  – So these 4 points should better be accurate…

• Detect sub-pixel coordinates
  – E.g. Harris corner detector
    (specialized method can be faster and more accurate)
  – Strongly reduces jitter!

• Undistort corner coordinates
  – Remove radial distortion from lens
Marker Tracking – Pose Estimation

• Calculates the marker position and rotation relative to the camera

• Initial estimation directly from homography
  – Direct estimation from 4 2D - 2D correspondences
  – Very fast, but coarse
  – Jitters a lot...

• Refinement via Gauss-Newton iteration
  – 6 parameters (3 for position, 3 for rotation) to refine
  – Covered in Brian’s talk…
Less obtrusive Markers

- Keep 4 corner points for pose estimation
- Keep marker uniquely identifiable
- Minimize area occluded by marker structure
Dot Markers

- Overlay existing content with dots for easier tracking
- Dots cover only ~1% of the overall area
- 4 dots form a cell uniquely identifiable via template matching (32x32 pixels)
- Major problem: finding the regular grid of dots under perspective transformation

Original idea in [Rohs, 2007]
Markers in Action (Recorded from Phone)
Natural Feature Tracking

• More difficult than marker tracking
  – Markers are designed for their purpose
  – The natural environment is not…

• Less well established methods
  – Every year new ideas are proposed

• Usually much slower than marker tracking

• Simplest case - 2D target
SIFT in Action

Overall: 3.46 ms
Find Pts: 0.91 ms
Track Pts: 0.13 ms
Features: 0.78 ms
Outliers: 0.65 ms
Pose: 0.29 ms

Corners: 112
Matched Features: 59
Wrong Rotation: 0
Bad Linetest: 1
Bad Homographytest: 0
Correct: 58
From Tracking: 44
Levels: 46 12 0 0 0 0 0 0 0 0
Rotation: 5
Avg. Reproj. Err: 1.32
Overview on SIFT descriptor

• Keypoint localization
  – Searching for minima and maxima in scale space
    (also provides scale estimation for scale invariance)
  – Requires building pyramid of Difference of Gaussian
  – Major performance bottleneck of SIFT

• Feature description
  – Estimation of dominant 2D feature orientations
  – Orientation histogram of 4x4 sub-regions (128 bins)

• Feature matching
  – Feature database stored as k-d tree for sub-linear search time
PhonySIFT – Keypoint Detection

- Scale-space search way too slow…
- Replaced with FAST* corner detector
- Much faster, but scale estimation is lost
- Reintroduce scale robustness by building database on multiple scales

* [Rosten, 2005]
PhonySIFT – Descriptor Creation

• Estimate dominant keypoint orientation using gradient histogram
• Compensate for detected orientation
• Create descriptor from 3x3 sub-regions with 4 directions = 36 bins instead of 128
  – All weights pre-computed!
PhonySIFT – Descriptor Matching

- Brute force matching not an option
- K-d tree still not efficient enough
- New approach: Spill forest
  - Multiple spill trees (k-d tree with overlap)
  - Randomized dimensions for pivoting
  - Voting to merge results from different trees

![Memory Consumption Chart]

Overlap

- Memory Consumption vs. Number of Trees
PhonySIFT – Outlier Removal

• Cascade of removal techniques
• Start with cheapest, finish with most expensive…
  – Overall rotation check
  – Line tests
  – RANSAC for homography estimation
Overall rotation check

- SIFT provides keypoint rotation for free
- All keypoints must have same relative rotation
- Look at histogram and keep only peaks
Line test

- Pick two correspondences \(\rightarrow\) define line
- All other must be on the same side
PhonySIFT – Keypoint Tracking

• Most expensive step: descriptor creation
• Tracking of keypoints reduces descriptor creation frequency
  – Track both good and bad (unmatched) keypoints
  – Extract 8x8 patch for each keypoint
  – Check all keypoints at 25 pixel distance using template matching
PhonySIFT – Pose Refinement

• Based on Gauss-Newton iteration
  – Part of tracking pipeline that mostly benefits from floating point usage
  – Can still be implemented effectively in fixed point
  – Pose from homography makes good starting point

• Typically 2-4 iterations are enough…
Performance of SIFT and Ferns modified for mobile phone tracking

![Bar chart showing the comparison of tracking times for different devices and configurations.](image-url)
Speed Analysis for PhoneSIFT

Corner Detection  Feature describe and match  Outliers  Pose

Describing  Matching

Blur  Orient  Rotate  Vector  Spill Forest  SSD
NFT with SIFT on a Mobile Phone
## Markers vs Natural Feature Tracking

<table>
<thead>
<tr>
<th></th>
<th>Markers</th>
<th>NFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robustness</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>Speed</td>
<td>![ ✔️ ]</td>
<td>![ ✘️ ]</td>
</tr>
<tr>
<td>Memory requirements</td>
<td>![ ✔️ ]</td>
<td>![ ✘️ ]</td>
</tr>
<tr>
<td>Scalability</td>
<td>![ ✔️ ]</td>
<td>![ ✘️ ]</td>
</tr>
<tr>
<td>Visual clutter</td>
<td>![ ✘️ ]</td>
<td>![ ✔️ ]</td>
</tr>
<tr>
<td>Self-descriptive (data payload)</td>
<td>![ ✔️ ]</td>
<td>![ ✘️ ]</td>
</tr>
</tbody>
</table>
Detection in every Frame

• This is what most „trackers“ do…
• Targets are detected every frame
• Popular because detection and pose estimation are solved simultaneously
Doing it better: Dedicated Detection and Tracking

Target Recognition System

- Activate recognition system if tracking was lost
- Activate tracking system if target was found

Target Tracking System
PatchTracker Workflow (1)

• Target Detection:
  – find feature in reference image

• Target Tracking:
  – Take previous pose and apply motion model
    • Get estimate for what we are looking for
  – Create affine warped patches of reference features
    • Closely resemble how the feature should look in the camera image
  – Project patches into camera image and use normalized cross correlation (NCC) to match
Patch Prediction and Warping
Patch motion measurements

- Search half-res with low number of feature
  - E.g. 25 features, search radius of 5 pixels
- Refine estimated pose
- Search full-res with many features
  - E.g. 100 features, search radius of 2 pixels
- Refine for final pose
PatchTracker Workflow Analysis

• Affine warped patches allow very strong affine transformations (tilt close to 90°)
• NCC allows severe lighting changes
• 5 pixel search radius at half-res allows “wide” base-line
  – 5 x 2 x 20Hz = 200 pixels/sec
• Tracking 100 features at full-res strongly reduces jitter
PatchTracker in Action (1)
Detection and Tracking
PatchTracker in Action (2)
When does it break?

- **Losing Target**
  - Starting Frame
  - Tracking degrades
  - Last tracked Frame
  - First lost Frame

- **Occlusion**

- **Tilt**
When does it break?

Motion Blur

Reflection

Starting Frame  Tracking degrades  Last tracked Frame  First lost Frame
How fast is it?

We tested all trackers against the “simple” sequence (1) of the robustness tests of section V.D, since we wanted to prevent tracking failures as much as possible to measure full frames only.

During the first frame several lookup tables are created. Some these tables are for our fixed-point mathematics implementation (e.g. sine/cosine tables); others are specific to PhonySIFT. PhonyFerns and the PatchTracker do not use custom lookup tables. Creating the fixed-point math tables takes 38ms on the mobile phone and far below 0.1ms on the PC (it is not possible to measure this reliably, since it is executed only once per application call). The PhonySIFT lookup tables require 121ms to fill on the mobile phone and 0.3ms on the PC. Figure 10 shows the performance results on both mobile phone and PC. These measurements do not include the timings for the first frame.
## Orthogonal Strengths and Weaknesses

<table>
<thead>
<tr>
<th>Feature</th>
<th>SIFT/Ferns</th>
<th>PatchTracker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognize many targets</td>
<td>✔️</td>
<td>❌</td>
</tr>
<tr>
<td>Detect target</td>
<td>✔️</td>
<td>❌</td>
</tr>
<tr>
<td>Initialize tracking</td>
<td>✔️</td>
<td>❌</td>
</tr>
<tr>
<td>Speed</td>
<td>❌</td>
<td>✔️</td>
</tr>
<tr>
<td>Robust to blur</td>
<td>❌</td>
<td>✔️</td>
</tr>
<tr>
<td>Robust to tilt</td>
<td>❌</td>
<td>✔️</td>
</tr>
<tr>
<td>Robust to lighting changes</td>
<td>□</td>
<td>✔️</td>
</tr>
</tbody>
</table>
No need to stay in 2D

• Patches don’t need to come from a single plane
• Per feature
  – 3D position
  – Plane normal for affine warp
  – Multiple reference images
• Continue as before ...
Features in 3D

- Different geometry per per feature
- Defined by 3D shape
Tracking 3D Objects
Robustness

• Tracking loss
  – fast motion
  – occlusions
  – bad lighting
  – ...

• Trade-off between Delay and Robustness
Combination with Inertial Sensors

- Kalman filter for fusion
Tracking vs Initialization

Tracking

• Frame-to-frame
  – Relative
  – No start !
  – Fast

• Real-time experience
• Not robust

Initialization

• Without prior information
  – Absolut
  – Slow

• Required for
  – Initialization
  – recovery
  – robustness !
Lab 3 - Tracking with Vuforia toolkit

• https://ar.qualcomm.at/qdevnet/

• Register for an account
• And bring your own images

• More in the afternoon
Exam

• Written exam, 45 mins
• 6 - 8 questions with written answers
  – no calculation
• Cover material from block 1 - 3
  – slides
  – exercises

• In the last session of block 4