# **Tracking Algorithm Comparison and Implementation Guide**

## **1. Methodology for Comparing Two Tracking Algorithms**

**This document describes a repeatable methodology for comparing two visual tracking algorithms on Raspberry Pi boards using a shared LCD screen. The aim is to objectively measure speed and accuracy.**

### **Hardware Layout**

**- Large LCD/monitor to display prerecorded test videos  
- Raspberry Pi #1 runs tracker A  
- Raspberry Pi #2 runs tracker B  
- Identical cameras for both Pis  
- Rigid frame to hold LCD and cameras at fixed geometry  
- Power and cooling to avoid CPU throttling**

### **Test Content**

**Prepare several annotated reference videos covering translation, rotation, scale changes, occlusion and lighting changes. Store bounding boxes for evaluation.**

### **Software Setup**

**Configure each Pi with the same OS, libraries, and CPU governor set to performance. Log bounding boxes and processing time per frame.**

### **Calibration & Synchronization**

**Align cameras with the LCD. Use a visual flash for synchronization. Optionally calibrate lenses.**

### **Metrics**

**- FPS and latency  
- IoU accuracy vs ground truth  
- Success rate at IoU thresholds  
- Robustness (loss events and recovery time)  
- Drift and CPU temperature**

### **Procedure**

**Mount hardware, play video, run both trackers simultaneously, collect logs, compute metrics with scripts, repeat for robustness.**

## **2. Algorithm Implementations**

### **2.1 One-click: GrabCut → KLT + Similarity (RANSAC)**

**This algorithm initializes tracking with a single click. GrabCut segments the object, giving a bounding box. Shi–Tomasi features are extracted inside the box. Points are tracked frame-to-frame with Lucas–Kanade optical flow. A similarity transform is estimated using RANSAC to update the box. Periodically, new features are seeded.**

### **2.2 ORB Re-detect + KLT + PnP**

**This approach maintains 3D pose. ORB detects and matches features between the template and current frame. Homography and solvePnP compute the pose. Between detections, KLT optical flow tracks points for speed.**

## **3. Compilation Guides**

**Both implementations use OpenCV and C++17.  
  
Windows (MSVC + vcpkg):  
1) Install vcpkg and OpenCV (with contrib for ORB).  
2) Configure:  
 cmake -B build -S . -DCMAKE\_TOOLCHAIN\_FILE=path\to\vcpkg\scripts\buildsystems\vcpkg.cmake -A x64  
3) Build:  
 cmake --build build --config Release  
  
Linux (Debian/Ubuntu):  
 sudo apt update && sudo apt install build-essential cmake pkg-config libopencv-dev libopencv-contrib-dev  
 mkdir build && cd build && cmake .. && make -j$(nproc)**

## **4. Notes**

**- Ensure same resolution, exposure, and focus on both cameras.  
- Keep GrabCut iteration count small (2–4) for fast initialization.  
- ORB re-detect should run every 1–2s or on confidence drop.  
- Use threading and ROI to optimize FPS.  
- For pose estimation, calibrate the camera intrinsics.**