COSC450 Assignment 1 - Report

Arguments for the improved AR tracker are as follows:

Assignment1 <video source> <calibration file> <target> <tracker>

<video source> is either a device index (0 for the first camera) or a filename
<calibration file> contains intrinsic parameters
<target> gives chessboard dimensions, or an image of the target to track
<tracker> is the method used to track the target and estimate the pose:

- 'Chessboard' uses chessboard tracking

- 'SIFT_BF' uses SIFT features with BruteForce matching in each frame, and restricted feature matching

- 'SIFT_FLANN' uses SIFT features with k-NN matching

- 'ORB_BF' uses ORB features with BruteForce matching in each frame

- 'KLT' uses KLT tracking from frame to frame

Examples:
Assignment1 "C:\WindowsExample\SampleData\mists.mp4", " C:\WindowsExample\ calibData\calib.txt", " C:\WindowsExample\ SampleData\mists.png", "SIFT_BF"

Assignment1 "C:/LinuxMacExample/SampleData/hartleyZisserman.mp4", "C:/LinuxMacExample/calibData/calib.txt", "C:/LinuxMacExample/SampleData/hartleyZisserman.jpg", "KLT"

ORB Features

I was unable to implement ORB tracking successfully, as the included code compiles and runs but is wildly inaccurate. The ORB feature detection and use of Hamming distance are correct, however the quality and/or selection of matches may be problematic.

Taking a closer look at the execution and online implementations, I believe the knnMatch() method is not suited to binary descriptors. I attempted using the match() method but could not grasp the required arguments. It is also possible that the knnMatch method requires different parameters and thresholds for saving matches, but I was unable to find them when experimenting. The crossCheck option, while meant to give better matches with binary descriptors, caused a runtime error on my system.

ORB is meant to give fewer and higher quality matches than SIFT, so although less matches were found in execution than the comparative methods, the issue could alternatively be the handling of those matches. However, when debugging I could see no evidence of the matched points being any different in format or requiring different methods than the SIFT tracker to plug into solvePnP().

kd-based Matching

The SIFT_FLANN tracker initializes by detecting target features with SIFT and adding their descriptors to a FlannBasedMatcher, which uses *k*d-trees for more efficient indexing than brute force when finding feature correspondences.

When updating, the tracker detects features in the frame and knn-matches them to the target descriptors with k=2, filtering matches with a ratio of 0.8 for ambiguity. A homography between the matching target and frame points is found, and points that were determined by RANSAC to be inliers added to a final 3D (z = 0) target point vector and 2D frame point vector. These are then used to estimate the camera pose of that frame.

Restricted Feature Matching

Restricted feature matching is implemented on the SIFT brute-force tracker by way of a bounding box calculated from the previous frame's target corners.

The first frame is initialized with a bounding box encasing the entire image, and the following regions of interest are calculated after camera pose is found in the update method.

The target corners are projected into the frame using the same methods as KLT_Tracker, and the bounding box is calculated using an expansion factor of 1.2 times the width and the height (limited by the frame borders) to allow for moderate movement of the target in the next frame.

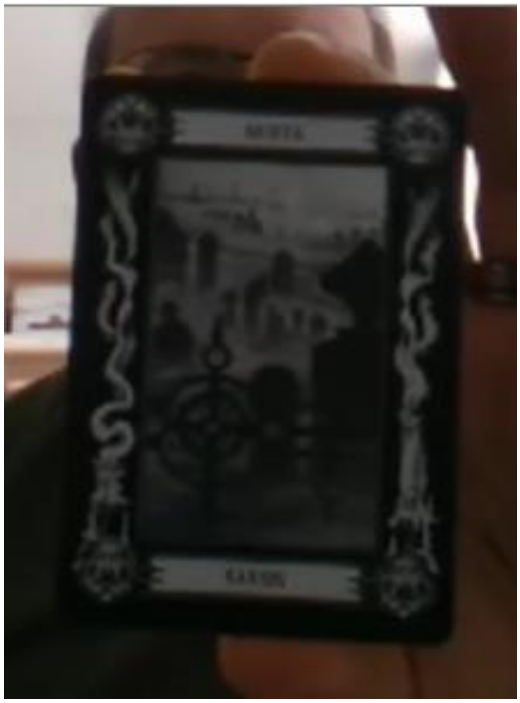

Figure 1 - restricted region of interest with expansion of 1.2x

The region-of-interest class member of SIFT_BF_Tracker is updated with the new bounding rectangle fields, and the following frame update applies a mask to the feature detection based on the rectangle of interest.

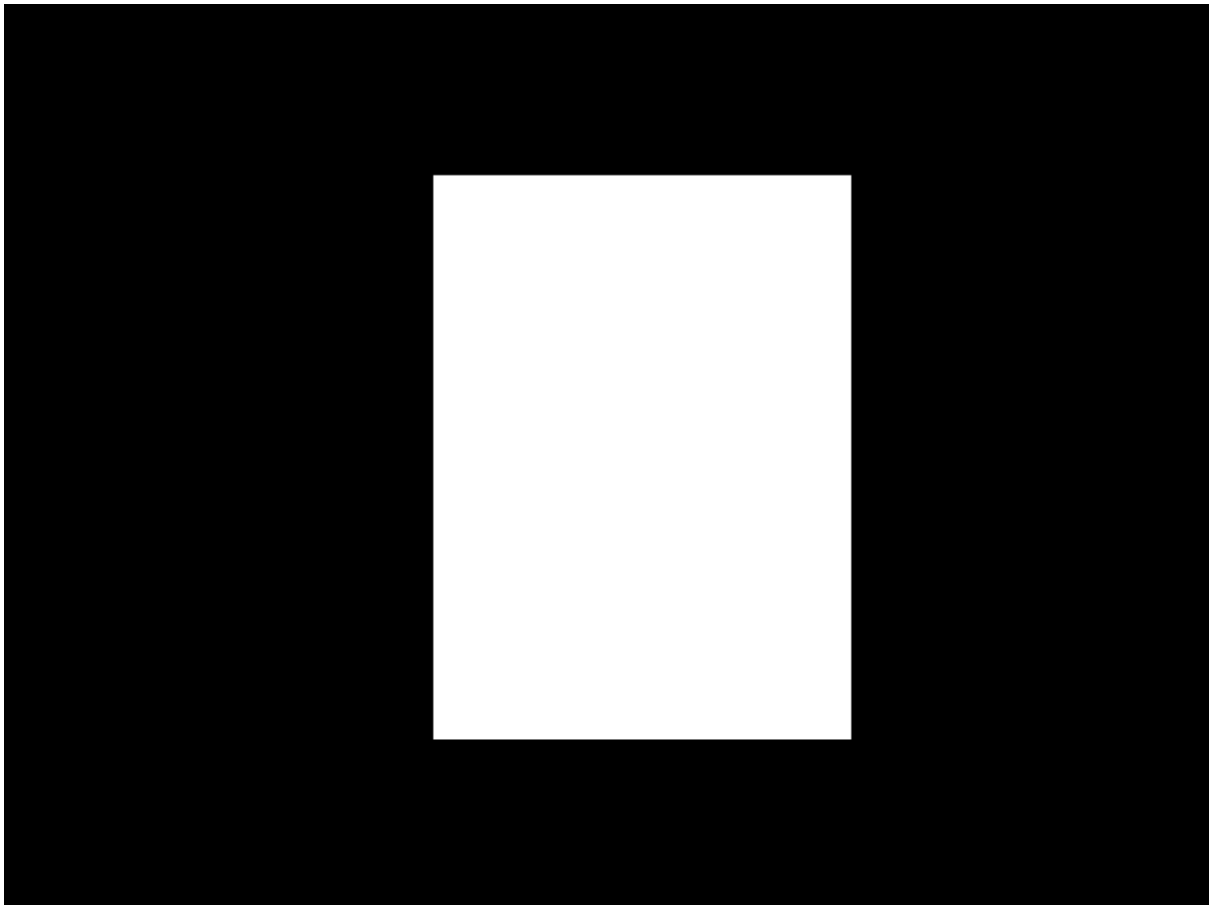

Figure 2 – mask used to restrict feature matching of frame

Initially a direct applying of the rectangle to the frame matrix was tried, but this caused all points found to be relative to that particular rectangle of interest, skewing the co-ordinate system for the next frame. This could be corrected by an offset operation on all points, but using a rectangular mask instead is more computationally efficient.

Kande-Lucas-Tomasi Tracking

When initializing, the KLT tracker needs an initial estimate of the target location, which is found with a SIFT_BF_Tracker estimate of the camera pose. The target image corner estimates are projected into the image, and a homography is found between the projections and the corners

Shi-Tomasi features are then found and transformed into the target image plane with the homography. The transformed features within the 2D bounds of the image are saved in 2D and 3D with z = 0.

When updating, the tracked features from the previous frame have a homography found between them and the target features from initializing. The Shi-Tomasi features successfully tracked and matching the homography, and their corresponding target features, are used to estimate the camera pose.

If less than 4 points are successfully tracked and match the homography (the minimum required for method solvePnP) , the tracker is re-initialized with reset target feature vectors.

Method testing

The test sequences for comparing trackers was the two provided videos of the “Mists” card and the textbook.

The speed of each method was measured by the average frames per second over a selection of 50 or 100 frames, for ease of measurement. This was chosen to fairly compare the overall performance of consistent methods like SIFT_FLANN, with faster methods that sometimes slow down upon re-initializing, like KLT.

```
// Update the tracker
pose = tracker->update(frame, camCal);
double fps = 1.0 / timer.read();

if (pose.tracked) {
    std::cout << fps << " frames/second" << std::endl;
    frames++;
    avg += fps;
    if (frames == 50)
        std::cout << "AVERAGE FPS " << avg/frames << std::endl;
} else {
    std::cout << "X " << fps << " frame/second" << std::endl;
}
```

Accuracy was measured by observation of the number of inaccurate mappings of the target in the measured frames. One of the factors in the n=50 frame tests was that original SIFT_BF Tracker, in particular, ran so slowly on the mists video that that human assessment of long-term accuracy became tedious.

Three frame collections were selected to test examples of translation, rotation and obscuring of the target that occurred in the complete mists video. A collection of frames from the textbook video was also tested to compare the impact of a visually different target. Due to this tests running faster, therefore allowing a longer accuracy recording, the frame number was extended to 100.

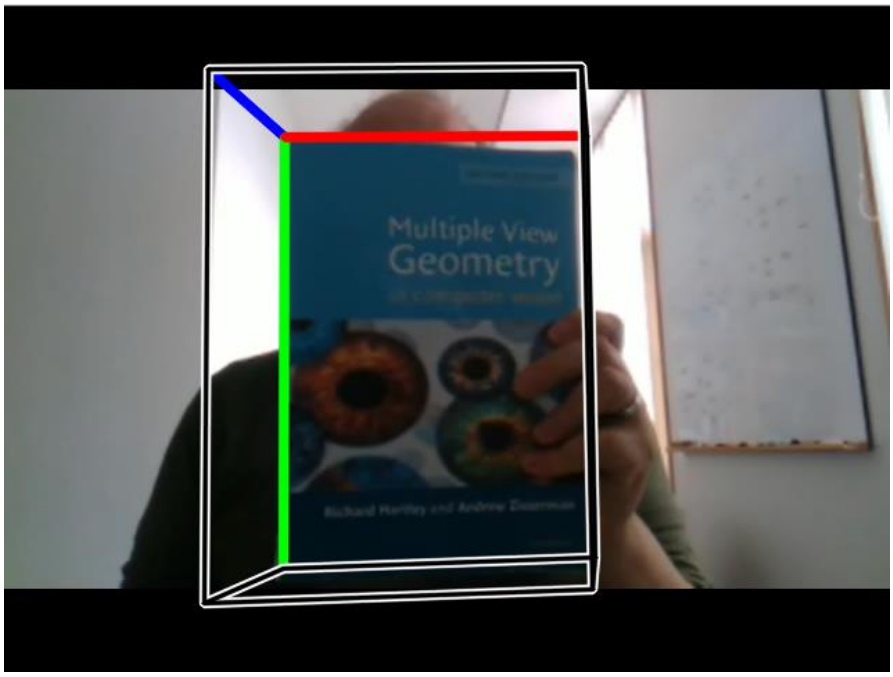

Figure 3 - A passing frame. Small inaccuracies were allowed

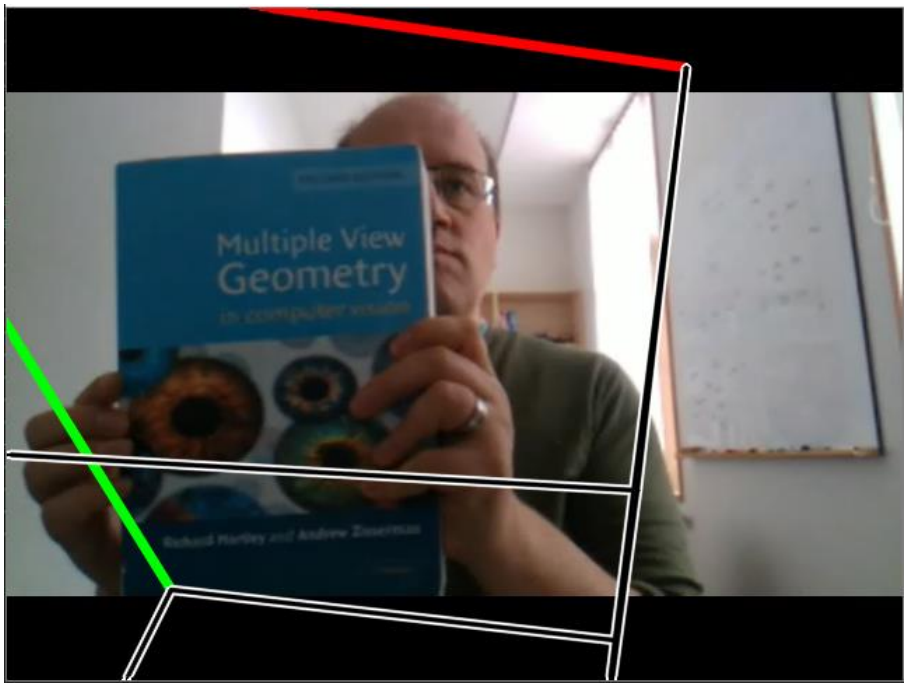

Figure 4 - A failing frame

Results

Mists video – frames 1-50

This frame collection was chosen to test tracking with a translating object.

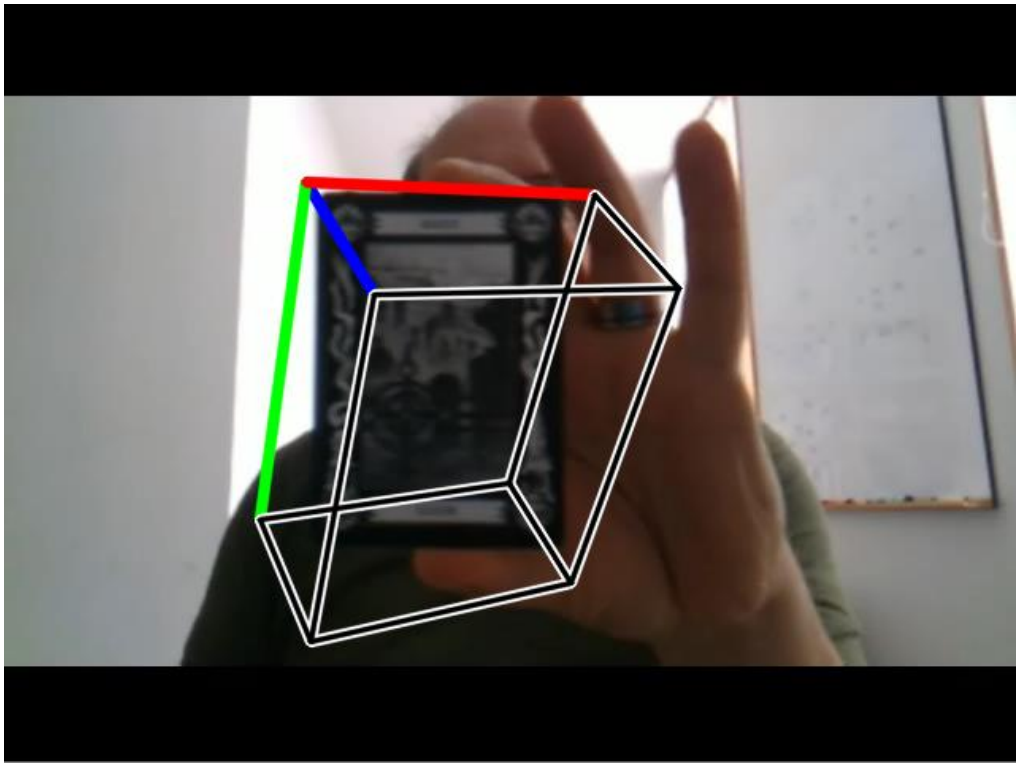

Figure 5 - an incorrect frame from KLT_Tracker

	SIFT_BF (no restriction)	**SIFT_FLANN**	**SIFT_BF (restricted)**	**KLT**
Avg FPS	0.723101	1.77504	0.865841	6.45536
Wrong frames	6%	6%	6%	22%

The three SIFT-based trackers showed similar levels of accuracy (93%), which was to be expected given their same method of feature detection.

Within SIFT, restricting feature matching gave a modest improvement in speed (≈20%), while *k*d-based matching gave a large improvement (≈145%).

KLT Tracking was less accurate at 78%, but significantly faster than any of the other methods.

Mists video – frames 201-250

This frame collection was chosen to test tracking with a rotating object.

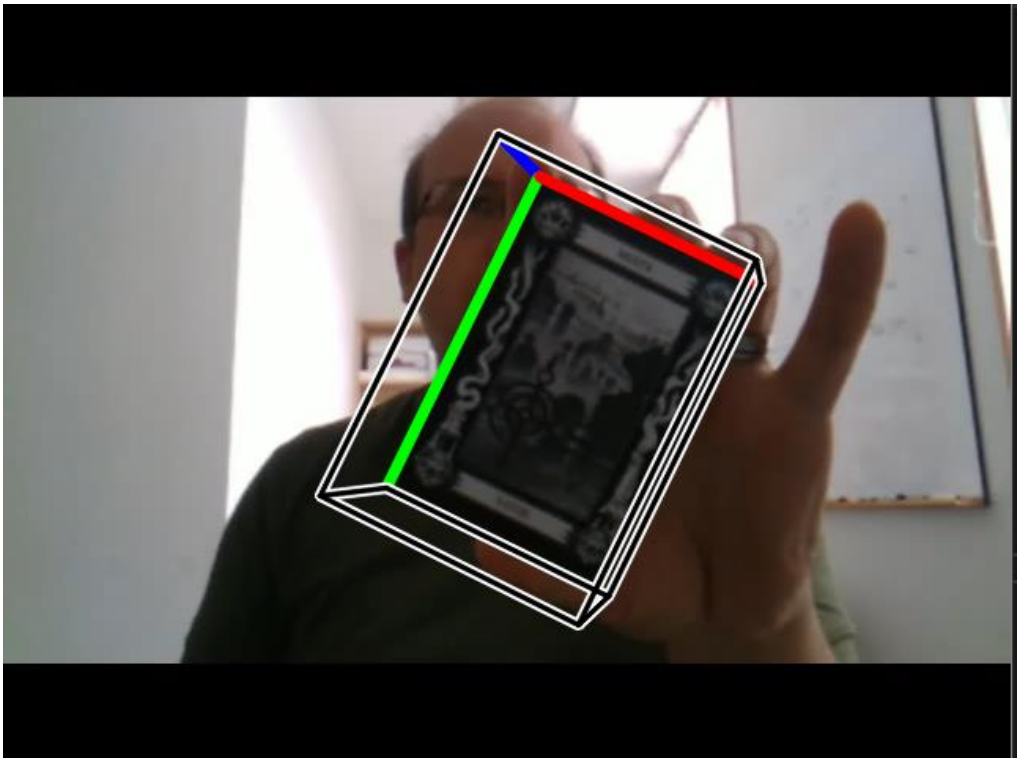

Figure 6 - A correct frame from SIFT_FLANN_Tracker

	SIFT_BF (no restriction)	**SIFT_FLANN**	**SIFT_BF (restricted)**	**KLT**
Avg FPS	0.54924	1.13614	0.634216	2.55208
Wrong frames	0%	0%	0%	~80%

The rotating object slowed speed for all trackers, and was effectively untrackable by the KLT tracker.

Mists video – frames 471 – 520

This frame collection was chosen to test tracking an obscured object.

Figure 7 - A correct frame from restricted SIFT_BF_Tracker

	SIFT_BF (no restriction)	**SIFT_FLANN**	**SIFT_BF (restricted)**	**KLT**
Avg FPS	0.599476	1.74051	0.84544	2.33175
Wrong frames	0%	0%	0%	~90%

The obscured target decreased the speed but not the accuracy of the SIFT_BF tracker. The SIFT_FLANN and restricted SIFT trackers performed comparably with the non-obscured translating target, while the KLT tracker became even more inaccurate.

Textbook video – frames 1-100

This frame collection was chosen to test tracking with a different translating and obscured object

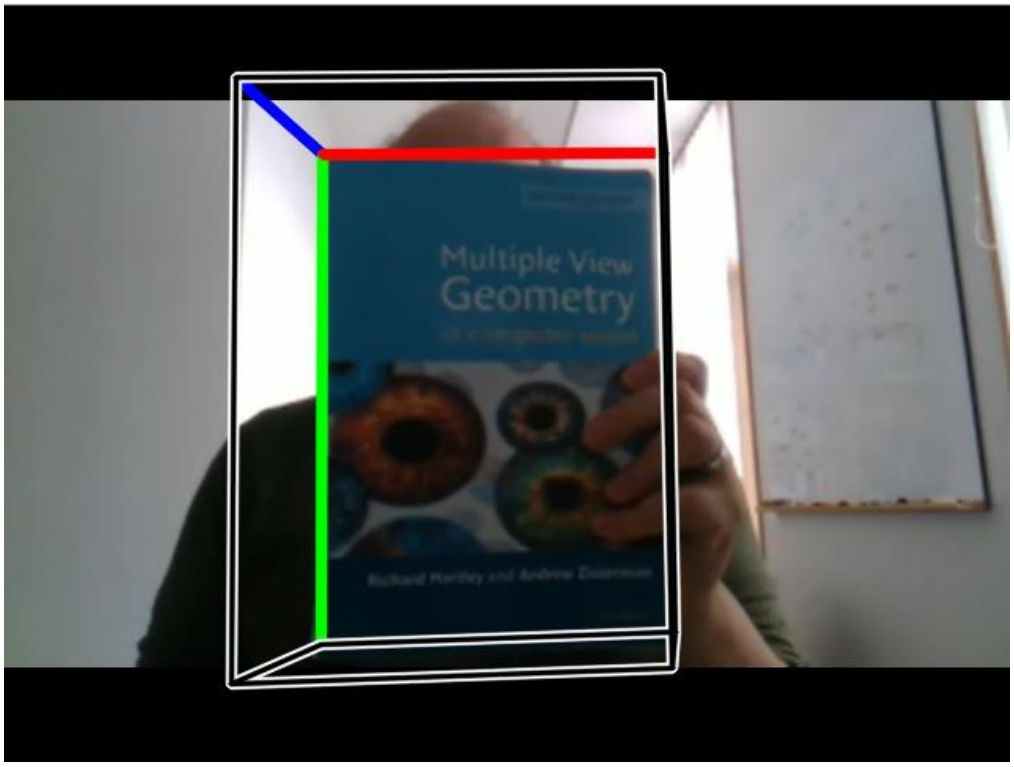

Figure 8 - a correct frame from SIFT_BF_Tracker

	SIFT_BF (no restriction)	**SIFT_FLANN**	**SIFT_BF (restricted)**	**KLT**
Avg FPS	1.4379	1.75553	1.47535	6.28169
Wrong frames	0%	0%	0%	19%

The change in target uniformly improved speed and accuracy across all trackers compared to the translating mists card sequence.

The textbook target was also successfully tracked when obscured by the KLT_Tracker, unlike the mists target.

Discussion

The feature matching gave a very minimal improvement in speed over non-tracked methods, however I believe this may have been to do with inefficient implementation of the bounding box creation and application. Combining FLANN-based matching and a more efficient bounding box would have created the fastest of the accurate methods (in the absence of ORB detection).

As said in the assignment specification, motion prediction was outside the scope of this assignment. A rectangular bounding box was chosen for its ease in expanding and catering to any movement or rotation, but using a Kalman filter (or similar) could allow for a smaller and more accurate region of interest in further improvements. Given knowledge of the target shape and assuming such an algorithm exists, prediction of the rotated shape of the target could also occur. Particularly when the card or chessboard rotates in 3D towards or away from the camera, its 2D shape changes quickly in a way that thwarts a too-small quadrilateral bounding area. In a context where an object with distinct rotated profiles is rotating frequently in a predictable way, and there is much to gain from reducing the feature detecting area, this could be advantageous.

The KLT tracker ran extremely fast, but was somewhat inaccurate with translation. Many of the incorrect boxes were highly skewed into the frame's black borders, so some recognition of physically impossible target shapes could be implemented to indicate the need to re-initialize on these frames.

The KLT tracker was also unable to track the rotated and obscured mists target at all. The target mappings generated were usually matching at least one corner, but highly distorted. It's unclear as to why this occurred, but some combination of motion tracking and target shape awareness could be used in the future to re-initialize on these incorrect mappings.

It did, however, track the obscured textbook successfully. The textbook gave greater accuracy in general than the mists card – this may have been due to the textbook being more distinguishable from the dark sweater section of the background than the mists card (even in greyscale for Shi-Tomasi features)

For human-like accuracy determination on a larger scale, an image recognition neural network could be used in the future to assess target matching over long image sequences and slow executions. While this perhaps supersedes the point of a Computer Vision feature tracker, the two approaches can inform each other's design patterns. A conventional vision algorithm also has the advantage of being clearer and slightly more accountable in its logic, perhaps relevant to sensitive images and decision contexts like criminal facial recognition.

References

ORB feature detection research:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html
https://blog.francium.tech/feature-detection-and-matching-with-opencv-5fd2394a590

Restricted feature detection:
https://answers.opencv.org/question/14942/create-a-mask-with-a-roi/

KLT method syntax:
https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html
https://github.com/opencv/opencv/blob/master/samples/cpp/lkdemo.cpp