Feature Tracking for AR

COSC450 Assignment 1

Due: Monday, 20 April 2020, 11:59pm

1 Overview

The AR tracker demonstrated so far uses feature matching to locate the target in each frame. There are several ways that this can be improved, and in this assignment you will explore three of them:

1. Use more efficient feature detection and matching,
2. Restrict feature detection to a smaller region of the image,
3. Use a feature tracker, rather than matching.

You will write a program using the OpenCV library to implement these approaches. A skeleton solution is provided in `~steven/Public/COSC450/2020/Assignment1`. You will then conduct experiments to evaluate the speed and reliability of these two approaches, comparing them to one another and to the brute-force SIFT matcher provided in the skeleton code.

1.1 Workload Expectations

The total workload for the paper is around 240 hours, so a 20% assignment should take around 45-50 hours to complete. The algorithms required for this assignment are all implemented in OpenCV, but you may have to conduct some research to find out how to use the methods – sometimes the format of input and output is not immediately obvious from the documentation. There are, however, a lot of examples online. It is OK to refer to these, but make sure you provide appropriate citations in your code and report.

In addition to implementing different approaches, a key aspect of this assignment is evaluating them. There are two main criteria for evaluation – speed and reliability. Speed is relatively easy to measure, and a simple `Timer` class is included in the skeleton code. Reliability is harder to measure, but you could consider experimenting with videos that have different speeds of motion, angles of view on the target, or levels of occlusion.

While the tracker can run from a live video stream, it is best to use recorded sequences for evaluation. This ensures that all methods are getting the same input, and allows you to set up specific test cases to measure reliability.

2 Tracking Methods

2.1 More Efficient Feature Matching

The feature based tracker provided uses SIFT features [1] and brute-force matching. SIFT gives good quality matches, but other methods can be faster and nearly as good. ORB features [4], in particular are often used for real-time tracking. These are implemented in OpenCV as `cv::ORB`, but require some adjustments to the matching routines as they use binary descriptors, rather than a vector of numbers. This means that the Hamming distance needs to be used rather than a Euclidean distance.

A second way to speed up the matching process is to use more efficient indexing when finding feature correspondences. A common way to do this is to use kd-trees [3], which are implemented in OpenCV as `cv::FlannBasedMatcher`. This is an alternative to

For the Assignment: Implement a matching-based `Tracker` similar to `SIFT_BF_Tracker`, but using ORB features and kd-based matching. You should then conduct experiments to see whether this is faster; whether any speed increases come primarily from the use of ORB features or kd-tree matching; and what (if any) changes there are in the reliability of the tracking.

2.2 Restricted Feature Matching

The OpenCV feature detectors can take a mask to determine which parts of the image are searched for features. This can be set to `cv::noArray()` to search the whole image. To search just part of the image, you can pass in a mask image that is black and white, with features only being detected in the white part. The difficult part is determining which part of the image to search.

One approach is to project the corners of the target into the image. There are a few options as to how to proceed from there, as illustrated in Figure 1:

- You could make a mask using the quadrilateral around the target.
- You could make a bounding box, which is a simpler shape to draw. If you use this approach, you can call a featue detector with just a rectangular region of interest, rather than a mask
  `cv::Rect roi(leftX, topY, width, height);`
  `detector->detectAndCompute(image(roi), cv::noArray(), ...);`
- You could extend either of these regions a little to allow for motion of the target. A more advanced approach would be to predict the movement (using a Kalman filter or similar), but that is beyond the scope of this assignment.

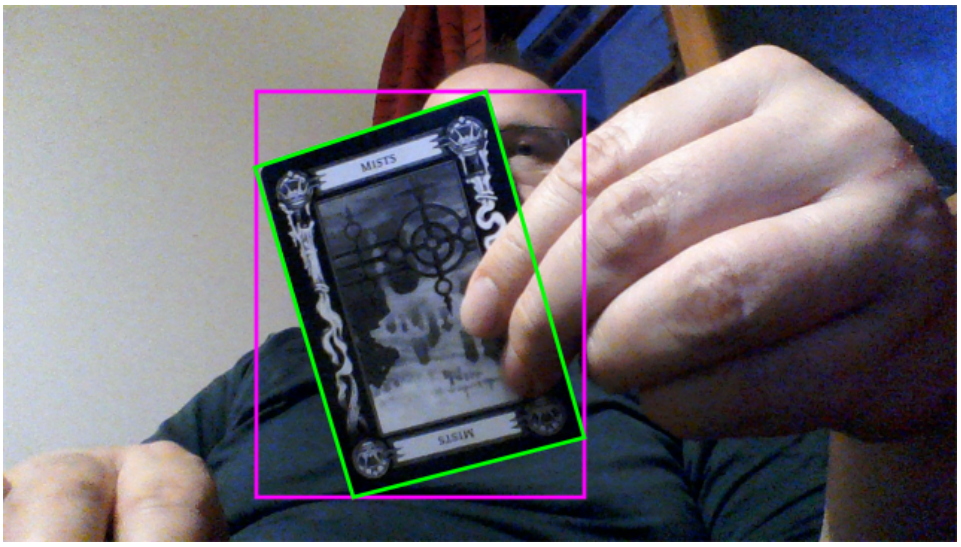

Figure 1: The search for matching features could be restricted to the target area (green box) or a bounding box (magenta box). Either of these could be extended to allow for target motion.

2.3 Feature Tracking

Using SIFT or ORB features to find the target is effective, but ignores prior knowledge when tracing from frame-to-frame. A more efficient approach is to track features starting from their previous position, and Kande-Lucas-Tomasi (KLT) tracking is a common way to do this. Rough pseudocode for such a tracker could be:

```
KLT_Tracker::initialise(target, frame, calibration) {

    //Build a SIFT or ORB matcher from the target image

    //Locate the target and comptue pose using SIFT/ORB matches

    //Find Shi-Tomasi features in that region, and estimate their target locations

    //Return the initial pose from SIFT/ORB
}

KLT_Tracer::update(frame) {

    //Track features from the last frame using Lucas-Kande

    if (sufficient features tracked) {
```

```
        //Use their (remembered) target locations to estimate pose
    } else {
        //Reinitialise tracking from the target image
    }

}
```

To initialise the tracker we still need to do SIFT (or ORB) matching. The KLT tracker needs an initial estimate of the target location, which we can get using these methods. Then we find Shi-Tomasi features [5], which are designed to be easy to track. Using the region-based restriction on feature matching from the last section we can initialise just features on the target in the current frame. These can then be tracked with the Lucas-Kanade method [2], but we need to find the corresponding locations in the target image. You can do this by finding a homography between the current frame and the target based on the SIFT/ORB features – see the `update` method of the `SIFT_BF_Tracker` for an example. This can then be used to transfer the Shi-Tomasi locations to the target image.

When tracking, we can then update the locations of the features in the current frame. That the locations of the features in the target image won't change, but some of the features may fail to track correctly. You can check for this by computing a homography between the current frame locations and the original target image locations. Tracked features will be inliers in this computation. If there are not many features being tracked you can re-initialise the tracker.

Some OpenCV methods you will find useful are:

- `perspectiveTransform` – Can be used to apply a homography to a `std::vector` of `cv::Point`s
- `goodFeaturesToTrack` – Finds corners to track using the Shi-Tomasi feature detector.
- `calcOpticalFlowPyrLK` – Matches features between consecutive frames using the Lucas-Kanade tracking method.

For the Assignment: Implement a KLT-based `Tracker` using SIFT or ORB to initialise the tracking, but Shi-Tomasi features and Lucas-Kanade tracking once an initial pose has been found. You should then conduct experiments to see whether this is faster than the matching-based tracking; what (if any) changes there are in the reliability of the tracking. You should also implement some method to detect tracking failure and re-initialise using SIFT Or ORB.

3 Deliverables and Marking Scheme

For the assignment you should deliver source code for your trackers and a report.

The code delivered should include the `.h` and `.cpp` source files, but not executables, intermediate files, etc. You should also provide any special instructions on how to build your code if this differs from the skeleton.

Your report should include:

- Instructions for running your code.
- A brief description of how each method works and any interesting implementation details ($\frac{1}{2}$ to 1 page per method).
- A description of how you evaluated the methods' speed and reliability. This would include the design of experiments, selection of test image sequences, etc. (2-3 pages)
- The results of these experiments comparing the brute-force SIFT tracker and the three you have implemented. (3-4 pages)
- Discussion and closing remarks reflecting on your results and providing any ideas for further enhancement. (1-2 pages)

Page counts are guidelines only, and include allowance for figures such as pictures from test sequences, illustrations of tracking results (good and bad), tables and plots of results, etc.

There are **20** marks available for this assignment. Most (**15**) will be allocated to the report, with an emphasis on the experimental design, presentation of the results, and your ability to discuss the results and draw conclusions from them. Your report should be clearly presented, and make appropriate use of sectioning, figures, mathematics, tables, and references.

The remaining **5** marks will be for your code. I will be looking for code that is *correct*, *clear*, and *concise*, in that order. While the assignment is about determining which methods are faster, there are no additional marks for improving the individual approaches. You should use the OpenCV implementations of algorithms where possible, as they are well-tested and optimised. The focus is on comparing the different approaches, not on developing better versions of them.

References

[1] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[2] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of Image Understanding Workshop*, pages 121–130, 1984.

[3] Maruius Muja and David Lowe. FLANN – fast library for approximate nearest neighbours user manual. Technical report, Department of Computer Science, University of British Columbia, 2009.

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, 2011.

[5] Jianbo Shi and Carlo Tomasi. Good features to track. In *International Conference on Computer VIsion and Pattern Recognition*, pages 593—600, 1994.