2D Feature Tracking Results Analysis

Ada Lazuli

2024-01-01

Code Updates and Implementation

Data Loading

Implementing an input buffer for holding loaded images was the first action item completed for the project. The buffer was implemented as a first in first out structure that holds a limit quantity of images, with that quantity being constrained by a variable set within the application.

When new images are load that would exceed the buffer's specified size, the oldest images are removed from the buffer. The benefit of this being that system resources for data storage are minimized in memory, leaving more resources for processing.

Detector Implementation

The second phase of the project was it implement a plethora of detectors for keypoint generation on the images loaded in the buffer.

The first detector implemented was **HARRIS**. HARRIS took the most work to implement and utilizes triple layer iteration, which could yield performance issues. Additionally, the HARRIS implementation relied several functions found within openCV 4.1

The remaining detectors, listed below, were implemented strictly using openCV:

1. FAST
2. BRISK
3. ORB
4. AKAZE
5. SIFT

Input Cropping

The third phase of was to crop the keypoints to a bounding box that contains a singular car. This step is important for later evaluation of detector performance and keypoint matching since the ground truth that the vehicle is the same throughout the images is known.

The cropping feature was implemented by iterating through through the generated keypoints and removing any keypoints that were outside of the bounding box.

Descriptor Implementation

The fourth phase was to implement several descriptors to process the keypoints. Using functions provided by openCV, the following descriptors were added to the project:

1. BRISK
2. BRIEF
3. ORB
4. FREAK
5. AKAZE
6. SIFT

Keypoint Matching

The fifth phase was to implement methods for matching processed keypoints between consecutive images. To accomplish this, FLANN and K-Nearest Neighbor matching was implemented in the project.

FLANN was implemented using openCV functions, however, there was a slight complication where varying data types needed to be controlled for. Depending on the descriptor used, there were moments when the data type changed from the expected 32 bit floating point value. This was controlled for by utilizing a conditional to test for the incorrect data type then convert to a float 32 using openCV functions.

Additionally, the K-Nearest Neighbor matcher was implemented by using a the `knnMatch` method of the openCV DescriptorMatcher class.

Filtering

The sixth and final update to the project was the implmentation of a match filter. The match filter was implemented by iterating over the matched keypoints and evalute the match distance. The match distance would be checked against a specified threshold and used for determining whether to keep or drop the match.

Performance Analysis

Detector Performance

The purpose of this section is to review the performance of the various detectors implemented in the project and draw conclusions around performance. The quantity of **keypoints** generated and **duration** of detection will be the only metrics considered in this section.

The first step is to load the data and overview the comma separated values (CSV) contents.

```r
# load the aggregated csv
df <- read.csv("detector_tests.csv")
glimpse(df)
```

```
## Rows: 60
## Columns: 3
## $ detector  <chr> "orb", "orb", "orb", "orb", "orb", "orb", "orb", "orb", "orb~
## $ keypoints <int> 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 5063, 4952~
## $ duration  <dbl> 20.360200, 3.467500, 14.660900, 6.151090, 5.922500, 6.365720~
```

The CSV has 60 rows, with three columns per row. The first column details the detector used, the second column records the number of key points generated by the detector for an image, and the third column captures the duration of time needed for keypoint generation.

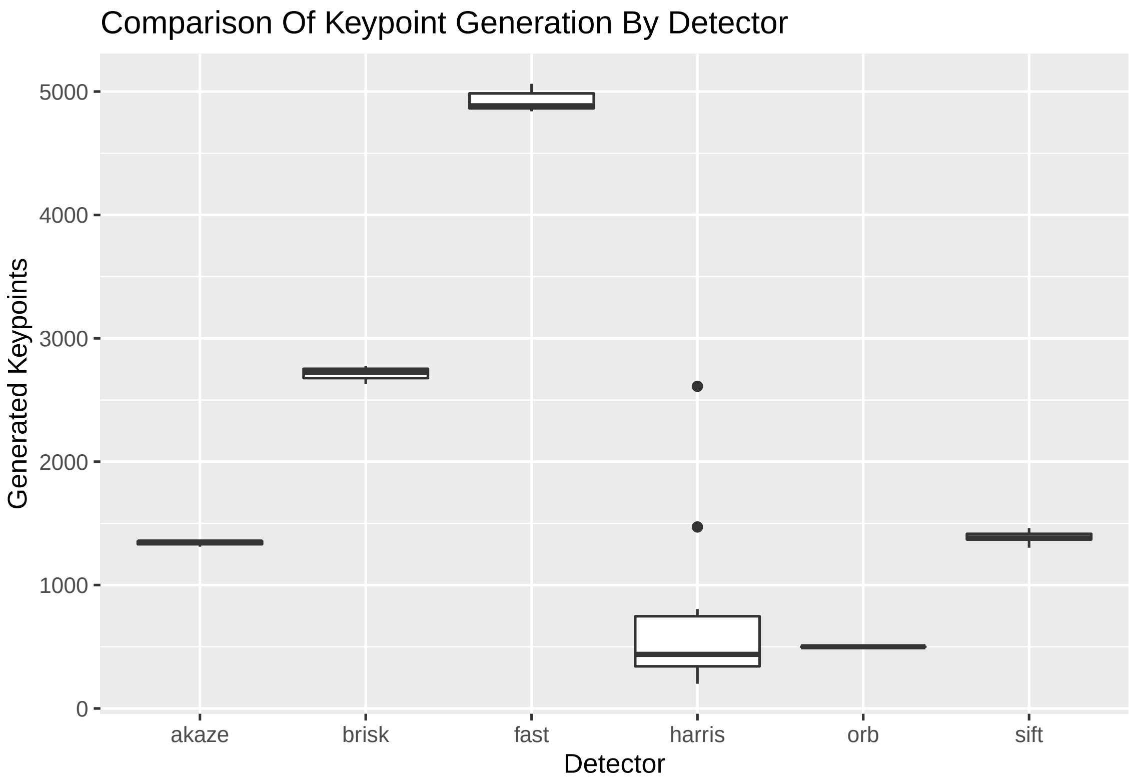
Additionally, the following code block shows that 6 detectors are uniformly represented in the CSV, with each having 10 associated observations.

```
df %>% count(detector)
```

```
##   detector  n
## 1    akaze 10
## 2    brisk 10
## 3     fast 10
## 4   harris 10
## 5      orb 10
## 6     sift 10
```

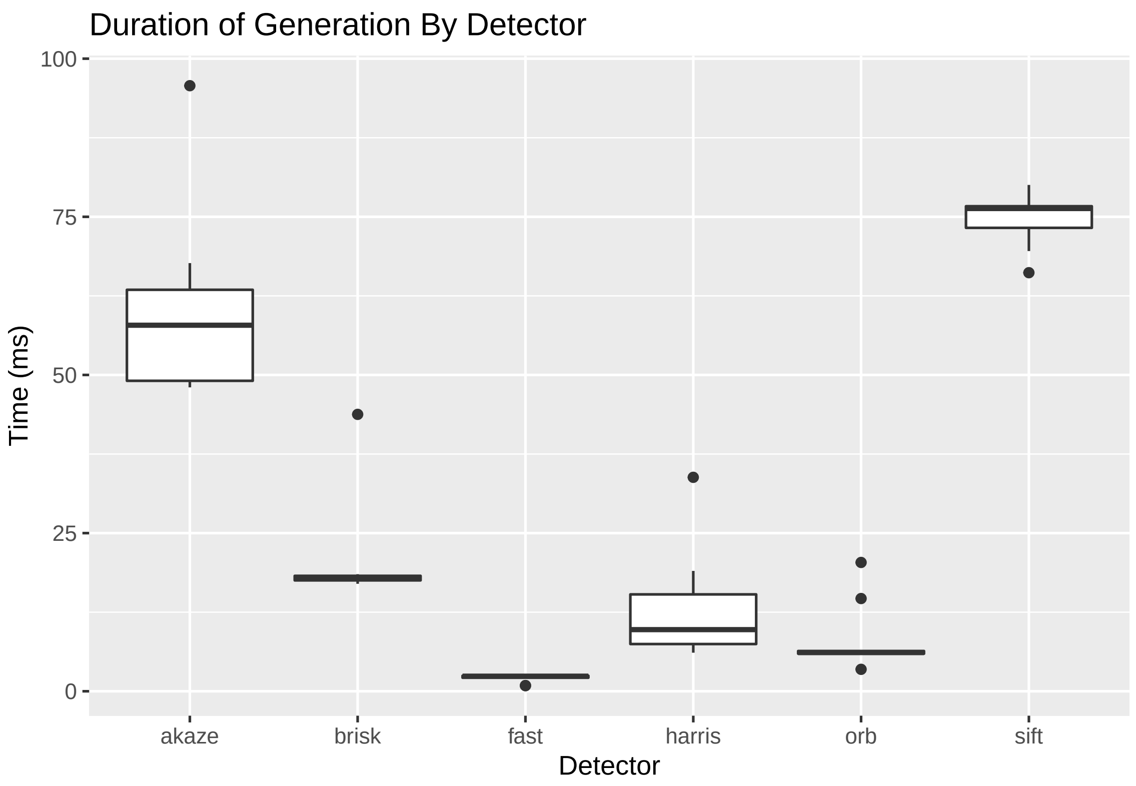
A box and whisker plot was used to review the distribution of keypoint generation by the various detectors for 10 images. The plot shows that most of the detectors were consistent, with *FAST* tending to generate the most keypoints. The *HARRIS* detector was implemented with some custom flow control and exhibited the highest variability.

```
ggplot(df, aes(x = detector, y = keypoints)) + geom_boxplot() + xlab("Detector") +
  ylab("Generated Keypoints") + ggtitle("Comparison Of Keypoint Generation By Detector")
```



Similarly, a box and whisker plot was used to review the distribution of the time used by the detectors for keypoint generation. The plot showed that the *AKAZE* detector varied the most, *SIFT* generally took the longest, and *FAST* was generally the fastest.

```r
ggplot(df, aes(x = detector, y = duration)) + geom_boxplot() + xlab("Detector") +
  ylab("Time (ms)") + ggtitle("Duration of Generation By Detector")
```



Detector Descriptor Performance

A second CSV was created during experimentation that contains details about various runs using different combinations of detectors and descriptors. Additionally, the keypoint matching algorithm and the images used were held constant throughout the experiments.

```r
df <- read.csv("detector_descriptor_experiments.csv")
glimpse(df)
```

```
## Rows: 315
## Columns: 6
## $ detector            <chr> "orb", "orb", "orb", "orb", "orb", "orb", "orb", "~
## $ descriptor          <chr> "brisk", "brisk", "brisk", "brisk", "brisk", "bris~
## $ keypoints           <int> 500, 500, 500, 500, 500, 500, 500, 500, 500, 4952,~
## $ detector.duration   <dbl> 3.529930, 7.709430, 3.157630, 2.886860, 2.900540, ~
## $ descriptor.duration <dbl> 0.580986, 0.570177, 0.611642, 0.615170, 0.664459, ~
## $ matches             <int> 82, 93, 95, 103, 101, 115, 119, 118, 116, 420, 431~
```

The CSV contains 315 rows with 6 columns. Each row captures the following details for each detector-descriptor pairing:

1. The **detector** used for keypoint generation
2. The **descriptor** used for keypoint description

3. The number of **keypoints** generated by the detector
4. The amount of **time** used by the detector
5. The amount of **time** used by the descriptor
6. The number of keypoint **matches** between two images

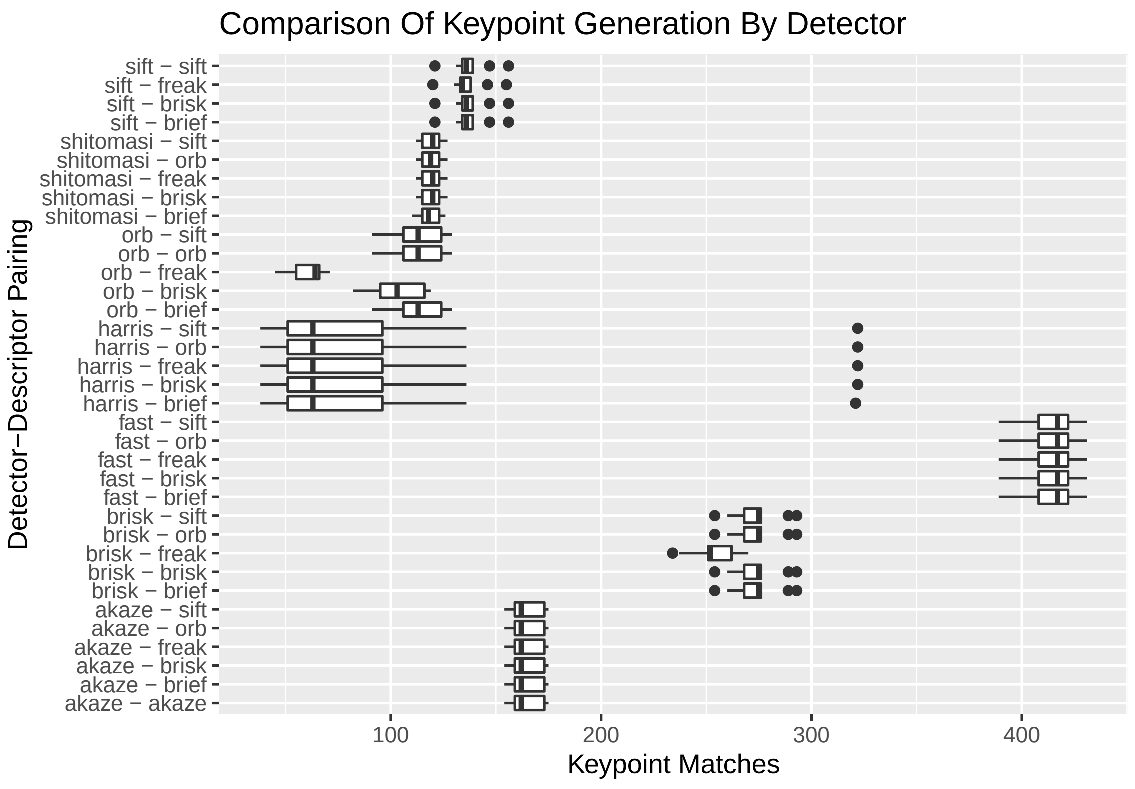
An important aspect to note is that the descriptors are not represented in the data equally. The *AKAZE* descriptor is only compatible with the *AKAZE* detector, hence it is represented the least in the data. Additionally, use of the *ORB* descriptor with the *SIFT* detector resulted in memory overflow errors and could not be recorded, leaving 9 less examples in the data compared to the other descriptors.

```
df %>% count(descriptor)
```

```
##   descriptor  n
## 1      akaze  9
## 2      brief 63
## 3      brisk 63
## 4      freak 63
## 5        orb 54
## 6       sift 63
```

The matches generated by each detector-descriptor pair were analyzed using a box and whisker plot. *HARRIS* and *FAST* had the highest variability and *FAST* had a tendency for the highest number of keypoint matches. Moreover, the plot showed that the detector played the largest role in performance, compared to the descriptor.

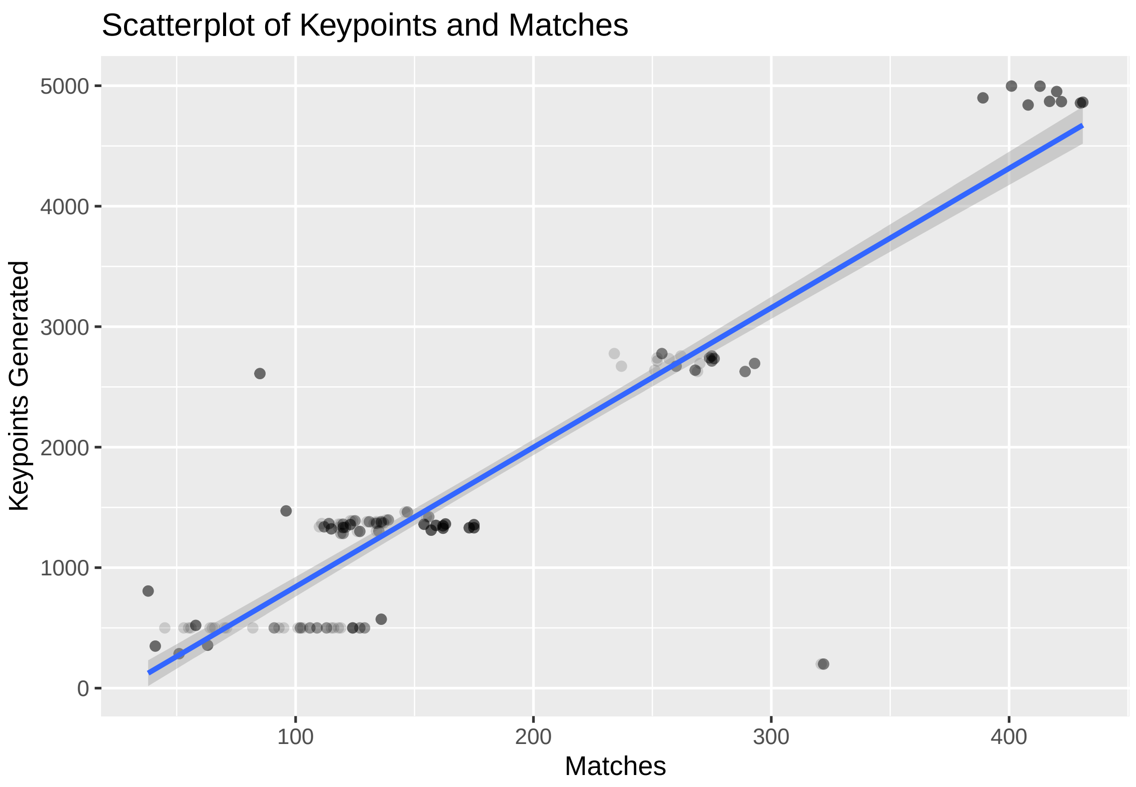
```
df <- df %>% mutate(pair = paste(detector, '-', descriptor))
ggplot(df, aes(x = pair, y = matches)) + geom_boxplot() +
  xlab("Detector-Descriptor Pairing") + ylab("Keypoint Matches") +
  ggtitle("Comparison Of Keypoint Generation By Detector") + coord_flip()
```



A scatter plot was used to further explore the relationship between the number of keypoints and the number of matches. The plot showed that the number of matches increased as the the number of keypoints increased. Therefore, detectors that generate more keypoints, such as *FAST* would naturally have more matches.

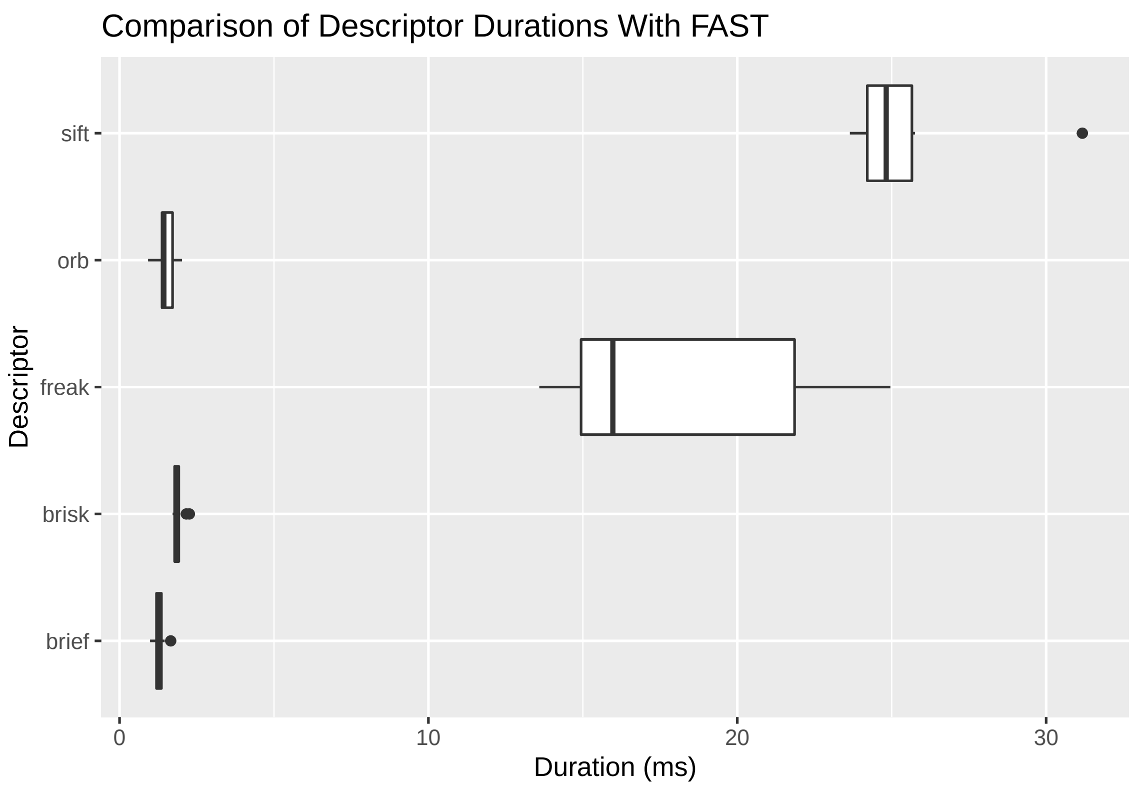
```
ggplot(df, aes(x = matches, y = keypoints)) + geom_point(alpha = 0.15) +
  xlab("Matches") + ylab("Keypoints Generated") +
  ggtitle("Scatterplot of Keypoints and Matches") + geom_smooth(method='lm')
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



Lastly, the duration for each descriptor was reviewed using a box and whisker plot. The plot showed that the *FREAK* descriptor had the highest variability, *SIFT* took the longest, and *BREIF* was the fastest.

```
ggplot(df %>% filter(detector == 'fast'), aes(x = descriptor, y = descriptor.duration)) +
  geom_boxplot() + xlab("Descriptor") + ylab("Duration (ms)") +
  ggtitle("Comparison of Descriptor Durations With FAST") + coord_flip()
```



Conclusion

After reviewing all of the data collected, the top three pairings of detector and descriptor would to be:

1. **FAST** with **BRIEF**.
2. **FAST** with **ORB**.
3. **FAST** with **BRISK**.

Overall, using the **FAST** with **BRIEF** is the best pairing of detector and descriptor. This combination results in the highest keypoint generation, matching, and shortest duration.