Focal Plane Visual Feature Generation and Matching on a Pixel Processor Array

Supplementary Material

A. Supplementary videos

We provide a video of feature generation and matching running on the real SCAMP camera. In the video, the left-hand display shows features which are generated from each frame, and matched against features of a stored "keyframe" shown on the right-hand display. The green squares shown in each image represent the keypoints extracted via the FAST algorithm, for each of which a binary descriptor is generated within the SCAMP's digital registers. The red lines then represent the matching of keypoint features between the frames, based on these descriptors. The SCAMP camera also outputs two 4-bit images, the latest frame and the stored keyframe, but this is only for visualization.

B. Technical details

B.1. FAST keypoint number control

The PPA FAST algorithm deploys an adaptive intensity difference (threshold) between the center pixel and its neighboring pixels to regulate the density of keypoints. In our method, when the number of keypoints exceeds the number of PPA rows, redundant keypoints are culled, and the threshold is decreased for stricter keypoint selection in the next frame. Conversely, if the detected keypoints are not enough, the threshold is increased for more keypoints' detection. This strategy enables the control of the number of keypoints within a target range.

B.2. BRIEF generation shifting optimization

In computing BRIEF descriptors on PPA (Algorithm 3) of Section 3.1, many pairs of points from the captured image need to be brought together into the same location for comparison. The image movement involved in this process thus needs to be as efficient as possible to keep the computational cost down. Two copies of the image are manipulated to bring these pairs together, where for each comparison a copy can be either (1) shifted, or (2) replaced with a copy of the original image (undoing any previous shifting) and then shifted. The total required number of shifts can be significantly reduced by correctly choosing which of these two options is used to perform shifting at every step as given by Algorithm 4. The order of movement for point pairs is determined by a greedy policy based on *seven state changes* shown in Figure 11. This planning of image movement is optimized on PC in advance, generating *code* that can be run upon PPA.

On PC, a series of point pairs for the descriptors sampling pattern is randomly selected from the normal distribution.

Algorithm 3 BRIEF: point comparison

Require: (1) Input image frame: $Image$;
(2) Pixel offsets for the descriptor bit x_1, y_1, x_2, y_2;
(3) Keypoints mask Key;
(4) Analog registers B, C, D;

1: $B \leftarrow$ shift $Image$ by (x_1, y_1)
2: $C \leftarrow$ shift $Image$ by (x_2, y_2)
3: $D \leftarrow B$ - C
4: $Result \leftarrow 0$
5: **where** $(D > 0)$
6: $\quad Result \leftarrow Key$
7: **end where**

Output: $Result$

The order in which these point pairs are compared, along with the shifting involved, is determined as follows. First the point pair with the smallest number of shifting steps is selected. Next the number of shifting steps involved in going from this selected point pair to each remaining point pair is calculated using the *seven state changes* policy. The point pair of minimal steps then becomes the newly selected pair, and this process is repeated iteratively until no point pairs remain, all the while recording the sequence in which pairs were selected. In order to take account of the noise buildup when shifting analogue data, a limit on the number of shifting steps is applied, after which the shifted register's content is replaced by the original image. In addition, this process can also be used to generate optimized PPA code for any other similar binary descriptors.

B.3. Details of On-Sensor Descriptor Movement

First, this section introduces the Off-sensor row planner. Next, we illustrate two essential masks essential for descriptor movement, transmission channel $R9$ and destination $R10$. We also explain the operation of the *scamp5_flood* function, which is used in moving the descriptors of different keypoints to their assigned rows, 1-bit at a time.

B.3.1. Off-sensor Row Planner

In section 3.2, each keypoint is assigned one row of PEs for the storage of descriptors. After reading out the keypoint positions from the sensors, Algorithm 5(Off-sensor row planner) is executed on the micro-controller to determine the unique row for each keypoint.

B.3.2. Rules of scamp5_flood

The flooding of SCAMP is performed using two digital registers, referred to as R_{source} and R_{mask}. Any "1s" in the

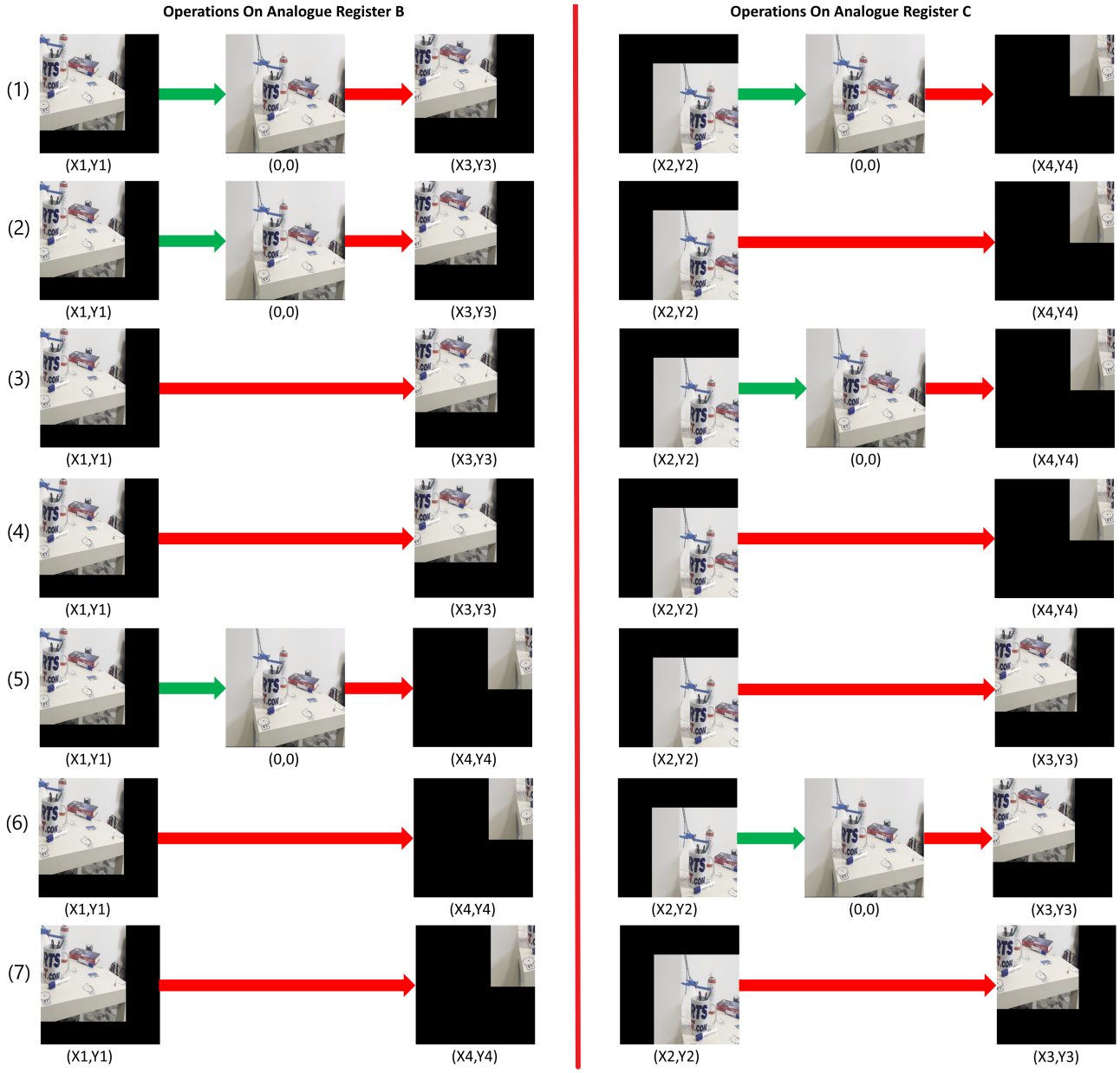

Figure 11. *Seven state changes* involves two point pairs among three analogue registers A, B, C, each change depicting a district movement pattern of B and C. The shifting steps of each type are different from each other. The coordinates $(X1, Y1)$, $(X2, Y2)$, $(X3, Y3)$, and $(X4, Y4)$ represent the positions of the image center within the corresponding analogue registers. Analogue register A holds the original image data, centered at $(0, 0)$. Green arrows show an overwrite operation, where image data in the corresponding register is covered with that from register A. Red arrows represent a direct *shift* operation, where data in the current register is directly shifted to a target location (X, Y).

R_{source} register will spread outwards, flipping neighboring "0s", flooding their surroundings. However, this spreading is restricted to only where R_{mask} contains "1s", allowing flooding to be controlled to specific areas.

B.3.3. Transfer 1-bit descriptor to the assigned row

In Figure 12, 1-bit of a keypoint's descriptor is transferred vertically to its assigned row in two steps.

1. First R_{mask} is setup to bridge the gap between the descriptor's current row, and its assigned row. Then *scamp5_flood* is called using the descriptor's content as

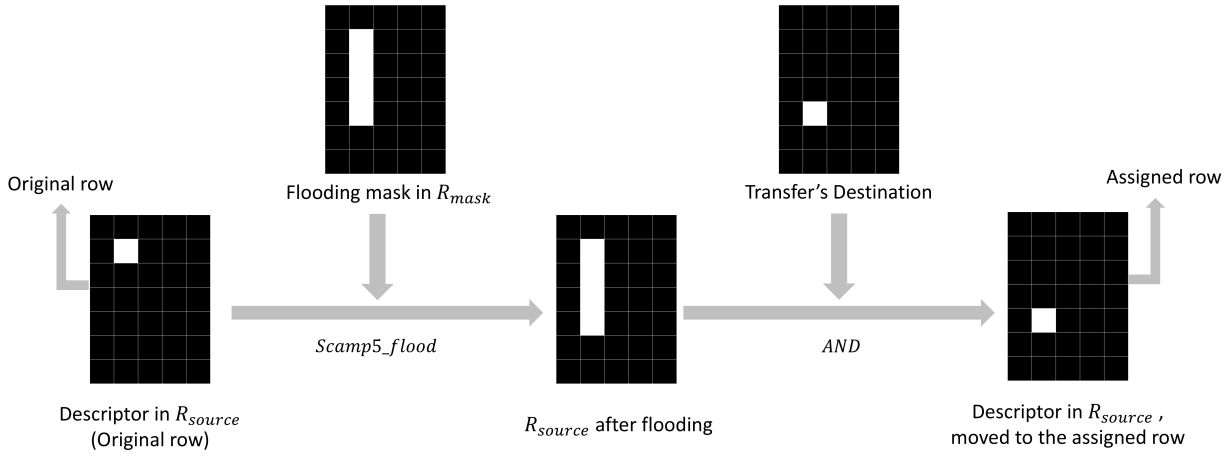

Figure 12. Transmitting 1-bit descriptor of one keypoint to the assigned row.

Algorithm 4 Optimization of BRIEF movement for PPA

Require: Random point pairs from $sampling\ pattern$;
Initialize $Cache$, $step\_limit$;
Select point pair with minimal shifting steps as Ps and put other point pairs into $Cache$;
while $Cache \neq$ Null **do**
  for Pc in $Cache$ **do**
    $seven\_state\_change(Ps,Pc)$;
    Update $step\_min$, $State$, $best\_pair$;
  end for
  if $step\_limit$ + $step\_min \leq$ threshold **then**
    add $State$ to $output\_cache$;
    $Ps = best\_pair$;
  else
    Search Ps with minimal shifting steps in $Cache$;
    Add $State$ of Ps to $output\_cache$;
  end if
  Remove Ps from $Cache$;
  Update $step\_limit$;
end while
Output: $output\_cache$

Algorithm 5 Off-sensor row planner

Require: Keypoint position key_pos;
Initialize $row\_occupied$;
for $point$ in key_pos **do**
  if $point$ not in $row\_occupied$ **then**
    Add $point$ to $Planner$ and $row\_occupied$;
  else
    Search $correct\_position$ for storage;
    if $found$ **then**
      Add $correct\_position$ and $point$ to $Planner$ and $row\_occupied$;
    end if
  end if
end for
Output: $Planner$

R_{source}. If the descriptor bit was a "1", it will be flooded vertically into the assigned row.

2. Next an AND operation is executed between R_{source} and digital register containing descriptor destination, containing "1" for the descriptor's assigned row, within the descriptor's column.

B.3.4. Transmission channel $R9$ and Destination $R10$

The Off-sensor row planner in Algorithm 5 assigns each keypoint a target row it should be moved to. Digital register $R9$ contains the flooding mask used to move the descriptors of multiple keypoints at once, with an area for each keypoint starting from its position, and connecting to the row where the planner schedules. An example of this is shown for a single keypoint by R_{mask} in Figure 12. The digital register $R10$ contains the destination for each keypoint's descriptor (The transfer's destination in Figure 12 is one of them). Using *scamp5_flood* and the content of $R9$ and $R10$, descriptors of different keypoints can be transferred to their assigned rows in parallel, 1-bit at a time.

B.3.5. Flooding Mask Constraints

A 1-bit descriptor moves to its destination using a *flooding mask*, as illustrated in Figure 12. To ensure the correctness in descriptor propagation, the following constraints are imposed on the selection of *flooding mask* for different keypoints on the same column shown in Figure 13.

1. Keypoints within the same column should be assigned non-adjacent *flooding masks*.
2. The *flooding masks* of keypoints in the same column

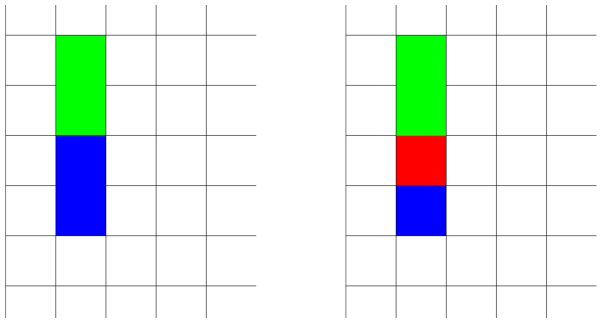

Figure 13. Two constraints on *flooding mask*. Green and Blue regions represent the *flooding mask* of two keypoints located in the same column. The red region denotes the overlapping area between these two masks.

Algorithm 6 MultiSandCastleSum

Require: Register R7; Descriptor length $descriptor\_len$;

$CLR(RN, RS)$;
$scamp5\_load\_pattern$ in register $SELECT$ and $R10$ with the pattern (0, -1, 255, 256-$descriptor\_len$);
$shift$ $R10$ one pixel in $west$ direction and reverse the value of $R10$;
for $descriptor\_len$ in loop **do**
  $RW = R7$; $RE = NOT(R7)$;
  $//falling\ down\ one\ pixel\ from\ R7\ to\ R8$ $DNEWS0(R8, R7)$;
  $//mask\ the\ error\ during\ falling\ down$ $IMP(R9, R7, R8)$;
  Update $R7$ via register $SELECT$ and $R8$;
  Create mask of error in $R8$ based on $R9$ and $R10$;
  Eliminate the error $R8$ from $R7$;
end for

Output: Digital register $R7$;

should not overlap with each other.

The selection of *correct position* for keypoints in Algorithm 5 also follows these rules. If a valid position satisfying both conditions cannot be found for a given keypoint, that keypoint is simply removed from the digital register of FAST keypoints for computational efficiency. Optionally, Algorithm 5 and 1 can be executed again for these unassigned keypoints as not losing any of them.

B.4. Details of Analogue bit summation

In section 4.3, we illustrate how to find the best descriptor matching upon analogue computation in short. This method currently supports descriptors of up to 128 bits in length. Specifically, for a 128-bit descriptor, the process is divided into four key stages, as shown in Figure 14.

- **Digital to Analogue:** Convert the digital matching result into the corresponding analogue form.
- **Regional Analogue Diffusion:** The matching is then split into two adjacent regions, each consisting of 64 bits. Each region performs **Analogue Diffusion** row by row simultaneously.
- **Average the score:** The final matching score is computed by averaging the values from the corresponding columns of the two adjacent regions.
- **Thresholding:** The best match is determined by applying a thresholding process. The initial matching threshold is set to 80% of the descriptor length. If multiple matches exceed this threshold, the threshold is increased to 90% (i.e., 80%+10%, where 10% is an offset value) for a second round of matching. If no matches are found, the threshold is decreased to 70% (i.e., 80%-10%). The offset value is halved in each subsequent iteration for further searching.

Analogue Diffusion. This method can uniformly diffuse the analogue signals of a specified region. Unlike the incremental stacking process of *MultiSandCastleSum* (Algorithm 6, which progressively places "1" pixels, Analogue Diffusion operates similarly to the release of water through an open sluice gate, enabling faster signal propagation. In addition, any pixel in the matching region can represent the matching score due to the uniform diffusion of signals in this region.

C. Additional Experiments and Details

As mentioned in section 5, we also consider the FAST algorithm's influence in the following test. Our system is evaluated based on two situations, and the keypoints are extracted from all images via the FAST algorithm.

- Performing our system without any tests. The accuracy equals the number of correct matchings divided by the number of key points joining the matching process.
- Performing our system with algorithm *Cross Check*. The accuracy equals the number of correct matchings divided by the number of key points filtered by *Cross Check*.

Figure 15, 16 show that the accuracy of matching largely increases with the algorithms *Cross Check* since they filter a lot of erroneous matchings. In addition, the analogue bit summation method has a similar performance to the digital one when using the *Cross Check* method.

- FPS Calculation: The total processing time per frame includes the components: FAST Keypoint extraction, feature management (descriptor generation and storage), digital/analogue feature matching, and matches readout. Using the example with 50 keypoints and digital feature matching, the processing time breakdown as follows: 500 µs for FAST keypoint extraction, 705 µs for descriptor generation, 828 µs for descriptor storage, 53×50 µs for digital feature matching, and 100µs for matches readout, resulting in a frame rate of 209 FPS.

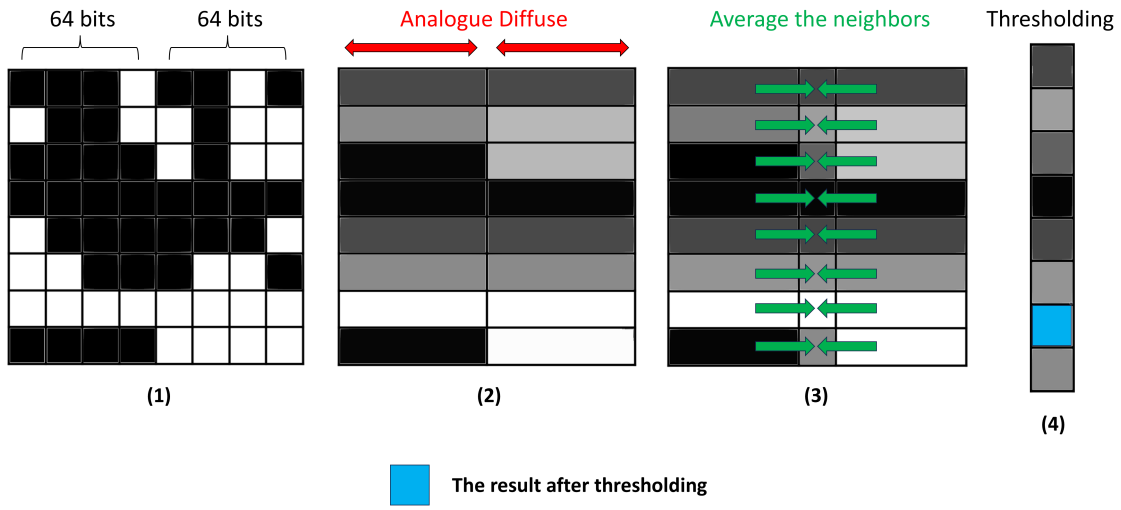

Figure 14. Four steps of Analogue bit summation on 128-bit descriptor. (1) The analogue matching result is converted from the digital result. (2) Split the 128-bit matching result into two regions and diffuse the result in each region simultaneously. (3) Average the values from these regions. (4) Get the result after thresholding.

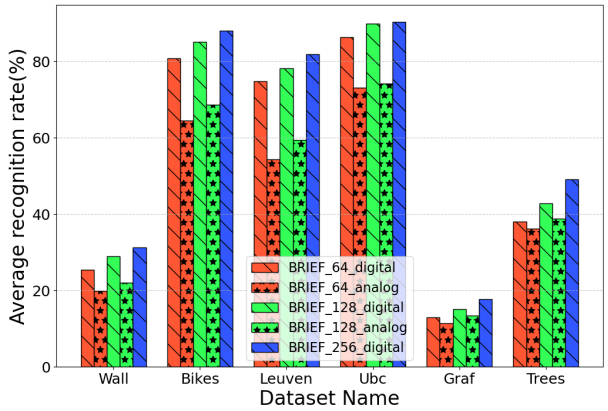

Figure 15. Recognition rates on six image Sequences without any tests. 64, 128, and 256 in the BRIEF descriptors' names are the length in bits. digital/analogue in the name represents the digital/analogue bit summation methods separately.

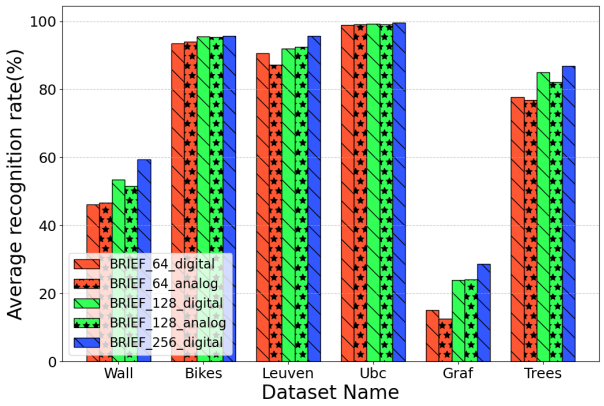

Figure 16. Recognition rates on six image Sequences with cross-check test. 64, 128, and 256 in the BRIEF descriptors' names are the length in bits. digital/analogue in the name represents the digital/analogue bit summation methods separately.

- Hermitian matrices: The Hermitian matrices in our method refer to homography matrices provided by the dataset, used to project points from an input image to different reference frames.
- Seq2Seq: This method performs sequence-based matching against sequences in the database. This method is more robust as occasional frame-level mismatches have a small impact on re-localization.
- The performance of [7] on the Oxford Robotic Car Dataset: The previous PPA work[7], which utilized an earlier generation PPA (SCAMP-5), reported re-localisation accuracy of 64.95% using a 32-bit binary descriptor per image at 315 FPS, and 80.4% using low-resolution images at 386 FPS.

Dataset	Descriptor type	Accuray	Time
Oxford Robotic Car	Binary descriptor	64.95%	3.17ms
	Low-resolution images	80.4%	2.59ms

Table 6. Peformance of [7] on Oxford Robotic Car Dataset.