Camera Tracking on Focal-Plane Sensor-Processor Arrays

Thomas Debrunner*[1], Sajad Saeedi*[1], Laurie Bose[2],
Andrew J Davison[1], and Paul H J Kelly[1]

[1] Imperial College London, UK
[2] University of Bristol, UK

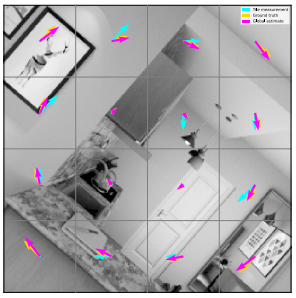
Abstract. Focal-Plane Sensor-Processor (FPSP) Arrays are new sensors with parallel image capturing and processing capabilities built into one device. This design eliminates the need for image transfer and thus increases frame rate significantly. Additionally, recent FPSP which are based on analogue technology, consume much less power compared to conventional digital cameras. However, implementing well-known pose estimation and camera tracking algorithms on FPSP is a challenging problem because the processing elements on FPSP have very limited resources. FPSP also require a different programming paradigm. * This paper presents several contributions to camera tracking on FPSP, using limited instruction sets and memory available for each pixel. We demonstrate that FPSP are able to estimate camera pose at very high-frame rates with low power consumption and high accuracy. Simulated and real-world experiments demonstrate the effectiveness of the proposed methods.

Keywords: Focal-Plane Sensor-Processor (FPSP) Arrays, Visual Odometry

1 Introduction

In modern image processing systems, an image is captured by the camera sensor first, then it is transfered to the main memory to be processed by a CPU, GPU or some other processing device. Data transfer incurs delays in transmission and also consumes power. At higher frame rates, the data transmission becomes a serious bottleneck. For instance to transfer an 8 bit, 256×256 image at 10,000 fps, a bus capable of transmitting approximately 650 MB/s is required [1]. Applications such as Simultaneous Localization and Mapping (SLAM), Convolutional Neural Networks (CNN), and Virtual Reality (VR) are examples that benefit not only from high frame rate, but also could save resources if they consume less energy. Dynamic Vision Sensors (DVS)s, also known as event cameras, approach this problem by only reporting pixel values that changed in intensity to the host system [2], [3]. Cellular Vision Chips, such as the ACE400 [4], ACE16K [5], MIPA4K [6], and the various revisions of the SCAMP chip [7], [8], [9], also known as Focal-Plane Sensor-Processor (FPSP) Arrays, introduce the concept of 'sensing and processing in the focal plane'. FPSP not only increase frame rate but also have the ability to reduce power consumption. However FPSP such as the recent SCAMP-5 chip [9], which is the focus of this paper, have very limited instruction sets and local memory.

*These authors contributed equally.



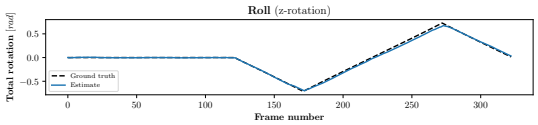


Fig. 1: On Focal-Plane Sensor-Processor (FPSP) Arrays, image capturing and processing is done on the focal plane chip. (left) This figure demonstrates a screen-shot from the Apron simulator of the tiling-based camera tracking algorithm. (right) This figure shows the ground truth, measured tile vectors, and estimated motion of the camera. The proposed camera tracking algorithm can run at 700 fps with 1.23 W of power consumption.

Therefore developing new applications for FPSP is generally a challenging problem, both algorithmically and in the implementation.

In this paper, we propose a novel camera tracking algorithm which runs on FPSP. We present a keyframing method to store images in image-plane. Then a 2DoF tracking algorithm is explained that estimates yaw and pitch motions of the camera. Based on the 2DoF algorithm, we build a 4DoF algorithm which estimates full orientation and translation along the principal axis of the camera. The proposed camera tracking algorithm opens up the research path to use FPSP in robotics and applications which require high frame rate and low power consumption, such as SLAM and VR. Fig. 1 demonstrates a screen-shot of the algorithm developed for camera tracking. On the left, the vectors demonstrate the ground truth (yellow), the measured tile vectors (green), as well as the estimated motion vectors (magenta) for each of the small tiles. The aggregated results from the tiles are used to estimate the orientation of the camera. The estimated angle with its ground truth are on the right. Our algorithm is able to recover the 4DoF pose of the camera accurately at very high frame rates, while consuming only (an estimated) 1.23 W of power.

1.1 Literature Review

This section briefly introduces FPSP and Visual Odometry algorithms.

Focal-Plane Sensor-Processor (FPSP) Arrays Programmable Focal-Plane processors have been around for a long time. While there was theoretical interest in the topic, real implementations of processors on image sensors did not appear until the mid 1990s. For example, the ACE400 [4] is a CMOS chip capable of storing and processing four binary images. It is a 20×22 processor array modeled after a Cellular Neural Network (CNN) Universal Machine [10].

Interesting applications using the ACE400 chip were shown [11]. One application was the detection and classification of simple objects printed on a rotating ring at 10,000 fps. The other application involved the analysis of individual sparks from a car spark plug. The unconventionally high frame rate allowed them to capture around seven frames per spark to properly analyze its properties.

A different approach was taken in [12], first describing a single, analogue, sampled current microprocessor, leading to the SCAMP vision chip [7]. Unlike the CNN based chips [10], which use parallel, hard wired convolution templates for most operations,

the SCAMP consists of a grid of analogue general-purpose processors operating in a single instruction multiple data (SIMD) manner. Despite having speed disadvantages, Dudek argues that the sequential SIMD processor is a better practical choice, due to smaller circuit area and better achievable accuracy [13].

Multiple versions of the SCAMP chip have been implemented, starting from a very small 21×21 prototype [14] to the more recent 256×256 array [9]. All implementations share a very similar architecture. Our work is developed for the latest version of the chip, the SCAMP-5. Carey et al. describe SCAMP-5 as consisting of a square array of 256×256 analogue processing units (PE) along with control and readout circuitry to drive the chip. On this chip, a single instruction stream is distributed to all processing elements simultaneously. This means that all processing elements execute the same instruction on their local data simultaneously. The algorithm that is performed on the system is thus defined by a sequence of instruction words, issued by the system controller [9].

Visual Odometry Nister et al. coined the term Visual Odometry (VO), as an analogy to wheel odometry. VO is the method of estimating a camera's location and pose solely from analyzing the video feed and a known initial position and orientation. Pioneering results were shown by Nister et al. utilizing a Random Sample Consensus (RANSAC, [15]) approach to filter out outliers [16]. More recent approaches in feature tracking based algorithms and RANSAC are presented in the works by [17], implementing a VO system for the Mars Exploration Rover. Another contribution showed an implementation of a purely incremental monocular VO system [18]. A similar approach as [19] is followed in this paper. Another appearance-based approach was presented in [20] utilizing the Fourier Mellin transformation for a global full frame comparison, which was shown to be more accurate and also computationally more efficient than optic flow methods. Yet another approach was followed with [21], where an approach based on sparse optical flow is presented. More recently, [22] showed a real-time VO solution that also incorporates depth information from RGB-D sensors. Event cameras, as alternative capturing devices, have also been used for tracking [23], optical flow [24], corner detection [25], pose estimation [2], [26], [27], [28], [29], and also multi-sensor pose estimation [30], [31]. The only other visual odeomtery paper on FPSP is the one proposed by Bose et. al [32]. It is a 4DoF odometry algorithm, that uses image gradient to estimate rotations and translation along optical axis.

In this paper, we adopt a tile-based approach to estimate the motion and the rigid-body motion. This idea has been used for 3DoF motion estimation with event cameras by Conradt [26]; however, our approach is 4DOF and also it has been applied to FPSP.

The rest of the paper is organised as follows. Sec. 2 presents the proposed pose estimation method. Sec. 3 shows the experimental results, and 4 summarizes the paper.

2 Proposed Method

This section presents the contributions of this paper. First an approach for storing grayscale images upon the digital registers of the FPSP is presented. Both 2DoF and 4DoF tracking algorithm are then introduced, making use of this grayscale image storage for storing a keyframe to which the current image is compared to in order to deduce camera motion. In this paper, we use left-handed coordinate system, where the z-axis

Algorithm 1 Grayscale image storage

```
1: procedure AnalogTo4Bit(analog_img, R1, R2, R3, R4)
2:     CLEAR(R1, R2, R3, R4)                          ▷ Clear digital register content
3:
4:     mov(A, analog_img)                             ▷ Create of copy of the desired analog image
5:     where(A)                                       ▷ Select pixels for which register A > 0
6:         SET(R4)                                    ▷ Set the 4^th bit in selected pixels
7:     all()                                          ▷ Select all pixels
8:
9:     mov(A, analog_img)
10:    add(A, 128)                                    ▷ Increase the value of A by 128 in all pixels
11:    WHERE(R4)                                      ▷ Select pixels for which 4^th bit is set
12:        sub(A, 128)                                ▷ Decrease the value of A by 127 in selected pixels
13:    all()
14:    sub(A, 64)                                     ▷ Decrease the value of A by 64 in all pixels
15:    where(A)
16:        SET(R3)                                    ▷ Set the 3^rd bit in selected pixels
17:    all()
18:
19:    mov(A, analog_img)
20:    add(A, 128)
21:    WHERE(R4)
22:        sub(A, 128)
23:    WHERE(R3)
24:        sub(A, 64)
25:    all()
26:    sub(A, 32)
27:    where(A)
28:        SET(R2)                                    ▷ Set the 2^nd bit in selected pixels
29:    all()
30:
31:    mov(A, analog_img)
32:    add(A, 128)
33:    WHERE(R4)
34:        sub(A, 128)
35:    WHERE(R3)
36:        sub(A, 64)
37:    WHERE(R2)
38:        sub(A, 32)
39:    all()
40:    sub(A, 16)
41:    where(A)
42:        SET(R1)                                    ▷ Set the 1^st bit in selected pixels
43:    all()
44: return R1, R2, R3, R4
```

is along the principle axis of the camera, x points to right, and y points up. Rotations along x, y, z axes are pitch, yaw, and roll. We use the order x-y-z for Euler angles.

2.1 Grayscale Image Storage on FPSP

The SCAMP-5 vision chip is a grid of 256×256, each cell of the grid with a processing element with 7 analog registers (AReg) and 13 1-bit digital registers (DReg). Thus, 7 different grayscale and 13 binary images can be stored and processed in-plane [9]. Content stored within ARegs decays over the course of a few seconds making them unsuitable for storing captured grayscale images. Instead such image storage must use the DRegs. A single digital register "plane" can store a binary image at full 256×256 resolution. By combining N such binary images, an N bit approximation of a grayscale

Algorithm 2 Grayscale image retrieval

```
1: procedure 4BitToAnalog(analog_img, R1, R2, R3, R4)
2:     mov(analog_img, −128)                          ▷ Move -128 into analog_img in all pixels
3:
4:     R0 = AND(R1, R2, R3, R4)
5:     where(R0)          ▷ Select pixels storing value 16 (i.e. where 1st,2nd,3rd and 4th bits are set)
6:         mov(analog_img, 128)          ▷ Move the value of 128 into analog_img for selected pixels
7:     all()
8:
9:     R0 = AND(NOT(R1), R2, R3, R4)
10:    where(R0)                                      ▷ Select pixels storing value 15
11:        mov(analog_img, 112)          ▷ Move the value of 112 into analog_img for selected pixels
12:    all()
13:
14:    R0 = AND(R1, NOT(R2), R3, R4)
15:    where(R0)                                      ▷ Select pixels storing value 14
16:        mov(analog_img, 92)
17:    all()
18:    ...
19:
20:    ...
21:    R0 = AND(R1, NOT(R2), NOT(R3), NOT(R4))
22:    where(R0)                                      ▷ Select pixels storing value 1
23:        mov(analog_img, 16)
24:    all()
25: return analog_img
```

image can be stored across N digital register planes. In this work we use four digital registers within each pixel for storing a 4-bit grayscale image used as a keyframe by the odometry algorithm. This involves both the process of extracting a 4-bit digital representation from an analog grayscale image, and the process of reconstructing an analog grayscale image from the data stored in the four digital registers.

Algorithm 1 is used to convert a captured image stored in an analog register plane, to a 4bit digital image. The $R1$, $R2$, $R3$ and $R4$ digital registers of each pixel are used to store the converted image, each storing a single bit of the 4 bit value, with the most significant bit being stored in $R4$ through to the least significant in $R1$. Accordingly Algorithm 2 outlines the process used to reconstruct a grayscale image within a given analogue register from the data stored across the four digital register planes $R1, R2, R3, R4$. Fig. 2 illustrates this conversion process, comparing the original captured analogue image to its reconstructed 4 bit approximation along with the content of the $R1$, $R2$, $R3$ and $R4$ registers used as storage.

2.2 2DoF Camera Tracking

The 2DoF camera tracking algorithm is a gradient-based algorithm similar to [32]. The 4DoF algorithm builds on top of 2DoF algorithm. The algorithm estimates pitch and

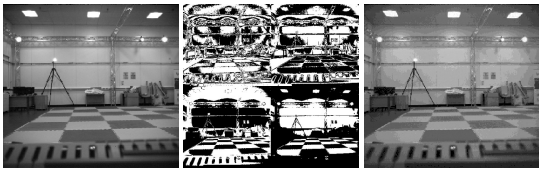


Fig. 2: Left: captured grayscale image. Middle: Four digital registers storing the 4bit representation of the original grayscale image. Right: Grayscale image reconstructed from the data stored within the four digital registers.

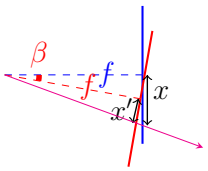


Fig. 3: Apparent position of a pixel after rotation. The blue and red thick lines represent the image plane before and after a rotation by β. The magenta line is the vector pointing to a real world location. The values x and x' are the distances from the image center (in x-direction) where we expect to find the real world location in both images.

yaw motions. Since we rule out translation, a pitch or yaw rotation should result in the same apparent movement for all visible objects on the image plane. The idea of the algorithm is to store the last keyframe in a register on the FPSP, and to capture a new one. A new keyframe is generated when the total some of the pixel values are more than a threshold. We assume that if the camera did rotate by a small angle between the two frames, the new frame should essentially be equivalent to the old frame but shifted some pixels into a certain direction. Generally, for an arbitrary yaw or pitch angles, not all pixels in the new frame are shifted by the same offset. The images are transformed by a homography which does in general not respect lengths and angles in the image plane; however, this assumption can be assumed to be true for FPSP since they can capture images at very high frame rates, and thus the motion between consecutive images are small. Computing the sum of absolute difference between a shifted version of the new frame and the old frame gives us a measure on how good the matching of the two frames is. Finding motion between the frames is therefore equivalent to finding a shift offset that gives an alignment with the lowest sum of absolute difference value. We assume that any motion of the camera between two frames is purely yaw and pitch motion, according to the camera's coordinate system.

Let $I^{(t)}(x, y)$ be the pixel value of a stored image at position x, y at time t, f be the focal length of the camera and let the camera be free of distortions. Assume a camera rotation of β radians (yaw) and α radians (pitch) between two consecutive frames.

Fig. 3 shows the situation in which we rotate the camera around the $y - axis$ (yaw). A point seen at a distance of x (along the x-axis). After the transformation, we find the same point at x' in the new image. The relation between the two points is:

$$x' = f \cdot \tan(\arctan \frac{x}{f} - \beta), \tag{1}$$

which can be reformulated to

$$x' = f \cdot \frac{\frac{x}{f} - \tan(\beta)}{1 + \tan(\beta)\frac{x}{f}}. \tag{2}$$

Assuming small β, and using approximation $\tan(\beta) \approx \beta$, $\tan(\beta)\frac{x}{f} \approx 0$, Eq. (2) is:

$$x' \approx x - \beta \cdot f. \tag{3}$$

Eq. (3) shows that a point previously found at pixel location (x, y) should now have moved to location $(x - \beta f, y - \alpha f)$. If we further assume similar lighting conditions for both frames at time t and $t + 1$, we can say that for the image intensity I at time t:

$$\left| I^{(t)}(x, y) - I^{(t+1)}(x - \beta f, y - \alpha f) \right| \approx 0 \tag{4}$$

The sum of absolute differences is defined as:

$$\mathrm{SAD}(u,v) = \sum_x \sum_y \left| I^{(t)}(x,y) - I^{(t+1)}(x+u, y+v) \right| \tag{5}$$

Finding the camera rotation between two consecutive frames thus becomes a search problem with the goal to minimize the cost function $\mathrm{SAD}(u,v)$.

$$(\alpha, \beta) = \frac{1}{f} \cdot \underset{u,v}{\mathrm{argmin}}(\mathrm{SAD}(u,v)) \tag{6}$$

With a given frequency r, we can estimate the angular velocities as $\omega_x = r \cdot \alpha$ and $\omega_y = r \cdot \beta$. The angles can then be determined using rotation matrices.

To do the optimization in Eq (6), we perform the search for the best alignment by using a gradient descent inspired approach. We starts out with $u = 0$ and $v = 0$, assuming there was no movement between the frames. The sum of absolute difference for no movement gets stored as the initial best value. From there, we shift the current frame one pixel in all four directions, computing the sums of absolute differences for each direction. Note that shift operations in any direction are a very cheap operation on FPSP. We keep the minimum value obtained as well as the direction we took to get there. This identifies the direction to shift the image in the next iteration, assuming we always go into the direction which leads to the lowest immediate value. After identifying the direction, we add the direction to the result variables u or v and shift the current frame in the obtained direction. We start the next iteration of the process. Again, we assume no further movement between the current frame and the last frame. If we detect a direction in which we can get an even lower minimal sum, we go into this direction and start the next iteration. The process ends when we identified a local minima, which manifests itself that we get higher sums of absolute differences in all four directions. we assume the problem to be locally convex. The process reports the (scaled) value of the angular velocities back to the host CPU.

2.3 4DoF Camera Tracking

This section introduces a novel method to extend the 2DoF approach presented in Section 2.2 with two additional degrees of freedom; roll and z-translation. The algorithm is based on splitting the image into tiles, and estimating a displacement vector for each tile. The global movement is then estimated by a regression on the measured vectors to base vectors, which are explained next.

Approach The approach chosen for this problem splits the image into N square tiles. The SCAMP system allows to address pixels efficiently when the number of the square tiles are powers of four. Experimentally, $N = 16$ tiles perform better than 4 or 64 tiles. The algorithm estimates the displacement vector for each tile according to the gradient descent based algorithm introduced in Section 2.2. For each of the four degrees of freedom, we have an expectation of how the individual vectors should be aligned, if a motion in this dimension occurs. This vectors are called base vectors. Fig. 4 shows the base vectors for each tile, for each motion. The algorithm then uses a statistical

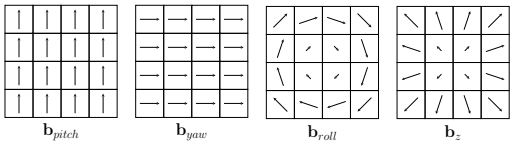


Fig. 4: Base vectors for each tile given a certain mode of motion between two frames. Any legal motion the camera experiences between two frames is assumed to appear as a linear combination of these vectors, according to Eq. (12).

method, to match the 16 measured displacement vectors to a linear combination of the base vectors. Unlike in Section 2.2, we can not expect the same approximate apparent movements of pixels in the whole image plane. Instead, this assumption is reduced to only hold for individual tiles. For each tile, we compute the $[u, v]$ vector according to Eq. (6), referred to as 'measurement'. These measurements are then used to estimate the motion as follows.

Let $\mathbf{m}$ be the measured apparent motion vector components found in each tile.

$$\mathbf{m} = \begin{bmatrix} u_1 & v_1 & u_2 & v_2 & \dots & u_{16} & v_{16} \end{bmatrix}^\mathsf{T}, \tag{7}$$

where u_i and v_i are the measurement vectors for tile i, $i = 1..N$, u_1 and v_1 are the vectors of the top right tile, and u_{16} and v_{16} are the vectors of the bottom left tile. Let $\mathbf{b}_{yaw}$, $\mathbf{b}_{pitch}$, $\mathbf{b}_{roll}$, and $\mathbf{b}_z$ be the vectors in the same form as $\mathbf{m}$, but normalized with the components of their corresponding base vector field. The resulting normalized $\mathbf{b}$ vectors are:

$$\mathbf{b}_{pitch} = \tfrac{1}{4}\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^\mathsf{T}, \tag{8}$$

$$\mathbf{b}_{yaw} = \tfrac{1}{4}\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^\mathsf{T}, \tag{9}$$

$$\mathbf{b}_{roll} = \tfrac{1}{4\sqrt{10}}\begin{bmatrix} 3 & -3 & 3 & -1 & 3 & 1 & 3 & 3 & 1 & -3 & 1 & -1 & 1 & 1 & 1 & 3 & -1 & -3 & -1 & -1 & -1 & 1 & -1 & 3 & -3 & -3 & -3 & -1 & -3 & 1 & -3 & 3 \end{bmatrix}^\mathsf{T}, \tag{10}$$

$$\mathbf{b}_{z} = \tfrac{1}{4\sqrt{10}}\begin{bmatrix} -3 & -3 & -1 & -3 & 1 & -3 & 3 & -3 & -3 & -1 & -1 & -1 & 1 & -1 & 3 & -1 & -3 & 1 & -1 & 1 & 1 & 1 & 3 & 1 & -3 & 3 & -1 & 3 & 1 & 3 & 3 & 3 \end{bmatrix}^\mathsf{T}. \tag{11}$$

Pitch and yaw base vectors are the same for every partitioning of the image, except that their normalizer are different. The base vectors for the z motion are uniformly expanding form the center of the image with their lengths being proportional to the displacement of their tile center and center of the image. The roll base vector for each tile is orthogonal to the z base vector. Based on the four base vector fields, the model is

$$\mathbf{m} - (\alpha \cdot \mathbf{b}_{yaw} + \beta \cdot \mathbf{b}_{pitch} + \gamma \cdot \mathbf{b}_{roll} + \delta \cdot \mathbf{b}_z) = \epsilon\,, \tag{12}$$

with ϵ being the error we encounter in our estimate. The objective is to minimize ϵ, by choosing proper α, β, γ, and δ values. The intuition behind this model is that it describes a way to linearly decompose the measured $\mathbf{m}$ vector into a linear combination of the known $\mathbf{b}$ vectors. The parameters $\alpha, \beta, \gamma, \delta$ from the linear combination are then the scaled, angular velocities in their respective direction. The presented form is equivalent to a standard generalized linear regression problem, for which there exist multiple solution strategies.

OLS solution A solution to the problem is the Ordinary Least Squares (OLS) approach. This approach requires us to assume, that the error on the estimates follows a Gaussian

distribution. Given $B = \begin{bmatrix}\mathbf{b}_{yaw} & \mathbf{b}_{pitch} & \mathbf{b}_{roll} & \mathbf{b}_z\end{bmatrix}$ we can rewrite Eq. (12) as

$$\mathbf{m} - B \begin{bmatrix}\alpha & \beta & \gamma & \delta\end{bmatrix}^\mathsf{T} = \epsilon. \tag{13}$$

Then the OLS solution is

$$\begin{bmatrix}\alpha & \beta & \gamma & \delta\end{bmatrix}^\mathsf{T} = \left(B^\mathsf{T}B\right)^{-1} B^\mathsf{T}\mathbf{m} = B^\mathsf{T}\mathbf{m}. \tag{14}$$

Since the columns of matrix B are orthonormal, it follows that $B^\mathsf{T}B = B^{-1}B = I$. In every frame, the apparent $x-y$ vectors are measured in all 16 tiles. The CPU then computes the matrix vector product $B^\mathsf{T}\mathbf{m}$ to get the motion estimations in each of the four dimensions. Note that B is constant and is known at compile time. Motion estimation is therefore just a linear combination of the vector components in $\mathbf{m}$. This operation requires at most $4 \cdot 32 = 128$ multiply and add operations on a standard CPU and should thus easily fit into the computational budget of even very limited processing devices. One can expect to reduce this even further by explicitly multiplying the expression out and exploiting the sparse nature of B.

RANSAC solution In visual odometry, a robust estimation that assumes outliers to be present, RANSAC [15], has been successfully applied in the past [16]. In addition to OLS, RANSAC is used to eliminate outliers from the regression problem in Eq. (12).

3 Experiments

The algorithms has been implemented on the Apron [33] environment which can also compile and load the code to the SCAMP-5 hardware. First several simulated results are shown using the Apron simulator, then the real-world experiments using the actual SCAMP-5 is presented. For 2DoF, all of the algorithm was implemented on the FPSP. For the 4DoF, the measurement of the individual tile vectors takes place on the FPSP. Then for the simulation with Apron, the matrix multiplication for the actual estimate are done a CPU, and for the real-world experiments, the matrix multiplication happens on a ARM Cortex M4 + M0 dual core (200 MHz) micro-controller, available on the chip. Yaw and pitch are estimated using the ordinary least squares method, roll and z-translation are estimated using a 5-sample RANSAC with 90% inlier confidence threshold.

Fig. 5: Example frames from the test datasets. It is expected that frames with low variance are harder to track.

Table 1: Mean and standard deviations of squared angular velocity errors. (both datasets)

Units: $[\frac{rad^2}{s^2}]$	Mean Error	Std. Error
Living room	7.038 e-06	8.963 e-06
Office	5.831 e-06	7.902 e-06

3.1 Trajectories

For Apron simulation, we created two set of synthetic datasets: one set with a real-world trajectory, and one set with synthetic trajectory. The real-world trajectory, recorded with

the gyroscope of a smartphone, was used in two synthetic environments to create synthetic datasets: office scene and living room scene. The trajectories were ported to Persistence Of Vision Raytracer, POVRay [34] to render realistic images from the scenes. Fig. 3 shows some frames taken from both datasets. The frames captured with the camera pointing to the ceiling may pose the biggest challenge to the algorithm.

3.2 Camera Tracking on Apron

Two experiments were performed in Apron: real-world trajectory and simulated trajectory. In the real-world trajectory experiment, the camera remains at the same place throughout the entire scene, and only the full rotation is analyzed. The trajectory was tested in *living room* and *office* environments. In the synthetic trajectory experiment, a forward facing camera is mounted on a person moving forward. The trajectory was rendered in two different environments, a *kitchen* scene and a *sofa* scene.

Fig. 6 shows the estimated angular velocities for 4DoF tracking algorithm on the *living room* and the *office* datasets. The shaded areas at the bottom give an indication about the variance of the dataset at the given frames. It is expected that the less variance there is in the images, the harder it gets for the algorithm to successfully track the motion. Table 1 shows the mean and standard deviations of the squared error of the 4DoF algorithm in the three degrees of freedom (rotations) pose estimation task.

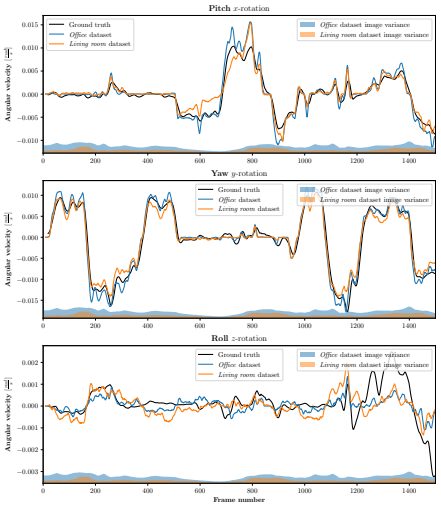


Fig. 6: Estimated angular velocities for *living room* and *office* datasets. The shaded regions at the bottom symbolize the image variance of the datasets.

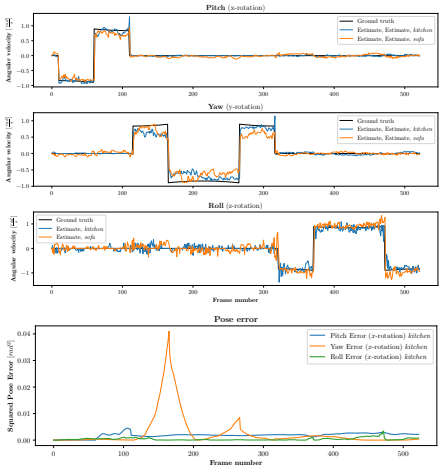


Fig. 7: 4DoF algorithm. (Top 3) angular velocities in *kitchen* and *sofa* datasats. (Bottom) estimated pose error on *kitchen* dataset in full 4DoF estimation

The tracking of yaw and pitch angles is consistent. The roll rotation estimate is considerably less accurate than the yaw and pitch estimate. Table 2 and Table 3 show the mean squared and the maximum individual errors reached on the sequence.

Table 2: Means and standard deviations of the squared errors of the estimated poses on *living room* and *office* datasets.

Units: $[rad^2]$	Mean Error	Std. Error	Max. Error	End Error
Living room	0.108	0.079	0.322	0.214
Office	0.042	0.046	0.194	0.063

Table 3: Maximum and end errors for the individual dimensions of the estimated poses on *living room* and *office* datasets.

Units: $[rad]$	Max Pitch	Max Yaw	Max Roll	End Pitch	End Yaw	End Roll
Living room	0.498	0.290	-0.271	0.362	0.266	-0.109
Office	0.376	0.193	-0.214	0.136	0.193	-0.084

The algorithm was also tested on a synthetic trajectory showing the point of view from a person walking slowly towards a kitchen counter (*kitchen* dataset) and towards a sofa in a room (*sofa* dataset). Fig. 7 shows the estimated angular velocities (Top 3 plots) and camera orientation error (bottom) in global coordinates. The mean translation error for *kitchen* is 0.16 m and for *sofa* is 0.08 m (not shown on the plots). The *sofa* dataset shows a better performance at translation estimation, whereas the *kitchen* dataset exhibits a better performance in orientation estimation. Orientation errors are relatively small with largest squared error being around $0.04\ rad$. Contrary to the expectation, both were encountered in the yaw direction, rather than roll and z directions. Since we are dealing with a monocular system, the scale of the translational movement can not be automatically established. The scale was chosen using prior information.

Table 4: Orientation errors for *kitchen* dataset.

Units: $[rad^2]$	Pitch	Yaw	Roll
Avg. Squared Error	0.0056	0.0026	0.0009
Max. Squared Error	0.0085	0.0398	0.0078

Table 5: Orientation errors for *sofa* dataset.

Units: $[rad^2]$	Pitch	Yaw	Roll
Avg. Squared Error	0.0161	0.0042	0.0043
Max. Squared Error	0.0301	0.0464	0.0300

3.3 Camera Tracking on SCAMP-5

The motion estimation produced by the proposed approach running upon SCAMP-5 was evaluated by direct comparison to motion capture ground truth. Figure 8 shows the yaw, pitch and roll estimations produced. Throughout the SCAMP-5 was able to maintain a frame-rate between 400-500 Hz. Though noisy the produced estimation was able to operate under high angular velocities which would significantly blur a standard camera image, rendering it unfit for use in motion estimation. Further results and comparisons are given in this paper's corresponding video.

3.4 Performance

The number of SCAMP operations required to perform the operation on the vision chip were obtained from [35]. The power consumption of the vision chip at the maximum instruction rate of 10 MHz is 1.23 W and an idle power consumption is 0.2 mW [9]. Power consumption for a fixed frame rate below the maximum frame rate is computed under the assumption that the processor finishes a frame at maximum clock speed and then switches to idle state until the beginning of the new frame. Table 6 shows estimated

Fig. 8: Graphs of angular rate estimates produce by SCAMP-5 at 400-500Hz under a combination of orientation changes. Light colors represent the estimation, where as dark represent motion captured ground-truth.

performance calculations for the algorithm. It clearly shows that using a FPSP can give extremely high frame rates at a very low energy consumptions.

Table 6: Estimated performance values for the algorithms on the SCAMP chip.

	2DoF algorithm	4DoF algorithm
Avg. cycles per frame	846.72	13,547.52
Std. cycles per frame	314.58	5,033.28
Theoretical max. FPS.	11810	738.125
Power @ max. FPS	1.23 W	1.23 W
Estimated power @ 60FPS	6.4 mW	100.2 mW

4 Conclusions

This paper presented several contributions to the areas of camera tracking on Focal-Plane Sensor-Processor (FPSP) Arrays. The simple structure, high achievable frame rate and low power consumption achievable with the algorithms allow for promising future applications. This is the first algorithm to use a tiling method for pose estimation. It was shown that the global motion of the camera can be detected using only the vectors measured in the individual tiles by means of a model fitting approach. An Ordinary Least Squares and a RANSAC based approach for the model fitting was presented. The evaluation of the algorithm showed good tracking performance on synthetic and real-world trajectories. 4DoF camera tracking is ideal for applications where the agent is mechanically restricted to move into one direction, such as wheeled robots and cars.

This research was performed in an attempt to get accurate tracking at high frame rates on energy limited devices, such as small aerial vehicles or virtual-reality headsets. The massive frame rate provided by the FPSP allows for simpler tracking algorithms using larger approximations. This opens up an interesting design space involving the trade off between frame rate and complexity. Also, the base vectors were estimated manually. Learning-based methods are a promising direction for future work.

References

1. Carey, S.J., Barr, D.R., Wang, B., Lopich, A., Dudek, P.: Locating high speed multiple objects using a SCAMP-5 vision-chip. In: Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on, IEEE (2012) 1–2
2. Censi, A., Scaramuzza, D.: Low-latency event-based visual odometry. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE (2014) 703–710
3. Mueggler, E., Huber, B., Scaramuzza, D.: Event-based, 6-DOF pose tracking for high-speed maneuvers. In: Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, IEEE (2014) 2761–2768
4. Dominguez-Castro, R., Espejo, S., Rodriguez-Vazquez, A., Carmona, R.A., Foldesy, P., Zarándy, Á., Szolgay, P., Szirányi, T., Roska, T.: A 0.8-/spl mu/m CMOS two-dimensional programmable mixed-signal focal-plane array processor with on-chip binary imaging and instructions storage. IEEE Journal of Solid-State Circuits **32**(7) (1997) 1013–1026
5. Linan, G., Espejo, S., Dominguez-Castro, R., Rodriguez-Vazquez, A.: Architectural and basic circuit considerations for a flexible 128×128 mixed-signal SIMD vision chip. Analog Integrated Circuits and Signal Processing **33**(2) (2002) 179–190
6. Poikonen, J., Laiho, M., Paasio, A.: MIPA4k: A 64×64 cell mixed-mode image processor array. In: Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on, IEEE (2009) 1927–1930
7. Dudek, P., Hicks, P.J.: A general-purpose processor-per-pixel analog SIMD vision chip. IEEE Transactions on Circuits and Systems I: Regular Papers **52**(1) (2005) 13–20
8. Dudek, P.: Implementation of SIMD vision chip with 128×128 array of analogue processing elements. In: Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on, IEEE (2005) 5806–5809
9. Carey, S.J., Lopich, A., Barr, D.R., Wang, B., Dudek, P.: A 100,000 fps vision sensor with embedded 535GOPS/W 256×256 SIMD processor array. In: VLSI Circuits (VLSIC), 2013 Symposium on, IEEE (2013) C182–C183
10. Chua, L.O., Yang, L.: Cellular neural networks: Theory. IEEE Transactions on Circuits and Systems **35**(10) (Oct 1988) 1257–1272
11. Zarandy, A., Dominguez-Castro, R., Espejo, S.: Ultra-high frame rate focal plane image sensor and processor. IEEE Sensors Journal **2**(6) (2002) 559–565
12. Dudek, P., Hicks, P.J.: A CMOS general-purpose sampled-data analog processing element. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing **47**(5) (2000) 467–473
13. Dudek, P.: Accuracy and efficiency of grey-level image filtering on VLSI cellular processor arrays. In: Proc. CNNA. (2004) 123–128
14. Dudek, P., Hicks, P.J.: An analogue SIMD focal-plane processor array. In: Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on. Volume 4., IEEE (2001) 490–493
15. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM **24**(6) (1981) 381–395
16. Nister, D., Naroditsky, O., Bergen, J.: Visual odometry. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2004. Volume 1. (2004) 652–659 Vol.1
17. Cheng, Y., Maimone, M., Matthies, L.: Visual odometry on the Mars exploration rovers. In: Systems, Man and Cybernetics, 2005 IEEE International Conference on. Volume 1., IEEE (2005) 903–910

18. Tardif, J.P., Pavlidis, Y., Daniilidis, K.: Monocular visual odometry in urban environments using an omnidirectional camera. In: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, IEEE (2008) 2531–2538
19. Milford, M.J., Wyeth, G.F.: Single camera vision-only SLAM on a suburban road network. In: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, IEEE (2008) 3684–3689
20. Goecke, R., Asthana, A., Pettersson, N., Petersson, L.: Visual vehicle egomotion estimation using the Fourier-Mellin transform. In: Intelligent Vehicles Symposium, 2007 IEEE, IEEE (2007) 450–455
21. Corke, P., Strelow, D., Singh, S.: Omnidirectional visual odometry for a planetary rover. In: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. Volume 4., IEEE (2004) 4007–4012
22. Steinbrücker, F., Sturm, J., Cremers, D.: Real-time visual odometry from dense RGB-D images. In: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, IEEE (2011) 719–722
23. Kim, H., Handa, A., Benosman, R., Ieng, S.H., Davison, A.: Simultaneous mosaicing and tracking with an event camera. In: Proceedings of the British Machine Vision Conference, BMVA Press (2014)
24. Bardow, P., Davison, A.J., Leutenegger, S.: Simultaneous optical flow and intensity estimation from an event camera. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2016) 884–892
25. Mueggler, E., Bartolozzi, C., Scaramuzza, D.: Fast event-based corner detection. In: BMVC. (2017)
26. Conradt, J.: On-board real-time optic-flow for miniature event-based vision sensors. In: 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO). (Dec 2015) 1858–1863
27. Rebecq, H., Gallego, G., Mueggler, E., Scaramuzza, D.: EMVS: Event-based multi-view stereo—3D reconstruction with an event camera in real-time. International Journal of Computer Vision (Nov 2017)
28. Gallego, G., Lund, J.E.A., Mueggler, E., Rebecq, H., Delbruck, T., Scaramuzza, D.: Event-based, 6-DOF camera tracking from photometric depth maps. IEEE Transactions on Pattern Analysis and Machine Intelligence **PP**(99) (2017) 1–1
29. Weikersdorfer, D., Hoffmann, R., Conradt, J.: Simultaneous localization and mapping for event-based vision systems. In Chen, M., Leibe, B., Neumann, B., eds.: Computer Vision Systems, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 133–142
30. Vidal, A.R., Rebecq, H., Horstschaefer, T., Scaramuzza, D.: Ultimate SLAM? combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. IEEE Robotics and Automation Letters **3**(2) (April 2018) 994–1001
31. Rebecq, H., Horstschaefer, T., Scaramuzza, D.: Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In: BMVC. (2017)
32. Bose, L., Chen, J., Carey, S.J., Dudek, P., Mayol-Cuevas, W.: Visual odometry for pixel processor arrays. In: 2017 IEEE International Conference on Computer Vision (ICCV). (Oct 2017) 4614–4622
33. Barr, D.R., Dudek, P.: A cellular processor array simulation and hardware prototyping tool. In: Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on, IEEE (2008) 213–218
34. POV-Team: POVRay. http://www.povray.org/
35. Chen, J.: SCAMP-5c vision system. https://github.com/jianingchen/scamp5c (2016)