Event-Based Feature Tracking Using the Iterative Closest Point Algorithm

Joseph Adeola, Preeti Verma, and Moses Ebere

Abstract—A groundbreaking revolution has been sparked in computer vision and robotics through the emergence of event-based vision sensors, represented by the Dynamic and Active-pixel Vision sensor (DAVIS). These sensors are capable of generating asynchronous brightness changes and synchronous grayscale frames, offering unmatched advantages in high temporal resolution, minimal latency, and an extended dynamic range. Developing specialized algorithms tailored to their unique output becomes crucial to realize their transformative potential fully. This research implements an innovative methodology for event-based feature tracking, leveraging the powerful Iterative Closest Point (ICP) algorithm. By intelligently grouping events and transforming the data into image frames at predefined intervals, computational complexity is significantly reduced, enabling seamless perception and analysis in robotics applications. The procedure seamlessly integrates efficient event binning, leveraging the Shi-Tomasi and Canny edge detectors for feature detection and harnessing the robust capabilities of the ICP algorithm for accurate feature tracking. Rigorous feasibility evaluation and exceptional performance validation unequivocally demonstrate the untapped potential of event-based vision sensors, ushering in a new era of enhanced robotics capabilities and real-time perception in dynamic environments. This research sets the stage for exciting advancements in robotics perception, propelling us toward a future shaped by the remarkable power of event-based vision sensors.

Index Terms—Events, Feature Detection, Feature Tracking, ICP.

1 Introduction

Event-based vision sensors have gained significant attention in computer vision and robotics due to their unique characteristics and wide range of applications. These sensors, such as the Dynamic and Active-pixel Vision sensor (DAVIS), offer advantages like high temporal resolution, minimal latency, and extended dynamic range. However, fully utilizing the potential of event-based cameras requires specialized algorithms tailored to their unconventional output.

This research paper aims to implement a methodology for event-based feature tracking using the Iterative Closest Point (ICP) algorithm, inspired by [1]. The research experiment employs the shapes_6dof dataset acquired from the DAVIS 240C Datasets [2]. An efficient processing strategy is explored to address the computational challenges posed by the high frequency and large volume of events. The proposed methodology adopts a grouped event processing approach, collectively converting events into image frames. This optimization reduces computational complexity and enhances the performance of the feature tracker.

The methodology encompasses crucial steps for efficient event-based feature tracking. Event binning is employed to condense event data and alleviate the computational load. This technique groups events within intervals and transforms them into binary image frames, improving the efficiency of data handling and analysis. The Shi-Tomasi feature detector is used for corner detection, while the Canny edge detector generates binary edge maps. Local edge-map patches around detected corner points serve as model points for feature tracking.

Accurate feature tracking is executed by using the ICP algorithm, which aligns model points with data points from event frames. The algorithm estimates transformations that minimize distances between corresponding points, ensuring optimal alignment and robust tracking. A feature reinitialization process is implemented to handle situations where the number of features falls below a minimum threshold.

This research aims to demonstrate the feasibility and performance of event-based feature tracking using the ICP algorithm. The implemented methodology, evaluation, and obtained results will contribute valuable insights to understanding event-based vision sensors. The research sets the stage for improved perception and analysis in robotics and computer vision applications.

2 Related Work

In recent years, significant advancements have been made in event-based vision sensors, leading to the development of various algorithms and techniques for efficient processing and analysis of event data. Zhu et al. [3] proposed a novel approach in soft data association, using an intertwined expectation-maximization scheme and optical flow computation to calculate association probabilities. Mitrokhin et al. [4] introduced a unique event stream representation for object tracking, eliminating the need for explicit optical flow computation. Iacono et al. [5] explored the use of deep learning techniques for object detection using visual events on the iCub robotic platform. The study explores the influence of temporal integration of event data and introduces a

• *Joseph Adeola, Preeti Verma, and Moses Ebere are Erasmus Mundus Master's students in Intelligent Field Robotic Systems at the University of Girona, Spain.*
E-mail: adeolajosepholoruntoba@gmail.com, preeti8600verma@gmail.com, moseschukaebere@gmail.com,

Manuscript submitted on June 12, 2023.

novel pipeline that exploits mature frame-based algorithms to expedite event-based dataset annotation.

Afshar et al. [6] contributed to space situational awareness by introducing an optical space imaging dataset captured using event-based neuromorphic vision sensors. This publicly-accessible dataset includes day and night time recordings, diverse labeled targets, and simultaneous co-collections from different event-based sensors. Chen et al. [7] presented a groundbreaking deep neural network that learns and predicts object-level motion/transforms in event-based object tracking. This approach utilizes a unique representation called Time-Surface with Linear Time Decay and achieves exceptional performance, particularly in challenging scenarios characterized by high-speed motion and low illumination conditions.

3 METHODOLOGY

In this section, we describe the methodology we employed for event-based feature tracking using the Iterative Closest Point (ICP) algorithm. Our approach addresses the computational challenges posed by the high frequency and large volume of events generated by event-based cameras. Instead of processing each event individually, we adopted a grouped approach where we group events within a certain interval and convert them into image frames collectively. For feature detection, we utilized the Shi-Tomasi feature detector for corner detection and applied the Canny edge detector for edge generation. Next, we utilized the ICP algorithm for feature tracking, aligning the model points and data points extracted from the event frames. To handle situations where the number of features fell below a minimum threshold, we implemented a feature reinitialization process. Our methodology provides an efficient and effective solution for event-based feature tracking.

The subsequent subsections elucidate each step in our methodology, covering event binning, feature detection, the feature tracker, feature reinitialization, and ground truth estimation. These sections delve into the techniques and algorithms utilized, offering a comprehensive understanding of our approach to event-based feature tracking with the ICP algorithm.

3.1 Event Binning

Processing the DAVIS camera data on an events-by-events basis can be computationally expensive due to the high frequency and large volume of events generated. Each event corresponds to a specific pixel location and its polarity, resulting in a substantial amount of data that needs to be processed and analyzed.

Thus, performing feature tracking on an events-by-events basis would require handling and analyzing each event individually, which can impose a significant computational burden. The real-time processing of a large number of events can pose challenges in terms of memory usage, computational power, and overall system efficiency.

To address these complexities and optimize computational resources, we adopted a grouped approach for event processing. Instead of processing each event individually, we grouped the events within a certain interval and converted them collectively into image frames.

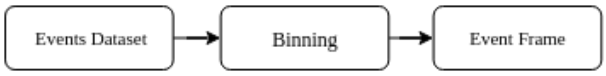


Fig. 1: Event Binning Flowchart

In our implementation of the feature tracker for the DAVIS camera, we carefully selected an appropriate interval for grouping the events. This interval was chosen with the aim of balancing the thickness and sparsity of the resulting edges in the image frames.

Within each interval, we assigned a pixel value of 255 to the pixel locations where events were triggered, irrespective of their polarity. By doing so, we obtained binary image frames where the presence of an event was represented by a white pixel.

This grouping approach allowed us to condense the event data and reduce the computational load associated with feature tracking. By processing the events collectively within intervals, we could handle and analyze the data more efficiently, optimizing the overall performance of the feature tracker.

3.2 Feature Detection

The feature detection process in our algorithm involves several steps. First, the Shi-Tomasi feature detector is used to identify corner points on the intensity image frame. The Shi-Tomasi detector is a widely-used method for detecting corners in images and works by computing the minimum eigenvalue of the second-moment matrix of image gradients at each pixel and identifying corners as points with high minimum eigenvalues. Next, the intensity frame is converted to grayscale and a Canny edge detector is applied to generate a binary where 1 represents an edge pixel and 0 represents non-edge pixels. Following this, local edge-map patches are drawn around corner points. These patches represent small regions of the image that contain edges or corners. Specifically, we used a square patch of size 19 x 19 centered on each corner point. The choice of patch size can significantly impact feature tracking performance. Larger patches capture more information but are computationally expensive, while smaller patches are faster but may not capture enough information. Therefore, we chose the patch size to balance accuracy and efficiency. In our implementation, we applied a mask on the intensity image before detecting corners to restrict the corner detector from detecting corners close to the border of the image. This is to ensure that patches drawn on the image don't fall outside the image. The patch size is an adjustable parameter that can be tuned as desired.

After this, the edge pixels inside a patch are used to define model points. These model points are extracted for all corner points detected by the Shi-Tomasi corner detector. After extracting the model points for all corner points, outlier corners with insufficient edge pixels (model points) around them are removed to improve the accuracy of the feature tracking algorithm. This is due to the fact that the Shi-Tomasi corner detector is known for its robustness in detecting salient corners, which may not necessarily have edge pixels around them. In our implementation, we consider corners with less than five model points outliers and delete the corner from the list of features we want to track.

Algorithm 1 DAVIS Feature Tracker

Feature detection:
- Detect corner points on the frame (Shi-Tomasi detector).
- Run Canny edge detector (returns a binary image, 1 if edge pixel, 0 otherwise).
- Extract local edge-map patches around corner points, and convert them into model point sets.

Feature tracking:
- Get the event frame with timestamp closest to intensity frame and start from there

for each event frame **do**
  - Extract data point set.
  for each corresponding data point set **do**
    - Estimate the registration parameters between the data and the model point sets (ICP).
    - Transform the model points and corners
  end for
end for

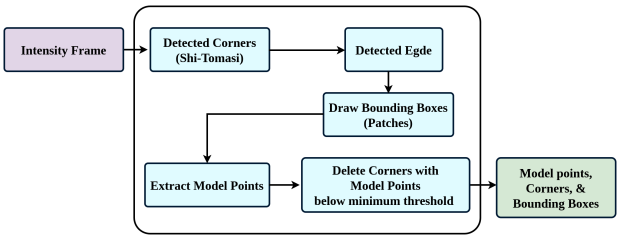


Fig. 2: Event Binning Flowchart

In our method, a constant supply of intensity frames is unnecessary as they are solely utilized for initializing features. Whenever the number of tracked features falls below the minimum threshold, frames can be obtained as required.

3.3 Feature Tracker

To track features, the binned event frames and the model points set obtained from the feature detector are required. The feature tracking algorithm follows a flowchart as depicted in Figure 3.

3.3.1 Data Point Extraction

For each feature i, data points are extracted from the patch centered around that feature. These data points correspond to pixel locations where events were triggered, specifically pixels within the patch with a value of 255. This extraction process is performed for all patches in a given event frame. To enhance the robustness of the subsequent ICP registration algorithm against noise, outliers are eliminated. Data points located more than 3 pixels away from any model point within the patch are discarded. This filtering step ensures the retention of events triggered near edge pixel locations. The resulting set of model points and data points are then passed to the ICP registration algorithm.

3.3.2 ICP Registration

The Iterative Closest Point (ICP) algorithm is a widely used technique in the field of point cloud registration. It serves as a fundamental tool for aligning two or more point clouds. For the registration process, we used the point-to-point variant of the ICP algorithm, which is available in the open-source Python library, open3d. This aim was to align the model points (denoted as Q) with data points (denoted as P) to determine the optimal transformation (denoted as T) that minimizes the distances between corresponding points.

The registration process begins by initializing an estimate of the transformation between the two sets, which we assume to be the identity transformation. Each data point (p) is paired with its nearest corresponding model point (q) based on the Euclidean distance metric, establishing point-to-point correspondences between the sets. This correspondence set is denoted as $K = \{(p, q)\}$.

An iterative refinement process is then employed to estimate a transformation that aligns the data points with the model points. This refinement is achieved by minimizing the distances between corresponding points, as defined by the objective function:

$$E(T) = \sum_{(p,q)\in K} \|p - Tq\|^2 \tag{1}$$

Here, p represents a data point, q represents a corresponding model point, and T represents the transformation matrix. The objective function $E(T)$ measures the squared Euclidean distances between the transformed model points (Tq) and the data points (p), summed over all the correspondences in the set K.

The iterations continue until convergence is reached, as determined by predefined convergence criteria. Once the algorithm converges, the final estimated transformation (T) represents the alignment and registration between the model points and data points.

This transformation is subsequently applied to the features and the model point set for estimation. The transformed features and model point set is then used as input for tracking features in the next event frame.

3.4 Feature Reinitialization

In our tracking algorithm, features are discarded under certain conditions, such as when they get too close to the image border or when the number of model points or data points falls below a minimum threshold, rendering the ICP algorithm unable to function properly.

To address the situation when the number of remaining features drops below the minimum threshold, we employ a feature reinitialization process. This process involves selecting the next intensity frame in the sequence, which has the timestamp closest to the last event frame where tracking was suspended. We then perform feature detection on this new intensity frame and resume tracking.

However, if the number of features detected in the new intensity frame is less than the minimum number of features required for tracking, we skip that frame and proceed to the next frame in the sequence. This iterative process continues until we find an intensity frame where the number of detected features meets or exceeds the minimum threshold.

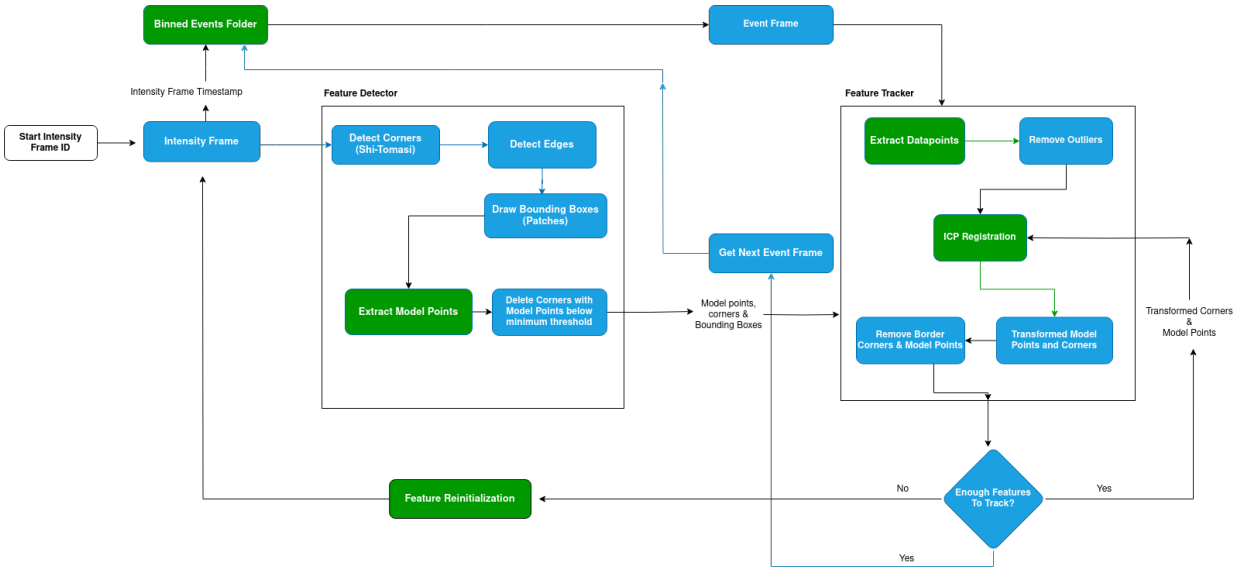


Fig. 3: Feature Tracker Flowchart Design

The minimum number of features necessary for successful tracking is dynamically determined based on the number of features detected by the feature detector. Specifically, we set the threshold for the minimum number of features to track at 70% of the total number of features detected in the intensity image. This threshold value can be adjusted according to individual preferences or specific application requirements.

By implementing this feature reinitialization approach, we ensure that the tracking algorithm maintains an adequate number of features for robust and reliable tracking of the target object or region of interest. This strategy enhances the overall accuracy and resilience of the tracking process, even in scenarios where features may be lost or become insufficient for successful tracking.

3.5 Ground Truth Estimate

To obtain a ground truth estimate of the feature trajectory over time, the following steps were taken. Firstly, the Harris corner detector was used to detect corner features in the first intensity image frame. However, to bias the subsequent feature detection towards corners, the detected corners were dilated. This dilation step helps to enhance the detection of corner points since the subsequent algorithm, SIFT, primarily focuses on detecting features in an image. By emphasizing corner points, the subsequent steps can more accurately track the desired features.

Next, the SIFT algorithm was applied to compute keypoints and descriptors for the dilated corner features in both the first and second image frames. By finding good matches between the corner features in these two frames, correspondences were established. To estimate the transformation between the two frames, RANSAC (Random Sample Consensus) was used. RANSAC helps to robustly estimate the homography transformation by considering inlier correspondences while disregarding outliers. This homography transformation represents the relationship between the inlier corner features in the first frame and the second frame.

Using the obtained homography transformation, the corner features in the first frame were transformed to estimate their new locations in the second frame. SIFT keypoints and descriptors were then computed for these transformed features.

After obtaining the transformed features with their respective keypoints and descriptors, they served as the expected positions for feature tracking in the second frame. The iterative process then proceeded by finding matches between two consecutive frames and so on. The transformed features from the previous iteration acted as the reference positions for matching in each subsequent frame. By repeating this process, the feature tracking algorithm established correspondences and estimated the trajectory of the features across multiple frames. This iterative approach allowed for continuous tracking and updating of the feature positions as the sequence of frames progressed.

3.6 Joint Autonomous Task

The joint autonomous task within our project focused on detecting an Aruco marker while a Kobuki Turtlebot2 robot autonomously explores its environment. Once the marker is detected, the robot estimates its pose and autonomously navigates toward it. This subsection provides an overview of the implementation of the perception side, specifically the marker detection and pose estimation.

For marker detection, we developed a perception module utilizing the OpenCV library. The module processed the robot's camera images and employed a marker detection algorithm to identify ArUco markers in the environment.

To estimate the poses of the detected markers, a pose estimation algorithm was implemented. It utilized the corner points of the markers and camera parameters to calculate their positions and orientations relative to the robot's coordinate system.

To enhance visualization, we provided graphical representations of the marker orientations overlaid on the camera images. This facilitated a clear understanding of the markers' positions and orientations.

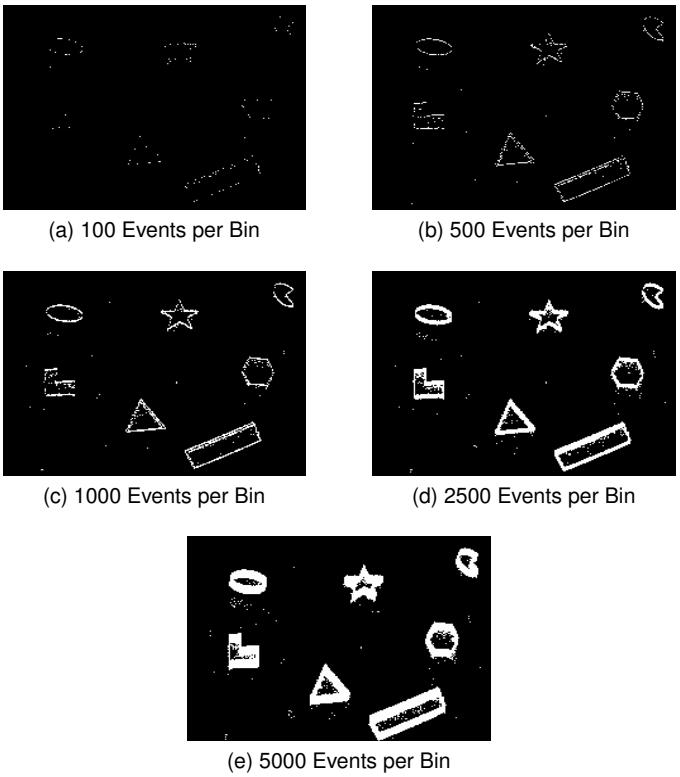
Finally, leveraging the estimated poses, the robots were programmed to autonomously navigate toward the detected markers. The navigation system utilized the marker's pose information to generate appropriate motion commands, enabling the robots to reach the desired marker locations.

By integrating the perception and navigation modules into the overall autonomous exploration system, our robots successfully detected ArUco markers, estimated their poses, and autonomously navigated toward them, accomplishing the joint autonomous task. The simulation and real-world test results for the joint autonomous task are presented in the result section.

4 Results and Discussion

We evaluated the performance of the event-based feature tracker on different scenes - most of which were based on scenes with objects that have well-defined edges. To progressively build up to the final results of this project, the following subsections present intermediate results of different parts of the integration pipeline in Figure 3.

4.1 Binning Events



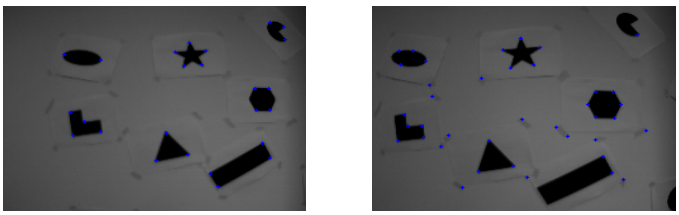
(a) 100 Events per Bin (b) 500 Events per Bin (c) 1000 Events per Bin (d) 2500 Events per Bin (e) 5000 Events per Bin

Fig. 4: Images for different binning sizes. The higher the size of each bin, the thicker the edges obtained.

The size of each bin is one parameter that must be tailored to the dataset being used. Choosing a low bin size could yield data points that are not sufficient to capture the actual transformation of the features of interest. On the other end of the spectrum, a high bin size could introduce what we consider motion blur for events. This happens when the data in the local region was generated during very high camera movements; therefore, a high bin size would not be able to account for seemingly little but important intermediate motions. The upshot of this would be 'jumps' in the tracking output. For the dataset represented in Figure 4 (i.e., the shapes_6DOF dataset by Mueggler, E. et al. [2]), a bin of 1000 events was used for the final implementation.

4.2 Model Points

4.2.1 Corners



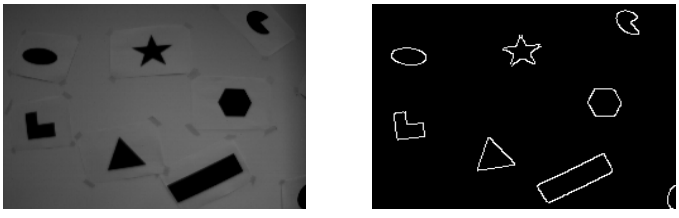
(a) Harris Corners (b) Shi-Tomasi Corners

Fig. 5: Detected Corners.

The results of the two corner detection algorithms explored - Harris and Shi-Tomasi - are presented in Figure 5. The detected corners are shown using blue crosses. There are significantly more corners detected using the Shi-Tomasi algorithm; however, this seems to be a deviation from what is a direct conclusion of the scoring functions of these algorithms in the literature. Being an improvement over the Harris corner detection, the Shi-Tomasi corner detector is designed to detect less spurious corners. In the Figure 5a, the Harris detector misses some real corners which the Shi-Tomasi detector succeeds in picking up (Figure 5b). This was instrumental in the selection of the latter alongside the advantages it offers in reducing the number of feature reinitializations required to keep the tracker running for long periods.

4.2.2 Edges

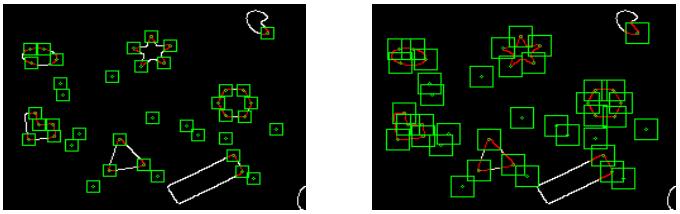
Figure 6b is the binary edge-map counterpart of Figure 6a. The edge image was obtained using the Canny edge detector and hysteresis thresholding to emphasize strong edges and score weak edges based on their connectivity to strong edges.



(a) Reference Intensity Image (b) Canny Edges

Fig. 6: Edge Detection.

4.2.3 Patches



(a) 10x10 Pixel Patches (b) 19x19 Pixel Patches

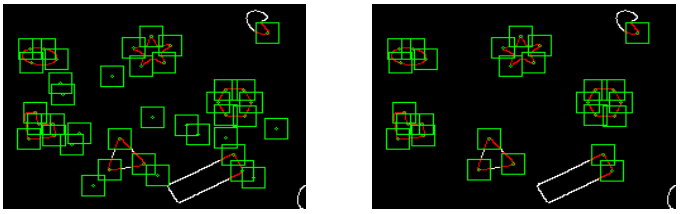
Fig. 7: Local Edge-Map Patches.

The patch size is a tunable parameter; therefore, it is more preferably determined experimentally. In Figure 7, we visualize two patch sizes - 10x10 in 7a 19x19 in 7b - as green boxes over the detected corners and edges. There are as many patches as there are detected corners because the candidate features to be tracked are the corners. It is worth noting that the larger the patch size, the higher the expected level of overlap between patches, especially in the case where the candidate corners are close pixel-wise. This could constitute a problem in the following case:

- If the transformation returned by the ICP algorithm is wrong for one patch and correct for the other due to noisy data like spurious edges, this wrong result will have a larger sphere of influence beyond the feature centered on the patch.

4.2.4 Outlier Rejection

The results in Figure 8a show some patches within which there are little to no edge pixels (or model points) present; therefore, these non-informative patches need to be discarded as earlier indicated in the flowchart in Figure 2. Figure 8b shows the effect of the outlier rejection on model points.



(a) Before Outlier Rejection (b) After Outlier Rejection

Fig. 8: Outlier Rejection for Model Points.

Following the outlier rejection implementation on the model point set, Figure 8b shows the final outcome of the feature detection module of the integration pipeline.

4.3 Data Points

Figure 9 shows the data point highlighted in red within the bounding boxes/patches in green.

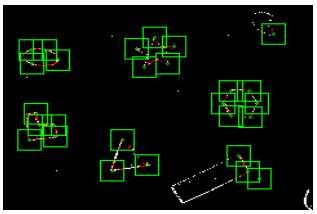
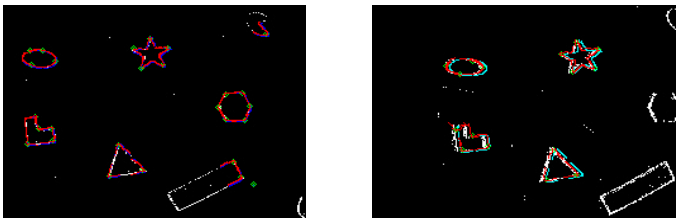


Fig. 9: Datapoints Extracted from Events within Patches

4.4 ICP Results and Reinitialization

In Figure 10a, the result of the ICP implementation is shown. The model points before and after applying ICP are seen in red and blue colors respectively. In this particular case, the ICP algorithm returned a near-identity transformation for all the patches.



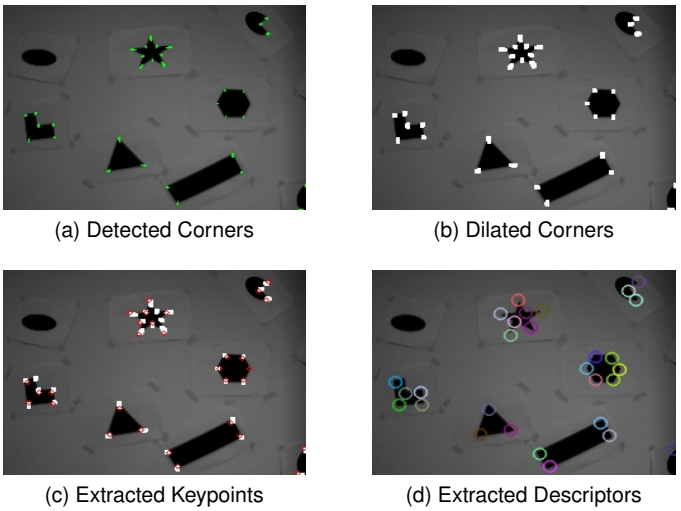
(a) Before Outlier Rejection (b) After Outlier Rejection

Fig. 10: Outlier Rejection for Model Points.

Figure 10b is added to visualize feature reinitialization. This reinitialization occurred about 85,000 events after the result in Figure 10a. The change in color of the transformed model points (light blue) signifies that the features being tracked fell below the stipulated threshold and the reinitialization sub-pipeline was triggered. This is evident by the fact that the three left-most shapes in Figure 10a are now too close to the border in Figure 10b.

4.5 Ground Truth

4.5.1 Feature Detection



(a) Detected Corners (b) Dilated Corners

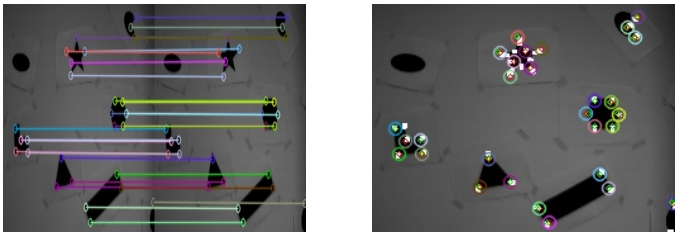
(c) Extracted Keypoints (d) Extracted Descriptors

Fig. 11: Ground Truth - Feature Detection.

Similar to the event-based feature tracking this report focuses on, the ground truth estimation also has a feature detection pipeline which is visually elaborated in 11. From (a) to (d), Harris corners are detected, dilated (to create a bias towards corners for the SIFT algorithm), 'converted' to keypoints and used to compute descriptors.

4.5.2 *Feature Tracking*

Figure 12a shows the established correspondences between two consecutive intensity images. From this, RANSAC is used to estimate the tomography, and the resulting transformation is applied to the features in the older frame to obtain their expected positions in the new frame. Figure 12b illustrates the result of transforming the corners from the old to the new frame.



(a) Detected Corners (b) Dilated Corners

Fig. 12: Feature Detection for the Ground Truth.

4.6 Final Tracking Results

Figures 13a, 13b, 13c, and 13d show the trajectory of the model points and the interest points (corners) between two different intensity image frames.

4.7 Joint Autonomous Task

The results of the ArUco marker detection employed in the joint autonomous task are contained in this section. Due to marked differences in the camera parameters in the simulation and on the real robot, further practical tuning measures were applied during the transition from simulation to real experiments.

4.7.1 *Simulation*

Figure 14 shows the detection of an ArUco marker from the ArUco Original dictionary in OpenCV. The marker was attached to a simulation object in a world scene specially prepared in the Stonefish Simulator. This detection was during an autonomous exploration run by the Kobuki Turtlebot 2 robot, and it was implemented as a generic behavior on the behavior tree, which was used as the coordinating sequencer for the joint autonomous task.

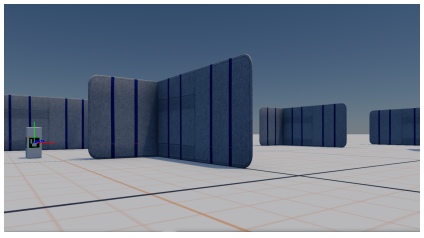
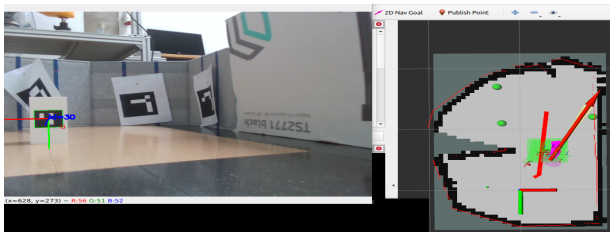


Fig. 14: ArUco Detection in Simulation

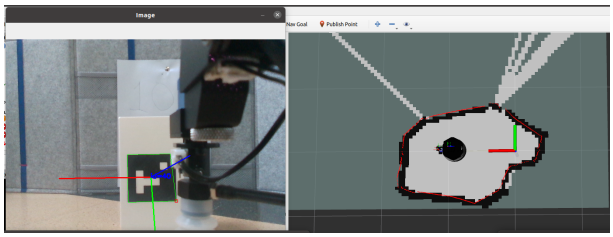
4.7.2 *Real Experiment*

In the experiments, ArUco markers from the '4X4_50' of OpenCV were applied to lightweight boxes. Figures 16a and 16b show ArUcos detected on the left of the screen and the explored map on the right-hand side. The detected aruco is overlayed with a bounding box and a cartesian axis to show what the underlying algorithm sees. In the map on the right, the green window shows where the robot is, the axis is the world frame, the red arrow signifies the localization from the SLAM algorithm also running, and the red line shows the path generated by the path-planning which works in sync with exploration. Figure 16c shows another instance where the manipulation behavior is about to take over following the aruco detection and subsequent move-to-pick behaviors.



(a) ArUco Detection during exploration of Map 1

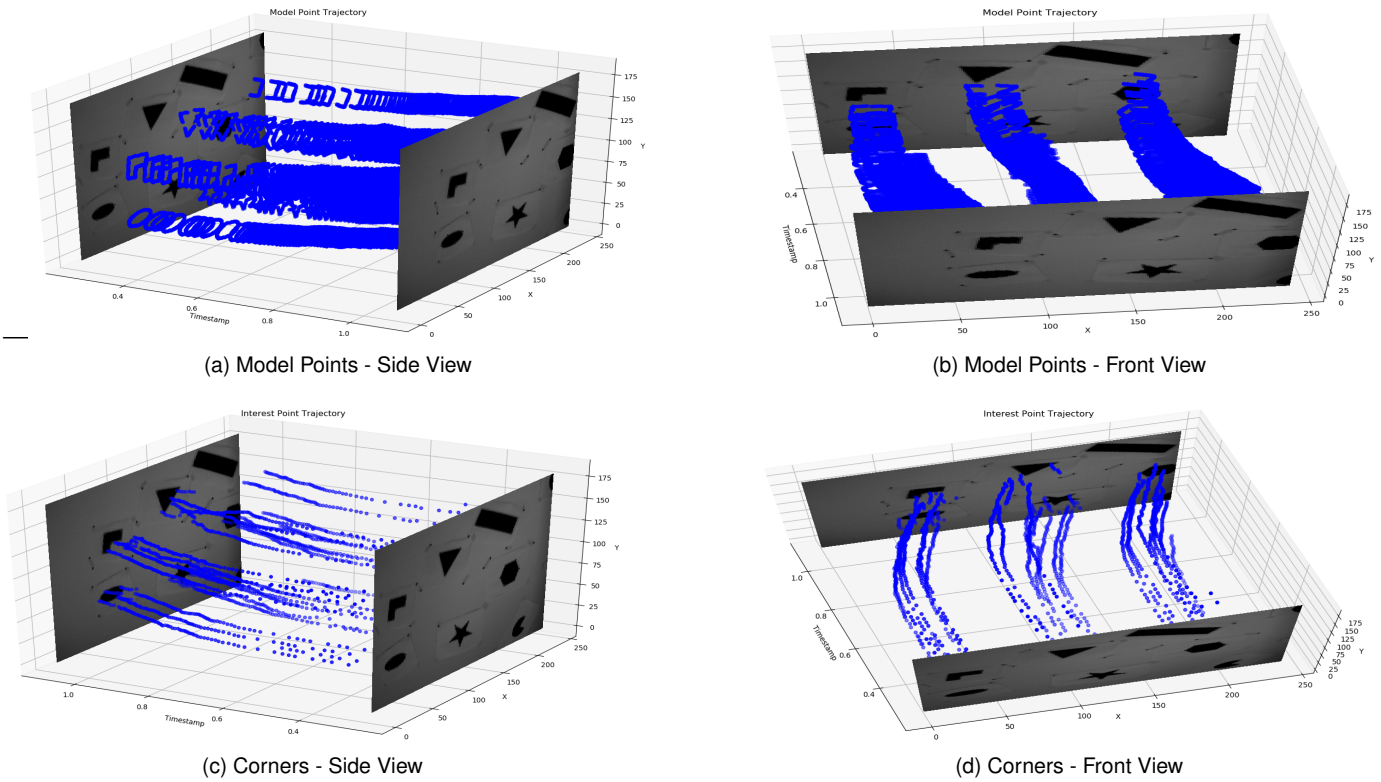
(b) ArUco Detection during exploration of Map 2



(c) ArUco Re-Detection before Manipulation (Pick) in Map 3

Fig. 16: ArUco Detection alongside Exploration, Localization, Mapping, and Manipulation.

One notable difference between detection in simulation and in the real world is the latency in the camera used. In the simulation, the camera topic publishes at a rate of 30Hz; however, the camera on the real robot has a publishing rate of about 1.7Hz. This constituted high levels of latency that hampered the ease of implementation in the joint autonomous task. To mitigate this, a special behavior was designed to keep the ArUco detection running in parallel (see the Detect_ArUco behavior in Figure 15) while tasks like exploration, localization, mapping, and manipulation from the other hands-on project were being executed. The final behavior tree of the joint autonomous task is found in Figure 15. The behaviors that integrate the perception hands-on are encircled with pink boxes, those of hands-on



(a) Model Points - Side View (b) Model Points - Front View

(c) Corners - Side View (d) Corners - Front View

Fig. 13: Trajectory of model points and corners between Two Intensity Image Frames.

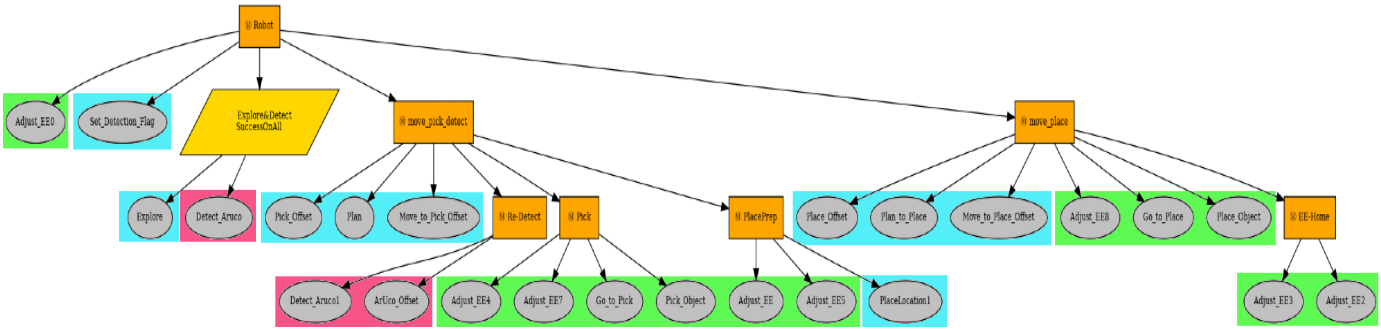


Fig. 15: Behavior Tree for the Joint Autonomous Task

intervention are encircled with green boxes, and those of hands-on planning are encircled with blue boxes. Throughout the autonomous task, pose-based ICP SLAM is used, so it is not included as a behavior.

5 Conclusion and Future Work

This research paper evaluated the effectiveness of a methodology for event-based feature tracking using the Iterative Closest Point (ICP) algorithm. The study addressed the computational challenges associated with high-frequency and large-volume event data by adopting a grouped approach to convert events into image frames. The Shi-Tomasi feature detector was utilized for corner detection, while the Canny edge detector was employed for edge generation. The evaluation of the methodology involved extensive experimentation and analysis. The results demonstrated the robustness and accuracy of the ICP algorithm in tracking features in event-based data. Additionally, a feature reinitialization process was introduced to handle situations where the number of features fell below a minimum threshold. The evaluation showcased the viability of the method, contributing to the advancement of real-time computer vision applications.

Future research in event-based feature tracking with ICP can explore alternative feature representations, such as tracking model points in overlapping patches and monitoring resulting transformations to prevent erroneous results. Additionally, implementing an adaptive binning strategy that adjusts spatial resolution based on motion characteristics and feature density holds promise for improving efficiency and accuracy. These advancements have the potential to significantly enhance the performance and robustness of event-based feature tracking systems, expanding their applicability in various real-time computer vision scenarios.

ACKNOWLEDGMENTS

The authors express their gratitude to supervisors Dr. Estrela Gracias, Nuno Ricardo, and Jad Mansour for their invaluable support throughout this research project.

REFERENCES

[1] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, "Low-latency visual odometry using event-based feature tracks," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 16-23, 2016. doi: 10.1109/IROS.2016.7758089

[2] Mueggler, E. et al. (2017) 'The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam', *The International Journal of Robotics Research*, 36(2), pp. 142–149. doi:10.1177/0278364917691115.

[3] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based feature tracking with probabilistic data association," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4465-4470.

[4] A. Mitrokhin, C. Ferm"uller, C. Parameshwara, and Y. Aloimonos, "Event-based moving object detection and tracking," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1-9.

[5] M. Iacono, S. Weber, A. Glover, and C. Bartolozzi, "Towards event-driven object detection with off-the-shelf deep learning," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1-9, 2018.

[6] S. Afshar, A. P. Nicholson, A. van Schaik, and G. Cohen, "Event-based object detection and tracking for space situational awareness," *IEEE Sensors Journal*, vol. 20, no. 24, pp. 15117-15132, 2020.

[7] H. Chen, D. Suter, Q. Wu, and H. Wang, "End-to-end learning of object motion estimation from retinal events for event-based object tracking," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 10534-10541, 2020.

APPENDIX

The project execution results, in the form of videos, are available for access through the provided links:

- **Link for results video**: Result folder link