Real-time Grasping using Event-driven Camera

Università di Genova

Hocine DELALA

DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering

University of Genova

Supervisors:
Prof. Carmine Tommaso Recchiuto

In partial fulfillment of the requirements for the degree of

Laurea Magistrale in Robotics Engineering

October 14, 2025

I would like to dedicate this thesis to my family; without them, I would not be where I am today. To my mother, my first and most faithful supporter; to my father, whose trust and quiet support made every goal feel within reach; and to my grandparents, aunts, and uncles, whose prayers, wisdom, and constant encouragement sustained me.

A special dedication goes to my twin brother Hassan, whose belief in me and steady support kept me moving forward.

I also dedicate this work to my friends for their support and good wishes—especially to Baba, who supported me throughout this thesis and the journey.

Declaration of Originality

I, Hocine DELALA, hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Carmine Recchiuto, for his guidance, patience, and insightful feedback throughout this work. His expertise and encouragement were essential at every stage of the thesis.

I am also grateful to the Robotics Engineering faculty and staff at the University of Genova, and to my colleagues and friends in the lab, for their support, discussions, and willingness to help.

My heartfelt thanks go to my family and friends for their unconditional support and belief in me. I also wish to acknowledge all the teachers who have taught me—from elementary school through my university studies—for laying the foundations that made this journey possible.

Finally, I am grateful to all those who contributed, directly or indirectly, to the completion of this thesis.

Abstract

This thesis presents a real-time robotic grasping pipeline using an eye-in-hand event-based vision system for industrial pick-and-place on conveyor lines, targeting both stopped (static) and moving (dynamic) items. Event-based cameras provide asynchronous measurements with microsecond-level latency and high dynamic range, making them well suited to low-latency manipulation. An initial evaluation of event-based Multi-View Stereo (EMVS) for 3D reconstruction revealed strong sensitivity to pose accuracy and parameter tuning. In response, a lightweight perception pipeline requiring only object bounding dimensions (rather than full CAD models) was developed that operates directly on the 2D event stream and avoids dense reconstruction.

The method clusters events in the image plane using DBSCAN, fits a rotated rectangle, and validates it via a size-consistency check using known top-face dimensions and object height. The validated centroid is then back-projected to 3D pose in the robot base frame using calibrated intrinsics and the robot's TF chain. This geometric prior approach is suited to industrial pick-and-place scenarios where object dimensions are catalogued but detailed geometry varies. Implemented in ROS, the system supports two actuation modes: (i) search-then-grasp for stationary objects; and (ii) track-then-predictive-grasp for moving objects via Position-Based Visual Servoing (PBVS) with a velocity estimation followed by a predictive intercept.

Experiments on a Kinova Gen3 demonstrated centroid accuracy ranging from 3.39 mm to 8.85 mm, achieving 5.31 mm mean error using only natural edges. Grasp success rates were 80-100% in static scenarios. Dynamic experiments were conducted as proof-of-concept using manually moved objects, confirming tracking ability but revealing reduced grasp accuracy due to unstable object speed. The results indicate that event-driven, reconstruction-free perception is a practical, modular solution for both static and dynamic scenarios, though further validation with controlled conveyor speeds and computational benchmarking is required for production deployment.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

Robotic systems have become essential in modern industry and daily life, enhancing automation, precision, and efficiency. Within this broader landscape, robotic grasping is a cornerstone of manipulation and autonomy, underpinning applications in manufacturing, logistics, service robotics, and assistive technologies. The ability to reliably detect, track, and grasp objects in real time is critical for operation in static and dynamic settings, such as factory lines with conveyor-borne parts.

Traditional vision pipelines for grasping rely on frame-based cameras, which capture images at fixed frame rates and produce dense but redundant data. While effective in static scenes, these systems struggle in fast or unpredictable scenarios due to motion blur, limited dynamic range, and high latency.

Event-driven cameras, also known as neuromorphic or dynamic vision sensors, offer a compelling alternative. Rather than capturing frames, they asynchronously report per-pixel brightness changes, yielding sparse data streams with microsecond resolution. This sensing modality provides low latency, robustness to motion blur, high dynamic range, and greatly reduced data redundancy, making event-based vision particularly well suited to robotic grasping tasks where speed and precision are paramount.

1.2 Motivation

Although event-driven cameras are increasingly applied to perception tasks such as visual odometry and 3D reconstruction, their integration into robotic grasping pipelines remains comparatively under-explored. A representative state-of-the-art pipeline reconstructs a metric scene representation and then derives grasp

poses from it. In practice, this often means reconstructing depth via Event-based Multi-View Stereo (EMVS) [29] and using the resulting map to localize the target and plan a grasp. However, this strategy is highly sensitive to pose estimation errors, requires careful parameter tuning, and involves heavy computation. This challenge motivated the development of lightweight, event-driven methods that can estimate graspable features from asynchronous event data without full scene reconstruction. For industrial applications where object dimensions are known a priori from part catalogues, such methods can exploit simple geometric priors to achieve robust detection without the complexity of full 3D modeling. In particular, a pipeline that can reliably detect object centroids and orientations, stabilize these estimates over time, and integrate seamlessly with robotic control frameworks would enable practical real-time grasping in both stationary and conveyor-like scenarios.

The work presented in this thesis addresses this by designing and experimentally validating a modular perception and control pipeline based on event clustering. The system is tested on a Kinova Gen3 robotic arm with a Robotiq 2F-85 gripper and a DAVIS346 event camera, demonstrating reliable performance in grasping both static and moving objects.

1.3 Thesis Outline

This thesis is organized as follows:

- **Chapter 2 – State of the Art:** reviews event-driven cameras, their advantages over frame-based sensors; explores relevant concepts, techniques, and existing research in event-based perception, clustering, and robotic grasping, it also presents a comparison between existing approaches and our proposed approach. Visual servoing techniques are also introduced as a foundation for the proposed dynamic grasping application.

- **Chapter 3 – Hardware and Software Setup:** describes the experimental platform used in this thesis, including the DAVIS346 event camera, the Kinova Gen3 robotic arm with Robotiq gripper, and the ROS-based software framework used for calibration, control, and motion planning. This chapter also explains how these software and hardware components are related to each other, specifically the DAVIS346 camera and the gripper.

- **Chapter 4 – Methodology:** describes the proposed grasping pipeline, covering both perception and control. It begins by analyzing the limitations of EMVS-based 3D reconstruction, then introduces a clustering approach for centroid and in-plane orientation estimation. Finally, it details

the techniques for detecting, tracking, and grasping objects in both static and moving scenarios.

- **Chapter 5 – Experimental Results:** provides quantitative evaluations of the proposed system in static scenarios; for dynamic grasping, it reports a proof-of-concept demonstration. The static-object experiments assess accuracy, robustness to variations in object texture and orientation, and success rates. The chapter concludes with a summary and discussion of the results.
- **Chapter 6 – Conclusions and Future Work:** summarizes the contributions and key findings of the thesis, discusses observed limitations, and suggests directions for future research in event-driven robotic grasping.

Chapter 2

State of the Art

This chapter reviews the state of the art in object grasping using event-driven cameras. Event-driven cameras are introduced, highlighting their benefits for real-time applications. The chapter first covers the principles of event-driven sensors and the core differences between them and normal frame-based cameras. Additionally, we discuss some event-based perception techniques (3D reconstruction, clustering, etc.). Finally, we discuss visual servoing and grasping techniques using event cameras.

2.1 Event-Based Vision: Principles and Sensors

Frame-based image sensors are widely used in machine vision, but they suffer from high power consumption and a high processing latency. The event-driven dynamic vision sensor (DVS) [23] reduces redundant data for post-processing by only reporting changes in scene reflectance with latencies and temporal precision down to microseconds. These features, along with the wide dynamic range (100 dB) of the DVS, make these sensors particularly suitable for robotics and real-time tracking, where they can reduce the system-level power consumption by a factor of 100 compared to conventional image sensors [2].

Event-based vision sensors represent a fundamental shift from traditional frame-based acquisition. Rather than capturing entire images at fixed intervals, these sensors detect changes in intensity at each pixel independently, resulting in a sparse asynchronous data stream. This approach not only reduces redundancy but also provides key advantages such as high dynamic range, low latency, and microsecond-level temporal resolution. These devices are inspired by biological retinas and form the core of neuromorphic engineering in vision systems [10].

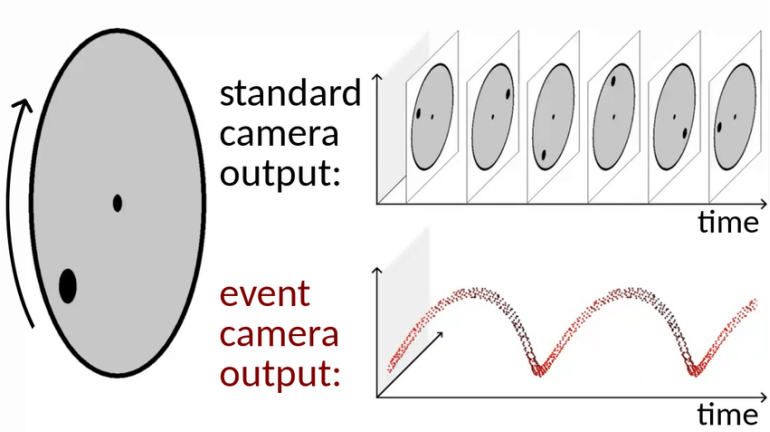


Figure 2.1: Comparison of the output of a standard frame-based camera and an event camera when facing a black dot on a rotating disk [11].

2.1.1 Event-Based vs. Frame-Based Cameras

Event cameras offer numerous potential advantages over standard cameras, and this can be summarized in the following points: Event cameras differ from conventional frame cameras in that they are bio-inspired sensors. Instead of capturing images at a fixed rate, they asynchronously measure per-pixel brightness changes, and output a stream of events that encode the time, location, and sign of the brightness changes [10].

Event-based cameras offer a significantly higher temporal resolution compared to their frame-based counterparts. This is because events are detected and timestamped with microsecond precision, a direct result of the fast, asynchronous analog circuitry. This inherent speed allows event cameras to capture very fast motions without the motion blur that is typical of traditional frame-based cameras [10]. This capability is particularly crucial for robotic applications, such as high-speed grasping, where precise and rapid sensing is required.

Another key advantage is their exceptionally high dynamic range (HDR), which often exceeds 120 dB, far surpassing the 60 dB of most high-quality frame-based cameras. This allows event cameras to operate effectively in challenging lighting conditions, from very dark environments to scenes with direct sunlight. This is made possible because each pixel's photoreceptor operates on a logarithmic scale and independently adapts to the light stimulus, similar to how a biological retina functions [10].

Furthermore, event-based sensors exhibit minimal latency. Since there is no need to wait for a global exposure time or an entire frame to be read out, a change in brightness is transmitted almost instantaneously. In ideal lab conditions, latency can be as low as 10 μs, with real-world latency typically in the

sub-millisecond range. This near-zero latency is a critical feature for reactive systems where timely decision-making is essential.

Finally, event cameras are highly power-efficient. By transmitting only the data corresponding to the brightness changes, they eliminate the energy consumed by processing redundant information. On the die level, most cameras use only about 10 mW, and entire embedded systems that include both the sensor and a processor have been shown to consume less than 100 mW [10]. This makes them ideal for power-constrained applications, such as battery-powered mobile robots and drones.

2.1.2 Overview of Event Camera Sensors

Several commercial event cameras are available, including the one utilized in this thesis. They are still research prototypes, although most of them may be obtained through commercial goods or research partnerships. There exist events-only cameras as well as hybrid cameras, which output both frames and events.

2.1.2.1 Events-only Cameras

Event-only vision sensors, such as the DVS128 [25], only output asynchronous event streams based on intensity changes, offering significant advantages in terms of low latency and high dynamic range (up to 120 dB) [10]. These sensors are ideal for applications such as robotic motion detection or gesture recognition, where high temporal resolution is required [25]. However, the absence of intensity frames limits their use in traditional image-based tasks without additional data processing. DVS cameras (e.g., SciDVS) [14] are highly sensitive and are often used in research where rapid scene changes are the focus.

Another example of a modern DVS-only camera is the iniVation DVXplorer series [19]. The DVXplorer Mini is a mass-produced DVS sensor with a VGA (640×480) resolution. It is manufactured using 90 nm BSI CIS technology. The camera is designed to produce a high-throughput event output of up to 450 million events per second (MEPS). It has a sub-millisecond latency and a temporal resolution of 200 μs. The camera offers a dynamic range of up to approximately 110 dB. A notable feature is that it does not output intensity frames; however, similar images can be reconstructed from the event data using the provided DV software.

2.1.2.2 Hybrid Cameras

Hybrid vision sensors (HVS) integrate asynchronous event detection with a frame-based synchronous imager into a single sensor. This dual functionality is designed

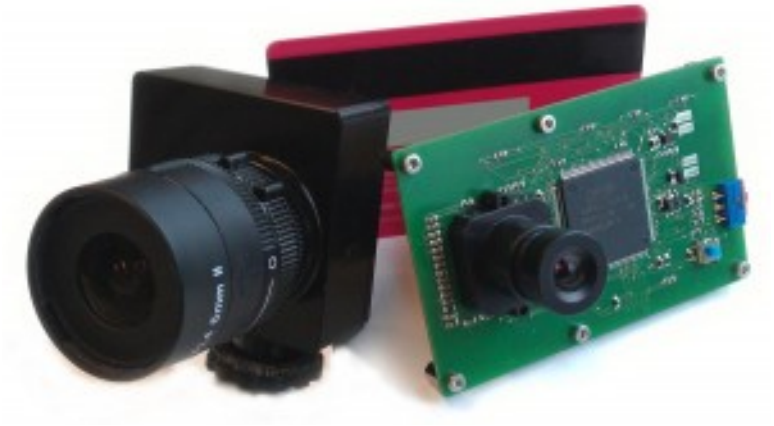


Figure 2.2: DVS and DAVIS – Dynamic Vision Sensors. [18].

to leverage the benefits of both sensing modalities: the high temporal resolution and low-latency output of event-based systems, along with the absolute intensity information from traditional frame-based cameras. This combination proves especially advantageous for tasks such as object recognition, tracking, and classification [10].

A prominent example of this technology is the DAVIS (Dynamic and Active-pixel Vision Sensor), introduced by Brandli et al. [2]. The DAVIS integrates a Dynamic Vision Sensor (DVS) with an Active Pixel Sensor (APS) on the same pixel array, enabling simultaneous output of events and grayscale frames. This unified architecture allows the sensor to operate in hybrid mode, supporting both asynchronous and synchronous data acquisition.

One widely used model in this family is the DAVIS346, developed by iniVation [19]. It provides a resolution of 346×260 pixels for both the DVS and APS outputs. The DVS component achieves a dynamic range of up to 120 dB and a minimum latency of approximately 20 μs. The APS offers a dynamic range of around 56.7 dB and supports configurable shutter modes, including global and rolling shutter. Additionally, the sensor includes a 6-axis IMU and communicates via USB 3.0, making it suitable for robotic perception applications [33].

In this work, I used the DAVIS346 camera primarily for event-based perception, particularly for object clustering and robotic grasping tasks. A deliberate design choice was made to rely exclusively on the event data output from the DVS. Although the DAVIS346 camera also provides grayscale frames, these were used only for overlay visualization during debugging and analysis. By avoiding any dependency on the APS frames, the proposed architecture remains lightweight and more broadly applicable, ensuring compatibility with both hybrid and event-only

cameras. This makes the system flexible and transferable to other event-driven sensors that do not provide intensity images. The DAVIS346 camera is fully compatible with the Robot Operating System (ROS), allowing easy integration with existing robotic frameworks thanks to the ROS driver [30], which facilitates streaming of event data from the camera into the ROS ecosystem for further processing.

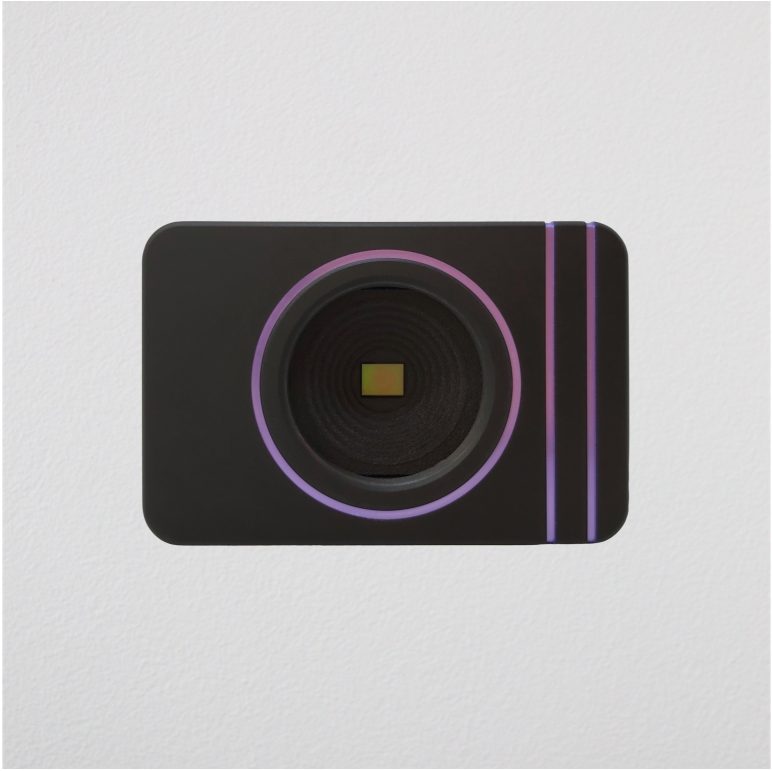


Figure 2.3: DVXplorer camera manufactured by Inivation. [19].

2.2 Event-based Perception Methods

In this section, we present a comprehensive review of the state-of-the-art methods for processing and interpreting the asynchronous event streams generated by event-cameras.

Event-based cameras, such as the DAVIS346 [19], generate asynchronous streams of pixel-level brightness changes, or *events*, rather than conventional frames. Each event $e_i = (x_i, y_i, t_i, p_i)$ encodes the pixel coordinates (x_i, y_i), timestamp t_i, and polarity p_i, indicating a brightness increase or decrease [25]. This

paradigm allows for high temporal resolution, low-latency sensing, and robustness to motion blur, making it particularly suitable for real-time robotic perception.

In our work, only the DVS events were utilized for object clustering and detection. Grayscale frames were used only for overlay visualization and debugging purposes. Initially, we explored 3D reconstruction techniques such as EMVS, but practical limitations led us to adopt a simpler 2D clustering approach based on DBSCAN.

2.2.1 3D reconstruction approaches (EMVS, voxel-based methods)

Event-based 3D reconstruction aims to recover scene geometry from sparse and asynchronous events. Traditional frame-based stereo methods rely on dense pixel correspondences, whereas event-based methods accumulate information across time and viewpoints.

EMVS (Event-based Multi-View Stereo) Event-based Multi-View Stereo (EMVS), introduced by Rebecq et al. [29], is a real-time 3D reconstruction algorithm that leverages the asynchronous stream of events produced by dynamic vision sensors (DVS). Unlike traditional stereo systems, EMVS accumulates events across time as a camera moves, without relying on synchronized frames.

The key concept is the construction of a **Disparity Space Image (DSI)**, a 3D volume that encodes the likelihood of a point in space corresponding to a 3D surface.

Each event is defined as:

$$e_i = (x_i, y_i, t_i, p_i), \tag{2.1}$$

where (x_i, y_i) are the pixel coordinates, t_i is the timestamp, and $p_i \in \{-1, 1\}$ is the polarity.

Projection Model For a given candidate depth z and known camera pose T_k at time t_k, a pixel (x_i, y_i) defines a 3D point X_i^z along the viewing ray:

$$X_i^z = \pi^{-1}(x_i, y_i, z), \tag{2.2}$$

where π^{-1} is the back-projection using camera intrinsics.

This 3D point is then projected into all past views using:

$$(x'_{ik}, y'_{ik}) = \pi(T_k^{-1} X_i^z), \tag{2.3}$$

where T_k is the camera pose and π is the projection function.

DSI Accumulation The Disparity Space Image is defined as:

$$DSI(x, y, z) = \sum_{k=1}^{K} \delta(I_k(\pi(T_k^{-1} X^z)) - p), \tag{2.4}$$

where I_k is the event image at time t_k, p is the event polarity, and δ is a contrast measure. High values of DSI indicate that a surface exists at depth z along the viewing ray of (x, y).

Depth Map Estimation The depth at each pixel (x, y) is computed by maximizing the DSI across depths:

$$\hat{z}(x, y) = \arg\max_{z} DSI(x, y, z) \tag{2.5}$$

Reconstruction Output The output is a semi-dense point cloud formed by projecting $\hat{z}(x, y)$ into 3D space. The result is a depth map or 3D point cloud that can be used for further tasks like visual odometry or mapping.

Limitations Observed in This Work Although EMVS is elegant and works in real-time, it requires:

- Accurate camera poses from a visual odometry or SLAM system,
- A sufficiently dense and noise-free stream of events,
- Structured camera motion (e.g., lateral parallax) to generate disparity.

In our experiments, the method was tested using the DAVIS346 event camera and poses from a visual odometry system. However, due to noise, sparse events, and non-ideal camera motion, the DSI was too noisy to extract reliable depth maps. Consequently, EMVS was not used in the final pipeline.

2.2.2 Event-based visual odometry (EVO, UltimateSLAM, etc.)

Event-based visual odometry (VO) aims to estimate a camera's 6-DoF motion using the asynchronous stream of events produced by neuromorphic sensors. Unlike conventional frame-based VO systems that suffer from motion blur and latency, event-based VO leverages the high temporal resolution and dynamic range of event cameras. One of the most prominent contributions in this domain is EVO [28], a purely geometric approach to real-time, parallel tracking and mapping using only events.

EVO consists of two interleaved modules: a pose tracking module and a mapping module. The tracking module aligns a live event image, constructed by aggregating recent events, against a projection of the current semi-dense 3D map M, using an edge-based inverse compositional Lucas-Kanade (IC-LK) method [15]. The objective function minimized during the tracking is the following:

$$E(T) = \sum_{\mathbf{u}} \left[M(W(\mathbf{u}; \Delta T)) - I(W(\mathbf{u}; T)) \right]^2 , \tag{2.6}$$

where $W(\mathbf{u}; T)$ is a 3D rigid-body warp applied to pixel $\mathbf{u}$ using transformation T, and the sum is over all pixels $\mathbf{u}$ with known depth $d_{\mathbf{u}}$. The warp is defined by the following.

$$W(\mathbf{u}; T) := \pi \left(T \cdot \pi^{-1}(\mathbf{u}, d_{\mathbf{u}}) \right) , \tag{2.7}$$

where π and π^{-1} denote the camera projection and its inverse. The transformation T is parametrized in the Lie group SE(3) using exponential coordinates.

Simultaneously, the mapping module uses EMVS [29] to generate a semi-dense 3D map. It back-projects events using the estimated camera poses and accumulates them into a voxel grid called the Disparity Space Image (DSI), where each voxel stores a ray density. The peaks in this volume indicate probable 3D edge locations and are extracted to form a point cloud.

This parallel pipeline allows EVO to operate in real-time on a standard CPU, outputting hundreds of pose estimates per second. Notably, EVO does not require grayscale frames or intensity reconstruction, avoiding error propagation and reducing computational complexity. The system has demonstrated high accuracy and robustness in challenging scenes, including aggressive motions and extreme lighting variations.

2.2.3 Event clustering and segmentation (DBSCAN, K-Means, spatio-temporal clustering)

Event-based sensors produce sparse, asynchronous spatio-temporal streams, unlike conventional cameras that generate dense image frames. In order to extract high-level understanding from these event clouds—such as identifying and localizing objects—grouping the raw events into coherent spatial structures is essential. This process, known as event clustering, separates informative foreground activity from background noise.

Centroid-Based Clustering (e.g., K-Means) K-Means is a widely used partitioning algorithm that clusters data into a predefined number of groups k by minimizing intra-cluster variance [26]. It works by initializing k centroids,

assigning each data point to the nearest centroid, and then updating centroids as the mean of the assigned points. The process repeats until convergence. However, K-Means presents key limitations when applied to event-based vision:

- **Predefined Cluster Count (k):** In robotic scenarios, the number of objects or motion patterns is often unknown and time-varying. K-Means requires k to be known in advance, which is impractical.
- **Assumption of Cluster Shape:** K-Means implicitly assumes clusters are convex and isotropic (spherical), which does not align with the irregular and elongated shapes of event clusters produced by real-world motion.

Spatio-Temporal Clustering via Time-Slicing Event data inherently possess spatio-temporal structure (x, y, t, p), making clustering directly in this multi-dimensional space valuable for capturing object trajectories and motion dynamics. A notable approach is the Hierarchy of Time Surfaces (HOTS), which builds multi-layer time-decaying representations and extracts spatio-temporal features for classification [22]. Despite its effectiveness, HOTS and similar time-surface methods can be computationally demanding.

Alternatively, some works employ event clustering for object tracking—such as the method developed by Barranco et al., which performs real-time clustering for multi-target tracking using only event data [1].

DBSCAN: Density-Based Clustering The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [7] overcomes K-Means' limitations and is particularly well-suited for event segmentation.

DBSCAN defines clusters as regions of high event density separated by areas of low density. Each point is classified based on two parameters:

- ε: radius of the neighborhood (proximity threshold)
- `minPts`: minimum number of points within ε to form a cluster

Using these, each event e_i is classified as:

- A **core point** if its ε-neighborhood contains at least `minPts` events.
- A **border point** if it is within ε of a core point but does not meet the `minPts` threshold itself.
- **Noise** otherwise.

The Euclidean distance is defined as:

$$d(e_i, e_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{2.8}$$

In some approaches, a temporal term is added:

$$d_{spatio-temporal}(e_i, e_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + \alpha^2(t_i - t_j)^2}, \tag{2.9}$$

where α balances the time and spatial dimensions.

Our Implementation In this project, we used DBSCAN in two stages:

- In the 2D image plane: events from the last $T = 100$ms are clustered to detect the object.
- In the 3D world: after projecting clustered events using camera intrinsics and fixed depth, DBSCAN is re-applied to filter noisy points.

We used the `scikit-learn` implementation in Python:

```
clustering = DBSCAN(eps=0.03, min_samples=20).fit(xyz)
labels = clustering.labels_
```

The centroid of the largest valid cluster was computed as:

$$(x_c, y_c) = \frac{1}{N} \sum_{i=1}^{N} (x_i, y_i), \quad \text{for events in cluster} \tag{2.10}$$

This centroid was projected to 3D using intrinsic parameters and a fixed Z-depth. The 3D point was then transformed to the robot base frame via TF and used for tracking and grasping the object.

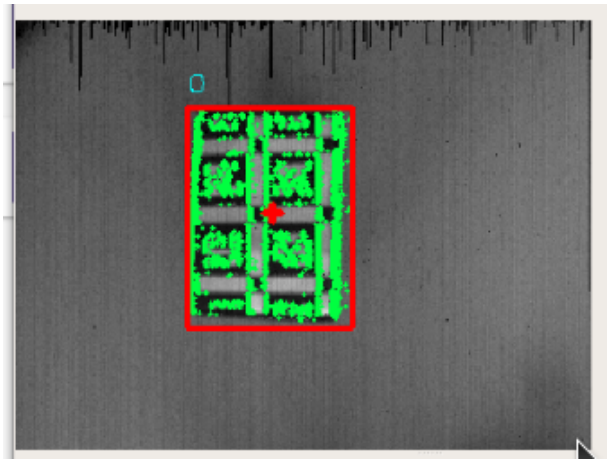


Figure 2.4: Example of DBSCAN clustering output.

2.3 Grasping Using Event-Based Cameras

Robotic grasping is a cornerstone of manipulation, fundamentally relying on the accurate and low-latency estimation of an object's 6-DoF pose. Traditional approaches using frame-based cameras can be effective in static environments but often fail when objects are in motion, suffering from motion blur that corrupts visual information and degrades pose accuracy. Furthermore, their fixed frame rates impose a significant latency limit on the perception-action loop [16]. Event-based cameras offer a compelling alternative for dynamic grasping scenarios. Their high temporal resolution (microsecond-level), high dynamic range (HDR), and sparse data output make them naturally robust to motion blur and challenging lighting conditions. This has motivated two broad families of event-driven grasping pipelines.

2.3.1 Model-Based Approaches (MBA)

These methods assume a known 3D CAD model of the target object. The core challenge is to align this model with the incoming event data. Events, which naturally highlight edges and contours, can be accumulated into point clouds or edge-like feature maps. Pose estimation is then framed as a registration problem, aligning the model's projected silhouette with the observed event features. For instance, Kim et al. [20] proposed a method for real-time 3D reconstruction and 6-DoF tracking using only event data. Similarly, Li and Stueckler [24] introduced a probabilistic framework that fuses events and frames to accurately estimate 6-DoF object motion. More recently, EDOPT [12] demonstrated stereo event-based dynamic object pose tracking using a hybrid registration scheme. Motion execution typically uses Position-Based Visual Servoing (PBVS) since a full Cartesian pose is available. These approaches are highly accurate but are inherently limited to known object geometries because they require offline object modeling and registration. [16].

2.3.2 Model-Free Approaches (MFA)

Distinct from model-based pipelines that align a known CAD shape to event-derived features, the model-free approach estimates graspable poses directly from the event stream without an object model.

In Huang *et al.*, the pipeline contains three main stages: (i) event-domain instance segmentation, (ii) robust centroid/orientation extraction with depth support, and (iii) velocity-based visual servoing and grasp execution [16].

(1) Event-domain instance segmentation (MEMS). The authors propose a Multiple-object Event-based Mean-Shift (MEMS) algorithm that clusters events jointly in space and time (x, y, t) by ascending the mean-shift vector of a spatio–temporal kernel density estimate. Practically, each event iteratively "shifts" along the local mean-shift direction until convergence to a mode, producing per-object segments without any prior on object size or shape. To meet real-time constraints, MEMS is accelerated via (i) an *acceleration* term α that extrapolates the step along the mean-shift direction (with an empirically safe regime around $\alpha \approx 0.35$), and (ii) a *uniform down-sampling* factor β that reduces event load while improving robustness to noise. With $\beta > 4$, the reported per-event processing approaches $\sim 1\,\mu$s; an E-score improvement of at least $\sim 16\%$ is already observed at $\beta = 2$ (balancing speed and F1) [16].

(2) Robust centroid and depth support. After instance segmentation, a robust 2D centroid and in-plane orientation are computed using a stacked set of *surfaces of active events* and corner evidence: events are mapped to SAE/SAFE/SALE/SAVE layers (raw event timestamps, e-Harris event-corners, corner-concentration heatmaps, and a stabilized "virtual" centroid). This yields a centroid/orientation estimate that is less sensitive to transient noise than a naive mean of raw events. Metric depth is required for 3D alignment; the authors acquire and *freeze* a scene depth map via EMVS during an initial short motion, then reuse this depth (object-wise or per-cluster) while tracking and grasping—thus the model-free pipeline relies on depth but not on a CAD model [16].

(3) Event-based Visual Servoing (EVS) and grasp. Control is velocity-based visual servoing (rather than PBVS): the gripper first eliminates the *position* error to the robust centroid and then removes the *orientation* error, using the depth-supported centroid/orientation as feedback. After alignment, a simple grasp policy is executed (BarrettHand in the paper). The end-to-end framework (Algorithm 2 in [16]) follows: initial motion $\rightarrow$ EMVS + MEMS to cache depth and object centroids; then, for each object ID, run MEMS segmentation and robust centroiding, perform EVS to null position/orientation errors, grasp, and iterate [16].

Acceptance/rejection logic. Crucially, candidate objects are *not* accepted or rejected using any geometric size prior (e.g., pixel width/height). Instead, acceptance is driven by event-domain consistency: clusters must appear as coherent density modes of the spatio–temporal KDE (MEMS convergence) and be stabilized by corner-based heatmap support for centroid quality; clusters that fail these consistency tests are rejected [16].

2.3.3 Positioning Our Approach within the Model-Free Grasping Literature

Object Priors and Method Classification:
In the sense of Huang *et al.* [16], model-free methods require no object-specific 3D models or CAD templates. Our approach occupies a distinct category: we require lightweight geometric priors—specifically, the object's top-face dimensions (length, width) and height—but not full 3D shape models. We term this a geometric prior approach to distinguish it from both fully model-free methods (which assume no shape knowledge) and model-based methods (which require complete CAD geometry). This design choice targets industrial conveyor scenarios where object bounding dimensions are known in advance (e.g., in warehouse management systems) but detailed geometry varies across production batches or packaging types. Concretely, we cluster events in the image plane, fit a rotated bounding rectangle to each cluster, and then accept or reject clusters via a size-consistency check based on the known top-face dimensions and the current camera–object offset. This retains the spirit of model-free grasping (no CAD), while using minimal priors to make our clustering decisive and robust at low latency.

Depth without 3D reconstruction:
Compared with Huang *et al.* [16], who reconstruct depth maps via EMVS [29], our approach does not estimate depth from vision. Instead, we assume a known object height z_{obj} and use the robot's kinematic transform $T_{\text{base}\leftarrow\text{cam}}$ (TF) to compute the camera–object distance along the optical axis:

$$Z \;=\; \big(T_{\text{base}\leftarrow\text{cam}}\big)_{3,4} \;-\; z_{obj}.$$

This Z is then used both to (i) predict the expected pixel footprint of the rectangle (for accepting/rejecting clusters) and (ii) back-project the accepted centroid to 3D (see Sec. 4.2). The consequence is a much simpler, lower-latency perception module with no heavy reconstruction, only calibrated intrinsics and an accurate TF chain. The trade-off is that z_{obj} must be known (or bounded) and the top surface should be approximately planar relative to the camera.

Implications and scope:
This design choice makes our pipeline simpler and faster than methods that require dense or semi-dense 3D recovery: we only need $(\mathbf{K}, T_{\text{base}\leftarrow\text{cam}}, z_{obj})$ to deliver a metric centroid and yaw for further processing (PBVS). In exchange, we assume approximate knowledge of top-face dimensions and object height, which tailors

the method to box-like items or other shapes whose top projection can be approximated by a rectangle. Generalizing to arbitrary geometries (e.g., spheres or unknown heights) would require replacing the size-consistency test with a shape-agnostic acceptance criterion and introducing an alternative source of metric depth.

2.4 Visual Servoing in Robotics

Visual Servoing (VS) denotes the family of feedback controllers that use visual measurements to regulate a robot to a desired pose. Let $s \in \mathbb{R}^m$ be a vector of visual features and $s^\star$ their desired values. The control objective is to drive the error $e = s - s^\star \to 0$ by commanding the camera (or end-effector) spatial velocity $\mathbf{v}_c \in \mathbb{R}^6$ (linear and angular velocities). Visual Servoing systems are broadly categorized into two main classes [3, 17]. Figure 2.5 illustrates these two Visual Servoing control scheme; Position-Based VS and Image-Based VS.

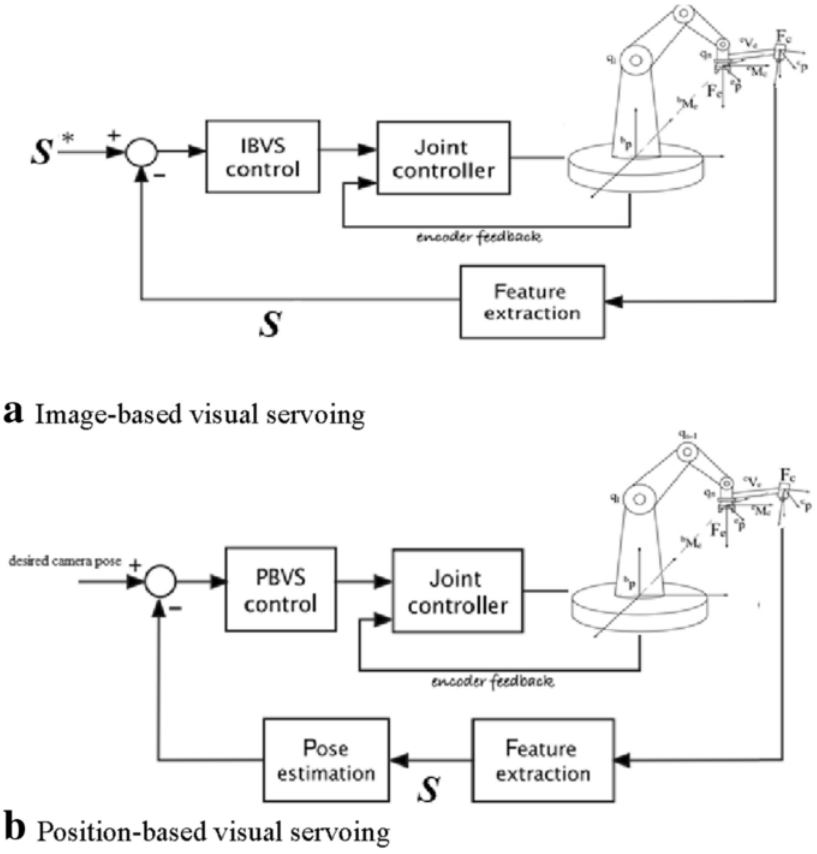


Figure 2.5: PBVS and IBVS control schemes [5].

2.4.1 Image-Based Visual Servoing (IBVS)

In IBVS, the features s are defined in the *image plane* (e.g., point coordinates, line parameters, centroid, moments). The image kinematics relate the feature time derivative to the camera twist through the *interaction matrix* (also called image Jacobian) $L_s(s, Z)$:

$$\dot{s} \;=\; L_s(s, Z)\, \mathbf{v}_c.$$

For example, for a point with normalized image coordinates (x, y) and depth Z, one has the well-known interaction matrix

$$L_{\text{point}} \;=\; \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix},$$

possibly composed for multiple features and scaled by camera intrinsics [3]. A standard IBVS control law is

$$\mathbf{v}_c \;=\; -\lambda\, L_s^{+}(s, Z)\, e,$$

with $\lambda > 0$ a gain and L_s^{+} the (damped) pseudoinverse.

This approach is generally robust to camera calibration errors, as the control loop operates directly in the sensor's coordinate system. However, it can lead to unpredictable Cartesian trajectories for the robot's end-effector and may suffer from singularities or camera retreat issues where the camera moves away from the target to satisfy image constraints[3, 17].

2.4.2 Position-Based Visual Servoing (PBVS)

PBVS first estimates the *3D pose* of the target, $T_{c\leftarrow o} \in \mathrm{SE}(3)$ (object w.r.t. camera), from visual measurements; the error is then defined in Cartesian space. If $T^{\star}_{c\leftarrow o}$ is the desired pose, a common choice is the body-frame error

$$T_e \;=\; (T_{c\leftarrow o})^{-1}\, T^{\star}_{c\leftarrow o}, \quad \text{with } \mathbf{t}_e \in \mathbb{R}^3,\ \boldsymbol{\omega}_e \in \mathbb{R}^3 \text{ from } \log(T_e).$$

The control is then

$$\mathbf{v}_c \;=\; -\lambda \begin{bmatrix} \mathbf{t}_e \\ \boldsymbol{\omega}_e \end{bmatrix},$$

optionally with decoupled gains/saturation on translation vs. rotation [3, 17].

Traditional Visual Servoing systems are limited by the frame rate of the camera, which introduces latency and limits the control loop's bandwidth. Event cameras, by providing continuous visual feedback with microsecond latency, enable control loops to run at much higher frequencies (kHz range), allowing robots to track and interact with highly dynamic targets that would be impossible with standard cameras [16].

In this thesis we adopt a PBVS approach: the perception system delivers a metric 3D target centroid from events, and a PD/PBVS controller commands the end-effector velocities to null the Cartesian error allowing an accurate tracking.

Chapter 3

Hardware and Software Configuration and Utilization

In this chapter, we present a detailed overview of the software and hardware configurations that were essential for implementing and validating our proposed approach. The goal is to provide a clear understanding of the tools, frameworks, and setup procedures used throughout the development and experimentation phases of this project.

The chapter is structured into two main sections: (1) Software Tool, where we describe the core software components that supported perception, calibration, and robot control. This includes Kalibr for camera calibration, MoveIt for robot motion planning, and the ROS framework that enabled real-time communication and coordinate transformations between system components. (2) Hardware Tools, which outlines the physical devices used in the setup. We introduce the DAVIS346 event-based camera, the Kinova Gen3 robotic arm, and finally we describe the integrated camera-robot configuration designed for real-time object detection, grasping and visual servoing tasks.

3.1 Software Tools

In this section we present the software tools used in our implementation. These tools include the calibration software Kalibr, MoveIt for planning and executing robot motion, and ROS for inter-module communication.

3.1.1 Kalibr

Kalibr is a state-of-the-art calibration toolbox designed for accurate estimation of camera intrinsics, camera-IMU extrinsics, and temporal offsets between sensors.

Developed by the Autonomous Systems Lab at ETH Zurich, it supports multi-camera and multi-sensor calibration scenarios and is widely used in robotics and computer vision applications [9].

In this project, Kalibr was employed to calibrate the DAVIS346 event-based camera. Although the DVS sensor itself does not output standard frames, its grayscale APS (Active Pixel Sensor) output was utilized to extract corner features for calibration. A standard aprilGrid pattern was used to estimate the camera's intrinsic parameters, including focal lengths, principal point, and distortion coefficients. These parameters were then used in the perception pipeline to undistort events and project them accurately into 3D space.

The resulting calibration data ensured consistency between the event-based measurements and the rest of the system, particularly for back-projecting clustered event centroids and transforming them into the robot's base frame. The GitHub repository for the Kalibr toolbox is available at [8].

3.1.2 MoveIt

MoveIt is a powerful, open-source motion planning framework designed for the Robot Operating System (ROS), widely adopted in the field of robotic manipulation. It provides an integrated set of tools for motion planning, kinematics (FK/IK), collision checking, and trajectory execution [4].

Built to reduce the barrier to entry in robotics software development, MoveIt combines modular components such as the OMPL motion planners, collision detection via FCL, and state representation through the planning scene interface. Additionally, the MoveIt Task Constructor extension enabled hierarchical task modeling, allowing the execution of complex behaviors such as pick-and-place by composing sequences of motion primitives [13].

In our project, MoveIt was used to control the Kinova Gen3 robotic arm for both arm trajectory planning and gripper actuation (opening/closing), fully integrated within a ROS-based architecture. The framework's high-level abstraction and ROS compatibility allowed seamless integration with other perception and control modules. Its flexibility and transparent interface made it an ideal backbone for our manipulation pipeline.

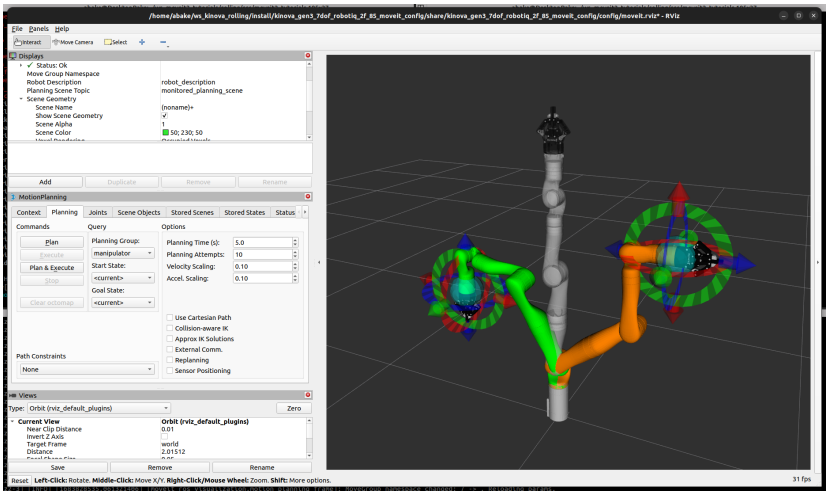


Figure 3.1: MoveIt RViz motion planning interface controlling the Kinova Gen3 robot [6].

3.1.3 Robot Operating System (ROS)

The Robot Operating System (ROS) is a flexible framework for writing robot software. It provides essential tools and libraries for developing, building, and debugging robotic applications in a modular and distributed manner. ROS follows a publish/subscribe architecture, where nodes communicate through topics, services, and actions, enabling real-time inter-process communication in robotic systems [27].

In this work, ROS Kinetic was used as the middleware for integrating all robotic components. Due to compatibility constraints with other tools and packages, the entire ROS stack was executed within a Docker container. This approach ensured an isolated, reproducible, and platform-independent development environment. The container was configured to access USB devices and GPU resources to enable real-time interaction with the DAVIS346 event-based camera and the Kinova Gen3 robotic arm.

ROS was responsible for interfacing with camera and robot drivers, broadcasting transformations (TF), executing motion plans via MoveIt, and controlling the robotic gripper and arm through dedicated nodes. It served as the central communication backbone throughout the grasping pipeline.

3.2 Hardware Tools

In this section, we present the Hardware tools used in our project. We describe the perception unit (DAVIS346 camera) and the manipulation unit (Kinova Gen3

and robotiq gripper), followed by the integrated camera–robot setup.

3.2.1 DAVIS346 Camera

In this project, the DAVIS346 camera was used exclusively as an event-based perception sensor to detect and localize objects for robotic grasping. Although the sensor provides both asynchronous events and grayscale frames, only the event stream was utilized in the perception pipeline. The grayscale frames were accessed selectively for overlay visualization during debugging and evaluation.

The camera was connected and streamed into the ROS ecosystem using the open-source 'rpg dvs ros' driver [30], within a Docker container running ROS Kinetic. This configuration allowed seamless integration with other ROS nodes responsible for clustering, transform broadcasting, and motion planning. The DAVIS346's USB 3.0 interface enabled high-throughput event capture without noticeable delays, and its compact size made it easy to mount in an eye-in-hand configuration on the Kinova Gen3 manipulator.

Figure 3.2: DAVIS346 camera manufactured by Inivation. [19].

The camera's high temporal resolution and low-latency event output made it particularly suited for detecting fast scene changes and enabling real-time clustering of event data. These properties are especially advantageous in dynamic robotic environments where conventional frame-based methods suffer from motion blur or high latency.

3.2.2 Kinova Gen3 Robot

The Kinova Gen3 is a lightweight, modular robotic arm developed for advanced manipulation tasks in research and industrial applications. In this project, the 6-DoF version of the Gen3 was used, which offers a balance between reach, payload, and responsiveness. The arm has a maximum reach of approximately 891 mm and a continuous payload capacity of up to 2 kg across its full range of motion. It weighs 7.2 kg, supports infinite actuator joint rotation, and can be powered with a 24 V DC supply. The maximum end-effector speed is approximately 50 cm/s, and the system is rated for operation in ambient temperatures between -30 °C and 35 °C [31].

For user interaction, the Gen3 includes a physical joystick for manual control and supports Ethernet connectivity, allowing seamless integration into local networks for real-time programming, monitoring, or remote diagnostics.

The robot is equipped with the Robotiq 2F-85 adaptive gripper [32], a parallel two-finger gripper designed for precision and compliant grasping. It supports position, velocity, and effort control and is well suited for research scenarios involving lightweight object manipulation.

Figure 3.3: Kinova Gen3 robotic arm equipped with the 2F adaptive gripper [31].

To control the Gen3 from ROS, we used the official Kinova Kortex ROS driver, available through the ros kortex package [21]. This driver provides full access to

the robot's low- and high-level functionalities, including arm movement, gripper actuation, and status monitoring. The driver publishes joint states, exposes ROS services and action servers for motion execution, and includes URDF and MoveIt configuration files for simulation and planning.

Additionally, the Kinova Web App was used to visualize and test robot behavior before full integration. The Web App allows users to manually control the robot, lock/unlock joints, and monitor the robot state directly via a browser-based interface hosted on the robot's internal web server.

3.3 Camera and Robot Setup

In our project, the **DAVIS346 event camera** was mounted on the Kinova Gen3 robotic arm in an eye-in-hand configuration, meaning the camera is physically attached to the robot's end-effector and moves synchronously with it during manipulation tasks. The sensor was affixed to the gripper using a high-strength double-sided adhesive, ensuring mechanical stability throughout motion.

To ensure accurate perception and spatial alignment within the ROS framework, a dedicated coordinate frame named `dvx_camera_link` was defined and attached to the robot's `ee_link` (end-effector frame). The transformation between the camera and the robot was measured manually, including both the translation and the rotation components. The camera's Z-axis was oriented to point forward along the gripper's approach direction, while the X and Y axes were calibrated accordingly using visual feedback in RViz.

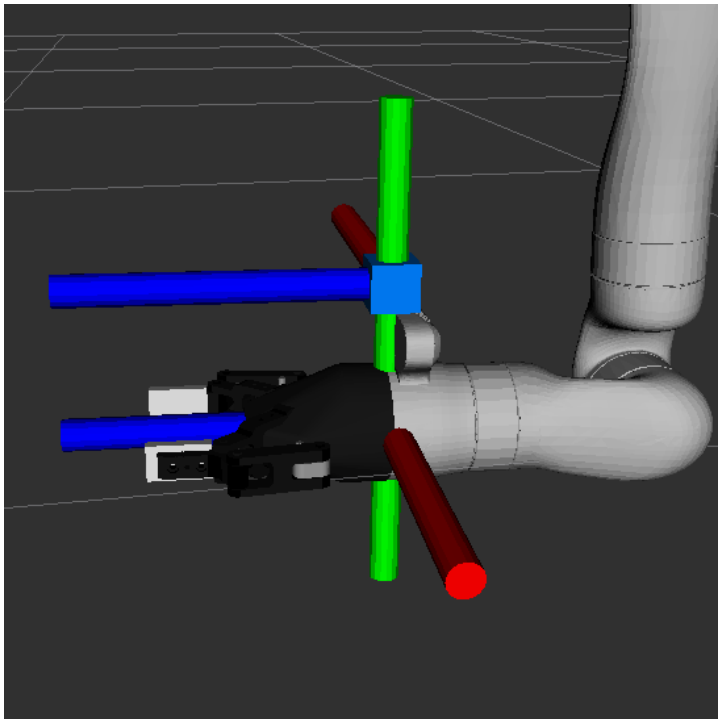


Figure 3.4: Visualization of the End-Effector to Camera Transformation in RViz. The blue cube represents the DAVIS346 camera, and the colored axes (red: X, green: Y, blue: Z) represent its coordinate frame.

This transformation was encoded in the robot's `URDF/XACRO` description using a static `<joint>` and corresponding `<link>`, enabling continuous TF broadcasting between the robot and camera frames. This integration allowed seamless visualization and processing of the camera data within the robot's kinematic tree, facilitating tasks such as event-based clustering, 3D pose estimation, and visual servoing. Additionally, it allowed the camera's position to be expressed relative to the base frame (`base_link`), enabling accurate transformation of perception outputs into the robot's task space.

Figure 3.5: DAVIS346 event camera mounted on the Kinova Gen3 gripper using a fixed eye-in-hand configuration.

Chapter 4

Methodology

This chapter presents the core methodology developed for event-based robotic grasping. It details the perception algorithms, control techniques, and system architecture designed to translate the sparse, asynchronous data from an event camera into successful robotic manipulation.

The narrative of this chapter follows the project's developmental trajectory. We begin by documenting our initial investigation into 3D scene reconstruction using the EMVS algorithm, as introduced in Chapter 2. We then analyze the limitations of this approach, which necessitated a pivot towards a more direct and computationally efficient 2D clustering method for object detection. The remainder of the chapter is dedicated to this successful pipeline, detailing its application to two distinct grasping scenarios: first, the grasping of a static object, and second, the more complex task of tracking and grasping a moving object. Our implementation was using ROS kinetic, and our ROS nodes were Python scripts.

4.1 Initial Approach 3D Scene Reconstruction

The design of our methodology was heavily informed by the state-of-the-art in event-based grasping, particularly the framework proposed by Huang et al. in their work on real-time grasping strategies [16]. They identified two primary pathways for event-based perception: a model-based approach and a model-free approach. The model-based strategy, which we investigated first, relies on building an explicit 3D model of the scene from multiple viewpoints. The conceptual workflow involves using an algorithm to generate a 3D point cloud, followed by point cloud processing techniques (e.g., clustering, registration) to identify and locate the target object.

Inspired by this model-based paradigm, our initial approach was to use the Event-based Multi-View Stereo (EMVS) algorithm [29] to perform the 3D re-

construction. The plan was to generate a semi-dense point cloud of the scene and then apply a 3D clustering algorithm to segment the object and determine its pose. This approach was chosen for its potential to provide a rich, geometric understanding of the environment, but as we will demonstrate, it presented significant practical challenges.

4.1.1 Offline Experiments with EMVS

To validate this approach, we conducted a series of offline experiments. The first step was to create a high-quality dataset by recording a rosbag file during a controlled robot motion. This dataset was designed to provide the EMVS algorithm with all its required inputs:

- **Events (/dvs/events):** The raw event stream from the DAVIS346 camera, capturing changes in brightness as the robot's end-effector moved relative to the scene. The events were published at a rate of 30 Hz.
- **Camera Intrinsics (/dvs/camera_info):** The calibrated camera matrix and distortion parameters, essential for the projection operations within the EMVS pipeline.
- **Camera Pose (/dvs/pose):** A high-frequency (100 Hz) stream of the camera's 6-DOF pose in the robot's base_link frame. This was obtained by listening to the /tf topic, which provided the static transformation from the robot's end-effector to the rigidly mounted camera as explained in 3.3.

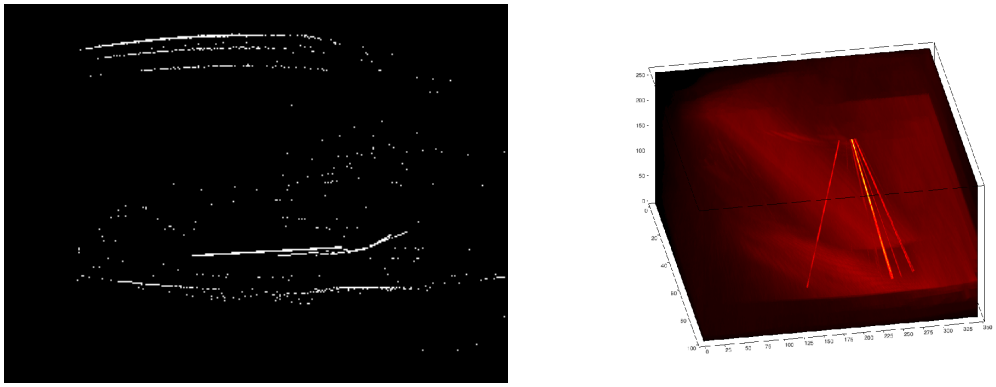
Following the setup instructions provided in the official EMVS repository [34], I processed our recorded rosbag data. The algorithm's performance is highly dependent on its numerous parameters, and I performed extensive tuning. In the following, I present two representative results from this tuning process.

The first attempt used a broad depth range suitable for general scene reconstruction. The parameters were set as follows:

- –min-depth=0.3, –max-depth=2.0
- –dimZ=100 (Number of depth planes)
- –adaptive-threshold-c=2.0

The output, shown in Figure 4.1, was not as expected. The resulting semi-dense mask (a); which is a binary image that highlights the pixels in the depth map where reliable depth estimates have been computed from sufficient event data and pose accuracy, as seen in the figure, it was extremely sparse, containing only

a few disconnected points. The corresponding Disparity Space Image (DSI) (b), which should ideally show a clear surface, is dominated by noise, indicating that the algorithm could not find a consistent geometric structure.



(a) Semidense mask.

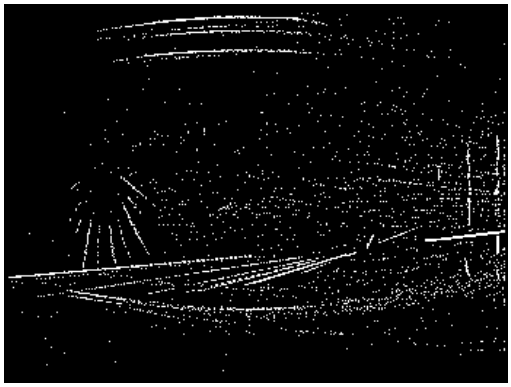
(b) Disparity Space Image (DSI).

Figure 4.1: Offline EMVS results with a broad depth range. The reconstruction is sparse and unusable.

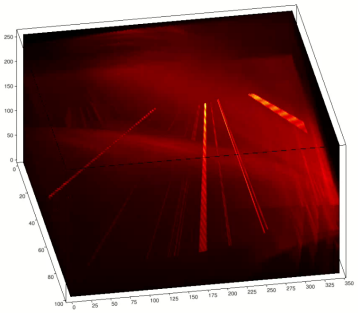
For the second attempt, we significantly constrained the search space to the known location of the object, providing the algorithm with a strong prior.

- –min-depth=0.1, –max-depth=0.5
- –dimX=346, –dimY=260, –dimZ=100
- –adaptive-threshold-c=2.0

Despite these favorable conditions, the results shown in Figure 4.2 were still inadequate. While the semi-dense mask (a) contains more points than the previous attempt, they remain too sparse and fragmented to form a coherent object shape. The DSI (b) likewise fails to converge on a distinct surface.



(a) Semidense mask.



(b) Disparity Space Image (DSI).

Figure 4.2: Offline EMVS results with a constrained depth range. The reconstruction quality remains insufficient for object segmentation.

4.1.2 Limitations and Rationale for Pivot

The experimental results consistently demonstrated that the 3D reconstruction approach was not viable for our application, due to several fundamental limitations:

1. **Extreme Parameter Sensitivity:** As shown, the quality of the reconstruction is highly sensitive to a large number of parameters (e.g., depth range, number of disparity planes, adaptive thresholds). Finding a robust set of parameters that worked across different scenarios proved to be exceptionally difficult and time-consuming.

2. **Requirement for Precise Pose Data:** The EMVS algorithm relies on highly accurate camera pose estimates to integrate information over time. Any small errors or jitter in the provided trajectory can lead to a failure in reconstruction, as seen in our results.

3. **High Computational Cost:** The algorithm is computationally intensive, requiring significant processing time even for offline-recorded data on our development machine. This characteristic makes it unsuitable for a real-time robotic application where low-latency perception is critical.

Given these challenges, it became clear that pursuing a full 3D reconstruction was both impractical and unnecessary for our application. The poor quality of the point clouds meant that any subsequent clustering step would be unreliable. Consequently, we revised the methodology and decided to forgo explicit 3D reconstruction and instead develop a method that operates directly on the 2D event

stream. This new approach, detailed in the following section, aims to leverage the high temporal resolution of the event camera for direct object detection, and proves to be more robust and computationally efficient.

Beyond these algorithmic limitations, the targeted application of this thesis is industrial-style pick-and-place where the object lies on a planar support at a known height (e.g., a tabletop or conveyor belt) and the camera is rigidly mounted to the end-effector (eye-in-hand). In this setting, the distance between the camera and the object surface along the optical axis (with the camera optical axis approximately aligned with the base z-axis) can be obtained without performing dense 3D reconstruction, by exploiting the known (or bounded) object height:

$$Z \;=\; \big(T_{\text{base}\leftarrow\text{cam}}\big)_{3,4} \;-\; z_{obj},$$

with z_{obj} the (known or bounded) object height and $T_{\text{base}\leftarrow\text{cam}}$ provided by the calibrated TF chain. This known Z suffices to (i) predict the expected pixel footprint of a candidate cluster and accept/reject it via a simple size-consistency check, and (ii) back-project the accepted centroid to metric 3D for control. Under these operating constraints, a lightweight 2D event-clustering approach empirically delivers lower latency, fewer failure modes, and better real-time behavior than the tested 3D reconstruction approach, while meeting the accuracy required for grasping. Hence, we adopt the direct 2D pipeline detailed in the following sections.

4.2 Core Perception Pipeline: 2D Clustering and Pose Estimation

The core of our perception pipeline was a robust, two-stage clustering algorithm designed to estimate the 3D pose $\big(x, y, z, \theta\big)$ (position and orientation) of a target object from the sparse, asynchronous data stream of an event camera. This algorithm was fundamental to both the static object grasping and moving-object grasping approaches, providing a consistent method for transforming raw event data into actionable robotic commands. The process was divided into two main stages: (1) single-frame 2D event clustering and pose hypothesis generation, and (2) temporal 3D pose filtering and stabilization.

In the first stage, events accumulated over a short time window are clustered in the 2D image plane using the DBSCAN algorithm. This identifies spatially proximate groups of events that may correspond to a moving object or edge. For each significant cluster, a rotated rectangle is fitted to approximate the object's

shape. This rectangle is then validated by comparing its dimensions to the expected pixel-space dimensions of the known object, which are calculated based on the object's physical size and its estimated depth from the camera. This size-based filtering is crucial for rejecting spurious clusters caused by noise or other objects in the scene. For each accepted 2D cluster, we fit a rotated rectangle and use its pixel center (u, v) and the longest-edge direction to recover a metric pose. The distance between the camera and the object plane is obtained from the eye–in–hand TF and the known object height z_{obj}:

$$Z \;=\; \big(T_{\text{base}\leftarrow\text{cam}}\big)_{3,4} \;-\; z_{obj}.$$

Given the camera intrinsics $\mathbf{K}$, the 2D center is back-projected to the camera frame and then to the robot base frame

$$[X_c, Y_c, Z]^\top = Z\,\mathbf{K}^{-1}[u, v, 1]^\top, \qquad [x, y, z, 1]^\top = T_{\text{base}\leftarrow\text{cam}}\,[X_c, Y_c, Z, 1]^\top.$$

The yaw θ is obtained by back-projecting the endpoints of the rectangle's longest edge and applying $\theta = \text{atan2}((\mathbf{p}_2 - \mathbf{p}_1)_y,\ (\mathbf{p}_2 - \mathbf{p}_1)_x)$ in the base XY-plane. Full algorithmic details follow bellow (Alg. 1).

This stage outputs a single, albeit potentially noisy, 3D pose estimate for that moment in time. All 2D→3D mapping (back-projection with $\mathbf{K}$ and $T_{\text{base}\leftarrow\text{cam}}$ and the computation of Z) is performed *entirely in Stage 1.* Stage 2 operates only on the resulting stream of metric poses $\{(x, y, z, \theta)_t\}$ and does not reprocess events or image data.

In the second stage, these individual pose estimates, generated continuously over time, are filtered to produce a stable and accurate result. A history of the most recent pose estimates is maintained. The DBSCAN algorithm is applied again, but this time in the 3D Cartesian space of the pose history. This step identifies the most consistent and dense group of pose estimates, effectively filtering out transient outliers. The final stabilized object pose is then calculated as the arithmetic mean of the positions and orientations of the poses belonging to this dominant 3D cluster. This filtered pose is then used either to command a final grasp or to provide continuous feedback to a visual servoing controller for the moving-object grasping approach.

The method was used as the fundamental perception block for both static and dynamic object grasping tasks, with variations in its execution policy.

4.2.1 Stage 1: 2D Event Segmentation and 3D Back-Projection (Pose Hypothesis)

The first stage is responsible for generating a raw but complete 3D pose hypothesis from a single slice of recent events. The logic is summarized in Algorithm 1.

All geometric operations use the TF transformation chain between the base frame of the robot `base_link` and the camera frame `dvx_camera_link` and the calibrated intrinsics, $\mathbf{K}$, of the DAVIS346 camera. The algorithm's behavior is governed by several key ROS parameters, for instance: the event accumulation time window, `time_window` (Δt); the 2D clustering parameters, `cluster_eps` (px) and `min_samples`; the known physical dimensions of the object, `object_dims` = $[\ell, w, z_{obj}]$ (m); and the 3D temporal filtering parameters, ε_{3D} (m) and $\texttt{minPts}_{3D}$.

The following is an explanation of the workflow of this stage of the approach: At a fixed rate, the node extracts the most recent time slice of events.

$$\mathcal{E}_{\Delta t} = \{\, e_i = (x_i, y_i, t_i, p_i) \mid t_{\text{now}} - t_i \leq \Delta t \,\}.$$

If $|\mathcal{E}_{\Delta t}| < \texttt{min\_recent\_events}$, the cycle is skipped to avoid degenerate fits. The image points $\mathcal{P} = \{(x_i, y_i)\}_{e_i \in \mathcal{E}_{\Delta t}}$ are clustered by DBSCAN in pixel space with ($\varepsilon = \texttt{cluster\_eps}, \texttt{minPts} = \texttt{min\_samples}$), yielding clusters $\{\mathcal{C}_k\}$ and outliers.

For each non-noisy cluster $\mathcal{C}_k$, we fit a rotated rectangle $\mathcal{R}_k = ((u, v), (u_{px}, h_{px}), \phi)$ via `cv::minAreaRect`. Given the known physical dimensions of the object (length and width) (ℓ, w) and a depth prior to that

$$Z = t_z - z_{obj},$$

where t_z is the camera's z translation in `base_link`(i.e., the camera height above the base) and z_{obj} is the known object height. The resulting Z represents the depth (distance) from the camera lens to the surface of the object in `base_link`.

we predict the expected long/short pixel extents using the pinhole model (with $f = \frac{f_x + f_y}{2}$):

$$\widehat{L} = \frac{f\ \max(\ell, w)}{Z}, \qquad \widehat{S} = \frac{f\ \min(\ell, w)}{Z}.$$

Let $L \geq S$ be the sorted pair $\{u_{px}, h_{px}\}$. A size-consistency score filters spurious clusters:

$$\text{err} = \sqrt{\left(\frac{L - \widehat{L}}{\widehat{L}}\right)^2 + \left(\frac{S - \widehat{S}}{\widehat{S}}\right)^2},$$

and the best cluster is the one with the smallest error below a tolerance. If no cluster passes, the cycle yields no pose estimated.

For the accepted rectangle, the 2D centroid (u, v) is back-projected to 3D using the following transformation:

$$\begin{bmatrix} X_c \\ Y_c \\ Z \end{bmatrix} = Z\,\mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T_{\text{base}\leftarrow\text{camera}} \begin{bmatrix} X_c \\ Y_c \\ Z \\ 1 \end{bmatrix}.$$

The yaw θ that represents the orientation of the object found is computed from the longest rectangle edge: Let (u_1, v_1) and (u_2, v_2) be adjacent corner pixels forming the long edge; back-project both to the base frame to obtain $\mathbf{p}_1 = (x_1, y_1, \cdot)$ and $\mathbf{p}_2 = (x_2, y_2, \cdot)$, then

$$\theta = \operatorname{atan2}\big(y_2 - y_1,\ x_2 - x_1\big).$$

The outputs at this stage are a potentially noisy object centroid pose estimation (x, y, z, θ) and a diagnostic overlay image (events in green, rectangle in blue) that is used just for visualization.

Algorithm 1 Single-Frame Pose Hypothesis Generation

Require: Events $\mathcal{E}_{\Delta t}$ (2D points), transform $T_{\text{base}\leftarrow\text{cam}}$, intrinsics $\mathbf{K}$, dims $[\ell, w, h]$, tolerance SizeTol
Ensure: Raw pose (x, y, z, θ) and fitted rectangle Rect

$\mathcal{C}_{\text{2D}} \leftarrow \text{DBSCAN}(\mathcal{E}_{\Delta t})$
$Z_{\text{cam}} \leftarrow (T_{\text{base}\leftarrow\text{cam}})_{3,4} - h$ (*camera–object plane distance*)
$(\widehat{L}, \widehat{S}) \leftarrow \text{ProjectToPixelSpace}(\ell, w, \mathbf{K}, Z_{\text{cam}})$
$\text{Best} \leftarrow \varnothing,\ \ \text{Err}_{\min} \leftarrow \infty$
for each cluster $C \in \mathcal{C}_{\text{2D}}$ **do**
  if $|C| < \text{MinPts}$ **then**
    continue
  end if
  $\text{Rect} \leftarrow \text{FitRotatedRectangle}(C)$
  $(w_{px}, h_{px}) \leftarrow \text{GetPixelDimensions}(\text{Rect});\ \ (L, S) \leftarrow \text{sort\_desc}(w_{px}, h_{px})$
  $\text{err} \leftarrow \sqrt{\big(\frac{L-\widehat{L}}{\widehat{L}}\big)^2 + \big(\frac{S-\widehat{S}}{\widehat{S}}\big)^2}$
  if $\text{err} < \text{SizeTol}$ **and** $\text{err} < \text{Err}_{\min}$ **then**
    $\text{Best} \leftarrow (\text{Rect}, Z_{\text{cam}}),\ \text{Err}_{\min} \leftarrow \text{err}$
  end if
end for
if $\text{Best} = \varnothing$ **then**
  return `None`
end if
$(\text{Rect}_*, Z_*) \leftarrow \text{Best}$
$(u, v) \leftarrow \text{Center}(\text{Rect}_*)$
$[X_c, Y_c, Z_*]^\top \leftarrow Z_* \mathbf{K}^{-1}[u, v, 1]^\top$
$[x, y, z]^\top \leftarrow \big(T_{\text{base}\leftarrow\text{cam}}[X_c, Y_c, Z_*, 1]^\top\big)_{1:3}$
$(u_1, v_1), (u_2, v_2) \leftarrow \text{LongestEdgePixels}(\text{Rect}_*)$
$\mathbf{p}_1, \mathbf{p}_2 \leftarrow \text{BackProjectPair}((u_1, v_1), (u_2, v_2), Z_*, \mathbf{K}, T_{\text{base}\leftarrow\text{cam}})$
$\theta \leftarrow \operatorname{atan2}\big((\mathbf{p}_2 - \mathbf{p}_1)_y,\ (\mathbf{p}_2 - \mathbf{p}_1)_x\big)$
return $\big((x, y, z, \theta),\ \text{Rect}_*\big)$

4.2.2 Stage 2: Temporal 3D Filtering and Stabilization

The pose hypotheses generated by Stage 1 are noisy due to event jitter and partial views. Stage 2 is designed to filter these raw estimates over time to produce a single, stable, and accurate pose. This is achieved by maintaining a history of the most recent pose hypotheses, $\mathcal{H} = (x_j, y_j, z_j, \theta_j)$.

The core of this stage is to apply the DBSCAN algorithm again, but this time to the 3D Cartesian positions, (x_j, y_j, z_j), within the history. This identifies the largest and most consistent cluster of pose estimates, effectively rejecting transient outliers. The final stabilized position, $(\bar{x}, \bar{y}, \bar{z})$, is the arithmetic mean of the positions in this dominant 3D cluster. The stabilized yaw, $\bar{\theta}$, is computed using the circular mean of the corresponding angles to correctly handle wrap-around issues. This filtering logic is summarized in Algorithm 2.

The specific strategy for how this stage is executed—i.e., whether it runs once at the end of a batch or continuously on a sliding window—differs between the static and dynamic grasping applications, as will be detailed in Sections 4.3 and 4.4.

Algorithm 2 Temporal 3D Pose Filtering

Input:
  Pose_History: Collection of recent raw 3D poses (x, y, z, θ).
  ε_{3D}: DBSCAN search radius in meters.
  $Min\_Samples_{\text{3D}}$: Minimum poses for a dense cluster.
Output:
  $Pose_{\text{stable}}$: Filtered, stable 3D pose $(\bar{x}, \bar{y}, \bar{z}, \bar{\theta})$.

procedure FilterPoseHistory($Pose\_History, \varepsilon_{\text{3D}}, Min\_Samples_{\text{3D}}$)
  $Positions_{\text{3D}} \leftarrow$ ExtractPositions($Pose\_History$)
  $Clusters_{\text{3D}} \leftarrow$ DBSCAN($Positions_{\text{3D}}$, eps $= \varepsilon_{\text{3D}}$, min_samples $= Min\_Samples_{\text{3D}}$)
  if no clusters found in $Clusters_{\text{3D}}$ **then**
    return None
  end if
  $Largest\_Cluster\_Indices \leftarrow$ FindLargestCluster($Clusters_{\text{3D}}$)
  $Stable\_Poses \leftarrow$ SelectPoses($Pose\_History, Largest\_Cluster\_Indices$)
  $(\bar{x}, \bar{y}, \bar{z}) \leftarrow \frac{1}{|Stable\_Poses|} \sum_{p \in Stable\_Poses} p.\text{position}$
  $\bar{\theta} \leftarrow \text{atan2}\left(\sum_{p \in Stable\_Poses} \sin(p.\theta), \sum_{p \in Stable\_Poses} \cos(p.\theta)\right)$
  $Pose_{\text{stable}} \leftarrow (\bar{x}, \bar{y}, \bar{z}, \bar{\theta})$
  return $Pose_{\text{stable}}$
end procedure

4.3 Application 1: Static Object Grasping Approach

At this stage, our system is able to extract a 3D filtered object centroid in the robot base, here in this section we talk about the actuation of the robot to perform the exploration and the grasping of the object in the scene.

The first practical application of our event-based perception pipeline is the grasping of a static object that rests on a planar support (e.g., a conveyor section temporarily stopped for grasping). We assume the object's top surface is approximately planar and parallel to the support. The object height is known a priori, z_{obj}; hence the distance from the camera optical center to the object plane is $Z = \left(T_{\text{base}\leftarrow\text{cam}}\right)_{3,4} - z_{obj}$, obtained from the calibrated TF chain, so no depth estimation is required.

The robot then performs a short exploratory motion to generate a sufficient number of events from the object's corners. While this exploration is in progress, the clustering node processes the accumulated events and returns a single, precise pose (centroid position and in-plane orientation) of the object. Finally, the robot executes the grasp based on this computed pose.

The whole framework of the proposed static object grasping approach is illustrated in Fig. 4.3.

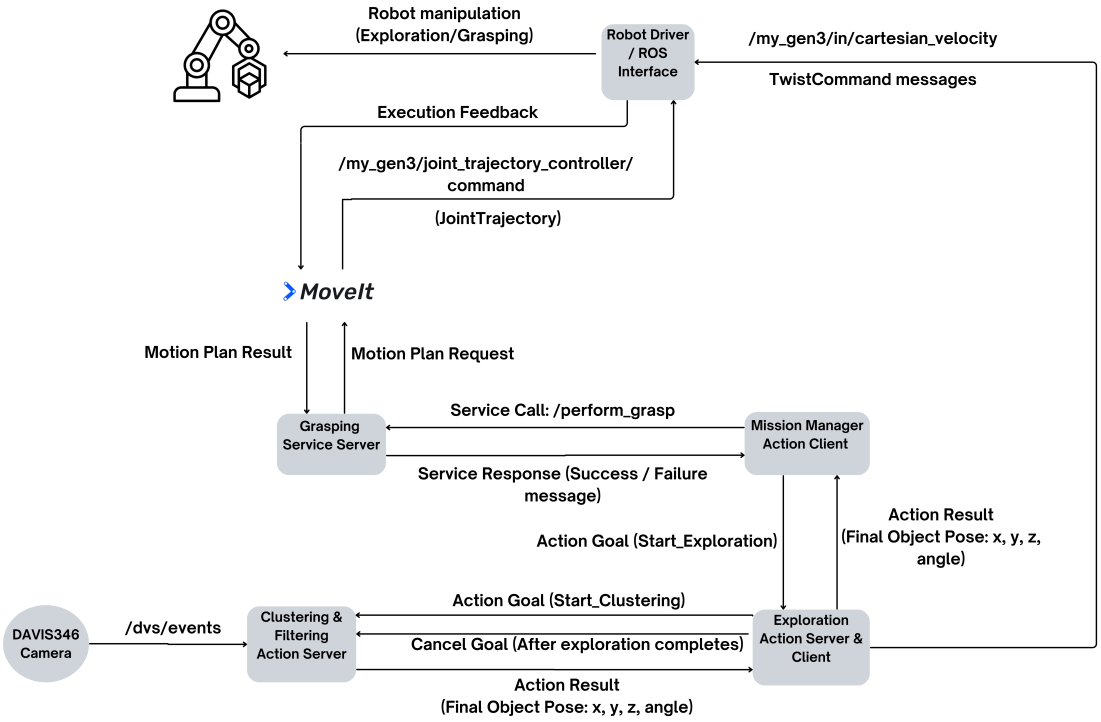


Figure 4.3: The framework for the static object grasping pipeline, illustrating the interaction between the main ROS nodes, MoveIt, and the robot driver.

The static grasping pipeline is composed of four primary ROS nodes: mission manager node, exploration server node, clustering and filtering node, and grasping node. The interaction between these components is primarily managed through ROS Actions and ROS Services.

Mission Manager Node (`mission_manager`) The `mission manager node` serves as the high-level state machine that orchestrates the entire sequence from searching and detecting the object to grasping. It did not perform any perception or motion itself but instead delegates tasks to the other specialized nodes in the correct order. Its primary role was to initiate the exploration, wait for the perception pipeline to return a final object centroid pose, and then command the robot to grasp the object at that pose.

- ROS Interface

- **Action Client**: It is an action client to the `exploration_server`. It initiates the process by sending a goal to this server.

- **Service Client**: It is a client (proxy) to the `/perform_grasp` service. After receiving a successful result from the exploration and clustering phase, it calls this service to execute the physical grasp.

- Workflow Upon startup, the node connects to the `exploration_server` action server and waits for the `/perform_grasp` service to become available. It then sends a goal to the `exploration_server` to begin the concurrent exploration and perception phase. The mission manager then waits for the action of exploration to complete, which signifies that a final object pose has been computed. Upon receiving the result containing the filtered 3D centroid and orientation, it saves this pose into a `GraspObjectRequest` message and calls the `/perform_grasp` service to complete the grasping phase.

Exploration Server (`exploration_node.py`) : The `exploration node` is the first phase of the mission where it is responsible for generating the initial motion of the robot arm required to produce events from the object's corners and features, while simultaneously managing the perception node. Its role is to execute pre-defined movement patterns (e.g., zigzag, square) while triggering the event clustering process to start.

- ROS Interface

- **Action Server**: It provides the `exploration_server` action, which is triggered by a goal from the `mission_manager`.
- **Action Client**: It is an action client to the `cluster_events` action server. When it begins exploration, it immediately sends a goal to the clustering server to start accumulating event data.
- **Publisher**: It publishes `kortex_driver/TwistCommand` messages to `/my_gen3/in/cartesian_velocity` to command the robot's end-effector velocity.

- Workflow The node first waits for a goal from the `mission_manager`. Upon receiving it, the server immediately forwards a goal to the `cluster_events` action server, effectively enabling the perception pipeline. It then begins publishing a sequence of velocity commands to execute a chosen exploratory path. After completing the movement, it stops the robot and cancels the goal sent to the `cluster_events` server. This cancellation tells the perception node to stop collecting data and process all the poses it has gathered. Finally, it waits for the result from `cluster_events` (the final object pose) and passes this result back to the `mission_manager`, completing its action.

Clustering and Filtering node :
The `clustering node` is the core perception node for the static grasping task, implementing the two-stage clustering algorithm. Its role is to listen to event data, detect potential object poses in real-time, and, upon request, filter all collected poses to find the most stable and accurate one. The node implementation was described in details in the previous section 4.2.

- ROS Interface

- **Action Server**: It provides the `cluster_events` action. It is called by the `exploration_server` to start the perception process. Upon completion (preemption), it sends the final computed object pose back to the `exploration_server` as the action's result.
- **Subscriber**: It subscribes to `/dvs/events` to receive the raw event stream from the camera.
- **Publisher**: It publishes `sensor_msgs/Image` messages to `/object/overlay` for visualization and debugging.

- Workflow When its action is activated by the `exploration_server`, the node enters a processing loop. In each iteration, it gathers recent events and performs Stage 1 of the perception pipeline: pose hypothesis generation, as detailed in Section 4.2, Each successfully generated pose is stored in a list. This loop continues as long as the action is active. When the `exploration_server` preempts or cancels the action, the loop terminates. The node then executes Stage 2 of the pipeline (Section 4.2.2), applying 3D DBSCAN clustering to the entire list of collected pose hypotheses to find the most stable cluster. The centroid of this final cluster is calculated and returned as a single, definitive action result. This contrasts with a tracking approach, which would instead publish a continuous stream of filtered poses as feedback.

Grasping Node (`grasping_node_orient.py`) The `grasping_node` is responsible for the final manipulation phase. It interfaces with the robot's motion planner (MoveIt) and gripper controller to execute the physical grasp. Its role is to receive a target object pose and perform a sequence of motions to approach, grasp, lift, and place the object.

- ROS Interface

- **Service Server**: It provides the `/perform_grasp` service of type `GraspObject`. It is called by the `mission_manager` and receives the final object pose as part of the service request.
- **Action Client**: It is an action client for the gripper controller, sending goals to open or close the gripper.
- **MoveIt Interface**: It uses the `MoveGroupCommander` class to plan and execute Cartesian motions for the robot arm.

- Workflow The node waits for a request on the `/perform_grasp` service. Upon receiving a target pose (x, y, z, and angle), it commands the arm through a predefined sequence of waypoints using MoveIt. This sequence includes moving to a "pre-grasp" position above the object that is just an offset in z of the target pose, then adjusting the wrist orientation depending on the object orientation, once the orientation is adjusted, the arm is then ready to go down for grasp, the robot goes down to a fixed z that depends on the object height, the gripper then is closed, and the object is lifted and moved to a release position, and finally the robot returns to the starting position (Home position). Once the sequence is complete, it returns a `GraspObjectResponse` indicating success or failure to the `mission_manager`.

Algorithm 3 Object Grasping and Delivery Sequence

Input: $P_{\text{target}}, \theta_{\text{target}} \leftarrow (x, y, z, \theta)$ Target object centroid position and orientation (yaw angle)
Output: Success or Failure status

Initialize MoveIt Commander and Gripper Action Client
$P_{home} \leftarrow$ GetCurrentArmPose() (*Record the starting pose*)
(***1. Approach***)
$P_{pregrasp} \leftarrow P_{home}$
$P_{pregrasp}.position \leftarrow (P_{target}.x, P_{target}.y, 0.22m)$ (*Set pre-grasp 10cm above grasp height*)
MoveArmTo($P_{pregrasp}$)
(***2. Orient***)
$P_{orient} \leftarrow P_{pregrasp}$
$\theta_{grasp} \leftarrow \theta_{target} + \pi/2$ (*Calculate grasp angle for side grasp*)
$Q_{grasp} \leftarrow$ EulerToQuaternion$(\pi, 0, \theta_{grasp})$
$P_{orient}.orientation \leftarrow Q_{grasp}$
MoveArmTo(P_{orient})
(***3. Grasp***)
$P_{grasp} \leftarrow P_{orient}$
$P_{grasp}.position.z \leftarrow 0.12m$ (*Move down to final grasp height*)
MoveArmTo(P_{grasp})
CloseGripper()
(***4. Deliver***)
$P_{lift} \leftarrow$ GetCurrentArmPose()
$P_{lift}.position.z \leftarrow P_{lift}.position.z + 0.10m$ (*Lift object by 10cm*)
MoveArmTo(P_{lift})
$P_{dropoff} \leftarrow P_{lift}$
$P_{dropoff}.position \leftarrow (P_{target}.x, P_{target}.y - 0.25m, 0.15m)$ (*Move to drop-off location*)
MoveArmTo($P_{dropoff}$)
OpenGripper()
(***5. Return Home***)
$P_{postdrop} \leftarrow$ GetCurrentArmPose()
$P_{postdrop}.position.z \leftarrow P_{postdrop}.position.z + 0.10m$ (*Raise arm after release*)
MoveArmTo($P_{postdrop}$)
MoveArmTo(P_{home})
ShutdownMoveIt()
return Success

Algorithm 3 summarizes the complete grasp planning and execution sequence implemented in the grasping_node. It details the grasping strategy, breaking down the process into five distinct phases: approaching the object, orienting the gripper for a side grasp, executing the grasp, delivering the object to a drop-off location, and returning to a home position. The algorithm illustrates how the target pose and orientation, provided by the perception pipeline, are used to parameterize a series of Cartesian movements.

Object Orientation Estimation In addition to localizing the object's centroid, the pipeline also estimates its orientation (yaw angle) in the robot's base frame, which is crucial for aligning the gripper correctly for a stable grasp. This orientation estimation is performed geometrically using the 2D event cluster identified by DBSCAN. The process, implemented in the clustering logic, follows these steps:

1. A minimum area bounding rectangle is fitted to the 2D event cluster using the OpenCV function 'cv2.minAreaRect'. This function returns the center, dimensions, and rotation angle of the rectangle in the image plane.

2. The algorithm identifies the longest side of this 2D rectangle, which represents the primary axis of the object as seen by the camera.

3. The two endpoints of this longest side are then back-projected from their 2D pixel coordinates into the robot's 3D base frame. This transformation uses the camera's intrinsic matrix and the known object height to find their real-world 3D positions.

4. The 3D yaw angle of the object is calculated by applying the 'atan2' function to the difference in the X and Y coordinates of these two 3D points. This yields a robust estimation of the object's orientation relative to the robot.

5. Finally this angle value is returned as a feedback allong with the centroid and used by the grasping function to perform the grasp considering the object orientation.

4.4 Application 2: Dynamic Object Grasping (Visual Servoing)

The "search-then-grasp" strategy was effective for static objects, but it is not suitable for tasks involving motion. If the object moves, the computed grasp pose becomes invalid, leading to mission failure. To address this, we developed

a second application based on a "Track and Chase" logic, enabling the robot to first track the moving object and once it has a reliable estimate of the object's planar velocity, it performs a predictive grasp.

We assume that the object moves on a conveyor at a fixed, known height z_{obj}. The object's top surface is approximately planar and parallel to the support. With an eye-in-hand camera, the camera–object distance along the optical axis is

$$Z = \left(T_{\text{base}\leftarrow\text{cam}}\right)_{3,4} - z_{obj},$$

so no depth estimation is required.

Unlike the static object grasping, this approach relies on a continuous, real-time feedback loop for the object pose from the perception system(Clustering) streams and a PBVS controller uses this feedback to track the object. A short history of positions is used to estimate the planar velocity; once the estimate stabilizes, the controller predicts an intercept point and executes a downward approach and grasp.

System Architecture and Continuous Feedback The core architectural change from the static pipeline is the modification of the perception node to provide a continuous stream of data rather than a single object pose result. Instead of waiting for an exploration phase to finish, the system immediately begins tracking the object and adjusting the robot's position in real-time. This is achieved through two key components: a modified clustering node for continuous feedback and a grasp manager that implements the servoing and prediction logic.

Clustering for Servoing (`clustering_servoing.py`) To enable real-time tracking, the perception node was adjusted to operate in a continuous feedback mode. The `clustering_servoing.py` node implements this critical change.

- **Role:** To continuously detect the object in the event stream and provide a stable pose estimate at a high frequency.
- **Key Modification:** Unlike the static grasping node which accumulated all poses for a final batch computation, this node uses a sliding window of recent pose hypotheses (a `deque` in the implementation). In every processing cycle, it performs both Stage 1 (new pose hypothesis) and Stage 2 (filtering the sliding window) of the perception pipeline.
- **ROS Interface:**
  - **Action Server (`cluster_events`):** The action is activated once at the beginning of the mission. Instead of returning a single result, it

continuously publishes the latest stable object pose on the action's feedback topic. This is the key mechanism for providing real-time data to the rest of the system.

– **Subscribers/Publishers:** It subscribes to `/dvs/events` and publishes a visualization overlay, same as the static version.

This modification transforms the perception pipeline from a single-shot estimator into a continuous pose estimator, providing the real-time stream of target poses necessary for visual servoing.

4.4.1 Position-Based Visual Servoing (PBVS) for Tracking

With a continuous stream of object poses available, the first challenge is to make the robot follow the moving target. We implement this using a Position-Based Visual Servoing (PBVS) scheme, managed by the `TrackingPredictiveGraspManager` node (e.g., `track_and_chase2.py`). The goal of this initial tracking phase was to keep the moving object within the camera's field of view and to gather a history of its positions for velocity estimation.

The `TrackingPredictiveGraspManager` node acts as the central controller for this task. Upon receiving a start command, it enters a `TRACKING` state. In this state, it subscribes to the feedback topic of the `cluster_events` action server which contains the object centroid pose. For each new centroid received, it calculates the error between the object's position and the center of the end-effector's frame(tool frame). This error is then fed into a Proportional-Derivative (PD) controller that generates velocity commands (`TwistCommand`) to manipulate the robot arm.

The control law for the servoing is defined as:

$$v_{control} = K_p \cdot e + K_d \cdot \dot{e} \tag{4.1}$$

where e is the position error of the object in the tool frame, $\dot{e}$ is the time derivative of the error, and K_p and K_d are the proportional and derivative gains, respectively. This controller commands the robot to move in such a way that the error is minimized, effectively causing the end-effector to follow the object's movements in the XY plane. During this phase, the node stores the history of received object positions and their timestamps in a buffer.

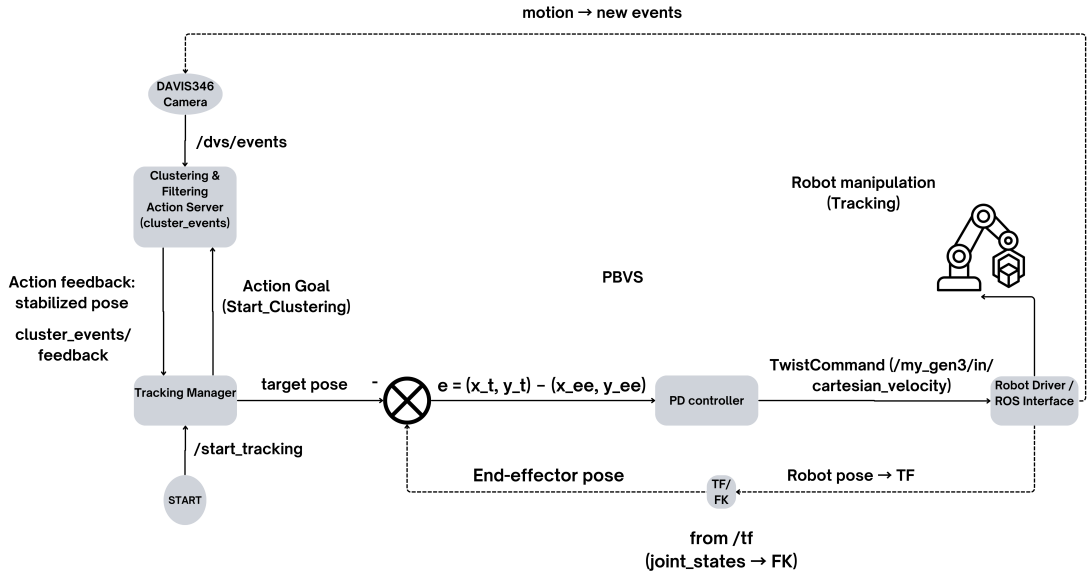


Figure 4.4: The system architecture for real-time object tracking using Position-Based Visual Servoing (PBVS).

The framework of the proposed approach for performing real-time tracking of a dynamic object is illustrated in Fig. 4.4. The architecture is based on a Position-Based Visual Servoing (PBVS) control loop. An event-based perception node provides a continuous stream of the object's target pose. The Tracking Manager then calculates the error between this target and the robot's current end-effector pose. This error is fed to a PD controller that generates velocity commands, closing the loop, and enabling the robot to follow the object in real-time doing a sort of servoing.

4.4.2 Grasping a Moving Target: Prediction and Grasping

Simply servoing to the object's current position is insufficient for a successful grasp due to system latency. By the time the robot's gripper moves to the object's last known location, the object will have already moved. To overcome this, the system must predict the object's future position and aim for that intercept point. This is handled by the `ESTIMATING` and `BLIND_DESCENT` states in the `TrackingPredictiveGraspManager`.

4.4.2.1 Velocity Estimation

Once a sufficient number of position measurements have been collected during the `TRACKING` phase, the system transitions to the `ESTIMATING` state. Here, it uses the stored history of positions and timestamps to compute the object's velocity. We implemented and tested two methods for this:

Linear Regression: A simple and effective method where a first-degree polynomial (`np.polyfit`) is fitted to the time-stamped position data. The slope of the resulting line provides a robust estimate of the object's average velocity (v_x, v_y) during the tracking period.

Kalman Filter: While simple methods like linear regression over the stored position history were considered, we implemented a Kalman Filter to process the incoming stream of centroid data. This approach was chosen for its superior performance in real-world scenarios, as it provides a continuously refined and smoothed estimate of the object's state while being inherently robust to measurement noise from the perception pipeline. The filter is configured with a 4-dimensional state vector $\mathbf{x} = [x, y, v_x, v_y]^T$, representing the object's position and velocity in the 2D plane. The filter operates in a standard predict-update cycle:

1. **Predict:** At each time step Δt, the filter predicts the next state based on a linear motion model: $\mathbf{x}_{k|k-1} = \mathbf{F}_k \mathbf{x}_{k-1|k-1}$. The state transition matrix $\mathbf{F}_k$ incorporates Δt to update the position based on the current velocity estimate.

2. **Update:** When a new measurement $\mathbf{z}_k$ (a noisy centroid position) arrives from the `clustering_servoing` node, the filter computes the Kalman Gain and updates its state estimate by blending the prediction with the new measurement. This corrects the state based on real-world data.

Upon transitioning to the `ESTIMATING` state, the system captures the final, stabilized state from the Kalman Filter. The velocity components, v_x and v_y, become the crucial inputs for the next phase.

4.4.2.2 Blind Grasping with Feed-Forward Control

After estimating the object's velocity, the system stops the visual servoing and transitions to a "blind" interception phase (`BLIND_DESCENT`). In this state, it no longer uses the live feedback from the camera. Instead, it uses the estimated velocity to continuously predict the object's future trajectory.

The core of this phase is a high-frequency control loop (running at 50 Hz in our implementation) that dynamically calculates a target intercept point for each cycle. This predicted point is calculated using the classic linear motion equation:

$$p_{predicted}(t) = p_{start} + v_{est} \cdot \Delta t \tag{4.2}$$

where p_{start} is the object's position at the beginning of the blind phase, v_{est} is the constant velocity obtained from the Kalman Filter, and Δt is the time elapsed since the start of the phase.

The robot's motion is now governed by a controller that aims to minimize the error between the end-effector's current position and the object's predicted future position. The control law is augmented with a feed-forward term based on the estimated velocity:

$$v_{control} = K_p \cdot e_{pred} + v_{est} \tag{4.3}$$

where e_{pred} is the error between the end-effector and the predicted object position, and v_{est} is the estimated velocity from the previous stage. This feed-forward term commands the robot to move with the object, while the P term corrects for any remaining error. The robot simultaneously descends in the Z-axis to the required grasping height.

Once the error between the end-effector and the predicted intercept point falls below a small threshold, the system triggers the final grasp, closing the gripper to capture the moving object.

Grasp Execution and Task Completion The blind interception phase concludes when the robot's end-effector reaches the predicted intercept point. This is determined by checking if the position error in both the XY plane and the Z-axis falls below a predefined threshold (`dead_zone_xy` and `dead_zone_z`). Once this condition is met, the system triggers the final grasp and executes a pre-programmed post-grasp sequence to complete the task:

1. **Grasp:** The gripper is commanded to close, securing the object.
2. **Lift:** The robot moves the object vertically upwards to clear the workspace.
3. **Move To Drop-off and Release** The arm retreats and moves to a designated drop-off location, the gripper opens, releasing the object.
4. **Return Home:** The robot returns to its initial home pose, ready for the next command.

Chapter 5

Experimental Results

This chapter presents our experimental evaluation of the proposed event-based grasping pipeline. The experiments were divided into two main categories: grasping a static object (Section 5.1) and tracking and grasping a moving object (Section 5.2). We analyze the system's performance by varying several key parameters, including the robot's exploration trajectory, the visual texture of the target object, and the object's orientation.

5.1 Static Object Grasping Experiments

Objective and Goals:

The first set of experiments assessed the static-object grasping pipeline and evaluate the accuracy and reliability of the perception module under different conditions and determining the optimal parameters for robust performance.

Experimental Setup:

The experimental evaluation was conducted using a 6-DOF Kinova Gen3 robotic manipulator equipped with a Robotiq 2F-85 gripper. An iniVation DAVIS346 event-based camera was mounted on the robot's end-effector in an "eye-in-hand configuration". The entire software pipeline, including the ROS nodes for perception, control, and mission management, was run on a computer with Ubuntu 24.04.1 LTS with ROS Kinetic in a Docker container.

Figure 5.1: Experiment setup used for the experiments, consists of the Kinova Gen3 arm, the 2f-85 Robotiq gripper, and the DAVIS346 camera in an eye-in-hand configuration.

The target object was a rectangular box (cuboid), of size 8.5 x 6.0 x 14.5 cm. To evaluate the system's performance under various conditions, the object surface was covered with different high-contrast patterns. The patterns used were the following:

- A University of Genova (UNIGE) logo with distinct borders.
- A UNIGE logo with a dotted background.
- A RICE Lab logo with a high-frequency zigzag background.
- A standard checkerboard pattern.
- A label-free condition, using only the object's inherent features (text).

Figure 5.2: The different printed patterns used to evaluate the performance of the event-based clustering algorithm.

Additionally, two primary end-effector motion trajectories were used in order to generate events, because the more events we have, the better the clustering will be:

- Planar square movement, which was a square-shaped path in the XY plane above the object, with a side length of approximately 8 cm.

- Rotational Yaw Sweep movement, which was a rotational movement around the end-effector's yaw axis.

The robot setup was the same for all the experiments; the parameters that have been changed were the exploration movement of the robot (square or yaw sweep), the object surface patterns, and the object orientation.

Evaluation Metrics:
The performance of the event-driven static object detection and grasping pipeline was evaluated using the following quantitative metrics:

- **Euclidean Distance Error**
  The spatial accuracy of the detected centroid was measured as the Euclidean distance between the ground-truth (reference) centroid position $(x_{\text{ref}}, y_{\text{ref}})$ and the detected centroid using our vision module $(x_{\text{det}}, y_{\text{det}})$ represented in the robot base frame:

  $$e = \sqrt{(x_{\text{det}} - x_{\text{ref}})^2 + (y_{\text{det}} - y_{\text{ref}})^2}$$

  In the case of the static object, the ground-truth position was established by manually moving the robot's end effector to the center of the object and recording its pose. This metric was expressed in meters. For each configuration, the mean and standard deviation across trials were computed to assess both accuracy and consistency.

- **Events Count**
  The number of events that fall inside the bounding box detected of the object during each grasp experiment. This metric was used to compare how

different motion trajectories, object surface patterns, and object orientations affected the camera's event generation.

- **Grasp Success Rate**
  The percentage of trials in which the robot successfully grasped, lifted, and held the object. This metric was calculated as follow:

$$\text{Success Rate} = \frac{\text{Successful Grasps}}{\text{Total Attempts}} \times 100\%$$

5.1.1 Performance with Square Trajectory Exploration

The first tested exploration motion was a square trajectory, where the end-effector moves in a 20x20 cm square pattern (in the x and y directions), ensuring that the DAVIS346 observes the object from multiple viewpoints to make sure enough events were generated. For each trial, the robot executed the trajectory, processed the accumulated events to detect the object and calculate its centroid, and then attempted to grasp it. The experiment was carried out for different setups; the object with a plain surface and with four different high-contrast patterns presented in Figure 5.1.

The reference centroid of the object (only the positions x and y because z was fixed) was measured manually using a meter with respect to the robot base, and also double checked using MoveIt commander where we placed the robot end-effector in the correct centroid position and using MoveIt (current position), we displayed this position the `base_link` frame, these measurements gave us the following reference centroid:

$$(x_{\text{ref}}, y_{\text{ref}}) = (0.553, -0.041) \text{ m}$$

This value was used as the ground truth for the Euclidean distance error calculation presented in the results Table 5.1.

For each of the five object texture configurations (No Pattern and four distinct patterns), we conducted 10 consecutive grasp attempts. The performance was evaluated by recording the centroid estimated error, the final grasp outcome (success or failure), and the number of events generated for each object found.

The detailed results of the grasping attempts are presented in Tables 5.1, 5.3, and 5.2.

Trial	No pattern	RICE zigzag	UNIGE borders	UNIGE dots	Checkerboard
1	12.50	5.10	11.70	11.70	3.60
2	1.40	5.00	8.40	10.70	2.80
3	3.60	2.20	8.20	11.70	1.60
4	0.00	5.10	6.40	5.10	2.80
5	2.20	8.50	9.20	10.40	2.40
6	10.00	2.20	7.00	9.40	3.60
7	5.10	3.60	7.80	6.70	3.60
8	5.10	3.60	9.40	6.40	5.80
9	7.10	2.20	8.60	7.60	3.20
10	6.10	3.20	7.80	5.80	4.50
Average	**5.31**	**4.07**	**8.45**	**8.55**	**3.39**
Std. dev.	**3.25**	**2.11**	**1.90**	**2.35**	**1.42**

Table 5.1: Centroid error (mm) for square exploration over 10 trials.

Trial	No pattern	RICE zigzag	UNIGE borders	UNIGE dots	Checkerboard
1	921	2762	2155	2752	1985
2	1395	2510	2003	2747	2071
3	1402	2798	2298	2707	2099
4	1439	2752	2186	2584	1932
5	1358	2756	2192	2636	2032
6	1754	2730	2174	2636	1998
7	1567	2744	2198	2680	2036
8	1683	2689	2177	2736	2018
9	1568	2705	2266	2747	1889
10	1532	2770	2275	2735	2055
Average	**1462**	**2722**	**2192**	**2696**	**2012**

Table 5.2: Event counts within the detected object window for square exploration (10 trials).

Trial	No pattern	RICE zigzag	UNIGE borders	UNIGE dots	Checkerboard
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	0	1	0	1	1
5	1	1	1	1	1
6	1	1	1	1	1
7	1	1	1	1	1
8	1	1	0	1	1
9	1	1	1	1	1
10	1	1	1	1	1
Success (%)	**90**	**100**	**80**	**100**	**100**

Table 5.3: Grasp outcome and success rate for square exploration (1 = success, 0 = failure).

5.1.1.1 Discussion of The Results

The results from the square trajectory exploration experiments prove that our proposed event-driven pipeline is highly effective for static object grasping, achieving millimeter-level accuracy and a high grasp success rate across all configurations, and it also demonstrates a strong correlation between the visual texture of the object, the number of events generated, and the final grasping performance.

As shown in Table 5.2, the "No pattern" condition generated the fewest events (average of 1462), which corresponds to an average centroid error of 5.31 mm (Table 5.1) and a success rate of 90% (Table 5.3). Conversely, the "RICE zigzag" and "UNIGE dots" patterns produced the most events. However, the "Checkerboard" pattern represents the optimal balance; despite generating fewer events than the zigzag pattern, it achieved the lowest average centroid error with just 3.39 mm, and also a perfect 100% success rate.

The "UNIGE borders" and "UNIGE dots" patterns performed noticeably worse, with average errors exceeding 8 mm. This is likely because the features are either too sparse (dots) or concentrated only at the edges (borders), leading to a less reliable centroid calculation compared to the dense features of the checkerboard and zigzag patterns.

It is also important to note a key experimental limitation observed during the trials. In all recorded failure cases, the centroid was correctly identified within a graspable tolerance and grasp failures were due to a MoveIt trajectory planning anomaly. The planner would occasionally generate a non-linear path with a large joint rotation. Due to the physical constraint of our setup, for instance, the

short camera cable, these trajectories had to be manually aborted (Pressing the Emergency Stop) to prevent equipment damage. These instances are recorded as failures in Table 5.3 (e.g., the single failure for "No pattern" and the two failures for "UNIGE borders"), even though our perception pipeline had succeeded in finding a correct centroid.

It is particularly noteworthy that the "No pattern" configuration achieved a respectable 5.31 mm average error and 90% success rate despite relying solely on the object's natural edges and any inherent surface features (such as printed text). This demonstrates that the pipeline does not strictly require high-contrast artificial patterns to function effectively. The ability to detect and grasp objects based on their natural boundaries alone is crucial for real-world industrial applications, such as handling pharmaceutical packaging, cardboard boxes, or retail items that bear only printed labels or logos rather than structured visual markers. This robustness to texture-free surfaces validates the practical applicability of the geometric prior approach in scenarios where adding fiducial markers or patterns is impractical or undesirable.

5.1.2 Performance with Rotation (Yaw Sweep) Trajectory Exploration

The second exploration motion is a "yaw-sweep" trajectory. In this configuration, the robot's end-effector remains at a fixed XY position above the object and rotates its wrist (axis 6) back and forth around its vertical axis. This motion primarily generates events from the object's vertical edges. To ensure a consistent comparison, 10 trials were conducted for each of the three used object texture configurations: "No pattern", "RICE zigzag", and "UNIGE dots".

The reference centroid of the object (only the positions x and y because z was fixed) was measured manually using a meter with respect to the robot base, these measurements gave us the following reference centroid:

$$(x_{\text{ref}}, y_{\text{ref}}) = (0.553, -0.041) \text{ m}$$

This value was used as the ground truth for the Euclidean distance error calculation presented in the results Table 5.4.

The detailed results of the grasp attempts are presented in Tables 5.4, 5.5, and 5.6.

Trial	No pattern	RICE zigzag	UNIGE dots
1	5.10	12.50	6.40
2	3.60	5.10	7.60
3	8.00	7.60	7.10
4	4.10	11.20	9.10
5	1.00	10.00	6.40
6	6.20	10.60	6.40
7	2.50	10.00	7.60
8	7.50	7.30	6.10
9	4.80	7.10	5.40
10	2.90	7.10	8.20
Average	**4.57**	**8.85**	**7.03**
Std. dev.	**2.22**	**2.33**	**1.11**

Table 5.4: Centroid error (mm) for yaw sweep exploration over 10 trials.

Trial	No pattern	RICE zigzag	UNIGE dots
1	1902	2080	2523
2	1056	2529	2491
3	834	2148	2540
4	1550	2460	2462
5	1611	2474	2612
6	1700	2452	2532
7	1100	2098	2237
8	950	2536	2542
9	1400	2589	2545
10	1850	2559	2327
Average	**1395**	**2393**	**2487**

Table 5.5: Event counts for yaw sweep exploration over 10 trials.

Trial	No pattern	RICE zigzag	UNIGE dots
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
Success (%)	**100**	**100**	**100**

Table 5.6: Grasp outcome for yaw sweep exploration (1 = success, 0 = failure).

5.1.2.1 Discussion of Yaw Sweep Trajectory Results

The results of the yaw–sweep exploration further validate the effectiveness of the event-driven pipeline for static object grasping, but also reveal a different relationship between object texture, event count, and localization accuracy than the one observed with the square trajectory.

As reported in Table 5.5, the "No pattern" condition generated the fewest events (average of 1395). Nevertheless, it achieved the lowest average centroid error of 4.57 mm in Table 5.4, outperforming both "RICE zigzag" (8.85 mm) and "UNIGE dots" (7.03 mm). This apparently counter-intuitive result suggests that, under a pure yaw motion, the most informative visual cues are the object's vertical edges. The rotation activates these two contours cleanly and symmetrically, yielding a compact event cluster and a stable rectangle fit. By contrast, adding dense internal textures (zigzag, dots) increases the number of events but also introduces asymmetric, high-frequency activity inside the silhouette, which can bias the rectangle fit and degrade the centroid estimate. In other words, for yaw sweeps, event quality (clean edge activation) matters more than event quantity.

Despite these differences across textures, the perception module maintained millimeter-level accuracy for all tested patterns, within a graspable tolerance for our gripper. This experiment therefore underscores that the best texture depends on the exploration motion: rich patterns help under translational exploration, while simple edge-only appearance can be preferable for rotational (yaw) exploration.

5.1.3 Performance With Various Object Orientation

One of the important requirements for a practical grasping system is the ability to handle objects in arbitrary orientations. To evaluate the robustness of our system to different object orientations, we conducted a final experiment in the static object case.

Using the "UNIGE dots" pattern and the square exploration trajectory configuration, the object was placed at five different orientations: 0, 30, 45, -45, and 90 degrees. For each orientation, a complete grasp cycle was performed, and the centroid error and grasp outcome were recorded. The results are presented in Table 5.7.

Orientation (degrees)	Centroid Error (mm)	Grasp Success
0	4.07	Success
30	4.25	Success
45	4.51	Success
-45	4.48	Success
90	4.15	Success

Table 5.7: Performance with various object orientation.

5.1.3.1 Discussion of Orientation Results

The results in Table 5.7 demonstrate that the system is highly robust to changes in object orientation. The grasp was successful for all angles tested and the centroid error remained consistently low and acceptable. This experiment confirms that the pipeline described in our methodology can reliably determine the complete correct object pose (position and orientation) in a flexible manner, making it well suited for real-world applications(conveyor belt, for instance) where the object's rotation cannot be guaranteed.

5.1.4 General Discussion and Comparison

To summarize the findings of all static object grasping experiments, Table 5.8 highlights the best results for each exploration trajectory and pattern. The comparison highlights the impact of both motion type and object texture on perception accuracy and grasp reliability.

Explore Motion	Pattern	Avg. Centroid Err (mm)	Success (%)
Square	RICE zigzag	4.07	100
Square	Checkerboard	3.39	100
Yaw Sweep	No pattern	4.57	100
Yaw Sweep	UNIGE dots	7.03	100

Table 5.8: Comparative results of best performance for static object grasping.

The results demonstrate that both the exploration motion and the object's visual texture significantly affect the perception and grasping performance. The square trajectory combined with a high-contrast pattern (especially "RICE zigzag" or checkerboard) consistently yielded the lowest centroid errors and perfect grasp success rates. The yaw sweep motion, while robust, showed slightly higher errors, and its performance depended more on the object's natural edges than on added patterns.

Across all static experiments, the system achieved millimeter-level localization accuracy and reliable grasping, regardless of object orientation. The orientation estimation logic proved robust, enabling successful grasps for all tested angles. Grasp failures, when present, were attributed to external planning constraints rather than perception errors.

In summary, the static object grasping pipeline is highly effective and flexible, capable of accurate and repeatable performance under varied conditions. These findings provide a strong foundation for the next section, where the system's capabilities will be evaluated on moving objects.

5.2 Moving Object Grasping Experiment

Objective and Motivation
The objective of this set of experiments is to demonstrate the feasibility and robustness of our event-based perception and control pipeline to track and grasp an object in motion, simulating a conveyor scenario. The goal is to demonstrate proof of concept dynamic grasping under realistic but material-limited laboratory conditions.

Experimental Setup and Limitations
The experiments were carried out using the same hardware as in the static object case: a 6-DOF Kinova Gen3 robotic manipulator with a Robotiq 2F-85 gripper and an iniVation DAVIS346 event-based camera in an eye-in-hand configuration, as a pattern for the object surface, we used the "RICE zigzag" pattern, as it showed good results in the static object grasping experiments.
Due to the absence of a motorized conveyor, the motion of the object was simulated by placing the target object on a long sheet of paper and pulling it manually across the workspace, as shown in Figure 5.3. This setup approximates a conveyor scenario, although with less control over speed and trajectory repeatability.

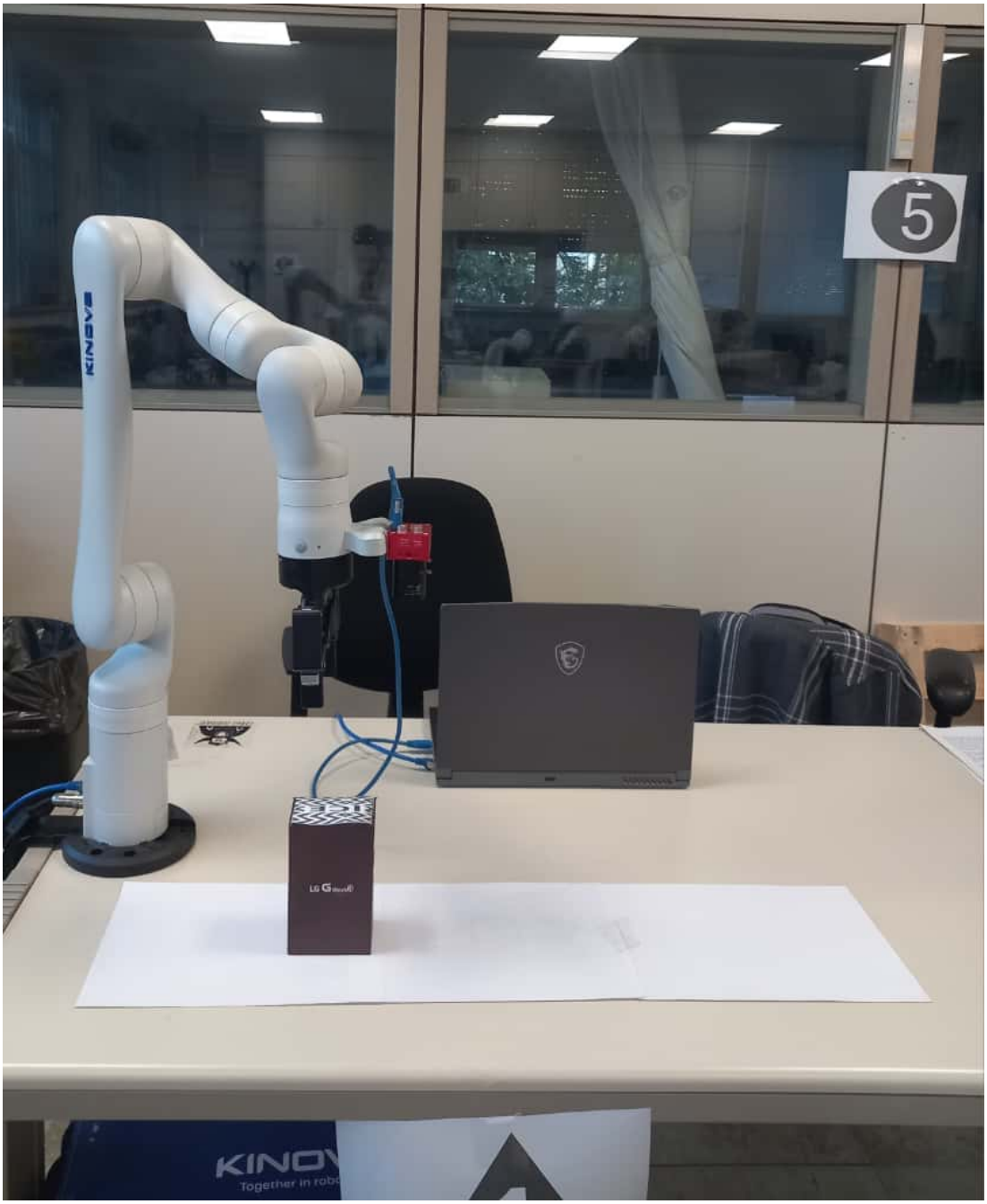


Figure 5.3: Experimental setup for moving object grasping: the object is placed on a long sheet of paper and pulled by hand to simulate a conveyor motion.

The main limitation of this setup is the inability to precisely move the object with a constant and smooth speed and to guarantee perfectly repeatable trajecto-

ries at each grasping trial. As a result, the experiments serve as a proof-of-concept rather than a quantitative benchmark.

Experimental Procedure
Our experiments were carried out in two phases, first a tracking system where the robot's end-effector was commanded to follow the moving object in real time using position-based visual servoing (PBVS) processing the centroid data from the clustering system, minimizing the error in the X and Y directions as the object moved. This phase confirms the system's ability to continuously track a moving target.

In the second phase, the system tracked the object for a short period, collecting a sequence of centroid positions. A Kalman filter was used to estimate the object's velocity, after which the robot executed a predictive intercept and grasp. Both straight line (X direction only) and diagonal (XY direction) object motions were tested during our experiments, by sliding the object on the paper simulating a conveyor belt.

5.2.1 Results and Observations

Despite the lack of precise speed control of the object (the object was moved by hand), the system consistently succeeded in tracking and intercepting the moving object in both tested object motion trajectories (straight line and diagonal). However, it is important to note that while the robot successfully closed the gripper on the object during motion, the contact point was often near the object's edge rather than at its centroid that's a result of an effect of the object's irregular and imprecise speed.

These results demonstrate the effectiveness of the event-based perception and predictive control pipeline for dynamic interception, while also highlighting that accurate and stable object motion is essential to achieve centroid-accurate grasps. The results also serve as a proof of concept for the proposed approach and the experiments validate that the pipeline can handle real-time tracking and grasping of moving objects, even under non-ideal, manually simulated conditions.

To further support these findings, we provide video recordings of the experiments at this Dropbox folder. These videos show the full grasping cycle and visually confirm the system's performance.

Also, the QR code in Fig. 5.4 encodes the same link as above, for convenient access from a printed copy.

Figure 5.4: QR code linking to the Dropbox folder with the grasping videos.

5.2.2 Discussion and Future Work

The dynamic grasping experiments served as a crucial proof-of-concept for integrating the event-based vision pipeline into a real-time robotic system. The main contribution lies in demonstrating the successful closed-loop integration of event-based perception with predictive control under real motion conditions. While the lack of a motorized conveyor limited rigorous quantitative benchmarking as we could not systematically measure grasp accuracy as a function of object speed or trajectory complexity, the successful interceptions across varied manual trajectories (including straight-line and diagonal paths) validate three key technical claims underpinning the proposed system.

First, the clustering pipeline provides sufficiently stable pose estimates for continuous Position-Based Visual Servoing (PBVS) tracking. Second, the Kalman filter extracts the object's velocity with sufficient accuracy to enable interception, though residual prediction errors resulted in off-center contact points (near the object edge rather than its centroid, as visible in the video recordings). Finally, the feed-forward control based on this prediction successfully compensates for the combined system latency (perception, communication, and robot response), enabling an effective intercept grasp. These results firmly establish the feasibility of the approach and provide strong evidence for the robustness of the event-based perception component.

Looking to future work, we must address the primary limitation of these experiments by integrating a motorized conveyor system. This will enable systematic variation and precise measurement of object speed, allowing for a rigorous and

quantitative benchmarking of the dynamic grasping performance. The success rate can then be characterized as a function of object speed, which will fully explore the performance envelope of the event-based perception pipeline.

Chapter 6

Conclusions

Traditional grasping vision pipelines rely on frame-based cameras, which capture images at fixed frame rates and produce dense but redundant data. While effective in static scenes, these systems struggle in fast or unpredictable scenarios due to motion blur, limited dynamic range, and high latency. Event-driven cameras offer a compelling alternative. Rather than capturing frames, they asynchronously report per-pixel brightness changes, yielding sparse data streams with microsecond resolution. This thesis presented a real-time grasping framework driven by an event-based vision sensor and tailored to conveyor-style manipulation.

Motivated by the latency and robustness advantages of event cameras, we designed a perception pipeline that operates directly on the asynchronous event stream and avoids dense 3D reconstruction by relying only on object bounding dimensions as geometric priors. The pipeline converts events into a stable metric pose of the target object and supports two applications: a "search–then–grasp" strategy for static targets, where exploration is performed and once a stable centroid is found the grasping is performed. And a "track–then-predictive-grasp" strategy for moving targets using Position–Based Visual Servoing (PBVS) for tracking and velocity estimation, followed by a predictive intercept for grasping.

The system achieves millimetre–level localization in the static setting (3.39 mm mean centroid error on checkerboard targets) Notably, the system demonstrated reliable performance even without artificial patterns or markers, achieving a respectable 5.31 mm average error and a high rate of successful graspings (e.g., 90% for square trajectories and 100% for yaw sweeps) using only the object's natural edges and inherent surface features. This capability is particularly valuable for industrial applications where adding fiducial markers to products is impractical or undesirable, such as handling pharmaceutical packaging or retail items with only printed labels. In dynamic trials, the object was tracked using PBVS to collect position data, and a Kalman Filter was applied to estimate its velocity

and pose for interception. Most failures were due to the object being moved manually, which introduced unstable and imprecise speeds that reduced the accuracy of the KF predictions, rather than limitations in perception or motion planning.

6.1 Limitations and Practical Considerations

During the development of the system, and after performing the experiments and evaluating the results, some limitations were identified. In the following, we state these limitations and we suggest improvements to overcome it:

- **Evaluation set-up: missing motorized conveyor.** Dynamic experiments used manually moved objects instead of a belt with controlled speed. Results therefore constitute a proof of concept rather than a full quantitative evaluation by varying the object speed.
  Improvement: integrating a motorized conveyor to ensure a stable speed of the object and a reference success rate precision versus object speed, acceleration, and path complexity.

- **Motion-planning latency and occasional failures (MoveIt).** Grasp timing was sometimes dominated by MoveIt planning latency, with occasional plan failures; these were independent of the perception module.
  Improvement: MoveIt was used as a convenient solution for motion planning. However, in a real-world implementation, a more dedicated and optimized control system could significantly reduce motion execution time and overcome planning problems.

- **Geometric prior and pose output.** The pipeline assumes a known object height and a top surface approximately planar and parallel to the support; it estimates centroid and in-plane yaw (not full 6-DoF pose).
  Improvement: adding an auxiliary depth source (e.g. stereo setup) when height is unknown, also adopting shape-agnostic acceptance criteria to generalize beyond box-like objects, extending to full resulting pose when required by the application.

6.2 Future Work and Practical Roadmap

For future work, the following directions are proposed:

- **Auxiliary depth source and learned cluster validation** Introducing an optional depth source removes the fixed-height assumption and unlocks full metric scale when object height is unknown, e.g., via a stereo setup with two event cameras or an event camera paired with a depth sensor. Beyond depth, cluster validation can leverage learned, event-native recognition: lightweight classifiers (e.g., CNNs on time-surfaces or event-count images) can assign a detection confidence and output an approximate class label and 6-DoF pose (position and orientation) for each candidate cluster, re-ranking detections to reject clutter and texture-poor false positives.

- **Instrumented conveyor and multi-object benchmarking** From an evaluation standpoint, introducing a motorized conveyor will enable systematic benchmarking across belt speeds, accelerations, and path geometries (straight, diagonal, curved). This setup also facilitates repeatable tests of robustness to texture, illumination, and occlusion. Scaling to multi-object scenes is a natural next step: running per-cluster pose tracks with simple identity management and scheduling grasps over several candidates would bring the system closer to real industrial flows.

- **Event–frame fusion (hybrid perception).** Finally, an idea can be to fuse the DAVIS346 grayscale frames with the event pipeline to form a hybrid system. While events provide microsecond, low-latency updates, low-rate APS frames (10–30 Hz) can stabilize estimation when event activity drops (e.g., slow motion, low texture, or illumination changes). APS edges/corners can help validate event clusters, providing drift-free centroid corrections. This hybrid strategy is expected to improve detected centroid stability and orientation accuracy, and maintain continuity without resorting to dense reconstruction.

Appendix A

Project GitHub Repository

The source code for this thesis is available on GitHub: `https://github.com/hocinedl/Event-Grasping-Pipeline`.

References

[1] Barranco, F., Fermuller, C. & Ros, E. (2018). Real-time clustering and multi-target tracking using event-based sensors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE. 12

[2] Brandli, C., Berner, R., Yang, M., Liu, S.C. & Delbruck, T. (2014). A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid-State Circuits*, **49**, 2333–2341. 4, 7

[3] Chaumette, F. & Hutchinson, S. (2006). Visual servo control. i. basic approaches. *IEEE Robot. Autom. Mag.*, **13**, 82–90. 17, 18

[4] Coleman, D.T., Sucan, I.A., Chitta, S. & Correll, N. (2014). Reducing the barrier to entry of complex robotic software: A MoveIt! case study. 21

[5] Cong, V.D. & Hanh, L.D. (2023). A review and performance comparison of visual servoing controls. *Int. J. Intell. Robot. Appl.*, **7**, 65–90. vii, 17

[6] Contributors, M. (2025). Moveit: Motion planning framework for ros. [Online]. Available: `https://moveit.ros.org`. vii, 22

[7] Ester, M., Kriegel, H.P., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, 226–231, AAAI Press. 12

[8] ETH Zurich - Autonomous Systems Lab (2025). Kalibr: Calibration toolbox for cameras (and imus). `https://github.com/ethz-asl/kalibr`, accessed: 2025-09-07. 21

[9] Furgale, P., Rehder, J. & Siegwart, R. (2013). Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE. 21

[10] Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A.J., Conradt, J., Daniilidis, K. & Scaramuzza, D. (2022). Event-based vision: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, **44**, 154–180. 4, 5, 6, 7

[11] Gehrig, D., Rebecq, H., Gallego, G. & Scaramuzza, D. (2018). Asynchronous, photometric feature tracking using events and frames. In *Computer Vision – ECCV 2018*, Lecture notes in computer science, 766–781, Springer International Publishing, Cham. vii, 5

[12] Glover, A., Gava, L., Li, Z. & Bartolozzi, C. (2024). EDOPT: Event-camera 6-DoF dynamic object pose tracking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 37, 18200–18206, IEEE. 14

[13] Gorner, M., Haschke, R., Ritter, H. & Zhang, J. (2019). MoveIt! task constructor for task-level motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, IEEE. 21

[14] Graca, R., Zhou, S., McReynolds, B. & Delbruck, T. (2024). Scidvs: A scientific event camera with 1.7*ESSERC*. 6

[15] Hinzmann, T. & Siegwart, R. (2021). SD-6DoF-ICLK: Sparse and deep inverse compositional Lucas-Kanade algorithm on SE(3). 11

[16] Huang, X., Halwani, M., Muthusamy, R., Ayyad, A., Swart, D., Seneviratne, L., Gan, D. & Zweiri, Y. (2022). Real-time grasping strategies using event camera. *J. Intell. Manuf.*, **33**, 593–615. 14, 15, 16, 18, 28

[17] Hutchinson, S., Hager, G.D. & Corke, P.I. (1996). A tutorial on visual servo control. *IEEE Trans. Rob. Autom.*, **12**, 651–670. 17, 18

[18] iniLabs AG (2025). Products - dvs and davis cameras. `https://inilabs.com/products/`, accessed: 2025-08-30. vii, 7

[19] iniVation AG (2025). Discontinued products - inivation documentation. `https://docs.inivation.com/hardware/discontinued-products/index.html`, accessed: 2025-08-17. vii, 6, 7, 8, 23

[20] Kim, H., Leutenegger, S. & Davison, A.J. (2016). Real-time 3D reconstruction and 6-DoF tracking with an event camera. In *Computer Vision – ECCV 2016*, Lecture notes in computer science, 349–364, Springer International Publishing, Cham. 14

[21] Kinova, K.R. (2025). ros_kortex: Kinova gen3 ros driver. https://github.com/Kinovarobotics/ros_kortex, accessed: 2025-09-08. 24

[22] Lagorce, X., Orchard, G., Galluppi, F., Shi, B.E. & Benosman, R.B. (2017). HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, **39**, 1346–1359. 12

[23] Lenero-Bardallo, J.A., Serrano-Gotarredona, T. & Linares-Barranco, B. (2011). A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor. *IEEE J. Solid-State Circuits*, **46**, 1443–1455. 4

[24] Li, H. & Stueckler, J. (2021). Tracking 6-DoF object motion from events and frames. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE. 14

[25] Lichtsteiner, P., Posch, C. & Delbruck, T. (2008). A 128×128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits*, **43**, 566–576. 6, 8

[26] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, **28**, 129–137. 11

[27] Open Source Robotics Foundation (2025). Robot operating system (ros). [Online]. Available: https://www.ros.org. 22

[28] Rebecq, H., Horstschaefer, T., Gallego, G. & Scaramuzza, D. (2017). EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real time. *IEEE Robot. Autom. Lett.*, **2**, 593–600. 10

[29] Rebecq, H., Gallego, G., Mueggler, E. & Scaramuzza, D. (2018). EMVS: Event-based multi-view stereo—3d reconstruction with an event camera in real-time. *Int. J. Comput. Vis.*, **126**, 1394–1414. 2, 9, 11, 16, 28

[30] Robotics & Perception Group, U.o.Z. (2018). rpg_dvs_ros: Ros driver for event-based vision sensors (davis, dvs). https://github.com/uzh-rpg/rpg_dvs_ros, accessed: 2025-08-30. 8, 23

[31] Robotics, K. (2025). Kinova gen3 ultra-lightweight robot arm. [Online]. Available: https://www.kinovarobotics.com/product/gen3-robots. vii, 24

[32] Robotiq (2025). Robotiq gripper 2f-85. [Online]. Available: https://robotiq.com/fr/produits/mains-adaptives#Two-Finger-Gripper. 24

[33] Taverni, G., Paul Moeys, D., Li, C., Cavaco, C., Motsnyi, V., San Segundo Bello, D. & Delbruck, T. (2018). Front and back illuminated dynamic and active pixel vision sensors comparison. *IEEE Trans. Circuits Syst. II Express Briefs*, **65**, 677–681. 7

[34] uzh-rpg (2025). This is the code for the 2018 ijcv paper emvs: Event-based multi-view stereo - 3d reconstruction with an event camera in real-time. [Online]. Available: `https://github.com/uzh-rpg/rpg_emvs`. 29