2023

Object Tracking with an Event Camera

Nader Ahmed , '23

Rezhwan A. Kamal , '23

Swarthmore College

Engineering 090 - Senior Design Project

Object Tracking with an Event Camera

Final Report

Creators:

Nader Ahmed

Rezhwan Kamal

Supervisors:

Professor Matt Zucker

Professor Stephen Phillips

Acknowledgments

We are grateful for the help of our faculty supervisors Professor Matt Zucker and Professor Stephen Phillips, whose guidance throughout the semester was vital for our progress on this project. We would also like to thank the UPenn GRASP Lab—specifically graduate students Kenneth Chaney and Fernando Cladera Ojeda—for allowing us to borrow and experiment with their EVK-3 sensor during the fall and showing us some of the work being done with event cameras at the Pennovation Center.

Abstract

In this project, an event-based camera was used to develop a functional proof of concept for a procedural 3-D cube tracking system. Hough space shape detection was used to track the position of cube edges in the image space, and an optimization-based algorithm was used in conjunction for finding the pose of the cube. Additionally, a ghosting effect was seen following the trailing edge of events. A sensor model was developed for simulating and investigating this phenomenon.

Table of Contents

1. Introduction 3

1.1. Event-Based Vision 3

1.2. The EVK-4 HD Sensor 4

2. Tracking a 2D Object 4

2.1. Hough Circle Transformation 4

2.2. Results and Discussion 5

3. Sensor Modeling 7

3.1. SONY IMX-636 7

3.2. Infinite Impulse Response Filter 8

3.3. Simulation 8

3.4. Results and Discussion 9

4. Tracking a 3D Object 10

4.1. Line Detection 10

4.2. Pinhole Camera Model 12

4.3. Obtaining camera intrinsics 13

4.4. Cube pose estimation 15

4.5. Tracking simulation 16

5. Engineering Design 19

6. Conclusion 19

References 20

1. Introduction

1.1. Event-Based Vision

The event camera is a type of vision sensor that has gained widespread attention in the research community over the past decade. Unlike a traditional camera, which constructs snapshots of the physical scene using the accumulated light hitting each sensor at a particular framerate, an event camera detects changes in light intensity that occur at each pixel in the sensor as they occur [1]. Each change, or "event," is a signal that consists of a timestamp, the location of the pixel, and its polarity (indicating a positive or negative change in light intensity). Since events are captured asynchronously and not tied to a specific frame, event cameras can capture high-speed motion with fewer computational resources. Event-based vision sensors are also more bandwidth-efficient than traditional cameras given the sparsity of events in most use cases. Because of these key advantages, event cameras are well-suited for real-time vision sensing applications such as autonomous vehicles and augmented reality.

While the problem of tracking objects in motion using a traditional camera is well-explored, the literature on object tracking using event-based vision sensors is comparatively sparse. Therefore, in order to create an end-to-end system for procedural object tracking and detection using an event stream, it is necessary to implement key components such as feature extraction and pose estimation. The goal of this project is to produce working implementations of these subsystems and integrate them to achieve accurate real-time tracking of the pose of a moving cube.

1.2. The EVK-4 HD Sensor

At the time of writing, the EVK-4 HD is the newest vision sensor sold commercially by Prophesee, a French company that is credited with producing the most advanced neuromorphic vision systems [2]. The camera can be used in tandem with the Metavision SDK, a “C++ and Python SDK for event-based data processing” which is also provided by Prophesee [3]. This project makes use of the SDK’s Calibration module for initial calibration, as well as Metavision Studio, a GUI-based tool that provides functionality for recording from a live event camera and configuring bias settings [4].

2. Tracking a 2D Object

As a first step of this project, a working proof of concept for a 2D disk tracking system was produced. The system was tested on a .raw file of events associated with the motion of a falling disk. The results obtained using this deliverable were evaluated qualitatively and used to indicate if the project was on track to carry out the more complex task of tracking a cube.

2.1. Hough Circle Transformation

The Hough transformation is a feature extraction approach that helps to extract lines, curves, and other types of features from visual data. At a high level, it requires a parametric representation of the desired feature, as well as a discretized parameter space that represents all possible features that could be extracted [5]. Each entry on the parameter space represents a unique feature, or candidate, that can be extracted. In the simple case of black figure points on a white background, each figure point is treated as a “vote” for the set of possible features that it coincides with. These “votes” are accumulated in a matrix that represents the parameter space, and the local maxima of the accumulator matrix represent the likely features that can be extracted.

OpenCV provides built-in support for different Hough transform algorithms, including the circle transform [6]. In this case, the parameter space was three-dimensional, containing the Cartesian coordinates of the centroid of the circle to be extracted, along with its radius. Since this implementation works on conventional images, it was necessary to batch the events in a recording into discrete frames, then convert each batch of events into figure points on a traditional black-on-white image. Each frame was then passed into the Hough circle transform function, and the circle with the highest number of "votes" was returned.

It must be noted that this rudimentary approach was agnostic to event polarity, and, by converting to traditional images, it also failed to take advantage of the low-bandwidth capabilities of event cameras. As this was only a proof of concept, however, the primary concern here was in regard to the feasibility of the system for accurate object tracking.

2.2. Results and Discussion

This initial deliverable provided mixed results. On one hand, the process of getting an event feed from the EVK-4 sensor and passing the converted image data into a Hough transform algorithm was a demonstrable success, and the functional end-to-end system had promising qualitative results. However, upon further investigation of the event feed, the trailing edge events had a notable temporal lag, and a "ghosting" effect was observed behind the wake of the object, as can be seen below. This was an unexpected effect that warranted further investigation as it had the potential to negatively impact more complex object tracking systems. The investigation of this "ghosting" effect is the primary goal of the next stage of the project, and it will be covered in greater detail in the next section.

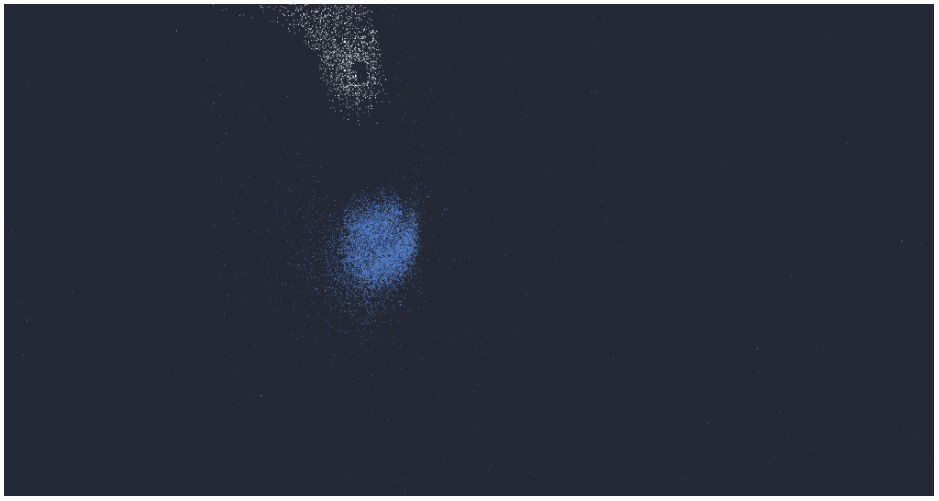


Figure 1. Events from frame of falling disk video (positive events in white, negative in blue).

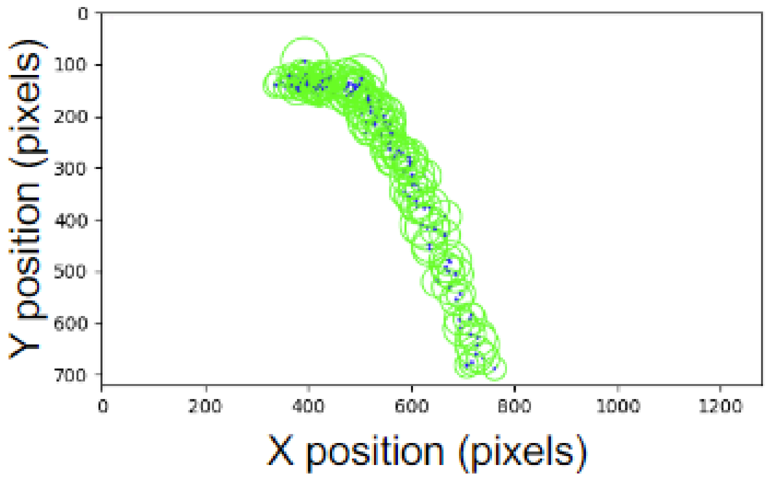


Figure 2. Overlay of extracted circles across every frame in the recording of a falling disk.

3. Sensor Modeling

One key assumption we made was that for objects with high-contrast edges, the sensor would act as an idealized edge detector. We anticipated seeing a symmetric feed of positive and negative events at the leading and trailing edges. However, our results from the 2D tracking were informative in that we learned this was not the reality. This prompted a change in plans; we sought to create a model to conceptualize how the camera's underlying sensor functions.

3.1. SONY IMX-636

The EVK-4 uses SONY's IMX-636ES as its sensor, which is built for event-based vision [7]. The datasheet for this sensor is proprietary, and SONY was unable to provide further information regarding it. However, some basic information could be gleaned from the promotional content on their event-based sensors. Namely, it was possible to obtain a high-level schematic of the analog system, displayed below, which demonstrates the process through which light rays are used to generate events.

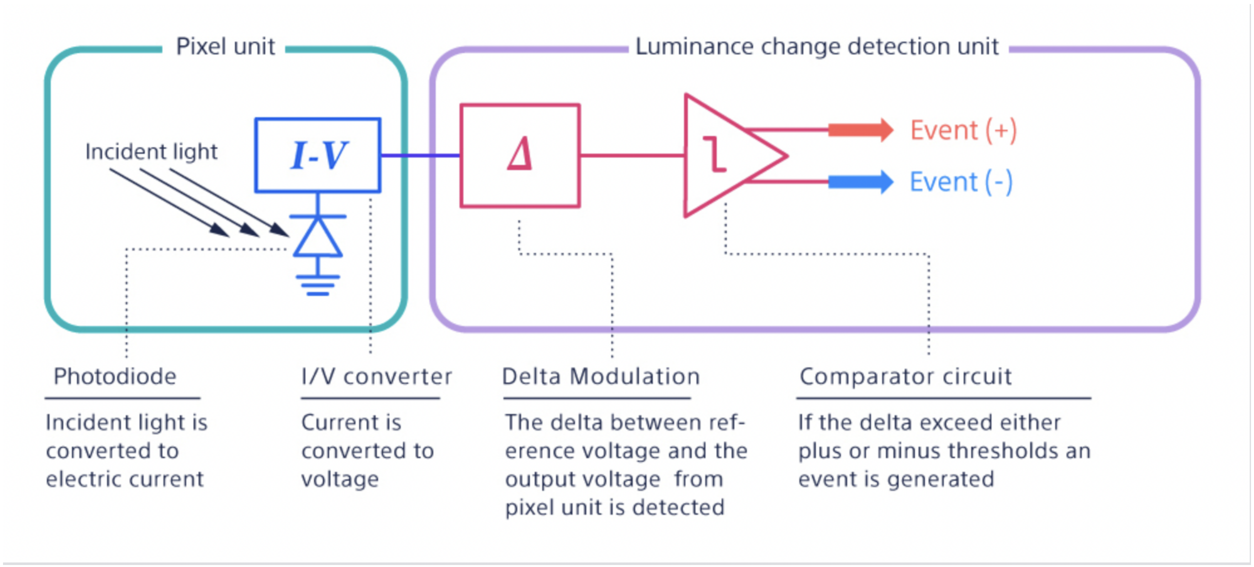

Pixel unit
Luminance change detection unit
Incident light
I-V
Δ
Event (+)
Event (-)
Photodiode
Incident light is converted to electric current
I/V converter
Current is converted to voltage
Delta Modulation
The delta between ref-erence voltage and the output voltage from pixel unit is detected
Comparator circuit
If the delta exceed either plus or minus thresholds an event is generated

Figure 3. Abstract schematic of the IMX-636ES sensor at the pixel level.

As illustrated in the figure, incident light gathered by the photodiode is converted to voltage in the pixel unit. This voltage is compared to the reference, which determines the voltage at which a positive or negative event is generated. If the change is substantial enough, an event signal is fired, and the reference voltage is updated for future differential measurements.

3.2. Infinite Impulse Response Filter

One key thing to note about the schematic above is that the analog sensor takes time to charge and discharge at the pixel level. This reality limits how quickly events can be generated relative to changes in incident brightness. Although an ideal event camera has no motion blur, this idealization breaks down in certain situations because of the sensor's inevitable analog limitations.

We suspected that this charging and discharging had an associated time constant similar in nature to a first-order RC circuit. To replicate this effect, we passed incident light intensity data through a first-order low-pass filter, which acted to smooth out the input signal in the time domain. A first-order Butterworth filter was chosen to preserve the accuracy of the input given its relatively flat passband.

3.3. Simulation

To test if the sensor's analog limitations were responsible for the ghosting, the sensor model was made to be tested on synthetic light intensity data: a rectangle moving across the image space. The Butterworth filter was then applied to the intensity data, and the logarithm of the result was taken to get the voltage. Events were then fired probabilistically based on the difference between the reference and actual voltage.

3.4. Results and Discussion

The results of the sensor model were promising. The events generated from the synthetic data using the sensor model showed an identical "ghosting" effect to what was observed in the initial 2-D object tracking attempt. This showed that this was in fact, due to the analog limitations of the circuit behind the sensor.

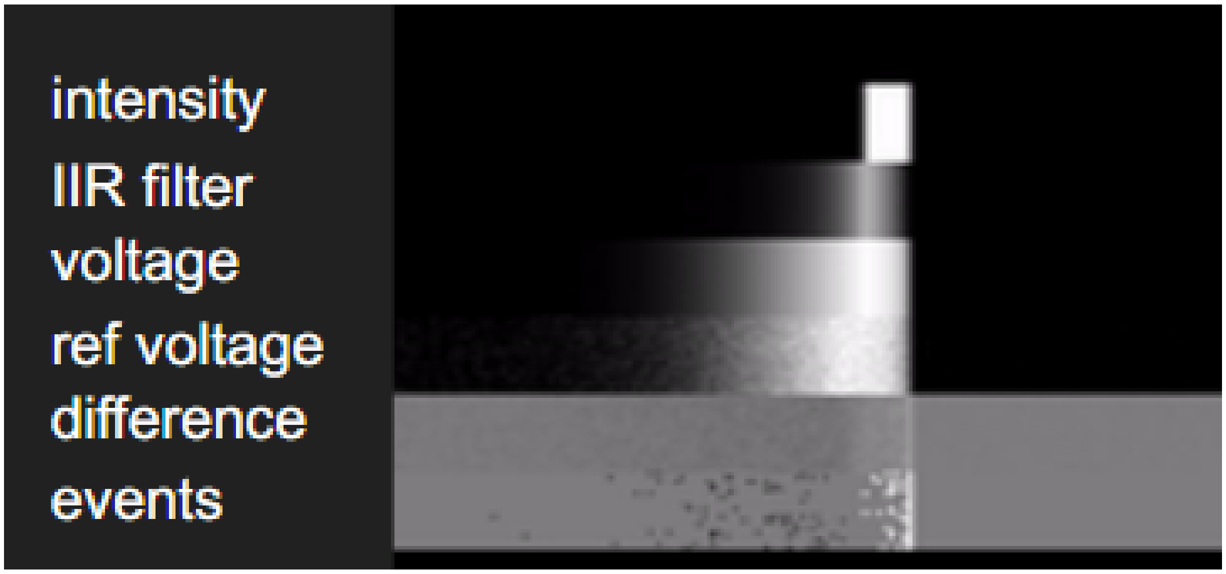


Figure 4. One frame of our sensor model simulation results, showing (top to bottom) light intensity, filtered intensity, intensity as voltage, reference voltage, voltage differences, and generated events (positive in white, negative in gray).

4. Tracking a 3D Object

In March, Prophesee released Metavision SDK 4.0 which added support for a Spatio-Temporal Contrast (STC) filter. It was a direct solution to the ghosting problem we had experienced from the 2D tracking–this filter processed and removed events of the same polarity within some threshold. This was great news–and allowed us to pivot back towards 3D object tracking. Our object of interest was a Rubik's cube with grid lines in a 3x3 pattern.

Figure 5: 3x3 Rubik's cube

4.1. Line Detection

Developing a robust line detection algorithm was an intuitive choice given the distinct straight lines present on the edges and interior of the Rubik's cube. In section 2.1 the Hough circle transformation was used for the detection of circles within images of batched events plotted at their particular pixel coordinate location. We used a related approach for detecting lines; the Hough line transform. This includes a variation in the accumulator space, now based on identifying lines parameterized by a radius *r* and angle *θ*.

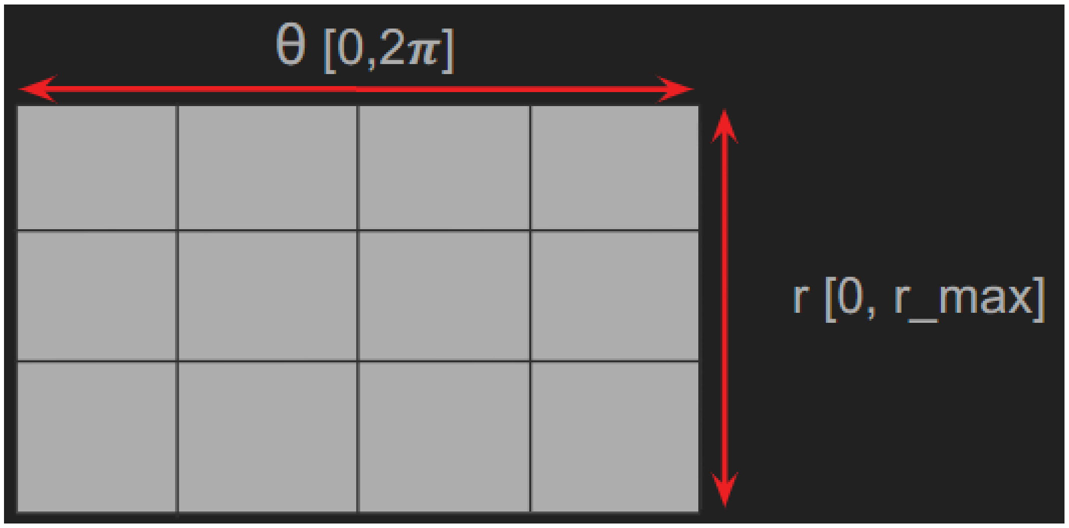
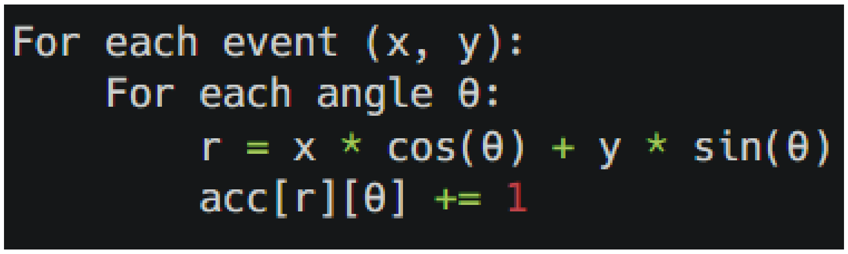


Figure 6: Visualization of the Hough space for line detection

However, the more significant difference in this approach was leveraging the unique data stream provided by our event camera. Instead of creating images with plotted events and feeding them into OpenCV's Hough detection algorithms, we processed the events individually and accumulated votes for the parameters of the subset of lines that pass through them.



```
For each event (x, y):
    For each angle θ:
        r = x * cos(θ) + y * sin(θ)
        acc[r][θ] += 1
```

Figure 7: Pseudocode for Hough space line accumulator

This was an improvement in that we were processing solely the events, and not doing excessive computation in looking at an entire image of batched events. Notably, the speed of this detection was further optimized by using vectorized NumPy arrays, which draw on faster pre-compiled code to do mathematical operations. For detecting the local maxima of the Hough accumulator, we used SciPy's maximum filter within a neighborhood size of 20 and found the accumulator positions which corresponded with the computed maximum values.

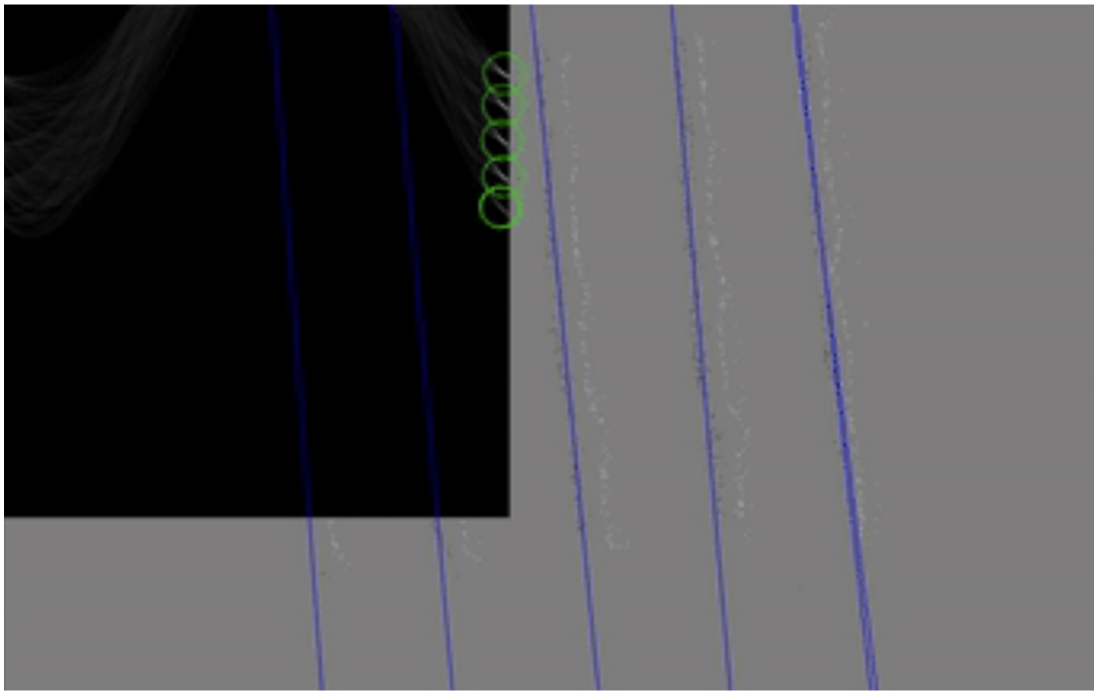


Figure 8: Hough line detector demo. Top left: Hough space with circles indicating local maxima and neighborhood size

As seen in Figure 8, our line detector was largely successful at identifying the edges of the Rubik's cube; the blue lines overlap with the positive events (colored black). For this demo, we merely moved the Rubik's cube from left to right, so we did not observe any horizontal lines (as they were parallel to the direction of movement). We leverage this line detector later on in our development of a 3D tracker.

4.2. Pinhole Camera Model

Fundamental to computer vision, the pinhole camera model approximates how real-world cameras work by mathematically describing the relationship between points in a 3D space and their corresponding projections onto a 2D image plane. In this model, there is a small aperture or "pinhole" and an image plane positioned at a distance f (the focal length) away from it. The pinhole is the optical center *c,* and it is the entry point for light rays from the 3D object before being projected onto the image.

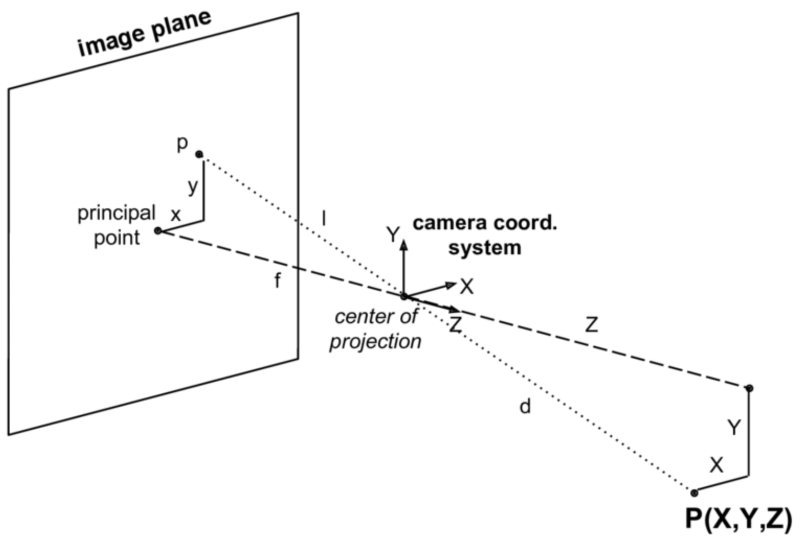


Figure 9. Pinhole camera model [8]

Given that our goal is to track the 3D pose of the cube, we fittingly used the pinhole camera model and its corresponding OpenCV implementations for camera projective math.

4.3. Obtaining camera intrinsics

Finding camera intrinsic parameters, or its camera matrix and distortion coefficients, is important for any type of work involving camera projection. The camera matrix is as follows:

$$K = \begin{vmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix}$$

where (f_x, f_y) is the focal length and (c_x, c_y) is the optical center of the camera. Distortion coefficients are parameters used for correcting both radial and tangential distortion. Radial distortion occurs when straight lines appear curved, while tangential distortion arises from the lens and imaging plane not being parallel. This comes from the inherent imperfections in lenses and other components. Typically, these coefficients are represented by the vector *d:*

$$d = \left[k_1, k_2, p_1, p_2, k_3\right]$$

where k are the radial coefficients and p the tangential.

The canonical procedure for obtaining these intrinsics is by capturing images of a checkerboard in various positions and orientations relative to the camera. The well-defined geometric structure of the board facilitates the detection of its features. The procedure can be summarized as follows:

1) Detect the 2D corner points on the image plane (OpenCV uses Harris corner detection). These are the observed 2D points.
2) Associate observed points with their corresponding 3D points in the world coordinate frame (based on the known checkerboard dimensions).
3) Estimate camera parameters by minimizing the reprojection error between the observed 2D points and ones from projecting the associated 3D points.
4) Use nonlinear least squares optimization to minimize the reprojection error between the known and estimated 2D points.

With an event camera, however, the checkerboard must be blinking in order to be visible for corner detection. We were able to calibrate our camera with a program provided in the Metavision SDK, obtaining the following intrinsics with an overall RMSE of 3.5 pixels:

$$K = \begin{vmatrix} 1730.0 & 0.0 & 298.0 \\ 0.0 & 1730.0 & 364.0 \\ 0.0 & 0.0 & 1.0 \end{vmatrix},\ d = \left[-.03,\ 0.16,\ 0.0,\ -0.06,\ 0.0\right]$$

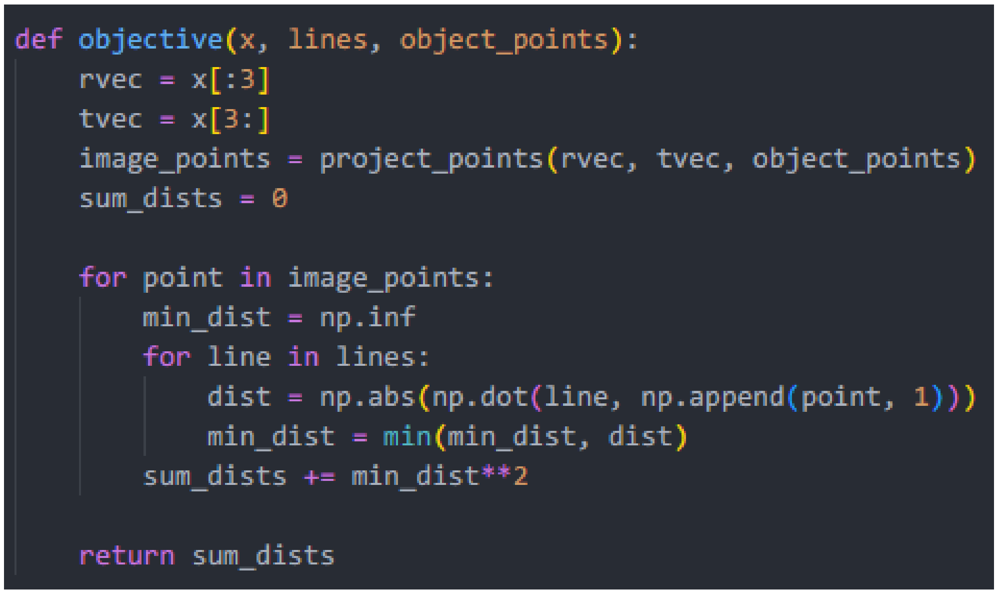
Moreover, we qualitatively observed that the reprojected points lined up well with the original checkerboard corners.

4.4. Cube Pose Estimation

The pieces for a cube tracker had now been assembled; we developed a robust line detector and obtained the intrinsic parameters of our camera. In a similar nature to the camera calibration process, our cube tracker was designed to leverage the inverse problem: finding 2D projections of the cube.

In this case, we provided (1) our known object points of the Rubik's cube, (2) our camera intrinsics, and (3) our guess for the 3D pose to OpenCV's projectPoints function to obtain 2D image projections. Similar to how camera calibration used the known checkerboard features as ground truth, we used our line detector.

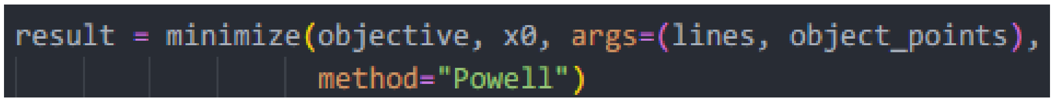
We defined an objective function that computed the squared distance of each projected point to every line and took the *x* as the set of variables to optimize. The vector *x* is precisely what we set out to find: the pose of the cube. Its first three elements are its rotation vector, and the last three elements are its translation vector.



```python
def objective(x, lines, object_points):
    rvec = x[:3]
    tvec = x[3:]
    image_points = project_points(rvec, tvec, object_points)
    sum_dists = 0

    for point in image_points:
        min_dist = np.inf
        for line in lines:
            dist = np.abs(np.dot(line, np.append(point, 1)))
            min_dist = min(min_dist, dist)
        sum_dists += min_dist**2

    return sum_dists
```



```python
result = minimize(objective, x0, args=(lines, object_points),
                  method="Powell")
```

Figure 10: Objective function for optimization

We then used SciPy to find the values of *x* which minimize the objective function. We opted to use the Powell method because it does not require the function to be differentiable. In doing this, we found the rotation and translation vectors which, when used for projecting from 3D to 2D, would yield image points along our detected lines.

4.5. Tracking simulation

Up to this point, we had developed a method for obtaining the pose of the cube given some initial guess for it. The concern we sought to address, however, was whether or not our optimization results for arbitrary initial guesses were accurate. We thus developed a simulation with synthetic data to test if the cube was trackable while moving. In Figure 11, the following is plotted:

- Green circles: initial points projected using an initial guess for the pose.
- Red circles: points projected using the true 3D pose with a small amount of Gaussian noise added.
- Purple circles: points projected using the optimized cube pose.

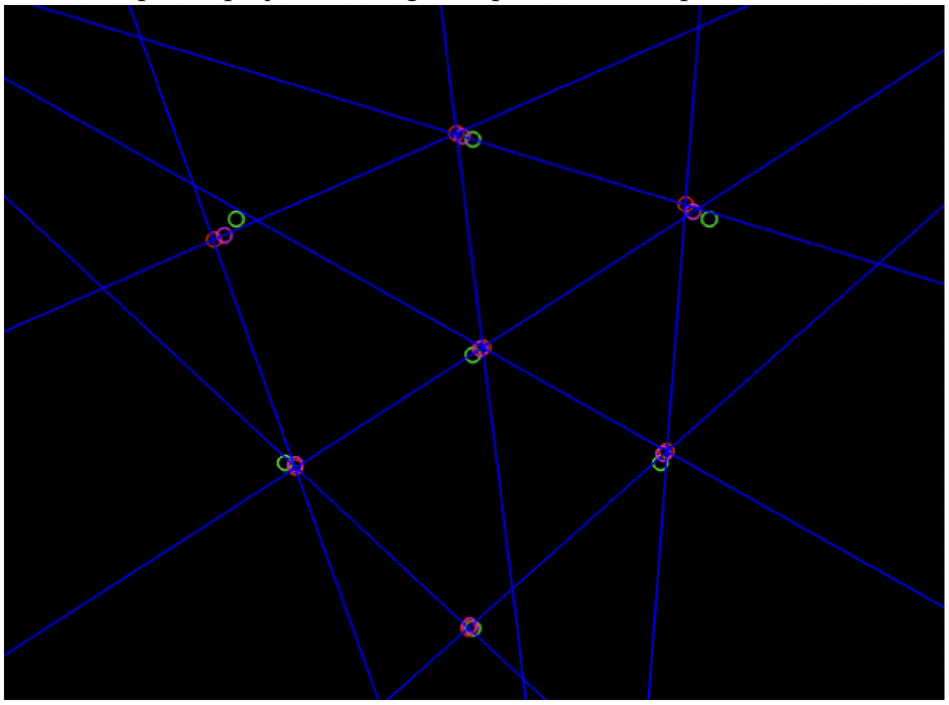


Figure 11: Synthetic cube tracking. Green: starter image points. Red: Perturbed points. Purple: Points from optimization.

As seen in this first trial, the optimized points did not line up with the perturbed ones. We then attempted adding three additional object points in between the corners and found success:

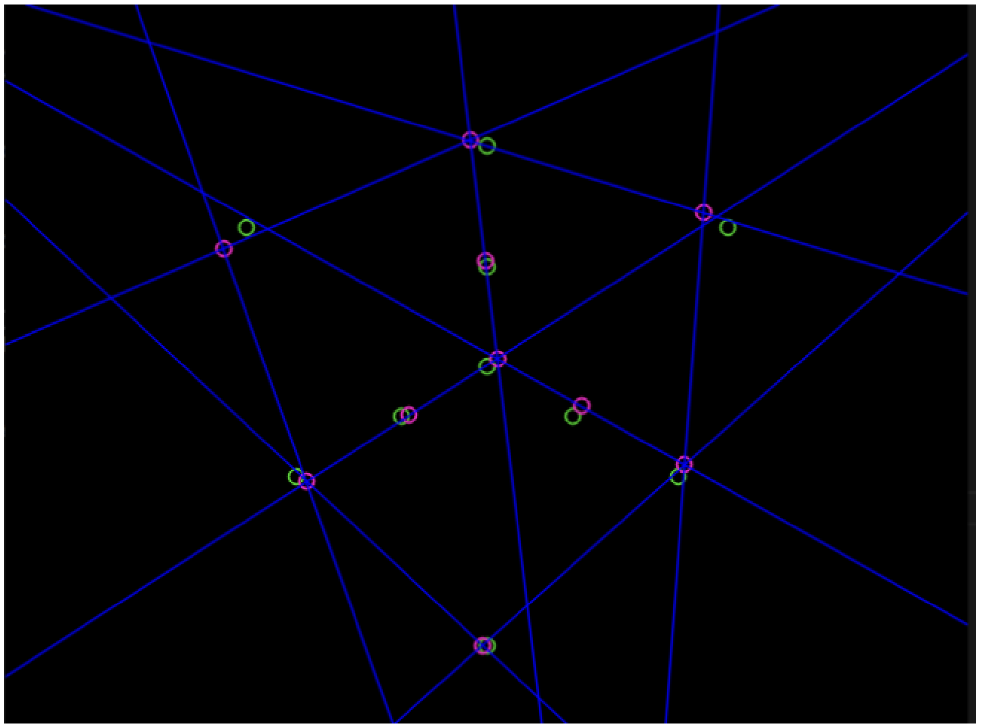


Figure 12: Synthetic cube tracking with additional points. Green: starter image points. Red/purple (overlapping): points from perturbed and optimized poses.

It is evident that with additional data, our optimizer was able to discover the pose corresponding with the correct projection despite the perturbations from the initial guess. It made the optimization a bit more specific in that it now restricted corner points to line intersections and middle points to just the lines themselves.

Overall, this indicated that the pose could be calculated in a live cube-tracking application, where the previously estimated pose is fed in as a guess for the current. This hinges on the requirement that the movement between frames is not exceedingly high.

5. Engineering Design

The constraints which governed this design problem were largely technical. Given that the event-based vision is more of a recent research topic, there was not much pre-existing literature or information regarding the best techniques for designing an object tracker. As discussed in section 2.2, the ghosting phenomenon we observed consumed several weeks of our time until a software update was released by our camera manufacturer. Thus, the most pressing constraint was the novelty of the camera itself.

Our requirement heading into the project was to develop subsystems and demonstrations along the way to keep us on a guided track. We performed many simulations doing relatively simpler tasks to guarantee the performance of certain components of the project. In evaluating our final solution, it is reasonable to say that we were mostly successful. While we did not get the chance to test the tracker on real data, we have sufficient reason to believe that it would be successful with further implementation.

By writing well-maintained and readable code, we upheld the integrity of our work in compliance with professional standards. Given the relatively low amount of physical components to this project, there were no real safety concerns or codes which are applicable. We furthermore did not survey any groups or record data from any external sources.

6. Conclusion

We set out on this project with the intent of developing a robust object tracker using an event camera. In doing so, we ended up designing a system that integrates several subsystems, including Metavision's SDK, Sony's IMX-636ES, OpenCV, and more. Through our iterative development, we built a 2D object tracker, a sensor model, and a proof of concept for a 3D tracker with promising results using synthetic data. Naturally, the strongest direction for future research is testing the 3D tracker with real data from the event camera. It would also be fruitful to investigate neural-network based approaches, as they are commonly used in conjunction with event cameras.

References

[1] Prophesee. “Event Cameras Represent a Shift towards Next Generation Machine Vision.” PROPHESEE, 22 Mar. 2022, https://www.prophesee.ai/2022/03/17/what-is-an-event-camera/.

[2] “Event Camera Evaluation Kit 4 HD imx636 Prophesee-Sony.” *PROPHESEE*, 28 Apr. 2023, https://www.prophesee.ai/event-camera-evk4/.

[3] “Metavision SDK.” Metavision SDK - Metavision SDK Docs 4.0.1 Documentation, https://docs.prophesee.ai/stable/metavision_sdk/.

[4] “Metavision Studio.” *Metavision Studio - Metavision SDK Docs 4.0.1 Documentation*, https://docs.prophesee.ai/stable/metavision_studio/index.html.

[5] Duda, Richard O., and Peter E. Hart. “Use of the Hough Transformation to Detect Lines and Curves in Pictures.” *Communications of the ACM*, vol. 15, no. 1, Jan. 1972, pp. 11–15, https://doi.org/10.1145/361237.361242.

[6] “OpenCV: Hough Circle Transform.” *Docs.opencv.org*, docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html.

[7] “Event-Based Vision Sensor（EVs）technology: Technology.” Sony Semiconductor SSolutions Group, www.sony-semicon.com/en/technology/industry/evs.html. Accessed 8 May 2023.

[8] Krylov, A. S., & Chubarov, A. S. (2018). A camera calibration method based on the single image of a sphere. Journal of Communications Technology and Electronics, 63(12), 1439-1446.