Project A: Event-based Tracking Using Kalman Filter and Particle Filter on Road-UAE Dataset

Submitted By:
Mohammed Tarnini
Department of Mechanical and Nuclear Engineering
100049735@ku.ac.ae

Submitted To:
Prof. Jorge Dias
Department of Electrical Engineering
jorge.dias@ku.ac.ae

***Abstract*—This study presents a generic vehicle tracking solution using Kalman and Particle Filters. The method is implemented on a video captured through a camera attached inside a running car in the highway traffic for vehicle motion detection. The system processes RGB video frames by converting them into event-based data that are analyzed through a combination of machine vision techniques: median filtering and binary thresholding. These preprocessing steps are followed by object tracking to ensure accurate detection. The Kalman Filter plays a key role in predicting and correcting the trajectories of vehicles, providing smooth and consistent position estimates even in dynamic scenarios. Particle Filtering on the other hand maintains a set of particles to represent the probable positions of tracked vehicles. However, rapid motion of vehicles relative to the camera brings in sudden shifts at the time of resampling. The proposed framework has proven highly effective for vehicle tracking with both appropriate reliability and accuracy. The results obtained confirm the capability of the system in handling the dynamic traffic scenarios of the real world, therefore, making it valuable in real-time applications for traffic monitoring and intelligent transportation systems.**

***Index Terms*—kalman filter, particle filter, tracking**

I. Introduction

Visual tracking is a crucial component of modern computer vision, with applications spanning autonomous navigation, video surveillance, and human-robot interaction [1]–[3]. It involves continuously identifying the position of a moving object in video data and monitoring its trajectory, providing valuable insights into dynamic environments and enabling rapid system responses. However, real-world tracking faces challenges such as fluctuating lighting conditions and object occlusions, which can hinder performance. To address these issues, various techniques have been developed, including training on object trajectories to predict movement patterns [4], [5]. Such training, typically conducted offline, lays the groundwork for effective real-time tracking in challenging scenarios.

In contrast, online tracking operates without prior training, relying on past positions and current observations to estimate the object's new position. This method is sensitive to noise in the observations, which may lower the reliability of the system and increase the possibility of losing the tracked target [1]. The common solution to noisy observations is the incorporation of knowledge about prior target paths, enhancing estimation accuracy [6]. One of the most usable tools for this purpose is the Kalman Filter, a mathematical method well known for its ability to estimate states in time-dependent systems. The Kalman Filter copes very well with dynamic environment challenges described by the motion equations, and its ability to predict future states based on current measurements makes it highly reliable for real-time tracking systems [7]. Beyond trackings, the Kalman Filter has demonstrated versatility in many applications, from the field of medical image processing [8], to machine vision [9], proving to be adaptable and robust when handling data which is usually noisy and complex.

Despite the effectiveness of the Kalman Filter, there are certain limitations in object tracking, such as dependence on prior knowledge of the target location and current measurement of its states. If the target becomes occluded by an obstacle, the tracking accuracy may deviate from the actual position. The Kalman Filter works in mainly two steps: process prediction and correction, as shown in Figure 1. In the prediction step, the state estimate is updated by projecting it forward in time using the system's dynamics and control input:

$$\hat{x}_k^- = F\hat{x}_{k-1} + Bu_k,$$

where $\hat{x}_k^-$ is the predicted state at time k, F is the state transition matrix, $\hat{x}_{k-1}$ is the previous state estimate, B is the control input matrix, and u_k is the control vector. The prediction covariance is updated as:

$$P_k^- = FP_{k-1}F^T + Q,$$

where P_k^- is the predicted covariance, P_{k-1} is the previous covariance, and Q is the process noise covariance. In the correction step, the Kalman gain is computed:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1},$$

where K_k is the Kalman gain, H is the observation matrix, and R is the measurement noise covariance. The state estimate is then updated as:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-),$$

where $\hat{x}_k$ is the updated state estimate and z_k is the measurement at time k. Finally, the covariance is updated as:

$$P_k = (I - K_k H)P_k^-.$$

Although these two steps improve the accuracy of the estimation, the fact that the Kalman Filter depends on a single

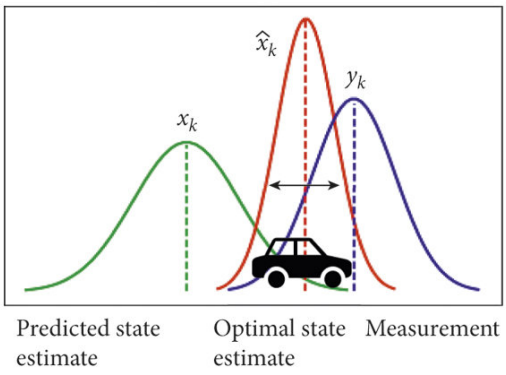

Fig. 1. Kalman Filter representation

frame makes it very vulnerable during occlusion [6]. Another disadvantage of the method is that it cannot incorporate the stored trajectories from the environment for fine-tuning the prediction. The tracking systems that have tried to overcome this depend on the stored paths capturing typical movement patterns within an environment, achieved through preprocessing, clustering, and modeling. Preprocessing often involves techniques like normalization and dimensionality reduction to standardize trajectories for easier processing [10].

In contrast, a Particle Filter is a much adaptive, non-parametric way that turns out to be highly potent for nonlinear and non-Gaussian dynamics. The tracker works well in scenarios when tracking may be complicated or non-regular. The basic idea behind the Particle Filter is the representation of probability distributions in the form of a number of particles, each corresponded to a particular position of the object. This enables the modeling of multimodal distributions and, by doing so, lets the filter keep multiple hypotheses on the object's state. Particle Filter is self-adaptive with respect to varying uncertainty, thus being inherently robust for dynamic scenes. It works by approximating the posterior distribution of the object state using a set of weighted particles, where the weights of these particles reflect the likelihoods of each particle representing the true state. In the predict step, each particle is updated and propagated forward based on the motion model, allowing the filter to predict potential future states of the object. This particle-based approach enables the filter to cope more effectively with abrupt changes or uncertainties in the object's motion.

$$x_k^{(i)} = f(x_{k-1}^{(i)}, u_k) + \epsilon_k^{(i)},$$

where $x_k^{(i)}$ is the state of the i-th particle at time k, $f(\cdot)$ is the motion model, u_k is the control input, and $\epsilon_k^{(i)}$ is the process noise for the i-th particle. In the update step, the weight of each particle is computed based on the likelihood of the measurement z_k given the particle's predicted state:

$$w_k^{(i)} = p(z_k|x_k^{(i)}),$$

where $w_k^{(i)}$ is the weight of the i-th particle. The particles are then resampled based on their weights to focus on the regions with higher likelihood:

$$x_k^{(i)} \sim \text{Multinomial}(w_k^{(i)}).$$

Finally, the state estimate is computed as the weighted average of the particles:

$$\hat{x}_k = \frac{1}{N} \sum_{i=1}^{N} w_k^{(i)} x_k^{(i)},$$

where N is the total number of particles.

As particles are weighted in respect to their likelihood, resampling guarantees the fact that particles with higher weights may be emphasized in the next iteration. Particle Filter has become very popular for applications where tracking accuracy is preferred over computational efficiency since it can handle abrupt changes in motion, appearance variations, and partial occlusions. However, Particle Filter requires heavy computation, especially as the number of particles grows larger, which may impact real-time performance.

This study focuses on examining and comparing the effectiveness of Kalman Filter and Particle Filter methods within the context of visual tracking. The objective is to evaluate their adaptability and determine the optimal choice of filter based on computational efficiency and tracking precision tailored to various applications. By exploring the trade-offs inherent in these two approaches, the study aims to provide valuable insights that guide their implementation in diverse real-time tracking systems. The study also offers an in-depth understanding of the strengths and limitations of both filters, highlighting how these factors influence performance across different scenarios. Additionally, the findings aim to inform decision-making regarding the appropriate filter to achieve specific objectives, whether prioritizing computational speed or tracking accuracy.

The structure of the study is as follows: Section II provides an overview of the methodologies employed in the analysis. Section III presents the experimental findings and interprets their implications. Finally, Section IV outlines the conclusions drawn from the study and proposes directions for future work.

II. Methodology

A. ROAD-UAE Dataset

ROAD-UAE Dataset consists of 20 videos that are all 3 minutes long. These videos are captured from a moving vehicle mainly on the highways of Abu Dhabi and Dubai cities in United Arab Emirates. Variety of cars, buildings, signs, and pedestrians are captured in those videos making the dataset ideal for tracking multiple objects. This dataset can also be used to train tracking methods that can be used on autonomous cars. Figure 2 shows a frame from on of the videos of ROAD-UAE dataset.

To obtain events that will be used for kalman filter tracking and particle filter tracking, v2e toolbox [11] was used. Firstly, the default parameters of v2e toolbox were used to convert the

Fig. 2. Frame taken from one of the videos of ROAD-UAE dataset

videos of the dataset to events. These parameters are shown in Table I. The output events video when using these parameters is shown in Figure 3. It is clear that the events are very noisy from using the default parameters.

TABLE I
DEFAULT PARAMETERS OF THE V2E TOOLBOX

Parameter	Value
dvs_exposure	0.033
input_frame_rate	30
input_slowmotion_factor	1
pos_thres	0.2
neg_thres	0.2
sigma_thres	0.03
cutoff_hz	30
leak_rate_hz	0.1
shot_noise_rate_hz	5

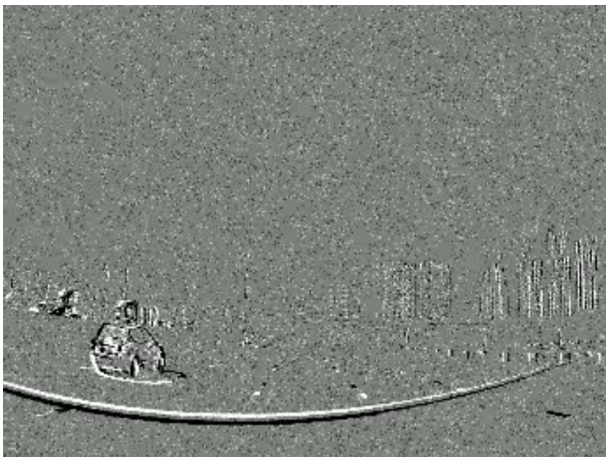

Fig. 3. Noisy event from using the default parameters of v2e toolbox

So, the parameters of v2e were adjust as seen in Table II to give cleaner events that can be used for tracking. The cleaner event obtained from the adjusted parameters is shown in Figure 4. It is clear that from these events, there are only white and black pixels where there are objects, unlike the noisy events where there are white and black pixels spread across the whole frame. This particular information will be vital for detection of objects seen in the next sub-section.

TABLE II
ADJUSTED PARAMETERS OF THE V2E TOOLBOX TO GIVE CLEANER EVENTS.

Parameter	Value
dvs_exposure	0.033
input_frame_rate	30
input_slowmotion_factor	1
pos_thres	0.3
neg_thres	0.3
sigma_thres	0.07
cutoff_hz	10
leak_rate_hz	0.03
shot_noise_rate_hz	0.5

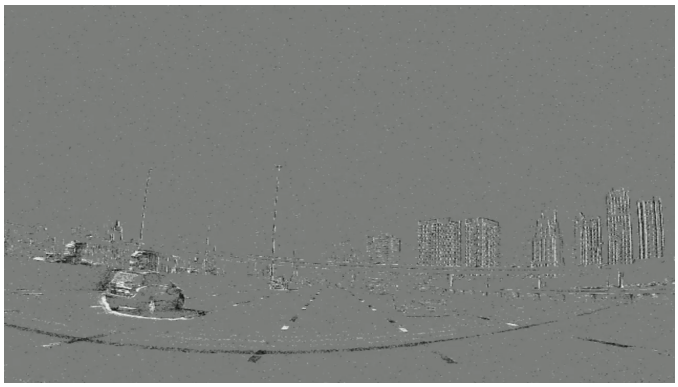

Fig. 4. Cleaner event from using the adjusted parameters of v2e toolbox

B. Detection using Machine Vision Techniques

The process of detection starts with the application of denoising to the frames of the event video. A 2D median filter *medfilt2* is used in order to remove the remaining salt-and-pepper noise in events videos. The median filter works as a smoothing operator by replacing the value of each pixel by the median value of neighboring pixels; hence, this operation does not blur the image much. This step is crucial since noisy data usually reduces object detection accuracy significantly.

After denoising, the frame is then transformed into grayscale by the *rgb2gray* function. This will make things easier for analysis since color information may not be required to detect the objects. Working with grayscale images reduces computational complexity because a grayscale image has only intensity values that are important for foreground detection.

A thresholding operation is done to separate the objects from the background. In this step, pixels with grayscale values above a certain threshold are classified as part of the foreground and set to white (binary value 1), while those with values below the threshold are classified as part of the background and set to black (binary value 0). This results in a binary image where the foreground objects are highlighted. The thresholding equation is:

$$\text{binaryFrame}(i,j) = \begin{cases} 1, & \text{if p}(i,j) > \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

Figure 5 shows a frame after the preprocessing methods were implemented on the events. After the preprocessing steps, object detection is carried out on the binary image. Th *bwlabel* function labels the connected components of the binary image. Each connected region is assigned a unique label, which corresponds to a potential object or event. This labeling step is crucial for identifying distinct objects that may be tracked over time. Once the regions are labeled, the *regionprops* function is used to extract key properties of each labeled region. Specifically, the centroid and area of each region are calculated. The centroid provides the location of the object in the image, which is essential for tracking its movement across frames. The area represents the size of the object, which is used to filter out irrelevant small regions. By applying an area threshold, only objects larger than a certain size are considered valid detections. This helps to eliminate noise caused by small, insignificant regions that could otherwise lead to false detections.

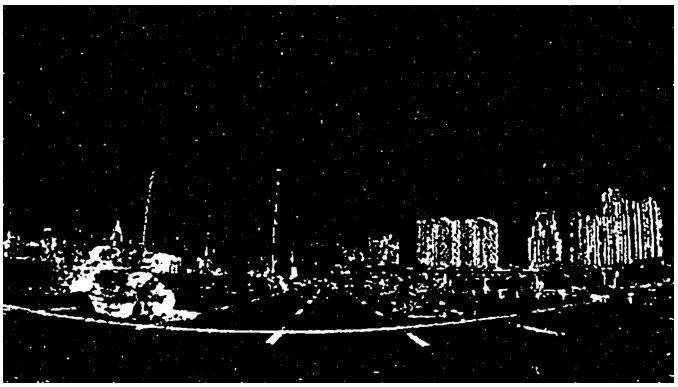

Fig. 5. Preprocessed event for detection and tracking

C. Kalman Filter Tracking

Each detected vehicle was tracked with the *vision.KalmanFilter*, initialized with a *ConstantVelocity* model. This model works well for those situations where the object motion is nearly non-accelerating, like that of vehicles on the highway. The Kalman Filter provides an estimate of what the next state an object will be in, given its current position and velocity, when only position measurements are available. Each filter was configured using the function *configureKalmanFilter* with a setup of initial parameters for state transition, measurement noise, and process noise. A small measurement noise covariance *[1 1]* was used to allow little inaccuracy in the measured values, while the process noise covariance *[0.1 0.1]* allowed for small potential changes in the object motion, such as acceleration or deceleration.

In every frame, it was the job of the *predict* function to estimate its future position based on this state. If a matching detection was found, the refinement of this prediction came through in updating the filter's estimate via the *correct* function. Predictions were matched with any new detections within a 20-pixel threshold of distance. If an object's detection fell within that, the filter's state was to be updated. In cases where no matching detection was found within the threshold, it relied solely on prediction by the Kalman Filter and incremented an age counter, tracking the continuity of the track for persistence.

Any detections that were not assigned to an already existing track in each frame were initialized with new Kalman Filters using the *configureKalmanFilter* function. This process added newly detected vehicles entering the scene to the list of tracked objects. Conversely, filters corresponding to tracks that failed to find a match over a defined number of consecutive frames (*maxTrackAge = 10*) were deleted. Such removal reduced computational overhead by removing inactive tracks and hence concentrated resources on active and relevant objects in the scene. This systematic approach enabled the effective and efficient tracking of vehicles in dynamic environments.

D. Particle Filter Tracking

For tracking the detected objects, a particle filter was initialized and maintained for each vehicle, whose data was stored in the *particleFilters* cell array. The filter started by scattering particles around the centroid of each detected object, following a normal distribution. Each object was represented by *numParticles* set to 100 and a *spreadRadius* of 5, forming a particle cloud near the object's initial position. This particle cloud approximated the object's location; since no prior information was present on the exact position of an object, all particles got equal weights at the initialization stage, as all were equally important. In each frame, the particles associated with an object were updated. For each particle, the Euclidean distance to the detected object's centroid was computed; this distance was used to compute the likelihood function. This probability was modeled as a Gaussian function of the distance, where the value is higher for particles closer to the detected object. The weights of the particles were updated based on this likelihood so that particles closer to the detection would have greater influence in representing the position of the object.

Next, the *randsample* function was used to resample the particles based on their updated weights. This step kept the particles close to the detected position and removed those that were far from it, thereby increasing tracking accuracy by condensing the filter around the true position of the object. To simulate movement, Gaussian noise was added to the positions of the particles using *noiseLevel*. Then, it calculates the weighted average of particle positions to get the estimated centroid for each object, yielding a robust position estimate in case of noise or occlusion. The function *insertMarker* was used for visualization purposes to mark the detected object centroid in green, the individual particles in red, and the estimated particle centroid in blue. This visual feedback showed how the particles are distributed, clustered around the

estimated position, and hence contributed to the tracking of each vehicle.

III. Results and Discussion

Results across Figure 6 illustrate how effective Kalman Filter has been able to predict vehicle trajectories and positions for some static building locations; there is strong consistency of the predicted position compared with the true paths in this object. A small lag exists between the predicted and corrected position, which characteristically relates to the inner update nature of the Kalman Filter. This lag occurs because the filter takes some time to incorporate new measurements in order to fine-tune its predictions.

However, despite this slight lag, the general tracking accuracy is still quite high and rarely deviates from the actual positions. This further consolidates the strength and viability of the Kalman Filter in handling dynamic tracking situations. Its performance, as seen above, ensures that it can function well in applications that involve the tracking of moving objects in real-time environments.

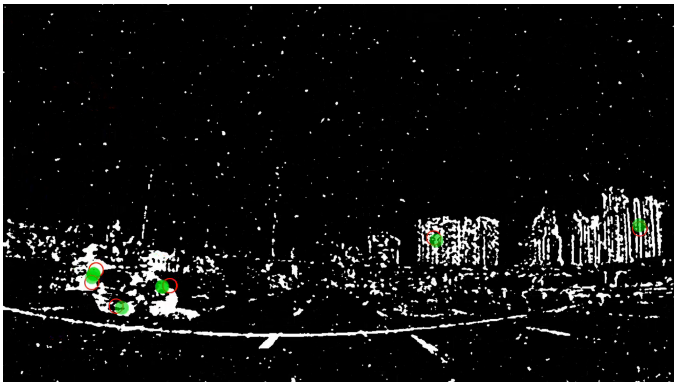

Fig. 6. Kalman Filter tracking on a preprocessed event frame. red circle: predicted position, green circle: corrected location

Figure 7 shows the unprocessed video frames with detected masks overlayed to highlight regions of interest as determined by the detection algorithm. For better visualization, masks are overlayed as semi-transparent to clearly see how well the detections fit into the raw footage. This visualization serves as a means of validation that the detection process can be transformed back to the original video before converting it to frames. The overlay gives an idea of the spatial coverage and precision of the masks regarding the exact capture of intended objects by the system without false positives or missed detections. In this figure, mask regions are directly compared to unprocessed background parts. A demonstration video that shows original video, preprocessing, tracking and overlaying back to original video is attached with this report called **Kalman_Filter.mp4**.

For particle filter, the median filter used had a larger size of 5x5 to ensure that there is no salt and pepper noise on the frames. This is because the scattered small white pixels were massively impacting the performance of the particle filter. Figure 8 shows the performance of particle filter tracking on

Fig. 7. Tracking markers from events being overlayed on the original video

the preprocessed frame. In the process, the filter tracked the position with an acceptable precision. However, it is clear that there is a neighborhood of particles at areas where there is no visible object to be detected. This can be explained by the fact that same objects enter and leave the frame very fast, so particles take some time to readjust to visible objects due to resampling by the filter. Hence, the particle filter was more inaccurate when compared to the smooth tracking done by the Kalman Filter.

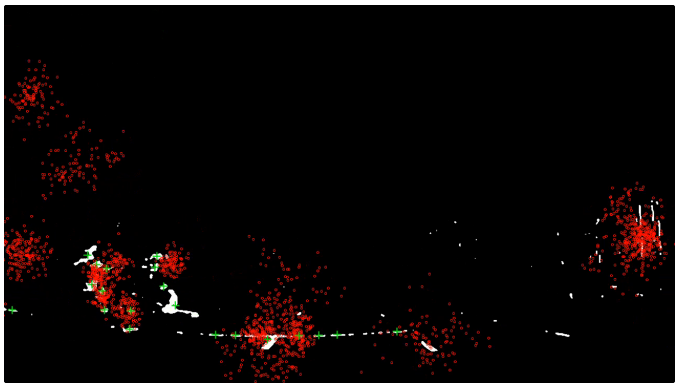

Fig. 8. Particle Filter tracking on preprocessed video. red dots: particles, green cross: detection from binary threshold.

Similar to Kalman Filter tracking, the markers were overlayed successfully back to the original video before converting it to frames. This is seen in Figure 9. The inaccurate position of particles seen on the left might be due to the presence of road lights or signs on the left in some frames. This causes the particles to cluster in that area expecting an object, but it does not occur frequently. In addition to that, the Particle Filter tracking took about 3 times more time than Kalman Filter tracking when both are implemented on the same three minutes video. This further proves the superiority of the Kalman Filter over the Particle Filter in this specific application. A demonstration video that shows original video, preprocessing, tracking and overlaying back to original video is attached with this report called **Particle_Filter.mp4**.

Fig. 9. Tracking markers from events being overlayed on the original video

IV. Conclusion

This work presented a broad application of tracking algorithms using the ROAD-UAE dataset, which is a diverse collection of videos captured on highways in Abu Dhabi and Dubai. The richness of this dataset in objects, such as vehicles, buildings, signs, and pedestrians, makes it ideal for the training and testing of object detection and tracking methods in autonomous systems. Event-based data created with the v2e toolbox provided the background to the implementation and comparison of different tracking techniques, namely the Kalman filters and particle filters.

Results showed that the Kalman filter was better in estimating vehicle trajectories. The predictability of future states with effective correction mechanisms resulted in continuous and correct object tracking, even in dynamic conditions. On the other hand, the particle filter presented higher levels of noise in the estimates. Noisy output from the particle filter, added to its computational overhead, resulted in its lower efficiency in this application compared to the Kalman filter.

These findings raise the importance of choosing appropriate tracking methods according to specific tasks. The strong performance of the Kalman filter suggests that this could be a suitable approach in real-time applications for autonomous driving, where accurate and efficient object tracking is crucial. Future work should focus on advanced hybrid methods that integrate the strengths of both approaches to further enhance tracking performance in more challenging conditions.

References

[1] S. R. E. Datondji, Y. Dupuis, P. Subirats, and P. Vasseur, "A survey of vision-based traffic monitoring of road intersections," *IEEE transactions on intelligent transportation systems*, vol. 17, no. 10, pp. 2681–2698, 2016.

[2] X. Zhao, D. Dawson, W. A. Sarasua, and S. T. Birchfield, "Automated traffic surveillance system with aerial camera arrays imagery: Macroscopic data collection with vehicle tracking," *Journal of Computing in Civil Engineering*, vol. 31, no. 3, p. 04016072, 2017.

[3] J. Jung, I. Yoon, and J. Paik, "Object occlusion detection using automatic camera calibration for a wide-area video surveillance system," *Sensors*, vol. 16, no. 7, p. 982, 2016.

[4] B. T. Morris and M. M. Trivedi, "Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 11, pp. 2287–2301, 2011.

[5] A. Alahi, V. Ramanathan, K. Goel, A. Robicquet, A. A. Sadeghian, L. Fei-Fei, and S. Savarese, "Learning to predict human behavior in crowded scenes," in *Group and Crowd Behavior for Computer Vision*. Elsevier, 2017, pp. 183–207.

[6] F. Farahi and H. S. Yazdi, "Probabilistic kalman filter for moving object tracking," *Signal Processing: Image Communication*, vol. 82, p. 115751, 2020.

[7] M. Mehta, C. Goyal, M. Srivastava, and R. Jain, "Real time object detection and tracking: Histogram matching and kalman filter approach," in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, vol. 5. IEEE, 2010, pp. 796–801.

[8] J. Bersvendsen, F. Orderud, R. J. Massey, K. Fosså, O. Gerard, S. Urheim, and E. Samset, "Automated segmentation of the right ventricle in 3d echocardiography: a kalman filter state estimation approach," *IEEE transactions on medical imaging*, vol. 35, no. 1, pp. 42–51, 2015.

[9] M. Marshall and H. Lipkin, "Adaptive kalman filter control law for visual servoing," in *2016 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2016, pp. 379–386.

[10] M. Briers, A. Doucet, and S. Maskell, "Smoothing algorithms for state–space models," *Annals of the Institute of Statistical Mathematics*, vol. 62, pp. 61–89, 2010.

[11] Y. Hu, S.-C. Liu, and T. Delbruck, "v2e: From video frames to realistic dvs events," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 1312–1321.