Video Object Tracking & Kalman Filter Implementation

Shabbir, Abubaker [a]

University of London City, robotics, AI, and autonomous systems, Computer Science Department

Nomenclature

X	Position in meters
V	Velocity in meters per second
P	Covariance matrix, representing uncertainty in the state estimate.
K	Kalman gain, a factor that adjusts the estimate based on measurement uncertainty.
R	Measurement noise covariance matrix.
Q	Process noise covariance matrix.
$\hat{X}$	Predicted state estimate in the Kalman filter.
Z	Measurement or observed value
H	Measurement model matrix in the Kalman filter
DoF	Degree Of Freedom

RGB	Colour Space (Red, Green & Blue)
Kalman Filter	A recursive algorithm for estimating the state of a dynamic system from noisy measurements
Covariance Matrix	A matrix that expresses the variance and covariance of random variables
Object Tracking	A process of identifying and tracking the motion of objects in a video sequence
MATLAB	A programming environment used for mathematical modelling and algorithm implementation.

Abstract

This research examines two essential elements in contemporary signal processing and communications: video object tracking and the Kalman filter. The initial segment entails a comprehensive experiment in video object tracking, which includes selecting an appropriate film, implementing a specific tracking method, and analysing findings utilising MATLAB's Computer Vision Toolbox. We outline the procedure sequentially and evaluate the outcomes to assess the efficacy of the algorithm. The second section delves into the Kalman filter, starting with an explanation of covariance matrices and their significance in probabilistic scenarios. The notion of Kalman gain is examined, along with illustrations of its real-world applications. The Kalman filter mathematically and computationally extends the input aircraft location and velocity data.

MATLAB will be used to simulate and verify the extended positions and velocities.

This coursework illustrates the actual applications of advanced signal processing techniques, emphasising their significance in areas such as surveillance, navigation, and predictive modelling.

1. Introduction, Part 1: Video Object Tracking

"Formula 1 is the pinnacle of motorsport and the world's most prestigious motor racing competition... making daring decisions in the blink of an eye—and at 370 km/h." *(Everything You Need to Know about F1—Drivers, Teams, Cars, Circuits, and More | Formula 1®, n.d.)*

Formula 1, a multibillionaire industry, is described as the "pinnacle of motorsport" through its numerous events being held worldwide and high-end car manufacturers coming together, like Mercedes and Aston Martin, to run their best vehicles to the test.

It is almost crucial that technology is up to par, given all the investment from car manufacturers themselves as well as F1 fans. Currently, to this date, F1 has employed the following technologies in regard to tracking: transponders, timing loops, GPS receivers, and computer vision. (*How F1's Timing Tech Works and Why It's Crucial for Racing Accuracy*, n.d.)

Going into computer vision in more detail: *"An algorithm that uses pixel counting to determine if a car has exceeded track limits."* In this specific part of the project, the focus will lie on creating an object-tracking system that can identify and monitor the motion of these motorsport vehicles. This will propose an alternative to the many methods F1 has already employed for motorsport tracking.

This system will be primarily set up on MATLAB and will make use of the Computer Vision Toolbox functions. Prior to executing the racetrack, a basic video of a ball movement, obstructed by a paper, will be initiated first to ensure the code type used is the most optimal from the selection below, as well as to ensure the code delivers expected results.

1.1 Testing Phase

The motorsport presents only 1 DoF, that being translation along the x-axis, whereas the motorsport video will show 3 forms of translation (x, y, and z). Hence the code is predicted to handle the ball movement almost flawlessly, whereas the motorsport will present some flaws, which are later dived into further.

The following test is influenced by MATLAB's tutorials in object tracking. (*Object Tracking | Student Competition: Computer Vision Training - YouTube*, n.d.-a)

Here is a frame walkthrough of the ball's movement:

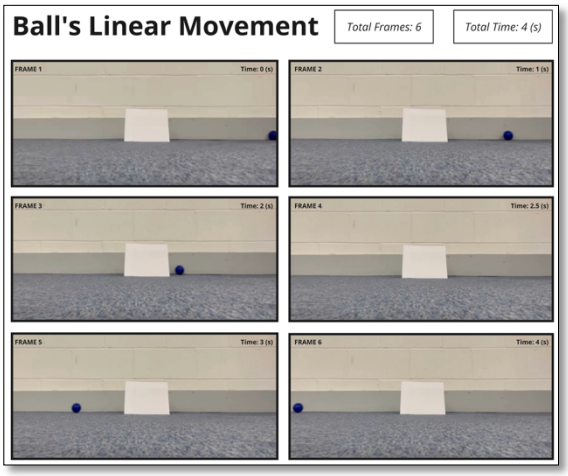


Figure 1. Ball's Linear Translation

1.2a - Histogram-Based Ball Tracking

Initially, the ***'histogram-based tracking'*** system is to be employed in MATLAB. To understand its full nature, its iterative process is explained as follows in 4 steps:

[1] The ball to be tracked is represented as an *pixel histogram*, which can either be represented via an RGB colour space.

[2] A *density map* is created, which has elements of higher weight, signifying pixels are similar to the ball's histogram, and lower weight, being pixels dissimilar to the object's histogram. *Note: every pixel is assigned a weight.

$$x_{center} = \frac{\sum(x_i - w_i)}{\sum w_i} \qquad y_{center} = \frac{\sum(y_i - w_i)}{\sum w_i}$$

- X_i & Yi: Pixel Coordinates
- Wi: individuals pixels weight
- ∑Wi: Total sum of weight in each search window

Figure 2. Pixel Weight Centroid Formulas

[3] *Computation of centroid* (centre of mass) of the weighted pixels. This specific value is where the pixel density is maximised; also referred to as the weighted sum. On computation of centroid, the program dynamically shifts the centroid search window accordingly.

[4] This *iterative process* is executed continuously until the search window converges to a stable position, being the centre of the densest region of pixels.

Continuously adaptive mean shift
This is an advancement to a technique named mean shift algorithm' that possesses an iterative nature of shifting a search window to the densest region of pixels at a fixed size. The advancement is in how this adaptative method resizes the search window dynamically; this allows the program to consider the object shrinking (moves further away from reference), growing larger (moves closer to the reference point), and most importantly, rotation.

1.3a: Histogram Tracking Results

Below presented are the results obtained via histogram tracking as well as a flow chart [appendix] on how the MATLAB algorithm operates:

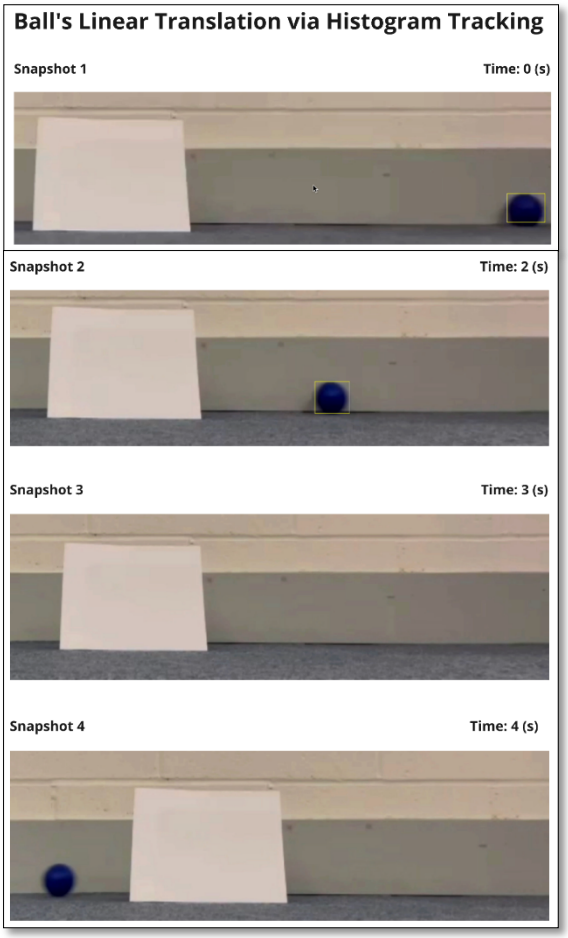


Figure 3. Histogram Tracking Results

The following flowchart, presented in figure 4 on the appendix page, illustrates the workflow of the histogram tracking program. The algorithm works by initially allowing the user to create a bounding box. As the tracking phase begins, it will iteratively process each video frame and determine whether the tracking of the ball is still occurring. If tracking confidence is high, the object is marked with a bounding box; if not, the program will attempt to recover via segmenting the frame (isolation of specific parts) to detect the ball again. Given the re-detection is successful, the tracker is re-initialised, and tracking presumes, if not that a particular set of frames is left without a bounding box.

As shown per snapshots 1 and 2 in Fig. 3, the algorithm successfully is able to track the ball until the paper obstruction, which

then, upon passing the paper and making its way to the other side, fails to re-track.

Snapshot 4 is where this method showcases its dependability on colour recognition.

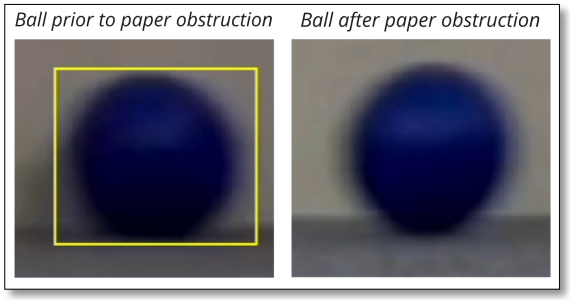


Figure 4. Ball tracking close view

Fig. 4 illustrates Snapshot 4 zoomed in before and after the paper obstruction. Visible to the naked eye, there is a change in brightness, which seems to be a lighter variation after the paper obstruction.

The alteration in brightness and possible fluctuation in the ball's colour appearance impacts the efficacy of the histogram-based tracker, influenced by the room's lighting. Histogram-based techniques depend on stable colour distributions to ensure tracking (SpringerLink, 2019). When brightness fluctuates, the ball's colour histogram fails to align with the tracker's reference, resulting in diminished confidence and eventual tracking loss. Furthermore, occlusion from the paper temporarily obscures the ball, impairing the tracker's capacity to juxtapose the frame with the original histogram *(Meshgi, 2015)...*

A study on managing partial occlusion says that histogram-based trackers aren't very good at dealing with changes in brightness and depend a lot on features that are visible and stay the same. This makes it hard to re-detect after occlusion. *Chu et al. (2015)* The tracker fails to retrieve the ball due to a diminished variation in brightness and occlusion, both of which distort the target's original histogram.

Even a partial reemergence (successful retracting) can lead to failure if the tracker is unable to distinguish the target from the surrounding background or variations in lighting conditions; this here seemed to be the case. *(Lim et al., 2015)*

Below Fig. 5 presents a schematic of the confidence score, presenting failure in re-tracking once the ball passes the paper and out.

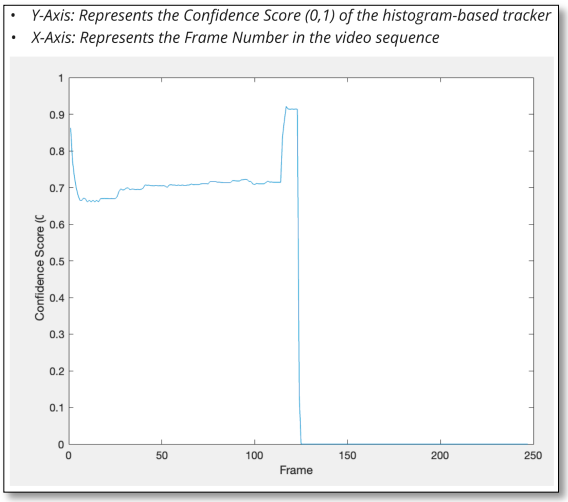


Figure 5. Tracker confidence over time per frame

[a] Frames 0–5, the confidence score decreases from a high value (~0.85) to approximately 0.7, likely due to the tracker's initial adjustments as it commences locating and tracking the ball.

[b] Frames 20-110, the confidence score remains consistently stable at approximately 0.7, indicating that the tracker effectively monitors the ball during this interval, with negligible fluctuations in appearance or movement.

[c] Frames in the region of 110, the confidence score markedly rises to approximately 0.95, suggesting that the tracker realigns more precisely with the ball, potentially due to enhanced visibility or lighting conditions.

[d] Frames 120: Final frame, the confidence score declines to zero,

indicating that the tracker has entirely lost sight of the ball, potentially due to total occlusion and variations in background noise and lighting.

In summary, the histogram tracking method failed to re-track due to segmentation failure (background noise/lighting). Its inability to predict the position of the ball in the future led to the failure of re-tracking after the paper obstruction. Hence, the next method presented takes future predictions of the ball trajectory into account.

1.2c - Kalman-Filter Ball Tracking

The Kalman filter consists of 2 major steps: (*Object Tracking | Student Competition: Computer Vision Training - YouTube*, n.d.-a)

[1] *Prediction* - The algorithm will consider the trajectory and movement (motion model) as well as previous states to predict the object's next move (current states); this, in return, offers more accurate tracking. The motion model may present itself to have either a constant velocity or constant acceleration, which in the ball translation movement isn't the case, as the ball over time will reduce in speed at a non-constant rate due to environmental factors. This comes along with the problem of inaccurate predictions. In contrast, given the racetrack is animation-based, a constant velocity (m/s) and acceleration of 0 (m/s^2) is guaranteed.

[2] *Correction* – The Kalman filter will utilise an object detection mechanism in order to correct itself, where it will iteratively return feed the algorithm with information on the centre of the object. Here specifically is a part where it contrasts with the histogram-tracking method, as it feeds back the algorithm with the centre of the object rather than the centre of the bounding box.

1.3d - Kalman-Filter Tracking Results

Below presented are the results obtained via Kalman tracking a flow chart [appendix] as well as how the MATLAB algorithm operates:

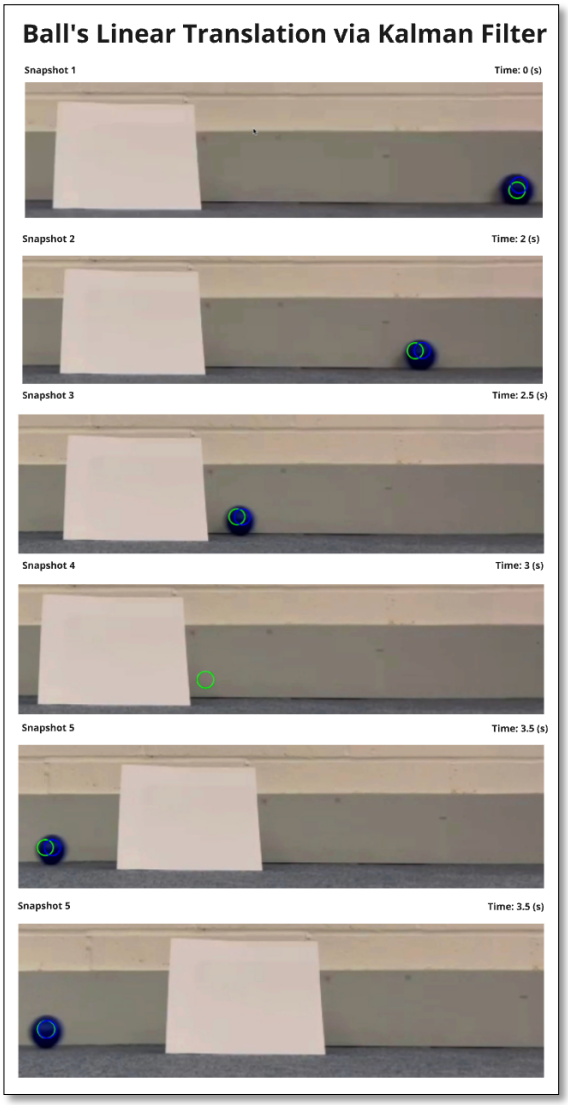


Figure 6. Kalman Filter Tracking Results

Fig. 6 presents 5 snapshots of the balls linear translation. Each snapshot consists of 2 circles, which have been assigned a colour. Blue has been assigned to mark the present positioning of the ball (via histogram-based tracking), whereas green marks the predicted position of the ball (via Kalman filter).

This algorithm successfully retracks after the paper obstruction, which was a flaw in histogram-based tracking alone. Snapshots

5 and 6 clearly show how the Kalman filter aids histogram-based tracking via both blue and green circles appearing.

The distinction between the Kalman filter and histogram-based tracking flowcharts (appendix page) will aid in explaining how the program has changed. Steps [1-4], as explained on pages 2-3, are present; given Kalman filter just acts as an optimisation to histogram-based tracking. The additional steps are presented below with respects to the original directory order [1-4]:

After steps [1-3], present the following:

[3a] Kalman filter prediction: We compute the centroid prior to this specific step and then feed it into the Kalman filter. The filter will detect the next position based on the motion model, in which this specific algorithm employs the constant velocity model. A green circle then visually illustrates this prediction. The velocity constant model necessitates fewer parameters and computations than the acceleration constant model, making it the most suitable option in this case. This is due to the simplicity of the ball movement, which is a single translational linear movement (Kalman Filter in One Dimension, n.d.).

[3b] Correction with detected position— Given the histogram has successfully identified the ball's centroid positioning, the Kalman filter will here attempt to further correct the accuracy based on the centroid and present the correction via a blue circle.

[3c] Tracking during occlusions: As a redundancy, if it's the case where step [3b] fails (centroid not found), the Kalman presents a detection based on its own mechanism; hence, tracking was achieved after paper obstruction.

[3D] Trajectory smoothing and output Given this video was recorded in comparison to the animation present in the next section, it was subject to lots of background noise, which the Kalman filter here will filter out, ensuring the algorithm is able to detect the ball's manoeuvrability more accurately.

Lastly, the iterative process of the above steps continues until the end of the video/frames. [4]

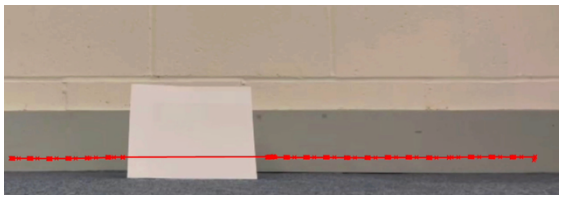


Figure 8. Ball Tracking Trajectory

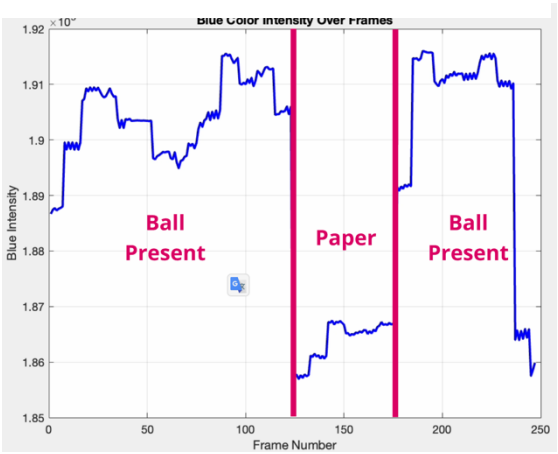


Figure 7. Blue intensity over time

Figs. 7 & 8 further reinforce the validity of the Kalman algorithm. Fig. 7 illustrates a red hue that is less defined on paper than it is outside the paper. It has also excluded the ball from this snapshot, clearly showing how it has been able to recognise it entirely. Figure 8 illustrates how the intensity of the blue colour decreases and then increases, demonstrating the algorithm's capability to re-track.

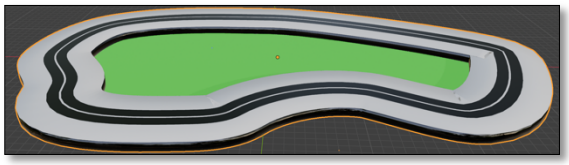
All in all, using the Kalman filter severely resulted in the ball's re-tracking. Moving forward, this method will be implemented into a more complex animation, the racetrack, given that it has non-linear motion and is suspectable to 3 DoF, and

the algorithm will now have to accommodate for multi-object tracking.

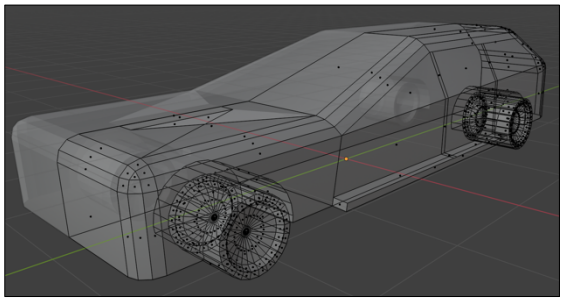
1.2e: Kalman Vehicle Tracking

Motorsport track video creation

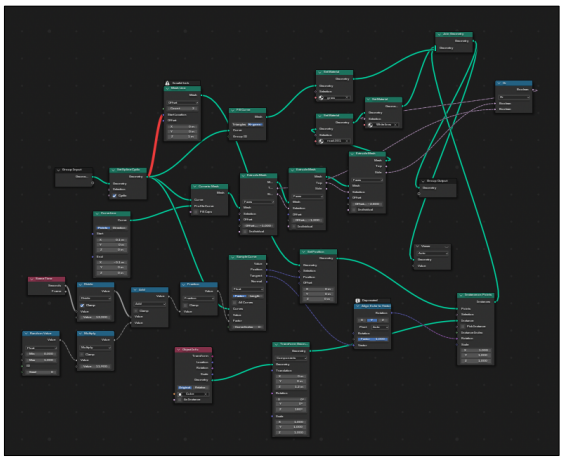
The animation consisted of 3 parts:
a) Track design/setup, as shown below:



b) Vehicle design/setup, as shown below:



c) Physics Setup:



The animation was exclusively created using the Blender program. We configured the physics to allow the cars to loop under a counter control, employing a constant colour gradient to generate various colours within the loop. Neurons operate the vehicle's mobility, monitoring boundaries along the track to ensure it stays on the correct path. Boolean functions arrange the vehicles in the correct order.

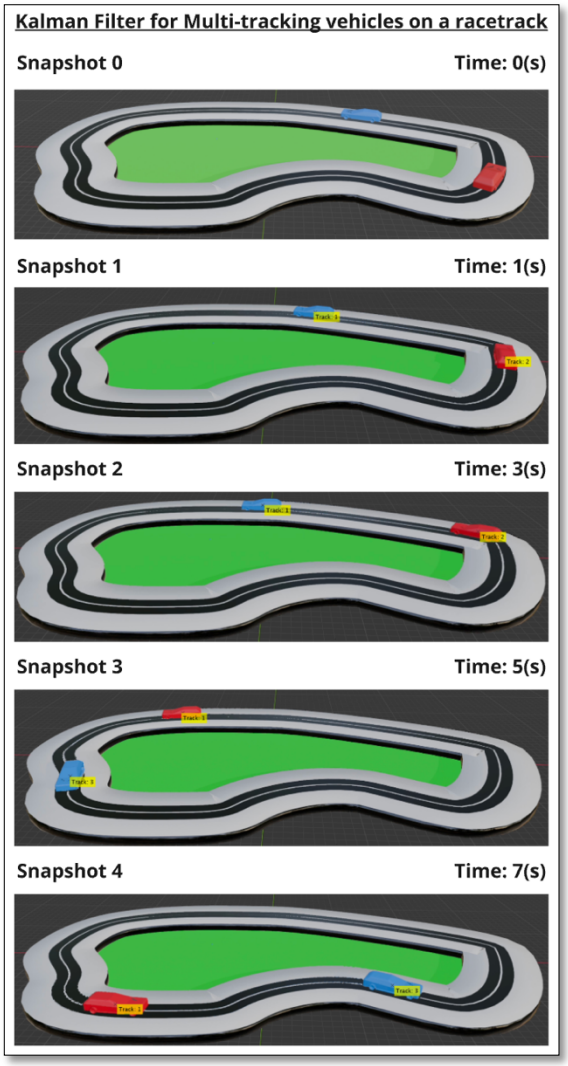


Figure 9. Racetrack Kalman Filter Multi Tracking Snapshots

As described above, this specific track will come with two additional complexities. The first is the non-linearity element, and the second is the program's ability to track multiple vehicles simultaneously.

To utilise the algorithm's multi-object tracking capability, one must implement a track management system. The tracks represent the 'hypothesised paths' of the vehicles over time, with each vehicle assigned its own track. The system works in the following order:

[a] Initialisation: When a vehicle is detected, a track is assigned to it, in which the track will record its position and velocity and give it a label in return: car [number].

[b] Prediction: Using the motion model (constant velocity), the Kalman filter will predict the vehicle's next frame.

[c] Association: A 'cost function' (the distance between the predicted and current positions) matches the vehicle's current position to existing tracks.

If the 'cost' or 'distance' falls within the range, we will update the track accordingly.When the cost exceeds the range, we create a new track. However, we delete the track if it fails to receive valid detections for a predefined amount of time.

[Refer to Appendix for full code]

1.3f: Kalman Vehicle Tracking Results

Here are the following tracking snapshots, on each of the 4 corners of the track:
Snapshots 1 and 2 -
The system assigns Track 1 to the blue car and Track 2 to the red car at Time 1–3s. The system confirms the successful initialisation of the Kalman filter and the accurate assignment of tracks to the identified vehicles.

Snapshots 3 and 4 –
Time 5s and Time 7s reassign the blue car to Track 3, while Track 2 disappears. This shows that the red car was lost and therefore, when re-tracked, assigned track 3. Snapshot 2 presents how the system eventually removed its track.

Track reallocations & removals –

[1] Track Deletion, Red Car:
Snapshots 1 & 2 originally designated the red vehicle as Track 2, but its inability to receive valid detections for 10 consecutive frames led to its un-assignment. Given this is an animation, potential complications such as occlusion (due to the rotational movement around the track) may have led to the track's consecutive invisible count exceeding the maxInvisibleCount threshold of 10 frames.

[2] New Track Creation, Blue Car:
For analogous reasons, the Kalman Filter temporarily unassigned the blue automobile, recognised it as a new entity, and initiated a new track (Track 3). The reassignment occurred due to the potential temporary loss of detection on the blue car's prior track (Track 1).

Despite this vigorous process, the algorithm still manages to successfully track both vehicles moving along the track. Further optimisations may be to use the acceleration model and allow the algorithm to have the ability to learn about a vehicle in 3 dimensions rather than the 2 dimensions, which led to slightly the creation of new tracks as per above.

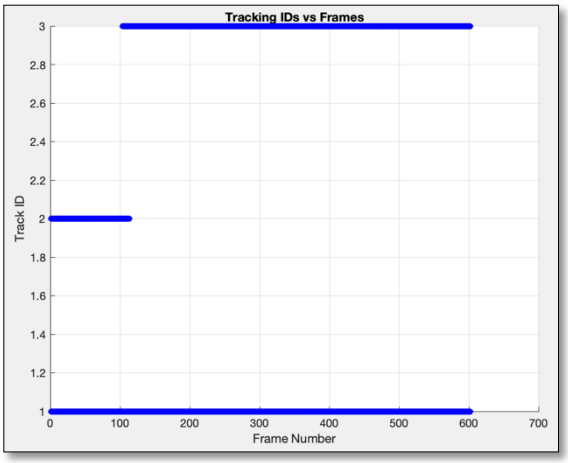


Figure 10. Tracks over time/frames processed

Fig. 10 graphically presents the creation of track 3 and deletion of 2, after each other, at frame 100. The overlap presents the efficiency in that track 3 is already deployed a few frames before the maxInvisibleCount threshold of 10 frames.

Fig. 11 presents a simplified flowchart in which the histogram-based tracking and Kalman filter initialisation and process are further explored in the appendix page. It also presents the ‘Gaussian blob' and the 2 ‘Morphological Operation’.

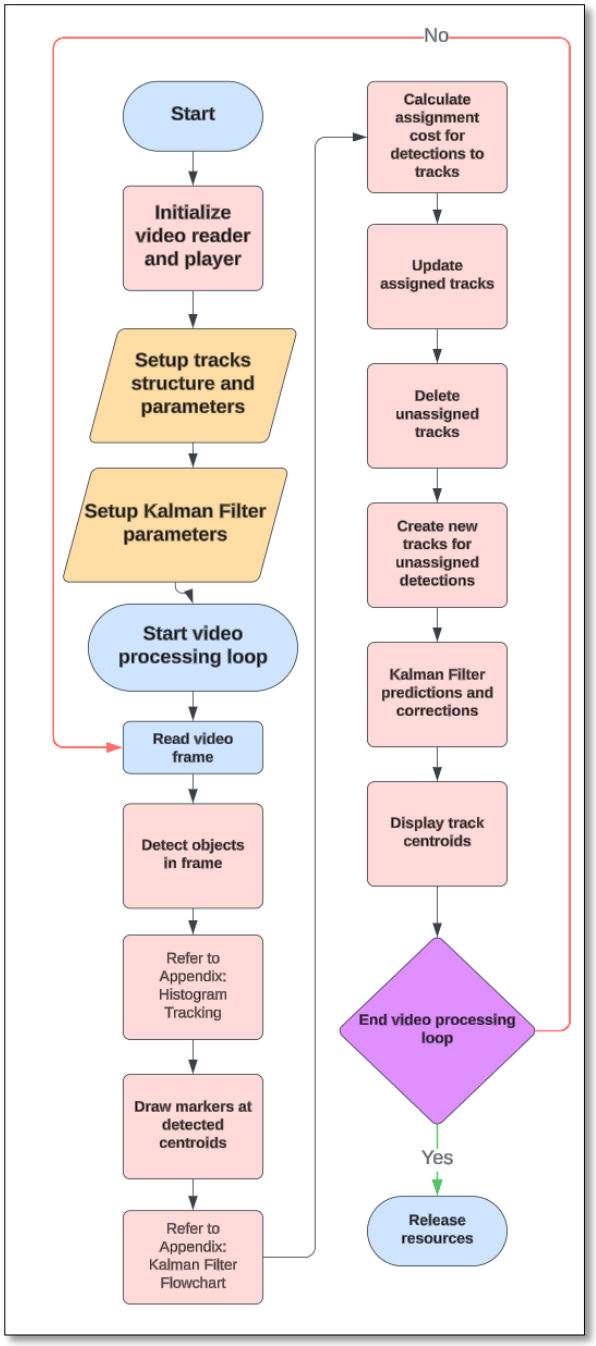


Figure 11. Kalman Filter Multi Tracking Flowchart

The parameter types [1-3] relevant to this flowchart are elucidated and detailed below:

[1] Kalman filter parameters;
a) Motion Model:
Constant Velocity
b) Initial Estimate Error:
Value: [1 x 10^6 1 x 10^6]
c) Motion Noise:
Value: [20 20]
d) Measurement Noise:
Value: 10

[2] Track management parameters;
a) Maximum invisible frames:
Value: 10
b) Cost of Non-Assignment:
Value: 50

[3] Gaussian Blob detection parameters;
a) Minimum Blob Area:
Value: 2000 (*Vision.BlobAnalysis*, n.d.)
* A blob is a region where pixels share a similar colour intensity; therefore, specific to this animation, it will represent the vehicles. Value 2000 for blob area is the minimum value the area should be to be considered valid; setting hue conditions as per below narrowed down the search of other colours and therefore slightly improved algorithm efficiency.
(BlobAnalysis, n.d.)

b) HSV Thresholds:
Blue Car: Hue: [0.55, 0.65], Saturation: > 0.3, Value: > 0.2
Red Car: Hue (lower range): [0.0, 0.05], Hue (upper range): [0.9, 1.0], Saturation: > 0.4, Value: > 0.2
* When running the algorithm without the hue, the grass patch in the middle was assigned a tracking value; hence, when applying the hues above, the algorithm narrowed down the colours to search for; an alternative to this approach would be to simply set a maximum blob value.

c) Morphological Operations:
A disc-shaped structuring element with a radius of 5.
*This is used in imopen to clean up noise in the binary mask generated due to the blob setup in the algorithm. It consists of 2 main operations:
1) *Erosion:* shrinks or fully removes small structures in the binary mask.
2) *Dilation:* expands the remaining structures to restore their original size after erosion.
(*Kalman Filter in One Dimension*, n.d.)

Part 2 Kalman Filter

Variance Formula (1D Case)

$$\sigma_x^2 = Var(x) = \frac{\sum(x_i - \bar{x})^2}{N}$$

Were

- σ_x^2: Variance of x
- x_i: Individual measurements
- $\bar{x}$: Mean of the measurements
- N: Total number of measurements

Equation 1. Variance Formula (1D Case)

Covariance Formula (2 Variables)

$$Cov(x, y) = \sigma_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N}$$

Were:

- σ_{xy}: Covariance between x and y
- x_i, y_i: Individual measurements of x and y
- $\bar{x}, \bar{y}$ Means of x and y

Equation 2. Covariance Formula (2 Variables)

Kalman Gain (K_k) Formula

$$k_k = \frac{P_k H^T}{HP_K H^T + R}$$

Were:

- k_k: Kalman gain
- P_k : State covariance matrix
- H: Observation matrix
- R: Measurement noise covariance matrix

Equation 3. Kalman Gain (Kk) Formula

Variance-Covariance Matrix (3D Case)

$$p = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_z^2 \end{bmatrix}$$

Were:

- Diagonal terms $\sigma_x^2, \sigma_y^2, \sigma_z^2$): Variances of x, y, z
- Off-diagonal terms (σ_{xy}, σ_{xz}, σ_{yz}): Covariances between pairs of variables

Equation 4. Variance-Covariance Matrix (3D Case)

State Covariance Matrix (P_k) in Kalman Filter

$$P_k = AP_{K-1}A^T + Q$$

Were:

- P_k: State covariance matrix at time k
- A: State transition matrix
- Q: Process noise covariance matrix

Equation 5. State Covariance Matrix (Pk) in Kalman Filter

Update Step of Kalman Filter

$$P_k = (I - K_K H)\, P_K$$

Were:

- P_k: Updated state covariance matrix
- I: Identity matrix
- K_K: Kalman gain
- H: Observation matrix

Equation 6. Update Step of Kalman Filter

Variance as Standard Deviation

$$\sigma_x = \sqrt{\sigma_x^2}$$

Were:

- σ_x: Standard deviation
- $\sqrt{\sigma_x^2}$: Variance

Equation 7. Variance as Standard Deviation

Variance-Covariance Matrix (2D Case)

$$p = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}$$

Were:

- σ_x^2, σ_y^2: Variances of x and y
- σ_{xy}, σ_{yx} : Covariance between x and y

Equation 8. Variance-Covariance Matrix (2D Case)

Covariance as an Expectation

$$Cov(x,y) = E\,[(\,x - E[x])\,(y - E[y])\,]$$

Were:

- E: Expected value (mean) of the variable

Equation 9. Covariance as an Expectation

2.1. Covariance matrix explained via probabilistic expressions

In order to explain the covariance matrix, one must understand 'Kalman filters,' *"a mathematical algorithm that estimates the state of a system using noisy measurements."* Rudolf Kalman, a Hungarian engineer, invented the Kalman filter, which finds its application in various fields such as computer vision, guidance and navigation systems, econometrics, and signal processing, among others. (*Kalman Filter - MATLAB & Simulink*, n.d.)

For clarity, the Kalman filter operates via an iterative process that minimises 'the mean of the squared error.' It takes in two major inputs: actual measurements fed into the system and, secondly, a prediction of what the system believes it should be. Following this, the system applies a weighting factor to determine the most appropriate input for a specific frame or time. This process is referred to as 'sensor fusion', which is done iteratively as demonstrated with Equation [6]. *(Brown, 2009)*

The covariance matrix plays a fundamental role within the Kalman filter, as it acts as the backbone of how much uncertainty, or 'the mean of the squared error,' there is between specific variables in play at a certain state frame. The following study on Kalman filters (*Covariance Matrix Explained With Pictures—The Kalman Filter*, n.d.) explains the covariance matrix in terms of error ellipse/ellipsoid visuals.

Like in the word itself, a covariance matrix is a matrix that, e.g., holds an object's position as well as its velocity. Combining these two characteristics into a single matrix enables it to discern the impact of each on the other.

- The matrix's covariance increases as the object's position and velocity rise.
- As the object's position and velocity decrease, the covariance of the matrix decreases.

The Kalman filter uses a covariance matrix to represent the error of a multidimensional Gaussian-distributed dataset; therefore, applying equation 1 alone cannot represent the mean, resulting in a bell-shaped curve as shown in Fig. 10:

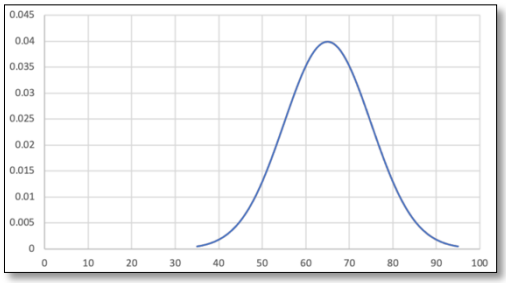


Figure 10. 1D Normal Distribution

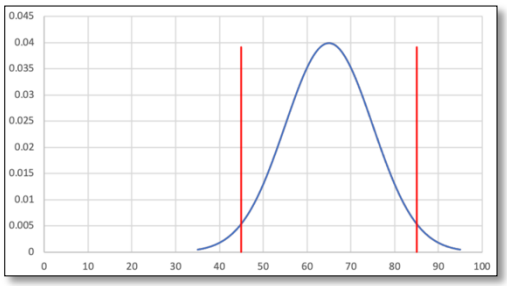


Figure 11. 1D Normal distribution presenting confidence regions

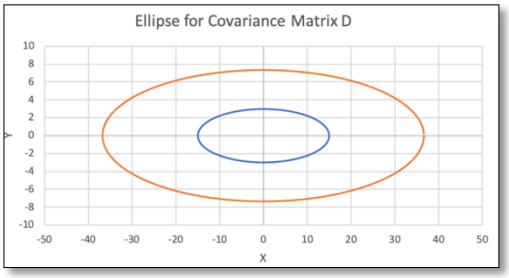


Figure 12. Covariance Matrix without Correlation between X and Y

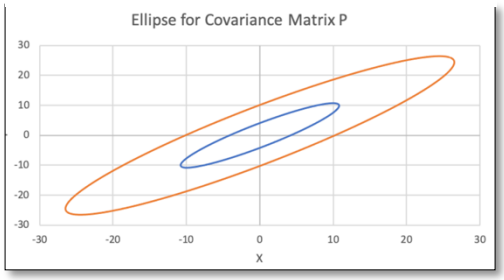
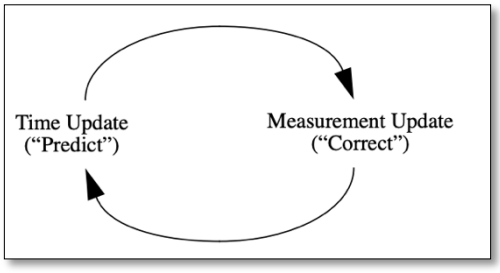


Figure 13. Covariance Matrix with Correlation between X and Y



Another crucial element of the covariance matrix is 'confidence intervals.' Fig. 11 presents a region that illustrates the level of certainty and reliability in the predicted data generated by the filter; it represents a range of values surrounding a point estimate.

Since Kalman filters work on more than one dimension, Fig. 11's red lines, which show the algorithm's acceptability mechanism, aren't a true representation of the visual demonstration because the filters are "multidimensional." Instead, Kalman filters will use a covariance matrix of 2x2, 3x3, or higher. presented in equations [8] and [4].

To represent a covariance matrix visual, an ellipse is presented as per Fig.12 & 13.

Fig. 13 The diagonal elongation of the ellipse results from its tilt. This tilt reflects a linear relationship between X and Y: as X increases, Y tends to either increase (positive correlation) or decrease (negative correlation); the stronger the correlation, the more elongated the ellipse becomes. In relation to equation [8], the covariance matrix will contain a non-zero off-diagonal term $[\sigma_{xy} \neq 0]$, indicating that changes in X are linked to changes in Y.

In contrast, Fig.12 shows not tilt and has alignment with the axis and therefore means X and Y are independent of each other. Similarity in relation to equation [8] the matrix component $[\sigma_{xy} = 0]$ which also indicates its independency.

*Note depending on the computation via equation [2], will determine the positioning of these ellipses.

Welch & Bishop (1994) conducted a study that included the demonstration of the iterative loop nature of Kalman filters and reinforced the concept of constant correction attempts during execution. Equations [5] & [6] govern this loop, highlighting the crucial role of the dynamic covariance matrix.

2.2. Examples of co-variance matrix implementation

2.2a, 3x3 Navigation & Object Tracking in Autonomous Vehicles, Mathematical Walkthrough

Figure 14. (What Are Self-Driving Cars? How Do They Work? | Built In)

Covariance matrices are crucial for sensor fusion in autonomous cars to ascertain the position, velocity, and orientation of the vehicle in three-dimensional space. This entails integrating data from several sensors, including LiDAR, cameras, GPS, and IMUs, to guarantee precise navigation and obstacle identification. Below is a detailed explanation of each of these three elements on which the matrix will be based.

1) LiDAR - emits a laser pulse that reflects off things within the scene. The system then determines the time it takes for the reflected light to return to the source, using this information to create a distance map. *(What Is LiDAR, and How Does It Work? | Synopsys, n.d.)*

2) GPS - supplies the vehicle's exact coordinates, encompassing the longitude, latitude, and altitude. The vehicle uses this data to determine its position in relation to road margins, pedestrian crossings, and nearby automobiles. *What Are Self-Driving Cars? How Do They Work? | Built In, n.d.)*

3) The IMU - "measures a vehicle's acceleration, angular velocity, and orientation, and provides critical information for autonomous vehicles" *(What Is an IMU? » OxTS, n.d.)*

The three sensors mentioned above provide measurements with varying levels of uncertainty. Therefore, these uncertainties meet the requirements for modelling a 3x3 covariance matrix. * Note: Typically, an autonomous system of this type considers both velocity and position parameters, resulting in a 6x6 covariance matrix. For simplicity, only position is considered, resulting in a 3x3 covariance matrix.

Below are rough values that allow for a more visual demonstration:

Table 1. Sensor Fusion parameters

Sensor	Axis	Variance	Covariance with Other Axes
GPS	X	4	$\sigma_{xy} = 1.5 \quad \sigma_{xz} = 1.2$
	Y	3	$\sigma_{xy} = 1.5 \quad \sigma_{yz} = 0.8$
	Z	2	$\sigma_{xz} = 1.2 \quad \sigma_{yz} = 0.8$
IMU	X	1.5	$\sigma_{xy} = 1.2 \quad \sigma_{xz} = 0.5$
	Y	1.5	$\sigma_{xy} = 1.2 \quad \sigma_{yz} = 0.6$
	Z	1.2	$\sigma_{xz} = 0.5 \quad \sigma_{yz} = 0.6$
LiDAR	X	0.8	$\sigma_{xy} = 0.3 \quad \sigma_{xz} = 0.2$

	Y	0.8	$\sigma_{xy} = 0.3 \quad \sigma_{yz} = 0.2$
	Z	0.5	$\sigma_{xz} = 0.2 \quad \sigma_{yz} = 0.2$

Here are the following formulas, showing the types of covariance matrix's:

1) Initial Covariance Matrix [P]; ***indicates the preliminary uncertainties in the state estimation.*** **[Equation 4]**

$$p = \begin{bmatrix} 4 & 1.5 & 0.8 \\ 1.5 & 3 & 0.6 \\ 0.8 & 0.6 & 2 \end{bmatrix}$$

Covariance and variance can be extracted from this matrix, given the variance is values diagonal and covariance un-diagonal:
Variances (4, 3, 2) Covariances (1.5, 1.2, 0.8)

2) Measurement covariance [R_R]; ***describes sensor noise and inaccuracies.***

Diagonal Element computations:

1) For x: GPS = 4 , IMU = 1.5, LiDAR = 0.8

Average variance $\frac{4+1.5+0.8}{3} = 2.5$

2) For y: GPS = 3, IMU = 1.5, LiDAR = 0.8

Average variance $\frac{3+1.5+0.8}{3} = 2.0$

3) For z: GPS = 2, IMU = 1.2, LiDAR = 0.5

Average variance $\frac{2+1.2+0.5}{3} = 1.5$

Un-diagonal Elements computations:

4) For σ_{xy}: GPS = 1.5, IMU = 1.2 , LiDAR = 0.3

Average variance $\frac{1.5+1.2+0.3}{3} = 0.5$

5) For σ_{xy}: GPS = 1.2, IMU = 0.5 , LiDAR = 0.2

Average variance $\frac{1.2+0.5+0.2}{3} = 0.2$

6) For σ_{xy}: GPS = 0.8, IMU = 0.6 , LiDAR = 0.2

Average variance $\frac{0.8+0.6+0.2}{3} = 0.1$

$$R_R = \begin{bmatrix} 2.5 & 0.5 & 0.2 \\ 0.5 & 2 & 0.1 \\ 0.2 & 0.1 & 1.5 \end{bmatrix}$$

Similarly, variance and covariance can be extracted from this matrix.
Variances (2.5, 2.0, 1.5) Covariances (0.5, 0.2, 0.1)

3) Process covariance [Q_Q]; ***accounts for process model uncertainties.***
Given the GPS simply reports the car's position periodically but does not help model how the car moves dynamically, hence we only compute for LiDAR and IMU averages.

Diagonal Element computations:

1) For x: IMU = 1.5, LiDAR = 0.8

Average variance $\frac{1.5+0.8}{3} = 1.2$

Un-diagonal Elements computations:

4) For σ_{xy}: IMU = 1.2 , LiDAR = 0.3

Average variance $\frac{1.2+0.3}{3} = 0.75$

2) For y: IMU = 1.5, LiDAR = 0.8

Average variance $\frac{1.5+0.8}{3} = 1.2$

3) For z: IMU = 1.2, LiDAR = 0.5

Average variance $\frac{1.2+0.5}{3} = 0.85$

5) For σ_{xy}: IMU = 0.5 , LiDAR = 0.2

Average variance $\frac{0.5+0.2}{3} = 0.35$

6) For σ_{xy}: IMU = 0.6 , LiDAR = 0.2

Average variance $\frac{0.6+0.2}{3} = 0.4$

$$Q_Q = \begin{bmatrix} 1.2 & 0.75 & 0.35 \\ 0.75 & 1.2 & 0.4 \\ 0.35 & 0.4 & 0.85 \end{bmatrix}$$

Similarly, variance and covariance can be extracted from this matrix.
Variances (1.2, 1.0, 0.8) Covariances (0.3, 0.2, 0.1)

2.2b, Real Life Applications

Financial Portfolio Management

Application: In finance, covariance matrices are essential for constructing diversified investment portfolios. (*The Significance and Applications of Covariance Matrix - Javatpoint*, n.d.)

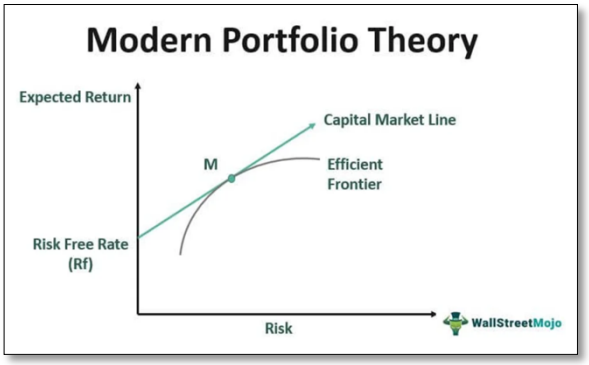


Figure 15. MPT Model

A covariance matrix quantifies the relationships between asset returns in a portfolio. This model specifically implements Modern Portfolio Theory, which utilises covariance to compute the risk (variance) of a portfolio.

Fig. 15 presents a curve showing the set of optimal portfolios offering the highest expected return for a given level of risk; this computation relies heavily on the covariance matrix.

- **Positive** σ_{xy}: Indicates that asset returns tend to move in the same direction.

- **Negative** σ_{xy}: Suggests that asset returns move in opposite directions.

- **Zero** σ_{xy}: Implies no linear relationship between asset returns.

Signal Processing in Telecommunications

Application: In telecommunications, a system called **Multiple Input Multiple Output is implemented, which requires a covariance matrix to allow for the commutations and process of signal analysis.** (*Covariance Matrix: Definition, Formula with Solved Examples - GeeksforGeeks*, n.d.)

The covariance matrix will hold the following elements:

Diagonal Elements: Represents the power (variance) of individual signal channels.

Un-Diagonal Elements: Indicate the correlation between pairs of signal channels;

how the signals interfere or reinforce each other.

- Positive σ_{xy}: Channels move together which causes a constructive interference via reducing diversity.

- Negative σ_{xy}: Opposing channels provide diversity gain which enhances the robustness to fading.

- Zero σ_{xy}: Independent channels will maximize the data throughout as well as provide multiplexing efficiency.

2.3. Examples of Kalman Gain implementation

Kalman Gain, K is pivotal in the Kalman filter. It assesses the degree of reliance on fresh measurements in relation to the existing state estimate. At each stage, it undergoes revisions to reconcile the uncertainty between the forecasted values and the arriving data. The computation of Kalman's gain is performed using equation [3]:

$$k_k = \frac{P_k H^T}{H P_K H^T + R}$$

The Kalman filter predicts the next state (x_p) based on the system's dynamics via a state transition model. Simultaneously, it computes the predicted covariance matrix (P); which quantifies the uncertainty in the predicted state. The covariance matrix P is revised to incorporate the new state estimate following the acquisition of the measurement via equation [6]:

$$P_k = (I - K_K H)\, P_K$$

Here presented is different scenarios of 'Kalman Gain':

[1] High Kalman Gain (K_k to 1)

- A high Kalman Gain suggests that the measurement is more reliable than the prediction; hence the measurement will most likely influence the updated state.

- *E.g., Autonomous Vehicles: (linking back to previous example on 2.2)*

A LiDAR measurement's high accuracy (low covariance R) results in a high k_k, which forces the system to rely on the LiDAR for position correction heavily. (Lin & Wu, 2021) LiDAR measures distances using laser pulses as well as the time of flight (the time it takes for the laser to hit an object and return to the sensor). The method provides mm to cm level precision, especially in controlled environments. These distance measurements will result in low measurement noise; which leads to a low R in the Kalman filter.

[2] Low Kalman Gain (K_k to 0)
-A low Kalman Gain means that the prediction is more reliable than the measurement, hence updated state remains closer to the prediction.

- *Example: Aviation: When a radar signal becomes noisy (high R), the system trusts the inertial navigation system's prediction instead.* *(Wu et al., 2024)*

In aviation, systems often rely on radar signals to determine the position and velocity of aircraft. However, radar measurements can become noisy due to environmental factors such as:

[1] Atmospheric disturbances (e.g., rain, fog, or turbulence); all interfere with the radar signal. *(Zhang et al., 2023)*
[2] Signal reflections from ground clutter or other nearby objects; lead to inaccuracies.
[3] Low signal-to-noise ratio; when the radar signal weakens over long ranges.
[4] The measurement noise covariance (R) increases in these scenarios, which indicates that the radar data is less reliable. (*Signal-to-Noise Ratio*, n.d.)

The Kalman filter recognizes the high measurement noise (R) and assigns a lower Kalman Gain (K_k). This in return will reduce the weight given to radar measurements within the state update. Instead, the system will now rely more heavily on its inertial navigation system predictions; which use accelerometer as well as gyroscope data to estimate position and velocity.

[3] Moderate Kalman Gain ($0< K_k<1$)

- A balanced Kalman Gain reflects that both the prediction and measurement are moderately reliable, hence the updated state is a weighted average of the prediction and measurement.

- *Example: Robot Localisation: A robot combines wheel encoder data (moderate accuracy) with GPS readings (also moderate accuracy). The Kalman Gain gives appropriate weight to both.* *(Skobeleva et al. 2017)* *Let's take the example of an, a robot navigating a warehouse:*

-Wheel Encoders: Measure distance travelled based on wheel rotations. (These measurements are subject to drift and slippage, depending on environmental factors).
-GPS: Provides positional data, however, it can be affected, due to a signal loss.
The Kalman filter dynamically computes K_k and the Kalman Gain will improve its localisation via balancing data GPS and wheel encoder data. This balancing achieved is referred to as a blended estimate, which aims to minimise the impact of the weaknesses in either sensor.

2.4. Aircraft Iteration Computation via mathematical computation and MATLAB usage

This tasks consists of continuing the motion of the aircraft via Kalman filter iterations. The table below presents the first 3 iterations' value of tables, which are to be continued up to 7 iterations.

Iteration	Predicted State X_kp (Position, Velocity)	Process Covariance Matrix P_kp	New State X_k (Position, Velocity)	Kalman Gain K	Previous Covariance Matrix P_k
1st	[4281, 282]	[425, 0], [0, 25]	[4272.5, 282]	[0.405, 0], [0, 0.410]	[253, 0], [0, 14.8]
2nd	[4555.5, 284]	[287.8, 14.8], [14.8, 14.8]	[4553.8, 284.3]	[0.300, 0], [0, 0.291]	187.5, 0], [0, 10.5]
3rd	[4839.1, 286.3]	[187.5, 0], [0, 10.5]	[4843.9, 286.2]	[0.231, 0], [0, 0.226]	[144.2, 0], [0, 8.1]

- To compute for iterations 4 and above, the MATLAB code will take 2 inputs for the last iterations, being:

a) Y_k – Previous state, which is represented via a vector (based of velocity and position)

b) P_k - Previous Covariance Matrix P_k

- For iteration 4 computation, Y_k for iteration 3 and p_k will need to be extracted as following:

1) Iteration 3, Y_k – [4860:286]

2) iteration 3, p_k – [[144.2, 0] : [0, 8.1]]

The MATLAB code has been given the instruction to iterate up to 7 iterations. Here are the following results for the next iterations as well as a graphical representation of Kalman's gains position as well as velocity:

Iteration	Predicted State (x_p)	Kalman Filter Estimate (x)	State Covariance (P)
4th (X_4 = 4860 m, V_3 = 286 m/sec)	[5131.10, 288.20]	[5078.07, 285.49]	$\begin{bmatrix} 121.4954 & 5.3269 \\ 5.3269 & 6.5559 \end{bmatrix}$
5th (X_5 = 5110m, V_3 = 287 m/sec)	[1.00, 2.00]	[976.07, 112.39]	$\begin{bmatrix} 111.2814 & 8.2624 \\ 8.2624 & 5.4130 \end{bmatrix}$
6th (X_6 = 5360 m, V_3 = 285 m/sec)	[1.00, 2.00]	[993.60, 122.04]	$\begin{bmatrix} 106.7258 & 9.8579 \\ 9.8579 & 4.5180 \end{bmatrix}$
7th (X_7 = 5615 m, V_3 = 287 m/sec)	[1.00, 2.00]	[1026.19, 127.56]	$\begin{bmatrix} 104.7625 & 10.6319 \\ 10.6319 & 3.7969 \end{bmatrix}$

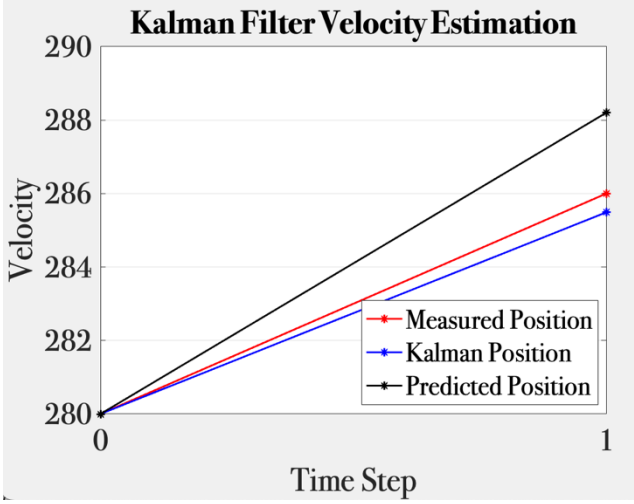


Figure 16. Velocity estimation via Kalman Gain

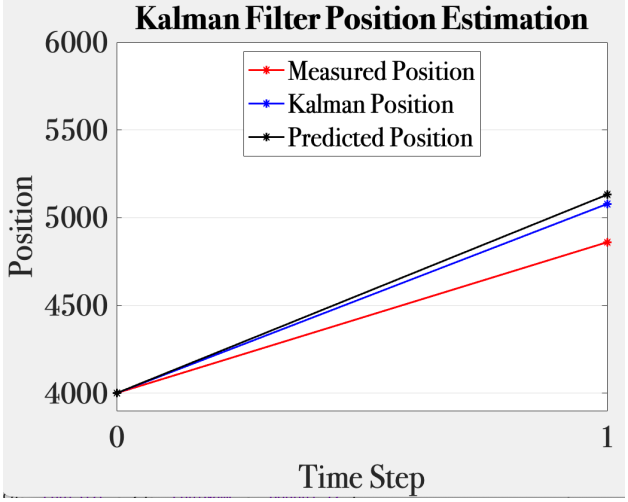


Figure 17. Position estimation via Kalman Gain

Mathematical Manual Computation for Iteration 4:

[1] State Prediction X_{kp}:

Using the formula:

$$X_{kp} = F \cdot X_{k-1} + G \cdot u_k$$

Substitute values:

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

$$u_k = 2$$

$$X_{k-1} = \begin{bmatrix} 4843.9 \\ 286.2 \end{bmatrix}$$

Calculation:

$$X_{kp} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4843.9 \\ 286.2 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \cdot 2$$

$$X_{kp} = \begin{bmatrix} 4843.9 + 286.2 \\ 286.2 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Result:

$$X_{kp} = \begin{bmatrix} 5131.1 \\ 288.2 \end{bmatrix}$$

[2] Covariance Prediction P_{kp}:

Using the formula:

$$P_{kp} = (F \cdot P_{k-1} \cdot F^T) + Q$$

Substitute values:

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$P_{k-1} = \begin{bmatrix} 144.2 & 0 \\ 0 & 8.1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Calculation:

$$P_{kp} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 144.2 & 0 \\ 0 & 8.1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$P_{kp} = \begin{bmatrix} 152.3 & 8.1 \\ 8.1 & 8.1 \end{bmatrix}$$

Result:

$$P_{kp} = \begin{bmatrix} 152.3 & 8.1 \\ 8.1 & 8.1 \end{bmatrix}$$

[3] Kalman Gain K_k:

Using the formula:
$K_k = P_{kp} \cdot H^T \cdot (H \cdot P_{kp} \cdot H_T + R)^{-1}$

Substitute values:
$P_{kp} = \begin{bmatrix} 152.3 & 8.1 \\ 8.1 & 8.1 \end{bmatrix}$

$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$R = \begin{bmatrix} 625 & 0 \\ 0 & 36 \end{bmatrix}$

Calculation:
$H \cdot P_{kp} \cdot H^T + R =$
$\begin{bmatrix} 152.3 + 625 & 8.1 \\ 8.1 & 8.1 + 36 \end{bmatrix} = \begin{bmatrix} 777.3 & 8.1 \\ 8.1 & 44.1 \end{bmatrix}$

$\text{Inverse} = \frac{1}{\begin{bmatrix} 152.3 & 8.1 \\ 8.1 & 8.1 \end{bmatrix}} =$
$\begin{bmatrix} 0.00129 & -0.00024 \\ -0.00024 & 0.02217 \end{bmatrix}$

$K_k = \begin{bmatrix} 152.3 & 8.1 \\ 8.1 & 8.1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$\cdot \begin{bmatrix} 0.00129 & -0.00024 \\ -0.00024 & 0.02217 \end{bmatrix}$

$K_k = \begin{bmatrix} 0.1944 & 0.1480 \\ 0.0085 & 0.1821 \end{bmatrix}$

[4] Updated State X_k:

Using the formula:
$X_k = X_{kp} + K_k \cdot (Y_k - H \cdot X_{kp})$

Substitute values:
$X_{kp} = \begin{bmatrix} 5131.1 \\ 288.2 \end{bmatrix}$

$Y_k = \begin{bmatrix} 4860 \\ 286 \end{bmatrix}$

$K_k = \begin{bmatrix} 0.1944 & 0.1480 \\ 0.0085 & 0.1821 \end{bmatrix}$

Calculation:
$Y_k - H \cdot X\_kp =$
$\begin{bmatrix} 4860 \\ 286 \end{bmatrix} - \begin{bmatrix} 5131.1 \\ 288.2 \end{bmatrix} = \begin{bmatrix} -271.1 \\ -2.2 \end{bmatrix}$

$K_k \cdot [Y_k - H \cdot X_{kp}] = \begin{bmatrix} -53.03 \\ -2.71 \end{bmatrix}$

$X_k = \begin{bmatrix} 5131.1 \\ 288.2 \end{bmatrix} + \begin{bmatrix} -53.03 \\ -2.71 \end{bmatrix}$

$X_k = \begin{bmatrix} 5078.07 \\ 285.49 \end{bmatrix}$

[5] Updated Covariance P_k:

Using the formula:
$P_k = (I - K_k \cdot H) \cdot P_{kp}$

Substitute values:
$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$K_k = \begin{bmatrix} 0.1944 & 0.1480 \\ 0.0085 & 0.1821 \end{bmatrix}$

$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$P_{kp} = \begin{bmatrix} 152.3 & 8.1 \\ 8.1 & 8.1 \end{bmatrix}$

Calculation:
$(I - K_k \cdot H) =$

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.1944 & 0.1480 \\ 0.0085 & 0.1821 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} =$
$\begin{bmatrix} 0.8056 & -0.1480 \\ -0.0085 & 0.8179 \end{bmatrix}$

$P_k \approx \begin{bmatrix} 121.47 & 5.33 \\ 5.33 & 6.56 \end{bmatrix}$

This result matches MATLAB present above and therefore validates the code's ability to perform iterations successfully according to Kalman filters.

Fig.16. and Fig.17. both present how Kalman gains updated estimates are a weighted average of predictions and measurements. The velocity graph on Fig.16, seems to be more promising in contrast to Fig.17, the position graph. In both graphs the blue line (Kalman position) & black line (predicted position) seem to align closely, which indicates how the Kalman filter significantly reduces the noise in the measured data

References

Blob Analysis. (n.d.). Retrieved December 19, 2024, from https://uk.mathworks.com/help/vision/ref/blobanalysis.html

Brown, S. D. (2009). Transfer of Multivariate Calibration Models. *Comprehensive Chemometrics*, *3*, 345–378. https://doi.org/10.1016/B978-044452701-1.00077-6

Chu, H., Song, Q., Yuan, H., Xie, Z., Zhang, R., & Jiang, W. (2015). Research of Mean Shift target tracking with Spatiogram Corrected Background-Weighted Histogram. *2015 IEEE International Conference on Information and Automation, ICIA 2015 - In Conjunction with 2015 IEEE International Conference on Automation and Logistics*, 1942–1946. https://doi.org/10.1109/ICINFA.2015.7279606

Covariance Matrix: Definition, Formula with Solved Examples - GeeksforGeeks. (n.d.). Retrieved December 19, 2024, from https://www.geeksforgeeks.org/covariance-matrix/

Everything you need to know about F1 – Drivers, teams, cars, circuits and more | Formula 1®. (n.d.). Retrieved December 13, 2024, from https://www.formula1.com/en/latest/article/drivers-teams-cars-circuits-and-more-everything-you-need-to-know-about.7iQfL3Rivf1comzdqV5jwc

How F1's Timing Tech Works and Why It's Crucial for Racing Accuracy. (n.d.). Retrieved December 13, 2024, from https://www.v-hr.com/blog/how-f1s-timing-tech-works-and-why-its-important/

Kalman Filter - MATLAB & Simulink. (n.d.). Retrieved December 19, 2024, from https://uk.mathworks.com/discovery/kalman-filter.html

Kalman Filter in one dimension. (n.d.). Retrieved December 18, 2024, from https://www.kalmanfilter.net/kalman1d.html?

Lim, S. H., Mat Isa, N. A., Ooi, C. H., & Toh, K. K. V. (2015). A new histogram equalization method for digital image enhancement and brightness preservation. *Signal, Image and Video Processing*, *9*(3), 675–689. https://doi.org/10.1007/S11760-013-0500-Z/TABLES/2

Lin, S. L., & Wu, B. H. (2021). Application of Kalman Filter to Improve 3D LiDAR Signals of Autonomous Vehicles in Adverse Weather. *Applied Sciences 2021, Vol. 11, Page 3018*, *11*(7), 3018. https://doi.org/10.3390/APP11073018

Meshgi, K. (2015). The State-of-the-Art in Handling Occlusions for Visual Object Tracking. *IEICE Transactions on Information and Systems*. https://www.academia.edu/59457792/The_State_of_the_Art_in_Handling_Occlusions_for_Visual_Object_Tracking

Object Tracking | Student Competition: Computer Vision Training - YouTube. (n.d.-a). Retrieved December 18, 2024, from https://www.youtube.com/watch?v=0jAC9sMQQuM&t=789s

Object Tracking | Student Competition: Computer Vision Training - YouTube. (n.d.-b). Retrieved December 19, 2024, from https://www.youtube.com/watch?v=0jAC9sMQQuM&t=883s

Signal-to-Noise Ratio. (n.d.). Retrieved December 20, 2024, from https://www.licor.com/bio/help/empiria_studio/software/empiria_studio/calculate/blot/signal-to-noise.html

The Significance and Applications of Covariance Matrix - Javatpoint. (n.d.). Retrieved December 19, 2024, from https://www.javatpoint.com/the-significance-and-applications-of-covariance-matrix?

vision.BlobAnalysis. (n.d.). Retrieved December 19, 2024, from https://uk.mathworks.com/help/vision/ref/vision.blobanalysis-system-object.html

Wu, Q. ;, Yang, R. ;, Liu, K. ;, Xu, Y. ;, Miao, J. ;, Sun, M., Dual, K., Wu, Q., Yang, R., Liu, K., Xu, Y., Miao, J., & Sun, M. (2024). Dual Kalman Filter Based on a Single Direction under Colored Measurement Noise for INS-Based Integrated Human Localization. *Electronics 2024, Vol. 13, Page 3027*, *13*(15), 3027. https://doi.org/10.3390/ELECTRONICS13153027

Zhang, Y., Carballo, A., Yang, H., & Takeda, K. (2023). Perception and sensing for autonomous vehicles under adverse weather conditions: A survey. *ISPRS Journal of Photogrammetry and Remote Sensing*, *196*, 146–177. https://doi.org/10.1016/J.ISPRSJPRS.2022.12.021

- Note: Self-produced content:
  Fig. 1. Ball Linear Translational Movement
  Figs 9. Racetrack Animation, developed on Blender

Appendix

Figure 10. Histogram Tracking Flowchart

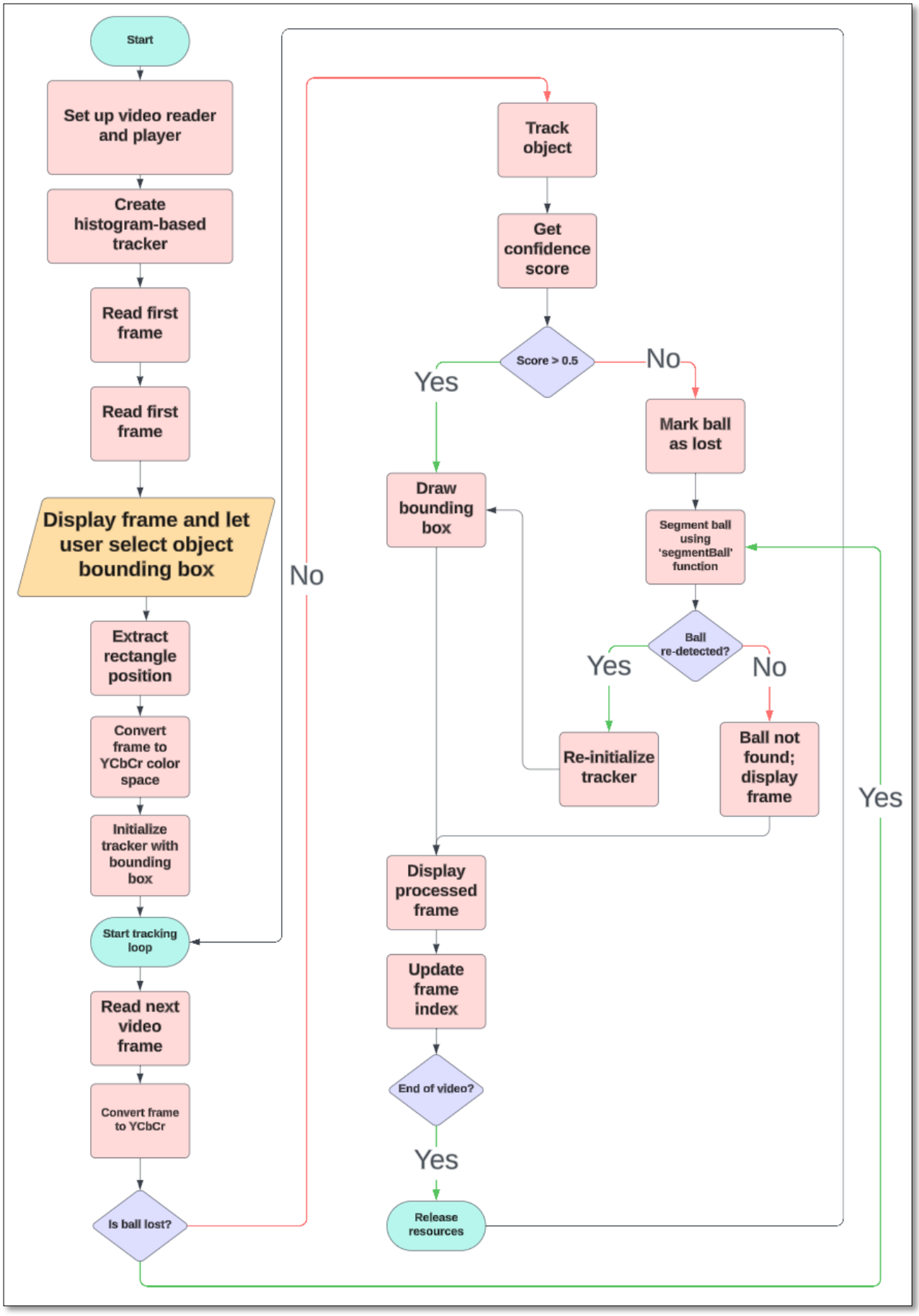
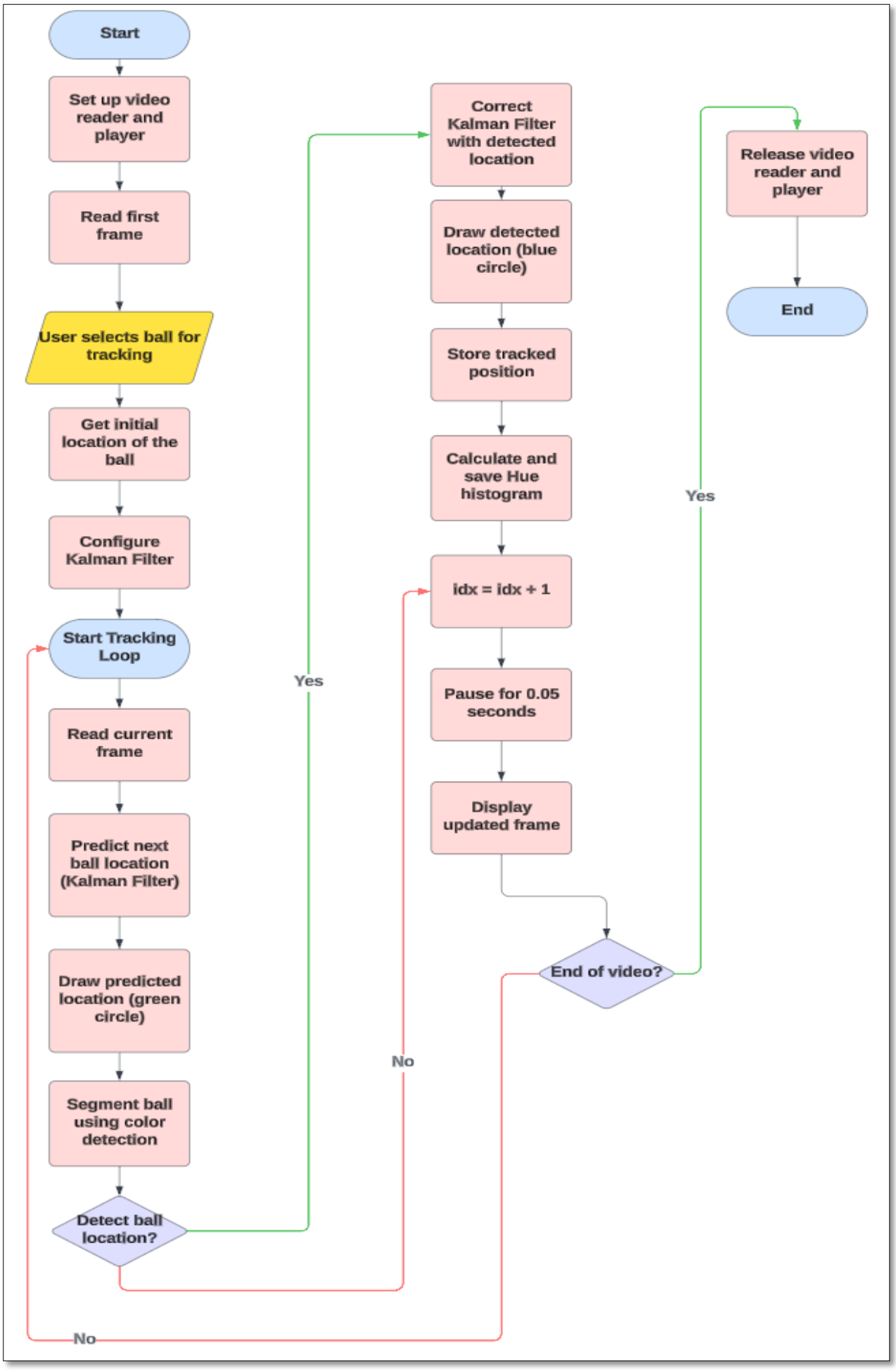


Figure 11. Kalman Filter Tracking Flowchart



1.2g: Kalman Filter Code, Key Features Breakdown
(*Object Tracking | Student Competition: Computer Vision Training - YouTube*, n.d.-b)

```
% Load the video file & set up the video player
videoFReader = vision.VideoFileReader('racetrackcars.mov','VideoOutputDataType','double');
videoPlayer = vision.DeployableVideoPlayer;

% Initialise tracking structure and parameters
tracks = struct(...
    'id', {}, ...                       % Track ID assigned
    'centroid', {}, ...                 % Holds current centroid of the track
    'kalmanFilter', {}, ...             % Kalman filter for each track
    'consecutiveInvisibleCount', {}); % Invisible frame count
nextId = 1;                             % ID for the next new track

% Kalman filter configuration
param.motionModel = 'ConstantVelocity'; % Assumes constant velocity motion, explained in
% report in further depth
param.initialEstimateError = [1e6 1e6]; % Large initial uncertainty
param.motionNoise = [20 20];             % Process noise covariance
param.measurementNoise = 10;             % Measurement of noise covariance

% Tracking parameters
maxInvisibleCount = 10;                 % Maximum allowed invisible frames for this algorithm
costOfNonAssignment = 50;               % Cost for unassigned tracks (weighting factor)

% Initialises frame counters as well as track history
frameCount = 0;
trackHistory = [];                      % Stores tracking history
```

[1] Initial Setup

```
% Loop through each frame in the video
while ~isDone(videoFReader)
    videoFrame = step(videoFReader);   % Read the current video frame
    frameCount = frameCount + 1;       % Increment frame count

    % Detects objects and their centroids in the frame
    [~, centroids] = segmentBall(videoFrame, 2000); % Minimum area for detection

    % Mark detected centroids on the frame, assigned as a black circle
    if ~isempty(centroids)
        videoFrame = insertMarker(videoFrame, centroids, 'x', 'color', 'black', 'size', 5);
    end
```

[2] Main Processing Loop

```
 nDetections = size(centroids, 1);  % Number of detected objects
 nTracks = length(tracks);          % Number of active tracks

 % Predict the next state of each track using the Kalman filter
 for idx = 1:nTracks
     tracks(idx).centroid = predict(tracks(idx).kalmanFilter);
 end

 % Compute assignment cost matrix for tracks and detections
 cost = zeros(nTracks, nDetections);
 for idx = 1:nTracks
     cost(idx, :) = distance(tracks(idx).kalmanFilter, centroids); % Compute Euclidean distance
 end

 % Assign detections to tracks
 [assignments, unassignedTracks, unassignedDetections] = ...
     assignDetectionsToTracks(cost, costOfNonAssignment);
```

[3] Predict and Assign Tracks

```
% Update assigned tracks with corresponding detections
for idx = 1:size(assignments, 1)
    trackIdx = assignments(idx, 1);
    detectionIdx = assignments(idx, 2);
    correct(tracks(trackIdx).kalmanFilter, centroids(detectionIdx, :));
    tracks(trackIdx).consecutiveInvisibleCount = 0;
end

% Handle unassigned tracks and remove lost tracks
toBeDeleted = false(nTracks, 1);
for idx = 1:length(unassignedTracks)
    ind = unassignedTracks(idx);
    tracks(ind).consecutiveInvisibleCount = tracks(ind).consecutiveInvisibleCount + 1;
    if tracks(ind).consecutiveInvisibleCount > maxInvisibleCount
        toBeDeleted(ind) = true;
    end
end
tracks(toBeDeleted) = [];
```

[4] Update and Manage Tracks

```
% Create new tracks for unassigned detections
for idx = 1:size(unassignedDetections, 1)
    kalmanFilter = configureKalmanFilter(...
        param.motionModel, ...
        centroids(unassignedDetections(idx), :), ...
        param.initialEstimateError, ...
        param.motionNoise, ...
        param.measurementNoise);

    tracks(end+1).id = nextId;      % Assigns a new track ID
    tracks(end).kalmanFilter = kalmanFilter;
    tracks(end).centroid = centroids(unassignedDetections(idx), :);
    tracks(end).consecutiveInvisibleCount = 0;
    nextId = nextId + 1;
end
```

[5] Create New Tracks

```
%  This loop handles the creation of new tracks for detection, that were not assigned to
% existing tracks already. For each unassigned detection, it will configure a new Kalman filter.
  % Stores the active track IDs as well as history
  activeTrackIDs = [tracks.id];
  trackHistory(frameCount).frame = frameCount;
  trackHistory(frameCount).ids = activeTrackIDs;

  % Annotates the track on the frame, track 1 and track 2 as visible on
  % the animation
  for idx = 1:length(tracks)
      videoFrame = insertText(videoFrame, ...
          tracks(idx).centroid, ...
          ['Track: ' num2str(tracks(idx).id)], ...
          'FontSize', 20, ...
          'BoxColor', 'yellow', ...
          'TextColor', 'black', ...
          'BoxOpacity', 0.6);
  end
```

Boxes 3-5, explain how tracks come into play within this algorithm.

- kalmanFilter = configureKalmanFilter(...)

This specific function will initialise a new Kalman filter for an unassigned detection.
The filter is configured with parameters, as presented: motion model, initial state estimate, process noise, and measurement noise.

- if tracks(ind).consecutiveInvisibleCount > maxInvisibleCount

Check if the track has been invisible for too many frames (defined by maxInvisibleCount). Following this is the code line: toBeDeleted(ind) = true, which will recognise if a track has been insufficient for a long period, resulting in its deletion (which was the case for our track animation, where track 2 was deleted, resulting in track 3)

-

[6] Display Tracks and Save History

```
    % Displays the video frame
    step(videoPlayer, videoFrame);
    pause(0.1) % Slight pause used to establish better visibility & clearly visible when
    % executing the program, runs much slower than the orginal video/animation.
end

% clear cache memory
release(videoPlayer);
release(videoFReader);

% Plot the tracking history graph
figure;
hold on;
for idx = 1:length(trackHistory)
    frame = trackHistory(idx).frame;
    ids = trackHistory(idx).ids;
    for idIdx = 1:length(ids)
        plot(frame, ids(idIdx), 'bo', 'MarkerSize', 5, 'MarkerFaceColor', 'b');
    end
end
title('Tracking IDs vs Frames');
xlabel('Frame Number');
ylabel('Track ID');
grid on;
hold off;
```

[7] Clean Up and Plot Results

```
% Function to segment objects in the frame based on color thresholds
function [boxLoc, centroidLoc] = segmentBall(videoFrame, minArea)
    persistent hBlob
    if isempty(hBlob)
        hBlob = vision.BlobAnalysis;
        hBlob.AreaOutputPort = false;
        hBlob.ExcludeBorderBlobs = true;
        hBlob.MinimumBlobArea = minArea;
    end

    % Converts the video frame to HSV color space for segmentation
    Ihsv = rgb2hsv(videoFrame);
    hue = Ihsv(:, :, 1);
    sat = Ihsv(:, :, 2);
    val = Ihsv(:, :, 3);

    % Detects blue and red objects based on HSV thresholds
    BW_blue = (hue >= 0.55 & hue <= 0.65) & (sat > 0.3) & (val > 0.2);
    BW_red1 = (hue >= 0.0 & hue <= 0.05) & (sat > 0.4) & (val > 0.2);
    BW_red2 = (hue >= 0.9 & hue <= 1.0) & (sat > 0.4) & (val > 0.2);
    BW_red = BW_red1 | BW_red2;

    BW = BW_blue | BW_red;            % Combines detected regions
    BW = imopen(BW, strel('disk', 5)); % Morphological opening in order remove noise

    [centroidLoc, boxLoc] = step(hBlob, BW); % Extracts object centroids and bounding boxes
    boxLoc = round(boxLoc);
end
```

[8] Segment Ball Function

The following code uses the BlobAnalysis System object (hBlob) in order to detect as well as analyse, the connected regions (blobs) in a binary image.

The hBlob object is initialised in a way, to ignore small blobs (MinimumBlobArea = minArea) as well as to exclude blobs which touch the image out-skirts.
Following this, comes the step(hBlob, BW) function which extracts centroids as well as the bounding boxes of blobs in the binary image; which in return will correspond to the detected red and blue objects.

The regions of interest (blobs) are first segmented using colour thresholds in HSV space (rather than RGB), as explained above in racetrack analysis, the program was initiated without the stating the hue spectrum, in which led to another track being assigned to the grass patch in the middle, however it still successfully, tracked the vehicles going around.