Sensor Fusion of Visual Odometry and High-Gain Observer for Application to Vehicle Localization

W. Jacques[1], H. Bessafa[2], A. Zemouche[2], and R. Rajamani[3], Z. Belkhatir[1,*]

***Abstract*—This paper investigates the integration of visual odometry with a state-space observer, as an integral part of a Simultaneous localization and Mapping (SLAM) system, for robust vehicle localization. SLAM is a key part of many Advanced Driver Assistance Systems (ADASs), and plays a crucial role in the increasing number of autonomous driving systems. A novel fusion algorithm that relies on already integrated vehicle sensors with the additional cost-effective visual data is proposed. The proposed system enables a correction of the vehicle's path from monocular visual SLAM system using a theoretically-proven nonlinear high-gain observer. Moreover, an Extended Kalman Filter (EKF) enhances the vehicle's path localization by integrating GNSS data with the visual odometry and the high-gain observer. The proposed system accurately and quickly provides the vehicle with its position on the road, and a mapping of the world it is in. The performance of the integrated localization system is shown through different simulated scenarios, which mimic real-life occurrences, using data from the CARLA simulator. The system is shown to leverage the presence of visual odometry and the nonlinear observer to accurately localize the vehicle even when the GNSS or RTK base station signal is lost.**

I. Introduction

FOR the past thirty years, automotive safety has been steadily increasing, due to developments in advanced driver assistance systems (ADAS). These systems take control from the driver, therefore mitigating risks due to human error [1]. Examples include anti-lock braking, traction control, electronic stability programs, and more advanced systems like automatic braking and lane-assist systems [2]. Recently, there has been vast development in self-driving automotive systems, eliminating human error altogether, leading to reduced crashes, congestion, and harmful emissions [3], [4]. As automotive ADASs and autonomous driving systems become more advanced, it is increasingly required that the vehicle constructs a map of the surrounding environment, as well as recognise its position within this map. This process is much like a human driver does, and allows the vehicle's safety systems to respond to developing hazards around the vehicle. This is given the term simultaneous localization and mapping (SLAM) [5].

[1] William Jacques and Zehor Belkhatir are with the Department of Electronics and Computer Science, University of Southampton, University Road, Southampton, SO17 1BJ, UK.

[2] Hichem Bessafa and Ali Zemouche are with the Université de Lorraine, CNRS, CRAN, F-57000 Metz, France.

[3] Rajesh Rajamani is with the Laboratory for Innovation in Sensing, Estimation and Control, Department of Mechanical Engineering, University of Minnesota, United States.

[*] Corresponding author (Email: z.belkhatir@soton.ac.uk)

Research reported in this paper was supported by University of Southampton and partially funded by the ANR agency under the project ArtISMo ANR-20-CE48-0015.

Techniques for both localization and mapping have developed as requirements change; systems have grown from mono-sensor solutions like Global Network Satellite System (GNSS), also referred to as Global Positioning System (GPS), or wheeled odometry [6], to multi-sensor systems where a collection of sensors are fused together to improve the output [7], [8]. The authors in [6] compared different SLAM methods for autonomous race cars. This shows the downside of using wheeled-odometry techniques, due to speed differentials between the wheels and the vehicle. Our system improves upon [6] by fusing the kinematic model in the high-gain observer to correct for any drift introduced by the other sensors. While our current study does not focus on aggressive driving like [6], the automotive environment is still extremely harsh for sensors, and it is common for issues to arise. For example, this could be temporary sensor failure; obscured GNSS data behind buildings, trees, or bridges could cause failure to provide a position. The satellite signals are also affected by atmospheric conditions and reflections, meaning the received data may be inaccurate. Our proposed system can avoid bad sensor data corrupting the system by fusing data from different sensors to allow redundancy. An example of visual-sensor fusion is the technique proposed in [8], taking a visual-inertial fusion approach, which uses a technique called Inverse Perspective Mapping to turn the 3D camera view into a bird's-eye view image. Following this, feature-detection of the road markings provide a useful output of just the centre line, from which the vehicle's lateral velocity can be calculated. This data is fused with an Inertial Measurement Unit (IMU) using a Kalman Filter. Our paper integrates an advancement in the form of resilience, whereby it can cope with sensor failure. The system in [8] is also only aimed at use in two wheeled vehicles and not applied on four-wheeled vehicles. Furthermore, our system produces an absolute position output, which allows further applicability to connected vehicle networks, where absolute position knowledge is needed for cooperative localization [9].

The recent study [7] proposes a solution for detection and positioning of other vehicles, specifically to be mounted on an electronic scooter. The system uses data from a two-dimensional light detection and range (LIDAR) sensor combined with data from a monocular camera. Despite the aim of achieving a cost-effective approach, the LIDAR sensor used costs $650, which is still comparatively more expensive than other sensors. The automotive industry works with low profit margins, and sensors are only added when their requirement outweighs their cost. This also extends to using cheaper sensors to replace more costly ones and using complex techniques to lessens the sensor's disadvantages. One research

solution is the use of soft sensors like observers. Nonlinear observers have found multiple uses in the automotive field with many applications, e.g., estimation of complex internal vehicle parameters such as side slip angle [10], vehicle tracking [11] [12] [13], and cyber-physical attacks detection [14].

This cost-saving is further seen in industry applications; a recent controversy arose when Tesla [15] began removing radar systems in favour of “Tesla Vision", an advanced camera system. This reduction in sensing capability is for a good reason, due to the relatively low cost of cameras in comparison to LIDAR; cameras are as much as 100 times cheaper than the cheapest LIDAR systems [16], [7]. Fully autonomous taxis are currently in use in America, to mixed reviews [17], some praising the quickly progressing technology, others unwilling to accept small imperfections in the final product. Unlike Tesla, these taxis contain many additional sensors atop the roof: LIDAR, cameras and additional radar sensors. The fact that both these solutions struggle to produce “perfect" results suggests that autonomous capability is not limited by sensors, but by how they are used.

In this paper, an observer-based sensor fusion design system is proposed to address the aforementioned challenges. A by-product of the proposed fusion framework is a reduction in sensor and system cost, a quality of utmost importance for the automotive sector. Moreover, the fusion combination of the nonlinear observer and visual odometry system is built upon a modular form, allowing sensing systems to be interchanged if the sensor requirements change; for example, the observer-based orientation sensor is interchangeable with a compass-based sensor. This allows a wide applicability in the automotive sector, with the possibility to use different, or additional sensors with little change to the design. The proposed system’s absolute position output gives it another advantage, where its output can be used as part of a connected vehicle network to provide the network with data on the absolute positioning of each vehicle. This connectivity can provide many benefits, as the sharing of a vehicle’s position can improve the accuracy, resilience, and robustness of the whole network’s SLAM [9]. The main contributions of the paper can be summarised as follows:

- Cheap fusion framework relying on affordable sensors.
- Novel combination of nonlinear observers, namely high-gain observer, and visual SLAM for more accurate and resilient vehicle tracking.
- Robust estimation of the absolute position of the vehicle even during GNSS sensor failure, which is a very useful property for cooperative localization in connected-car networks.
- Extensive experimental tests using automotive-specific simulation software, namely CARLA.

The remainder of the paper is organized as follows. Section II introduces the problems faced by traditional techniques, and provides background to the modular blocks used by the proposed system. Section III proposes the novel techniques introduced, including combination of the modular blocks, as well as providing a visualisation of the flow and processing of sensor data through the entire system. Finally, Section IV analyses the numerical implementation and CARLA experimental results to conclude the paper and outline potential future works in Section V.

II. Problem Statement

This section provides necessary background details about the selected sensors and technologies along with the challenges that are faced during the implementation of a SLAM system, all of which are essential to the proposed fusion localization solution to be provided in the next section.

A. Sensor Selection

One seemingly trivial element of the system is the selection of suitable sensors. As discussed earlier, there are tight cost constraints in the automotive sector, leading to the selection of sensor solutions which are either already implemented, or inexpensive to integrate additionally. Another consideration that we account for is about the number of sensors. Moreover, fusion of multiple sensors may require an additional complexity in either software or hardware, which may also increase cost. From this, a trade-off between cost, accuracy, and reliability must be made.

In this study, four sensors have been selected to be used in the vehicle localization system. The first is the wheel speed sensor that provides each wheel’s rotational velocity, and hence, by using a vehicle kinematic model similar to that in [18], an indirect relative position can be provided. The second sensor is the steering angle sensor, which measures the angle of the front wheels relative to the car body. This can be combined with the knowledge of the wheel speed to improve the accuracy of localization. Both of these sensors have been chosen since they are usually already implemented on the vehicle for use in stability control systems like [19], making the system very cost-effective. Accuracy over short distances is good, due to both sensor’s relatively precise outputs. Despite this, a system like this is very prone to drift, due to wheel slip, lateral movement, and in general, aggressive driving [6].

The third sensor used is GNSS, a system that uses the triangulation of satellites to obtain a position on the Earth. This sensor was chosen since it is one of the only sensors that provide a direct, absolute measurement of the vehicle’s position, making it essential for applications related to connected vehicle networks. This triangulation can provide accuracy of 1.5m in the best case using sensors such as in [20]. The technique using only satellite measurements is henceforth referred to as local GNSS. If a navigation system is fitted, a lower-accuracy local GNSS sensor will likely already be integrated into the vehicle, reducing the cost. Furthermore, local GNSS can be improved using a technique called Real Time Kinematics (RTK) [21]. RTK uses transmitted signals from a reference base station to improve the accuracy of measurement. This allows sub-1m accuracy in sensors like the ones from [22], down to 1cm accuracy in sensors from [23]. Although this usually comes with a higher cost over regular local-only GNSS, this cost is justified for high-accuracy applications. The only major issue facing GNSS systems is the requirement for satellite visibility. If the satellites are hidden from view, for example when going

under bridges or in tree cover, less accurate, or no GNSS position data will be outputted. In addition, RTK sensors face the additional requirement for visibility to the stationary base stations. Without these, the accuracy will reduce to local-only GNSS, or no position output. Furthermore, start-up times vary drastically, with some modules taking multiple minutes to lock onto satellites. Due to these unreliable characteristics, the fusion with the other sensors is essential and required to provide a solution resilient to these challenges.

The last sensor selected is a monocular RGB camera. Cameras can convey significant amounts of information, which coupled with their ever-falling cost, makes them a cost-effective choice. Despite this, the sheer amount of data output by the camera requires the use of complex computer-vision techniques and high performance computing hardware to extract useful data. While this does go against the low-cost ethos, the extractable data far outweighs the cost outlay. A monocular camera has been chosen over stereoscopic to lower cost and complexity. Due to the low cost of the only additional sensor, the camera, the proposed system opens the door to wider applicability, offering large possibility for little outlay.

The following section investigates the required mono-sensor systems in more detail: the High-Gain Observer, and Visual SLAM algorithm.

B. High-Gain Observer

The role of the high-gain observer is to estimate the internal states of the vehicle, by using knowledge of the vehicle's dynamics, and measurements from external sensors. Since these internal states are not directly measured, this is called a soft sensing approach. In this case, the orientation of the vehicle is estimated, which will later be used to offset any bias and drift of the entire system's estimated path. This, in essence, will correct the relative orientation of the visual odometry's path to an absolute orientation, which allows the proposed system to maintain an accurately oriented path even when losing absolute sensor data, i.e., GNSS.

The model-based observer used in this study relies on the vehicle kinematic model in [13]. Its variables are portrayed in Figure 1.

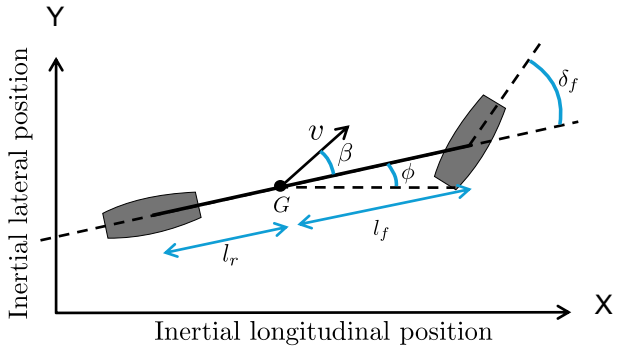

Fig. 1. Vehicle Kinematic Model. Reproduced from [13].

These dynamics are described by the following vehicle kinematic equations:

$$\begin{bmatrix}\dot{X}\\ \dot{Y}\\ \dot{\phi}\\ \dot{\delta}_f\end{bmatrix} = \begin{bmatrix} v\cos(\phi+\beta)\\ v\sin(\phi+\beta)\\ \frac{v}{l_f+l_r}\tan(\delta_f)\cos(\beta)\\ 0\end{bmatrix}, \tag{1}$$

where

- X, Y is the vehicle's relative position in the coordinate space.
- ϕ is the vehicle's orientation with respect to the x axis.
- v is the vehicle's longitudinal speed.
- β is the slip angle. The derivative of this is assumed to be 0.
- δ_f is the steering angle. The derivative of this is assumed to be 0.
- l_f, l_r are the distances from each axle to the centre of gravity of the vehicle, front and rear respectively. When added together, they define the wheelbase.

We use the following relationship between the slip and steering angles :

$$\beta = \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\delta_f)\right), \tag{2}$$

where the vehicle slip angle β is assumed to be slowly varying, hence its rate ($\dot{\beta}$) is assumed to be zero. We can rewrite (2) as follows:

$$\frac{\tan(\beta)}{l_r} = \frac{\tan(\delta_f)}{l_r+l_f}, \tag{3}$$

and substituting in (1) we get:

$$\dot{\phi} = \frac{v}{l_r}\sin\beta. \tag{4}$$

The vehicle's position is measured, and used as the output equation of the observer:

$$y = \begin{bmatrix}X\\ Y\end{bmatrix}, \tag{5}$$

where y is the measurement output.

The used observer is a high-gain observer designed and proposed in [11], where it is shown to have applications to vehicle tracking, which makes it a relevant fit for this study. The implemented observer leverages the second design outlined in [11]. To make this observer design applicable, the kinematic model must be transformed into a triangular system $z \in \mathbb{R}^6$ using the following change in variable:

$$\begin{cases} z_1 = y_1 = X,\\ z_2 = \dot{z_1} = \dot{X} = v\cos(\phi+\beta),\\ z_3 = \dot{z_2} = \ddot{X} = -v\sin(\phi+\beta)\dot{\phi} = -\dot{\phi}\times z_5,\\ z_4 = y_2 = Y,\\ z_5 = \dot{z_4} = \dot{Y} = v\sin(\phi+\beta),\\ z_6 = \dot{z_5} = \ddot{Y} = v\cos(\phi+\beta)\dot{\phi} = \dot{\phi}\times z_2, \end{cases} \tag{6}$$

where $\dot{\phi}$ is defined in (1). The transformed nonlinear system in (6) can be written in a triangular form as follows:

$$\begin{cases} \dot{z} = A_{sys}z + B_{sys}f(z)\\ y = C_{sys}z\\ \omega = h(z) \end{cases} \tag{7}$$

where

$$A_{sys} = \begin{bmatrix} a & 0_{3\times3} \\ 0_{3\times3} & a \end{bmatrix},\ B_{sys} = \begin{bmatrix} b & 0_{3\times1} \\ 0_{3\times1} & b \end{bmatrix},$$
$$C_{sys} = \begin{bmatrix} c & 0_{1\times3} \\ 0_{1\times3} & c \end{bmatrix},\ \text{and}\ f(z) = \begin{bmatrix} f_1(z) \\ f_2(z) \end{bmatrix}, \quad (8)$$

$0_{n\times m}$ denotes the zero matrix of n rows and m columns,

$$a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},\ b = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},\ c = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix},\ \text{and}$$
$$f_1(z) = -z_2 \times \dot{\phi}^2, f_2(z) = -z_5 \times \dot{\phi}^2, \quad (9)$$

$f, h : \mathbb{R}^n \to \mathbb{R}$ satisfy the Lipschitz property formulated under the form

$$|f(x'_1, \ldots, x'_n) - f(x_1, \ldots, x_n)| \leq \gamma_f \sum_{j=1}^{n} |x'_j - x_j| \quad (10)$$

where h is defined similarly, and $\dot{\phi}$ is related to the new state as follows:

$$\dot{\phi} = \frac{v}{l_r} \sin\beta = \frac{z_2 z_6 - z_3 z_5}{z_2^2 + z_5^2} \quad (11)$$

The high-gain observer proposed in [11] takes into account the presence of an additional measurement ω. It is given by the following dynamics [1]:

$$\dot{\hat{z}} = A_{sys}\hat{z} + B_{sys} f(\hat{z}) + L(y - C_{sys}\hat{z}) + M(\omega - h(\hat{z})), \quad (12)$$

where $\hat{z}$ is the transformed state estimate, ω denotes the additional measurement alongside with the output measurement vector y. In this case, the additional measurement is the velocity of the car, found from the wheel speed. The velocity of the car, v, can be calculated using (1) simply by the distance travelled by the car, and mapped into the triangular system:

$$v^2 = \dot{X}^2 + \dot{Y}^2 = {z_2}^2 + {z_5}^2, \quad (13)$$

which can be taken as an additional geometric constraint on the system. The additional constraint can thus be defined as follows:

$$h(z) = v = \sqrt{{z_2}^2 + {z_5}^2}. \quad (14)$$

Consequently, since the high-gain observer estimates the states in the triangular system (6), the non-transformed state ϕ of the system (1) does not exist in the transformed system. It follows that the estimates $\hat{\phi}$ and $\hat{v}$ can be computed as follows:

$$\hat{\phi}(t) = \tan^{-1}\left(\frac{\hat{z}_5(t)}{\hat{z}_2(t)}\right) - \beta, \quad \hat{v}(t) = \sqrt{\hat{z}_2^2(t) + \hat{z}_5^2(t)}\ , \quad (15)$$

where β is calculated as in (2), using δ_f as measured by the steering angle sensor. More details about the used LPV and LMI techniques along with the convexity principle to prove the observer's convergence and compute the observer's gains can be found in [11].

Remark 1: The additional measurement ω is used to ensure the transformed triangular system z is fully defined. Moreover, without the additional wheel speed measurement, the kinematic model (1) allows the orientation of the car to converge to two possible values, 180° apart, represented by either positive or negative wheel speed.

Remark 2: Since the steering angle δ_f is typically small, β, as defined in (2), is minimal. In tracking problems, obtaining another vehicle's steering angle is difficult, so this term may be omitted.

C. Visual SLAM

The visual odometry algorithm uses a feature-based extraction, following the structure of [24] and [25]. Oriented FAST and rotated BRIEF (ORB) feature detection [26] is used to effectively find tracking features. It will compare the image from the current and past frame, before using feature-tracking to match the images. The relative movement of these tracked-features allows the VO algorithm to output the estimated position of the vehicle, which is the output of the VO block. ORB features allows more data to be portrayed by each feature descriptor, with the added bonus of being invariant to scale, rotation and translation. This in turn allows a greater feature matching ability, and hence more accurate path tracking. The process of using ORB features to underpin the visual SLAM algorithm is called ORB-SLAM. In order to provide an accurate output, the ORB-SLAM algorithm requires significant parameterization. For example, an accurate model of the camera is required to match and triangulate the extracted features. Similarly, the feature extractor requires a set of parameters to reliably extract features, balancing computation time with accurate feature detection. Camera intrinsics define the type of image captured, and allow extraction of vital coordinate space translations during processes such as triangulation. More details regarding camera's intrinsic and extrinsic parameters can be found in [27].

The algorithm begins by extracting the ORB features from the first two frames, before attempting to match the set of features between the two images. If successful, the matched points can be triangulated using knowledge of the camera's parameters to map these matched points from 2D camera space into the 3D world coordinates. Following triangulation, the two initialisation frames are stored, along with their corresponding map points in memory, in the keyframe storage. In order to improve future path predictions, retrospective frames (views) and their corresponding features are stored in memory. This memory set is henceforth referred to as "Keyframe Storage". The connections between any linked keyframes, by pose and feature matches, are stored in the keyframe storage too. Figure 2 shows the structure of this storage graphically.

A similar storage system is also created for the 3D world coordinates of matched and triangulated features, called map points. Equally, like keyframe connections, the keyframe features that correspond to the map points are saved too, at least two correspondences per map point. The storage is schematically described in Figure 3. Once the number of keyframes, and hence map points have been found, the map point set is put through the bundle adjustment algorithm. This algorithm compares and matches features and map points between the current and past keyframes, including non-consecutive keyframes. An example could be a signpost; the

[1]Some of the used notations has been slightly changed from [11], to improve clarity and avoid confusion with variables defined later in this paper.

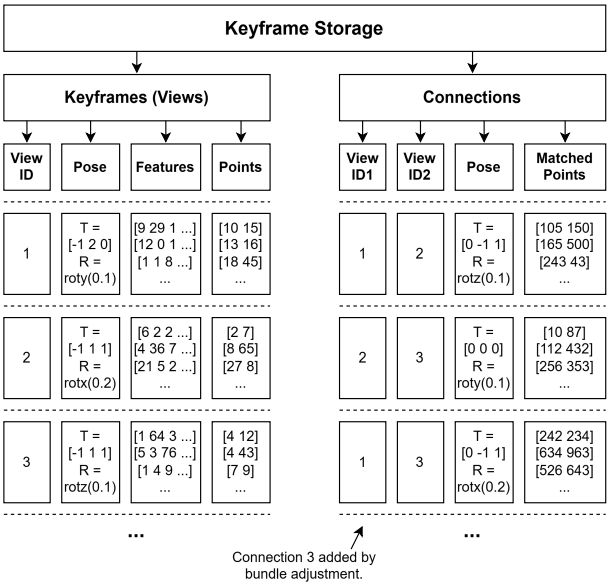

Fig. 2. Diagram portraying the keyframe storage. There is one view entry per keyframe, but there can be more connections than keyframes, due to non-consecutive frame connections found during bundle adjustment.

feature, and hence triangulated world point can be detected in multiple keyframes, so the matched map points are stored in the map point set, like entry three in Figure 3. Consequently, using the map point connections that the bundle adjustment has found, the algorithm will also create new connections between keyframes in the keyframe storage.

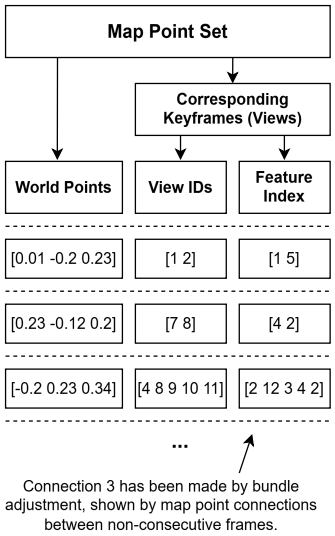

Fig. 3. Diagram portraying the map point set. There is one world point entry per detected, triangulated feature, but each point can hold more than two keyframe IDs, if that feature is detected in more than two keyframes, again found during bundle adjustment.

Following initialization, the main loop of the VO algorithm is similar to the map initialisation workflow, although the keyframe storage and the set of map points are used to improve our pose estimate. The flow begins as normal by extracting ORB features, before estimating the camera pose from the previous keyframe. This transformation is then applied to the previous map points, and features re-matched from these newly adapted map points. The matching of these map points can be iteratively improved; as the estimated pose improves, the map points come closer to the extracted features, aiding in matching them. In order to further improve this estimated position, bundle adjustment can be used to refine the map points further. After this process has completed, the algorithm will decide whether the current frame is a keyframe. This decision is characterized by two parameters that ensure a relatively constant stream of keyframes with respect to distance, not time. This attempts to create an optimal usage of storage. If a frame is not considered a keyframe, the algorithm will continue to the next frame.

Finally, each keyframe, or more specifically, each translation from one keyframe to the next, is collated into the set of 3D transformations, which contains a rotation and translation in 3D space. This is the output of the Visual Odometry algorithm.

III. Proposed Sensors Fusion Design For Complete System

The previous section described the mono-systems used in this study. The main contribution of this paper consists in the design of a novel combination of these blocks for more robust and accurate vehicle tracking. The system uses novel sensor fusion techniques to correct a visual odometry path, before further fusing GNSS data to provide a reliable, low-cost, error-correcting system. This system is able to overcome the challenges described in Section I including the challenge of modularity, lending its design to customisability dependant on the required hardware topology. A high-level overview of the proposed fusion system is provided in Figure 4.

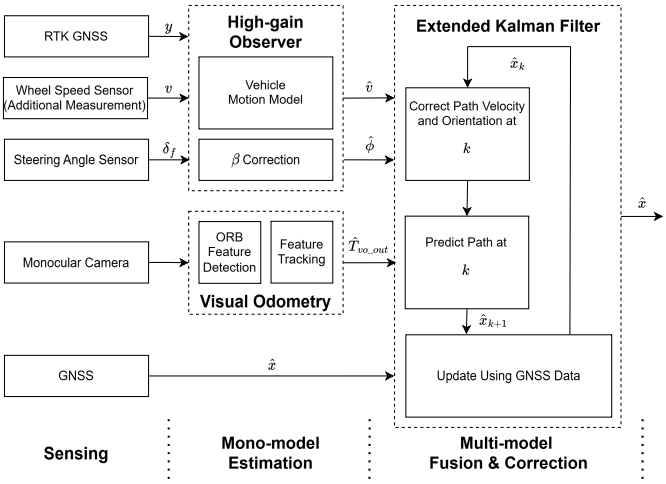

Fig. 4. High level overview of the proposed fusion framework. The three steps are mentioned at the bottom on the diagram, and follow from left to right.

This section provides details about the design of the complete fusion framework, following the three steps in Figure 4: Sensing, Mono-model Estimation, and Multi-model Fusion & Correction.

A. Sensing

The system's selected sensors were outlined in Section II. The GNSS data, as well as wheel speed and steering angle data, are input to the high-gain observer. One issue that arises due to the high-gain nature of the observer is commonly known as the peaking phenomenon. Because there is a high amplification of the error between output and the true value, any input noise causes high output noise, particularly shown at start-up in the transient phase, when this error is large. This means that any noise in the output vector y is propagated through the system. Since y is measured solely using the GNSS sensor, the precision of the sensor is directly proportional to the performance of the observer's estimation. Moreover, if the implemented GNSS sensor is a low-cost local sensor, its precision is relatively low, and the peaking phenomenon could degrade the output signal. To tame this response, a filter can be added to the GNSS data path ahead of the observer. When using higher-precision sensors such as RTK GNSS, this additional filtering is not required and can be disabled.

Remark 3: Alternatively, another method of providing filtered GNSS data to the observer would be using the estimated position from the system's main EKF, denoted $\hat{x}_k$ in Figure 4, as a feedback signal y into the high-gain observer provided in (12). The GNSS block has been separated in this paper for clarity, and to enable the calculation of each subsystem's output open-loop, for modularity, and ease of mono-model verification.

B. Mono-model Estimation

During the mono-model estimation, the high-gain observer takes in the current GNSS position of the vehicle, y, and the current velocity from the wheel speed sensors as an additional input. This estimates and outputs the current estimated orientation and velocity of the car, which is passed to the final EKF as shown in Figure 4. These output estimates are in triangular form, z, as shown in (15), and hence must be transformed back to values of orientation and velocity acceptable by the EKF. This transformation includes the calculation of β using the steering angle data. These transformations can be completed after the observer has finished, saving unnecessary computation if the values are not needed instantaneously. Furthermore, the high-gain block does not require any other inputs from other blocks and hence is fully modular from the system, meaning it could be swapped with another similar system if the application requires it. Another benefit of the modularity is the potential for parallel computation. Moreover, the VO and high-gain observer are distinct blocks, and use different input and output parameters, meaning they can be processed in parallel.

The path estimated by the VO algorithm is taken from the keyframe's relative poses. These poses contain a set of transformations, each containing a 3D rotation and translation, and each pose relative to the previous one. Since the multi-model EKF shown in Figure 4 requires the VO algorithm to output a single translation, the rotation is applied to the translation output by the VO system, transforming it into an absolute translation. The process of this removal of relative VO rotation is shown in Figure 5. The poses in Figure 5 are stored in the keyframe data, using the notation $Pose\{n\}.\{T/R\}$, where n represents the particular keyframe and T/R are the matrices which describe the pose's translation and rotation respectively. For simplicity, planar angles have been used to portray the rotations R, although R usually describes a 3D rotation.

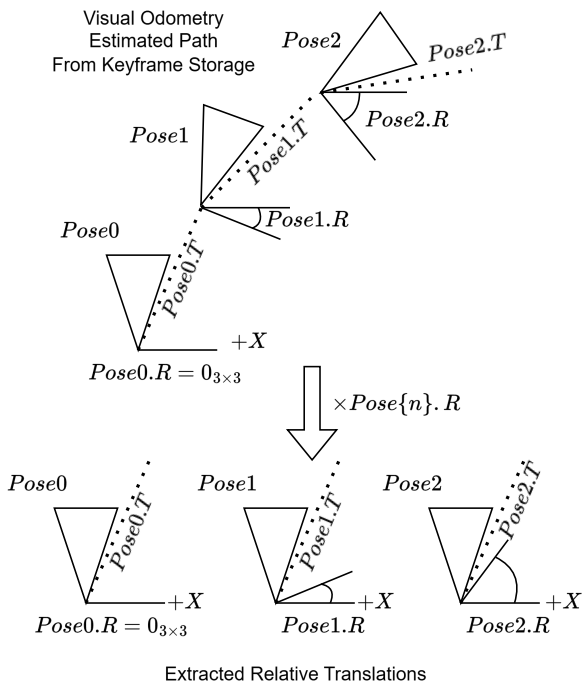

Fig. 5. A figure showing part of the proposed orientation correction process. Each keyframe in the Visual Odometery algorithm contains a relative pose. The absolute predicted path can be extracted at any timestep k by multiplying it, and all future pose's by the rotation matrix at timestep k. These poses can then be oriented to match those in the output system, as described in (16).

Before this translation can be used, it must be rotated into the same coordinate space as the other measurements in the EKF. Since the EKF state vector contains 2D planar coordinates, this process consists of removing the camera's initial orientation and reapplying the initial orientation of the new coordinate space. Furthermore, since the Z axis is known to be the vertical axis, the VO algorithm can remove any rotation in the Z axis before reapplying the initial orientation. This is done to mimic the other mono-model system's outputs. The latter two rotations of (16) orient this transformation into the same heading as the output system. Hence, the set of planar-corrected translations at time step k, denoted as $T_{pl\_corr} \in \mathbb{R}^{3\times1}$, can be found:

$$\begin{aligned} R &= Keyframe.R[k]R_{cam}^{T}R_{out}, \\ T_{pl\_corr}[k] &= Keyframe.T[k]R, \end{aligned} \tag{16}$$

where $Keyframe.T \in \mathbb{R}^{3\times1\times n}$ is the set of translation matrices output by the VO algorithm. Similarly, $Keyframe.R \in \mathbb{R}^{3\times3\times n}$ is the set of rotations (relative to the previous pose) output by the VO algorithm. The rotation matrices $R_{cam}, R_{out} \in \mathbb{R}^{3\times3}$ describe the rotations that map from the initial camera orientation to a heading in the positive

X axis, and then from there to the initial output orientation respectively. In other words, (16) transforms the keyframe translations from the visual odometry's coordinate system into the coordinate system used by the rest of the system. For example, if the visual odometry assumes the camera begins facing the negative X axis, then R_{cam} will be a rotation 180° in the z axis. Similarly, if the system is actually facing the positive Y axis, R_{out} will be a rotation -90° in the Z axis. These matrices could be combined into a single transformation, although have been separated to maintain the modularity of the blocks. R_{cam} is defined fully by the visual odometry algorithm, and R_{out}, in the proposed fusion framework, is defined fully by the vehicle's initial absolute heading. In future systems, it may be the case that the output system does not have an absolute heading, and so R_{out} can be changed to match the coordinate system the new system uses.

Due to an unknown camera extrinsic matrix, as well as the use of a monocular camera, the scale of the output can only be estimated. Due to this, the scale is estimated by the VO algorithm using the camera intrinsics determined by the camera sensor modelled. The estimated scale is then applied to each translation, and then later corrected using the velocity estimated by the observer. Given the velocity and time between keyframes, the corrected output distance in the direction of the VO output can be found. However, to achieve this, the visual odometry output data must be conveyed in a way to allow the translation and time to be displayed. This way, it can be easily interpreted by the Kalman filter system. Equation (17) shows the concatenation of the planar-corrected translation T_{pl_corr} computed by (16), with the frame time to give the final output $T_{vo\_out} \in \mathbb{R}^{4\times1}$.

$$T_{vo\_out} = \begin{bmatrix} T_{pl\_corr} \\ \Delta t_{[k-1]\to[k]} \end{bmatrix}, \tag{17}$$

where $\Delta t_{[k-1]\to[k]}$ is the time between the latest keyframe $[k]$ and its predecessor $[k-1]$. Algorithm 1 shows the steps of the proposed post-processing to the VO output. Some of the newly defined variables used in Algorithm 1 includes $KFStorage$, the Keyframe Storage, defined earlier in Figure 2. $Keyframe[k]$ is the kth entry in this storage. Each keyframe contains a pose, which includes a translation T, rotation R, and the time at which the keyframe was taken t. T_{scaled} represents the VO output translation, which has been scaled using the estimated camera scale, found from the camera intrinsics. This can be used as an additional step before (16), replacing $Keyframe.T[k]$. $observer\_\phi\_estimate\_valid$ is set if the observer has a correct orientation estimate, and $last\_known\_rotation$ is the last rotation before the observer estimate became invalid. Furthermore, there are a few functions specified, including $RotationMatrix(\mapsto coords_x)$. This function finds the rotation matrix which maps one coordinate system ($coords_x$) to the positive X axis. It follows that the transpose of this function will define the rotation matrix which maps the positive X axis back to the coordinate system ($coords_x$). The function $len(store)$ returns the number of elements contained in the storage $store$. $Pose(k)$ finds the kth pose in the keyframe storage. Lastly, $EstCamScale()$ finds the VO output scale, as estimated by the camera intrinsic matrix.

Algorithm 1 Visual Odometry Output Processing

procedure VOOUTPUTPROCESS($KFStorage$)
  ▷ *Keyframe Storage is input.* ◁
  $R_{cam} \leftarrow RotationMatrix(\mapsto coords_{cam})$
  ▷ *Rotation matrix that maps the camera to $+X$.* ◁
  if $OutputCoordinateSystemKnown()$ **then**
    $R_{out} \leftarrow RotationMatrix(\mapsto coords_{out})^T$
    ▷ *If the system has knowledge of the output, find rotation matrix that maps $+X$ to the output.* ◁
  else
    $R_{out} \leftarrow \mathbf{I}_3$
    ▷ *If unknown, assume output direction is $+X$.* ◁
  end if
  $Keyframe[0] \leftarrow [T = 0_{3\times1}, R = 0_{3\times3}, t = 0]$
  ▷ *Set initial pose equal to all zeros.* ◁
  for $k \leftarrow 1$ to $len(KFStorage)$ **do**
    $Keyframe[k] \leftarrow Pose(k)$ from $KFStorage$
    $T_{scaled} \leftarrow Keyframe.T[k] \times EstCamScale()$
    ▷ *Extract every transformation from storage, and scale it using the camera intrinsics' estimate.* ◁
    if observer_ϕ_estimate_valid **then**
      $R \leftarrow Keyframe.R[k] * {R_{cam}}^T * R_{out}$
      $last\_known\_rotation \leftarrow Keyframe.R[k]$
      ▷ *Apply equation (16).* ◁
    else
      $R \leftarrow last\_known\_rotation * {R_{cam}}^T * R_{out}$
    end if
    $T_{pl\_corr}[k] \leftarrow T_{scaled}[k] * R$
    $\Delta t_{[k-1]\to[k]} \leftarrow Keyframe.t[k] - Keyframe.t[k-1]$
    ▷ *Find frametime from frame k-1 to k...* ◁
    $T_{vo\_out} \leftarrow [T_{pl\_corr}[k]; \Delta t_{[k-1]\to[k]}]$
    return T_{vo_out}
    ▷ *... and concatenate onto the output.* ◁
  end for
end procedure

If the initial compass heading, ϕ, is known, this can be further applied to correct the relative VO path for orientation, turning it into an absolute heading. However, this is unnecessary, since ϕ is estimated by the observer, the relativity of the VO output is automatically corrected when it is combined with the other data streams in the final EKF. Since this estimated ϕ only describes the X and Y plane orientation, the observer correction automatically removes any drift in the Z axis. This correction process is further detailed in the next section.

C. Multi-model Fusion & Correction

As Figure 4 shows, the two mono-model systems compute their individual outputs separately, then fuse their respective outputs as they are combined during the multi-model EKF fusion framework system. The proposed EKF-based fusion algorithm takes in the estimated position from the visual odometry, T_{vo_out}, and the transformed state estimates from

the observer $\hat{z}$. The high-gain observer's predicted velocity and orientation are used to correct the visual odometry path's scale and orientation. This forms the *"predict"* half of the extended Kalman filter, as shown in the top half of Figure 6. Once the prediction stage of the Kalman filter has combined both sets of data, it outputs a corrected path.

This output is further fused to the absolute position, courtesy of the GNSS sensor, forming the *"update"* in the EKF, the lower half of Figure 6. Since the *"predicted"* path is relative to the unknown initial conditions, the update stage takes in the corrected path, and steers it closer to the absolute position, using the absolute GNSS data. Due to the fusion between the GNSS and the visual odometry both providing independent outputs, failure of either one allows the system to continue working, providing short-term redundancy. Similarly, the fusion will reduce drift introduced by any of the systems, due to inadequate models, or incorrect accidental feature tracking. These two halves combine together to form the novel EKF-based fusion algorithm. A diagram of the proposed EKF-fusion algorithm is shown in Figure 6.

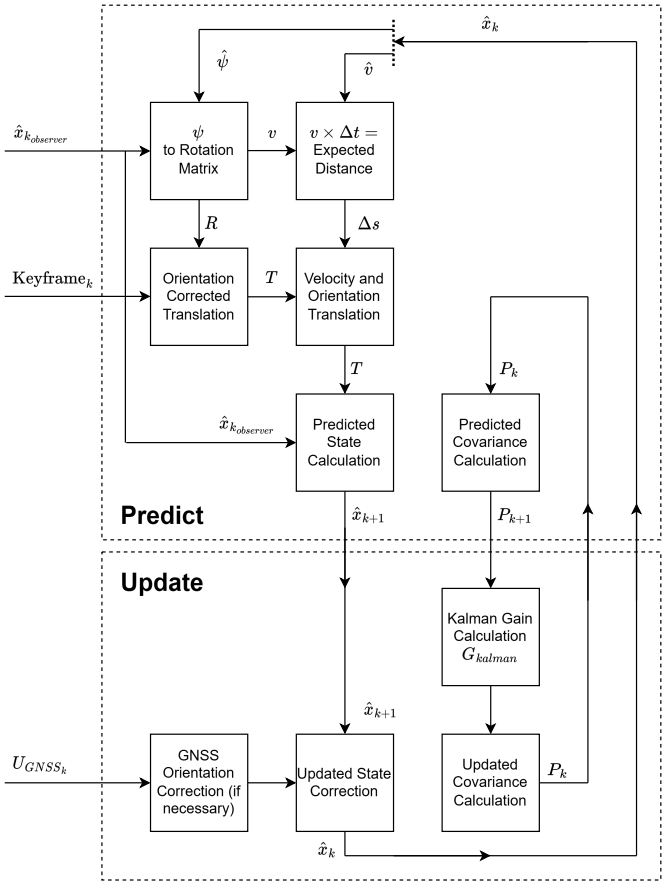

Fig. 6. The EKF prediction and update steps following standard equations with the extra preprocessing stages depicted. This diagram is representative of the EKF block in Figure 4.

As discussed previously, both the exact scale of the extracted VO path, and the initial orientation are unknown. Despite this, using the observer's predictions for the current velocity and orientation, this knowledge can be calculated. We note that before outputting the VO path, the VO algorithm uses the inverse camera rotation to obtain a planar coordinate-corrected translation. This translation can then be rotated, also planar, using ϕ to re-orient the camera path such that the translation faces in the correct direction. Assuming the observer's output has converged, using the orientation estimated by the high-gain observer $\hat{\phi}$ at time step k, a planar rotation to correct the orientation can be applied. The set of translations output by the visual odometry which have been corrected for orientation ϕ using (19) are denoted by $T_{\phi\_corr} \in \mathbb{R}^{3\times1\times n}$.

$$T_{\phi\_corr}[k] = \Phi \; T_{vo\_out}[k], \tag{18}$$

where Φ is the rotation matrix:

$$\Phi = \begin{bmatrix} \cos(\hat{\phi}[k]) & -\sin(\hat{\phi}[k]) & 0 & 0 \\ \sin(\hat{\phi}[k]) & \cos(\hat{\phi}[k]) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{19}$$

Therefore, the estimated ϕ from the observer can be used in the place of a compass, directly orienting the path. Unfortunately, this does mean the path accuracy is proportional to the observer's accuracy, meaning if the observer is slow to converge, the path may get worse. This is found to only be an issue at velocities close to zero, as the noise from the sensors outweighs the actual sensed value. Since the observer contains constraints on the expected inputs and states, the point of divergence can be easily quantified. As discussed previously in Section III-A, the system can include an optional GNSS filter to remove some of the noise input to the high-gain observer. This can be used for the secondary benefit of reducing this divergence point, since the effective noise floor of the sensor is lowered.

A similar technique is used to correct for the unknown scale output from the VO algorithm. The velocity of the car is estimated through the high-gain observer, from the additional wheel speed sensor measurement. This can be combined with the time taken between keyframes, a multiple of the frame-time, to find the distance covered in one translation. This can be corrected for by normalising the translation, before multiplying by the distance travelled, as follows:

$$T_{v\_corr}[k] = \frac{T_{vo\_out}[k]}{\|T_{vo\_out}[k]\|_2} \times \hat{v}[k] \times \Delta t_{[k-1]\to[k]}, \tag{20}$$

where $\hat{v}$ is the estimated velocity from the high-gain observer. Since the orientation and scale correction are associative, by correcting for both orientation using $\hat{\phi}$ given (18) and scale using $\hat{v}$ given (20):

$$T_{corrected}[k] = \frac{\Phi \; T_{vo\_out}[k]}{\|T_{vo\_out}[k]\|_2} \times \hat{v}[k] \times \Delta t_{[k-1]\to[k]}. \tag{21}$$

Given that (21) is made from (18) and (20), this also means the processes can be computed separately, or only one of them computed at a time. For example, if orientation prediction is not functioning, due to the lack of GNSS data, the scale can still be corrected without affecting the orientation correction. If this is the case, the last known orientation is still stored, and applied to orient the entire future path, with the relative

orientations still providing path tracking during the transient sensor-failure state.

To construct the EKF, the process is converted into a system of equations in order to output the prediction. These build upon the standard EKF equations, although contain additional terms for each of the prediction methods. Even though the system's output only contains the estimated position output $\hat{X}, \hat{Y}$, the observer's estimated state variables $\hat{v}$ and $\hat{\phi}$ are also stored in the EKF's state vector estimate, as shown in (22). These two extra terms are internally stored such that they are accessible to correct the path, as provided in (21). The error of this system is judged based upon the localization $\hat{X}, \hat{Y}$ accuracy, so the systems output is defined as $\hat{y} = C_{out}\hat{x}$ where

$$C_{out} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \hat{x} = \begin{bmatrix} \hat{X} & \hat{Y} & \hat{v} & \hat{\phi} \end{bmatrix}^T. \tag{22}$$

They are also optionally accessible at the output of the whole system with a change in C_{out} matrix. This could be in order to interface with other systems, or provide additional information if desired.

The EKF predict and update equations gain extra terms for the multiple systems being used for prediction: the observer and the visual odometry. The notation $\hat{x}_k$ is used to define the value of the predicted EKF state at time step k. Equation (23) describes the prediction equation of the next state of the system, $\hat{x}_{k+1}$. The equation uses a weighted sum of the previous estimate, as well as the combination of observer state estimates and the visual odometry translations. The observer state estimate only affects the lower two state variables, $\hat{v}$ and $\hat{\phi}$. Conversely, the VO path prediction only affects the positional states, $\hat{X}$ and $\hat{Y}$. However, if the GNSS data is not correct, then the observer will either fail to converge or incorrectly affect the EKFs orientation state. For this reason, the system will revert to uncorrected camera data using only the VO path to maintain tracking, maintaining reliability. The wheel-speed velocity correction also only happens if the velocity prediction is correct. Similarly, the position estimation depends on which systems are enabled. Equations (24) shows these as piecewise functions. The function $Diag(n,\ ...,\ m)$ represents a diagonal matrix, where the elements on the diagonal are given from n in the top left to m in the bottom right, and all other elements are zero.

$$\hat{x}_{k+1} = F\hat{x}_k + B\hat{x}_{observer} + BT_{camera}, \tag{23}$$

where $F = \mathbf{I}_4$, $B = Diag(\epsilon,\ \epsilon,\ 1,\ 1)$ and

$$T_{camera} = \begin{cases} T_{corrected} & \text{if observer estimate valid} \\ T_{uncorrected} & \text{if observer estimate invalid} \end{cases}$$

$$\epsilon = \begin{cases} 0.5 & \text{if both observer and camera enabled} \\ 1.0 & \text{if either observer and camera enabled} \\ 0.0 & \text{if neither enabled} \end{cases} \tag{24}$$

Despite the VO keyframe transformations being completely relative in translation and rotation, the uncorrected transformations $T_{uncorrected}$, computed as in (25), are not completely relative translations. This is because the system always keeps track of the last known correct absolute position and orientation T_{last_orient} of the vehicle. This means that while future uncorrected transformations will be relative to the last known absolute position, the system will still track in the correct direction, deviating from the absolute path only as far as the system drifts. The system also keeps track of the current velocity, in order to be able to correct the path to the same scale, T_{last_scale}, as before the sensor failure. If both the scale and orientation estimates are invalid, the system uses the most recent scale and orientation estimates T_{last_SandO}. Both orientation and scale correction systems work independently, allowing the system to use every working sensor at all times.

$$T_{uncorrected} = \begin{cases} T_{last\_scale} & \text{observer estimate } \phi \text{ valid and } v \text{ invalid} \\ T_{last\_orient} & \text{observer estimate } \phi \text{ invalid and } v \text{ valid} \\ T_{last\_SandO} & \text{all observer estimates invalid} \end{cases} \tag{25}$$

The estimated covariance of the next time step, $\hat{P}_{k+1}$ is calculated in (26), in accordance with the type of prediction made. For example, the covariance should not increase if the state variable has not been changed due to an incorrect or ignored sensor reading.

$$\hat{P}_{k+1} = F\hat{P}_k F^T + Q_{process} + BQ_{observer}B^T + \Phi Q_{camera}\Phi^T \tag{26}$$

where $Q_{process}, Q_{observer},$ and Q_{camera} are the covariance matrices for each system, which can be tuned to provide responses dependant on the weighting of each system.

$$\begin{aligned} Q_{process} &= \sigma^2_{process}\mathbb{I}^4, \\ Q_{observer} &= \sigma^2_{observer}Diag(Ob_{en},\ Ob_{en},\ v_{corr},\ \phi_{corr}), \\ Q_{camera} &= \sigma^2_{camera}Diag(Cam_{en},\ Cam_{en},\ 0,\ 0), \end{aligned} \tag{27}$$

where σ^2_x refers to the variance weighting of each system, including the process variance, Ob_{en} and Cam_{en} refer to the observer and camera being enabled respectively, and the v_{corr} and ϕ_{corr} refer to the velocity and orientation of the visual odometry being corrected respectively.

To provide a clear overview of the proposed EKF methods, algorithmic outlines of both halves of the EKF flow are detailed in Algorithms 2 and 3. The hierarchical structure of these algorithms is that the main EKF function is listed as EKFFUSION in Algorithm 3. This is the top level function, and calls each of the PREDICT (through OUTPUTCORRECTION) and UPDATE functions in Algorithms 2 and 3 respectively. Algorithm 2 contains a few newly defined variables: $new\_keyframe\_translation$ is set when a new keyframe has been added. Moreover, since the addition of keyframes is aperiodic, the system need only react when a new keyframe has been added. $observer\_v\_estimate\_valid$ is set when the observer's estimate of v is correct. In reality this is only zero when there is a sensed failure of the wheel speed sensors. Φ_{last} is the last known orientation correction value, in relation to T_{last_orient}. Lastly, the function $atan2(y, x)$ defines the four quadrant inverse tangent of y and x. It functions as the MATLAB function $atan2$ does.

Algorithm 2 Mono-model Correction & Prediction

procedure OUTPUTCORRECTION($k, \hat{x}, \hat{P}, z, \delta_f$)
  ▷ *Current timestep, KF state, covariance, and observer state is input.* ◁
  ▷ *z is the 6x1 triangular system estimate.* ◁
  $\hat{v}[k] \leftarrow \sqrt{\hat{z}_2[k]^2 + \hat{z}_5[k]^2}$
  $\beta = \tan^{-1}(l_r \times \tan(\delta_f)/(l_f + l_r))$
  $\hat{\phi}[k] \leftarrow atan2(\hat{z}_5[k], \hat{z}_2[k]) - \beta$
  ▷ *Calculate $\hat{\phi}, \hat{v}$ using (2) and (15).* ◁
  $d\hat{x}_{observer_1} \leftarrow z_1[k] - z_1[k-1]$
  $d\hat{x}_{observer_2} \leftarrow z_4[k] - z_4[k-1]$
  ▷ *z_1 and z_4 are $\hat{x}$ and $\hat{y}$.* ◁
  $d\hat{x}_{observer_3} \leftarrow \hat{v}[k] - \hat{v}[k-1]$
  $d\hat{x}_{observer_4} \leftarrow \hat{\phi}[k] - \hat{\phi}[k-1]$
  ▷ *Change in observer output estimates* ◁
  if new_keyframe_translation **then**
    ▷ *If new keyframe, hence new translation.* ◁
    $T_{vo\_out} \leftarrow$ VOOUTPUTPROCESS(kf_str)
    ▷ *Defined in Algorithm 1.* ◁
  else
    $T_{vo\_out} \leftarrow [0; 0; 0; 0]$
  end if
  $\hat{x}[k+1], \hat{P}[k+1] \leftarrow$ PREDICT($\hat{x}, \hat{P}, d\hat{x}_{observer}, T_{vo\_out}$)
end procedure
function PREDICT($\hat{x},\ \hat{P},\ d\hat{x}_{observer},\ T_{vo\_out}, \hat{v}$)
  ▷ *Input previous state and covariance, current observer and VO changes* ◁
  if observer_ϕ_estimate_valid **then**
    $\Phi_{last} \leftarrow \Phi$ from (19), using $\hat{\phi}[k] = d\hat{x}_{observer_4}$
  end if
  $T_{camera}[k] \leftarrow \Phi_{last} T_{pl\_corr}[k]$
  ▷ *Orient the translation T_{vo_out}, in (17). If orientation unknown, use Φ_{last}.* ◁
  if observer_v_estimate_correct **then**
    $s \leftarrow \hat{v}[k] * \Delta t_{[k-1]\rightarrow[k]}$
    ▷ *Find distance, and correct vector magnitude.* ◁
    $T_{camera}[k] \leftarrow T_{camera}[k]\ /\ \|T_{camera}[k]\| * s$
  end if
  $\hat{x}[k+1] \leftarrow F\hat{x}[k] + B\hat{x}_{observer} + BT_{camera}$ (23)
  $\hat{P}[k+1] \leftarrow F\hat{P}[k]F^T + Q_{process} + BQ_{observer}B^T + \Phi_{last}Q_{camera}\Phi_{last}^T$ (26)
  ▷ *Update State Estimate and Covariance* ◁
  return $\hat{x}[k+1], \hat{P}[k+1]$
end function

The update section of the EKF only contains one measurement, the GNSS sensor, meaning the correction can be performed using a similar structure to [10] or [6]. The update function begins by calculating the Kalman gain.

$$G_{kalman}[k] = \hat{P}[k]H^T(H\hat{P}[k]H^T + R)^{-1}, \qquad (28)$$

where $H = \mathbf{I}_4$ and $R = Diag(\sigma^2_{GNSS_x},\ \sigma^2_{GNSS_y},\ 0,\ 0)$, describing $\sigma^2_{GNSS_x}$ and $\sigma^2_{GNSS_y}$ as the variance of the GNSS sensor in the X and Y directions respectively. Lastly, the updates to the state variables, and the covariance matrix are defined as follows:

$$\begin{aligned} \hat{x}[k] &= \hat{x}[k] + G_{kalman}[k](U[k] - H\hat{x}[k]), \\ \hat{P}[k] &= (\mathbf{I}_4 - G_{kalman}[k]H)\hat{P}[k], \end{aligned} \qquad (29)$$

where $U[k]$ is the input sensor data being used to update the system's estimates. This data has been turned into the form $[U_{GNSS_x}[k],\ U_{GNSS_y}[k],\ \hat{x}_3[k],\ \hat{x}_4[k]]^T$, where $U_{GNSS_x}[k]$ and $U_{GNSS_y}[k]$ are the GNSS sensor's position measurements in X and Y respectively. This ensures that the previous prediction measurements are unaffected by the GNSS measurement. Algorithm 3 describes the full update function programmatically, as well as shows the overall system loop which completes the Kalman filter process. Line 6 shows the execution of the high-gain observer algorithm, with its output being the non-transformed triangular system z, as detailed in (6). Algorithm 3 contains the function $HighGainObserver(y, v)$, which returns the internal estimated states of the transformed system z, (15), given the plant output y and additional wheel speed measurement v. Its form was defined in Section II-B.

Algorithm 3 Multi-Model Update & Fusion

1: **procedure** EKF FUSION($v,\ \delta_f,\ U_{GNSS}, Frames_{cam}$)
2: ▷ *Wheel speed, GNSS data, and Camera Frames are input.* ◁
3: ▷ *Begin with timestep generation.* ◁
4: $N \leftarrow$ No. Timesteps
5: $y \leftarrow X, Y$ from U_{GNSS}
6: $z \leftarrow HighGainObserver(y, v)$
7: ▷ *GNSS and Observer run in advance.* ◁
8: $\hat{x}[1],\ \hat{P}[1] \leftarrow$ OUTPUTCORRECTION($0, x_0, P_0, z_0$)
9: ▷ *Initial timestep estimation* ◁
10: **for** $k \leftarrow 1 : N-1$ **do**
11: $\hat{x}[k],\ \hat{P}[k] \leftarrow$ UPDATE($\hat{x}, \hat{P}, U_{GNSS}[k]$)
12: ▷ *Update at current timestep ...* ◁
13: $\hat{x}[k+1],\ \hat{P}[k+1] \leftarrow$ OUTPUTCORRECTION($k, \hat{x}, \hat{P}, z, \delta_f$)
14: ▷ *... before predicting at the next timestep.* ◁
15: **end for**
16: **end procedure**
17: **function** UPDATE($\hat{x},\ \hat{P},\ U_{GNSS}[k]$)
18: ▷ *Update takes in current state, covariance, and GNSS Measurement.* ◁
19: $U[k] \leftarrow [U_{GNSS_x}[k],\ U_{GNSS_y}[k],\ \hat{x}_3[k],\ \hat{x}_4[k]]^T$
20: $G_{kalman}[k] \leftarrow \hat{P}[k]H^T(H\hat{P}[k]H^T + R)^{-1}$
21: ▷ *Rearrange GNSS input, calculate Kalman gain.* ◁
22: $\hat{x}[k] \leftarrow \hat{x}[k] + G_{kalman}[k](U[k] - H\hat{x}[k])$
23: $\hat{P}[k] \leftarrow (\mathbf{I}_4 - G_{kalman}[k]H)\hat{P}[k]$
24: ▷ *Update state estimates and covariance matrix.* ◁
25: **return** $\hat{x}[k], \hat{P}[k]$
26: **end function**

IV. SIMULATION RESULTS USING CARLA

The system was comprehensively tested, with each part of the full system tested individually before being combined, in order to prove each component of the system. An overview

of the software implementation is provided in Section IV-A. Next, the comparison of each mono-model system is shown in Section IV-B. Following that, the results of the full fusion framework are shown in Section IV-C.

A. Software

Each component of the fusion system is implemented in MATLAB. In order to provide realistic testing inputs, all sensor data is collected from dedicated automotive software CARLA. The simulated sensor's parameters have been selected to replicate their real-life counterparts, allowing for accurate experimental results.

1) CARLA: is an open-source package created for simulating real-world environments for testing and training autonomous driving algorithms [28]. In this work, it is used to model vehicles and sensors for gathering data, which are passed into the fusion system. The system's outputs, estimated position, and other parameters will be compared to the ground truth provided by the simulator in order to evaluate their performance. Figure 7 shows a selection of sensor outputs collected from inside the simulator. Each sensor has many configurable parameters, including noise and offsets.

```
SteeringAngle, 4.151114463806152, Speed, 7.660456865035259, Rotation, -86.60899353027344
SteeringAngle, 5.307870864868164, Speed, 7.671642690517057, Rotation, -86.5227279663086
SteeringAngle, 6.437304496765137, Speed, 7.670688715691614, Rotation, -86.42449188232422
SteeringAngle, 7.541878223419189S, Speed, 7.683134092082569, Rotation, -86.3196829663086
SteeringAngle, 7.750218868255615, Speed, 7.682294317832007, Rotation, -86.20035552978516
SteeringAngle, 7.697110652923584, Speed, 7.696517054517867, Rotation, -86.07677459716797
SteeringAngle, 7.644180774688721, Speed, 7.698934034043511, Rotation, -85.94165802001953
SteeringAngle, 7.570820331573486, Speed, 7.7143734828077255, Rotation, -85.80455780029297
SteeringAngle, 7.501430034637451, Speed, 7.719397299644282, Rotation, -85.65817260742188
SteeringAngle, 7.414157867431641, Speed, 7.7358567351298335, Rotation, -85.51174926757812
SteeringAngle, 7.333572864532471, Speed, 7.742587695786368, Rotation, -85.35797882080078
SteeringAngle, 7.237768173217773, Speed, 7.759759682027839, Rotation, -85.20561981201172
SteeringAngle, 7.150282382965088, Speed, 7.767377077938749, Rotation, -85.04729461669922
```

(a) An example of δ_f, v and ϕ data

(b) An example of the 1920 x 1080px RGB camera data

Fig. 7. Sensor data captured from CARLA.

The world, vehicles, and sensors are all controlled using Python scripts. These scripts act as clients which can communicate with the central CARLA server. This makes it possible to run multiple segregated scripts at the same time, to produce multiple user-controlled vehicles, each with individual sensors. This feature mimics real life, where each vehicle is individually controlled, without having influence on each another. In this implementation, a single instance of the CARLA server and client program are run locally, collecting data from a single vehicle. The path taken by the vehicle is predetermined by the script, and the server is told to produce deterministic results to provide repeatability of tests.

We modelled the vehicle as a Mercedes-Benz C-Class, and represents an average-sized sedan. This choice affects only the wheelbase. This is calculated by outputting each wheel position using a Python script, before calculating the vector magnitude between their positions. The four sensors needed to test the proposed fusion system are simulated by CARLA and modelled with parameters akin to their real-life counterparts. For example, the simulated RTK GNSS sensor is modelled from the u-blox ZED-X20P [23], a 25Hz RTK GNSS with an RTK accuracy of 1cm. This RTK sensor was used for all displayed simulations, without the use of the additional Kalman Filter input to the observer, discussed in Section III-A. As a low-cost alternative, the u-blox M8L [20] is an automotive specification local-only GNSS module, capable of running at 30Hz, with a measurement accuracy of 1.5m. This high accuracy is achieved using internal sensor fusion with a rudimentary inertial measurement unit. This local-only sensor can be bought for just over £25 and was only used to provide comparative results in Table II. Since the sensor has a lower precision than the RTK sensor, the additional KF, detailed in Section III-A, was used to reduce noise input to the observer. The camera used is modelled on the monocular OV5647 [16] sensor, which outputs 30Hz RGB frames of 1920x1080px. This camera was chosen due to its high enough resolution, coupled with the low price of under £10 per module. The wheel speed sensor is modelled as having discrete possible values, and measuring at a rate of 120Hz with an accuracy of 0.1kmph. Similarly, the steering angle sensor is modelled as having a sample rate of 120Hz, and an accuracy of 1.5°. The ground truth and actual orientation are also extracted from CARLA, but never given to the system.

2) MATLAB: All systems were implemented in MATLAB [29], chosen for its useful features, such as debugging, as well as good documentation and additional packages. YALMIP [30] is used to compute the LMI needed to compute the high-gain observer's gains, with MOSEK used as the solver [31]. The Matlab data structures $imageviewset$ and $worldpointset$ are used for the keyframe storage and map point set respectively.

The whole system requires heavy parameterisation. Table I, below, details some of these parameters, from the VO algorithm, and their corresponding descriptions. Some of these values are determined by the setup of the camera simulation in CARLA, more specifically the parameters related to the lens and camera sensor.

Name	Value	Description
fov	110	Camera Field of View (degrees)
image_size	[1080,1920]	Camera Sensor Size (px)
optical_centre	[960,540]	Point where lens does not distort light (px)
focal_length	672.2	Camera Sensor to Lens Distance (px)
scale_factor	1.2	ORB Features
num_levels	8	ORB Levels
num_points	1000	Target ORB Features per Frame
min_parallax	3	Minimum Angle Between Frames in First Keyframe Connection (degrees)

TABLE I
A BRIEF DESCRIPTION OF EACH PARAMETER IN THE VISUAL ODOMETRY INITIALISATION.

B. Mono-Model Accuracy and Efficiency

Before implementing the proposed fusion technique, each system is individually simulated in their mono-model form. The results of this, alongside the results achieved by fusion of multiple techniques are detailed in Figure 8. The systems presented are pure Visual Odometry (VO), the observer estimated position $(\hat{z}_1, \hat{z}_4)$, the corrected VO using the observer's orientation estimate $\hat{\psi}$, and wheel speed v. Lastly, the proposed solution shows the fusion of the corrected VO with RTK GNSS. The initial state of the transformed system is chosen to be $z_n(0) = 5,\ n = 1, ..., 6$, making the initial position of the original state vector $x_0 = [5, 5, 7.07, 0.79]$. Following this initialisation, the vehicle navigates a roundabout. This is a 270° turn counter-clockwise, taking the furthest exit of the roundabout. Each x-y graph in the results has a flipped y-axis to show the path from a top-down view.

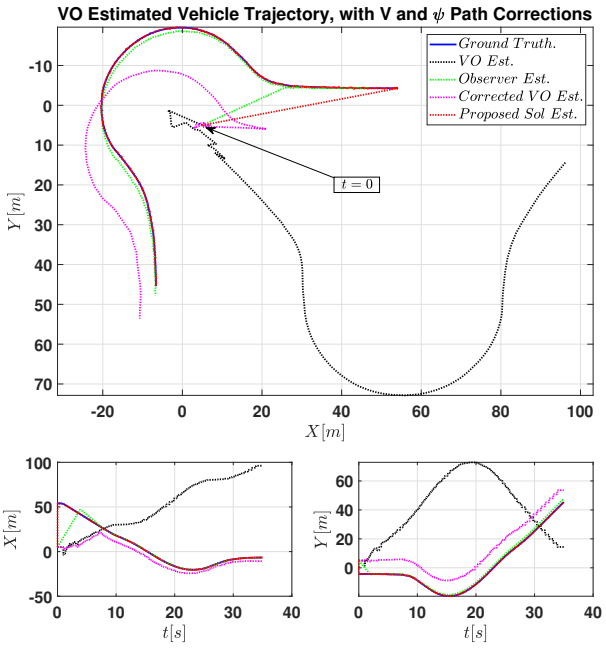

Fig. 8. CARLA Roundabout input, Showing a comparison between the mono-model estimations - Visual Odometry and High-Gain Observer, and multi-model systems - Observer corrected Visual Odometry, and the Proposed Fusion Framework. As shown by the lower position-time graphs, each system begins at $X = 5, Y = 5$, before taking a counter-clockwise roundabout.

There are two performance metrics that are noteworthy, namely the path replication, and the absolute error. The path replication concerns the shape of the estimated path, irrespective of orientation or scale. The absolute error is the numerical distance between the estimated path and the ground truth. The lower half of Figure 8 shows the temporal accuracy of the estimated vehicle position. The visual odometry, after overcoming its initial feature-matching transient stage, maintains a path tracking that replicates the true path well, implying good path replication. However, this solution alone contains a very large absolute error, not least due to its incorrectly scaled, rotated nature. This is due to the unknown absolute position, and hence the route is oriented and scaled incorrectly. On the other hand, techniques like the stand-alone high-gain observer match the absolute position much better, although they propagate a lot of noise through the system, leading to a lower path replication. The accuracy of the high-gain observer is found to be quite good, although it is found to lag slightly behind the true value, meaning the accuracy is not as good as the proposed system, or GNSS alone.

System	MRE (m)	Processing Time (s)
GNSS Sensor (Local Only) [20]	1.608	0.092
GNSS Sensor (RTK) [23]	0.169	0.092
Visual Odometry [24] [25]	80.850	286.102 (60.54×)
High-Gain Observer (Local) [11]	5.301	4.916 (1.04×)
High-Gain Observer (RTK) [11]	4.007	4.726 (1.00×)
Proposed HG-Corr VO (Local)	16.264	286.555 (60.63×)
Proposed HG-Corr VO (RTK)	17.007	286.438 (60.61×)
Proposed Fusion System (Local)	**0.877**	286.480 (60.62×)
Proposed Fusion System (RTK)	**0.143**	286.530 (60.63×)

TABLE II
MEAN RELATIVE ERROR (MRE) AND COMPUTATION TIME FOR EACH SENSOR SYSTEM.

Table II shows the mean relative error of each of the techniques, calculated as the sum of the mean error between the estimated position and the ground truth, for each time-step. Using inbuilt MATLAB timing, the computation time for each part of the system is calculated, and also displayed in Table II. This metric's data has been computed from the values produced by the comparison of each of the different systems, in Figure 8. In addition to these results, the same tests were repeated using the local GNSS only[20], with an Extended Kalman Filter (EKF) to filter the GNSS data ahead of the observer. This significantly improves the state estimates of the observer. The local-only system shows similar performance trends to the RTK system, although with a lower overall accuracy. In regard to computation time, as expected, the visual odometry takes significantly longer, over 60 times longer than the observer alone. Although there is most likely a lot of processing time in the display of each frame and graphing the result, the computation time still far exceeds any other option. With more processing power, or optimisation, this figure could drop drastically. Most importantly, the proposed fusion framework does not take a significant amount more computation time than the visual odometry alone.

C. Multi-Sensor Extended Kalman Filter Fusion Framework

Figure 9 shows the performance of the proposed fusion system. The vehicle performs the same roundabout manoeuvrer as in Figure 8, although the ground truth's initial position has been assumed to begin at the graph's origin. The observer and EKF's initial states are still the same as x_0 defined earlier. We note that the fusion system gives very promising results, proving that the integration of the four sensors is successful and adds value to each individual estimator. The path retains the correct orientation and shape, as well as self-correcting to the absolute position via the RTK GNSS fusion. Accuracy stays high, only slightly lowering during transient periods, staying under 0.2m absolute error most of the time, remaining

under 0.4m for the whole path. Convergence time is short, and low error values are achieved in a near-instantaneous transient phase.

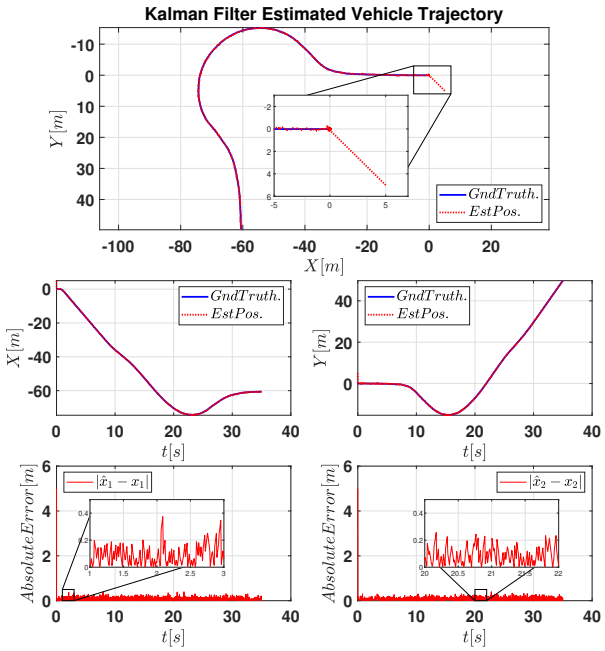

Fig. 9. CARLA Roundabout input, with Kalman filter fusion of RTK GNSS, Wheel Speed Sensor, and Visual Odometry.

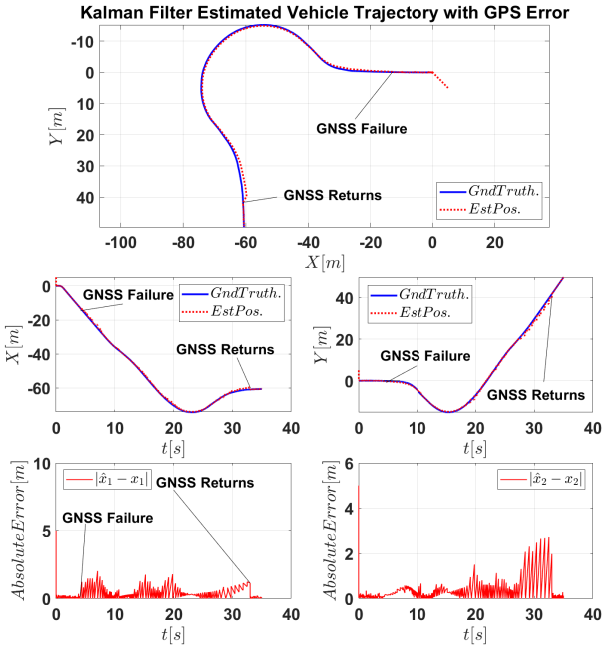

Fig. 10. CARLA Roundabout input, with complete RTK GNSS failure (no data) at $t = 4.25$, and return at $t = 33$.

D. Multi-Sensor EKF with Temporary Sensor Failure

Figure 10 portrays the performance of the whole proposed fusion system given the scenario where RTK GNSS readings would become unusable, for example, under a bridge. The system receives no signals from the RTK GNSS sensor between times $4.25s \leq t \leq 33s$, simulating a complete loss of both base station and satellite signals. This time is chosen especially to show resilience through a transient state; in this example, for the entire roundabout there is no RTK GNSS signal. Since the output vector for the observer is reliant on GNSS sensor data, the case of no signal is recognised, and the observer state estimates are consequently locked, until the RTK GNSS recovers. This means the only sensor in use is the camera. The performance of the camera provides enough data to fill in the space, and the path is tracked closely. Convergence speed and error initially is very good, and the path only drifts slightly over the 28.75s outage, reaching just over 2m error at peak. Immediately upon the GNSS return, the system re-converges on sub-0.2m accuracy.

V. Conclusion

This paper proposed a novel technique describing the combination of the high-gain nonlinear observer and visual odometry. This combination has allowed the systems to work independently, as modular blocks, while taking advantages from both systems to provide an accurate and robust tracking output. More specifically, the full fusion framework improved the mean relative error with respect to all of the explored mono-model techniques for little additional computing effort upon the visual odometry. The speed of convergence proves that the system is capable of retaining accuracy, and the good performance of the system when the GNSS sensor goes completely offline, with its data lost, proves the system's resilience.

By demonstrating resilience to sensor failure, the implemented system provides a novel design, which adds value to existing systems, as well as lays a path for future work through modularity. The system provides an accurate solution, which is not reliant on expensive radar equipment, but instead extracting value from pre-existing sensors.

The system's absolute position output would be beneficial for connected-car networks, so seeing a development towards a cooperative SLAM system would be a good future research task. This could be simulated using CARLA's multi-client capability to simulate a full connected-car network, as discussed earlier. Applications to experimental datasets like Kitti or NuScenes would be able to further prove the system's ability in a more focused application. Lastly, implementation of a state-of-the-art Visual Odometry algorithm could show large improvements in output performance, efficiency, and ease of deployment on real hardware.

References

[1] D. N. S. D. A. Salleh, "Study of vehicle localization optimization with visual odometry trajectory tracking," Ph.D. dissertation, Université Paris Saclay (COmUE), 2018.

[2] R. Rajamani, *Vehicle Dynamics and Control*. Springer, 2006.

[3] Đ. Petrović, R. Mijailović, and D. Pešić, "Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers," *Transportation Research Procedia*, vol. 45, pp. 161–168, 2020, transport Infrastructure and systems in a changing world. Towards a more sustainable, reliable and smarter mobility.TIS Roma 2019 Conference Proceedings. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146520301654

[4] M. Massar, I. Reza, S. M. Rahman, S. M. H. Abdullah, A. Jamal, and F. S. Al-Ismail, "Impacts of Autonomous Vehicles on Greenhouse Gas Emissions—Positive or Negative?" *International Journal of Environmental Research and Public Health*, vol. 18, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1660-4601/18/11/5567

[5] A. Singandhupe and H. M. La, "A Review of SLAM Techniques and Security in Autonomous Driving," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 602–607.

[6] K. Wahlqvist, "A Comparison of Motion Priors for EKF-SLAM in Autonomous Race Cars," Ph.D. dissertation, KTH ROYAL INSTITUTE OF TECHNOLOGY, 2019.

[7] H. Alai and R. Rajamani, "Low-cost camera and 2-D LIDAR fusion for target vehicle corner detection and tracking: Applications to micromobility devices," *Mechanical Systems and Signal Processing 206 (2024)*, 2023.

[8] M. Pryde, O. Alrazouk, L. Nehaoua, H. Hadj-Abdelkader, and H. Arioui, "Visual-inertial lateral velocity estimation for motorcycles using inverse perspective mapping," *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2022.

[9] L. Gao, X. Xia, Z. Zheng, H. Xiang, Z. Meng, X. Han, Z. Zhou, Y. He, Y. Wang, Z. Li, Y. Zhang, and J. Ma, "Cooperative Localization in Transportation 5.0," *IEEE Transactions on Intelligent Vehicles*, vol. 9, no. 3, pp. 4259–4264, 2024.

[10] L. Chu, Y. Zhang, M. Xu, Y. Shi, and O. Yang, "Design of a robust side slip angle observer using adaptive Kalman filter," in *2010 International Conference On Computer Design and Applications*, vol. 3, 2010, pp. V3–168–V3–171.

[11] H. Bessafa, C. Delattre, Z. Belkhatir, A. Zemouche, and R. Rajamani, "Nonlinear Observer Design Methods Based on High-Gain Methodology and LMIs with Application to Vehicle Tracking," *2023 American Control Conference (ACC)*, 2023.

[12] W. Jeon, A. Zemouche, and R. Rajamani, "Tracking of Vehicle Motion on Highways and Urban Roads Using a Nonlinear Observer," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 644–655, 2019.

[13] ——, "Nonlinear Observer for Vehicle Motion Tracking," *2018 Annual American Control Conference (ACC)*, 2018.

[14] M. Abbaszadeh and A. Zemouche, *Security and Resilience in Cyber-Physical Systems*. Springer Nature Switzerland AG, 2022.

[15] Tesla, "Tesla Vision Update:Replacing Ultrasonic Sensors with Tesla Vision," 2023. [Online]. Available: https://www.tesla.com/en_gb/support/transitioning-tesla-vision

[16] Omnivision, *OV5647 Datasheet*, 2009. [Online]. Available: https://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/ov5647_full.pdf

[17] Y. Lu, "'Lost Time for No Reason': How Driverless Taxis Are Stressing Cities," 2023. [Online]. Available: https://www.nytimes.com/2023/11/20/technology/driverless-taxis-cars-cities.html

[18] D. Wang and F. Qi, "Trajectory Planning for a Four-Wheel-Steering Vehicle," *IEEE International Conference on Robotics & Automation*, 2001.

[19] Bosch, "Electronic stability program," 2023. [Online]. Available: https://www.bosch-mobility.com/en/solutions/driving-safety/electronic-stability-program

[20] u blox, *NEO-M8L-06B Data sheet*, 2022. [Online]. Available: https://content.u-blox.com/sites/default/files/NEO-M8L-06B-ADR450_DataSheet_UBX-20058645.pdf

[21] M. Q. Hamdan, C. H. Foh, A. Ul Quddus, S. Hancock, O. Holland, and R. Woodling, "Using real-time kinematics algorithm in mission critical communication for accurate positioning and time correction over 5g and beyond networks," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, 2022, pp. 1–5.

[22] Novatel, "OEM729 Performance Specifications," 2023. [Online]. Available: https://docs.novatel.com/OEM7/Content/Technical_Specs_Receiver/OEM729_Performance_Specs.htm

[23] u blox, *ZED-X20P-00B Data sheet*, 2025. [Online]. Available: https://www.u-blox.com/sites/default/files/documents/ZED-X20P-00B_DataSheet_UBXDOC-963802114-12690.pdf

[24] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions On Robotics*, 2015.

[25] Mathworks, "Monocular Visual Simultaneous Localization and Mapping," 2024. [Online]. Available: https://uk.mathworks.com/help/vision/ug/monocular-visual-simultaneous-localization-and-mapping.html

[26] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *International Conference on Computer Vision*, sep 1999.

[27] S. Waslander and J. Kelly, "Visual Perception for Self-Driving Cars," 2019. [Online]. Available: https://www.coursera.org/learn/visual-perception-self-driving-cars/home/week/1

[28] Alexey Dosovitskiy and German Ros and Felipe Codevilla and Antonio Lopez and Vladlen Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[29] The MathWorks Inc., "MATLAB version: 23.2.0.2485118 (R2023b)," Natick, Massachusetts, United States, 2023. [Online]. Available: https://www.mathworks.com

[30] J. Löfberg, "YALMIP : A Toolbox for Modeling and Optimization in MATLAB," in *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[31] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 10.1.*, 2024. [Online]. Available: http://docs.mosek.com/latest/toolbox/index.html