A Navigation and Perception System for an Autonomous Race Car

Beatriz Tavares Silva
Supervised by: Prof. Alexandre José Malheiro Bernardino and Prof. Paulo José da Costa Branco
Instituto Superior Técnico, Lisboa, Portugal

June 2022

Abstract

This dissertation can be divided in two: image processing and sensor fusion, and was developed with the aim of designing a sensor system that makes it possible to turn one of PSEM's (Projecto de Sustentabilidade Energética Móvel) prototypes, GP17.Evo, into an autonomous race car. The sensors chosen for this system include a camera, a GNSS and IMU. The camera allows for relative positioning while the GNSS and IMU account for the global position of the car as well as an estimation of its direction and speed. The data generated by the sensor system would subsequently be transferred into a control unit. The detection of the race track limits was accomplished through image processing techniques, using OpenCV, and relying on footage from a race PSEM took part in. Then, two variations of the Kalman filter were implemented in order to determine the optimal choice in terms of sensor fusion. In a simulation environment, more specifically MATLAB, and based on a vehicle model of GP17.Evo, the Extended and Unscented Kalman filters were used to try to combine all of the data provided by various sensors. This work is the first step towards a fully functioning autonomous GP17.Evo that will hopefully be able to race in a new Greenpower category.
Keywords: PSEM, Autonomous Car, Navigation system, Sensor fusion

1. Introduction

PSEM is a student group at Instituto Superior Técnico whose goal is to build electric race cars so that they are as efficient as possible. PSEM represents the school at the Greenpower Challenge race where each competing team is free to build their car as they see fit, as long as they comply with the rules established by the committee. Over the years, PSEM has built several cars, but this work will focus on the upgrade of the 2017 prototype GP17.Evo.

1.1. Motivation

For PSEM, the overall weight of the car's components highly influences the performance during the race. The biggest setback when it comes to this, though, is the fact that the pilot weighs about the same as the fully assembled empty car, which means that removing this extra weight will allow it to reach higher speeds. At the moment, there is no category in the Greenpower Challenge for autonomous cars which means PSEM would be a pioneer in the competition. This is the main drive for the team as well as the main drive for this work, which will focus on the sensory aspect of a self-driving race car.

1.2. Objectives

The goal of this work is to propose a low-cost, basic sensor system for the autonomous GP17.Evo; through image processing, perform lane detection; and, lastly, implement two sensor fusion algorithms using a model of the car. To that end, a study on various sensors, namely a GNSS, IMU and a camera, will be conducted. Then, image processing techniques will be researched as well as sensor fusion methods. Finally, tests will be done to assess the robustness of the implementation.

1.3. Proposed system

The sensors chosen for the system include a GNSS, supplying position coordinates; an IMU, to extract parameters such as acceleration and an estimation of the heading; and a camera, allowing for relative positioning. A Raspberry Pi is meant to perform all of the processing and establish the connection to the camera and the Arduino, which is, in turn, connected to the GNSS/IMU module. All of the image processing will be done using OpenCV. The optimization of the trajectory and control of the car will be done in another work. In the end, the system proposed here will feed the eventual Control Unit all the useful information on variables such as position, speed and direction, in real-time, allowing the car to race autonomously.

2. State of the Art

Many of the techniques developed, especially those related to lane and obstacle detection, are a result of a need to increase safety on the road. TEINVEIN (TEchnology and INnnovation for VEhicle INtelligence) incorporates ADAS and is a project that puts a self-driving car on the road while, in racing, there are now competitions such as Roborace, Formula Student Driverless [1] and Indy Autonomous Challenge. As this work focuses exclusively on the sensory facet, we will explore their implementations as well as different combinations of sensors and ways of merging them.

2.1. Sensor Fusion

All of the recent studies done on this subject use a combination of sensors to increase the number of variables retrieved, with the GNSS being its most basic component.

We will take the example of TEINVEIN [2] whose system includes a LIDAR, a camera, a GPS-RTK, two radars, wheel encoders and IMU on the car's center of mass besides those integrated with the GPS. In a similar way, both AMZ Driverless versions [3, 1] share most of the sensors of TEINVEIN but positioned differently and making use of ROS.

These are considerably complex systems but if we take a step back and look at something akin to [4] where only a GNSS and IMU are utilized, we see that these work fairly well if there are no failures of the GNSS for extended periods of time. By introducing a camera into the equation [5], there is now the possibility of complementing data from the other two sensors, GNSS and IMU. Should we introduce two cameras, now stereo vision [6], depth can also be estimated.

The approaches to sensor fusion are vast [7] and aim to estimate the state of a system given all of the information gathered from the various devices. One of the most common methods is the Kalman Filter and its variations such as its non linear version EKF [3]; an improvement on the EKF, the UKF (Unscented Kalman Filter) [2]; and the less com-

putationally expensive while maintaining estimation accuracy, Multi-State Constraint Kalman Filter (MSCKF) [8].

2.2. Image Processing

One of the most famous implementations of lane detection is GOLD [9], which takes into account both lane and obstacle detection. They make use of the IPM technique to remove the perspective of the camera mounted on the car and make the lane lines parallel as if seen from a bird's eye view. Both GOLD and M. Aly in [10] define the image plane and the world space to create the transformation. In [11] after getting this bird's eye view, boundary modeling [12] is done, fitting road boundaries to a geometric model, and then lane detection. [13] applies the same concept but in reverse: first the lanes are detected and then for lane following the model is created.

In order to distinguish the road lines from everything else edge detection is often used. John Canny [14] developed the Canny edge detector that effectively detects edges based on variations of pixel intensity in the image [15, 16]. An alternative to this is to filter the image through color [17], as we know lane markings are usually lighter than the road (gray or black). The edge detection alone is not enough to detect the lines that create the lane or the curb so we must find a way to pinpoint them in each image. So, we can model the road and try to fit the edges found to it or we can apply a feature extraction method such as the Hough Transform [16] to detect the lines formed by the edges of the lane. As the Hough transform may not be enough, RANSAC (Random Sample Consensus) [10, 5] may follow it for a more robust detection.

3. System

This section will present, in a more detailed fashion, the elements that make up the system.

3.1. Camera

A camera will act as the eyes of the car and be used to locate it relative to the race track.OpenCV is a computer vision library that was chosen for processing what was captured by the camera as it has hundreds of algorithms for real time image processing. These include face and object recognition, but, most importantly, functions for edge and contour detection, as will be shown later.

3.1.1 Placement

The options for the camera's placement on the car are indicated in figure 1. Ideally, no structural changes should be made in order to ensure the car still complies with Greenpower's rule book and thus, position A is the best option as it is not obstructed by a pilot's helmet and is the safest and most stable place for it.

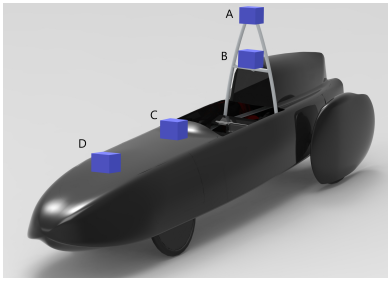


Figure 1: Camera placement possibilities.

3.2. GNSS and IMU

The GNSS and IMU integrated into a single device, the NEO-M8U. This module was chosen and has a built-in accelerometer and gyroscope and uses Untethered Dead Reckoning (UDR), which processes the signals from the GNSS and IMU together in order to get an as accurate as possible position when there is a poor GNSS signal. Some of the characteristics of this module include a horizontal accuracy of 2.5 m; a velocity accuracy pf 0.05 m/s; 1° of heading accuracy; and a refresh rate of 30Hz (at most).

The NEO-M8U needs an antenna, so AA.160.301111 by Taoglas was chosen. This antenna is able to connect to GPS, GLONASS and Galileo and will be connected to the NEO-M8U module.

3.3. Raspberry Pi

The Raspberry Pi will be the "on-board computer" of the car. The model that will be used is the Raspberry Pi 4B with 8GB of RAM. The operating system installed on it is the "Raspberry Pi OS", relatively similar to a Linux environment. The headless Raspberry Pi has no need for peripherals and the connection is made through SSH and VNC, making it possible to access the device without any additional hardware.

The Arduino Uno is a useful asset and will be connected to the GNSS/IMU module, since the libraries for that device are already available in Arduino's IDE through download. Accessing the data supplied by the NEO-M8U is thus made much easier. In order to be able to use this configuration, Arduino's IDE has to be installed on the Raspberry Pi and a connection through USB between these two has to be made.

3.4. Summary and limitations of the system

We must safeguard the fact that the camera was lent to PSEM for this work while all the other elements were purchased, so the total cost is divided in two, one with and one without the camera. The price of acquisition is 242.39€ without and 772.39€ with the camera.

Because the system proposed is not a very complex one, it is not possible to accommodate a high number of variables. Although the Greenpower Challenge is a race between many teams of students, each participating with their own car, the detection of other vehicles will not be taken into account, with the system focusing only on the race track. The refresh rate is dependent mostly on the camera and all processing associated with it as well as the Raspberry Pi's computing power. If the refresh rate is low, the car will not be able to reach higher speeds because data will be lost during the time the system is still processing past events. As a result of these hindrances, the system would only work under a specific set of conditions: well lit, clear environments at low speeds.

4. Methods

The system is summarized in figure 2, where we go from the gathering of data from the sensors to outputting variables that describe the car's movement and positioning. However, this section will concentrate solely on image processing and sensor fusion, using the term "Kalman Filter", in figure 2, to describe the EKF or UKF, as both have similar operations.

Many of the images shown in the following sections were taken from a film made during a race in which PSEM competed in October of 2021 with GP19. From now on, the color space RGB will be referred to as BGR interchangeably, as OpenCV performs all processing operations based on the latter component order.

Visual Studio 2019 with the 4.5.1 version of OpenCV for image processing and MATLAB 2021a for sensor fusion

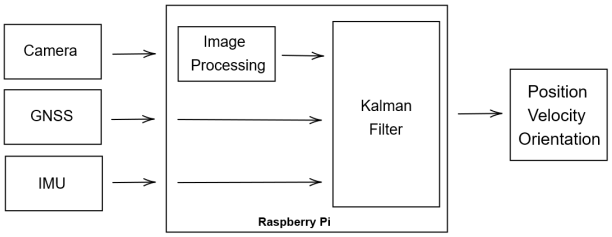


Figure 2: System Diagram.

Figure 5: Reference rectangle for the color gray.

were used for testing on an AMD Ryzen 7 @ 2.30 GHz PC.

4.1. Image Processing

The image processing stage is summarized in figure 3 where each new frame retrieved from the live video goes through a set of transformations. This section, and consequently section 5.1, where results are shown, are analyzed in a qualitative manner.

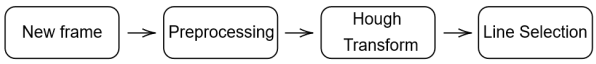


Figure 3: Image processing block diagram.

4.1.1 Preprocessing

The goal of preprocessing is to try to minimize the effect of noise and enhance the image's features for it to follow through to post-processing. Figure 4 describes the steps of this process.

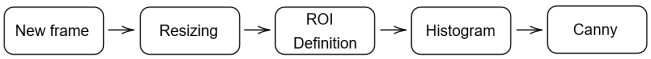


Figure 4: Preprocessing block diagram.

Region of Interest

Firstly, each new frame is scaled down by a factor of two and the ROI is defined, in this case, keeping its original width as to not lose any area of visible road on the right and left sides. The goal of the ROI is to focus only on the race track and attempt to eliminate as many unnecessary elements present in the image as possible, while also making the complete process faster. Here, there must be a compromise between trying to only see the race track and not mistakenly eliminating important features of it, especially when the car is very close to the boundaries of the track.

Histogram Back Projection

This portion of the implementation is divided in two parts: the histogram and the back projection. The goal of the histogram is to pinpoint the gray of the race track, from a reference, in this case, a rectangle of this particular color, much like the one in figure 5, whose size can be adjusted as needed. Then, the back projection creates a mask of the original image taking the histogram as its input. These operations can be done for a BGR or HSV image, but, as will be shown later, the HSV color space is not very reliable when it comes to detecting the color gray.

The function `calcHist()`, that calculates the histogram, analyses an input image (the rectangle of the race track) based on a quantization of each component of the color space given their ranges and creates a plot of the frequency of pixels with a given intensity. This plot is passed on to the `calcBackProject()` function so that it finds, in the original frame, how many pixels fit the distribution of the histogram. Ideally, this will create a mask that shows only the track in white and everything else in black and makes the edge detection much easier.

Canny Edge Detector

Edge detection is a useful asset, especially after color segmentation, because we get to pinpoint where the intensity of the pixels changes. The stage where the Canny Edge detection is applied is made up of four operations: a Gaussian Blur, the Canny detector itself, dilation and, finally, erosion. All of the operations that are not the Canny detector are used in order to strengthen the detection of the edges and ultimately of the lines formed by them in post-processing. The most important parameters are the kernel sizes for these operations, `GaussianBlur()`, `erode()` and `dilate()`, and the thresholds of the Canny function.

The Gaussian blur's kernel size is ruled by the following: if it is too high, nothing will be discernible, but if it is too small everything will pass through making the Canny detect things that would normally be eliminated by the blur. Dilation and erosion are relatively similar and the sizes of their kernels reflect, much like the blur, how the pixels are involved in the calculations that make up the output. The thresholds of `Canny()` are the inputs of the hysteresis performed and indicate which edges are considered real and which are not. The gradients calculated by the Canny edge detector that fall between the two threshold values defined are only accepted if connected to a pixel above the higher threshold.

4.1.2 Post-Processing

Now that our image contains mostly elements that are relevant for further processing, the PHT will be applied.

Hough Transform

Most lines will be straight or approximately straight and should be seen on either side of the car. A good way to detect these lines is through the Hough transform. OpenCV offers an implementation of the Probabilistic Hough transform, which detects line segments and will be applied based on the work developed in [18].

The Hough transform can be manipulated by changing its inputs, most importantly the `threshold`, `minLineLength` and `maxLineGap` and outputs each line found as a vector of four parameters: the x and y coordinates of its start and end points.

Line Selection

From the Hough Transform, a set of valid lines is gathered and we must filter them in order to determine the ones that are effectively the real ones. The filtering is done in two stages: eliminating clearly invalid lines and then choosing between the ones left which is the line that is closest to the center as it will match the edge of the track. The first stage eliminates detected lines that are features of the car (figure 6) through the bounds formed by the edges of the trapezoid, i.e. everything inside that area is seen as invalid. The helmet is not considered because the camera was not placed in the center but shifted slightly to the left in this footage. Lines that are nearly vertical (angles between 89° and 91°) or nearly horizontal (angles between

-1.1° and 1.1°) are also not accounted for as sometimes tire marks, rails or other landmarks are present, although this is a compromise that does not always pay off.

Figure 6: Trapezoid that eliminates features of the car.

The second stage makes use of the dot product. Figure 7 shows a sketch of the road where C is the center of the image; $\vec{v}$ is the vector from C to the line detected (in green, with orange for start and end points); and $\vec{n}$ is the normal vector of the line.

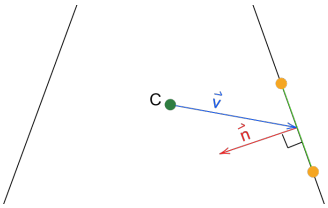


Figure 7: Sketch of the road and vectors for the line detected.

The goal is to determine the distance between the line and C through the dot product.

To try to fix the fact that sometimes there are no lines on both sides, two vectors of previous lines were created: `previousLinesLeft` and `previousLinesRight`. Both of these store, for each frame, the line detected. If, for the current frame, there is no line on one or both sides, we check these vectors and repeat the last found line.

The FPS was also a crucial deciding factor as a lower FPS means a bigger delay between the evaluation of the track and the decision making by the control unit.

4.2. Sensor Fusion

The merging of the data gathered from the sensors will be done using two techniques: the EKF and the UKF.

4.2.1 Extended Kalman Filter

The EKF [19] has two steps: the prediction, where we estimate the state and update the error covariance matrix, and the correction, where predictions are corrected and the covariance matrix is once again updated.

Equation (1) introduces the state vector $\mathtt{x}$ as a function of f, depending on the previous state of the system, an input vector u, and w, the process noise.

$$\mathtt{x}_k = f(\mathtt{x}_{k-1}, u_{k-1}, w_{k-1}) \tag{1}$$

The measurement vector, y_k, is described in equation (2) as a function of h. This function receives as inputs the current state prediction, $\mathtt{x}$, and the measurement noise, v_k.

$$\mathtt{y}_k = h(\mathtt{x}_k, v_k) \tag{2}$$

As mentioned above, the variables w_k and v_k are the process and measurement noise, more specifically, a Gaussian noise with zero mean and covariances $\mathbf{Q}_k$ and $\mathbf{R}_k$, respectively. $\mathbf{F}$, $\mathbf{W}$, $\mathbf{H}$ and $\mathbf{V}$ are the Jacobian matrices of f in respect to $\mathtt{x}_k$, of f in respect to w, of h in respect to $\mathtt{x}_k$, and of h in respect to v.

During the "Predict" stage, the algorithm tries to estimate the *a priori* state vector and error covariance, $\hat{\mathtt{x}}_k^-$ in equation and $\mathbf{P}_k^-$ given the *a posteriori* estimate from the previous state, the control vector and estimate error covariance.

Then, for the "Correct" portion of the implementation, we now calculate the Kalman gain, $\mathbf{K}_k$, given the measurement, while also updating the error covariance.

In practice, the EKF and the UKF, will be implemented based on the work developed in [20], where a controller for GP17.Evo is designed so that the car can run on a racing simulator. First, the state vector $\mathtt{x} = \begin{bmatrix} v_x & v_y & \dot{\psi} & e_\psi & d_y & s & \psi & x & y \end{bmatrix}^T$, as partially defined by [20], contemplates the following variables: the longitudinal (v_x [m/s]) and lateral (v_y [m/s]) velocities, the yaw rate ($\dot{\psi}$ [rad/s]), heading error (e_ψ [rad]), lateral distance error (d_y [m]) and a measure of progress (s [m]). Additionally, for this work, the car's heading, ψ [rad], and position, x [m] and y [m], will be appended. While v_x, v_y, $\dot{\psi}$, ψ, x and y are relatively straightforward, the remaining three variables are specific to [20]. The error e_ψ and d_y refer to the the deviation of the heading angle to a previously planned trajectory and to the distance of the car to the middle of the track, and s corresponds to the distance the car has travelled, denoted by perpendicular progress lines along the middle of the track.

The input vector $u = \begin{bmatrix} a & \delta \end{bmatrix}^T$ shows which variables control the vehicle at every time step, in this case, a [m/s^2], its longitudinal acceleration, and δ [rad], the steering angle.

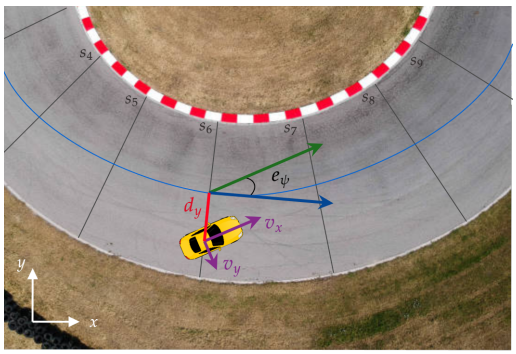


Figure 8: Diagram of the track and state variables.

Figure 8 is a representation of the car on the track and some of the variables of the state vector. The progress lines in gray, s_4, s_5, s_6 and so on, intersect the blue line that goes along the middle of the track. The velocities v_x and v_y are drawn in purple and d_y, in red, shows the distance from the car to the center of the track. Finally, e_ψ is the angle between the green and the blue vectors, translating into the difference in the value of the heading between the planned trajectory and the car's actual direction. The coordinate system is also present in the lower left corner of the image. In order to be able to calculate x and y [21], the heading, ψ, had to be added to the state vector and consequently to the state function. Given this, and after adding the last three expressions for these additional variables, function f is complete, as evidenced by equation (3).

$$f(\mathrm{x};u)=\begin{bmatrix}\dot{\psi}v_y+a\\ \frac{C_{\alpha f}+C_{\alpha r}}{m}\frac{v_y}{v_x}+\frac{l_fC_{\alpha f}-l_rC_{\alpha r}}{m}\frac{\dot{\psi}}{v_x}-\dot{\psi}v_x-\frac{C_{\alpha f}}{m}\delta\\ \frac{l_fC_{\alpha f}-l_rC_{\alpha r}}{I_z}\frac{v_y}{v_x}+\frac{l_f^2C_{\alpha f}+l_r^2C_{\alpha r}}{I_z}\frac{\dot{\psi}}{v_x}-\frac{l_fC_{\alpha f}}{I_z}\delta\\ \dot{\psi}-\frac{v_x\cos e_\psi-v_y\sin e_\psi}{1-d_yC(s)}C(s)\\ v_x\sin e_\psi+v_y\cos e_\psi\\ \frac{v_x\cos e_\psi-v_y\sin e_\psi}{1-d_yC(s)}\\ \dot{\psi}\\ v_x\cos\psi-v_y\sin\psi\\ v_x\sin\psi+v_y\cos\psi\end{bmatrix} \tag{3}$$

Before moving on, we must safeguard the fact that, in this model, there is a singularity for $v_x = 0$. To overcome this, whenever v_x is close to zero, the input u forcibly feeds the system an acceleration so that it never reaches this value. In addition, the track is assumed to have a constant width. In $\mathrm{y} = \begin{bmatrix} v_x & d_y & \psi & x & y \end{bmatrix}^T$, the variables taken from the sensors are shown, creating the vector y. The first two, x and y, would be retrieved from the GNSS, v_x and ψ come from the IMU and d_y would be given by the camera. The measurement function assumes that all the variables are measured directly from the sensors.

The definition of the noise covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ are in equations (4) and (5), respectively.

$$\mathbf{Q} = \mathrm{diag}\left(\sigma_{v_x}^2,\ \sigma_{v_y}^2,\ \sigma_{\dot{\psi}}^2,\ \sigma_{e_\psi}^2,\ \sigma_{d_y}^2,\ \sigma_s^2,\ \sigma_\psi^2,\ \sigma_x^2,\ \sigma_y^2\right) \tag{4}$$

$$\mathbf{R} = \mathrm{diag}\left(\sigma_{v_x}^2,\ \sigma_{d_y}^2,\ \sigma_\psi^2,\ \sigma_x^2,\ \sigma_y^2\right) \tag{5}$$

The Jacobian of f is shown in (6).

$$\mathbf{F}=\begin{bmatrix}0&\frac{\partial f_1}{\partial v_y}&\frac{\partial f_1}{\partial\dot{\psi}}&0&0&0&0&0&0\\ \frac{\partial f_2}{\partial v_x}&\frac{\partial f_2}{\partial v_y}&\frac{\partial f_2}{\partial\dot{\psi}}&0&0&0&0&0&0\\ \frac{\partial f_3}{\partial v_x}&\frac{\partial f_3}{\partial v_y}&\frac{\partial f_3}{\partial\dot{\psi}}&0&0&0&0&0&0\\ \frac{\partial f_4}{\partial v_x}&\frac{\partial f_4}{\partial v_y}&1&\frac{\partial f_4}{\partial e_\psi}&\frac{\partial f_4}{\partial d_y}&\frac{\partial f_4}{\partial s}&0&0&0\\ \frac{\partial f_5}{\partial v_x}&\frac{\partial f_5}{\partial v_y}&0&\frac{\partial f_5}{\partial e_\psi}&0&0&0&0&0\\ \frac{\partial f_6}{\partial v_x}&\frac{\partial f_6}{\partial v_y}&0&\frac{\partial f_6}{\partial e_\psi}&\frac{\partial f_6}{\partial d_y}&\frac{\partial f_6}{\partial s}&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ \frac{\partial f_8}{\partial v_x}&\frac{\partial f_8}{\partial v_y}&0&0&0&0&\frac{\partial f_8}{\partial\psi}&0&0\\ \frac{\partial f_9}{\partial v_x}&\frac{\partial f_9}{\partial v_y}&0&0&0&0&\frac{\partial f_9}{\partial\psi}&0&0\end{bmatrix} \tag{6}$$

Equation (7) defines $\mathbf{H}$, the Jacobian of h.

$$\mathbf{H}=\begin{bmatrix}1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\end{bmatrix} \tag{7}$$

4.2.2 Unscented Kalman Filter

The UKF [22] is overall quite similar to the EKF in the way that it also estimates the state of a given system and then, at its correction stage, attempts to improve it based on a received measurement. It is generally used for highly non-linear models and does not require the Jacobians $\mathbf{F}$ and $\mathbf{H}$. Instead, the calculation of the sigma points and the scaled unscented transformation are at the center of this variation of the Kalman filter. All vectors, functions and matrices are the same as those in section 4.2.1. The goal is to propagate x, of dimension n_x through a function g, as shown in equation (8), outputting y, of dimension n_y.

$$\mathrm{y} = g(\mathrm{x}) \tag{8}$$

This variable x has mean $\overline{\mathrm{x}}$ and covariance P_x. To calculate the statistics of y one must create a set of $2n_\mathrm{x}+1$ sigma points, χ_i, or weighted samples, that are chosen to get the true mean and covariance of the prior x.

The variables $\lambda = \alpha^2(n_\mathrm{x}+\kappa) - n_\mathrm{x}$, κ and α are scaling parameters, while β is chosen to add knowledge of the previous x. $W_i^{(c)}$ and $W_i^{(m)}$ denote the first and second order weights of each point, respectively. Then, the propagation of the sigma points is done through equation (9).

$$\mathrm{Y}_i = g(\chi_i) \qquad\qquad i = 0, \ldots, 2n_\mathrm{x} \tag{9}$$

And the mean and covariance of y are calculated as such:

$$\overline{\mathrm{y}} = \sum_{i=0}^{2n_\mathrm{x}} W_i^{(m)} \mathrm{Y}_i \tag{10a}$$

$$\mathbf{P}_\mathrm{y} = \sum_{i=0}^{2n_\mathrm{x}} W_i^{(c)} \{\mathrm{Y}_i - \overline{y}\}\{\mathrm{Y}_i - \overline{y}\}^T \tag{10b}$$

The UKF requires an initialization of the state vector, x, the error covariance, process and measurement noises, and now, α, κ and β, which, consequently, help calculate the first and second order weights. After initializing these parameters, the next step is to calculate the sigma points followed by the state prediction update. Afterwards, we have the measurement update where the Kalman gain is calculated as well as $\mathbf{P}_{\tilde{y}\tilde{y}}$, the covariance matrix of the predicted measurement, and $\mathbf{P}_{\mathrm{x}_t\mathrm{y}_t}$, the cross-covariance matrix between the state vector and the measurement updates. Lastly, the actual update of x and $\mathbf{P}$ occurs given the measurement introduced in the filter.

As a final remark, the values for α, β and κ must be chosen in such a way that they aid the filter [22]. It is advised to have a $\kappa \geq 0$ to make sure that the covariance matrix is positive semidefinite. By default, κ is often set to zero. Finally, the paramater β should be larger than or equal to zero, and α should have a small value, typically between 0 and 1. For Gaussian distributions $\beta = 2$ is ideal.

5. Results

This section will present the results obtained by using the techniques shown in section 4. The final solution is achieved based on the most computationally efficient approaches as well as the conditions the car will ideally race in.

5.1. Line Detection

This section details the outcome of the experiments regarding the image processing stage, whose first step includes the definition of the ROI. Because we are not necessarily interested in eliminating any elements from either side, the width is kept and the length of each frame simply becomes a given percentage of the original one. The four differently sized ROI's for a specific frame from the video of the race were: 67% as proposed in [23], and then 55%, 50% and 45%.

A ROI at 67% is not ideal, since everything that is not relevant for the line detection is still in frame. On the other end of this, taking into account only 45% of the length means often not seeing the edge of the track, also making

Figure 9: ROI at 50% for a slight curve left.

Figure 10: ROI at 50% in a straight.

it invalid. The ROI at 50% is slightly better because it removes the patch of gray on the right next to the grass and part of the rail seen in the background. However, as stated before, this will always be a compromise, as evidenced by figure 9. Here, the ROI is set at 50%, but it is not enough to detect the lines that limit the track on both sides, as we can only see the red and white curb on the left. The ROI in figure 10 though, which is also at 50%, fares well, as we see more of the lines that define the limits.

The last method is the histogram back projection, applied given the selected area present in figure 5. The quantization of each channel (of HSV or BGR) divides the histogram into intervals or bins and plays an important role in the output of the back projection. Figures 11 and 12 indicate the number of bins chosen for each trial.

Figure 11: HSV bins [10, 5, 5].

Figure 12: BGR bins [3, 3, 3].

Decreasing the number of bins for each channel of BGR and HSV results in a better output, although HSV has only marginally improved with this reduction. No matter how much one tries to change the number of bins of each channel in the HSV color space, its output is always noisy, making BGR better suited for this task. It makes sense, overall, that the higher the number of bins, the worse the match between the gray rectangle that creates the histogram and the original image, as less bins envelop a higher number of "different colors", in this case, shades of gray.

After having the limits of the track highlighted in terms of color, we move on to the Canny edge detector stage, made up of the blur, Canny, dilation and erosion. For a kernel of 3x3, the output is only slightly noisy, while for a kernel of 5 there are few distinguishable features in the image, as everything else looks like a random combination of lines. This means that if it is impossible to remove any noise from the image before applying the edge detection, we will prefer a smaller sized kernel. Applying the Canny after having blurred the image is an improvement, as some of the edges detected on the road that were just noise have disappeared and the line is much clearer. The dilation and erosion serve to close the gaps left open by the Canny. This stage's influence, especially that of Canny, dilation and erosion, is better understood after applying the PHT as, only then, are we able to see how their presence, thresholds and kernel sizes affect the final line detection.

5.1.1 Probabilistic Hough Transform

The PHT is ready to be put into action and must be performed on a binary image. The BGR histogram back projection is fast and eliminates noise before any other image processing technique is applied. On the other hand, the inverse mask for red and white in HSV is also a strong contender, even though it is too slow, especially when we create two separate masks, as the lines (or curb) might be white only or red and white. Their main advantage is the fact that they are able to clearly define the edges of the track. All of the tests were carried out by trial and error, as there is no sure way to know which parameters work best beforehand and each specific technique requires its own set of values.

5.1.2 Solution

The complete image processing stage, as was implemented, will be presented here and will follow the steps shown in section 4.1. First, the ROI will be set at 50% to eliminate unwanted objects in the distance but still allow us to see the foreground which contains the race track followed by the BGR histogram back projection. As we know that further ahead, when performing the Canny edge detection, and, consequently, the PHT, the lines created by the shape of the car and its different contrasting colors will be highlighted, a trapezoid is defined, as explained in subsection 4.1.2. This trapezoid constitutes the elimination of invalid lines in two stages: first, it is painted black, so that the algorithm does not see the car itself, and then through the equations that describe the trapezoid's bounds, the lines detected here are excluded. Ideally, the camera is placed in the middle of the roll bar and not on its left side, which means that the trapezoid will cover both the car and the helmet. Obviously, in the future, the size of the area that hides the car must be adjusted to fit the new camera placement.

The histogram back projection is the next step in the process, which requires first the definition of the rectangle, 150 by 25 pixels, that selects an area of the track and does not encompass any other element present there. Figure 13 shows the outcome of the histogram back projection for the BGR color space.

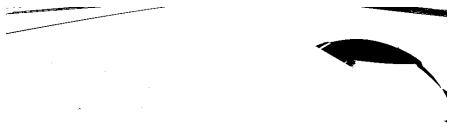


Figure 13: Histogram back projection for BGR.

It was established that the goal would be to have as few lines as possible in order to make line filtering easier. So, the Canny and the PHT parameters were chosen together and with this in mind. The blur has a kernel of 5x5, dilation and erosion of 3x3. The Canny has as a lower threshold of 200, upper threshold of 600 and kernel size of 3. Lastly, the PHT's (figure 14) parameters are $[rho,\ theta,\ threshold,\ minLineLength,\ maxLineGap] = [2,\ \pi/180,\ 40,\ 100,\ 10]$. As we can see, in the last figure, the rails in the background are detected and there are a couple of lines belonging to their edges in the outcome of the PHT.

To make sure the algorithm only sees the track limits,

Figure 14: Probabilistic Hough transform.

line filtering now takes place. Implementing this means that the output now looks like the one in figure 15 where there is only one line on the right side and one line on the left side.

Figure 15: After line filtering.

For the left and right sides of the image, a reference distance and index are created to save the smallest distance from the line to the center and its corresponding position in a vector, which holds all the objects of class "Line". A "for loop" goes through all the results of the PHT, which discards the invalid lines and puts the remaining ones through a filter function. Here, the distance to the center is evaluated and at the end of each frame there is a chosen line for each side of the image. If nothing is detected in the current frame for either side, the last detected line is repeated. Part of the reason for this last part is tied to the fact that, as the camera oscillates, sometimes a line that is still visible stops being detected as its edges become smaller or bigger, but we still want to see it. On the other hand, if, in fact, it does not exist, the algorithm will assume that there is an edge present anyway.

5.1.3 Exceptions

Sometimes the solution presented earlier did not work as expected and, ultimately, failed. Figure 16's right side line does not exactly match the line of the track. This happens throughout a lot of the frames, because the histogram back projection is noisy near the edges and when the Canny is applied this noise is considered instead, even though it is slightly off the real position of the line. Other exceptions include when both lines detected belong to the starting grid marked on the road; when there is no line on either side of the image and the last detected is replicated in the wrong place; lastly, sometimes the shade of gray of the track changes abruptly and this causes the algorithm to assume a line where there is none.

Figure 16: Wrong detection.

5.2. Variations of the Kalman Filter

This section will focus on the implementation of the Extended and Unscented Kalman filters, making use of the already existing car model and controller for GP17.Evo. The initial state estimate, for both the EKF and the UKF, x_0, is usually zero for all the state variables, but, due to the singularity for $v_x = 0$ m/s, as mentioned above in section 4.2.1, the longitudinal velocity will be the only parameter with a value other than 0. The initial values for x, $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ are given in equations (11) to (14). $\mathbf{P}$ is set at a low value for each variable because we have great confidence in the initial estimate given to the system. The process noise, $\mathbf{Q}$, is not calculated, but arrived at through trial and error. The first $\mathbf{Q}$ to be tested will be the one from [24] for v_x, v_y, $\dot{\psi}$, ψ, x and y. The remaining e_ψ, d_y and s were based on these previous values, with s set higher due to its range. The measurement noise, $\mathbf{R}$, comes from the sensors' accuracy and, because of this, will not be changed.

$$\mathrm{x}_0 = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{11}$$

$$\mathbf{P}_0 = \mathrm{diag}\,(10^{-3},\ 10^{-3},\ 10^{-3},\ 10^{-3},\ 10^{-3},\ 10^{-3},\ 10^{-3},\ 10^{-3},\ 10^{-3}) \tag{12}$$

$$\mathbf{Q}_0 = \mathrm{diag}\,(0.5^2,\ 0.5^2,\ 0.001^2,\ 0.001^2,\ 0.5^2,\ 3^2,\ 0.001^2,\ 0.5^2,\ 0.5^2) \tag{13}$$

$$\mathbf{R}_0 = \mathrm{diag}\,(0.05^2,\ 2.5^2,\ 0.0175^2,\ 2.5^2,\ 2.5^2) \tag{14}$$

The work of [20] replicates the behavior of the car for four different maximum values of v_x, which are 25, 50, 75, and 100 Km/h. The trials described in the following sections will be performed and tuned for the $v_x = 100$ Km/h case.

The metric used to compare the results obtained is the RMS, as in equation (15), where $x_{prediction}$ is the filter's estimate for each element of x and x_{true} is the state variable's true value.

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{prediction} - x_{true})^2} \tag{15}$$

Tables 2 and 3 present each trial done for the EKF and UKF, respectively. Every state variable has an RMS value and a matching value in the process noise covariance matrix $\mathbf{Q}$, both in its unit. The error covariance, $\mathbf{P}$, is already squared in equation 12 and both tables. The first trial was conducted using the vectors and matrices of equations (11) to (14), which are kept through during the following runs unless specified otherwise. Furthermore, the estimate output by the filter is in red, the true value of each given variable is in blue, and the measurements are in green, in the figures illustrating the results of the simulations in the following sections. The less relevant trials were removed.

5.2.1 Value Range

Here, we present the interval of values of each state variable in the case of $v_x = 100$ Km/h. The "Maximum" and "Minimum" rows of table 1 are the true values of the variables obtained through the simulations performed.

5.2.2 Extended Kalman Filter

Given the initial parameters defined above, the filter was run. The fact that the variables without a measurement perform worse than the rest stands out immediately and is expected exactly because of that. However, even if some deviation from the true value was anticipated, the filter must be tuned so that it does not output results that are completely unreasonable. For example, v_y goes from about 0 to about 150 m/s throughout the run. Meanwhile, when performing a complete lap, the true v_y is not only always lower than 4 m/s (in absolute value) but it varies less than the same amount. Both these observations are evidenced in table 2 through the RMS value. The most striking failures of this first trial, besides v_y, are $\dot{\psi}$, s and e_ψ. All other variables, v_x, d_y, ψ, x and y have acceptable errors.

Table 1: Maximum and minimum values for the state variables.

	v_x	v_y	$\dot{\psi}$	e_ψ	d_y	s	ψ	x	y
Maximum	28	3.5	0.72	0.49	9.7	1936	7.55	293	474
Minimum	0.1	-3.8	-0.84	-0.5	-9.72	10^{-4}	-0.02	-98	0

Table 2: EKF trials for 100 Km/h.

Trial		v_x [m/s]	v_y [m/s]	$\dot{\psi}$ [rad/s]	e_ψ [rad]	d_y [m]	s [m]	ψ [rad]	x [m]	y [m]
1	$\mathbf{P}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	$\mathbf{Q}$	0.5	0.5	0.001	0.001	0.5	3	0.001	0.5	0.5
	RMS	0.266	85.434	21.678	0.749	5.509	699.786	0.108	7.196	6.424
3	$\mathbf{P}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	$\mathbf{Q}$	0.01	10^{-4}	10^{-5}	10^{-4}	0.001	0.1	0.001	0.1	0.1
	RMS	0.618	110.65	30.833	2.266	41.602	726.71	0.343	30.251	46.970
4	$\mathbf{P}$	10^{-3}	1	1	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	$\mathbf{Q}$	0.5	0.001	10^{-5}	0.001	0.5	0.5	0.5	0.5	0.5
	RMS	0.268	113.109	33.477	2.722	2.250	1×10^3	0.023	2.004	1.912
5	$\mathbf{P}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	$\mathbf{Q}$	0.05	0.001	0.001	0.001	0.5	0.05	0.5	0.5	0.5
	RMS	0.237	89.248	27.421	1.449	2.844	670.33	0.023	1.966	2.632

Trial 3 has a process covariance matrix, $\mathbf{Q}$, with even smaller values than before across its diagonal and is also a failure. All variables performed worse than in trial 1. The $\mathbf{Q}$ chosen for trial 4 translates into a very unstable v_y and $\dot{\psi}$ at the start of the simulation. They are also further away from x_{true} and from the initial estimate given. To fix this, their value in $\mathbf{P}_0$ was increased to 1, as evidenced in table 2. This helped to fix some of the flaws in trial 1, such as d_y, ψ, x, and y. On the other hand, s now has an error of an even greater order of magnitude, 10^3, v_y and e_ψ have suffered a significant increase, and the same goes for $\dot{\psi}$. So, not only did trial 4 fail, but reducing $\mathbf{P}$ did not prove to be efficient. Finally, trial 5 is an attempt at combining all the previous trials in order to find the best solution. It is also the last try for the EKF as adjusting the values of $\mathbf{Q}$ any further leads nowhere. One always ends up having to relinquish some state variables in favor of others, since, as function f in equation (3) shows, most of them are dependent on each other. This trial has the lowest RMS for v_x, s, ψ and x and ranks second for d_y and y and third for v_y, $\dot{\psi}$ and e_ψ. Still, none of these is very far from their respective lowest RMS found in other trials. The greatest adversity encountered was maintaining a stable s, or at the very least one that did not deviate too much from the true value. The same applies to v_y, $\dot{\psi}$ and e_ψ.

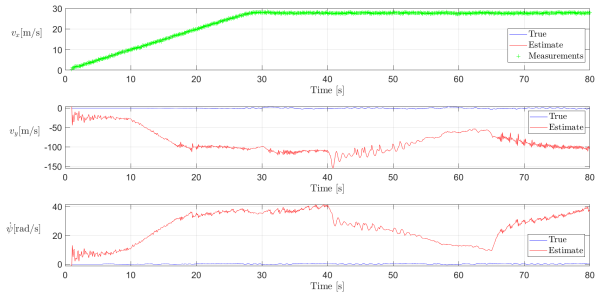
The graphs in figure 17 also give good insight to the shortcomings of the EKF mentioned earlier. It is clear at a first glance that v_y, $\dot{\psi}$, e_ψ and s did not do very well. Both v_y and $\dot{\psi}$ begin by increasing their distance from the real value, eventually getting closer to it, but quickly falling off shortly after. At first, e_ψ is close to x_{true}, but, as time passes, it begins to continuously move further away. In the same way, s becomes constant at about $t = 30$ s with the occasional increase or decrease, despite having started off well. Finally, d_y follows the blue line all the way through to the end, but is not very consistent, especially between $t =$ 10 s and $t = 40$ s and then again between $t = 65$ s and $t =$ 80 s.

5.2.3 Unscented Kalman Filter

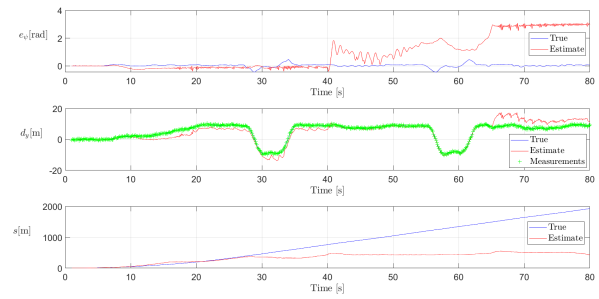
The first step of the UKF, as stated before, is also the initialization. As default values, κ will be set at 0, α at 10^{-3} and β at 2 [25]. The state vector is defined by equation (11) and the same goes for the error covariance matrix, equation (12), process noise covariance, equation (13) and measurement noise covariance, equation (14).

There is an issue with the implementation of the UKF during the computation of the sigma points. Part of this process makes use of the Cholesky decomposition which requires a positive semidefinite covariance matrix, $\mathbf{P}$. However, after the first 7 iterations of the filter, this matrix ceases to meet the requirements of the decomposition, as in, it is no longer positive semidefinite. To fix this, simple modifications, such as changing the values of κ, α and β or replacing $\mathbf{P}$ by $\mathbf{P}_{chol} = \frac{\mathbf{P}+\mathbf{P}^T}{2}$, were put in place but did not work, so the SRUKF [26] will be implemented, as it guarantees the positive semidefiniteness of $\mathbf{P}$.

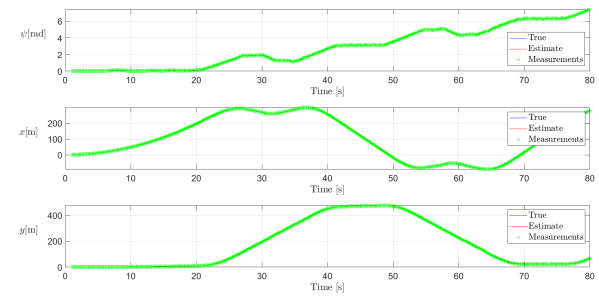
This solution, however, does not work for the vectors and matrices used in trial 1, equations (11) to (14), so v_x in matrix $\mathbf{Q}$ was changed to 0.1 m/s. Table 3 lists all of the trials conducted for the UKF. The first trial, with the exception of v_x, is exactly the same the one in table 2 and is not satisfactory. Four out of nine variables have an RMS of 10^6 or higher, which is completely unacceptable. Despite this, the remaining v_x, d_y, ψ, x and y, at least in comparison to the EKF, are decent but could be improved upon. In trial 3, unlike what happened in the EKF, lowering $\mathbf{Q}$ for the most problematic variables (v_y, $\dot{\psi}$, e_ψ and s) improves the results significantly. In any case, these four state variables, given table 1, will still introduce an error into the system that should be mitigated. Similarly to what happened in the fourth trial of the EKF, as shown in figure 18(a), both v_y and $\dot{\psi}$ are unstable in the beginning of the simulation. So, for trial 4 of the UKF their values in $\mathbf{P}_0$ were increased to 1. This made v_x marginally better, and kept ψ the same. In comparison to trial 3, while most variables experienced an increase to their RMS, it was not very significant with the exception of v_y, $\dot{\psi}$ and d_y, all of which nearly doubled.



(a) Output of variables v_x, v_y and $\dot{\psi}$.

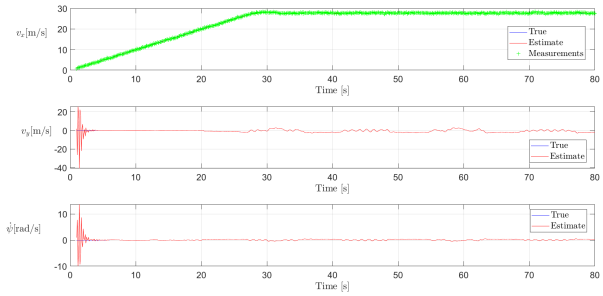


(b) Output of variables e_ψ, d_y and s.

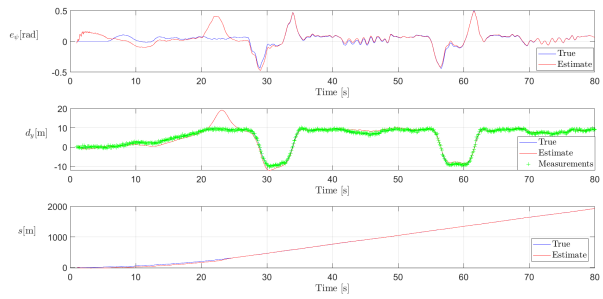


(c) Output of variables ψ, x and y.

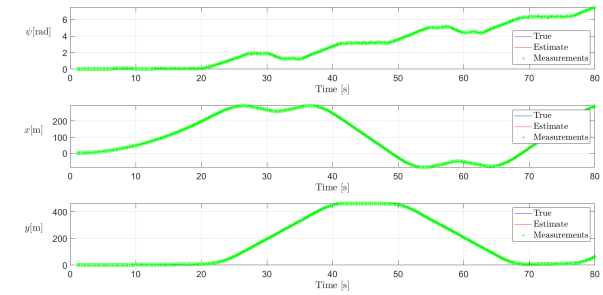
Figure 17: Trial 5 - EKF output for 100 Km/h.



(a) Output of variables v_x, v_y and $\dot{\psi}$.



(b) Output of variable3 e_ψ, d_y and s.



(c) Output of variables ψ, x and y.

Figure 18: Trial 3 - UKF output for 100 Km/h.

Other combinations not listed here were tried, but none of them succeeded in outperforming trial 3.

The UKF's weakness in trial 3 is also s, but, this time, its error is about 40 times smaller, which is a huge improvement. The estimate of s seems to follow the true value (blue line) quite faithfully most of the time. The lateral velocity also has a high error considering the fact that the highest true v_y is slightly below 3 m/s (in absolute value). On the other hand, in the beginning, v_y oscillates considerably, as does $\dot{\psi}$, contributing to their high error, but both these variables then stabilize and are close to the the blue line throughout. e_ψ is quite unstable, even having a peak right before $t = 25$ s while x_{true} remains approximately constant. It fluctuates until $t \approx 30$ s, after which it becomes more faithful to its true value. d_y is almost always slightly off, and like e_ψ deviates considerably in the interval of $20 < t < 30$ s. The last three state variables, ψ, x and y, have a small error, and are generally always close to the real value.

We should note that, despite the fact that the SRUKF was implemented, there were times when the filter failed to run because $\mathbf{P}$ stopped being positive semidefinite. As before, this happened mostly at the start after very few iterations.

5.2.4 Summary

The two filters had different performances throughout the trials, but, as expected, once both had been tuned, the UKF was more accurate when it comes to estimating the system's state. If we analyse trials 5 (Table 2 for the EKF) and 3 (Table 3 for the UKF), we can conclude that for most of the variables, in this case v_x, d_y, ψ, x and y, there is only a marginal difference between the errors of both filters. The UKF actually performs worse for v_x and x, but, again, the difference is not very significant. On the other hand, when the EKF is the one to do worse, it is usually by one or more orders of magnitude, as happens with v_y, $\dot{\psi}$, e_ψ and s. This is also visible in figures 17 and 18 when it comes to these last few that were not properly estimated, as well as in the close up of the remaining variables.

When applying the same parameters for both filters with only a limited number of measurements, the UKF is clearly the weaker algorithm, as is evident from trials 1 and 2 in tables 2 and 3. This means that tuning plays an important part in the performance of the filters and here neither of them have been tuned. On the flip side, this trial has a measurement associated with every variable and does not leave any to be estimated solely on the basis of all the others, thus decreasing the propagation of errors among them. All in all, both filters perform similarly and well.

Table 3: UKF trials for 100 Km/h.

Trial		v_x [m/s]	v_y [m/s]	$\dot{\psi}$ [rad/s]	e_ψ [rad]	d_y [m]	s [m]	ψ [rad]	x [m]	y [m]
1	$\mathbf{P}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	$\mathbf{Q}$	0.1	0.5	0.001	0.001	0.5	3	0.001	0.5	0.5
	RMS	0.321	4×10^{6}	4×10^{6}	1×10^{7}	4.074	2×10^{7}	0.023	4.959	8.103
3	$\mathbf{P}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	$\mathbf{Q}$	0.01	10^{-4}	10^{-4}	10^{-4}	0.005	0.05	0.5	0.5	0.5
	RMS	0.292	2.401	0.850	0.069	1.730	17.603	0.019	2.527	2.532
4	$\mathbf{Q}$	0.01	10^{-4}	10^{-4}	10^{-4}	0.005	0.05	0.5	0.5	0.5
	$\mathbf{P}$	10^{-3}	1	1	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
	RMS	0.287	4.984	1.873	0.081	2.123	20.554	0.019	2.693	2.548

To conclude, if we were to choose a filter to estimate the state of the system, the UKF is the logical choice. The only setback is the fact that oftentimes it fails because the error covariance matrix, $\mathbf{P}$, stops being positive semidefinite. The UKF simply has a better performance and the EKF does not output accurate enough results to relay to a control unit.

6. Conclusions

The objectives of this work defined in section 1 were the proposal of a cheap and simple system for GP17.Evo, which includes a GNSS, IMU and a camera; the creation of a line detection algorithm to pinpoint race track limits; and the implementation of two sensor fusion methods, both variations of the Kalman filter. If we take into account the results obtained, we can conclude that the objectives were achieved although all present some limitations. Despite these, this work is a starting point for PSEM to turn GP17.Evo into an autonomous car.

The work developed in this dissertation is far from over and improvements on what has been accomplished as well as new additions will be suggested in this section. The system proposed in section 3 is relatively simple, as was the goal, but, should the opportunity arise, more sensors could be added to the original setup. When it comes to line detection, detecting curved lines with accuracy is not implemented yet. This could be done by applying RANSAC [10] or creating a line model [13] to follow curved lane limits. Sensor fusion can also be improved upon, namely, in the tuning of the parameters $\mathbf{P}$ and $\mathbf{Q}$ of the EKF and UKF through the analysis of the model of the car. The introduction of more or better sensors will also influence the performance of the filters, through the matrix $\mathbf{R}$, making it more accurate and reliable in the event of a sensor failure.

In the future, the goal is to actually have an autonomous GP17.Evo by combining the sensor system with the control and planning unit and have the car race in real conditions.

References

[1] Miguel Valls et al. Design of an autonomous racecar: Perception, state estimation and system integration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2048–2055, 2018.

[2] Stefano Arrigoni et al. Design of a prototypical platform for autonomous and connected vehicles. In *2021 AEIT International Conference on Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pages 1–6, 2021.

[3] Juraj Kabzan et al. AMZ Driverless: The Full Autonomous Racing System. *J. Field Robotics*, 37:1267–1294, 2020.

[4] Francois Caron et al. GPS/IMU data fusion using multisensor Kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 2006.

[5] Tianxing Chu et al. Monocular camera/imu/gnss integration for ground vehicle navigation in challenging gnss environments. *Sensors*, 12:3162–85, 12 2012.

[6] Simone Mentasti. *Embedded AI and Sensor Fusion for Autonomous Vehicles*. PhD thesis, Politecnico di Milano, 2021.

[7] R.C. Luo et al. Multisensor fusion and integration: approaches, applications, and future research directions. *IEEE Sensors Journal*, 2(2):107–119, 2002.

[8] Mingyang Li et al. online temporal calibration for camera-imu systems: Theory and algorithms. *The International Journal of Robotics Research*, pages 947–964, 05 2014.

[9] M. Bertozzi et al. GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81, 1998.

[10] M. Aly. Real time detection of lane markers in urban streets. In *2008 IEEE Intelligent Vehicles Symposium*, pages 7–12, 2008.

[11] Zhenqiang Ying et al. Robust lane marking detection using boundary-based inverse perspective mapping. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1921–1925, 2016.

[12] Shengyan Zhou et al. A novel lane detection based on geometrical model and Gabor filter. In *2010 IEEE Intelligent Vehicles Symposium*, pages 59–64, 2010.

[13] Cláudio Rosito Jung et al. Lane following and lane departure using a linear-parabolic model. *Image and Vision Computing*, 23(13):1192–1202, 2005.

[14] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), 1986.

[15] Abdulhakam.AM. Assidiq et al. Real time lane detection for autonomous vehicles. In *2008 International Conference on Computer and Communication Engineering*, pages 82–88, 2008.

[16] D. C. Andrade et al. A Novel Strategy for Road Lane Detection and Tracking Based on a Vehicle's Forward Monocular Camera. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1497–1507, 2019.

[17] Kuo-Yu Chiu et al. Lane Detection using Color-Based Segmentation. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pages 706–711, 2005.

[18] J. Matas et al. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78(1):119–137, 2000.

[19] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. *Proc. Siggraph Course*, 8, 01 2006.

[20] Miguel D'Ajuda. Trajectory Planning, Guidance and Control of a Race Car. Master's thesis, Instituto Superior Técnico, 2022.

[21] M. Bersani et al. Vehicle state estimation based on kalman filters. In *2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pages 1–6, 2019.

[22] Rudolph Merwe et al. The Unscented Particle Filter. *NIPS*, 13, 01 2001.

[23] Jingwei Cao et al. Lane Detection Algorithm for Intelligent Vehicles in Complex Road Conditions and Dynamic Environments. *Sensors*, 19(14), 2019.

[24] L. Zhao et al. An Extended Kalman Filter Algorithm for Integrating GPS and Low Cost Dead Reckoning System Data for Vehicle Performance and Emissions Monitoring. *Journal of Navigation*, 56(2):257–275, 2003.

[25] E.A. Wan et al. The Unscented Kalman Filter for Nonlinear Estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000.

[26] R. Van der Merwe et al. The Square-Root Unscented Kalman Filter for state and parameter-estimation. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 6, pages 3461–3464 vol.6, 2001.