3.
Error Generation Procedures

Most controllers are designed to follow a reference signal or to minimize the error between the desired value of a variable and the one obtained by some measuring method. Considering that this research is entirely based on simulation results, it is necessary to define how to model the sensor system perception of the current position related to the reference trajectory.

The chosen control reference is the vehicle's desired path, which is stored as detailed in [15]. Two different error definitions are introduced in this section: the present-based trajectory error and the future-based trajectory error. Both are detailed and tested.

3.1. Track Construction Model

3.1.1. Presentation and Description

The defined track model [15] is stored as a matrix, represented in Equation (3.1) and describing the track as a combination of straight lines and circle arcs. The columns store information about corresponding road parts. Null values in the second line of column i means that the current stretch is a straight line. The first line of that column is the stretch's length, given by the variable $L(i)$. On the other hand, for a curve stretch represented in column j, the first line shows the curve's radius and the second line defines its angle, $-R(j)$ and $\alpha(j)$ respectively.

$$Track = \begin{bmatrix} L(1) & R(2) & \cdots & R(n-1) & L(n) \\ 0 & \alpha(2) & \cdots & \alpha(n-1) & 0 \end{bmatrix} \tag{3.1}$$

Each column information of the track matrix can be used to obtain the positions and orientations of the starting point for each track stretch i, named

$(x_T(i),y_T(i))$ and $\beta(i)$ respectively. Therefore, $(x_T(1),y_T(1))$ are the coordinates of the first track stretch and $\beta(1)$ is its angle with the horizontal.

In order to obtain the coordinates of the subsequent track stretches, the track construction algorithm developed uses simple geometric relations, as shown in Figure 3.1. Equations (3.2) and (3.3) detail these calculations, for straight lines and curves respectively.

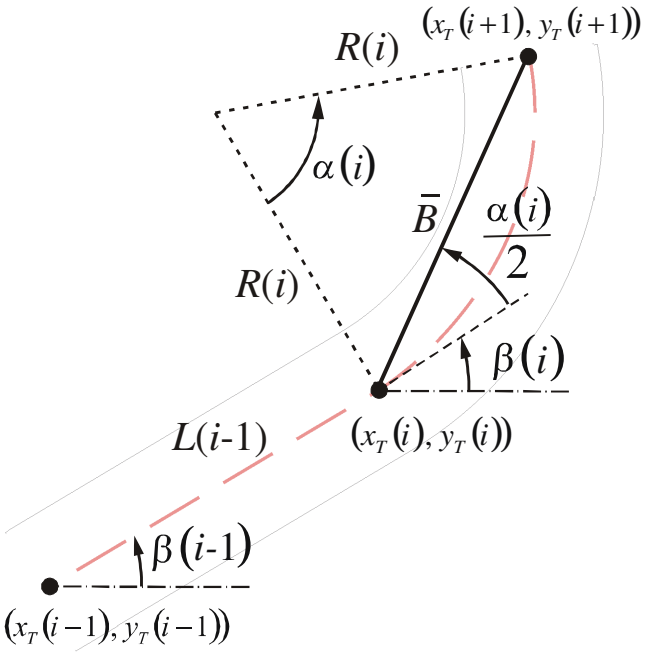

Figure 3.1 - Track stretch Coordinate Calculus.

$$\begin{cases} x_T(i) = x_T(i-1) + L(i-1)\cdot\cos(\beta(i-1)) \\ y_T(i) = y_T(i-1) + L(i-1)\cdot\sin(\beta(i-1)) \\ \beta(i) = \beta(i-1) \end{cases} \quad ; for\ \ i = \{2,3,...,n\} \tag{3.2}$$

$$\begin{cases} \bar{B} = \sqrt{2\cdot R(i-1)^2 - 2\cdot|R(i-1)|\cdot\cos(|\alpha(i-1)|)} \\ x_T(i) = x_T(i-1) + \bar{B}\cdot\cos\left(\beta(i-1) + \dfrac{\alpha(i-1)}{2}\right) \\ y_T(i) = y_T(i-1) + \bar{B}\cdot\sin\left(\beta(i-1) + \dfrac{\alpha(i-1)}{2}\right) \\ \beta(i) = \beta(i-1) + \alpha(i-1) \end{cases} \quad ; for\ \ i = \{2,3,...,n\} \tag{3.3}$$

For all vehicle models, the defined track is completely planar, allowing a 2–dimentional representation from center line trajectory. However, is assumed a simplification hypothesis that the whole track has a single lane width: l_w.

3.1.2. Validation Tests

The validation test for the track model is simple and should enable the construction of a graphical representation of any track directly from its data matrix. For example, in Table 3.1 all the information of a sample track is detailed. The curve lengths, $L(2)$ and $L(4)$, are calculated by multiplying the curve angle in radians and the curve radius in meters.

Table 3.1 – Detailed Track Information.

Track Part	Type	R [m]	α [°]	L [m]	l_w [m]
1	Straigth Line	-	-	100	8
2	Right Curve	20	180	62.8	8
3	Straigth Line	-	-	100	8
4	Right Curve	20	180	62.8	8

The respective data matrix is defined in Equation (3.4). Note that the signs in $R(2)$, $R(4)$, $\alpha(2)$ and $\alpha(4)$ indicate the side of the curve. According to the alignment of the positive X axis with the car front, a left curve is positive and a right curve is negative. Figure 3.2 shows the representation for that matrix and the gray arrow indicates the track starting point and orientation.

$$Track = \begin{bmatrix} 100 & -20 & 100 & -20 \\ 0 & -\pi & 0 & -\pi \end{bmatrix} \tag{3.4}$$

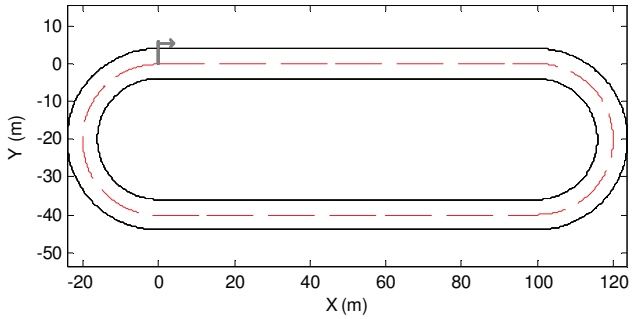

Figure 3.2 – Track Model Validation Test: Oval Circuit.

Although the center line information can be written as a combination of straight lines and arcs of circle, the trajectory itself is treated as a sequence of

points where the car center of mass should move along. The car's path is obtained by iterations of the vehicle model directly commanded or as a result of the optimization process.

In the error generation procedure the reference signal is the global coordinate of a desired path along time. This implies in keeping much information, but allows the direct connection between the optimization and controller simulation blocks, increasing modularity.

3.2. Traveled Track Distance Correction

3.2.1. Presentation and Description

An important calculation is the travelled track distance correction. This distance represents the center line projection of any point on the track. It can be better visualized in Figure 3.3.

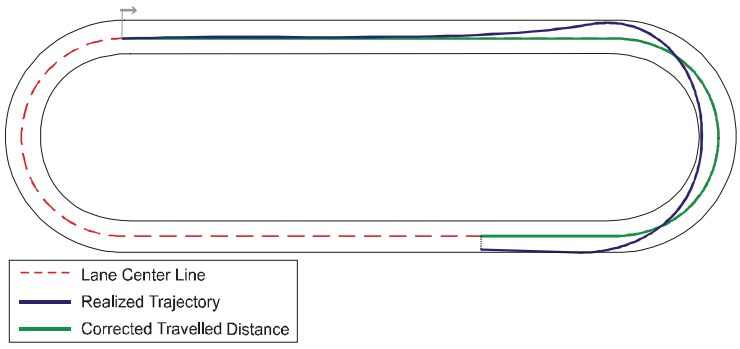

Figure 3.3 – Graphic Representation of the Traveled Track Distance Correction.

In order to adopt a pattern in the track representation, all figures respect the same color scheme shown in the legend of Figure 3.3. Human drivers do not use the elapsed time to locate themselves on the path. Instead, they use visual information to identify, for example, that they are at the end of a curve or getting close to a straight line.

Initially, the current car position must be related to a specific track stretch to enable its center line recognition. Therefore, to identify on which track stretch the car position should be projected, the track side limits must be calculated. Maintaining the same discretization of the coordinates of the track stretch position, $(x_T(i),y_T(i))$, and orientation, $\beta(i)$, the track limits for the left and right sides of the road are shown in Equations (3.5) and (3.6), respectively.

$$\lim_{Left}(i)=\begin{bmatrix} x_T(i)+\left(\frac{l_w}{2}\right)\cdot\cos\beta(i) & x_T(i+1)+\left(\frac{l_w}{2}\right)\cdot\cos\beta(i+1) \\ y_T(i)+\left(\frac{l_w}{2}\right)\cdot\sin\beta(i) & y_T(i+1)+\left(\frac{l_w}{2}\right)\cdot\sin\beta(i+1) \end{bmatrix} \tag{3.5}$$

$$\lim_{Right}(i)=\begin{bmatrix} x_T(i)-\left(\frac{l_w}{2}\right)\cdot\cos\beta(i) & x_T(i+1)-\left(\frac{l_w}{2}\right)\cdot\cos\beta(i+1) \\ y_T(i)-\left(\frac{l_w}{2}\right)\cdot\sin\beta(i) & y_T(i+1)-\left(\frac{l_w}{2}\right)\cdot\sin\beta(i+1) \end{bmatrix} \tag{3.6}$$

Once the (x,y) point is located inside the track it can be projected, generating the center line corresponding position of the car, (x_C,y_C). This projection, which consists, for the straight parts of the track, of finding the interception point between lines r and p, is graphically represented in Figure 3.4. Notice that r is the track stretch center line and p is a line perpendicular to r that contains the point (x,y).

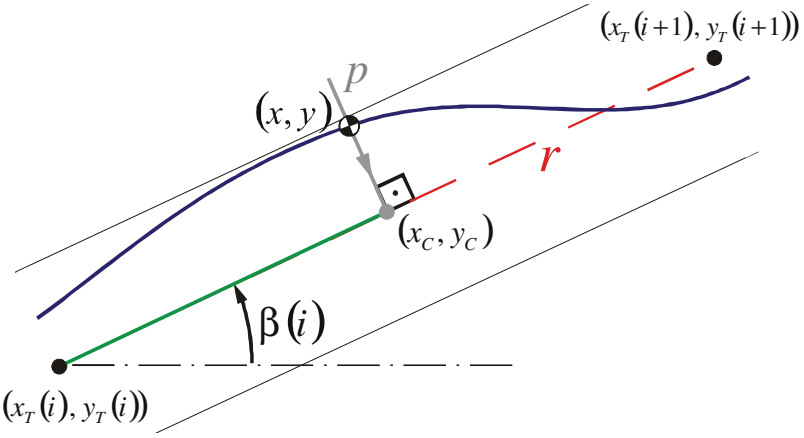

Figure 3.4 – Straight Line's Projection Procedure.

The implemented algorithm begins by identifying on which track stretch the car (x,y) is located. Since the car always starts from the first stretch, this search consists of analyzing the current vehicle position; if it is outside the limits of stretch i, the stretch counter is incremented.

If the identified track stretch is a straight line, the angular and linear coefficients of the center line equation are calculated. The variables r_1 and r_2, are given below, where the line r equation is defined by $y_r = r_1.x_r + r_2$:

$$r_1 = \tan\left(\arctan\left(\frac{x_T(i+1) - x_T(i)}{y_T(i+1) - y_T(i)} \right) + \frac{\pi}{2} \right)$$

$$r_2 = y - r_1 \cdot x$$

The same calculation is repeated for the line p equation, shown in gray in Figure 3.4. This line must be perpendicular to the r line and the car position (x,y) must be a point of p. The line p equation is $y_p = p_1.x_p + p_2$, and the coefficients p_1 and p_2 are determined by:

$$p_1 = \tan\left(\arctan(r_1) + \frac{\pi}{2} \right)$$

$$p_2 = y - p_1 \cdot x$$

The projected point that corresponds to the vehicle position is (x_C,y_C), calculated from the interception between lines r and p. The length of each track stretch j, $L(j)$ has already been calculated analytically. The corrected Travelled Track Distance of (x,y), $d(x,y)$ is then obtained through Equation (3.7).

$$x_C = \frac{p_2 - r_2}{r_1 - p_1}$$

$$y_C = p_1 \cdot x_C + p_2$$

$$d(x,y) = \sqrt{(x_C - x_T(i))^2 + (y_C - y_T(i))^2} + \sum_{j=1}^{i-1} L(j) \qquad (3.7)$$

If the track stretch is a curve, the center line perpendicular projection must be calculated in another way. Instead, the radius line that crosses the car position (x,y) should be used, maintaining the orthogonal characteristic of this operation.

Considering that all curves are arcs of circle, polar coordinates are used in the projection calculation. Some auxiliary variables, especially angles, are necessary and defined below.

The angle between the global X-axis and the first point radial line is Δ. That angle is easily obtained from the orientation of the track stretch first point, $\beta(i)$, and the sign of the curve angle, $\alpha(i)$:

$$\Delta = \beta(i) - \text{sign}(\alpha(i)) \cdot \frac{\pi}{2}$$

The point (Cx,Cy) is the center of curvature of the track stretch i and is defined by using the inverse polar transform. Hence, using the curve radius, $R(i)$, the arc angle Δ, and the coordinates of the previous stretch starting point's ($x_T(i-1)$,$y_T(i-1)$), the transform can be written as:

$$\begin{bmatrix} C_x \\ C_y \end{bmatrix} = \begin{bmatrix} |R(i)| \cdot \cos\Delta \\ |R(i)| \cdot \sin\Delta \end{bmatrix} + \begin{bmatrix} x_T(i-1) \\ y_T(i-1) \end{bmatrix}$$

The orientation of the desired radius line that crosses the car position, γ, is obtained from the vertical and horizontal variation between the center of curvature and the vehicle position, ∂_x and ∂_y. Finally, the corrected traveled track distance, $d(x,y)$, is obtained as shown in Equation (3.8). The graphical representation of the curve path algorithm can be seen in Figure 3.5.

$$\partial_x = x - C_x$$
$$\partial_y = y - C_y$$
$$\gamma = \pi - |\Delta| + \text{sign}(\alpha(i)) \cdot \arctan\left(\frac{\partial_y}{\partial_x}\right)$$

$$d(x,y) = \gamma \cdot R(i) + \sum_{j=1}^{i-1} L(j) \tag{3.8}$$

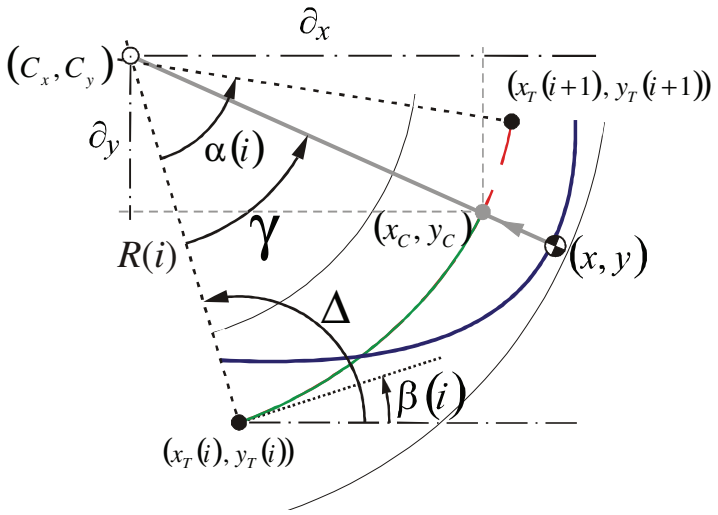

Figure 3.5 – Curve's Projection Procedure.

3.2.2. Validation Tests

To validate the procedure, the track defined in Table 3.1 was implemented in the Simulink® environment. A test trajectory, tangent to the second curve of this track, is defined manually. This trajectory does not coincide with the lane center line and, in order to generate the acceleration profile input, the correct travelled track distance should be calculated.

In Figure 3.6, the lane center line, the trajectory performed by the vehicle and the corrected center line projection are plotted together. The track travelled distance of the last point indicated by a black "x" in the same figure is 196.20 m. Comparing it to the analytical distances showed in Table 3.1 and considering that the car is approximately in the first third of the second straight line, the corrected distance should be: $d = 100 + 20\pi + 100/3 \approx 196.16$.

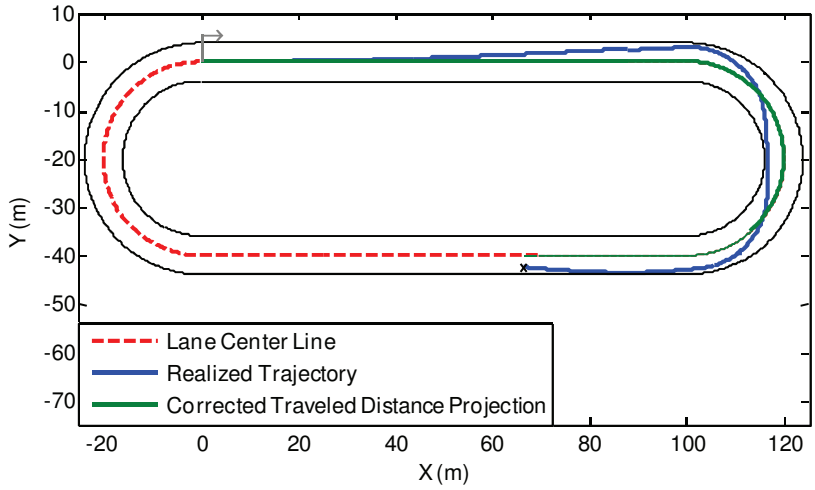

Figure 3.6 – Traveled Distance Correction Validation Test: Projection Procedure.

3.3. Present-based Trajectory Error Definition

3.3.1. Presentation and Description

As mentioned before, two different error generation methods were developed with the purpose of understanding and modeling how humans identify a deviation from a desired path. The first one, detailed in this section, consists of comparing the instant car position and orientation to desired values of those variables on the closest point in the reference trajectory.

Human are able to perform this task automatically. However, in the case of a computational search for the trajectory's closest point, much information must be stored and computed. As the car initial position is always the first point, this is surely the closest point to the reference, therefore creating the base for the recursive calculation.

The index of the closest point is i_{cp}, which is represented in Figure 3.7. A region around this point is also stored in order to minimize the search domain in the next iteration. The vehicle's reference system (X_C,Y_C) appears in gray and it is placed on the vehicle center of mass. The track reference system is placed on the closest point and is represented by the axes in black (X_P,Y_P). The car's yaw angle is θ, and the angle between the trajectory tangents at the i_{cp} point and the global X-axis is λ.

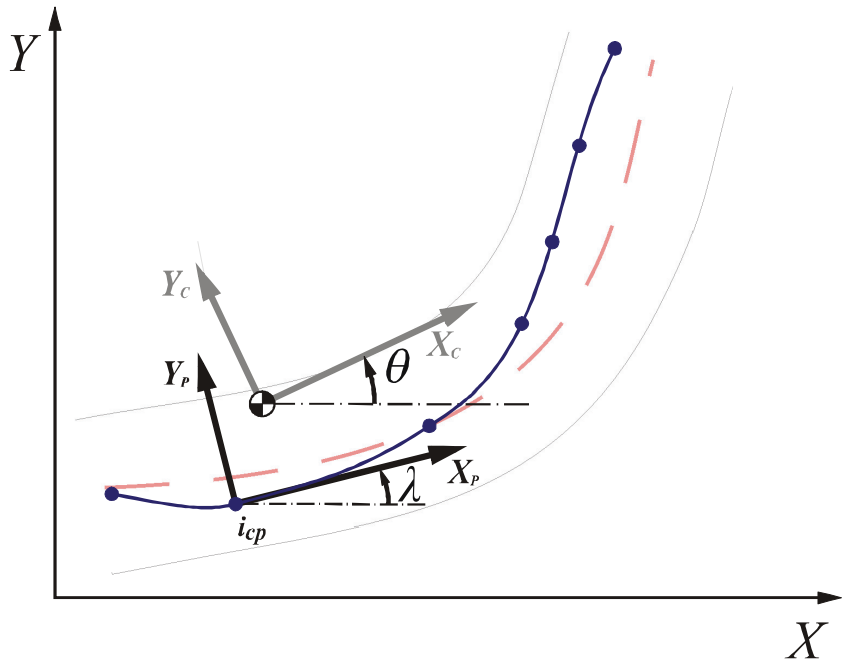

Figure 3.7 – Present-based Trajectory Error: Graphic Representation.

The closest point index calculation is repeated throughout the iterations; before updating i_{cp}, the previous iteration index value is stored in the variable i_{last}. The region is stored as the array of the points indexes, a_{reg}, which generally contains the closest point index itself and some points after and before.

The ideal gaps are symmetrical around the closest point and may be changed manually by the variable n_{gap}. Its determination must consider the vehicle speed and simulation time. The algorithm must treat some situations where the number of points after and before the closest point is not the same. The first contour condition is when i_{last} is equal or smaller than n_{gap}, and the region is defined as in Equation (3.9).

$$a_{reg} = \{ 1 \ , \ 2 \ , \ \dots \ , \ i_{last} \ , \ \dots \ , \ i_{last} + n_{gap} - 1 \ , \ i_{last} + n_{gap} \} \tag{3.9}$$

Another special situation is when the vehicle approaches the end of the trajectory. If i_{last} plus n_{gap} are equal or higher than the trajectory array length l_{traj}, the region array is written as in Equation (3.10).

$$a_{reg} = \{ i_{last} - n_{gap} \ , \ i_{last} - n_{gap} + 1 \ , \ \dots \ , \ i_{last} \ , \ \dots \ , \ l_{traj} - 1 \ , \ l_{traj} \} \tag{3.10}$$

It is generally possible to map a complete search region with the number of elements, determined by $2.n_{gap}+1$. The region of possible closest points is then updated recursively as shown in Equation (2.3).

$$a_{reg} = \{ i_{last} - n_{gap} \ , \ \dots \ , \ i_{last} \ , \ \dots \ , \ i_{last} + n_{gap} \ \} \tag{3.11}$$

The closest point is the point in the region array with minimum Euclidean distance to the car's center of mass, (x,y). Equations (3.12) show the implementation of this calculation and also the i_{cp} determination.

$$\begin{cases} d_{reg}(i) = \sqrt{\left(x_{traj}(i) - x\right)^2 + \left(y_{traj}(i) - y\right)^2} \Leftrightarrow i \in a_{reg} \\ i_{cp} = \min\left(d_{reg}\right) \end{cases} \tag{3.12}$$

Calculating the car's position and orientation regarding the trajectory is the last step for obtaining the error components. The strict changes from global coordinates (X,Y) to local reference coordinates, (X_P,Y_P) and (X_C,Y_C), are shown in Equations (3.13) and (3.14).

$$ {}_G R_P = \begin{bmatrix} \cos\lambda & \sin\lambda \\ -\sin\lambda & \cos\lambda \end{bmatrix} \tag{3.13} $$

$$ {}_G R_C = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{3.14} $$

By multiplying (x,y) by the rotation matrix ${}_G R_P$ the car position can be referred to the local system. Hence, as shown in Equations (3.15), the position error, E_P, is equal to the y_P component with the opposite sign. This sign change intends to automatically indicate if the car is too much to the left or to the right side.

$$ \begin{cases} \begin{bmatrix} x_P \\ y_P \end{bmatrix} = {}_G R_P \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_{traj}(i_{cp}) \\ y_{traj}(i_{cp}) \end{bmatrix} \\ E_P = y_P \end{cases} \tag{3.15} $$

Considering that the treated tracks can be closed circuits, λ and θ may assume values in other quadrants as well. The matrices described in Equations (3.13) and (3.14) can be used to express the rotation between the track's and vehicle's local reference systems. The matrix manipulation is:

$$ {}_G R_C = {}_G R_P \cdot {}_P R_C $$

$$ \left({}_G R_P^{-1}\right) {}_G R_C = \left({}_G R_P^{-1}\right) {}_G R_P \cdot {}_P R_C \rightarrow {}_G R_P^{-1} \cdot {}_G R_C = I \cdot {}_P R_C $$

$$ {}_P R_C = {}_G R_P^{-1} \cdot {}_G R_C \quad \therefore \quad {}_G R_C \text{ is Orthogonal} \Leftrightarrow {}_G R_P^{-1} = {}_G R_P^{T} $$

Therefore, the orientation error, E_a, can be obtained from the rotational matrix ${}_P R_C$. As seen in Equation (3.16), all the matrix positions are known trigonometric functions of E_a.

$$ {}_{P}R_{C} = {}_{G}R_{P}^{\ T} \cdot {}_{G}R_{C} = \begin{bmatrix} \cos E_a & \sin E_a \\ -\sin E_a & \cos E_a \end{bmatrix} \tag{3.16} $$

Both error components calculated here were used by the controller developed in Chapter 5. They are sufficient to guarantee that any controller has the information not only to follow the track center, but also to perform it with the correct orientation.

3.3.2. Validation Tests

The error generation procedure is directly applied to a predefined trajectory and to the car's condition. Tests shown here intend to confirm the expected error behavior in specific conditions, and are not related to any real application or vehicle model.

The primary and most important test consists of evaluating the model response. The algorithm is submitted to known inputs and then it is checked whether the error calculated components represent those inputs properly. The Simulink® Block Diagram used in this test is shown in Figure 2.13. The car global displacement is represented by x, y and θ.

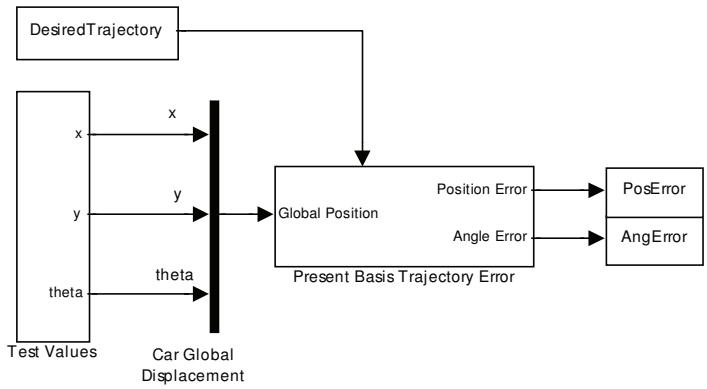

Figure 3.8 – Present-based Trajectory Error Validation 1st Test: Simulink Block Diagram.

The car displacement for this test is defined as a straight line with constant speed. In the block diagram, x is a ramp function, and y and θ are constants defined here as 0. This and the desired trajectorycan be seen in Figure 3.9. The output error components are shown in Figure 3.10.

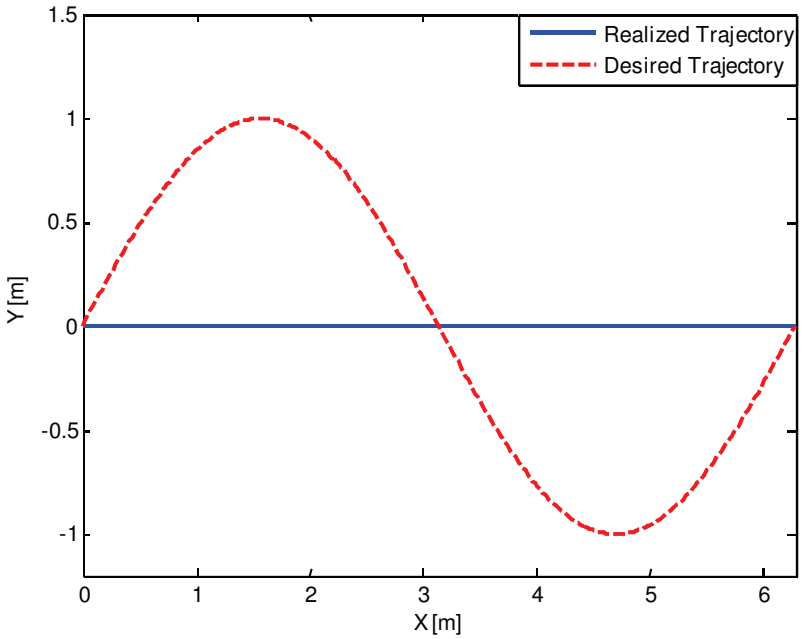

Figure 3.9 – Present-based Trajectory Error Validation 1st Test: Inputs.

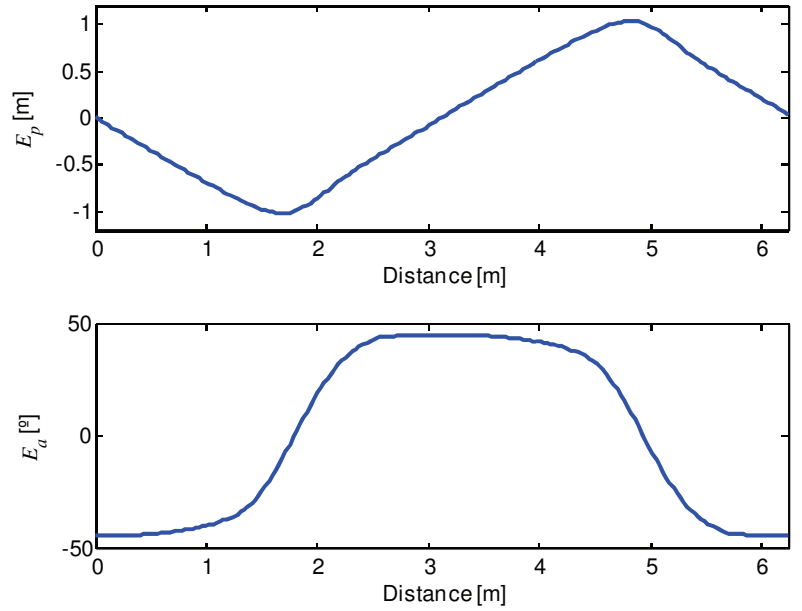

Figure 3.10 – Present-based Trajectory Error Validation 1st Test: Outputs.

It can be seen in Figure 3.10 that the test response is coherent with the given inputs. Notice that the position error, E_p, does not repeat a sinusoidal profile. The orientation error, E_a, varies from –45º to 45º in accordance with the same sinusoidal inclination angles.

An important point is to analyze the responses when the trajectory is a closed circuit. As the error reference system follows the trajectory, it is interesting

to see how a full turn affects the error values. Another relevant matter is whether the error keeps close to zero when the car manages to chase the trajectory.

This final experiment will test both situations simultaneously. The Simulink® Block Diagram is modified so that it submits the error algorithm to a circular trajectory and a similar car displacement.

The defined inputs can be seen in Figure 3.11. Visually, they are exactly the same. However, the circle equation is graphically represented in a numeric environment by a polygon with a finite number of faces. Therefore, as these two data sets were created separately, the polygons are not identical.

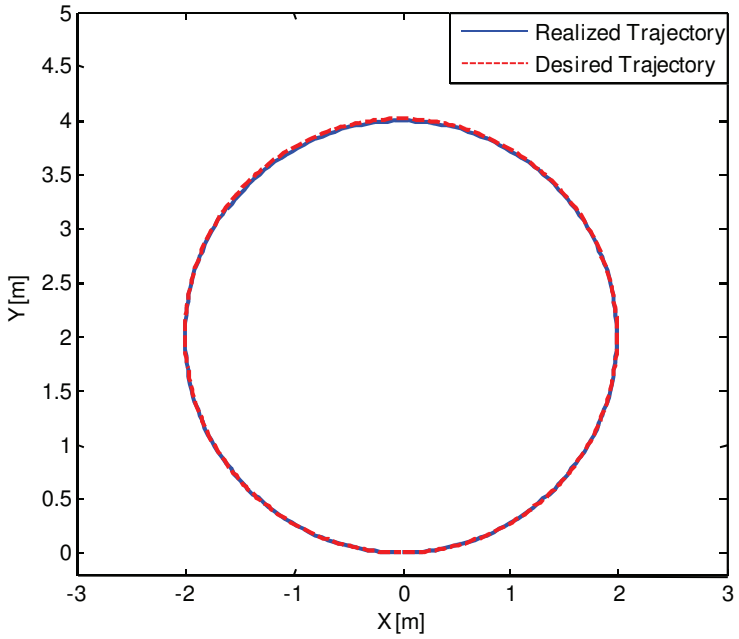

Figure 3.11 – Present-based Trajectory Error Validation 2nd Test: Inputs.

As shown in Figure 3.12, the position error, E_p, does not present significant values throughout the trajectory. Nevertheless, the orientation error E_a, is more susceptible to it.

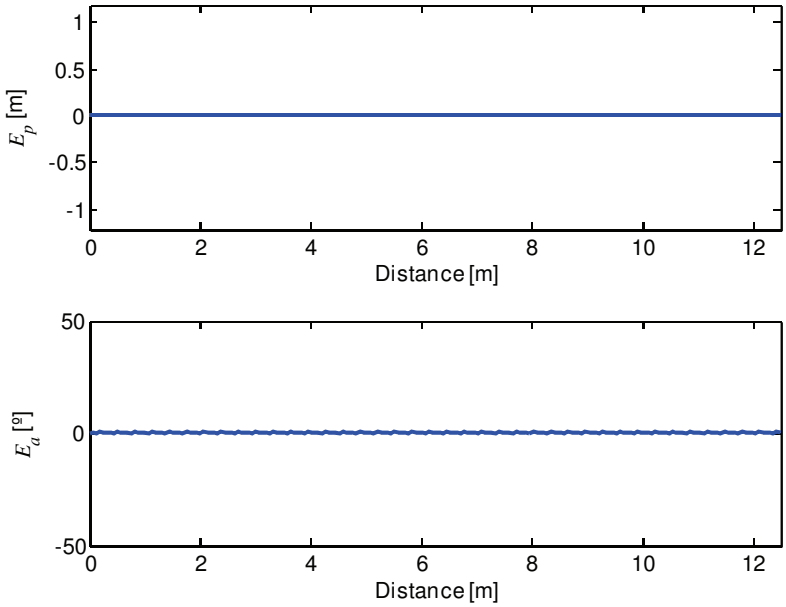

Figure 3.12 – Present-based Trajectory Error Validation Second Test: Outputs.

Despite the small numeric disturbance, this test confirms the expectations. It shows that when the car follows a trajectory, the error signal remains close to zero. In addition, it can be observed that the error generation procedure is suited to closed circuits.

3.4. Future-based Trajectory Error Definition

3.4.1. Presentation and Description

The error analysis presented above represents the perception of a driver in a car "without windows and with a small hole on the floor". That driver is able to notice deviations just after they occur, and only then react to them.

Differently from the present-based trajectory error, the future-based trajectory error considers road information ahead, which is more representative of the way human beings drive.

As the previous method, the car initial position is always the first point of the trajectory. A region of possible closest points is also kept stored to minimize the search domain, and two local reference systems are again placed on the car's center of mass and on the closest point.

Instead of getting error information only from i_{cp}, a number of forward steps, n_{FS}, are also analyzed. Those forward positions may be collected sequentially or by ignoring some intermediate data. The step size is defined by variable δ_i. A schematic procedure is illustrated in Figure 3.13.

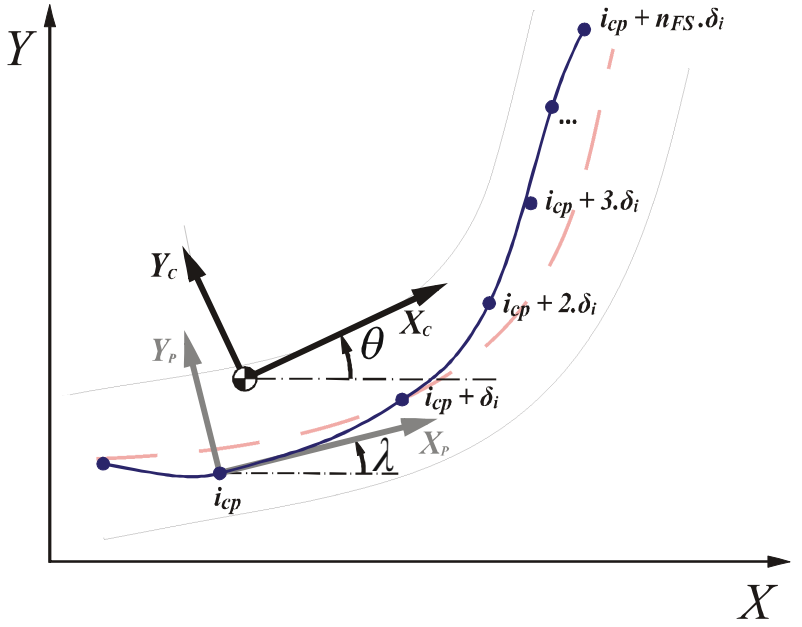

Figure 3.13 – Future-based Trajectory Error: Graphic Representation.

Once more, the closest point index calculations are repeated throughout the iterations and the previous iteration index value is stored in i_{last}. The a_{reg} array is determined by the algorithm described in Equations (3.9), (3.10) and (2.3); the closest point, i_{cp}, is the same as shown in Equation (3.12).

The new algorithm begins by looking at the present trajectory information regarding the vehicle. In the present-based trajectory error, the car position, (x,y), considered the track's reference system. Here the rotation matrix ${}_GR_C$, detailed in Equation (3.14), is used for expressing the track points with respect to the car's local reference system. From i_{cp} and for all the desired forward steps, that calculation is detailed in Equation (3.17).

$$\begin{bmatrix} x_C(j) \\ y_C(j) \end{bmatrix} = {}_GR_C \cdot \begin{bmatrix} x_{traj}(i) \\ y_{traj}(i) \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{cases} k = \{0, 1, 2, \ldots, n_{FS}\} \\ i = i_{cp} + k \cdot \delta_i \\ j = k + 1 \end{cases} \tag{3.17}$$

The main idea is to consider the influence of all points in the errorcalculation. A natural method is to compute the mean value among the y_C components of all track information obtained above. However, in order to add more flexibility to the model, the array w_{FS} is defined to calculate the position error as a weighted average, as shown in (3.18).

$$E_p = \frac{1}{n_{FS}+1}\sum_{j=1}^{n_{FS}+1}\left(-y_C(j)\cdot w_{FS}(j)\right) \tag{3.18}$$

It is very important to keep the convention that the positive error is related to the car's left. Therefore, y_C is written negatively, because in this case the error reference system is placed in the car.

The orientation error generation procedure implemented here also considers the future information. It starts with the calculation of the ${}_CR_{P,}\ {}_CR_P$ matrix for each point *j* ahead. To calculate the contributions of each future point to the orientation error, $E_a(j)$, it is necessary to invert the trigonometric functions in each rotation matrix. Those operations are detailed below and the error definition appears in Equation (3.19).

$$\begin{aligned}
&{}_GR_P\Big|_j = {}_GR_C \cdot {}_CR_P\Big|_j \\
&\left({}_GR_C^{-1}\right)\cdot {}_CR_P\Big|_j = \left({}_GR_C^{-1}\right)\cdot {}_GR_C \cdot {}_CR_P\Big|_j \\
&{}_GR_C^{-1}\cdot {}_GR_P\Big|_j = I\cdot {}_PR_C \quad \therefore \quad {}_GR_C \text{ is orthogonal} \Leftrightarrow {}_GR_P^{-1} = {}_GR_P^{T} \\
&{}_CR_P\Big|_j = {}_GR_C^{T}\cdot {}_GR_P\Big|_j = \begin{bmatrix} \cos E_a(j) & \sin E_a(j) \\ -\sin E_a(j) & \cos E_a(j) \end{bmatrix}
\end{aligned}$$

Erro! Não é possível criar objetos a partir de códigos de campo de edição. (3.19)

3.4.2. Validation Tests

Tests shown here have the same structure as those in 3.3.2. One evaluates if the responses are conceptually correct, and the other checks its behavior in critical situations.

The Simulink® Block Diagram used in this validation analysis and in all following tests is shown in Figure 3.14. Here the future-based trajectory error is the tested block.

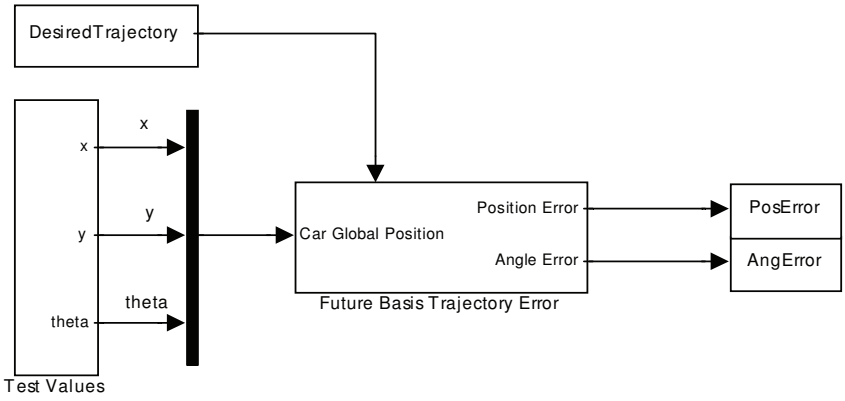

Figure 3.14 – Future-based Trajectory Error Validation Tests: Simulink Block Diagram.

Again the car displacement is established as a straight line with constant speed, x is a ramp function, y and θ are constants set to 0. The desired trajectory is plotted together with the car displacement in Figure 3.15.

As seen in the previous section, the future-based trajectory error algorithm allows variations in the number of considered forward steps, n_{FS}, and also in the sample interval, δ_i. Different values for n_{FS} and δ_i were tested for the same inputs used above; results can be seen in Figure 3.16.

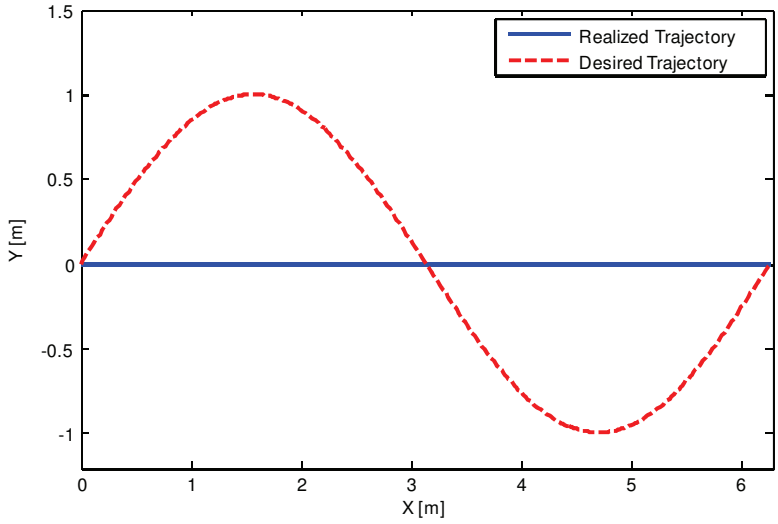

Figure 3.15 – Future-based Trajectory Error Validation 1st Test: Inputs.

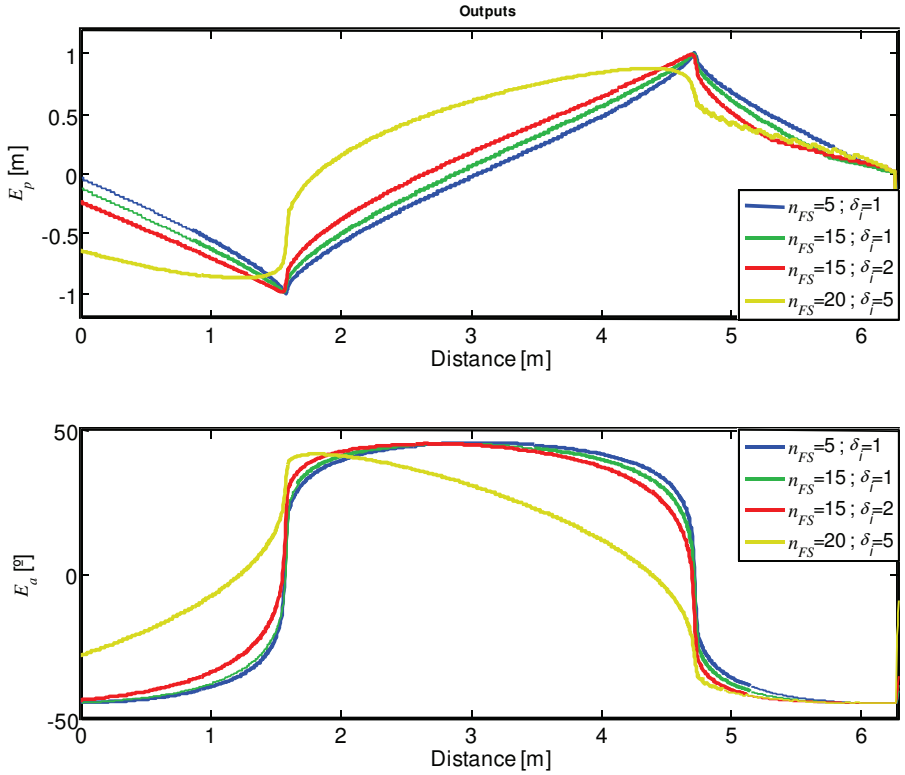

Figure 3.16 – Future-based Trajectory Error Validation 1st Test: Outputs.

The first case responses were very similar to those obtained through the present-based trajectory error procedure. Knowing that, in simulation, the intervals are of 0.001s, n_{FS} = 5 and δ_i = 1, the error procedure considers information from few centimeters ahead only.

In the next case, with n_{FS} = 15 and δ_i = 1, the future information caused some changes to E_p and E_a graphic shapes. However, as the step size between the evaluated points was still one, the future information does not compensate the computational effort in increasing the number of forward steps.

Relevant results were obtained in the next two cases. With the same n_{FS} but with a step size equal to two, fifteen points were evaluated; the thirtieth step ahead was considered in error calculations. The graphs presented in Figure 3.16 clearly show how the error signals were influenced by future information.

E_p now grows faster while the trajectory is diverging from the car; when the future distance is decreasing, the position error follows that path asymptotically. E_a also presents this behavior of anticipating and smoothing the sinusoidal transitions. It is important to remember that this response could

change drastically with the array's weight w_{FS} (defined uniformly here in order to give the same importance to all points evaluated by error procedure).

The second test consists of applying a coincident trajectory and car displacement inputs that should represent a closed contour. Therefore, a circular trajectory was used and the car displacement was defined as a cosine in x, a sine in y and a ramp in θ. In Figure 3.17, the test inputs can be seen. Those are the same inputs given in the second test of the present-based trajectory error procedure.

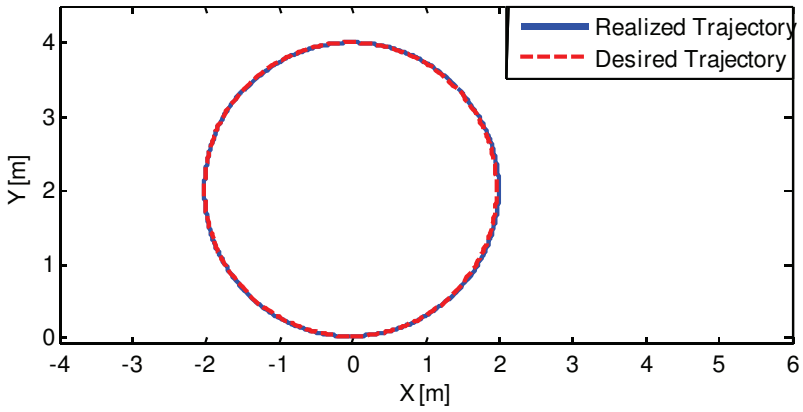

Figure 3.17 – Future-based Trajectory Error Validation 2nd Test: Inputs.

Calculated error outputs for different values of n_{FS} and δ_i appear in Figure 3.18.

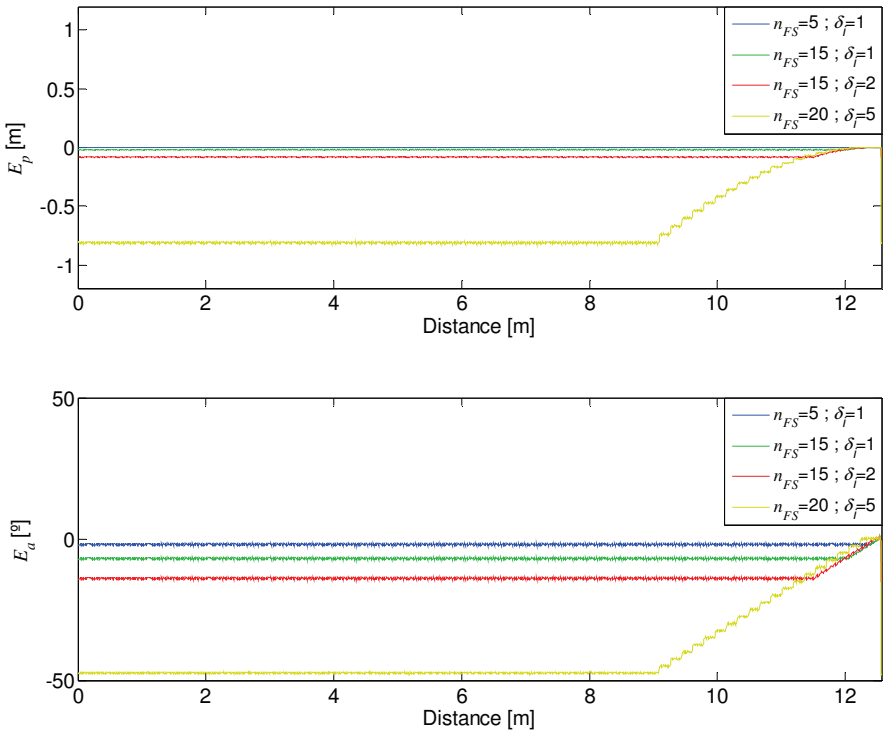

Figure 3.18 – Future-based Trajectory Error Validation 2nd Test: Outputs.

Despite some small disturbance due to numerical difference between the circular trajectory and the car displacement, this test confirms the expectations. Although the car follows the trajectory, the position error increases with n_{FS} and δ_i. This occurs because a larger part of the circle ahead from the car's position is evaluated by the error procedure.

The orientation error repeats the same behavior. This was also expected, once the trajectory is ending and the error procedure does not have more future points to evaluate.