Universitá degli Studi di Trento

Dipartimento di Ingegneria Industriale

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

Tesi di Laurea

Algoritmi per la Valutazione del Contatto tra Pneumatico e Strada in Soft Real Time

Laureando:

Davide Stocco

Relatore:

Prof. Enrico Bertolazzi

Anno Accademico 2019 · 2020

Sommario

This dissertation details …

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli acronimi

Introduzione 1

1.1 Obiettivi

La motivazione di questa tesi sta nella trovata collaborazione tra il *Dipartimento di Ingegneria Industriale* dell'Università di Trento e *AnteMotion S.r.l.*, azienda specializzata in realtà virtuale e simulazione multibody per il campo *automotive*. In particolare il modello di veicolo e pnumatico precedentemente studiati da Larcher in [4] sarà integrato nel simulatore di guida in tempo reale di AnteMotion. Pertanto, lo sviluppo del modello è stato finalizzato a minimizzare i tempi di compilazione massimizzando invece l'accuratezza. La necessità di sviluppare un algoritmo che calcoli i parametri dell'interazione tra terreno e pneumatico getta le basi per il lavoro svolto.

1.2 Il problema

La *simulazione* risolve alcuni dei problemi relativi al mondo della progettazione in modo sicuro ed efficiente, senza la necessità di costruire un prototipo dell'oggetto fisico. A differenza della modellazione fisica, che può coinvolgere il sistema reale o una copia in scala di esso, la simulazione è basata sulla tecnologia digitale e utilizza algoritmi ed equazioni per rappresentare il mondo reale al fine di imitare l'esperimento reale. Ciò comporta diversi vantaggi in termini di tempo, costi e sicurezza. Infatti, il modello digitale può essere facilmente riconfigurato e analizzato,

mentre questo è solitamente impossibile o troppo oneroso del punto di vista di tempi e/o costi da fare con il sistema reale [5]. Al giorno d'oggi esistono numerosi modelli di veicolo e pneumatico. Certamente, più semplice è il modello più veloce è la risoluzione delle equazioni costituenti, quindi a seconda delle applicazioni deve essere scelto il modello con la giusta complessità. Per la maggior parte delle applicazioni di guida autonoma, un modello semplice è sufficiente per caratterizzare con un livello di dettaglio sufficiente il comportamento del veicolo, e poiché queste analisi sono molto spesso fatte con l'ausilio di *Hardware in the Loop* (HIL), il modello dinamico del veicolo deve essere risolto in tempo reale con tipico passo di tempo di 1 millisecondo. Il vincolo in tempo reale implica la scelta un modello di veicolo che sia velocemente risolvibile, ciò significa che i modelli semplici con pochi parametri, di solito modelli lineari a due ruote, sono particolarmente adatti per questo tipo di applicazioni. Tuttavia ci sono alcune situazioni che richiedono modelli più dettagliati, come ad esempio l'azione prodotta da un *Advanced Driver-Assistance Systems* (ADAS), ovvero una manovra di sicurezza come l'elusione degli ostacoli o una frenata di emergenza, poiché il veicolo è spinto nella maggior parte dei casi al limite delle sue prestazioni [3]. In queste condizioni di guida si devono tenere conto di molti fattori come ad esempio il comportamento degli pneumatici, che si sposta nella regione non lineare e i fenomeni transitori non sono più trascurabili. Ciò significa che un modello più dettagliato di quello utilizzato per la guida in condizioni "standard". L'accuratezza dinamica del modello è di grande importanza per ricavare previsioni realistiche delle prestazioni del veicolo e del sistema di controllo. È importante notare che modellare in modo esaustivo tutti i sistemi di un'auto sarebbe un compito estremamente arduo e a volte anche impossibile. Esistono quindi modelli empirici come il modello della *Magic Formula* di Hans Pacejka e il modello *Fiala* che cercano di imitare il reale comportamento del sistema. Il calcolo dei parametri di questo tipo di modelli richiede l'interpolazione di un set di dati di grandi dimensioni, e può quindi essere numericamente inefficiente o comunque troppo oneroso in termini di tempo.

Per studiare il comportamento del sistema in diversi scenari di guida, i moderni strumenti di simulazione spesso richiedono una grande quantità di di dati e l'unico modo per ottenerli è esecuzione stessa di migliaia di simulazioni. A questo proposito è quindi necessario un conducente virtuale o artificiale in grado di controllare il veicolo fino ai limiti di guidabilità. Pertanto, la caratteristica principale di un driver artificiale o virtuale è la capacità di guidare una varietà di veicoli con diverse

caratteristiche dinamiche. Indipendentemente dall'architettura e dalla metodologia utilizzata per sviluppare il conducente artificiale, deve utilizzare una sorta di modello dinamico che rappresenta il comportamento del veicolo controllato. Si tratta dunque di una sorta di modello di comportamento dinamico del veicolo.
Lo scopo di questo lavoro si collega a quello già svolto da Larcher in [4], dove grazie a un modello di veicolo completo con 14 gradi di libertà ha fornito un modello in grado di catturare con un livello di dettaglio appropriato il comportamento del veicolo quando viene spinto ai suoi limiti di maneggevolezza. La necessità di calcolare in tempo reale i parametri di input per il modello di ruota scelto da [4] definisce l'obiettivo di questo lavoro. Ovvero di avere una libreria scritta in `C++`, che con alcuni semplici parametri in input come la denominazione ETRTO dello pneumatico e la posizione nello spazio, calcola i dati relativi all'interazione pneumatico strada quali l'intersezione del punto sotto il centro ruota, l'area di contatto, e l'inclinazione locale del piano strada. Il tutto cercando di minimizzare i tempi di compilazione.

1.3 Convenzioni e Notazioni

1.3.1 Sistemi di Riferimento

La convenzione utilizzata per definire gli assi del sistema di riferimento della vettura è la *International Organization for Standardization* (ISO) 8855.

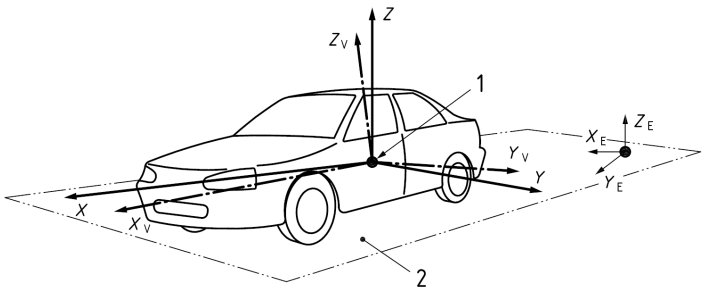


Figura 1.1: Rappresentazione degli assi del sistema di riferimento della vettura secondo la convenzione ISO-V.

Il sistema di riferimento della ruota è conforme alla convenzione ISO-C, la cui disposizione degli assi è illustrata nella Figura 1.2. L'origine del sistema di riferimento del vettore ruota è posta in corrispondenza del centro della ruota mentre

posizione e orientamento relativi rispetto al sistema di riferimento del telaio sono definiti attraverso il modello della sospensione descritto in [4].

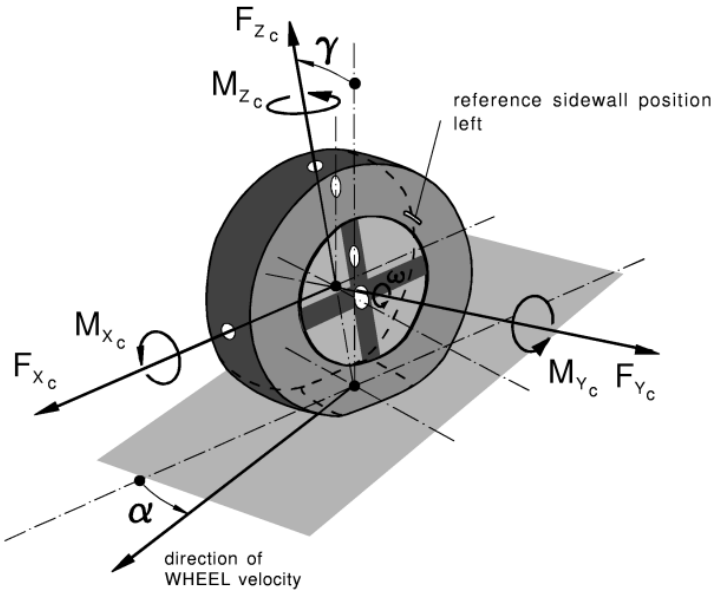


Figura 1.2: Rappresentazione degli assi del sistema di riferimento dello pneumatico secondo la convenzione ISO-C.

1.3.2 Matrice di Trasformazione

Per descrivere sia l'orientamento che la posizione di un sistema di assi nello spazio, viene introdotta la *matrice roto-traslazione*, chiamata anche *matrice di trasformazione*. Questa notazione permette di impiegare le operazioni matrice-vettore per l'analisi di posizione, velocità e accelerazione. La forma generale di una matrice di trasformazione è del tipo:

$$T_m = \left[\begin{array}{ccc|c} & [R_m] & & \begin{matrix} O_{mx} \\ O_{my} \\ O_{mz} \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{1.1}$$

dove R_m è la matrice di rotazione 3×3 del sistema di riferimento in movimento e O_{mx}, O_{my} e O_{mz} sono le coordinate della sua origine nel sistema di riferimento

assoluto o nativo. L'introduzione dell'elemento fittizio 1 nel vettore della posizione di origine e la successiva spaziatura interna zero della matrice rende possibili le moltiplicazioni matrice-vettore, rendendo la matrice di trasformazione un modo compatto e conveniente per la descrizione dei sistemi di riferimento. Si noti che per i vettori, le informazioni traslazionali vengono trascurate imponendo l'elemento fittizio pari a 0.

Il Pneumatico 2

2.1 Introduzione

Gli pneumatici sono probabilmente i componenti più complessi di un'auto in quanto combinano decine di componenti che devono essere formati, assemblati e curati insieme. Il loro successo finale dipende dalla loro capacità di fondere tutti i componenti separati in un prodotto coeso che soddisfa le esigenze del conducente [7]. Gli pneumatici sono caratterizzati da un comportamento altamente non lineare con una dipendenza da diversi fattori costruttivi e ambientali. Tuttavia, le forze di contatto possono essere descritte completamente da un vettore di forza risultante applicato in un punto specifico della patch di contatto e da un vettore di coppia, come illustrato nella Figura 2.1.

Componenti cruciali per la movimentazione dei veicoli e il comportamento di guida, le forze degli pneumatici richiedono particolare attenzione e cura soprattutto quando, lungo il comportamento stazionario, il comportamento non stazionario deve essere coperto. Attualmente, è possibile identificare tre gruppi di modelli:

- modelli matematici;
- modelli fisici;
- combinazione dei precedenti.

La prima tipologia di modello tenta di rappresentare le caratteristiche fisiche del pneumatico attraverso una descrizione puramente matematica. Pertanto questi tipi

F_x	forza longitudinale
F_y	forza laterale
F_z	forza verticale
T_x	coppia di sovrasterzo
T_y	resistenza al rotolamento
T_z	coppia di autoallineamento

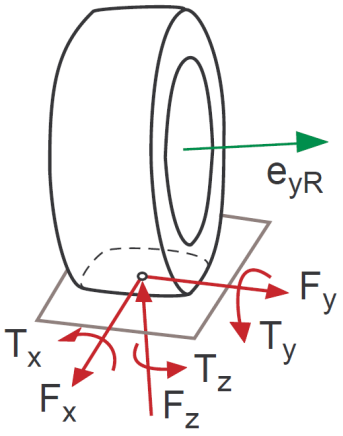


Figura 2.1: Forze e coppie generate dal contatto pneumatico-strada.

di modelli partono da un curve caratteristiche ricavate sperimentalmente e cercano di derivare un comportamento approssimativo dall'interpolazione di dati. Un esempio ben noto di questo approccio è il modello di Pacejka o *Magic Formula Tire Model* [6]. Questo tipo di modellazione è adatta per la simulazione di manovre di guida, dove il comportamento di interesse è per lo più la gestione del veicolo e le frequenze di uscita sono ben al di sotto delle frequenze di risonanza della cintura dello pneumatici. I modelli fisici o i modelli ad alta frequenza, come i modelli agli elementi finiti, sono in grado di rilevare fenomeni di frequenza più elevata come le vibrazioni della membrana. Ciò permette di valutare il comfort di guida di un veicolo. Dal punto di vista del calcolo, i modelli fisici complessi richiedono molto tempo al computer per essere risolto, nonché molti dati. Dall'altro lato, i modelli matematici sono veloci in termini di calcolo, ma richiedono un'accurata pre-elaborazione dei dati sperimentali. La terza tipologia di modelli consiste in un'estensione dei modelli matematici attraverso le leggi fisiche al fine di coprire una gamma di frequenza più ampia.
Il modello di pneumatico sviluppato nel modello di veicolo presentato da Larcher in [4] si basa sulla Magic Formula 6.2. Una panoramica generale sui modelli della Magic Formula e sul calcolo dello slip degli pneumatici è descritta nelle sezioni seguenti, mentre l'insieme completo delle equazioni del modello implementato è riportato nell'Appendice. Come vedremo nel Capitolo 3 il modello di pneumatico deve essere combinato con un'interfaccia di pneumatico/strada modellata adeguata per ottenere risultati significativi.

2.2 Geometria dello Pneumatico secondo ETRTO

Quando si fa riferimento ai dati puramente geometrici, viene utilizzata una forma abbreviata della notazione completa prevista dall'ente di normazione ETRTO. Assumendo di avere un pneumatico generico la notazione che identificherà la geometria sarà del tipo $a/b\mathrm{R}c$. Dove:

- a rappresenta larghezza nominale del pneumatico nel punto più largo;
- b rappresenta percentuale dell'altezza della spalla dello pneumatico in relazione alla larghezza dello stesso;
- c rappresenta il diametro dei cerchi ai quali lo pneumatico si adatta.

Facendo un esempio, 195/55R16 significherebbe che la larghezza nominale del pneumatico è di circa 195 mm nel punto più largo, l'altezza della spalla dello pneumatico è il 55% della larghezza, ovvero 107 mm in questo caso, e che il pneumatico si adatta a dei cerchi di 16 pollici di diametro. Con questa notazione è possibile calcolare direttamente il diametro esterno teorico dello pneumatico tramite la seguente:

$$\phi_e = \frac{2ab}{25.4} + c \quad [\text{in}] \qquad \phi_e = 2ab + 25.4c \quad [\text{mm}] \tag{2.1}$$

Riprendendo l'esempio usato sopra, il diametro esterno risulterà dunque 24.44 in o 621 mm.

Meno comunemente usato negli Stati Uniti e in Europa (ma spesso in Giappone) è una notazione che indica l'intero diametro del pneumatico invece delle proporzioni dell'altezza della parete laterale, quindi non secondo ETRTO. Per fare lo stesso esempio, una ruota da 16 pollici avrebbe un diametro di 406 mm. L'aggiunta del doppio dell'altezza del pneumatico (2×107 mm) produce un diametro totale di 620 mm. Quindi, un pneumatico 195/55R16 potrebbe in alternativa essere etichettato 195/620R16.

Anche se questo è teoricamente ambiguo, in pratica queste due notazioni possono essere facilmente distinte perché l'altezza della parete laterale di un pneumatico automobilistico è in genere molto inferiore alla larghezza. Quindi, quando l'altezza è espressa come percentuale della larghezza, è quasi sempre inferiore al 100% (e certamente meno del 200%). Al contrario, i diametri degli pneumatici del veicolo sono sempre superiori a 200 mm. Pertanto, se il secondo numero è superiore a 200, allora è quasi certo che viene utilizzata la notazione giapponese, se è inferiore a 200 allora viene utilizzata la notazione USA/europea.

Figura 2.2: Esempio di misure, secondo la notazione ETRTO, riportate sulla spalla dello pneumatico.

2.3 Il Modello della *Magic Formula*

2.3.1 La *Magic Formula*

Uno dei modelli di pneumatici più utilizzati è il cosiddetto modello *Magic Formula* sviluppato da Egbert Bakker e Pacejka in [1]. Questo modello è stato poi rivisto e l'ultima versione è riportata in [6]. Il modello Magic Formula consiste in una pura descrizione matematica del rapporto input-output del contatto pneumatico-strada. Questa formulazione collega le variabili di forza con lo slip rigido del corpo che vengono trattati nelle sezioni successive. La forma generale della funzione di descrizione può essere scritta come:

$$y(x) = D\sin\{C\arctan[B(x+S_h) - E(B(x+S_h) - \arctan(B(x+S_h)))]\} + S_v \tag{2.2}$$

dove:

- B rappresenta il fattore di rigidezza;
- C rappresenta il fattore di forma;
- D rappresenta il falore massimo della forza o coppia;
- E rappresenta il fattore di curvatura;
- S_v rappresenta lo spostamento in verticale della curva caratteristica;
- S_h rappresenta lo spostamento in orizzontale della curva caratteristica.

e dove $y(x)$ può essere la forza longitudinale F_x , la forza laterale F_y o la coppia di autoallineamento M_z, mentre x è la componente di slip corrispondente. In Figura 2.3 sono illustrate le curve caratteristiche generiche degli pneumatici derivate con il metodo della Magic Formula.

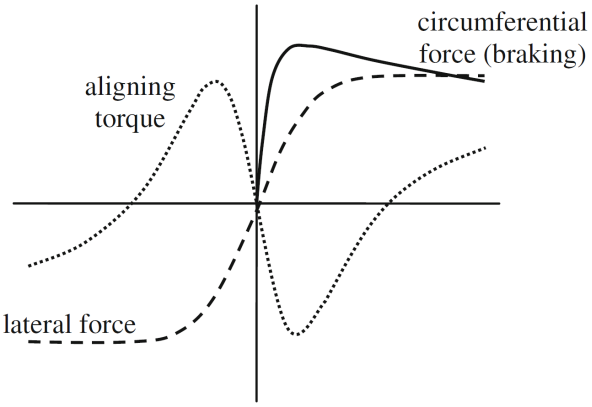


FIGURA 2.3: Curve caratteristiche generiche degli pneumatici derivate con il metodo della Magic Formula

2.3.2 Contatto con la Superficie Stradale

La posizione e l'orientamento della ruota in relazione al sistema fissato a terra sono dati dal telaio di riferimento del vettore ruota RF_{wh_i} che viene calcolato risolvendo le equazioni dinamiche del sistema ottenuto nel Capitolo 2 in [4] istante per istante. Supponendo che il profilo stradale potrebbe essere rappresentato da una funzione arbitraria di due coordinate spaziali

$$z = z(x, y) \tag{2.3}$$

su una irregolare, il punto di contatto P non può essere calcolato direttamente. Così, come prima approssimazione siamo in grado di identificare un punto P, che è definito come una semplice traslazione del centro ruota M:

$$P^\star = M - R_0 \boldsymbol{e}_{zC} \begin{bmatrix} x^\star \\ y^\star \\ z^\star \end{bmatrix} \tag{2.4}$$

dove R_0 è il raggio dello pneumatico indeformato e $\boldsymbol{e}_{zC}$ è il vettore unitario che definisce l'asse z_c del sistema di riferimento del vettore ruota.

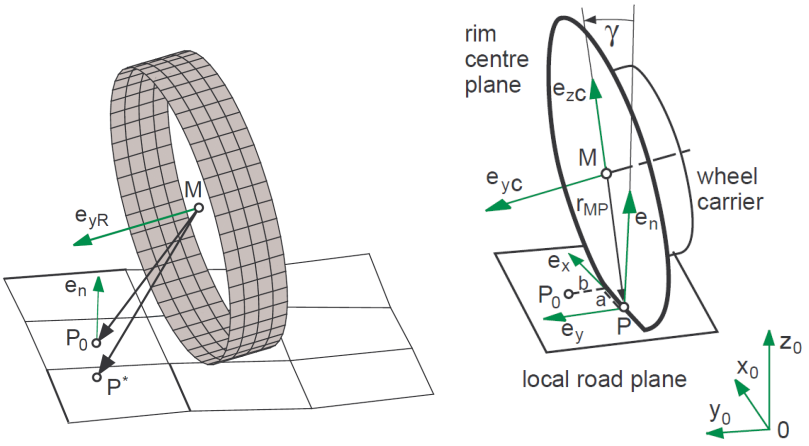


Figura 2.4: Geometria del contatto pneumatico-strada.

La prima stima del sistema di riferimento del punto di contatto $RF_{PC^\star}$ è un frame con origine in $P^\star$ e orientamento dell'asse definito dall'orientamento dell'asse del sistema portante della ruota.

$$RF_{PC^\star} = \left[\begin{array}{ccc|c} & [R_{RF_{wh}}] & & \begin{matrix} x^\star \\ y^\star \\ z^\star \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.5}$$

Ora, i vettori di unità $\boldsymbol{e}_x$ ed $\boldsymbol{e}_y$, che descrivono il piano locale nel punto P, possono essere ottenuti dalle seguenti equazioni:

$$\boldsymbol{e}_x = \frac{\boldsymbol{e}_{yC} \times \boldsymbol{e}_n}{|\boldsymbol{e}_{yC} \times \boldsymbol{e}_n|} \qquad \boldsymbol{e}_y = \boldsymbol{e}_n \times \boldsymbol{e}_x \tag{2.6}$$

Al fine di ottenere una buona approssimazione del piano pista locale in termini di inclinazione longitudinale e laterale, sono stati utilizzati un insieme di quattro punti di campionamento $(Q_1^\star, Q_2^\star, Q_3^\star, Q_4^\star)$ che sono rappresentati graficamente in Figura 2.5. I punti di campionamento sono definiti sul piano locale del punto di contatto $RF_{PC^\star}$, poiché lo spostamento longitudinale e laterale dall'origine del sistema, che è $P^\star$. I vettori di spostamento sono definiti come:

$$\begin{aligned} {}_{PC^\star}\boldsymbol{r}_{Q^\star_{1,2}} &= \pm\Delta x \\ {}_{PC^\star}\boldsymbol{r}_{Q^\star_{3,4}} &= \pm\Delta y \end{aligned} \tag{2.7}$$

e quindi, i quattro punti di campionamento sono:

$$
\begin{aligned}
{}^{P^\star}\boldsymbol{r}_{Q^\star_{1,2}} &= P^\star \pm \Delta x \boldsymbol{e}_{xPC^\star} \\
{}^{P^\star}\boldsymbol{r}_{Q^\star_{3,4}} &= P^\star \pm \Delta y \boldsymbol{e}_{yPC^\star}
\end{aligned}
\tag{2.8}
$$

Al fine di campionare la patch di contatto nel modo più efficiente possibile, le distanze di Δx e Δy, dell'equazione precedente, vengono regolate in base al raggio del pneumatico indeformato R_0 e alla larghezza del pneumatico B. I valori di queste due quantità possono essere trovate in letteratura e sono $\Delta x = 0.1R_0$ e $\Delta x = 0.3B$. Attraverso questa definizione, si può ottenere un comportamento realistico durante la simulazione.

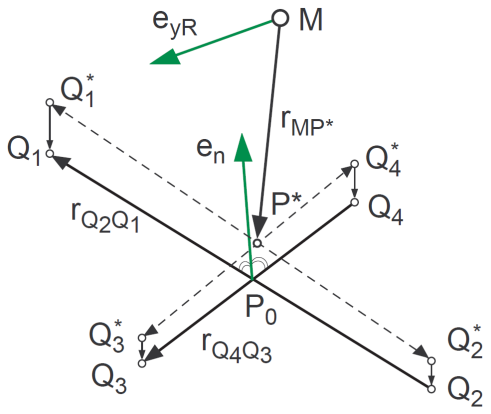


FIGURA 2.5: Punti campionati nel piano locale della superficie stradale.

Ora il componente traccia z in corrispondenza dei quattro punti campione viene valutato attraverso la funzione $z(x, y)$ (Eq. 3.1), quindi, aggiornando la terza coordinata dei punti di sondaggio $Q^\star_i$, otteniamo i corrispondenti punti campione Q_i sulla superficie della pista locale. La linea fissata dai punti Q_1, Q_2 e rispettivamente Q_3, Q_4, può ora essere utilizzata per definire il vettore normale del piano della pista locale (Figura??). Pertanto, il vettore normale è definito come:

$$
\boldsymbol{e}_n = \frac{\boldsymbol{r}_{Q_1Q_2} \times \boldsymbol{r}_{Q_4Q_3}}{|\boldsymbol{r}_{Q_1Q_2} \times \boldsymbol{r}_{Q_4Q_3}|}
\tag{2.9}
$$

dove sono $\boldsymbol{r}_{Q_2Q_1}$ e $\boldsymbol{r}_{Q_4Q_3}$ sono i vettori che puntano rispettivamente da Q_1 a Q_2 e da Q_3 a Q_4. Applicando Eq 3.4 è ora possibile calcolare i vettori unitari $\boldsymbol{e}_x$ e $\boldsymbol{e}_y$ del piano di locale del punto di contatto. Il punto di contatto P si ottiene aggiornando le coordinate del primo punto di prova $P^\star$, con il valore medio delle tre coordinate

spaziali dei quattro punti campione.

$$P = \frac{1}{4} \begin{bmatrix} \sum_{i=1}^{4} x_i \\ \sum_{i=1}^{4} y_i \\ \sum_{i=1}^{4} z_i \end{bmatrix} \tag{2.10}$$

Infine possiamo mettere assieme tutte le componenti del piano di riferimento del punto di contatto finale ottenendo:

$$RF_{PC} = \left[\begin{array}{ccc|c} & & & x_P \\ [\boldsymbol{e}_x] & [\boldsymbol{e}_y] & [\boldsymbol{e}_z] & y_P \\ & & & z_P \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.11}$$

Attraverso questo approccio di modellazione, le informazioni della traccia locale normal vector $\boldsymbol{e}_n$, insieme al punto di contatto locale P sono in grado di rappresentare l'irregolarità locale in modo soddisfacente. Come accade in realtà, bordi taglienti o discontinuità del manto stradale saranno smussate da questo approccio. Alcuni casi dimostrativi sono illustrati nella Figura 2.6.

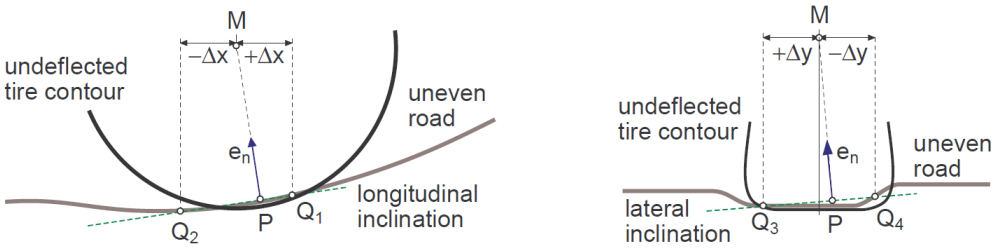


Figura 2.6: Inclinazione longitudinale e laterale del piano strada locale.

Algoritmi Geometrici 3

3.1 Introduzione

La geometria computazionale è la branca dell'Informatica che studia le strutture dati e gli algoritmi efficienti per la soluzione di problemi di natura geometrica e la loro implementazione al calcolatore. Storicamente, è considerato uno dei campi più antichi del calcolo, anche se la geometria computazionale moderna è uno sviluppo recente. La ragione principale per lo sviluppo della geometria computazionale è stata dovuta ai progressi compiuti nella computer grafica, *Computer-Aided Design* (CAD), *Computer-Aided Manufacturing* (CAM) e nella visualizzazione matematica. Ad oggi, le applicazioni della geometria computazionale si trovano nella robotica, nella progettazione di circuiti integrati, nella visione artificiale, in *Computer-Aided Engineering* (CAE) e nel *Geographic Information Systems* (GIS).
I rami principali della geometria computazionale sono:

- *Calcolo combinatorio* (o *geometria algoritmica*), che si occupa di oggetti geometrici come entità discrete. Ad esempio, può essere utilizzato per determinare il poliedro o il poligono più piccolo che contiene tutti i punti forniti, o più formalmente, dato un insieme di punti, si deve determinare il più piccolo insieme convesso che li contenga tutti (problema dell'inviluppo convesso).
- *Geometria di calcolo* numerica (o *Computer-Aided Geometric Design* (CAGD)), che si occupa principalmente di rappresentare oggetti del mondo reale in forme adatte per i calcoli informatici nei sistemi CAD e CAM. Questo ramo

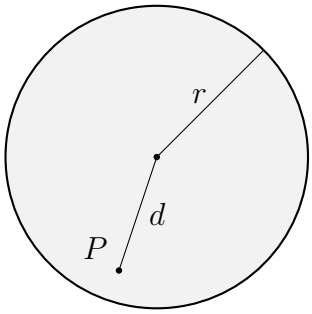


Figura 3.1: Point-circle intersection problem scheme.

può essere visto come uno sviluppo della geometria descrittiva ed è spesso considerato un ramo della computer grafica o del CAD. Entità importanti di questo ramo sono superfici e curve parametriche, come ad esempio le *spline* e *curve di Bézier*.

In questo capitolo tutti gli algoritmi che verranno utilizzati in seguito durante l'analisi geometrica dell'intersezione tra pneumatico e superfice stradale saranno trattati. Questi algoritmi sono la soluzione di alcuni semplici ma molto importanti problemi, che devono essere risolti in modo efficiente. In particolare le intersezioni tra:

- punto e segmento (sul piano);
- punto e circonferenza (sul piano);
- raggio e circonferenza (sul piano);
- raggio e triangolo (sullo spazio);

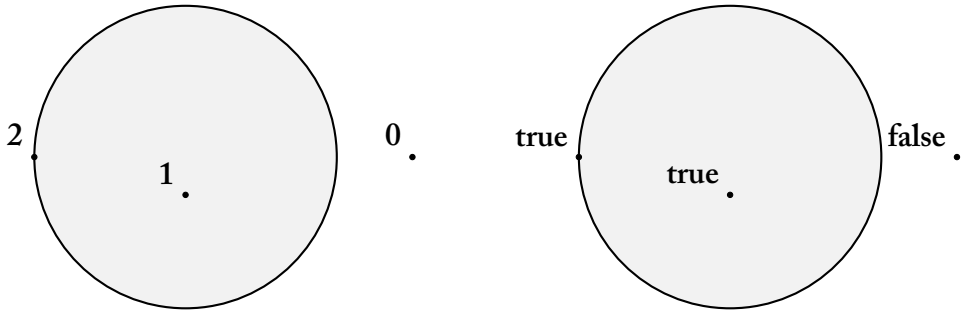
saranno esaminato al fine di trovare la massima prestazione in termini di *efficienza computazionale*.

3.2 Point-Segment Intersection

3.3 Point-Circle Intersection

Having a circle with center $C = (x_c, y_c)$ and radius r, the problem consists in finding out whether a query point $P = (x_p, y_p)$ is inside, outside or on the circle. The solution to the problem is simple: the distance between the circle center C and the query point P is given by the *Pythagorean theorem* as

$$d = \sqrt{(x_p - x_c)^2 + (y_p - y_c)^2} \tag{3.1}$$



(a) Integer output (b) Boolean output

FIGURA 3.2: Output schemes of the point-circle intersection problem.

The query point P is *inside* the circle if $d < r$, on the circle if $d = r$, and *outside* the circle if $d > r$. Little work can be saved by comparing d^2 with r^2 instead: the point P is *inside* the circle if $d^2 < r^2$, on the circle if $d^2 = r^2$, and *outside* the circle if $d^2 > r^2$. Thus, the final comparison will be between the number $(x_p - x_c)^2 + (y_p - y_c)^2$ and r^2.
The *inputs* of the point-circle intersection algorithm are:

- the circle center $C = (x_c, y_c)$;
- the circle radius r;
- a query point $P = (x_p, y_p)$.

The *output* could be an integer which value is:

- 0 if the point is outside;
- 1 if the point is inside;
- 2 if the point is on the circle.

Another option could be a boolean which value is:

- false if the point is outside;
- true if the point is inside or on the circle.

On Figura 3.3 the schemes for the point-circle intersection algorithm with integer and boolean outputs are reported.

With integer output

$d = (x_p - x_c)^2 + (y_p - y_c)^2$
if $(d > r^2)\{$
  return 0
$\}$ **else if** $(d < r^2)\{$
  return 1
$\}$ **else** $\{$
  $// d = r^2$
  return 2
$\}$

With boolean output

$d = (x_p - x_c)^2 + (y_p - y_c)^2$
if $(d > r^2)\{$
  return false
$\}$ **else** $\{$
  $// d <= r^2$
  return true
$\}$

Figura 3.3: Point-circle intersection algorithm schemes.

3.4 Ray-Circle Intersection

3.5 Ray-Triangle Intersection

Having a triangle with vertices (V_1, V_2, V_3) and a ray R with origin R_O and direction R_D, the problem consists in finding out whether the ray hits or not the triangle and if so, where is the intersection point P. Over the last decades, plenty of algo-

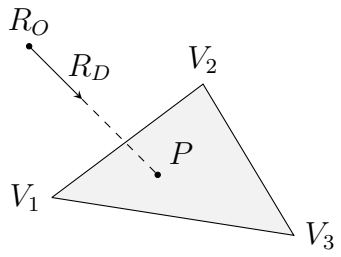


Figura 3.4: Ray-triangle intersection problem scheme.

rithms for solving this problem had been purposed, so there are several solutions to the ray/triangle or ray-triangle intersection problem. Three of the most relevant algorithms are:

- *Badouel* algorithm;
- *Segura* algorithm;
- *Möller–Trumbore* algorithm.

As Jiménez, Segura e Feito states in [2], the Möller-Trumbore's is the faster algorithm when the normal and/or the projection plane have not been previously stored,

as in this thesis.

3.5.1 Möller-Trumbore Algorithm

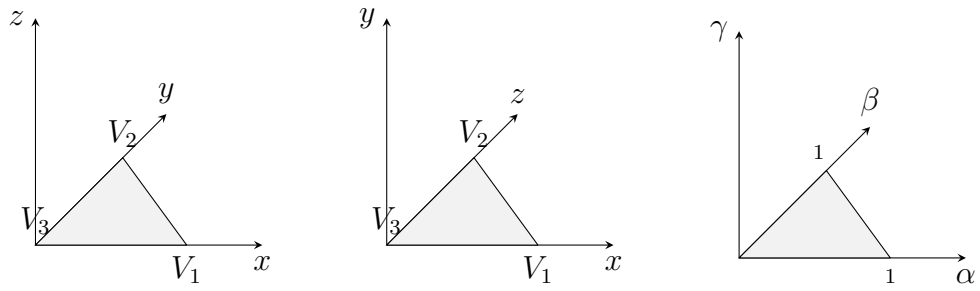


Figura 3.5: Transformation and base change of ray in Möller-Trumbore algorithm.

The inputs of the Möller-Trumbore algorithm are:

- Triangle vertices (V_1, V_2, V_3);
- Segment points (Q_1, Q_2).

With back-face culling

$Q = Q_2 - Q_1$
$E_1 = V_2 - V_1$
$E_2 = V_3 - V_1$
$A = Q \times E_2$
$D = A \cdot E_1$
if $(D > \varepsilon)\{$
  $T = Q_1 - V_1$
  $u = A \cdot T$
  if $(u < 0.0 \,||\, u > D)\{$
    return false
  $\}$
  $B = T \times E_1$
  $v = B \cdot Q$
  if $(v < 0.0 \,||\, u + v > D)\{$
    return false
  $\}$
$\}$ **else if** $(D < -\varepsilon)\{$
  $T = Q_1 - V_1$
  $u = A \cdot T$
  if $(u > 0.0 \,||\, u < D)\{$
    return false
  $\}$
  $B = T \times E_1$
  $v = B \cdot Q$
  if $(v > 0.0 \,||\, u + v < D)\{$
    return false
  $\}$
$\}$ **else** $\{$
  return false
$\}$
$D_{inv} = 1.0/D$
$t = (B \cdot E_2) * D_{inv}$
if $(t > 0.0)\{$
  $P = Q + D * t$
  return true
$\}$ **else** $\{$
  return false
$\}$

Without back-face culling

$Q = Q_2 - Q_1$
$E_1 = V_2 - V_1$
$E_2 = V_3 - V_1$
$A = Q \times E_2$
$D = A \cdot E_1$
if $(D < \varepsilon)\{$
  return false
$\}$
$T = Q_1 - V_1$
$u = A \cdot T$
if $(u < 0.0 \,||\, u > D)\{$
  return false
$\}$
$B = T \times E_1$
$v = B \cdot Q$
if $(v < 0.0 \,||\, u + v > D)\{$
  return false
$\}$
$D_{inv} = 1.0/D$
$t = (B \cdot E_2) * D_{inv}$
if $(t > 0.0)\{$
  $P = Q + D * t$
  return true
$\}$ **else** $\{$
  return false
$\}$

Figura 3.6: Ray-triangle intersection algorithm schemes.

Il Codice C++ 4

Conclusioni e Lavoro Futuro 5

A Chapter of Examples 6

6.1 A Table

Feature	Misuse-based	Anomaly-based
Modeled activity:	Malicious	Normal
Detection method:	Matching	Deviation
Threats detected:	Known	Any
False negatives:	High	Low
False positives:	Low	High
Maintenance cost:	High	Low
Attack desc.:	Accurate	Absent
System design:	Easy	Difficult

Tabella 6.1: Duality between misuse- and anomaly-based intrusion detection techniques. Note that, an anomaly-based CAMs can detect "Any" threat, under the assumption that an attack always generates a deviation in the modeled activity.

6.2 Code

```
1  /* ... */ cd['<'] = {0.1, 0.11} cd['a'] = {0.01, 0.2} cd['b'] =
2  {0.13, 0.23} /* ... */
3
4  b = decode(arg3_value);
```

```

if ( !(cd['c'][0] < count('c', b) < cd['c'][1]) ||\
     !(cd['<'][0] < count('<', b) < cd['<'][1]) ||\
     ... || ...)  fire_alert("Anomalous content detected!");
/* ... */
```

6.3 A Sideways Table

Approach	Time	Header	Payload	Stochastic	Determ.	Clustering
[phad]		•				•
[kruegel:sac2002:anomaly]		•	•	•		
[protocolanom]		•		•	•	
[ramadas]			•			•
[rules-payl]	•		•		•	
[zanero-savaresi]		•	•			•
[wang:raid2004:payl]			•	•		
[zanero-pattern]		•	•			•
[DBLP:conf/iwia/BolzoniEHZ06]		•	•			•
[wang:raid2006:anagram]			•	•		

Tabella 6.2: Taxonomy of the selected state of the art approaches for network-based anomaly detection.

6.4 A Figure

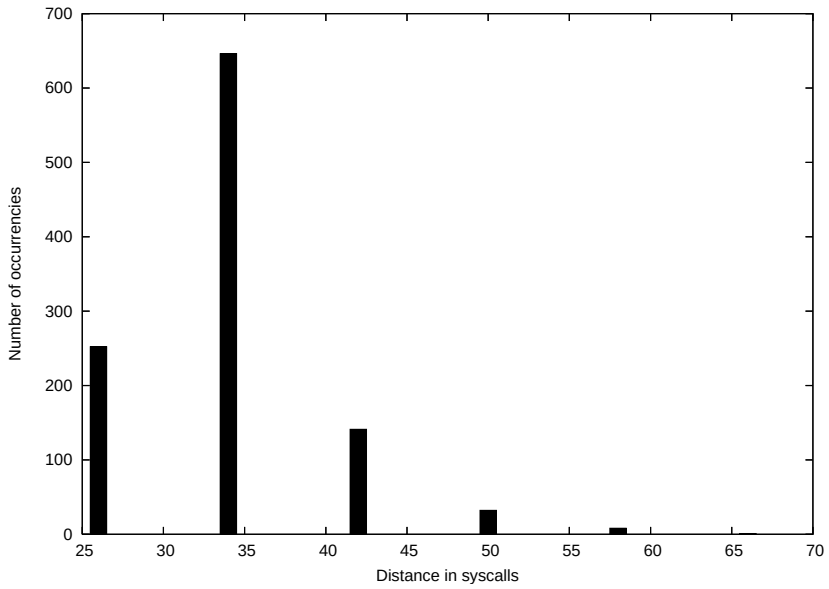


Figura 6.1: `telnetd`: distribution of the number of other system calls among two `execve` system calls (i.e., distance between two consecutive `execve`).

6.5 Bulleted List

- O =“Intrusion”, $\neg O$ =“Non-intrusion”;
- A =“Alert reported”, $\neg A$ =“No alert reported”.

6.6 Numbered List

1. O =“Intrusion”, $\neg O$ =“Non-intrusion”;
2. A =“Alert reported”, $\neg A$ =“No alert reported”.

6.7 A Description

Time refers to the use of *timestamp* information, extracted from network packets, to model normal packets. For example, normal packets may be modeled by their minimum and maximum inter-arrival time.

6.8 An Equation

$$d_a(i,j) := \begin{cases} K_a + \alpha_a \delta_a(i,j) & \text{if the elements are different} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

6.9 A Theorem, Proposition & Proof

Theorem 6.9.1 $a^2 + b^2 = c^2$

Proposition 6.9.2 $3 + 3 = 6$

Proof 6.9.1 *For any finite set* $\{p_1, p_2, ..., p_n\}$ *of primes, consider* $m = p_1 p_2 ... p_n + 1$. *If* m *is prime it is not in the set since* $m > p_i$ *for all* i. *If* m *is not prime it has a prime divisor* p. *If* p *is one of the* p_i *then* p *is a divisor of* $p_1 p_2 ... p_n$ *and hence is a divisor of* $(m - p_1 p_2 ... p_n) = 1$, *which is impossible; so* p *is not in the set. Hence a finite set* $\{p_1, p_2, ..., p_n\}$ *cannot be the collection of all primes.*

6.10 Definition

Definition 6.10.1 (Anomaly-based CAM) *An* anomaly-based CAM *is a type of CAM that generate alerts* $\mathbb{A}$ *by relying on normal activity profiles.*

6.11 A Remark

Remark 1 *Although the network stack implementation may vary from system to system (e.g.,* Windows *and* Cisco *platforms have different implementation of CAM).*

6.12 An Example

Example 6.12.1 (Misuse *vs.* Anomaly) *A misuse-based system* M *and an anomaly-based system* A *process the same log containing a full dump of the system calls invoked by the kernel of an audited machine. Log entries are in the form:*

```
<function_name>(<arg1_value>, <arg2_value>, ...)
```

6.13 Note

Note 6.13.1 (Inspection layer) *Although the network stack implementation may vary from system to system (e.g.,* Windows *and* Cisco *platforms have different implementation of CAM), it is important to underline that the notion of IP, TCP, HTTP* packet *is well defined in a system-agnostic way, while the notion of* operating system activity *is rather vague and by no means standardized.*

Codice della Libreria C++ A

A.1 RoadRDF.hh

```
1 ///
2 /// file: MeshRDF.hh
3 ///
4
5 #pragma once
6 #include <AABBtree.hh>
7 #include <Eigen/Dense>  // Eigen linear algebra Library
8 #include <cmath>        // Math.h - STD math Library
9 #include <fstream>      // fStream - STD File I/O Library
10 #include <iostream>     // Iostream - STD I/O Library
11 #include <string>       // String - STD String Library
12 #include <vector>       // Vector - STD Vector/Array Library
13 #include <memory>
14
15 // Print progress to console while loading (large models)
16 #define RDF_CONSOLE_OUTPUT
17
18 #ifndef RDF_ERROR
19   #define RDF_ERROR(MSG) {                 \
20     std::ostringstream ost; ost << MSG;    \
```

```
21     throw std::runtime_error( ost.str() ); \
22   }
23 #endif
24
25 #ifndef RDF_ASSERT
26   #define RDF_ASSERT(COND,MSG) \
27     if ( !(COND) ) RDF_ERROR( MSG )
28 #endif
29
30 //! RDF mesh computations routine
31 namespace RDF {
32
33   typedef double real_type;  //!< Real number type
34   typedef int    int_type;   //!< Integer number type
35
36   typedef Eigen::Vector2d vec2;  //!< 2D vector type
37   typedef Eigen::Vector3d vec3;  //!< 3D vector type
38   typedef Eigen::Matrix3d mat3;  //!< 3x3 matrix type
39
40   typedef std::basic_ostream<char> ostream_type;  //!< Output
       stream type
41
42   //! Class that handle triangle bounding box.
43   class BBox3D {
44   private:
45     real_type Xmin;  //!< Xmin shadow domain point
46     real_type Ymin;  //!< Ymin shadow domain point
47     real_type Xmax;  //!< Xmax shadow domain point
48     real_type Ymax;  //!< Ymax shadow domain point
49
50   public:
51
52     //! Default constructor for orientation object.
53     BBox3D() {}
54
55     //! Variable set constructor for Bounding box object.
56     BBox3D( vec3 const Vertices[3] ) {
```

```
57        updateBBox3D( Vertices );
58      }
59
60      //! Set Xmin shadow domain point.
61      void
62      setXmin(real_type const _Xmin) { Xmin = _Xmin; }
63
64      //! Set Ymin shadow domain point.
65      void
66      setYmin(real_type const _Ymin) { Ymin = _Ymin; }
67
68      //! Set Xmax shadow domain point.
69      void
70      setXmax(real_type const _Xmax) { Xmax = _Xmax; }
71
72      //! Set Ymax shadow domain point.
73      void
74      setYmax(real_type const _Ymax) { Ymax = _Ymax; }
75
76      //! Get Xmin shadow domain point.
77      real_type
78      getXmin() const { return Xmin; }
79
80      //! Get Ymin shadow domain point.
81      real_type
82      getYmin() const { return Ymin; }
83
84      //! Get Xmax shadow domain point.
85      real_type
86      getXmax() const { return Xmax; }
87
88      //! Get Ymax shadow domain point.
89      real_type
90      getYmax() const { return Ymax; }
91
92      //! Clear the bounding box domain.
93      void clear(void);
```

```
94
95    //! Print bounding box vertices.
96    void
97    print(ostream_type & stream) const {
98      stream
99        << "BBOX (xmin,ymin,xmax,ymax) = ( " << Xmin << ", " <<
              Ymin
100       << ", " << Xmax << ", " << Ymax << " )\n";
101   }
102
103   //! Update the bounding box domain with multiple input
         triangles object.
104   void
105   updateBBox3D( vec3 const Vertices[3] );
106 };
107
108 //! Class for handling triangles.
109 class Triangle3D {
110 private:
111   vec3      Vertices[3];  //!< Vertices vector
112   real_type Friction;     //!< Face friction coefficient
113   BBox3D    TriangleBBox; //!< Triangle bounding box
114
115 public:
116   //! Variable set constructor for orientation object.
117   Triangle3D( vec3 const _Vertices[3], real_type _Friction ) {
118     Vertices[0] = _Vertices[0];
119     Vertices[1] = _Vertices[1];
120     Vertices[2] = _Vertices[2];
121     Friction    = _Friction;
122     TriangleBBox.updateBBox3D(Vertices);
123   }
124
125   //! Get triangle face normal versor.
126   vec3
127   Normal(void) const {
128     vec3 d1 = Vertices[1] - Vertices[0];
```

```
129       vec3 d2 = Vertices[2] - Vertices[0];
130       return d1.cross(d2).normalized();
131     }
132
133     //! Set vertices vector and update bounding box domain.
134     void
135     setVertices( vec3 const _Vertices[3] ) {
136       Vertices[0] = _Vertices[0];
137       Vertices[1] = _Vertices[1];
138       Vertices[2] = _Vertices[2];
139       TriangleBBox.updateBBox3D(Vertices);
140     }
141
142     //! Set friction.
143     void
144     setFriction( real_type _Friction ) { Friction = _Friction; }
145
146     //! Get i-th vertex.
147     vec3 const &
148     getithVertex( unsigned i ) const { return Vertices[i]; }
149
150     //! Get friction coefficent on the face.
151     real_type
152     getFriction(void) const  { return Friction; }
153
154     //! Get triangle bonding box.
155     BBox3D const &
156     getBBox(void) const { return TriangleBBox; }
157
158     //! Print vertices information.
159     void
160     print( ostream_type & stream ) const {
161       stream
162         << "V1:\t" << Vertices[0] << '\n'
163         << "V2:\t" << Vertices[1] << '\n'
164         << "V3:\t" << Vertices[2] << std::endl;
165     }
```

```
166 };
167
168 //! Algorithms for RDF mesh computations routine.
169 namespace algorithms {
170
171   //! Split a string into a string array at a given token.
172   void
173   split(
174     std::string const        & in,    //!< Input string
175     std::vector<std::string> & out,   //!< Output string vector
176     std::string const        & token  //!< Token
177   );
178
179   //! Get tail of string after first token and possibly
          following spaces.
180   std::string
181   tail( std::string const & in );
182
183   //! Get first token of string.
184   std::string
185   firstToken( std::string const & in );
186
187   //! Get element at given index position.
188   template<typename T>
189   T const &
190   getElement(
191     std::vector<T> const & elements,  //!< Elements vector
192     std::string    const & index      //!< Index position
193   );
194 }  // namespace algorithms
195
196 //! Class for handlinf mesh surface object.
197 class MeshSurface {
198 private:
199   std::vector<std::shared_ptr<Triangle3D> > PtrTriangleVec;
          //!< Triangles vector list
200   std::vector<G2lib::BBox::PtrBBox>         PtrBBoxVec;
```

```
        //!< Bounding boxes pointers
201     G2lib::AABBtree::PtrAABB PtrTree
202                         = std::make_shared<G2lib::AABBtree>();
                              //!< Mesh tree pointer
203
204     MeshSurface( MeshSurface const & ) = delete; // costruttore
           di copia
205     MeshSurface & operator = ( MeshSurface const & ) = delete; //
            operatore di copia
206
207   public:
208     // Default set constructor for mesh object.
209     MeshSurface() {};
210
211     // Variable set constructor for mesh object.
212     MeshSurface( std::vector<std::shared_ptr<Triangle3D> > const
           & _PtrTriangleVec ) {
213       this->PtrTriangleVec = _PtrTriangleVec;
214       updatePtrBBox();
215       PtrTree->build(PtrBBoxVec);
216     };
217
218     // Variable set constructor for mesh object.
219     MeshSurface( std::string const & Path ){
220       bool load = LoadFile(Path);
221       RDF_ASSERT( load, "Error while reading file" );
222     }
223
224     //! Get all triangles inside the mesh as a vector.
225     std::vector<std::shared_ptr<Triangle3D> > const &
226     getTrianglesPtr(void) const
227     { return PtrTriangleVec; }
228
229     //! Get i-th triangle.
230     std::shared_ptr<Triangle3D> const &
231     getithTrianglePtr( unsigned i ) const
232     { return PtrTriangleVec[i]; }
```

```
233
234     //! Get AABB tree.
235     G2lib::AABBtree::PtrAABB
236     getAABBPtr(void) const
237     { return PtrTree; }
238
239     //! Print data in file.
240     void
241     printData( std::string const & FileName );
242
243     //! Get the mesh G2lib bounding boxes pointers vector.
244     std::vector<G2lib::BBox::PtrBBox> const &
245     getPtrBBox() const
246     { return PtrBBoxVec; }
247
248     //! Copy the mesh.
249     void
250     set( MeshSurface const & in ) {
251       this->PtrTriangleVec = in.PtrTriangleVec;
252       this->PtrBBoxVec  = in.PtrBBoxVec;
253       this->PtrTree  = in.PtrTree;
254     }
255
256     //! Load the RDF model and print information on a file.
257     //! If RDF model is properly loaded true value is returned.
258     bool
259     LoadFile( std::string const & Path );
260
261   private:
262     //! Update the mesh G2lib bounding boxes pointers vector.
263     void updatePtrBBox(void);
264
265     //! Generate vertices from a list of positions face line.
266     void
267     GenVerticesFromRawRDF(
268       std::vector<vec3> const & iNodes,
269       std::string       const & icurline,
```

```
270       vec3                        oVerts[3]
271     );
272   };
273
274 }  // namespace RDF
275
276 ///
277 /// eof: MeshRDF.hh
278 ///
```

A.2 RoadRDF.cc

```
1 #include "RoadRDF.hh"  // RDF file extention Loader
2
3
4 //! RDF mesh computations routine
5 namespace RDF {
6
7   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        - - - - - - - -
8   // class BBox3D
9   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        - - - - - - - -
10
11  //! Clear the bounding box domain.
12  void
13  BBox3D::clear(void) {
14    Xmin = std::numeric_limits<real_type>::quiet_NaN();;
15    Ymin = std::numeric_limits<real_type>::quiet_NaN();;
16    Xmax = std::numeric_limits<real_type>::quiet_NaN();;
17    Ymax = std::numeric_limits<real_type>::quiet_NaN();;
18  }
19
20  //! Update the bounding box domain with multiple input
        triangles object.
21  void
22  BBox3D::updateBBox3D( vec3 const Vertices[3] ) {
```

```
23    G2lib::minmax3( Vertices[0][0], Vertices[1][0], Vertices
          [2][0], Xmin, Xmax );
24    G2lib::minmax3( Vertices[0][1], Vertices[1][1], Vertices
          [2][1], Ymin, Ymax );
25  }
26
27  // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        - - - - - - - -
28  // class MeshSurface
29  // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        - - - - - - - -
30
31  //! Print data in file.
32  void
33  MeshSurface::printData( std::string const & FileName ) {
34    // Create/Open Out.txt
35    std::ofstream file(FileName);
36
37    // Print introduction
38    file
39      << "LOADED RDF MESH DATA\n\n"
40      << "Legend:\n"
41      << "\tVi: i-th vertex\n"
42      << "\t N: normal to the face\n"
43      << "\t F: friction coefficient\n\n";
44
45    for ( unsigned i = 0; i < PtrTriangleVec.size(); ++i ) {
46      Triangle3D const & Ti = *PtrTriangleVec[i];
47      vec3 const & V0 = Ti.getithVertex(0);
48      vec3 const & V1 = Ti.getithVertex(1);
49      vec3 const & V2 = Ti.getithVertex(2);
50      vec3          N = Ti.Normal();
51
52      // Print vertices, normal and friction
53      file
54        << "TRIANGLE " << i
55        << "\n\tV0:\t" << V0[0] << ", " << V0[1] << ", " << V0[2]
```

```
56        << "\n\tV1:\t" << V1[0] << ", " << V1[1] << ", " << V1[2]
57        << "\n\tV2:\t" << V2[0] << ", " << V2[1] << ", " << V2[2]
58        << "\n\t N:\t" << N[0]  << ", " << N[1]  << ", " << N[2]
59        << "\n\t F:\t" << Ti.getFriction()
60        << "\n\n";
61    }
62    // Close File
63    file.close();
64  }
65
66  //! Update the mesh G2lib bounding boxes pointers vector
67  void
68  MeshSurface::updatePtrBBox(void) {
69    PtrBBoxVec.clear();
70    RDF::BBox3D iBBox;
71    for (unsigned id = 0; id < PtrTriangleVec.size(); ++id) {
72      iBBox = (*PtrTriangleVec[id]).getBBox();
73      PtrBBoxVec.push_back(G2lib::BBox::PtrBBox(
74          new G2lib::BBox(iBBox.getXmin(), iBBox.getYmin(), iBBox
              .getXmax(),
75                          iBBox.getYmax(), id, 0)));
76      iBBox.clear();
77    }
78  }
79
80  //! Load the RDF model and print information on a file
81  bool
82  MeshSurface::LoadFile( std::string const & Path ) {
83    // Check if the file is an ".rdf" file, if not return false
84    if (Path.substr(Path.size() - 4, 4) != ".rdf") {
85      std::cerr << "Not a RDF file\n";
86      return false;
87    }
88
89     // Check if the file had been correctly open, if not return
         false
90    std::ifstream file(Path);
```

```
91      if (!file.is_open()) {
92        std::cerr << "RDF file not opened\n";
93        return false;
94      }
95
96      // Vector for nodes coordinates
97      std::vector<vec3> Nodes;
98
99      bool nodes_parse    = false;
100     bool elements_parse = false;
101
102 #ifdef RDF_CONSOLE_OUTPUT
103     int_type const outputEveryNth = 5000;
104     int_type outputIndicator      = outputEveryNth;
105 #endif
106
107     std::string curline;
108     while (std::getline(file, curline)) {
109 #ifdef RDF_CONSOLE_OUTPUT
110       if ((outputIndicator = ((outputIndicator + 1) %
              outputEveryNth)) == 1) {
111         std::cout
112           << "\r- "
113           << "Loading mesh..."
114           << "\t triangles > "
115           << PtrTriangleVec.size() << std::endl;
116       }
117 #endif
118
119       std::string token = algorithms::firstToken(curline);
120       if ( token == "[NODES]" || token == "NODES" ) {
121         nodes_parse    = true;
122         elements_parse = false;
123         continue;
124       } else if (token == "[ELEMENTS]" || token == "ELEMENTS") {
125         nodes_parse    = false;
126         elements_parse = true;
```

```
127         continue;
128       } else if (token[0] == '{') {
129         // commento multiriga, continua a leggere fino a che
              trovo '}'
130         continue;
131       } else if (token[0] == '%' || token[0] == '#' || token[0]
           == '\r') {
132         // Check comments or empty lines
133         continue;
134       }
135
136       // Generate a vertex position
137       if (nodes_parse) {
138         std::vector<std::string> spos;
139         vec3 vpos;
140
141         algorithms::split(algorithms::tail(curline), spos, " ");
142
143         vpos[0] = std::stod(spos[0]);
144         vpos[1] = std::stod(spos[1]);
145         vpos[2] = std::stod(spos[2]);
146         Nodes.push_back(vpos);
147       }
148
149       // Generate a face (vertices & indices)
150       if (elements_parse) {
151         // Generate the triangle vertices from the elements
152         vec3 iVerts[3];
153         GenVerticesFromRawRDF( Nodes, curline, iVerts );
154
155         // Get the triangle friction from current line
156         std::vector<std::string> curlinevec;
157         algorithms::split(curline, curlinevec, " ");
158         real_type iFriction = std::stod(curlinevec[4]);
159
160         // Create a shared pointer for the last triangle and push
              it in the pointer vector
```

```
161         PtrTriangleVec.push_back(std::shared_ptr<Triangle3D>(new
                Triangle3D(iVerts,iFriction)));
162       }
163     }
164
165 #ifdef RDF_CONSOLE_OUTPUT
166     std::cout << std::endl;
167 #endif
168
169     file.close();
170
171     if (PtrTriangleVec.empty()) {
172       perror("Loaded mesh is empty");
173       return false;
174     } else {
175       // Update the local intersected triangles list
176       updatePtrBBox();
177       PtrTree->build(PtrBBoxVec);
178       return true;
179     }
180   }
181
182   // Generate vertices from a list of positions face line.
183   void
184   MeshSurface::GenVerticesFromRawRDF(
185     std::vector<vec3> const & iNodes,
186     std::string       const & icurline,
187     vec3                      oVerts[3]
188   ) {
189     std::vector<std::string> svert;
190     vec3                     vVert;
191     algorithms::split( icurline, svert, " " );
192
193     int_type control_size = int(svert.size() - 4);
194     for ( int i = 1; i < int(svert.size() - control_size); ++i )
            {
195       // Calculate and store the vertex
```

```
196       vVert = algorithms::getElement(iNodes, svert[i]);
197       oVerts[i-1] = vVert; // CONTROLLARE i <=
              3!!!!!!!!!!!!!!!!!!!!!!
198     }
199   }
200
201   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          - - - - - - - -
202   // namespace algorithms
203   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          - - - - - - - -
204
205   //! Holds all of the algorithms needed for the mesh processing
206   namespace algorithms {
207
208     //! Split a string into a string array at a given token.
209     void
210     split(
211       std::string const        & in,    //!< Input string
212       std::vector<std::string> & out,   //!< Output string vector
213       std::string const        & token  //!< Token
214     ) {
215       out.clear();
216
217       std::string temp;
218
219       for ( int i = 0; i < int(in.size()); ++i ) {
220         std::string test = in.substr(i, token.size());
221         if (test == token) {
222           if (!temp.empty()) {
223             out.push_back(temp);
224             temp.clear();
225             i += (int)token.size() - 1;
226           } else {
227             out.push_back("");
228           }
229         } else if (i + token.size() >= in.size()) {
```

```
230           temp += in.substr(i, token.size());
231           out.push_back(temp);
232           break;
233         } else {
234           temp += in[i];
235         }
236       }
237     }
238
239     //! Get tail of string after first token and possibly
          following spaces.
240     std::string
241     tail(std::string const & in ) {
242       size_t token_start = in.find_first_not_of(" \t");
243       size_t space_start = in.find_first_of(" \t", token_start);
244       size_t tail_start  = in.find_first_not_of(" \t",
            space_start);
245       size_t tail_end    = in.find_last_not_of(" \t");
246       if (tail_start != std::string::npos && tail_end != std::
            string::npos) {
247         return in.substr(tail_start, tail_end - tail_start + 1);
248       } else if (tail_start != std::string::npos) {
249         return in.substr(tail_start);
250       }
251       return "";
252     }
253
254     //! Get first token of string.
255     std::string
256     firstToken( std::string const & in ) {
257       if (!in.empty()) {
258         size_t token_start = in.find_first_not_of(" \t\r\n");
259         if (token_start != std::string::npos) {
260           size_t token_end = in.find_first_of(" \t\r\n",
                token_start);
261           if (token_end != std::string::npos) {
262             return in.substr(token_start, token_end - token_start
```

```
                );
263           } else {
264             return in.substr(token_start);
265           }
266         }
267       }
268       return "";
269     }
270
271
272     //! Get element at given index position.
273     template<typename T>
274     T const &
275     getElement(
276       std::vector<T> const & elements,  //!< Elements vector
277       std::string    const & index         //!< Index position
278     ) {
279       // std::cout << "Index: " << index << std::endl;
280       int_type id = std::stoi(index);
281       if ( id < 0 ) std::cerr << "ELEMENTS indexes cannot be
            negative\n";
282       return elements[id - 1];
283     }
284
285   }  // namespace algorithms
286 }  // namespace RDF
287
288 ///
289 /// eof: MeshRDF.hh
290 ///
```

A.3 PatchTire.hh

```
1 ///
2 /// file: PatchTire.hh
3 ///
4
```

```
5 /*!
6
7 \mainpage
8
9 Contact Patch Evaluation
10 ========================
11
12 Master's Thesis
13
14 Algorithm for Tire Contact Patch Evaluation in Soft Real-Time
15
16 Academic Year 2019 · 2020
17
18 Graduant:
19 --------
20
21 Davide Stocco\n
22 Department of Industrial Engineering\n
23 University of Trento\n
24 davide.stocco@studenti.unitn.it
25
26 Supervisor:
27 ----------
28
29 Prof. Enrico Bertolazzi\n
30 Department of Industrial Engineering\n
31 University of Trento\n
32 enrico.bertolazzi@unitn.it
33
34 */
35
36 #pragma once
37
38 #include <Eigen/Dense>  // Eigen linear algebra Library
39 #include <chrono>       // chrono - STD Time Measurement Library
40 #include <cmath>        // Math.h - STD math Library
41 #include <fstream>      // fStream - STD File I/O Library
```

```
42 #include <iostream>     // Iostream - STD I/O Library
43 #include <string>       // String - STD String Library
44 #include <vector>       // Vector - STD Vector/Array Library
45
46 #include "RoadRDF.hh"  // RDF file extention Loader
47
48 //! Tire computations routine
49 namespace PatchTire {
50
51   typedef double real_type;  //!< Real number type
52   typedef int    int_type;   //!< Integer number type
53
54   typedef Eigen::Vector2d vec2;  //!< 2D vector type
55   typedef Eigen::Vector3d vec3;  //!< 3D vector type
56   typedef Eigen::Vector4d vec4;  //!< 2D vector type
57   typedef Eigen::Matrix3d mat3;  //!< 3x3 matrix type
58   typedef Eigen::Matrix4d mat4;  //!< 4x4 matrix type
59
60   typedef std::basic_ostream<char> ostream_type;  //!< Output
       stream type
61
62   static real_type quiteNaN =
63       std::numeric_limits<real_type>::quiet_NaN();  //!< Not-a-
           Number type
64
65   static mat3 mat3_NaN = mat3::Constant(quiteNaN); //!< Not-a-
       Number 3x3 matrix type
66   static vec3 vec3_NaN = vec3::Constant(quiteNaN); //!< Not-a-
       Number 3D vactor type
67
68   real_type const epsilon = std::numeric_limits<real_type>::
       epsilon();
69
70   //! Class for handling tire disks.
71   class Disk {
72   private:
73     vec2              OriginXZ;       //!< X0,Z0 origin vector
```

```
74    real_type         Y;              //!< Y0 (= D) origin
          coordinate or offset from center
75    real_type         R;              //!< Disk radius
76    std::vector<vec2> PointsSequence; //!< Sampled terrain points
          vector
77    real_type         PatchLength;    //!< Local patch length
78
79    Disk const & operator = ( Disk const & ); // operatore di
          copia
80
81  public:
82    //! Variable set constructor for orientation object.
83    Disk(
84      vec2 const & _OriginXZ,  //!< X0,Z0 origin coordinate
85      real_type    _Y,    //!< Y0 (= D) origin coordinate
86      real_type    _R     //!< Disk radius
87    ) {
88      OriginXZ = _OriginXZ;
89      Y        = _Y;
90      R        = _R;
91    }
92
93    //! Copy class Disk.
94    void
95    set( Disk const & obj ) {
96      this->OriginXZ       = obj.OriginXZ;
97      this->Y              = obj.Y;
98      this->R              = obj.R;
99      this->PatchLength    = obj.PatchLength;
100     this->PointsSequence = obj.PointsSequence;
101   }
102
103   //! Set origin vector.
104   void
105   setOriginXZ( vec2 const & _OriginXZ ) { OriginXZ = _OriginXZ;
          }
106
```

```
107     //! Get origin vector XZ-axes coordinates.
108     vec2 const &
109     getOriginXZ(void) const { return OriginXZ; }
110
111     //! Get origin Y-axis coordinate.
112     real_type
113     getOriginY(void) const { return Y; }
114
115     //! Get the overall point sequance length inside the disk.
116     real_type
117     getPatchLength(void) const { return PatchLength; }
118
119     //! Set sampled terrain points vector and calculate area
            between disk and
120     //! segment.
121     void
122     setPointsSequence( std::vector<vec2> const & _PointsSequence
            ) {
123       PointsSequence.clear();
124       PointsSequence = _PointsSequence;
125       updatePatchLength();
126     }
127
128   private:
129     //! Calculate area between disk and segment.
130     void updatePatchLength(void);
131   };
132
133   //! Class for tire shadow bounding box
134   class Shadow {
135   private:
136     real_type Xmin;                                  //!< Xmin
            shadow domain point
137     real_type Ymin;                                  //!< Ymin
            shadow domain point
138     real_type Xmax;                                  //!< Xmax
            shadow domain point
```

```
139   real_type Ymax;                                    //!< Ymax
          shadow domain point
140   std::vector<G2lib::BBox::PtrBBox> PtrBBoxVec;  //!< Bounding
          boxes pointers
141   G2lib::AABBtree::PtrAABB PtrTree =
142       std::make_shared<G2lib::AABBtree>();  //!< Mesh tree
              pointer
143
144   Shadow( Shadow const & ); // costruttore di copia
145   Shadow const & operator = ( Shadow const & ); // operatore di
          copia
146
147 public:
148   //! Default constructor for orientation object.
149   Shadow() {}
150
151   //! Variable set constructor for orientation object.
152   Shadow(
153     real_type _Xmin,  //!< Xmin shadow domain point
154     real_type _Ymin,  //!< Ymin shadow domain point
155     real_type _Xmax,  //!< Xmax shadow domain point
156     real_type _Ymax   //!< Ymax shadow domain point
157   ) {
158     Xmin = _Xmin;
159     Ymin = _Ymin;
160     Xmax = _Xmax;
161     Ymax = _Ymax;
162     updatePtrBBox();
163     PtrTree->build(PtrBBoxVec);
164   }
165
166   //! Set class member Xmin, second argument boolean allows to
          update the
167   //! AABB tree of the tire shadow after the change of domain.
168   void
169   setXmin(
170     real_type const _Xmin,              //!< Xmin shadow domain
```

```
          point
171       bool            updateTree = false  //!< Optional boolean
             for updating the tire shadow AABB tree
172     ) {
173       Xmin = _Xmin;
174       if ( updateTree == true ) {
175         updatePtrBBox();
176         PtrTree->build(PtrBBoxVec);
177       }
178     }
179
180     //! Set class member Ymin, second argument boolean allows to
           update the
181     //! AABB tree of the tire shadow.
182     void
183     setYmin(
184       real_type const _Ymin,              //!< Ymin shadow domain
             point
185       bool            updateTree = false  //!< Optional boolean
             for updating the tire shadow AABB tree
186     ) {
187       Ymin = _Ymin;
188       if (updateTree == true) {
189         updatePtrBBox();
190         PtrTree->build(PtrBBoxVec);
191       }
192     }
193
194     //! Set class member Xmax, second argument boolean allows to
           update the
195     //! AABB tree of the tire shadow.
196     void
197     setXmax(
198       real_type const _Xmax,              //!< Xmax shadow domain
             point
199       bool            updateTree = false  //!< Optional boolean
             for updating the tire shadow AABB tree
```

```
200   ) {
201     Xmax = _Xmax;
202     if (updateTree == true) {
203       updatePtrBBox();
204       PtrTree->build(PtrBBoxVec);
205     }
206   }
207
208   //! Set class member Ymax, second argument boolean allows to
          update the
209   //! AABB tree of the tire shadow.
210   void
211   setYmax(
212     real_type const _Ymax,              //!< Ymax shadow domain
            point
213     bool            updateTree = false  //!< Optional boolean
          for updating the tire shadow AABB tree
214   ) {
215     Ymax = _Ymax;
216     if (updateTree == true) {
217       updatePtrBBox();
218       PtrTree->build(PtrBBoxVec);
219     }
220   }
221
222   //! Get tire shadow AABB tree.
223   G2lib::AABBtree::PtrAABB
224   getAABBPtr(void) const
225   { return PtrTree; }
226
227   //! Get Xmin shadow domain point.
228   real_type
229   getXmin() const { return Xmin; }
230
231   //! Get Ymin shadow domain point.
232   real_type
233   getYmin() const { return Ymin; }
```

```
234
235     //! Get Xmax shadow domain point.
236     real_type
237     getXmax() const { return Xmax; }
238
239     //! Get Ymax shadow domain point.
240     real_type
241     getYmax() const { return Ymax; }
242
243     //! Clear the tire shadow domain.
244     void
245     clear(void) {
246       Xmin = quiteNaN;
247       Ymin = quiteNaN;
248       Xmax = quiteNaN;
249       Ymax = quiteNaN;
250     }
251
252     //! Print tire shadow bounding box vertices.
253     void
254     print( ostream_type & stream ) const {
255       stream
256         << "BBOX (xmin,ymin,xmax,ymax) = ( " << Xmin << ", " <<
              Ymin
257         << ", " << Xmax << ", " << Ymax << " )\n";
258     }
259
260     //! Get the mesh G2lib bounding boxes pointers vector.
261     std::vector<G2lib::BBox::PtrBBox> const &
262     getPtrBBox() const
263     { return PtrBBoxVec; }
264
265   private:
266     //! Update the mesh G2lib bounding boxes pointers vector.
267     void
268     updatePtrBBox(void) {
269       PtrBBoxVec.clear();
```

```
270       PtrBBoxVec.push_back(
271         G2lib::BBox::PtrBBox(
272           new G2lib::BBox(Xmin, Ymin, Xmax, Ymax, 0, 0)
273         )
274       );
275     }
276   };
277
278   //! Class for handling tire origin, yaw angle and camber angle
279   class Orientation {
280   private:
281     vec3      Origin = vec3_NaN;  //!< Origin position (default
          NaN)
282     mat3      RotMat = mat3_NaN;  //!< 3x3 rotation matrix (
          default NaN)
283     real_type Yaw;      //!< Yaw angle [rad]
284     real_type Camber;   //!< Camber angle [rad]
285     vec3      Xversor;  //!< X-axis versor
286     vec3      Yversor;  //!< Y-axis versor
287     vec3      Zversor;  //!< Z-axis versor
288
289     Orientation( Orientation const & ) = delete; // costruttore
          di copia
290     Orientation & operator = ( Orientation const & ) = delete; //
           operatore di copia
291
292   public:
293     //! Default constructor for orientation object.
294     Orientation() {}
295
296     //! Variable set constructor for orientation object.
297     Orientation(
298       vec3    & _Origin,      //!< Origin position
299       real_type _Yaw,    //!< Yaw angle [rad]
300       real_type _Camber  //!< Camber angle [rad]
301     ) {
302       Origin = _Origin;
```

```
303       Yaw    = _Yaw;
304       Camber = _Camber;
305       updateRotMat();
306     }
307
308     //! Check if the object is empty
309     bool
310     isEmpty(void) {
311       if ( Origin != Origin || RotMat != RotMat){
312         return true;
313       } else {
314         return false;
315       }
316     }
317
318     //! Get current rotation matrix.
319     mat3 const &
320     getRotMat(void) const { return RotMat; }
321
322     //! Get current X-axis versor.
323     vec3 const &
324     getX(void) const { return Xversor; }
325
326     //! Get current Y-axis versor.
327     vec3 const &
328     getY(void) const { return Yversor; }
329
330     //! Get current Z-axis versor.
331     vec3 const &
332     getZ(void) const { return Zversor; }
333
334     //! Get camber angle.
335     real_type
336     getCamber(void) const { return Camber; }
337
338     //! Get yaw angle.
339     real_type
```

```
340     getYaw(void) const { return Yaw; }
341
342     //! Get origin vector.
343     vec3
344     getOrigin(void) const { return Origin; }
345
346     //! Set camber angle.
347     void
348     setCamber( real_type const _Camber ) {
349       Camber = _Camber;
350       updateRotMat();
351     }
352
353     //! Set yaw angle.
354     void
355     setYaw( real_type const _Yaw ) {
356       Yaw = _Yaw;
357       updateRotMat();
358     }
359
360     //! Set origin.
361     void
362     setOrigin( vec3 const & _Origin ) { Origin = _Origin; }
363
364     //! Set rotation matrix.
365     void
366     setRotMat( mat3 const & _RotMat ) { RotMat = _RotMat; }
367
368     //! Copy class Orientation.
369     void set( Orientation const & in ) {
370       this->Origin = in.Origin;
371       this->Yaw    = in.Yaw;
372       this->Camber = in.Camber;
373       updateRotMat();
374     }
375
376   private:
```

```
377   //! Get axes versors components.
378   void
379   updateRotMat(void);
380 };
381
382 //! Class for handling the patch evaluation precision.
383 class SolverPrecision {
384 private:
385   int_type Xdiv = 20;  //!< Number of divisions in X-axis
386   int_type Ydiv = 20;  //!< Number of divisions in Y-axis
387
388   SolverPrecision( SolverPrecision const & ) = delete; //
        costruttore di copia
389   SolverPrecision & operator = ( SolverPrecision const & ) =
        delete; // operatore di copia
390
391 public:
392   //! Default constructor for SolverPrecision object.
393   SolverPrecision() {}
394
395   //! Variable set constructor for SolverPrecision object.
396   SolverPrecision(
397     int_type _Xdiv, //!< Number of divisions in X-axis
398     int_type _Ydiv  //!< Number of divisions in Y-axis
399   ) {
400     Xdiv = _Xdiv;
401     Ydiv = _Ydiv;
402   }
403
404   //! Get number of divisions in X-axis.
405   int_type
406   getXdiv(void) const { return Xdiv; }
407
408   //! Get number of divisions in Y-axis.
409   int_type
410   getYdiv(void) const { return Ydiv; }
411
```

```
412     //! Set number of divisions in X-axis.
413     void
414     setXdiv( int_type const _Xdiv ) { Xdiv = _Xdiv; }
415
416     //! Set number of divisions in Y-axis.
417     void
418     setYdiv( int_type const _Ydiv ) { Ydiv = _Ydiv; }
419
420     //! Copy the solver precision class.
421     void
422     set( SolverPrecision const & in ) {
423       this->Xdiv = in.Xdiv;
424       this->Ydiv = in.Ydiv;
425     }
426   };
427
428   //! Class for handling ETRTO tire data.
429   class ETRTO {
430   private:
431     real_type SectionWidth;  //!< Tire section width[mm]
432     real_type AspectRatio;   //!< Tire aspect ratio [%]
433     real_type RimDiameter;   //!< Rim diameter [in]
434     real_type SidewallHeight;   //!< Rim diameter [in]
435     real_type TireDiameter;   //!< Rim diameter [in]
436
437     ETRTO( ETRTO const & ); // costruttore di copia
438
439   public:
440     //! Default constructor for orientation object.
441     ETRTO() {}
442
443     //! Variable set constructor for tire object.
444     ETRTO(
445       real_type _SectionWidth,  //!< Tire section width[mm]
446       real_type _AspectRatio,   //!< Tire aspect ratio [%]
447       real_type _RimDiameter    //!< Rim diameter [in]
448     ) {
```

```
449       SectionWidth = _SectionWidth;
450       AspectRatio  = _AspectRatio;
451       RimDiameter  = _RimDiameter;
452       calcSidewallHeight();
453       calcTireDiameter();
454     }
455
456     //! Operator =.
457     ETRTO const &
458     operator = ( ETRTO const & rhs ) {
459       this->SectionWidth = rhs.SectionWidth;
460       this->AspectRatio  = rhs.AspectRatio;
461       this->RimDiameter  = rhs.RimDiameter;
462       calcSidewallHeight();
463       calcTireDiameter();
464       return *this;
465     }
466
467     //! Get sidewall height [m].
468     real_type
469     getSidewallHeight(void) const
470     { return SidewallHeight; }
471
472     //! Get external tire diameter [m].
473     real_type
474     getTireDiameter(void) const
475     { return TireDiameter; }
476
477     //! Get section width [m].
478     real_type
479     getSectionWidth(void) const
480     { return SectionWidth / 1000.0; }
481
482     //! Display tire data.
483     void
484     print( ostream_type & stream ) const {
485       stream
```

```
486         << "Current Tire Data:\n"
487         << "\tSection width = " << SectionWidth << " mm\n"
488         << "\tAspect ratio  = " << AspectRatio << " %\n"
489         << "\tRim diameter  = " << RimDiameter << " in\n"
490         << "\tS.wall Height = " << getSidewallHeight() * 1000 <<
               " mm\n"
491         << "\tTire diameter = " << getTireDiameter() * 1000 << "
               mm\n\n";
492     }
493
494   private:
495     //! Calculate sidewall height [m].
496     void
497     calcSidewallHeight(void)
498     { SidewallHeight = SectionWidth / 1000.0 * AspectRatio / 100;
          }
499
500     //! Calculate external tire diameter [m].
501     void
502     calcTireDiameter(void)
503     { TireDiameter = RimDiameter * 0.0254 + getSidewallHeight() *
          2.0; }
504   };
505
506   //! Algorithms for tire computations routine.
507   namespace algorithms {
508
509     //! Check if a ray hits a triangle object through Möller-
          Trumbore
510     //! intersection algorithm.
511     bool
512     rayIntersectsTriangle(
513       vec3 const      & RayOrigin,        //!< Ray origin
             position
514       vec3 const      & RayVector,        //!< Ray direction
             vector
515       RDF::Triangle3D & Triangle,         //!< Triangle object
```

```
516       vec3            & IntersectionPoint //!< Intersection point
517     );
518
519     //! Find the intersection points between a circle and a line
            segment.
520     //! Output integer gives the number of intersection points.
521     int_type
522     segmentIntersectsCircle(
523       vec2 const & Origin,         //!< Circle origin position
524       real_type    R,              //!< Circle radius
525       vec2 const & Point_1,        //!< Line segment point 1
526       vec2 const & Point_2,        //!< Line segment point 2
527       vec2       & Intersection_1, //!< Intersection point 2
528       vec2       & Intersection_2  //!< Intersection point 2
529     );
530
531     //! Check if a point lays inside or outside a circumfererence
            .
532     //! If output bool is true the point is inside the
            circumfererence,
533     //! otherwise it is outside.
534     bool
535     pointInsideCircle(
536       vec2 const & Origin,  //!< Circle origin position
537       real_type    R,    //!< Circle radius
538       vec2 const & Point    //!< Query point
539     );
540
541     //! Check if a point lays inside or outside a line segment.
542     //! Warning: The point query point must be on the same rect
            of the line
543     //! segment.
544     bool
545     pointOnSegment(
546       vec2 const & Point_1,  //!< Line segment point 1
547       vec2 const & Point_2,  //!< Line segment point 2
548       vec2 const & Point     //!< Query point
```

```
549     );
550
551     //! Update the rectangular shadow domain of the tire in X and
            Y-axis
552     void
553     updateShadow(
554       Shadow            & iShadow,   //!< Shadow bounding box
              object
555       ETRTO       const & TireDenom, //!< ETRTO tire denomination
              object
556       Orientation const & Orient     //!< Orientation object
557       );
558
559     //! Update the local intersected triangles list
560     std::vector<std::shared_ptr<RDF::Triangle3D> >
561     updateIntersectionList(
562       Shadow            const & iShadow, //!< Shadow bounding box
              object
563       RDF::MeshSurface        & iMesh    //! RDF mesh object
              pointer
564     );
565
566   }  // namespace algorithms
567
568   //! Class for evaluating the contact patch
569   class TireDisks {
570   private:
571     std::vector<Disk> DiskVector;  //! Disks instance vector
572     ETRTO             TireDenom;   //!< ETRTO tire denomination
          object
573     SolverPrecision   Precision;   //!< Solver precision object
574     Orientation       Orient;      //!< Orientation object
575     Shadow            iShadow;     //!< Shadow bounding box
          object
576     RDF::MeshSurface  iMesh;       //! RDF mesh object pointer
577     std::vector<std::shared_ptr<RDF::Triangle3D> >
          intersectionTriPtr; //!< Local intersected triangles
```

```
      vector
578   Eigen::MatrixXd      intersectionGrid;  //! Local sampling
      grid
579
580   TireDisks( TireDisks const & ); // costruttore di copia
581   TireDisks const & operator = ( TireDisks const & ); //
      operatore di copia
582
583 public:
584   //! Variable set constructor for tire object.
585   TireDisks(
586     ETRTO           const & _TireDenom,
587     SolverPrecision const & _Precision
588   ) {
589     TireDenom        = _TireDenom;
590     Precision.set(_Precision);
591     intersectionGrid = Eigen::MatrixXd( Precision.getYdiv() +
        1, Precision.getXdiv() + 1 );  // Y sampling (rows), X
        sampling (columns)
592     std::vector<real_type> Dvec = offsetDisk(Precision.getYdiv
        ());
593     real_type R                 = TireDenom.getTireDiameter() /
         2.0;
594     for ( int_type i = 0; i <= Precision.getYdiv(); ++i )
595       DiskVector.push_back( Disk( vec2(0, 0), Dvec[i], R) );
596   }
597
598   //! Set the tire orientation
599   void
600   setOrientation( Orientation const & _Orient ) {
601     Orient.set(_Orient);
602     algorithms::updateShadow(iShadow, TireDenom, Orient);
603     intersectionTriPtr = algorithms::updateIntersectionList(
        iShadow, iMesh);
604   }
605
606   //! Set the terrain mesh
```

```
607     void
608     setMesh( RDF::MeshSurface const & _iMesh ) {
609       iMesh.set(_iMesh);
610     }
611
612     //! Get orientation information
613     Orientation const &
614     getOrientation(void) const
615     { return Orient; }
616
617     //! Get orientation information
618     SolverPrecision const &
619     getPrecision(void) const
620     { return Precision; }
621
622     //! Get grid step on X-axis
623     real_type
624     getXstep(void) const
625     { return TireDenom.getTireDiameter() / Precision.getXdiv(); }
626
627     //! Get grid step on Y-axis
628     real_type
629     getYstep(void) const
630     { return TireDenom.getSectionWidth() / Precision.getYdiv(); }
631
632     //! Get i-th intantiated disk
633     Disk const &
634     getithDisk( int_type const i ) const
635     { return DiskVector[i]; }
636
637     //! Perform triangles sampling
638     void
639     gridSampling( bool print = false );
640
641     //! Find single point intersection between tire and mesh (
            Pacejka Single
642     //! Contact Point) with AABB tree
```

```
643     real_type
644     MF_Pacejka_SCP( bool print = false );
645
646     //! Evaluate the contact patch between tire and mesh (Magic
           Formula
647     //! Swift Contact Patch Evaluation)
648     real_type
649     MF_Swift_PE( bool print = false );
650
651     //! Evaluate the local effective road plane (Magic Formula
652     //! Swift Effective road plane)
653     std::vector<real_type>
654     MF_Swift_ERP( bool print = false );
655
656     void
657     Move(
658       vec3      const & start,   //! Starting position
659       vec3      const & arrival, //! Arrival position
660       real_type const & freq,    //!< Sampling frequency [Hz]
661       real_type const & speed,   //!< Tire speed [m/s]
662       bool              print
663     );
664
665   private:
666     std::vector<real_type>
667     offsetDisk( int_type const n );
668
669     //! Find nearest intersection to origin
670     real_type
671     calculateMagnitude( std::vector<vec3> const &
           IntersectionPointVec );
672   };
673
674   //! Class for evaluating the contact patch evaluation with
       Magic Formula.
675   class TireMF {
676   private:
```

```
677     ETRTO              TireDenom; //!< ETRTO tire denomination
            object
678     Orientation        Orient;    //!< Orientation object
679     Shadow             iShadow;   //!< Shadow bounding box object
680     RDF::MeshSurface   iMesh;     //! RDF mesh object pointer
681     vec3               Qvec[4];   //! Local sampling grid
682     std::vector<std::shared_ptr<RDF::Triangle3D> >
            intersectionTriPtr; //!< Local intersected triangles
            vector
683
684     TireMF( TireMF const & ); // costruttore di copia
685     TireMF const & operator = ( TireMF const & ); // operatore di
            copia
686
687   public:
688     //! Variable set constructor for tire object.
689     TireMF(
690       ETRTO             const & _TireDenom,
691       RDF::MeshSurface const & _iMesh
692     ) {
693       TireDenom = _TireDenom;
694       iMesh.set(_iMesh);
695     }
696
697     //! Set the tire orientation.
698     void
699     setOrientation( Orientation const & _Orient ) {
700       Orient.set(_Orient);
701     }
702
703     //! Get orientation information.
704     Orientation const &
705     getOrientation(void) const
706     { return Orient; }
707
708     //! Set camber angle.
709     void
```

```
710     setCamber( real_type const Camber ) { Orient.setCamber(Camber
            ); }
711
712     //! Set yaw angle.
713     void
714     setYaw( real_type const Yaw ) { Orient.setYaw(Yaw); }
715
716     //! Set camber angle.
717     void
718     setOrigin( vec3 const & Origin ) { Orient.setOrigin(Origin);
            }
719
720     //! Set camber angle.
721     void
722     setTotalTM( mat4 const & TM ) {
723       Orient.setOrigin(TM.block<3,1>(0,3));
724       Orient.setRotMat(TM.block<3,3>(0,0));
725     }
726
727     //! Set the terrain mesh.
728     void
729     setMesh( RDF::MeshSurface const & _iMesh ) {
730       iMesh.set(_iMesh);
731     }
732
733     //! Find the normal vector of the local track plane and local
            contact point.
734     std::vector<vec3>
735     MF_62( bool print = false );
736
737     //! Move the current tire from a starting point to an ending
            point.
738     void
739     Move(
740       vec3      const & start,   //! Starting position
741       vec3      const & arrival, //! Arrival position
742       real_type const & freq,    //!< Sampling frequency [Hz]
```

```
743       real_type const & speed,   //!< Tire speed [m/s]
744       bool              print
745     );
746
747   private:
748     //! Perform triangles sampling.
749     void
750     terrainSampling(void);
751   };
752
753 }  // namespace PatchTire
754
755 ///
756 /// eof: PatchTire.hh
757 ///
```

A.4 PatchTire.cc

```
1 #include "PatchTire.hh"  // RDF file extention Loader
2
3 //! Tire computations routine
4 namespace PatchTire {
5
6   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - -
7   // class Disk
8   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - -
9
10  //! Calculate area between disk and segment
11  void
12  Disk::updatePatchLength(void) {
13    // Reset class variable
14    PatchLength = 0.0;
15
16    for ( unsigned i = 0; i < PointsSequence.size() - 1; ++i ) {
17      vec2     Intersection_1, Intersection_2;
```

```
18      vec2     Point_1 = PointsSequence[i];
19      vec2     Point_2 = PointsSequence[i + 1];
20      int_type Type    = algorithms::segmentIntersectsCircle(
21        OriginXZ, R, Point_1, Point_2, Intersection_1,
              Intersection_2
22      );
23      // std::cout << " " << Type;
24      if ( Type == 0 ) {
25        // No contact points, the line segment is not into the
              Disk
26        continue;
27      } else if (Type == 1) {
28        // Tangent, no length added
29        continue;
30      } else if (Type == 2) {
31        // Check whether the two segment points are into the
              circle
32        bool Pose_pt1 = algorithms::pointInsideCircle(OriginXZ, R
              , Point_1);
33        bool Pose_pt2 = algorithms::pointInsideCircle(OriginXZ, R
              , Point_2);
34
35          // Check whether the two intersection points are onto
                the line segment
36        bool Pose_int1 = algorithms::pointOnSegment(Point_1,
              Point_2, Intersection_1);
37        bool Pose_int2 = algorithms::pointOnSegment(Point_1,
              Point_2, Intersection_2);
38
39        // Cases
40        // Line segment Point_1 and line segment Point_2 into the
               circle,
41        // intersection points oustide line segment
42        if ( Pose_pt1 && Pose_pt2 && !Pose_int1 && !Pose_int2 ) {
43          PatchLength += (Point_2 - Point_1).norm();
44          continue;
45        }
```

```
46        // Intersection points into the line segment and line
              segment points
47        // outside the circle
48        else if ( !Pose_pt1 && !Pose_pt2 && Pose_int1 &&
              Pose_int2 ) {
49          PatchLength += (Intersection_2 - Intersection_1).norm()
                ;
50          continue;
51        }
52        // Line segment Point_1 outside the circle, line segment
              Point_2
53        // inside the circle
54        else if ( !Pose_pt1 && Pose_pt2 ) {
55          if ( Pose_int1 && !Pose_int2 ) {
56            // Add length from Intersection_1 to Point_2
57            PatchLength += (Intersection_1 - Point_2).norm();
58            continue;
59          } else if ( Pose_int2 && !Pose_int1 ) {
60            // Add length from Intersection_2 to Point_2
61            PatchLength += (Intersection_2 - Point_2).norm();
62            continue;
63          }
64        }
65        // Line segment Point_1 inside the circle, line segment
              Point_2
66        // outside the circle
67        else if ( Pose_pt1 && !Pose_pt2 ) {
68          if ( Pose_int1 && !Pose_int2 ) {
69            // Add length from Intersection_1 to Point_1
70            PatchLength += (Intersection_1 - Point_1).norm();
71            continue;
72          } else if ( !Pose_int1 && Pose_int2 ) {
73            // Add length from Intersection_2 to Point_1
74            PatchLength += (Intersection_2 - Point_1).norm();
75            continue;
76          }
77        }
```

```
78      }
79    }
80    // std::cout << "Length: " << PatchLength << std::endl;
81  }
82
83  // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - -
84  // class Orientation
85  // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - -
86
87  //! Get axes versors components.
88  void
89  Orientation::updateRotMat(void) {
90    // Transformation matrix for Z-axis rotation
91    mat3 Rot_Z;
92    Rot_Z << cos(Yaw), -sin(Yaw), 0,
93             sin(Yaw),  cos(Yaw), 0,
94                    0,         0, 1;
95    // Transformation matrix for X-axis rotation
96    mat3 Rot_X;
97    Rot_X << 1,           0,            0,
98             0, cos(Camber), -sin(Camber),
99             0, sin(Camber),  cos(Camber);
100
101   //Update Rotation Matrix and Versors
102   RotMat = Rot_Z * Rot_X;
103   Xversor = RotMat * vec3(1.0, 0.0, 0.0);
104   Yversor = RotMat * vec3(0.0, 1.0, 0.0);
105   Zversor = RotMat * vec3(0.0, 0.0, 1.0);
106 }
107
108 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - -
109 // namespace algorithms
110 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - -
```

```
111
112 //! Holds all of the algorithms needed for the contact patch
       evaluation
113 namespace algorithms {
114
115   //! Check if a ray hits a triangle object through Möller-
         Trumbore
116   //! intersection algorithm.
117   bool
118   rayIntersectsTriangle(
119     vec3 const      & RayOrigin,        //!< Ray origin
          position
120     vec3 const      & RayDirection,     //!< Ray direction
          vector
121     RDF::Triangle3D & Triangle,         //! Triangle object
122     vec3            & IntersectionPoint //! Intersection point
123   ) {
124     vec3      E1  = Triangle.getithVertex(1) - Triangle.
          getithVertex(0);
125     vec3      E2  = Triangle.getithVertex(2) - Triangle.
          getithVertex(0);
126     vec3      A   = RayDirection.cross(E2);
127     real_type det = A.dot(E1);
128     real_type t_param;
129
130     if ( det > epsilon ) {
131       vec3      T = RayOrigin - Triangle.getithVertex(0);
132       real_type u = A.dot(T);
133       if ( u < 0.0 || u > det ) return false;
134       vec3      B = T.cross(E1);
135       real_type v = B.dot(RayDirection);
136       if ( v < 0.0 || u + v > det ) return false;
137       t_param = (B.dot(E2))/det;
138     } else if ( det < -epsilon ) {
139       vec3      T = RayOrigin - Triangle.getithVertex(0);
140       real_type u = A.dot(T);
141       if ( u > 0.0 || u < det ) return false;
```

```
142       vec3      B = T.cross(E1);
143       real_type v = B.dot(RayDirection);
144       if ( v > 0.0 || u + v < det ) return false;
145       t_param = (B.dot(E2))/det;
146     } else {
147       return false;
148     }
149     // At this stage we can compute t to find out where the
          intersection
150     // point is on the line
151     if ( t_param >= 0 ) { // ray intersection
152       IntersectionPoint = RayOrigin + RayDirection * t_param;
153       return true;
154     } else {
155       // This means that there is a line intersection on
            negative side
156       return false;
157     }
158   }
159
160   //! Find the intersection points between a circle and a line
        segment.
161   //! Output integer gives the number of intersection points.
162   int_type
163   segmentIntersectsCircle(
164     vec2 const & Origin,
165     real_type    R,
166     vec2 const & Point_1,
167     vec2 const & Point_2,
168     vec2       & Intersect_1,
169     vec2       & Intersect_2
170   ) {
171     real_type t_param;
172
173     vec2      d   = Point_2 - Point_1;
174     vec2      P1O = Point_1 - Origin;
175     real_type A   = d.dot(d);
```

```
176       real_type B   = 2 * d.dot(P1O);
177       real_type C   = P1O.dot(P1O) - R*R;
178       real_type det = B*B - 4 * A * C;
179       if ( A <= epsilon || det < 0 ) {
180         // No real solutions
181         Intersect_1 = vec2(quiteNaN, quiteNaN);
182         Intersect_2 = vec2(quiteNaN, quiteNaN);
183         return 0;
184       } else if ( det == 0.0 ) {
185         // One solution
186         t_param     = -B / (2*A);
187         Intersect_1 = Point_1 + t_param * d;
188         Intersect_2 = vec2(quiteNaN, quiteNaN);
189         return 1;
190       } else {
191         // Two solutions
192         t_param = (-B + std::sqrt(det)) / (2 * A);
193         Intersect_1 = Point_1 + t_param * d;
194         t_param = (-B - std::sqrt(det)) / (2 * A);
195         Intersect_2 = Point_1 + t_param * d;
196         return 2;
197       }
198     }
199
200     //! Check if a point lays inside or outside a circumfererence
          .
201     //! If output bool is true the point is inside the
          circumfererence,
202     //! otherwise it is outside.
203     bool
204     pointInsideCircle(
205       vec2 const & Origin,
206       real_type    R,
207       vec2 const & Point
208     ) {
209       // Compare radius of circle with distance
210       // of its center from given point
```

```
211       vec2 PO = Point - Origin;
212       return PO.dot(PO) <= R*R;
213     }
214
215     //! Check if a point lays inside or outside a line segment.
216     //! Warning: The point query point must be on the same rect
            of the line
217     //! segment.
218     bool
219     pointOnSegment(
220       vec2 const & Point_1,
221       vec2 const & Point_2,
222       vec2 const & Point
223     ) {
224       // A and B are the extremities of the current segment C is
              the point to
225       // check
226
227       // Create the vector AB
228       vec2 AB = Point_2 - Point_1;
229       // Create the vector AC
230       vec2 AC = Point - Point_1;
231
232       // Compute the cross product of AB and AC
233       // Check if the three points are aligned (cross product is
              null)
234       // if ((AB.cross3(AC)).squaredNorm() > epsilon)
235       // return false;
236
237       // Compute the dot product of vectors
238       real_type KAC = AB.dot(AC);
239       if ( KAC      < -epsilon ) return false;
240       if ( abs(KAC) < epsilon  ) return true;
241
242       // Compute the square of the segment lenght
243       real_type KAB = AB.dot(AB);
244       if ( KAC > KAB ) return false;
```

```
245       if ( abs(KAC - KAB) < epsilon ) return true;
246
247       // The point is on the segment
248       return true;
249     }
250
251     //! Update the rectangular shadow domain of the tire in X and
            Y-axis
252     void
253     updateShadow(
254       Shadow            & iShadow,   //!< Shadow bounding box
              object
255       ETRTO       const & TireDenom, //!< ETRTO tire denomination
              object
256       Orientation const & Orient     //!< Orientation object
257       ) {
258       // Calculate maximum covered space
259       real_type diagonal = hypot( TireDenom.getSectionWidth(),
260                                   TireDenom.getTireDiameter() ) /
                                        2;
261
262       // Increment shadow to take in account camber angle
263       real_type inc = 1.1;
264
265       // Set new tire shadow domain
266       iShadow.setXmax(Orient.getOrigin()[0] + inc * diagonal);
267       iShadow.setXmin(Orient.getOrigin()[0] - inc * diagonal);
268       iShadow.setYmax(Orient.getOrigin()[1] + inc * diagonal);
269       iShadow.setYmin(Orient.getOrigin()[1] - inc * diagonal,
              true);
270     }
271
272     //! Update the local intersected triangles list
273     std::vector<std::shared_ptr<RDF::Triangle3D> >
274     updateIntersectionList(
275       Shadow            const & iShadow, //!< Shadow bounding box
              object
```

```
276       RDF::MeshSurface        & iMesh   //! RDF mesh object
              pointer
277       ) {
278       G2lib::AABBtree::VecPairPtrBBox intersectionList;
279       std::vector<std::shared_ptr<RDF::Triangle3D> >
              intersectionTriPtr;
280       (*iMesh.getAABBPtr()).intersect(*iShadow.getAABBPtr(),
              intersectionList);
281       for ( unsigned i = 0; i < intersectionList.size(); ++i ) {
282         intersectionTriPtr.push_back(
283           iMesh.getithTrianglePtr((*intersectionList[i].first).Id
                  ())
284         );
285         //std::cout << (*intersectionList[i].first).Id() << "\t";
286       }
287       return intersectionTriPtr;
288     }
289
290   }  // namespace algorithms
291
292   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          - - - - - - -
293   // class TireDisks
294   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          - - - - - - -
295
296   void
297   TireDisks::gridSampling(bool print) {
298
299     // Orient the sampling
300     mat3 MatrixInv = Orient.getRotMat().inverse();
301
302     // Storing indexers
303     unsigned i = 0;
304     unsigned j = 0;
305     std::vector<vec2> PointsSequence;
306     vec3 IntersectionPoint;
```

```
307
308     for ( real_type y = -TireDenom.getSectionWidth() / 2.0;
309           y <= TireDenom.getSectionWidth() / 2.0;
310           y += getYstep(), ++j ) {
311       for ( real_type x = -TireDenom.getTireDiameter() / 2.0;
312             x <= TireDenom.getTireDiameter() / 2.0;
313             x += getXstep(), ++i ) {
314         // Update ray center in tire coordinates
315         vec3 curCenter = Orient.getOrigin() + MatrixInv * vec3(x,
                y, 0);
316         for ( unsigned t = 0; t < intersectionTriPtr.size(); ++t
               ) {
317           // Find intersection
318           if ( algorithms::rayIntersectsTriangle(
319             Orient.getOrigin(),
320             -Orient.getZ(),  // careful to the minus sign !!!
321             *intersectionTriPtr[t],
322             IntersectionPoint
323           ) ) {
324             // Store results
325             real_type z = (IntersectionPoint - curCenter).norm();
326             intersectionGrid(j, i) = z;  // Y sampling (rows), X
                   sampling (columns)
327             PointsSequence.push_back(vec2(x, z));
328             break;
329           } else {
330             intersectionGrid(j, i) = -1.0;  // Y sampling (rows),
                    X sampling (columns)
331           }
332         }
333       }
334       // Calculate chain length inside the circle
335       DiskVector[j].setPointsSequence(PointsSequence);
336
337       if ( print ) {
338         std::cout
339           << "Disk " << j << " of " << DiskVector.size() - 1
```

```
340           << " -> Chain: ";
341         for ( unsigned k = 0; k < PointsSequence.size(); ++k )
342           std::cout << PointsSequence[k][1] << " ";
343         std::cout << std::endl;
344       }
345       PointsSequence.clear();
346
347       // Update indexer
348       i = 0;
349     }
350     if ( print ) std::cout << std::endl << intersectionGrid <<
          std::endl;
351   }
352
353
354   //! Find single point intersection between tire and mesh (
        Pacejka Single
355   //! Contact Point) with AABB tree
356   real_type
357   TireDisks::MF_Pacejka_SCP(bool print) {
358     // Ray-Triangle intesection Point
359     std::vector<vec3> IntersectionPointVec;
360     real_type Magnitude = quiteNaN;
361
362     vec3 IntersectionPoint;
363     for ( unsigned i = 0; i < intersectionTriPtr.size(); ++i ) {
364       if ( algorithms::rayIntersectsTriangle(
365               Orient.getOrigin(), -Orient.getZ(),  // careful to
                    the minus sign !!!
366               *intersectionTriPtr[i], IntersectionPoint
367            ) ) {
368           // Store intersection point in proper vector
369           IntersectionPointVec.push_back(IntersectionPoint);
370       }
371     }
372     // Find nearest intersection to origin
373     Magnitude = calculateMagnitude(IntersectionPointVec);
```

```
374
375   // Display information
376   if ( print )
377     std::cout
378       << "Single contact point for Pacejka MF -> "
379       << IntersectionPointVec.size() << " intersections found"
380       << "\n\tInt. Point  = X " << IntersectionPoint[0]
381       << ", Y " << IntersectionPoint[1]
382       << ", Z " << IntersectionPoint[2]
383       << "\n\tMagnitude   = " << Magnitude
384       << "\n\n";
385   return Magnitude;
386 }
387
388 //! Evaluate the contact patch between tire and mesh (Magic
      Formula
389 //! Swift Contact Patch Evaluation)
390 real_type
391 TireDisks::MF_Swift_PE(bool print) {
392   real_type PatchArea = 0.0;
393   real_type Ystep     = getYstep();
394   for ( unsigned i = 0; i < DiskVector.size(); ++i )
395     PatchArea += DiskVector[i].getPatchLength() * Ystep;
396   if ( print )
397     std::cout
398       << "Contact Patch Area for Swift MF -> "
399       << PatchArea * pow(10, 6) << " mm^2\n\n";
400   return PatchArea;
401 }
402
403 //! Evaluate the local effective road plane (Magic Formula
404 //! Swift Effective road plane)
405 std::vector<real_type>
406 TireDisks::MF_Swift_ERP( bool print ) {
407   int_type nrows = int_type( intersectionGrid.rows() );
408   int_type ncols = int_type( intersectionGrid.cols() );
409
```

```
410   real_type angle_RotY  = 0.0;
411   real_type deltaY_RotY = 0.0;
412
413   real_type angle_RotX  = 0.0;
414   real_type deltaY_RotX = 0.0;
415
416   real_type deltaX_RotY = TireDenom.getTireDiameter();
417   real_type deltaX_RotX = TireDenom.getSectionWidth();
418
419   for ( int_type i = 0; i < nrows; ++i ) {
420     deltaY_RotY = intersectionGrid(i, 0) - intersectionGrid(i,
          ncols - 1);
421     angle_RotY += atan2(deltaY_RotY, deltaX_RotY);
422   }
423   angle_RotY /= nrows;
424
425   for (int_type i = 0; i < ncols; i++) {
426     deltaY_RotX = intersectionGrid(0, i) - intersectionGrid(
          nrows - 1, i);
427     angle_RotX += atan2(deltaY_RotX, deltaX_RotX);
428   }
429   angle_RotX /= ncols;
430
431   if ( print )
432     std::cout
433       << "Effective Road Plane for Swift MF -> "
434       << "X: " << angle_RotX * 180 / G2lib::m_pi << "°, "
435       << "Y: " << angle_RotY * 180 / G2lib::m_pi << "°\n\n";
436
437   return std::vector<real_type>(angle_RotX, angle_RotY);
438 }
439
440 void
441 TireDisks::Move(
442   vec3      const & start,    //! Starting position
443   vec3      const & arrival,  //! Arrival position
444   real_type const & freq,     //!< Sampling frequency [Hz]
```

```
445     real_type const & speed,    //!< Tire speed [m/s]
446     bool              print
447   ) {
448     // Set current position
449     vec3 curpos = start;
450
451     // Set and initialize orientation
452     real_type Yaw    = 0;
453     real_type Camber = 0;
454
455     real_type nstep = (arrival - start).norm() / speed * freq;
456     vec3 step       = (arrival - start) / nstep;
457
458     // Start chronometer
459     auto start_move = std::chrono::system_clock::now();
460
461     for ( unsigned i = 0; i < nstep; ++i ) {
462       // Set current orientation
463       Orientation Pose(curpos, Yaw, Camber);
464       setOrientation(Pose);
465
466       // Evaluate Single Contact Point
467       MF_Pacejka_SCP(print);
468
469       // Perform grid sampling
470       gridSampling(false);
471
472       // Evaluate Patch
473       MF_Swift_PE(print);
474
475       // Evaluate Effective Road Plane
476       MF_Swift_ERP(print);
477
478       // Update current position
479       curpos += step;
480     }
481     // Stop chronometer
```

```
482   auto end_move = std::chrono::system_clock::now();
483
484   // This constructs a duration object using milliseconds
485   auto elapsed_move = std::chrono::duration_cast<std::chrono::
          microseconds>(
486         end_move - start_move);
487   std::cout
488     << "Execution time = " << elapsed_move.count() / 1000.0 <<
          " ms\n"
489     << "Step execution time = "
490     << (elapsed_move.count() / 1000.0) / nstep << " ms\n";
491 }
492
493   std::vector<real_type>
494   TireDisks::offsetDisk( int_type n ) {
495     std::vector<real_type> offsetVec;
496     for ( int_type i = 0; i < n; ++i )
497       // Index from Y positive to Y negative
498       offsetVec.push_back( TireDenom.getSectionWidth() / 2.0 -
499                            TireDenom.getSectionWidth() * i / n
                                 );
500     return offsetVec;
501   }
502
503   //! Find nearest intersection to origin
504   real_type
505   TireDisks::calculateMagnitude(
506     std::vector<vec3> const & IntersectionPointVec
507   ) {
508     real_type iMagnitude = quiteNaN;
509     real_type Magnitude  = quiteNaN;
510     for ( unsigned i = 0; i < IntersectionPointVec.size(); ++i
          ) {
511       iMagnitude = TireDenom.getTireDiameter() / 2.0 -
512                    (IntersectionPointVec[i] - Orient.getOrigin
                        ()).norm();
513       // std::cout << "iMagnitude " << i << ": " << iMagnitude
```

```
            << std::endl;
514         if ( i == 0 ) {
515           Magnitude = iMagnitude;
516           continue;
517         } else {
518           if (iMagnitude > Magnitude)
519             Magnitude = iMagnitude;
520         }
521       }
522       return Magnitude;
523     }
524
525   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        - - - - - - -
526   // class TireMF
527   // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        - - - - - - -
528
529   void
530   TireMF::terrainSampling(void) {
531     // Calculate Delta_X and Delta_Y
532     real_type Delta_X = 0.1 * TireDenom.getTireDiameter();
533     real_type Delta_Y = 0.3 * TireDenom.getSectionWidth();
534     // Store the four sample positions
535     vec3 Qpos[4];
536     Qpos[0] = Orient.getOrigin() + vec3(Delta_X, 0.0, 0.0);
537     Qpos[1] = Orient.getOrigin() - vec3(Delta_X, 0.0, 0.0);
538     Qpos[2] = Orient.getOrigin() + vec3(0.0, Delta_Y, 0.0);
539     Qpos[3] = Orient.getOrigin() - vec3(0.0, Delta_Y, 0.0);
540     // Find intersection in the four positions
541     vec3 IntersectionPoint;
542     std::vector<vec3> IntersectionPointVec;
543     for( unsigned i = 0; i < 4; ++i ) {
544       for ( unsigned t = 0; t < intersectionTriPtr.size(); ++t )
            {
545         if ( algorithms::rayIntersectsTriangle(
546           Qpos[i],
```

```
547           -Orient.getZ(),  // careful to the minus sign !!!
548           *intersectionTriPtr[t],
549           IntersectionPoint)
550         ) {
551           // Store results
552           IntersectionPointVec.push_back(IntersectionPoint);
553         }
554       }
555       // Select the correct intersection point
556       if ( IntersectionPointVec.size() > 1 ) {
557         for ( unsigned j = 0; j < IntersectionPointVec.size(); ++
              j ) {
558           real_type zRealPoint, ziPoint;
559           ziPoint = IntersectionPointVec[j][2];
560           if (j == 0) {
561             zRealPoint = ziPoint;
562           } else if (j > 0 && ziPoint < zRealPoint) {
563             Qvec[i] = IntersectionPointVec[j];
564             zRealPoint = ziPoint;
565           }
566         }
567       } else if ( IntersectionPointVec.size() == 0 ) {
568           std::cerr << "There is no terrain under the tire!\n\n";
569       } else {
570         Qvec[i] = IntersectionPointVec[0];
571       }
572       IntersectionPointVec.clear();
573     }
574   }
575
576   //! Find the normal vector of the local track plane and local
        contact point.
577   std::vector<vec3>
578   TireMF::MF_62( bool print ) {
579     // Check if there is an orientation
580     if ( Orient.isEmpty() ){
581       RDF_ERROR("No orientation given to the tire!");
```

```
582       return std::vector<vec3>{vec3_NaN, vec3_NaN};
583     }
584     // Update the intersected triangles list
585     algorithms::updateShadow(iShadow, TireDenom, Orient);
586     intersectionTriPtr = algorithms::updateIntersectionList(
          iShadow, iMesh);
587     // Perform the terrain sampling;
588     terrainSampling();
589     // Calculate normal of the local track plane
590     vec3 e_n = ((Qvec[0]-Qvec[1]).cross(Qvec[2]-Qvec[3])).
          normalized();
591     // Calculate first guess of local contact point
592     vec3 P_star = vec3(0.0, 0.0, 0.0);
593     for (unsigned i = 0; i < 4; ++i)
594     { P_star += Qvec[i]; }
595     P_star /= 4;
596     // Calculate real local contact point
597     real_type dist = (Orient.getOrigin() - P_star).dot(e_n);
598     vec3 P = Orient.getOrigin() - e_n * dist;
599     // Print the results
600     if (print) {
601       std::cout
602         << "Magic Formula 6.2 parameters\n"
603         << "\tNormal vector of the local track plane\n"
604         << "\ten = [ " << e_n[0] << ", " << e_n[1] << ", " << e_n
              [2] << " ]\n"
605         << "\tFirst guess of the local contact point\n"
606         << "\tP* = [ " << P_star[0] << ", " << P_star[1] << ", "
              << P_star[2] << " ]\n"
607         << "\tReal local contact point\n"
608         << "\tP  = [ " << P[0] << ", " << P[1] << ", " << P[2] <<
               " ]\n\n";
609     }
610     return std::vector<vec3>{e_n, P_star};
611   }
612
613   //! Move the current tire from a starting point to an ending
```

```
    point.
614 void
615 TireMF::Move(
616   vec3      const & start,    //! Starting position
617   vec3      const & arrival,  //! Arrival position
618   real_type const & freq,     //!< Sampling frequency [Hz]
619   real_type const & speed,    //!< Tire speed [m/s]
620   bool              print
621 ) {
622   // Set current position
623   vec3 curpose = start;
624
625   // Calculate parameters for the movement
626   real_type nstep = (arrival - start).norm() / speed * freq;
627   vec3 step       = (arrival - start) / nstep;
628
629   // Set current orientation
630   Orientation Pose(curpose, 0.0, 0.0);
631   setOrientation(Pose);
632
633   // Start chronometer
634   auto start_move = std::chrono::system_clock::now();
635
636   for ( unsigned i = 0; i < nstep; ++i ) {
637     // Update position
638     setOrigin(curpose);
639     // Calculate Magic Formula 6.2 parameters
640     MF_62(print);
641     // Update current position
642     curpose += step;
643   }
644   // Stop chronometer
645   auto end_move = std::chrono::system_clock::now();
646
647   // This constructs a duration object using milliseconds
648   auto elapsed_move = std::chrono::duration_cast<std::chrono::
       microseconds>(
```

```
649           end_move - start_move);
650     std::cout
651       << "Step execution time = "
652       << (elapsed_move.count() / 1000.0) / nstep << std::
              setprecision(3) << " ms\n";
653   }
654
655 } // namespace PatchTire
```

Codice dei Tests B

B.1 Tests Geometrici

B.1.1 geometry-test1.cc

```
1 // GEOMETRY TEST 1 - RAY/TRIANGLE INTERSECTION ON TRIANGLE EDGE
2
3 #include <fstream>   // fStream - STD File I/O Library
4 #include <iostream>  // Iostream - STD I/O Library
5 #include <string>    // String - STD String Library
6
7 #include "PatchTire.hh"  // Tire Data Processing
8 #include "RoadRDF.hh"    // Tire Data Processing
9
10 // Main function
11 int
12 main() {
13   std::cout
14     << " GEOMETRY TEST 1 - RAY/TRIANGLE INTERSECTION ON TRIANGLE
          EDGE\n"
15     << "Angle\tIntersections\n";
16
17   RDF::vec3 V1[3];
```

```
18  V1[0] = RDF::vec3(1.0, 0.0, 0.0);
19  V1[1] = RDF::vec3(0.0, 1.0, 0.0);
20  V1[2] = RDF::vec3(-1.0, 0.0, 0.0);
21
22  RDF::vec3 V2[3];
23  V2[0] = RDF::vec3(-1.0, 0.0, 0.0);
24  V2[1] = RDF::vec3(0.0, -1.0, 0.0);
25  V2[2] = RDF::vec3(1.0, 0.0, 0.0);
26
27  // Initialize generic Triangle3D
28  RDF::Triangle3D Triangle1(V1, 0.0);
29  RDF::Triangle3D Triangle2(V2, 0.0);
30
31  // Initialize rotation matrix
32  RDF::mat3 Rot_X;
33
34  // Initialize intersection point
35  RDF::vec3 IntersectionPointTri1, IntersectionPointTri2;
36  bool IntersectionBoolTri1, IntersectionBoolTri2;
37
38  // Initialize Ray
39  RDF::vec3 RayOrigin    = RDF::vec3(0.0, 0.0, 0.0);
40  RDF::vec3 RayDirection = RDF::vec3(0.0, 0.0, -1.0);
41
42  // Perform intersection at 0.5° step
43  for ( RDF::real_type angle = 0;
44        angle < G2lib::m_pi;
45        angle += G2lib::m_pi / 360.0 ) {
46
47    Rot_X << 1,          0,           0,
48             0, cos(angle), -sin(angle),
49             0, sin(angle),  cos(angle);
50
51    // Initialize vertices
52    RDF::vec3 VerticesTri1[3], VerticesTri2[3];
53
54    VerticesTri1[0] = Rot_X * V1[0];
```

```
55    VerticesTri1[1] = Rot_X * V1[1];
56    VerticesTri1[2] = Rot_X * V1[2];
57
58    VerticesTri2[0] = Rot_X * V2[0];
59    VerticesTri2[1] = Rot_X * V2[1];
60    VerticesTri2[2] = Rot_X * V2[2];
61
62    Triangle1.setVertices(VerticesTri1);
63    Triangle2.setVertices(VerticesTri2);
64
65    IntersectionBoolTri1 = PatchTire::algorithms::
          rayIntersectsTriangle(
66      RayOrigin, RayDirection, Triangle1, IntersectionPointTri1
67    );
68    IntersectionBoolTri2 = PatchTire::algorithms::
          rayIntersectsTriangle(
69      RayOrigin, RayDirection, Triangle2, IntersectionPointTri2
70    );
71
72    std::cout
73      << angle * 180.0 / G2lib::m_pi << "°\t"
74      << "T1 -> " << IntersectionBoolTri1 << ", T2 -> "
75      << IntersectionBoolTri2 << std::endl;
76
77    // ERROR if no one of the two triangles is hit
78    if ( !IntersectionBoolTri1 && !IntersectionBoolTri2 ) {
79      std::cout << "GEOMETRY TEST 1: Failed\n";
80      break;
81    }
82  }
83
84  // Print triangle normal vector
85  RDF::vec3 N1 = Triangle1.Normal();
86  RDF::vec3 N2 = Triangle2.Normal();
87  std::cout
88    << "\nTriangle 1 face normal = [" << N1[0] << ", " << N1[1]
          << ", " << N1[2] << "]"
```

```
89      << "\nTriangle 2 face normal = [" << N2[0] << ", " << N2[1]
            << ", " << N2[2] << "]"
90      << "\n\n\nGEOMETRY TEST 1: Completed\n";
91
92    // Exit the program
93    return 0;
94  }
```

B.1.2 geometry-test2.cc

```
1  // GEOMETRY TEST 2 - SEGMENT CIRCLE INTERSECTION
2
3  #include <fstream>   // fStream - STD File I/O Library
4  #include <iostream>  // Iostream - STD I/O Library
5  #include <string>    // String - STD String Library
6
7  #include "PatchTire.hh"  // Tire Data Processing
8  #include "RoadRDF.hh"    // Tire Data Processing
9
10 // Main function
11 int
12 main() {
13   // Initialize circle origin and radius
14   RDF::vec2 Origin = RDF::vec2(0.0, 0.0);
15   RDF::real_type R = 1.0;
16
17   // Initialize segments points
18   RDF::vec2 SegIn1PtA = RDF::vec2(0.0, 0.0);
19   RDF::vec2 SegIn1PtB = RDF::vec2(0.0, 1.0);
20
21   RDF::vec2 SegIn2PtA = RDF::vec2(-2.0, 0.0);
22   RDF::vec2 SegIn2PtB = RDF::vec2(2.0, 0.0);
23
24   RDF::vec2 SegOutPtA = RDF::vec2(1.0, 2.0);
25   RDF::vec2 SegOutPtB = RDF::vec2(-1.0, 2.0);
26
27   RDF::vec2 SegTangPtA = RDF::vec2(1.0, 1.0);
```

```
28 RDF::vec2 SegTangPtB = RDF::vec2(-1.0, 1.0);
29
30 // Initialize intersection points and output types
31 RDF::vec2 IntSegIn1_1, IntSegIn1_2, IntSegIn2_1, IntSegIn2_2,
       IntSegOut_1,
32     IntSegOut_2, IntSegTang_1, IntSegTang_2;
33 RDF::int_type PtIn1, PtIn2, PtOut, PtTang;
34
35 // Calculate intersections
36 PtIn1 = PatchTire::algorithms::segmentIntersectsCircle(
37   Origin, R, SegIn1PtA, SegIn1PtB, IntSegIn1_1, IntSegIn1_2
38 );
39 PtIn2 = PatchTire::algorithms::segmentIntersectsCircle(
40   Origin, R, SegIn2PtA, SegIn2PtB, IntSegIn2_1, IntSegIn2_2
41 );
42 PtOut = PatchTire::algorithms::segmentIntersectsCircle(
43   Origin, R, SegOutPtA, SegOutPtB, IntSegOut_1, IntSegOut_2
44 );
45 PtTang = PatchTire::algorithms::segmentIntersectsCircle(
46   Origin, R, SegTangPtA, SegTangPtB, IntSegTang_1, IntSegTang_2
47 );
48
49 // Diplay results
50 std::cout
51   << "GEOMETRY TEST 2 - SEGMENT CIRCLE INTERSECTION\n\n"
52   << "Radius = " << R << std::endl
53   << "Origin = [" << Origin[0] << ", " << Origin[1] << "]\n"
54   << std::endl
55   << "Segment 1 with two intersections -> " << PtIn1 << "
       intersections found\n"
56   << "Segment Point A\t= [" << SegIn1PtA[0] << ", " <<
       SegIn1PtA[1] << "]\n"
57   << "Segment Point B\t= [" << SegIn1PtB[0] << ", " <<
       SegIn1PtB[1] << "]\n"
58   << "Intersection Point 1\t= [" << IntSegIn1_1[0] << ", " <<
       IntSegIn1_1[1] << "]\n"
59   << "Intersection Point 2\t= [" << IntSegIn1_2[0] << ", " <<
```

```
        IntSegIn1_2[1] << "]\n"
60    << std::endl
61    << "Segment 2 with two intersections -> " << PtIn2 << "
        intersections found\n"
62    << "Segment Point A\t= [" << SegIn2PtA[0] << ", " <<
        SegIn2PtA[1] << "]\n"
63    << "Segment Point B\t= [" << SegIn2PtB[0] << ", " <<
        SegIn2PtB[1] << "]\n"
64    << "Intersection Point 1\t= [" << IntSegIn2_1[0] << ", " <<
        IntSegIn2_1[1] << "]\n"
65    << "Intersection Point 2\t= [" << IntSegIn2_2[0] << ", " <<
        IntSegIn2_2[1] << "]\n"
66    << std::endl
67    << "Segment with no intersections -> " << PtOut << "
        intersections found\n"
68    << "Segment Point A\t= [" << SegOutPtA[0] << ", " <<
        SegOutPtA[1] << "]\n"
69    << "Segment Point B\t= [" << SegOutPtB[0] << ", " <<
        SegOutPtB[1] << "]\n"
70    << "Intersection Point 1\t= [" << IntSegOut_1[0] << ", " <<
        IntSegOut_1[1] << "]\n"
71    << "Intersection Point 2\t= [" << IntSegOut_2[0] << ", " <<
        IntSegOut_2[1] << "]\n"
72    << std::endl
73    << "Segment with one intersection -> " << PtTang << "
        intersection found\n"
74    << "Segment Point A\t= [" << SegTangPtA[0] << ", " <<
        SegTangPtA[1] << "]\n"
75    << "Segment Point B\t= [" << SegTangPtB[0] << ", " <<
        SegTangPtB[1] << "]\n"
76    << "Intersection Point 1\t= [" << IntSegTang_1[0] << ", " <<
        IntSegTang_1[1] << "]\n"
77    << "Intersection Point 2\t= [" << IntSegTang_2[0] << ", " <<
        IntSegTang_2[1] << "]\n"
78    << "\nCheck the results...\n"
79    << "\nGEOMETRY TEST 2: Completed\n";
80
```

```
81   // Exit the program
82   return 0;
83 }
```

B.1.3 geometry-test3.cc

```
1 // GEOMETRY TEST 3 - POINT INSIDE CIRCLE
2
3 #include <fstream>   // fStream - STD File I/O Library
4 #include <iostream>  // Iostream - STD I/O Library
5 #include <string>    // String - STD String Library
6
7 #include "PatchTire.hh"  // Tire Data Processing
8 #include "RoadRDF.hh"    // Tire Data Processing
9
10 // Main function
11 int
12 main() {
13   // Initialize circle origin and radius
14   RDF::vec2 Origin = RDF::vec2(0.0, 0.0);
15   RDF::real_type R = 1.0;
16
17   // Query points and intersection bools
18   RDF::vec2 PointIn     = RDF::vec2(0.0, 0.0);
19   RDF::vec2 PointOut    = RDF::vec2(2.0, 0.0);
20   RDF::vec2 PointBorder = RDF::vec2(1.0, 0.0);
21
22   bool PtInBool, PtOutBool, PtBordBool;
23
24   // Calculate intersection
25   PtInBool   = PatchTire::algorithms::pointInsideCircle(Origin, R
       , PointIn);
26   PtOutBool  = PatchTire::algorithms::pointInsideCircle(Origin, R
       , PointOut);
27   PtBordBool = PatchTire::algorithms::pointInsideCircle(Origin, R
       , PointBorder);
28
```

```
29  std::cout
30    << "GEOMETRY TEST 3 - POINT INSIDE CIRCLE\n\n"
31    << "Radius = " << R << std::endl
32    << "Origin = [" << Origin[0] << ", " << Origin[1] << "]\n\n";
33
34  // Show results
35  if ( PtInBool && !PtOutBool && PtBordBool ) {
36    std::cout
37      << "Point inside\t= ["
38      << PointIn[0] << ", " << PointIn[1] << "] -> Bool = " <<
            PtInBool << std::endl
39      << "Point outside\t= ["
40      << PointOut[0] << ", " << PointOut[1]  << "] -> Bool = " <<
             PtOutBool << std::endl
41      << "Point on border\t= ["
42      << PointBorder[0] << ", " << PointBorder[1] << "] -> Bool =
             "<< PtBordBool
43      << std::endl;
44  } else {
45    std::cout << "GEOMETRY TEST 3: Failed";
46  }
47
48  std::cout << "\nGEOMETRY TEST 3: Completed\n";
49
50  // Exit the program
51  return 0;
52 }
```

B.1.4 geometry-test4.cc

```
1 // GEOMETRY TEST 4 - POINT ON SEGMENT
2
3 #include <fstream>   // fStream - STD File I/O Library
4 #include <iostream>  // Iostream - STD I/O Library
5 #include <string>    // String - STD String Library
6
7 #include "PatchTire.hh"  // Tire Data Processing
```

```
8 #include "RoadRDF.hh"    // Tire Data Processing
9
10 // Main function
11 int
12 main() {
13   // Initialize segment points
14   RDF::vec2 PointA = RDF::vec2(0.0, 0.0);
15   RDF::vec2 PointB = RDF::vec2(1.0, 1.0);
16
17   // Query points and intersection bools
18   RDF::vec2 PointIn     = RDF::vec2(0.5, 0.5);
19   RDF::vec2 PointOut    = RDF::vec2(-1.0, -1.0);
20   RDF::vec2 PointBorder = RDF::vec2(1.0, 1.0);
21
22   // Calculate intersection
23   bool PtInBool   = PatchTire::algorithms::pointOnSegment(PointA,
         PointB, PointIn);
24   bool PtOutBool  = PatchTire::algorithms::pointOnSegment(PointA,
         PointB, PointOut);
25   bool PtBordBool = PatchTire::algorithms::pointOnSegment(PointA,
         PointB, PointBorder);
26
27   std::cout
28     << "GEOMETRY TEST 4 - POINT ON SEGMENT\n\n"
29     << "Point A = [" << PointA[0] << ", " << PointA[1] << "]\n"
30     << "Point B = [" << PointB[0] << ", " << PointB[1] << "]\n\n
          ";
31
32   // Show results
33   if ( PtInBool && !PtOutBool && PtBordBool ) {
34     std::cout
35       << "Point inside\t= ["
36       << PointIn[0] << ", " << PointIn[1] << "] -> Bool = " <<
            PtInBool
37       << "\nPoint outside\t= ["
38       << PointOut[0] << ", " << PointOut[1] << "] -> Bool = " <<
            PtOutBool
```

```
39      << "\nPoint on border\t= ["
40      << PointBorder[0] << ", " << PointBorder[1] << "] -> Bool =
            " << PtBordBool
41      << std::endl;
42  } else {
43    std::cout << "GEOMETRY TEST 4: Failed";
44  }
45
46  std::cout << "\nGEOMETRY TEST 4: Completed\n";
47
48  // Exit the program
49  return 0;
50 }
```

B.2 Tests per il Modello Magic Formula

B.2.1 MF-test1.cc

```
1 // PATCH EVALUATION TEST 1 - LOAD THE DATA FROM THE RDF FILE THEN
      PRINT IT INTO
2 // A FILE Out.txt. THEN CHARGE THE TIRE DATA AND ASSOCIATE THE
     CURRENT MESH TO
3 // IT.
4
5 #include <chrono>    // chrono - STD Time Measurement Library
6 #include <fstream>   // fStream - STD File I/O Library
7 #include <iostream>  // Iostream - STD I/O Library
8
9 #include "PatchTire.hh" // Tire Data Processing
10 #include "RoadRDF.hh"   // Tire Data Processing
11 #include "TicToc.hh"    // Processing Time Library
12
13 // Main function
14 int
15 main() {
16
17  try {
```

```
18
19    // Instantiate a TicToc object
20    TicToc tictoc;
21
22    std::cout
23      << "MAGIC FORMULA EVALUATION TEST 1 - LOAD THE DATA FROM
          THE RDF FILE"
24         "THEN PRINT IT INTO A FILE Out.txt. THEN CHARGE THE TIRE
              DATA AND"
25         "ASSOCIATE THE CURRENT MESH TO IT.\n\n";
26
27    // Start chronometer
28    tictoc.tic();
29
30    // Load .rdf File
31    RDF::MeshSurface Road;
32    bool checkLoad = Road.LoadFile("./RDF_files/Eight.rdf");
33    RDF_ASSERT( checkLoad, "Error during RDF file reading" );
34
35    // Print OutMesh.txt file
36    Road.printData("OutMesh.txt");
37
38    // Make a new tire
39    PatchTire::ETRTO Tire;
40    Tire = PatchTire::ETRTO(205, 45, 15);
41
42    // Display current tire data on command line
43    Tire.print(std::cout);
44
45    // Orient the tire in the space
46    double Yaw                  = 3.14 / 2;
47    double Camber               = -3.14 / 4;
48    RDF::vec3 Origin( 0.0, 0.0, 0.26 );
49    PatchTire::Orientation Pose(Origin, Yaw, Camber);
50
51    // Initialize the Magic Formula Tire
52    PatchTire::TireMF MFTire( Tire, Road );
```

```
53
54    // Set an orientation
55    MFTire.setOrientation(Pose);
56
57    // Stop chronometer
58    tictoc.toc();
59
60    // This constructs a duration object using milliseconds
61    std::cout
62      << "Execution time = " << tictoc.elapsed_ms() << " ms\n"
63      << "\nCheck the results...\n"
64      << "\nMAGIC FORMULA TEST 1: Completed\n";
65
66  } catch ( std::exception const & exc ) {
67    std::cerr << exc.what() << '\n';
68  }
69  catch (...) {
70    std::cerr << "Unknown error\n";
71  }
72 }
```

B.2.2 MF-test2.cc

```
1 // PATCH EVALUATION TEST 2 - CHECK MF_Pacejka_SCP INTERSECTION
2
3 #include <fstream>   // fStream - STD File I/O Library
4 #include <iostream>  // Iostream - STD I/O Library
5 #include <string>    // String - STD String Library
6
7 #include "PatchTire.hh" // Tire Data Processing
8 #include "RoadRDF.hh"   // Tire Data Processing
9 #include "TicToc.hh"    // Processing Time Library
10
11 // Main function
12 int
13 main() {
14
```

```
15 try {
16
17   // Instantiate a TicToc object
18   TicToc tictoc;
19
20   std::cout
21     << "MAGIC FORMULA EVALUATION TEST 2 - CHECK MF_62 FUNCTION\
        n\n";
22
23   // Initialize a quite big triangle
24   RDF::vec3  Vertices[3];
25   Vertices[0] = RDF::vec3(100.0, 0.0, 0.0);
26   Vertices[1] = RDF::vec3(0.0, 100.0, 0.0);
27   Vertices[2] = RDF::vec3(0.0, -100.0, 0.0);
28   std::vector<std::shared_ptr<RDF::Triangle3D> > PtrTriangleVec
        ;
29   PtrTriangleVec.push_back( std::shared_ptr<RDF::Triangle3D>(
        new RDF::Triangle3D(Vertices, 0.0) ) );
30
31   // Build the mesh
32   RDF::MeshSurface Road(PtrTriangleVec);
33
34   // Make a new tire
35   PatchTire::ETRTO Tire;
36   Tire = PatchTire::ETRTO(205, 45, 15);
37
38   // Display current tire data on command line
39   Tire.print(std::cout);
40
41   // Orient the tire in the space
42   double Yaw                  = 0.0;
43   double Camber               = 0.0;
44   RDF::vec3 Origin( 50.0, 0.0, 0.26 );
45   PatchTire::Orientation Pose(Origin, Yaw, Camber);
46
47   // Initialize the Magic Formula Tire
48   PatchTire::TireMF MFTire( Tire, Road );
```

```
49
50    // Start chronometer
51    tictoc.tic();
52
53    // Set an orientation
54    // Comment to check no orientation error
55    MFTire.setOrientation(Pose);
56
57    // Calculate Magic Formula 6.2 parameters
58    std::vector<RDF::vec3> MFParams = MFTire.MF_62(true);
59
60    // Stop chronometer
61    tictoc.toc();
62
63    // This constructs a duration object using milliseconds
64    std::cout
65      << "Execution time = " << tictoc.elapsed_ms() << " ms\n"
66      << "\nCheck the results...\n"
67      << "\nMAGIC FORMULA TEST 2: Completed\n";
68
69  } catch ( std::exception const & exc ) {
70    std::cerr << exc.what() << '\n';
71  }
72  catch (...) {
73    std::cerr << "Unknown error\n";
74  }
75 }
```

B.2.3 MF-test3.cc

```
1 // PATCH EVALUATION TEST 2 - CHECK MF_Pacejka_SCP INTERSECTION
2
3 #include <fstream>   // fStream - STD File I/O Library
4 #include <iostream>  // Iostream - STD I/O Library
5 #include <string>    // String - STD String Library
6
7 #include "PatchTire.hh" // Tire Data Processing
```

```
8 #include "RoadRDF.hh"   // Tire Data Processing
9 #include "TicToc.hh"    // Processing Time Library
10
11 // Main function
12 int
13 main() {
14
15   try {
16
17     // Instantiate a TicToc object
18     TicToc tictoc;
19
20     std::cout
21       << "MAGIC FORMULA EVALUATION TEST 3 - CHECK TireMF::Move
           FUNCTION\n\n";
22
23     // Load .rdf File
24     RDF::MeshSurface Road;
25     bool checkLoad = Road.LoadFile("./RDF_files/Eight.rdf");
26     RDF_ASSERT( checkLoad, "Error during RDF file reading" );
27
28     // Make a new tire
29     PatchTire::ETRTO Tire;
30     Tire = PatchTire::ETRTO(205, 45, 15);
31
32     // Display current tire data on command line
33     Tire.print(std::cout);
34
35     // Initialize the Magic Formula Tire
36     PatchTire::TireMF MFTire( Tire, Road );
37
38     // Start chronometer
39     tictoc.tic();
40
41     // Move the tire
42     double freq  = 10;  // Hz
43     double speed = 7;     // m/s
```

```
44      //RDF::vec3 startpos   = RDF::vec3(0.0, 15.0, 0.26);
45      //RDF::vec3 arrivalpos = RDF::vec3(3.0, 15.0, 0.26);
46      RDF::vec3 startpos   = RDF::vec3(0.0, 0.0, 0.26);
47      RDF::vec3 arrivalpos = RDF::vec3(5.0, 0.0, 0.26);
48
49      // Move the tire
50      MFTire.Move(startpos, arrivalpos, freq, speed, false);
51
52      // Stop chronometer
53      tictoc.toc();
54
55      // This constructs a duration object using milliseconds
56      std::cout
57        << "Execution time = " << tictoc.elapsed_ms() << " ms\n"
58        << "\nCheck the results...\n"
59        << "\nMAGIC FORMULA TEST 3: Completed\n";
60
61    } catch ( std::exception const & exc ) {
62      std::cerr << exc.what() << '\n';
63    }
64    catch (...) {
65      std::cerr << "Unknown error\n";
66    }
67  }
```

Bibliografia

[1] Lars Nyborg Egbert Bakker e Hans B. Pacejka. "Tyre Modelling for Use in Vehicle Dynamics Studies". In: *SAE Transactions* 96 (1987), pp. 190–204. ISSN: 0096736X.

[2] Juan J. Jiménez, Rafael J. Segura e Francisco R. Feito. "A Robust Segment/-Triangle Intersection Algorithm for Interference Tests. Efficiency Study". In: *Comput. Geom. Theory Appl.* 43.5 (lug. 2010), pp. 474–492. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2009.10.001. URL: http://dx.doi.org/10.1016/j.comgeo.2009.10.001.

[3] Dick De Waard Karel A. Brookhuis e Wiel H. Janssen. "Behavioural impacts of advanced driver assistance systems–an overview". In: *European Journal of Transport and Infrastructure Research 1.3* (2019).

[4] Matteo Larcher. "Development of a 14 Degrees of Freedom Vehicle Model for Realtime Simulations in 3D Environment". Master Thesis. University of Trento.

[5] Anu Maria. "Introduction to modeling and simulation". In: *Winter simulation conference* 29 (gen. 1997), pp. 7–13.

[6] Hans Pacejka. *Tire and vehicle dynamics, 3rd Edition*. 2012.

[7] Georg Rill. *Road vehicle dynamics: fundamentals and modeling*. 2011.