Corso di Laurea Magistrale in Ingegneria Informatica e Robotica

DIPARTIMENTO DI INGEGNERIA

Tesina Robot Mobili Intelligenti

Titolo:

Parametrizzazione delle dimensioni di un veicolo in UE4 e realizzazione di un motion model basato sugli encoder di un veicolo in UE4

Studenti: Valerio Brunacci, Giovanni Fagotti

Introduzione

Questa tesina nasce con l'idea di proseguire e integrare il lavoro che l'ISAR Lab svolge quotidianamente, è stata infatti concordata e supportata da alcuni dei suoi componenti.
I due task che sono stati svolti sono, quindi, conseguenti ai progetti portati avanti dal gruppo di ricerca sopracitato.
Entrando nel dettaglio, entrambi i lavori sono funzionali alla realizzazione di una simulazione di guida completa e quanto più possibile indipendenti dai parametri intrinseci del veicolo. In prima istanza si è cercato di astrarre le caratteristiche di un veicolo quali massa, raggio delle ruote, potenza del motore, coppia e altri; questo al fine di poter variare le dimensioni del veicolo e ottenere comunque le stesse prestazioni.
Quanto appena detto è descritto nella prima parte di questa tesina.
Nella seconda parte si è invece posto il focus sull'implementazione dell'odometria associata ad una autovettura. Quì si è anche svolta una simulazione, sfruttando ROS, che ha consentito di confrontare la traiettoria prodotta dalla proposta codificata con il ground truth (GPS di UE4).

Parte 1: rendere costanti le prestazioni di un veicolo in UE4 rispetto ai parametri dipendenti

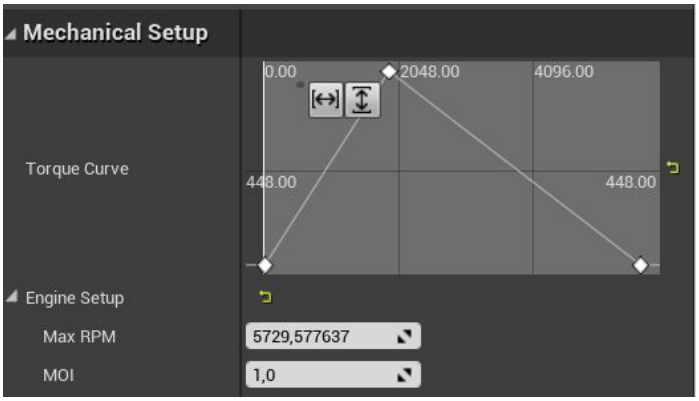
Nella prima parte del progetto, abbiamo analizzato quali parametri devono essere modificati in caso di aumento o diminuzione della dimensione di un veicolo in UE4, senza che le prestazioni dello stesso vengano intaccate.
Per prima cosa, si procede andando a modificare la dimensione della mesh, che inizialmente è settata di default a 1.

(Questa sezione si trova nel menu Mesh, all'interno delle impostazione del Blueprint Vehicle)
Una volta fatto questo, si nota come le ruote del veicolo affondano nel terreno. Per risolvere questo problema, si deve andare ad agire sulla dimensione della "blueprint wheel" andando a moltiplicare il valore "Shape radius" che troviamo di default, per il fattore di scala che abbiamo applicato alla mesh.

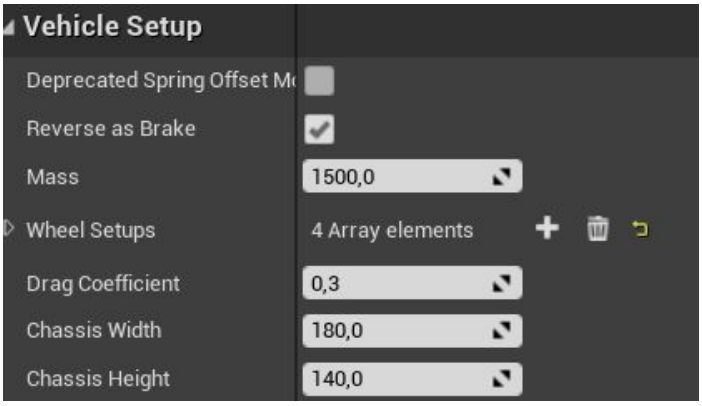
Una volta effettuati questi passaggi, avremo un veicolo di dimensione diversa da quella iniziale ma, se si prova a guidarlo, si noterà che le prestazioni sono molto diverse. Per veicoli di dimensione minore, si avrà una violenta accelerazione e successivamente, il veicolo diventerà poco maneggevole ed instabile.
Per dimensioni maggiori invece, il veicolo avrà un'accelerazione molto bassa e si muoverà lentamente.

Per risolvere questo problema, si dovrà andare ad agire all'interno della classe blueprint del veicolo nella sezione VehicleMovement.

- Veicolo più piccolo del normale: In questo caso, bisogna andare a modificare i parametri relativi alla coppia e al momento di inerzia(MOI) del motore, nella sezione "Engine Setup". La coppia dovrà essere abbassata per permettere un'accelerazione più dolce e una velocità massima limitata, in modo da non far sbandare il veicolo. Per quando riguarda il momento di inerzia invece, deve essere aumentato, in modo da far girare il motore con meno facilità per avere una progressione più costante e meno istantanea.
- Veicolo più grande del normale: Per veicoli di dimensioni maggiori invece, si va ad agire sempre sugli stessi parametri, ma al contrario ovviamente. Quindi dovremo aumentare la coppia, anche se, rispetto al caso del veicolo più piccolo, la coppia dovrà essere aumentata di molto, addirittura facendole raggiungere valori di 10-20 mila N/m. Per quanto riguarda il MOI invece, dovrà essere abbassato, in modo da permettere al motore di girare più facilmente, così da imprimere un'accelerazione più grande al veicolo. Nel caso di veicolo più grande, bisognerà ridurre anche il coefficiente di drag, cioè il coefficiente di attrito dell'aria sul veicolo. Questo perchè, essendo più imponente il veicolo, verrà frenato dall'aria che gli sbatte contro, quindi riducendo questo coefficiente, si avrà più accelerazione che velocità massima.



(Impostazioni relative al motore)



(Impostazione relativa al coefficiente di drag)

Una volta sistemati questi valori, è necessario andare a verificare gli RPM, cioè i giri del motore. Questo perchè, se sono troppo bassi, le marce verranno inserite tutte insieme, quando ancora il veicolo non ha raggiunto la massima coppia, facendo perdere prestazione al veicolo. Se il valore degli RPM è troppo alto invece, non si riuscirà ad inserire tutte le marce, perdendo accelerazione e di conseguenza anche velocità massima. Per controllare questo parametro, una volta che si è alla guida il veicolo si deve visualizzare il debug dei valori della macchina tramite il comando “\show debug”. Una volta visualizzati i parametri e iniziata a muovere la macchina, si può controllare il comportamento degli RPM e delle relative marce, in modo da aggiustare il parametro.

Parte 2: Realizzazione di un motion model a partire dai dati odometrici ricavati da UE4

Nella seconda parte dell'elaborato, si è realizzato un motion model per calcolare la traiettoria completa del veicolo a partire dai dati degli encoder restituiti da UE4.
Il veicolo che verrà utilizzato, è lo stesso implementato nella prima parte.
A questo punto, grazie alle classe c++, il plugin e le api scritte dall'IsarLab, si è in grado di comandare il veicolo tramite un socket python che, oltre a fornire questa funzionalità, restituisce anche i dati della macchina che serviranno poi per implementare il motion model.
Unreal quindi, ci restituirà l'angolo di sterzo delle 4 ruote, il valore dei 4 encoder delle ruote, il raggio delle ruote e la distanza delle ruote, sia anteriori che posteriori dal centro di massa del veicolo.
Ora, con questi dati a disposizione, possiamo procedere con l'implementazione del motion model.
Analizzando il nostro veicolo, si può notare che si basa sullo sterzo di ackermann, quindi dovremo usare delle formule adatte a questo modello.
Nel nostro progetto, viene utilizzata un'approssimazione di questo modello, che ci permette di poter calcolare la posa della macchina con i dati a nostra disposizione.
Il modello approssimato utilizzato, è quello della bicicletta, che ha prestazioni molto simili a quello del modello completo.

Kinematic Bicycle Model

La seconda sezione di questa tesina si pone come scopo quello di riuscire a stimare la posa di un veicolo a 4 ruote tramite esclusivamente i dati odometrici degli econder relativi alle ruote dell'autovettura.

Gli Encoder

Le informazioni quindi a disposizione sono:

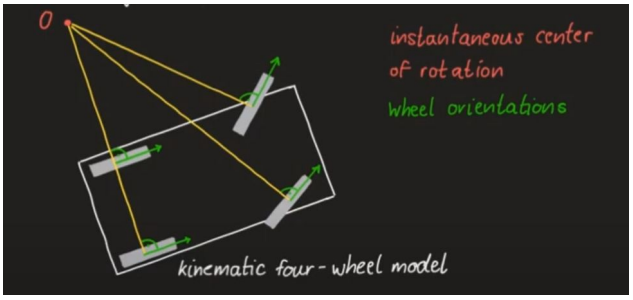
- la posizione relativa dell'encoder rotativo
- l'orientamento della ruota rispetto al corpo macchina fornita da un'encoder assoluto

Il primo informa riguardo quanto un pneumatico ha ruotato su se stesso rispetto al passo di campionamento precedente, ad esempio se al tempo t=0 esso segna 5° e al tempo t=1 segna 10° significa che la ruota ha ruotato di 5 gradi su se stessa; tale trasduttore è anche in grado di fornire la direzione di rotazione: un segno negativo nel simulatore da noi utilizzato sta a significare movimento in avanti e viceversa. Si noti inoltre che questo encoder ha una soglia di saturazione posta a 1800°.

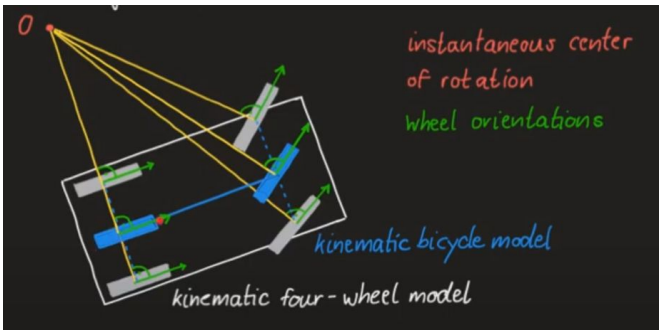
Il secondo essendo un encoder assoluto è in grado di dire esattamente l'angolo che la ruota forma con l'asse longitudinale della vettura; esso è detto steering e rappresenta la direzione delle ruote sterzanti della vettura. Un angolo di 10° in questo caso indica che le ruote sono fuori asse con la vettura di 10°. Si noti come questa informazione non ci dice la direzione della vettura nel suo complesso in quanto è un corpo composto e si muove secondo archi di circonferenza ben precisi.

Vista la scarsa documentazione fornita da Unreal riguardo le unità di misura e il funzionamento dei suoi encoder, la comprensione delle informazioni appena esposte è risultata discretamente complessa.

Il Modello Utilizzato



Un veicolo classico a 4 ruote e 2 assi è un mezzo olonomico, ossia vincolato a muoversi secondo precise restrizioni dovute alla sua configurazione.
Questo fa sì che la sterzata del veicolo avvenga secondo il principio di Ackermann, il quale afferma che, quando un veicolo effettua una sterzata, le linee immaginarie ad angolo retto rispetto al piano di mezzeria delle ruote, devono incontrarsi in uno stesso punto, detto punto di istantanea rotazione, esso quindi è il punto effettivamente rispetto al quale la macchina ruota.



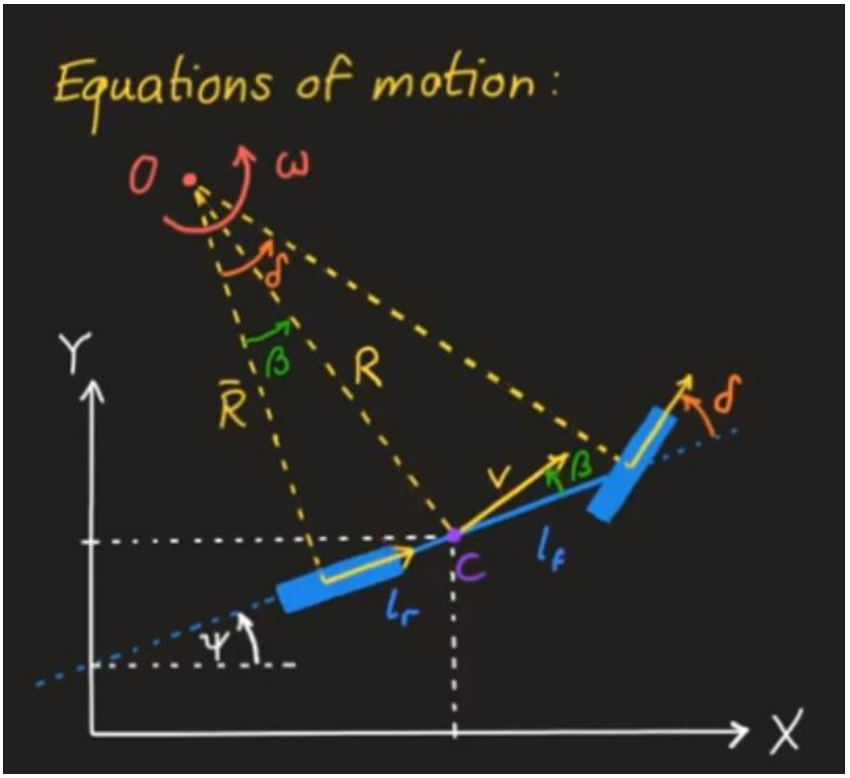
Questo modello, come si può notare dalla figura precedente, prevede che le 2 ruote sterzanti anteriori, abbiano angolazioni differenti per rispettare il principio sopracitato.

In questo elaborato si è deciso di approssimare il modello appena descritto con quello di un Biciclo, in cui si ha quindi 1 solo asse longitudinale (si perde quella trasversale) e una sola ruota sterzante.

Questo modello riesce ad avere buone approssimazioni del modello completo a basse velocità e per angoli di sterzata non grandi (perde circa 3° di sterzata ogni 30°).

Il vantaggio, tuttavia, al netto delle approssimazioni, consiste in una netta semplificazione delle equazioni che descrivono il modello e della loro implementazione.

Il modello definitivo sfruttato per produrre il codice correlato a questo documento è basato sul modello mostrato nella seguente figura:

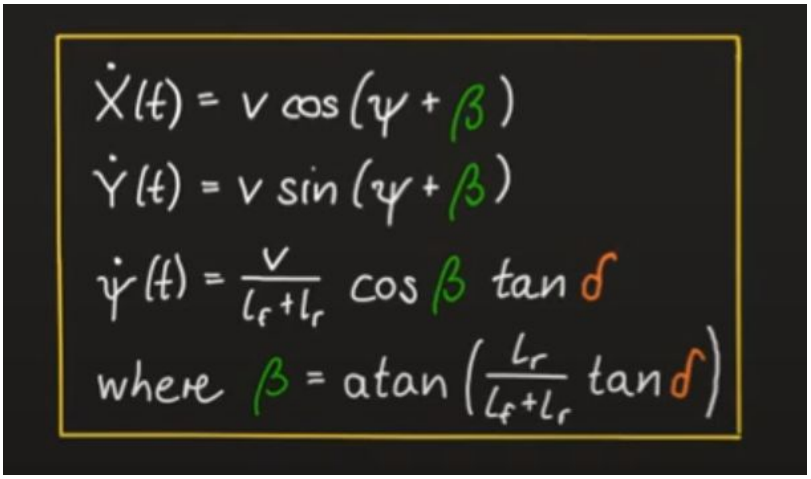


Con riferimento a quanto descritto in precedenza, di seguito è presente un elenco di associazione tra i simboli della figura e il loro significato:

- δ: angolo di sterzata della ruota; è stato calcolato come la media dei 2 angoli di sterzata relativi alle 2 ruote sterzanti del veicolo a 4 simulato.
- C: centro di massa del veicolo; è il punto rispetto al quale si calcolano velocità longitudinale e orientamento del biciclo
- Lr e Lf: distanze rispettivamente dal centro verso la ruota posteriore e verso quella anteriore.
- R: raggio della circonferenza di cui il veicolo ne sta percorrendo un arco.
- ω: velocità angolare con la quale il veicolo si muove rispetto al centro di rotazione istantaneo O.
- β: sideslip angle o anche detto angolo di scivolamento; si noti come esso dipenda dalla posizione di C: tanto più C è vicino alla ruota sterzante tanto più esso si avvicinerà a δ, viceversa tanto più C andrà verso la ruota non sterzante (quindi Lr tende a 0) tanto più β tende a 0.
- v: vettore che rappresenta la velocità del veicolo
- Ψ: anche detto yaw angle esso è l'angolo che rappresenta l'orientamento del veicolo rispetto al sistema di riferimento fissato.

Si noti che per l'implementazione di questo modello si è scelto di sfruttare come sistema di riferimento fissato quello relativo alla posa iniziale del veicolo.
Inoltre nell'implementazione è stata considerata la velocità e lo steering costanti nell'intervallo di campionamento.

Il modello matematico finale è il seguente:



Come si può notare dalle formule tale modello assume come input 2 quantità:

- la velocità v
- l'angolo di sterzata Sigma

Come descritto nel precedente paragrafo si ha a disposizione solamente δ direttamente mentre, per quanto riguarda v, è necessario calcolarla. Per svolgere ciò è stato necessario scrivere un metodo che dal dato relativo estrae il dato assoluto:

```
def ComputeVelocity(self, wheelsRotationAngle, wheelIndex,):
    '''
    This function allows to compute the wheel linear velocity.
    Parmameters
    ----------
    Return
    ----------
    '''
    # sign compute: note thath it is inverted because of UE4 reverse
implementation
    # NOTE: in UE4 negative encoder value rapresents a forward motion and
viceversa.
    rotationSign = 1
    if (wheelsRotationAngle <= 0):
        rotationSign = 1
    else:
        rotationSign = -1

    # this rapresent the threeshold after whichi consider encoder
saturation. note: this refers to the old encoder value
    hightSaturationThreshoold = 1200
    # this rapresent the threeshold below whichi consider encoder
saturation. Note: this refers to the new encoder value
    lowSaturationThreshoold = 200
    # this value save how much the wheel is rotate from the previous step
to the actual
    rotationAngleDifference = 0

    # convert into positiv logic: forward is now negative for UE4
    oldRA = self.oldRotationAngle[wheelIndex] * (-1)
    newRA = wheelsRotationAngle * (-1)

    # TODO: I think the following if can better coding with switch case
and other structure
    # and several cases can be compress,but for quickest learning i
prefer to leave       everything as it is.
    # For testing tasks i don't think this should be a computational
problem!
```

```
   # Forward
   if(oldRA > 0 and newRA > 0):
       # forward without encoder saturation
       if(newRA > oldRA):
           rotationAngleDifference = newRA - oldRA
       # forward with encoder saturation
       if(oldRA > newRA and (oldRA > hightSaturationThreshoold and newRA
< lowSaturationThreshoold)):
           rotationAngleDifference = 1800 - (oldRA - newRA)

   # Forward from negative value to positives
   if(oldRA < 0 and newRA > 0):
       rotationSign = 1
       rotationAngleDifference = newRA-oldRA

   # Forward from negative value to smaller negative ones
   if(oldRA < 0 and newRA < 0):  # both negative
       # be sure that i'm not going to saturating the encoder
       if(oldRA > -hightSaturationThreshoold):
           if(oldRA < newRA):  # moving to zero value
               rotationSign = 1
               rotationAngleDifference = newRA - oldRA

   # Backward
   if(oldRA < 0 and newRA < 0):
       # backward without encoder saturation
       if(newRA < oldRA):
           rotationAngleDifference = oldRA - newRA
       # backward without encoder saturation
       if(oldRA < -hightSaturationThreshoold and newRA >
-lowSaturationThreshoold):
           rotationAngleDifference = 1800 + (oldRA - newRA)

   # Backward from positive value and remaining in positive value
   if(oldRA > 0 and newRA > 0):  # positive values
       # be sure that i'm not going to saturating the encoder
       if(oldRA < hightSaturationThreshoold):
           if(oldRA > newRA):  # backward motion
               rotationSign = -1
               rotationAngleDifference = oldRA-newRA

   # Backward from positive value to negative ones
   if(oldRA > 0 and newRA < 0):
       rotationSign = -1
```

```
        rotationAngleDifference = oldRA - newRA

    # the wheel circunference
    wheelCircumference = 2*math.pi*self.wheelRadius
    # the velocity of the wheel: space/delta_T
    v = (wheelCircumference *
(abs(rotationAngleDifference)/360))/self.delta_T

    # update the memory variable with new old value
    self.oldRotationAngle[wheelIndex] = wheelsRotationAngle

    return [v, rotationSign]
```

In questo codice come si può notare vengono sfruttate 2 soglie:

- hightSaturationThreshoold
- lowSaturationThreeeshold

Tali soglie sono necessarie in quanto servono a capire quando l'encoder ha saturato e quindi ha azzerato il suo valore ed è ripartito; in particolare la soglia "alta" serve a capire quanto al passo precedente l'encoder si è avvicinato alla saturazione. La soglia bassa serve ad accertarsi che il nuovo valore dell'encoder ricada "vicino" allo 0.
In sostanza queste due quantità aiutano a capire la differenza tra una situazione di saturazione e una in cui semplicemente la ruota ha cambiato direzione senza però che l'encoder abbia cambiato segno.

Il calcolo di v è effettuato in linea di principio nella seguente maniera dal metodo mostrato sopra:

- viene estratta la differenza tra il valore attuale dell'encoder e quello memorizzato del passo precedente
- si calcola il segno di rotazione: un intero dal valore di 1 o -1 che indica il senso di rotazione
- si calcola la circonferenza delle ruote noto il raggio delle stesse
- si calcola v come lo spazio percorso dalla ruota (numero di giri fatti dalla ruota * circonferenza delle ruote) diviso l'intervallo di campionamento del modello (delta_T)
- si aggiorna la variabile che tiene in memoria i valori precedenti con i nuovi valori

Il secondo metodo sfruttato è “OdometryCar”, tale metodo si occupa dell'implementazione delle equazioni mostrate sopra. È importante notare come qui venga richiamato il metodo appena descritto e come esso sia fatto girare singolarmente per ogni ruota e successivamente si vada a prendere la media delle velocità prodotte; questo è necessario in quanto non tutte le ruote “rotolano” alla stessa maniera e quindi bisogna fare una scelta; quanto appena fatto notare è risultato un punto critico in quanto in base alle proprietà del veicolo è opportuno scegliere un’approssimazione adeguata della velocità (v): si pensi ad un veicolo con una coppia molto alta, esso tenderà a far slittare le ruote trainanti se accellerato rapidamente, quindi i dati odometri di quelle ruote risulteranno poco affidabili ai fini della stima della traiettoria, sarebbe quindi più corretto eseguire la media solamente delle ruote non trainanti.

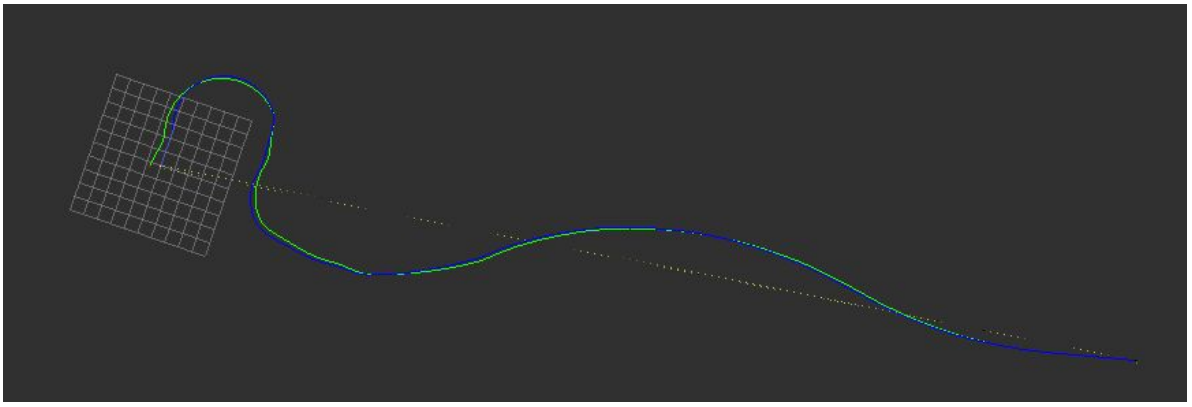
```
def OdometryCar(self, wheelsSteerAngle, wheelsRotationAngle, delta_t):
   # this Odometry model is based on the equations shows in
https://www.youtube.com/watch?v=HqNdBiej23I

   # NOTE: this is an assumption!!!
   # We approssimate the steerangle of the car by mean doing
   # the mean of the front-steering-wheels, that are, for our car, the
only that changing their direction.
   # according to  http://www.ce.unipr.it/~medici/ctrl.pdf for relative
small angle the approssimation is fine

   # print('wheelsSteerAngle: '+str(wheelsSteerAngle))

   wheelsSteerAngle = np.mean([wheelsSteerAngle[0],
wheelsSteerAngle[1]])

   # vehicle velocity
   # compute the velocity for each wheel
   wheelsRotationAngle = np.asarray(wheelsRotationAngle)

   v0 = self.ComputeVelocity(wheelsRotationAngle[0], 0, delta_t)
   v1 = self.ComputeVelocity(wheelsRotationAngle[1], 1, delta_t)
   v2 = self.ComputeVelocity(wheelsRotationAngle[2], 2, delta_t)
   v3 = self.ComputeVelocity(wheelsRotationAngle[3], 3, delta_t)

   allV = np.array([v0[0], v1[0], v2[0], v3[0]])

   # NOTE: APPORSSIMATION!!
   # we decide to take the mean of the 4 wheel, but it is possibile also
```

```
to use
   # the slowest or faster 2 wheel, or the fronts ones, or the back
ones.
   # It is your choise

   # CODE to use the 2 fasters
   # max1 = np.sort(allV)[len(allV)-1]
   # max2 = np.sort(allV)[len(allV)-2]
   # v = np.mean([max1, max2])

   # Code to use the mean
   v = np.mean(allV)

   # can happen that some wheel became negative before the others
   # and the car is not going backward ( positive logic motion ( see
inside ComputeVelocity))
   # so we decide thath the direction of movement of the car change only
when all the wheel are rotating
   # in the same towards
   rotationSign = v0[1]
   if (v0[1] == v1[1] == v2[1] == v3[1]):
       rotationSign = v0[1]

   # Compute vehicle orientation angle
   # change steering sign to be compliance with UE4 gps data
   if(rotationSign == 1):
       yawAngle = wheelsSteerAngle * (-1)  # this value is in degree
   else:
       yawAngle = wheelsSteerAngle   # this value is in degree

   yawAngle = yawAngle * (math.pi/180)  # convert degree to radians

   # vehicle sideslip Angle  (beta)
   tan = math.tan(yawAngle)
   beta = math.atan((self.Lr/(self.Lr+self.Lr))
                    * tan)  # beta is in radians

   # Compute of the velocity kinematic model

   # first compute the car orientation
   yawRate = (v/(self.Lr*2)) * math.cos(beta) * math.tan(yawAngle)
   newYawAngle = yawRate * delta_t

   # second compute the components of the velocity vector.
   # NOTE: with bycicle model we ar assuming that this vector is
```

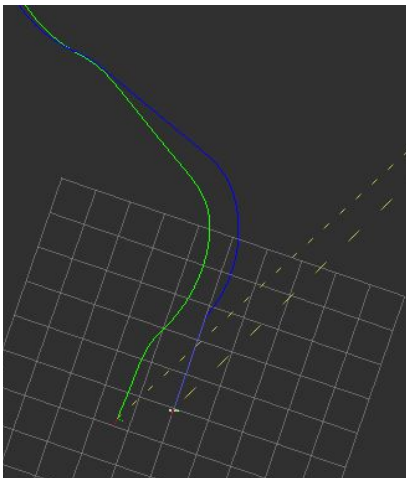
```
positioned
    # at distance Lr from the rear wheel and Lf from the front wheel
    vx = v * math.cos(newYawAngle + beta)
    vy = v * math.sin(newYawAngle + beta)

    # From velocity model to odometric model
    dx = vx * delta_t * rotationSign
    dy = vy * delta_t * rotationSign

    return [dx, dy, newYawAngle]
```

Prestazioni del modello implementato

Per l'interfacciamento con l'agente di unreal engine 4 si è sfruttata l'interfaccia sviluppata nel laboratorio di robotica, la quale ha consentito di misurare le prestazioni del modello potendolo confrontare con i dati assoluti relativi alla posa forniti dal gps integrato di UE4.
Tale confronto lo si è effettuato appoggiandosi alla piattaforma ROS e con la quale si è andati a visualizzare le due traiettorie tramite rviz.

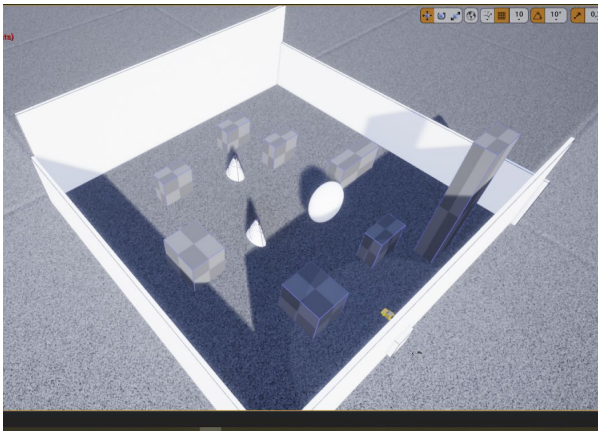


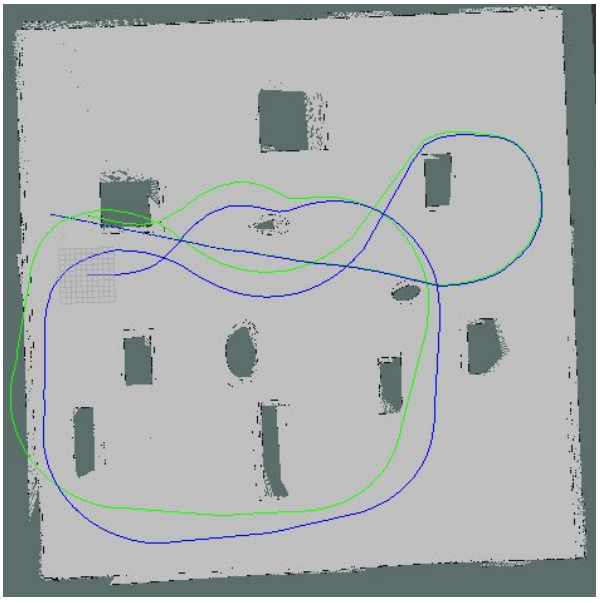
Nell'immagine precedente in Blu è disegnata la traiettoria costruita grazie all'odometria, dunque sui dati degli encoder, in Verde la traiettoria reale del veicolo fornita dal GPS simulato di UE4.



La traiettoria precedente è stata creata muovendo manualmente la vettura nel simulatore e effettuando "piccole" sterzate, questo perché si è riscontrata una difficoltà del modello quando l'angolo di sterzata è elevato.
Ciò è evidente nell'immagine affianco.

Grazie alle funzionalità avanzate di ROS e Unreal è stato possibile anche effettuare una prova del funzionamento dell'algoritmo di SLAM Gmapping.
In input è stato fornita la posa generata dall'odometria, in output viene restituita la mappa dell'ambiente di test.
Il sensore sfruttato è un laser con 200° di visione sul piano sagittale e con un fondo-scala di 15 metri; esso per ogni grado acquisisce 10 campioni.





Sopra è riportato a sinistra il livello di Unreal dove si è svolta la simulazione e di fianco la mappa costruita da GMapping. In quest'ultima si può notare in blu la traiettoria costruita dal modello implementato mentre in verde è riportata la traiettoria costruita tramite il GPS.

Conclusioni

La prima parte di questo lavoro ha estrapolato dal backend di UE4 le variabili grazie alle quali è possibile modificare un veicolo.
Per ottenere le stesse prestazioni con parametri differenti, è necessario sia scalare alcuni di questi con fattori moltiplicativi, sia tunarne manualmente altri basandosi sul feedback del simulatore.

Nella seconda parte di questo lavoro si è implementato e valutato un modello cinematico per un veicolo a 4 ruote.
Le prestazioni del modello testato sono risultate ambigue:

- in condizioni favorevoli (basse velocità, piccoli angoli di sterzata, slittamento delle ruote pressoché nullo) mostra prestazioni buone.
- in condizioni più complicate (velocità elevata, angoli di sterzata netti e sostenuti, slittamento delle ruote presente) sono esaltati i limiti prestazionali dovuti alle approssimazioni effettuate.

Si ritiene che la più importante approssimazione effettuata sia quella di usare il modello di un biciclo al posto di quello completo con 4 ruote.
Una seconda importante approssimazione risiede nel fare la media dello steering delle ruote sterzanti (anteriore dx e anteriore sx); ciò è necessario in quanto il modello scelto prevede una sola ruota sterzante, mentre il veicolo simulato, ovviamente, ne presenta due. Queste, rispettando il principio di Ackermann, presentano un differente angolo di sterzata.

BIBLIOGRAFIA

- UWheeledVehicleMovementComponent
- Vehicle Dynamics & Control - 05 Kinematic bicycle model
- Simple Understanding of Kinematic Bicycle Model | by Yan Ding
- https://github.com/valiokei/Bycicle-Kinematic-Odometry/tree/master