Turtlebot 3 Autonomous Driving

Trotti Francesco VR398637
Moreno Bragaglio VR423929

November 20, 2018

Università degli studi di Verona

Introduzione

Autonomous Driving

Il package "autonomous driving" del turtlebot3 si occupa di identificare:

- Posizione all'interno delle carreggiate
- Semafori
- Parcheggi
- Cartelli stradali
- Passaggi nei tunnel

Scopo del progetto

Il task si è concentrato principalmente sui nodi riguardanti la line detection dell'intero package. Lo scopo finale è di eliminare dalle carreggiate la linea gialla e sostituirla con una linea bianca utilizzando come turtlebot3 il waffle, e di eseguire delle prime prove nella realtà.

Problematiche e risoluzioni

Sono emerse delle problematiche riguardo il turtlebot3 waffle in quanto l'altezza della camera è più bassa del modello burger creando così problemi nella proiezione tramite la matrice di omografia. Si è deciso quindi di utilizzare il burger ma non essendo disponibile all'interno dell'università si è proceduti in simulata.

Installazione

Prerequisiti

- UBUNTU 16.04
- ROS kinetic
- OpenCV 3.0
- Workspace catkin per la compilazione

Clone del package

- Clonare dalla repository github il package AutoraceTurtlebot3ROS in catkin_ws/src
- ```
  $ git clone https://github.com/FrancescoTrotti/
      AutoraceTurtlebot3ROS.git
  ```
- Eseguire catkin_make
  ```

Contenuto del package

Nel package sono presenti diverse cartelle:

- config/ parametri della camera per la proiezione e per la detect delle linee (file .yaml)
- launch/ contiene i launch file per l'esecuzione dei nodi
- node/ contiene i codici per i 3 nodi
- param/ contiene i parametri per identificare le linee
- urdf/ contiene tutti i modelli di turtlebot3 per gli export
- word/ contiene la descrizione del mondo gazebo per la simulazione

Esecuzione e funzionamento dei nodi

Omografia

I primi due nodi si occupano di eliminare la prospettiva nei frame.

- `image_projection`: prende in input l'immagine della camera e ne crea la matrice omografica per eliminarne la prospettiva.
- `image_compensation`: compensa l'immagine proiettata ma non ha alcun effetto in simulata.

Camera input e frame proiettati

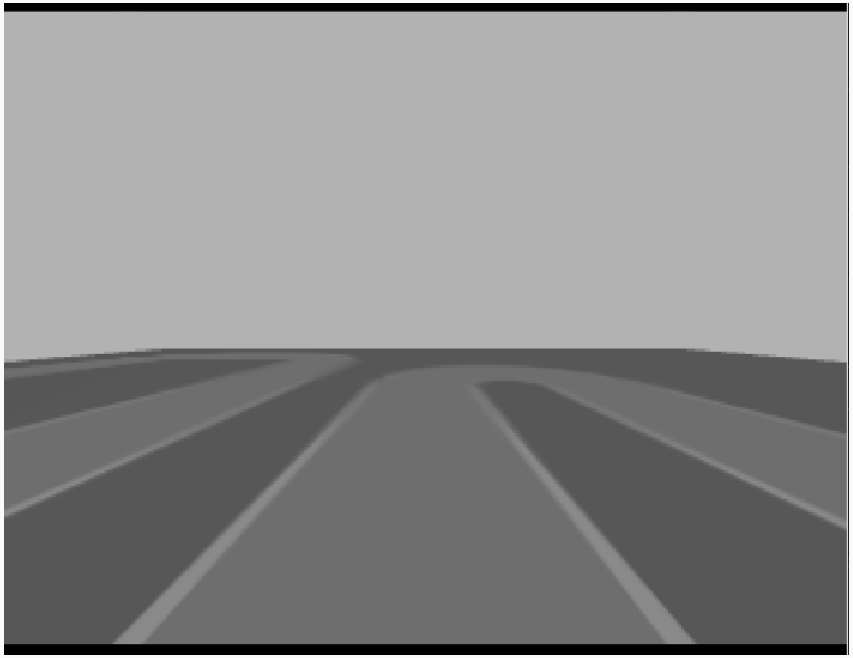


Figure 1: Immagine ripresa dalla camera del Turtlebot3 burger

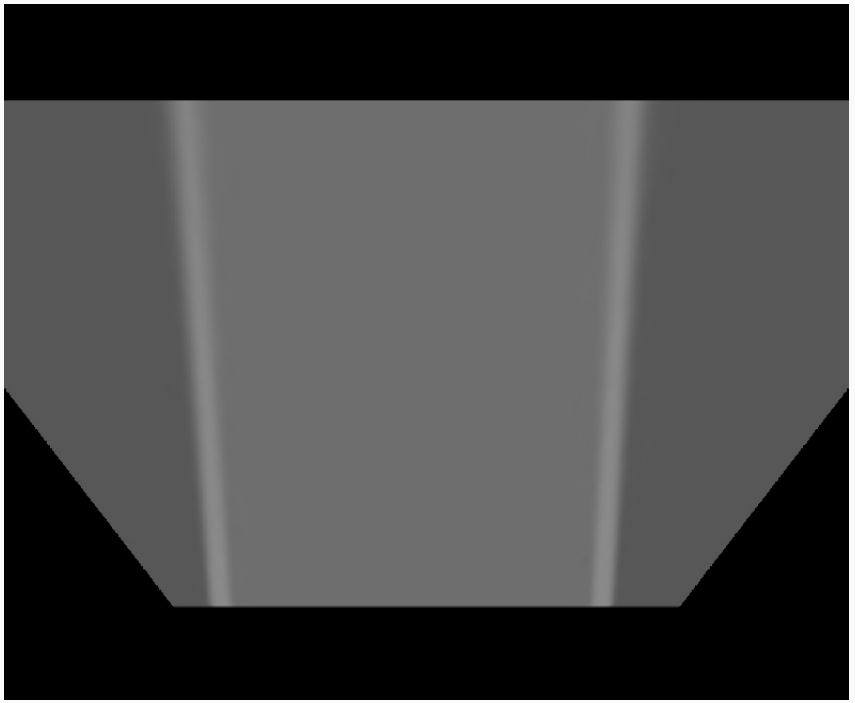


Figure 2: Immagine proiettata

Detect line

L'ultimo nodo si occupa di fare la detect vera e propria delle linee

- `detect_lane`: prende in input il frame proiettato, identifica le due linee bianche e calcola frame per frame la distanza normalizzata tra le due linee, tracciando una linea gialla, che il turtlebot3 dovrà seguire per restare in carreggiata.

Lane Detection

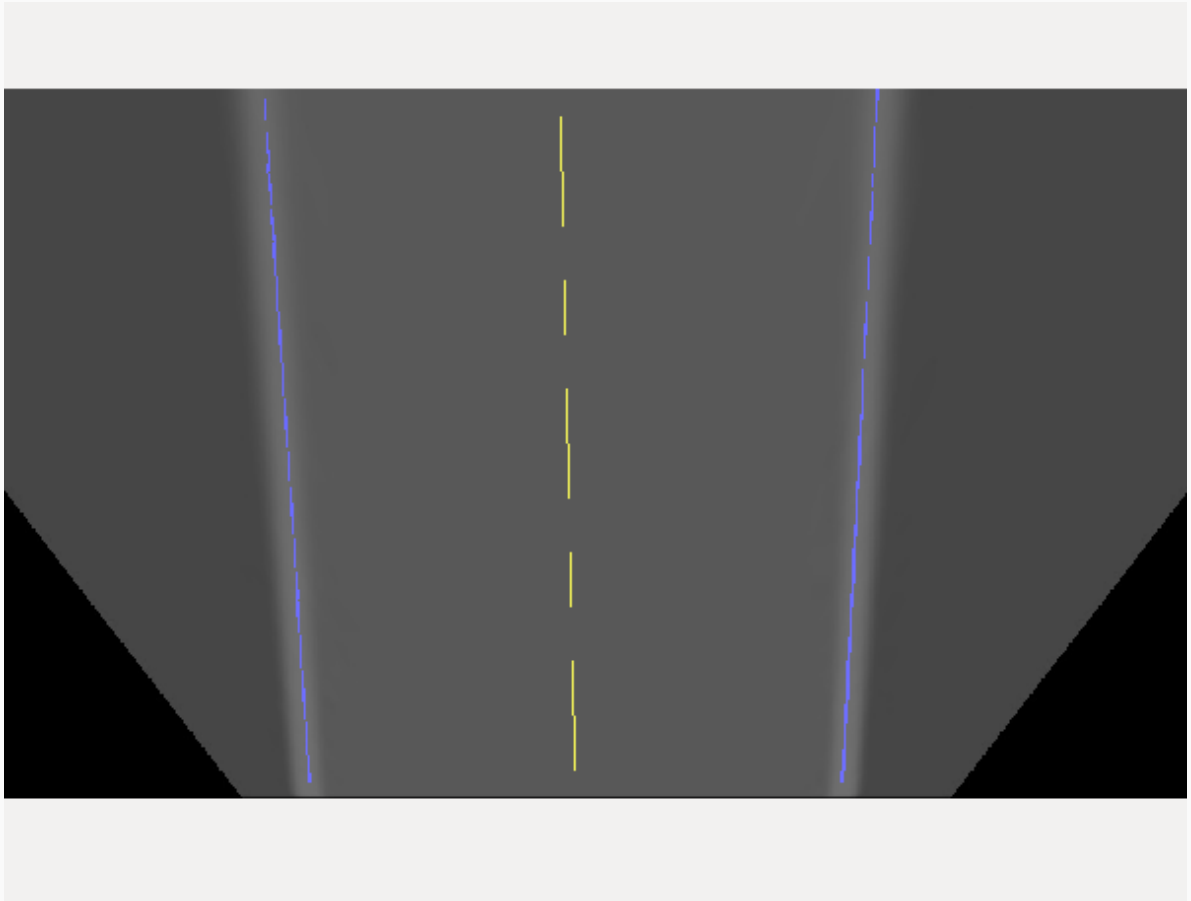


Figure 3: Le linee blu identificano le due linee bianche; la linea gialla centrale è l'insieme dei punti medi normalizzati

Esecuzione simulazione

Comandi esecuzione

In un terminale eseguire:

```
$ roscore
```

In un secondo terminale eseguire:

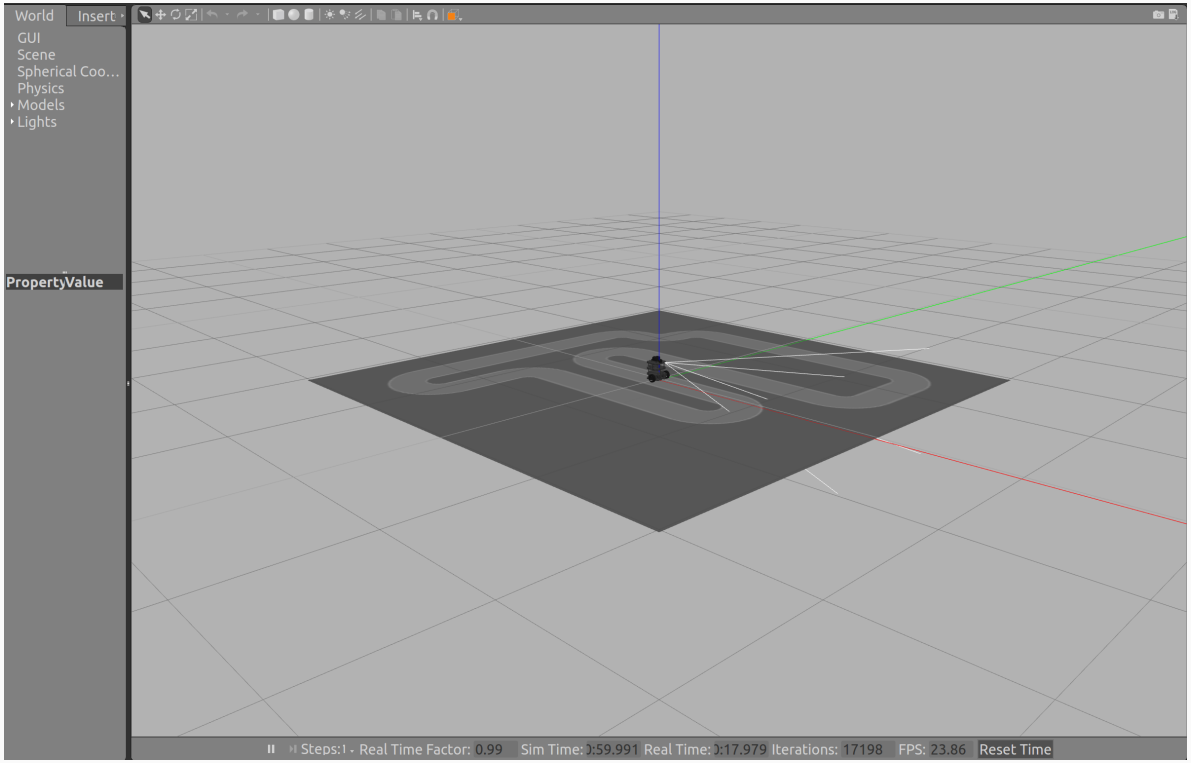
```
$ export TURTLEBOT3_MODEL=burger_pi
$ roslaunch autorace gazebo.launch
```

In un terzo terminale eseguire:

```
$ roslaunch autorace autorace.launch
```

Ambiente di simulazione

Eseguendo il file launch di gazebo viene costruito un mondo simulato in cui la pista e il turtlebot3 sono nella posizione x=0, y=0, z=0



Utilizzo di Rviz

Sottoscrivendosi ai topic in *RViz* è possibile visualizzare ciò che la camera del turtlebot3 riprende, inoltre è possibile visualizzare l'immagine con applicata la matrice omografica. Precisamente nei topic: `/camera/rgb/image_raw` and `/camera/projection/image`.

Prospettive future e conclusioni

Sviluppi futuri

- Dare i comandi ai motori sottoscrivendosi al topic cmd_vel
- Alleggerire il codice e il livello computazionale così da poterlo applicare alla realtà costruendo una pista.
- Applicare il tutto al turtlebot3 waffle

Conclusioni

Questo lavoro, vista la non possibilità di poter utilizzare il waffle, ci ha permesso di approfondire l'attività sul burger.

Video dimostrativo: ▸ Link video