Control of Mobile Robots

Project 2021-2022

Mirko Usuelli (10570238) – mirko.usuelli@mail.polimi.it
M.Sc. Computer Science and Engineering

February 10, 2022

Prof. Luca Bascetta, Politecnico di Milano.

---

1 PI Trajectory Tracking Controller

Starting from the definition of PI controller, I set up the error as the difference of the generated trajectory – x_{ref} and y_{ref} – compared with the measured position of a point belonging to the robot chasis :

$$\begin{cases} e_x(t) = x_{ref}(t) - x_P(t) \\ e_y(t) = y_{ref}(t) - y_P(t) \end{cases}$$

Afterwards, in order to exploit the PI contribution, I computed the feedback error for the velocities according to the meausered and reference position:

$$\begin{cases} v_x(t) = \dot{x}_{ref}(t) + K_{Px} \cdot \left(e_x(t) + \frac{1}{T_{Ix}} \cdot \int e_x(t)dt \right) \\ v_y(t) = \dot{y}_{ref}(t) + K_{Py} \cdot \left(e_y(t) + \frac{1}{T_{Iy}} \cdot \int e_y(t)dt \right) \end{cases}$$

The integrable contribution has been implemented using Backward Euler discretisation:

$$I(t) = \frac{K_P}{T_I} \cdot \int e(t)dt$$

$$\frac{dI(t)}{dt} = \frac{K_P}{T_I} \cdot e(t)$$

$$\frac{I(t) - I(t-1)}{T_S} \simeq \frac{K_P}{T_I} \cdot e(t)$$

$$I(t) \simeq I(t-1) + T_S \cdot \frac{K_P}{T_I} \cdot e(t)$$

Obtaining as final result:

$$\begin{cases} v_x(t) = \dot{x}_{ref}(t) + K_{Px} \cdot e_x(t) + I_x(t-1) + T_S \cdot \frac{K_{Px}}{T_{Ix}} \cdot e_x(t) \\ v_y(t) = \dot{y}_{ref}(t) + K_{Py} \cdot e_y(t) + I_y(t-1) + T_S \cdot \frac{K_{Py}}{T_{Iy}} \cdot e_y(t) \end{cases}$$

Later on, I tuned the PI controller parameters adopting a *trial-and-error* approach as empirical method[1]. At first, I set up a loss function as objective function to be minimized in a grid search – namely a cumulative error in absolute value – as far as concerned for parameters K_{Px} and K_{Py}:

$$\begin{cases} error_x = \Sigma_t |e_x(t)| = \Sigma_t |x_{ref}(t) - x_P(t)| \\ error_y = \Sigma_t |e_y(t)| = \Sigma_t |y_{ref}(t) - y_P(t)| \end{cases}$$

Then, I chose as reference trajectory a Step Function in purpose of measuring reactivity to this sudden input in both component $x(t)$ and $y(t)$ by plotting results through a python script:

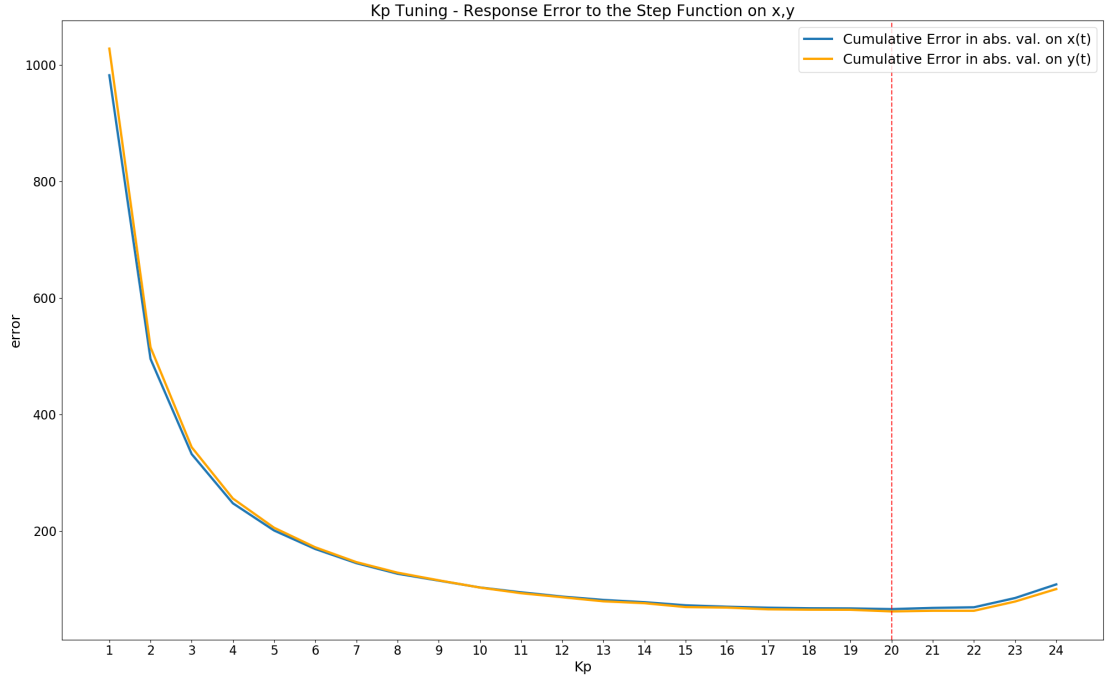

Figure 1: K_{Px} and K_{Py} tuning optimization.

The minimization process clearly identified 20 as the best value for both K_{Px} and K_{Py}.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
error_x	981.98	495.45	332.22	247.89	201.25	169.51	145.2	127.02	115.2	103.47
error_y	1027.71	515.5	344.01	256.04	205.96	172.39	146.92	128.84	115.73	103.3
	11	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	**20**
error_x	95.16	87.82	82.08	78.21	72.83	70.47	68.74	67.78	67.51	**66.34**
error_y	93.64	86.71	79.7	76.49	69.78	68.96	65.98	65.35	65.18	**62.37**
	21	*22*	*23*	*24*						
error_x	68.38	69.47	85.32	108.75						
error_y	63.48	63.46	79.33	100.76						

Table 1: Experimental results for K_{Px} and K_{Py} tuning optimization.

After this, I tuned in the same empirical manner T_{Ix} and T_{Iy} parameters by setting $T_S = 0.001s$ as the overall integration time of the system.

However, the experimental results highlighted the fact that as lower as T_I was, as worse and unstable were the loss function performances. Thus, I decided to discard the $I(t)$ contribution in the PI controller and take just the proportional gain in purpose of preserving the lowest error minimization found so far, otherwise there would have not be any relevant improvement.

Resuming the tuning process, I chose the parameters settings as:

- $K_{Px}, K_{Py} = 20$
- $T_{Ix}, T_{Iy} = 1$
- $T_S = 0$

Where T_{Ix}, T_{Iy}, T_S where specifically chosen in purpose to neglect the Integral contribution according to the data reported above.

2 Bicycle Kinematic Model – `car_traj_ctrl` package

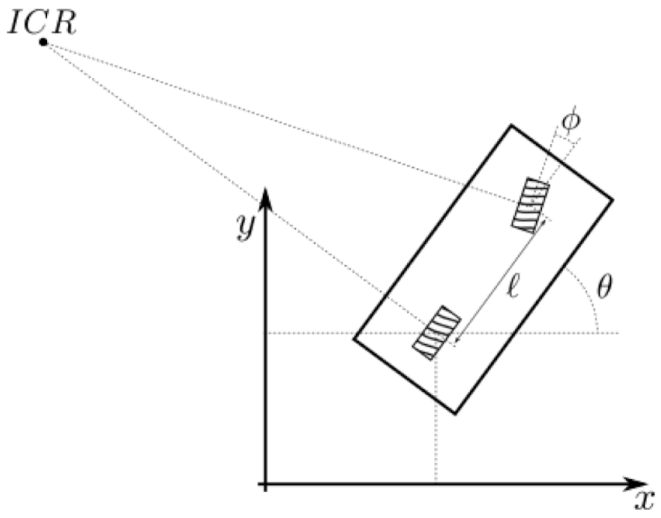

Figure 2: Bicycle Kinematic Model

Starting from the baseline Bicycle Kinematic Model, the parameters that characterise the robot were:

- Wheelbase $L = 0.26\,m$

I set up the initial configuration $q = [x\ y\ \theta\ \phi]^T = [0\ 0\ 0\ 0]^T$, where:

- x : longitudinal position
- y : lateral position
- θ : heading orientation
- ϕ : front steering wheel orientation

Whereas the chosen trajectory was an 8-shaped circuit on the XY plane having the following characteristic equations as source of reference for the mobile robot:

$$\begin{cases} x_{ref} = a \cdot sin(\frac{2\pi}{T} \cdot t) \\ \dot{x}_{ref} = \frac{2\pi}{T} \cdot a \cdot cos(\frac{2\pi}{T} \cdot t) \\ y_{ref} = a \cdot sin(\frac{2\pi}{T} \cdot t) \cdot cos(\frac{2\pi}{T} \cdot t) \\ \dot{y}_{ref} = \frac{2\pi}{T} \cdot a \cdot (cos(\frac{2\pi}{T} \cdot t)^2 - sin(\frac{2\pi}{T} \cdot t)^2) \end{cases}$$

Where $a = 2\,m$ and, in my case, the characteristic period corresponded to $T = 6.3\,s$.

The adopted Control System receives the generated 8-shaped trajectory in terms of x_{ref}, y_{ref} and $\dot{x}_{ref}, \dot{y}_{ref}$ in purpose of exploiting the above mentioned feedback error:

$$\begin{cases} v_x = \dot{x}_{ref} + K_{Px} \cdot (x_{ref} - x_P) \\ v_y = \dot{y}_{ref} + K_{Py} \cdot (y_{ref} - y_P) \end{cases}$$

Then, in order to introduce a Closed-Loop Feeback Controller, I needed to set an arbitrary point P belonging to the Robot Chasis with distance $\varepsilon = 0.05\,m$, consequently obtaining the point P position as:

$$\begin{cases} x_P = x + \varepsilon \cdot cos(\theta) \\ y_P = y + \varepsilon \cdot sin(\theta) \end{cases}$$

Through this relation it is now possible to perfectly exploit the tuning error $e(t)$.

Assuming the canonical form of the Rear-Wheel Bicycle Kinematic Model – configuration $q = [x\ y\ \theta]^T$ – characterized by the following state equations:

$$\begin{cases} \dot{x} = v \cdot cos(\theta) \\ \dot{y} = v \cdot sin(\theta) \\ \dot{\theta} = \frac{v}{L} \cdot tan(\phi) \end{cases}$$

For the rear-wheel drive bicycle we need to assume that the steering rate limit is so high that the steering angle can be changed instantaneously. The model can be controlled by the command variable $u = [v\ \phi]^T$ by writing them according to the unicycle kinematic model Feedback Linearising Law:

$$\begin{cases} v = v_x \cdot cos(\theta) + v_y \cdot sin(\theta) \\ \omega = \frac{1}{\varepsilon} \cdot (v_y \cdot cos(\theta) - v_x \cdot sin(\theta)) \end{cases}$$

Considering $\omega = \frac{v}{L} tan(\phi)$, I obtain:

$$\begin{cases} v = v_x \cdot cos(\theta) + v_y \cdot sin(\theta) \\ \phi = arctan\left(\frac{L}{\varepsilon} \cdot \frac{v_y \cdot cos(\theta) - v_x \cdot sin(\theta)}{v_x \cdot cos(\theta) + v_y \cdot sin(\theta)}\right) \end{cases}$$

At the end, by linking all the reported equations, the overall Control System appears as shown below:

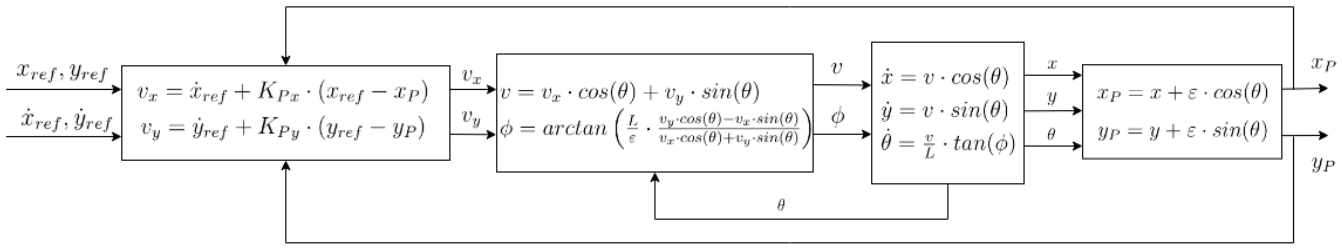

Figure 3: Closed-Loop PI Trajectory Tracking Controller

Concluding this section, I resume that the maximum tracking error recorded in absolute value along the trajectory – in x and y direction – across $1\,min$ of time recordings has been:

- $error_{xMAX} = 0.0141\,m$
- $error_{yMAX} = 0.0314\,m$

Experimental results produced by the descripted system are reported through the following figures produced through a python script:

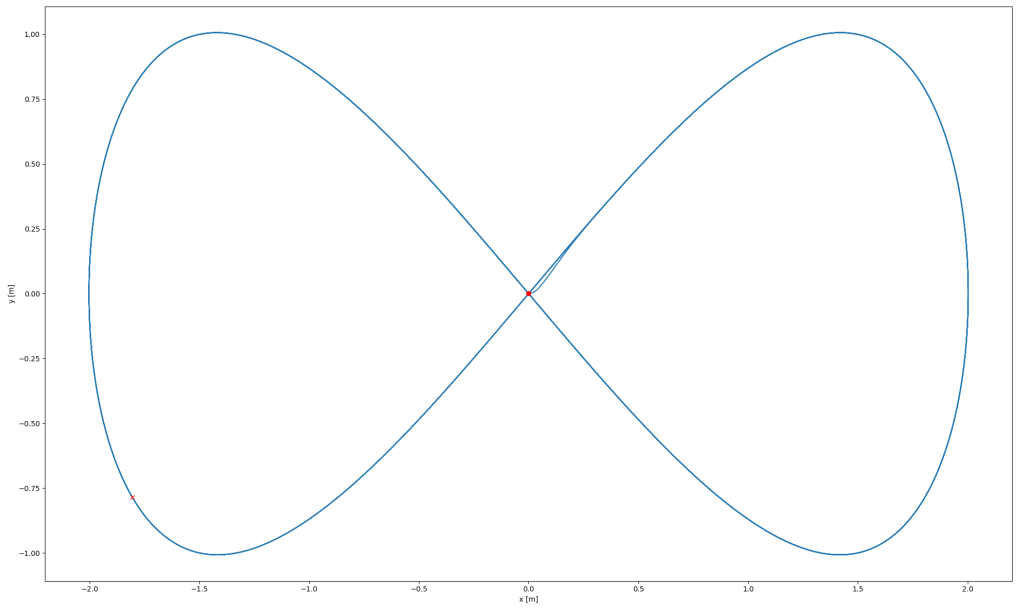

Figure 4: Robot 8-Shaped Trajectory in the XY plane

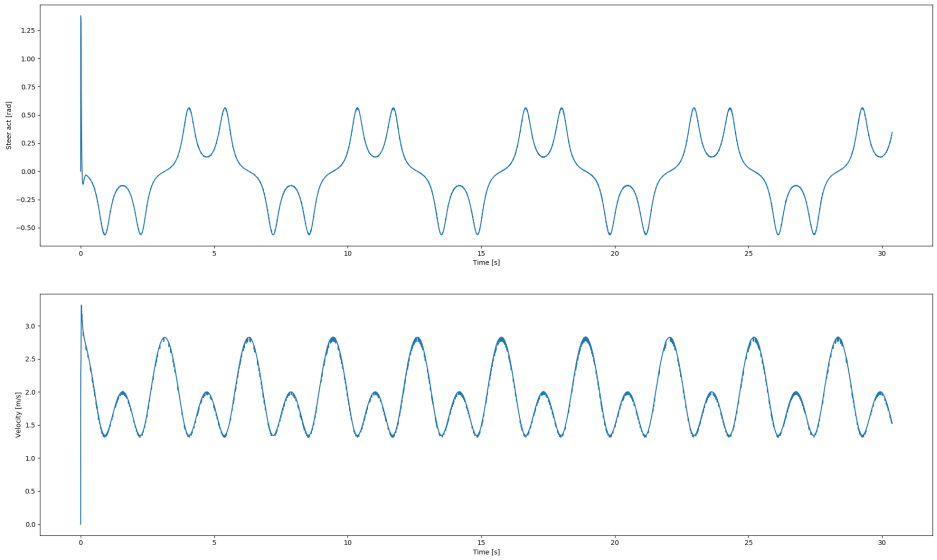

Figure 5: Commands $u = [v\ \phi]^T$ across time

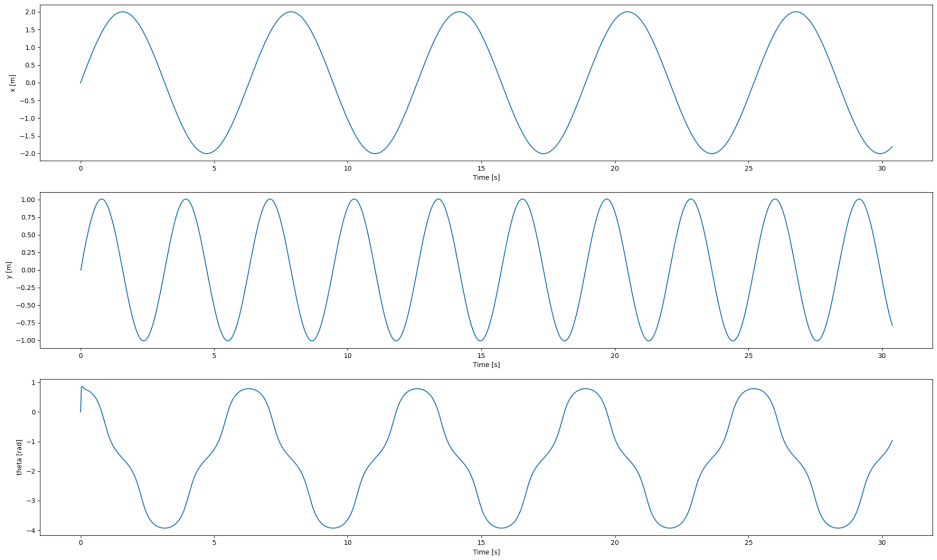

Figure 6: Configuration variables x, y, θ across time

3 Single-Track Dynamic Model – `car_simulator` package

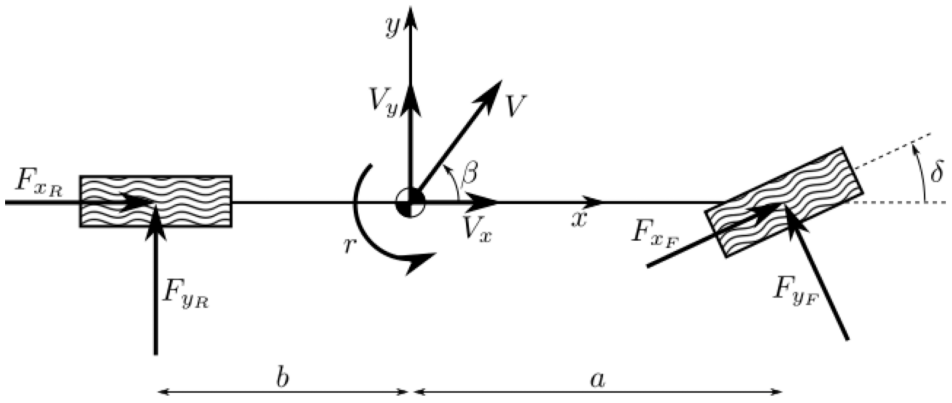

Figure 7: Bicycle Dynamic Model

Starting from the baseline Bicycle Dynamic Model, the forces acting on the system were:

- F_{xR} and F_{xF} : rear (R) and frontal (F) traction / bracking forces
- F_{yR} and F_{yF} : rear (R) and frontal (F) tire-ground interaction forces

While the parameters that characterise the robot were:

- mass $m = 1.2\,kg$
- distance of the center of gravity from the front axle $a = 0.14\,m$
- distance of the center of gravity from the rear axle $b = 0.12\,m$
- ground coefficient $\mu = 0.385$
- front wheel cornering stiffness $C_{\alpha F} = 50\,N/rad$
- rear wheel cornering stiffness $C_{\alpha R} = 120\,N/rad$
- yaw inertia $I_z = 0.028\,kg\,m^2$

The absolute velocity of the vehicle $V = \sqrt{V_x^2 + V_y^2}$ and the longitudinal (V_x) and lateral (V_y) velocities are related through the sideslip angle:

$$\beta = arctan\left(\frac{V_y}{V_x}\right)$$

In order to compute the tire slip angles I need the expression of the front and rear wheel velocities:

- $V_{xF} = V \cdot cos(\beta) \approx V$
- $V_{xR} = V \cdot cos(\beta) \approx V$
- $V_{yF} = V \cdot sin(\beta) + a \cdot r \approx V \cdot \beta + a \cdot r$
- $V_{yR} = V \cdot sin(\beta) - b \cdot r \approx V \cdot \beta - b \cdot r$

Then, I can now write the slip angles relations as:

$$\begin{cases} \alpha_F = arctan\left(\frac{V_{yF}}{V_{xF}}\right) - \delta \approx \frac{V_{yF}}{V_{xF}} - \delta \approx \frac{V \cdot \beta + a \cdot r}{V} - \delta = \beta + \frac{a \cdot r}{V} - \delta \\ \alpha_R = arctan\left(\frac{V_{yF}}{V_{xF}}\right) \approx \frac{V_{yF}}{V_{xF}} \approx \frac{V \cdot \beta - b \cdot r}{V} = \beta - \frac{b \cdot r}{V} \end{cases}$$

Where r is the yaw rate and δ is the angle of the front steering wheel.

I can write the Tyre Model of the lateral forces acting on the robot, assuming it as a Linear Model:

$$\begin{cases} F_{yF} = -C_{\alpha F} \cdot \alpha_F \\ F_{yR} = -C_{\alpha R} \cdot \alpha_R \end{cases}$$

At the end, the Bicycle Dynamic Model can be written as follow:

$$\begin{cases} \dot{x} = V \cdot cos(\psi) \\ \dot{y} = V \cdot sin(\psi) \\ \dot{\psi} = r \\ \dot{\beta} = \frac{F_{yF} + F_{yR}}{m \cdot V} \cdot cos(\beta) - r \\ \dot{r} = \frac{a \cdot F_{yF} - b \cdot F_{yR}}{I_z} \end{cases}$$

The system has been tested following a simple trajectory as requested by setting both commands $u = [V \ r]^T$ with a Step Function at time $t_i = 0\ s$ until $t_f = 5\ s$ where the commands were set to zero again.

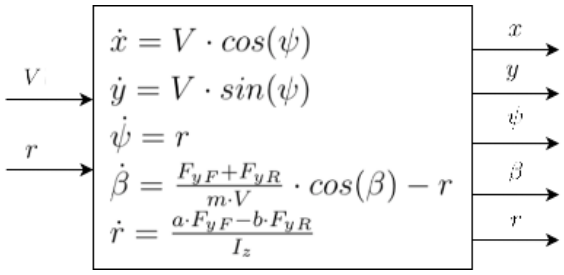

Figure 8: Open-Loop system

Experimental results produced by the descripted system are reported through the following figures produced through a python script:

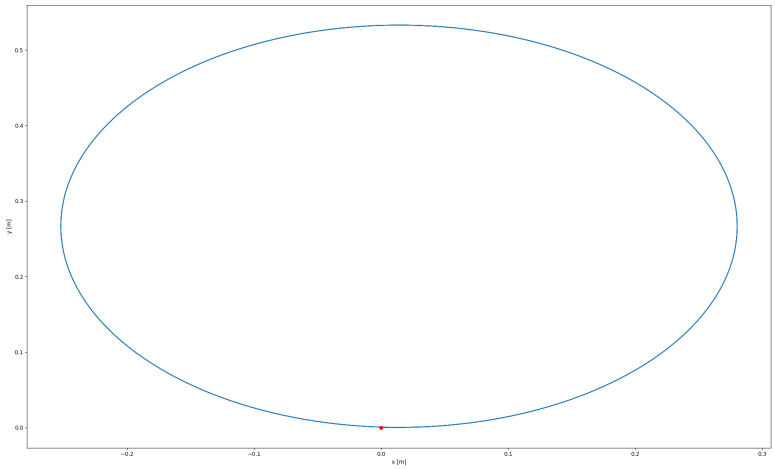

Figure 9: Simple Trajectory.

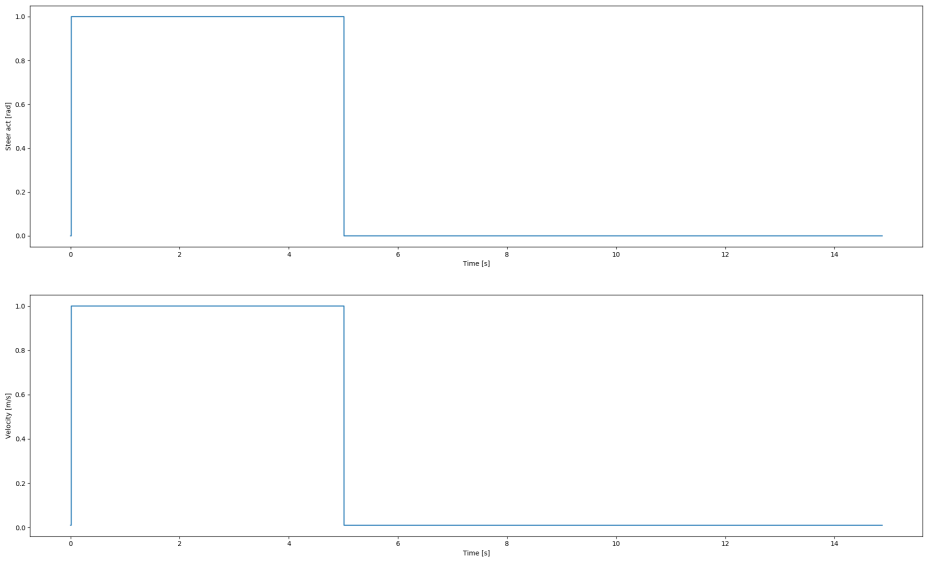

Figure 10: Commands $u = [r\ V]^T$ as a Step Funciton across time.

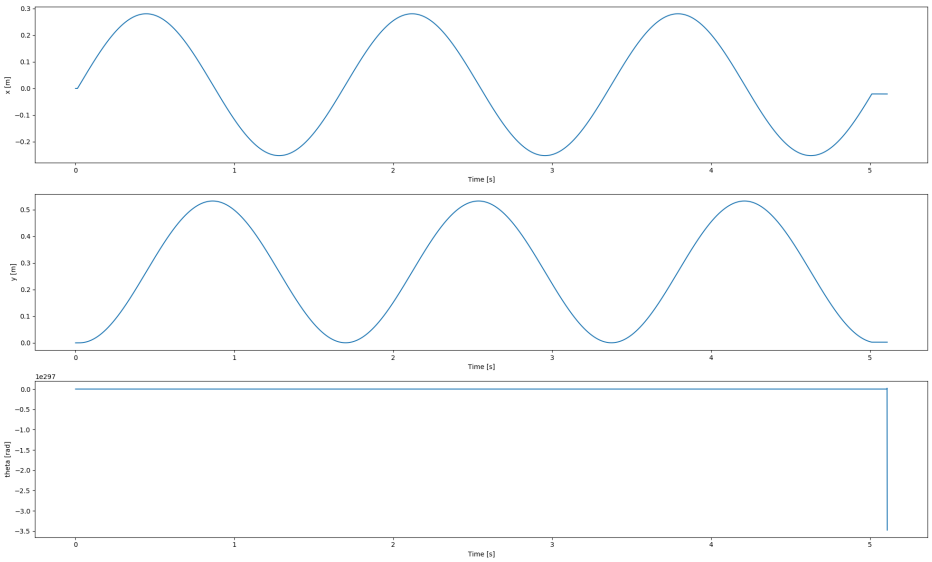

Figure 11: Configuration variables across time

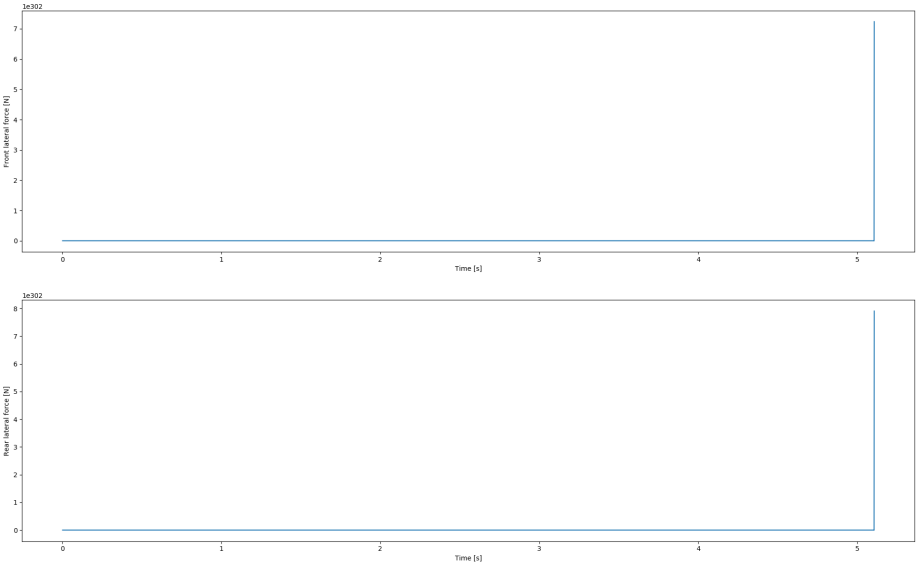

Figure 12: Front lateral force F_{yF} and Rear lateral force F_{yR}

4 Instructions

1. Insert the ROS packages `car_traj_ctrl` for the Bicycle Kinematic Model and `car_simulator` for the Single-Track Dynamic Model by typing:

```
$ mv car_traj_ctrl <path>/catkin_ws/src
$ mv car_simulator <path>/catkin_ws/src
```

2. Go to your own `catkin_ws` folder in the system and recompile everything by doing:

```
$ cd <path>/catkin_ws
$ source ./devel/setup.bash
$ catkin_make
```

3. Then run ROS core by typing:

```
$ roscore
```

4.1 `car_traj_ctrl` package (Bicycle Kinematic Model)

1. Open a new terminal and record the robot activity through a `rosbag` in this manner by reading the topic `/car_state`:

```
$ cd <path>/catkin_ws/src/car_traj_ctrl/script
$ rosbag record -O test_eight /car_state
```

2. Open a new terminal and launch the robot simulation as:

```
$ cd <path>/catkin_ws
$ source ./devel/setup.bash
$ roslaunch car_traj_ctrl test_car_eight.launch
```

3. Go back to the terminal you ran `rosbag` and press `ctrl + c`

4. Do the same in the terminal which is running the robot simulation

5. Go back to the terminal where you stopped `rosbag` and run the following python script by typing on the bash:

```
$ python plot_result.py test_eight.bag
```

6. Now you can see the result produced by the execution you recorded.

4.2 `car_simulator` package (Single-Track Dynamic Model)

1. Open a new terminal and record the robot activity through a `rosbag` in this manner by reading the topic `/car_state`:

```
$ cd <path>/catkin_ws/src/car_simulator/script
$ rosbag record -O test_simple /car_state
```

2. Open a new terminal and launch the robot simulation as:

```
$ cd <path>/catkin_ws
$ source ./devel/setup.bash
$ roslaunch car_simulator test_car_simple.launch
```

3. Go back to the terminal you ran `rosbag` and press `ctrl + c`

4. Do the same in the terminal which is running the robot simulation

5. Go back to the terminal where you stopped `rosbag` and run the following python script by typing on the bash:

```
$ python plot_result.py test_simple.bag
```

6. Now you can see the result produced by the execution you recorded.

References

[1] Kevin Lynch, (author of Modern Robotics: Mechanics, Planning, and Control), Empirical adjustment of the PID gain – Northwestern University – [*youtu.be/uXnDwojRb1g*]