Unicycle and bicycle robot control

David FILLIAT - Alexandre CHAPOUTOT
ENSTA Paris

2 décembre 2020

1 Introduction

In this practical work, we will implement very simple control methods, using PID controllers to guide a unicycle or a bicycle robot towards a goal or along a path. For this, we will use the python code available on the course Moodle. The provided code makes it possible to simulate the motion of unicycle and bicycle robots using the controllers you will have to write. It requires the installation of the `numpy`[1] and `matplotlib`[2] python packages.

Upload your report as a pdf file that includes your answers to the questions and the code you wrote on the Moodle.

2 Unicycle control

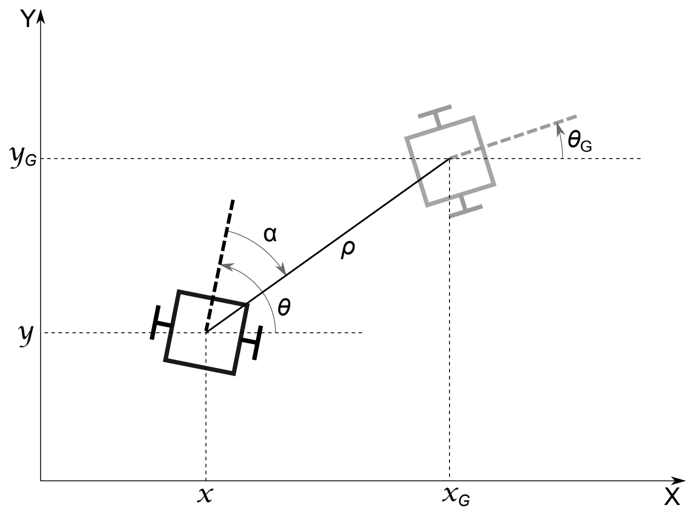

FIGURE 1 – Control of a unicycle towards a position

In this first part, you have to write a proportional controller in order to guide a unicycle robot towards a given pose (Figure 1). The unicycle has limited forward ($\|v\| < 1m.s^{-1}$) and rotation speed ($\|\omega\| < \pi rad.s^{-1}$). Acceleration is also limited ($\|\frac{dv}{dt}\| < 10m.s^{-2}$ and $\|\frac{d\omega}{dt}\| < 10rad.s^{-2}$).

2.1 Question 1 - Unicycle control

You have to implement two Proportional controllers, a first one to bring the robot to the goal point and a second one to adjust the orientation at the goal :

— Far from the goal ($\rho > 0.05$), the controller will use ρ et α shown in figure 1 in order to compute translation and rotation speed :

1. `https://numpy.org/`
2. `https://matplotlib.org/`

$$
\begin{aligned}
\rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\
\alpha &= \arctan\frac{(y_G - y)}{(x_G - x)} - \theta \\
v &= K_\rho \times \rho \\
\omega &= K_\alpha \times \alpha
\end{aligned}
$$

In order to reach the goal faster without making large circular trajectory, you can rotate the robot without moving forward when the direction is too far from the goal direction :

$$
\begin{aligned}
\text{if } |\alpha| &> \alpha_{max} \\
v &= 0
\end{aligned}
$$

— When the robot is close to the goal ($\rho < 0.05$), the controller with use the angle β to compute rotation speed :

$$
\begin{aligned}
\beta &= \theta_G - \theta \\
\omega &= K_\beta \times \beta
\end{aligned}
$$

You have to implement this method in the `unicycle_to_pose_control` function in the `unicycle_to_pose.py` file. Remember to use the `np.atan2(y,x)` function to compute the arctangent in order to compute the right angle when the two x and y values are negative. Also use the provided function `vm.angle_wrap` to reduce angles consistently between $-\pi$ and π.

The `unicycle_to_pose.py` script will test your method from several starting positions on a circle to reach the circle center. Adjust the controller gains and the α_{max} parameter to quickly reach the goal by limiting oscillations (you should have a score below 3500).

3 Control of a bicycle

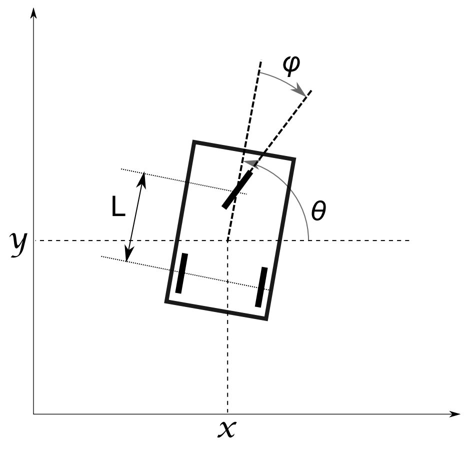

FIGURE 2 – Bicycle model.

We are now going to work on the control of a bicycle model (Figure 2). This model has a limited translation speed ($\|v\| < 1 m.s^{-1}$) and a limited steering wheel angle ($\|\phi\| < 1.2 rad$). In addition, the acceleration is also limited ($\|\frac{dv}{dt}\| < 10 m.s^{-2}$) as well as the rotation speed of the steering wheel ($\|\frac{d\phi}{dt}\| < 8 rad.s^{-1}$).

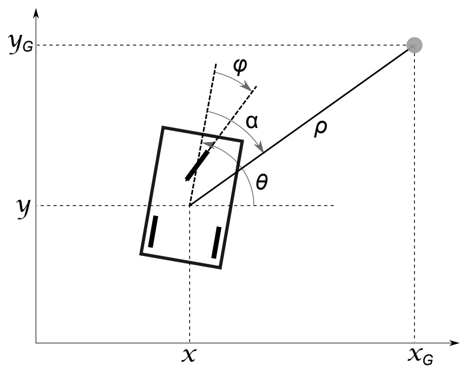

FIGURE 3 – Control of a bicycle towards a point.

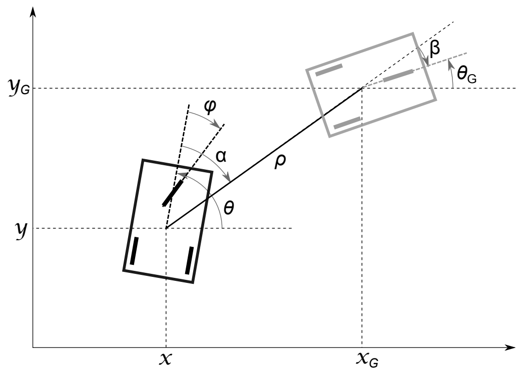

FIGURE 4 – Control of a bicycle towards a pose.

3.1 Question 2 - Control of a bicycle towards a point

You should first write a Proportional controller to guide the robot towards a point (Figure 3). You should use the following equations :

$$\begin{aligned}
\rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\
\alpha &= \arctan \frac{(y_G - y)}{(x_G - x)} - \theta \\
v &= K_\rho \times \rho \\
\varphi &= K_\alpha \times \alpha
\end{aligned}$$

You have to implement this method in the `bicycle_to_point_control` function in the `bicycle_to_point.py` file.

The `bicycle_to_point.py` script will test your method from several starting positions on a circle to reach the circle center. Adjust the controller gains to quickly reach the goal by limiting oscillations (you should have a score below 2800).

3.2 Question 3 - Control of a bicycle towards a pose

We now have to build a proportional controller that guides the robot to a position with a fixed final orientation (Figure 4). For this, you will write a controller using the following equations :

$$\begin{aligned}
\rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\
\alpha &= \arctan \frac{(y_G - y)}{(x_G - x)} - \theta \\
\beta &= \theta_G - \arctan \frac{(y_G - y)}{(x_G - x)} \\
v &= K_\rho \times \rho \\
\varphi &= K_\alpha \times \alpha + K_\beta \times \beta \\
\textit{avec} \quad & K_\beta < 0
\end{aligned}$$

You have to implement this method in the `bicycle_to_pose_control` function in the `bicycle_to_pose.py` file.

The `bicycle_to_pose.py` script will test your method from several starting positions on a circle to reach the circle center. Adjust the controller gains to quickly reach the goal by limiting oscillations (you should have a score below 3500).

3.3 Question 4 - Control of a bicycle following a path

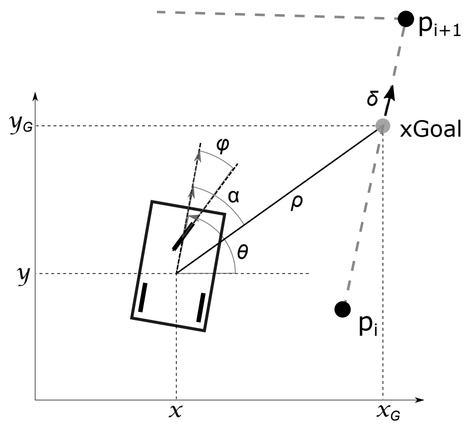

FIGURE 5 – Control of a bicycle following a path.

You now have to write a controller that guides the robot to follow a path defined by a set of waypoints (Figure 5) using the simple Pure Pursuit approach [Coulter, 1992].

At each timestep you need to determine a point on the trajectory at a given lookahead distance ρ (for example $\rho = 0.5$) from the robot, and use the proportional controller from question 2 to compute the speed that will guide the robot toward this point. A simple method is to take a point on the path and move it towards the next waypoint with a small fixed step δ when the robot's distance to that point is less than ρ. When the point reaches the next waypoint of the path, the following waypoint should be used.

You have to implement this method in the `bicycle_to_path_control` function in the `bicycle_to_path.py` file.

The `bicycle_to_path.py` script will test your method on a fixed path. Optimize the parameters (lookahead and controller gains) to follow the path as closely as possible. Explain what happens when the lookahead distance becomes too large.

Références

[Coulter, 1992] Coulter, R. C. (1992). Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST.