Lab 2: Kinematics and Control

ME 597: Autonomous Systems

At this point in the semester we have a fully assembled and tested an autonomous robot. We know how to use the individual components that go into that car. And we have some idea of how to program things (hopefully). The first two control methods we are going to explore for our car are timing based control and line following control

With a timing based controller we will prescribe the different actions for the robot to take based on the time since start. This sort of control architecture is easy to get up and running and sometimes works well enough for highly repeatable systems. With more real world scenarios however, timing control performance degrades rapidly as disturbances continue to grow through the operation of the robot.

The basic idea of a timing based robot is to describe the required motion as a set of maneuvers.

Move	Time (s)
Forward	2
Right Turn	0.5
Forward	1
Right Turn	0.5
Backward	2

TABLE I: Example timing based control

This can be accomplished using the time module, if-else structures, and loops. Basically what you are going to want to do is determine the time, determine what speeds you should send to the motors from your if-else structure, and then act on those speeds. This will involve a lot of trial and error, especially to determine how to effectively turn.

For line following control each car has a set of three IR emitters and detecters. This is useful for detecting if there is a bright or dark surface underneath the vehicle. Basic line following logic effectively simplifies down to the following:

1) Measure with three line modules
2) If middle triggered go forward
3) If left triggered turn left a bit
4) If right triggered go right a bit

This can be extended with memory so that if the robot gets lost it navigates back towards where it last saw the line.

I. Assignment

A. Time Based Control

1) Set up a timing based control function to draw a 0.5 meter box counterclockwise, once
2) Set up a timing based control function to draw a 1 meter box clockwise, repeating
3) Do not use `sleep`, `delay` or any other blocking approach to measure time.

B. Line Follower

1) Set up a simple line following control, test, and observe the disadvantages of this approach.
2) Set up a line following control function with a simple memory so that if the vehicle loses the line it will turn towards where it last saw the line.
3) For all tests, if an obstacle is blocking the path, stop at a safe distance.

C. Linear Control

The goal of this assignment is to create a PID controller to help the uctronics car stop one meter an obstacle.

1) Create a script called `pid_controller.py` in `aut_sys/scripts` and implement the following:
   a) Create a Subscriber that reads from the `/distance` topic.
   b) Create one publisher that writes into the `/motors` topic to move the robot.
   c) Set `rate = rospy.Rate(10)` and use `rate.sleep()` in a loop to maintain a particular rate for the subscriber callback function. Remember that the `rate` function defines the execution speed of your node. In this particular case, the configured frequency is $10Hz$, which means that the controller node will be scheduled for execution every $100ms$. You can tune this parameter, setting it too high will cause an unnecessary load, while setting it to a low value can make your controller to respond very slow.
   d) In the callback function, get the distance measured from the ultrasonic sensor, calculate the error to the target position, use a PID controller to control the velocity, and then publish the velocity (`/motors`) and position (`/distance`).
2) A launch file called `run_pid_control.launch` will be provided. Copy it into the `/launch` folder. This file will:
   a) Launch the main script `pid_controller.py`
   b) Run `rqt_plot`. This node should subscribe to `/cmd_vel/linear/x` and `/distance`. We use it to monitor the robot's behavior in a real-time plot.
3) Launch `run_mission.launch`. See the response in `rqt_plot`. Try to make the settling time as small as possible by tuning the controller.

D. Inverse Kinematics - Mecanum Wheeled Robot

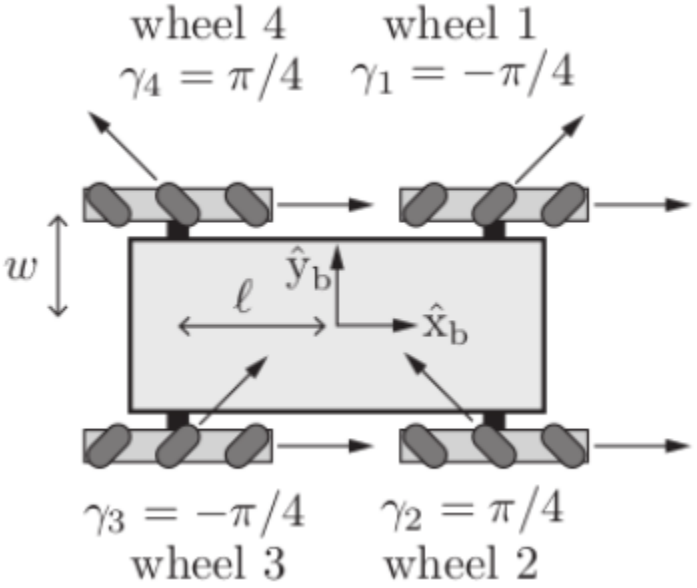

Fig. 1: FIGURE 1: A four mecanum wheeled robot

The final assignment of this lab will help you understand the derivation of inverse kinematics. Here you will see an image of a four-wheeled robot. Unlike the wheeled robots we've seen in lecture, this robot is equipped with mecanum wheels. Unlike a normal wheel, a mecanum wheel has rollers attached

to the circumference of the wheel. These wheels divert the wheel velocity along the roller direction. The wheel constraints of a single mecanum wheel are:

$$V_w \cdot u_r = r\dot{\phi}cos(\gamma) \tag{1}$$

$$V_w \cdot u_t = 0 \tag{2}$$

$$u_r \cdot u_t = 0 \tag{3}$$

where γ is the angle between the roller direction and the axle of the mecanum wheel, u_r is the unit vector pointing in the direction that the roller rolls and u_t is the unit vector pointing in the direction of the roller axle. These 2 constraints are the rolling and no slip constraints respectively.

Finally, the kinematic model of a four wheeled mecanum robot is:

$$\dot{\phi} = 1/r \begin{bmatrix} 1 & -1 & -(w+l) \\ 1 & 1 & (w+l) \\ 1 & -1 & (w+l) \\ 1 & 1 & -(w+l) \end{bmatrix} \dot{\zeta_R} \tag{4}$$

1) Re-derive the kinematic model using the wheel constraints and the procedure and equations discussed in lecture.
2) Using your model, derive the inverse kinematic equation that describe the transformation from the inertial reference frame position space to joint space and the kinematic equation that describes the inverse operation.

II. Deliverables

1) Python node file of the time based control.
2) Python node file of the simple line follower based control.
3) Python node file of the memory line follower based control.
4) Python node file of the PID Controller.
5) Procedure to solve the inverse kinematics problem.

III. Rubric

- **10 pts** - Time Based Control.
  - 5 pts - `time_control.py` works as requested.
  - 5 pts - Properly commented code.
- **30 (+10 bonus) pts** - Line Follower.
  - 25 pts - `line_follower.py` works as requested.
  - 10 bonus pts - `line_follower_memory.py` works as requested.
  - 5 pts - Properly commented code.
- **30 pts** - Linear Control.
  - 25 pts - `pid_control.py` works as requested.
  - 5 pts - Properly commented code.
- **30 pts** - Inverse Kinematics - Mecanum Wheeled Robot.
  - 15 pts - Forward kinematics.
  - 15 pts - Inverse kinematics.