MP2 Walkthrough

Minghao Jiang

Eric Liang

Professor Katie Driggs-Campbell

MP2 - Vehicle model and control

- Demo due 3/25/2021, report due 3/26/2021
- In this MP, there are 2 written questions and 3 implementation questions
- The written questions are related to theories of PID control which will help you understand the gains of PID controller ;
- For the coding section, you will develop a waypoint following controller and use the controller to drive the GEM vehicle in racetrack in Gazebo.

Written Questions

Problem 1 (15 points). Consider the two dimensional ODE system described by:

$$\dot{x} = x^2 + y$$
$$\dot{y} = x - y + a,$$

where a is a parameter of the model. Find all the equilibrium points of this system.

Problem 2 (20 points). Consider the 2-dimensional linear time invariant system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & v \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = Ax + Bu,$$

where v is a model parameter. We would like to design a state-feedback controller to make the system asymptotically stable. Let the feedback law be of the form:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = - \begin{bmatrix} k_{11} & k_{12} \\ 0 & k_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -Kx.$$

Write down the equations for the closed loop system. Suppose the gain k_{22} is set to be 0. Write down conditions on k_{11} and k_{12} or specific values that makes the closed loop system asymptotically stable. Show your work.

Waypoint Following Controller

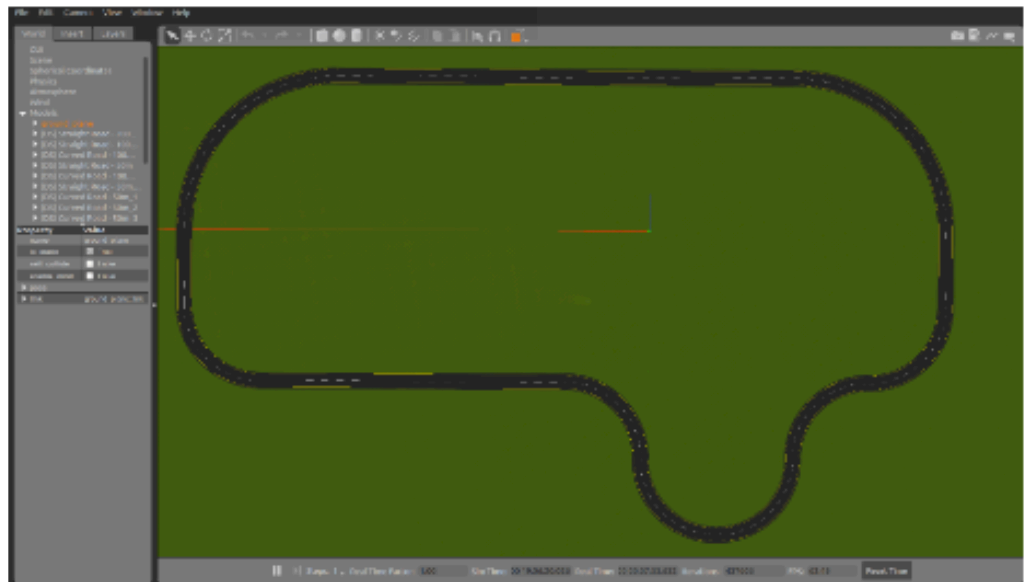
- Same track as MP1
- Preset waypoints are provided in *waypoint_list.py*
- Need to implement ***execute*** function in *controller.py*
- To run the controller, type:

  python main.py
- Code location:

  [mp-directory]/src/mp2/src
- Compiling using: *catkin_make*
- *Launch simluation*

  source devel/setup.bash

  roslaunch mp2 mp2.launch



ROS messages

- GazeboModelState
  - Provide state of GEM Car: Position, Orientation, Velocity, Angular Velocity

```
# Set Gazebo Model pose and twist
string model_name              # model to set state (pose and twist)
geometry_msgs/Pose pose        # desired pose in reference frame
geometry_msgs/Twist twist      # desired twist in reference frame
string reference_frame         # set pose/twist relative to the frame of this entity (Body/Model)
                               # leave empty or "world" or "map" defaults to world-frame
```

- AckermannDrive
  - It's used for setting desired speed and steering angle through actuators
  - Pass the control input(speed, steering angle) calculated by your controller

```
float32 steering_angle          # desired virtual angle (radians)
float32 steering_angle_velocity # desired rate of change (radians/s)
float32 speed                   # desired forward speed (m/s)
float32 acceleration            # desired acceleration (m/s^2)
float32 jerk                    # desired jerk (m/s^3)
```

- Refer to http://docs.ros.org/ for more information

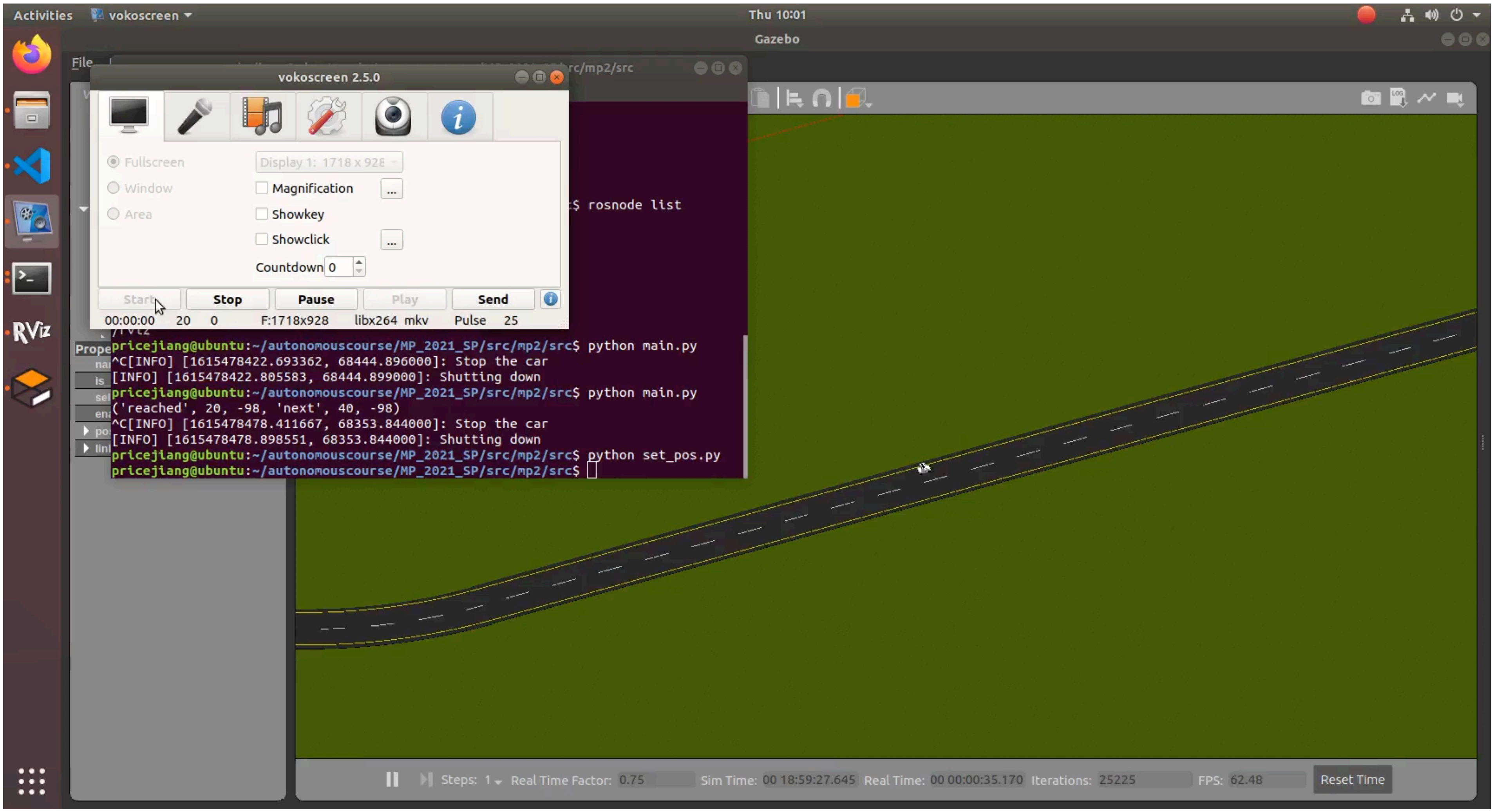
Implementing PD Controller

- For the ***execute*** function, it has 2 input arguments:
  - Reference State $[x_{ref}, y_{ref}, \theta_{ref}, v_{ref}]$
  - Current State $[x_B, y_B, \theta_B, v_B]$
  - You need to calculate the error vector $[\delta_x, \delta_y, \delta_\theta, \delta_v]$ defined as:

$$\delta_x = cos(\theta_B) * (x_{ref} - x_B) + sin(\theta_B) * (y_{ref} - y_B)$$
$$\delta_y = -sin(\theta_B) * (x_{ref} - x_B) + cos(\theta_B) * (y_{ref} - y_B)$$
$$\delta_\theta = \theta_{ref} - \theta_B$$
$$\delta_v = v_{ref} - v_B$$

  - With the error vector, you can calculate control input $u = [v, \delta]$ by

$$u = K * \delta \quad \text{where} \quad K = \begin{bmatrix} k_x & 0 & 0 & k_v \\ 0 & k_y & k_\theta & 0 \end{bmatrix}$$


Activities
vokoscreen
Thu 10:01
Gazebo
vokoscreen 2.5.0
Fullscreen
Window
Area
Magnification
Showkey
Showclick
Countdown 0
Start
Stop
Pause
Play
Send
00:00:00 20 0 F:1718x928 libx264 mkv Pulse 25
$ rosnode list
pricejiang@ubuntu:~/autonomouscourse/MP_2021_SP/src/mp2/src$ python main.py
^C[INFO] [1615478422.693362, 68444.896000]: Stop the car
[INFO] [1615478422.805583, 68444.899000]: Shutting down
pricejiang@ubuntu:~/autonomouscourse/MP_2021_SP/src/mp2/src$ python main.py
('reached', 20, -98, 'next', 40, -98)
^C[INFO] [1615478478.411667, 68353.844000]: Stop the car
[INFO] [1615478478.898551, 68353.844000]: Shutting down
pricejiang@ubuntu:~/autonomouscourse/MP_2021_SP/src/mp2/src$ python set_pos.py
pricejiang@ubuntu:~/autonomouscourse/MP_2021_SP/src/mp2/src$
Steps: 1 Real Time Factor: 0.75 Sim Time: 00 18:59:27.645 Real Time: 00 00:00:35.170 Iterations: 25225 FPS: 62.48
Reset Time

Demo Instruction

- Students need to show their controller:
  - is capable of driving the car for the whole loop
  - can drive the car stably on the road
  - is able to control the car following the preset waypoints and not deviating from the road

Questions?