Manual

Getting started

This system is an Autonomous Bluetooth car racing environment. The projector displays the track that the cars will be driving on, to the large screen. The first menu item you need to select is a map that your car will be driving on. Next you chose your car to drive with. Next you chose a strategy file which is either autonomous self-driving car or the manual car feature. Once you have selected these three items from the menu you can press start and begin driving!

Starting the system

1. Turn the power point on
2. Press the “on” button on the projector
3. Download the code for the program directly by first clicking this link: https://github.com/cits3200-autonomous-car/CITS3200_AutonomousCar/tree/final Click the green “Clone or Download” button and download as a ZIP
4. From downloads find the zip file and right click then click extract to the Desktop
5. Run insall.sh and select “Execute in terminal” to install all required dependencies
6. Run exec.sh and select “Execute in terminal” to run the program

Navigating through the menu

7. Take a car from the charging point, the on button is underneath the car at the rear end it is small and dark red, once the car is on you should hear a noise
8. Using the X-box controller select a map out of 3 options
9. On the menu select the car corresponding to the colour of the car you have turned on

Select a strategy file

- Using the X and Y button on the Xbox controller select a strategy file. This selection is located underneath the car selection you can chose a manual or autonomous driving feature which is “passive.py”.
- Manual car feature: This allows you to drive the car yourself, you can turn the horn and lights on if you wish as well
- Autonomous car feature: This feature is where the system controls the driving of the car, as a user you don’t have to do anything but you can turn on the horn or lights if you wish.

Entering a scenario

9. Once you have made your selections, press the green Start button on the menu to enter the scenario
10. If the “User-Defined” map option has been selected, set up the cones in pairs as desired and then press A on the controller or Enter on the keyboard
11. Place the car in its cell, this allows the computer vision to calibrate
12. Once the car has been identified a countdown will begin

13. After the countdown has ended the timer will start and the car will start driving autonomously if a strategy file was selected or manual control will be enabled if manual was selected. Lap times can be triggered and the race ended by using the keyboard.

Xbox 360 controller

• Menu:
  - o D-Pad: Navigation
  - o A: Select menu item / enable car
  - o B: Disable car / exit scenario
  - o X / Y: Change strategy file

- Scenario:
  - o B: Exit to menu
  - o Xbox button: Enable debug mode

- Manual mode:
  - o Right trigger: Hold to move forward – speed is determined by strength of press
  - o Left trigger: Hold to move backwards – speed is determined by strength of press o Left stick: Steering
  - o X: Hold to activate horn
  - o A: Stop the horn
  - o B: Toggle police siren
  - o Y: Toggle headlights
  - o Left/right bumpers: Toggle left/right indicator lights
  - o Xbox button: Enable debug mode

Keyboard

- Menu:
  - o Arrow keys: Navigation
  - o Enter: Select menu item / enable car
  - o Escape: Disable car / exit scenario
  - o Space / Backspace: Change strategy file
  - o **Shift + Escape: Exit the program**

  Scenario:
  - o Escape: Exit to menu
  - o Space: Trigger lap timer
  - o Shift + Space: Finish race
  - o **Shift + Escape: Exit the program**

Code

Structure – what each of the files does

- maps – contains all the maps that will be available in the program
- media – contains misc. images
- strategies – contains the strategy files that you can access from menu

- tracker -blob_detector.py – A simple blob detector attuned to the cars shape -calibrator.py – For calibrating the camera perspective -camera.py – For reading frames from the camera -core.py – Tracker imports, classes and constants -mosse_tracker.py – The MOSSE tracker -orientation_finder.py – Methods for determining car orientation
- zenwheels -cars.csv – Information about the fleet -comms.py – For sending commands to the cars via Bluetooth -protocol.py – The communication protocol specification
- agent.py – contains agent class, methods of the car, contains current way point index of the car from information in world.py.
- constants.py – general constants and paths
- display.py – sets up the screen which has all the menu items handles input, and sets up the joy stick to navigate the menu.
- main.py – entry point
- vehicle.py – sets up the properties of the vehicle
- vision.py – works with the computer vision, calibrates and tracks the cars location.
- world.py – updates the location of the car