Version : 0.1
Authors :

Cruise control Project
User Manual

Crtl Alt Elite :

Euhnye YOON - Piyuan SONG - Abishek Anil MUDALE - Alexandre LIN - Adrien SALLÉ

Teachers :

Ms LETOURNEUR – Mr PERRIER

Introduction

This project uses modular design and manufacture cruise control system. The modular system is provided by Lego. The project is divided into three parts: bronze, silver and gold. The bronze goal will enable the car to find its own way and monitor obstacles, while the silver goal will enable the car to navigate more complex routes, with the ultimate goal being that the car can detect roadside panels. This article will provide you with information about Lego, sensors and how to use the software.

Table of content

1. Requirements

1.1. Cruise control

Cruise control system or CCS. The use of electronic technology, within a certain speed range, the driver does not control the accelerator pedal, but can ensure that the car at a set speed of a stable electronic control device. The car equipped with this device on the highway, can save the driver frequently step on the accelerator this person for action and automatically maintain the pre-set speed, which can greatly reduce the fatigue of the driver, improve the stability, safety, comfort and fuel economy when driving.

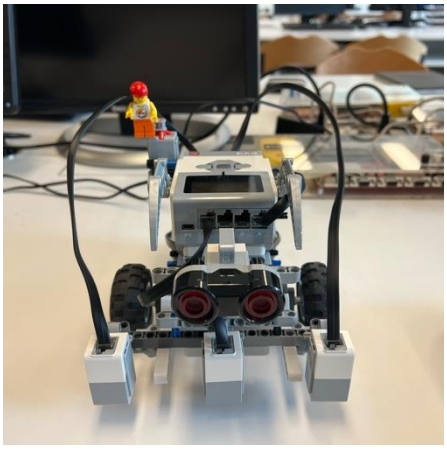

Figure 1: Cruise control verification vehicle

The project is based on the LEGO EV3, designed and produced by CtrlAltElite team.

1.2. Line Following And Map

Different objectives require different test routes, so we need to use the drawing tool to create the required map. Use a simple drawing tool and split the map into different parts so that you can use it multiple times and splice it into different maps.

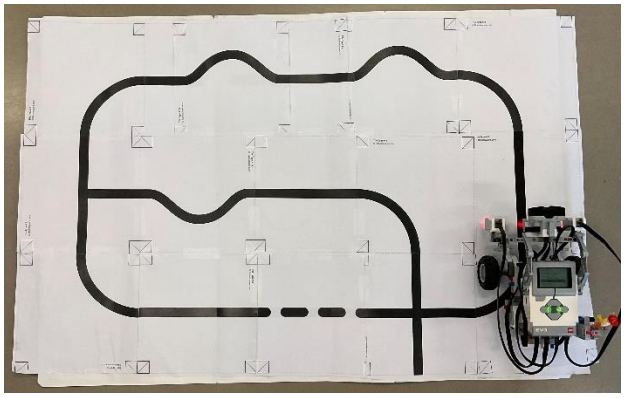

Figure 2: Silver target map sample

Use tape to paste the map on the back, and put cardboard on the bottom to ensure that the map is smooth, so that the test section is more accurate.

Test only need to put the sensor and the route to it, start the car on the computer to complete the route test.

1.3. Assembly of robots

This time, Lego robot EV3 series is selected. The box contains a simple assembly process book, EV3 series sensors, a controller and numerous parts. Select some of the parts to complete the project. It is divided into hardware part and software part, which will be explained one by one in the following sections.

Figure 3: The Lego EV3

2. Hardware

2.1. Frame of vehicle

Cruise control The Lego EV3 car features a simple frame design. The Lego EV3 comes with a handy build manual in the parts box. By building the manual, we can build a complete simple vehicle step by step. We have two motors on each side of the vehicle that act as driving motors and power the Lego car. The EV3 bricks are connected at the top to communicate with the vehicle, kept intact with the help of different Lego parts and connectors. At this point the simple vehicle installation is complete.

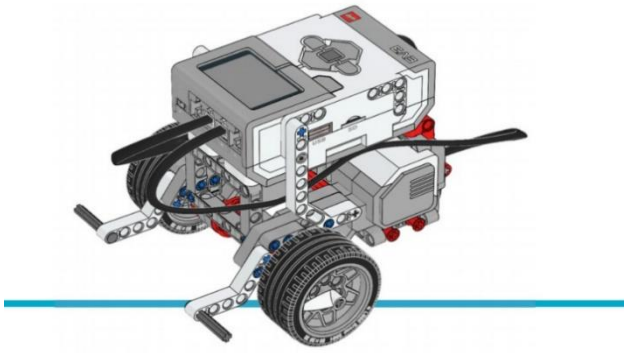

Figure 4: Example diagram of ev3 route

2.2. Selection and installation of sensors

In the present Cruise control car there are in total 5 sensors used, the selection of these sensors is done by the means of their properties such as there are 3 color sensors used to detect the black line and follow the path upfront above which an ultrasonic sensor is connected to detect the obstacle coming in front of the vehicle, this sensor is placed at a slight elevation above the color sensor and finally a push sensor is used to guide the vehicle to start and stop which is placed conveniently next to the Lego EV3 brick.

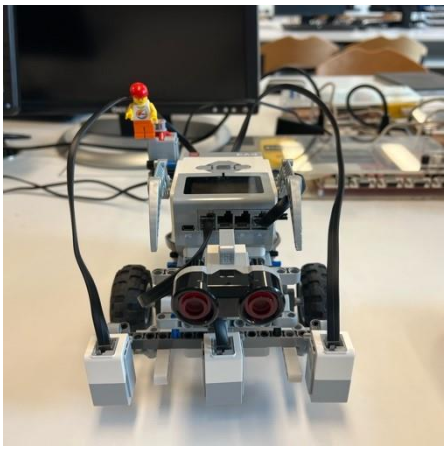

Figure 5: Sensor Installation Example

2.3. Connection between ev3 and sensor

The connection application is easy to implement with the help of various Lego pieces which have unique shape and dimensions and small connector joints, these makes a Lego vehicle simple to construct but also gives the freedom to construct a complex model. In this vehicle the car is constructed in a way that the EV3 brick is faced up which makes it easy to handle and operate alongside a push sensor and three color sensors at front placed equally distant from each other and an ultrasonic sensor placed just above the color sensors to detect the obstacles, these sensors and EV3 brick are mutually connected by means of multiplexer placed behind the Lego EV3 brick.

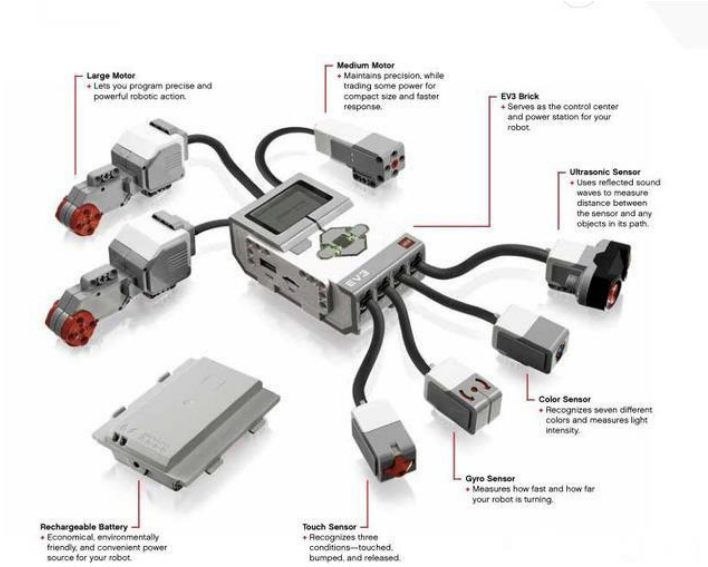

Figure 6 :Sensor connection method

In most cases, the sensor interface provided by the EV3 controller is adequate. If the number of sensors required exceeds the number of interfaces, a multiplexer is required. The multiplexer can provide three additional interfaces, allowing access to more sensors. Just connect the main head to the EV3 controller and connect the additional sensors to C1.C2 and C3.

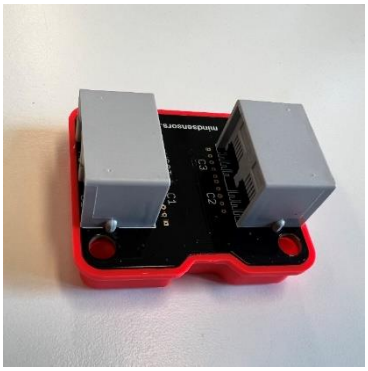

Figure 7 : Lego multiplexer

3. Software

3.1. Integrated development environment

EV3 of Lego comes with EV3 brick, and ev3dev can be used as the operating system of brick. This is a Linux environment. To create our code, we used Visual Studio code, which allows us to do autocompletion and basic spell checking as we code.

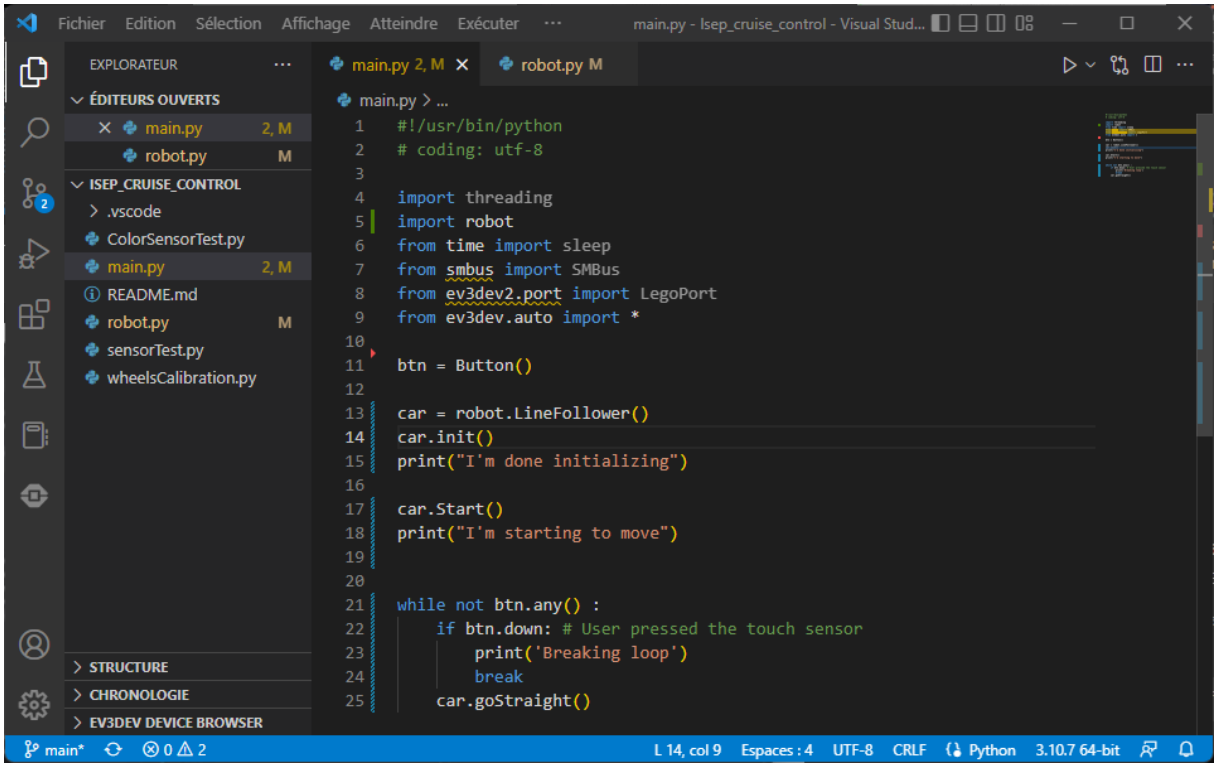

Figure 8 : Screenshot of Visual Studio Code

Visual Studio code can be searched and downloaded using google, and the tutorial can be viewed on the official website. Create a Linux code environment in the software and edit it.

Along with the extension ev3dev-browser for Visual Studio Code, we are able to send our program to the board through ssh.

3.2. Ev3dev library

To ensure the stability of our robot, we decided to use the PID control technique. This method is highly used in the robotic domain and has the advantage of being efficient and quick to set up, although the premise might be though to understand for beginners.

The following example is the PID block diagram required for the silver target. You can adjust values of PID controller (kp, ki, kd) depends on your environment.

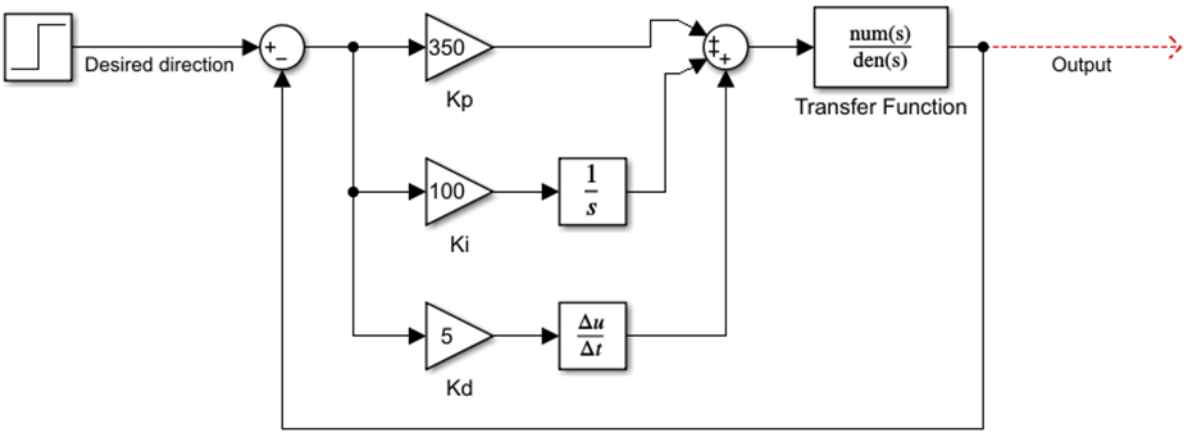

Figure 9 : PID control model

Table of figures

Annexes

```
#!/usr/bin/python
# coding: utf-8

import threading
from time import sleep

from ev3dev.auto import *

# ------Input--------
power = 30
target = 43
kp = float(0.85) # Proportional gain, Start value 1
kd = float(0.46)         # Derivative gain, Start value 0
ki = float(0.15) # Integral gain, Start value 0
direction = 1 # Direction (-1 for black line on the left or 1 for black line on the right)
minRef = 11 # Sensor min value
maxRef = 89 # Sensor max value
# --------------------

#SOUND init
speaker = Sound()
speaker.speak("Hellow")

#MUX Init
muxC1port = LegoPort("in1:i2c80:mux1")
muxC1port.set_device="lego-ev3-us"
sleep(1) # need to wait for sensors to be loaded. 0.5 seconds is not enough.

ul = UltrasonicSensor("in1:i2c80:mux1"); assert ul.connected

# Sensors
col= ColorSensor(INPUT_3);                    assert col.connected
col_left = ColorSensor(INPUT_2)
col_right = ColorSensor(INPUT_4)

# Change color sensor mode
col.mode = 'COL-REFLECT'

btn = Button()
```

```
#Global values
ulCorrection = 0
turnRight = False
turnLeft = False

#Motors
left_motor = LargeMotor(OUTPUT_D);  assert left_motor.connected
right_motor = LargeMotor(OUTPUT_A); assert right_motor.connected



#Function for ultrasonic sensor
def readingSensors():
        distance = ul.value()/10
        #Computing a correction depending on the measured distance
        if distance < 20 and distance > 10 :
                correction = -100 / distance
        elif distance <= 10 :
                correction = - power
        else :
                correction = 0
        return correction

#Function called to compute power for each motors
def steering(course, power,correction):
        power_left = power_right = power + correction
        s = (50 - abs(float(course))) / 50

        if course >= 0:
                power_right *= s
                if course > 100:
                        power_right = - power
        else:
                power_left *= s
                if course < -100:
                        power_left = - power

        return (int(power_left), int(power_right))

#Main function
def run(power, target, kp, kd, ki, direction, minRef, maxRef):
```

```
lastError = error = integral = 0
left_motor.run_direct()
right_motor.run_direct()

#thread = threading(readingSensors)


while not btn.any() :
        if btn.down: # User pressed the touch sensor
                print('Breaking loop')
                break


        ulCorrection = readingSensors()
        #Reading sensors for crossroad & interruption detection
        refRead = col.value()
        val_left = col_left.value()
        val_right = col_right.value()



        if (val_left < 20) & (val_right < 20):
                print('LEFT : ' + str(val_left) + ' RIGHT : '+str(val_right))
                if kp == float(1.2) :
                        kp = float(0.8)
                elif kp == float(0.8) :
                        kp = float(1.2)
                integral = 0
                turnLeft(kp)
        if (val_left > 70) & (val_right > 70) & (refRead > 70):
                print('Interuption')
                straightForward(pow)
        else:
                #PID computing
                print("PID Control")
                error = target - (100 * ( refRead - minRef ) / ( maxRef - minRef ))
                derivative = error - lastError
                lastError = error
                integral = float(0.5) * integral + error
                course = (kp * error + kd * derivative +ki * integral) * direction

                for (motor, pow) in zip((left_motor, right_motor), steering(course, power,ulCorrection)):
```

```
                motor.duty_cycle_sp = pow

                sleep(0.01) # Aprox 100 Hz

def turnLeft(kp):
        speaker.speak("CROSSROAD " + str(kp))
        left_motor.duty_cycle_sp = 55
        right_motor.duty_cycle_sp = 0
        sleep(1)

def turnRight():
        speaker.speak("CROSSROAD!")
        left_motor.duty_cycle_sp = 0
        right_motor.duty_cycle_sp = 60
        sleep(1)

def straightForward(pow):
        left_motor.duty_cycle_sp = pow
        right_motor.duty_cycle_sp = pow
        sleep(0.5)



run(power, target, kp, kd, ki, direction, minRef, maxRef)

# Stop the motors before exiting.
print ('Stopping motors')
left_motor.stop()
right_motor.stop()
```