Arduino Version

After the program is downloaded, the car chassis performs actions in the following set order:

① **Move forward for 4 seconds.**

② **Move backward for 4 seconds.**

③ **Turn left for 4 seconds.**

④ **Return to the initial position.**

⑤ **Forward to the right for 4 seconds.**

⑥ **Return to the initial position.**

Each action is followed by a 1-second interval.

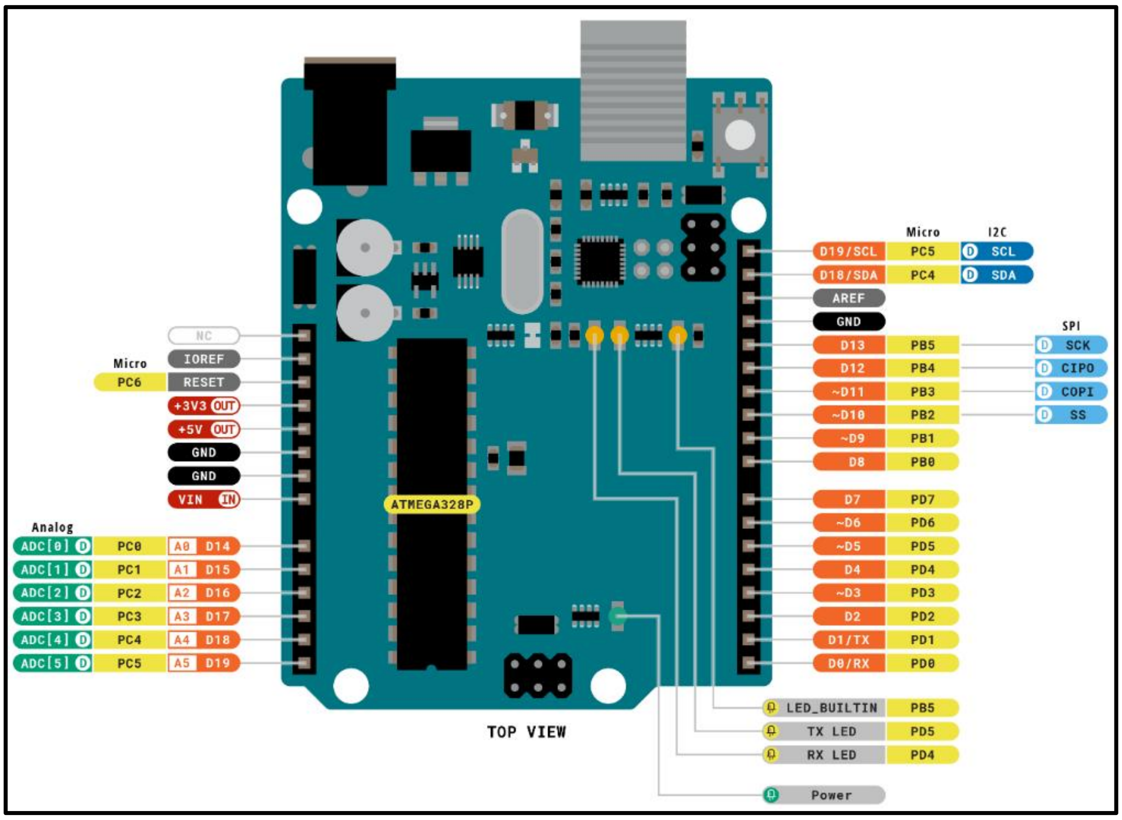
1. Hardware Introduction

1.1 Arduino UNO Controller

Arduino is a convenient, flexible, and user-friendly open-source electronic prototyping platform. It features 14 digital input/output pins (with 6 capable of PWM output), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power socket, an ICSP header, and a reset button.

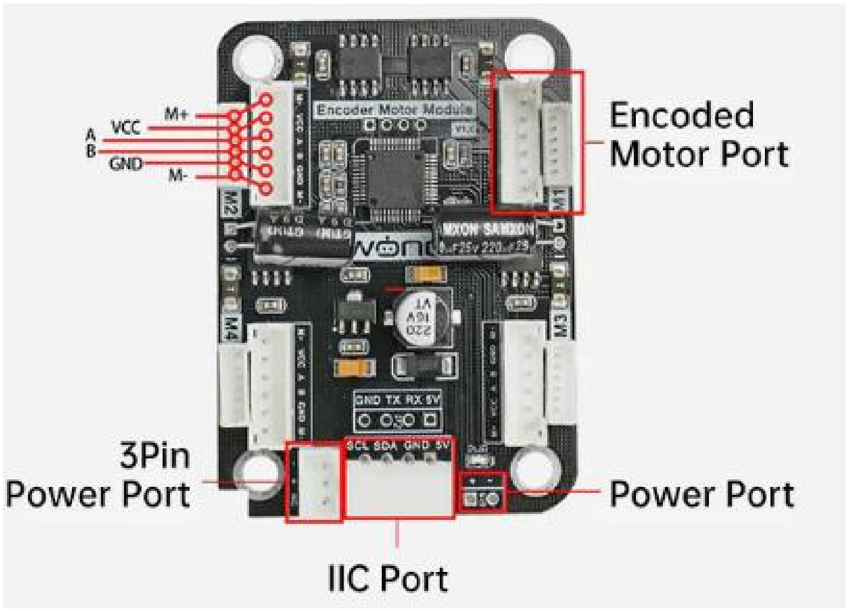
The following diagram illustrates the physical pin layout of Arduino UNO.

Please refer to your specific Arduino UNO controller for accurate details.



1.2 4-Channel Encoder Motor Driver

This is a motor drive module designed to work with a microcontroller for driving TT motors or magnetic encoder motors. Each channel is equipped with an SA8339 motor drive chip, and its voltage range is DC 3V-12V. The specific voltage depends on the voltage requirements of the connected motor. The interface distribution is illustrated in the figure below:



The introduction to the interface on the driver is as below:

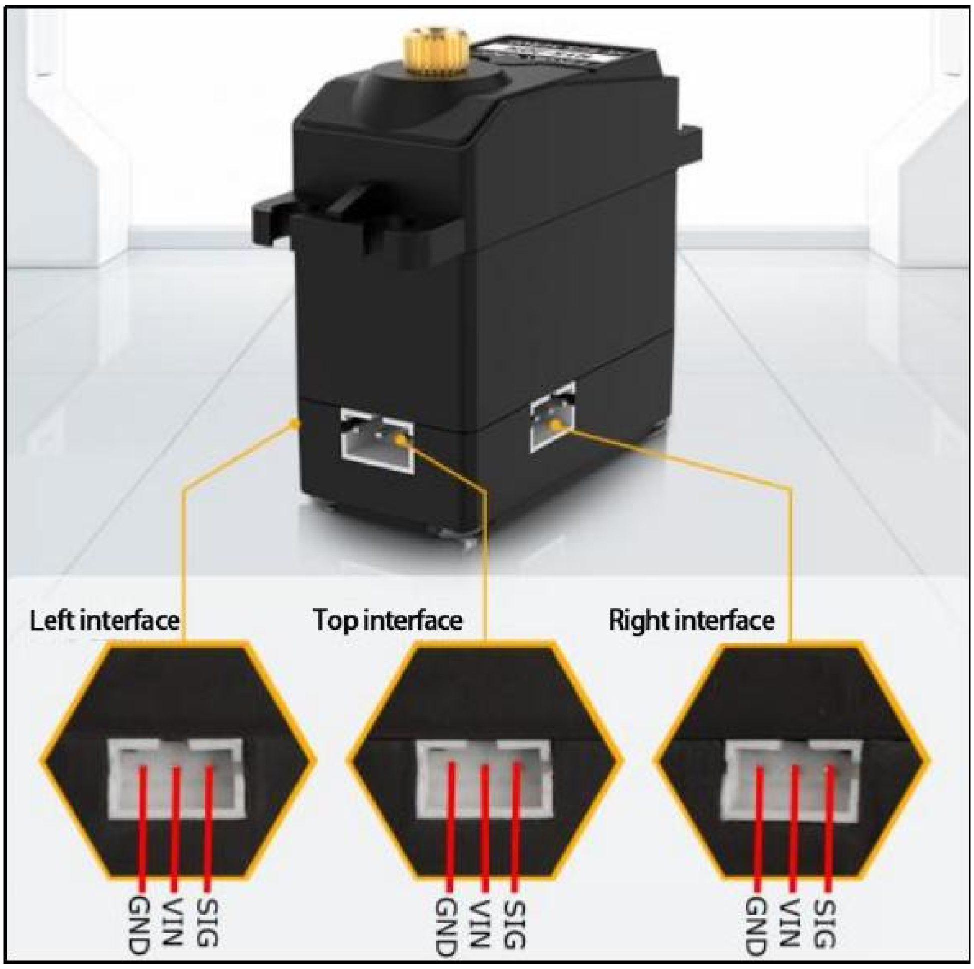
Interface type	NO.	Function
Encoder motor interface	GND	Negative electrode of the Hall power
	A	A-phase pulse signal output terminal
	B	B-phase pulse signal output terminal
	VCC	Positive electrode of the Hall power
	M+	Positive electrode of the motor power supply
	M-	Positive electrode of the motor power supply
	Note: The voltage between VCC and GND is determined based on the power supply voltage of the microcontroller used. Typically, 3.3V or 5V is used. When the spindle rotates clockwise, the output pulse signal of channel A is ahead of channel B; when the spindle rotates counterclockwise, the signal of channel A is behind channel B. The voltage between M+ and M- is determined based on the voltage requirements of the motor used.	

IIC	SCL	Clock line
	SDA	Bi-directional data line
	GND	Power ground line
	5V	5V DC output
3Pin power port	-	Power negative electrode
	+	Power positive input
	NC	Empty
Power port	+	Power positive input
	-	Power negative electrode

1.3 Steering Servo

The steering servo in this chassis utilizes the HTS-20H bus servo model.

The HTS-20H bus servo integrates servo drive, motor, and bus servo signal. It is controlled by serial commands, and the serial baud rate is 115200. You can send corresponding commands to the servo, based on the communication protocol provided with the product. This allows you to control the servo rotation or read servo information. It can be switched to stepper motor mode.



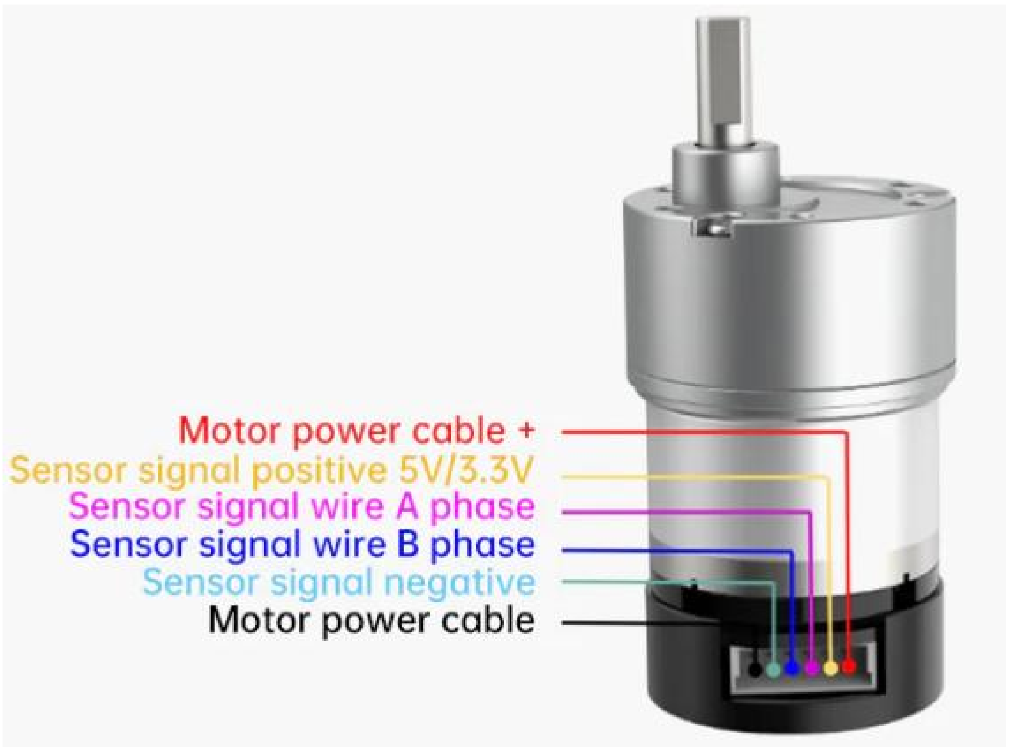
The diagram above illustrates the wiring port distribution for the bus servo, accompanied by the pin distribution table below.

Note: When the servo and the microcontroller are powered by different sources, ensure both power supplies are grounded together.

PIN	Description
GND	Ground wire
VIN	Power input
SIG	Signal terminal, which is a half-duplex UART asynchronous serial interface.

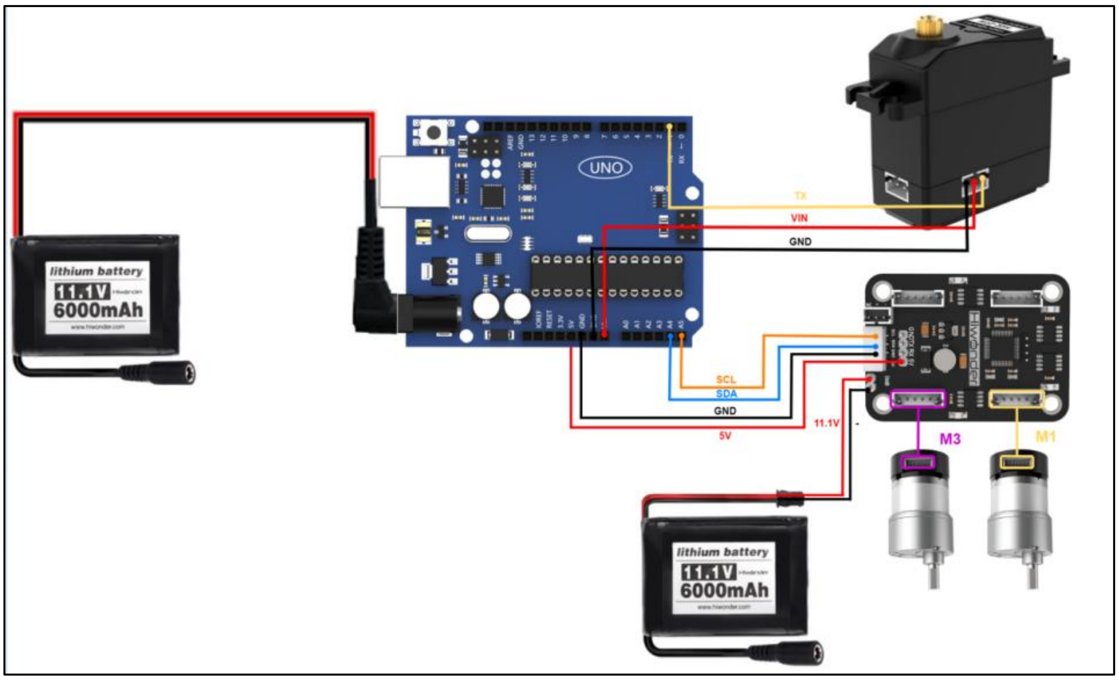
1.4 Encoder Geared Motor

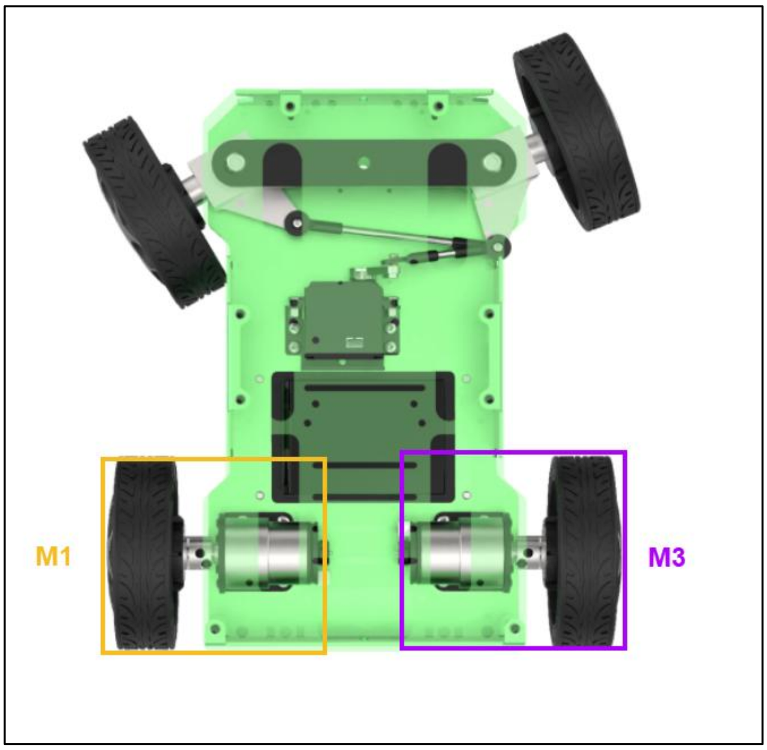
The motor model employed in this chassis is JGB37-520R90-12. Here's the breakdown: "J" signifies a DC motor, "GB" denotes an eccentric output shaft, "37" indicates the diameter of the reduction box, "520" represents the motor model, "R90" stands for the reduction ratio of 1:90, and "12" signifies the rated voltage of 12V. Please refer to the interface description illustrated in the figure below:


Motor power cable +
Sensor signal positive 5V/3.3V
Sensor signal wire A phase
Sensor signal wire B phase
Sensor signal negative
Motor power cable

2. Wiring

The Arduino Uno is outfitted with a 4-channel motor driver. It operates using an 11.1V 6000mAh lithium battery to power the motor, while a separate 11.1V battery is utilized to power the steering servo. Refer to the image below for the Arduino UNO wiring diagram.





3. Environment Configuration and Program Download

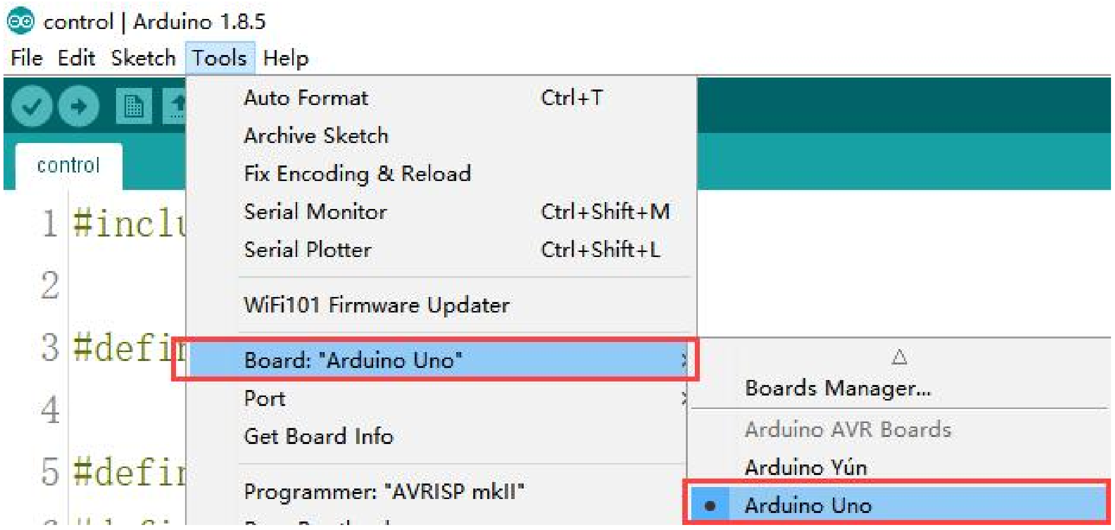
3.1 Environment Configuration

Prior to downloading, ensure that the Arduino IDE is installed on your computer. You can find the software package in the “2.Software/2.1 Arduino IDE”.

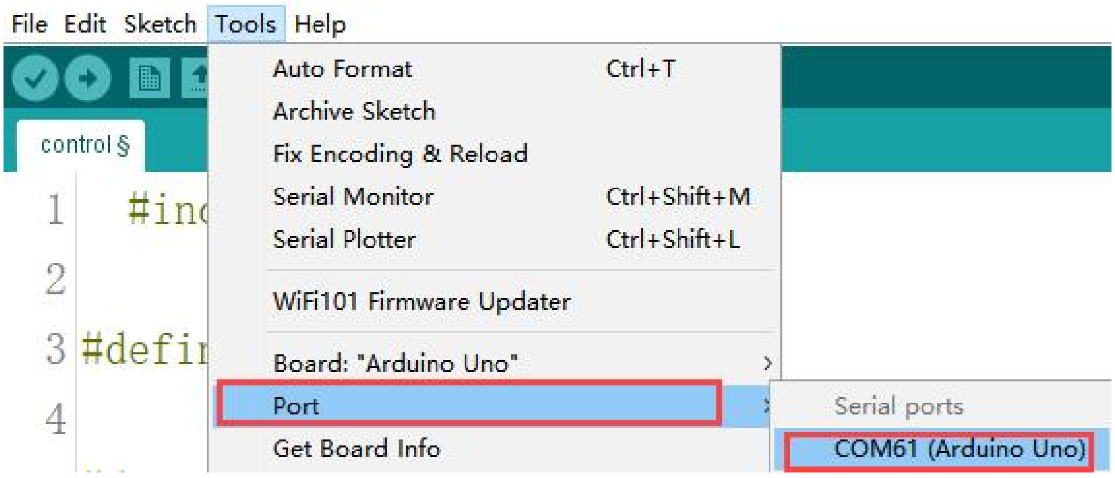
3.2 Program Running

Open the “control.ino” program saved in “2.Program File\Arduino Version” using Arduino IDE.

1) Choose the Arduino development board type. In this case, select “Arduino Uno” .

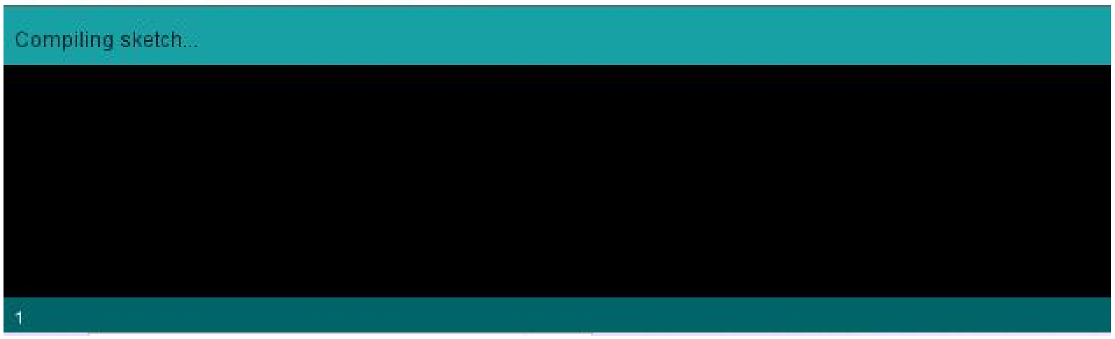


2) Select the USB port the Arduino currently connecting to your computer. The IDE will detect it automatically; in this case, choose 'COM56'.

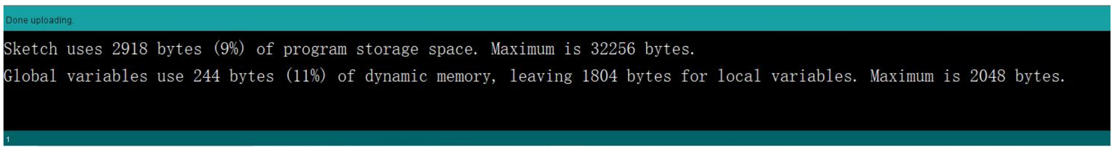


3) Connect Arduino UNO to the computer. Select ‘Arduino UNO’ in the tool bar, and click-on ➔ to download the program.

4) The software will compile the program automatically. Please wait until the compilation process is successfully completed.


Compiling sketch...

5) Wait for the program to finish uploading.


Done uploading.

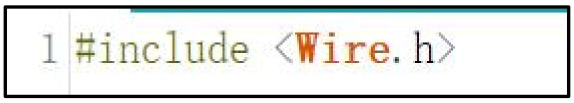
```
Sketch uses 2918 bytes (9%) of program storage space. Maximum is 32256 bytes.
Global variables use 244 bytes (11%) of dynamic memory, leaving 1804 bytes for local variables. Maximum is 2048 bytes.
```

3.3 Program Outcome

Once the program is downloaded, the car chassis executes the following actions in sequence: 1. Move forward for 4 seconds; 2. Move backward for 4 seconds; 3. Turn left for 4 seconds; 4. Return to the initial position; 5. Move forward to the right for 4 seconds; 6. Return to the original position. There is a 1-second interval between each action.

4. Program Analysis

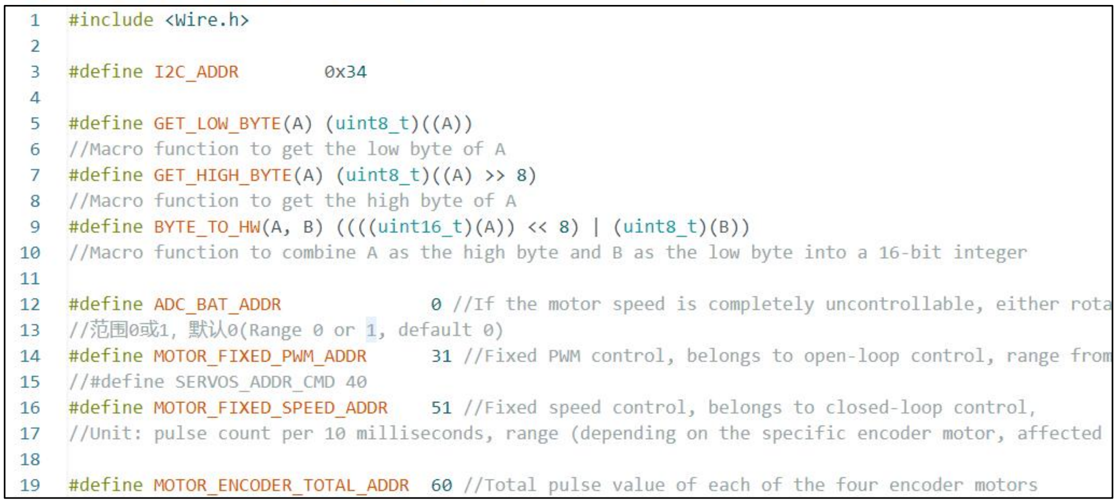
- **Import Necessary Library**



```
#include <Wire.h>
```

The library is integrated into the Arduino IDE. To add it, navigate to '**Sketch -> Include Library**.' It incorporates write methods for I2C communication, enabling the control of motor rotation.

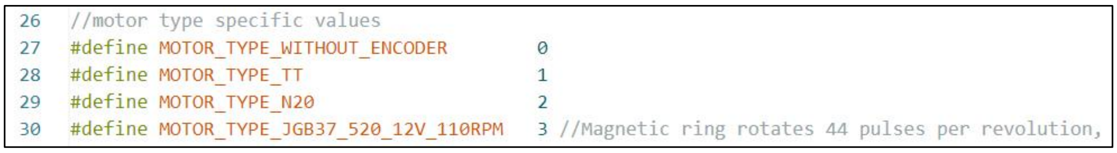
- **Initialize Communication Address**



```
#include <Wire.h>

#define I2C_ADDR        0x34

#define GET_LOW_BYTE(A) (uint8_t)((A))
//Macro function to get the low byte of A
#define GET_HIGH_BYTE(A) (uint8_t)((A) >> 8)
//Macro function to get the high byte of A
#define BYTE_TO_HW(A, B) ((((uint16_t)(A)) << 8) | (uint8_t)(B))
//Macro function to combine A as the high byte and B as the low byte into a 16-bit integer

#define ADC_BAT_ADDR                0 //If the motor speed is completely uncontrollable, either rota
//范围0或1，默认0(Range 0 or 1, default 0)
#define MOTOR_FIXED_PWM_ADDR      31 //Fixed PWM control, belongs to open-loop control, range from
//#define SERVOS_ADDR_CMD 40
#define MOTOR_FIXED_SPEED_ADDR    51 //Fixed speed control, belongs to closed-loop control,
//Unit: pulse count per 10 milliseconds, range (depending on the specific encoder motor, affected

#define MOTOR_ENCODER_TOTAL_ADDR  60 //Total pulse value of each of the four encoder motors
```

Define the I2C communication address and address codes for different types of encoded motors as macros, facilitating subsequent calls. The I2C communication address connected to the driver board is set as 0x34; this value is hardware-specific, and the default can be retained here. The encoded motor type is designated as 20, and its direction polarity is defined as 21. It's essential to note that these two numbers represent the address positions for writing parameters, not the actual parameter values. These values are hardware-specific, and for simplicity, the default values can be maintained in this context.

- **Initialize Motor Type**



```
//motor type specific values
#define MOTOR_TYPE_WITHOUT_ENCODER        0
#define MOTOR_TYPE_TT                     1
#define MOTOR_TYPE_N20                    2
#define MOTOR_TYPE_JGB37_520_12V_110RPM   3 //Magnetic ring rotates 44 pulses per revolution,
```

The 4-channel motor driver module is versatile and supports different motor types, such as TTL, N20, and JGB motors. Here, we use macro definitions to specify these types. For this development, JGB motors are employed, and their motor type is macro-defined as 3.

- **Declare Servo Control Function**

```
//Control the servo to rotate to a certain position
void LobotSerialServoMove(HardwareSerial &SerialX, uint8_t id, int16_t position, uint16_t time)
{
  byte buf[10];
  if(position < 0)
    position = 0;
  if(position > 1000)
    position = 1000;
  buf[0] = buf[1] = 0x55;
  buf[2] = id;
  buf[3] = 7;
  buf[4] = 1;
  buf[5] = GET_LOW_BYTE(position);
  buf[6] = GET_HIGH_BYTE(position);
  buf[7] = GET_LOW_BYTE(time);
  buf[8] = GET_HIGH_BYTE(time);
  buf[9] = LobotCheckSum(buf);

  SerialX.write(buf, 10);
}
```

In this function, four parameters need to be passed in. The first one is SerialX, which represents the method of signal transmission. Generally, the value passed in is the serial port “Serial”. The specific usage of this value will be used in the process of controlling the movement of the car chassis later. The second parameter is id, which is the ID number of the bus servo. Since the bus servo has the function of setting the servo ID number, if the ID number of the current bus servo is unknown when you receive the car, use the parameter 254 to control all the unknown bus servos. The third parameter is position, which is used to control the rotation position of the servo. Similarly, this parameter will be used in the process of controlling the movement of the car chassis. The last parameter is time, which controls the time required for the servo to rotate to the target position, in units of milliseconds.

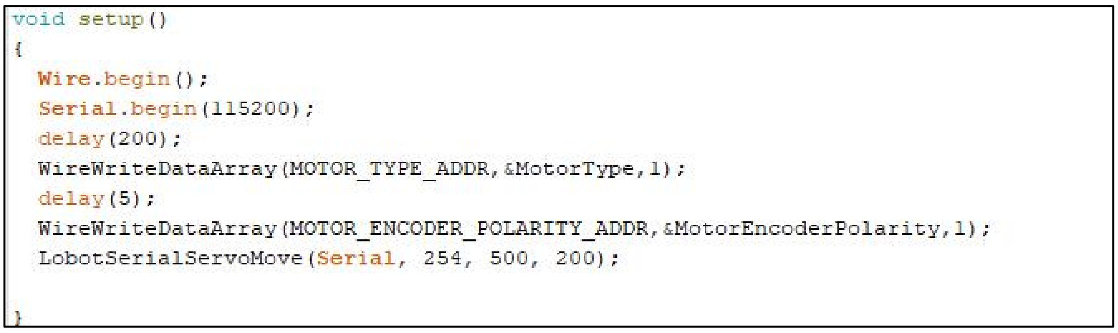
- **Configure & Control Motor**

```
uint8_t MotorType = MOTOR_TYPE_JGB37_520_12V_110RPM;     //Motor mode setting
uint8_t MotorEncoderPolarity = 0;     //Motor polarity setting

int8_t car_forward[4]={-23,0,23,0};    //Forward
int8_t car_retreat[4]={23,0,-23,0};    //Backward
int8_t car_stop[4]={0,0,0,0};
```

The lines "MOTOR_TYPE_JGB37_520_12V_110RPM" and "MotorEncoderPolarity = 0" serve to define the motor type and encoder polarity, respectively.

"MOTOR_TYPE_JGB37_520_12V_110RPM" is a predefined constant representing a specific motor model with a 12V voltage requirement and a maximum speed of 110 rpm. Meanwhile, "MotorEncoderPolarity" being set to 0 indicates that the encoder polarity is configured to its default setting.

- **Initialization**



```
void setup()
{
  Wire.begin();
  Serial.begin(115200);
  delay(200);
  WireWriteDataArray(MOTOR_TYPE_ADDR,&MotorType,1);
  delay(5);
  WireWriteDataArray(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1);
  LobotSerialServoMove(Serial, 254, 500, 200);

}
```

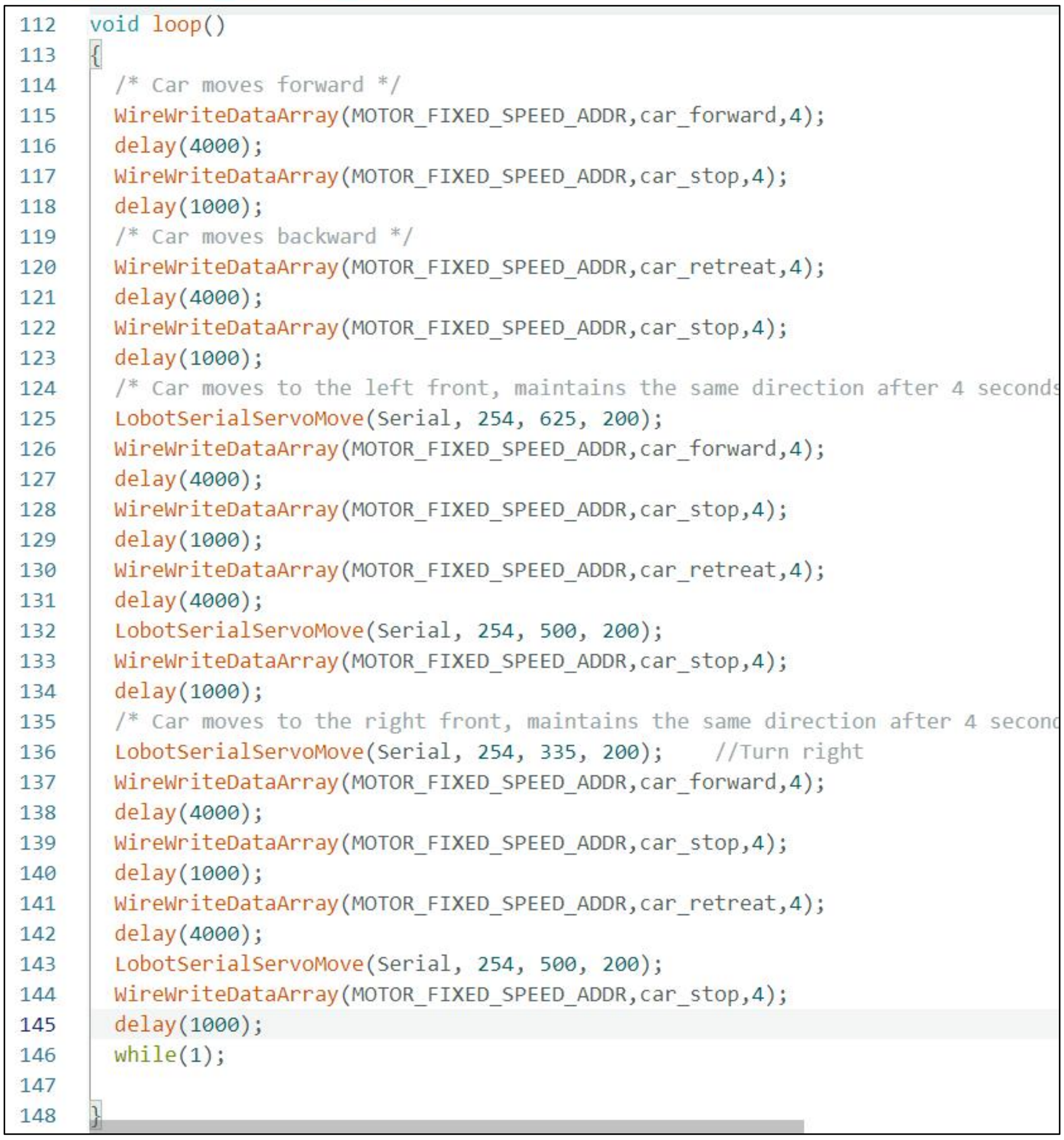
The function "Wire.begin();" initiates I2C communication.

The function "Serial.begin();" is used to set the baud rate of serial port to communicate with the bus servo. Set the baud rate to 115200.

"WireWriteDataArray(MOTOR_TYPE_ADDR,&MotorType,1)" and "WireWriteDataArray(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1)" utilize the I2C protocol to transmit the motor type and encoder polarity settings to specified addresses.

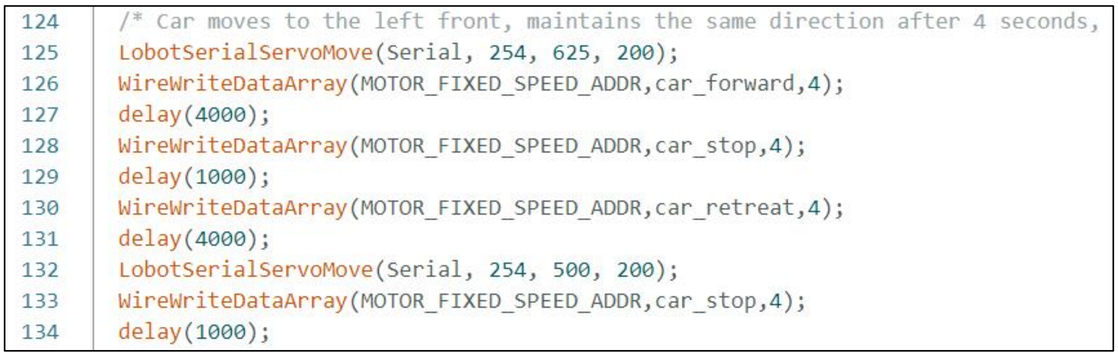
The function “LobotSerialServoMove(Serial, 254, 500, 200)” is used to control the rotation of the bus servo. Serial indicates the use of serial communication. 254 represents the ID of the servo. When the servo has no ID by default, or you want to control all connected bus servos, set the value to 254. 500 indicates the position of the rotation, which is set to the neutral position. 200 represents the time of rotation, in units of milliseconds.

- **Main Function**



```
void loop()
{
  /* Car moves forward */
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_forward,4);
  delay(4000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  /* Car moves backward */
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_retreat,4);
  delay(4000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  /* Car moves to the left front, maintains the same direction after 4 second
  LobotSerialServoMove(Serial, 254, 625, 200);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_forward,4);
  delay(4000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_retreat,4);
  delay(4000);
  LobotSerialServoMove(Serial, 254, 500, 200);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  /* Car moves to the right front, maintains the same direction after 4 secon
  LobotSerialServoMove(Serial, 254, 335, 200);    //Turn right
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_forward,4);
  delay(4000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_retreat,4);
  delay(4000);
  LobotSerialServoMove(Serial, 254, 500, 200);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  while(1);

}
```

In the main function, the motor's operating mode is determined using the "WireWriteDataArray" function. Let's focus on the left turning of the car as an example.



```
  /* Car moves to the left front, maintains the same direction after 4 seconds,
  LobotSerialServoMove(Serial, 254, 625, 200);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_forward,4);
  delay(4000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_retreat,4);
  delay(4000);
  LobotSerialServoMove(Serial, 254, 500, 200);
  WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,car_stop,4);
  delay(1000);
```

In "LobotSerialServoMove(Serial, 254, 625, 200)", "Serial" indicates the use of serial communication. 254 represents the ID address of the bus servo. Setting it to 254 means that all bus servo addresses on the serial port are selected. 625 represents the current position of the bus servo. The value for the neutral position of the servo is 500. Therefore, the value for rotating the bus servo in the opposite direction to the one on the car should be less than 500. It is recommended to set the range to 335 to 625. 200 represents the time required for the servo to reach the target position after rotating to the target point. "WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR, car_retreat, 4)" establishes the forward mode. This mode corresponds to the "int8_t car_forward[4] = {-23,0,23,0}" function, which sets the motor rotation speed for the M1 and M3 interfaces. Given that the motors' installation directions are opposite, the speed control values in "car_forward" include both positive and negative values -23 and 23. Adjusting the absolute value of this parameter allows fine-tuning of the motor speed: the larger the absolute value, the faster the speed. Subsequent control of the car chassis to achieve backward movement can be implemented by changing the “car_forward” to “car_stop”.

5. Development Notices

1) Given that the Arduino Uno operates at a rated working voltage of 5V, the encoding motor utilized in this car chassis (Ackerman) requires a 12V power supply for operation.

2) Utilizing the IIC interface solution of the 4-ch encoded motor drive module to provide power is unfeasible. This is because the 5V provided by this interface serves solely as a voltage input and cannot be used for output. Additionally, it is not advisable to employ other interfaces of the motor driver module to power the Arduino Uno, as this could lead to unstable voltage output from the Arduino Uno.

3) To ensure the steering gear, responsible for controlling the vehicle's steering, receives its required operating voltage, a 11.1V lithium battery is

employed in the tutorial. This battery powers the Arduino Uno main control board, thereby facilitating the steering gear's operation through the corresponding pin interface.