Project 3: Autonomous Car

Presented by Leo Zhao

Table of contents

Project Goals

Can be Controlled via Bluetooth

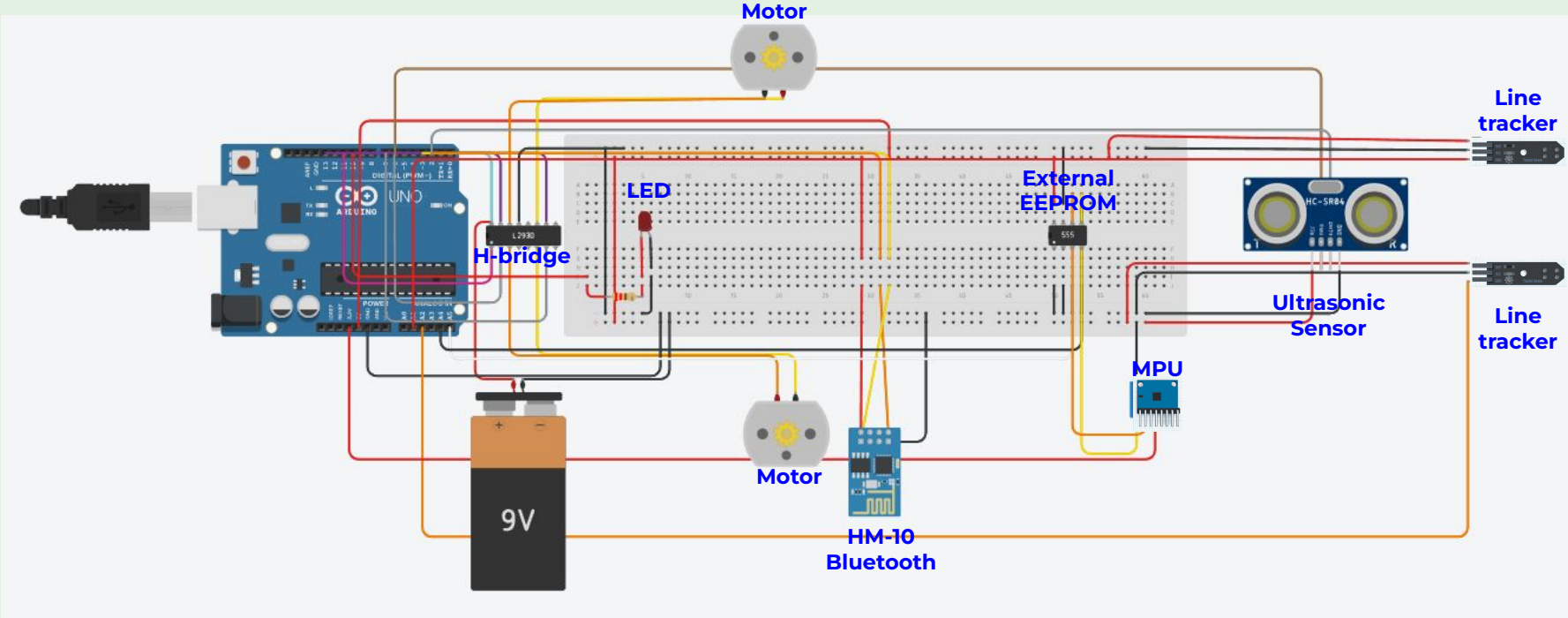
Crash Avoidance

Line Tracking

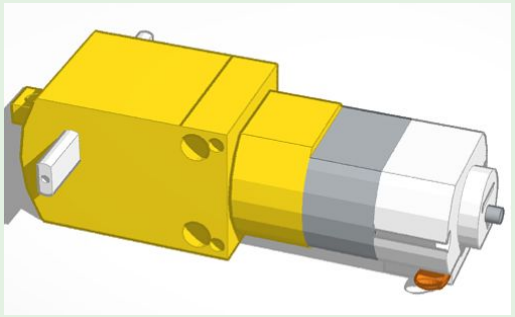
Record Acceleration in EEPROM

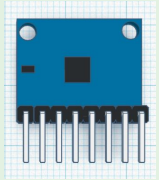
Read the Recorded Acceleration from EEPROM

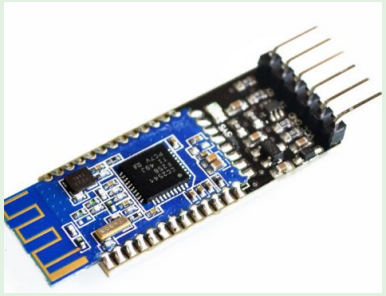
TinkerCAD Schematic

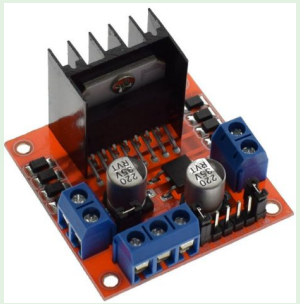


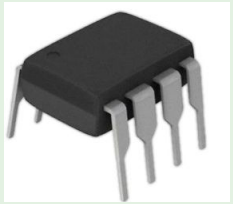
Important Parts

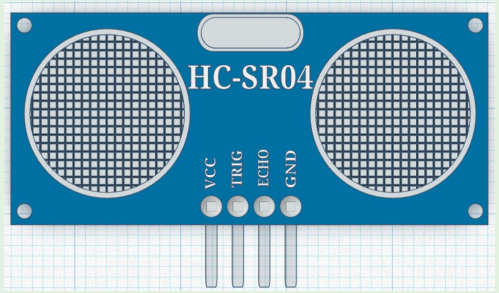












Modes of Operation

JoySticks Controlled Mode:

- Controls the vehicle with the joystick on Microblue App
- The LED is permanently off when no crash is detected
- If a crash is detected, flash the LED, and disable the vehicle from moving forward
- When the switch is pressed, switch to autonomous driving mode

Autonomous Driving Mode:

- The vehicle will follow a black line on the floor
- The LED is permanently on when no crash is detected
- If a crash is detected, flash the LED, and stop the vehicle completely
- When the switch is pressed, switch to joystick controlled mode

Explanation of Code

MPU.h

- Uses Wire.h (I2C protocol)
- void recordAccelRegisters(): reads **acceleration data** from the acceleration registers
- void processAccelData(): convert acceleration data to **units of G's**
- void recordGyroRegisters(): reads **gyro data** from the gyroregisters
- void processGyroData(): convert gyro data to **units of degrees**

RGB_TOGGLE.h

- void flashLED(): flash the red LED every 100 milliseconds using **time-controlled** if loops
- void turnOnLED(): write HIGH to the red LED (turn it on permanently)
- void turnOffLED(): write LOW to the red LED (turn it off permanently)

LINE_TRACKING.h

- int readLineTracker(int lineTracker): read the **analog value** from lineTracker, and then **convert it to a digital value**. Return 0 if the analog value is less than 100, 1 if the value is greater than 100
- void lineTracking(): If **a crash is detected**, brake the motors; else, if **both sensors read 0 or 1**, move the car forward; **if the left sensor reads 0 and the right sensor read 1**, make the car turn right; **if the left sensor reads 1 and the right sensor read 0**, make the car turn left

Explanation of Code

EEPROM.h

- Uses Wire.h (I2C protocol)
- void writeFloatEEPROM (int deviceaddress, unsigned int eeaddress, float data): break a float into **an array of bytes**, write them in sequence in EEPROM
- float readFloatEEPROM (int deviceaddress, unsigned int eeaddress): reads an array of bytes from EEPROM, then assemble them into a float and return it
- void updateMaxAccel(): keep updating the maximums during the drive
- void writeMaxAccel(): write the **maximum X, Y, and Z accelerations** to the EEPROM
- void readMaxAccel(): read the **maximum X, Y, and Z accelerations** from EEPROM, then print to Serial

CRASH_SENSOR.h

- Modified the code from Week 14 lab to work **without delay**
- float measureDistance(): Write LOW to trigPin; After 2000 us, write HIGH to trigPin; After 2015 us, write LOW to trigPin; After 2025 us, read pinTime from echoPin, then calculate the **target distance in inches**. Return target distance
- bool detectCrash(): measure target distance; if target distance is **less than 5 inches**, brake the motors, flash the LED, return true; otherwise, turn of or off the LED based on the driving mode and return false

Drive.h

- Downloaded from this website, modified to work with my car
- void drive(int throttle, int steering): If throttle is 0, brake the motors; if throttle is greater than 0, set motors to move forward, otherwise, set the motors to move backwards. Based on throttle and steering, map the speeds into **mappedSpeed** and **reducedSpeed**. If steering is greater than 0, turn right by reducing right motor speed; otherwise, turn left by reducing left motor speed
- void joystick(String id, String value): if id is “d1”, get the throttle and steering from the joystick. if a **crash is detected**, only allow the car to move **backwards** with the joystick; otherwise, control the car to move in **any direction** using the **drive function**

Explanation of Code

In the main file:

- Downloaded from this website, modified to work with my car
- Include all the header files, then setup all sensors and BLESerial
- Create a **struct** for **Bluetooth message**
- struct BLEMessage getBLEMessage() (copied from the website): get a bluetooth message from the app if there is one; **Read id and value if the signature code 1 is detected**
- loop():
    - Detect crash
    - Get a bluetooth message from the Microblue App, if possible
    - If the **switch** on the app is flipped, **change the drive mode** between joystick driving and autonomous driving
    - if the drive mode is **joystick driving**, control the car to move with the **joystick function**
    - If the drive mode is **autonomous driving**, make the car follow a black line using the **lineTracking function**
    - Write **maximum acceleration data** to EEPROM

Data Collected During Test Drive

Unfortunately, due to some issues with the EEPROM, I was not able to write/read acceleration from the EEPROM.

I assumed that something was wrong with my EEPROM chip, but the program still did not work after I swapped a new one.

Conclusion - the Challenges

EEPROM

- Initially, I wanted to write acceleration data to the EEPROM every second until the EEPROM is full
- However, I discovered that doing so would damage the EEPROM
- Therefore, I changed the code so the max accelerations keep getting updated, but are only written to the EEPROM when a button is pressed

Line Tracking

- I realized that the position of the sensors affect the performance of the system. At first, I put the sensors on the back of the car with the motors in front, which did not work great because the car would turn in a direction where both sensors deviate from the line
- Then I put the sensors really close to the two motors, which greatly improved the accuracy of the system.
- I still had to adjust the sensitivity of the sensors to get the car stay on the line completely
- Due to the internal friction of the motors, the car would sometimes get stuck. I have yet found a solution for this problem.

Conclusion - What's next?

- Use different **motors**, so that we have more precise control over the speed and direction of the vehicle
- Improve the **line tracking algorithm**: make the vehicle follow a black line at a higher speed
- Add a **LCD screen** to the vehicle to display instantaneous data, like we did for project 2
- Implement more **autonomous features**, such as signal tracking

References

https://app.arduino.cc/sketches/1fd78c62-4b63-40cc-91d5-ef4814c58afd?view-mode=preview

Thank You for Watching