BBM434 Car Following Object Embedded Systems

Project Report

Mert IŞIK *(*mert_1535@icloud.com*)*

Mücahit FINDIK*(*mucahitfindik97@gmail.com*)*

June 22, 2019

İçindekiler

1.Introduction

Overview of the car following object

Our robot is a kind of car that follows the object in front of it. Their function is to measure the distance from the object in front of it, detect whether the object is on the right or left, power the engines according to the distances measured, and steer the vehicle.

The car following object provides the user to measure the distance as he / she moves the object and to activate the dc motors. It does this by keeping the distance to the object. Figure 1.1 illustrates the hardware layout of the car following object.

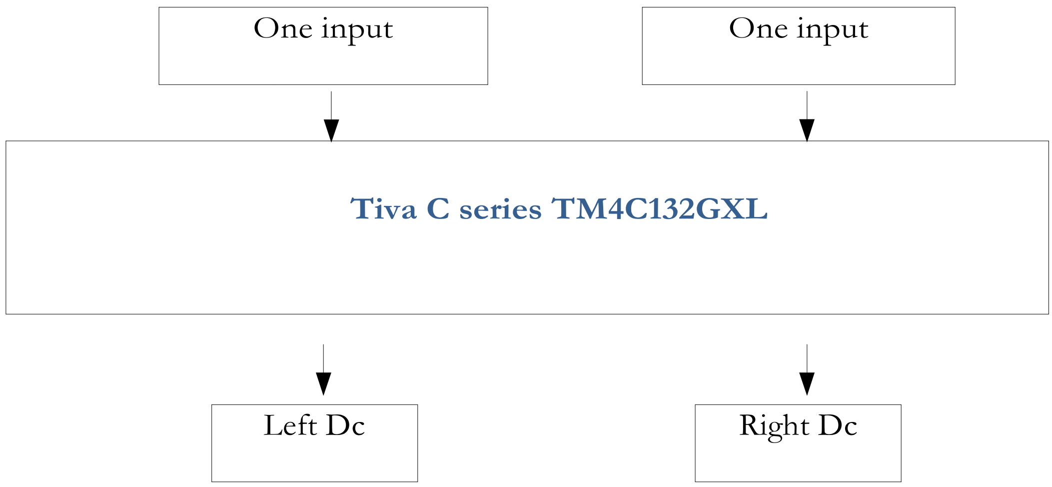


Figure 1.1.: *Hardware layout of the Car Following Object*

Real time distance measuring embedded device

The Car Following Object should measure distance reliably. Because it basically moves the motors according to the distance it measures. When the user changes the position of the object, it detects it with distance sensors and energizes the motors. User unpredictable object movements require our robot to constantly check distance. We used hardware interrupts for this. It starts the engines based on these interruptions.

Report outline

The details of the various functions of the Car Following Object will then be highlighted. The report wraps some photos of the running Car Following Object. Hardware schemes and code snippets of the project will be shown. You will also receive the complete project code.

2. Real-time measure consideration

Real-time tasks

Our vehicle following the object must measure the distance perfectly in real time. We used interrupts for this. When using the distance sensor interruption, we thought of the operating logic. The distance sensor will be explained in more detail below.

Operating logic of the distance sensor

Distance sensors have 4 pins. These are Gnd, Vcc, Echo and Trig. We used Gnd and Vcc to get power and connect the power to earth.The important thing is that we need to connect the Vcc pin to the Vbus (5V) pin of our launcpad.

We send ultraviolet rays from the sensor periodically with the echo pin. With the Echo pin, the sensor sends the rays, and these rays strike object which it is in front of the car and return. Using the trigger pin, these rays are captured at the periodic time we set by it. So we define interrupts to our trig pins.

We need to keep the time between the ultraviolet rays coming out of the sensor and caught by the sensor. Because this time will help us measure distance. We get this time using systickTime. Then we can measure the distance of the object when we divide this time with the speed of the ray.

For example, as soon as we start sending beams from the echo pin, I reset the systickTimer (t0 = 0) and activate the timer. I then check the sysctickTimer elapsed time (t1=1) when the trig pin interrupt is caught. This is the time difference between an ultraviolet ray hitting and returning. When I divide this time difference by the velocity of the ultraviolet beam ($3x10^8$m/s), I get the distance.

t1-t0 (trig time-beginning time)
v (velocity of ultraviolet ray)

X=V/T $\longrightarrow$ X=velocity of ultraviolet ray/(trig time-beginning time)

Code and Description

```c
void Timer4_Init(void(*task)(void), unsigned long period){

  SYSCTL_RCGCTIMER_R |= 0x10;   // 0) activate TIMER4
  PeriodicTask[4] = task;       // user function
  TIMER4_CTL_R = 0x00000000;    // 1) disable TIMER4A during setup
  TIMER4_CFG_R = 0x00000000;    // 2) configure for 32-bit mode
  TIMER4_TAMR_R = 0x00000002;   // 3) configure for periodic mode, default down-count settings
  TIMER4_TAILR_R = period-1;    // 4) reload value
  TIMER4_TAPR_R = 0;            // 5) bus clock resolution
  TIMER4_ICR_R = 0x00000001;    // 6) clear TIMER4A timeout flag
  TIMER4_IMR_R = 0x00000001;    // 7) arm timeout interrupt
  NVIC_PRI17_R = (NVIC_PRI17_R&0xFF00FFFF)|0x00800000; // 8) priority 4   (23-21)
// interrupts enabled in the main program after all devices initialized
// vector number 86, interrupt number 70
  NVIC_EN2_R = 1<<(70-64);      // 9) enable IRQ 70 in NVIC
  TIMER4_CTL_R = 0x00000001;    // 10) enable TIMER4A
}
```

We used Timer4 in this project. Notice that we set the priority to 4. This timer is used to determine the period of ultraviolet radiation that we will send.

```c
void SysTick_Init(unsigned long period)
{
  Period = period;   // to be used in delay functions

  NVIC_ST_CTRL_R = 0;                    // disable SysTick during setup
  NVIC_ST_RELOAD_R = period - 1 ;        // reload value (max: 2^24 - 1)
  NVIC_ST_CURRENT_R = 0;                 // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;  // priority 2
  NVIC_ST_CTRL_R = 0x07;                 // enable SysTick with core clock and interrupts

  EnableInterrupts();
}

/** @brief  Executed every (Period / CLOCK_FREQ) sec
  * @input  None
  * @output None
  */
void SysTick_Handler(void)
{
  Counts++;
}
```

We have set the SystickTimer initial settings and set its priority value to 2.This timer measures the time between the going and return of the ultraviolet ray.

```c
void OmniControl_Init(void)
{
  Timer4_Init(&Timer4_Task, 6*80000);    // timer period: 6ms
}

_
```

We used PLL and increased the speed of our launcpad to 80 MHz. So we sent a beam every six milliseconds.

```c
void Timer4A_Handler(void){
  TIMER4_ICR_R = TIMER_ICR_TATOCINT;// acknowledge TIMER4A timeout
  (*PeriodicTask[4])();             // execute user task

}
```

With Timer_Handler, we called our Timertask4 function.

```c
void Timer4_Task(void)
{
  /* Ultrasonic Distance Calculation */
  if(timerCount%4 == 0)       // if 24 ms has passed (this timing is important for proper measurement)
  {
    if(ult2.done == 1)            // if second ultrasonic sensor calculated distance
    {
      ult2.done = 0;
      Ultrasonic1_sendTrigger();  // activate first ultrasonic
    }
    else if(ult1.done == 1)       // if first ultrasonic sensor calculated distance
    {
      ult1.done = 0;
      Ultrasonic2_sendTrigger();  // activate second ultrasonic
    }
  }
  timerCount++;
```

Since we use 2 distance sensors, we need to synchronize them.For this purpose, we sent the beam in turn according to the beam sending conditions.

```c
  */
void Ultrasonic1_sendTrigger(void)
{
  GPIO_PORTB_DATA_R&=~0x04;  // Trigpin: LOW
  delay_us(2);               // wait 2 us
  GPIO_PORTB_DATA_R|=0x04;   // Trigpin: HIGH
  delay_us(10);              // wait 10 us for triggering
  GPIO_PORTB_DATA_R&=~0x04;  // Trigpin: LOW
}
```

When sending the ray from the distance sensors, we first stopped sending the ray and waited for 2 microseconds and sent the ray again for 10 microseconds. Then we stopped sending the ray again.

```c
void GPIOPortE_UltrasonicTask(void)
{
  if(GPIO_PORTE_RIS_R & GPIO_PORTE_PE0_M)     // PE0 interrupt occurred
  {
    GPIO_PORTE_ICR_R |= GPIO_PORTE_PE0_M;     // ack flag0
    ult1.flag++;

    if(GPIO_PORTE_DATA_R & GPIO_PORTE_PE0_M) // if PE0 is high
      ult1.first_time = Counts;              // measure first time
    else                                     // if PE0 is low
    {
      ult1.second_time = Counts;             // measure second time
      ult1.distMeasure = 1;
    }

    if(ult1.distMeasure == 1)  // calculate distance only after echo pin goes low
    {
      ult1.change = (ult1.second_time - ult1.first_time) / 1000.0; // time change in ms
      ult1.dist = (ult1.second_time - ult1.first_time) / 58.82;    // calculate distance in cm
      ult1.distMeasure = 0;
      ult1.done = 1;          // measurement is completed
      if(ult1.dist > 50){
          ult1.dist=50; // if the distance is greater than 50 cm, make it 50 to prevent unwanted wall following behavior.

      }
    }
```

We calculated the time between the going and return of the ultraviolet ray using Counts variable. Because of the beam emerges from the sensor and turns back, we also divide the result into two. If the result is greater than 50 cm, we made our distance variable 50.

Operating logic of dc motor

We used the L298N motor drive to control the motors. This motor drive operates in the 9V-32V range. This motor drive has 10 pins. 6 of them are input and 4 of them are output. 3 inputs and 2 outputs are used for one motor. Thus, the L298N can run 2 motors. The 2 output pins are connected directly to the motor pin. 3 input pins are used to determine the direction and speed of a motor.

In our project, we used 2 input pins because the speed of the motors is constant. The input pins determine the direction of the motor. For example, when we give 10, it goes forward, when we give 01, it goes back, and when we give 11 or 00, the engine stops.

```
void GPIO_PortA_Init()
{
  //Initializing the Ports for Motor Driver
  unsigned long volatile delay;
  SYSCTL_RCGC2_R|=0x00000001;           // activate clock for port A
  delay=SYSCTL_RCGC2_R;                 // dummy delay
  GPIO_PORTA_CR_R|=0x78;                // allow changes to PA3,4,5,6
  GPIO_PORTA_AMSEL_R&=~0x78;           // disable analog mode selection on PA3,4,5,6
  GPIO_PORTA_PCTL_R&=~0x0FFFF000;
  GPIO_PORTA_DIR_R|=0x78;              // all of them output
  GPIO_PORTA_AFSEL_R&=~0x78;            // disable alt funct on PA3,4,5,6
  GPIO_PORTA_DEN_R|=0x78;               // enable digital I/O on PA3,4,5,6


}
```

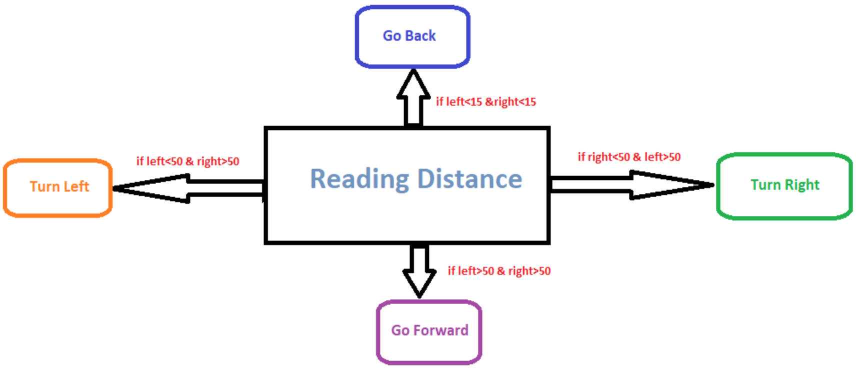
The outputs for our two motors are 3,4,5,6 of A pin.

```
#define PORTA_PIN234567 0x78;//0xFC
#define PORTA_CLK_EN 0x01;

#define FORWARD 0x28//0101
#define BACKWARD 0x50//1010
#define LEFT  0x68//1101
#define RIGHT  0x38//0111
#define BREAK  0xFC   //111111
```

In the picture of above , there are pins value to motors go forward or back.

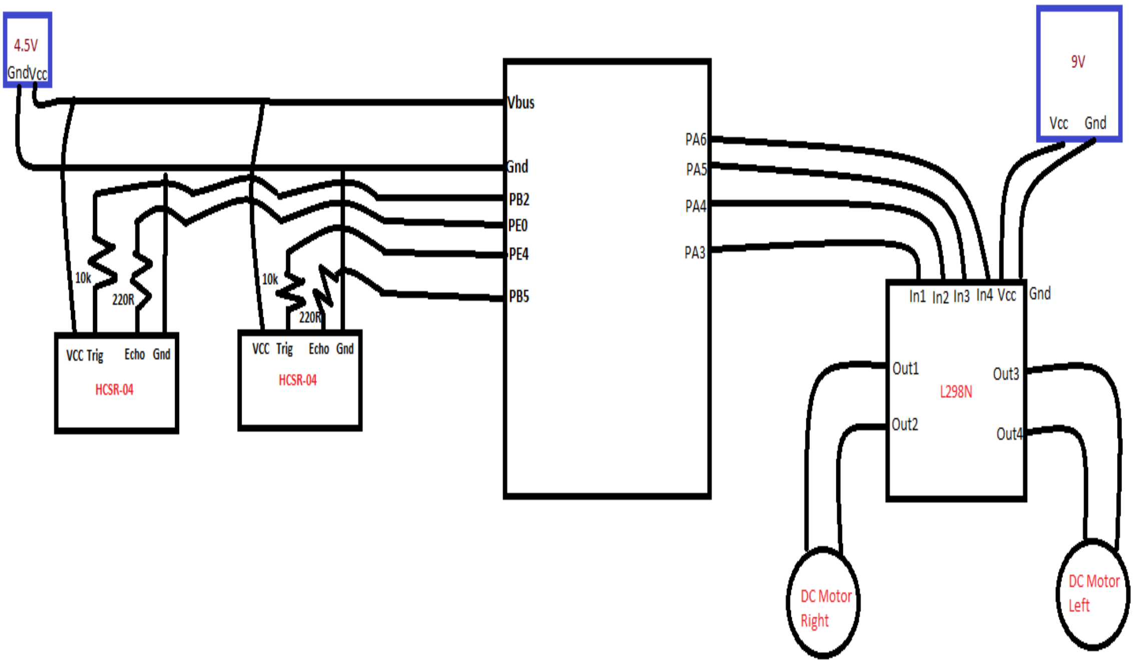
3. Algorithm Design



We have two distance sensors on the right and left. If the value of the right distance sensor is greater than 50 and the value of the left distance sensor is less than 50, it must turn left. If the value of the left distance sensor is greater than 50 and the value of the right distance sensor is less than 50, it must turn right. If the value of the right distance and left distance sensor is greater than 50, it needs to go forward. If the value of the right distance and left distance sensor is less than 15, it must go back to maintain the distance. If there is a situation other than these situations, it is enough to stop.

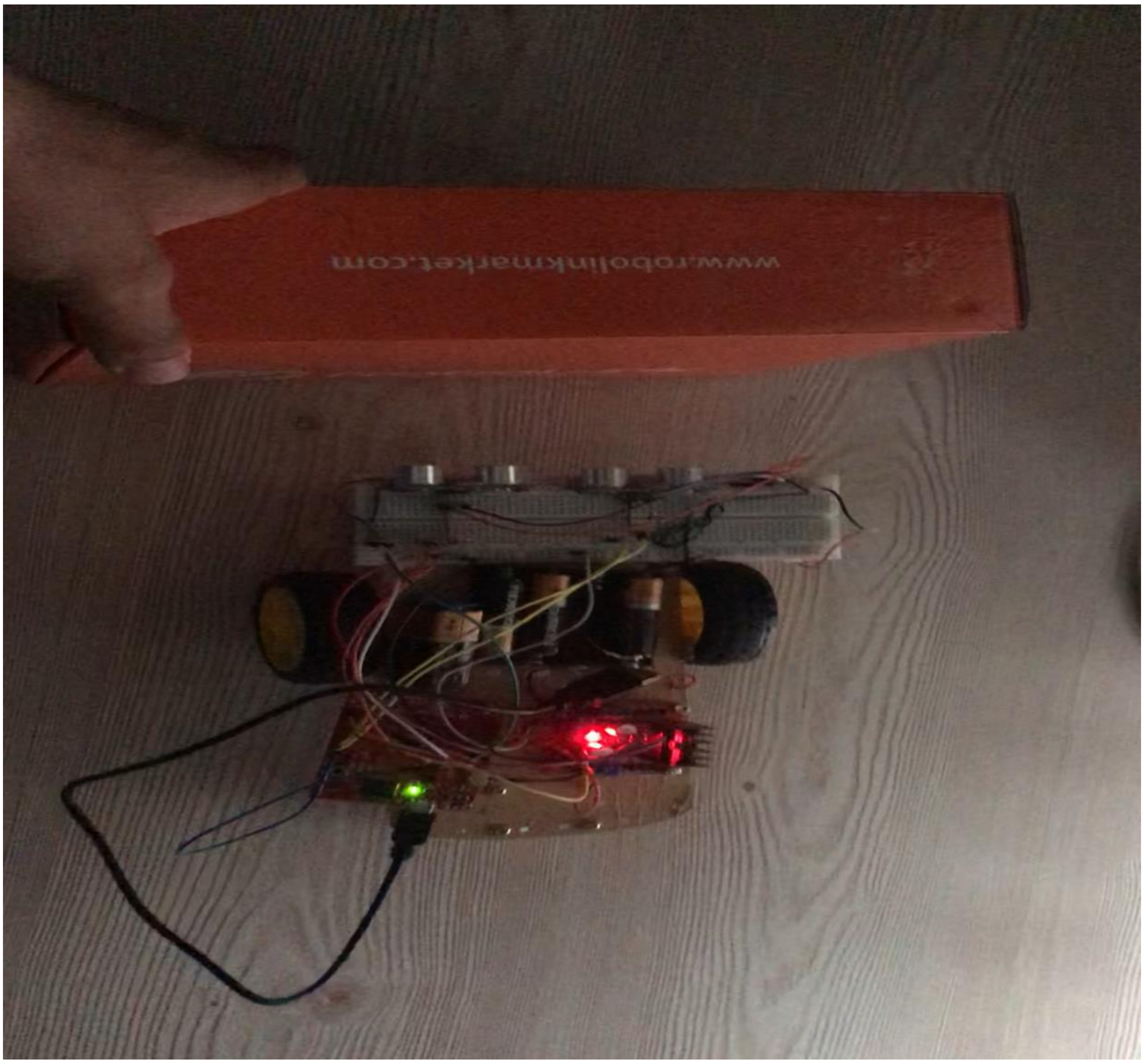
```
void MotorControl(){
  if(previous_distance1>50&&previous_distance2>50){
    moveForward();
  }
  else if(previous_distance1<15&&previous_distance2<15){
    moveBackward();
  }
  else if(previous_distance1<50&&previous_distance2>50){
    moveRight();
  }
  else if(previous_distance1>50&&previous_distance2<50){
    moveLeft();
  }
else{
  Stop();
}
```

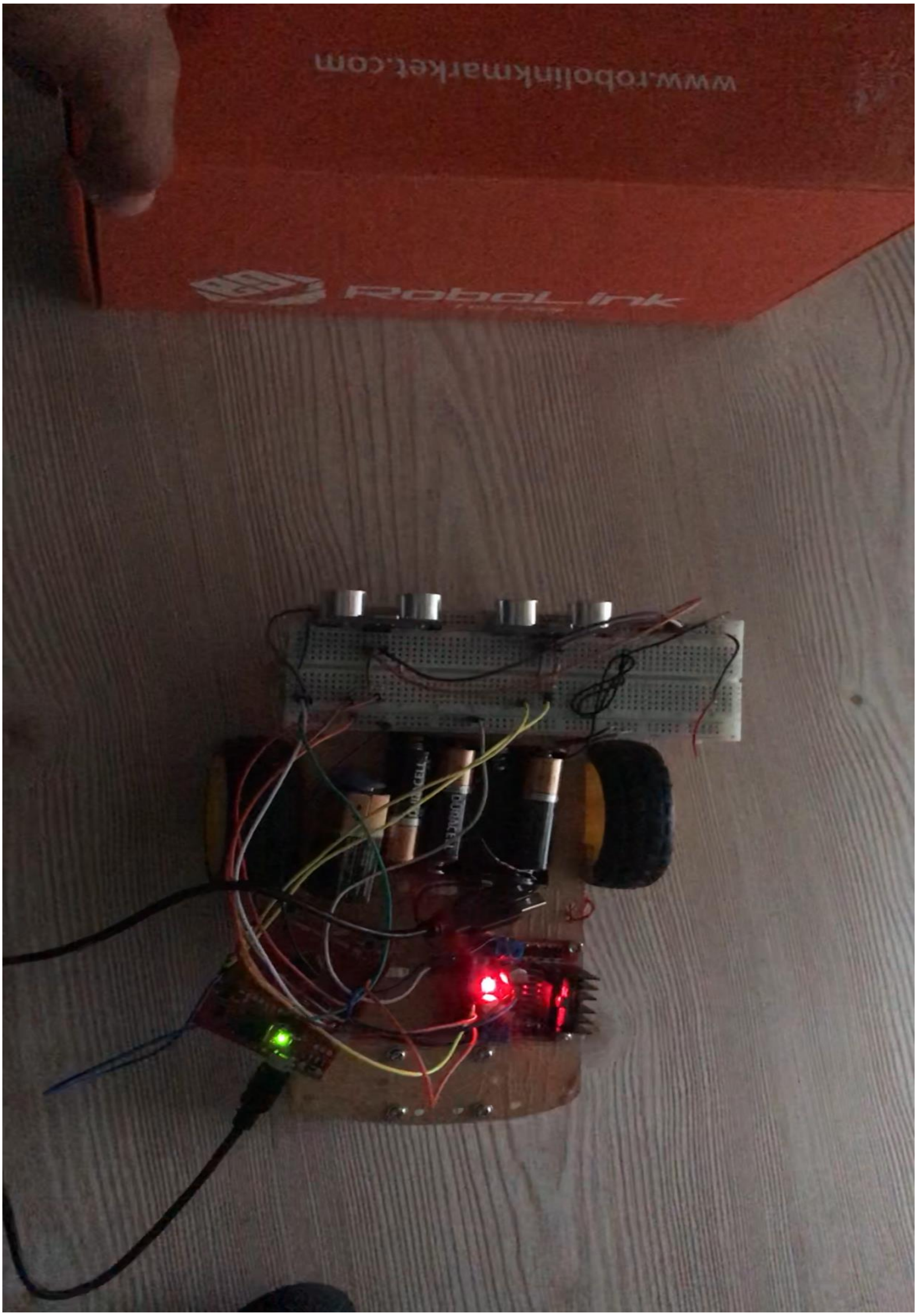
4. Electric Circuit



5. Photographs and Videos

VIDEO LINK → https://youtu.be/7WZ3ohHNNVc




www.robolinkmarket.com
RoboLink