BAŞKENT UNIVERSITY

ENGINEERING FACULTY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EEM 332 - MICROPROCESSORS
TERM PROJECT

ARDUINO BASED CAR WITH REMOTE CONTROL WITH HTTP SERVER

Samet Bayat

22293730

May 2023

CONTENTS

*** THE OPEN-SOURCE DRIVER AND EXAMPLE CODES WHICH I UTILIZED CAN BE FOUND IN**

https://github.com/espressif/esp32-camera

**** YOU CAN REACH THE DIGITAL COPY OF THIS PAPER AND THE FULL CODE VIA**

https://sametbayat.me/samet_332_term_code.md **(CODE)**

https://sametbayat.me/samet_332_term_report.pdf **(DIGITAL COPY)**

INTRODUCTION

For my microprocessors term project, I decided to create a remote-control car using the ESP32-CAM microcontroller module. The ESP32-CAM is a powerful system-on-chip that includes a camera and Wi-Fi capabilities, making it a great choice for applications that require real-time monitoring and control. In this report, I will discuss my experience building and testing the remote-control car, including the specific components and software tools that I used, as well as the challenges I encountered and how I overcame them. Additionally, I will explore the potential applications and implications of this project for the field of microprocessors and beyond.

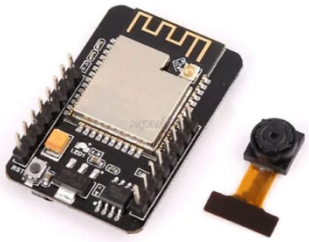
ACKNOWLEDGMENTS

Before I dive into the details of my project, I would like to take a moment to express my gratitude to Hamit Erdem, who provided me with a solid foundation in digital logic, microprocessors, and assembly language. I'll never forget his wisdom and the stories that he shared with us.

ESP32-CAM SPECIFICATIONS & COMPARISON (VS. ARDUINO)

ESP32-CAM a versatile and cost-effective solution for a wide range of applications, including home automation, security systems, and robotics. The camera and Wi-Fi features of the module are particularly useful for real-time, remote monitoring and control of physical systems.

ESP32-CAM



- System-on-Chip: ESP32 dual-core processor
- Flash Memory: 4MB
  - SPI Flash default 32Mbit
  - RAM internal520KB+external 4M PSRAM
- Camera: OV2640 2MP camera
- Wi-Fi
- Bluetooth: Bluetooth 4.2 BLE
- Interfaces: UART, SPI, I2C, I2S, CAN, PWM, ADC, DAC
- Operating Voltage: 5V DC
- Dimensions: 27mm x 40.5mm

- Real-time camera and Wi-Fi features, remote monitoring and control
- More powerful processor than most Arduino boards
- Support for a wide range of interfaces
- Can handle more complex tasks than most Arduino boards

vs.ARDUINO:

- Single-core processor
- Less flash memory than ESP32-CAM
- Limited connectivity options
- Capabilities:
  - Limited connectivity options compared to ESP32-CAM
  - May be better suited for simpler projects

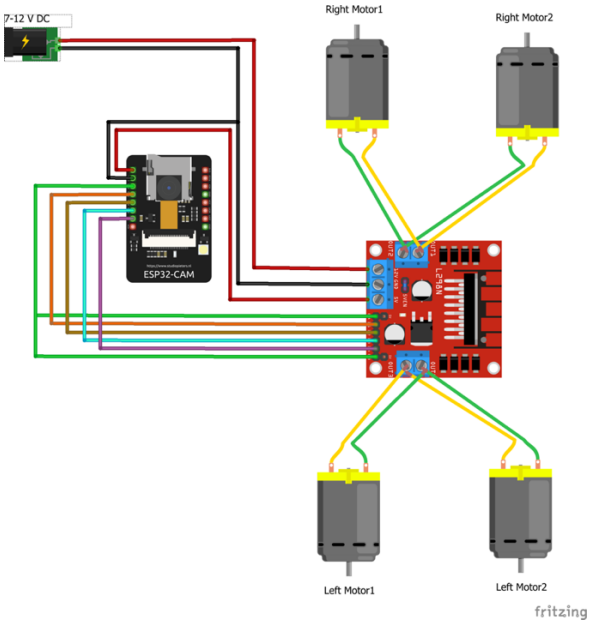
The Arduino Uno, one of the most popular Arduino boards, has a clock speed of 16 MHz (megahertz). The ESP32-CAM module, on the other hand, has a clock speed of 240 MHz by default, which is significantly faster than the Arduino. Having a faster clock speed means that the ESP32-CAM can perform computations and execute code more quickly, which can be advantageous for certain applications. However, it also has some drawbacks such as higher power consumption and the need for more advanced components to support the higher clock speed.

SOME FLUKES AND THE METHODS HOW I ACHIEVE THEM

The difference between WiFi.softAP() and WiFi.begin() is that WiFi.softAP() sets up a soft access point whereas WiFi.begin() connects to an existing access point. A soft access point (soft AP) is a feature that allows a device to act as an access point for other devices to connect to it directly instead of connecting to a router2. On the other hand, WiFi.begin() is used to connect to an existing access point. So for html server broadcast, due to bugs I changed had to downgrade the WiFi protocol to basic one.

In testing phase, my major problem was the battery's itself. I used 9.6V NiCD battery. (Which I remove it from old RC car). Despite the Nickel-Cadmium batteries' long cycle life and charging efficiency, battery drains relatively fast due to 10+ years of fatigue. Yet for certain period, it still could handle 4 motors' and microprocessor's power need so well. Additionally, the NiCD batteries are prone to a phenomenon called "memory effect", that means if I continue to charge it regularly, it's efficiency may be better in time.

Due to the WiFi, processor capabilities/limitations, old http server libraries and lack of higher battery capacity, I had to remove the ultrasound distance sensor from the set up. It seems the computational cost of streaming feature is so high. (if I set for less resolution < SVGA(800x600) the system might handle additional distance measurement ability, yet I decided the optimal standards for my set up for 2MP cam was SVGA.) Plus I lack of some advanced http server protocol data transfer/handshaking. It seems that with better optimization methods, the system could need less processing cost.



ESP32 Cam Pins	L298N Driver pins
IO12	enA
IO13	IN1
IO15	IN2
IO14	IN3
IO2	IN4
IO12	enB
5V	5V
GND	GND

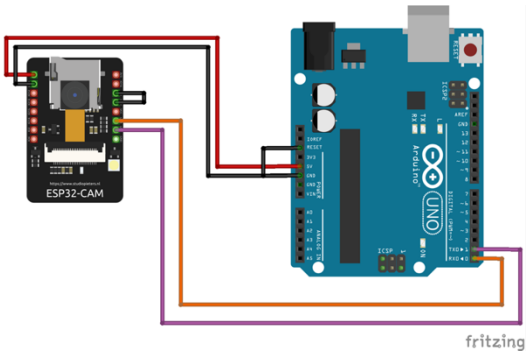


FIGURE. (ABOVE) SET UP OF THE CIRCUIT.

(RIGHT) FLASHING SET UP OF ESP32-CAM USING ARDUINO UNO

OTHER COMPONENTS

DC MOTOR

DC motors can be controlled by microprocessors using a four-quadrant switching circuit[1]. Microprocessors can be used for speed control, servo position control, and monitoring functions[2]. A microprocessor-based control system for a DC motor can be developed using standard hardware1. The processes of a general servo system may be used to generate the firing pulse by going through the pulse generation program by means of the interrupt signal[3].

1. ieeexplore.ieee.org 2. trid.trb.org 3. eeeguide.com

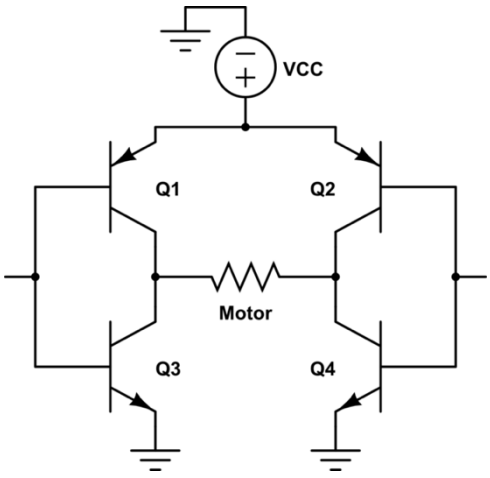
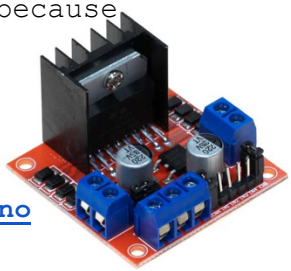


FIGURE. H-BRIDGE EXAMPLE

L298N MOTOR DRIVER

The L298N chip contains two standard H-bridges capable of driving a pair of DC motors[1]. It is a high-power motor driver module which is used to drive DC and Stepper Motors[2]. The L298N module can control up to four DC motors, or two DC motors with direction control and speed control because of its dual H-bridge circuit[2].



1. lastminuteengineers.com
2. technobyte.org

See also:
GitHub - KROIA/L298N_MotorDriver: Arduino driver library for DC-Motors

WEB SERVER DETAIL:

AsyncWebServer is a library for creating a web server using the ESP32/ESP8266 Arduino platform. It provides a way to handle incoming HTTP requests asynchronously, meaning that multiple requests can be handled simultaneously without blocking the server.

The library uses the concept of handlers to define how to respond to different types of HTTP requests, such as GET or POST requests. Handlers can be created for specific paths, allowing for different responses based on the URL requested. AsyncWebServer also provides support for serving static files, such as HTML, CSS, and JavaScript files, from the file system of the ESP32/ESP8266. Additionally, it supports serving dynamic content by allowing the server to generate responses on-the-fly based on the incoming request.

FIGURE. THE BROWSER UI.

```
// Here the related libraries added to the code.
#include "esp_camera.h"
#include <WiFi.h>
#include <Arduino.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <iostream>
#include <sstream>

//The pinEn variable typically controls the
motor's speed,
//while pinIN1 and pinIN2 control the direction of
rotation.
struct MOTOR_PINS
{
  int pinEn;
  int pinIN1;
  int pinIN2;
};

std::vector<MOTOR_PINS> motorPins =
{
 {12, 13, 15},  //RIGHT_MOTOR Pins (EnA, IN1, IN2)
 {12, 14, 2},  //LEFT_MOTOR  Pins (EnB, IN3, IN4)
};
//Assign values to variables that we are going to
frequently use.
#define LIGHT_PIN 4
#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4
#define STOP 0
#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1
#define FORWARD 1
#define BACKWARD -1

const int PWMFreq = 1000; /* 1 KHz */
const int PWMResolution = 8;
const int PWMSpeedChannel = 2;
const int PWMLightChannel = 3;

//Initialize and assign port to http server to
stream and control.
AsyncWebServer server(80);
AsyncWebSocket wsCamera("/Camera");
AsyncWebSocket wsCarInput("/CarInput");
uint32_t cameraClientId = 0;

// Define user browser UI using HTML.
// Show the live stream on top and control buttons
below the video.
// The direction commands is sending via
("MoveCar","assigned_number")
// The assigned_number shown is pseudo code
represents the number that assigned for certain
acts by myself. 1->UP, 0->STOP, etc.
const char* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(
       PSUEDO HTML
       LIVE STREAM
       LIGHT, SPEED DIRECTION CONTROL BTNS
       **VISIT THE LINK FOR FULL CODE
)HTMLHOMEPAGE";

// This function helps us to arrange motor acts.
It is designed to control the rotation
//direction of a motor by setting the appropriate
digital pins of the motor driver module.
void rotateMotor(int motorNumber, int
motorDirection)
{
  if (motorDirection == FORWARD)
  {
    digitalWrite(motorPins[motorNumber].pinIN1,
HIGH);
    digitalWrite(motorPins[motorNumber].pinIN2,
LOW);
  }
  else if (motorDirection == BACKWARD)
  {
    digitalWrite(motorPins[motorNumber].pinIN1,
LOW);
    digitalWrite(motorPins[motorNumber].pinIN2,
HIGH);
  }
  else
  {
    digitalWrite(motorPins[motorNumber].pinIN1,
LOW);
    digitalWrite(motorPins[motorNumber].pinIN2,
LOW);
  }
}

// When the moveCar func. called, the motors'
current situation rearranging in order to the user
input.
void moveCar(int inputValue)
{
  Serial.printf("Got value as %d\n", inputValue);
  switch(inputValue)
  {

    case UP:
      rotateMotor(RIGHT_MOTOR, FORWARD);
      rotateMotor(LEFT_MOTOR, FORWARD);
      break;

    case DOWN:
      rotateMotor(RIGHT_MOTOR, BACKWARD);
      rotateMotor(LEFT_MOTOR, BACKWARD);
      break;

    case LEFT:
      rotateMotor(RIGHT_MOTOR, FORWARD);
      rotateMotor(LEFT_MOTOR, BACKWARD);
      break;

    case RIGHT:
      rotateMotor(RIGHT_MOTOR, BACKWARD);
      rotateMotor(LEFT_MOTOR, FORWARD);
      break;

    case STOP:
      rotateMotor(RIGHT_MOTOR, STOP);
      rotateMotor(LEFT_MOTOR, STOP);
      break;

    default:
      rotateMotor(RIGHT_MOTOR, STOP);
      rotateMotor(LEFT_MOTOR, STOP);
      break;
  }
}

// This function helps us to communicate, transfer
and monitorize datas between our microprocessor,
its side-parts (motors, drivers, etc.) and
browser(user UI).
void handleRoot(AsyncWebServerRequest *request)
{
  request->send_P(200, "text/html", htmlHomePage);
}
```

```
void handleNotFound(AsyncWebServerRequest
*request)
{
    request->send(404, "text/plain", "File Not
Found");
}

void onCarInputWebSocketEvent(AsyncWebSocket
*server,
                      AsyncWebSocketClient
*client,
                      AwsEventType type,
                      void *arg,
                      uint8_t *data,
                      size_t len)
{
  switch (type)
  {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u
connected from %s\n", client->id(), client-
>remoteIP().toString().c_str());
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u
disconnected\n", client->id());
      moveCar(0);
      ledcWrite(PWMLightChannel, 0);
      break;
    case WS_EVT_DATA:
      AwsFrameInfo *info;
      info = (AwsFrameInfo*)arg;
      if (info->final && info->index == 0 && info-
>len == len && info->opcode == WS_TEXT)
      {
        std::string myData = "";
        myData.assign((char *)data, len);
        std::istringstream ss(myData);
        std::string key, value;
        std::getline(ss, key, ',');
        std::getline(ss, value, ',');
        Serial.printf("Key [%s] Value[%s]\n",
key.c_str(), value.c_str());
        int valueInt = atoi(value.c_str());
        if (key == "MoveCar")
        {
          moveCar(valueInt);
        }
        else if (key == "Speed")
        {
          ledcWrite(PWMSpeedChannel, valueInt);
        }
        else if (key == "Light")
        {
          ledcWrite(PWMLightChannel, valueInt);
        }
      }
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
    default:
      break;
  }
}

void onCameraWebSocketEvent(AsyncWebSocket
*server,  AsyncWebSocketClient *client,
                      AwsEventType type,
                      void *arg,
                      uint8_t *data,
                      size_t len)
{
  switch (type)
  {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u
connected from %s\n", client->id(), client-
>remoteIP().toString().c_str());
      cameraClientId = client->id();
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u
disconnected\n", client->id());
      cameraClientId = 0;
      break;
    case WS_EVT_DATA:
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
    default:
      break;
  }
}

#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

const char* ssid = "Note10lite";
const char* password = "***PASSWORD***";

void setupLedFlash(int pin);

void sendCameraPicture()
{
  if (cameraClientId == 0)
  {
    return;
  }
  //capture a frame
  camera_fb_t * fb = esp_camera_fb_get();
  if (!fb)
  {
      Serial.println("Frame buffer could not be
acquired");
      return;
  }
  wsCamera.binary(cameraClientId, fb->buf, fb-
>len);
  esp_camera_fb_return(fb);
  //Wait for message to be delivered
  while (true)
  {
    AsyncWebSocketClient * clientPointer =
wsCamera.client(cameraClientId);
    if (!clientPointer || !(clientPointer-
>queueIsFull()))
    {
      break;
    }
    delay(1);
  }
}

void setUpPinModes()
{
  //Set up PWM
  ledcSetup(PWMSpeedChannel, PWMFreq,
PWMResolution);
  ledcSetup(PWMLightChannel, PWMFreq,
PWMResolution);

  for (int i = 0; i < motorPins.size(); i++)
  {
    pinMode(motorPins[i].pinEn, OUTPUT);
```

```
    pinMode(motorPins[i].pinIN1, OUTPUT);
    pinMode(motorPins[i].pinIN2, OUTPUT);

    /* Attach the PWM Channel to the motor enb Pin
*/
    ledcAttachPin(motorPins[i].pinEn,
PWMSpeedChannel);
  }
  moveCar(STOP);

  pinMode(LIGHT_PIN, OUTPUT);
  ledcAttachPin(LIGHT_PIN, PWMLightChannel);
}



// For streaming we use 20MHz cloCk speed here.
For smoother outputs, in our case (ESP32-cam) can
achieve up to 240MHz in theory.
//Yet our sample could not. (Note: Arduino
clk_speed = 16MHz.)
void setup() {

  setUpPinModes();
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  //setup camera values
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000; //20MHz
  config.frame_size = FRAMESIZE_SVGA; //800x600
  config.pixel_format = PIXFORMAT_JPEG; // for
streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for
face detection/recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  if(config.pixel_format == PIXFORMAT_JPEG){
    if(psramFound()){
      config.jpeg_quality = 10;
      config.fb_count = 1;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not
available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  }

  // camera initialize
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error
0x%x", err);
    return;
  }
    if (psramFound())
  {
    heap_caps_malloc_extmem_enable(20000);
    Serial.printf("PSRAM initialized. malloc to
take memory from psram above this size");
  }

  sensor_t * s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and
colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1); // flip it back
    s->set_brightness(s, 1); // up the brightness
just a bit
    s->set_saturation(s, -2); // lower the
saturation
  }
  // drop down frame size for higher initial frame
rate
  if(config.pixel_format == PIXFORMAT_JPEG){
    s->set_framesize(s, FRAMESIZE_QVGA);
  }

  WiFi.begin(ssid, password);
  WiFi.setSleep(false);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

//  startCameraServer();

server.on("/", HTTP_GET, handleRoot);
  server.onNotFound(handleNotFound);

  wsCamera.onEvent(onCameraWebSocketEvent);
  server.addHandler(&wsCamera);

  wsCarInput.onEvent(onCarInputWebSocketEvent);
  server.addHandler(&wsCarInput);

  server.begin();
  Serial.println("HTTP server started");

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {

  wsCamera.cleanupClients();
  wsCarInput.cleanupClients();
  sendCameraPicture();
}
```