**Robotic Buggy with Camera and Ultrasonic Sensor**

# Abstract

The project is to develop a buggy that can collect visual data and measure distance while traversing through a maze. It adds the ESP32-CAM camera module to the previous version of buggy that has motors and ultrasonic sensor. The image captured by ESP32-CAM will be transmitted with the distance data through Bluetooth module embedded within it. To receive the data, a Python script is used to establish Bluetooth Connection as a serial port and save the image as JPG file with the time stamp and the distance as the file name.

Table of Contents

[Abstract 1](#_Toc199542457)

[Goals 1](#_Toc199542458)

[Background 2](#_Toc199542459)

[Method 2](#_Toc199542460)

[MSP432 3](#_Toc199542461)

[ESP32-CAM 7](#_Toc199542462)

[Serial Port 8](#_Toc199542463)

[Camera on ESP32-CAM 9](#_Toc199542464)

[Bluetooth Serial 9](#_Toc199542465)

[Python Script 11](#_Toc199542466)

[Procedures 11](#_Toc199542467)

[Difficulties and Solutions 11](#_Toc199542468)

[Result and Conclusion 13](#_Toc199542469)

[Appendix 14](#_Toc199542470)

# Goals

The goal of this project is to develop an enhanced buggy that integrates an ESP32-CAM module to capture and transmit visual and distance data via Bluetooth.

# Background

Bluetooth is a short-range wireless communication standard. Bluetooth uses low power wireless link at 2.4GHz, giving it a short range of 10 meters and vulnerable to interference. It is mainly used for portable devices to exchange files with host, like wireless headphone or speaker to mobile phone or laptop. (1)

# Method

The buggy has four main components: MSP432P401R microcontroller (MSP432), ESP32-CAM, HC-SR04 ultrasonic sensor, and two motors with H-bridge. The connection is shown in the diagram below.

A red circuit board with wires

Description automatically generated

**Figure 1.** Schematic of the Buggy

The motor and the ultrasonic sensor are using the same method as the previous version, except the Trig and Echo pins for the ultrasonic sensor are moved to P4.6 and P4.7. And the previous code has been organized into different files based on the functionality. See Appendix for the code. The ESP32-CAM is connected to MSP432 by the UART port. So, they can communicate with the byte stream.

A screenshot of a computer

Description automatically generated

**Figure 2.** Diagram of the Buggy Set-up

## MSP432

The buggy has a state machine in MSP432 to control its behavior. It has four states: *INIT*, *IDLE*, *MOVE*, and *SCAN.* The buggy starts with INIT state to initialize all components.

**case** *INIT*:

//this state initialize all components

Clock\_init();

motors\_init(); //motors for moving

UART2\_init(); //uart2(3.2, 3.3) for communicating with ESP32 cam

TIMERA0\_init();

US\_init();

currState = *IDLE*;

**break**;

Then it moves to IDLE state to wait for the ESP32-CAM to be ready. If it receives “go” from the camera module, it enables interrupt service routine to start the timer and ultrasonic sensor. Then it proceeds to the MOVE state.

**case** IDLE:

//this state wait for the ESP32 cam to be ready and send "go" through uart port

UART2\_gets(readbuf);

if(strncmp(readbuf, "go",2) == 0)

{

\_\_enable\_irq();

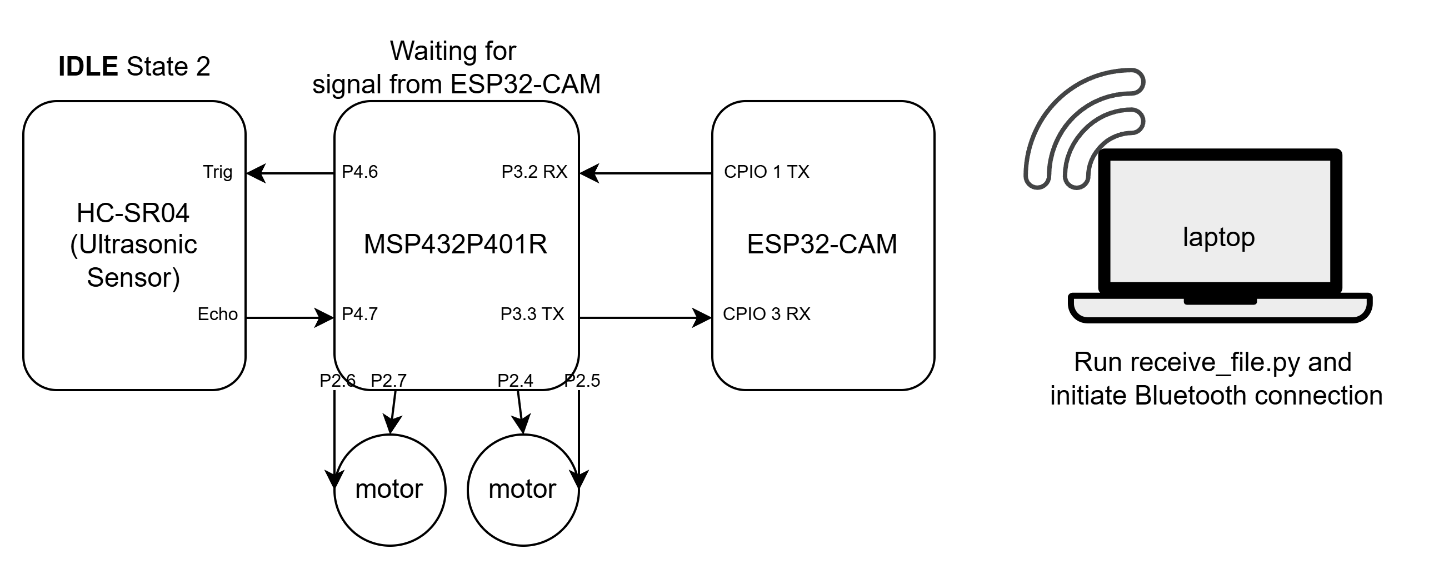
// enables interrupts to start timer for time stamps, irq for ultrasonic sensor

currState = MOVE; // starts moving

}

**break;**

A screenshot of a device

Description automatically generated

A screenshot of a computer

Description automatically generated

**Figure 3.** Flow Chart of IDLE State

In the MOVE state, the buggy will move forward and start scanning. The step buggy takes will be increased each time, so with the turn, the buggy will move in a triangular pattern.

**case** *MOVE*:

forward(unit\_step \* step); //the forward steps will gradually increase

step++;

currState = *SCAN*;

**break;**

In the SCAN state, the buggy will take two 120 degree turns. For each turn, it measures the distance by the ultrasonic sensor and gets the current time stamp. It sends the information to the ESP32-CAM through UART2 and waits for the ready signal to move on. It will end up taking a 240 degree turn and then change to MOVE state.

**case** *SCAN*:

**for**(i = 0; i < 2; i++){

distance = measure\_dist();

time\_stamp = get\_time\_stamp();

**sprintf**(writebuf, "%s, %d\n", time\_stamp, distance); // format the output string

UART2\_puts(writebuf); //write to ESP32 cam

UART2\_gets(readbuf); //wait for ready signal from ESP32 cam

turn(-120); //it will end up turning for 240 degree

}

currState = *MOVE*;

**break**;

}

**A close-up of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Figure 4.** Flow Chart of SCAN State

The buggy will continue to oscillate between MOVE and SCAN state to move in triangular spiral and collect images. The reset button on the MSP432 can be used to break out of this loop. The buggy will then be reset and waiting in IDLE state. To start the run again, the laptop needs to restart the python script to establish Bluetooth connection again.

## ESP32-CAM

The ESP32-CAM is also a microcontroller, and it can be programmed using Arduino IDE

A close-up of a computer chip

Description automatically generated

**Figure 5.** ESP32-CAm with OV2640 Camera

The ESP32-CAM doesn’t have a micro-USB port. It can be connected with ESP32-CAM-MB and use its micro-USB port. The micro-USB port is used for uploading the compiled program. In case when ESP32-CAM-MB isn’t available,FTDI jumper can be used for converting micro-USB to UART serial port. (2)

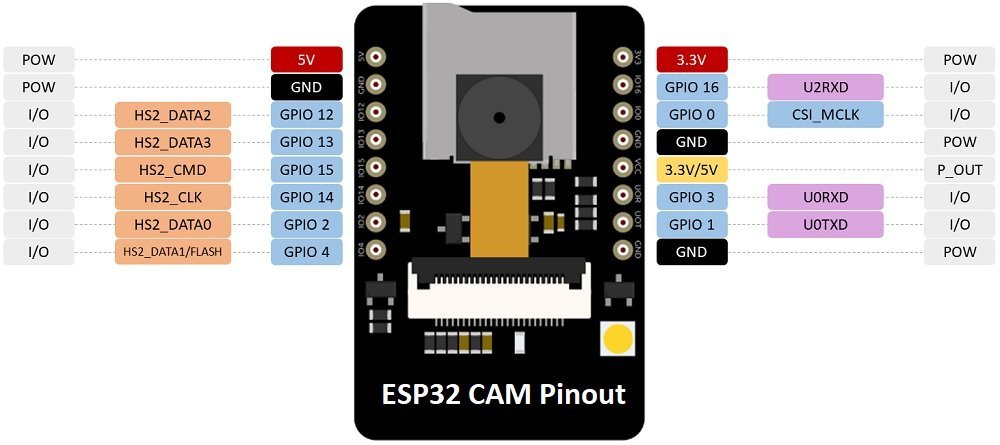
To compile ESP32 on Arduino, add the following link to Arduino IDE under Preferences->Additional Boards Manager->Additional Boards Manager URLs

<https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json>

Then go to Tools->Board->Board Manager. Install ESP32 by Expressif Systems.

Once Installed, go to Tools->Board->ESP32 Arduino->AI Thinker ESP32-CAM.

Connect ESP32 Cam and upload compiled code.



**Figure 6.** ESP32 CAM Pin Configuration

## Serial Port

ESP32-CAM only has one full UART port (GPIO 1 and 3), which in our case will be used to communicate with MSP432 with its UART2 (P3.2 and P3.3). GPIO 1 and 3 are also used for sketch uploading. And because of the board configuration of ESP32-CAM, this serial port isn’t compatible with terminal emulator like Teraterm. It can be properly interpreted by Arduino serial monitor if the board configuration is loaded. The problem seems to be that ESP32-CAM disabled DTR and RTS, and if these are used, the board will keep rebooting. However, the exact solution to configure Teraterm to fit with this setting is still unknown. This will also mess up the transmission if directly connected to external Bluetooth module, like HC-06, for serial transmission. Direct connection to other microcontrollers like MSP432 through UART seems to be fine.

In order to receive serial data on a computer, pyserial library is used. It requires Python 2.7 or Python 3.4 and newer, which can be downloaded from official website with all released version <https://www.python.org/downloads/> . And pyserial can be installed through pip through command line “python -m pip install pyserial”. (3) To establish the connection, it needs to create a Serial object with the port name and baud rate, like the example below:

com = serial.Serial()

com.port = port\_name

com.baudrate = baud

com.timeout = 1

com.dsrdtr = False

com.open()

To resolve the issue of asserting DTR and RTS, it needs to initiate a port object and set dsrdtr to False before establishing the connection. With connection properly established, it can receive bytes from serial port and save it as jpg file.

## Camera on ESP32-CAM

ESP32-CAM needs to be configured and initialized to capture images. The specific configuration is in the Appendix. After it properly initialized, function esp\_camera\_fb\_get() will return a struct of frame buffer(4):

typedef struct {

uint8\_t \* buf; /\*!< Pointer to the pixel data \*/

size\_t len; /\*!< Length of the buffer in bytes \*/

size\_t width; /\*!< Width of the buffer in pixels \*/

size\_t height; /\*!< Height of the buffer in pixels \*/

pixformat\_t format; /\*!< Format of the pixel data \*/

struct timeval timestamp; /\*!< Timestamp since boot of the first DMA buffer of the frame \*/

} camera\_fb\_t;

The frame buffer is at the buf pointer with the size of len. If it is properly configured, the buffer will be in the jpeg format. And these data can be transmitted as byte stream through serial port with Serial.write(fb->buf, fb->len). After each usage of the frame buffer, the memory needs to be freed by function esp\_camera\_fb\_return(fb).

To take better quality pictures, it is recommended to use the flashlight at GPIO 4 and take one picture first to let the camera adjust itself and take the picture again. Like the example below

## Bluetooth Serial

camera\_fb\_t \* fb = NULL;

// Take Picture with Camera

pinMode(4, OUTPUT); //GPIO 4 is the LED flashlight

digitalWrite(4, HIGH);

fb = esp\_camera\_fb\_get();

esp\_camera\_fb\_return(fb); //need to free frame buffer to prevent memory overflow

//Taking the picture twice allows the camera to adjust itself. It usually works better under brighter environment

delay(1000);

fb = esp\_camera\_fb\_get();

digitalWrite(4, LOW);

The ESP32-CAM has embedded Bluetooth module that can be initiated using BluetoothSerial.h.

With the function SerialBT.begin(device\_name), ESP32-CAM will appear with the provide name as the Bluetooth device. It can be connected to a computer with “find Bluetooth Device” option. After connected, two serial ports will be added for connection.

void btCallback(esp\_spp\_cb\_event\_t event, esp\_spp\_cb\_param\_t \*param){

if(event == ESP\_SPP\_SRV\_OPEN\_EVT){

//this event corresponds to establishing bluetooth connection

//The "go" triggers MSP432 to start moving

Serial.println("go");

}

}

Bluetooth serial has similar functionality as regular serial port like read, write, print, avail, etc. It can also register call back functions using SerialBT.register\_callback(btCallback); This is used to detect the event of establishing Bluetooth connection.

When the connection is established, ESP32-CAM sends the String “go\n” to the MSP432 through default serial port (GPIO 1 and 3). This corresponds to the “go” MSP432 is waiting for in the IDLE state.

After the connection is established, ESP32-CAM will be waiting for MSP432 to send the time stamp and the distance through UART. Each time ESP32-CAM receives the data, it will take a picture and transmit all the information through Bluetooth. The byte stream for each picture follows the structure below, so the receiver can parse all information properly.

/\*

\* structure of bytes stream transferred

\* transmission started\n

\* time stamp, distance\n

\* file size(bytes)\n

\* file bytes

\*/

The Bluetooth transmission is similar with regular serial port with SerialBT.write(fb->buf, fb->len). It seems like this process can’t be done under the call back function, or else the data will be lost through Bluetooth transfer. After all transmission through Bluetooth, ESP32-CAM will send an arbitrary character to MSP432 as ready signal, marking it’s ready for the next picture.

## Python Script

The Python script is used to receive images on the computer. The pyserial library can read bytes from the serial port until a newline character by readline() or until the given number of bytes with read. After receiving all the information, it will create an image file with the time stamp and distance as the file name.

        if(value == b'transmission started\n'):

            name = com.readline()

            file\_name = str(name, encoding="UTF-8")[:-1]

            print(file\_name)

            value = com.readline()

            line = str(value, encoding="UTF-8")

            file\_len = int(line)

            print(f'file size: {file\_len} bytes')

            start = time.time() #calculate tranmission time

            file\_buf = com.read(file\_len)

            print(len(file\_buf))

            with open(file\_name+".jpg", "wb") as f: #save the image with time stamp, distance as file name

                f.write(file\_buf)

            end = time.time()

            print("picture captured")

            print(end - start)

            num +=1

## Procedures

1. Program MSP432 and ESP32-CAM
2. Connect all components together and power on, the buggy should then be waiting for Bluetooth connection
3. If the ESP32-CAM hasn’t been added as Bluetooth devices, use “Find Bluetooth Device” option on the computer to add it.
4. Run the Python Script on the computer, the buggy will start running and sending images to the computer
5. To stop the buggy, press the reset button on the MSP432. The buggy will reboot and waiting for Bluetooth connection.
6. To start another run, run the Python script on the computer again.

# Difficulties and Solutions

A summary of difficulties encountered during the project and their solutions

* Can’t read from ESP32-CAM serial port and the board is halted

Mostly likely because ESP32-CAM disabled DTR and RTS, and if these are used, the board will keep rebooting. So make sure the other end also disable these features. Serial monitor in Arduino IDE can communicate with it properly if the corresponding board configuration is loaded. If the connection is established through pyserial, initiate a port object and set dsrdtr to False before establishing the connection, like the example below

com = serial.Serial()

com.port = port\_name

com.baudrate = baud

com.timeout = 1

com.dsrdtr = False

com.open()

* Bytes are dropped during Bluetooth transmission

Avoid starting the transmission after the callback function. This is probably because callback function limits the stack size to handle function calls and frame buffer. Smaller file size means a lower chance of dropping, and the size of image can be adjusted by setting config.frame\_size to FRAMESIZE\_ + QVGA, CIF, VGA, SVGA, XGA, SXGA, UXGA(size in increasing order). The regular Bluetooth connection should be reliable enough. To ensure a fully reliable transfer, certain kinds of protocol need to be implemented. For example, similar to TCP, the image file can be divided into packets and only send out the next packet after the receiver acknowledged the previous packet. However, this is an overkill and will significantly slow down the transmission.

* esp\_camera\_fb\_get() fails

Probably because the previous frame buffer wasn’t freed by esp\_camera\_fb\_return(). The camera can only hold the number of frame buffer according to config.fb\_count

* picture is green with a lot of noise

The camera performs better under bright light. The quality of the picture can be improved by having multiple tries because the camera will configure itself after taking pictures.

# Result and Conclusion

A small black robot with wires and wheels

Description automatically generatedA toy car with wires

Description automatically generated

**Figure 7.** Side and Top view of Buggy

A collage of images of buildings

Description automatically generated

**Figure 8.** Received Example Images

The figures above are the final version of the buggy, and the image received in the test run. The Python Script can continuously receive images captured by the buggy and create files accordingly. This fulfills the goal of the project.

To further improve the buggy, we could use PCB for the circuit connection, which will greatly reduce the size of the buggy. The picture quality can be improved by using a different camera module. Bluetooth transmission is relatively slow due to its baud rate. We can speed up this process by changing to other wireless transmission like WiFi. The current buggy only takes command for starting the traversal and can only be terminated by the reset button. With Bluetooth or other transmission, it can potentially take in more commands like pause, continue, or even remote control of directions.

**Reference**

1. "About us – Bluetooth Technology Website". Bluetooth.com.
2. "ESP32-CAM Complete Guide". <https://www.diyengineers.com/2023/04/13/esp32-cam-complete-guide/>
3. "pyserial documentation". <https://pyserial.readthedocs.io/en/latest/pyserial.html>
4. "ESP32 Camera Driver". <https://github.com/espressif/esp32-camera/blob/master/README.md>

# Appendix

Box folder for code <https://uofi.box.com/s/8n2sgsv3pwzodkihhqbszfubwpxs37ac>

bluetooth\_cam.ino

#include "esp\_camera.h"

#include "Arduino.h"

#include "FS.h"                // SD Card ESP32

#include "SD\_MMC.h"            // SD Card ESP32

#include "soc/soc.h"           // Disable brownour problems

#include "soc/rtc\_cntl\_reg.h"  // Disable brownour problems

#include "driver/rtc\_io.h"

#include <EEPROM.h>            // read and write from flash memory

#include "BluetoothSerial.h"   //bluetooth

// define the number of bytes you want to access

#define EEPROM\_SIZE 1

RTC\_DATA\_ATTR int bootCount = 0;

// Pin definition for CAMERA\_MODEL\_AI\_THINKER

#define PWDN\_GPIO\_NUM     32

#define RESET\_GPIO\_NUM    -1

#define XCLK\_GPIO\_NUM      0

#define SIOD\_GPIO\_NUM     26

#define SIOC\_GPIO\_NUM     27

#define Y9\_GPIO\_NUM       35

#define Y8\_GPIO\_NUM       34

#define Y7\_GPIO\_NUM       39

#define Y6\_GPIO\_NUM       36

#define Y5\_GPIO\_NUM       21

#define Y4\_GPIO\_NUM       19

#define Y3\_GPIO\_NUM       18

#define Y2\_GPIO\_NUM        5

#define VSYNC\_GPIO\_NUM    25

#define HREF\_GPIO\_NUM     23

#define PCLK\_GPIO\_NUM     22

//Example SerialtoSerialBT

// Check if Bluetooth is available

#if !defined(CONFIG\_BT\_ENABLED) || !defined(CONFIG\_BLUEDROID\_ENABLED)

#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it

#endif

// Check Serial Port Profile

#if !defined(CONFIG\_BT\_SPP\_ENABLED)

#error Serial Port Profile for Bluetooth is not available or not enabled. It is only available for the ESP32 chip.

#endif

#define CHUNK\_SIZE (size\_t)32

BluetoothSerial SerialBT;

String device\_name = "ESP32\_CAM\_BT";

void setup() {

  WRITE\_PERI\_REG(RTC\_CNTL\_BROWN\_OUT\_REG, 0); //disable brownout detector

  Serial.begin(9600);   //The MSP432 will be communicating through default serial port (GPIO 1 TX, GPIO 3 RX)

  cam\_init();

  bt\_init();

}

void loop() {

  String read;

  if(Serial.available())

  {

    //read time stamp and distance from MSP432

    read = Serial.readString();

    /\*

    \* structure of bytes stream transferred

    \* transmission started\n

    \* time stamp, distance\n

    \* file size(bytes)\n

    \* file bytes

    \*/

    SerialBT.printf("transmission started\n");

    SerialBT.print(read);

    take\_picture();

    //tell MSP432 the cam is ready again

    Serial.print("r\n");

  }

}

void cam\_init()

{

  Serial.setDebugOutput(false);

  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\_sscb\_sda = SIOD\_GPIO\_NUM;

  config.pin\_sscb\_scl = SIOC\_GPIO\_NUM;

  config.pin\_pwdn = PWDN\_GPIO\_NUM;

  config.pin\_reset = RESET\_GPIO\_NUM;

  config.xclk\_freq\_hz = 20000000;

  config.pixel\_format = PIXFORMAT\_JPEG;

  //ESP32 cam will most likelt have psram, so it will be using UXGA

  if(psramFound()){

    Serial.print("Using UXGA\n");

    config.frame\_size = FRAMESIZE\_UXGA; // FRAMESIZE\_ + QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA

    config.jpeg\_quality = 10;

    config.fb\_count = 2;    //it can holds two frame buffer at the same time

  } else {

    Serial.print("Using SVGA\n");

    config.frame\_size = FRAMESIZE\_SVGA;

    config.jpeg\_quality = 12;

    config.fb\_count = 1;

  }

    // Init Camera

  esp\_err\_t err = esp\_camera\_init(&config);

  if (err != ESP\_OK) {

    Serial.printf("Camera init failed with error 0x%x", err);

    return;

  }

}

void bt\_init()

{

  SerialBT.begin(device\_name);

  SerialBT.register\_callback(btCallback);

  //Serial.println("The device started, now you can pair it with bluetooth");

}

void btCallback(esp\_spp\_cb\_event\_t event, esp\_spp\_cb\_param\_t \*param){

  //Serial Port Profile (SPP) API

  if(event == ESP\_SPP\_SRV\_OPEN\_EVT){

    //this event corresponds to establishing bluetooth connection

    //The "go" triggers MSP432 to start moving

    Serial.println("go");

  }

}

void take\_picture()

{

  camera\_fb\_t \* fb = NULL;

  // Take Picture with Camera

  pinMode(4, OUTPUT);   //GPIO 4 is the LED flashlight

  digitalWrite(4, HIGH);

  fb = esp\_camera\_fb\_get();

  esp\_camera\_fb\_return(fb); //need to free frame buffer to prevent memory overflow

  //Taking the picture twice allows the camera to adjust itself. It usually works better under brighter environment

  delay(1000);

  fb = esp\_camera\_fb\_get();

  digitalWrite(4, LOW);

  if(!fb) {

    //notify each side the capture failed

    Serial.println("Camera capture failed");

    SerialBT.println("Camera capture failed");

    return;

  }

  SerialBT.printf("%u\n",fb->len);

  SerialBT.flush();

  delay(10);

  SerialBT.write(fb->buf, fb->len);

  SerialBT.flush(); //make sure to transmit the whole file

  esp\_camera\_fb\_return(fb);

}

receive\_file.py

import serial, time

import sys

if \_\_name\_\_ == '\_\_main\_\_':

    if len(sys.argv) != 3:

        #using the following command to run the script, baudrate is 115200 with bluetooth

        print("python script serial\_port baudrate")

        #using the following command to list the available ports

        print("python -m serial.tools.list\_ports")

        exit()

    port\_name = sys.argv[1]

    baud = sys.argv[2]

    #Create a Serial object, configure the parameters, then open the connection

    #This is especially important when using wired connection with ESP32 cam

    #because it will try to use dsrdtr regardless of the setting, which reset the board

    com = serial.Serial()

    com.port = port\_name

    com.baudrate = baud

    com.timeout = 10

    com.dsrdtr = False

    com.open()

    print("serial connected")

    num = 1 #number of pictures received

    while True:

        print(num)

        value = com.readline()

        line = str(value, encoding="UTF-8")

        print(line)

        '''

        structure of bytes stream transferred:

        transmission started\n

        time stamp, distance\n

        file size(bytes)\n

        file bytes

        '''

        if(value == b'transmission started\n'):

            name = com.readline()

            file\_name = str(name, encoding="UTF-8")[:-1]

            print(file\_name)

            value = com.readline()

            line = str(value, encoding="UTF-8")

            file\_len = int(line)

            print(f'file size: {file\_len} bytes')

            start = time.time() #calculate tranmission time

            file\_buf = com.read(file\_len)

            print(len(file\_buf))

            with open(file\_name+".jpg", "wb") as f: #save the image with time stamp, distance as file name

                f.write(file\_buf)

            end = time.time()

            print("picture captured")

            print(end - start)

            num +=1

main.c

#include "msp.h"

#include <stdio.h>

#include <string.h>

#include "uart.h"

#include "motor.h"

#include "us.h"

#define unit\_step 1000

enum buggieState {

    INIT,

    IDLE,

    MOVE,

    SCAN

};

void main(void)

{

    WDT\_A->CTL = WDT\_A\_CTL\_PW |             // Stop watchdog timer

            WDT\_A\_CTL\_HOLD;

    char readbuf[128];

    char writebuf[128];

    int step = 1;       //number of forward step taken at each round

    long distance;      //distance measured by ultrasonic sensor

    char\* time\_stamp;   //time stamp starting

    int i;

    enum buggieState currState = INIT;

    while(1)

    {

        switch(currState)

        {

        case INIT:

            //this state initialize all components

            Clock\_init();

            motors\_init();      //motors for moving

            UART2\_init();       //uart2(3.2, 3.3) for communicating with ESP32 cam

            TIMERA0\_init();

            US\_init();

            currState = IDLE;

            break;

        case IDLE:

            //this state wait for the ESP32 cam to be ready and send "go" through uart port

            UART2\_gets(readbuf);

            if(strncmp(readbuf, "go",2) == 0)

            {

                \_\_enable\_irq();             // enables interrupts to start timer for time stamps, irq for ultrasonic sensor

                currState = MOVE;           // starts moving

            }

            break;

        case MOVE:

            forward(unit\_step \* step);      //the forward steps will gradually increase

            step++;

            currState = SCAN;

            break;

        case SCAN:

            for(i = 0; i < 2; i++){

                distance = measure\_dist();

                time\_stamp = get\_time\_stamp();

                sprintf(writebuf, "%s, %d\n", time\_stamp, distance); // format the output string

                UART2\_puts(writebuf);   //write to ESP32 cam

                UART2\_gets(readbuf);    //wait for ready signal from ESP32 cam

                turn(-120);         //it will end up turning for 240 degree

            }

            currState = MOVE;

            break;

        }

    }

}

**motor.h**

#include "msp.h"

#include <stdio.h>

#include "SysTick0.h"

void motors\_init(); // Initialize ports controlling H-Bridge

void forward(int time); // Move buggy forward for a time in ms

void backward(int time); // Move buggy backward for a time in ms

void turn(int angle); // Turn buggy by an angle in degrees

**motor.c**

#include "motor.h"

#define timePerAngle 10 // Constant to translate

void left\_fw(); // Helper functions for H-Bridge

void left\_bw();

void left\_off();

void right\_fw();

void right\_bw();

void right\_off();

void motors\_init() {

P2->OUT = 0x00; // Set P2.4-2.7 to low GPIO out, 2.4, 2.5 for the left, 2.6, 2.7 for the right

P2->SEL0 = 0x00;

P2->SEL1 = 0x00;

P2->DIR = 0xF0;

return;

}

void left\_fw() {

P2->OUT &= (~BIT5);

P2->OUT |= BIT4;

return;

}

void left\_bw() {

P2->OUT &= (~BIT4);

P2->OUT |= BIT5;

return;

}

void left\_off() {

P2->OUT &= (~(BIT4 | BIT5));

return;

}

void right\_fw() {

P2->OUT &= (~BIT7);

P2->OUT |= BIT6;

return;

}

void right\_bw() {

P2->OUT &= (~BIT6);

P2->OUT |= BIT7;

return;

}

void right\_off() {

P2->OUT &= (~(BIT6 | BIT7));

return;

}

void forward(int time) {

right\_fw(); // Make both motors spin forward for a set time

left\_fw();

SysTick\_Delayms(time);

right\_off(); // Turn off both motors

left\_off();

return;

}

void backward(int time) {

right\_bw(); // Make both motors spin backwards for a set time

left\_bw();

SysTick\_Delayms(time);

right\_off(); // Turn off both motors

left\_off();

return;

}

void turn(int angle) {

if (angle == 0 || angle > 180 || angle < -180) { // Invalid angles

} else if (angle > 0){ // Counter-clockwise turns, spin right wheel forward and left wheel backwards

right\_fw();

left\_bw();

SysTick\_Delayms(angle \* timePerAngle);

right\_off();

left\_off();

} else { // Clockwise turns, spin left wheel forward and right wheel backwards

right\_bw();

left\_fw();

SysTick\_Delayms(angle \* (-1) \* timePerAngle);

right\_off();

left\_off();

}

return;

}

**uart.h**

#include "msp.h"

//UART0 is at the micro-USB port. UART2 is at pin 3.2 and pin 3.3. These two ports allow serial communication between the microcontroller and the ESP32-CAM.

//This function initializes UART0.

void UART0\_init();

//This function sends string through UART0 by spin waiting.

int UART0\_puts(const char \*str);

//This function initializes UART2.

void UART2\_init();

//This function sends string through UART2 by spin waiting. The transmit pin is pin 3.3.

int UART2\_puts(const char \*str);

//This function receives string through UART2 by spin waiting. The receive pin is pin 3.2.

//It returns after the receiving line feed ‘\n’ or end of the string ‘\0’

int UART2\_gets(char \*str);

**uart.c**

#include "msp.h"

#include "uart.h"

void UART0\_init();

int UART0\_puts(const char \*str);

void UART2\_init();

int UART2\_puts(const char \*str);

int UART2\_gets(char \*str);

/\*\*

\* This function is initializing the UART0(micro-USB port) for sending distance message to the console with baud rate of 9600

\*

\*/

void UART0\_init()

{

// Configure UART pins

P1->SEL0 |= BIT2 | BIT3; // set 2-UART pin as secondary function

// Configure UART

EUSCI\_A0->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Put eUSCI in reset

EUSCI\_A0->CTLW0 |= EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // Configure eUSCI clock source for SMCLK

// Baud Rate calculation

// 12000000/(16\*9600) = 78.125

// Fractional portion = 0.125

// User's Guide Table 21-4: UCBRSx = 0x10

// UCBRFx = int ( (78.125-78)\*16) = 2

EUSCI\_A0->BRW = 78; // 12000000/16/9600

EUSCI\_A0->MCTLW = (2 << EUSCI\_A\_MCTLW\_BRF\_OFS) |

EUSCI\_A\_MCTLW\_OS16;

EUSCI\_A0->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // take eUSCI out of reset mode

EUSCI\_A0->IFG &= ~EUSCI\_A\_IFG\_RXIFG; // Clear eUSCI RX interrupt flag

EUSCI\_A0->IE &= ~EUSCI\_A\_IE\_RXIE; // Disable USCI\_A0 RX interrupt

}

/\*\*

\* This function puts a string to transmit buffer in the UART0, which will be sent to console

\*/

int UART0\_puts(const char \*str)

{

int status = -1;

if (str != '\0')

{

status = 0;

while (\*str != '\0')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

/\* Transmit data \*/

EUSCI\_A0->TXBUF = \*str;

/\* If there is a line-feed, add a carriage return \*/

if (\*str == '\n')

{

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG))

;

EUSCI\_A0->TXBUF = '\r';

}

str++;

}

}

return status;

}

/\*\*

\* This function is initializing the UART2(pin 3.2,3.3) for sending distance message to the console with baud rate of 9600

\*

\*/

void UART2\_init()

{

//pin 3.2 RX pin 3.3 TX

// Configure UART pins

P3->SEL0 |= BIT2 | BIT3; // set 2-UART pin as secondary function

P3->SEL1 &= ~(BIT2 | BIT3);

// Configure UART

EUSCI\_A2->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Put eUSCI in reset

EUSCI\_A2->CTLW0 |= EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // Configure eUSCI clock source for SMCLK

// Baud Rate calculation

// 12000000/(16\*9600) = 78.125

// Fractional portion = 0.125

// User's Guide Table 21-4: UCBRSx = 0x10

// UCBRFx = int ( (78.125-78)\*16) = 2

EUSCI\_A2->BRW = 78; // 12000000/16/9600

EUSCI\_A2->MCTLW = (2 << EUSCI\_A\_MCTLW\_BRF\_OFS) |

EUSCI\_A\_MCTLW\_OS16;

EUSCI\_A2->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // take eUSCI out of reset mode

EUSCI\_A2->IFG &= ~EUSCI\_A\_IFG\_RXIFG; // Clear eUSCI RX interrupt flag

EUSCI\_A2->IE &= ~EUSCI\_A\_IE\_RXIE; // Disable USCI\_A2 RX interrupt

}

/\*\*

\* This function puts a string to transmit buffer in the UART2, which will be sent to console

\*/

int UART2\_puts(const char \*str)

{

int status = -1;

if (str != '\0') {

status = 0;

while (\*str != '\0') {

/\* Wait for the transmit buffer to be ready \*/

while (!(EUSCI\_A2->IFG & EUSCI\_A\_IFG\_TXIFG));

/\* Transmit data \*/

EUSCI\_A2->TXBUF = \*str;

str++;

}

}

return status;

}

int UART2\_gets(char \*str)

{

while (1) {

/\* Wait for the receive buffer to be ready \*/

while (!(EUSCI\_A2->IFG & EUSCI\_A\_IFG\_RXIFG));

/\* Transmit data \*/

\*str = EUSCI\_A2->RXBUF;

if(\*str == '\n' || \*str == '\0')

{

break;

}

str++;

}

return 1;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**"us.h"**

#include "msp.h"

#include <stdint.h>

#include <stdio.h>

//higher number has lower priority

#define PORT4\_PRIO 4

#define TIMEA0\_PRIO 3

#define INTR\_PRD 1000

#define DIST\_DIV 1400

#define SAMPLE\_SIZE 3

//This function initializes the clock signal

void Clock\_init();

//This function initializes the TIMERA0 for recording time\_stamp

void TIMERA0\_init();

//This function initializes pins for trigger ultrasonic sensor and record echo signals.

//It uses pin 4.6 as trigger and pin 4.7 for echo. It also registers IRQ for receiving interrupt from rising or falling edge at pin 4.7

void US\_init();

//This function measures the distance between the ultrasonic sensor and the object in front of it.

//It triggers the sensor several times and return the average value in cm.

//The specific size of the sample can be modified by changing the macro in us.h. The range of distance is (0, 300)

long measure\_dist();

//This function returns the time passed since the start of the measurement (marked by US\_init). The time stamp is record as â€œmin,secâ€ in string

char\* get\_time\_stamp();

void Delay(uint32\_t tick);

us.c

#include "us.h"

long intrcnt = 0;

long clktick = 0;

long start = 0;

long end = 0;

int min = 0;

int sec = 0;

char buffer[20];

long US\_scan\_once();

void Clock\_init()

{

//setting up clock signals

CS->KEY = CS\_KEY\_VAL; // Unlock CS module for register access

CS->CTL0 = 0; // Reset tuning parameters

CS->CTL0 = CS\_CTL0\_DCORSEL\_3; // Set DCO to 24MHz

CS->CTL1 = CS\_CTL1\_SELA\_2 | // Select ACLK = REFO

CS\_CTL1\_SELS\_3 | // SMCLK = DCO

CS\_CTL1\_SELM\_3; // MCLK = DCO

CS->KEY = 0; // Lock CS module from unintended accesses

}

void TIMERA0\_init()

{

//setting up timer A0

TIMER\_A0->CCTL[0] = TIMER\_A\_CCTLN\_CCIE; // CCR0 interrupt enabled

TIMER\_A0->CCR[0] = INTR\_PRD - 1; // interrupt is raised for every 1000 clock tick

TIMER\_A0->CTL = TIMER\_A\_CTL\_TASSEL\_2 | TIMER\_A\_CTL\_MC\_\_UP | TIMER\_A\_CTL\_CLR; // SMCLK, upmode, TA clear

// Enable Timer interrupt for timing and set priority as 3

NVIC\_SetPriority(TA0\_0\_IRQn, TIMEA0\_PRIO);

NVIC\_EnableIRQ(TA0\_0\_IRQn);

}

void US\_init()

{

// Configure GPIO

//set P4 as GPIO

P4->SEL0 = 0;

P4->SEL1 = 0;

// uses P4.7 as echo

P4->DIR &= ~BIT7; // P4.7 as input pin

P4->REN |= BIT7; // P4.7 pull resistor enabled

P4->OUT &= ~BIT7; // P4.7 selected as pull down (active low)

// uses P4.6 as trigger

P4->DIR |= BIT6; // trigger pin as output

// receiving interrupt from P4.7

P4->IFG = 0; // clean pending interrupt flag

P4->IES &= ~BIT7; // enable rising edge interrupt

P4->IE |= BIT7; // enable interrupt

// Enable Port4 interrupt for echo and set priority as 4

NVIC\_SetPriority(PORT4\_IRQn, PORT4\_PRIO);

NVIC\_EnableIRQ(PORT4\_IRQn);

}

/\*\*

\* This function delay for the given amount of clock cycle

\*/

void Delay(uint32\_t tick)

{

// initialize timer32 1 with give amount of tick

TIMER32\_1->LOAD = tick;

//no prescaler, periodic wrapping mode, disable interrupt, 32-bit timer

TIMER32\_1->CONTROL = 0xc2;

//spin wait until the time is reached

while ((TIMER32\_1->RIS & 1) == 0)

;

TIMER32\_1->INTCLR = 0; //clear raw interrupt flag

}

long US\_scan\_once()

{

// uses pin 4.6 for trigger

P4->OUT |= BIT6; // generate pulse

Delay(240); // 240 cycle in 24mhz is about 10us

P4->OUT &= ~BIT6; // stop pulse

P4->IFG = 0; // clear P4 interrupt just in case anything happened before

P4->IES &= ~BIT7; // wait for rising edge on ECHO pin

Delay(660000); //set this to 100us - 60ms

if(end > start)

{

return (end - start) / DIST\_DIV; // converting ECHO time into cm

}

else

{

return 300;

}

}

long measure\_dist()

{

long res = 0;

int i;

for(i = 0; i < SAMPLE\_SIZE; i++)

{

long dist = US\_scan\_once();

if(dist > 300)

res += 300;

else

res += dist;

}

return res/SAMPLE\_SIZE;

}

char\* get\_time\_stamp()

{

sec = intrcnt / 24000;

min = sec / 60;

sec %= 60;

sprintf(buffer, "%d,%d", min, sec);

return buffer;

}

/\*\*

\* Port4 interrupt service routine, start timing echo from rising edge to falling edge

\*/

void PORT4\_IRQHandler(void)

{

// check if interrupt is pending for P4.7

if (P4->IFG & BIT7)

{

//if we are checking for raising edge, we are at the beginning of the echo

if (!(P4->IES & BIT7 ))

{

//record starting time of the echo

start = intrcnt \* 1000 + (long) TIMER\_A0->R;

//now checks for falling edge, which is the end of the echo

P4->IES |= BIT7;

}

else

{

end = intrcnt \* 1000 + (long) TIMER\_A0->R; //calculating ECHO length

//now back to checking rising edge, which is the beginning of the echo

P4->IES &= ~BIT7;

}

P4->IFG &= ~BIT7; //clear flag

}

}

/\*\*

\* TimerA0 interrupt service routine, updates count of interrupt and clears interrupt flag

\*/

void TA0\_0\_IRQHandler(void)

{

// Interrupt gets triggered for every 1000 clock cycle in SMCLK

intrcnt++;

TIMER\_A0->CCTL[0] &= ~TIMER\_A\_CCTLN\_CCIFG;

}

Systick0.h

// Header to initialize SysTick for simple delay in ms or us

#include "msp.h"

#include <stdio.h>

void SysTick\_Init();

void SysTick\_Delayms();

void SysTick\_Delayus();

Systick0.c

// Functions to initialize SysTick for simple delay in ms or us

#include "msp.h"

#include <stdio.h>

#include "SysTick0.h"

void SysTick\_Init();

void SysTick\_Delayms();

void SysTick\_Delayus();

// Function to initialize SysTick without interrupt enable

// No longer necessary, delay functions work indpendently

void SysTick\_Init(){

SysTick->CTRL = 0; // disable

SysTick->LOAD = 0x00FFFFFF; // set reload to max

SysTick->VAL = 0; // clear counter value

SysTick->CTRL = 0x00000005 ; // set CLKSOURCE to core clock and enable to 1

// clock is 3MHz, CTRL has interrupts disabled

// tech reference manual 2.4.4.1

return;

}

// Delay function with argument in milliseconds

void SysTick\_Delayus(int time){

SysTick->LOAD = (time \* 3 - 1); // 3000t / 3 MHz = t ms

// minimum 25 us, scales oddly until ~100 us

SysTick->VAL = 0; // any write to VAL clears VAL, restart

SysTick->CTRL = 0x00000005 ; // set CLKSOURCE to core clock and enable to 1

while(!(SysTick->CTRL & 0x00010000)); // stays in loop until COUNTFLAG is true

return;

}

// Delay function with argument in milliseconds

void SysTick\_Delayms(int time){

SysTick->LOAD = (time \* 3000 - 1); // 3000t / 3 MHz = t ms

SysTick->VAL = 0; // any write to VAL clears VAL, restart

SysTick->CTRL = 0x00000005 ; // set CLKSOURCE to core clock and enable to 1

while(!(SysTick->CTRL & 0x00010000)); // stays in loop until COUNTFLAG is true

return;

}