ДОДАТОК А

Лістинг програмного коду пульта керування

```
//connecting libraries
#include "WiFi.h"
#include "esp_now.h"
#include "TFT_eSPI.h"
#include "TJpg_Decoder.h"
#include "GyverButton.h"
#include "SPI.h"
#include "ArduinoWebsockets.h"

const char *ssid = "Enter Wi-Fi name"; //connecting to Wi-Fi
const char *password = "Enter Wi-Fi password";

#define leftJoystickX_pin 36
#define leftJoystickY_pin 34

#define rightJoystickX_pin 39
#define rightJoystickY_pin 38

#define autoLed    21
#define handLed    22
#define mineLed    5

GButton AutoModeButt (19);
GButton HandModeButt (18);
GButton ManipulatorButt (23);
GButton JoystickSwitchButt (1);

using namespace websockets;
WebsocketsServer server;
WebsocketsClient client;

//display setting
TFT_eSPI tft = TFT_eSPI();         // Invoke custom library

bool tft_output(int16_t x, int16_t y, uint16_t w, uint16_t h, uint16_t* bitmap)
{
   // Stop further decoding as image is running off bottom of screen
  if ( y >= tft.height() ) return 0;

  // This function will clip the image block rendering automatically at the TFT boundaries
```

```
  tft.pushImage(x, y, w, h, bitmap);

  // This might work instead if you adapt the sketch to use the Adafruit_GFX library
  // tft.drawRGBBitmap(x, y, bitmap, w, h);

  // Return 1 to decode next block
  return 1;
}

//mac-address of the receivering (Robot) board
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

//data sending structure
typedef struct struct_message {
  byte leftJoystickX;
  byte leftJoystickY;
  byte rightJoystickX;
  byte rightJoystickY;
  String mode;
  String arm;
  String gripper;
  String leftObstacle;
  String frontObstacle;
  String rightObstacle;
  String mineDetected;
} struct_message;
struct_message data;

//get data from CAM
typedef struct struct_Message {
  String leftObstacle;
  String frontObstacle;
  String rightObstacle;
  String mineDetected;
} struct_Message;
struct_Message dataCAM;

esp_now_peer_info_t peerInfo;

void getDataCam(){
  data.leftObstacle = dataCAM.leftObstacle;
  data.frontObstacle = dataCAM.frontObstacle;
  data.rightObstacle = dataCAM.rightObstacle;
  data.mineDetected = dataCAM.mineDetected;
}
```

```
void OnDataSent(const uint8_t * 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void OnDataRecv(const uint8_t * 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, const uint8_t *incomingData, int len) {
  memcpy(&dataCAM, incomingData, sizeof(dataCAM));
  Serial.print("Bytes received: ");
  Serial.println(len);
  Serial.print("Left Obstacle: ");
  Serial.println(dataCAM.leftObstacle);
  Serial.print("Front Obstacle: ");
  Serial.println(dataCAM.frontObstacle);
  Serial.print("Right Obstacle: ");
  Serial.println(dataCAM.rightObstacle);
  Serial.println();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  tft.begin();
  tft.setRotation(3);
  tft.setTextColor(0xFFFF, 0x0000);
  tft.fillScreen(TFT_BLUE);
  tft.setSwapBytes(true); // We need to swap the colour bytes (endianess)

  // The jpeg image can be scaled by a factor of 1, 2, 4, or 8
  TJpgDec.setJpgScale(1);

  // The decoder must be given the exact name of the rendering function above
  TJpgDec.setCallback(tft_output);

  Serial.println();
  Serial.println("Setting AP...");
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP Address : ");
  Serial.println(IP);
```

```
  server.listen(8888);

  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  pinMode(autoLed, OUTPUT);
  pinMode(handLed, OUTPUT);
  pinMode(mineLed, OUTPUT);

  esp_now_register_recv_cb(OnDataRecv);
}

void HandMode() {
  data.leftJoystickX = analogRead(leftJoystickX_pin);
  data.leftJoystickY = analogRead(leftJoystickY_pin);
  data.rightJoystickX = analogRead(rightJoystickX_pin);
  data.rightJoystickY = analogRead(rightJoystickY_pin);
}

//for switching right joystick modes
void setTimeout(uint16_t 3000);

void loop() {
  AutoModeButt.tick();
  HandModeButt.tick();
  ManipulatorButt.tick();
  joystickSwitchButt.tick();

  HandMode();

  if(server.poll()){
    client = server.accept();
  }

  if(client.available()){
    client.poll();

    WebsocketsMessage msg = client.readBlocking();

    uint32_t t = millis();

    // Get the width and height in pixels of the jpeg if you wish
```

```
    uint16_t w = 0, h = 0;
    TJpgDec.getJpgSize(&w, &h, (const uint8_t*)msg.c_str(), msg.length());
    Serial.print("Width = "); Serial.print(w); Serial.print(", height = ");
Serial.println(h);

    // Draw the image, top left at 0,0
    TJpgDec.drawJpg(0, 0, (const uint8_t*)msg.c_str(), msg.length());

    // How much time did rendering take
    t = millis() - t;
    Serial.print(t); Serial.println(" ms");
  }
  if (AutoModeButt.isPress()){
    //avto control mode
    digitalWrite(autoLed, HIGH);
    digitalWrite(handLed, LOW);
    data.mode = "Auto";
    getDataCam();
  }
  if (HandModeButt.isPress()){
    digitalWrite(handLed, HIGH);
    digitalWrite(autoLed, LOW);
    data.mode = "Hand";
    HandMode();
  }
  if (ManipulatorButt.isPress()){
    data.arm = "Enable";
    if (JoystickSwitchButt.isPress()){
      data.gripper = "True";
    }
  }
  else if (ManipulatorButt.isHolded()){
    data.arm = "Disable";
  }
  esp_err_t result = esp_now_send(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, (uint8_t
*) &data, sizeof(data));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
}
```

ДОДАТОК Б

Лістинг програмного коду системи керування роботом-розмінувальником

```
//connecting libraries
#include "WiFi.h"
#include "esp_now.h"
#include "ESP32Servo.h"
#include "Robojax_L298N_DC_motor.h"

const char *ssid = "Enter Wi-Fi name"; //connecting to Wi-Fi
const char *password = "Enter Wi-Fi password";

// drive motor 1 initializing
#define ENA 19
#define IN1 18
#define IN2 16

// drive motor 2 initializing
#define IN3 12
#define IN4 13
#define ENB 5

const int CCW = 2;
const int CW  = 1;

#define motor1 1
#define motor2 2

//manipulator initializing

#define SERVO_FORWARD_STEP_ANGLE 1
#define SERVO_BACKWARD_STEP_ANGLE -1

struct ServoPins
{
  Servo servo;
  int servoPin;
  String servoName;
  int initialPosition;
};

std::vector<ServoPins> servoPins =
{
  { Servo(), 17 , "Base", 90},
```

```
  { Servo(), 25 , "Shoulder", 90},
  { Servo(), 26 , "Elbow", 90},
  { Servo(), 27 , "Gripper", 90},
};

//starting Robojax_L298N_DC_motor library
Robojax_L298N_DC_motor motor(IN1, IN2, ENA, IN3, IN4, ENB, true);

//motors setting
struct MotorsState {
  int speed_motor1 = 0;
  int speed_motor2 = 0;
  char direction_motor1 = CCW;
  char direction_motor2 = CW;
} motors_state;

//get data from Pult
typedef struct struct_message {
  byte leftJoystickX;
  byte leftJoystickY;
  byte rightJoystickX;
  byte rightJoystickY;
  String mode;
  String arm;
  String gripper;
  String leftObstacle;
  String frontObstacle;
  String rightObstacle;
  String mineDetected;
} struct_message;
struct_message receiverData;

//travel functions
void Stop(){
  motors.brake(1);
  motors.brake(2);
}

void Forvard() {
  motors_state.direction_motor2 = CW;
  motors_state.direction_motor1 = CCW;
  motors.rotate(motor2,                    motors_state.speed_motor2,
motors_state.direction_motor2);
  motors.rotate(motor1,                    motors_state.speed_motor1,
motors_state.direction_motor1);
```

```
}

void Backward(){
  motors_state.direction_motor2 = CCW;
  motors_state.direction_motor1 = CW;
  motors.rotate(motor2,                                    motors_state.speed_motor2,
motors_state.direction_motor2);
  motors.rotate(motor1,                                    motors_state.speed_motor1,
motors_state.direction_motor1);
}

void Left(){
  motors_state.direction_motor1 = CCW;
  motors.rotate(motor1,                                    motors_state.speed_motor1,
motors_state.direction_motor1);
}

void Right(){
  motors_state.direction_motor2 = CW;
  motors.rotate(motor2,                                    motors_state.speed_motor2,
motors_state.direction_motor2);
}

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  if (len == 0)
  {
    return;
  }
  memcpy(&receiverData, incomingData, sizeof(receiverData));

  if (receiverData.arm == "Disable"){

    //scale the value to the interval 0-180
    int motorY = map(receiverData.rightJoystickY, 0, 1023, 0, 180);
    int motorX = map(receiverData.rightJoystickX, 0, 1023, 0, 180);
    int armY = map(receiverData.leftJoystickY, 0, 1023, 0, 180);
    int armX = map(receiverData.leftJoystickX, 0, 1023, 0, 180);


    if(motorY > 90){                  //forward
      Forvard();
    }
    else if(motorY < 90){              //backward
```

```
  Backward();
 }

 if(motorX < 90){              //Right
  Right();
 }
 else if(motorX > 90){             //Left
  Left();
 }

 if(motorX && motorY = 90)  {
  Stop()
 }

 //Base
 if (armX > 90)
   {
    writeServoValues(0, SERVO_FORWARD_STEP_ANGLE);
   }
 else if (armX < 90)
   {
    writeServoValues(0, SERVO_BACKWARD_STEP_ANGLE);
   }

 //Shoulder
 if (armY > 90)
   {
    writeServoValues(1, SERVO_FORWARD_STEP_ANGLE);
   }
 else if (armY < 90)
   {
    writeServoValues(1, SERVO_BACKWARD_STEP_ANGLE);
   }
}

else if (receiverData.arm == "Enable") {
 int rightY = map( receiverData.rightJoystickY, 0, 1023, 0, 180 );
 int rightX = map( receiverData.rightJoystickX, 0, 1023, 0, 180 );
 int leftY = map( receiverData.leftJoystickY, 0, 1023, 0, 180 );
 int leftX = map( receiverData.leftJoystickX, 0, 1023, 0, 180 );

 if (receiverData.arm == "Enable") {

  //Base
  if (leftX > 90)
```

```
    {
      writeServoValues(0, SERVO_FORWARD_STEP_ANGLE);
    }
    else if (leftX < 90)
    {
      writeServoValues(0, SERVO_BACKWARD_STEP_ANGLE);
    }

    //Shoulder
    if (leftY > 90)
    {
      writeServoValues(1, SERVO_FORWARD_STEP_ANGLE);
    }
    else if (leftY < 90)
    {
      writeServoValues(1, SERVO_BACKWARD_STEP_ANGLE);
    }

    //Elbow
    if (rightX > 90)
    {
      writeServoValues(2, SERVO_FORWARD_STEP_ANGLE);
    }
    else if (rightX < 90)
    {
      writeServoValues(2, SERVO_BACKWARD_STEP_ANGLE);
    }

    //Gripper
    if (rightY > 90)
    {
      writeServoValues(3, SERVO_FORWARD_STEP_ANGLE);
    }
    else if (rightY < 90)
    {
      writeServoValues(3, SERVO_BACKWARD_STEP_ANGLE);
    }

    //if left joistic was presd
    if (receiverData.gripper == "True")
    {
      gripperSwitch = !gripperSwitch;  //Toggle gripper close / open
      gripperSwitch ? writeServoValues(3, 170, true) :   writeServoValues(3, 100,
true) ;
      receiverData.gripper == "False";
```

```
      }

      delay(10);
    }
  }
}

void setup()
{
  Serial.begin(115200);
  motor.begin();

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");


  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  //for check
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  //receiving data
  esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
  if (receiverData.mode == "Auto"){

    Forvard();

    if (receiverData.leftObstacle == "True"){
      if (receiverData.mineDetected == "True"){
        Stop();
        receiverData.mode = "Hand";
      }
      else if (receiverData.rightObstacle == "False"){
```

```
Right();
delay(700);
if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.leftObstacle == "False"){
    Left();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.rightObstacle == "False"){
            Right();
            delay(700);
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else if (receiverData.rightObstacle == "False"){
          Right();
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(1500);
            if (receiverData.leftObstacle == "False"){
              Left();
              delay(700);
              if (receiverData.frontObstacle == "False"){
                Forvard();
                delay(700);
                if (receiverData.rightObstacle == "False"){
                  Right();
                  delay(700);
                } else {
                  Stop();
                  receiverData.mode = "Hand";
                }
              } else {
                Stop();
```

```
          receiverData.mode = "Hand";
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
     } else {
      Stop();
      receiverData.mode = "Hand";
     }
    }
  } else if (receiverData.frontObstacle == "False"){
   Forvard();
   delay(1500);
   if (receiverData.leftObstacle == "False"){
   Left();
   delay(700);
   if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.rightObstacle == "False"){
     Right();
     delay(700);
    } else {
     Stop();
     receiverData.mode = "Hand";
    }
   } else if (receiverData.rightObstacle == "False"){
    Right();
    delay(700);
    if (receiverData.frontObstacle == "False"){
     Forvard();
     delay(1500);
     if (receiverData.leftObstacle == "False"){
      Left();
      delay(700);
      if (receiverData.frontObstacle == "False"){
       Forvard();
       delay(700);
       if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
       } else {
        Stop();
        receiverData.mode = "Hand";
```

```
          }
         } else {
          Stop();
          receiverData.mode = "Hand";
         }
        } else {
         Stop();
         receiverData.mode = "Hand";
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
      }
     } else {
      Stop();
      receiverData.mode = "Hand";
     }
    } else if (receiverData.rightObstacle == "False"){
     Right();
     delay(700);
     if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
       Left();
       delay(700);
       if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.leftObstacle == "False"){
         Left();
         delay(700);
         if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.rightObstacle == "False"){
           Right();
           delay(700);
          } else {
           Stop();
           receiverData.mode = "Hand";
          }
         } else if (receiverData.rightObstacle == "False"){
          Right();
```

```
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(1500);
            if (receiverData.leftObstacle == "False"){
              Left();
              delay(700);
              if (receiverData.frontObstacle == "False"){
                Forvard();
                delay(700);
                if (receiverData.rightObstacle == "False"){
                  Right();
                  delay(700);
                } else {
                  Stop();
                  receiverData.mode = "Hand";
                }
              } else {
                Stop();
                receiverData.mode = "Hand";
              }
            } else {
              Stop();
              receiverData.mode = "Hand";
            }
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  } else {
    Stop();
    receiverData.mode = "Hand";
  }
} else {
  Stop();
  receiverData.mode = "Hand";
```

```
    }
   }
  } else if (receiverData.frontObstacle == "False"){
   Forvard();
   delay(1500);
   if (receiverData.leftObstacle == "False"){
      Left();
      delay(700);
      if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.rightObstacle == "False"){
          Right();
          delay(700);
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.leftObstacle == "False"){
            Left();
            delay(700);
            if (receiverData.frontObstacle == "False"){
              Forvard();
              delay(700);
              if (receiverData.rightObstacle == "False"){
                Right();
                delay(700);
              } else {
                Stop();
                receiverData.mode = "Hand";
              }
            } else {
              Stop();
              receiverData.mode = "Hand";
            }
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
```

```
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  }
} else if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.leftObstacle == "False"){
  Left();
  delay(700);
  if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.rightObstacle == "False"){
      Right();
      delay(700);
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  } else if (receiverData.rightObstacle == "False"){
    Right();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(700);
          if (receiverData.rightObstacle == "False"){
            Right();
            delay(700);
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
```

```
        Stop();
        receiverData.mode = "Hand";
       }
      } else {
       Stop();
       receiverData.mode = "Hand";
      }
     }
    } else {
     Stop();
     receiverData.mode = "Hand";
    }
   } else if (receiverData.rightObstacle == "False"){
    Right();
    delay(700);
    if (receiverData.frontObstacle == "False"){
     Forvard();
     delay(1500);
     if (receiverData.leftObstacle == "False"){
      Left();
      delay(700);
      if (receiverData.frontObstacle == "False"){
       Forvard();
       delay(1500);
       if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
        if (receiverData.frontObstacle == "False"){
         Forvard();
         delay(1500);
         if (receiverData.rightObstacle == "False"){
          Right();
          delay(250);
         } else {
          Stop();
          receiverData.mode = "Hand";
         }
        } else if (receiverData.rightObstacle == "False"){
         Right();
         delay(700);
         if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.leftObstacle == "False"){
           Left();
```

```
                delay(700);
                if (receiverData.frontObstacle == "False"){
                  Forvard();
                  delay(700);
                  if (receiverData.rightObstacle == "False"){
                    Right();
                    delay(700);
                  } else {
                    Stop();
                    receiverData.mode = "Hand";
                  }
                } else {
                  Stop();
                  receiverData.mode = "Hand";
                }
              } else {
                Stop();
                receiverData.mode = "Hand";
              }
            } else {
              Stop();
              receiverData.mode = "Hand";
            }
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  }
} else {
  Stop();
  receiverData.mode = "Hand";
}
```

```
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  }
}

if (receiverData.rightObstacle == "True"){
  if (receiverData.mineDetected == "True"){
    Stop();
    receiverData.mode = "Hand";
  }

  else if (receiverData.leftObstacle == "False"){
    Left();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.rightObstacle == "False"){
            Right();
            delay(700);
            if (receiverData.frontObstacle == "False"){
              Forvard();
              delay(1500);
              if (receiverData.leftObstacle == "False"){
                Left();
                delay(700);
              } else {
                Stop();
                receiverData.mode = "Hand";
              }
            } else {
              Stop();
              receiverData.mode = "Hand";
```

```
      }
    } else if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.leftObstacle == "False"){
            Left();
            delay(700);
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  } else if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.rightObstacle == "False"){
      Right();
      delay(700);
      if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.rightObstacle == "False"){
          Right();
          delay(700);
```

```
  if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.leftObstacle == "False"){
      Left();
      delay(700);
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  } else {
    Stop();
    receiverData.mode = "Hand";
  }
 } else if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.rightObstacle == "False"){
    Right();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  } else {
    Stop();
    receiverData.mode = "Hand";
  }
 } else {
  Stop();
  receiverData.mode = "Hand";
```

```
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
      } else {
       Stop();
       receiverData.mode = "Hand";
      }
    } else {
     Stop();
     receiverData.mode = "Hand";
    }
  } else {
   Stop();
   receiverData.mode = "Hand";
  }
}

if (receiverData.frontObstacle == "True"){
  if (receiverData.mineDetected == "True"){
    Stop();
    receiverData.mode = "Hand";
  }

  else if (receiverData.rightObstacle == "False"){
      Right();
      delay(700);
      if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.leftObstacle == "False"){
          Left();
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(1500);
            if (receiverData.leftObstacle == "False"){
              Left();
              delay(700);
              if (receiverData.frontObstacle == "False"){
                Forvard();
                delay(1500);
                if (receiverData.rightObstacle == "False"){
                  Right();
```

```
      delay(700);
     } else {
      Stop();
      receiverData.mode = "Hand";
     }
    } else if (receiverData.rightObstacle == "False"){
     Right();
     delay(700);
     if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
       Left();
       delay(700);
       if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(700);
        if (receiverData.rightObstacle == "False"){
         Right();
         delay(700);
        } else {
         Stop();
         receiverData.mode = "Hand";
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
      } else {
       Stop();
       receiverData.mode = "Hand";
      }
     } else {
      Stop();
      receiverData.mode = "Hand";
     }
    }
   } else if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.leftObstacle == "False"){
    Left();
    delay(700);
    if (receiverData.frontObstacle == "False"){
     Forvard();
```

```
      delay(1500);
      if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else if (receiverData.rightObstacle == "False"){
      Right();
      delay(700);
      if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.leftObstacle == "False"){
          Left();
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(700);
            if (receiverData.rightObstacle == "False"){
              Right();
              delay(250);
            } else {
              Stop();
              receiverData.mode = "Hand";
            }
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    }
  } else {
    Stop();
    receiverData.mode = "Hand";
  }
} else if (receiverData.rightObstacle == "False"){
```

```
Right();
delay(700);
if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.leftObstacle == "False"){
    Left();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.rightObstacle == "False"){
            Right();
            delay(700);
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else if (receiverData.rightObstacle == "False"){
          Right();
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(1500);
            if (receiverData.leftObstacle == "False"){
              Left();
              delay(700);
              if (receiverData.frontObstacle == "False"){
                Forvard();
                delay(700);
                if (receiverData.rightObstacle == "False"){
                  Right();
                  delay(700);
                } else {
                  Stop();
                  receiverData.mode = "Hand";
                }
              } else {
                Stop();
```

```
                receiverData.mode = "Hand";
              }
             } else {
              Stop();
              receiverData.mode = "Hand";
             }
            } else {
             Stop();
             receiverData.mode = "Hand";
            }
           }
          } else {
           Stop();
           receiverData.mode = "Hand";
          }
         } else {
          Stop();
          receiverData.mode = "Hand";
         }
        } else {
         Stop();
         receiverData.mode = "Hand";
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
      } else {
       Stop();
       receiverData.mode = "Hand";
      }
     }
    } else if (receiverData.frontObstacle == "False"){
     Forvard();
     delay(1500);
     if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
        if (receiverData.frontObstacle == "False"){
         Forvard();
         delay(1500);
         if (receiverData.rightObstacle == "False"){
          Right();
          delay(700);
         } else {
          Stop();
          receiverData.mode = "Hand";
         }
        } else if (receiverData.rightObstacle == "False"){
```

```
    Right();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
        Left();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(700);
          if (receiverData.rightObstacle == "False"){
            Right();
            delay(700);
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  }
} else if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.leftObstacle == "False"){
  Left();
  delay(700);
  if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.rightObstacle == "False"){
      Right();
      delay(700);
    } else {
      Stop();
```

```
      receiverData.mode = "Hand";
     }
    } else if (receiverData.rightObstacle == "False"){
     Right();
     delay(700);
     if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.leftObstacle == "False"){
       Left();
       delay(700);
       if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(700);
        if (receiverData.rightObstacle == "False"){
         Right();
         delay(700);
        } else {
         Stop();
         receiverData.mode = "Hand";
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
      } else {
       Stop();
       receiverData.mode = "Hand";
      }
     } else {
      Stop();
      receiverData.mode = "Hand";
     }
    }
   } else {
    Stop();
    receiverData.mode = "Hand";
   }
  } else if (receiverData.rightObstacle == "False"){
   Right();
   delay(700);
   if (receiverData.frontObstacle == "False"){
    Forvard();
    delay(1500);
    if (receiverData.leftObstacle == "False"){
```

```
Left();
delay(700);
if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.leftObstacle == "False"){
    Left();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else if (receiverData.rightObstacle == "False"){
      Right();
      delay(700);
      if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.leftObstacle == "False"){
          Left();
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(700);
            if (receiverData.rightObstacle == "False"){
              Right();
              delay(700);
            } else {
              Stop();
              receiverData.mode = "Hand";
            }
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
```

```
                } else {
                  Stop();
                  receiverData.mode = "Hand";
                }
               }
              } else {
               Stop();
               receiverData.mode = "Hand";
              }
             } else {
              Stop();
              receiverData.mode = "Hand";
             }
            } else {
             Stop();
             receiverData.mode = "Hand";
            }
           } else {
            Stop();
            receiverData.mode = "Hand";
           }
          } else {
           Stop();
           receiverData.mode = "Hand";
          }
         } else {
          Stop();
          receiverData.mode = "Hand";
         }
        } else {
         Stop();
         receiverData.mode = "Hand";
        }
       } else {
        Stop();
        receiverData.mode = "Hand";
       }
      } else if (receiverData.leftObstacle == "False"){
       Left();
       delay(700);
       if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.rightObstacle == "False"){
         Right();
         delay(700);
         if (receiverData.frontObstacle == "False"){
          Forvard();
```

```
delay(1500);
if (receiverData.rightObstacle == "False"){
 Right();
 delay(700);
 if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.leftObstacle == "False"){
   Left();
   delay(700);
  } else {
   Stop();
   receiverData.mode = "Hand";
  }
 } else {
  Stop();
  receiverData.mode = "Hand";
 }
} else if (receiverData.frontObstacle == "False"){
 Forvard();
 delay(1500);
 if (receiverData.rightObstacle == "False"){
  Right();
  delay(700);
  if (receiverData.frontObstacle == "False"){
   Forvard();
   delay(1500);
   if (receiverData.leftObstacle == "False"){
    Left();
    delay(700);
   } else {
    Stop();
    receiverData.mode = "Hand";
   }
  } else {
   Stop();
   receiverData.mode = "Hand";
  }
 } else {
  Stop();
  receiverData.mode = "Hand";
 }
} else {
 Stop();
 receiverData.mode = "Hand";
```

```
    }
  } else {
    Stop();
    receiverData.mode = "Hand";
  }
} else if (receiverData.frontObstacle == "False"){
  Forvard();
  delay(1500);
  if (receiverData.rightObstacle == "False"){
    Right();
    delay(700);
    if (receiverData.frontObstacle == "False"){
      Forvard();
      delay(1500);
      if (receiverData.rightObstacle == "False"){
        Right();
        delay(700);
        if (receiverData.frontObstacle == "False"){
          Forvard();
          delay(1500);
          if (receiverData.leftObstacle == "False"){
            Left();
            delay(700);
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else if (receiverData.frontObstacle == "False"){
        Forvard();
        delay(1500);
        if (receiverData.rightObstacle == "False"){
          Right();
          delay(700);
          if (receiverData.frontObstacle == "False"){
            Forvard();
            delay(1500);
            if (receiverData.leftObstacle == "False"){
              Left();
              delay(700);
            } else {
              Stop();
```

```
                receiverData.mode = "Hand";
              }
            } else {
              Stop();
              receiverData.mode = "Hand";
            }
          } else {
            Stop();
            receiverData.mode = "Hand";
          }
        } else {
          Stop();
          receiverData.mode = "Hand";
        }
      } else {
        Stop();
        receiverData.mode = "Hand";
      }
    } else {
      Stop();
      receiverData.mode = "Hand";
    }
  } else {
    Stop();
    receiverData.mode = "Hand";
  }
 }
}
```

ДОДАТОК В

Лістинг програмного коду модуля автоматизації

```
#include "WiFi.h"
#include "esp_now.h"
#include "esp_camera.h"
#include <Demining_System_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"
#include "SPI.h"
#include "TFT_eSPI.h"
#include "TJpg_Decoder.h"

#define 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

#define EI_CAMERA_RAW_FRAME_BUFFER_COLS          320
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS          240
#define EI_CAMERA_FRAME_BYTE_SIZE                3

static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
static bool is_initialised = false;
uint8_t *snapshot_buf; //points to the output of the capture

static camera_config_t camera_config = {
    .pin_pwdn = PWDN_GPIO_NUM,
    .pin_reset = RESET_GPIO_NUM,
    .pin_xclk = XCLK_GPIO_NUM,
    .pin_sscb_sda = SIOD_GPIO_NUM,
    .pin_sscb_scl = SIOC_GPIO_NUM,

    .pin_d7 = Y9_GPIO_NUM,
    .pin_d6 = Y8_GPIO_NUM,
    .pin_d5 = Y7_GPIO_NUM,
    .pin_d4 = Y6_GPIO_NUM,
    .pin_d3 = Y5_GPIO_NUM,
    .pin_d2 = Y4_GPIO_NUM,
    .pin_d1 = Y3_GPIO_NUM,
    .pin_d0 = Y2_GPIO_NUM,
    .pin_vsync = VSYNC_GPIO_NUM,
    .pin_href = HREF_GPIO_NUM,
    .pin_pclk = PCLK_GPIO_NUM,
```

```
    //XCLK 20MHz or 10MHz for OV2640 double FPS (Experimental)
    .xclk_freq_hz = 20000000,
    .ledc_timer = LEDC_TIMER_0,
    .ledc_channel = LEDC_CHANNEL_0,

    .pixel_format = PIXFORMAT_JPEG, //YUV422,GRAYSCALE,RGB565,JPEG
    .frame_size = FRAMESIZE_QVGA,    //QQVGA-UXGA Do not use sizes above QVGA when not JPEG

    .jpeg_quality = 12, //0-63 lower number means higher quality
    .fb_count = 1,      //if more than one, i2s runs in continuous mode. Use only with JPEG
    .fb_location = CAMERA_FB_IN_PSRAM,
    .grab_mode = CAMERA_GRAB_WHEN_EMPTY,
};

/* Function definitions ------------------------------------------------------- */
bool ei_camera_init(void);
void ei_camera_deinit(void);
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf) ;

/**
* @brief      Arduino setup function
*/

const char *ssid = "Enter Wi-Fi name"; //connecting to Wi-Fi
const char *password = "Enter Wi-Fi password";

//mac-address of the receivering (Pult) board
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
```

```
//data sending structure
typedef struct struct_message {
  String leftObstacle;
  String frontObstacle;
  String rightObstacle;
  String mineDetected;
} struct_message;
struct_message dataCAM;

esp_now_peer_info_t peerInfo;

const int leftSensor = 12;
const int frontSensor = 13;
const int rightSensor = 14;

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, PultMacAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  pinMode (leftSensor, INPUT);
  pinMode (frontSensor, INPUT);
  pinMode (rightSensor, INPUT);

  while (!Serial);
```

```
    Serial.println("Edge Impulse Inferencing Demo");
    if (ei_camera_init() == false) {
        ei_printf("Failed to initialize Camera!\r\n");
    }
    else {
        ei_printf("Camera initialized\r\n");
    }

    ei_printf("\nStarting continious inference in 2 seconds...\n");
    ei_sleep(2000);
}

/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/

void loop() {
     // instead of wait_ms, we'll wait on the signal, this allows threads to cancel us...
    if (ei_sleep(5) != EI_IMPULSE_OK) {
        return;
    }

    snapshot_buf = (uint8_t*)malloc(EI_CAMERA_RAW_FRAME_BUFFER_COLS *
EI_CAMERA_RAW_FRAME_BUFFER_ROWS                                      *
EI_CAMERA_FRAME_BYTE_SIZE);

    // check if allocation was successful
    if(snapshot_buf == nullptr) {
```

```
        ei_printf("ERR: Failed to allocate snapshot buffer!\n");
        return;
    }

    ei::signal_t signal;
    signal.total_length = EI_CLASSIFIER_INPUT_WIDTH * EI_CLASSIFIER_INPUT_HEIGHT;
    signal.get_data = &ei_camera_get_data;

    if (ei_camera_capture((size_t)EI_CLASSIFIER_INPUT_WIDTH, (size_t)EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf) == false) {
        ei_printf("Failed to capture image\r\n");
        free(snapshot_buf);
        return;
    }

    // Run the classifier
    ei_impulse_result_t result = { 0 };

    EI_IMPULSE_ERROR err = run_classifier(&signal, &result, debug_nn);
    if (err != EI_IMPULSE_OK) {
        ei_printf("ERR: Failed to run classifier (%d)\n", err);
        return;
    }

    // print the predictions
    ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",
                result.timing.dsp, result.timing.classification, result.timing.anomaly);

  #if EI_CLASSIFIER_OBJECT_DETECTION == 1
```

```
    dataCAM.mineDetected = "True";
    bool bb_found = result.bounding_boxes[0].value > 0;
    for (size_t ix = 0; ix < result.bounding_boxes_count; ix++) {
        auto bb = result.bounding_boxes[ix];
        if (bb.value == 0) {
            continue;
        }
        ei_printf("    %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\n", bb.label, bb.value,
bb.x, bb.y, bb.width, bb.height);
    }
    if (!bb_found) {
        ei_printf("    No objects found\n");
    }
  #else
    dataCAM.mineDetected = "False";
    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        ei_printf("    %s: %.5f\n", result.classification[ix].label,
                                result.classification[ix].value);
    }
  #endif

  #if EI_CLASSIFIER_HAS_ANOMALY == 1
        ei_printf("    anomaly score: %.3f\n", result.anomaly);
  #endif
  free(snapshot_buf);

  delay(100);
  if(digitalRead(12) == LOW)  {
    dataCAM.leftObstacle = "True";
  } else{
```

```
    dataCAM.leftObstacle = "False";
  }
  if(digitalRead(13) == LOW)  {
    dataCAM.frontObstacle = "True";
  } else{
    dataCAM.frontObstacle = "False";
  }
  if(digitalRead(14) == LOW)  {
    dataCAM.rightObstacle = "True";
  } else{
    dataCAM.rightObstacle = "False";
  }
}

**
 * @brief   Setup image sensor & start streaming
 *
 * @retval  false if initialisation failed
 */
bool ei_camera_init(void) {

    if (is_initialised) return true;

    //initialize the camera
    esp_err_t err = esp_camera_init(&camera_config);
    if (err != ESP_OK) {
      Serial.printf("Camera init failed with error 0x%x\n", err);
      return false;
    }
```

```
    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, 0); // lower the saturation
    }

#if defined(CAMERA_MODEL_M5STACK_WIDE)
    s->set_vflip(s, 1);
    s->set_hmirror(s, 1);
#elif defined(CAMERA_MODEL_ESP_EYE)
    s->set_vflip(s, 1);
    s->set_hmirror(s, 1);
    s->set_awb_gain(s, 1);
#endif

    is_initialised = true;
    return true;
}

/**
 * @brief      Stop streaming of sensor data
 */
void ei_camera_deinit(void) {

    //deinitialize the camera
    esp_err_t err = esp_camera_deinit();

    if (err != ESP_OK)
```

```
    {
        ei_printf("Camera deinit failed\n");
        return;
    }

    is_initialised = false;
    return;
}


/**
 * @brief      Capture, rescale and crop image
 *
 * @param[in]  img_width     width of output image
 * @param[in]  img_height    height of output image
 * @param[in]  out_buf       pointer to store output image, NULL may be used
 *                           if ei_camera_frame_buffer is to be used for capture and resize/cropping.
 *
 * @retval     false if not initialised, image captured, rescaled or cropped failed
 *
 */
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf) {
    bool do_resize = false;

    if (!is_initialised) {
        ei_printf("ERR: Camera is not initialized\r\n");
        return false;
    }

    camera_fb_t *fb = esp_camera_fb_get();
```

```
 if (!fb) {
    ei_printf("Camera capture failed\n");
    return false;
 }

bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT_JPEG, snapshot_buf);

esp_camera_fb_return(fb);

if(!converted){
   ei_printf("Conversion failed\n");
   return false;
}

 if ((img_width != EI_CAMERA_RAW_FRAME_BUFFER_COLS)
    || (img_height != EI_CAMERA_RAW_FRAME_BUFFER_ROWS)) {
    do_resize = true;
 }

 if (do_resize) {
    ei::image::processing::crop_and_interpolate_rgb888(
    out_buf,
    EI_CAMERA_RAW_FRAME_BUFFER_COLS,
    EI_CAMERA_RAW_FRAME_BUFFER_ROWS,
    out_buf,
    img_width,
    img_height);
 }
```

```
    return true;
}

static int ei_camera_get_data(size_t offset, size_t length, float *out_ptr)
{
    // we already have a RGB888 buffer, so recalculate offset into pixel index
    size_t pixel_ix = offset * 3;
    size_t pixels_left = length;
    size_t out_ptr_ix = 0;

    while (pixels_left != 0) {
        out_ptr[out_ptr_ix] = (snapshot_buf[pixel_ix + 2] << 16) + (snapshot_buf[pixel_ix
+ 1] << 8) + snapshot_buf[pixel_ix];

        // go to the next pixel
        out_ptr_ix++;
        pixel_ix+=3;
        pixels_left--;
    }
    // and done!
    return 0;
}
#if     !defined(EI_CLASSIFIER_SENSOR)     ||     EI_CLASSIFIER_SENSOR     !=
EI_CLASSIFIER_SENSOR_CAMERA
#error "Invalid model for current sensor"
#endif
```

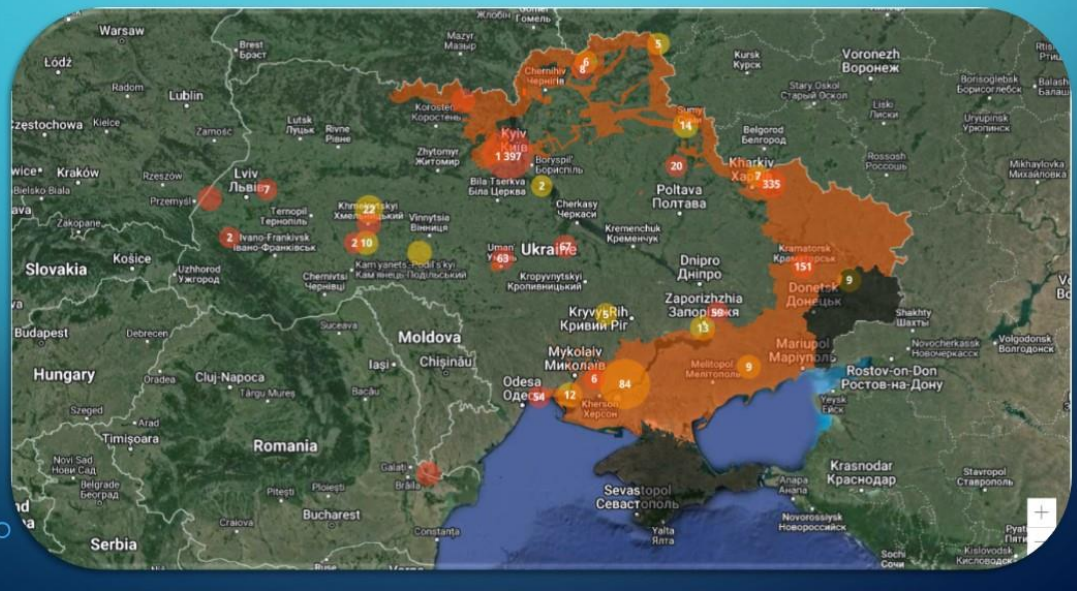
ДОДАТОК Г

Демонстраційний матеріал

РОЗРОБЛЕННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ДЛЯ ДИСТАНЦІЙНОГО КЕРУВАННЯ РОБОТОМ-РОЗМІНУВАЛЬНИКОМ

Виконав: Курбанов Н. Р.
Ст. гр. АКТАКІТ-20-2
Керівник: Хрустальова С. В.
Кафкдра: КІТАР

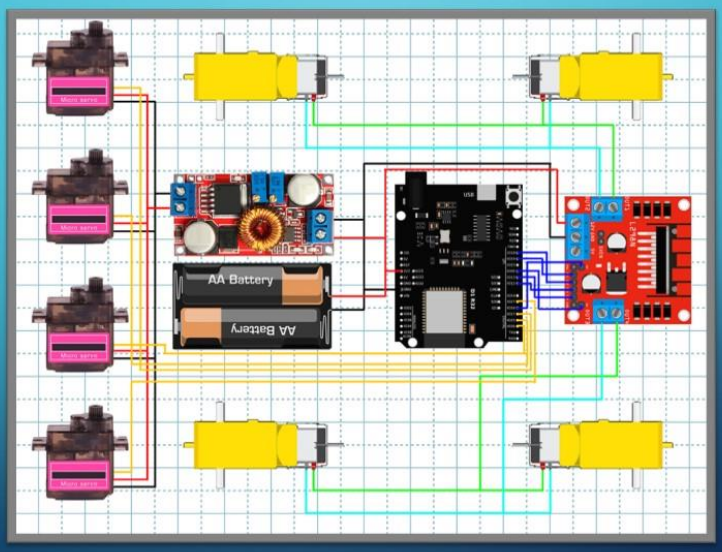
АКТУАЛЬНІСТЬ РОБОТИ



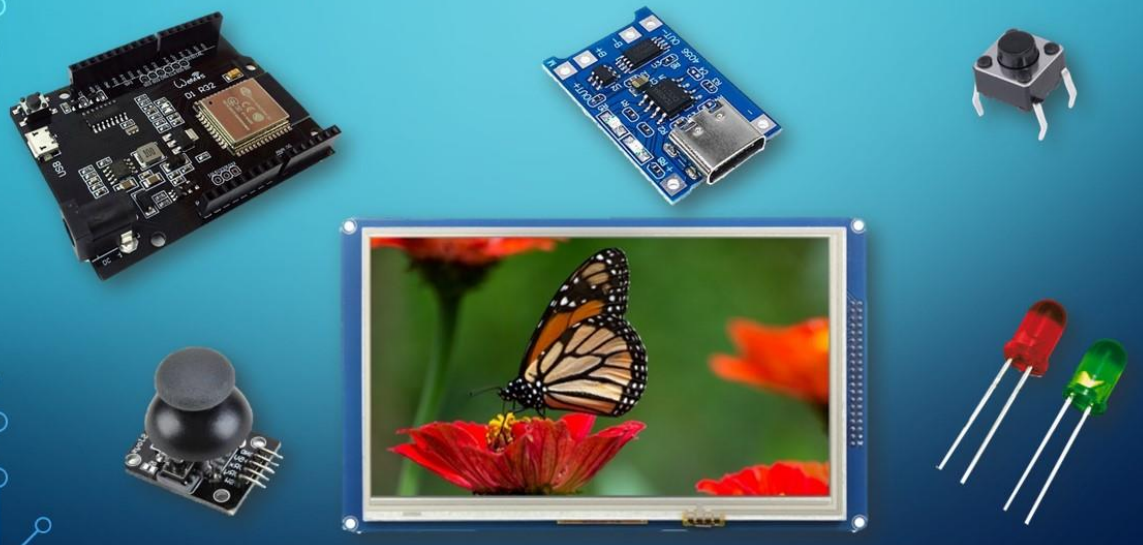

ОСНОВНІ ЕЛЕМЕНТИ СИСТЕМИ
ESP
NOW
Mine Discovered
Auto
Hand
Auto
Hand


КОМПОНЕНТИ
РОБОТА-РОЗМІНУВАЛЬНИКА
MG90S
Micro servo
D1 R32
USB

СХЕМА ПІДКЛЮЧЕННЯ КОМПОНЕНТІВ РОБОТА-РОЗМІНУВАЛЬНИКА



КОМПОНЕНТИ ПУЛЬТА КЕРУВАННЯ



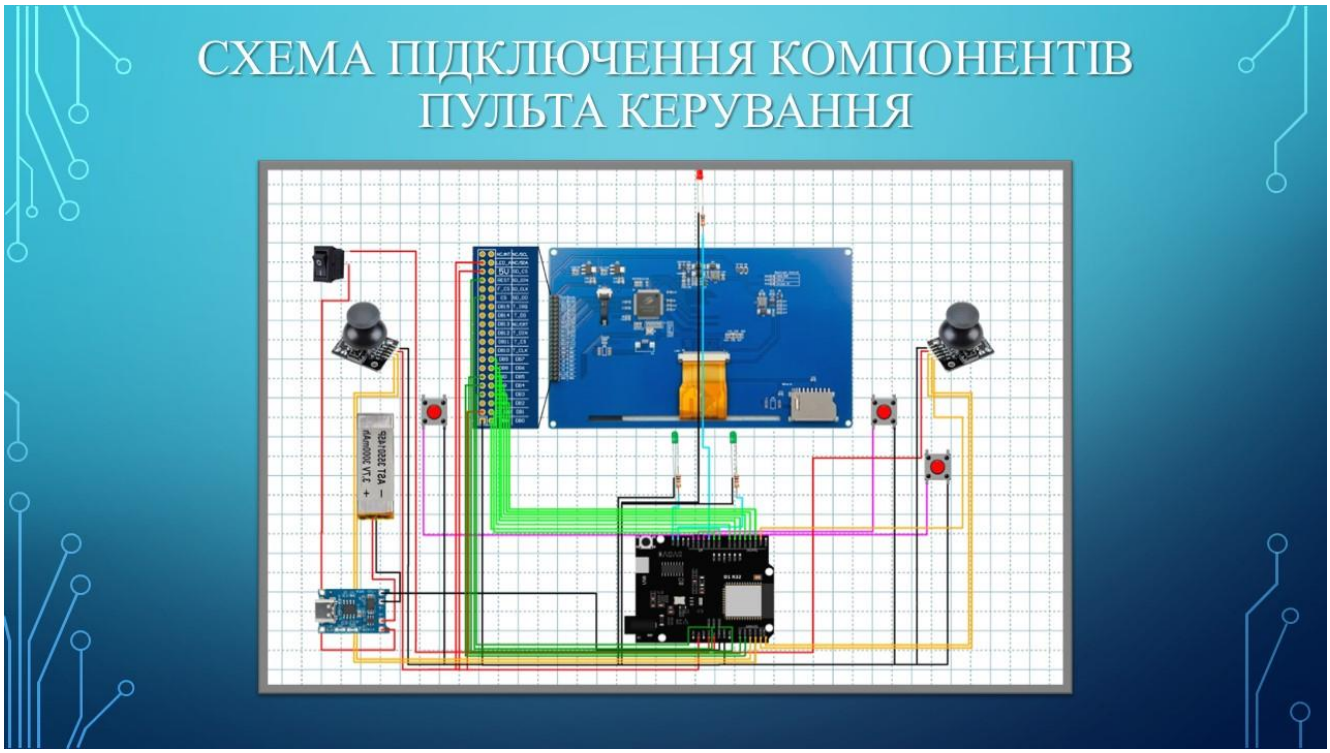
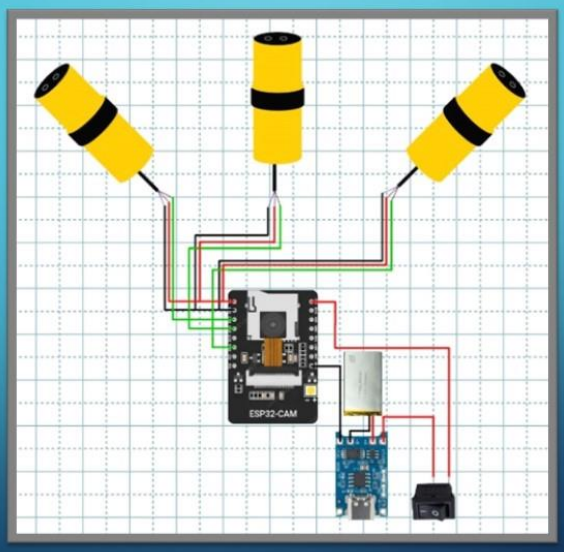


СХЕМА ПІДКЛЮЧЕННЯ КОМПОНЕНТІВ
ПУЛЬТА КЕРУВАННЯ


КОМПОНЕНТИ
МОДУЛЯ АВТОМАТИЗАЦІЇ
ESP32-CAM

СХЕМА ПІДКЛЮЧЕННЯ КОМПОНЕНТІВ МОДУЛЯ АВТОМАТИЗАЦІЇ



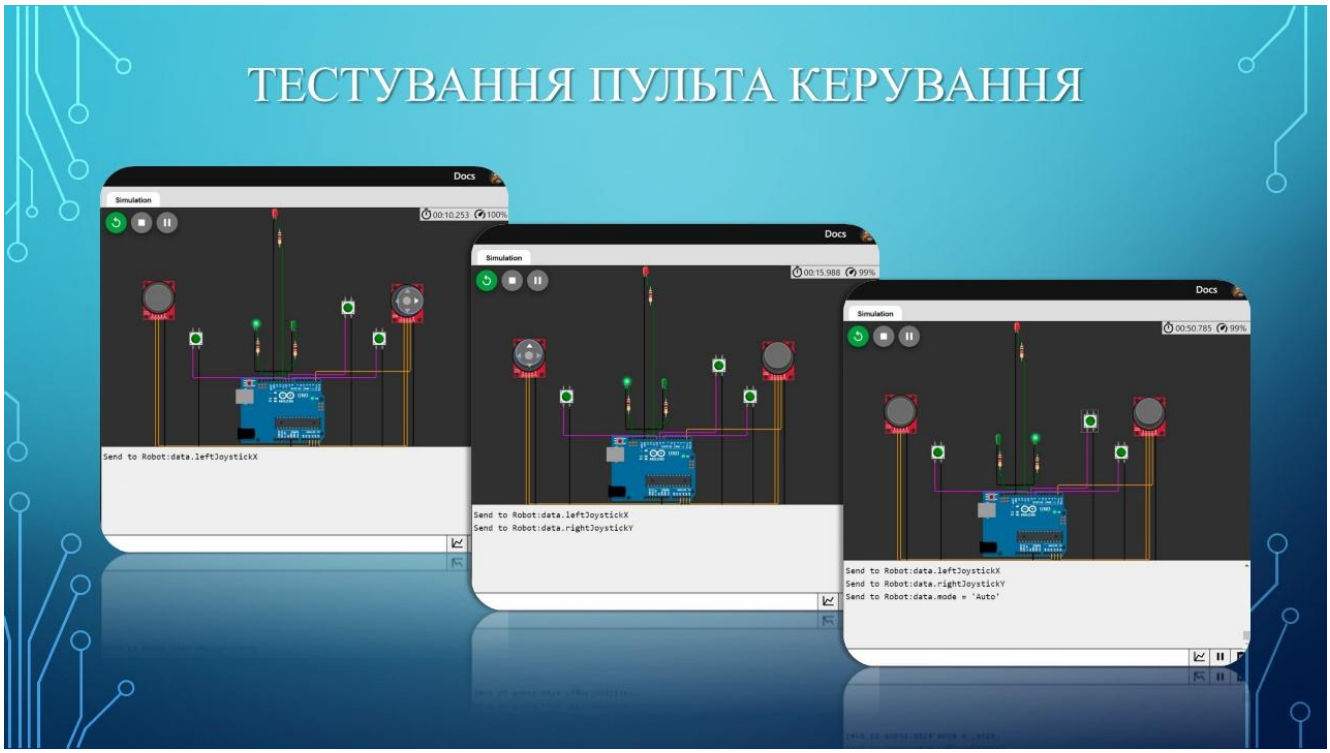
ЗАСОБИ РОЗРОБКИ ТА ТЕСТУВАННЯ

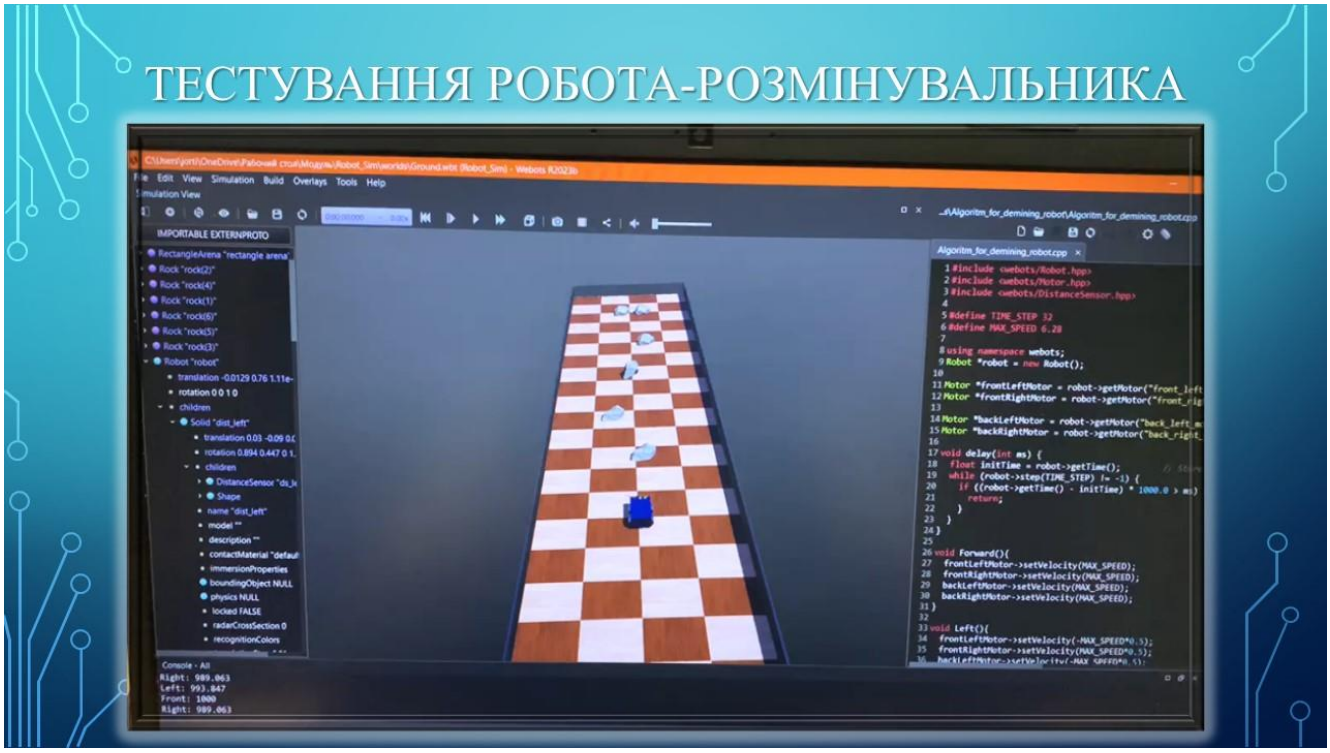
WOKWI
Online Arduino Simulator

ARDUINO

EDGE IMPULSE

Webots
robot simulation


ТЕСТУВАННЯ ПУЛЬТА КЕРУВАННЯ
Docs
Simulation
Send to Robot:data.leftJoystickX
Send to Robot:data.rightJoystickY
Send to Robot:data.mode = 'Auto'


ТЕСТУВАННЯ РОБОТА-РОЗМІНУВАЛЬНИКА

ТЕСТУВАННЯ МОДУЛЯ АВТОМАТИЗАЦІЇ

ДЯКУЮ ЗА УВАГУ!

№ документа	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 505400.008 ПЗ	Пояснювальна записка	А4, 74 с.
		Додаткові матеріали	
2		Лістинг програмного коду	А4, 5 с.
		пульта керування	
3		Лістинг програмного коду	А4, 29 с.
		системи керування	
		роботом-розмінувальником	
4		Лістинг програмного коду	А4, 12 с.
		модуля автоматизації	
5		Демонстраційний матеріал	*.pptx, 7 с.
		у вигляді презентації	

					ГЮИК. 505400.008 ВД			
Зм.	Арк.	№ докум.	Підп.	Дата				
Розроб.		Курбанов Н.Р.			Розроблення системи автоматизації для дистанційного керування роботом-розмінувальником	Літ.	Аркуш	Аркушів
Перевір.		Хрустальова С.В.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.								
Затв.		Невлюдов І.Ш.						