DOKUMENTACIJA IMPLEMENTACIJE

Simulacija joystick kontrolera za PC

Emrah Žunić, Mirnes Fehrić, Kerim Šikalo, Faris Aljić

15. juni 2025.

Sadržaj

1 Pregled arhitekture sistema

Sistem za simulaciju joystick kontrolera implementiran je kao distribuirana aplikacija koja se sastoji od tri glavne komponente:

- **Raspberry Pi Pico W** - hardverski kontroler sa analognim joystick-om i digitalnim tasterima
- **MQTT broker** - centralni posrednik za komunikaciju (broker.hivemq.com)
- **PC aplikacija** - Python skripta koja implementira virtuelni joystick (vJoy)

1.1 Komunikacijski tok podataka

1. Pico W čita analogne vrijednosti sa joystick osa (GP28, GP27) i digitalna stanja tastera (GP0-GP3)
2. Podaci se formatiraju i šalju preko WiFi na MQTT broker
3. PC aplikacija prima MQTT poruke i obrađuje podatke
4. Obrađeni podaci se proslijeđuju vJoy driver-u
5. vJoy kreira virtuelni joystick dostupan svim aplikacijama

2 Implementacija na Raspberry Pi Pico W

2.1 Hardverska konfiguracija

GPIO mapiranje pinova

- **GP28** - X osa joystick-a (analogni ulaz, ADC kanal)
- **GP27** - Y osa joystick-a (analogni ulaz, ADC kanal)
- **GP0** - Lijevi žmigavac (digitalni ulaz, pull-up otpor)
- **GP1** - Desni žmigavac (digitalni ulaz, pull-up otpor)
- **GP2** - Sirena (digitalni ulaz, pull-up otpor)
- **GP3** - Brisač (digitalni ulaz, pull-up otpor)

2.2 Inicijalizacija hardverskih komponenti

```
# Inicijalizacija analognih pinova za joystick
x_axis = ADC(Pin(28))
y_axis = ADC(Pin(27))

# Inicijalizacija digitalnih pinova za tastere (pull-up otpor)
button_lijevo = Pin(0, Pin.IN, Pin.PULL_UP)    # GP0
```

```
button_desno = Pin(1, Pin.IN, Pin.PULL_UP)      # GP1
button_sirena = Pin(2, Pin.IN, Pin.PULL_UP)     # GP2
button_brisac = Pin(3, Pin.IN, Pin.PULL_UP)     # GP3
```

Listing 1: Setup GPIO pinova

2.3 MQTT konfiguracija

```
SSID = "Zhunky"
PASSWORD = "emre12345"
MQTT_BROKER = "broker.hivemq.com"
MQTT_PORT = 1883
CLIENT_ID = "joystick-client"

# MQTT teme
MQTT_TOPIC_X = b"tema/x"
MQTT_TOPIC_Y = b"tema/y"
MQTT_TOPIC_LIJEVO = b"tema/lijevo"
MQTT_TOPIC_DESNO = b"tema/desno"
MQTT_TOPIC_SIRENA = b"tema/sirena"
MQTT_TOPIC_BRISAC = b"tema/brisac"
```

Listing 2: MQTT postavke

2.4 WiFi konekcija

```
def do_connect():
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        print("Povezivanje na WiFi mrezu...")
        sta_if.active(True)
        sta_if.connect(SSID, PASSWORD)
        while not sta_if.isconnected():
            time.sleep(0.5)
    print("Uspjesno povezan:", sta_if.ifconfig())
```

Listing 3: WiFi konekcija funkcija

2.5 Toggle logika za tastere

Implementiran je toggle mehanizam koji mijenja stanje na svaki pritisak tastera:

```
# Varijable za pracenje stanja tastera
last_button_state = {
    "lijevo": 1,    # 1 = nije pritisnut (pull-up)
    "desno": 1,
    "sirena": 1,
    "brisac": 1
}

# Varijable za toggle stanje (ukljuceno/iskljuceno)
toggle_state = {
    "lijevo": False,
    "desno": False,
    "sirena": False,
```

```
    "brisac": False
}

def check_button_press(button, button_name, topic, client):
    """Provjeri je li taster pritisnut i posalji MQTT poruku"""
    current_state = button.value()

    # Detektuj pritisak tastera (promjena sa 1 na 0 zbog pull-up)
    if last_button_state[button_name] == 1 and current_state == 0:
        # Taster je pritisnut - promijeni toggle stanje
        toggle_state[button_name] = not toggle_state[button_name]

        # Posalji MQTT poruku
        message = "1" if toggle_state[button_name] else "0"
        client.publish(topic, message)

        status = "UKLJUCENO" if toggle_state[button_name] else "
    ISKLJUCENO"
        print(f"{button_name.upper()}: {status}")

        # Kratka pauza da se izbjegne bounce
        time.sleep(0.05)

    # Azuriraj poslednje stanje
    last_button_state[button_name] = current_state
```

Listing 4: Toggle implementacija

2.6 Glavna petlja

```
def main():
    do_connect()

    client = MQTTClient(CLIENT_ID, MQTT_BROKER, port=MQTT_PORT)
    client.connect()
    print("MQTT klijent povezan.")
    print("Joystick i tasteri aktivni...")

    while True:
        # Citaj joystick osi
        x_val = x_axis.read_u16()  # 0-65535
        y_val = y_axis.read_u16()  # 0-65535

        # Posalji joystick podatke
        client.publish(MQTT_TOPIC_X, str(x_val))
        client.publish(MQTT_TOPIC_Y, str(y_val))

        # Provjeri tastere
        check_button_press(button_lijevo, "lijevo", MQTT_TOPIC_LIJEVO,
    client)
        check_button_press(button_desno, "desno", MQTT_TOPIC_DESNO,
    client)
        check_button_press(button_sirena, "sirena", MQTT_TOPIC_SIRENA,
    client)
        check_button_press(button_brisac, "brisac", MQTT_TOPIC_BRISAC,
    client)

```

```
24          print("Joystick - X:", x_val, "Y:", y_val)
25
26          time.sleep(0.2)  # 5Hz update rate
```

Listing 5: Main loop za prikupljanje podataka

3 Implementacija PC aplikacije

3.1 MQTT i vJoy konfiguracija

```
1  import paho.mqtt.client as mqtt
2  import pyvjoy
3
4  # MQTT postavke
5  MQTT_BROKER = "broker.hivemq.com"
6  MQTT_PORT = 1883
7  MQTT_TOPIC_X = "tema/x"
8  MQTT_TOPIC_Y = "tema/y"
9  MQTT_TOPIC_LIJEVO = "tema/lijevo"
10 MQTT_TOPIC_DESNO = "tema/desno"
11 MQTT_TOPIC_SIRENA = "tema/sirena"
12 MQTT_TOPIC_BRISAC = "tema/brisac"
13
14 # vJoy setup
15 vj = pyvjoy.VJoyDevice(1)
16 MAX_AXIS_VALUE = 0x8000  # 32768
17 CENTER_AXIS_VALUE = MAX_AXIS_VALUE // 2  # 16384
18
19 # Deadzone postavke
20 DEADZONE_PERCENT = 0.10
21 INPUT_MIN = 0
22 INPUT_MAX = 65535
23 INPUT_CENTER = INPUT_MAX // 2  # 32767
```

Listing 6: PC aplikacija postavke

3.2 Algoritam za skaliranje joystick vrijednosti

Ključna funkcija koja pretvara ADC vrijednosti (0-65535) u vJoy format (0-32768) sa deadzonom:

```
1  def scale_input_to_vjoy_axis(input_value: int, input_min: int = 0,
       input_max: int = 65535) -> int:
2      """
3      Pretvori int vrijednost iz raspona [0..65535] u int raspon
       [0..32768].
4      Ukljucuje deadzone od 10% oko srednje vrijednosti.
5      """
6      # Ogranici ulaznu vrijednost na valjan raspon
7      if input_value < input_min:
8          input_value = input_min
9      if input_value > input_max:
10         input_value = input_max
11
12     # Izracunaj deadzone granice
```

```
    deadzone_range = (input_max - input_min) * DEADZONE_PERCENT / 2
    deadzone_min = INPUT_CENTER - deadzone_range
    deadzone_max = INPUT_CENTER + deadzone_range

    # Provjeri je li vrijednost u deadzone
    if deadzone_min <= input_value <= deadzone_max:
        return CENTER_AXIS_VALUE  # Vrati srednju vrijednost za vJoy

    # Skaliraj vrijednost izvan deadzone
    if input_value < deadzone_min:
        # Donji dio - skaliraj od 0 do CENTER_AXIS_VALUE
        span_input = deadzone_min - input_min
        span_output = CENTER_AXIS_VALUE
        normalized = (input_value - input_min) / span_input
        result = int(normalized * span_output)
    else:  # input_value > deadzone_max
        # Gornji dio - skaliraj od CENTER_AXIS_VALUE do MAX_AXIS_VALUE
        span_input = input_max - deadzone_max
        span_output = MAX_AXIS_VALUE - CENTER_AXIS_VALUE
        normalized = (input_value - deadzone_max) / span_input
        result = int(CENTER_AXIS_VALUE + (normalized * span_output))

    return result
```

Listing 7: Skaliranje sa deadzone algoritmom

3.3 MQTT message handling

```
def on_message(client, userdata, msg):
    """
    Callback funkcija za obradu svih MQTT poruka
    """
    try:
        payload_str = msg.payload.decode('utf-8')

        # Obradi poruke za osi (X i Y)
        if msg.topic in [MQTT_TOPIC_X, MQTT_TOPIC_Y]:
            axis_value = int(payload_str)
            vjoy_value = scale_input_to_vjoy_axis(axis_value)

            if msg.topic == MQTT_TOPIC_X:
                vj.set_axis(pyvjoy.HID_USAGE_X, vjoy_value)
                print(f"X os: {axis_value} -> {vjoy_value}")
            elif msg.topic == MQTT_TOPIC_Y:
                vj.set_axis(pyvjoy.HID_USAGE_Y, vjoy_value)
                print(f"Y os: {axis_value} -> {vjoy_value}")

        # Obradi poruke za tastere (tastovi 1-4)
        elif msg.topic in [MQTT_TOPIC_LIJEVO, MQTT_TOPIC_DESNO,
    MQTT_TOPIC_SIRENA, MQTT_TOPIC_BRISAC]:
            button_value = int(payload_str)
            button_pressed = (button_value == 1)

            if msg.topic == MQTT_TOPIC_LIJEVO:
                vj.set_button(1, button_pressed)
                print(f"Lijevi zmigavac (button 1): {'UKLJUCEN' if
    button_pressed else 'ISKLJUCEN'}")
```

```
            elif msg.topic == MQTT_TOPIC_DESNO:
                vj.set_button(2, button_pressed)
                print(f"Desni zmigavac (button 2): {'UKLJUCEN' if button_pressed else 'ISKLJUCEN'}")
            elif msg.topic == MQTT_TOPIC_SIRENA:
                vj.set_button(3, button_pressed)
                print(f"Sirena (button 3): {'UKLJUCENA' if button_pressed else 'ISKLJUCENA'}")
            elif msg.topic == MQTT_TOPIC_BRISAC:
                vj.set_button(4, button_pressed)
                print(f"Brisac (button 4): {'UKLJUCEN' if button_pressed else 'ISKLJUCEN'}")

        except Exception as e:
            print(f"Greska pri parsiranju/podacima: {e}")
```

Listing 8: Obrada MQTT poruka

3.4 MQTT klijent setup

```
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Spojen na MQTT broker.")
        print(f"Pretplata na teme: {MQTT_TOPIC_X}, {MQTT_TOPIC_Y}")
        print(f"Pretplata na teme za tastere: {MQTT_TOPIC_LIJEVO}, {MQTT_TOPIC_DESNO}, {MQTT_TOPIC_SIRENA}, {MQTT_TOPIC_BRISAC}")

        # Pretplati se na sve teme
        client.subscribe(MQTT_TOPIC_X)
        client.subscribe(MQTT_TOPIC_Y)
        client.subscribe(MQTT_TOPIC_LIJEVO)
        client.subscribe(MQTT_TOPIC_DESNO)
        client.subscribe(MQTT_TOPIC_SIRENA)
        client.subscribe(MQTT_TOPIC_BRISAC)

        # Postavi pocetne vrijednosti na sredinu za osi
        vj.set_axis(pyvjoy.HID_USAGE_X, CENTER_AXIS_VALUE)
        vj.set_axis(pyvjoy.HID_USAGE_Y, CENTER_AXIS_VALUE)
        print(f"vJoy osi postavljene na srednju vrijednost ({CENTER_AXIS_VALUE})")

        # Postavi sve tastove kao otpustene
        for i in range(1, 5):
            vj.set_button(i, False)
        print("vJoy tastovi 1-4 postavljeni na otpusteno stanje")

    else:
        print(f"Neuspjesno spajanje, kod greske: {rc}")

def main():
    # Inicijaliziraj MQTT klijenta
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message

    # Spoji se na broker
    client.connect(MQTT_BROKER, MQTT_PORT, keepalive=60)
```

```
36
37      # Pokreni loop koji ce u pozadini slusati poruke
38      client.loop_start()
39
40      try:
41          print("MQTT klijent pokrenut. Ceka se na poruke...")
42          print(f"Deadzone: {DEADZONE_PERCENT*100}% oko srednje
    vrijednosti ({INPUT_CENTER})")
43          print("Mapiranje tastera:")
44          print("  tema/lijevo -> vJoy button 1 (lijevi zmigavac)")
45          print("  tema/desno -> vJoy button 2 (desni zmigavac)")
46          print("  tema/sirena -> vJoy button 3 (sirena)")
47          print("  tema/brisac -> vJoy button 4 (brisac)")
48          while True:
49              time.sleep(1)
50      except KeyboardInterrupt:
51          print("Prekid rada skripte. Zatvaram konekciju i izlazim...")
52      finally:
53          client.loop_stop()
54          client.disconnect()
```

Listing 9: MQTT klijent inicijalizacija

4 Tehnički detalji

4.1 Deadzone implementacija

Deadzone od 10% oko srednje vrijednosti (32767) eliminiše neželjeno kretanje kada je joystick u neutralnoj poziciji. Algoritam dijeli input raspon na tri segmenta:

1. **Donji segment**: $[0, center - deadzone] \rightarrow [0, vJoy_{center}]$
2. **Deadzone**: $[center - deadzone, center + deadzone] \rightarrow vJoy_{center}$
3. **Gornji segment**: $[center + deadzone, 65535] \rightarrow [vJoy_{center}, vJoy_{max}]$

4.2 MQTT protokol

Koristi se jednostavan string protokol:

- **Osi**: numerička vrijednost kao string (npr. ”32767”)
- **Tasteri**: 0 za isključeno, 1 za uključeno

4.3 vJoy integracija

vJoy biblioteka omogućava kreiranje virtuelnog joystick uređaja sa:

- 2 analogne ose (X, Y)
- 4 digitalna dugmeta
- HID protokol kompatibilnost sa Windows aplikacijama

5 Performanse sistema

5.1 Update rate

Sistem koristi 5Hz update rate (200ms interval) što predstavlja balans između brzina odziva sistema i mrežnih opterećenja.

5.2 Kašnjenje

Tipično ukupno kašnjene sistema: 15-60ms, što se sastoji od:

- ADC uzorkovanje: $\sim 1\mu$s
- WiFi prijenos: 10-50ms
- MQTT brokerska obrada: 1-5ms
- PC aplikacijska obrada: $\sim$1ms
- vJoy driver: $\sim$1ms

6 Error handling

Implementiran je osnovni error handling:

- Try-catch blokovi u MQTT message parsing
- Kontrolisano gašenje sa KeyboardInterrupt
- Cleanup konekcija u finally bloku
- WiFi rekonekcija u while petlji

7 Zaključak

Implementirani sistem demonstrira uspješnu integraciju embedded hardware-a (Pico W) sa desktop aplikacijom preko MQTT protokola. Ključne komponente uključuju deadzone algoritam za stabilnost joystick-a, toggle logiku za tastere, i pouzdanu MQTT komunikaciju.