記事元
https://riragon.com/04-network-remote-rccar-test/

raspberry Pi5送信側 IP:192.168.10.202 send.py

```
import socket
from inputs import get_gamepad
import sys
import threading
import time
import select

# 送信先のIPとポート(受信側)
server_ip =192.168.10.201"
server_port = 5000

# 接続確認信号を受信するためのIPとポート(送信側)
local_ip = "0.0.0.0"
local_port = 6000  # 受信側からの確認信号を受け取るためのポート

# ソケットの設定(UDP)
send_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
recv_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
recv_sock.bind((local_ip, local_port))
recv_sock.setblocking(False)  # ノンブロッキングモードに設定

print("接続待機中...")

# ゲームパッドの取得
try:
    devices = get_gamepad()
    gamepad = devices[0] if devices else None
    if gamepad:
        print(f"ゲームパッドが接続されました")
    else:
        raise IndexError
except IndexError:
    print("エラー: ゲームパッドが接続されていません。プログラムを終了します。")
    sys.exit(1)

print(f"{server_ip}:{server_port} にデータを送信します。")
print("操作信号を送信します。")

# 接続確認用の変数とロック
ack_lock = threading.Lock()
last_ack_time = time.time()
ack_timeout = 0.5  # 500ms
force_stop = False
b_button_pressed_time = None
connection_lost = False

def receive_ack():
    global last_ack_time, force_stop, connection_lost
    while True:
        try:
            readable, _, _ = select.select([recv_sock], [], [], 0)
            if recv_sock in readable:
                data, addr = recv_sock.recvfrom(1024)
                message = data.decode('utf-8')
                if message == "ACK":
                    with ack_lock:
                        last_ack_time = time.time()
                    if connection_lost:
                        print("接続が再開されました。")
                        connection_lost = False
        except Exception:
```

```
            pass  # ノンブロッキングなので、データがない場合はパス

        # ACKタイムアウトのチェック
        with ack_lock:
            if time.time() - last_ack_time > ack_timeout:
                if not connection_lost:
                    print("接続確認信号が途切れました。操作信号の送信を停止します。")
                    connection_lost = True
                    force_stop = True  # 接続途絶時に操作信号の送信を停止

        time.sleep(0.1)  # 100msごとにチェック

def send_keep_alive():
    while True:
        try:
            send_sock.sendto("KEEP_ALIVE".encode('utf-8'), (server_ip, server_port))
        except Exception as e:
            print(f"KEEP_ALIVE送信エラー: {e}")
        time.sleep(0.1)  # 100msごとに送信

# ACK受信スレッドの開始
ack_thread = threading.Thread(target=receive_ack, daemon=True)
ack_thread.start()

# KEEP_ALIVE送信スレッドの開始
keep_alive_thread = threading.Thread(target=send_keep_alive, daemon=True)
keep_alive_thread.start()

try:
    while True:
        events = get_gamepad()
        for event in events:
            # Bボタンの長押し検出（BTN_EASTに変更）
            if event.ev_type == "Key" and event.code == "BTN_EAST":
                if event.state == 1:
                    b_button_pressed_time = time.time()
                    print("Bボタンが押されました。")
                elif event.state == 0 and b_button_pressed_time:
                    pressed_duration = time.time() - b_button_pressed_time
                    b_button_pressed_time = None
                    print(f"Bボタンが離されました。押下時間: {pressed_duration:.2f}秒")
                    if pressed_duration >= 1.0:
                        # Bボタン長押しで操作信号の送信を再開
                        force_stop = False
                        with ack_lock:
                            last_ack_time = time.time()
                        print("Bボタンが1秒以上押されました。操作信号の送信を再開します。")

            if force_stop:
                continue  # 操作信号の送信を停止

            if event.ev_type == "Absolute" and event.code in ["ABS_X", "ABS_RY"]:
                axis_value = event.state / 32767  # -1 から 1 の範囲に正規化
                message = f"{event.code}:{axis_value:.3f}"
                try:
                    send_sock.sendto(message.encode('utf-8'), (server_ip, server_port))
                    # 操作信号の表示
                    print(f"送信: {message}")
                except Exception as e:
                    print(f"操作信号送信エラー: {e}")

except KeyboardInterrupt:
    print("\nプログラムを終了します。")
except Exception as e:
    print(f"エラーが発生しました: {e}")
finally:
    send_sock.close()
    recv_sock.close()
    print("ソケットを閉じました。")
```

raspberry Pi4受信側 IP:192.168.10.201 receive.py

```
import socket
from gpiozero import PWMOutputDevice
from gpiozero.pins.pigpio import PiGPIOFactory
import time
import sys
import threading
import select

# 受信側のIPとポート
local_ip = "0.0.0.0"
local_port = 5000

# 送信側のIPとポート(接続確認信号の送信先)
client_ip = "192.168.10.202"
client_port = 6000  # 送信側がACKを受信するためのポート

# ソケットの設定
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((local_ip, local_port))
sock.setblocking(False)  # ノンブロッキングモードに設定

print(f"ポート {local_port} でデータを待機しています...")
print("接続待機中...")

# GPIOの設定
factory = PiGPIOFactory()

# サーボとESCの初期化
try:
    servo = PWMOutputDevice(12, pin_factory=factory, frequency=50)
    esc = PWMOutputDevice(13, pin_factory=factory, frequency=50)
except Exception as e:
    print(f"GPIOの初期化中にエラーが発生しました: {e}")
    sys.exit(1)

# 操作信号受信確認用の変数とロック
last_received_time = time.time()
control_loss = True  # 初期状態では接続待機中
client_ip_lock = threading.Lock()

def send_ack():
    while True:
        with client_ip_lock:
            current_client_ip = client_ip
        try:
            sock.sendto("ACK".encode('utf-8'), (current_client_ip, client_port))
        except Exception:
            pass  # ソケットエラー時はパス
        time.sleep(0.1)  # 100msごとに送信

def set_servo_angle(angle):
    min_pulse_width = 0.5 / 1000
    max_pulse_width = 2.4 / 1000
    frame_width = 1 / 50

    pulse_width = ((angle + 90) / 180) * (max_pulse_width - min_pulse_width) + min_pulse_width
    duty_cycle = pulse_width / frame_width
    servo.value = duty_cycle

def set_esc_speed(pulse_width_ms):
```

```
    pulse_width_s = pulse_width_ms / 1000
    frame_width_s = 1 / 50
    duty_cycle = pulse_width_s / frame_width_s
    esc.value = duty_cycle

def initialize_devices():
    set_servo_angle(0)
    set_esc_speed(1.5)
    time.sleep(1)

def shutdown_devices():
    set_servo_angle(0)
    set_esc_speed(1.5)
    time.sleep(1)
    servo.close()
    esc.close()

def main():
    global last_received_time, control_loss, client_ip

    # デバイスの初期化
    initialize_devices()

    # ACK送信スレッドの開始
    ack_thread = threading.Thread(target=send_ack, daemon=True)
    ack_thread.start()

    try:
        while True:
            readable, _, _ = select.select([sock], [], [], 0)
            if sock in readable:
                data, addr = sock.recvfrom(1024)
                message = data.decode('utf-8')

                # 送信元IPを更新(動的IPに対応)
                with client_ip_lock:
                    client_ip = addr[0]

                last_received_time = time.time()
                if control_loss:
                    control_loss = False
                    print("接続が再開されました。")

                if message == "KEEP_ALIVE":
                    continue  # 接続維持用メッセージなので処理をスキップ

                # データの処理
                if ":" in message:
                    code, value = message.split(":")
                    axis_value = float(value)
                    if code == "ABS_X":
                        angle = axis_value * 90
                        angle = max(min(angle, 90), -90)
                        set_servo_angle(angle)
                        print(f"受信: {message} -> サーボ角度を {angle:.1f} 度に設定")

                    elif code == "ABS_RY":
                        dead_zone = 0.05
                        if abs(axis_value) < dead_zone:
                            axis_value = 0

                        pulse_width = 1.5 + (axis_value * 0.1)
                        pulse_width = max(min(pulse_width, 1.6), 1.4)

                        set_esc_speed(pulse_width)
                        print(f"受信: {message} -> ESCパルス幅を {pulse_width:.2f} msに設定")

            # 接続確認信号が一定時間受信されなかった場合
            if time.time() - last_received_time > 0.5 and not control_loss:
```

```
                control_loss = True
                set_esc_speed(1.5)  # ESCパルスを1.5msに強制固定
                print("接続確認信号が途切れました。ESCパルスを1.5msに強制固定します。")
                print("接続待機中...")

            # 過度なCPU使用を防ぐための短い待機
            time.sleep(0.01)

    except KeyboardInterrupt:
        print("\nプログラムを終了します。")
    except Exception as e:
        print(f"エラーが発生しました: {e}")
    finally:
        shutdown_devices()
        sock.close()
        print("デバイスとソケットを安全な状態に設定しました。")

if __name__ == "__main__":
    main()
```