Laboratorium z przedmiotu Systemy wbudowane (SW)

Sprawozdanie

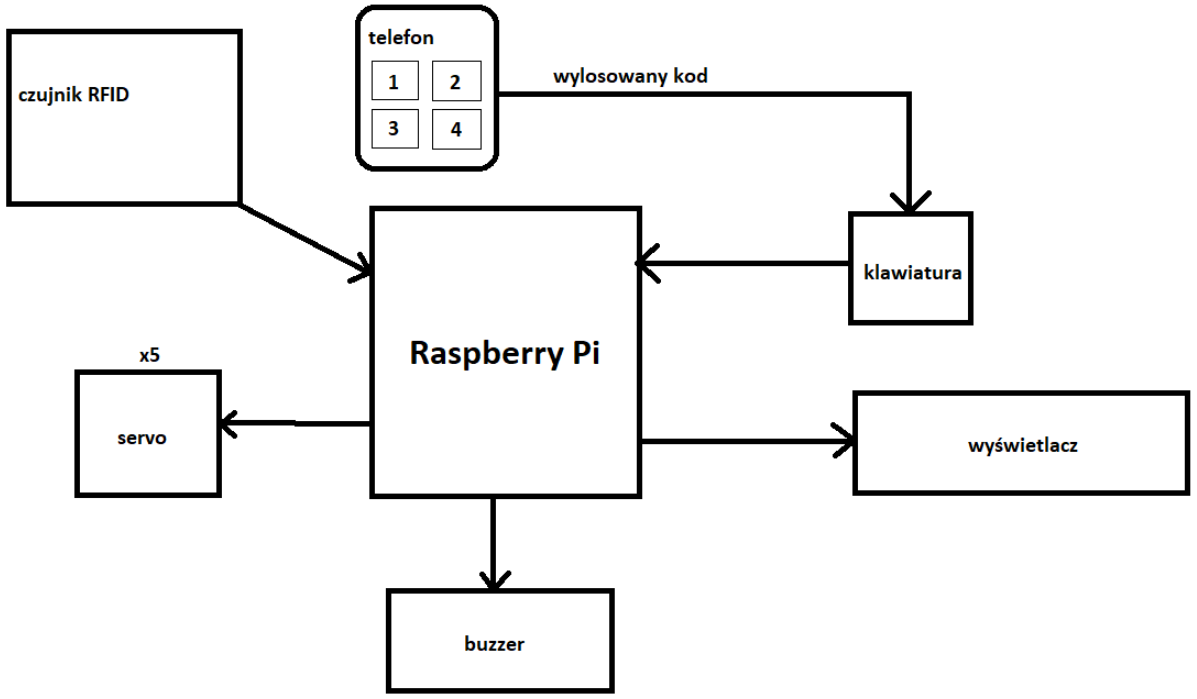
Nazwa projektu: Skrzynka-self z Minecrafta

Prowadzący: Ariel Antonowicz	*Autorzy:* *148182* *148081* *150251*	*Grupa dziekańska:*	*i5.1*
		Ocena:	

Cel projektu:

Stworzenie skrzynki na przedmioty zabezpieczone jednorazowo wygenerowanym hasłem z telefonu (na zasadzie blik). Łączymy się za pomocą Wi-Fi i wpisujemy hasło, które następnie otwiera odpowiednią skrytkę. Skrytki nie mogą być otwarte jednocześnie. Buzzer odpala się 3 sekundy przed zamknięciem i kiedy zostało wpisane nieprawidłowe hasło.

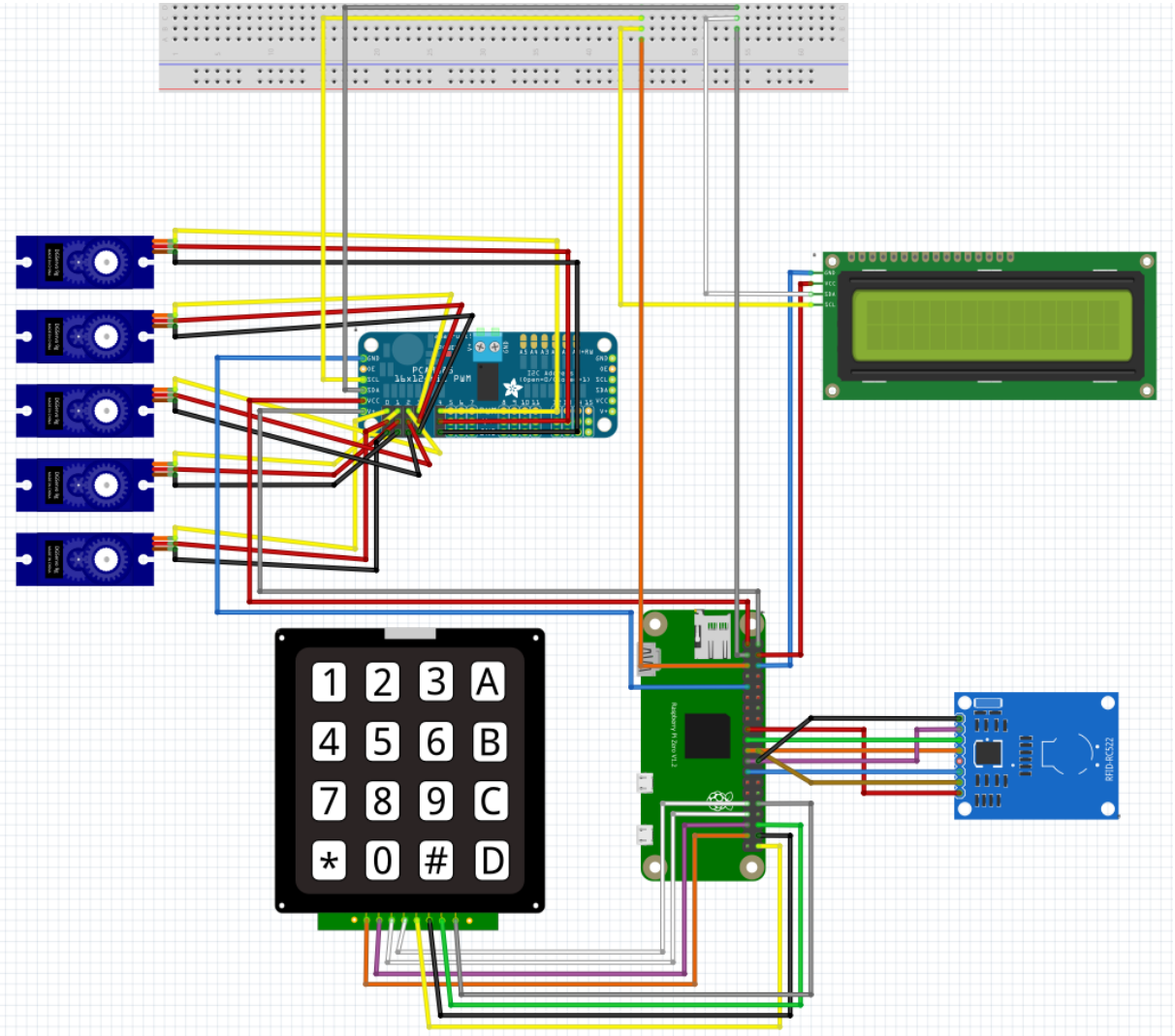
Schemat:

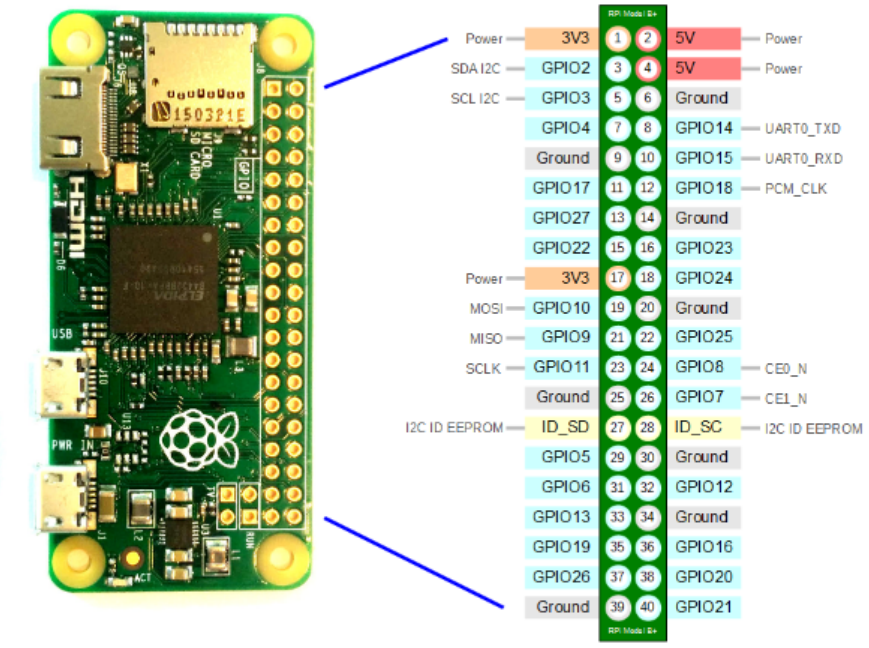


***Wykorzystana platforma sprzętowa, czujniki pomiarowe, elementy wykonawcze**:*

servo x5, klawiatura, czujnik RFID, aplikacja na telefonie, buzzer

Schemat połączeniowy z wykorzystaniem programu Fritzing





Wykorzystane elementy

- **1x Czytnik kart RFID RC522**
- **1x 16 kanałowy servo motor driver**
- **1x Wyświetlacz LCD 16X2 Alphanumeric Display**
- **1x Raspberry Pi Zero W**
- **1x 16-klawiszowa klawiatura 4x4 DWT-0261**
- **3x Małe Servo SG90 9g**
- **1x Małe ServoTG9e 9g**
- **1x Duże Servo ACOMS AS-12**

Projekt a realizacja

Z początku wgraliśmy na kartę SD Raspberry Pi OS. Następnie podłączyliśmy kartę do Raspberry Pi Zero W, które umożliwia łączenie się poprzez sieć Wi-Fi. Zamontowaliśmy czujniki i większość elementów. Mieliśmy problem z Pythonem (odinstalowano go, po czym zainstalowano, ale nie była to najlepsza decyzja). Pomogło ponowne wgranie systemu na kartę SD. Trzeba też było odpowiednio skonfigurować wstępne ustawienia i opcje (co będzie pokazane niżej).
Wszystkie elementy celu zostały spełnione poza ostatnim - buzzerem. Uznaliśmy, że jeśli servo wydaje dźwięki przy otwieraniu to nie potrzeba innego dźwięku, który oznajmia że coś się dzieje ze skrzynką, ponadto wszystko jest wyświetlane na ekranie.
Nie spodziewaliśmy się, że zostanie wykorzystane aż tyle pinów w Raspberry Pi stąd pomysł z driverem do serv. Dzięki temu wszystkie serva spokojnie mieszczą się na urządzeniu, a można również dołączyć większą liczbę serv niż 4.
Do projektu chcieliśmy dołączyć również płytkę umożliwiającą korzystanie z Raspberry Pi przewodowo ale okazała się nieprzydatna. Cały program został właściwie stworzony przez Putty - protokół SSH dzięki któremu mogliśmy pracować zdalnie poprzez tunel przekierowujący IP do localhosta w Raspberry Pi.
Aplikacja na telefon została napisana w Kotlinie. Połączenie odbywa się poprzez udostępnienie telefonu jako routera i podłączenie aplikacji do IP Raspberry Pi. Aplikacja postuje na serwerze, która jest na płytce informacje o numerze skrytki i kodzie do tej skrytki a RPi to odbiera.

Możliwości rozwoju projektu

Skrzynka jest prototypem, który przypomina paczkomat. Wystarczy kliknąć odpowiedni przycisk na telefonie, a skrytka sama się otworzy. Można powiedzieć że stworzyliśmy paczkomat v2 z 4 skrytkami.

Najważniejsze fragmenty kodu z komentarzami

Pobrane biblioteki i inicjalizacja zmiennych

```
#!/usr/bin/python

# ------------ LIBRARIES SECTION ------------
import time
import RPi.GPIO as GPIO
# server
import requests
# rfid
from mfrc522 import SimpleMFRC522
# LCD
from rpi_lcd import LCD
# keyboard
from pad4pi import rpi_gpio
# servo driver
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
import board
import busio
import adafruit_pca9685
from adafruit_servokit import ServoKit

# ------------ STARTER PACK STUFF SECTION ------------
i2c = busio.I2C(board.SCL, board.SDA)
pca = adafruit_pca9685.PCA9685(i2c)     # driver
kit = ServoKit(channels=16)             # driver
pca.frequency = 50                      # set frequency for servos driver
card = SimpleMFRC522()                  # rfid
lcd = LCD(address=0x3f)                 # lcd

keyboard = ""                           # keyboard input
KEYPAD = [
    ["1","2","3","A"],
    ["4","5","6","B"],
    ["7","8","9","C"],
    ["*","0","#","D"]
    ]
ROW_PINS = [21, 20, 16, 12]             # set pins BCM mode
COL_PINS = [26, 19, 13, 6]              # set pins BCM mode
factory = rpi_gpio.KeypadFactory()
keypad = factory.create_keypad(keypad=KEYPAD,row_pins=ROW_PINS, col_pins=COL_PINS,
gpio_mode=GPIO.BCM)
```

Aktywacja klawiatury z dołączonej biblioteki

```
def activate_keyboard():
    keypad.registerKeyPressHandler(key_pressed)
```

```
def key_pressed(key):
    global keyboard
    try:
        keyboard += str(key)
    except ValueError:
        print("F")
```

Pobieranie danych z serwera

```
def get_from_server():
    global server, case, code, if_open
    server = eval(requests.get("http://0.0.0.0:8183").text)
    code, case = str(server['code']), int(server['case'])
```

Główny program

```
activate_keyboard()                         # włączono odczytywanie klawiatury
while True:
    set_servos()                        # wyzerowanie pozycji serv na stan początkowy
    try:
        while str(keyboard) == "" or str(keyboard[len(keyboard)-1]) not in exit_btns:
            # tylko ostatnia litera inputu umożliwia wyjście ze skrzynki
            lcd_text("Hi! ", 1)
            lcd_text("Put your card!", 2)
            id, text = card.read()          # oczekiwanie na kartę
            lcd_show_card_id()
            if id == 551701029536:          # odczytywanie tylko jednej karty
                id = 0
                open_box()                  # otworzenie skrzynki
                time.sleep(4)
                while str(keyboard) == "" or str(keyboard[len(keyboard)-1]) not in
exit_btns:
                    get_from_server()# pobranie ostatnio wygenerowanego kodu
                    check_if_wrong_password() # sprawdzenie czy wejście nie
                                        # przekroczyło długości ciągu 4 cyfrowego PINu
                    lcd_show_password()     # odświeżenie ekranu
                    time.sleep(0.5)   # by nie zajmować zbytnio procesora

                    if str(code) == str(keyboard):
                        # post_info_to_the_server(code, case, "cant")
                        keyboard = ""
                        open_for_x_seconds(8) # ograniczony czas otwarcia skrzynki
                        lcd_text("Thats all! ", 1)
                        lcd_text("THX!", 2)
                        time.sleep(3)       # chwila na przeczytanie tekstu
```

```
                        close_box()         # zamknięcie skrzynki
                        lcd.clear()
                        break
            else:
                lcd_text("Wrong card! ", 1) # obsługa niepoprawnej karty
                lcd_text("Try again!", 2)
    except KeyboardInterrupt:               # naciśnięcie ctrl+C z poziomu konsoli
        lcd.clear()
        print('interrupted!')
        break
    except TypeError:                       # w razie błędu typu (sprawdza czy do
                                            # serwera wysłano poprawne dane)
        lcd_text("My processor", 1)
        lcd_text("is broken :(", 2)

    if str(keyboard[len(keyboard)-1] ) in exit_btns:
        keyboard=""
        lcd_text("My job here...", 1)
        lcd_text("...is done", 2)
        time.sleep(2)
    else:
        keyboard=""
        lcd_text("Bye!", 1)
        lcd_text("Have a nice day!", 2)
        time.sleep(2)
    cleanup()      # zakończenie działania i czyszczenie (zacznie
                   # od nowa wyświetlać procedurę wejścia do skrzynki)
```

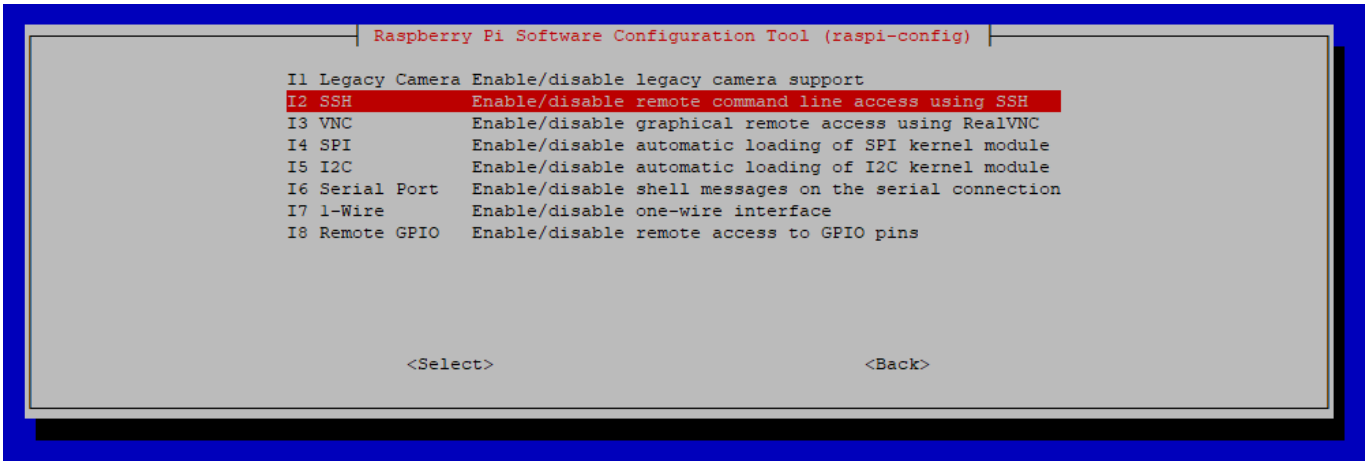
Uruchamiamy skrzynkę za pomocą funkcji

```
def open_box():
    kit.continuous_servo[0].throttle = 1
```

Uruchamiamy również serwer z odrębnego pliku server.py

```
# uruchomienie serwera
with socketserver.TCPServer(("0.0.0.0", PORT), MyServer) as httpd:
    print("serving at 0.0.0.0:{}".format(PORT))
    httpd.serve_forever()
```

Tutaj należało udostępnić opcje I2, I4, I5, które umożliwiały prawidłowe działanie SSH, I2C, SPI



Autostart
Za pomocą komendy *sudo nano /etc/rc.local* należało dodać aplikacje autostartu.

```
sudo python /home/helena/server.py &
sudo python /home/helena/main.py 2>&1 | tee /home/helena/log.txt &
```

log.txt informował o ewentualnych błędach programu

Wi-Fi
Bardzo ważnym elementem było dodanie do ***wpa_supplicant.conf*** linijki informującej o dodatkowym Wi-Fi, dzięki czemu RPi łączy się automatycznie z pierwszą napotkaną siecią, a w razie niepowodzenia z kolejną zapisaną w pliku.

Osiągnęliśmy to poleceniem:
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

```
network={
    ssid="Test Wifi Network"
    psk="SecretPassWord"
}
```

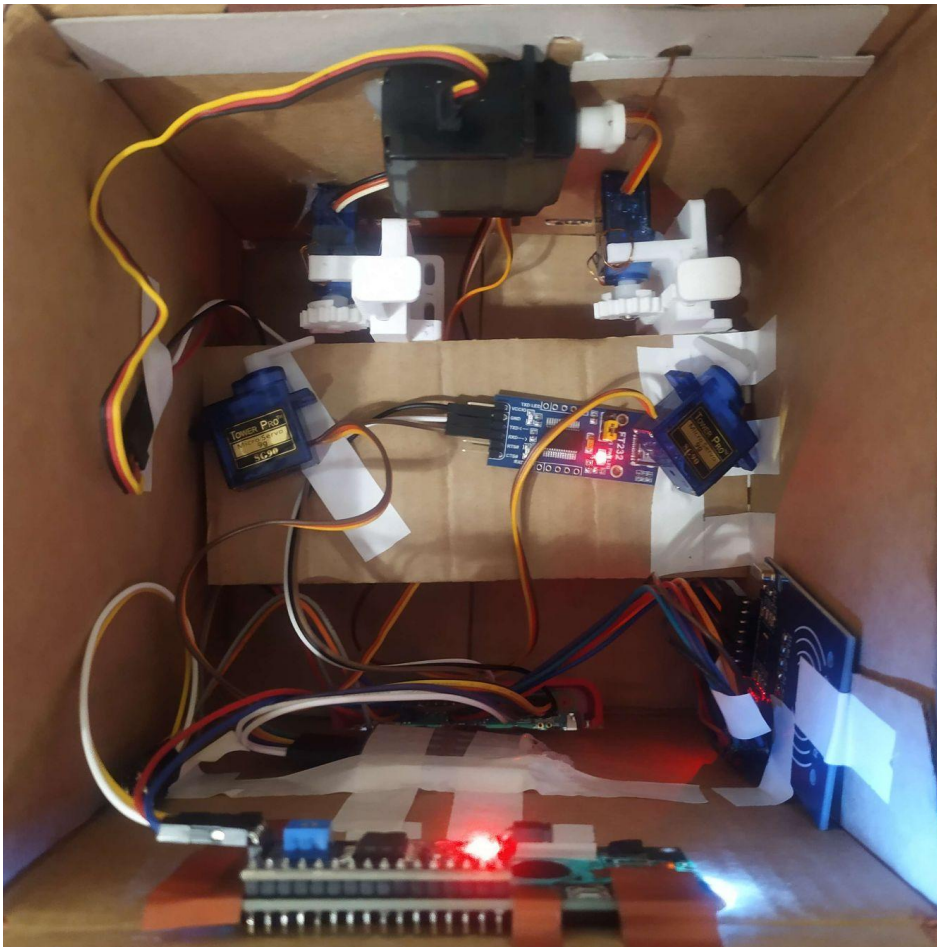
Zdjęcia skrzynki - przykładowe napisy na skrzynce i zawartość pudełka


Your card ID:
5517010295 36


Password:
***

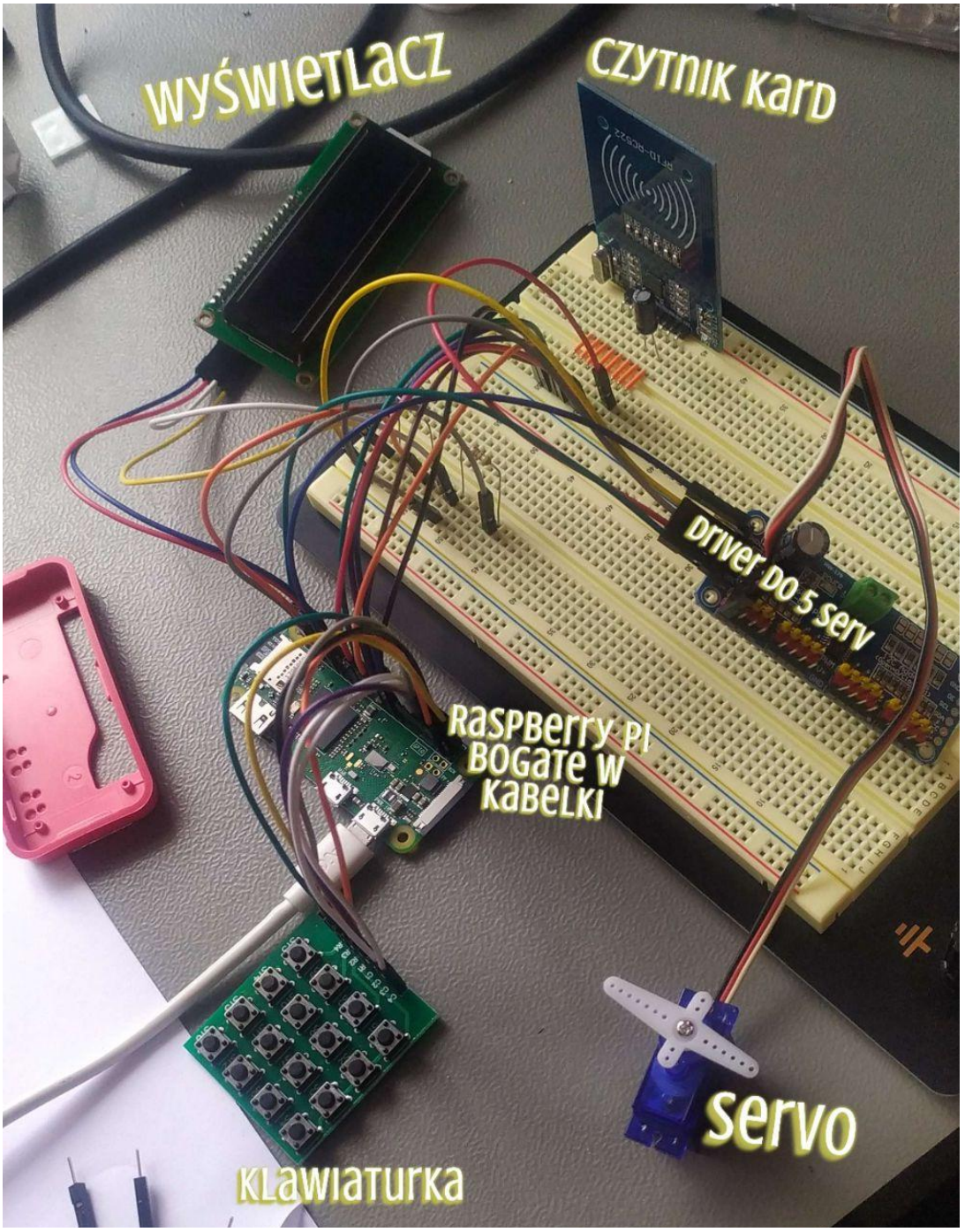

Box: 4
5

na zakończenie skrzynka wyświetla powyższy napis



driver znajduje się pod kładką z kartonu

Zdjęcia elementów i przewodów na płytce uniwersalnej w fazie testów



Zrzut ekranu aplikacji

Podsumowanie i wnioski

Projekt zdecydowanie przybliżył nam działanie płytki Raspberry Pi i jej potencjał. Wystarczy odpowiedni czujnik i odpowiednie elementy wykonawcze aby stworzyć za pomocą niepozornego komputerka system, który może być stosowany w domu. Pracy było sporo, jeszcze więcej było błędów, ale po 2 tygodniach zadanie udało się wykonać.