

TECNICA-MENTE 2019 – CICLO SUPERIOR

ESCUELA DE EDUCACIÓN SECUNDARIA TECNICA Nº: 9 Antonio José Rodríguez.

DISTRITO: Lanús.

REGIÓN: 2.

TÍTULO DEL PROYECTO: Sistema Auxiliar para Personas con Limitaciones Motrices.

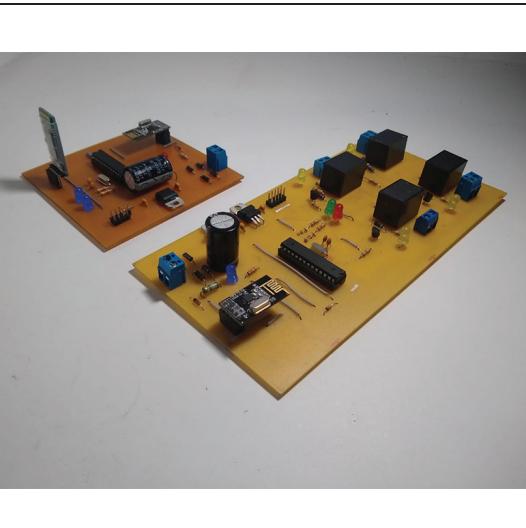
Equipo responsable: Zatloukal Maule, Julián; Da Cruz, Agustín Tomás; Villegas Cabral, Rocío.

Docente Tutor: Otero, Diego; Castro, Pujol; Della Paolera, Sergio.

1. Objetivo del Proyecto

Las relaciones cercanas con personas de capacidades limitadas nos llevaron a plantearnos la idea de implementar una ayuda para mejorar su calidad de vida a partir de nuestros conocimientos como técnicos electrónicos. A principio de año, junto a nuestros profesores, ideamos un servicio sistema para beneficiarlos; al cual luego lo bautizamos **Sistema Auxiliar para Personas con Limitaciones Motrices (SAPLM)**.

El objetivo que quisieramos alcanzar consta en facilitar su vida cotidiana a través de la simplificación de los mandos y controles de sistemas tanto eléctricos como electromecánicos hogareños sin la necesidad de un tercero que lo asista en estas tareas, como podría ser desde un simple encendido de luces hasta regular la calefacción mediante un simple comando por voz procesado por teléfono celular.



2. Descripción del Proyecto

Este sistema se compone en primera instancia de un teléfono móvil Android, el cual el paciente dispone en todo momento, en este se reciben y procesan los comandos de voz mediante la aplicación diseñada, programada y desarrollada por nosotros. Esta aplicación fue desarrollada en Android Studio, el programa oficial de Android para desarrollo de aplicaciones móviles, y programada en lenguaje Java. Ésta integra dos librerías las cuales resuelven el reconocimiento de voz, pocket-sphinx y la API oficial de Google destinada a este propósito.

Estos comandos luego de ser procesados en la aplicación se envían mediante Bluetooth a lo que nosotros denominamos la placa principal o maestra. La cual, inmediatamente a través de un sistema de radiofrecuencia, retransmite el comando recibido a la placa de mando correspondiente. Estas llamadas placas de mando son aquellas que interactúan directamente con el dispositivo o maquinaria a controlar, reciben un simple comando mediante el sistema de radiofrecuencia enviado desde la placa central y cumplen la orden.

Concretamente en nuestro proyecto se exponen dos placas de mando, una al cual la llamamos "Placa de mandos de potencia" y a la otra "Placa de mandos motrices". Ambas se componen esencialmente del modulo de radiofrecuencia junto a un microcontrolador de 8 bits de la familia AVR y de todos los elementos para la regulación del voltaje. La "Placa de mandos de potencia" dispone de 4 contactos electromecánicos los cuales controlaran elementos simples, como una estufa, ventilador, luz, velador u otros. En cuanto a la "Placa de mandos motrices" esta dispone de dos drivers para motores paso a paso los cuales manejaran una camilla y una cortina.

El proyecto está planteado de manera extensible de manera que soporte la adición de nuevas placas de mando al sistema ya planteado; luego de una correspondiente actualización del firmware de la placa principal.

3. ESTADO DE DESARROLLO DEL PROYECTO:

Luego de cuatro meses de desarrollo nos encontramos con una firme base del proyecto ya armada. Largas tardes y noches resultaron en una aplicación capaz de recibir comandos de voz con una exactitud competente y de transmitirlos mediante el protocolo Bluetooth. Los firmware de las placas ya están asentados.

Todas las placas se encuentran diseñadas pero por el momento solo construimos dos de ellas, las cuales funcionan a la perfección. La placa principal sufrió un rediseño luego de que en una jornada de trabajo caigamos en la cuenta de que esta tenía una configuración de conexiones errónea. Se decidió construir primero la "Placa de mandos de potencia" debido a su simplicidad, esta no trajo dificultades al armado y sus modificaciones fueron mínimas luego de ser completada.

Con ambas placas en marcha y una aplicación funcional el sistema está listo para ser utilizado en situaciones sencillas como prender una estufa o luz; sin embargo no lo son para alguien con capacidades diferentes.

Diagrama de bloques

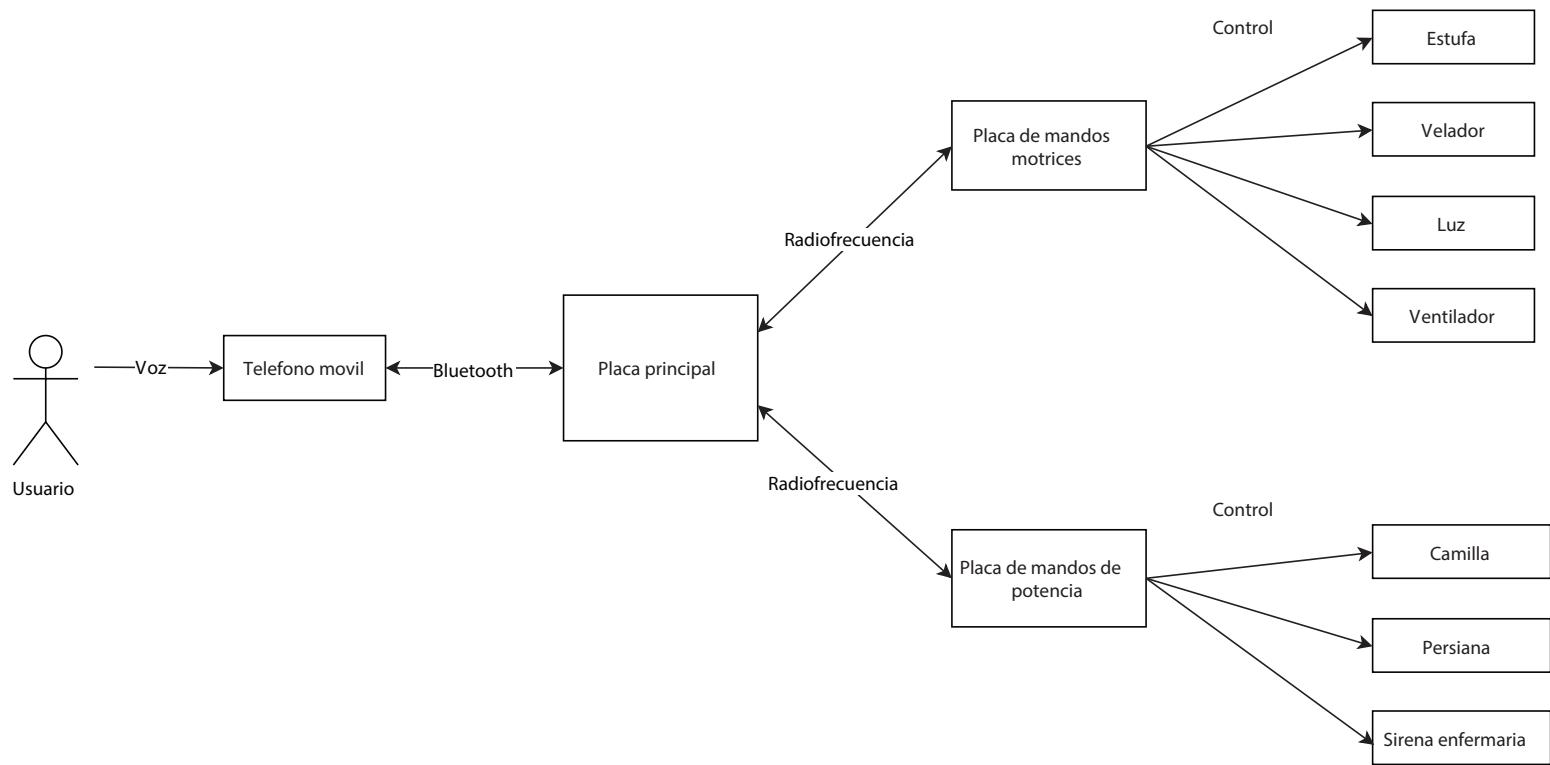
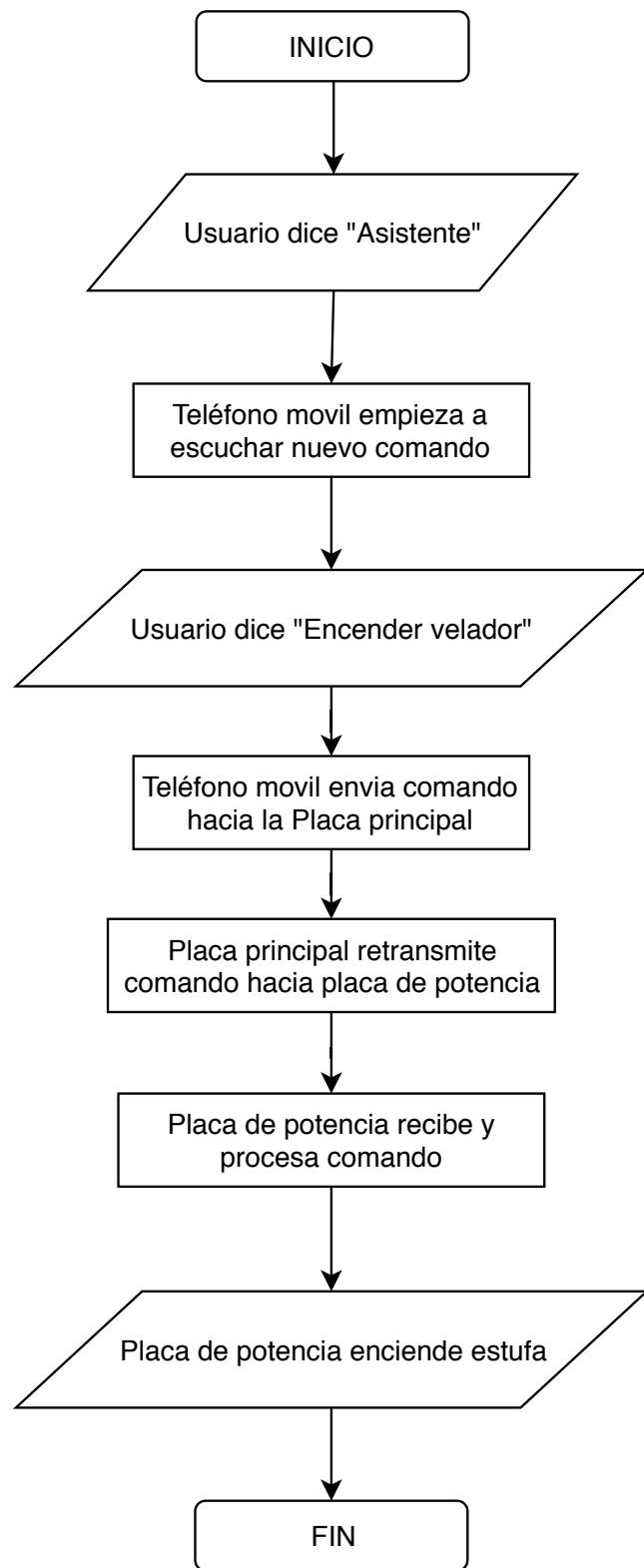


Diagrama de flujo para encender una estufa



Comandos por voz

- Palabra clave : Asistente

Comandos disponibles

Encender estufa	Apagar estufa
Encender velador	Apagar velador
Encender ventilador	Apagar ventilador
Llamar enfermería	

Estados de escucha



Escucha
deshabilitada

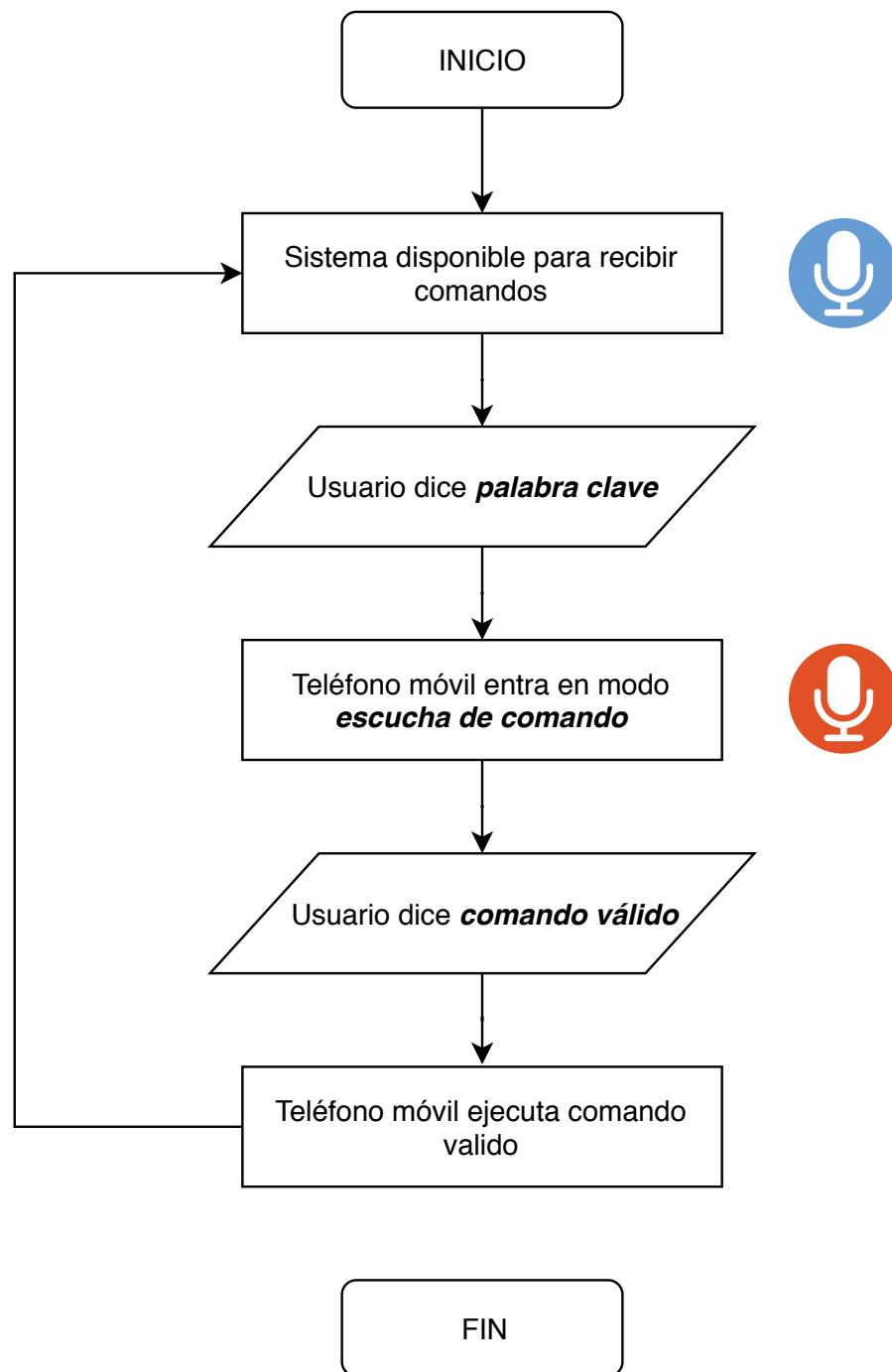


Escuchando
palabra clave



Escuchando
comando

Diagrama de flujo generico para accionar comando



Interfaz gráfica de la aplicación

- Pantalla principal



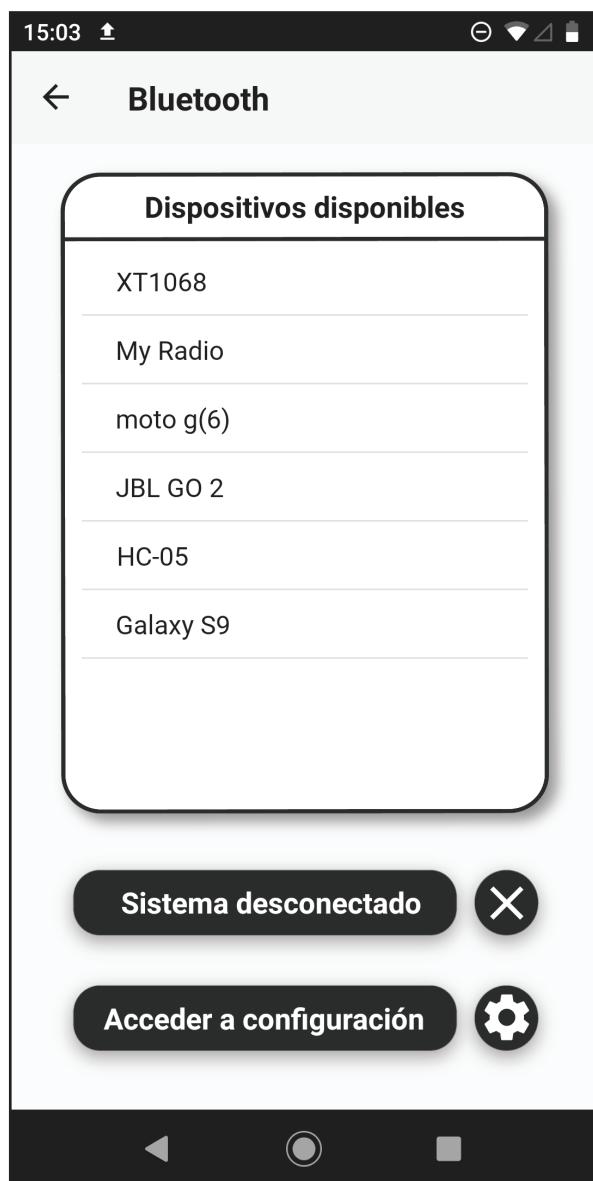
Interfaz gráfica de la aplicación

- **Menú de configuraciones**



Interfaz gráfica de la aplicación

- **Menu de Bluetooth**



CODIGO DE FUENTE PLACA DE POTENCIA

```
1  
2 #define F_CPU 16000000UL  
3  
4 #define bit_get(p,m) ((p) & (m))  
5 #define bit_set(p,m) ((p) |= (m))  
6 #define bit_clear(p,m) ((p) &= ~(m))  
7 #define bit_flip(p,m) ((p) ^= (m))  
8 #define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))  
9 #define BIT(x) (0x01 << (x))  
10 #define LONGBIT(x) ((unsigned long)0x00000001 << (x))  
11  
12 #include "nrf24.h"  
13 #include "Command_Handler.h"  
14  
15 #include <avr/io.h>  
16 #include <string.h>  
17 #include <stdlib.h>  
18 #include <util/delay.h>  
19  
20 bool initRF();  
21 void initIO();  
22 void faultyRF_Alarm();  
23  
24 int main(void)  
25 {  
26     initIO();  
27     initRF();  
28  
29     while (1)  
30     {  
31         if(nrf24_dataReady())  
32         {  
33             bit_clear(PORTB, BIT(0));  
34  
35             nrf24_getData(command_buffer);  
36  
37             bit_set(PORTD, BIT(7));  
38             _delay_ms(500);  
39             commandType currentCommand;  
40             bool success = decomposeCommand(command_buffer, &currentCommand,    ↴  
41                         parameter);  
42             if (success) { currentCommand.handlerFunction(); }  
43             bit_clear(PORTD, BIT(7));  
44         }  
45         if (nrf24_checkAvailability()==false) { while(initRF()==false); }  
46     }  
47 }  
48  
49 void initIO(){  
50     /*  
51         Input/Output pin initialization
```

```
52      1 : OUTPUT | 0 : INPUT | 0b76543210 Bit order
53      ATTACHMENTS
54          RELAY 0    : PD3           |  OUTPUT
55          RELAY 1    : PD2           |  OUTPUT
56          RELAY 2    : PD6           |  OUTPUT
57          RELAY 3    : PD5           |  OUTPUT
58          RED LED   : PD7           |  OUTPUT
59          GREEN LED  : PB0          |  OUTPUT
60      nRF24L01
61          CE    : PC0           |  OUTPUT
62          CSN   : PC1           |  OUTPUT
63          MISO  : PDD0 (MSPIM MISO ATMEGA) |  INPUT
64          MOSI  : PDI1 (MSPIM MOSI ATMEGA) |  OUTPUT
65          SCK   : PD4 (MSPIM XCK)     |  OUTPUT
66      */
67  DDRD = 0b11111110;
68  DDRB = 0b00101001;
69  DDRC = 0b11011111;
70
71  PORTD = 0b00000000;
72  PORTC = 0b00000000;
73  PORTB = 0b00000000;
74 }
75
76 bool initRF(){
77     uint8_t tx_address[5] = {0xD7,0xD7,0xD7,0xD7,0xD7};
78     uint8_t rx_address[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
79
80     initliazeMemory();
81
82     /* Power down module */
83     nrf24_powerDown();
84
85     nrf24_init();
86
87     /* Channel #112 , payload length: 32 */
88     nrf24_config(112,32);
89
90     /* Check module configuration */
91     if (nrf24_checkConfig()==false) { faultyRF_Alarm(); return false; }
92
93     /* Set the device addresses */
94     nrf24_tx_address(tx_address);
95     nrf24_rx_address(rx_address);
96
97     /* Power up in receive mode */
98     nrf24_powerUpRx();
99
100    return true;
101 }
102
103 void faultyRF_Alarm(){
```

```
104     bit_clear(PORTD, BIT(7));
105     for (uint8_t x = 0; x < 6; x++)
106     {
107         bit_flip(PORTD, BIT(7));
108         _delay_ms(125);
109     }
110 }
112
113
114
115
```

```
1
2
3 #ifndef COMMAND_HANDLER_H_
4 #define COMMAND_HANDLER_H_
5
6 #include <stdbool.h>
7 #include <stdint.h>
8
9 #ifndef nullptr
10 #define nullptr ((void *)0)
11 #endif
12
13 #ifndef F_CPU
14 #define F_CPU           16000000UL
15 #endif
16
17 #define AVAILABLE_COMMANDS 6
18 #define COMMAND_BUFFER_SIZE 32
19 #define PARAMETER_BUFFER_SIZE 28
20
21 #ifndef BIT_MANIPULATION_MACRO
22 #define BIT_MANIPULATION_MACRO 1
23 #define bit_get(p,m) ((p) & (m))
24 #define bit_set(p,m) ((p) |= (m))
25 #define bit_clear(p,m) ((p) &= ~(m))
26 #define bit_flip(p,m) ((p) ^= (m))
27 #define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
28 #define BIT(x) (0x01 << (x))
29 #define LONGBIT(x) ((unsigned long)0x00000001 << (x))
30 #endif
31
32 typedef struct commandType {
33     const char *commandBase;
34     uint8_t nParameters;
35     void (*handlerFunction)();
36 } commandType;
37
38 void *parameter[3];
39 uint8_t *command_buffer;
40 extern bool initliazeMemory();
41 bool memoryInitialized;
42 extern void TURN_RELAY_ON_HANDLE(), TURN_RELAY_OFF_HANDLE(),
        BUILT_IN_LED_TEST_HANDLER(), TURN_EVERYTHING_ON_HANDLER(),
        TURN_EVERYTHING_OFF_HANDLER(), CALL_NURSE_HANDLER(); ↗  
↗
43
44 extern void composeCommand(void* output_buffer, commandType* commandT, void** ↗
    inputParameter);
45 extern bool decomposeCommand(void* input_buffer, commandType* commandT, void** ↗
    outputParameter);
46
47
48 #endif /* COMMAND_HANDLER_H_ */
```

```
1 #include "Command_Handler.h"
2 #include "nrf24.h"
3 #include <stdbool.h>
4 #include <string.h>
5 #include <stdlib.h>
6 #include <stdint.h>
7 #include <avr/io.h>
8 #include <util/delay.h>
9
10
11
12 const commandType availableCommand[AVAILABLE_COMMANDS] = {
13     { .commandBase = "TURN_RELAY_ON", .nParameters = 1, .handlerFunction =
14         &TURN_RELAY_ON_HANDLE},
15     { .commandBase = "TURN_RELAY_OFF", .nParameters = 1, .handlerFunction =
16         &TURN_RELAY_OFF_HANDLE},
17     { .commandBase = "BUILT_IN_LED_TEST", .nParameters = 0, .handlerFunction =
18         &BUILT_IN_LED_TEST_HANDLER},
19     { .commandBase = "TURN_EVERYTHING_ON", .nParameters = 0, .handlerFunction =
20         &TURN_EVERYTHING_ON_HANDLER},
21     { .commandBase = "TURN_EVERYTHING_OFF", .nParameters = 0, .handlerFunction =
22         &TURN_EVERYTHING_OFF_HANDLER},
23     { .commandBase = "CALL_NURSE", .nParameters = 0, .handlerFunction =
24         &CALL_NURSE_HANDLE}
25 };
26
27 bool initliazeMemory(){
28     if(memoryInitialized) return false;
29     parameter[0] = (void*)calloc(28,1);
30     parameter[1] = (void*)calloc(28,1);
31     parameter[2] = (void*)calloc(28,1);
32     command_buffer = (uint8_t*)calloc(32,1);
33     if(parameter[0]==nullptr||parameter[1]==nullptr||parameter[2]==nullptr||
34         command_buffer==nullptr) return false;
35     memoryInitialized = true;
36     return true;
37 }
38
39
40 void composeCommand(void* output_buffer, commandType* commandT, void***
41 inputParameter){
42     strcpy(output_buffer, commandT->commandBase);
43     char* startParamPTR = (char*)(output_buffer+strlen(commandT->commandBase));
44     char* endParamPTR = (char*)(startParamPTR+1+strlen(*inputParameter));
45
46     for (uint8_t index = 0; index < commandT->nParameters; index++){
47         *startParamPTR='[';
48         strcpy(startParamPTR+1, *inputParameter);
49         *endParamPTR=']';
50         startParamPTR=(endParamPTR+1);
51         if (index!=(commandT->nParameters-1)){
52             inputParameter++;
53         }
54     }
55 }
```

```
45         uint8_t len = strlen(*inputParameter);
46         endParamPTR = (char*)(startParamPTR+len+1);
47     }
48 }
49 *startParamPTR='\\0';
50 }
51
52 bool decomposeCommand(void* input_buffer, commandType* commandT, void** outputParameter){      ↵
53
54     for (uint8_t index = 0; index < AVAILABLE_COMMANDS; index++){
55         if (memmem(input_buffer, COMMAND_BUFFER_SIZE, availableCommand
56             [index].commandBase, strlen(availableCommand[index].commandBase))!=
57             nullptr)
58         {
59             *commandT = availableCommand[index]; break;
60         }
61     else if (index==(AVAILABLE_COMMANDS-1)) { return false;}
62 }
63
64     for (uint8_t x = 0; x < commandT->nParameters; x++){
65         uint8_t* startNumPTR = memchr(input_buffer, '[', COMMAND_BUFFER_SIZE);
66         uint8_t* endNumPTR = memchr(input_buffer, ']', COMMAND_BUFFER_SIZE);
67         if (startNumPTR==nullptr||endNumPTR==nullptr) { if(x==0) return false;
68             break; }
69         (*startNumPTR) = 0x20;
70         (*endNumPTR) = 0x20;
71         startNumPTR++;
72         uint32_t bytes = ((endNumPTR)) - ((startNumPTR));
73         if (bytes>PARAMETER_BUFFER_SIZE) return false;
74         memcpy(outputParameter[x], startNumPTR, bytes);
75     }
76
77     return true;
78 }
79
80 void TURN_RELAY_ON_HANDLE() {
81     uint8_t relayIndex = atoi(parameter[0]);
82     switch (relayIndex) {
83         case 0:
84             bit_set(PORTD, BIT(3));
85             break;
86         case 1:
87             bit_set(PORTD, BIT(2));
88             break;
89         case 2:
90             bit_set(PORTD, BIT(6));
91             break;
92     }
```

```
93 }
94
95 void TURN_RELAY_OFF_HANDLE() {
96     uint8_t relayIndex = atoi(parameter[0]);
97     switch (relayIndex) {
98         case 0:
99             bit_clear(PORTD, BIT(3));
100            break;
101        case 1:
102            bit_clear(PORTD, BIT(2));
103            break;
104        case 2:
105            bit_clear(PORTD, BIT(6));
106            break;
107        case 3:
108            bit_clear(PORTD, BIT(5));
109            break;
110    }
111 }
112
113 void BUILT_IN_LED_TEST_HANDLER(){
114     for (uint8_t x = 0; x < 8; x++) {
115         bit_flip(PORTD, BIT(7));
116         bit_flip(PORTB, BIT(0));
117         _delay_ms(250);
118     }
119     bit_clear(PORTD, BIT(7));
120     bit_clear(PORTB, BIT(0));
121 }
122
123 void TURN_EVERYTHING_ON_HANDLER(){
124     bit_set(PORTD, BIT(3));
125     bit_set(PORTD, BIT(2));
126     bit_set(PORTD, BIT(6));
127     bit_set(PORTD, BIT(5));
128 }
129
130 void TURN_EVERYTHING_OFF_HANDLER(){
131     bit_clear(PORTD, BIT(3));
132     bit_clear(PORTD, BIT(2));
133     bit_clear(PORTD, BIT(6));
134     bit_clear(PORTD, BIT(5));
135 }
136
137 void CALL_NURSE_HANDLE(){
138     bit_set(PORTD, BIT(5));
139     _delay_ms(500);
140     bit_clear(PORTD, BIT(5));
141     _delay_ms(500);
142     bit_set(PORTD, BIT(5));
143     _delay_ms(500);
144     bit_clear(PORTD, BIT(5));
```

```
145     _delay_ms(500);
146     bit_set(PORTD, BIT(5));
147     _delay_ms(500);
148     bit_clear(PORTD, BIT(5));
149 }
150
```

```
1 #ifndef NRF24
2 #define NRF24
3
4 #include "nRF24L01_Definitions.h"
5 #include <stdint.h>
6 #include <stdbool.h>
7 #include <util/delay.h>
8
9 #define LOW 0
10 #define HIGH 1
11
12 #define nrf24_ADDR_LEN 5
13 #define nrf24_CONFIG ((1<<EN_CRC)|(0<<CRCO))
14
15 #define NRF24_TRANSMISSION_OK 0
16 #define NRF24_MESSAGE_LOST 1
17
18 void nrf24_init();
19 void nrf24_rx_address(uint8_t* adr);
20 void nrf24_tx_address(uint8_t* adr);
21 void nrf24_config(uint8_t channel, uint8_t pay_length);
22 bool nrf24_checkRegister(uint8_t reg, uint8_t desiredValue, uint8_t len);
23 bool nrf24_checkConfig();
24 bool nrf24_checkAvailability();
25
26
27 uint8_t nrf24_dataReady();
28 uint8_t nrf24_isSending();
29 uint8_t nrf24_getStatus();
30 uint8_t nrf24_rxFifoEmpty();
31
32 void nrf24_send(uint8_t* value);
33 void nrf24_getData(uint8_t* data);
34
35 uint8_t nrf24_payloadLength();
36
37 uint8_t nrf24_lastMessageStatus();
38 uint8_t nrf24_retransmissionCount();
39
40 uint8_t nrf24_payload_length();
41
42 void nrf24_powerUpRx();
43 void nrf24_powerUpTx();
44 void nrf24_powerDown();
45
46 uint8_t spi_transfer(uint8_t tx);
47 void nrf24_transmitSync(uint8_t* dataout, uint8_t len);
48 void nrf24_transferSync(uint8_t* dataout, uint8_t* datain, uint8_t len);
49 void nrf24_configRegister(uint8_t reg, uint8_t value);
50 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len);
51 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len);
52
```

```
53 extern void nrf24_setupPins();  
54  
55 extern void nrf24_ce_digitalWrite(uint8_t state);  
56  
57 extern void nrf24_csn_digitalWrite(uint8_t state);  
58  
59 extern void nrf24_sck_digitalWrite(uint8_t state);  
60  
61 extern void nrf24_mosi_digitalWrite(uint8_t state);  
62  
63 extern uint8_t nrf24_miso_digitalRead();  
64  
65 #endif  
66
```

```
1  
2 #define UCPHA0 1  
3 #define F_CPU 8000000UL  
4 #define BAUD_RATE 9600UL  
5 #define UBRR_VALUE ((F_CPU)/(2UL*BAUD_RATE))-1  
6  
7 #include "nrf24.h"  
8 #include <avr/io.h>  
9  
10 uint8_t payload_len;  
11 uint8_t selectedChannel;  
12  
13 void nrf24_init()  
14 {  
15     nrf24_setupPins();  
16     nrf24_ce_digitalWrite(LOW);  
17     nrf24_csn_digitalWrite(HIGH);  
18 }  
19  
20 void nrf24_config(uint8_t channel, uint8_t pay_length)  
21 {  
22     /* Use static payload length ... */  
23     payload_len = pay_length;  
24     selectedChannel = channel;  
25     // Set RF channel  
26     nrf24_configRegister(RF_CH,channel);  
27     // Set length of incoming payload  
28     nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...  
29     nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe  
30     nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used  
31     nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used  
32     nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used  
33     nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used  
34     // 1 Mbps, TX gain: 0dbm  
35     nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));  
36     // CRC enable, 1 byte CRC length  
37     nrf24_configRegister(CONFIG,nrf24_CONFIG);  
38     // Auto Acknowledgment  
39     nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|  
        (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));  
40     // Enable RX addresses  
41     nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|  
        (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));  
42     // Auto retransmit delay: 1000 us and Up to 15 retransmit trials  
43     nrf24_configRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC));  
44     // Dynamic length configurations: No dynamic length  
45     nrf24_configRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|  
        (0<<DPL_P4)|(0<<DPL_P5));  
46  
47 }  
48  
49 bool nrf24_checkConfig(){
```

```
50     // Check all registers
51     if (nrf24_checkRegister(RF_CH, selectedChannel,1)==false) return false;
52     if (nrf24_checkRegister(RX_PW_P0, 0x00,1)==false) return false;
53     if (nrf24_checkRegister(RX_PW_P1, payload_len,1)==false) return false;
54     if (nrf24_checkRegister(RX_PW_P2, 0x00,1)==false) return false;
55     if (nrf24_checkRegister(RX_PW_P3, 0x00,1)==false) return false;
56     if (nrf24_checkRegister(RX_PW_P4, 0x00,1)==false) return false;
57     if (nrf24_checkRegister(RX_PW_P5, 0x00,1)==false) return false;
58     if (nrf24_checkRegister(RF_SETUP, (0<<RF_DR)|(0x03<<RF_PWR),1)==false)    ↵
59         return false;
60     if (nrf24_checkRegister(CONFIG,nrf24_CONFIG,1)==false) return false;
61     if (nrf24_checkRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|    ↵
62         (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5),1)==false) return false;
63     if (nrf24_checkRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC),1)==false) return ↵
64         false;
65     if (nrf24_checkRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|    ↵
66         (0<<DPL_P3)|(0<<DPL_P4)|(0<<DPL_P5),1)==false) return false;
67     return true;
68 }
69
70
71
72
73 /* Set the RX address */
74 void nrf24_rx_address(uint8_t * adr)
75 {
76     nrf24_ce_digitalWrite(LOW);
77     nrf24_writeRegister(RX_ADDR_P1,adr,nrf24_ADDR_LEN);
78     nrf24_ce_digitalWrite(HIGH);
79 }
80
81 /* Returns the payload length */
82 uint8_t nrf24_payload_length()
83 {
84     return payload_len;
85 }
86
87 /* Set the TX address */
88 void nrf24_tx_address(uint8_t* adr)
89 {
90     /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */
91     nrf24_writeRegister(RX_ADDR_P0,adr,nrf24_ADDR_LEN);
92     nrf24_writeRegister(TX_ADDR,adr,nrf24_ADDR_LEN);
93 }
94
95 /* Checks if data is available for reading */
96 /* Returns 1 if data is ready ... */
```

```
97 uint8_t nrf24_dataReady()
98 {
99     // See note in getData() function - just checking RX_DR isn't good enough
100    uint8_t status = nrf24_getStatus();
101
102    // We can short circuit on RX_DR, but if it's not set, we still need
103    // to check the FIFO for any pending packets
104    if ( status & (1 << RX_DR) )
105    {
106        return 1;
107    }
108
109    return !nrf24_rxFifoEmpty();;
110 }
111
112 /* Checks if receive FIFO is empty or not */
113 uint8_t nrf24_rxFifoEmpty()
114 {
115     uint8_t fifoStatus;
116
117     nrf24_readRegister(FIFO_STATUS,&fifoStatus,1);
118
119     return (fifoStatus & (1 << RX_EMPTY));
120 }
121
122 /* Returns the length of data waiting in the RX fifo */
123 uint8_t nrf24_payloadLength()
124 {
125     uint8_t status;
126     nrf24_csn_digitalWrite(LOW);
127     spi_transfer(R_RX_PL_WID);
128     status = spi_transfer(0x00);
129     nrf24_csn_digitalWrite(HIGH);
130     return status;
131 }
132
133 /* Reads payload bytes into data array */
134 void nrf24_getData(uint8_t* data)
135 {
136     /* Pull down chip select */
137     nrf24_csn_digitalWrite(LOW);
138
139     /* Send cmd to read rx payload */
140     spi_transfer( R_RX_PAYLOAD );
141
142     /* Read payload */
143     nrf24_transferSync(data,data,payload_len);
144
145     /* Pull up chip select */
146     nrf24_csn_digitalWrite(HIGH);
147
148     /* Reset status register */
```

```
149     nrf24_configRegister(STATUS,(1<<RX_DR));
150 }
151
152 /* Returns the number of retransmissions occurred for the last message */
153 uint8_t nrf24_retransmissionCount()
154 {
155     uint8_t rv;
156     nrf24_readRegister(OBSERVE_TX,&rv,1);
157     rv = rv & 0x0F;
158     return rv;
159 }
160
161 // Sends a data package to the default address. Be sure to send the correct
162 // amount of bytes as configured as payload on the receiver.
163 void nrf24_send(uint8_t* value)
164 {
165     /* Go to Standby-I first */
166     nrf24_ce_digitalWrite(LOW);
167
168     /* Set to transmitter mode , Power up if needed */
169     nrf24_powerUpTx();
170
171     /* Do we really need to flush TX fifo each time ? */
172 #if 1
173     /* Pull down chip select */
174     nrf24_csn_digitalWrite(LOW);
175
176     /* Write cmd to flush transmit FIFO */
177     spi_transfer(FLUSH_TX);
178
179     /* Pull up chip select */
180     nrf24_csn_digitalWrite(HIGH);
181 #endif
182
183     /* Pull down chip select */
184     nrf24_csn_digitalWrite(LOW);
185
186     /* Write cmd to write payload */
187     spi_transfer(W_TX_PAYLOAD);
188
189     /* Write payload */
190     nrf24_transmitSync(value,payload_len);
191
192     /* Pull up chip select */
193     nrf24_csn_digitalWrite(HIGH);
194
195     /* Start the transmission */
196     nrf24_ce_digitalWrite(HIGH);
197 }
198
199 uint8_t nrf24_isSending()
200 {
```

```
201     uint8_t status;
202
203     /* read the current status */
204     status = nrf24_getStatus();
205
206     /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
207     if((status & ((1 << TX_DS) | (1 << MAX_RT))))
208     {
209         return 0; /* false */
210     }
211
212     return 1; /* true */
213
214 }
215
216 uint8_t nrf24_getStatus()
217 {
218     uint8_t rv;
219     nrf24_csn_digitalWrite(LOW);
220     rv = spi_transfer(NOP);
221     nrf24_csn_digitalWrite(HIGH);
222     return rv;
223 }
224
225 uint8_t nrf24_lastMessageStatus()
226 {
227     uint8_t rv;
228
229     rv = nrf24_getStatus();
230
231     /* Transmission went OK */
232     if((rv & ((1 << TX_DS))))
233     {
234         return NRF24_TRANSMISSION_OK;
235     }
236     /* Maximum retransmission count is reached */
237     /* Last message probably went missing ... */
238     else if((rv & ((1 << MAX_RT))))
239     {
240         return NRF24_MESSAGE_LOST;
241     }
242     /* Probably still sending ... */
243     else
244     {
245         return 0xFF;
246     }
247 }
248
249 void nrf24_powerUpRx()
250 {
251     nrf24_csn_digitalWrite(LOW);
252     spi_transfer(FLUSH_RX);
```

```
253     nrf24_csn_digitalWrite(HIGH);
254
255     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
256
257     nrf24_ce_digitalWrite(LOW);
258     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
259     nrf24_ce_digitalWrite(HIGH);
260
261     _delay_ms(5);
262 }
263
264 void nrf24_powerUpTx()
265 {
266     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
267
268     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
269
270     _delay_ms(5);
271 }
272
273 void nrf24_powerDown()
274 {
275     nrf24_ce_digitalWrite(LOW);
276     nrf24_configRegister(CONFIG,nrf24_CONFIG);
277
278     _delay_ms(5);
279 }
280
281 uint8_t spi_transfer(uint8_t tx)
282 {
283     uint8_t i = 0;
284     uint8_t rx = 0;
285
286     nrf24_sck_digitalWrite(LOW);
287
288     for(i=0;i<8;i++)
289     {
290
291         if(tx & (1<<(7-i)))
292         {
293             nrf24_mosi_digitalWrite(HIGH);
294         }
295         else
296         {
297             nrf24_mosi_digitalWrite(LOW);
298         }
299
300         nrf24_sck_digitalWrite(HIGH);
301
302         rx = rx << 1;
303         if(nrf24_miso_digitalRead())
304         {
```

```
305         rx |= 0x01;
306     }
307
308     nrf24_sck_digitalWrite(LOW);
309
310 }
311
312     return rx;
313 }
314
315 /* send and receive multiple bytes over SPI */
316 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
317 {
318     uint8_t i;
319
320     for(i=0;i<len;i++)
321     {
322         datain[i] = spi_transfer(dataout[i]);
323     }
324
325 }
326
327 /* send multiple bytes over SPI */
328 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
329 {
330     uint8_t i;
331
332     for(i=0;i<len;i++)
333     {
334         spi_transfer(dataout[i]);
335     }
336
337 }
338
339 /* Clocks only one byte into the given nrf24 register */
340 void nrf24_configRegister(uint8_t reg, uint8_t value)
341 {
342     nrf24_csn_digitalWrite(LOW);
343     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
344     spi_transfer(value);
345     nrf24_csn_digitalWrite(HIGH);
346 }
347
348 /* Read single register from nrf24 */
349 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
350 {
351     nrf24_csn_digitalWrite(LOW);
352     spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
353     nrf24_transferSync(value,value,len);
354     nrf24_csn_digitalWrite(HIGH);
355 }
356
```

```
357 /* Write to a single register of nrf24 */
358 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
359 {
360     nrf24_csn_digitalWrite(LOW);
361     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
362     nrf24_transmitSync(value,len);
363     nrf24_csn_digitalWrite(HIGH);
364 }
365
366 /* Check single register from nrf24 */
367 bool nrf24_checkRegister(uint8_t reg, uint8_t desiredValue, uint8_t len)
368 {
369     uint8_t registerValue;
370     nrf24_readRegister(reg,&registerValue,len);
371     if (registerValue==desiredValue) { return true; } else { return false; }
372 }
373
374 #define RF_DDR  DDRD
375 #define RF_PORT PORTD
376 #define RF_PIN  PIND
377
378 #define CE_CSN_DDR  DDRC
379 #define CE_CSN_PORT PORTC
380 #define CE_CSN_PIN  PINC
381
382 #define MISO_BIT_POS    0
383 #define MOSI_BIT_POS    1
384 #define SCK_BIT_POS     4
385
386 #define CE_BIT_POS      0
387 #define CSN_BIT_POS     1
388
389 #define set_bit(reg,bit) reg |= (1<<bit)
390 #define clr_bit(reg,bit) reg &= ~(1<<bit)
391 #define check_bit(reg,bit) (reg&(1<<bit))
392
393 /* ----- */
394
395 void nrf24_setupPins()
396 {
397     set_bit(CE_CSN_DDR, CE_BIT_POS); // CE output
398     set_bit(CE_CSN_DDR, CSN_BIT_POS); // CSN output
399
400     clr_bit(RF_DDR, MISO_BIT_POS); // MISO input
401     set_bit(RF_DDR, MOSI_BIT_POS); // MOSI output
402     set_bit(RF_DDR, SCK_BIT_POS); // SCK output
403 }
404 /* ----- */
405 void nrf24_ce_digitalWrite(uint8_t state)
406 {
407     if(state)
408     {
```

```
409     set_bit(CE_CSN_PORT, CE_BIT_POS);
410 }
411 else
412 {
413     clr_bit(CE_CSN_PORT, CE_BIT_POS);
414 }
415 }
416 /* -----
417 void nrf24_csn_digitalWrite(uint8_t state)
418 {
419     if(state)
420     {
421         set_bit(CE_CSN_PORT, CSN_BIT_POS);
422     }
423     else
424     {
425         clr_bit(CE_CSN_PORT, CSN_BIT_POS);
426     }
427 }
428 /* -----
429 void nrf24_sck_digitalWrite(uint8_t state)
430 {
431     if(state)
432     {
433         set_bit(RF_PORT, SCK_BIT_POS);
434     }
435     else
436     {
437         clr_bit(RF_PORT, SCK_BIT_POS);
438     }
439 }
440 /* -----
441 void nrf24_mosi_digitalWrite(uint8_t state)
442 {
443     if(state)
444     {
445         set_bit(RF_PORT, MOSI_BIT_POS);
446     }
447     else
448     {
449         clr_bit(RF_PORT, MOSI_BIT_POS);
450     }
451 }
452 /* -----
453 uint8_t nrf24_miso_digitalRead()
454 {
455     return check_bit(RF_PIN, MISO_BIT_POS);
456 }
457 /* -----
```

CODIGO DE FUENTE PLACA PRINCIPAL

```
1 #define F_CPU 16000000UL
2
3 #include <avr/io.h>
4 #include <util/delay.h>
5 #include <avr/interrupt.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <stdbool.h>
9 #include <stdint.h>
10
11 #include "UART_Bluetooth.h"
12 #include "nrf24.h"
13
14 void initIO();
15 void initRF();
16 char messageTest[] = "UART TESTING COMMANDS! \n";
17
18 int main(void)
19 {
20     sei(); // Interrupts on
21     initBluetoothUart();
22     initIO();
23     initRF();
24     setupReceiveMode();
25     while (1)
26     {
27         while(!commandAvailable);
28         processReceivedLine();
29         setupReceiveMode();
30     }
31 }
32
33
34 void initIO(){
35     /*
36         Input/Output pin initialization
37         1 : OUTPUT | 0 : INPUT | 0b76543210 Bit order
38         HC-05
39             TX      : PD0 (RX ATMEGA) | INPUT
40             RX      : PD1 (TX ATMEGA) | OUTPUT
41             KEY/ENABLE : PD2          | OUTPUT
42             STATE    : PC5          | INPUT
43         nRF24L01
44             CE      : PC0          | OUTPUT
45             CSN    : PC1          | OUTPUT
46             MISO   : PD0 (MSPIM MISO ATMEGA) | INPUT
47             MOSI   : PD1 (MSPIM MOSI ATMEGA) | OUTPUT
48             SCK    : PD4 (MSPIM XCK) | OUTPUT
49     */
50     DDRD = 0b11111110;
51     DDRB = 0b00101001;
52     DDRC = 0b11011111;
```

```
53     bit_clear(PORTD, BIT(2));
54 }
55
56 void initRF(){
57     uint8_t tx_address[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
58     uint8_t rx_address[5] = {0xD7,0xD7,0xD7,0xD7,0xD7};
59
60     nrf24_init();
61
62     /* Channel #112 , payload length: 32 */
63     nrf24_config(112,32);
64
65     /* Set the device addresses */
66     nrf24_tx_address(tx_address);
67     nrf24_rx_address(rx_address);
68 }
69
70
71
72
73
```

```
1
2
3 #ifndef COMMAND_HANDLER_H_
4 #define COMMAND_HANDLER_H_
5
6 #ifndef nullptr
7 #define nullptr ((void *)0)
8 #endif
9
10 #ifndef F_CPU
11 #define F_CPU           16000000UL
12 #endif
13
14 #include <stdbool.h>
15 #include <stdint.h>
16 #include <stdio.h>
17 #include <string.h>
18 #include <stdlib.h>
19 #include <avr/io.h>
20 #include <util/delay.h>
21
22 #define AVAILABLE_COMMANDS 11
23 #define COMMAND_BUFFER_SIZE 32
24 #define PARAMETER_BUFFER_SIZE 28
25
26 #ifndef BIT_MANIPULATION_MACRO
27 #define BIT_MANIPULATION_MACRO 1
28 #define bit_get(p,m) ((p) & (m))
29 #define bit_set(p,m) ((p) |= (m))
30 #define bit_clear(p,m) ((p) &= ~(m))
31 #define bit_flip(p,m) ((p) ^= (m))
32 #define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
33 #define BIT(x) (0x01 << (x))
34 #define LONGBIT(x) ((unsigned long)0x00000001 << (x))
35 #endif
36
37 typedef struct commandType {
38     const char *commandBase;
39     uint8_t nParameters;
40     void (*handlerFunction)();
41 } commandType;
42
43 void *parameter[3];
44 uint8_t *command_buffer;
45 extern bool initliazeMemory();
46 bool memoryInitialized;
47 extern void ROTATE_FORWARDS_HANDLE(), ROTATE_BACKWARDS_HANDLE(),
48     TURN_LED_ON_HANDLE(), TURN_LED_OFF_HANDLE(), TURN_RELAY_ON_HANDLE(),
49     TURN_RELAY_OFF_HANDLE();                                     ↵
50 extern void UART_TEST_HANDLER(), BUILT_IN_LED_TEST_HANDLER(),
51     TURN_EVERYTHING_ON_HANDLE(), TURN_EVERYTHING_OFF_HANDLE(), CALL_NURSE_HANDLE();    ↵
52
```

```
50 extern void composeCommand(void* output_buffer, commandType* commandT, void**  
    inputParameter);  
51 extern bool decomposeCommand(void* input_buffer, commandType* commandT, void**  
    outputParameter);  
52  
53  
54 #endif /* COMMAND_HANDLER_H */
```

```
1
2 #include "Command_Handler.h"
3 #include "UART_Bluetooth.h"
4 #include "nrf24.h"
5
6
7 const commandType availableCommand[AVAILABLE_COMMANDS] = {
8     { .commandBase = "ROTATE_FORWARDS", .nParameters = 3, .handlerFunction = &ROTATE_FORWARDS_HANDLE},
9     { .commandBase = "ROTATE_BACKWARDS", .nParameters = 1, .handlerFunction = &ROTATE_BACKWARDS_HANDLE},
10    { .commandBase = "TURN_LED_ON", .nParameters = 1, .handlerFunction = &TURN_LED_ON_HANDLE},
11    { .commandBase = "TURN_LED_OFF", .nParameters = 1, .handlerFunction = &TURN_LED_OFF_HANDLE},
12    { .commandBase = "TURN_RELAY_ON", .nParameters = 1, .handlerFunction = &TURN_RELAY_ON_HANDLE},
13    { .commandBase = "TURN_RELAY_OFF", .nParameters = 1, .handlerFunction = &TURN_RELAY_OFF_HANDLE},
14    { .commandBase = "UART_TEST", .nParameters = 0, .handlerFunction = &UART_TEST_HANDLER},
15    { .commandBase = "BUILT_IN_LED_TEST", .nParameters = 0, .handlerFunction = &BUILT_IN_LED_TEST_HANDLER},
16    { .commandBase = "TURN_EVERYTHING_ON", .nParameters = 0, .handlerFunction = &TURN_EVERYTHING_ON_HANDLE},
17    { .commandBase = "TURN_EVERYTHING_OFF", .nParameters = 0, .handlerFunction = &TURN_EVERYTHING_OFF_HANDLE},
18    { .commandBase = "CALL_NURSE", .nParameters = 0, .handlerFunction = &CALL_NURSE_HANDLE}
19 };
20
21 bool initliazeMemory(){
22     if(memoryInitialized) return false;
23     parameter[0] = (void*)calloc(28,1);
24     parameter[1] = (void*)calloc(28,1);
25     parameter[2] = (void*)calloc(28,1);
26     command_buffer = (uint8_t*)calloc(32,1);
27     if(parameter[0]==nullptr||parameter[1]==nullptr||parameter[2]==nullptr||
28         command_buffer==nullptr) return false;
29     memoryInitialized = true;
30     return true;
31 }
32
33 void composeCommand(void* output_buffer, commandType* commandT, void** inputParameter){
34     strcpy(output_buffer, commandT->commandBase);
35     char* startParamPTR = (char*)(output_buffer+strlen(commandT->commandBase));
36     char* endParamPTR = (char*)(startParamPTR+1+strlen(*inputParameter));
37
38     for (uint8_t index = 0; index < commandT->nParameters; index++){
39         *startParamPTR='[';
```

```
40         strcpy(startParamPTR+1, *inputParameter);
41         *endParamPTR=']';
42         startParamPTR=(endParamPTR+1);
43         if (index!=(commandT->nParameters-1)){
44             inputParameter++;
45             uint8_t len = strlen(*inputParameter);
46             endParamPTR = (char*)(startParamPTR+len+1);
47         }
48     }
49     *startParamPTR='\0';
50 }
51
52 bool decomposeCommand(void* input_buffer, commandType* commandT, void** outputParameter){
53
54     for (uint8_t index = 0; index < AVAILABLE_COMMANDS; index++){
55         if (memmem(input_buffer, COMMAND_BUFFER_SIZE, availableCommand
56 [index].commandBase, strlen(availableCommand[index].commandBase))!=
57         nullptr)
58         {
59             *commandT = availableCommand[index]; break;
60         }
61         else if (index==(AVAILABLE_COMMANDS-1)) { return false;}
62     }
63     for (uint8_t x = 0; x < commandT->nParameters; x++){
64         uint8_t* startNumPTR = memchr(input_buffer, '[', COMMAND_BUFFER_SIZE);
65         uint8_t* endNumPTR = memchr(input_buffer, ']', COMMAND_BUFFER_SIZE);
66         if (startNumPTR==nullptr||endNumPTR==nullptr) { if(x==0) return false; }
67         (*startNumPTR) = 0x20;
68         (*endNumPTR) = 0x20;
69         startNumPTR++;
70         uint32_t bytes = ((endNumPTR)) - ((startNumPTR));
71         if (bytes>PARAMETER_BUFFER_SIZE) return false;
72         memcpy(outputParameter[x], startNumPTR, bytes);
73     }
74     return true;
75 }
76
77 void ROTATE_FORWARDS_HANDLE() {}
78
79 void ROTATE_BACKWARDS_HANDLE() {}
80
81 void TURN_LED_ON_HANDLE() {}
82
83 void TURN_LED_OFF_HANDLE() {}
84
85 void TURN_RELAY_ON_HANDLE() {
86     composeCommand(command_buffer, &availableCommand[4], parameter);
87 }
```

```
88     nrf24_send(command_buffer);
89     while(nrf24_isSending());
90
91     uint8_t messageStatus = nrf24_lastMessageStatus();
92     if(messageStatus == NRF24_TRANSISSTION_OK) { transmitMessageSync("Successful RF transmission! \n", 29); }
93     else if(messageStatus == NRF24_MESSAGE_LOST) { transmitMessageSync("Failure on RF transmission! \n", 29); }
94
95     uint8_t retransmissionCount = nrf24_retransmissionCount();
96     char* retransmissionString = malloc(32);
97     sprintf(retransmissionString, "Retransmission count: %d \n",
98             retransmissionCount);
99     transmitMessageSync(retransmissionString, strlen(retransmissionString));
100    free(retransmissionString);
101 }
102 void TURN_RELAY_OFF_HANDLE() {
103     composeCommand(command_buffer, &availableCommand[5], parameter);
104
105     nrf24_send(command_buffer);
106     while(nrf24_isSending());
107
108     uint8_t messageStatus = nrf24_lastMessageStatus();
109     if(messageStatus == NRF24_TRANSISSTION_OK) { transmitMessageSync("Successful RF transmission! \n", 29); }
110     else if(messageStatus == NRF24_MESSAGE_LOST) { transmitMessageSync("Failure on RF transmission! \n", 29); }
111
112     uint8_t retransmissionCount = nrf24_retransmissionCount();
113     char* retransmissionString = malloc(32);
114     sprintf(retransmissionString, "Retransmission count: %d \n",
115             retransmissionCount);
116     transmitMessageSync(retransmissionString, strlen(retransmissionString));
117     free(retransmissionString);
118 }
119 void UART_TEST_HANDLER() {
120     transmitMessageSync("Successful UART transmission!\n", 30);
121 }
122
123 void BUILT_IN_LED_TEST_HANDLER(){
124     for (uint8_t x = 0; x < 8; x++) {
125         bit_flip(PORTD, BIT(7));
126         bit_flip(PORTB, BIT(0));
127         _delay_ms(250);
128     }
129     bit_clear(PORTD, BIT(7));
130     bit_clear(PORTB, BIT(0));
131 }
132
133 void TURN_EVERYTHING_ON_HANDLE(){
```

```
134     composeCommand(command_buffer, &availableCommand[8], parameter);
135
136     nrf24_send(command_buffer);
137     while(nrf24_isSending());
138
139     uint8_t messageStatus = nrf24_lastMessageStatus();
140     if(messageStatus == NRF24_TRANSMISSON_OK) { transmitMessageSync("Successful
141         RF transmission! \n", 29); }
142     else if(messageStatus == NRF24_MESSAGE_LOST) { transmitMessageSync("Failure
143         on RF transmission! \n", 29); }
144
145     uint8_t retransmissionCount = nrf24_retransmissionCount();
146     char* retransmissionString = malloc(32);
147     sprintf(retransmissionString, "Retransmission count: %d \n",
148             retransmissionCount);
149     transmitMessageSync(retransmissionString, strlen(retransmissionString));
150     free(retransmissionString);
151 }
152
153 void TURN_EVERYTHING_OFF_HANDLE(){
154     composeCommand(command_buffer, &availableCommand[9], parameter);
155
156     nrf24_send(command_buffer);
157     while(nrf24_isSending());
158
159     uint8_t messageStatus = nrf24_lastMessageStatus();
160     if(messageStatus == NRF24_TRANSMISSON_OK) { transmitMessageSync("Successful
161         RF transmission! \n", 29); }
162     else if(messageStatus == NRF24_MESSAGE_LOST) { transmitMessageSync("Failure
163         on RF transmission! \n", 29); }
164
165     uint8_t retransmissionCount = nrf24_retransmissionCount();
166     char* retransmissionString = malloc(32);
167     sprintf(retransmissionString, "Retransmission count: %d \n",
168             retransmissionCount);
169     transmitMessageSync(retransmissionString, strlen(retransmissionString));
170     free(retransmissionString);
171 }
172
173 void CALL_NURSE_HANDLE(){
174     composeCommand(command_buffer, &availableCommand[10], parameter);
175
176     nrf24_send(command_buffer);
177     while(nrf24_isSending());
178
179     uint8_t messageStatus = nrf24_lastMessageStatus();
180     if(messageStatus == NRF24_TRANSMISSON_OK) { transmitMessageSync("Successful
181         RF transmission! \n", 29); }
182     else if(messageStatus == NRF24_MESSAGE_LOST) { transmitMessageSync("Failure
183         on RF transmission! \n", 29); }
184
185     uint8_t retransmissionCount = nrf24_retransmissionCount();
```

```
178     char* retransmissionString = malloc(32);
179     sprintf(retransmissionString, "Retransmission count: %d \n",
180             retransmissionCount);
180     transmitMessageSync(retransmissionString, strlen(retransmissionString));
181     free(retransmissionString);
182 }
183
184
```

```
1 #ifndef NRF24
2 #define NRF24
3
4 #include "nRF24L01_Definitions.h"
5 #include <stdint.h>
6
7 #define LOW 0
8 #define HIGH 1
9
10#define nrf24_ADDR_LEN 5
11#define nrf24_CONFIG ((1<<EN_CRC)|(0<<CRCO))
12
13#define NRF24_TRANSMISSON_OK 0
14#define NRF24_MESSAGE_LOST 1
15
16 void nrf24_init();
17 void nrf24_rx_address(uint8_t* adr);
18 void nrf24_tx_address(uint8_t* adr);
19 void nrf24_config(uint8_t channel, uint8_t pay_length);
20
21 uint8_t nrf24_dataReady();
22 uint8_t nrf24_isSending();
23 uint8_t nrf24_getStatus();
24 uint8_t nrf24_rxFifoEmpty();
25
26 void nrf24_send(uint8_t* value);
27 void nrf24_getData(uint8_t* data);
28
29 uint8_t nrf24_payloadLength();
30
31 uint8_t nrf24_lastMessageStatus();
32 uint8_t nrf24_retransmissionCount();
33
34 uint8_t nrf24_payload_length();
35
36 void nrf24_powerUpRx();
37 void nrf24_powerUpTx();
38 void nrf24_powerDown();
39
40 uint8_t spi_transfer(uint8_t tx);
41 void nrf24_transmitSync(uint8_t* dataout, uint8_t len);
42 void nrf24_transferSync(uint8_t* dataout, uint8_t* datain, uint8_t len);
43 void nrf24_configRegister(uint8_t reg, uint8_t value);
44 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len);
45 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len);
46
47 extern void nrf24_setupPins();
48
49 extern void nrf24_ce_digitalWrite(uint8_t state);
50
51 extern void nrf24_csn_digitalWrite(uint8_t state);
52
```

```
53 extern void nrf24_sck_digitalWrite(uint8_t state);
54
55 extern void nrf24_mosi_digitalWrite(uint8_t state);
56
57 extern uint8_t nrf24_miso_digitalRead();
58
59 #endif
60
```

```
1  
2 #define UCPHA0 1  
3 #define F_CPU 8000000UL  
4 #define BAUD_RATE 9600UL  
5 #define UBRR_VALUE ((F_CPU)/(2UL*BAUD_RATE))-1  
6  
7 #include "nrf24.h"  
8 #include <avr/io.h>  
9  
10 uint8_t payload_len;  
11  
12 void nrf24_init()  
13 {  
14     nrf24_setupPins();  
15     nrf24_ce_digitalWrite(LOW);  
16     nrf24_csn_digitalWrite(HIGH);  
17 }  
18  
19 void nrf24_config(uint8_t channel, uint8_t pay_length)  
20 {  
21     /* Use static payload length ... */  
22     payload_len = pay_length;  
23  
24     // Set RF channel  
25     nrf24_configRegister(RF_CH,channel);  
26  
27     // Set length of incoming payload  
28     nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...  
29     nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe  
30     nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used  
31     nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used  
32     nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used  
33     nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used  
34  
35     // 1 Mbps, TX gain: 0dbm  
36     nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));  
37  
38     // CRC enable, 1 byte CRC length  
39     nrf24_configRegister(CONFIG,nrf24_CONFIG);  
40  
41     // Auto Acknowledgment  
42     nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|  
        (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));  
43  
44     // Enable RX addresses  
45     nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|  
        (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));  
46  
47     // Auto retransmit delay: 1000 us and Up to 15 retransmit trials  
48     nrf24_configRegister(SETPUP_RETR,(0x04<<ARD)|(0x0F<<ARC));  
49  
50     // Dynamic length configurations: No dynamic length
```

```
51     nrf24_configRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|  
      (0<<DPL_P4)|(0<<DPL_P5));  
52  
53     // Start listening  
54     nrf24_powerUpRx();  
55 }  
56  
57 /* Set the RX address */  
58 void nrf24_rx_address(uint8_t * adr)  
59 {  
60     nrf24_ce_digitalWrite(LOW);  
61     nrf24_writeRegister(RX_ADDR_P1,adr,nrf24_ADDR_LEN);  
62     nrf24_ce_digitalWrite(HIGH);  
63 }  
64  
65 /* Returns the payload length */  
66 uint8_t nrf24_payload_length()  
67 {  
68     return payload_len;  
69 }  
70  
71 /* Set the TX address */  
72 void nrf24_tx_address(uint8_t* adr)  
73 {  
74     /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */  
75     nrf24_writeRegister(RX_ADDR_P0,adr,nrf24_ADDR_LEN);  
76     nrf24_writeRegister(TX_ADDR,adr,nrf24_ADDR_LEN);  
77 }  
78  
79 /* Checks if data is available for reading */  
80 /* Returns 1 if data is ready ... */  
81 uint8_t nrf24_dataReady()  
82 {  
83     // See note in getData() function - just checking RX_DR isn't good enough  
84     uint8_t status = nrf24_getStatus();  
85  
86     // We can short circuit on RX_DR, but if it's not set, we still need  
87     // to check the FIFO for any pending packets  
88     if ( status & (1 << RX_DR) )  
89     {  
90         return 1;  
91     }  
92  
93     return !nrf24_rx_fifoEmpty();;  
94 }  
95  
96 /* Checks if receive FIFO is empty or not */  
97 uint8_t nrf24_rx_fifoEmpty()  
98 {  
99     uint8_t fifoStatus;  
100    nrf24_readRegister(FIFO_STATUS,&fifoStatus,1);
```

```
102     return (fifoStatus & (1 << RX_EMPTY));
103 }
104
105
106 /* Returns the length of data waiting in the RX fifo */
107 uint8_t nrf24_payloadLength()
108 {
109     uint8_t status;
110     nrf24_csn_digitalWrite(LOW);
111     spi_transfer(R_RX_PL_WID);
112     status = spi_transfer(0x00);
113     nrf24_csn_digitalWrite(HIGH);
114     return status;
115 }
116
117 /* Reads payload bytes into data array */
118 void nrf24_getData(uint8_t* data)
119 {
120     /* Pull down chip select */
121     nrf24_csn_digitalWrite(LOW);
122
123     /* Send cmd to read rx payload */
124     spi_transfer( R_RX_PAYLOAD );
125
126     /* Read payload */
127     nrf24_transferSync(data,data,payload_len);
128
129     /* Pull up chip select */
130     nrf24_csn_digitalWrite(HIGH);
131
132     /* Reset status register */
133     nrf24_configRegister(STATUS,(1<<RX_DR));
134 }
135
136 /* Returns the number of retransmissions occured for the last message */
137 uint8_t nrf24_retransmissionCount()
138 {
139     uint8_t rv;
140     nrf24_readRegister(OBSERVE_TX,&rv,1);
141     rv = rv & 0x0F;
142     return rv;
143 }
144
145 // Sends a data package to the default address. Be sure to send the correct
146 // amount of bytes as configured as payload on the receiver.
147 void nrf24_send(uint8_t* value)
148 {
149     /* Go to Standby-I first */
150     nrf24_ce_digitalWrite(LOW);
151
152     /* Set to transmitter mode , Power up if needed */
153     nrf24_powerUpTx();
```

```
154
155     /* Do we really need to flush TX fifo each time ? */
156     #if 1
157         /* Pull down chip select */
158         nrf24_csn_digitalWrite(LOW);
159
160         /* Write cmd to flush transmit FIFO */
161         spi_transfer(FLUSH_TX);
162
163         /* Pull up chip select */
164         nrf24_csn_digitalWrite(HIGH);
165     #endif
166
167     /* Pull down chip select */
168     nrf24_csn_digitalWrite(LOW);
169
170     /* Write cmd to write payload */
171     spi_transfer(W_TX_PAYLOAD);
172
173     /* Write payload */
174     nrf24_transmitSync(value,payload_len);
175
176     /* Pull up chip select */
177     nrf24_csn_digitalWrite(HIGH);
178
179     /* Start the transmission */
180     nrf24_ce_digitalWrite(HIGH);
181 }
182
183 uint8_t nrf24_isSending()
184 {
185     uint8_t status;
186
187     /* read the current status */
188     status = nrf24_getStatus();
189
190     /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
191     if((status & ((1 << TX_DS) | (1 << MAX_RT))))
192     {
193         return 0; /* false */
194     }
195
196     return 1; /* true */
197 }
198
199
200 uint8_t nrf24_getStatus()
201 {
202     uint8_t rv;
203     nrf24_csn_digitalWrite(LOW);
204     rv = spi_transfer(NOP);
205     nrf24_csn_digitalWrite(HIGH);
```

```
206     return rv;
207 }
208
209 uint8_t nrf24_lastMessageStatus()
210 {
211     uint8_t rv;
212
213     rv = nrf24_getStatus();
214
215     /* Transmission went OK */
216     if((rv & ((1 << TX_DS))))
217     {
218         return NRF24_TRANSMISSION_OK;
219     }
220     /* Maximum retransmission count is reached */
221     /* Last message probably went missing ... */
222     else if((rv & ((1 << MAX_RT))))
223     {
224         return NRF24_MESSAGE_LOST;
225     }
226     /* Probably still sending ... */
227     else
228     {
229         return 0xFF;
230     }
231 }
232
233 void nrf24_powerUpRx()
234 {
235     nrf24_csn_digitalWrite(LOW);
236     spi_transfer(FETCH_RX);
237     nrf24_csn_digitalWrite(HIGH);
238
239     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
240
241     nrf24_ce_digitalWrite(LOW);
242     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
243     nrf24_ce_digitalWrite(HIGH);
244 }
245
246 void nrf24_powerUpTx()
247 {
248     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
249
250     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
251 }
252
253 void nrf24_powerDown()
254 {
255     nrf24_ce_digitalWrite(LOW);
256     nrf24_configRegister(CONFIG,nrf24_CONFIG);
257 }
```

```
258
259 uint8_t spi_transfer(uint8_t tx)
260 {
261     uint8_t i = 0;
262     uint8_t rx = 0;
263
264     nrf24_sck_digitalWrite(LOW);
265
266     for(i=0;i<8;i++)
267     {
268
269         if(tx & (1<<(7-i)))
270         {
271             nrf24_mosi_digitalWrite(HIGH);
272         }
273         else
274         {
275             nrf24_mosi_digitalWrite(LOW);
276         }
277
278         nrf24_sck_digitalWrite(HIGH);
279
280         rx = rx << 1;
281         if(nrf24_miso_digitalRead())
282         {
283             rx |= 0x01;
284         }
285
286         nrf24_sck_digitalWrite(LOW);
287     }
288
289     return rx;
290 }
291
292 /* send and receive multiple bytes over SPI */
293 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
294 {
295     uint8_t i;
296
297     for(i=0;i<len;i++)
298     {
299         datain[i] = spi_transfer(dataout[i]);
300     }
301
302 }
303
304 /* send multiple bytes over SPI */
305 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
306 {
307     uint8_t i;
308
309 }
```

```
310     for(i=0;i<len;i++)
311     {
312         spi_transfer(dataout[i]);
313     }
314 }
315 }
316
317 /* Clocks only one byte into the given nrf24 register */
318 void nrf24_configRegister(uint8_t reg, uint8_t value)
319 {
320     nrf24_csn_digitalWrite(LOW);
321     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
322     spi_transfer(value);
323     nrf24_csn_digitalWrite(HIGH);
324 }
325
326 /* Read single register from nrf24 */
327 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
328 {
329     nrf24_csn_digitalWrite(LOW);
330     spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
331     nrf24_transmitSync(value,value,len);
332     nrf24_csn_digitalWrite(HIGH);
333 }
334
335 /* Write to a single register of nrf24 */
336 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
337 {
338     nrf24_csn_digitalWrite(LOW);
339     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
340     nrf24_transmitSync(value,len);
341     nrf24_csn_digitalWrite(HIGH);
342 }
343
344 #define RF_DDR DDRC
345 #define RF_PORT PORTC
346 #define RF_PIN PINC
347
348 #define set_bit(reg,bit) reg |= (1<<bit)
349 #define clr_bit(reg,bit) reg &= ~(1<<bit)
350 #define check_bit(reg,bit) (reg&(1<<bit))
351
352 /* ----- */
353
354 void nrf24_setupPins()
355 {
356     set_bit(RF_DDR,0); // CE output
357     set_bit(RF_DDR,1); // CSN output
358     set_bit(RF_DDR,2); // SCK output
359     set_bit(RF_DDR,3); // MOSI output
360     clr_bit(RF_DDR,4); // MISO input
361 }
```

```
362 /* -----
363 void nrf24_ce_digitalWrite(uint8_t state)
364 {
365     if(state)
366     {
367         set_bit(RF_PORT,0);
368     }
369     else
370     {
371         clr_bit(RF_PORT,0);
372     }
373 }
374 /* -----
375 void nrf24_csn_digitalWrite(uint8_t state)
376 {
377     if(state)
378     {
379         set_bit(RF_PORT,1);
380     }
381     else
382     {
383         clr_bit(RF_PORT,1);
384     }
385 }
386 /* -----
387 void nrf24_sck_digitalWrite(uint8_t state)
388 {
389     if(state)
390     {
391         set_bit(RF_PORT,2);
392     }
393     else
394     {
395         clr_bit(RF_PORT,2);
396     }
397 }
398 /* -----
399 void nrf24_mosi_digitalWrite(uint8_t state)
400 {
401     if(state)
402     {
403         set_bit(RF_PORT,3);
404     }
405     else
406     {
407         clr_bit(RF_PORT,3);
408     }
409 }
410 /* -----
411 uint8_t nrf24_miso_digitalRead()
412 {
413     return check_bit(RF_PIN,4);
```

```
414 }
415 /* ----- */
416
```

```
1
2
3 #ifndef UART_BLUETOOTH_H_
4 #define UART_BLUETOOTH_H_
5
6
7 #include <stdbool.h>
8 #include <stdint.h>
9
10 #ifndef F_CPU
11 #define F_CPU          16000000UL
12 #endif
13
14 #ifndef BAUD
15 #define BAUD          9600
16 #endif
17
18 #ifndef BRC
19 #define BRC           F_CPU/8/BAUD-1
20 #endif
21
22 #ifndef nullptr
23 #define nullptr        nullptr ((void*)0)
24 #endif
25
26 #define uartBufferSize      32
27 #define uartEndMsgChar     '$'
28 #define uartCarriageReturnChar '/'
29
30 #ifndef BIT_MANIPULATION_MACRO
31 #define BIT_MANIPULATION_MACRO 1
32 #define bit_get(p,m) ((p) & (m))
33 #define bit_set(p,m) ((p) |= (m))
34 #define bit_clear(p,m) ((p) &= ~(m))
35 #define bit_flip(p,m) ((p) ^= (m))
36 #define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
37 #define BIT(x) (0x01 << (x))
38 #define LONGBIT(x) ((unsigned long)0x00000001 << (x))
39 #endif
40
41
42 extern bool commandAvailable;
43
44 extern void initBluetoothUart();
45 extern void transmitMessage(uint8_t* message, uint8_t length);
46 extern void transmitMessageSync(uint8_t* message, uint8_t length);
47 extern bool transmissionState();
48 extern void setupReceiveMode();
49 extern void processReceivedLine();
50 extern void disableUART();
51
52
```

53
54 #endif /* UART_BLUETOOTH_H */

```
1
2
3 #include "UART_Bluetooth.h"
4 #include <avr/io.h>
5 #include <avr/interrupt.h>
6 #include "Command_Handler.h"
7 #include <stdlib.h>
8 #include <string.h>
9
10 uint8_t* uartBufferPos;
11 uint8_t* uartTxMessageEnd;
12 bool commandAvailable;
13
14 void initBluetoothUart(){
15     // UART Initialization : 8-bit : No parity bit : 1 stop bit
16     UBRR0H = (BRC >> 8); UBRR0L = BRC; // UART BAUDRATE
17     UCSR0A |= (1 << U2X0); // DOUBLE UART SPEED
18     UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // 8-BIT CHARACTER SIZE
19
20     // Setup UART buffer
21     initliazeMemory();
22     uartBufferPos = command_buffer;
23 }
24
25 void transmitMessage(uint8_t* message, uint8_t length){
26     while (!(UCSR0A & (1<<UDRE0)));
27     uartBufferPos = command_buffer;
28     uartTxMessageEnd = (command_buffer+length);
29     memcpy(command_buffer, message, length);
30     UCSR0A |= (1<<TXC0) | (1<<RXC0);
31     UCSR0B |= (1<<TXEN0) | (1<<TXCIE0);
32     UCSR0B &=~(1<<RXEN0) &~(1<<RXCIE0);
33
34     uartBufferPos++;
35     UDR0 = *(command_buffer);
36 }
37
38 void transmitMessageSync(uint8_t* message, uint8_t length){
39     while (!(UCSR0A & (1<<UDRE0)));
40     uartBufferPos = command_buffer;
41     uartTxMessageEnd = (command_buffer+length);
42     memcpy(command_buffer, message, length);
43     UCSR0A |= (1<<TXC0) | (1<<RXC0);
44     UCSR0B |= (1<<TXEN0) | (1<<TXCIE0);
45     UCSR0B &=~(1<<RXEN0) &~(1<<RXCIE0);
46
47     uartBufferPos++;
48     UDR0 = *(command_buffer);
49
50     while (transmissionState());
51
52 }
```

```
53
54     bool transmissionState(){
55         // True : Currently transmitting | False : Transmission finished
56         if (uartBufferPos!=uartTxMessageEnd)
57         {
58             return true;
59         }
60         else
61         {
62             return false;
63         }
64     }
65
66
67     void setupReceiveMode(){
68         while (!(UCSR0A & (1<<UDRE0)));
69         uartBufferPos = command_buffer;
70
71         UCSR0A |= (1<<RXC0) | (1<<TXC0);
72         UCSR0B &= ~(1<<TXEN0) &~(1<<TXCIE0);
73         UCSR0B |= (1<<RXEN0) | (1<<RXCIE0);
74     }
75
76     void processReceivedLine(){
77         commandAvailable = false;
78
79         commandType currentCommand;
80         bool success = decomposeCommand(command_buffer, &currentCommand, parameter);
81         if(success) currentCommand.handlerFunction();
82     }
83
84     void disableUART(){
85         UCSR0B &= ~(1<<TXEN0) &~(1<<TXCIE0);
86         UCSR0B &= ~(1<<RXEN0) &~(1<<RXCIE0);
87     }
88
89     ISR(USART_TX_vect){
90         if (uartBufferPos!=uartTxMessageEnd){
91             UDR0 = *uartBufferPos;
92             uartBufferPos++;
93         }
94     }
95
96     ISR(USART_RX_vect){
97         if(uartBufferPos!=(command_buffer+uartBufferSize)) {
98             *uartBufferPos=UDR0;
99             if (*uartBufferPos!=uartEndMsgChar) {
100                 if(*uartBufferPos!=uartCarriageReturnChar) {uartBufferPos++;} else    ↵
101                 { uartBufferPos = command_buffer; }
102             }
103             else { disableUART(); commandAvailable = true; }
104         } else {uartBufferPos = command_buffer;}
105     }
```


CODIGO DE FUENTE APLICACIÓN ANDROID

```

1 package com.example.gui_design.bluetoothActivities;
2
3 import android.graphics.Color;
4 import android.graphics.PorterDuff;
5 import android.graphics.drawable.ColorDrawable;
6 import android.graphics.drawable.Drawable;
7 import android.support.v7.widget.RecyclerView;
8 import android.view.LayoutInflater;
9 import android.view.View;
10 import android.view.ViewGroup;
11 import android.widget.ImageView;
12 import android.widget.TextView;
13
14 import com.example.gui_design.Application;
15 import com.example.gui_design.R;
16
17 import java.util.List;
18
19 public class DeviceListAdapter extends RecyclerView.Adapter<DeviceListAdapter.ViewHolder> {
20
21     private List<AvailableDevice> deviceList;
22
23     public class ViewHolder extends RecyclerView.ViewHolder {
24         public TextView textView_device_name;
25         public ImageView imgView_device_state_icon;
26
27         public ViewHolder(View view) {
28             super(view);
29             textView_device_name = (TextView) view.findViewById(R.id.textView_bluetooth_device_name);
30             imgView_device_state_icon = (ImageView) view.findViewById(R.id.
31             imageView_bluetooth_device_state_icon);
32         }
33     }
34
35     public DeviceListAdapter(List<AvailableDevice> deviceList) {
36         this.deviceList = deviceList;
37     }
38
39     @Override
40     public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
41         View itemView = LayoutInflater.from(parent.getContext())
42             .inflate(R.layout.bluetooth_list_layout_style, parent, false);
43
44         return new ViewHolder(itemView);
45     }
46
47     @Override
48     public void onBindViewHolder(ViewHolder holder, int position) {
49         AvailableDevice device = deviceList.get(position);
50         holder.textView_device_name.setText(device.getDeviceName());
51         holder.textView_device_name.setTextAppearance(device.getTextAppearance());
52
53         int iconID = device.getDeviceStateIconID();
54
55         if (iconID!=0) {
56             Drawable icon = Application.getContext().getDrawable(iconID);
57             icon.setColorFilter(Application.getContext().getColor(R.color.colorPrimary), PorterDuff.
58             Mode.SRC_ATOP);
59             holder.imgView_device_state_icon.setImageDrawable(icon);
60         } else {
61             holder.imgView_device_state_icon.clearColorFilter();
62             Drawable transparentDrawable = new ColorDrawable(Color.TRANSPARENT);
63             holder.imgView_device_state_icon.setImageDrawable(transparentDrawable);
64         }
65     }
66
67     @Override
68     public int getItemCount() {
69         return deviceList.size();
70     }
71

```

```

1 package com.example.gui_design;
2
3 import android.content.Intent;
4 import android.os.Bundle;
5 import android.speech.RecognitionListener;
6 import android.speech.RecognizerIntent;
7 import android.speech.SpeechRecognizer;
8 import android.view.View;
9
10 import com.example.gui_design.bluetoothActivities.BluetoothConnection;
11
12 import org.apache.commons.lang3.StringUtils;
13
14 import java.text.Normalizer;
15 import java.util.ArrayList;
16 import java.util.Arrays;
17
18 public class GoogleSpeechRecognizer extends MainActivity implements RecognitionListener {
19
20     private static SpeechRecognizer speechRecognizer = null;
21
22     public static void startSpeechListening(){
23         String language = "es-ES";
24         Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
25         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,language);
26         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, language);
27         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_PREFERENCE, language);
28         intent.putExtra(RecognizerIntent.EXTRA_ONLY_RETURN_LANGUAGE_PREFERENCE, language);
29         intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE, Application.getActivity().
getPackageName());
30         intent.putExtra(RecognizerIntent.EXTRA_PARTIAL_RESULTS,true);
31         speechRecognizer.startListening(intent);
32     }
33
34     public static void setupRecognizer(){
35         speechRecognizer = SpeechRecognizer.createSpeechRecognizer(Application.getContext());
36         speechRecognizer.setRecognitionListener(new GoogleSpeechRecognizer());
37     }
38
39     public static void shutdownRecognizer(){
40         speechRecognizer.stopListening();
41         speechRecognizer.cancel();
42         speechRecognizer.destroy();
43         speechRecognizer = null;
44         System.gc();
45     }
46
47     public static void forceStopRecognizer(){
48         speechRecognizer.stopListening();
49         speechRecognizer.cancel();
50
51         PocketsphinxListener.startListening();
52     }
53
54
55     public static void analizeListenerResults(ArrayList<String> data, float[] confidenceScore){
56         outerloop:
57         for (int i = 0; i < data.size(); i++) {
58             String hypothesis = Normalizer.normalize((String) data.get(i)).toLowerCase(), Normalizer.
Form.NFKD;
59             String[] hypothesisSplit = hypothesis.split(" ");
60
61             if (Arrays.asList(hypothesisSplit).contains("encender") && Arrays.asList(hypothesisSplit).
contains("velador")) {
62                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_ON, 0);
63                 break outerloop;
64             }
65
66             if (Arrays.asList(hypothesisSplit).contains("encender") && Arrays.asList(hypothesisSplit).
contains("estufa")) {
67                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_ON, 1);
68                 break outerloop;
69             }
70
71             if (Arrays.asList(hypothesisSplit).contains("encender") && Arrays.asList(hypothesisSplit).
contains("ventilador")) {
72                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_ON, 2);

```

```

73             break outerloop;
74         }
75
76         if (Arrays.asList(hypothesisSplit).contains("encender") && Arrays.asList(hypothesisSplit)
77 .contains("luz")) {
78             BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_ON, 3);
79             break outerloop;
80         }
81
82         if (Arrays.asList(hypothesisSplit).contains("apagar") && Arrays.asList(hypothesisSplit).
83 contains("velador")) {
84             BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_OFF, 0)
85 ;
86             break outerloop;
87         }
88
89         if (Arrays.asList(hypothesisSplit).contains("apagar") && Arrays.asList(hypothesisSplit).
90 contains("estufa")) {
91             BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_OFF, 1)
92 ;
93             break outerloop;
94         }
95
96         if (Arrays.asList(hypothesisSplit).contains("apagar") && Arrays.asList(hypothesisSplit).
97 contains("ventilador")) {
98             BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_RELAY_OFF, 2)
99 ;
100            break outerloop;
101
102            if (Arrays.asList(hypothesisSplit).contains("encender") && Arrays.asList(hypothesisSplit)
103 .contains("todo")) {
104                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.TURN_EVERYTHING_ON
105 , 0);
106                 break outerloop;
107             }
108
109             if (Arrays.asList(hypothesisSplit).contains("apagar") && Arrays.asList(hypothesisSplit).
110 contains("todo")) {
111                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.
112 TURN_EVERYTHING_OFF, 0);
113                 break outerloop;
114             }
115
116             if (Arrays.asList(hypothesisSplit).contains("llamar") && Arrays.asList(hypothesisSplit).
117 contains("enfermera")) {
118                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.CALL_NURSE, 0);
119                 break outerloop;
120             }
121
122             if (Arrays.asList(hypothesisSplit).contains("encender") && Arrays.asList(hypothesisSplit)
123 .contains("rele")) {
124                 for (String word : hypothesisSplit) {
125                     if (StringUtils.isNumeric(word) == true) {
126                         int number = Integer.parseInt(word);
127                         BluetoothConnection.composeAndSend(BluetoothConnection.commandType.
128 TURN_RELAY_ON, number);
129                         break outerloop;
130                     }
131                 }
132             if (Arrays.asList(hypothesisSplit).contains("apagar") && Arrays.asList(hypothesisSplit).

```

```

132 contains("rele")) {
133         for (String word : hypothesisSplit) {
134             if (StringUtils.isNumeric(word) == true) {
135                 int number = Integer.parseInt(word);
136                 BluetoothConnection.composeAndSend(BluetoothConnection.commandType.
137                     TURN_RELAY_OFF, number);
138             }
139         }
140     }
141 }
142 }
143 }
144
145 @Override
146 public void onReadyForSpeech(Bundle params) {
147     MicrophoneHandler.setState(MicrophoneHandler.MicrophoneState.LISTENING_COMMAND);
148 }
149
150 @Override
151 public void onBeginningOfSpeech() {
152 }
153
154 @Override
155 public void onRmsChanged(float rmsdB) {
156 }
157
158 @Override
159 public void onBufferReceived(byte[] buffer) {
160 }
161
162 @Override
163 public void onEndOfSpeech() {
164     Application.toastMessage("End of speech");
165 }
166
167 @Override
168 public void onError(int error) {
169     Application.toastMessage("On error" + error);
170     MicrophoneHandler.setState(MicrophoneHandler.MicrophoneState.LISTENING_KEYWORD);
171     PocketsphinxListener.startListening();
172 }
173
174 @Override
175 public void onResults(Bundle results) {
176     String str = new String();
177     ArrayList<String> resultsArrayList = results.getStringArrayList(SpeechRecognizer.
178         RESULTS_RECOGNITION);
179     float[] confidenceScore = results.getFloatArray(SpeechRecognizer.CONFIDENCE_SCORES);
180     analizeListenerResults(resultsArrayList, confidenceScore);
181
182     StringBuilder builder = new StringBuilder();
183     for (String line : resultsArrayList) {
184         builder.append(line + "\n");
185     }
186
187     MainActivity.textView_speechRecognitionResults.setText(builder.toString());
188     MainActivity.textView_speechRecognitionResults.setVisibility(View.VISIBLE);
189
190     MicrophoneHandler.setState(MicrophoneHandler.MicrophoneState.LISTENING_KEYWORD);
191
192     PocketsphinxListener.startListening();
193 }
194
195 @Override
196 public void onPartialResults(Bundle partialResults) {
197     String str = new String();
198     ArrayList<String> resultsArrayList = partialResults.getStringArrayList(SpeechRecognizer.
199         RESULTS_RECOGNITION);
200     float[] confidenceScore = partialResults.getFloatArray(SpeechRecognizer.CONFIDENCE_SCORES);
201
202     StringBuilder builder = new StringBuilder();
203

```

```
206     for (String line : resultsArrayList) {
207         builder.append(line + "\n");
208     }
209
210 }
211
212 @Override
213 public void onEvent(int eventType, Bundle params) {
214
215 }
216 }
217
```

```

1 package com.example.gui_design.bluetoothActivities;
2
3 import android.bluetooth.BluetoothAdapter;
4 import android.bluetooth.BluetoothDevice;
5 import android.bluetooth.BluetoothServerSocket;
6 import android.bluetooth.BluetoothSocket;
7 import android.content.BroadcastReceiver;
8 import android.content.Context;
9 import android.content.Intent;
10 import android.widget.Toast;
11
12 import com.example.gui_design.Application;
13
14 import java.io.InputStream;
15 import java.io.OutputStream;
16 import java.nio.charset.StandardCharsets;
17 import java.util.Set;
18 import java.util.UUID;
19 import java.util.concurrent.Executors;
20 import java.util.concurrent.ScheduledExecutorService;
21 import java.util.concurrent.ScheduledFuture;
22
23
24 public class BluetoothConnection extends BluetoothActivity {
25
26     static ScheduledExecutorService bluetoothThreadPool= Executors.newScheduledThreadPool(1);
27     static ScheduledFuture<?> bluetoothComThread;
28     public static OutputStream outputStream;
29     public static InputStream inputStream;
30     public static BluetoothAdapter btAdapter;
31     public static BluetoothDevice targetDevice;
32     public static BluetoothDevice[] bluetoothDevicesList;
33     public static BluetoothSocket btSocket;
34     public static BluetoothServerSocket btServerSocket;
35     public static UUID btUUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");
36     public static String btString = "HC-05";
37     public static Boolean bluetoothConnected = false;
38
39     public static void listAvailableBluetoothDevices(){
40         btAdapter = BluetoothAdapter.getDefaultAdapter();
41
42         if (btAdapter != null) {
43             if (btAdapter.isEnabled()) {
44                 Set<BluetoothDevice> bondedDevices = btAdapter.getBondedDevices();
45                 if(bondedDevices.size() > 0) {
46                     bluetoothDevicesList = bondedDevices.toArray(new BluetoothDevice[bondedDevices.
47                         size()]);
48                     String[] devicesNames = new String[bondedDevices.size()];
49                     deviceList.clear();
50
51                     for (int x = 0; x < bondedDevices.size(); x++) {
52                         devicesNames[x] = bluetoothDevicesList[x].getName();
53                         deviceList.add(new AvailableDevice(bluetoothDevicesList[x].getName(),
54                             AvailableDevice.DeviceState.DEFAULT));
55                     }
56
57                 }
58             } else {
59                 BluetoothActivity.setBluetoothStateGui(blueoothConnectionState.BLUETOOTH_DISABLED);
60             }
61         }
62     }
63
64
65     public static void connectBluetoothDevice(int deviceIndex){
66         targetDevice = bluetoothDevicesList[deviceIndex];
67
68         try{
69             btSocket = targetDevice.createRfcommSocketToServiceRecord(btUUID);
70             btSocket.connect();
71             inputStream = btSocket.getInputStream();
72             outputStream = btSocket.getOutputStream();
73             BluetoothActivity.setBluetoothStateGui(blueoothConnectionState.SYSTEM_CONNECTED);
74             deviceList.get(deviceIndex).setCurrentState(AvailableDevice.DeviceState.CONNECTED);
75             BluetoothActivity.mAdapter.notifyDataSetChanged();

```

File - C:\ Library256\Feria_2019\Interfaz_grafica\app\src\main\java\com\example\gui_design\bluetoothActivities\BluetoothConne

```
76     }catch (Throwable th){
77         Application.toastMessage(th.toString());
78     }
79
80 }
81
82
83
84
85
86     public static void connectionLost(){
87
88 }
89
90     public static void closeAllConnections(){
91         try {
92             if (bluetoothComThread != null) bluetoothComThread.cancel(true);
93             if (btSocket != null) btSocket.close();
94             if (outputStream != null) outputStream.close();
95             if (inputStream != null) inputStream.close();
96             bluetoothConnected = false;
97
98             btSocket = null;
99             outputStream = null;
100            inputStream = null;
101        } catch (Throwable th){
102
103    }
104 }
105
106     public enum commandType {TURN_RELAY_ON, TURN_RELAY_OFF, TURN_EVERYTHING_ON, TURN_EVERYTHING_OFF,
CALL_NURSE}
107
108     public static boolean composeAndSend(commandType cmdType, int parameter){
109         try {
110             String commandOutput = null;
111             switch (cmdType) {
112                 case TURN_RELAY_ON:
113                     commandOutput = String.format("/TURN_RELAY_ON[%s]", Integer.toString(parameter));
114                     break;
115                 case TURN_RELAY_OFF:
116                     commandOutput = String.format("/TURN_RELAY_OFF[%s]", Integer.toString(parameter));
117                     break;
118                 case TURN_EVERYTHING_ON:
119                     commandOutput = "/TURN_EVERYTHING_ON";
120                     break;
121                 case TURN_EVERYTHING_OFF:
122                     commandOutput = "/TURN_EVERYTHING_OFF";
123                     break;
124                 case CALL_NURSE:
125                     commandOutput = "/CALL_NURSE";
126                     break;
127             }
128
129             outputStream.write(commandOutput.getBytes(StandardCharsets.US_ASCII));
130             Application.toastMessage(commandOutput);
131             return true;
132         }catch (Throwable th) {
133             Application.toastMessage(th.toString());
134             return false;
135         }
136     }
137
138     public static BroadcastReceiver bluetoothAvailability = new BroadcastReceiver() {
139         @Override
140         public void onReceive(Context context, Intent intent) {
141             String action = intent.getAction();
142             BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
143
144             if (BluetoothDevice.ACTION_FOUND.equals(action)) {
145
146             }
147             else if (BluetoothDevice.ACTION_ACL_CONNECTED.equals(action)) {
148                 Toast.makeText(Application.getActivity(), "Device connected!", Toast.LENGTH_SHORT
).show();
149         }
150     }
151 }
```

```
149         }
150         else if (BluetoothDevice.ACTION_ACL_DISCONNECTED.equals(action)) {
151             Toast.makeText(Application.getActivity(), "Device disconnected!", Toast.LENGTH_SHORT)
152                 .show();
153             BluetoothConnection.connectionLost();
154             BluetoothConnection.closeAllConnections();
155         }
156     };
157
158     public static final BroadcastReceiver bluetoothAdapterState = new BroadcastReceiver() {
159         @Override
160         public void onReceive(Context context, Intent intent) {
161             final String action = intent.getAction();
162
163             if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
164                 final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
165                     BluetoothAdapter.ERROR);
166                 switch (state) {
167                     case BluetoothAdapter.STATE_OFF:
168                         break;
169                     case BluetoothAdapter.STATE_TURNING_OFF:
170                         break;
171                     case BluetoothAdapter.STATE_ON:
172                         break;
173                     case BluetoothAdapter.STATE_TURNING_ON:
174                         break;
175                 }
176             }
177         }
178     };
179 }
180
181 }
182 };
183 }
184 }
```

```

1 package com.example.gui_design.bluetoothActivities;
2
3 import android.graphics.PorterDuff;
4 import android.graphics.drawable.Drawable;
5 import android.os.Bundle;
6 import android.support.v4.content.ContextCompat;
7 import android.support.v7.app.AppCompatActivity;
8 import android.support.v7.widget.DefaultItemAnimator;
9 import android.support.v7.widget.DividerItemDecoration;
10 import android.support.v7.widget.LinearLayoutManager;
11 import android.support.v7.widget.RecyclerView;
12 import android.support.v7.widget.Toolbar;
13 import android.view.MenuItem;
14 import android.view.View;
15 import android.widget.ImageView;
16 import android.widget.TextView;
17
18 import com.example.gui_design.Application;
19 import com.example.gui_design.R;
20 import com.example.gui_design.settingsActivities.RecyclerTouchListener;
21
22 import java.util.ArrayList;
23 import java.util.List;
24
25 public class BluetoothActivity extends AppCompatActivity {
26     private static Toolbar toolbar;
27
28
29     public static List<AvailableDevice> deviceList = new ArrayList<>();
30     private RecyclerView recyclerView;
31     public static DeviceListAdapter mAdapter;
32
33     public static ImageView imgView_bluetoothStatusIcon;
34     public static TextView textView_bluetoothState;
35
36
37     @Override
38     protected void onCreate(Bundle savedInstanceState) {
39         super.onCreate(savedInstanceState);
40         setContentView(R.layout.bluetooth_layout);
41         Application.setActivity(this);
42         Application.setContext(this);
43
44         toolbar = this.findViewById(R.id.toolbar);
45         textView_bluetoothState = this.findViewById(R.id.textView_btStatus);
46         imgView_bluetoothStatusIcon = this.findViewById(R.id.imgView_btStatusIcon);
47
48         setSupportActionBar(toolbar);
49
50         // add back arrow to toolbar
51         if (getSupportActionBar() != null){
52             getSupportActionBar().setDisplayHomeAsUpEnabled(true);
53             getSupportActionBar().setDisplayShowHomeEnabled(true);
54
55             final Drawable upArrow = ContextCompat.getDrawable(this, R.drawable.
56             abc_ic_ab_back_material);
57             upArrow.setColorFilter(getColor(R.color.colorPrimary), PorterDuff.Mode.SRC_ATOP);
58             getSupportActionBar().setHomeAsUpIndicator(upArrow);
59         }
59
60         recyclerView = (RecyclerView) findViewById(R.id.deviceListRecyclerView);
61
62         mAdapter = new DeviceListAdapter(deviceList);
63
64         recyclerView.setHasFixedSize(true);
65
66         // vertical RecyclerView
67         // keep movie_list_row.xml width to `match_parent`
68         LinearLayoutManager mLayoutManager = new LinearLayoutManager(getApplicationContext());
69
70         // horizontal RecyclerView
71         // keep movie_list_row.xml width to `wrap_content`
72         // RecyclerView.LayoutManager mLayoutManager = new LinearLayoutManager(getApplicationContext
73         (), LinearLayoutManager.HORIZONTAL, false);
73
74         recyclerView.setLayoutManager(mLayoutManager);
75

```

```
76         // adding inbuilt divider line
77         recyclerView.addItemDecoration(new DividerItemDecoration(this, LinearLayoutManager.VERTICAL))
78     ;
79
80         // adding custom divider line with padding 16dp
81         //recyclerView.addItemDecoration(new MyDividerItemDecoration(this, LinearLayoutManager.HORIZONTAL, 16));
82
83         recyclerView.setItemAnimator(new DefaultItemAnimator());
84
85         // row click listener
86         recyclerView.addOnItemTouchListener(new RecyclerTouchListener(getApplicationContext(),
87             recyclerView, new RecyclerTouchListener.ClickListener() {
88
89                 @Override
90                 public void onClick(View view, int position) {
91                     AvailableDevice device = deviceList.get(position);
92
93                     BluetoothConnection.connectBluetoothDevice(position);
94                 }
95
96                 @Override
97                 public void onLongClick(View view, int position) {
98
99                 }
100             }));
101
102
103     @Override
104     public boolean onOptionsItemSelected(MenuItem item) {
105
106         if (item.getItemId() == android.R.id.home) // Press Back Icon
107         {
108             finish();
109         }
110
111         return super.onOptionsItemSelected(item);
112     }
113
114     public static enum bluetoothConnectionState { SYSTEM_DISCONNECTED, SYSTEM_CONNECTED,
115     BLUETOOTH_DISABLED}
116
117     public static void setBluetoothStateGui(bluetoothConnectionState state){
118
119         switch (state){
120             case SYSTEM_CONNECTED:
121                 textView_bluetoothState.setText(R.string.system_connected_message);
122                 imgView_bluetoothStatusIcon.setImageResource(R.drawable.ic_done_icon);
123                 break;
124             case BLUETOOTH_DISABLED:
125                 textView_bluetoothState.setText(R.string.bluetooth_disabled_message);
126                 imgView_bluetoothStatusIcon.setImageResource(R.drawable.ic_bluetooth_disabled);
127                 break;
128             case SYSTEM_DISCONNECTED:
129                 textView_bluetoothState.setText(R.string.system_disconnected_message);
130                 imgView_bluetoothStatusIcon.setImageResource(R.drawable.ic_close_icon);
131                 break;
132         }
133     }
134 }
```

```
1 package com.example.gui_design.bluetoothActivities;
2
3 import com.example.gui_design.R;
4
5 public class AvailableDevice {
6
7     public enum DeviceState { DEFAULT, DISCONNECTED, CONNECTED, LOADING}
8     private String deviceName;
9     private DeviceState currentState;
10    private int deviceStateIconID;
11
12    public AvailableDevice() {
13    }
14
15    public AvailableDevice(String deviceName, DeviceState defaultState) {
16        this.deviceName = deviceName;
17        this.currentState = defaultState;
18    }
19
20    public void setCurrentState(DeviceState currentState) {
21        this.currentState = currentState;
22    }
23
24    public DeviceState getCurrentState() {
25        return currentState;
26    }
27
28    public void setDeviceName(String deviceName) {
29        this.deviceName = deviceName;
30    }
31
32    public String getDeviceName() {
33        return deviceName;
34    }
35
36    public int getDeviceStateIconID() {
37        switch (currentState){
38            case DEFAULT:
39                return 0;
40            case CONNECTED:
41                return R.drawable.ic_done_icon;
42            case LOADING:
43                return R.drawable.loading_animation;
44            case DISCONNECTED:
45                return R.drawable.ic_close_icon;
46        }
47        return 0;
48    }
49
50    public int getTextAppearance(){
51        switch (currentState){
52            case DEFAULT:
53                return R.style.btDeviceListAppearance;
54            case CONNECTED:
55                return R.style.btDeviceListSelectedAppearance;
56            case LOADING:
57                return R.style.btDeviceListAppearance;
58            case DISCONNECTED:
59                return R.style.btDeviceListAppearance;
60        }
61        return R.style.btDeviceListAppearance;
62    }
63 }
64
```

```
1 package com.example.gui_design;
2
3 import android.app.Activity;
4 import android.content.Context;
5 import android.widget.Toast;
6
7 public class Application extends android.app.Application {
8     private static Activity mActivity;
9     private static Context mContext;
10
11    public static Activity getActivity() { return mActivity; }
12    public static void setActivity(Activity mActivitySet) { mActivity = mActivitySet; }
13
14    public static Context getContext() { return mContext; }
15    public static void setContext(Context mContextSet) { mContext = mContextSet; }
16
17    public static void toastMessage(final String message){
18        mActivity.runOnUiThread(new Runnable() {
19            @Override
20            public void run() {
21                Toast.makeText(mActivity, message, Toast.LENGTH_SHORT).show();
22            }
23        });
24    }
25
26
27 }
28
```

```
1 package com.example.gui_design;
2
3 import android.graphics.PorterDuff;
4 import android.graphics.drawable.Drawable;
5 import android.os.Bundle;
6 import android.support.v4.content.ContextCompat;
7 import android.support.v7.app.AppCompatActivity;
8 import android.support.v7.widget.Toolbar;
9 import android.view.MenuItem;
10
11 public class CommandsActivity extends AppCompatActivity {
12     private static Toolbar toolbar;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.commands_layout);
18         Application.setActivity(this);
19         Application.setContext(this);
20
21         toolbar = this.findViewById(R.id.toolbar);
22
23         setSupportActionBar(toolbar);
24
25         // add back arrow to toolbar
26         if (getSupportActionBar() != null){
27             getSupportActionBar().setDisplayHomeAsUpEnabled(true);
28             getSupportActionBar().setDisplayShowHomeEnabled(true);
29
30             final Drawable upArrow = ContextCompat.getDrawable(this, R.drawable.
31                 abc_ic_ab_back_material);
32             upArrow.setColorFilter(getColor(R.color.colorPrimary), PorterDuff.Mode.SRC_ATOP);
33             getSupportActionBar().setHomeAsUpIndicator(upArrow);
34         }
35     }
36
37     @Override
38     public boolean onOptionsItemSelected(MenuItem item) {
39         if (item.getItemId() == android.R.id.home) // Press Back Icon
40         {
41             finish();
42         }
43
44         return super.onOptionsItemSelected(item);
45     }
46 }
```

```

1 package com.example.gui_design;
2
3 import android.Manifest;
4 import android.content.pm.PackageManager;
5 import android.os.Bundle;
6 import android.support.v4.app.ActivityCompat;
7 import android.support.v4.content.ContextCompat;
8 import android.support.v7.app.AppCompatActivity;
9 import android.util.DisplayMetrics;
10 import android.view.View;
11 import android.widget.ImageButton;
12 import android.widget.TextView;
13
14 import com.example.gui_design.bluetoothActivities.BluetoothConnection;
15 import com.example.gui_design.settingsActivities.settings_layout;
16
17 public class MainActivity extends AppCompatActivity {
18
19     static ImageButton imgBtn_microphone;
20     static ImageButton imgBtn_microphoneBackground;
21     static ImageButton imgBtn_commands;
22     static ImageButton imgBtn_contacts;
23     static ImageButton imgBtn_bluetooth;
24     static ImageButton imgBtn_settings;
25     static ImageButton imgBtn_menu;
26     static MainMenuHandler mainMenuHandler;
27     static TextView textView_speechRecognitionResults;
28
29
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.activity_main);
34
35         if(savedInstanceState == null) {
36
37             mainMenuHandler = new MainMenuHandler(this, this);
38             Application.setActivity(this);
39             Application.setContext(this);
40             MainMenuHandler.dpiConstant = ((float) this.getResources().getDisplayMetrics().densityDpi
4 / DisplayMetrics.DENSITY_DEFAULT);
41
42             imgBtn_microphone = findViewById(R.id.imgBtn_microphone);
43             imgBtn_microphoneBackground = findViewById(R.id.imgBtn_microphone_background);
44             imgBtn_commands = findViewById(R.id.imgBtn_commands);
45             imgBtn_contacts = findViewById(R.id.imgBtn_contacts);
46             imgBtn_bluetooth = findViewById(R.id.imgBtn_bluetooth);
47             imgBtn_settings = findViewById(R.id.imgBtn_settings);
48             imgBtn_menu = findViewById(R.id.imgBtn_menu);
49             textView_speechRecognitionResults = findViewById(R.id.textView_speechRecognitionResults);
50
51             if (ContextCompat.checkSelfPermission(this,
52                     Manifest.permission.RECORD_AUDIO)
53                     != PackageManager.PERMISSION_GRANTED) {
54                 if (ActivityCompat.shouldShowRequestPermissionRationale(this,
55                         Manifest.permission.RECORD_AUDIO)) {
56                     } else {
57                         ActivityCompat.requestPermissions(this,
58                             new String[]{Manifest.permission.RECORD_AUDIO},
59                             527);
56                     }
57                 } else {
58                     GoogleSpeechRecognizer.setupRecognizer();
59                     PocketsphinxListener.setupRecognizer();
60                     PocketsphinxListener.startListening();
61
62                 }
63
64                 settings_layout.setupSettingsList();
65
66             }
67
68         }
69
70     }
71
72     }
73
74     @Override
75     public void onRequestPermissionsResult(int requestCode,
76                                         String permissions[], int[] grantResults) {

```

```
77         switch (requestCode) {
78             case 527: {
79                 // If request is cancelled, the result arrays are empty.
80                 if (grantResults.length > 0
81                     && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
82
83                     GoogleSpeechRecognizer.setupRecognizer();
84                     PocketsphinxListener.setupRecognizer();
85
86                 } else { }
87             }
88         }
89     }
90 }
91
92 public void onClickListener_menu(View view){ mainMenuHandler.menuButtonOnClick(); }
93
94 public void onClickListener_commands(View view){ mainMenuHandler.commandsButtonOnClick(); }
95
96 public void onClickListener_settings(View view){ mainMenuHandler.settingsButtonOnClick(); }
97
98 public void onClickListener_contacts(View view){ mainMenuHandler.contactsButtonOnClick(); }
99
100 public void onClickListener_bluetooth(View view){ mainMenuHandler.bluetoothButtonOnClick(); }
101
102 public void onClickListener_microphone(View view){ MicrophoneHandler.microphoneOnClick(); }
103
104 @Override
105 public void onDestroy() {
106     super.onDestroy();
107     PocketsphinxListener.shutdownRecognizer();
108     GoogleSpeechRecognizer.shutdownRecognizer();
109     BluetoothConnection.closeAllConnections();
110 }
111
112 }
113
```

```
1 package com.example.gui_design;
2
3 import android.animation.ValueAnimator;
4 import android.app.Activity;
5 import android.content.Context;
6 import android.content.Intent;
7 import android.view.View;
8 import android.view.animation.AccelerateDecelerateInterpolator;
9
10 import com.example.gui_design.bluetoothActivities.BluetoothActivity;
11 import com.example.gui_design.settingsActivities.settings_layout;
12
13 public class MainMenuHandler extends MainActivity{
14
15     public static float dpiConstant;
16     public static float radialMenuRadius = 263.0f;
17     public static boolean menuUncoiled = false;
18     public static boolean isAnimationRunning = false;
19     public static Context mainMenuContext;
20     public static Activity mainMenuActivity;
21
22     public MainMenuHandler(Context context, Activity activity){
23         this.mainMenuActivity = activity;
24         this.mainMenuContext = context;
25     }
26
27     public static void menuButtonOnClick(){
28         long animationDuration = 850;
29
30         if(isAnimationRunning==false) {
31             isAnimationRunning = true;
32
33             if (menuUncoiled == false) {
34                 ValueAnimator animatorCommands = ValueAnimator.ofFloat(360, 72);
35                 animatorCommands.setDuration(animationDuration);
36                 animatorCommands.setInterpolator(new AccelerateDecelerateInterpolator());
37                 animatorCommands.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
38                     @Override
39                     public void onAnimationUpdate(ValueAnimator animation) {
40                         float value = ((Float) (animation.getAnimatedValue()))
41                             .floatValue();
42
43                         float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
44                             radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
45                         float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
46                             radialMenuRadius * translationX));
47                         if (value > 180.0f) {
48                             translationY = -translationY;
49                         }
50
51                         imgBtn_commands.setTranslationX(convertDpToPixel(translationX));
52                         imgBtn_commands.setTranslationY(convertDpToPixel(translationY));
53                     }
54                 });
55                 imgBtn_commands.setVisibility(View.VISIBLE);
56                 animatorCommands.start();
57
58                 ValueAnimator animatorContacts = ValueAnimator.ofFloat(360, 144);
59                 animatorContacts.setDuration(animationDuration);
60                 animatorContacts.setInterpolator(new AccelerateDecelerateInterpolator());
61                 animatorContacts.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
62                     @Override
63                     public void onAnimationUpdate(ValueAnimator animation) {
64                         float value = ((Float) (animation.getAnimatedValue()))
65                             .floatValue();
66
67                         float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
68                             radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
69                         float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
70                             radialMenuRadius * translationX));
71                         if (value > 180.0f) {
72                             translationY = -translationY;
73                         }
74
75                         imgBtn_contacts.setTranslationX(convertDpToPixel(translationX));
76                         imgBtn_contacts.setTranslationY(convertDpToPixel(translationY));
77                     }
78                 });
79             }
80         }
81     }
82 }
```

```

74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
    }
    imgBtn_contacts.setVisibility(View.VISIBLE);
    animatorContacts.start();

    ValueAnimator animatorSettings = ValueAnimator.ofFloat(360, 216);
    animatorSettings.setDuration(animationDuration);
    animatorSettings.setInterpolator(new AccelerateDecelerateInterpolator());
    animatorSettings.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
        @Override
        public void onAnimationUpdate(ValueAnimator animation) {
            float value = ((Float) (animation.getAnimatedValue()))
                .floatValue();

            float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
                radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
            float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
                radialMenuRadius * translationX));
            if (value > 180.0f) {
                translationY = -translationY;
            }

            imgBtn_settings.setTranslationX(convertDpToPixel(translationX));
            imgBtn_settings.setTranslationY(convertDpToPixel(translationY));
        }
    });
    imgBtn_settings.setVisibility(View.VISIBLE);
    animatorSettings.start();

    ValueAnimator animatorBluetooth = ValueAnimator.ofFloat(360, 288);
    animatorBluetooth.setDuration(animationDuration);
    animatorBluetooth.setInterpolator(new AccelerateDecelerateInterpolator());
    animatorBluetooth.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
        @Override
        public void onAnimationUpdate(ValueAnimator animation) {
            float value = ((Float) (animation.getAnimatedValue()))
                .floatValue();

            float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
                radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
            float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
                radialMenuRadius * translationX));
            if (value > 180.0f) {
                translationY = -translationY;
            }

            imgBtn_bluetooth.setTranslationX(convertDpToPixel(translationX));
            imgBtn_bluetooth.setTranslationY(convertDpToPixel(translationY));
        }
    });
    imgBtn_bluetooth.setVisibility(View.VISIBLE);
    animatorBluetooth.start();

    imgBtn_commands.animate().alpha(1.0f).setDuration(animationDuration).setInterpolator(
        new AccelerateDecelerateInterpolator());
    imgBtn_contacts.animate().alpha(1.0f).setDuration(animationDuration).setInterpolator(
        new AccelerateDecelerateInterpolator());
    imgBtn_settings.animate().alpha(1.0f).setDuration(animationDuration).setInterpolator(
        new AccelerateDecelerateInterpolator());
    imgBtn_bluetooth.animate().alpha(1.0f).setDuration(animationDuration).setInterpolator(
        new AccelerateDecelerateInterpolator()).withEndAction(new Runnable() {
        @Override
        public void run() {
            isAnimationRunning = false;
        }
    });

    menuUncoiled = true;

} else {
    ValueAnimator animatorCommands = ValueAnimator.ofFloat(72, 360);
}

```

```

143         animatorCommands.setDuration(animationDuration);
144         animatorCommands.setInterpolator(new AccelerateDecelerateInterpolator());
145         animatorCommands.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
146             @Override
147             public void onAnimationUpdate(ValueAnimator animation) {
148                 float value = ((Float) (animation.getAnimatedValue()))
149                     .floatValue();
150
151                 float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
152                     radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
153                 float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
154                     radialMenuRadius * translationX));
155                 if (value > 180.0f) {
156                     translationY = -translationY;
157                 }
158
159                 imgBtn_commands.setTranslationX(convertDpToPixel(translationX));
160                 imgBtn_commands.setTranslationY(convertDpToPixel(translationY));
161             }
162         });
163         imgBtn_commands.setVisibility(View.VISIBLE);
164         animatorCommands.start();
165
166         ValueAnimator animatorContacts = ValueAnimator.ofFloat(144, 360);
167         animatorContacts.setDuration(animationDuration);
168         animatorContacts.setInterpolator(new AccelerateDecelerateInterpolator());
169         animatorContacts.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
170             @Override
171             public void onAnimationUpdate(ValueAnimator animation) {
172                 float value = ((Float) (animation.getAnimatedValue()))
173                     .floatValue();
174
175                 float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
176                     radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
177                 float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
178                     radialMenuRadius * translationX));
179                 if (value > 180.0f) {
180                     translationY = -translationY;
181                 }
182
183                 imgBtn_contacts.setTranslationX(convertDpToPixel(translationX));
184                 imgBtn_contacts.setTranslationY(convertDpToPixel(translationY));
185             }
186         });
187         imgBtn_contacts.setVisibility(View.VISIBLE);
188         animatorContacts.start();
189
190         ValueAnimator animatorSettings = ValueAnimator.ofFloat(216, 360);
191         animatorSettings.setDuration(animationDuration);
192         animatorSettings.setInterpolator(new AccelerateDecelerateInterpolator());
193         animatorSettings.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
194             @Override
195             public void onAnimationUpdate(ValueAnimator animation) {
196                 float value = ((Float) (animation.getAnimatedValue()))
197                     .floatValue();
198
199                 float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
200                     radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
201                 float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
202                     radialMenuRadius * translationX));
203                 if (value > 180.0f) {
204                     translationY = -translationY;
205                 }
206
207                 imgBtn_settings.setTranslationX(convertDpToPixel(translationX));
208                 imgBtn_settings.setTranslationY(convertDpToPixel(translationY));
209             }
210         });
211         imgBtn_settings.setVisibility(View.VISIBLE);
212         animatorSettings.start();
213
214         ValueAnimator animatorBluetooth = ValueAnimator.ofFloat(288, 360);
215         animatorBluetooth.setDuration(animationDuration);
216         animatorBluetooth.setInterpolator(new AccelerateDecelerateInterpolator());

```

```

214         animatorBluetooth.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
215             @Override
216             public void onAnimationUpdate(ValueAnimator animation) {
217                 float value = ((Float) (animation.getAnimatedValue()))
218                     .floatValue();
219
220                 float translationX = (float) (Math.cos(value * (Math.PI / 180.0f)) * (
221                     radialMenuRadius / 2.0f) - (radialMenuRadius / 2.0f));
222                 float translationY = (float) (Math.sqrt(-Math.pow(translationX, 2.0d) -
223                     radialMenuRadius * translationX));
224                 if (value > 180.0f) {
225                     translationY = -translationY;
226                 }
227
228                 imgBtn_bluetooth.setTranslationX(convertDpToPixel(translationX));
229                 imgBtn_bluetooth.setTranslationY(convertDpToPixel(translationY));
230             }
231             imgBtn_bluetooth.setVisibility(View.VISIBLE);
232             animatorBluetooth.start();
233
234             imgBtn_commands.animate().alpha(0.0f).setDuration(animationDuration).setInterpolator(
235                 new AccelerateDecelerateInterpolator());
236             imgBtn_contacts.animate().alpha(0.0f).setDuration(animationDuration).setInterpolator(
237                 new AccelerateDecelerateInterpolator());
238             imgBtn_settings.animate().alpha(0.0f).setDuration(animationDuration).setInterpolator(
239                 new AccelerateDecelerateInterpolator());
240             imgBtn_bluetooth.animate().alpha(0.0f).setDuration(animationDuration).setInterpolator(
241                 new AccelerateDecelerateInterpolator()).withEndAction(new Runnable() {
242                     @Override
243                     public void run() {
244                         isAnimationRunning = false;
245                     }
246                 });
247             menuUncoiled = false;
248         }
249     }
250
251     public static void commandsButtonOnClick() {
252         Context currentContext = Application.getContext();
253         Activity currentActivity = Application.getActivity();
254         Intent i = new Intent(currentContext, CommandsActivity.class);
255         currentContext.startActivity(i);
256     }
257
258     public static void contactsButtonOnClick() {
259         Context currentContext = Application.getContext();
260         Activity currentActivity = Application.getActivity();
261         Intent i = new Intent(currentContext, ContactsActivity.class);
262         currentContext.startActivity(i);
263     }
264
265     public static void bluetoothButtonOnClick() {
266         Context currentContext = Application.getContext();
267         Activity currentActivity = Application.getActivity();
268         Intent i = new Intent(currentContext, BluetoothActivity.class);
269         currentContext.startActivity(i);
270     }
271
272     public static void settingsButtonOnClick() {
273         Intent i = new Intent(Application.getContext(), settings_layout.class);
274         mainMenuContext.startActivity(i);
275         //mainMenuActivity.overridePendingTransition(R.anim.fade_in, R.anim.fade_out);
276     }
277
278
279
280
281
282
283     /**
284      * This method converts dp unit to equivalent pixels, depending on device density.

```

```
285     *
286     * param dp A value in dp (density independent pixels) unit. Which we need to convert into
287     * pixels
288     * return A float value to represent px equivalent to dp depending on device density
289     */
290     public static float convertDpToPixel(float dp){ return dp * dpiConstant; }
291
292     /**
293     * This method converts device specific pixels to density independent pixels.
294     *
295     * param px A value in px (pixels) unit. Which we need to convert into db
296     * return A float value to represent dp equivalent to px value
297     */
298     public static float convertPixelsToDp(float px){
299         return px / dpiConstant;
300     }
301 }
```

```

1 package com.example.gui_design;
2
3 import android.graphics.PorterDuff;
4 import android.view.animation.AccelerateDecelerateInterpolator;
5
6 import java.util.concurrent.Callable;
7 import java.util.concurrent.ExecutorService;
8 import java.util.concurrent.Executors;
9 import java.util.concurrent.Future;
10
11 public class MicrophoneHandler extends MainActivity {
12
13     public enum MicrophoneState { INHIBITED, LISTENING_KEYWORD, LISTENING_COMMAND}
14
15     static private MicrophoneState currentState = MicrophoneState.LISTENING_KEYWORD;
16
17     public static void microphoneOnClick(){
18         switch (currentState){
19             case LISTENING_KEYWORD:
20                 setState(MicrophoneState.INHIBITED);
21                 break;
22             case LISTENING_COMMAND:
23                 GoogleSpeechRecognizer.forceStopRecognizer();
24                 setState(MicrophoneState.LISTENING_KEYWORD);
25                 break;
26             case INHIBITED:
27                 setState(MicrophoneState.LISTENING_KEYWORD);
28                 break;
29         }
30     }
31
32
33     public static MicrophoneState getState(){
34         return currentState;
35     }
36
37     public static void setState(MicrophoneState desiredState){
38         switch (desiredState){
39             case INHIBITED:
40                 imgBtn_microphoneBackground.clearColorFilter();
41                 imgBtn_microphoneBackground.setColorFilter(Application.getContext().getColor(R.color.
ColorBackMicInhibited), PorterDuff.Mode.SRC_ATOP);
42                 imgBtn_microphone.setImageResource(R.drawable.ic_main_mic_off);
43                 endAnimation();
44                 currentState = MicrophoneState.INHIBITED;
45                 break;
46             case LISTENING_KEYWORD:
47                 imgBtn_microphoneBackground.clearColorFilter();
48                 imgBtn_microphoneBackground.setColorFilter(Application.getContext().getColor(R.color.
ColorBackMicKeyword), PorterDuff.Mode.SRC_ATOP);
49                 imgBtn_microphone.setImageResource(R.drawable.ic_main_mic_listening);
50                 endAnimation();
51                 currentState = MicrophoneState.LISTENING_KEYWORD;
52                 break;
53             case LISTENING_COMMAND:
54                 imgBtn_microphoneBackground.clearColorFilter();
55                 imgBtn_microphoneBackground.setColorFilter(Application.getContext().getColor(R.color.
ColorBackMicCommand), PorterDuff.Mode.SRC_ATOP);
56                 imgBtn_microphone.setImageResource(R.drawable.ic_main_mic_listening);
57                 startAnimation();
58                 currentState = MicrophoneState.LISTENING_COMMAND;
59                 break;
60         }
61     }
62
63     private static ExecutorService executor = Executors.newSingleThreadExecutor();
64     private static boolean terminateAnimation = false;
65     private static boolean animationHasFinished = false;
66
67     public static void startAnimation(){
68         terminateAnimation = false;
69         Future<Void> future = executor.submit(new Callable<Void>() {
70             public Void call() throws Exception {
71                 try {
72                     while (terminateAnimation == false) {
73                         imgBtn_microphoneBackground.animate().scaleX(0.90f).setDuration(350).
setInterpolator(new AccelerateDecelerateInterpolator());

```

```
74         imgBtn_microphone.animate().scaleX(0.90f).setDuration(350).setInterpolator(  
75             new AccelerateDecelerateInterpolator());  
76             imgBtn_microphoneBackground.animate().scaleY(0.90f).setDuration(350).  
77             setInterpolator(new AccelerateDecelerateInterpolator());  
78                 imgBtn_microphone.animate().scaleY(0.90f).setDuration(350).setInterpolator(  
79                     new AccelerateDecelerateInterpolator()).withEndAction(new Runnable() {  
80                         @Override  
81                         public void run() {  
82                             animationHasFinished = true;  
83                         }  
84                     });  
85                     while(animationHasFinished==false);  
86                     animationHasFinished= false;  
87                     imgBtn_microphoneBackground.animate().scaleX(1.00f).setDuration(350).  
88                     setInterpolator(new AccelerateDecelerateInterpolator());  
89                     imgBtn_microphone.animate().scaleX(1.00f).setDuration(350).setInterpolator(  
90                         new AccelerateDecelerateInterpolator());  
91                         imgBtn_microphoneBackground.animate().scaleY(1.00f).setDuration(350).  
92                         setInterpolator(new AccelerateDecelerateInterpolator());  
93                         imgBtn_microphone.animate().scaleY(1.00f).setDuration(350).setInterpolator(  
94                             new AccelerateDecelerateInterpolator()).withEndAction(new Runnable() {  
95                                 @Override  
96                                 public void run() {  
97                                     animationHasFinished = true;  
98                                 }  
99                             });  
100                             while(animationHasFinished==false);  
101                             animationHasFinished = false;  
102                             }  
103                         );  
104                     }  
105                     public static void endAnimation(){  
106                         terminateAnimation = true;  
107                     }  
108                 }  
109             }  
110         }
```

```

1 package com.example.gui_design;
2
3 import java.io.File;
4
5 import edu.cmu.pocketsphinx.Assets;
6 import edu.cmu.pocketsphinx.Hypothesis;
7 import edu.cmu.pocketsphinx.RecognitionListener;
8 import edu.cmu.pocketsphinx.SpeechRecognizer;
9 import edu.cmu.pocketsphinx.SpeechRecognizerSetup;
10
11 public final class PocketsphinxListener implements RecognitionListener {
12     private static SpeechRecognizer recognizer;
13     private static String KEYPHRASE = "asistente";
14     private static final String KWS_SEARCH = "wakeup";
15
16
17     @Override
18     public void onPartialResult(Hypothesis hypothesis) {
19         if (hypothesis == null)
20             return;
21
22         String text = hypothesis.getHypstr();
23
24         if (text.contains(KEYPHRASE)) {
25
26             Application.toastMessage(KEYPHRASE);
27             recognizer.stop();
28             //recognizer.cancel();
29             //recognizer.shutdown();
30             GoogleSpeechRecognizer.startSpeechListening();
31             //recognizer.startListening(KWS_SEARCH);
32         }
33     }
34
35
36
37     @Override
38     public void onResult(Hypothesis hypothesis) {
39         if (hypothesis != null) {
40             String text = hypothesis.getHypstr();
41
42         }
43     }
44
45     @Override
46     public void onBeginningOfSpeech() {
47     }
48
49     @Override
50     public void onEndOfSpeech() {
51
52     }
53
54     public static void setupRecognizer() {
55         try {
56             Assets assets = new Assets(getApplicationContext());
57             File assetDir = assets.syncAssets();
58             recognizer = SpeechRecognizerSetup.defaultSetup()
59                 .setAcousticModel(new File(assetDir, "es-es-ptm"))
60                 .setDictionary(new File(assetDir, "cmudict-es-es.dict"))
61                 .setKeywordThreshold(Float.MIN_VALUE)
62
63                 .setRawLogDir(assetDir) // To disable logging of raw audio comment out this call (takes a lot of space on the device)
64
65                 .getRecognizer();
66             recognizer.addListener(new PocketsphinxListener());
67             recognizer.addKeyphraseSearch(KWS_SEARCH, KEYPHRASE);
68         } catch (Throwable th) {
69             Application.toastMessage(th.toString());
70         }
71     }
72
73     public static void startListening(){
74         recognizer.startListening(KWS_SEARCH);
75     }
76

```

```
77     public static void shutdownRecognizer() {
78         if (recognizer != null) {
79             recognizer.cancel();
80             recognizer.shutdown();
81         }
82     }
83
84     public static void stopRecognizer() {
85         recognizer.stop();
86     }
87
88     @Override
89     public void onError(Exception error) {
90         Application.toastMessage("Error: " + error.toString());
91     }
92
93     @Override
94     public void onTimeout() {
95         Application.toastMessage("Timeout!");
96     }
97
98
99
100 }
101
102
```