Term Project Report

CSE 3442: Embedded Systems I

Ikechukwu Ofili

May 5, 2023

Contents

1 Introduction

The project is a haptic walking aid for visually impaired. It's job is to give haptic feedback to the user if certain proximity events are triggered. For example, if you are about to hit a wall or you are walking up a flight of stairs. The user can configure the walking aid as much as they need even tweaking the intensity of the vibration

2 Theory of Operation

To get a wholistic approach of the project, I feel we need to first understand the various parts that make up the project.

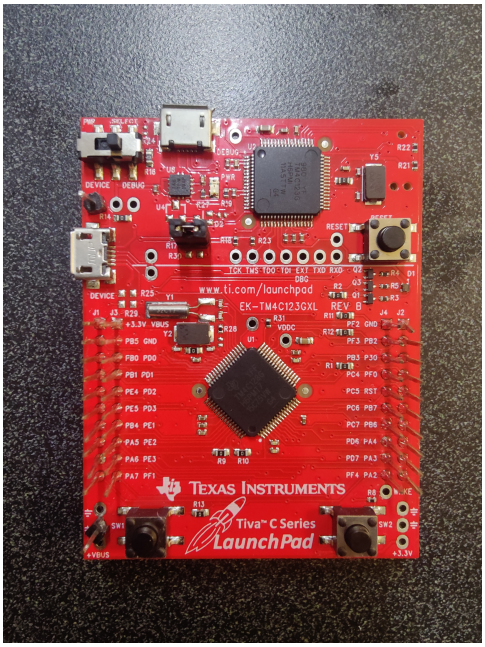

Figure 1: The Tiva C Series TM4C123GH6PM microcontroller board

First, we have the brain of the walking aid, the microcontroller board, *Texas Instruments* TM4C123GH6PM microcontroller. This board powers all the hardware, handles the logic for reading the sensors, choosing to turn on the motor or not, saving user config settings, etc.

This project also uses three ultra-sonic distance sensors which are the HC-SR04 sensors (I will link the datasheet here). The sensors are connected

to the microcontroller as GPIOs (general-purpose inputs and outputs) I use a timer to measure the amount of time that passed between when the sound wave was pulsed and when it bounces back which I can then use to calculate the distance using the speed of sound. Then, another timer that times is used to store the current distances measured by these sensors to a global variable `'dist'` ten times a second. After that, we then turn on (or off) the motor if the specified distance(s) warrants that.

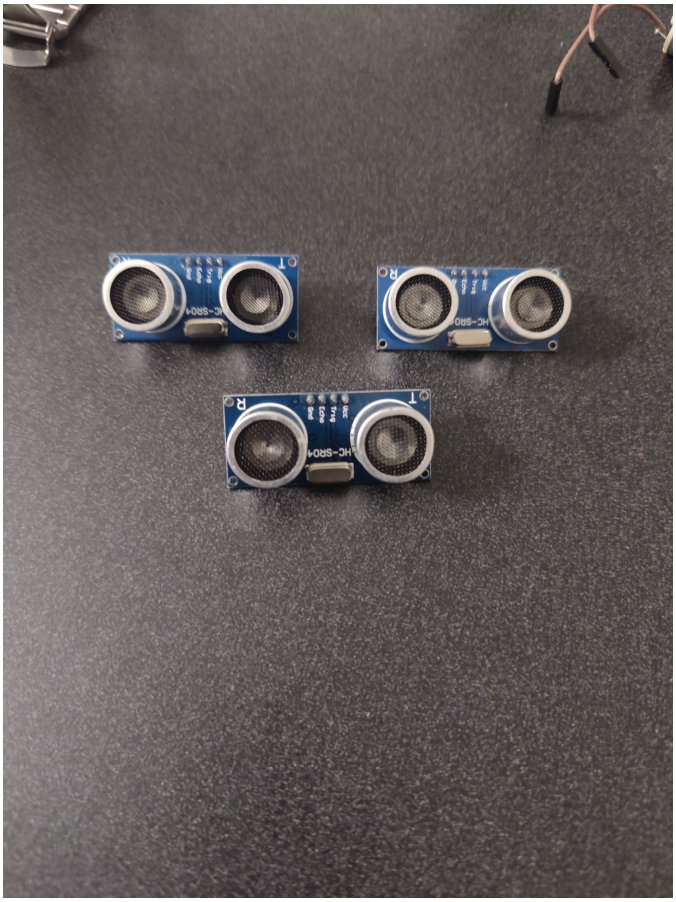

Figure 2: HC-SR04 Ultra-sonic sensors

The motor used draws a lot of current. Therefore, I could not power it directly using the microcontroller board but I had to use a MOSFET as a switch(ing amplifier)

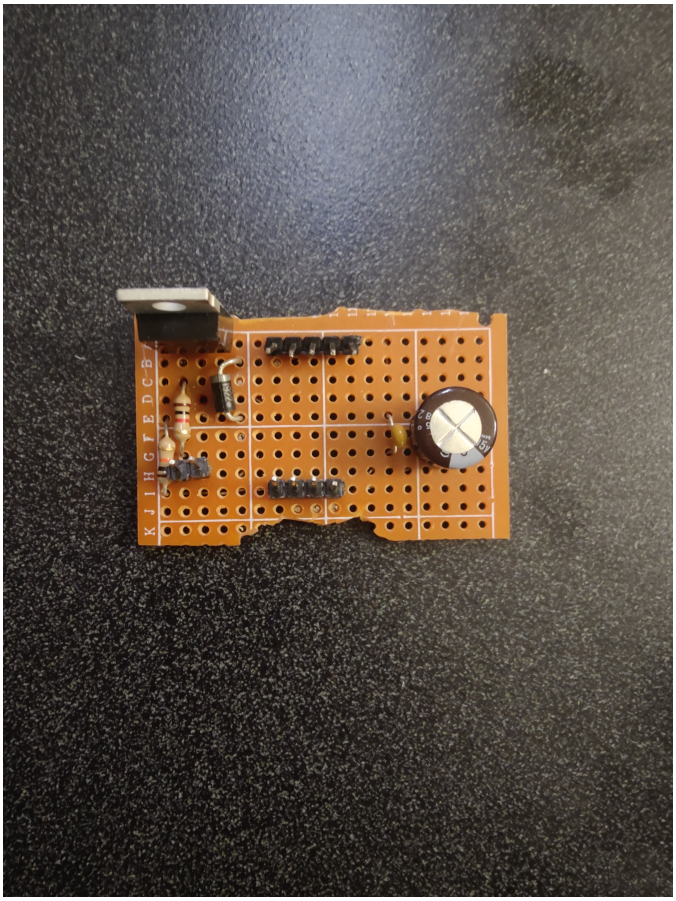

Figure 3: Switching circuit for powering the motor

A major part of this project is the event system. Users can add certain events that triggers the motors when the event is live. There are 16 simple events and 4 compound ones. Simple events simply vibrates the cane with a specified pattern when any of the three sensors senses an obstacle between a set distance. A compound event is the conjunction of two simple events. That is, it sets off when both events are live. The user can also set a specific pattern for those too. An event I used to test the final product and I find to be very useful is one to detect the next step while climbing up a flight of stairs. The events are also stored to long-term memory so that after power-off, user settings are not lost

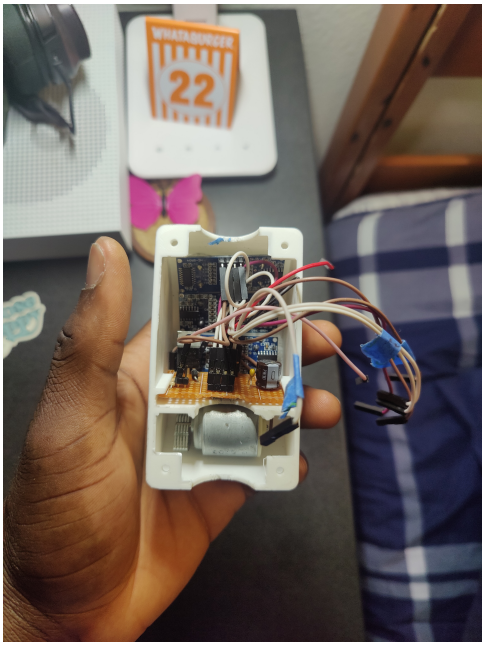

Figure 4: The sensors, motor and circuit all wired up in the plastic casing

Figure 5: A picture of the completed project

Finally, the completed project is housed in a 3D-printed plastic encasing and I used a PVC pipe as a mock-up walking stick for testing. A battery pack is also strapped on for powering the walking aid outside development.

3 Code

All code for this project was written in C. Development was done in TI's *Code Composer Studio* over a span of about 8 weeks. I have included all the files needed to build and run this project in its entirety below.

```
// Ikechukwuka Ofili

//-----------------------------------------------------------------------------
// Hardware Target
//-----------------------------------------------------------------------------

// Target Platform: EK-TM4C123GXL
// Target uC:       TM4C123GH6PM
// System Clock:    40 MHz
// Stack:           4096 bytes (needed for snprintf)

// Hardware configuration:
// Green LED:
//   PF3 drives an NPN transistor that powers the green LED
// Blue LED:
//   PF2 drives an NPN transistor that powers the blue LED
// UART Interface:
//   U0TX (PA1) and U0RX (PA0) are connected to the 2nd controller
//   The USB on the 2nd controller enumerates to an ICDI interface and a virtual
    COM port
//   Configured to 115,200 baud, 8N1


//-----------------------------------------------------------------------------
// Device includes, defines, and assembler directives
//-----------------------------------------------------------------------------

#include <inttypes.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include "clock.h"
#include "wait.h"
#include "uart0.h"
#include "eeprom.h"
#include "sensors.h"
#include "motor.h"
#include "tm4c123gh6pm.h"

#define RED_LED     (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)
    *32 + 1*4)))

// PortF masks
#define RED_LED_MASK 2

// Events
typedef struct _sevent
{
    uint8_t sensor;

    uint8_t on;
    uint8_t disabled;

    uint8_t pwm;
    uint32_t cyc;
    uint32_t on_time;
    uint32_t off_time;


    uint32_t min_mm;
    uint32_t max_mm;

} simpleEvent;


typedef struct _cevent
{
    uint8_t on;
    uint8_t disabled;
    uint8_t pwm;
    char    pad[4];

    uint32_t cyc;
    uint32_t on_time;
    uint32_t off_time;
```

```
    uint32_t event1;
    uint32_t event2;

} compoundEvent;

#define NUM_SIMPLE 16
#define NUM_COMPOUND 4

#define NUM_EVENTS (NUM_SIMPLE + NUM_COMPOUND)

simpleEvent simple_events[NUM_SIMPLE];
compoundEvent compound_events[NUM_COMPOUND];

#define SE_SIZE_WORDS ((sizeof(simple_events) + 3) >> 2)
#define CE_SIZE_WORDS ((sizeof(compound_events) + 3) >> 2)

// Globals
USER_DATA data;

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

// Initialize Hardware
void initHw()
{
    // Initialize system clock to 40 MHz
    initSystemClockTo40Mhz();

    // Enable clocks
    SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R5; //Port F
    SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R2; // Timer 2
    _delay_cycles(3);

    // Configure LED pins
    GPIO_PORTF_DIR_R |= RED_LED_MASK;  // bits 1 and 2 are outputs, other pins
        are inputs
    GPIO_PORTF_DR2R_R |= RED_LED_MASK; // set drive strength to 2mA (not needed
        since default configuration -- for clarity)
    GPIO_PORTF_DEN_R |= RED_LED_MASK;

    TIMER2_CTL_R  &= ~TIMER_CTL_TAEN;                 // turn-off timer before
        reconfiguring
    TIMER2_CFG_R   = TIMER_CFG_32_BIT_TIMER;          // configure as 32-bit timer
        (A+B)
    TIMER2_TAMR_R  = TIMER_TAMR_TAMR_PERIOD;          // configure for periodic
        mode (count down)
    TIMER2_TAILR_R = 8000000;                         // configure for 5Hz rate.
        The rate just has to be slower than how often we update we update dist
        variable
    TIMER2_IMR_R   = TIMER_IMR_TATOIM;                // turn-on interrupts
    TIMER2_CTL_R  |= TIMER_CTL_TAEN;
}

void enableTimer()
{
    NVIC_EN0_R = 1 << (INT_TIMER2A-16);               // turn-on interrupt 39 (
        TIMER2A) in NVIC
}

// ISR to update the states of the events
void timer2ISR()
{
    bool is_happening[NUM_SIMPLE];

    int i;
    char buf[16];
    for (i = 0; i < NUM_SIMPLE; ++i)
    {
        simpleEvent se = simple_events[i];
        is_happening[i] = !se.disabled && se.sensor < NUM_SENSORS && dist[se.
            sensor] >= se.min_mm && dist[se.sensor] <= se.max_mm;
    }

    for (i = NUM_COMPOUND - 1; i >= 0 ; --i)
    {
```

```
    compoundEvent ce = compound_events[i];
    if (!ce.disabled && ce.event1 < NUM_SIMPLE && is_happening[ce.event1] &&
         ce.event2 < NUM_SIMPLE && is_happening[ce.event2])
    {
        if (ce.on) {

            snprintf(buf, sizeof(buf), "Event %u!!\n> ", i);
            putsUart0(buf);

            // First, disable reading;
             stopReading();

             uint32_t N = ce.cyc;

            // Actually pulse motor
            while (N--)
            {
                // pulse on...
                setDutyCycle(ce.pwm);
                // wait
                waitMicrosecond(ce.on_time * 1000);
                // turn off
                setDutyCycle(0);
                // wait at low
                waitMicrosecond(ce.off_time * 1000);
            }

            RED_LED = 0;

            waitMicrosecond(50000);
            restartReading();

            TIMER2_ICR_R = TIMER_ICR_TATOCINT;
            return;
        }
    }
}

for (i = NUM_SIMPLE-1; i >= 0; --i)
{
    simpleEvent se = simple_events[i];
    if (!se.disabled && se.sensor < NUM_SENSORS && dist[se.sensor] >= se.
        min_mm && dist[se.sensor] <= se.max_mm) // or if (is_happening[i])
    {
        if (se.on)
        {
            snprintf(buf, sizeof(buf), "Event %u!!\n> ", i);
            putsUart0(buf);

            // First, disable reading;
            stopReading();

            int N = se.cyc;
            // Actually pulse motor
            while (N--)
            {
                // pulse on...
                setDutyCycle(se.pwm);
                // wait
                waitMicrosecond(se.on_time * 1000);
                // turn off
                setDutyCycle(0);
                // wait at low
                waitMicrosecond(se.off_time * 1000);
            }

            RED_LED = 0;

            waitMicrosecond(50000);
            restartReading();

            break;
        }
    }
}
```

```
    TIMER2_ICR_R = TIMER_ICR_TATOCINT;
}

void readEventsFromEeprom ()
{
    // Read the simple events first
    uint32_t offset = 0;

    uint32_t *as_words = (uint32_t *)simple_events;

    while (offset < SE_SIZE_WORDS)
    {
        as_words[offset] = readEeprom(offset);

        offset++;
    }

    as_words = (uint32_t *)compound_events;

    offset = 0;

    while (offset < CE_SIZE_WORDS)
    {
        as_words[offset] = readEeprom(offset + SE_SIZE_WORDS);

        offset++;
    }
}


void writeEventToEeprom (int idx)
{
    // Read the simple events first
    uint32_t offset;
    uint32_t *as_words;
    uint32_t blocks;

    if (idx < NUM_SIMPLE)
    {
        offset = idx * (SE_SIZE_WORDS/NUM_SIMPLE);
        as_words = (uint32_t *)simple_events;

        // Ceil to multiple of 4 to avoid losing data
        blocks = (sizeof(simpleEvent) + 3) >> 2;
    }
    else
    {
        idx -= NUM_SIMPLE;

        offset = SE_SIZE_WORDS + (idx * (CE_SIZE_WORDS/NUM_COMPOUND));
        as_words = (uint32_t *)compound_events;

        blocks = (sizeof(compoundEvent) + 3) >> 2;
    }

    int i;
    for (i = 0; i < blocks; ++i)
    {
        writeEeprom(offset + i, as_words[i]);
    }
}

void writeEventsToEeprom ( void )
{
    int i;
    for (i = 0; i < NUM_EVENTS; ++i)
    {
        writeEventToEeprom(i);
    }
}

void showPatterns()
{
    putsUart0("EVENT no.  PWM       TON       TOFF     BEATS\n");

    int i;
```

```
    bool wont_work[NUM_SIMPLE] = {0};

    char buf[128];
    for (i = 0; i < NUM_SIMPLE; ++i)
    {
        wont_work[i] = false;

        simpleEvent cur = simple_events[i];
        if (cur.pwm == 0xFF)
            snprintf(buf, sizeof(buf), "      %2"PRIu32"       -         -         -
                    -\n", i);
        else
            if (cur.disabled) {
                snprintf(buf, sizeof(buf), "      %2"PRIu8"     %3"PRIu8"*"\
                    " %7"PRIu32"*  %7"PRIu32"* %7"PRIu32"*\n", i, cur.pwm, cur.
                        on_time, cur.off_time, cur.cyc);
                wont_work[i] = true;
            }
            else
                snprintf(buf, sizeof(buf), "      %2"PRIu8"     %3"PRIu8\
                    "  %7"PRIu32"   %7"PRIu32"  %7"PRIu32"\n", i, cur.pwm, cur.
                        on_time, cur.off_time, cur.cyc);

        putsUart0(buf);
    }

    for (i = 0; i < NUM_COMPOUND; ++i)
    {
        compoundEvent cur = compound_events[i];
        if (cur.pwm == 0xFF)
            snprintf(buf, sizeof(buf), "      %2"PRIu32"       -         -         -
                    -\n", i + NUM_SIMPLE);
        else
        {
            if (cur.disabled || wont_work[cur.event1] || wont_work[cur.event2])
                snprintf(buf, sizeof(buf), "      %2"PRIu8"     %3"PRIu8"*"\
                    " %7"PRIu32"*  %7"PRIu32"* %7"PRIu32"*\n", i + NUM_SIMPLE,
                        cur.pwm, cur.on_time, cur.off_time, cur.cyc);
            else
                snprintf(buf, sizeof(buf), "      %2"PRIu8"     %3"PRIu8\
                    "  %7"PRIu32"   %7"PRIu32"  %7"PRIu32"\n", i + NUM_SIMPLE,
                        cur.pwm, cur.on_time, cur.off_time, cur.cyc);
        }

        putsUart0(buf);
    }
}

void showEvents()
{
    putsUart0("EVENT no. SENSOR MIN_DIST MAX_DIST\n");

    int i;
    char buf[128];
    for (i = 0; i < NUM_SIMPLE; ++i)
    {
        simpleEvent cur = simple_events[i];

        if (simple_events[i].disabled)
            snprintf(buf, sizeof(buf), "      %2"PRIu32"         -         -
                -\n", i);
        else
            snprintf(buf, sizeof(buf), "      %2"PRIu32"         %1"PRIu8\
                "  %7"PRIu32"  %7"PRIu32"\n", i, cur.sensor, cur.min_mm, cur
                    .max_mm);

        putsUart0(buf);
    }

    putsUart0("\nEVENT no. EVENT1 EVENT2 SENSOR1 MIN_DIST1 MAX_DIST1 SENSOR2
         MIN_DIST2 MAX_DIST2\n");

    for (i = 0; i < NUM_COMPOUND; ++i)
    {
        compoundEvent cur = compound_events[i];
```

```
            if (compound_events[i].disabled)
                snprintf(buf, sizeof(buf), "      %2"PRIu32"          -       -       -
                        -       -       -           -          -\n", i);
            else
            {

                simpleEvent event1 = simple_events[cur.event1], event2 =
                    simple_events[cur.event2];
                snprintf(buf, sizeof(buf), "      %2"PRIu32"        %2"PRIu32"      %2"
                    PRIu32\
                        "         %1"PRIu8"   %7"PRIu32"    %7"PRIu32"       %1"PRIu8"
                            %7"PRIu32"      %7"PRIu32"\n",
                        i, cur.event1, cur.event2,
                        event1.sensor, event1.min_mm, event1.max_mm,
                        event2.sensor, event2.min_mm, event2.max_mm);
            }

            putsUart0(buf);
        }
    }


    //-----------------------------------------------------------------------------
    // Main
    //-----------------------------------------------------------------------------

    #define DEBUG 1

    int main(void)
    {
        // Initialize hardware
        initHw();

        initMotor();

        initSensors();

        initEeprom();

        // Read the tables from EEPROM
        readEventsFromEeprom();

        enableTimer();

        // Setup UART0 baud rate
        initUart0();
        setUart0BaudRate(115200, 40e6);

        bool show_data = false;
        char str[50];

        putsUart0("> ");
        while (1)
        {
            if (kbhitUart0())
            {
                show_data = false;

                char buf[128];
                getsUart0(&data);

                parseFields(&data);

                bool bad = false;

                if (isCommand(&data, "reboot", 0))
                {
                    NVIC_APINT_R = NVIC_APINT_VECTKEY | NVIC_APINT_SYSRESETREQ;
                }
                else if (isCommand(&data, "show", 1))
                {
                    char *arg = getFieldString(&data, 1);

                    if (_isstreq(arg, "events"))
                    {
```

```
            showEvents();
        }
        else if (_isstreq(arg, "patterns"))
        {
            showPatterns();
        }

        else
            bad = true;
    }
    else if (isCommand(&data, "sensors", 0))
    {
        show_data = true;
    }
    else if (isCommand(&data, "event", 4))
    {
        uint32_t event = getFieldInteger(&data, 1);
        uint32_t sensor = getFieldInteger(&data, 2);
        uint32_t min = getFieldInteger(&data, 3);
        uint32_t max = getFieldInteger(&data, 4);

        if (sensor < NUM_SENSORS)
        {
            if (event < NUM_SIMPLE)
            {
                if (min > max)
                {
                    putsUart0("min cannot be greater than max\n");
                    bad = true;
                }
                else
                {
                    simple_events[event].sensor = sensor;
                    simple_events[event].min_mm = min;
                    simple_events[event].max_mm = max;

                    simple_events[event].disabled = 0;
                    simple_events[event].on = 0;

                    // A default pattern
                    simple_events[event].pwm = 75;
                    simple_events[event].cyc = 5;
                    simple_events[event].on_time = 400;
                    simple_events[event].off_time = 400;
                }

                writeEventToEeprom(event);

                if (DEBUG)
                {
                    snprintf(buf, sizeof(buf), "Event %u set to activate
                        when sensor %u detects an obstacle between %umm
                        and %umm\n",
                                event, sensor, min, max);
                    putsUart0(buf);
                }
            }
            else if (event < NUM_EVENTS)
                putsUart0("Compound events can only be set using the 'and
                    ' command\n");
            else
                bad = true;
        }
        else bad = true;
    }
    else if (isCommand(&data, "and", 3))
    {
        int32_t event = getFieldInteger(&data, 1) - NUM_SIMPLE;
        uint32_t event1 = getFieldInteger(&data, 2);
        uint32_t event2 = getFieldInteger(&data, 3);

        if (event >= 0) {

            compound_events[event].disabled = 0;
            compound_events[event].on = 0;
            compound_events[event].event1 = event1;
```

```
            compound_events[event].event2 = event2;

            if (DEBUG)
            {
                snprintf(buf, sizeof(buf), "Compound event %u set to
                    track simple events %u and %u\n",
                        event, event1, event2);

                putsUart0(buf);
            }

            // A default pattern
            compound_events[event].pwm = 80;
            compound_events[event].cyc = 5;
            compound_events[event].on_time = 500;
            compound_events[event].off_time = 150;

            writeEventToEeprom(event + NUM_SIMPLE);
        }
        else
        {
            putsUart0("Simple events are numbered 0 to 15\n");
        }
    }
    else if (isCommand(&data, "erase", 1))
    {
        char *arg = getFieldString(&data, 1);

        if (_isstreq(arg, "all"))
        {
            resetEeprom();

//                int j;
//                for (j = 0; j < NUM_EV)
            putsUart0("Cleared all events and their patterns\n");
        }
        else {
            uint32_t event = getFieldInteger(&data, 1);

            if (event < NUM_SIMPLE)
            {
                simple_events[event].disabled = 1;
                simple_events[event].pwm = 0xFF;
            }
            else if ((event - NUM_SIMPLE) < NUM_COMPOUND)
            {
                compound_events[event - NUM_SIMPLE].disabled = 1;
                compound_events[event - NUM_SIMPLE].pwm = 0xFF;
            }

            writeEventToEeprom(event);
        }
    }
    else if (isCommand(&data, "pattern", 5))
    {
        uint32_t event = getFieldInteger(&data, 1);
        uint32_t intensity = getFieldInteger(&data, 2);
        uint32_t cycles = getFieldInteger(&data, 3);
        uint32_t on = getFieldInteger(&data, 4);
        uint32_t off = getFieldInteger(&data, 5);

        if (event < NUM_SIMPLE)
        {
            simple_events[event].pwm = intensity;
            simple_events[event].cyc = cycles;
            simple_events[event].on_time = on;
            simple_events[event].off_time = off;
        }
        else if ((event - NUM_SIMPLE) < NUM_COMPOUND)
        {
            compound_events[event - NUM_SIMPLE].pwm = intensity;
            compound_events[event - NUM_SIMPLE].cyc = cycles;
            compound_events[event - NUM_SIMPLE].on_time = on;
            compound_events[event - NUM_SIMPLE].off_time = off;
        }
```

```
                writeEventToEeprom(event);
            }
            else if (isCommand(&data, "haptic", 2))
            {
                uint32_t event = getFieldInteger(&data, 1);
                char *arg = getFieldString(&data, 2);

                if (event < NUM_SIMPLE)

                    if (_isstreq(arg, "on"))
                        simple_events[event].on = 1;
                    else if (_isstreq(arg, "off"))
                        simple_events[event].on = 0;
                    else
                        bad = true;

                else if ((event - NUM_SIMPLE) < NUM_COMPOUND)

                    if (_isstreq(arg, "on"))
                        compound_events[event - NUM_SIMPLE].on = 1;
                    else if (_isstreq(arg, "off"))
                        compound_events[event - NUM_SIMPLE].on = 0;
                    else
                        bad = true;

                writeEventToEeprom(event);
            }
            else
            {
                putsUart0("Unrecognized command or not enough arguments\n");
            }

            if (bad)
            {
                putsUart0("Invalid argument for ");
                putsUart0(getFieldString(&data, 0));
                putsUart0(" command\n");
            }

            // prepare to read next input;
            putsUart0("> ");
        }

        if (show_data)
        {
            putsUart0("\n");
            snprintf(str, sizeof(str), "Channel 0: %5"PRIu32"mm\n", dist[0]);
            putsUart0(str);
            snprintf(str, sizeof(str), "Channel 1: %5"PRIu32"mm\n", dist[1]);
            putsUart0(str);
            snprintf(str, sizeof(str), "Channel 2: %5"PRIu32"mm\n> ", dist[2]);
            putsUart0(str);

            // wait for the user to actually read the data;
            waitMicrosecond(500000);
        }
        else
            waitMicrosecond(100000);
    }
}
```

4 Observations

The only observations that I noticed while designing and building this project was how unreliable the ultrasonic sensors are. Every once in a while, there will be a spike in the readings and some times the readings are wrong altogether if the obstacle is a smoother surface. However, for our case where we

only need a few centimeters of precision, this is fine (as long as the object can reflect the incoming sound waves)

5 Conclusions

In conclusion, I was able to build a haptic walking aid with user interaction and is actually fairly usable. I learned various embedded systems techniques and concepts along the way. It was truly a fun and educative project and I cannot wait for more like this