Lab 08

LIDAR

Jordan Pinson - 1002209221
Bryan Gonzalez - 1001443032
CSE 4355- Section 001
Dr. Jason Losh
Fall 2025

TM4C Code

Code Explanation

1. `initHW()` sets the system clock to 40 MHz, initializes UART0 (PC/terminal), UART1 (LIDAR), and a test LED module.

2. `send_STOP()`, `send_INFO()`, `send_SCAN()` send the LIDAR command bytes via UART1 (`0xA5` followed by the command code).

3. `toAsciiDecimal()` and `toAsciiHex()` convert integers into ASCII strings for UART printing.

4. `clearFIFO()` is intended to flush the LIDAR UART receive FIFO.

5. `main()`:

    - clears the terminal,

    - pulses an LED (emulate the motor pulse),

    - sends STOP, then INFO, reads 27 bytes of the INFO response and prints device descriptor, model, firmware, hardware, and serial number.

    - pulses the LED again, waits for motor spin-up (3 seconds).

    - enters an infinite loop:

        - clears FIFO, sends SCAN,

        - reads the scan response header (7 bytes) and then `SAMPLE_SIZE` samples (each sample 5 bytes)

        - decodes `angle` and `distance` per sample and stores them in arrays (skips zero distances by decrementing `i` to retry)

        - sends STOP

        - prints the raw readings

        - bubble sorts the (angle,distance) pairs by angle

- prints sorted readings
- computes the scanned area by summing the triangular wedge area between adjacent points using `0.5 * d0 * d1 * sin(deltaAngle)`
- prints `Area`
- loops to start again.

```
//-----------------------------------------------------------------------------
// Main
//-----------------------------------------------------------------------------

int main(void)
{
    char    str[128];
    uint8_t buff[32];
    int i, j;

//    int16_t quality[SAMPLE_SIZE];
    int16_t angle[SAMPLE_SIZE];
    int16_t distance[SAMPLE_SIZE];

    initHW();
    putsUart0("\033[2J\033[H"); //clear terminal
    putsUart0("Lab 8 - LIDAR\r\n\n");

    /*=========={ Pulse Motor }=====================================*/
    BLUE_LED = 1;
    waitMicrosecond(1e6);
    BLUE_LED = 0;

    /*=========={ Getting device info }=============================*/
    send_STOP();
    waitMicrosecond(1e3);

    clearFIFO();
    send_INFO();


    for(i = 0; i < 27; i++)
    {
        buff[i] = getcUart1();
    }
```

```
/*=========={ Parsing Device Info }==============================*/
putsUart0("Response Descriptor: \n\r\t");
for(i = 0; i < 7; i++)
{
    putsUart0(toAsciiHex(str, buff[i]));
    putsUart0("\n\r\t");
}
putsUart0("\n\r");

putsUart0("model Number: ");
putsUart0(toAsciiDecimal(str, buff[7], 3));
putsUart0("\n\r");

putsUart0("firmware minor: ");
putsUart0(toAsciiDecimal(str, buff[8], 3));
putsUart0("\n\r");

putsUart0("firmware major: ");
putsUart0(toAsciiDecimal(str, buff[9], 3));
putsUart0("\n\r");

putsUart0("hardware: ");
putsUart0(toAsciiDecimal(str, buff[10], 3));
putsUart0("\n\r");

putsUart0("Serial Number: \n\r\t");
for(i = 0; i < 16; i++)
    {
        putsUart0(toAsciiHex(str, buff[i]));
        putsUart0("\n\r\t");
    }
putsUart0("\n\r");


/*=========={ Starting Motor }===================================*/
BLUE_LED = 1;
waitMicrosecond(3e6); // wait to get up to speed
```

```
while(1)
{
    /*=========={ Start scan }==================================*/
    clearFIFO();
    send_SCAN();
    putsUart0("Starting to scan... \n\r");
    for(i = 0; i < 7; i++)
    {
        buff[i] = getcUart1();
    }

    int j;
    for(i = 0; i < SAMPLE_SIZE; i++)
    {
        for(j = 0; j < 5; j++)
        {
            buff[j] = getcUart1();
        }

        uint16_t a = (((buff[2]>>1) & 0xff) << 8) | (buff[1] & 0xff);
        a = a/64;

        uint16_t d =  ((buff[4] & 0xff)<<8) | (buff[3] & 0xff);
        d = d/4;


        if(d != 0)
        {
            distance[i] = d;
            angle[i] = a;
        }
        else        i--;
    }
    send_STOP();

    putsUart0("Output...\n\r");
    for(i = 0; i < SAMPLE_SIZE; i++)
    {

            putsUart0(toAsciiDecimal(str, angle[i], 6));
            putsUart0("\t");
            putsUart0(toAsciiDecimal(str, distance[i], 6));
            putsUart0("\n\r");
    }
```

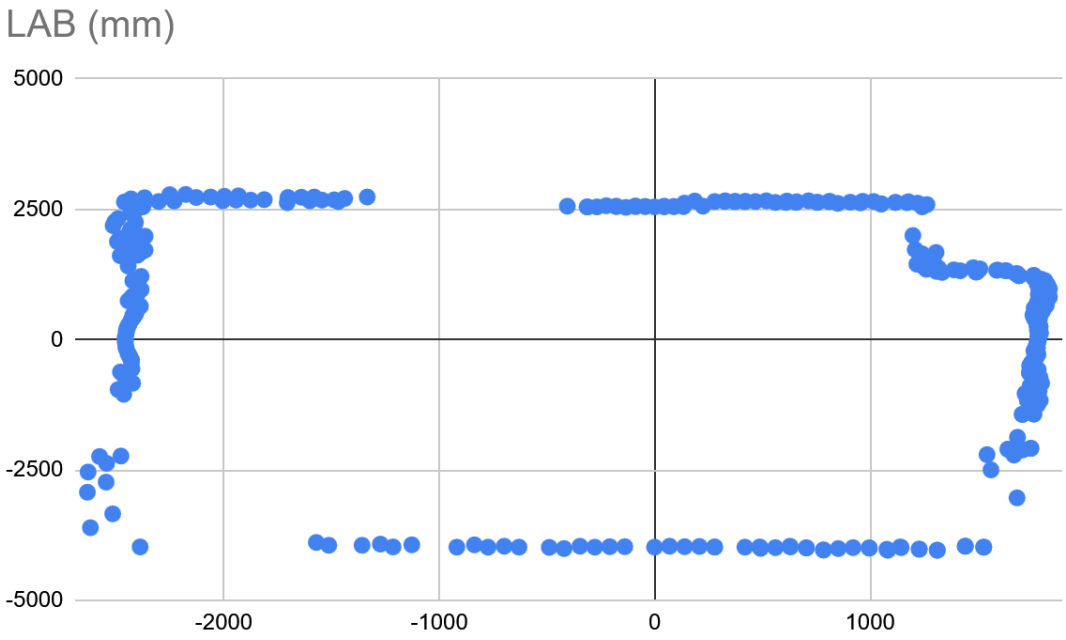
```
/*=========={ Bubble Sort (because I'm lazy but also I have more control) }================================*/
uint16_t tmp;
for(i = 0; i < SAMPLE_SIZE; i++)
{
    for(j = 0; j < SAMPLE_SIZE-1 - i; j++)
    {
        if(angle[j] > angle[j+1])
        {
            tmp = angle[j];
            angle[j] = angle[j+1];
            angle[j+1] = tmp;

            tmp = distance[j];
            distance[j] = distance[j+1];
            distance[j+1] = tmp;
        }
    }
}

putsUart0("Sorted...\n\r");
for(i = 0; i < SAMPLE_SIZE; i++)
{

        putsUart0(toAsciiDecimal(str, angle[i], 6));
        putsUart0("\t");
        putsUart0(toAsciiDecimal(str, distance[i], 6));
        putsUart0("\n\r");
}
```

```
/*=========={ Compute Area }================================*/
uint32_t area = 0;
for(i = 1; i < SAMPLE_SIZE-1; i++)
{
    float d0 = (float)distance[i] * 0.0393701; //convert to inches
    float d1 = (float)distance[i+1] * 0.0393701; //convert to inches

    float a0 = 2 * 3.14159 * (float)angle[i] /360; //Convert to radians
    float a1 = 2 * 3.14159 * (float)angle[i+1] /360; //Convert to radians

    area += 0.5 * d0 * d1 * sin(a1-a0);
}

putsUart0("\nArea: \n\r");
putsUart0(toAsciiDecimal(str, area, 6));
putsUart0(" in^2\n\r");
putsUart0("done\r\n");
```

Measurements

Angle (degrees)	Distance (mm)	Angle (radians)	Plot X	Plot Y	Distance (in)	Triangle Areas (in^2)
0	1781	0	1781	0	70.11815	42.87877
1	1780	0.017453293	1779.729	31.06528	70.07878	0
1	1781	0.017453293	1780.729	31.08274	70.11815	42.90286
2	1781	0.034906585	1779.915	62.156	70.11815	0
2	1785	0.034906585	1783.913	62.2956	70.27563	43.0475
3	1783	0.052359878	1780.556	93.31501	70.19689	0
3	1785	0.052359878	1782.554	93.41968	70.27563	43.31308
4	1794	0.06981317	1789.63	125.1431	70.62996	0
4	1795	0.06981317	1790.627	125.2129	70.66933	86.75834
6	1787	0.104719755	1777.211	186.7924	70.35437	0
6	1795	0.104719755	1785.167	187.6286	70.66933	43.62856
7	1797	0.122173048	1783.605	218.9992	70.74807	0
7	1797	0.122173048	1783.605	218.9992	70.74807	43.89592
8	1806	0.13962634	1788.424	251.3466	71.1024	0
8	1806	0.13962634	1788.424	251.3466	71.1024	43.89592
9	1797	0.157079633	1774.876	281.1127	70.74807	0
9	1797	0.157079633	1774.876	281.1127	70.74807	44.01745
10	1811	0.174532925	1783.487	314.4768	71.29925	0
10	1810	0.174532925	1782.502	314.3032	71.25988	132.8068
13	1809	0.226892803	1762.635	406.9365	71.22051	0
13	1809	0.226892803	1762.635	406.9365	71.22051	0
13	1834	0.226892803	1786.995	412.5602	72.20476	45.51911
14	1835	0.244346095	1780.493	443.9267	72.24413	45.14682
15	1819	0.261799388	1757.019	470.7918	71.61421	0
15	1818	0.261799388	1756.053	470.533	71.57484	45.39248
16	1846	0.27925268	1774.489	508.8266	72.6772	0

16	1846	0.27925268	1774.489	508.8266	72.6772	46.99046
17	1882	0.296705973	1799.766	550.2435	74.09453	0
17	1883	0.296705973	1800.722	550.5359	74.1339	47.47386
18	1864	0.314159265	1772.769	576.0077	73.38587	46.96963
19	1863	0.331612558	1761.501	606.5335	73.3465	0
19	1900	0.331612558	1796.485	618.5795	74.80319	48.82762
20	1900	0.34906585	1785.416	649.8383	74.80319	0
20	1935	0.34906585	1818.305	661.809	76.18114	50.74779
21	1939	0.366519143	1810.212	694.8755	76.33862	50.51175
22	1926	0.383972435	1785.756	721.4923	75.82681	0
22	1917	0.383972435	1777.411	718.1208	75.47248	50.535
23	1949	0.401425728	1794.064	761.535	76.73232	0
23	1955	0.401425728	1799.587	763.8794	76.96855	53.04397
…						
357	1777	6.23082543	1774.565	-93.001	69.96067	42.68632
358	1776	6.248278722	1774.918	-61.9815	69.9213	0
358	1776	6.248278722	1774.918	-61.9815	69.9213	42.75839
359	1780	6.265732015	1779.729	-31.0653	70.07878	0



Pictures