Introduction to Embedded Systems [CSE211s]

GPS Tracking System
Second Milestone

Name	ID
Mina Nasser Shehata	2100370
Mostafa Hamada Hassan Ahmed	2100587
Zeyad Magdy Roushdy	2100393
Ahmed Ashraf Ahmed	2100476
Mina Antony Samy Fahmy	2101022
Rana Ayman Hamdy	2100830
Farah Amr Mohamed	2101034
Mohamed Essam ElSayed	2101080
Mentalla hussein Sedik	2100872

Contents

List of Source Codes

List of Figures

1 Introduction

1.1 Project Description

This project aims to develop a GPS system using TM4C123G LaunchPad to store and plot motion trajectory. The coordinates of the motion trajectory and distance from starting point are recorded by the GPS module and are stored in the internal EEPROM of the TM4C123G LaunchPad. The resulting trajectory is then plotted using a Python program on the host laptop which the LaunchPad is connected to.
The GPS system has 4 modes:

- No Mode (Represented by Blue LED): The GPS system is waiting for user to press either SW1 or SW2 to enter Accumulating Mode or Retrieving Mode respectively
- Accumulating Mode (Represented by Red LED): The GPS system begins sets current position as a starting point, displays distance from starting point on the LCD, and sends motion trajectory to EEPROM.
- Retrieving Mode (Represented by Yellow LED): The GPS system waits for a trigger from the host laptop (Character **u**) and once triggered, sends EEPROM contents to the Python Trajectory drawer.
- Destination Reached Mode (Represented by green LED): This mode indicates that the destination has been reached either by passing the 100m mark from starting point or by manually pressing SW1 from Accumulating Mode. This mode is only active for brief moments before automatically switching again to No Mode.

There are two cases for the operation of the GPS system:

- Motion trajectory $<$ 100m:
  Starting with No Mode, we will switch to Accumulating mode (using SW1) to begin recording the trajectory (which is being stored in EEPROM) and distance from starting point (displayed on LCD). We then will manually record the destination point (using SW1) when it is reached. The GPS system switches to Destination Reached Mode then to No Mode. Then we can either run the Python trajectory drawer and switch to retrieving mode (using SW2) or switch to retrieving mode first and the run the Python trajectory drawer. Once the Python trajectory

drawer is run, it keeps sending triggers (character `u`) to the launchPad. If the launchPad is in retrieving mode, the stored trajectory in EEPROM is then sent to the Python trajectory drawer and the result is plotted.

- Motion trajectory > 100m:
  Similarly to the first case, we will start with No Mode and switch to Accumulating mode (using SW1) to begin recording the trajectory (which is being stored in EEPROM) and distance from starting point (displayed on LCD). The difference from the first case here is that after passing the 100m mark, the GPS system automatically switches to Destination Reached Mode then to No Mode. Then similarly to the first case, we can either run the Python trajectory drawer and switch to retrieving mode (using SW2) or switch to retrieving mode first and the run the Python trajectory drawer. Once the Python trajectory drawer is run, it keeps sending triggers (character `u`) to the launchPad. If the launchPad is in retrieving mode, the stored trajectory in EEPROM is then sent to the Python trajectory drawer and the result is plotted.

1.2 Project Structure

This project is divided into 4 Layers:

- Application Layer (APP)
  This layer includes the `main` function which calls functions from both HAL and MCAL layers
- Microcontroller Abstraction Layer (MCAL)
  This layer includes functions that control internal microcontroller peripherals like:
  - GPIO
  - Systick
  - UART0
  - UART5
  - EEPROM

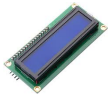
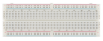
- Hardware Abstraction Layer (HAL) This layer includes functions that control external hardware peripherals like:
  - GPS Module
  - RGB LED
  - Switches
  - LCD
- Utilities This layer includes header files shared amongst all other layers, like:
  - STD_TYPES.h : contains `typedef`s for several datatypes
  - BIT_UTILITIES.h : contains multiple macros for manipulating/reading registers and certain bits of registers
  - Add To Print in Debug Serial : Embedded printf
  - Conversion.c Conversion.h : Conversion from string to float and vice versa
  - Hardfault: FPU handling

Additionally, we have a trajectory drawer written in Python in `TrajectoryDraw` folder.

1.3 Github Repository

You can find our project on this Github repository
https://github.com/minanasser54/ARM_M4_GPS
Each team member has contributed with one or more pull requests.

1.4 Components Used

Component	Quantity
Tiva C Series TM4C123G LaunchPad Evaluation Kit 	1
LCD display (16x2) 	1
Ublox NEO-6m GPS Module 	1
Resistors 	Multiple
Jumper Wires 	Multiple
Breadboard 	1

2 Source Code

2.1 APP

Source Code 2.1.1: main.c

```c
#include "../MCAL/GPIO/DIO_Driver.h"
#include "../MCAL/STK/Systick.h"
#include "../HAL/SW/SW.h"
#include "../HAL/SW/ext_sw.h"
#include "../HAL/GPS/GPS.h"
#include "../UTILITIES/Bit_Utilities.h"
#include "../UTILITIES/Conversion.h"
#include "../HAL/LED/LED_interface.h"
#include "../MCAL/UART0/UART0_PC.h"
#include "../MCAL/UART5/UART5.h"
#include "../MCAL/EEPROM/eeprom.h"
#include "../HAL/LCD/LCD.h"
#include "SAVE.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>

float Distance(float currentA , float currentB , float destA
↪  ,float destB);
float ToRad(float DegNum);
float ToDeg(float Num);

char* floatToString(float num, char* buffer, int bufferSize) {
    if (bufferSize < 32) {
        return NULL; // Buffer size too small
    }
    sprintf(buffer, "%f", num);
    return buffer;
}
void check_mode(int *ptr){
    if(!SW_ispressed(SW1)){
        *ptr=1;}
    if(!SW_ispressed(SW2)) {
        *ptr=2;}}

int main(void){
  ex_sw_init();
  eeprom_init();
```

```
  UART0_INIT();
  UART5Init();
  SysTick_Init();
  LED_LedInit();
  SW_Init(SW1);
  SW_Init(SW2);
  LCD_INIT();
  LCD_CMD(clear_display);
  LCD_CMD(cursorBlink);
  int mode =0;
  int *p = & mode;
  while(1){
    check_mode(p);
    if (mode == 1){
    float distance=0.0;
    float dLon=0.0;
    float dLat=0.0;
    float cLon=0.0;
    float cLat=0.0;

    LED_OffAll();
    LED_LedOn(LED_RED);
    LCD_CMD(clear_display);
    LCD_String("Accumulation Mode");
    delay_milli(2000);
    do{
        GPS_format();
        cLon=atof(getCurrentLongitude());
        cLat=atof(getCurrentLatitude());

        if(dLon≠0 && dLat≠0){
          LCD_CMD(clear_display);
          LCD_String("accumulating !!");
          distance+=Distance(cLon, cLat ,dLon  ,dLat); //data
           ↪  from gps
          delay_milli(500);
          save_eeprom(cLon);
          delay_milli(500);
          save_eeprom(cLat);
          delay_milli(500);
          char str[32];
          LCD_CMD(clear_display);
          delay_milli(500);
          LCD_String("D= ");
          floatToString(distance, str, sizeof(str));
```

```
          LCD_String(str); //display distance num on LCD
          delay_milli(3000);
          LCD_CMD(clear_display);
//                    printstr("\n dis = ");
//                    printstr(str);
//                    printstr("\n");
        }
        if (cLon && cLat){
        dLon =cLon;
        dLat =cLat;
        }
              //printstr(" \n SS");
        SysTick_Wait(4000);
              //printstr(" EE  \n");
        check_mode(p);
        if (!SW_ispressed(SW1) || mode≠1) break;
           }while(distance≤100 );

      //LED_LedOff(LED_RED);
      LED_OffAll();
      LED_LedOn(LED_GREEN);
      SysTick_Wait(5000);
      LED_OffAll();
      mode=0;
    }
    if (mode == 2){
      LED_OffAll();
      LED_LedOn(LED_GREEN);
      LED_LedOn(LED_RED);
      LCD_CMD(clear_display);
      LCD_String("Retrieving Mode");
      bool wait =true;

      do{
      wait=true;
      if((UART0_read()=='U')){
      wait=false;
      fetch_eeprom();
      eeprom_clear();
      }else{
      check_mode(p);}
      }while(wait && mode==2);
      mode=0;
      LCD_CMD(clear_display);
      LCD_String("Retrieving DONE");
```

```
      SysTick_Wait(3000);
      LCD_CMD(clear_display);
      LED_OffAll();
    }


    //LED_OffAll();
    LED_LedOn(LED_BLUE);
    LCD_CMD(clear_display);
    LCD_String("sw1 acc sw2 retrive");
    SysTick_Wait(10000);
   }
}

float ToDeg(float Num){
  int degree =(int) Num/100;
  float minutes =Num-(float)degree*100;
  return(degree+(minutes/60));
}

float ToRad(float DegNum){
    return (DegNum*(PI/180));
}
//float truncate(float copy_f32FloatValue)
//{
//    copy_f32FloatValue = floor(copy_f32FloatValue * 1000000) /
↪  (float)1000000.0;
//    return copy_f32FloatValue;
//}
//float trun(float copy_f32FloatValue)
//{
//    copy_f32FloatValue = floor(copy_f32FloatValue * 1000) /
↪  (float)1000.0;
//    return copy_f32FloatValue;
//}
float Distance(float currentA , float currentB , float destA
↪  ,float destB){
    float currentA_RAD = ToRad(ToDeg(currentA));
    float currentB_RAD = ToRad(ToDeg(currentB));
    float destA_RAD = ToRad(ToDeg(destA));
    float destB_RAD = ToRad(ToDeg(destB));
    float aDiff=destA_RAD-currentA_RAD;
    float bDiff=destB_RAD-currentB_RAD;
    float a=pow(sin(bDiff/2),2)+cos(currentB_RAD)*cos(destB_RAD
    ↪  )*pow(sin(aDiff/2),2);
```

```
    float Dis=2*atan2(sqrt(a),sqrt(1-a));
    return EARTH_RADIUS*Dis;
}
```

Source Code 2.1.2: SAVE.h

```
#ifndef __SAVE__
#define __SAVE__

void save_eeprom(float d1);
void fetch_eeprom(void);

#endif
```

Source Code 2.1.3: SAVE.c

```
#include "SAVE.h"
#include <stdint.h>
#include "../MCAL/EEPROM/eeprom.h"
#include "../MCAL/UART0/UART0_PC.h"

static uint8_t blk =0;
static uint8_t addr =0;


void save_eeprom(float d1){
  uint32_t d2 = (uint32_t)(d1*10000);
  if (blk < 32){
       eeprom_write(d2,addr++, blk);
  if (addr == 16) {
        blk++;
        addr = 0; // Reset address to start of the block
  }
  }else{}
}


void fetch_eeprom(void){
  char i;
  char j;
```

```
  for(i=0;i<32;i++){
    for(j=0;j<16;j++){
      char buffer[11]; // Make sure this is large enough to hold
       ↪  the converted string
      if (eeprom_read(j,i) ≠0){
      sprintf(buffer, "%lu", eeprom_read(j,i));
      printstr(buffer);
      printstr("\n");
    }
  }
}
  printstr("saved \n ");
}
```

2.2 MCAL

2.2.1 GPIO

Source Code 2.2.1: DIO_Driver.h

```
#ifndef DIODriver
#define DIODriver

#include "../../UTILITIES/tm4c123gh6pm.h"
#include "../../UTILITIES/Bit_Utilities.h"

//*************************************************************
 ↪  ****************
//
// Macros for PORTS
//
//*************************************************************
 ↪  ****************
#define DIO_PORTA      0
#define DIO_PORTB      1
#define DIO_PORTC      2
#define DIO_PORTD      3
#define DIO_PORTE      4
#define DIO_PORTF      5
//*************************************************************
 ↪  ****************
//
```

```
// Macros for PINS value
//
//*************************************************************
****************
#define DIO_OUTPUT        1
#define DIO_INPUT      0
#define DIO_HIGH      1
#define DIO_LOW        0
//*************************************************************
****************

void DIO_vPORTINIT(unsigned char portname);

void DIO_vSETPINDIR(unsigned char portname , unsigned char pin
 , unsigned char dir);

void DIO_vWRITEPIN(unsigned char portname , unsigned char pin ,
 unsigned char dir);


#endif

```

Source Code 2.2.2: DIO_Driver.c

```
#include "../../UTILITIES/tm4c123gh6pm.h"
#include "../../UTILITIES/Bit_Utilities.h"
#include "DIO_Driver.h"

void DIO_vPORTINIT(unsigned char portname){
  switch(portname){
    case 'A':
    case 'a':
      SET_BIT(SYSCTL_RCGCGPIO_R,0);
      while(READ_BIT(SYSCTL_PRGPIO_R,0)==0);
      GPIO_PORTA_LOCK_R |= 0x4c4f434b;
      GPIO_PORTA_CR_R |= 0xFF;
      GPIO_PORTA_DEN_R |= 0xFF;
      GPIO_PORTA_AMSEL_R &= ~0xFF;
      break;
    case 'B':
    case 'b':
      SET_BIT(SYSCTL_RCGCGPIO_R,1);
```

```
      while(READ_BIT(SYSCTL_PRGPIO_R,1)==0);
      GPIO_PORTB_LOCK_R |= 0x4c4f434b;
      GPIO_PORTB_CR_R |= 0xFF;
      GPIO_PORTB_DEN_R |= 0xFF;
      GPIO_PORTB_AMSEL_R &= ~0xFF;
      break;
    case 'C':
    case 'c':
      SET_BIT(SYSCTL_RCGCGPIO_R,2);
      while(READ_BIT(SYSCTL_PRGPIO_R,2)==0);
      GPIO_PORTC_LOCK_R |= 0x4c4f434b;
      GPIO_PORTC_CR_R |= 0xFF;
      GPIO_PORTC_DEN_R |= 0xFF;
      GPIO_PORTC_AMSEL_R &= ~0xFF;
      break;
    case 'D':
    case 'd':
      SET_BIT(SYSCTL_RCGCGPIO_R,3);
      while(READ_BIT(SYSCTL_PRGPIO_R,3)==0);
      GPIO_PORTD_LOCK_R |= 0x4c4f434b;
      GPIO_PORTD_CR_R |= 0xFF;
      GPIO_PORTD_DEN_R |= 0xFF;
      GPIO_PORTD_AMSEL_R &= ~0xFF;
      break;
    case 'E':
    case 'e':
      SET_BIT(SYSCTL_RCGCGPIO_R,4);
      while(READ_BIT(SYSCTL_PRGPIO_R,4)==0);
      GPIO_PORTE_LOCK_R |= 0x4c4f434b;
      GPIO_PORTE_CR_R |= 0xFF;
      GPIO_PORTE_DEN_R |= 0xFF;
      GPIO_PORTE_AMSEL_R &= ~0xFF;
      break;
    case 'F':
    case 'f':
      SET_BIT(SYSCTL_RCGCGPIO_R,5);
      while(READ_BIT(SYSCTL_PRGPIO_R,5)==0);
      GPIO_PORTF_LOCK_R |= 0x4c4f434b;
      GPIO_PORTF_CR_R |= 0xFF;
      GPIO_PORTF_DEN_R |= 0xFF;
      GPIO_PORTF_AMSEL_R &= ~0xFF;
      GPIO_PORTF_PUR_R    = 0x11;
      break;
  }
}
```

```
void DIO_vSETPINDIR(unsigned char portname , unsigned char pin
↪  , unsigned char dir){
  switch (portname){
    case'A':
    case'a':
      if (dir == 1 )
        SET_BIT(GPIO_PORTA_DIR_R, pin);
      else
        CLR_BIT(GPIO_PORTA_DIR_R, pin);
      break;
    case'B':
    case'b':
      if (dir == 1 )
        SET_BIT(GPIO_PORTB_DIR_R, pin);
      else
        CLR_BIT(GPIO_PORTB_DIR_R, pin);
      break;
    case'C':
    case'c':
      if (dir == 1 )
        SET_BIT(GPIO_PORTC_DIR_R, pin);
      else
        CLR_BIT(GPIO_PORTC_DIR_R, pin);
      break;
    case'D':
    case'd':
      if (dir == 1 )
        SET_BIT(GPIO_PORTD_DIR_R, pin);
      else
        CLR_BIT(GPIO_PORTD_DIR_R, pin);
      break;
    case'E':
    case'e':
      if (dir == 1 )
        SET_BIT(GPIO_PORTE_DIR_R, pin);
      else
        CLR_BIT(GPIO_PORTE_DIR_R, pin);
      break;
    case'F':
    case'f':
      if (dir == 1 )
        SET_BIT(GPIO_PORTF_DIR_R, pin);
      else
        CLR_BIT(GPIO_PORTF_DIR_R, pin);
      break;
```

```
  }
}
void DIO_vWRITEPIN(unsigned char portname , unsigned char pin ,
↪  unsigned char dir){
  switch (portname){
    case'A':
    case'a':
      if (dir == 1 )
        SET_BIT(GPIO_PORTA_DATA_R, pin);
      else
        CLR_BIT(GPIO_PORTA_DATA_R, pin);
      break;
    case'B':
    case'b':
      if (dir == 1 )
        SET_BIT(GPIO_PORTB_DATA_R, pin);
      else
        CLR_BIT(GPIO_PORTB_DATA_R, pin);
      break;
    case'C':
    case'c':
      if (dir == 1 )
        SET_BIT(GPIO_PORTC_DATA_R, pin);
      else
        CLR_BIT(GPIO_PORTC_DATA_R, pin);
      break;
    case'D':
    case'd':
      if (dir == 1 )
        SET_BIT(GPIO_PORTD_DATA_R, pin);
      else
        CLR_BIT(GPIO_PORTD_DATA_R, pin);
      break;
    case'E':
    case'e':
      if (dir == 1 )
        SET_BIT(GPIO_PORTE_DATA_R, pin);
      else
        CLR_BIT(GPIO_PORTE_DATA_R, pin);
      break;
    case'F':
    case'f':
      if (dir == 1 )
        SET_BIT(GPIO_PORTF_DATA_R, pin);
      else
```

```
        CLR_BIT(GPIO_PORTF_DATA_R, pin);
      break;
  }
}
void DIO_vTOGGLEPIN(unsigned char portname , unsigned char pin
↪  ){
  switch (portname){
    case'A':
    case'a':
      TOGGLE_BIT(GPIO_PORTA_DATA_R, pin);
      break;
    case'B':
    case'b':
      TOGGLE_BIT(GPIO_PORTB_DATA_R, pin);
      break;
    case'C':
    case'c':
      TOGGLE_BIT(GPIO_PORTC_DATA_R, pin);
      break;
    case'D':
    case'd':
      TOGGLE_BIT(GPIO_PORTD_DATA_R, pin);
      break;
    case'E':
    case'e':
      TOGGLE_BIT(GPIO_PORTE_DATA_R, pin);
      break;
    case'F':
    case'f':
      TOGGLE_BIT(GPIO_PORTF_DATA_R, pin);
      break;
  }
}
```

2.2.2 Systick

Source Code 2.2.3: Systick.h

```
#ifndef __Systick__
#define __Systick__


```

```
5
6   void SysTick_Init(void);
7   void systick_wait_1s(void);
8   void systick_wait_1ms(void);
9   void delay_IN_ms(int total);
10  void delay_IN_s(int total);
11  void systick_wait_1MICROs(void);
12  void delay_IN_MICROs(int total);
13  void SysTick_Wait(unsigned long delay);
14
15
16  #endif
```

Source Code 2.2.4: Systick.c

```
1   #include "../../UTILITIES/tm4c123gh6pm.h"
2   #include "Systick.h"
3
4   void SysTick_Init(void){
5     NVIC_ST_CTRL_R = 0;
6     NVIC_ST_CTRL_R = 0x00000005;
7   }
8
9    void systick_wait_1s(void)
10   {
11     NVIC_ST_RELOAD_R = (16000000) - 1;
12     NVIC_ST_CURRENT_R = 0X00;
13     while ((NVIC_ST_CTRL_R & NVIC_ST_CTRL_COUNT) == 0x00){}
14   }
15   void systick_wait_1ms(void)
16   {
17     NVIC_ST_RELOAD_R = (16000) - 1;
18     NVIC_ST_CURRENT_R = 0X00;
19     while ((NVIC_ST_CTRL_R & NVIC_ST_CTRL_COUNT) == 0x00) {}
20   }
21   void systick_wait_1MICROs(void)
22   {
23     NVIC_ST_RELOAD_R = (16) - 1;
24     NVIC_ST_CURRENT_R = 0X00;
25     while ((NVIC_ST_CTRL_R & NVIC_ST_CTRL_COUNT) == 0x00) {}
26   }
27
28   void delay_IN_ms(int total)
```

```
 {
   int i;
   for (i = 0; i < total; i++)
   {
     systick_wait_1ms();
   }
 }
 void delay_IN_MICROs(int total)
 {
   int i;
   for (i = 0; i < total; i++)
   {
     systick_wait_1MICROs();
   }
 }
 void delay_IN_s(int total)
 {
   int i;
   for (i=0; i < total; i++)
   {
     systick_wait_1s();
   }
 }

void SysTick_Wait(unsigned long delay){
  unsigned long i;
  for(i=0;i<delay;i++){
    systick_wait_1ms();
    //SysTick_Wait_10ms(80000);
  }
}
```

2.2.3 UART0

Source Code 2.2.5: UART0_PC.h

```
#include "stdint.h"

#ifndef __UART0__
#define __UART0__

#define GPIO_PA10_M 0X03
```

```
void UART0_INIT(void);
uint8_t UART0_Available(void);
char UART0_read(void);
void UART0_write(char data);
void getCommand(char *str,uint8_t maxLen);
void printstr(char *str);



#endif
```

Source Code 2.2.6: UART0_PC.c

```
#include "../../UTILITIES/tm4c123gh6pm.h"

#include "string.h"
#include "UART0_PC.h"

void UART0_INIT(){
 SYSCTL_RCGCUART_R |=SYSCTL_RCGCUART_R0;
 SYSCTL_RCGCGPIO_R |=SYSCTL_RCGCGPIO_R0;
 UART0_CTL_R &= ~UART_CTL_UARTEN;
 UART0_IBRD_R = 104;
 UART0_FBRD_R = 11;
 UART0_LCRH_R = (UART_LCRH_WLEN_8 | UART_LCRH_FEN);
 UART0_CTL_R |= (UART_CTL_UARTEN | UART_CTL_RXE | UART_CTL_TXE);
 GPIO_PORTA_AFSEL_R |= 0X03;
 GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R & ~0xff) |
  ↪  (GPIO_PCTL_PA0_U0RX  | GPIO_PCTL_PA1_U0TX);
 GPIO_PORTA_DEN_R |=0x03;
}


uint8_t UART0_Available(void){
  return ((UART0_FR_R &UART_FR_RXFE)==UART_FR_RXFE)? 0:1;
}
char UART0_read(void){
   while(UART0_Available() ≠1){};
    return UART0_DR_R & 0xFF;
    }

void UART0_write(char data){
  while ((UART0_FR_R&UART_FR_TXFF)≠0);
```

```
  UART0_DR_R = data;
}

void getCommand(char *str,uint8_t maxLen){
  char c;
  int8_t i;
  for(i =0;i<maxLen;i++){
    c = UART0_read();
    if(c=='\n' || c=='\r'){ break;
    }
    else str[i]=c;
    UART0_write(c);

  }

}
void printstr(char *str){
  while(*str){
    UART0_write(*str);
    str++;
  }}


```

2.2.4 UART5

Source Code 2.2.7: UART5.h

```
#ifndef __UART5__
#define __UART5__
#include "stdint.h"

void UART5Init(void);
int UART5_ReadAvailable(void);
char UART5_read(void);
void UART5_write(char c);
void U5_getCommand(char *str, int maxLen);
uint8_t UART5_Available(void);
void U5_printstr(char *str);

#endif
```

Source Code 2.2.8: UART5.c

```c
#include "../../UTILITIES/Bit_Utilities.h"
#include "../../UTILITIES/tm4c123gh6pm.h"
#include "UART5.h"
#include "stdint.h"
#include "string.h"

void UART5Init(){
 SYSCTL_RCGCUART_R |=SYSCTL_RCGCUART_R5;
 SYSCTL_RCGCGPIO_R |=SYSCTL_RCGCGPIO_R4;
 UART5_CTL_R &= ~UART_CTL_UARTEN;
// SET BRD
//BR = 9600 bits/sec
//16*10^6/(16*9600) = 104.16667
 UART5_IBRD_R = 104;
 UART5_FBRD_R = 11;
 //UART5_CC_R = UART_CC_CS_SYSCLK;
 UART5_LCRH_R = (UART_LCRH_WLEN_8 | UART_LCRH_FEN);
 UART5_CTL_R |= (UART_CTL_UARTEN | UART_CTL_RXE | UART_CTL_TXE);
 GPIO_PORTE_AFSEL_R |= 0X30;
 GPIO_PORTE_PCTL_R = (GPIO_PORTE_PCTL_R & ~0xff) |
  ↪  (GPIO_PCTL_PE4_U5RX  | GPIO_PCTL_PE5_U5TX);
 GPIO_PORTE_DEN_R |=0x30;
}

int UART5_ReadAvailable(void){
  return ((UART5_FR_R&UART_FR_RXFE)==UART_FR_RXFE)? 0:1;
}

char UART5_read(){
  while(UART5_ReadAvailable() ≠1);
  return  UART5_DR_R & 0xFF;
}

void UART5_write(char c){
 while ((UART5_FR_R&UART_FR_TXFF)≠0);
 UART5_DR_R = c;

}

void U5_getCommand(char *str, int maxLen){
  char c;
```

```
  int i;
  for ( i=0; i<maxLen; i++){
    c=UART5_read();
    if(c== '\n' || c=='\r' )break;
    else str[i] = c;
    UART5_write(c);
  }
}
uint8_t UART5_Available(void){
  return ((UART5_FR_R &UART_FR_RXFE)==UART_FR_RXFE)? 0:1;
}

void U5_printstr(char *str){
  while(*str){
    UART5_write(*str);
    str++;
  }
}
```

2.2.5 EEPROM

Source Code 2.2.9: EEPROM.h

```
#ifndef EEPROM_H
#define EEPROM_H
#include "../../MCAL/GPIO/DIO_Driver.h"
#include "../../UTILITIES/Bit_Utilities.h"
#include "../../UTILITIES/tm4c123gh6pm.h"
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#define BLOCK_SIZE 16
#define NUM_BLOCKS 32

void delay_n(int n);
void eeprom_init(void);
void eeprom_write(uint32_t data ,uint8_t addr,uint8_t blk);
uint32_t eeprom_read(uint8_t addr,uint8_t blk);
void eeprom_clear(void);
#endif
```

Source Code 2.2.10: EEPROM.c

```c
#include "eeprom.h"
#include <stdint.h>
#include <stdio.h>
#include "../../MCAL/GPIO/DIO_Driver.h"
#include "../../UTILITIES/Bit_Utilities.h"
#include "../../UTILITIES/tm4c123gh6pm.h"

void delay_n(int n)
{
  int i;
  for(i=0;i<n;i++)
  ;
}

int eeprom_start(void)
{
  SYSCTL_RCGCEEPROM_R = 0x01;
  delay_n(15);
  while(EEPROM_EEDONE_R & EEPROM_EEDONE_WORKING);
    if((EEPROM_EESUPP_R & EEPROM_EESUPP_ERETRY) |
    ↪  (EEPROM_EESUPP_R & EEPROM_EESUPP_PRETRY))
    {
      return -1;
    }
  SYSCTL_SREEPROM_R = SYSCTL_SREEPROM_R0;
  SYSCTL_SREEPROM_R = 0;
  delay_n(15);
  while(EEPROM_EEDONE_R & EEPROM_EEDONE_WORKING);
    if((EEPROM_EESUPP_R & EEPROM_EESUPP_ERETRY) |
    ↪  (EEPROM_EESUPP_R & EEPROM_EESUPP_PRETRY))
    {
      return -1;
    }
    return 1;
}
void eeprom_error_recovery(void)
{
  EEPROM_EESUPP_R = EEPROM_EESUPP_START;
  while(EEPROM_EEDONE_R & EEPROM_EEDONE_WORKING);
}
void eeprom_init(void)
{

```

```
    int flag = 0;
    flag = eeprom_start();
    if (flag) {
    } else {
        eeprom_error_recovery();
        flag = eeprom_start();
        if (flag) {

        } else {

        }
    }

}
void eeprom_write(uint32_t data,uint8_t addr,uint8_t blk)
{
  EEPROM_EEBLOCK_R = blk;//Block number
  EEPROM_EEOFFSET_R =  addr; //offset within the block
  EEPROM_EERDWR_R = data; //data written
  while(EEPROM_EEDONE_R & EEPROM_EEDONE_WORKING);
  delay_n(5);
}
void eeprom_clear(void){
  char i;
  char j;
  for(i=1;i≤32;i++){
          for(j=1;j≤16;j++){
            eeprom_write(0,j,i);
  }
  }}
uint32_t eeprom_read(uint8_t addr,uint8_t blk)
{
  uint32_t data;
  EEPROM_EEBLOCK_R = blk;//Block number
  EEPROM_EEOFFSET_R =  addr;
  data = EEPROM_EERDWR_R;
  while(EEPROM_EEDONE_R & EEPROM_EEDONE_WORKING);
  delay_n(5);
  return data;
}
```

2.3 HAL

2.3.1 GPS

Source Code 2.3.1: GPS.h

```h
#ifndef __GPS__
#define __GPS__

#define PI 3.141592654
#define EARTH_RADIUS 6371000

char*  GPS_read(void);
void GPS_format(void);
char* getCurrentLongitude(void);
char* getCurrentLatitude(void);
//char* getCurrentSpeed(void);


#endif
```

Source Code 2.3.2: GPS.c

```c
#include "GPS.h"
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "../../MCAL/UART5/UART5.h"
#include "../../MCAL/UART0/UART0_PC.h"
#include "../../MCAL/STK/Systick.h"
#include "../../UTILITIES/Conversion.h"
#include <stdio.h>
#define MAX_BUFFER_SIZE 256

char GPS[80];
char latitude[11], longitude[12];
//, speed[5]
/**
 * Description :
 * Checking Correct Log
 * Reading it from UART5 and saving it in GPS array
 * $GNRMC,204520.00,A,5109.0262239,N,11401.8407338,W,0.004,102.
 ↪  3,130522,0.0,E,D*3B
```

```
 */

void readLineFromGPS(char* buffer, int maxLength) {
    int i = 0;
    char c;
    bool foundStart = false;
    bool foundEnd = false;
    while (i < maxLength - 1) {
        // Assuming UART read function is named uart_read_char()
        c = UART5_read();
        UART5_write(c);
        // Check for the start of a new sentence
        if (c == '$') {
            // Reset the buffer and index
            i = 0;
            buffer[i++] = c;
            foundStart = true;
            foundEnd = false;
        } else if (foundStart && c == '*') {
            // End of data, null-terminate the string
            buffer[i] = '\0';
            foundEnd = true;
        } else if (foundStart && !foundEnd) {
            // Add character to buffer
            buffer[i++] = c;
        }
        if (foundEnd && strncmp(buffer, "$GPRMC,", 7) == 0) {
            break; // Exit loop if GPRMC sentence
        }
        if (foundEnd) {
            i = 0; // Reset index if not a GPRMC sentence
            foundStart = false;
            foundEnd = false;
        }
    }
}
//, char* speed
void parseGPRMC(char* sentence, char* latitude, char*
↪  longitude) {
    int i ;
    char* token;
    token = strtok(sentence, ",");
    if (token == NULL || strcmp(token, "$GPRMC") != 0) {
        return; // Not a valid GPRMC sentence
    }
```

```
    // Skip tokens we are not interested in
    for ( i = 0; i < 3; i++) {
        token = strtok(NULL, ",");
        if (token == NULL) {
            return; // Sentence is too short
        }
    }
    // Copy the latitude
    strncpy(latitude, token, 10);
    if (strcmp(token, "N") != 0) {
        token = strtok(NULL, ",");
        if (token == NULL) {
            return; // Sentence is too short
        }
    }
    for ( i = 0; i < 1; i++) {
        token = strtok(NULL, ",");
        if (token == NULL) {
            return; // Sentence is too short
        }
    }
    // Copy the longitude
    strncpy(longitude, token, 11);
    // Copy the longitude direction (E/W)
    // Skip the token if it is 'E', otherwise skip the next
    ↪  token (which is the longitude direction)
    if (strcmp(token, "E") != 0) {
        token = strtok(NULL, ",");
        if (token == NULL) {
            return; // Sentence is too short
        }
    }
    // Skip more tokens
    for ( i = 0; i < 1; i++) {
        token = strtok(NULL, ",");
        if (token == NULL) {
            return; // Sentence is too short
        }
    }
    // Copy the speed
    //strncpy(speed, token, 5);
}

char* GPS_read(){
  char buffer[MAX_BUFFER_SIZE];
```

```
109   readLineFromGPS(buffer, MAX_BUFFER_SIZE);
110   SysTick_Wait(1);
111   return buffer;
112 }
113 
114 void GPS_format(void){
115    strncpy(GPS,GPS_read(),80);
116           //printstr(GPS);
117         //  parseGPRMC(GPS, latitude, longitude, speed);
118           parseGPRMC(GPS, latitude, longitude);
119 //          printstr("\n");
120 //          printstr(latitude);
121 //          printstr("\n");
122 //          SysTick_Wait(1);
123 //          printstr(longitude);
124 //          printstr("\n");
125 //          SysTick_Wait(1);
126 //          printstr(speed);
127 //          printstr("\n");
128 }
129 char* getCurrentLongitude(){
130 return longitude;
131 }
132 char* getCurrentLatitude(){
133 return latitude;
134 }
135 
136 //char* getCurrentSpeed(){
137 //return speed;
138 //}
```

2.3.2 LED

Source Code 2.3.3: LED_interface.h

```
1 #ifndef __HAL_LED_INTERFACE_H__
2 #define __HAL_LED_INTERFACE_H__
3 
4 
5 /*********************************< MACROS for colours
  ↪  *********************************/
6 #define LED_RED       1
```

```h
#define LED_BLUE    2
#define LED_GREEN    3
/*********************************< _SELECT_MC_PIN_
 ↪  ************************************/
#define LED_RED_PIN      1
#define LED_BLUE_PIN     2
#define LED_GREEN_PIN    3
/*********************************< SOME_Masks
 ↪  ************************************/
#define PF123_mask             0x0E
#define PF_mask                0x20
/*********************************< functions for LEDs
 ↪  ********************************/
/** \brief  The function LED_voidLedInit initializes the
 ↪  direction of three LEDs, namely a blue LED, a green LED,
 *      and a red LED by configuring the corresponding pins of
  ↪  the microcontroller as output pins.
 *
 *      The function first calls DIO_voidInit function which
  ↪  initializes the microcontroller's general-purpose
  ↪  input/output (GPIO) pins,
 *      specifically the ones associated with port F.
 *
 *      Then, the function configures the direction of each LED
  ↪  by calling DIO_u8SetPinDirection function three times,
  ↪  passing the port number (DIO_u8_PORTF) and the pin numbers
  ↪  (LED_BLUE_PIN, LED_GREEN_PIN, and LED_RED_PIN) as
 *      arguments with the third argument being DIO_u8_OUTPUT to
  ↪  set the pins as output.
 *      This function is likely part of a larger program that
  ↪  controls the behavior of the LEDs.
 *
 * \param  void
 * \return  void
 *
 */
void LED_LedInit(void);

void LED_LedOn(unsigned char copy_u8LedColour);

void LED_LedOff(unsigned char copy_u8LedColour);

void LED_OffAll(void);


```

```
40
41  #endif //__HAL_LED_INTERFACE_H__
```

Source Code 2.3.4: LED.c

```
1   /**< UTIL */
2   #include "../../UTILITIES/Bit_Utilities.h"
3   #include "../../UTILITIES/tm4c123gh6pm.h"
4   /**< LED_HAL */
5   #include "../../HAL/LED/LED_interface.h"
6
7   void LED_LedInit (void){
8       SYSCTL_RCGCGPIO_R |= PF_mask;
9       while ((SYSCTL_PRGPIO_R & PF_mask) == 0){};
10      GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY ;
11      GPIO_PORTF_CR_R |= PF123_mask;
12      GPIO_PORTF_AMSEL_R &= ~PF123_mask;
13      GPIO_PORTF_PCTL_R &= ~0x0000FFF0;
14      GPIO_PORTF_AFSEL_R &= ~PF123_mask;
15      GPIO_PORTF_DIR_R |= PF123_mask;
16      GPIO_PORTF_DEN_R |= PF123_mask;
17      GPIO_PORTF_DATA_R &= ~PF123_mask;
18  }
19
20  void LED_LedOn(unsigned char copy_LedColour) {
21      switch (copy_LedColour) {
22          case LED_BLUE:
23              GPIO_PORTF_DATA_R |= (1 << LED_BLUE_PIN);
24              break;
25          case LED_GREEN:
26              GPIO_PORTF_DATA_R |= (1 << LED_GREEN_PIN);
27              break;
28          case LED_RED:
29              GPIO_PORTF_DATA_R |= (1 << LED_RED_PIN);
30              break;
31          default:
32              // Error state
33              break;
34      }
35  }
36
37  void LED_LedOff(unsigned char copy_LedColour) {
38      switch (copy_LedColour) {
```

```
        case LED_BLUE:
            GPIO_PORTF_DATA_R &= ~(1 << LED_BLUE_PIN);
            break;
        case LED_GREEN:
            GPIO_PORTF_DATA_R &= ~(1 << LED_GREEN_PIN);
            break;
        case LED_RED:
            GPIO_PORTF_DATA_R &= ~(1 << LED_RED_PIN);
            break;
        default:
            // Error state
            break;
    }
}

void LED_OffAll(void) {
    GPIO_PORTF_DATA_R &= ~((1 << LED_RED_PIN) | (1 << LED_BLUE_PIN) | (1 << LED_GREEN_PIN));
}
```

2.3.3 Switch

Source Code 2.3.5: SW.h

```

#ifndef __SW__
#define __SW__
#include <stdbool.h>


#define  SW1      0x10
#define  SW2      0x01
#define  PORTF     'F'



void            SW_Init(unsigned char S);
unsigned char   SW_Read(unsigned char S);
bool            SW_ispressed(unsigned char S);


```

```

#endif

```

Source Code 2.3.6: SW.c

```
#include <stdbool.h>
#include <stdio.h>

#include "../../MCAL/GPIO/DIO_Driver.h"
#include "../../UTILITIES/Bit_Utilities.h"
#include "../../UTILITIES/tm4c123gh6pm.h"
#include "SW.h"



unsigned char button_in;
unsigned char button_prev;



void SW_Init(unsigned char S)
{
    switch (S)
    {
    case SW1:
        DIO_vPORTINIT(PORTF);
        DIO_vSETPINDIR(PORTF,4,0);
        break;
    case SW2:
        DIO_vPORTINIT(PORTF);
        DIO_vSETPINDIR(PORTF,0,0);
        break;

    default:
        break;
    }

}

```

```
unsigned char SW_Read(unsigned char S)
{
    switch (S)
    {
    case SW1:

        return GPIO_PORTF_DATA_R &SW1;
        break;
    case SW2:
        return GPIO_PORTF_DATA_R &SW2;
        break;

    default:
        break;
    }
}



bool SW_ispressed(unsigned char S)
{
    switch (S)
    {
    case SW1:
        button_in = SW_Read(SW1);
        if (button_prev ≠ 0x10 && button_in == 0x10)   //rising
        ↪  edge
            return true;
        else
            return false;
        button_prev = button_in;
        break;


    case SW2:
        button_in = SW_Read(SW2);
        if (button_prev ≠ 0x01 && button_in == 0x01)   //rising
        ↪  edge
            return true;
        else
            return false;
        button_prev = button_in;
        break;

    default:
```

```
80          break;
81      }
82
83
84
85
86  }
```

2.3.4 LCD

Source Code 2.3.7: LCD.h

```
1   #include "tm4c123gh6pm.h"
2
3
4   #define LCD_RS (*((volatile unsigned long *)0x40004200))
    ↪  //PA.7 for register select pin
5   #define LCD_EN  (*((volatile unsigned long *)0x40004100))
    ↪  //PA.6 for enable pin
6   #define LCD_RW  (*((volatile unsigned long *)0x40004080))
    ↪  //PA.5 for rw pin
7   #define clear_display     0x01
8   #define returnHome        0x02
9   #define moveCursorRight   0x06
10  #define moveCursorLeft    0x08
11  #define Shift_Cursor_Left 0x10
12  #define shiftDisplayRight 0x1C
13  #define shiftDisplayLeft  0x18
14  #define cursorBlink       0x0F
15  #define cursorOff         0x0C
16  #define cursorOn          0x0E
17  #define Function_set_4bit 0x28
18  #define Function_set_8bit 0x38
19  #define Entry_mode        0x06
20  #define Function_8_bit    0x32
21  #define Set5x7FontSize    0x20
22  #define FirstRow          0x80
23  #define SecondRow         0xC0
24  void LCD_INIT(void);
25  void LCD_CMD(unsigned long cmd);
26  void LCD_WRITE (char data);
27  void SysTick_Wait_Timer(int delay);
```

```
28 void LCD_String(char *str);
29 void delay_milli(int i);
30 void delay_micro(int i);
```

Source Code 2.3.8: LCD.c

```
1  #include "tm4c123gh6pm.h"
2  #include "TM4C123.h"
3  #include "LCD.h"
4  #include "../../UTILITIES/Bit_Utilities.h"
5  typedef char * string;
6  /*The LCD initializes
7  the ports A5-6-7 as the control pins  for LCD and portB0-7
8  as the data pins*/
9  void LCD_INIT(void) {
10     // PA5 >> r/w, PA6 >> EN, PA7 >> RS
11     SET_BIT(SYSCTL_RCGCGPIO_R,0);   // allow the clock for
       ↪  PA5,6,7
12     while((SYSCTL_PRGPIO_R& (0x01)) == 0){}
13     GPIO_PORTA_AFSEL_R &= ~0xE0;    //disable alternative
       ↪  functions for PA5,6,7
14     GPIO_PORTA_AMSEL_R &= ~0XE0;    //disable analogue function
       ↪  for PA5,6,7
15     GPIO_PORTA_PCTL_R &= ~0XE0;     //regular digital pins
16     GPIO_PORTA_DIR_R |= 0XE0;       //set the direction of
       ↪  PA5,6,7 as output
17     GPIO_PORTA_DEN_R |= 0XE0;       //enable digital PA5,6,7
18     SET_BIT(SYSCTL_RCGCGPIO_R,1);
19     while((SYSCTL_PRGPIO_R& (0x01 << 1)) == 0){}
20
21     GPIO_PORTB_AFSEL_R &= ~0xff;    //disable alternative
       ↪  functions for portB
22     GPIO_PORTB_AMSEL_R &= ~0Xff;    //disable analogue function
23     GPIO_PORTB_PCTL_R &= ~0XFF;     //regular digital pins
24     GPIO_PORTB_DIR_R  |= 0XFF;      //set the direction of PB0-7
       ↪  as output
25     GPIO_PORTB_DEN_R  |= 0XFF;      //enable digital portB
26     delay_milli(10);
27
28
29
30 }
31
```

```
void LCD_CMD(unsigned long cmd) {
    GPIO_PORTB_DATA_R = cmd;    //set PB7-0 as the passed command to the function
    LCD_RS = 0x00;              //set PA7 register select pin to low
    LCD_RW = 0x00;              //set PA5 r/w pin to low
    LCD_EN = 0x40;              //set enable pin to high
    delay_micro(40);                    //short delay
    LCD_EN = 0x00;              //set enable pin to low
      delay_milli(10);

}

void LCD_WRITE (char data ) {
    GPIO_PORTB_DATA_R = data;  //write the data to PB7-0
    LCD_RS = 0x80;             //set PA7 to high
    LCD_RW = 0x00;             //set pA5 to low
    LCD_EN = 0x40;             //set the enable pin high
    delay_micro(40);                   //short delay
    LCD_EN = 0x00;             //set the enable pin to low
    delay_milli(10);

}

void SysTick_Wait_Timer(int delay)
{
  NVIC_ST_CTRL_R = 0; //disable Timer
  NVIC_ST_RELOAD_R = delay- 1;          //to make 0.1 secound as our Tiva has clk = 16 MHZ
  NVIC_ST_CURRENT_R = 0;                                //to clear counter value and underflow flag of counter
  NVIC_ST_CTRL_R |= 0x5;
  while((NVIC_ST_CTRL_R&0x00010000)==0) {
        ;                           /* wait for COUNT flag (bit 16) */
    }
}                         //to put at source clk 1 to get PROCESOR CLK NOT its 8th only and enable Timer
void delay_milli(int i)
{
 SysTick_Wait_Timer(i*16000);
}
void delay_micro(int i)
{
 SysTick_Wait_Timer(i*16);
```

```
}

void LCD_String(char* str)
{
    int i;
    for(i=0;str[i]≠0;i++)
    {
        LCD_WRITE(str[i]);
    }
}
```

2.4 Trajectory Drawer

Source Code 2.4.1: drawer.py

```
import pandas as pd
import matplotlib.pyplot as plt
import serial
PI =  3.141592654
def ToDeg(Num):
  degree = Num/100
  minutes =Num-float(degree*100)
  return(degree+(minutes/60))
def ToRad(DegNum):
    return (DegNum*(PI/180))

def draw(data_raw):
    if len(data_raw) % 2 ≠ 0:
        print("Invalid number of values. Each longitude should have a corresponding latitude.")
        return

    lon_values = []
    lat_values = []
    for i in range(0, len(data_raw), 2):
        lon_values.append(ToRad(round(ToDeg((float(data_raw[i])/10000)),5)))
        lat_values.append(ToRad(round(ToDeg((float(data_raw[i + 1])/10000)),5)))

    plt.plot(lon_values, lat_values, marker='o')
    plt.xlabel('Longitude')
```

```
    plt.ylabel('Latitude')
    plt.title('GPS Data')
    plt.show()

ser = serial.Serial(port='COM3',parity="N",baudrate=9600,stopbits=1,timeout=120)  # open first serial port
print(ser.portstr+" is runing !!")
ser.flushInput()
ser.flushOutput()
data_raw=[]

while True:
    if ser.writable():
        #user_input = input("Enter a U to send : ")
        #ser.write(user_input.encode())  # Send 'U' if writable
        ser.write(b'U')
    while ser.in_waiting > 0:
        line = ser.readline().decode().strip()  # Read and decode the data
        print(line)
        if line == "saved" :
            draw(data_raw)
            print("drawed")
        elif not line:
            print("=========BREAKED")
            break
        else:
            data_raw.append(line)  # Append received line to data_raw list



ser.close()


```

3 Tiva-C Launchpad Test

3.1 Full Videos

The full videos for the tests can be found on this link (also on our Github repository):
https://github.com/minanasser54/ARM_M4_GPS/tree/MinaNasser/Second%20Milestone%20_%20POC%20TEST

3.2 Test Case 1

Figure 3.1: Test Case 1 Trajectory

First We choose between accumulating and retrieving mode using SW1 or SW2 respectively. (Blue LED represents no mode)

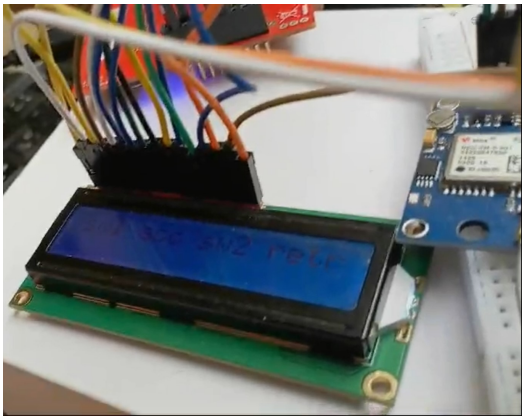


Figure 3.2: LCD showing (SW1:acc SW2:ret) and LED is blue

We will choose accumulating mode and move to desired destination with LED color being red. The EEPROM records current position periodically as we move:

Figure 3.3: Moving to destination with distance from starting point showing on LCD

Once we reach the destination, we will press SW2 to set destination reached and the LED turns green

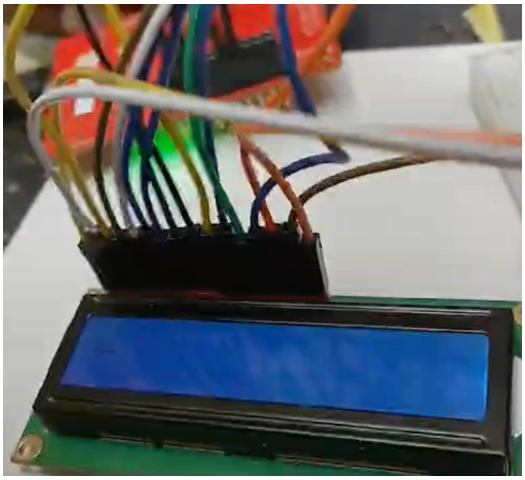


Figure 3.4: Destination reached

After that, No Mode is switched to automatically. We will remove and connect the launchpad from the host laptop in order to verify operation of EEPROM. Then, we will run the Python trajectory drawer to plot our trajectory. During runtime of the Python trajectory drawer, we will switch to retrieving mode using SW2 in order to send data from the launchpad to the host laptop (The LED turns yellow, representing retrieving mode)

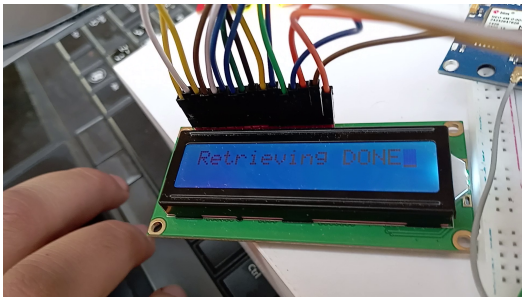


Figure 3.5: Retrieving mode: sending data from launchpad to Laptop

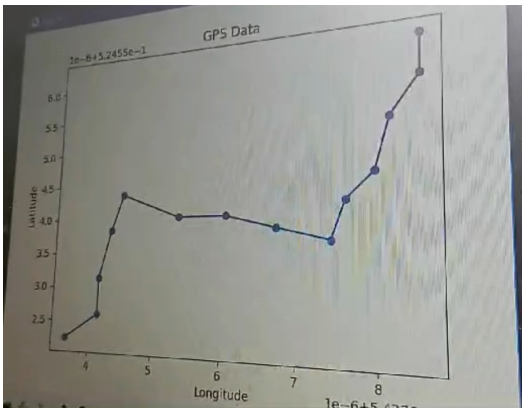


Figure 3.6: Trajectory plot of test case 1

3.3 Test Case 2

Figure 3.7: Test Case 2 Trajectory

The steps here are similar to Test Case 1 except that the LED will automatically turn green as we pass the 100m mark, switch to No Mode with LED turning to blue and similarly to Test Case 1 we will disconnect and connect the launchpad from the host laptop to verify EEPROM correct operation, run the Python trajectory drawer, switch to retrieving mode on the launchpad and observe the resulting trajectory plot.

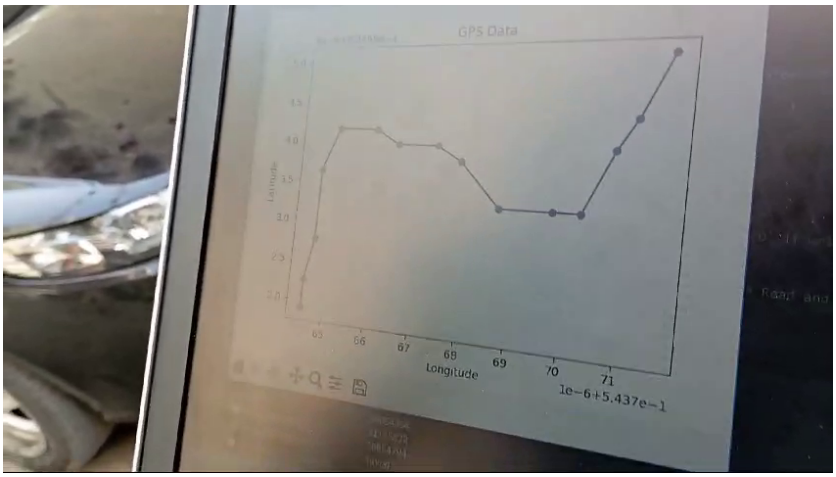


Figure 3.8: Trajectory plot for Test Case 2