Final Project Report

Tracking, Wirelessly Transmitting and Displaying User Motion

Group 1

Hoai Phuoc Truong – 260526454
Michael Kourlas - 260525163
Jacob Barnett – 260451446
Ravi Chaganti – 260469339

ECSE 426
Department of Electrical and Computer Engineering
McGill University

5th December, 2015

Table of Contents

1 Abstract

The aim of this group project is to design an embedded system which will track the trajectory of a user using CC2500 RF transceiver module, accelerometer and gyroscope sensors in LSM9DS1 chipset, STM32F4 and STM32F429i Discovery boards. The starting location is presumed to be the entrance to the microprocessor laboratory on the 4th floor of the Trottier building. The project requirements state that the board, which may be attached to the user by strapping it around the foot, should calculate and process accelerometer and gyroscope data until the user returns to the starting point and wirelessly transfers the data to another microprocessor board, which will plot the trajectory on an LCD display.

2 Problem Statement

One of the main objectives of the final group project is to configure the CC2500 RF transceiver module to enable wireless transmitter/receiver communication system between two STM boards. Driver functions must be properly implemented using SPI protocol to initialize the transceiver module and perform various operations including reading, writing, and flushing transmitter and receiver bytes. To configure wireless transmission at required frequency between the two STM boards, certain methods must be defined to transfer and receive packets of data without any loss of information.

The STM32F4 board must be configured with LSM9DS1 chipset to use accelerometer and gyroscope sensors to collect and process user-motion data in x and y coordinates. Driver functions must be properly implemented using SPI protocol to initialize the chipset and perform various operations including calibrating, filtering, and updating x and y coordinates. A button must be pressed to initiate the tracking process by storing coordinates into a buffer and to terminate the tracking process and transfer data to the STM32F429i board.

The STM32F429i board, which has an LCD display, must be used to display the trajectory of the user's motion on the 4th floor of the Trottier building. The board should be able to receive the coordinates from the STM32F4 board and plot the user's trajectory on the LCD display. Appropriate conversions of feet to pixels must take place of all the coordinates in the buffer to plot an accurate trajectory path.

All three sub-systems including wireless transmission, trajectory measurements, and LCD display must be successfully integrated to configure a fully functional user-motion trajectory system. Also, all the implementations, testing methods and observations must be properly documented.

3 Theory and Hypothesis

3.1 CC2500 RF Transceiver

CC2500 module is mainly used for low-power wireless designs at a frequency band of 2.4-2.483GHz. It has an integrated RF transceiver with a flexible baseband modem which offers data rate up to 500kBaud. It can be operated using generic SPI protocol to interface with a microprocessor system and also provides various key features including data packet handling/buffering and burst transmissions (Reference 1 in Appendix).

It offers a 64-byte TX/RX FIFO (first in first out) where TX-FIFO is write-only and RX-FIFO is read-only. It also enables user to send single byte instructions using command strobes to conduct operations including configuring receive mode and flushing TX/RX FIFO (which can only be issued in IDLE, TXFIFO_UNDERFLOW, RXFIFO_OVERFLOW states) by using SFTX (to acknowledge FIFO underflow) and SFRX (to acknowledge FIFO overflow).

To safely receive a packet from the sender, the receiver must have a frequency greater than that of transmitter. When the sender is in receive mode, the sender frequency is higher than the receiver frequency. For a good transmission, receiver frequency should be around twice the transmitter frequency.

3.2 Frequency Calculation

To determine the operating frequency of our channel for our group, the following equation is used:

$$channel_{frequency}(MHz) = \ base_{frequency}(MHz) + groupID \times 8\ (KHz)$$

Equation1: Operating Frequency

The channel frequency for our group is 2433.008MHz (base frequency is 2433 MHz). To store these frequencies in CC2500 registers, the following equation is used (Reference 1 in Appendix):

$$frequency_{carrier} = \frac{f_{XOSC}}{2^{16}} \times FREQ[23:0]$$

Equation2: Frequency Conversion

Using f_{XOSC} as 26MHz and $frequency_{carrier}$ as 2433.008MHz, we obtain $FREQ[23:0]$ as 6132657, which is 5d93c5 in hexadecimal (stored in register address as 5D, 93 and C5 in FREQ2, FREQ1 and FREQ0).

3.3 LCD Screen on STM32F429i Board

Liquid crystal display (LCD) has a 2D grid of pixels where each pixel is made of three segments and each segment is represented by red, green and blue. The LCD screen on STFM32F429i board is 320×240 pixels. The LTDC controller, which in the processor chipset, is interfaced to an ILI9341 controller, which is in the LCD module (Reference 2 in Appendix). Some of the key methods of LCD screen are given in stm32f429i_discovery_lcd.c which will be used in our implementation.

3.4 External Interrupts/GPIO Mapping

Since there are 16 types of GPIO pins each for GPIOA, GPIOB, GPIOC, GPIOD and GPIOE, the external interrupt lines are defined (Reference 3 in Appendix) according to the 16 types. For example, if we choose GPIOA_pin_1 on the processor to enable the external interrupt coming from the accelerometer sensor, the external interrupt line that will be used is EXTI1.

In our implementation of configuring the accelerometer and gyroscope sensors in LSM9DS1 chipset, we will be using GPIOE_pin_0 and GPIOE_pin_1 which are defined for the external interrupt line EXTI0 and EXTI1.

3.5 Least Square Approximation

To calibrate the readings that are measured from the accelerometer and gyroscope sensors, the least square approximation method will be used:

$$X = [w^T . w]^{-1} . w^T . Y$$

Equation 3: least square approximation to calculate X

$$Y = w.X$$

Equation 4: generic least square approximation

w is the N×4 matrix where N is the number of results that are being considered for calibration (Reference 4 in Appendix). X is the matrix which contains the normalization values or calibration parameters and Y is the output defined for a particular axis. Once the X matrix is obtained, constants will be added for each axis to each normalized values in X matrix for each axis.

3.6 Moving Average Filter

In our implementation of the moving average filter to smoothen the values of the output signal, we will be using a circular buffer as it is connected end-to-end and is easy to use to buffer continuous data streams. In this model, we will be taking the average of the number of elements based on the following equation (Reference 5 in Appendix):

$$y[i] = \frac{1}{M}\sum_{j=0}^{M-1} x[i+j]$$

Equation 5: moving average filter equation

where M is the filter depth (D), x is the circular buffer and y is the continuous moving average filter.

3.7 Serial Peripheral Interface (SPI) protocol

SPI protocol is used to read and write the registers from the external device with the help of 4 wires:

1. Chip Select (CS): low at the start of the transmission signal and high at the end
2. Serial Port Clock (SPC): controlled by SPI master, no transmission when CS is high
3. Serial Port Data Input (SPDI)
4. Serial Port Data Output (SPDO)

Data is written to the device when bit_0 is 0 (chip will enable SPDI) and data is read from the device when bit_0 is 1 (chip will enable SPDO). It will take 16 clock cycles for the read and write register commands to be completely executed. Connections for LSM9DS1 and CC2500 using SPI protocol to STM boards are described in detail in the implementation section (Reference 6 and Figure 14 in Appendix).

3.8 Timer's period equation

To calculate TIM's parameters of prescaler and period, the following equation will be used (Reference 7 in Appendix):

$$PWM\_frequency = \frac{\frac{default\_frequency}{prescaler + 1}}{TIM\_Period + 1}$$

Equation 6: Timer's Period Equation

3.9 Embedded Operating Systems

The purpose of any operating system is to manage access to hardware and to provide a set of services to applications running on that hardware. This is also true of embedded operating systems, including the real-time operating system used in this lab, RTX.

Embedded operating systems generally provide the following services (Reference 8 in Appendix):

1. Share CPU processing time between multiple tasks or threads by dividing the available time into slices and assigning slices to each task.
2. Provide common OS primitives, such as semaphores, mutexes, and signals, to transfer information and synchronize operations between threads and manage the use of shared resources.
3. Prevent applications from accessing memory that is not assigned to them.
4. Provide the above services with a very small memory footprint and a fast context switching time. Real-time operating systems in particular must ensure that events happen within a certain actual timeframe.

As a result, embedded operating systems also tend to simplify system designs and reduce application-specific code, since applications do not have to worry about providing these services.

The key disadvantage of embedded operating systems is that there are always memory and performance penalties associated with using them. The operating system itself requires some memory, and there is overhead involved in scheduling and context switching. Many applications do not need to use an embedded operating system, as the tasks they must perform are short and do not overlap; in such cases, it is more efficient to only use interrupts. However, this application does use many overlapping tasks, making the use of an embedded operating system worth the cost.

Another disadvantage is portability; each operating system uses a different interface, making it difficult to switch to another if it becomes necessary. This is partially mitigated through standard OS interfaces such as CMSIS-RTOS, which is used in this lab.

3.10 Operating System Threads

A thread is a single task that shares CPU time with other tasks. In CMSIS-RTOS, a thread consists of a C function and some associated data.

The operating system scheduler is responsible for executing each thread for a certain amount of time, then switching to another thread. The scheduler used in this lab is the RTX scheduler, a round-robin preemptive scheduler with priorities (Reference 9 in Appendix). Under the round-robin scheduling mechanism, time slices are assigned to each thread in circular order, and each slice is of equal size. However, higher priority threads are given more time slices than lower priority threads. Pre-emptive schedulers halt execution of a thread when performing switching; they do not wait for the thread to release control or terminate.

When switching between threads, the scheduler halts execution, saves the current thread context (which consists of the thread stack and some other attributes), loads the context of the new thread. Thread switching occurs so fast that it appears that the threads are being executed in parallel.

CMSIS-RTOS thread states and state transitions are shown in
Figure 1. A thread can be either active or inactive, and a thread that is active can be in one of three states: waiting, ready, or running. The thread that is currently being executed is in the running state; only one thread can be in this state at a time. Each thread in the ready state is ready to run and is waiting for the scheduler to preempt the running thread and move it into the running state. A thread in the waiting state is waiting for some event to occur, such as a signal or a mutex becoming free. It will not be moved into the running state by the scheduler until the event occurs and it moves into the ready state.

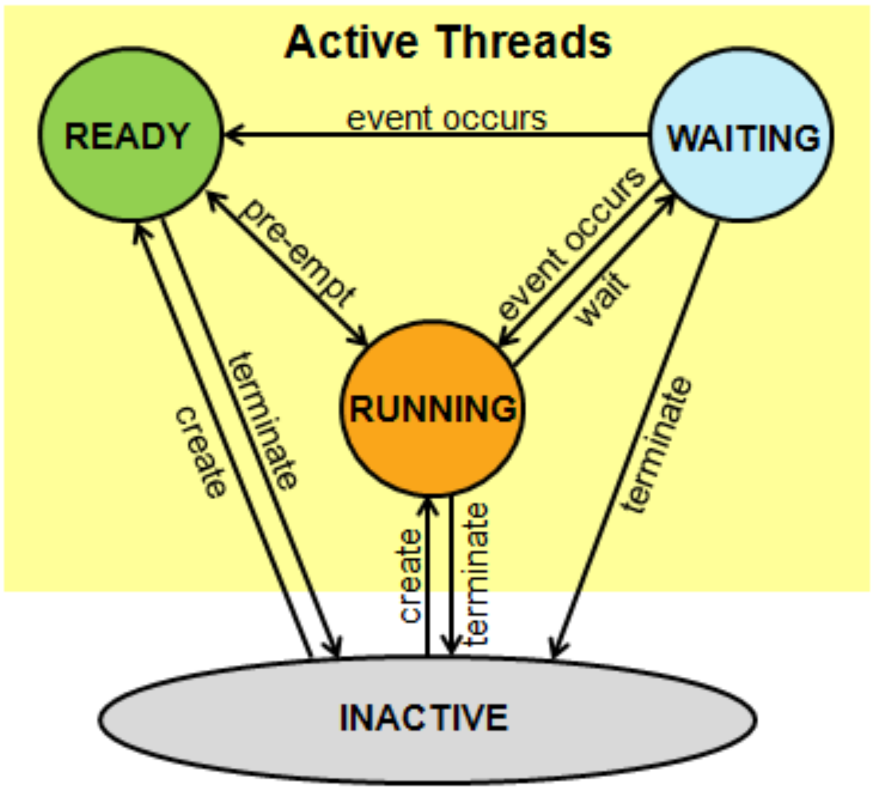


Figure 1: Thread states and state transitions

4 Implementation

4.1 LCD Configuration and Methods

We decided to display the user's trajectory on the LCD screen with an image of the 4th floor of Trottier as the background (as seen in Figure 2).

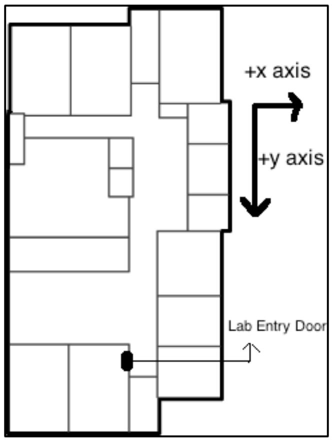


Figure 2: 4th Floor Map of the Trottier Building

The top-left corner is the starting point with 0 X 0 pixels and the size of the background image is 320 X 240 pixels. The entry door point of lab is defined by 163 X 266 pixels. We implemented msteps (Michael-steps, or one stride length of our teammate, Michael), where each step is approximately 2.5 feet and each foot is approximately 1.853 pixels on the LCD, to plot the user's trajectory which will be received from the measuring board in terms of x and y coordinates.

Some of the key methods that were implemented to configure the LCD screen are defined in Table 1:

Method	**Key Parameters**	**Explanation**
mstep_to_pixloc	mstep_coord	• Method will convert array of coordinates in units of msteps to pixels to display on LCD map
draw_points	xs, ys	• Method will draw xs and ys coordinates using LCD_DrawUni_Line function
draw_from_db		• Method will plot the background image of Trottier 4th floor and draw all the coordinates that are stored in a database
receive_and_plot		• Method can receive 2 types of commands: clear and plot/terminate • Clear will erase all the values in the database • Plot/terminate will plot all the values in the coordinates onto LCD and terminate other processes

		• Method will receive data packets from protocol_go_back_1_receive method
coordinate_db_insert_entry		• Method will store coordinates (x,y) into database in form [x1, y1, x2, y2,…]
coordinate_db_get_len coordinate_db_get_entry		• Methods will get the number of coordinates stored in database and retrieve single pair (x,y) from database

Table 1: LCD Configuration Methods

4.2 Trajectory Configuration and Methods

The first task was to write driver functions for LSM9DS1 chipset which is enabled with both accelerometer and gyroscope sensors. Table 2 summarizes low and high level initialization of chipset using SPI protocols.

Methods	Key Parameters	Explanation
lsm9ds1_low_level_init	• SPI -> SCK, MOSI, MISO, CS pin configurations • RCC_APB1PeriphClockCmd • RCC_AHB1PeriphClockCmd • SPI_I2S_DeInit • INT1, INT2 (interrupts)	• Method used to perform low-level initialization of accelerometer and gyroscope sensors • Configured and enabled SPI and connected SCK, MOSI, MISO, CS to pins 13 (GPIOB), 15 (GPIOB), 14 (GPIOB), 12 (GPIOB) on STMF32 board • APB1 used to enable SPI clock; AHB1 used to enable master slave SCK (clock), MOSI (input), MISO (output), CS (chip select), INT1 and INT2 (interrupts) GPIO clocks • INT1 and INT2 connected to GPIOE pins 0 and 1 to detect interrupts from

		accelerometer and gyroscope sensors
• lsm9ds1_acc_interrupt_low_level_init • lsm9ds1_gyro_interrupt_low_level_init	• RCC_AHB1PeriphClockCmd • RCC_APB2PeriphClockCmd • exti_init • nvic_init	• Methods used to perform low-level initialization of interrupts for accelerometer and gyroscope sensors • AHB1 used to enable SPI clock for INT1 and INT2; APB2 used to enable RCC_APB2Periph_SYSCFG • exti_init used to configure external interrupt INT1 at GPIOE_0 pin and INT2 at GPIOE_1 pin • nvic_init used to add IRQ (interrupt request) to the NVIC as global external interrupt to MCU; NVIC_IRQChannel set as EXTI_INT1 and EXTI_INT2 with preemptionPriority as 0 (highest priority)
• lsm9ds1_acc_init • lsm9ds1_gyro_init	• uint8_t buf • data_rate • full_scale • anti_aliasing • axes_enable • data_ready_interrupt_enabled	• Methods used to perform high-level initialization of accelerometer and gyroscope sensors • For accelerometer, will write data rate, full scale and anti-aliasing configurations to LSM9DS1_CTRL_REG6_XL and axes enable to LSM9DS1_CTRL_REG5_XL and will enable INT1 interrupt once data is ready • For accelerometer, will write data rate, full scale and anti-

		aliasing configurations to LSM9DS1_CTRL_REG1 and axes enable to LSM9DS1_CTRL_REG4 and will enable INT2 interrupt once data is ready

Table 2: LSM9DS1 initialization (accelerometer and gyroscope sensors)

Methods to read, write and fetch data from accelerometer and gyroscope sensors from memory address using SPI protocols are summarized in Table 3:

Methods	**Explanation**
lsm9ds1_send_recv_byte	Method sends and receives a single byte using SPI protocol
lsm9ds1_write	Method writes the specified bytes to the specified memory address within the accelerometer sensor via SPI protocol
lsm9ds1_read	Method reads the specified number of bytes to the specified memory address within the accelerometer sensor via SPI protocol
lsm9ds1_get_acceleration	Method used to get the current acceleration for x, y, and x axes from the accelerometer sensor via SPI protocol (determines sensitivity for value and then normalizes each value to return acceleration for x, y, and x axes)
lsm9ds1_get_angular_velocity	Method used to get the current angular velocity for x, y, and z axes from the gyroscope via SPI protocol (determines sensitivity for value and then normalizes each value to return angular velocity for x, y, and x axes)

Table 3: LSM9DS1 configuration methods

Methods to initialize accelerometer and gyroscope sensors and to calibrate and filter their measurements can be seen in Table 4:

Method	Key Parameters	Explanation
acc_init gyro_init	• axes_enabled	• Methods will initialize accelerometer and gyroscope sensors by enabling x, y and z axes and configuring ACC_TIM and GYRO_TIM
raw_to_calibrated		• Method will get calibrated accelerometer and gyroscope measurement using calibration matrices for each axis
raw_to_calibrated_filtered		• Method will get a calibrated and filtered accelerometer and gyroscope measurement using moving average filter for each axis_index
acc_update gyro_update	• acc_interrupt • gyro_interrupt	• acc_update method will be called by ACC_TIM to get and store the latest data from the accelerometer sensor and then reset the interrupt • gyro_update method will be called by GYRO_TIM to get and store the latest data from the gyroscope sensor and then reset the interrupt
TIM2_IRQHandler TIM1_IRQHandler	• acc_interrupt • gyro_interrupt	• TIM2_IRQHandler method will manage the interrupt handler for accelerometer sensor (ACC_TIM) • TIM1_IRQHandler method will manage the interrupt handler for gyroscope sensor (GYRO_TIM)

acc_get_x acc_get_y acc_get_z gyro_get_x gyro_get_y gyro_get_z		• acc_get_x, y, and z methods will get the filtered current acceleration in x, y, and z directions • gyro_get_x, y, and z methods will get the filtered angular velocity in x, y, and z directions

Table 4: Accelerometer and Gyroscope Sensor Methods

Map.c, step.c, and turn.c were implemented to process coordinates in terms of the number of steps and turns taken. In map.c, we defined the maximum number of coordinates, i.e. the maximum number of possible steps as 250. A step (left or right) or turn (left or right) is processed by adding a coordinate to the map with four possible orientations: positive and negative x and y directions. After each pair (x,y) of orientation is processed, its coordinates are copied to a buffer which will be transmitted to STM32F429i board. Files step.c and turn.c will check if a step or turn occurred given the latest accelerometer/gyroscope data and if their measurements are exceeding threshold values (explained in detail in Testing and Observation section).

4.3 Wireless Communication Configuration and Methods

The first task was to write driver functions for CC2500 RF transceiver. Table 5, Table 6 and Table 7 summarize chipset initialization using SPI protocols, methods and register configuration.

Method	**Key Parameters**	**Explanation**
CC2500_LowLevel_Init	• SPI -> SCK, MOSI, MISO, CS pin configurations • RCC_APB2PeriphClockCmd • RCC_AHB1PeriphClockCmd • SPI_I2S_DeInit	• Method to perform low-level initialization of interface that is used to drive CC2500 • Configure and enable SPI and connect SCK, MOSI, MISO, CS to pins 5 (GPIOA), 7 (GPIOA), 6 (GPIOA), 9 (GPIOB) on STMF32 board • APB2 used to enable SPI clock, AHB1 used to enable

		master slave SCK (clock), MOSI (input), MISO (output), and CS (chip select) GPIO clocks

Table 5: CC2500 Initialization (connected to both measuring board and LCD board)

Methods	Explanation
CC2500_Write CC2500_Read	• Write method writes 1 byte and Read method reads a block of data from CC2500 • pBuffer points to the buffer containing the data to be written, WriteAddr contains the internal address where byte is written to, NumByteToWrite and NumByteToRead are the number of bytes to be written and read from CC2500
CC2500_read_one CC2500_write_one	Read_one and Write_one methods will read and write one byte from/to CC2500
CC2500_get_state CC2500_get_part_num	get_state and get_part_num methods get the FSM state and part num of CC2500
CC2500_get_rxbytes CC2500_get_txbytes CC2500_flush_rx CC2500_flush_tx	• get_rxbytes and get_txbytes methods get the number of bytes in FX FIFO (first in first out) and TX FIFO of CC2500 • flush_rx and flush_tx methods will flush rx and tx FIFO from CC2500
CC2500_read_rx_one CC2500_write_tx_one CC2500_read_rx CC2500_write_tx	• read_rx_one and write_tx_one methods will read and write one byte from/to RX/TX FIFO of CC2500 • read_rx and read_tx methods will read and write multiple bytes from/to RX/TX FIFO of CC2500

CC2500_Reset	Reset method will reset the CC2500 chip by flushing tx and rx FIFO
CC2500_SendByte	SendByte method will send a byte through the SPI interface, wait to receive a byte, and return the byte received from the SPI bus

Table 6: CC2500 configuration methods (connected to both measuring board and LCD board)

Important Registers	Explanation
FIFOTHR	• Used to set threshold points in FIFO such that receiver always operates at higher frequency so that there won't be overflow of RXFIFO • Set as 7 (33 bytes in TX FIFO and 32 bytes in RX FIFO)
PKTLEN	• Set as 1 since we implemented our own checksum to accommodate longer packets
PKTCTRL1	• Set as 8 to disable automatic flush of RX FIFO since CRC (cyclic redundancy check) is not being used as we implemented our own checksum
PKTCTRL0	• Set as 0 to enable normal mode to use FIFOs for RX and TX
MCSM1	• Set as 0 which will switch state to IDLE after a packet has been received (in RX mode) and when a packet has been sent (in TX mode)

Table 7: Register Configuration for CC2500

We implemented a two-layer design for wireless communication system:

1. Layer 2 (data link layer: MAC): wireless_transmission_sm → FSM (finite state machine) running inside Layer 3
2. Layer 3 (network layer): protocol_go_back_1 → FSM (finite state machine)

Layer 2: Wireless Communication Basic Implementation

To check the length of the bytes in the packet that is being transmitted/received, a struct, my_check_sum, of parameters xor (all data bytes in packet), xor2 (data bytes in packet whose indices are divisible by 2), and xor3 (data bytes in packet whose indices are divisible by 3) was defined. Initially, the operation state of wireless transmission is set to idle (state 0) and the transmit, receive, and error states are defined as 2, 1, and 3. Method calculate_check_sum is used will calculate the sum of data bytes in the packet for xor, xor2 and xor3. Also, since no byte in the packet can be located at the START_PACKET and END_PACKET (indexes), check_sum (pointer to the location where checksum starts) is replaced with END_PACKET's index (subtracted with values 1, 2, 3, or 4) to avoid collision if a byte is located at start and end indexes.

To verify if the full packet that is being transmitted/received is in the right format, certain conditions must be met which are defined in wiresless_transmission_protocol_sanity_check method. The START_PACKET and END_PACKET parameters will check if it is indexed exactly at the beginning and end of the packet respectively. The packet is stored in full_packet buffer with length full_len. This method will verify if the packet length and sum of its bytes is correct and will return true. To encapsulate and check sum of the packets, wireless_transmission_protocol_encapsulate (which will encapsulate a packet and return length of the packet with header point and check sum) and wireless_transmission_protocol_checksum (will compare check sum with the expected sum) protocol methods were implemented. Also, methods wireless_transmission_encode and wireless_transmission_decode were defined to encode and decode packets with parameters dst (destination buffer where results are written), src (source buffer where input values are read), and len (length of the source buffer).

To receive and transmit packets, the following wireless transceiver methods have been implemented to initialize the basic API design (see Table 8).

Method	Key Parameters	Explanation
wireless_transmission_get_ received_packet	received_packet	• Method will retrieve received packet information (also check if packet is successfully retrieved or not) • Method will verify for packet length and sum and then decode the received packet into the received_packet struct
wireless_transmission_tran smit	packet len operation_mode	• Method will encapsulate the encoded data wirelessly transmit it • packet points to the data buffer and len is the length (in bytes) of the data buffer • Method will return true if transmission is a success (false otherwise) and set operation_mode to transmit mode (2)

wireless_transmission_retransmit		• Method will retransmit data if requested
wireless_transmission_receive_packet	operation_state	• Method will set the operation_state to receive mode (1)

Table 8: Transceiver Methods (for CC2500)

Layer 2: Main Operations of Transceiver

All the main operations have been defined in wireless_transmission_period method. First step is to verify if the operation_state is in transmit or receive mode. If in transmit mode, it will retrieve the state of CC2500 and flush TXFIFO if it's underflow state and set operation_state as error. If state is not in transmission mode for CC2500, it will send STX to request CC2500 to change its mode to transmission. Once CC2500 is in transmission mode, packets will be transferred byte to byte into TXFIFO and then reset the operation_state to idle mode.

If in receive mode, it will receive the state of CC2500 and flush RXFIFO if it's in overflow state and set operation_state as error. If state is not in receiver mode for CC2500, it will continuously send RTX until CC2500 is in receiver mode. Once CC2500 is in RX mode, it will read the number of bytes in RXFIFO byte by byte and process data by verifying state and end part of packets. If it is expecting END_PACKET, following operations will take place: if receiving END_PACKET, it will go to idle mode; if receiving START_PACKET, it will stop expecting END_PACKET; else it will write one byte into receiver buffer. Else if is not expecting END_packet, following operations will take place: if receiving START_PACKET, it will start expecting END_PACKET; else it will drop one received byte.

Figure 3 shows state diagrams for both receiver and sender ends for Layer 2:

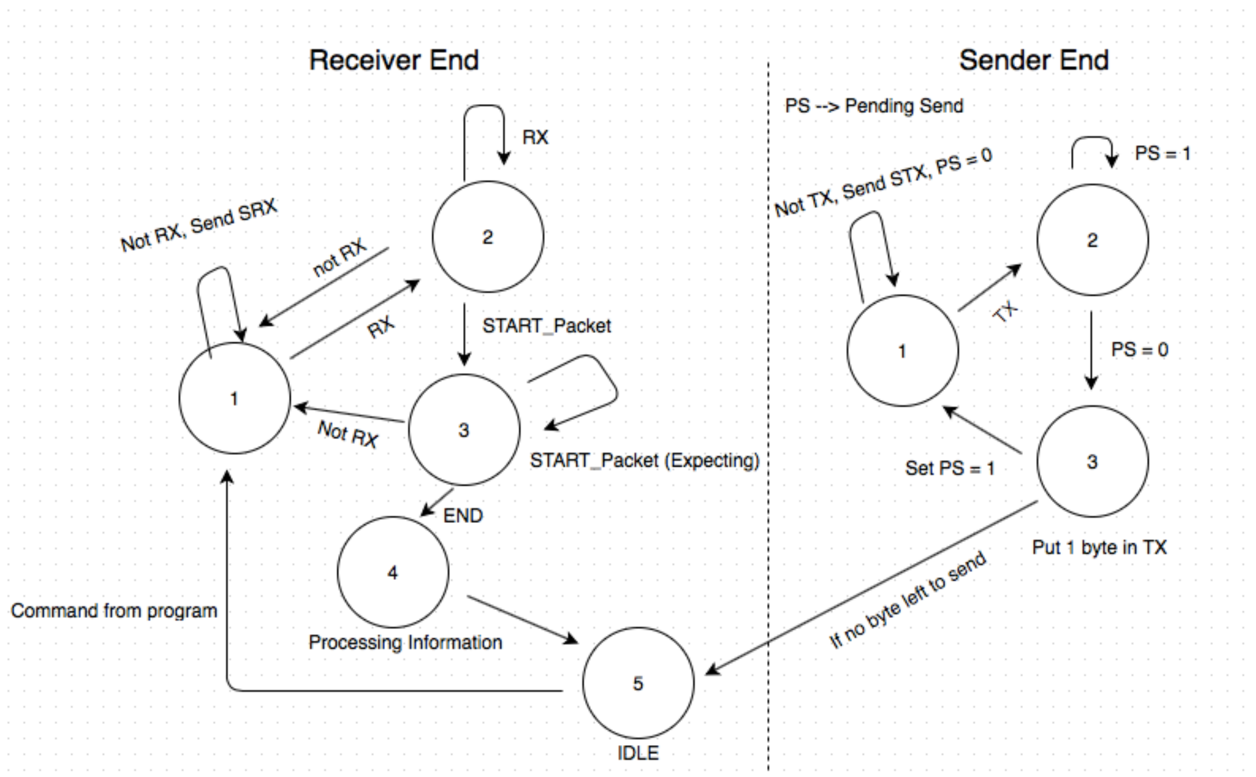


Figure3 : Receiver and Sender state diagrams for Layer 2

Layer 3: Transceiver Protocol: Go Back 1

This protocol (implemented in protocol_go_back_1.c) will instantiate the frequency adjustment process to enable transmission and receiver protocols that are implemented in wireless_transmission_sm.c (Layer 2). The following methods have been implemented to initialize the basic API for this protocol in Table 9:

Method	**Explanation**
protocol_go_back_1_init	• Method will initialize wireless_transmission_init • If mode is GO_BACK_ONE_MODE_SENDER (defined as 254), will set the SENDER state as idle (defined as 0) • Else will set the receiver state as idle (defined as 3)
protocol_go_back_1_get_state	• Method will retrieve and return the current state of internal FSM
protocol_go_back_1_get_received_data	• Method will retrieve the received new packet and copy data to dest parameter

protocol_go_back_1_receive	• Method will tell FSM to start the process of packet retrieval
protocol_go_back_1_send	• Method will tell FSM to start the process of packet transmission
next_id	• Method will get the next ID for the packet that will be sent (ranges from 1 to 250 which will not cover START_PACKET and END_PACKET)
consider_retransmit	• Method will count down time with timeout_count parameter (if 0, then retransmit data and change state to send mode)
transmit_ack	• ACK is a 1 byte packet which represents the ID of received packet • Method will transmit the ACK packet

Table 9: API for protocol_go_back_1

We implemented protocol_go_back_1_period method which will verify the mode (sender or receiver) and call protocol_go_back_1_periodic_sender/protocol_go_back_1_periodic_receiver appropriately. The following sequence of steps take place for each method:

protocol_go_back_1_periodic_sender:

1. If lower layer state (wireless_transmission_sm.c) is at error state, set state as error
2. If state is equal to send:
    a. If lower layer state is in idle state, then call wireless_transmission_receive_packet and set state to ACK
    b. If state is in ACK state:
        i. If lower layer state is not idle, then call consider_retransmit method
    c. Call wireless_transmission_get_received_packet method to receive the packet information (if transmission is not successful, call consider_retransmit method again)
    d. Reset state to idle once packet ID has been received

protocol_go_back_1_periodic_receiver:

1. If lower layer state (wireless_transmission_sm.c) is at error state, set state as error
2. If state is equal to receive:
    a. Call wireless_transmission_get_received_packet method to receive the packet information (if transmission is not successful, call wireless_transmission_receive_packet method)
    b. Call transmit_ack method and set state as ACK
3. If state is equal to ACK:
    a. Decrease ACK count

   b. If ACK count is 0, then set state as idle
   c. Else transmit call transmit_ACK method

Figure 4 shows state diagrams for both receiver and sender ends for Layer 3:

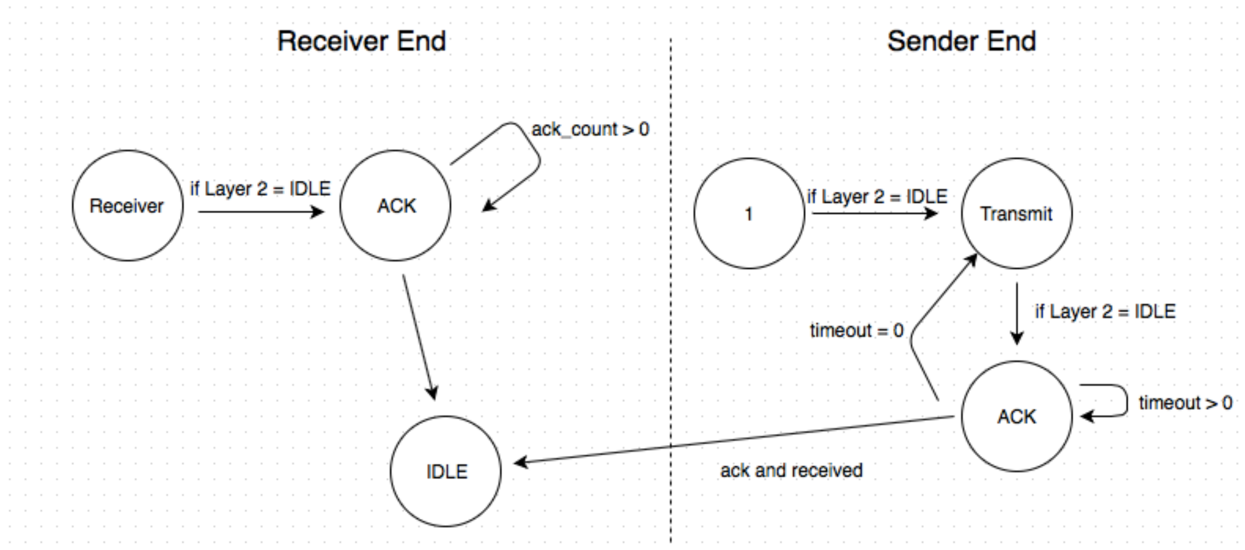


Figure 4: Receiver and Sender state diagrams for Layer 3

4.4 System Integration

To keep the integration process simple and efficient, we decided to divide our project into two parts:

1. **Displaying Board**
   This module configures both the LCD display board (STM32F429i) and wireless communication system (configured as SPI4). Once the coordinates (in feet) are received from the measuring board, feet-to-pixel conversion takes place and the corresponding trajectory is plotted on the LCD screen as a red line. Figure 5 shows the main method of this part of the project:

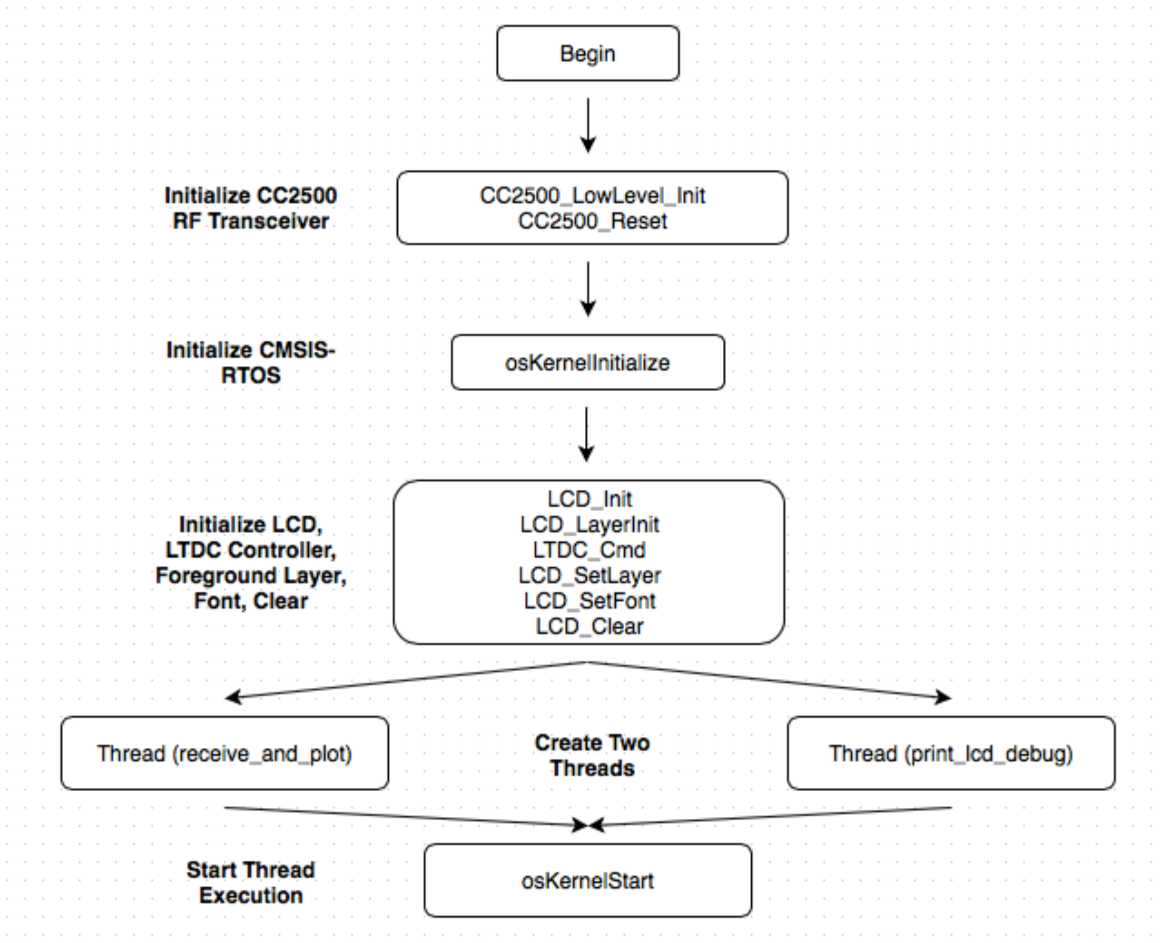


Figure 5: main method for displaying board

2. **Measuring Board**
   This module configures LSM9DS1 chipset (with accelerometer and gyroscope sensors), wireless communication system (configured as SPI2), and the measuring board (STM32F4). Once the user finishes his/her trajectory, coordinates which are stored in a buffer will be transmitted to the displaying board (using wireless_loop method). Figure 6 shows the main method of this part of the project:

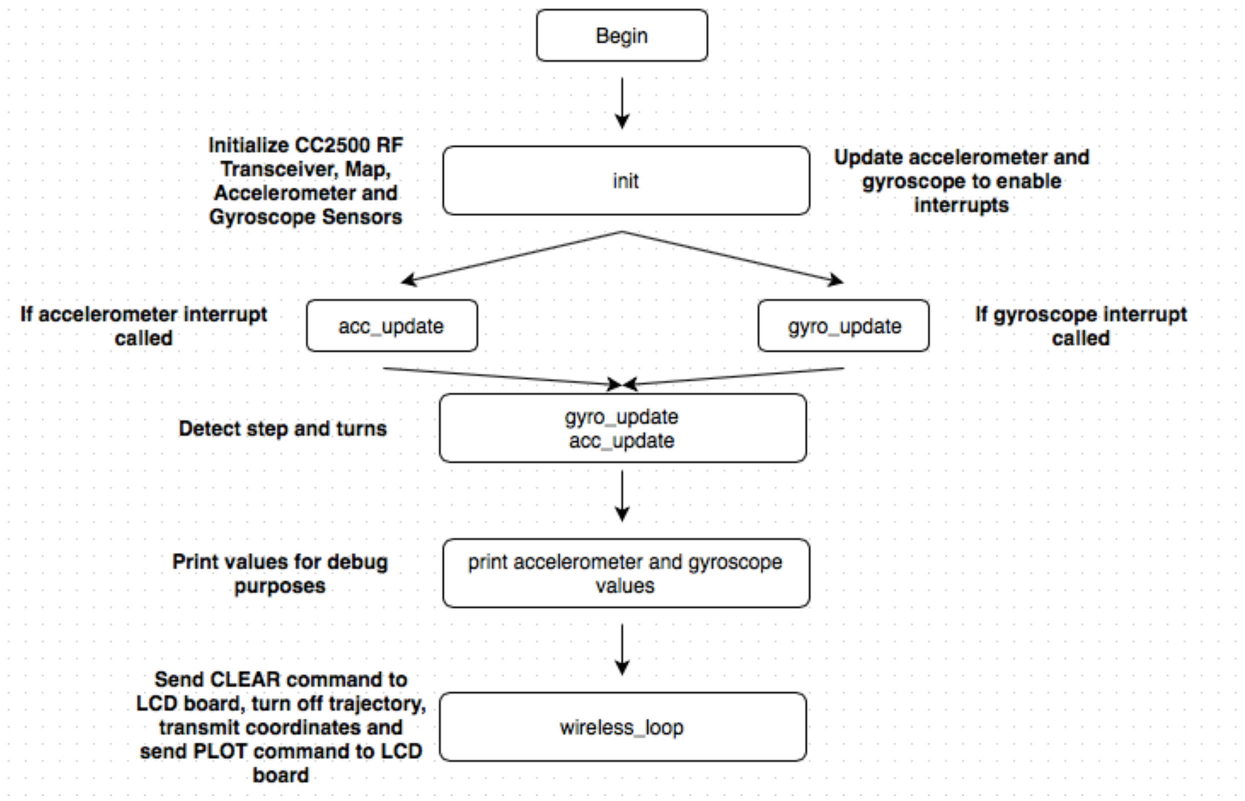

Begin
Initialize CC2500 RF Transceiver, Map, Accelerometer and Gyroscope Sensors
init
Update accelerometer and gyroscope to enable interrupts
If accelerometer interrupt called
acc_update
gyro_update
If gyroscope interrupt called
Detect step and turns
gyro_update acc_update
Print values for debug purposes
print accelerometer and gyroscope values
Send CLEAR command to LCD board, turn off trajectory, transmit coordinates and send PLOT command to LCD board
wireless_loop

Figure 6: main method for measuring board

GPIO pins for CC2500 and LSM9DS1 connected to measuring board and displaying board can be seen in Table 14 in Appendix.

5 Testing and Observation

5.1 LCD Screen (Testing: 28th November - 2nd December)

Initially, we mapped simple diagrams on the LCD display including drawing a horizontal line, full circle, triangle and text display. The next step was to set the background image as the 4th floor of the Trottier building. Since the floor image was not exactly oriented in 320X240 pixel format, we had to make slight modifications to the image format to completely display it on the LCD screen. In order to preserve the aspect ratio of the map, extra whitespace was added to the sides of the image instead of stretching the image.

Once the wireless communication system was complete, we decided to send the following dummy coordinates to test the LCD plot function with starting point as lab's entry door: 6 feet right, 12 feet up, 36 feet left, 15 feet up, 33 feet right, 12 feet up and 6 feet right. We observed the trajectory path on the LCD display which was almost accurate as seen in Figure 7:

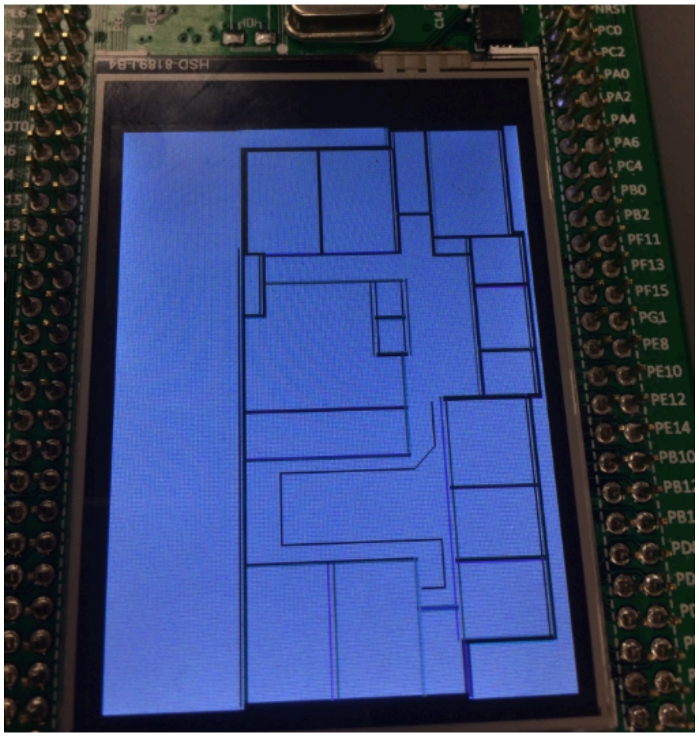


Figure 7: LCD test with dummy coordinates

After the measuring board was fully functional, we transmitted the coordinates from it which were received by the displaying board. The following trajectory path (red line) as seen in Figure 8 was observed which was very accurate:

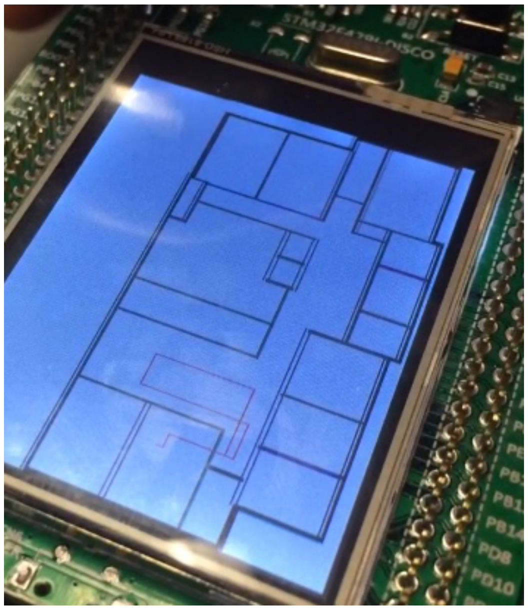


Figure 8: LCD test with actual coordinates

5.2 Testing of Accelerometer and Gyroscope Sensors

Accelerometer Sensor (Testing: 3rd December - 7th December)

After collecting measurements from the accelerometer sensor for 15 seconds, we observed that a right step occurred if a certain threshold was exceeded and a left step occurred if left step threshold was exceeded and then another threshold was exceeded. From Figure 9 and Figure 10, we observe that the threshold for left or right step is approximately 1100 mg/s^2. For left step and full step, the thresholds are approximately 600 mg/s^2 and 1100 mg/s^2. In Figure 9, the first 3 peaks are for left steps and then next 2 peaks are for the right steps.

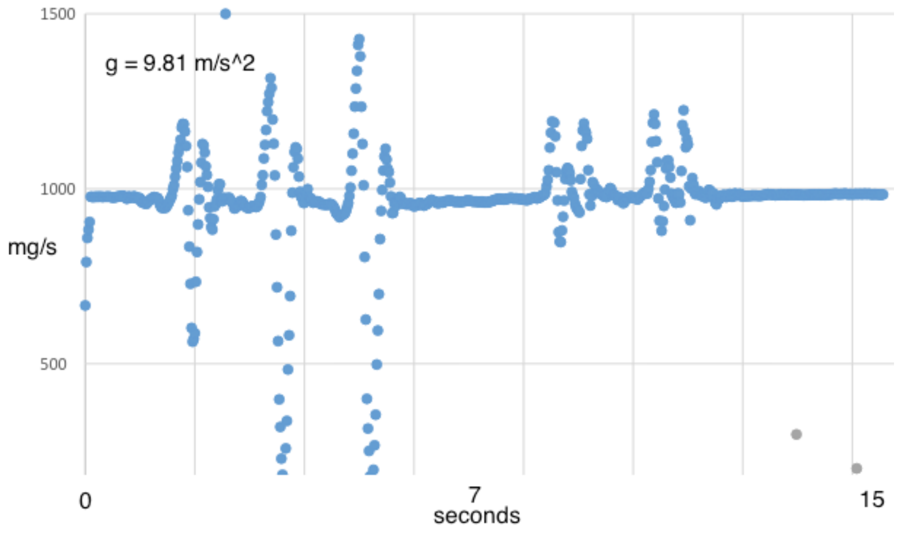


Figure 9: Accelerometer readings for separate left and right steps

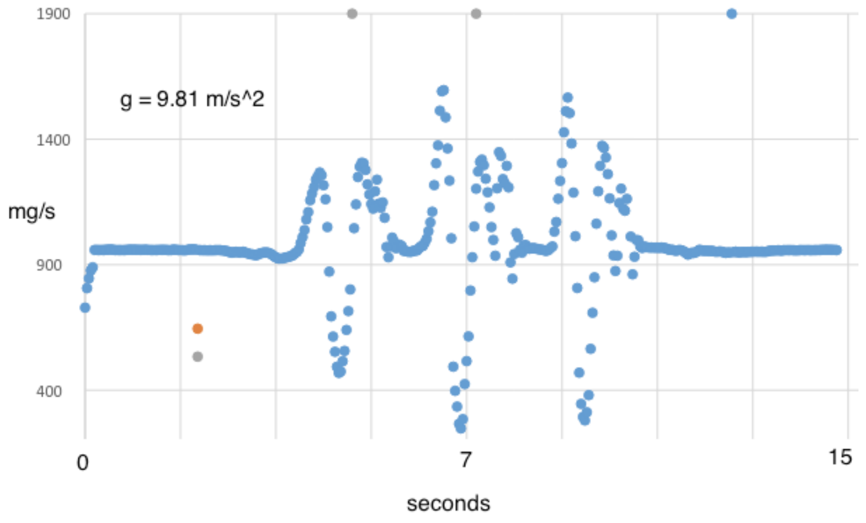


Figure 10: Accelerometer readings for combined steps (left and right)

To detect steps, we divided our tests into three parts as seen in Table 10:

Five Left Steps	Five Right Steps	Five Left/Right Alternating Steps
Right step 0	Right step 0	Right step 0
Left step 0	Left step 1	Left step 0
Left step 1	Right step 1	Right step 1
Right step 1	Right step 2	Left step 1
Left step 2	Right step 3	Left step 2
Right step 3	Right step 4	Right step 3

Table 10: Step Detection

The first step is always set as right step. We observe that for five left steps measured, the accelerometer sensor detected 3 left steps and 2 right steps and for five right steps measured, the accelerometer sensor detected 1 left step and 4 right steps. We deduced that for left steps and alternating steps, the step detection process was not accurate. However, for right steps, the step detection was quite accurate. Given these results, we decided to tie the measuring board to the right foot of the user to yield reliable results.

Gyroscope Sensor (Testing: 3rd December - 7th December)

After collecting measurements from the gyroscope sensor, we observed that a left turn (90 degrees) occurs when the x component of the gyroscope goes above a certain threshold and a right turn occurs when the x component of the gyroscope goes below a certain threshold. After plotting the measurements on a graph (in Figure 11), we observed that the left turn and right turn thresholds are 95000 milli degrees/s.

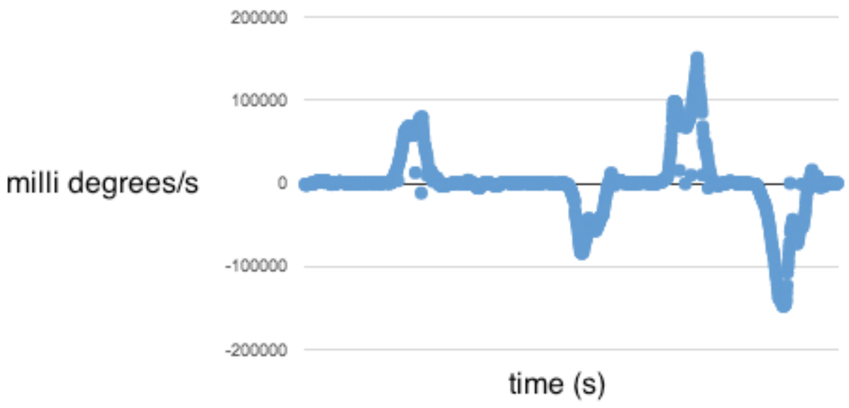


Figure 11: Gyroscope readings for left and right turn threshold limits

To detect turns, we divided our tests into two parts as seen in Table 11:

Left Turn Detection (4 turns)	Right Turn Detection (4 turns)
Left turn 0	Right turn 0
Left turn 1	Right turn 1
Left turn 2	Right turn 2
Left turn 3	Right turn 3

Table 11: Turn Detection

We observe that for all the left and right turns at 90 degrees are properly detected by the gyroscope sensor.

5.3 Wireless Communication System (Testing: 28th November - 4th December)

While testing the wireless communication system, we identified 3 key parameters which affect the time taken to transmit and receive data packets: packet count, packet length and distance between CC2500 chipsets. The testing data is recorded into table 13 in Appendix.

Packet Count

According to Table 13 in Appendix, it takes 21.1 seconds for 1 packet to be transmitted and received with packet length of 60 bytes and at distance of 0 feet (i.e. two CC2500 chips are very close). As we kept increasing the number of packets to 2 and 3 (with same packet length and distance), we observed that the time taken for transmission and reception were 42.2 seconds and 63.28 seconds. This highlighted an almost perfect linear relationship between packet count and time taken for packet transmission and reception given fixed packet length and distance.

Packet Length

According to Table 13 in Appendix, for 1 packet of length 12 bytes and distance of 0 feet, it took 8.14 seconds to transmit and receive data. For 1 packet of length 20, 40 and 60 bytes and distance of 0 feet, it took 10.52, 15,34 and 21.1 seconds. Hence we observed that as we increased the packet length, the time taken to transmit and receive data also increased.

Distance

According to table 13 in Appendix, it can be see that as the distance between two CC2500 chips increased, it took more time to transmit and receive data. For 3 packets of length 60 bytes each and at a distance of 2 feet, it took almost 336 seconds (worst case) to transmit and receive data. We also observed that the results wireless communication system was not reliable and practical for a distance greater than 2 feet.

5.4 Other Observations and Constraints

While operating at 2433.008 MH frequency, we observed that our wireless communication system was not able to properly receive and transmit data packets. This was because of other groups' wireless communication systems which were operating at almost similar frequencies which resulted in wave interferences and delayed the transmission and reception of data packets.

One of the biggest constraints was to carry a laptop that was connected to the measuring board which reduced the scope to walk freely. Also, each step and turn had to be very precise to ensure proper processing of coordinates of user's trajectory.

We observed that the cables, which were connected to all the devices including processor boards and chipsets were a bit loose. To handle this situation, we decided to use a breadboard and insert it inside the socks of the user which yielded greater stability and more accurate performance of the measuring board.

6 Conclusion

Given the hardware and software constraints, the overall design to track user's motion using accelerometer and gyroscope sensors and wirelessly transmit coordinates to the displaying board to plot the trajectory on the LCD screen was a success. All the coordinates were received and no information of data packets was lost during the wireless transmission and reception process.

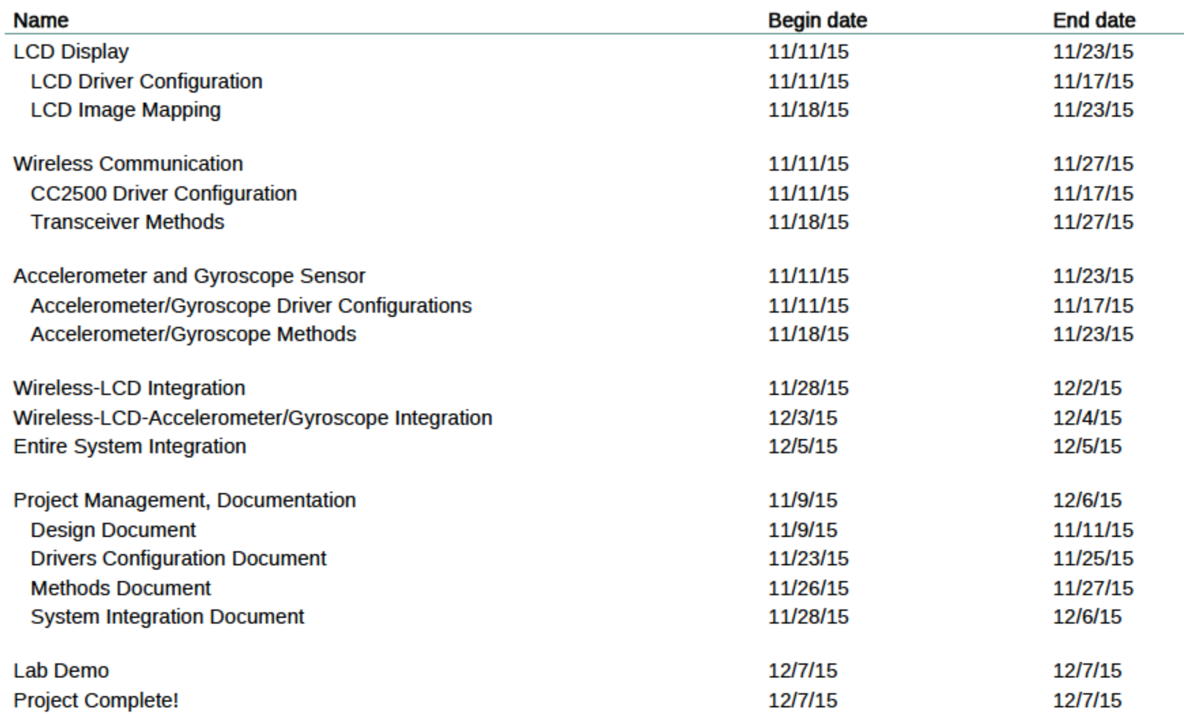
7 Task Breakdown

Table 12 summarizes the division of tasks between all the group members and Figure 12 and Figure 13 show the timeline of execution of all the tasks and Gantt Chart.

Tasks	**Member**	**Explanation**
Wireless Communication Configuration and Display Board Configuration	Hoai Phuoc Truong	Initialize and configure CC2500 transceiver module for RX/TX communication, implement driver functions using SPI protocol, high level logic for display board
Accelerometer Sensor Configuration and Data Collection Board Configuration	Michael Kourlas	Configure the accelerometer by implementing driver functions for the new board and use viterbi algorithm to continuously track

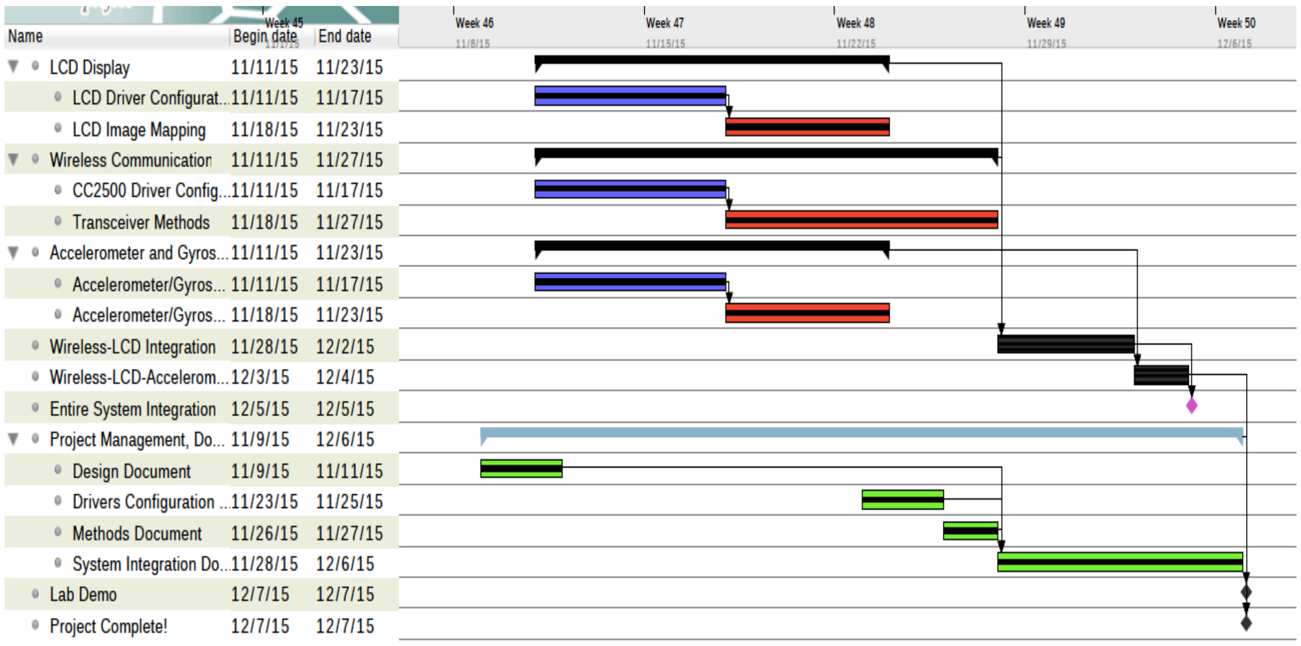
		movements, high level logic for data collection board
LCD Display Configuration and Display Board Configuration	Jacob Barnett	Configure LCD display on the board and implement driver functions, high level logic for display board
System Testing and Documentation	Ravi Chaganti	Collect data samples over multiple testing of individual and integrated system configurations (including successful and failed designs) and simultaneously document our implementation

Table 12: Division of labor among all important tasks



Name	Begin date	End date
LCD Display	11/11/15	11/23/15
LCD Driver Configuration	11/11/15	11/17/15
LCD Image Mapping	11/18/15	11/23/15
Wireless Communication	11/11/15	11/27/15
CC2500 Driver Configuration	11/11/15	11/17/15
Transceiver Methods	11/18/15	11/27/15
Accelerometer and Gyroscope Sensor	11/11/15	11/23/15
Accelerometer/Gyroscope Driver Configurations	11/11/15	11/17/15
Accelerometer/Gyroscope Methods	11/18/15	11/23/15
Wireless-LCD Integration	11/28/15	12/2/15
Wireless-LCD-Accelerometer/Gyroscope Integration	12/3/15	12/4/15
Entire System Integration	12/5/15	12/5/15
Project Management, Documentation	11/9/15	12/6/15
Design Document	11/9/15	11/11/15
Drivers Configuration Document	11/23/15	11/25/15
Methods Document	11/26/15	11/27/15
System Integration Document	11/28/15	12/6/15
Lab Demo	12/7/15	12/7/15
Project Complete!	12/7/15	12/7/15

Figure 12: Timeline of execution of all tasks



Name	Begin date	End date
LCD Display	11/11/15	11/23/15
LCD Driver Configurat...	11/11/15	11/17/15
LCD Image Mapping	11/18/15	11/23/15
Wireless Communication	11/11/15	11/27/15
CC2500 Driver Config...	11/11/15	11/17/15
Transceiver Methods	11/18/15	11/27/15
Accelerometer and Gyros...	11/11/15	11/23/15
Accelerometer/Gyros...	11/11/15	11/17/15
Accelerometer/Gyros...	11/18/15	11/23/15
Wireless-LCD Integration	11/28/15	12/2/15
Wireless-LCD-Accelerom...	12/3/15	12/4/15
Entire System Integration	12/5/15	12/5/15
Project Management, Do...	11/9/15	12/6/15
Design Document	11/9/15	11/11/15
Drivers Configuration ...	11/23/15	11/25/15
Methods Document	11/26/15	11/27/15
System Integration Do...	11/28/15	12/6/15
Lab Demo	12/7/15	12/7/15
Project Complete!	12/7/15	12/7/15

Figure 13: Gantt Chart

8 Appendix

8.1 References

1. Low-Cost Low-Power 2.4 GHz RF Transceiver. (n.d.). Retrieved December 7, 2015, pp. 1,19,21,23,24,65 from http://www.ti.com/lit/ds/swrs040c/swrs040c.pdf
2. Suyyagh, A. (2014, October 6). STM32F429 Discovery Board LCD Guide. Retrieved December 7, 2015, pp. 2,5.
3. Reference Manual. (n.d.). Retrieved November 1, 2015, pp. 208, from http://www.st.com/web/en/resource/technical/document/reference_manual/CD00171190.pdf
4. Application Note. (n.d.). Retrieved October 29, 2015, pp. 14,15,16, from http://www.st.com/web/en/resource/technical/document/application_note/CD00268887.pdf
5. Smith, S. (1997). Moving Average Filters. In The scientist and engineer's guide to digitalsignal processing (pp. 277-280). San Diego, Calif.: California Technical Pub.
6. Application Note - LIS302DL. (n.d.). Retrieved November 2, 2015, pp. 13,14, from http://www.st.com/web/en/resource/technical/document/application_note/CD00098549.pdf
7. Majerle, T. (2014, May 10). STM32F4 PWM tutorial with TIMERs - STM32F4 Discovery. Retrieved November 3, 2015, from http://stm32f4-discovery.com/2014/05/stm32f4-stm32f429-discovery-pwm-tutorial/

8. ECSE 456 Course Administration, Lecture 8, Montreal: McGill University, 2015.
9. A. Suyyagh and B. Nahill, *Real-Time Operating Systems: ARM's CMSIS as a case-study,* Montreal: McGill University, 2015.

8.2 Figures

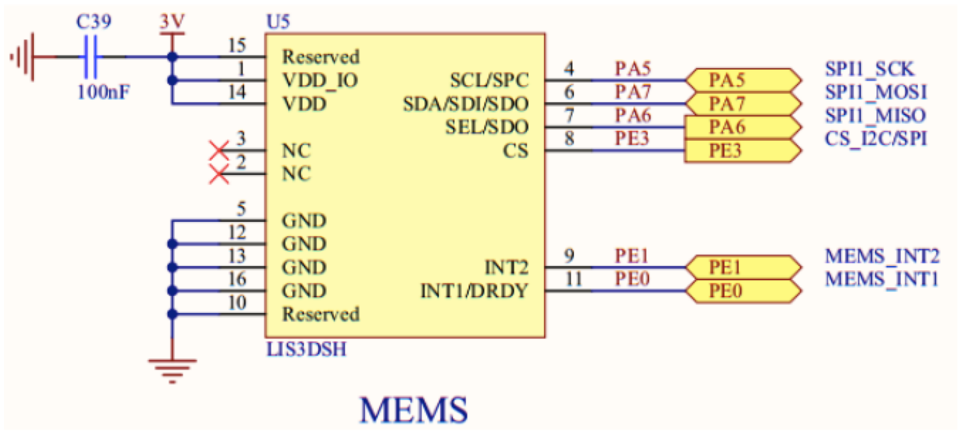


Figure 14: SPI protocol and initialization

8.3 Tables

Test No.	Packet count	Packet len(bytes)	Distance (ft)	Time (f[sys])	Time (s)
1	1	60	0	1055	21.1
2	1	60	0	1055	21.1
3	1	60	0	1071	21.42
4	1	60	0	1055	21.1
5	1	60	0	1054	21.08
6	2	60	0	2109	42.18
7	2	60	0	2109	42.18
8	2	60	0	2109	42.18
9	2	60	0	2109	42.18
10	2	60	0	2110	42.2
11	3	60	0	3164	63.28
12	3	60	0	3180	63.6
13	3	60	0	3164	63.28
14	3	60	0	3165	63.3
15	3	60	0	3163	63.26
16	6	60	0	6328	126.56
17	2	60	2	3239	64.78
18	2	60	2	3239	64.78
19	2	60	2	3238	64.76
20	2	60	2	3804	76.08
21	2	60	2	3804	76.08
22	3	60	2	6554	131.08
23	3	60	2	13429	268.58
24	3	60	2	13356	267.12
25	3	60	2	16848	336.96
26	3	60	2	13462	269.24

27	1	12	1	407	8.14
28	1	12	1	423	8.46
29	1	12	1	407	8.14
30	1	12	1	406	8.12
31	1	12	1	407	8.14
32	1	12	0	407	8.14
33	1	12	0	407	8.14
34	1	12	0	422	8.44
35	1	12	0	422	8.44
36	1	12	0	407	8.14
37	1	20	0	526	10.52
38	1	20	0	526	10.52
39	1	20	0	542	10.84
40	1	20	0	527	10.54
41	1	20	0	526	10.52
42	1	40	0	767	15.34
43	1	40	0	767	15.34
44	1	40	0	782	15.64
45	1	40	0	766	15.32

Table 13: Packet count, length and distance for testing of wireless communication system

Measuring board		
GPIO Pin	**Description**	**Functional chip**
PA5	SPI CLK	CC2500 (SPI1)
PA6	SPI MISO	
PA7	SPI MOSI	
PB9	SPI CS	
PB13	SPI CLK	LSM9DS1 (SPI2)
PB14	SPI MISO	
PB15	SPI MOSI	
PB12	SPI CS	
LCD board		
PE2	SPI CLK	CC2500 (SPI4)
PE5	SPI MOSI	
PE6	SPI MISO	
PC12	SPI CS	

Table 14: Peripheral chipsets GPIO pins usage