McGill University

Microprocessor Systems

ECSE 426

Patient Localization in Fixed Space Environment

Author:
Osama Abdulhadi
Kevin Cadieux
Maxime Grégoire
Vincent Arabian

Instructor:
Mark Coates

December 4, 2014

1 Abstract

In this experiment, a platform for localizing patients entering a restricted area was designed and built. The platform used both short and long-range measurements in order to localize a patient walking in a forbidden zone. A proximity sensor mounted on a servo motor was used to do the short-range positioning. The detection of patient entering the forbidden zone was done using the received signal strengths between different wireless links. The patient's coordinates were displayed on the LCD of the STM32F429 Discovery board.

2 Problem Statement

There is a need for a platform for patient positioning in a restricted area. Specifically, the platform should locate a patient entering a forbidden zone of 2 meters long by 3 meters wide. The patient position should be displayed on a LCD in near real time. To build this platform, the following electronic material is available:

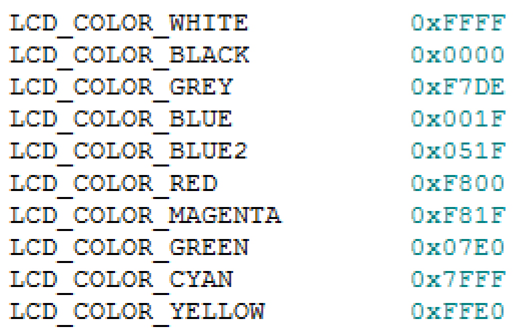
- 2 Discovery STM32F407 boards
- 1 Discovery STM32F429 board
- 3 CC2500 wireless SOC
- 1 Proximity sensor
- 1 Servo motor

The main challenge will be to track the patient's position correctly using a mix of received signal strength of wireless boards and readings from the proximity sensor.

3 Theory and Hypotheses

3.1 LCD Theory

The STM32F429 Discovery Board packs a LCD (liquid-crystal display) whose pixels are mapped to a subset of the board's RAM. The LCD resolution is 240 by 320 pixels. Using the provided LCD library, it is possible to draw lines, rectangles, circles, polygons and write texts at specific lines. Those shapes and texts can be displayed in the colors listed in Figure 1 using the associated codes.



LCD_COLOR_WHITE	0xFFFF
LCD_COLOR_BLACK	0x0000
LCD_COLOR_GREY	0xF7DE
LCD_COLOR_BLUE	0x001F
LCD_COLOR_BLUE2	0x051F
LCD_COLOR_RED	0xF800
LCD_COLOR_MAGENTA	0xF81F
LCD_COLOR_GREEN	0x07E0
LCD_COLOR_CYAN	0x7FFF
LCD_COLOR_YELLOW	0xFFE0

Figure 1 - The available LCD colors and their codes

It is also possible to display numerical pictures (in JPEG, BMP, PNG or GIF format) by hardcoding every pixel of the pictures in the code. A tool named STMImager has been created for that purpose.

The LCD has two visible layers, the background and the foreground. It is possible to adjust the transparency of the layers to create different effects. Different texts and drawings can be displayed on different layers. This makes it easy to create nice animations.

3.2 General-purpose Timers

The general-purpose hardware timers (TIM2-TIM5) can be set to tick at a frequency. These timers are then used to signal interrupts that will be handled by the nested vectored interrupt controller (NVIC) and execute a specific process. The NVIC already has pre-established internal connections to the general-purpose timers, so there is no additional multiplexing required to configure the TIM and NVIC, these handlers are defined in the driver for the TIM peripherals Equation 1. The timer's interrupting frequency can be calculated using the following formula:

$$\frac{\text{TimerClockingFrequency}}{(\text{Period} \times \text{prescalar})} = \text{desired rate}$$

Equation 1 - Desired Rate

3.3 Pulse-Width Modulation (PWM) & Motor Control

3.3.1 Description

Pulse-width modulation allows us to control the angle of the servo motor. It achieves this by modulating the duty cycle of the square waves. By controlling the width of the square wave, the digital signal can input a specific analog signal level to an electronic device. An example of different PWM signals is shown on Figure 2.

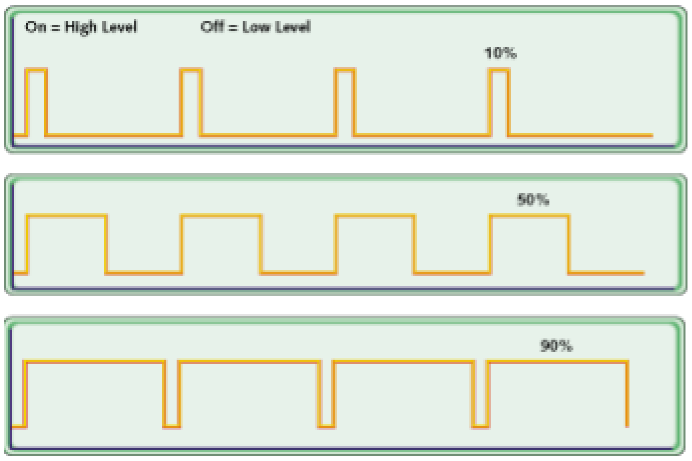


Figure 2 - Different Pulse Width Modulation Signals

3.3.2 Hardware Generation

The STM32F429 Discovery board has sets of general purpose full featured timers that can be used for PWM generation. Each timer has a set of channel that it is connected to and can send a PWM signal through an enabled channel to its associated GPIO pin.

3.3.3 Motor Control

The motor's datasheet (STMicroelectronics, 2014) that the pulse width for neutral position is 1500us. In order to rotate the motor head, this pulse width can be increased and reduced within a certain range. This range is specified between 600-2400μs, with a 20ms pulse cycle. The formula to calculate the pulse based on angle is the following:

$$pulse = Natural\ Pulse\ + (angle * multiplier)$$

Equation 2 - Pulse

Where the value of angle ranges between -90 and 90.

3.4 Proximity Sensor

The proximity sensor is a distance measuring sensor unit composed of an integrated combination of position sensitive detector, an infrared emitting diode, and a signal processing circuit.

4 Implementation

4.1 The solution architecture

The solution was built modularly, following the block diagram of **Error! Reference source not found.**.

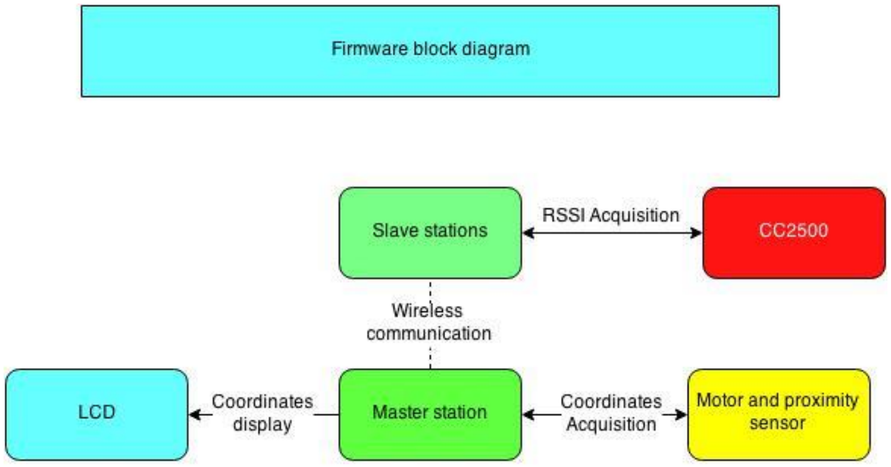


Figure 3 - Solution Block Diagram

4.2 CC2500 Driver

4.2.1 SPI Configuration

Communicating with the CC2500 wireless chip is done using the SPI protocol. For this reason, the main low level initialization task of the CC2500 driver consists of setting up one of the SPI modules available on either the STM32F407 or the STM32F429 board. The SPI settings differ depending on which STM board is being used. Table 1 describes the configuration for the F407 board, and Table 2 describes the configuration for the F429 board. Finally, Table 3 shows the configuration that is common to both boards.

Table 1 - SPI configuration for the F407 board.

Parameter	Value	Justification
SPI Module	SPI2	SPI2 is not tied to any other function, such as the accelerometer.
SCLK Pin	B13	SCLK for SPI2 is the alternate function of B13
CSN Pin	B12	CSN for SPI2 is the alternate function of B12
MISO Pin	B14	MISO for SPI2 is the alternate function of B14
MOSI Pin	B15	MOSI for SPI2 is the alternate function of B15
Baud Rate Prescaler	8	Minimum required prescaler to obtain a safe frequency for burst mode SPI transfers.
Peripheral Bus	APB1	SPI2 is connected to APB1.

Table 2 - SPI configuration for the F429 board.

Parameter	Value	Justification
SPI Module	SPI4	SPI4 is not tied to any other function, and is using free pins that are not tied to other modules such as the SDRAM or the LCD.
SCLK Pin	E2	SCLK for SPI4 is the alternate function of E2
CSN Pin	E4	CSN for SPI4 is the alternate function of E4
MISO Pin	E5	MISO for SPI4 is the alternate function of E5
MOSI Pin	E6	MOSI for SPI4 is the alternate function of E6
Baud Rate Prescaler	16	Minimum required prescaler to obtain a safe frequency for burst mode SPI transfers.
Peripheral Bus	APB2	SPI4 is connected to APB2.

Table 3 - Common SPI configuration for all boards.

Parameter	Value	Justification
Direction	Full Duplex	Required by the CC2500.
Data Size	8 bits	Required by the CC2500.

Clock Polarity	Active Low	Required to satisfy the SPI timing constraints of the CC2500.
Clock Phase	1 Edge	Required to satisfy the SPI timing constraints of the CC2500.
Slave Select Mode	Software	The slave select pin is managed by software, independently of the SPI module.
First Bit	MSB	Required by the CC2500.
CRC Polynomial	7	Default value.
SPI Mode	Master	The CC2500 is a slave.

The selection of the SPI pins and busses was guided by the datasheet of each ST board (STMicroelectronics, 2014) (STMicroelectronics, 2012). The other SPI configuration parameters were chosen to satisfy the constraints established by the CC2500 datasheet (Texas Instruments, 2014).

Special care needed to be taken when selecting the baud rate prescaler, as the selected SPI modules are not always connected to the same peripheral bus. Also, the clock management unit of the ST boards can be configured to provide different clock speeds. For all these reasons, the peripheral bus on which the SPI module is connected may vary in clock speed. To avoid having the user specify the baud prescaler every time such changes are made, the CC2500 driver we developed dynamically chooses the baud rate prescaler based on the current clock speed settings written in the reset and clock control (RCC) registers of the board. The baud rate prescaler values shown in Table 1 and Table 2 are the values that correspond to our specific RCC settings.

In addition to configuring the SPI module, the individual pins used in SPI communication had to be configured as well. The SCLK, MOSI, and MISO pin configuration can be seen in Table 4. Since the slave select pin is not managed by the SPI module hardware but with software, it needed to be configured differently. Its configuration is shown in Table 5.

Table 4 - GPIO configuration for SPI pins.

Parameter	Value	Justification
Mode	Alternate Function	SPI
Speed	50 MHz	Largely sufficient because the SPI clock will run at around 6.5 MHz.
Pull-Up/Pull-Down	Pull-Down	Input voltage is pulled down when no input is present.

Table 5 - GPIO configuration for the slave select SPI pin.

Parameter	Value	Justification
Mode	Output	Slave select is managed by software so a general output GPIO port is needed.
Speed	50 MHz	Largely sufficient because the SPI clock will run at around 6.5 MHz.

Output Type	Pull up / Pull down	Output needs to be pulled up to Vdd when the pin is set.

4.2.2 Programming Interface

Our CC2500 driver comes with a programming interface that allows the user to take advantage of every feature of the CC2500 chip.

4.2.2.1 Register and Command Strobe Macros

Every configuration register, status register, and command strobe can be named using the appropriate macro. All macros take the following form: CC2500_<name of register or command strobe>_<R or W (command strobes only)>. In the case of registers (configuration or status), the user only needs to provide its name (e.g. CC2500_PKTLEN). For command strobes, the user needs to append the name with either _R or _W, depending on whether the read or the write version of the strobe is required (e.g. CC2500_STX_R).

4.2.2.2 Configuration Register Setting Macros

All possible settings or values for the configuration and status registers are accessible via macros. For example, to configure the PKTCTRL0 register to use CRC with variable length packets, one would write: CC2500_PKTCTRL0_CRC_EN | CC2500_PKTCTRL0_Length_VARIABLE. We refer the reader to the cc2500.h header file to see a list of these macros, and to the CC2500 datasheet to understand what they do (Texas Instruments, 2014).

4.2.2.3 Initialization Structure

For convenience, an initialization structure called CC2500_InitTypeDef is made available. It contains a field for every configuration register of the CC2500 chip. This structure can be written to and passed to the CC2500_WriteConfig function to set up all of the CC2500 registers at once. The fields of this structure can be initialized to their default values using the CC2500_StructInit function.

4.2.2.4 Chip Status Structure

Most driver functions return an instance of the CC2500_StatusTypeDef structure, which contains all the status information of the chip that was returned from the previous interaction with it. The information available is:

- Chip Ready bit.
- Current state.
- FIFO bytes available.

4.2.2.5 Driver Functions

The driver provides many functions to interact with the CC2500 chip. These functions are described in Table 6.

Table 6 - CC2500 driver functions.

Name	**Parameters**	**Description**
CC2500_Init	None	Initializes SPI communication with the CC2500 chip. Must

		be called before anything else.
CC2500_WriteConfig	CC2500_InitTypeDef* init_struct	Writes all the configuration registers of the CC2500 chip based on the values found in <init_struct>.
CC2500_StructInit	CC2500_InitTypeDef* init_struct	Initializes an initialization structure to the default values.
CC2500_ReadConfigRegister	uint8_t addr uint8_t* destination	Reads the configuration register at the address specified in <addr>, and writes the content of this register in <destination>.
CC2500_ReadStatusRegister	uint8_t addr uint8_t* destination	Reads the status register at the address specified in <addr>, and writes ton content of this register in <destination>.
CC2500_ReadRxFIFO	uint8_t* destination, uint8_t nb_bytes	Reads the number of bytes specified in nb_bytes and writes them in <destination>.
CC2500_ReadPATABLE	CC2500_PATABLETypeDef* patable	Reads the PA TABLE from the CC2500 chip and writes it in <patable>.
CC2500_ReadConfiguration	CC2500_InitTypeDef* init_struct	Reads all the configuration registers from the CC2500 chip and fills <init_struct> with their values.
CC2500_WriteConfigRegister	uint8_t addr uint8_t byte	Writes <byte> into the configuration register specified by <addr>.
CC2500_WriteTxFIFO	uint8_t* data, uint8_t nb_bytes	Writes a number of bytes specified by <nb_bytes> into the TX FIFO using the bytes pointed to by <data>.
CC2500_WritePATABLE	CC2500_PATABLETypeDef* patable	Writes <patable> into the PA TABLE of the CC2500.

All but the first 3 functions in this table return a CC2500_StatusTypeDef structure, which contains the status of the chip. Also, the driver automatically opts for burst transfer SPI operations when it is advantageous to do so, such as during FIFO operations.

4.3 Wireless Module

The wireless module sits on top of the CC2500 driver and provides higher level functions that encapsulate all interactions with the CC2500 chip. It provides support for sending and receiving packets without forcing the user to worry about the intricacies of the CC2500 chip, such as initializing the chip with the correct parameters, or configuring interrupts.

4.3.1 CC2500 Configuration

Table 7 below shows how the CC2500 has been configured to obtain the desired behavior and packet format.

Table 7 - CC2500 configuration

Parameter	**Value**	**Justification**
PATABLE[0]	0xFF	Used for maximum output power.
FSCTRL1	0x0C	
FSCTRL0	0x00	FSCTRL values were provided by SmartRF Studio.
FREQ2	0x5D	
FREQ1	0x93	
FREQ0	0xC5	FREQ values were selected to obtain the desired frequency of 2,433,008 KHz.
MDMCFG4	0x0E	
MDMCFG3	0x3B	
MDMCFG2	0x73	
MDMCFG1	0xC2	
MDMCFG0	0xF8	MDMCFG values were mostly provided by SmartRF Studio. A slight modification has been made to MDMCFG1 to activate forward error correction on the packets by setting bit 7. We found through experimentation that this greatly reduced packet loss due to error.
DEVIATN	0x00	Provided by SmartRF Studio.
FREND1	0xB6	
FREND0	0x10	FREND values were provided by SmartRF Studio.
MCSM1	0x30 / 0x32	0x30 makes the chip go back to IDLE state after a packet has been sent or received. 0x32 has the same behavior except that the chip will stay TX after a packet has been sent. This is used when we wish to send a packet burst without going to IDLE mode between each packet. We have found through experimentation that this significantly speeds up burst transmissions.

MCSM0	0x18	Instructs the chip to use automatic calibration when going from IDLE state to RX or TX states.
FOCCFG	0x1D	Provided by SmartRF Studio.
BSCFG	0x1C	Provided by SmartRF Studio.
AGCTRL2	0xC7	
AGCTRL1	0x40	
AGCTRL0	0xB0	AGCTRL values were provided by SmartRF Studio.
FSCAL3	0xEA	
FSCAL2	0x0A	
FSCAL1	0x00	
FSCAL0	0x19	FSCAL values were provided by SmartRF Studio.
FIFOTHR	0x0F	Instructs the chip to use a threshold of 1 for the TX FIFO and a threshold of 64 for the RX FIFO. This value has been chosen to facilitate interrupt generation when a packet or packet burst has been sent (See IOCFG0 below).
IOCFG0	0x01 / 0x02	0x01 is used to instruct the chip to generate an interrupt either when the RX FIFO is above threshold or a packet has been sent. Since our packet length is below the RX FIFO threshold (see FIFOTHR above and PKTLEN below), the interrupt is always generated when a packet has been received. This setting is written to the CC2500 before receiving a packet. 0x02 is used to instruct the chip to generate an interrupt when the TX FIFO goes below the TX FIFO threshold. Since the TX FIFO threshold is 1 (see FIFOTHR above), this effectively means that an interrupt occurs when the TX FIFO is emptied, and thus when a packet or packet burst is finished being sent. This setting is written to the CC2500 before transmitting a single packet or a packet burst.
PKTCTRL1	0x07	Used to append the reception status at the end of every packet received and to perform an address check for every packet received.
PKTCTRL0	0x44	Used to perform data whitening on the packets, to enable CRC verification for each packet, and to use fixed packet length. We chose to use data whitening to obtain a smooth power distribution over the occupied bandwidth.
PKTLEN	0x2	Sets the fixed packet length to 2: 1 address byte + 1 payload byte.

ADDR	0x00	Default address. Will change every time the user wishes to receive a packet. The address is specified every time one of the ReceivePacket functions are called.
CHANNR	0x00	Channel 0 is always used.

4.3.2 Packet Format

For our purposes, the packet format only needs to support sending a very small number of commands to different addresses. The command can be encoded as a single byte. Therefore, only 1 address byte and 1 command byte are required, for a total of 2 bytes per packet.

4.3.3 Programming Interface

The programming interface is comprised of a WLESS_StatusCodeTypeDef structure and a small number of functions. All packet related functions (WLESS_SendPacket, WLESS_ReceivePacket, WLESS_SendPacketBurst, and WLESS_ReceivePacketVerified) return an instance of the WLESS_StatusCodeTypeDef structure to indicate if the operation was a success or a failure and why. Table 8 summarizes all the functions available to the user of this module. In this table, WLESS_PACKET_LENGTH has a value of 1 because only the payload byte is visible to the user. The wireless module takes charge of adding and removing the addresses from the packets.

Table 8 - Wireless module functions.

Name	**Parameters**	**Description**
WLESS_Init	None	Initializes the wireless module according to the CC2500 configuration shown in Table 7.
WLESS_SendPacket	uint8_t* packet_bytes, uint8_t address	Sends a number of bytes equal to WLESS_PACKET_LENGTH pointed to by <packet_bytes> to the address specified in <address>. The address is automatically added to the packet, so <packet_bytes> only contains the actual payload. This is a blocking function that waits until packet transmission is over.
WLESS_SendPacketBurst	uint8_t* packet_bytes, uint8_t address, uint8_t burst_size	Does the same operation as WLESS_SendPacket, except that it is repeated for the number of times specified in <burst_size>. This is typically used to send multiple copies of the same packet in order to increase the chance that at least one of them is received without error by the receiver. To perform this burst

		operation, the value of MCSM1 is changed to 0x32 to avoid having the CC2500 go back to IDLE between every packet. This is a blocking function that waits until the packet burst transmission is over.
WLESS_ReceivePacket	uint8_t address, uint8_t* packet_bytes	Receives a packet at the address specified in <address> and writes it at the location pointed to by <packet_bytes>. <packet_bytes> is expected to be of size WLESS_PACKET_LENGTH. The address is removed from the packet before writing into <packet_bytes>. This is a blocking function that waits until a packet is received.
WLESS_ReceivePacketVerified	uint8_t address, uint8_t* packet_bytes	Does the same operation as WLESS_ReceivePacket, except that it discards the packet and waits for another one if CRC verification has failed. This is a blocking function that waits until a packet that has passed CRC verification is received.
WLESS_GetLatestRSSI	None	Returns a byte containing the most recent RSSI value reported by the CC2500 chip when a packet was last received.
WLESS_GetLatestDecibelRSSI	None	Returns the same as WLESS_GetLatestRSSI, except that the value is converted into Db format.

4.4 Localization Algorithm

The localization of intruders is done using 3 wireless stations. The setup is displayed in Figure 4.

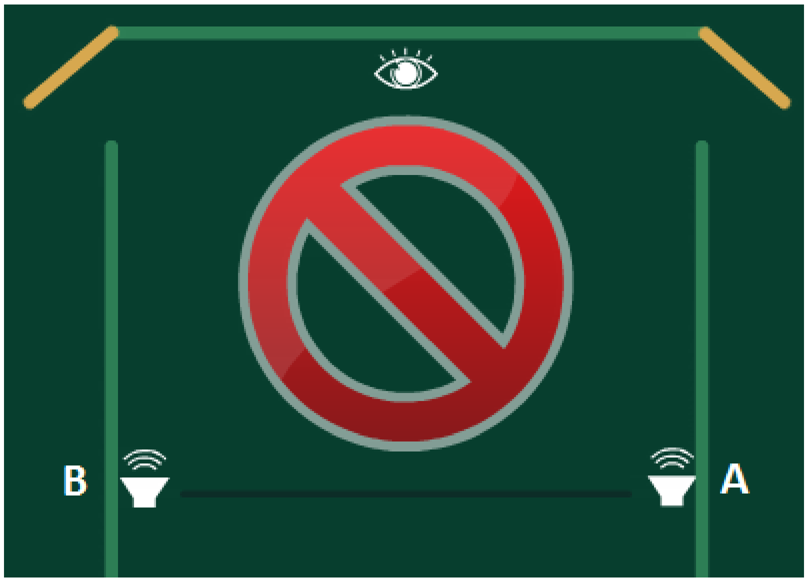


Figure 4 - Localization Setup

Stations A, B, and the main station (displayed as an eye icon on Figure 4) all have a different role to play. These are described in the sections below.

4.4.1 Sending Station A

This station continuously sends dummy packets addressed to station B. It never stops doing so.

4.4.2 Receiving Station B

This station continuously receives station A's packets and evaluates their received signal strength. If at some point the received signal strength significantly reduces below the baseline, a person is considered having passed between stations A and B. At this point, a burst of 10 "person detected" packets is sent to the main station, and station B goes back to evaluating the received signal strength of station A's packets to detect other persons. The reason for sending 10 packets to the main station is to make sure that the station receives at least one of them in the event that some packet losses occur. This process continues forever.

As various factors may affect the baseline received signal strength, it was impossible to have an absolute threshold at which a person is detected. Instead, every received signal strength indicator (RSSI) value is passed through a 200-value moving average filter, whose average is considered being the baseline. This way, the baseline value can adapt itself to the environment as it changes. If the RSSI value is 8 below this baseline, we consider that a person has passed.

Similarly, every packet received from A may be subject to noise, and therefore there may be times when a single isolated packet is below the threshold but that no one passed between the two stations. To mitigate this problem, an additional 20-value deep moving average filter was used, whose value is compared with the baseline.

4.4.3 Main Station

When the main station receives a “person detected” packet, it initiates a 180 degree sweep in front of itself using a proximity sensor attached to a servomotor. Once the intruder’s position is detected, it is displayed on the LCD screen. The 180 degree sweep is done in repetition until another “person detected” packet is received, at which point the person is considered having left the area and the sweeping stops. The sweeping continues again once a new “person detected” packet is received.

4.5 LCD Implementation

4.5.1 The representation of the patient’s location

Two metrics are available to determinate the patient’s coordinates: its numerical coordinates and its position on the map.

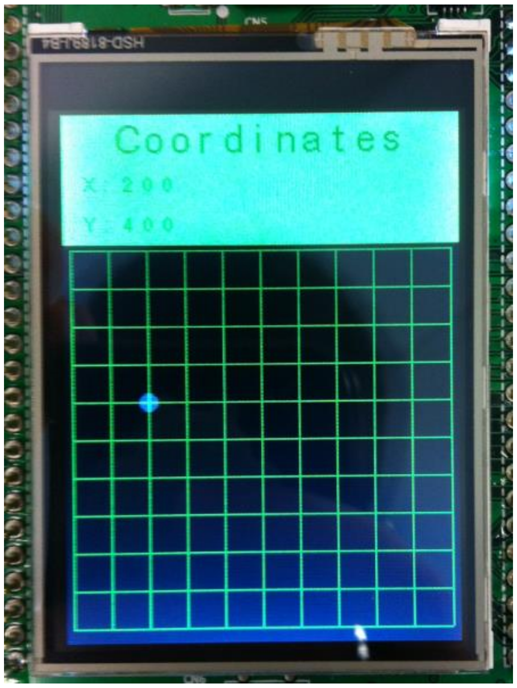


Figure 5 - The LCD display

4.5.1.1 The numerical coordinates

The patient’s location is represented on a 2-axis Cartesian system. As seen on Figure 5, these numerical coordinates are written on the top part of the display. They are rewritten using the LCD_DisplayStringLine function every time the function updateCoords is called.

4.5.1.2 The map

In order to display the position of the patient in real time, the LCD was used extensively. The first step was to display a map where the position of the patient would be shown. The map consists of a gridded square of 240 by 240 pixels displayed at the bottom part of the LCD as seen on Figure 5. The square was drawn using the LCD_DrawFullRect function. To make the grid, horizontal and vertical lines were drawn using the LCD_DrawLine function repeatedly. The map was drawn on the background layer, in order to make it easy to draw and erase patient's positions on the foreground layer while not having to redraw the whole grid. The map only needs to be drawn once at the initialization. The patient's position is represented by a white circle that is redrawn every time the function updateCoords is called.

4.6 Proximity Sensor

The proximity sensor is a point-to-point analogue module which detects the presence of objects at a distance between 20 and 150 cm. In order to extract information from this sensor the GPIO PA5 configuration had to be set to analogue, which is then connected to ADC1 and sampled at 25Hz.

The proximity sensor outputs a voltage reading, hence we had to conduct a set of experiments by placing an object at exact distances from the sensor and measuring the voltage response as shown in Table 9.

After measuring the voltage response, we used excel and MATLAB to find an accurate fitting function which maps voltage to distance as shown in Equation 3 and Figure 6 and Figure 7.

$$\boldsymbol{Y = 77504x^{-1}}$$

Equation 3 - Power Function

Proximity Sensor Readings

Distance (cm)	Voltage (V)
10	3738.65
15	3762.56
20	3408.68
30	2596.98
40	2009.91
50	1574.69
60	1178.18
70	1038.04
80	905.32
90	795.39
100	722.08
110	649.94
120	595.41
130	546.92
140	506.27
150	493.27
160	476.71

Table 9 - Proximity Sensor Readings

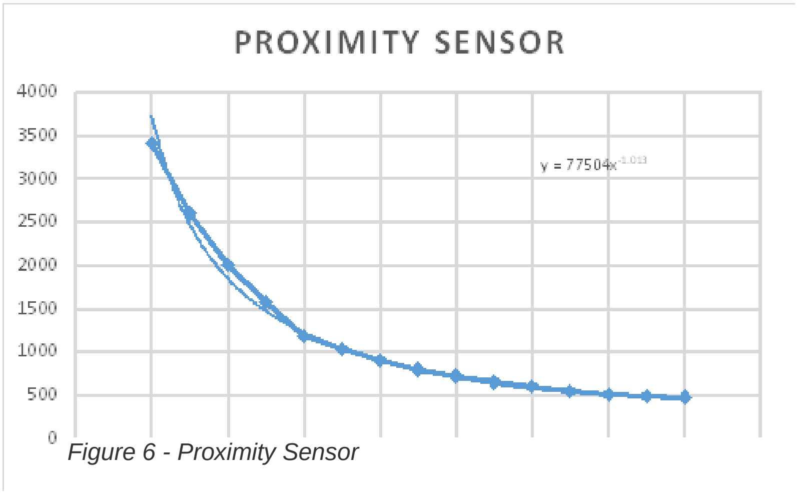


Figure 6 - Proximity Sensor

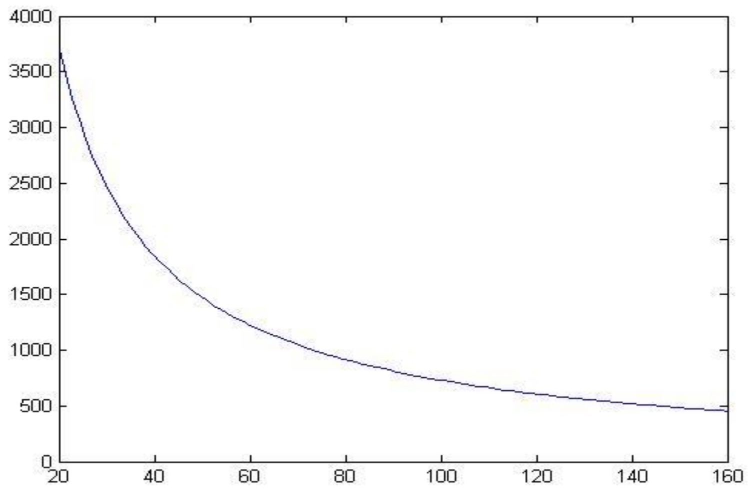


Figure 7 - Power Function

4.7 Pulse-Width Modulation (PWM) & Motor Control

4.7.1 GPIO Configuration

The GPIO pin PD 12 is enabled as it is associated with TIM4's channel 1 output. The operation mode is set to alternating function such that it uses the timer. The alternating configuration is then set to TIM4 and the chosen pin 12.

The interrupts service routine channel is set to the TIM4 interrupt service routine. The pre-emption priority of this interrupt is set to 0 as controlling the motor for the purpose of moving the proximity sensor is of the highest priority. The sub priority level is set to 1, by good practice, as if there was a case where there is another interrupt with the same pre-emption priority, the sub priority level shouldn't be the same in order to clearly distinguish pre-emption.

4.7.2 TIM & PWM Configuration

TIM4 was selected as the timer of choice to generate the PWM that is used by the motor. It was chosen as it is a full featured timer that supports PWM signal generation. Other timers provided by the board would have also suited our needs, as long as they support PWM signal generation. The prescaler and period values are chosen such that we get a 20ms pulse cycle and the pulse is set to 1500µs as specified in the datasheet.

4.7.3 Motor Implementation

Once the timer goes into interrupt service routine, the set compare function on the timer is called and sets the pulse to be the sum of 1500µs with the angle multiplied by 10. In order to have 180° coverage, the minimum pulse is 600µs and the upper boundary is 2400µs. This corresponds to angle values between -90° and 90°. The module that implements the motor, there is a function called setMotorAngle(int angle) that allows the angle for the motor to be set as it sets a global variable inside the motor file. That set angle is used in the calculation of the pulse during the interrupt service routine of the timer. Tests and Observations

4.7.4 Servo Motor Testing

In order to test servo motor, started by setting the timer parameters to the specified ones on the datasheet. We then modified those values in order to ensure to find the values that cover the 180° the most accurately. The Table 10 shows the different combination of parameters and the covered range.

Neutral Pulse	Multiplier	Range(°)
1450	10	87 to 88
1500	10	-90 to 90
1550	10	-86 to 87
1500	9.5	-86 to 86
1500	10.3	-92 to 92
1500	10.5	-92 to 93

Table 10 - Pulse Width Range Tests

Based on our tests, the datasheet defined values seem too correlated with the range we are looking for. The multiplier corresponds to the value used to augment the angle values such that they are converted to µs values for the purpose of calculating the pulse as specified by Equation 2 in the theory section for the PWM. Increasing the multiplier would “enlarge” each degree and that would result in either the trying to turn past its physical maximum and each software degree corresponding to more than a single physical degree. Changing the neutral pulse shifts the range towards the left or right.

4.8 Wireless Communication

In order to make sure that our wireless communication was robust, we chose to determine the percentage of packet loss that occurred during transmission between two stations. This metric is particularly important in our case because we have an event based system where an event is ideally signaled only once and assumed to be correctly received at the other end.

To test this metric, we programmed a station to send a packet every second, and another one to receive and display all the packets received. We found that a lot of packet losses occurred due to CRC errors. However, after enabling the forward error correction (FEC) feature of the CC2500 modem, these packet losses were greatly attenuated. The results are shown in Table 11 - Packet loss test results.

Table 11 - Packet loss test results.

Modem Mode	**Packet Loss (%)**	**Comment**
No FEC	32.6	Unacceptable packet loss percentage without FEC. This bad result may be explained by a noisy environment during the test.
FEC	1.7	Great improvement achieved with FEC. Packet loss is now a non-issue if using the packet burst capability of the wireless module.

5 Conclusion

It was possible to correctly detect a patient using our detection platform. Indeed, using wireless boards and their corresponding RSSI, the entrance of a patient in the forbidden zone was detected. Its position was then tracked using the servo-mounted proximity sensor. The range of the servo motor, as tested, was 180° and the maximum detection range of the sensor was 150 centimeters. The packet retrieval versus packet loss ratio for the wireless communications was 98.3%, which proves that our communication system is reliable.

During this project, we acquired knowledge on driver design, detection algorithms and wireless communication.

6 Bibliography

STMicroelectronics. (2012). UM1472 User Manual - STM32F4DISCOVERY - STM32F4 high-performance discovery board.

STMicroelectronics. (2014). STM32F427xx STM32F429xx Datasheet - production data.

Texas Instruments. (2014). CC2500 - Low-Cost Low-Power 2.4 GHz RF Transceiver. Dallas, Texas, United States.