ECSE 426-FINAL REPORT
3D PRINTING MACHINE

Group 6

Simon Ho -260479710

Sidney Ng -260507001

Chi-Wing Sit - 260482136

Meng Yin Tao -260480207

Kaichen Wang -260480833

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This experiment involves the development of a 3D mechanical printer system which is controlled by a pair of STM Discovery F4 microcontroller boards, as well as physical moving parts. The desired final product of the experiment includes two main subsystems. The first subsystem essentially comprises the user interface which allows a user to determine and control what the machine will print next, as well as to enable printing itself. This subsystem relies on the STM32F429 board, which includes a built-in LCD screen and to which an alphanumeric keypad is connected. On the output end of the machine, another subsystem uses the STM32F4 board to which three HiTech HS422 stepper motors are connected. These motors are in turn attached to a set of custom designed parts which can hold a pen, pencil or marker. To communicate between the two subsystems, wireless communication is set up using two TI MSP430-CC2500 wireless chipsets. Once the system is fully operational, the user can use the keypad buttons to switch between two different drawing modes; one for drawing basic predetermined shapes and the other for a sequence of user-determined lines. The user's inputs are displayed on the LCD, and once confirmed, control signals are sent wirelessly from one board to the other. The mechanical arms connected to the motors will then move in a 2D plane, as well as up and down, to draw the corresponding shapes.

PROBLEM STATEMENT

This experiment can be divided into several main components: the LCD display user interface, the motor configurations, the shape drawing algorithms, and the wireless communication. Many challenges can arise in the development of the individual components, and it is imperative to understand the theory behind each of the features to be implemented. As well, more problems can arise when the entire system is integrated as a whole and the components are put together. In order for the system to transmit data wirelessly between the two boards, a reliable way to communicate using the CC2500 transmitters is required. The challenge in this part of the experiment is in the writing of the drivers for the wireless chipsets. A large part of the driver's functionality is to enable the SPI protocol (as described in the Theory section) between the board and the wireless chip. Following the SPI, the actual wireless transmitter will need to be configured and the main challenge is to properly understanding the settings and the purpose of each. Assuming the SPI protocol is tested and functioning, the wireless transmission can still present many potential problems. Timing is very important when performing a transmission of data wirelessly. The LCD was also a challenge in itself, as a new technology that had not been used during the previous labs . A full understanding of the LCD board and drivers was a prerequisite to design a complete user interface. Finally, the shape drawing required to translate the rotational movement of the robot arms to a linear movement on a Cartesian plan for the pen.

THEORY AND HYPOTHESIS

HYPOTHESIS

The expected outcome of the experiment is a fully functional system that works according to the specifications of the experiment. If the human resources are managed properly and priority is given to the wireless transmission and the motor configurations, there will be enough time for the two separate components to be fully tested according to specifications. It is expected that the wireless transmission will be challenging to implement, but since the documentation available is quite comprehensive and the drivers available for the MEMS sensor of a previous experiment is similar in nature, a successful outcome is expected. For the motors and the shape-drawing algorithms, the trigonometry involved is straightforward, but it is expected that physically calibrating the moving parts will take a significant amount of time. It is also expected that the drawing/printing mechanism will be successful, but the accuracy of the drawing medium may vary given the physical limitations of the mechanical parts. Overall, as long as the theory detailed below is well understood, the individual components and their integration should be successful.

SPI

In order for the subsystems to communicate wirelessly, the microcontroller boards must first be able to communicate with their respective wireless transmitters. The communication protocol used to transmit data between the CC2500 wireless chip's registers and the board's processor is SPI. Once SPI compatible pins on the transmitter and the corresponding pins on the board are connected together, SPI can be enabled by setting one device as a master and the other as a slave. In this case, the board is set up as the master, while the CC2500 acts as slave. As seen in Figure 1 [2], the master (MCU) has three output signals: the clock (SCLK), the master-out (MOSI) and the chip select (CSn), as well as one input signal: master-in (MISO). On the other end, the slave (CC2500) device has a clock input (SCLK), slave-in input (SI), chip select input (CSn) and a slave-out output (SO).

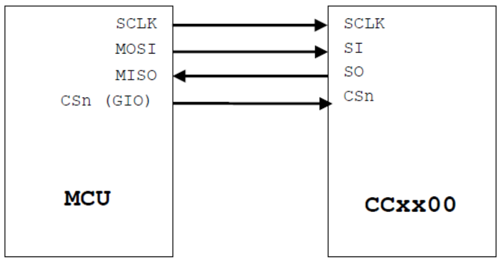


Figure 1-Four-wire SPI interface between master (left) and slave

The clock determines the frequency of the transmission since the period determines the amount of bits transferred per unit of time. In SPI, the master can initialize a transmission (read or write) of data contained in the slave's registers by sending and receiving bytes; a byte must be sent and a byte will be received for each transmission regardless of whether the desired action is a read or a write. Along with the data that is sent and received, SPI also requires an address of the desired source or destination register on the transmitter. This address needs to be in a specific format as seen in Figure 2 [2], where the MSB toggles between read (1) and write (0) mode, while the 2nd MSB determines whether only one register is accessed or many sequential registers are to be accessed in burst mode; bits 0-5 represents the actual register address value.

$R/\overline{W}$	B	A_5	A_4	A_3	A_2	A_1	A_0

Figure 2-Address header format for SPI

Once the pins for SPI are configured and the proper address header format is obtained, the board can begin communicating with the wireless transmitter's registers. Firstly, the address header is sent along the MOSI to the slave. In the case of a read (from a source register on the wireless chip) SPI command, a dummy byte can be sent and the data contained in the register corresponding to the address of the header will be returned by the SPI interface through the SO port. As for a write, after the address header has been sent, the data that needs to be written in the transmitter's register is sent via the MOSI pin; the returned value from the SO pin can be disregarded given the command is a write only.

To ensure proper functionality, SPI communication also requires proper timing between the signals sent through the four wires connecting the board to the wireless chip. Before the SPI transmission can begin, the CSn signal is first pulled low. Then, after waiting for the MISO signal to go low as well, the address header can be sent. After that, the actual transmission of data to the wireless chip can begin, with each byte sent also returning another byte. In the end, when the SPI transmission is complete, the CSn signal is pulled back up high.

CC2500 WIRELESS TRANSMITTER

Once the SPI protocol is configured and ready to read and write data to the wireless transmitter, the chipset itself can now be configured. The chip's settings which will determine the characteristics with which the pair of wireless transmitters will communicate with each other. As seen in Appendix A [3], these configurations cover a wide range of aspects relating to the hardware of the transmitter. Most of the settings can be configured using the default values or values provided in the CC2500 settings file, but in order for the two transmitters to communicate with each other while minimizing possible interference from other transmitters, the channel must be configured accordingly; the channel number (CHANNR register) must be set to identical values on both the transmitter and receiver chip.

STROBE COMMANDS

The CC2500 chipset has an internal Finite State Machine (FSM) that controls the state and operations that the chipset performs. In order to send instructions and change the state of the FSM, a strobe command can be sent via SPI. These SPI transmissions target specific strobe register addresses as shown in Table 1 [3], which lists five commonly-used strobe commands for wireless transmission and reception.

Table 1-Commonly used strobe commands for wireless transmission

Address	Strobe	Description
0x30	SRES	Reset chip.
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and MCSM0.FS_AUTOCAL=1.
0x35	STX	In IDLE state: Enable TX. Perform calibration first if MCSM0.FS_AUTOCAL=1. If in RX state and CCA is enabled: Only go to TX if channel is clear.
0x36	SIDLE	Exit RX / TX, turn off frequency synthesizer and exit Wake-On-Radio mode if applicable.
0x3D	SNOP	No operation. May be used to get access to the chip status byte.

The SRES command effectively resets the chip to its default state and this is performed when the chip is first configured and powered on. The SRX command enables the receive mode of the chip and enables data to be received from the wireless channel and placed into FIFO registers which can then in turn be read by the board through SPI. The STX command enables the chip's transmit mode and allows data written by the processor to the chip's FIFO registers to be broadcast wirelessly. Another command is the SIDLE strobe, which is used at the end of a wireless communication in order to allow both chips to exit the receive and transmit mode respectively. Finally, an important command that can be sent is the SNOP command. It acts as a dummy command in order to poll the FSM for a status byte which will indicate which state the transmitter is currently in.

WIRELESS TRANSMISSION

The process of sending data from the CC2500 chipset is straightforward. First write data to the transmit FIFO buffer via SPI. Then switch the state of the finite state machine to TX. Finally, wait until the state returns to IDLE to ensure that the packet has been sent. The process of receiving data is reversed. Set the finite state machine to RX and wait for an interrupt to indicate that data has arrived. Then wait for the state of the FSM to return to IDLE to indicate that the entire packet has been received. Then read data from the RX FIFO buffer via SPI.

LIQUID-CRYSTAL DISPLAY (LCD)

The STM32F429 LCD display available to us gives a graphical interface to send the plotting/printing commands to the controller board. The STM32F429 example files for the LCD operation already provide drivers and graphical packs such as specific board definitions, LCD configuration and layer initialization functions [1]. It also provides multiple functions which allows to draw various shapes (circles, rectangles, triangles, lines, ellipses, polygons). Here is an overview of the functions provided by the LCD driver:

Table 2-LCD diver API

Function name	Description
LCD_Init();	Configures the LCD controller GPIO pins as output and configures SPI port connected to the LCD
LCD_LayerUnit()	Initialize settings of LCD layers (window sizes) and sets color palettes for the layers
LTDC_cmd(ENABLE)	Enables the LTDC Controller
LCD_SetLayer(LCD_FOREGROUND_LAYER)	Select which layer we are currently working with

The STM32F429 example codes are referred as a reference for the LCD main function template to draw shapes.

A 8-pin, 16-key display keypad is used to select a desired shape and the LCD display the chosen shape on the screen. The method of scanning for key presses is the standard method of driving the row pins low and determining the active column, and then driving the columns low and determining the active row.

SERVO MOTOR CONTROL

The angle of the servo motor is determined by the pulse-width modulation (PWM) that is applied to the control wire. The servo position is not defined by the PWM duty cycle but only by the width of the pulse. The servo expects to see a pulse every 20 ms, that pulse will define the position of the motor, the motor position can go from 0° to 180°. For example, a 1.5 ms pulse will set the motor to the 0 degree position. Figure 3 shows the pulse width required for different angles. As a software timer would be affected by the thread executions, the PWM was implemented with a hardware timer that runs independently from the software to allow a more robust control over the servo motors.

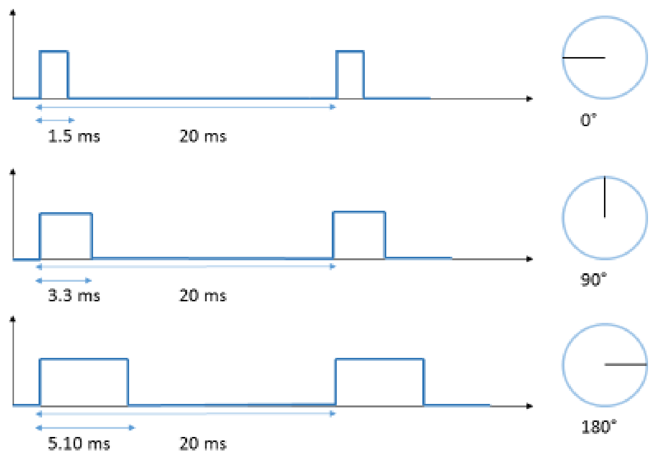


Figure 3-PWM of the servo motors

3D-DISPLACEMENT

The 3D mechanical printer system is composed of three HiTech HS422 stepper motors; two to control the xy-displacement of the moving parts and one to control the up and down. The first two motors are placed adjacent to each other and their center of rotation is shown in Figure 4 at location (-1.9, 0) and (1.9, 0) to control the moving arms. Each arm is composed of two segments, A and B. Segment A has length 7 cm and segment B has length 6.5 cm. A pen is attached to the end of segment B, connecting both arms. The position of the pen (x, y) is controlled by the angle of the motors (α+β) . Using geometry, one can solve for the motor angles given the desired (x, y) position.

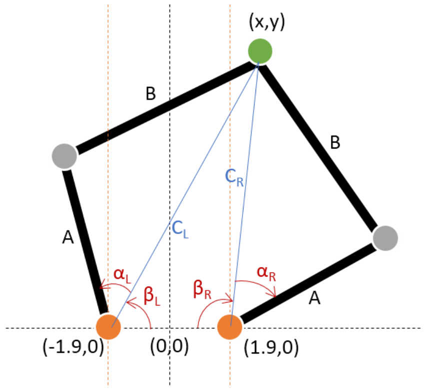


Figure 4-Moving Arms

From the figure above, one can see that the segments on each arm form two sides of a triangle, where the third side is simply the distance between the motor and (x, y). This distance C varies with the position of (x, y) and can be determined using *Pythagorean Theorem* by taking the positive root:

$$C_L = \sqrt{(x + 1.9)^2 + y_2} \quad \textbf{(1)}$$

$$C_R = \sqrt{(x - 1.9)^2 + y_2} \quad \textbf{(2)}$$

Knowing the side of a triangle, the *Cosine Law* can be used to find the angle between segment C and arm segment A (α). For instance, α_L can be solve:

$$\alpha_L = \arccos\left(\frac{B^2+C_L^2-A^2}{2BC_L}\right) \quad \textbf{(3)}$$

The same relationship applies to α_R:

$$\alpha_R = \arccos\left(\frac{B^2+C_R^2-A^2}{2BC_R}\right) \quad \textbf{(4)}$$

The angle between the x-axis and segment C (β) can be solved using the *arctan* function. Because of the special nature of the *arctan* function and the position of the motors, the computation for β must be done in five different cases depending on the x position of the pen.

- Case when $x > 1.9$:

$$\beta_L = \arctan\left(\frac{y}{x + 1.9}\right)$$

$$\beta_R = \pi - \arctan\left(\frac{y}{x - 1.9}\right)$$

- Case when $-1.9 < x < 1.9$:

$$\beta_L = \arctan\left(\frac{y}{x + 1.9}\right)$$

$$\beta_R = \arctan\left(\frac{y}{x - 1.9}\right)$$

- Case when $x < -1.9$:

$$\beta_L = \pi - \arctan\left(\frac{y}{-x - 1.9}\right)$$

$$\beta_R = \arctan\left(\frac{y}{1.9 - x}\right)$$

- Case when $x = 1.9$:

$$\beta_L = \pi - \arctan\left(\frac{y}{x + 1.9}\right)$$

$$\beta_R = \frac{\pi}{2}$$

- Case when $x = -1.9$:

$$\beta_L = \frac{\pi}{2}$$

$$\beta_R = \arctan\left(\frac{y}{-x + 1.9}\right)$$

4X4 KEYPAD

Users can interact with the system using a 4x4 keypad shown in Figure 5. All columns and rows need to be scanned in order to determine which key is pressed. To do so, each column pins and row pins are connect to the I/O ports as recorded in Appendix B. The column pins are initially set high. When a key is pressed, the corresponding row pin will be high as a result of a short between a column and a row. For instance, when a key on the first row is pressed, pin 5 will receive the signal. To determine the column on which the key is located, the rows are set high and a signal is expected on one of the column pins.

This implementation can only detect on key at any given time. If the user presses both key 3 and key 5, the keypad will return 2 since it scans the rows top-down and the columns from left to right. This limitation requires uses to press only one key at any time. Key debouncing issues are handled by storing the historical inputs. When two identical inputs are detected, the later one will be ignored until the input changes or no input is detected during a period of time.

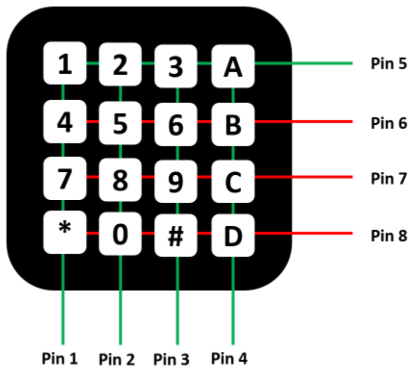


Figure 5-4x4 Keypad

IMPLEMENTATION

USER INPUT INTERPRETATION

The 3D printing machine allows the user to be in either the "game" mode or the "drawing" mode. The game mode allows the user to play the tic-tac-toe game against an AI, while the drawing mode lets the user draw a set of predefined shapes using the keypad. The different actions taken by the printing machine depending on the keypad input are defined in Table 3 and Figure 6.

Table 3-Keypad input mapping

Keypad Button	Output in Drawing Mode	Output in Game Mode
A	Square	---
B	Rectangle	---
C	Triangle	---
*****	Remain in Drawing Mode	Change to Drawing Mode
#	Change to Game Mode	Restart Game
0	---	---
1	Upper Left	Square 1
2	Up	Square 2
3	Upper Right	Square 3
4	Left	Square 4
5	Reset	Square 5
6	Right	Square 6
7	Lower Left	Square 7
8	Down	Square 8
9	Lower Right	Square 9

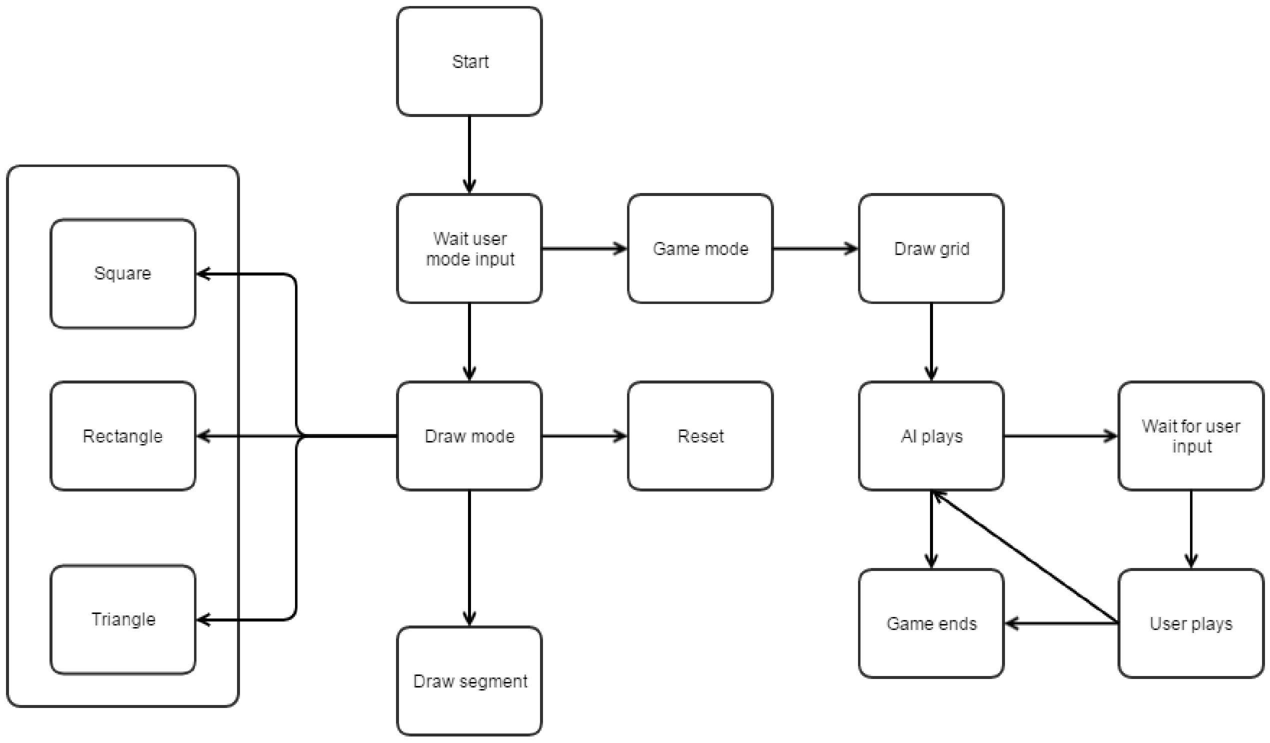


Figure 6-User input flow

LCD

On startup, the controller board is given to the user with the keypad connected to it. There are two modes of operation. The first mode is "on the fly drawing mode" where the user can draw predefined shapes (square, rectangle, triangle) or segments with angle and direction, each option mapped to a button. In order to obtain the right mode of operation, the shape, direction and mode are determined from the keyboard user input and the corresponding thread is called.

At the beginning of the mode, the motors move the drawing pen to coordinate (215,30). As we draw the shape on the fly, the current selection not yet sent will be blinking while the other ones already sent will be shown in solid color.2 threads for each shape are implemented to create the blinking effect. The function first draws the shape and then a simple osDelay(50) is used to get a continuous flashing animation.

SPI

The SPI implementation can be split into three major parts: configuration, read, and write.

The first is the configuration step, in which the SPI settings are set up. In terms of configuring the hardware, the wireless chip and the board are connected at specific pins using wires. On the wireless transmitter side, the SCLK input is located on pin 16, the SI input is on pin 18, the SO output is on pin 15, and the Chip select CSn input is on pin 17 [4]. As for microprocessor boards, the pin configurations that are compatible with the SPI protocol differ slightly between the STM32F429 LCD board and the STM32F4 board. For the STM32F429 board, the GPIO ports used are A for the SCLK (pin 2), MOSI (pin 6), and MISO (pin 5), and port C for the chip select CSn (pin 13); SPI 2 was selected to operate through those pins. As for the STM32F4 board, the GPIO port used is B for the SCLK (pin 13), MOSI (pin 15), MISO (pin 14), and chip select CSn (pin 12); SPI 4 was selected to operate through those pins. As well, to enable the external wireless device to communicate with the main board using SPI, the EXTI external interrupt lines are used and are connected to the boards NVIC peripheral in order to handle the interrupts raised by incoming data. The EXTI line used is line number 5 for the LCD board, and line number 14 for the regular STM32F4.

After completing the settings for the transmitter pins and board GPIO, the settings of the actual SPI protocol are configured as follows:

- Direction (SPI_Direction) is set to 2 lines with full duplex, because data needs to be sent in both directions
- Data size (SPI_DataSize) is set to 8 bits, because the data is expected to be sent in bytes
- Clock polarity (SPI_CPOL) is set to low, because SCLK is low in idle state.
- Clock phase (SPI_CPHA) to fist edge, because data should be centered on first positive edge of period
- Chip select/NSS signal (SPI_NSS) is set to soft, because there is only one slave device
- Transfer baud rate (SPI_BaudRatePrescaler) is set to have a prescaler value of 16
- First bit (SPI_FirstBit) is set to be the data's MSB
- Cyclic redundancy checking polynomial (SPI_CRCPolynomial) is set to 7, to ensure correct data
- SPI mode (SPI_Mode) is set to master mode, because the MCU (board) acts as the master

The second part of the SPI implementation is the read function. Both the SPI read and write involves sending a byte to the wireless transmitter and therefore both functions use SPI_I2S_SendData to send a byte. The read function first prepares the address header and ensures that the read/write byte is set to 1 and that the burst bit is set according to the number to bytes to read. After those two bits are concatenated to the source address and the address header is sent via the SPI_I2S_SendData function, which must wait until the data registers are empty before proceeding (a SPI_I2S_FLAG_TXE flag is checked using the SPI_I2S_GetFlagStatus function). After that, a dummy byte (0x00) is transmitted for each read and each corresponding return value is the value being read via SPI from the source register.

The third and final section of the SPI implementation is related to the write function. The is implemented almost identically to the read, except that instead of sending a dummy 0x00 byte, the actual data what needs to be written is sent and then transmitted via SPI to the destination register.

WIRELESS TRANSMISSION

To implement the wireless transmission, the wireless transmitter had to first be configured by using SPI to set the configuration registers on the chip. Most of the configurations were either left at their default values or set to the values from the CC2500 settings file. However, the wireless chip's channel was set to an arbitrary value chosen to be eight times the group number 6 to give channel 48 (0x30). After that, the actual wireless transmit and receive were implemented. The transmit first changes the state of the transmitter using the CC2500_Strobe function to SIDLE. The program waits for the state to stabilize in idle and then switches to either receive (RX) mode or transmit (TX) mode. For the transmit, the FIFO buffer in which the data will be placed is checked for potential overflow and proceeds to the transmission of data. For the receive end, no values are read until its FIFO buffer has been filled with at least one byte of data and proceeds.

For the purpose of the experiment, the source of the data being transmitted are the alphanumeric keypad presses. The transmitting CC2500 chip and the keypad are both connected to the LCD board and when a key is pressed by the user, the transmit function, which is waiting for an OS signal to proceed, can then continue. On the other board, the wireless chip is essentially continuously checking for incoming wireless data. Once the data has been received and verified to be a legal keypad value, the data can then be used as a command to activate specific features that can be carried out by the motors powered by the board.

ACCELEROMETER AND KALMAN FILTER

The raw data acquired from the accelerometer cannot be used directly. It is noisy and also most likely misaligned with the actual axes of the board. As a result, normalization and filtering must be performed on the data. For this application, we required the use of three different instances of a Kalman filter state, one for each of the accelerometer values. Using only one Kalman filter would have resulted in incorrect final calculations. Each individual input would have advanced the iteration when all three accelerometer values should be in the same iteration for different states. The filtered values are then passed to the tilt angle calculation.

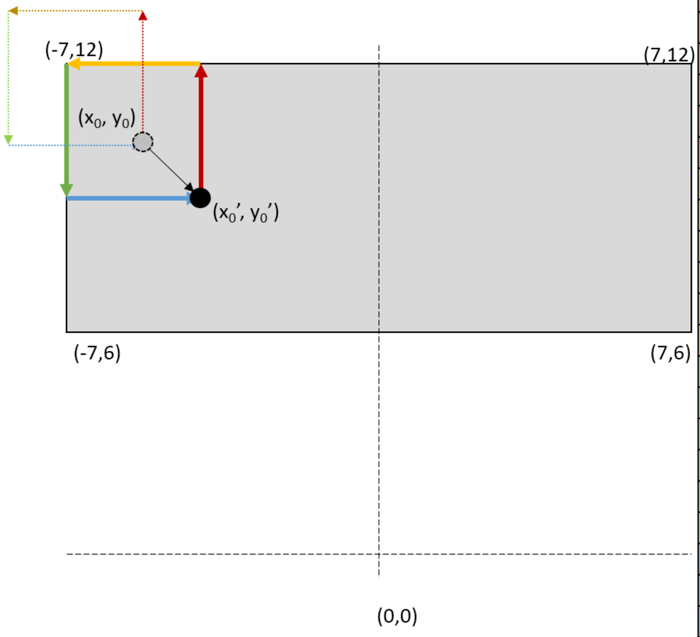
For each axis, the measurement noise covariance r was found by calculating the data variance. q was found empirically by comparing the general trend of the filtered data with different q. Finally, since the initial value of p, and k would not really affect the performance of the filter as they would rapidly converge to a certain value, we decided to set them to 1 and 0 respectively.

DRAWING MODE

In the “On the Fly Drawing Mode”, users can select either predefined shapes or 1-cm long segments in a direction of their choice. To draw a predefined shape or a segment, an array of (x,y) points are stored in a path. Depending on the starting point chosen by the user, the path array is adjusted. The motors update their angles by reading an element from the array every 50ms. In order to assure drawing straight lines, the interval between points are as small as 0.5 cm. The path array is shared by all the shapes, therefore the array size is fixed and has a length of 50 elements. The end of a path is marked by repeating the last coordinate.

Using the accelerometer, the user can also set the starting point (x_0, y_0) of the pen. A pitch value higher than 30° will increase the x-coordinate of the starting point, on the other hand a pitch angle lower than 30° will decrease the x-coordinate. Similarly the roll angle controls the y-coordinate of the pen and will control the vertical position of the pen.

The drawing area reachable by the pen is 14 cm wide by 6 cm tall. When user sets a starting point close to the edge of the drawing area, the system will move the starting point inward to ensure that the shape can be properly drawn. An example is illustrated in Figure 7. It is worth noticing that the last element of the array is repeated. This is used to notice the motors when they have reached the end of the path and the rest of the array can be ignore.



X_path[i]	Y_path[i]
x_0'	y_0'
x_0'	$y_0'+0.5$
x_0'	$y_0'+1$
x_0'	$y_0'+1.5$
x_0'	$y_0'+2$
x_0'	$y_0'+2.5$
x_0'	$y_0'+3$
$x_0'-0.5$	$y_0'+3$
$x_0'-1$	$y_0'+3$
$x_0'-1.5$	$y_0'+3$
$x_0'-2$	$y_0'+3$
$x_0'-2.5$	$y_0'+3$
$x_0'-3$	$y_0'+3$
$x_0'-3$	$y_0'+2.5$
$x_0'-3$	$y_0'+2$
$x_0'-3$	$y_0'+1.5$
$x_0'-3$	$y_0'+1$
$x_0'-3$	$y_0'+0.5$
$x_0'-3$	y_0'
$x_0'-2.5$	y_0'
$x_0'-2$	y_0'
$x_0'-1.5$	y_0'
$x_0'-1$	y_0'
$x_0'-0.5$	y_0'
x_0'	y_0'
x_0'	y_0'

Figure 7-Draw Square

GAME MODE

As special features, a tic-tac-toe game as well as an AI have been implemented into our 3D printing machine. This mode can be entered at any time by pressing the "#" key. Figure 8 shows the mapping between the each square of the grid to its numerical value on the keypad.

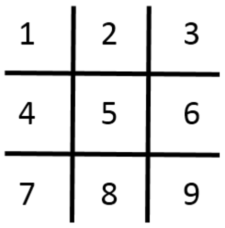


Figure 8-Tic-Tac-Toe key mapping

The AI has been designed to either win or tie the game. Before playing, the AI gets the current status of the board, the board is represented in the code as 1D array where the index corresponds to the previously defined key mapping. The AI first tries to play in a spot where he can win, if none exists, the AI will try to stop the player from winning. If the AI can neither win nor stop the player from winning, an array of the sequence of moves the AI should follow has been predefined, the AI will play in the first spot in the sequence that is available. Figure 9 illustrates the AI's logic flow.

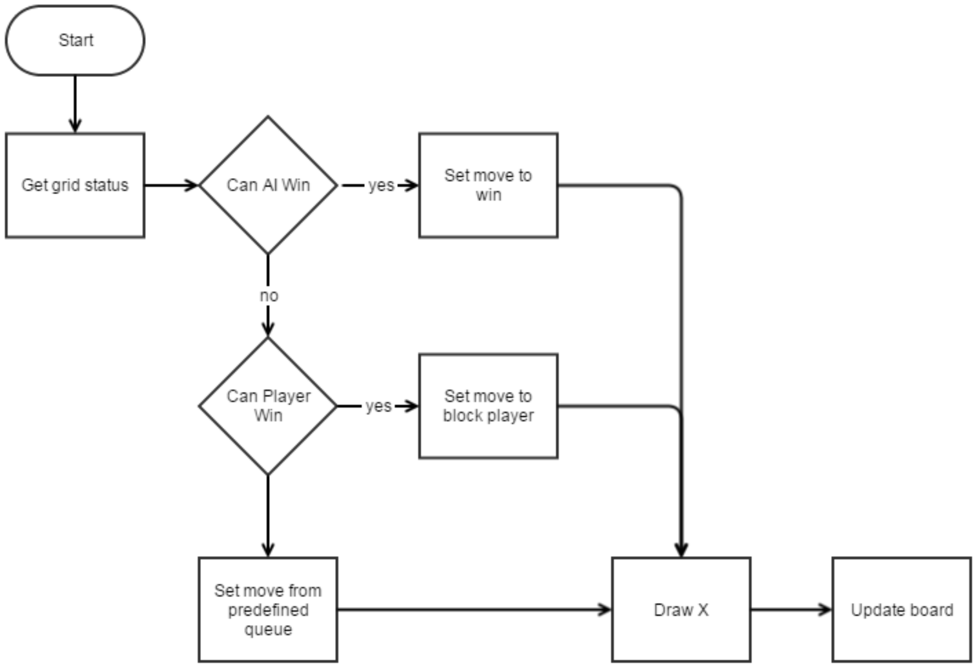


Figure 9-AI flowchart

TESTING AND OBSERVATIONS

SPI

The SPI protocol was tested by connecting the wireless chipset and continuously reading and writing values to the registers of the transmitter. A loop was used to automate the testing process and, at first, only one data value was written, read back and printed to the screen for each iteration of the loop. Following that, burst read and writes to registers were tested by creating test vectors. Since the data being transmitted was an unsigned 8-bit integer (uint8_t), there weren't many particular values to test. Instead, a large quantity of arbitrary data values were used as test data. SPI reads and writes were tested on the entire range of the 47 configuration registers (address 0 to 0x2E) which are compatible with both reads and writes.

STROBE COMMANDS

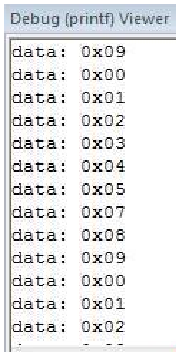
Strobe commands were confirmed to be working properly after the commands SIDLE, SRX, and STX were sent to their respective registers. After each strobe command was sent, enough time was given for the results to settle (no longer returning 100 or 110), the SNOP strobe allowed the current state of the wireless chipset's FSM to be returned and printed to the screen, which are listed in Table 1.

Table 4-State of CC2500 FSM after strobe command test

Strobe command (address)	State returned by Strobe command SNOP
SIDLE (0x36)	IDLE (000)
SRX (0x34)	RX (001)
STX (0x35)	TX (010)

WIRELESS TRANSMISSION

Testing of the wireless chip began with receiving data from the beacon which transmitted numeric codes zero to nine. As seen in Figure 10, the results obtained from the receive function yielded the correct data in the correct sequence.


Debug (printf) Viewer

```
data: 0x09
data: 0x00
data: 0x01
data: 0x02
data: 0x03
data: 0x04
data: 0x05
data: 0x07
data: 0x08
data: 0x09
data: 0x00
data: 0x01
data: 0x02
```

Figure 10-Beacon values received and printed for test

Then, the second step was to test both the transmit and receive capabilities of the system. Loops were created to continuously transmit bytes from the chip connected to the LCD board and the values were confirmed on the receiving end when they were printed onto the screen by the STM32F4 board. The main goal of the tests were to visualize any packet loss issues suffered. The test results concluded that packet loss can be reduced to about 5% of data when the two wireless chips are placed within a reasonable range of each other. As the distance between the two chips increases, so does the percentage of dropped packets. Through this empirical test, it was determined that the chips should be maintained within 30 cm to each other to minimize packet loss, which is perfectly reasonable for the purpose of this experiment.

Finally, once the other components of the experiment were implemented, including the alphanumeric keypad and its OS threads as well as the motors and the shape-drawing algorithms, final wireless tests were conducted. The test is fairly straightforward and results are observed empirically as well. For this integration test, the user presses a key and the data is sent wirelessly from one board to another and if the expected printing action is conducted, the transmission is considered a success. After multiple instances of this test, the wireless communication was determined to be a success for the purposes of this experiment.

KALMAN FILTER

The Kalman filter initial state parameters we chose were q = 0.025, r = 5, x = k = p = 0. x is the estimated value, and since each acceleration value will be in the range of 0 to 1, we set x to 0. To get the other values of the parameters, we plotted a comparison of filtered values with the raw data with varying parameter values. We held r constant at 5 and increased q from 0.0025 to 2500. Just by comparing visually, as the value of q rose, the signal became noisier. The optimal value we found at q = 0.025 and r = 5. The figures show the comparison for only the X values, but the filters for the Y and Z values gave the same result. Figure 11 illustrates the sampled data we measured initially as well as the filtered data in blue.

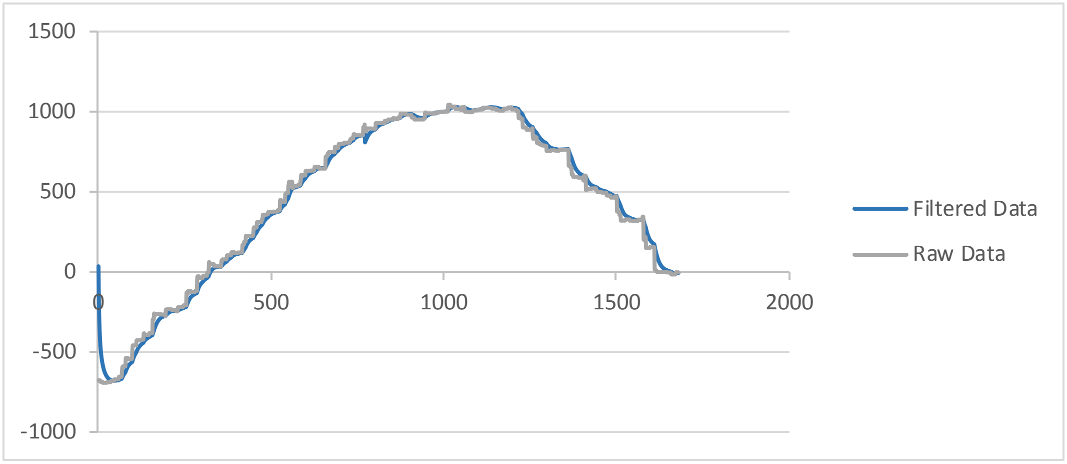


Figure 11-Raw and filtered data for the x-axis

VISUAL FEEDBACK

To test the system, we tried all the operational cases. By using the keypad directly on the RTX board, we could disregard some of the issues that the wireless might cause like packet losses. First, we tested the drawing mode by drawing the different shapes and segment direction at different starting point. We realized that the drawings were getting distorted as we approached the limits of the drawing area. Then, the game mode was tested by verifying all the possibilities where the AI wins or ties the game. The logic of the game was flawless but the symbols drawn on the last row were again distorted.

THREAD EXECUTION

When testing that the system works, we used the OS support provided by the debugging mode of the Keil IDE. From looking at the event viewer, a snapshot is given in Appendix C, we can tell that the system is performing as it should. The execution is dominated by the idle time, so the threads are changing states and waiting on the appropriate signals instead of blocking execution. Additionally, the threads are interleaving and no two threads are running at the same time. Looking at the System and Thread viewer, Appendix C, the thread configurations does not cause any errors. The stack load is not overflowing.

CONCLUSION

The final result of the experiment resulted in a system which met the basic requirement of sending instructions from one board to another board which will draw shapes. The system features three drawing modes. One to draw predetermined shapes, one to draw a path, and one that plays tic-tac-toe. A user can successfully choose which mode to use and the modes operate as specified. However, some issues remained unresolved. The drawn lines are not perfectly straight. As a result, the shapes are not perfectly drawn as well. In tic-tac-toe mode, the X's and the O's are misshapen on the last row. Additionally, the wireless communication between the boards is imperfect and if the sending frequency is too high, packets may be dropped. Implementing a handshake protocol between the boards would have solved this issue. Despite these issues, the system performs its basic functions. If the system functionality is extended by adding a camera at the motor board end, the system could function as a remote operating arm that can be used in situations deemed dangerous for humans.

REFERENCES

[1] A. Suyyagh, STM32F429 Discovery Board LCD Guide, Montreal, 2014, pp. 1-19

[2] S. Namtvedt, *Design Note DN503*, 1st ed. Dallas: Texas Instrument, 2007.

[3] *CC2500 Low-Cost Low-Power 2.4 GHz RF Transceiver*, 1st ed. Dallas: Texas Instrument, 2011.

[4] *eZ430-RF2500 Development Tool User's Guide*, 1st ed. Dallas: Texas Instrument, 2011.

APPENDIX

APPENDIX A

WIRELESS CONFIGURATION

Address	Register Name	Description
0x00	IOCFG2	GDO2 output pin configuration
0x01	IOCFG1	GDO1 output pin configuration
0x02	IOCFG0	GDO0 output pin configuration
0x03	FIFOTHR	RX FIFO and TX FIFO thresholds
0x04	SYNC1	Sync word, high byte
0x05	SYNC0	Sync word, low byte
0x06	PKTLEN	Packet length
0x07	PKTCTRL1	Packet automation control
0x08	PKTCTRL0	Packet automation control
0x09	ADDR	Device address
0x0A	CHANNR	Channel number
0x0B	FSCTRL1	Frequency synthesizer control
0x0C	FSCTRL0	Frequency synthesizer control
0x0D	FREQ2	Frequency control word, high byte
0x0E	FREQ1	Frequency control word, middle byte
0x0F	FREQ0	Frequency control word, low byte
0x10	MDMCFG4	Modem configuration
0x11	MDMCFG3	Modem configuration
0x12	MDMCFG2	Modem configuration
0x13	MDMCFG1	Modem configuration
0x14	MDMCFG0	Modem configuration
0x15	DEVIATN	Modem deviation setting
0x16	MCSM2	Main Radio Control State Machine configuration
0x17	MCSM1	Main Radio Control State Machine configuration
0x18	MCSM0	Main Radio Control State Machine configuration
0x19	FOCCFG	Frequency Offset Compensation configuration
0x1A	BSCFG	Bit Synchronization configuration
0x1B	AGCTRL2	AGC control
0x1C	AGCTRL1	AGC control
0x1D	AGCTRL0	AGC control
0x1E	WOREVT1	High byte Event 0 timeout

0x1F	WOREVT0	Low byte Event 0 timeout
0x20	WORCTRL	Wake On Radio control
0x21	FREND1	Front end RX configuration
0x22	FREND0	Front end TX configuration
0x23	FSCAL3	Frequency synthesizer calibration
0x24	FSCAL2	Frequency synthesizer calibration
0x25	FSCAL1	Frequency synthesizer calibration
0x26	FSCAL0	Frequency synthesizer calibration
0x27	RCCTRL1	RC oscillator configuration
0x28	RCCTRL0	RC oscillator configuration
0x29	FSTEST	Frequency synthesizer calibration control
0x2A	PTEST	Production test
0x2B	AGCTEST	AGC test
0x2C	TEST2	Various test settings
0x2D	TEST1	Various test settings
0x2E	TEST0	Various test settings

APPENDIX B

GPIO PORTS FOR WIRELESS

STM32F without LCD

Line	GPIO Pin
SPI2_NSS	GPIOB_12
SPI2_SCK	GPIOB_13
SPI2_MISO	GPIOB_14
SPI2_MOSI	GPIOB_15

STM32F with LCD

Line	GPIO Pin
SPI4_NSS	GPIOE_13
SPI4_SCK	GPIOC_2
SPI4_MISO	GPIOC_5
SPI4_MOSI	GPIOC_6

GPIO PORTS OF KEYPAD

STM32F with LCD

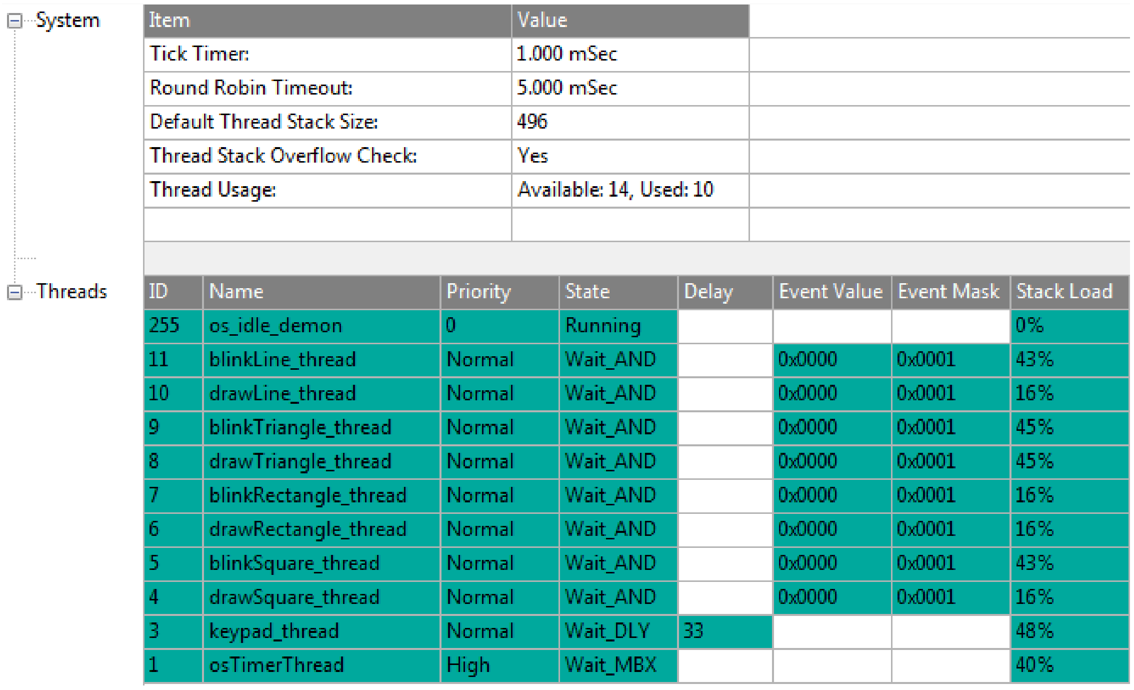
Keypad Pin Number	GPIO Pin
Pin 1 (column 1)	GPIOC_3
Pin 2 (column 2)	GPIOC_8
Pin 3 (column 3)	GPIOC_11
Pin 4 (column 4)	GPIOC_12
Pin 5 (row 1)	GPIOD_2
Pin 6 (row 2)	GPIOD_4
Pin 7 (row 3)	GPIOD_6
Pin 8 (row 4)	GPIOD_7

GPIO PORTS OF MOTORS

STM32F without LCD

Motor Number	GPIO Pin
Motor 0 (Left)	GPIOC_6
Motor 1 (Right)	GPIOC_7
Motor 2 (Up-Down)	GPIOC_8

APPENDIX C



System

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	496
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 14, Used: 10

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
11	blinkLine_thread	Normal	Wait_AND		0x0000	0x0001	43%
10	drawLine_thread	Normal	Wait_AND		0x0000	0x0001	16%
9	blinkTriangle_thread	Normal	Wait_AND		0x0000	0x0001	45%
8	drawTriangle_thread	Normal	Wait_AND		0x0000	0x0001	45%
7	blinkRectangle_thread	Normal	Wait_AND		0x0000	0x0001	16%
6	drawRectangle_thread	Normal	Wait_AND		0x0000	0x0001	16%
5	blinkSquare_thread	Normal	Wait_AND		0x0000	0x0001	43%
4	drawSquare_thread	Normal	Wait_AND		0x0000	0x0001	16%
3	keypad_thread	Normal	Wait_DLY	33			48%
1	osTimerThread	High	Wait_MBX				40%

Figure 12--System and thread viewer for LCD

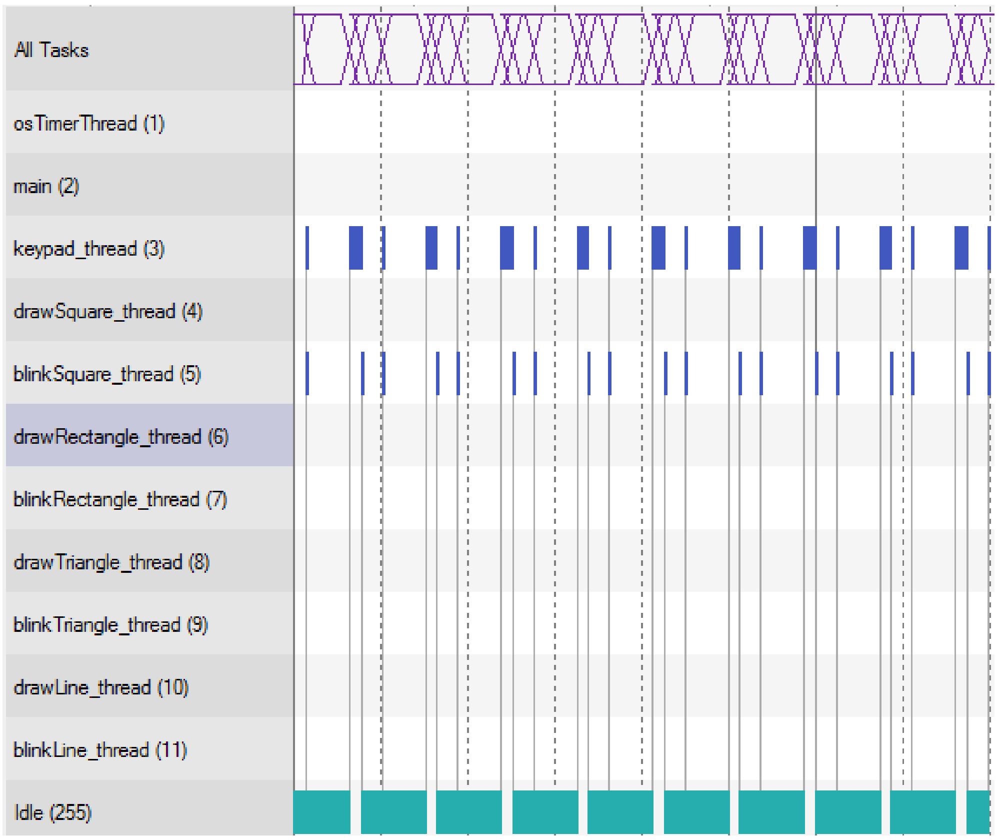
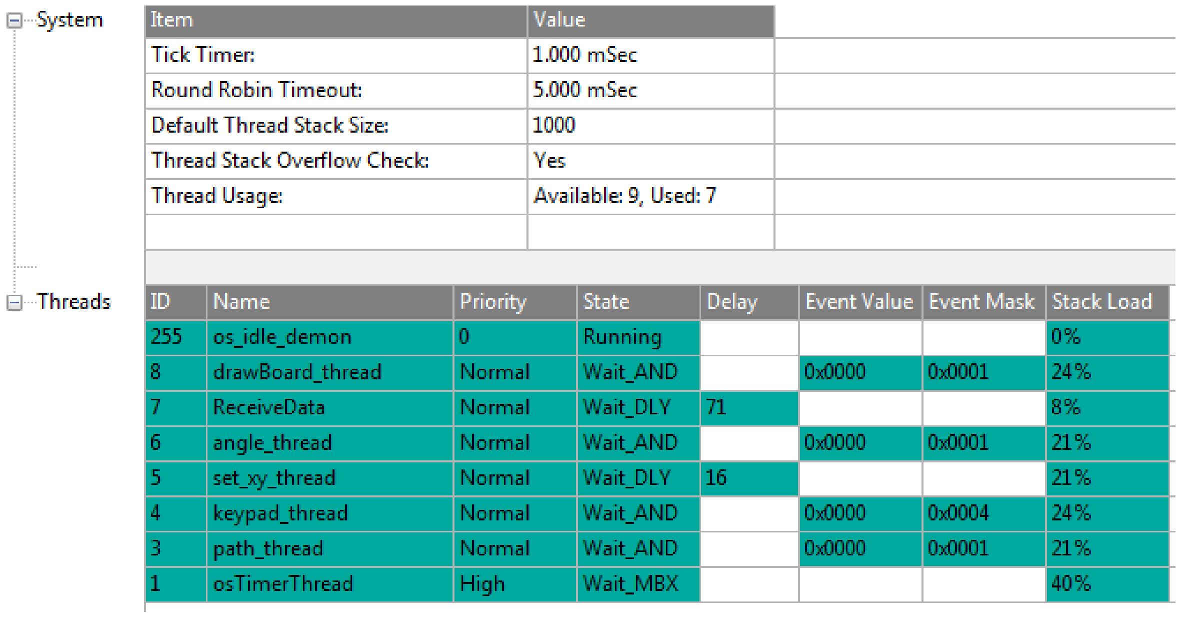


Figure 13-Keil OS Event Viewer debug window for LCD


System

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	1000
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 9, Used: 7

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
8	drawBoard_thread	Normal	Wait_AND		0x0000	0x0001	24%
7	ReceiveData	Normal	Wait_DLY	71			8%
6	angle_thread	Normal	Wait_AND		0x0000	0x0001	21%
5	set_xy_thread	Normal	Wait_DLY	16			21%
4	keypad_thread	Normal	Wait_AND		0x0000	0x0004	24%
3	path_thread	Normal	Wait_AND		0x0000	0x0001	21%
1	osTimerThread	High	Wait_MBX				40%

Figure 14-System and thread viewer for RTX

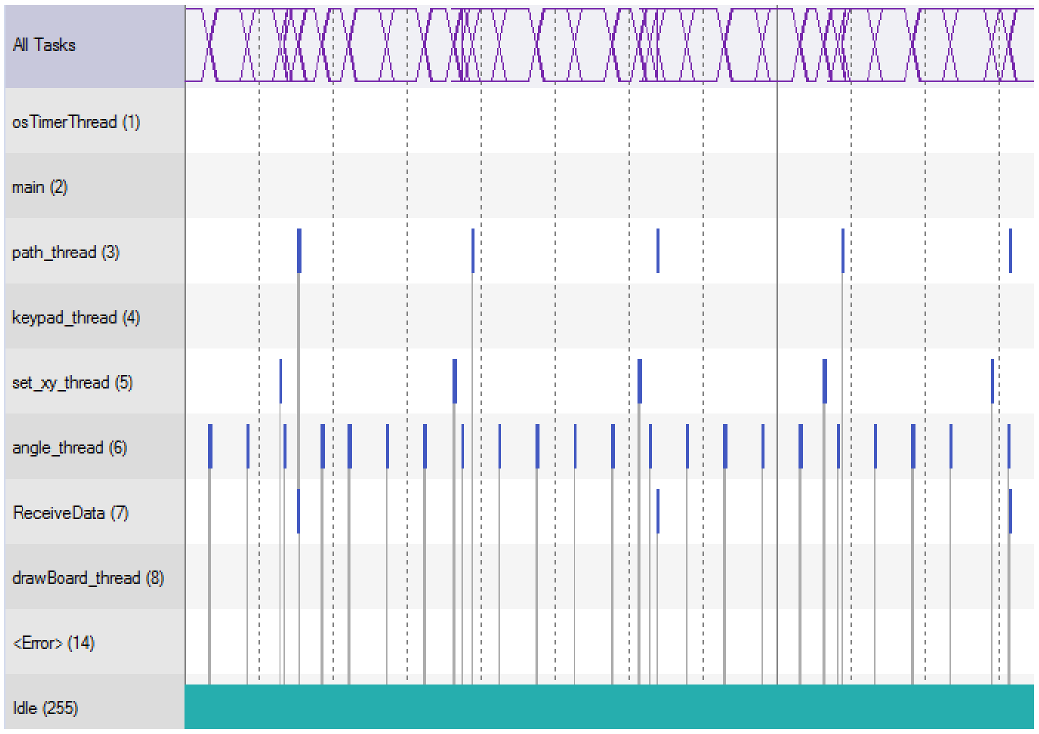
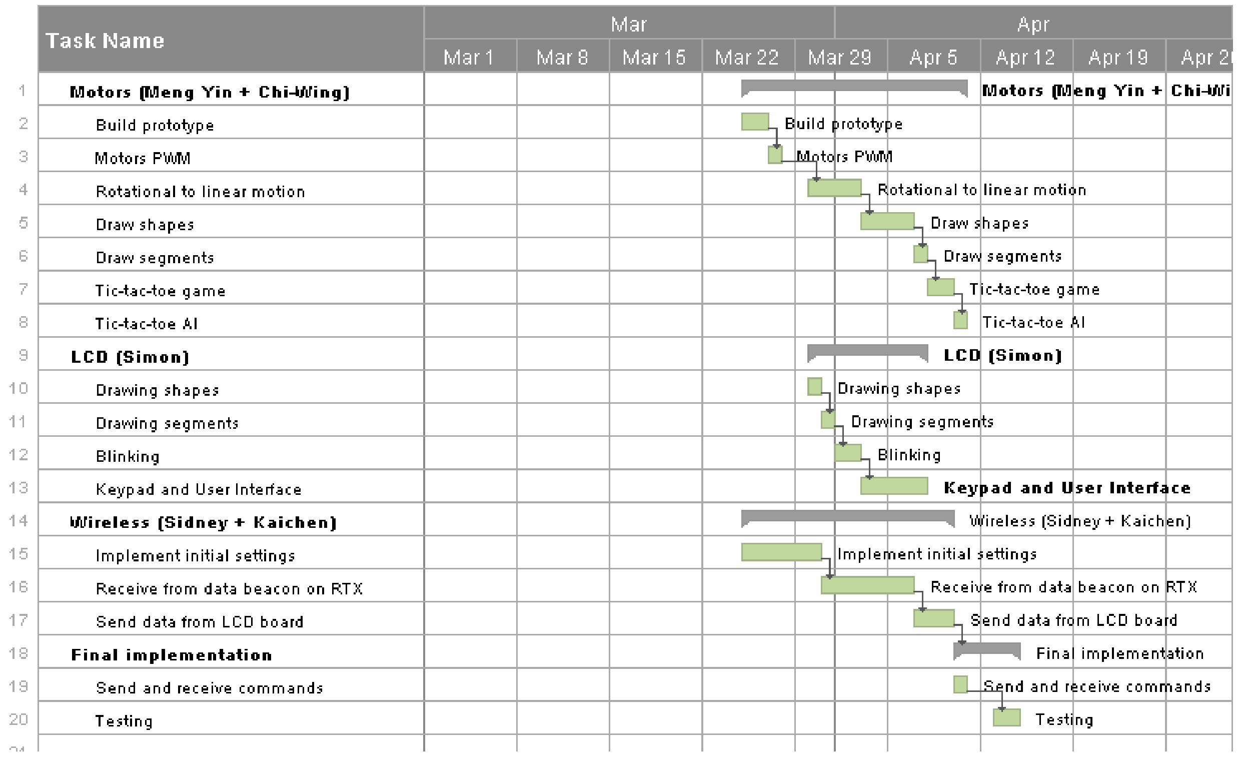


Figure 15-Keil OS Event Viewer debug window for RTX

APPENDIX D



	Task Name
1	**Motors (Meng Yin + Chi-Wing)**
2	Build prototype
3	Motors PWM
4	Rotational to linear motion
5	Draw shapes
6	Draw segments
7	Tic-tac-toe game
8	Tic-tac-toe AI
9	**LCD (Simon)**
10	Drawing shapes
11	Drawing segments
12	Blinking
13	Keypad and User Interface
14	**Wireless (Sidney + Kaichen)**
15	Implement initial settings
16	Receive from data beacon on RTX
17	Send data from LCD board
18	**Final implementation**
19	Send and receive commands
20	Testing

Figure 16-Gantt chart