CSUN | COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

ECE 528/L

Robotics and Embedded Systems with Lab

Final Project: Tilt/Gesture Controlled Robot

December 14th, 2025

Group 5
Sean Felker
Prachi Patel
Instructor: Aaron Nanas
Fall 2025

Table of Contents:

1 Introduction

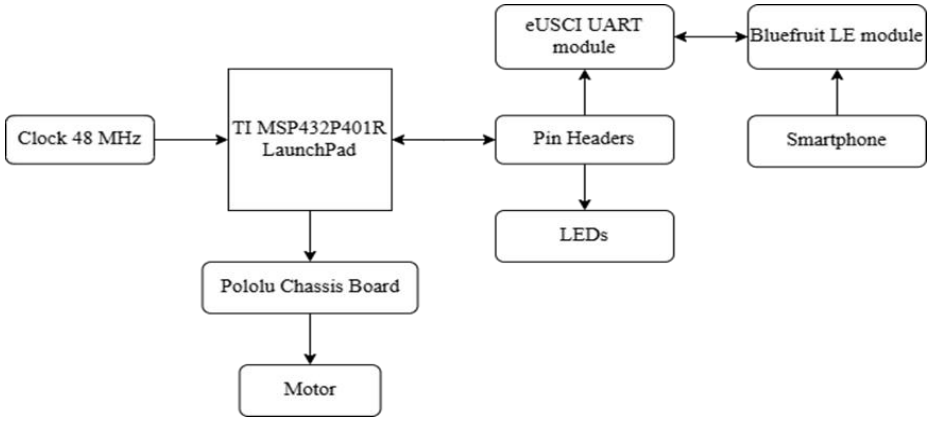
Advancement in human machine interaction technologies allow humans to communicate more effectively with robots and electrical devices. This project's main goal is to design a sensor-based hand gesture-controlled robot that responses the user's physical hand movement or tilting of a smartphone, which enables motions such as forward, backward, left, right, or diagonal without using traditional input devices such as remote or joysticks. The smartphone has built-in motion and orientation sensors that provide a quaternion reading through BluefruitConnect app. This quaternion sensor reading transmits to Adafruit Bluefruit LE UART module wirelessly and then the Bluefruit LE UART module will transmit that received reading to the MSP432P401R

microcontroller via UART serial communication. The MSP432P401R microcontroller drives the robot in the desired direction by adjusting PWM signals according to the data that is received. The main method of controlling the robot is when the user is holding and tilting their smartphone in a vertical/portrait orientation, but options for horizontal phone control mode and speed control mode have also been added.

2 Components

Description	Quantity	Manufacturer
MSP432 LaunchPad	1	Texas Instruments
USB-A to Micro-USB Cable	1	N/A
TI-RSLK MAX Chassis	1	Pololu
Adafruit BLE UART Module	1	Adafruit
Jumper Wires	6	N/A
iOS Smartphone	1	Foxconn, Pegatron
3.3 V DC Power Supply	1	Texas Instruments

3 Block Diagram



4 Pinouts

❖ **Pins for Motors and RGB LED:**

Pin Label	Connection	Purpose
P2.0, P2.1, P2.2	Direct to MSP432 LaunchPad	RGB LED to indicate command status
P2.6	Direct to MSP432 LaunchPad	Right motor PWM
P2.7	Direct to MSP432 LaunchPad	Left motor PWM
P3.6	Direct to MSP432 LaunchPad	Right motor sleep

P3.7	Direct to MSP432 LaunchPad	Left motor sleep
P5.4	Direct to MSP432 LaunchPad	Left motor direction
P5.5	Direct to MSP432 LaunchPad	Right motor direction
P1.1, P1.4	Direct to MSP432 LaunchPad	Pushbutton on chassis to switch mode
P8.0, P8.5	Direct to MSP432 LaunchPad	Yellow Chassis LED lights
P8.6, P8.7	Direct to MSP432 LaunchPad	Red Chassis LED lights

- **Adafruit BLE UART Module Pinout:**

Adafruit BLE UART Module	Connection	Purpose
MOD	P1.6 of the MSP432 LaunchPad	To select between the two modes: Command and Data
CTS	GND	BLE module can send data back to the microcontroller
TX0	P9.6(PM_UCA3RXD)	UART transmit pin from BLE to the microcontroller
RX1	P9.7(PM_UCA3TXD)	UART receive pin into BLE module from the microcontroller
VIN	3.3V	Power supply for BLE module
RTS	Not Connected(N/C)	When it is fine to send data to the BLE module, this pin will be low.
GND	Ground	Common ground for BLE module
DFU	Not Connected(N/C)	To force the BLE module to enter a special firmware update mode

5 Background and Methodology

A tilt / gesture-controlled robot utilizes some concepts of embedded systems such as C programming, GPIO (General Purpose Input / Output), PWM (Pulse Width Modulation), and UART serial communication.

- GPIO (General Purpose Input / Output) pins are used to indicate the driving status of the robot in directions such as forward, backward, left, and right by turning on the RGB LED in a specific color. It is also used to switch the orientation of the smartphone control mode between vertical and horizontal through pushbuttons on the microcontroller. The chassis lights are also used to indicate which control mode is currently active.
- The PWM (Pulse Width Modulation) signal frequency is 50 Hz and the count value in Up/Down mode is 15000. Speed control of the motor is achieved by varying the PWM

signal for a specified drive command of the robot. In this project, the duty cycle used to drive motors was 30%.

- The eUSCI_A module is configured and initialized to enable UART communication between the BLE module and the microcontroller using pins P9.6(PM_UCA3RXD) and P9.7(PM_UCA3TXD).
- The BLE module communicates wirelessly with BluefruitConnect app on an iOS smartphone and receives quaternion sensor data from it. The BLE module then transmits that received data to the microcontroller MSP432 via UART serial communication.
- Based on the transmitted quaternion readings from the BLE module, the microcontroller controls robot movements such as forward, backward, left, right, and diagonal directions. The RGB LED will also be determined based on which direction the robot is moving. For example, tilting the smartphone forward will cause the robot to drive forward and turn on the green color of the LED. An additional separate feature is that the robot will adjust the speed depending on how much smartphone is tilted by using a linear equation to scale a given quaternion reading up to a PWM value.

6 Analysis and Results

6.1 Quaternion sensor reading using BluefruitConnect app on a smartphone built-in sensor

The BluefruitConnect smartphone app includes a controller feature that provides different ways to connect with the BLE module such as sensor data, control pad, and color picker. The app sends data ten times per second. All the modes use different data formats for sending data from a smartphone to the connected BLE module wirelessly. Every data packet begins with a prefix that starts with "!" which is followed by character that specifies type of data being sent. Each sensor data value is a float type which has a 4 byte length.

There are different types of sensor data in the BluefruitConnect such as quaternion, accelerometer, gyroscope, magnetometer, and location. The sensor data format is shown in the below table:

Sensor Name	Prefix	Data Packet Format
Quaternion	!Q	['!'] ['Q'] [float x] [float y] [float z] [float w] [CRC]
Accelerometer	!A	['!'] ['A'] [float x] [float y] [float z] [CRC]
Gyroscope	!G	['!'] ['G'] [float x] [float y] [float z] [CRC]
Magnetometer	!M	['!'] ['M'] [float x] [float y] [float z] [CRC]
Location	!L	['!'] ['L'] [float lat.] [float long.] [float alt.] [CRC]

Table.1 Sensor Data Packet Format

The quaternion sensor readings combine data of the accelerometer, gyroscope, and magnetometer readings. It is used to represent motion and orientation in 3D space. All the values are limited to [-1,1], and it gives four floating points values qx, qy, qz, and qw.

• qx (roll): tilt forward and backward (+x axis and –x axis)

• qy (pitch): rotate left and right (+y axis and –y axis)

• qz (yaw): move left and right with gravity (+z axis and –z axis)

• qw : real/scalar part of the combined data

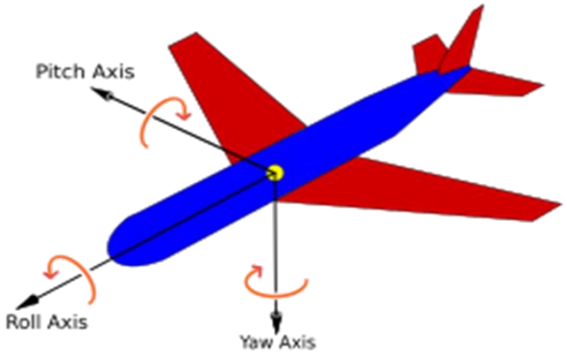


Figure.1- Source: Wikipedia

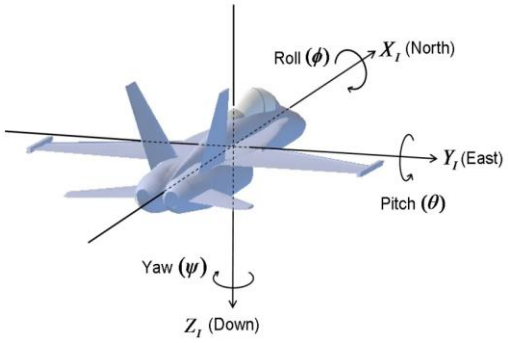


Figure.2

6.2 Configuration and functions for the BLE UART module

The BLE_UART_Init() function is used to configure the UART serial communication between the BLE module and MSP432 microcontroller. The pins P9.6 (PM_UCA3RXD) and P9.7 (PM_UCA3TXD) are configured as the primary module function to operate as the UART transmit and receive lines. The EUSCI_A3 module is initialized by configuring the bits in the CTLW0 register for UART communication parameters. These parameters include parity bit enable, bit order, length of the character, number of stop bits, reset state, UART mode enable, and clock source. The baud rate is set to 9600 bits per second, and the transmit and receive interrupts are enabled.

The BLE_UART_InChar() and BLE_UART_OutChar() functions send and receive single bytes by checking if the interrupt flag in the IFG register is set or not. If bit 1 of the IFG register is set, then the transmit buffer is empty, and a new byte is ready to be sent. If bit 0 of the IFG register is set, then the receive buffer is empty, and a new byte can be received. A character string is transmitted by using BLE_UART_OutString() function, which calls the BLE_UART_OutChar function in a loop until it reaches a null pointer, indicating the end of the string. To receive a string, the BLE_UART_InString() function is used to receive bytes in a loop until the carriage return character is received.

6.3 Reading quaternion data packet and extracting float values

In the beginning, we used the code from lab 5 to check if the BLE module was working or not. After that, we began to implement our own ideas and observed the transmitted data on the serial terminal. However, we observed unexpected results in how the quaternion packet was

being received. So, we began working on receiving the quaternion packet properly and storing each of its four readings into software.

The quaternion sensor data packet contains a total of 19 bytes. We created a function called BLE_UART_Read_Q_Packet(), which takes 19 bytes as a buffer size parameter. It reads a number of bytes (for our case, 19) and stores that reading into an array. This function is used to receive 19-byte data packets every 100 ms (0.1s). Using this function, the correct data was consistently being received, but the sensor was still transmitting an unexpected single byte at the beginning of each data packet. To resolve this, we copied the buffer array into a new array without the unexpected 1st byte. By using the memcpy C function, each of the quaternion values is copied from the receive buffer array into its own 4-byte float variable. If the data is not found in the buffer_pointer, it will print an error message. The Print_Quaternion(uint8_t *buffer_pointer) is used to print all the quaternion data on the serial terminal.

```
BLE UART Data: !, Q, 0.560363 -0.068003 -0.119994 0.816683
Packet size = 19
BLE UART char data: Ú!Qós▯?ðD‹½™¿õ½Q?
qx: 0.560363
qy: -0.068003
qz: -0.119994
qw: 0.816683
```

Figure. 3 Quaternion data packet format and extracted data

6.4 Features for this project

We used a preprocessor directive for conditional compilation for these three features.

6.4.1 Vertical Phone Mode

The main function contains all the initialized function which is needed for this project including DisableInterrupts(), Clock_Init48MHz(), LED2_Init(), Buttons_Init(), Chassis_Board_LEDs_Init(), EUSCI_A0_UART_Init_Printf(), BLE_UART_Init(), Motor_Init(), EnableInterrupts() and BLE_UART_Reset(). By using the BLE_UART_Read_Q_Packet() function, we receive and store a quaternion data packet. From the data packet, we extract the qx, qy, qz and qw floating point readings. The microcontroller checks the drive mode of the robot by checking the push button status. If the drive mode is zero, it will call the Process_BLE_UART_DATA_Vertical() function with four the reading parameters and it will turn on the red chassis lights.

The robot's movements are controlled by using only the qx and qy quaternion sensor reading which gives its motion and orientation data. By analyzing these quaternion sensor readings on a smartphone, we identified the range of the x axis and y axis for forward, backward, left, right,

and diagonal directions. From the analysis, If the qx value is positive and falls between 0.1 to 0.65, the robot moves forward by calling Motor_Forward() function using a 30% duty cycle and turning green LED on the chassis. By using these four functions, the robot movements are controlled: Motor_Forward(), Motor_Backward(), Motor_Left() and Motor_Right() using the nominal PWM value of 4500 which is a 30% duty cycle. The robot stops if both qx and qy reading values are in the range of [-0.1, 0.1]. The duty cycle formula is

$$Duty\ Cycle\ =\ \left(\frac{Compare\ Value}{Load\ Value}\right)\ \cdot\ 100$$

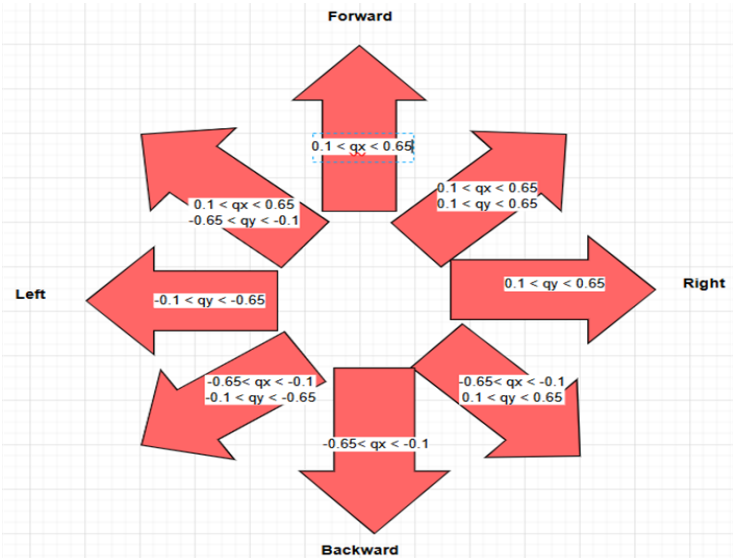


Figure. 4: Robot movement direction based on qx and qy quaternion values

6.4.2 Horizontal phone mode

First, the main function checks the drive mode by checking the status of the pushbutton. If the drive_mode is 1, that means it is in horizontal mode. It turns on the yellow light on the chassis and calls the Process_BLE_UART_DATA_Horizontal() function with quaternion parameters readings. In horizontal phone mode, robot movements are controlled by using qx and qy quaternion sensor readings. If the qy value is positive and falls between 0.1 to 0.65, the robot moves forward by calling the Motor_Forward() function using the nominal PWM value of 4500. The same four functions that are used in the vertical phone mode to control robot movements are also used in the horizontal mode with different conditions for various movement directions.

In the beginning, we were pressing buttons one after another, it was not switching modes, and the robot was driving based on the previous mode. After that, we analyzed that we must turn off the power button and turn it on again then press the button on the chassis. It switches the mode and drives accordingly to our idea. The vertical mode is called “gesture-controlled mode” and the horizontal mode is called “remote controlled mode”.

6.4.3 Speed control mode

The speed control mode allows users to control the speed of the robot for driving forward, backward, left, and right. It is not properly working for the diagonal directions. The speed depends on how much the user tilts the smartphone. By uncommenting the defined statement for this test case and commenting on the other two cases, we can execute this test case. For executing this test case, the Process_BLE_UART_DATA_Proportional() function is called with qx, qy,qz, and qw parameters. According to robot driving movements, it calls the Calculate_PWM() function to change the speed of the right and left motor. The Calculate_PWM() function takes the qx and qy values as parameters to calculate the PWM values using the following formula. The qx value is used for forward and backward directions. The qy value is substituted into the formula for left and right directions. The formula is

$$PWMValue = abs(qx \cdot 10000) - 500$$

To move in diagonal directions, there has to be a difference in the PWM values between the left and right motors. To do this for the speed control mode, we tried to calculate the PWM value as normal using the equation above for one motor. Then, this value is reduced by 33% to determine the PWM value of the other motor. This did not work from our testing, so more time is needed to figure out the diagonal directions for speed control mode.

7 Conclusions

In this project, we learned about how quaternion readings can be used to determine the orientation of an object. C programming was used to correctly receive and process the quaternion data packets sent by the smartphone app. UART communication, GPIO, and PWM were combined in this project to achieve the resulting tilt/gesture-controlled robot.

Overall, all the features are working properly, but there are many ways to improve this project with more time. Ultrasonic sensors could be added for object detection to allow the robot to automatically stop if it is about to crash into something. Additional time could be spent testing the vertical and horizontal modes to find more ideal quaternion value thresholds. This would make the robot more responsive and intuitive to the user driving it. The speed control mode works for forward, backward, left, and right directions, but more time is needed to implement diagonal directions.

8 References

- https://learn.adafruit.com/bluefruit-le-connect/controller
- https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
- https://medium.com/@dmdinithipurna/understanding-quaternions-67cf263c276f
- https://en.wikipedia.org/wiki/Aircraft_principal_axes
- https://www.geeksforgeeks.org/cpp/memcpy-in-cc/
- https://www.geeksforgeeks.org/cpp/cpp-preprocessor-directives-set-2/