

**College of Engineering**

**ELEC 491 – Electrical Engineering Design Project**

**Final Report**

**INTERACTIVE TETRIS GAME**

**Participant information:**

**VOLKAN TURGUT 69420**

**SARP TATLICI 69385**

**Project Advisor(s)**

**ALPER ERDOĞAN**

**09.06.2023**

Table of Contents

[ABSTRACT 3](#_Toc137512508)

[1. INTRODUCTION 5](#_Toc137512509)

[2. SYSTEM DESIGN 7](#_Toc137512510)

[2.1 Glove Design 7](#_Toc137512511)

[2.2 STM32 7](#_Toc137512512)

[2.3 ESP32 7](#_Toc137512513)

[2.4 Deep Learning Model 8](#_Toc137512514)

[2.5 IMU 8](#_Toc137512515)

[2.6 Wi-Fi 8](#_Toc137512516)

[2.7 Software of the Game 9](#_Toc137512517)

[3.ANALYSIS AND RESULTS 10](#_Toc137512518)

[4.CONCLUSION 11](#_Toc137512519)

[5. REFERENCES 11](#_Toc137512520)

[6.APPENDICES 11](#_Toc137512521)

## ABSTRACT

In this project, our goal was to design a microprocessor game. So, we thought about which game and which processor we will use this semester. Later, we thought that why would not we change the playing structure of a simple game and make it more fun? Eventually, we thought that we could change the playing mechanism of the Tetris game, with our Electrical Engineering knowledge. Since the Tetris game is played with a stick and a button, our main motivation was to change this mechanism to make the game more fun and interactive.

We designed a wireless glove trained with deep learning methods to understand 9 gestures and play the game accordingly. We used 2 microprocessors STM32 and ESP32. In STM32, we embed our innovative deep learning algorithm and trained it with our own well-prepared dataset. Also, we used IMU for the triggering effect which was read from STM32. ESP32 was used for wireless communication between the PC and the glove. Our project’s outcome did manage to fulfil what was promised in the proposal, with minor changes in design and training models.

In the future, our project can be used in many fun games such as Snake, Pinball, etc. With the glove we designed these games can be played wirelessly. Furthermore, we can add many gestures to change these games playing structure in order to make them more entertaining and interactive.

## INTRODUCTION

Our primary goal is to create a Tetris game that allows for interactive user interaction. The goal of our technology is to give kids an unmatched experience when playing older-generation video games. In contrast to the thousands of games currently available, we plan to develop an engaging game experience that won't be addicting and will, we hope, satisfy a child's daily gaming needs far more quickly than uninteresting PC games, hence reducing the average amount of time spent in front of a screen for the target audience.

Our objective was to achieve good, stable communication between master and slave, moreover, a strong deep learning algorithm that can recognize user hand motions and functional software. Furthermore, we had to implement an IMU-based trigger to switch between the commands. During the we have faced many problems. Such as glove design ideas, wi-fi connection, deep learning, SPI, STM32, and ESP32-based problems. To overcome these problems, we made lots of searches.

During the semester, we had 2 different glove designs. The first design was not looking good, and the flex sensors were not stable on it. We used that glove to make sure that flex sensors and IMU are working on the glove. Eventually, we designed a better-looking glove where the sensors and IMU are more stable on the glove as shown in Figure 1.

A picture containing furniture, indoor, chair, cable

Description automatically generated

Figure 1. Final Glove Design

We also faced STM32 and ESP32 problems. In the first place were only using ESP32 for both ADC and Wi-Fi. However, we saw that both ADC and Wi-Fi do not work in sync, and we concluded that ESP32 alone was not sufficient. Eventually, we started to use STM32. Now, ADC and deep learning model are in STM32, and the Wi-Fi connection is on ESP32. Another problem we faced was with IMU. From the data’s gx, gy, and gz only gz was working properly. Therefore, we used the gz value in order to trigger our system. We decided on a threshold value for gz and when the threshold is passed (rotating the glove to the right) the trigger is on.

## SYSTEM DESIGN

#### Glove Design

For a long time, we think about the glove design. The first glove we found was a construction worker glove and we sewed 5 flex sensors on the fingers. We used this design after we managed to reduce noise (hardware). Later, we wore the glove and read the sensor's values. However, we realized that our first design was not stable and not looking good. Later, we designed another glove that we sewed very stable and carefully. Also, we used patex to make solder points more stable. Furthermore, the IMU is on the glove sewed as well. The latest design was shown in Figure 1.

#### STM32

As we mentioned in the introduction ESP32 was not sufficient alone in this project. Afterward, we switch to STM32 as mentioned in our proposal. Before switching to STM32 we were thinking to use ESP and STM together, eventually, we had concerns about the communication between them. After a detailed search of their datasheets, we saw that it was manageable.

First, STM32 is a much better processor than ESP32 therefore we mainly focused on STM32. In STM32 we were reading the flex sensors values which were filtered with 0.1 microfarad capacitors. The read values are then classified into gestures by our trained model which was embedded in STM32. When the gesture is determined, STM32 sends this data to ESP32 through SPI. Moreover, to trigger our mechanism we used IMU. The IMU is located on the glove, and it is connected to STM32 with cables. The STM32 gets the IMU data and checks whether the data passes the threshold value that we decided. The value we were looking for was gz. When gz passes the threshold, our mechanism was triggered, and the gesture was performed in the game.

#### ESP32

In our project we used ESP32 Devkit Wroom. ESP32 is not quite as capable as STM32 as I mentioned previously. So, we wanted to work with simple as possible. During the semester we realised that ESP32 was not able to do ADC and Wi-Fi at the same time. So, we decided to use ESP32 as a communication tool. Between Bluetooth and Wi-Fi, we observed that Wi-Fi works better in ESP32. We have used Espressif IDE to use Wi-Fi communication between ESP32 and the PC. We have faced many issues such as ADC failure, Wi-Fi connection failure, especially with school Wi-Fi, and there was a lack of sources in order to use the Wi-Fi module in ESP32.

#### 2.4 Deep Learning Model

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, yazılım, bilgisayar simgesi, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu We used deep learning model for differentiating the gestures from each other. We could have used other methods which would be more inefficient that’s why we used machine learning to maximize our accuracy. Here is a schematic of our deep learning model.

Figure : Diagram of Deep Learning Model

Our deep learning model has 5 nodes in its input layer. These nodes are the flex sensors. Then we put two hidden layers. These hidden layers are 10 and 32 nodes respectively. We put these hidden layers to increase accuracy. We used Relu activation function in these layers because it is very fast to compute Relu and in hidden layers we need fast response because there are lots of interconnected nodes in these layers. But in the final layer there are 10 nodes that corresponds to 10 gestures. In this layer we used softmax because we need accuracy to be the maximum at this point. Later we converted this TensorFlow deep learning model into a TFLite model which is the TensorFlow model but compatible for microprocessor use case. We also looked at the datasheet and searched on internet which model files does our STM32 microprocessor support. Then we embed this model into our STM32 project. The model size is 2.45 KB. MACC is 647 and response time is approximately smaller than 0.1 seconds.

#### 2.5 IMU

IMU has played a significant role in our project. During the semester, we were thinking of IMU as a triggering mechanism for our game. In the beginning, we sewed the IMU and read the data in Arduino. The data was really good in Arduino, after we switched to STM32 from ESP32 and the glove design changed, everything for IMU got back where it started. Then, we connected the IMU to STM32 and used STMCube IDE for IMU activation. This part was the most challenging part for the IMU. The reason was again lack of information on the web. Eventually, we managed to get data from STM32, however the data we got was not sufficient. At this point, our engineering skills move in. We started to dig in the IMU data and made observations while our glove is being rotated. Finally, we concluded that when the glove is rotated, the gz data in the IMU was quickly oscillating between 0 and 8000. Later we defined a threshold. When the glove is rotated to right and if the data is above 6000, the triggering effect is performed in our game. To sum up, every time we want to trigger the move in the game, we have to rotate the glove right and perform the corresponding gesture you want to.

#### 2.6 Wi-Fi

We used ESP32 for wireless communication part. We transferred the data from STM32 to ESP32 by using SPI. We connected our ESP32 to PC hotspot. I also closed all the firewall on my PC because it rejects microprocessors connecting to my MQTT broker. I downloaded mosquito\_eclipse which is the software package for MQTT protocol. Then I set the mosquito and create the broker on windows cmd. Now broker is setup on my PC. My PC serves as a gateway at this point. ESP32 needs to publish data to a certain topic and certain port, and from another PC, we need to subscribe to that certain topic and port to listen those messages. 1881 port is the most suitable one because it does not enable TLS so that it does not require verification for security purposes. It would be better with a higher form of security, but things get a lot harder when building the project, that’s why for education purposes, we tried to keep things simple. The game was running on MAC and the broker was running on Windows. The game design was done on MAC and when it is carried to Windows, the scale of the game window was not compatible. MQTT was also not compatible with MAC so ESP32 connected to windows and published its messages onto Windows PC, and MAC was listening the messages dropped on Windows PC in short. A Python script was written for handling MQTT and controller part. We wrote a callback function where it triggers the function whenever a message is published to the topic that Python script was subscribed to. With proper libraries, a virtual keyboard instance was created and pressed corresponding keys.

#### 2.7 Software of the Game

The software design of the Interactive Tetris game was on Phyton. For the game, we import pygame, sys, random, and threading. Our game screen width and height were 400 and 1000 respectively, where the grid width and height were 10 and 25. We had 7 different blocks with 7 different colors. In our game, we have 3 screens which are the game screen, pause screen, and end game screen which are shown below. The Interactive Tetris game is very similar to a normal Tetris game where you can rotate and move the block but there are small additions by us.

A screen shot of a game

Description automatically generated with medium confidence A screen shot of a game

Description automatically generated with medium confidence A screenshot of a game

Description automatically generated

## 3.ANALYSIS AND RESULTS

Our design used to work very accurately and quite fast. We were able to distinguish hand gestures and receive them through Wi-Fi. However, before the demo day while we were getting some tests on new gestures we added to our deep learning model, we accidentally pull one of the flex sensors. Its position changed and even while our hand was unflexed, the flex sensor on our pinkie finger was flexed. And the amount of flex was getting bigger as we play the game and when we try to unflex it, then its position was changing. So, as we play with our glove more, our deep learning model were becoming less efficient in this case due to a small mistake. We had no time to sew it again because it was sewed by a tailor before. Also, the flex sensor started to make loose contact. We also had to re-solder the cables into our flex sensors, but our flex sensors were covered with epoxy which was hard as rock. We had no choice but to continue with this glove unfortunately. We tried to train a deep learning model with the damaged glove, and we thought that if it learns with damaged glove, it could produce accurate results. However, that is when we found that it makes loose contact. Still, we gave a shot, but it didn’t work well. It could have worked well if there was no loose contact, there was a small possibility.

Also, we used 1.5V 4 alkaline batteries. But after every 30 minutes we need to change the batteries because after some point, ESP32’s wi-fi module was being shut down due to insufficient amount of power. Our design required a lot of power because it was powering both microprocessors, there were an IMU connected via I2C and also there was a deep learning model running on STM32’s ARM processor. That is why we required a lot of power, and we were able to show the wireless part at the beginning of our demo day, but we weren’t later.

Before the incident on pinkie flex sensor, our trained model’s accuracy on the test set was 98.25% which was quite high. Also, we only encountered 1 or 2 wrong computed result of a hand gesture while trying and testing with real – time data.

## 4.CONCLUSION

## 

In conclusion, we designed a wireless controller glove that let’s users play Tetris game wirelessly with hand gestures. We have 5 flex sensors connected to STM32 microprocessor. We have also an IMU sensor attached to the STM32 microprocessor. We have a ESP32 connected to STM32 microprocessor. We enabled DMA (Direct Memory Access) for Analog to Digital conversion used on the flex sensors instead of polling the ADC because it was faster, and our model were handling a lot of things respectively, so we need to keep things as fast as possible.

We used timer interrupt for detecting the current angle (keeping the arm as the axis, so the angle of our wrist was calculated in the timer interrupt.) We used timer interrupt because this had to work every time, even though we were not performing a hand gesture because the IMU sensor was capable of measuring the angular velocity which ahs the unit of rad/sec. We need to discrete time integrate the angular velocity value to obtain the angle at all times. We put 60 degrees as the threshold and when the calculated angle exceeds the 60 degrees clockwise, the whole code was triggered. We used I2C for connecting the IMU to STM32 and we didn’t use library functions that IMU provided. We did the configuration, the reading all of the things by writing bytes into the corresponding memory address of IMU.

The whole code was first using DMA for ADC. After getting all the flex sensors analog values, it puts the flex values into the deep learning model. The model was created before the main loop. We called the deep learning model inside the if statement and run the inference with the given analog values. We mapped the gestures with binary encoding. It produced a binary number after the deep learning model.

Later we transferred the binary encoded number into ESP32 by using SPI. STM32 was the master device and ESP32 was the slave device. At ESP32 side, we used GPIO interrupt for detecting when the signal comes. We created a GPIO interrupt right on the slave select pin. Before transmitting the pin was first pulled to low, then message is transmitted then pin was pulled to high again by STM32. The slave select pin was an input for ESP32 because it is better for master device to control the flow of the communication and therefore it is usually the output pin for the master and input pin for the slave. By using interrupt, we detected when a signal comes and then we get the message from STM32 to ESP32. By the way, first three characters are 000 and after the first three characters, the significant part of the message takes place. Because while transmitting sometimes we read unwanted and weird characters at the first or first two characters when the message received to ESP32. To avoid or minimize the message loss, we put dummy characters for first three characters.

After receiving the message inside the interrupt, ESP32 published the message to the broker that is already configured and connected before the while loop. The ESP32 is connected to the 1883 port which does not use TLS certificate which eases the configuration process and connection process. However, it is not preferred because there were no security measurements at all.

Then on PC side there were two Python scripts. One was running the Tetris game and the other one was connecting to the mqtt broker, subscribing to the mqtt topic that the messages were publishing. We wrote a callback function in order to capture every message sent to the topic and created and instance of virtual keyboard which pressed the corresponding physical keyboard character with the given gestures. That is why, our model can be implemented on many games that uses keyboard or even mouse. However, our model is not designed to detect key holds or mouse drags so simple games are preferred. Still simple games could have been played or even with bigger command sets, you could write a whole essay. But our command set was limited because we were only allowed to use flex sensors for distinguishing the gestures. For further development, we could have taken all three axes of rotations from the IMU; inside a timer interrupt we could integrate the angular velocity all the time to get the angle value. Later we could have used the angle value in the deep learning model. But we tried it, and it was really challenging, and we had no time left. But if we were managed to do that, we would simply train a model that distinguishes symmetrical gestures which would increase our gesture set a lot. Also, the game would be more user friendly because the triggering function was rotating the wrist to right or clockwise. The gestures for moving left and moving right are two totally different gestures and after performing the gesture you have to turn your wrist to right or clockwise in order to trigger to whole system. People were confused at our stand and when they wanted to move the block to left, they rotated their wrist to left which didn’t do anything at all. A model with angles would be really good in conclusion.

## 5. REFERENCES

* [1] ESP32 series - espressif systems, https://www.espressif.com/sites/default/files/documentation/esp32\_datasheet\_en.pdf (accessed Jun. 13, 2023).
* [1] Alldatasheet.com, “STM32F429ZIT6 datasheet, PDF,” Alldatasheet, https://www.alldatasheet.com/view.jsp?Searchword=STM32F429ZIT6&sField=4 (accessed Jun. 14, 2023).

## 6.APPENDICES