B-Glove

Students

Amit Solomon

Aviel Simchi

Instructor

Boris Sosin

Supervisors

Yaron Honen

Boaz Sternfeld

Contents

Introduction 3

Project Timeline 3

Beginning 3

Arrival of the main components 3

First model 4

Second model 4

Glove 5

Problems and solutions 6

Suggestions for improvement 6

Hardware 7

Components 7

Hardware setup for ‘Black Pill’ 8

Software 10

Platforms and usages 10

Software setup for ‘Black Pill’ 10

Development 11

Controller 11

Adapter 12

Introduction

As the development and usage of VR and AR technologies increased in the last years there isn't yet a cheap and precise solution for **finger-tracking** problem.

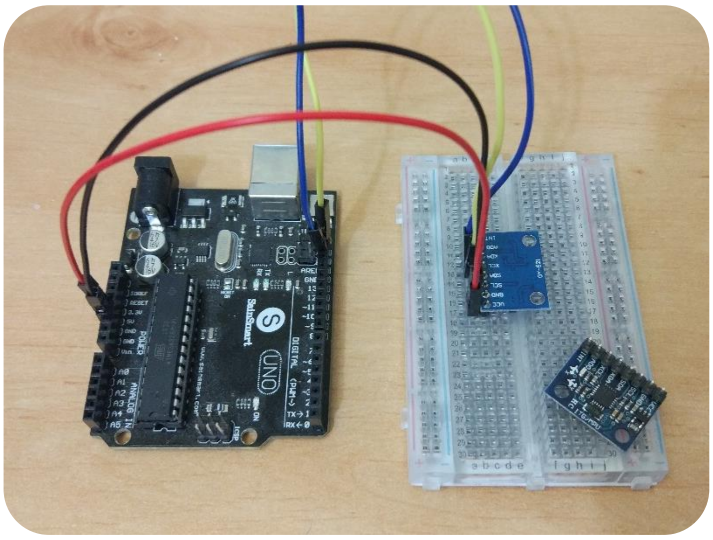
In our project we looked for creating a solution for this problem using IMUs (Inertial Measurement Unit) located on the fingertips and the back of the hand. With each IMU we were able to create a 3-dimentonal axis system and thus determine the relative direction of the hand and each finger to the ground and to each other.

As the HW team we provided a glove with the abilities described above and a C# adapter that can be used in unity environment.

Project Timeline

Beginning

1. First steps with Arduino Uno, Understanding the I2C protocol.
2. First soldering: MPU6050
3. Understanding the output of the Gyro and converting the raw data.
4. Displays the first orientation between two MPU6050 components, connected to Uno, and communicates with the I2C protocol, each with a different address.

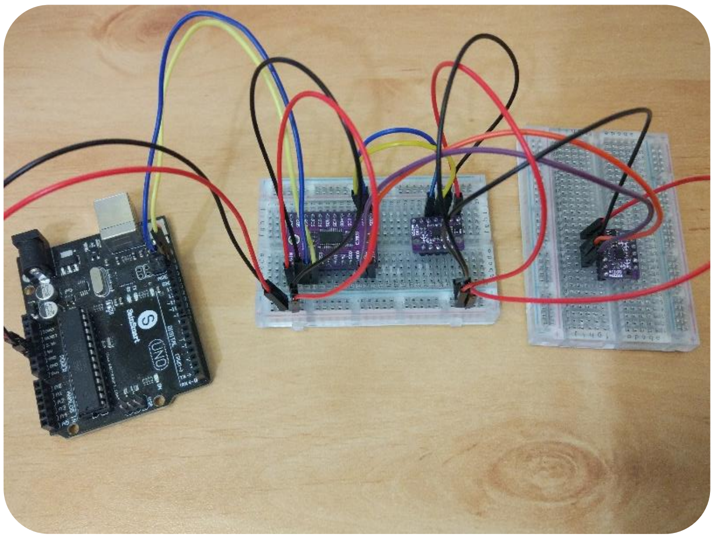


Arrival of the main components

5. Putting into use the BMI160 gyro models. These models have a higher accuracy and smaller dimensions which are suitable for installation on the fingers.
6. Learning and working with the TCA9548A Low-Voltage 8-Channel I2C Switch, adjust the libraries of BMI160 to support multi-use.

First model

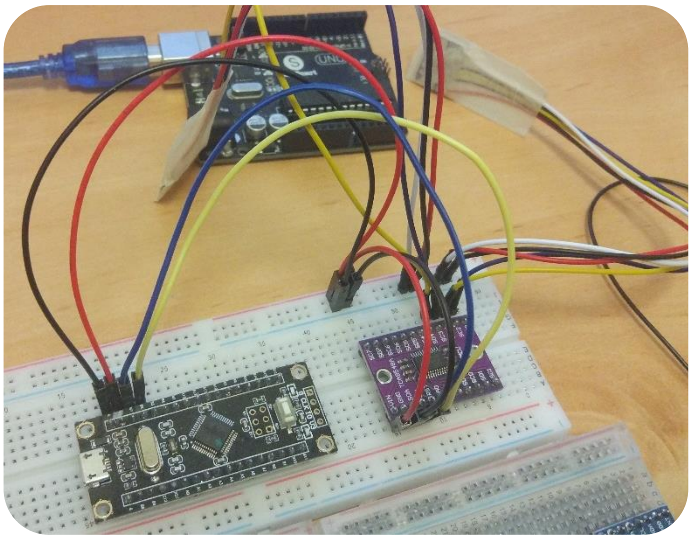
7. Arduino Uno + I2C Switch + 3xBMI160



   The results we received were unsatisfactory in terms of accuracy. We noticed that the problem is due to the lack of computing power of the Uno.
8. At this point we moved to develop the software side of the project in which we created:
   a. C# GloveAdapter: pulls from the serial port to internal database, all the additional calculations are made in this software module.
   b. C# PollerGlove: API to pull from the glove adapter to Unity environment. This interface will be used by the software team to update the hand 3D module.

Second model

9. STM32 + I2C Switch + 3xBMI160

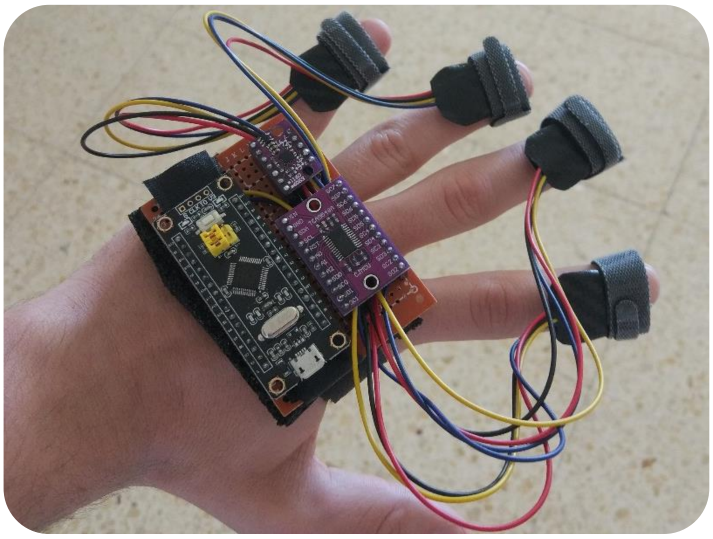


   To create the second model and improve the results, we first sold the STM32 Arm microcontroller. This supposed to be is much faster than the one we used the previous model.
10. After we did the soldering, we needed to figure out how to burn its bootloader, install the drivers and use the relevant libraries. (In the hardware section we will expand and show a guide on how we burned the bootloader using TTL to Serial Converter)

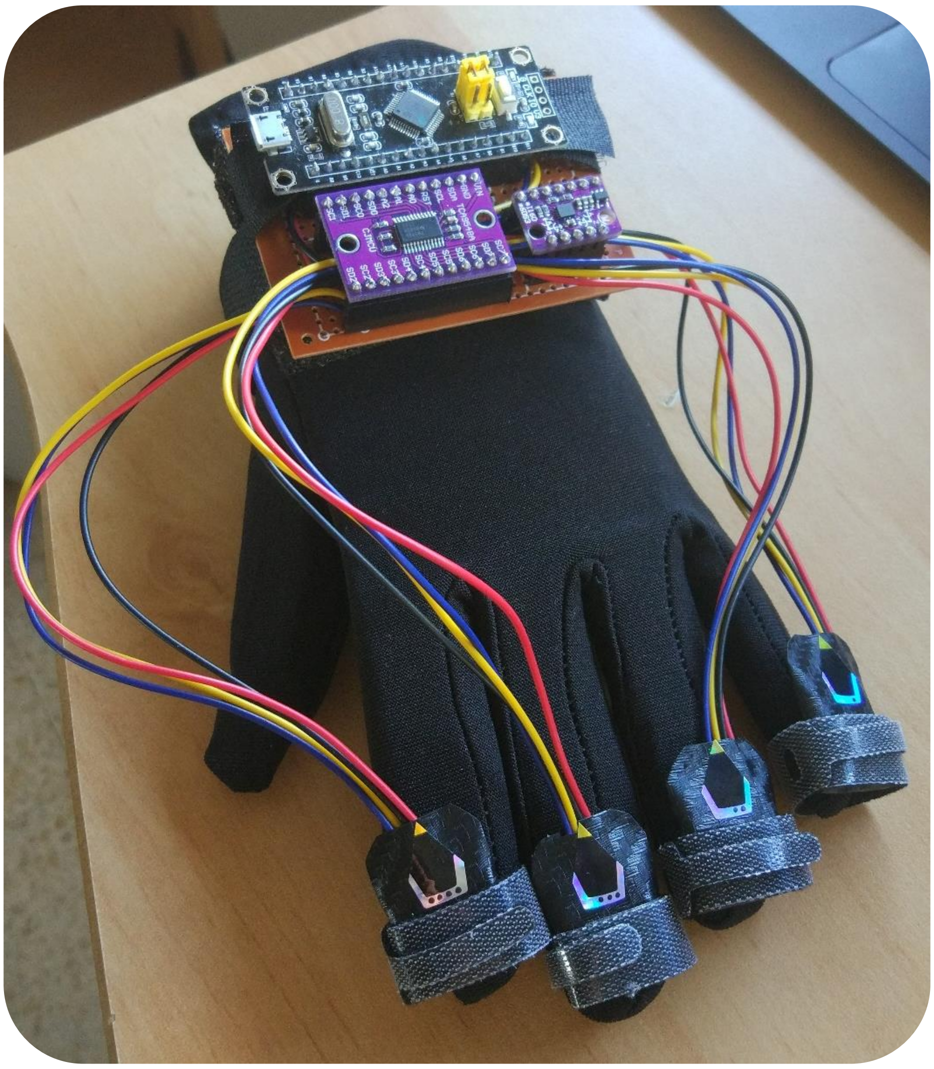
11. As we thought, the results have improved considerably with the STM32. So, we decided to continue to the next stage - building the glove.

Glove

12. After we created the main board that sits on the back of the hand, we examined several possible applications for the measurement components. Our conclusion was that the most convenient and easy to wear is the ring shape option.



13. Finally, we added stickers to the numbering of the components, and this is the final model on the artificial hand:



Problems and solutions

P: Inaccurate results.

S: Using the accelerometer to better calculation of the rotation and replacing the Arduino Uno with the STM32 Arm microcontroller for faster sampling rate.

P: Receiving invalid data (C# adapter).

S: Creating of a ‘safe-words’ at the beginning and the end of the packages. We chose the words so that the result of the XOR between them would be 255 and therefore, if there was a lost package between them, we knew about it and ignored the invalid datd. waited for the next opening word. Then again, we waited for the next opening ‘safe-word’.

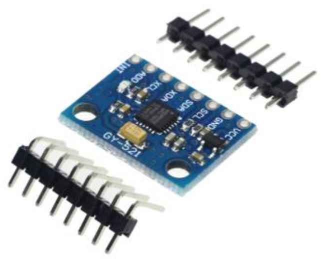
Suggestions for improvement

1. Using a Gyro sensor with magnetometer for improved accuracy.
2. Adding a Bluetooth module and battery.

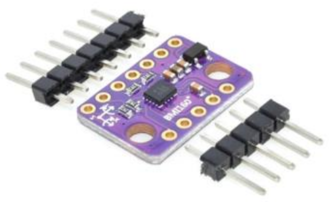
Hardware

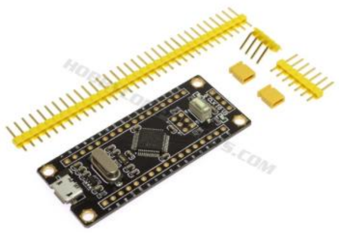
Components

Arduino Uno


MPU-6050 Six-Axis (Gyro + Accelerometer)

TCA9548 A Low-Voltage 8-Channel I2C Switch


BMI160 Small, low Power Inertial Measurement Unit

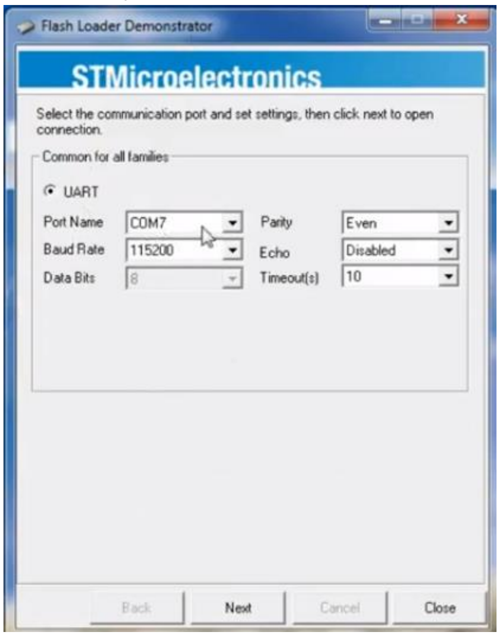

STM32F103C8 Mainstream Performance line, ARM Cortex-M3 MCU with 64 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN

TTL to Serial Converter

Hardware setup for 'Black Pill'

Most STM32 boards, including 'Black Pill', do not come with a USB bootloader installed so in order to use this controller a few setup steps should be done:

1. Download and install FlashLoaderDemonstrator and bootloader software: https://drive.google.com/file/d/1pijbw59Bw15fAqsFtYl-kHws_lXYikIl/view
2. Download drivers for Arduino STM32: https://github.com/rogerclarkmelbourne/Arduino_STM32
3. Set the hardware in the following order:
    a. Set Boot0 to 1 in the controller.
    b. Connect the TTL-adapter to the controller:
        i. TTL 3V<-> controller 3V
        ii. TTL GND <-> controller GND
        iii. TTL TX <-> controller A10
        iv. TTL RX <-> controller A9
    c. Connect the TTL adapter to the computer via USB port.
4. Open the Flash Loader Demonstrator:
    a. Select the port and set baud rate to 115200:



Select the communication port and set settings, then click next to open connection.

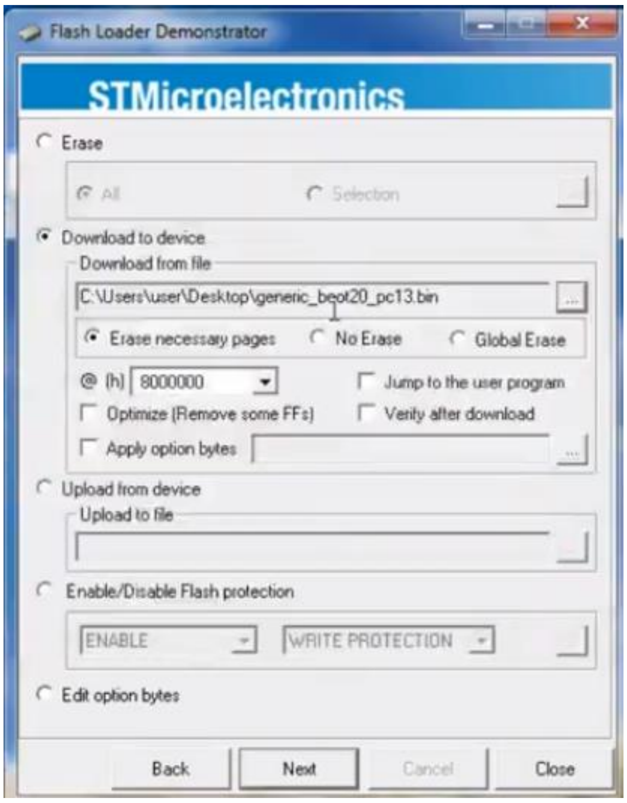
Common for all families

UART

Port Name	COM7	Parity	Even
Baud Rate	115200	Echo	Disabled
Data Bits	8	Timeout(s)	10

    b. Click next twice.

c. Select download to device and choose the bin file from step 1 and click next:



Download to device [✓]
Download from file: C:\Users\user\Desktop\generic_boot20_pc13.bin
Erase necessary pages [✓] No Erase Global Erase
@ (h) 8000000
Jump to the user program
Optimize (Remove some FFs)
Verify after download
Apply option bytes
Upload from device
Upload to file
Enable/Disable Flash protection
ENABLE
WRITE PROTECTION
Edit option bytes

d. The bootloader is installed.

5. Important! Before disconnecting the TTL adapter set Boot0 back to 0 or else you will need to repeat the whole process.
6. Open the Device Manger
   a. Under ports, right click on the Maple Serial, click Update Driver Software…
   b. Click on Browse my computer for driver software.
   c. Set the path to be drivers' folder from the extracted Arduino_STM32-master folder from step 2 and click Next.

Software

Platforms and usages

Arduino IDE – controller’s firmware.

Visual studio –software adapters.

Unity – software adapters and unity connectivity.

PyCharm – demo presentation.

Software setup for ‘Black Pill’

1. Open the Arduino IDE, click on File and Preferences.
2. Under Additional Boards Manager URLs add a comma and the following: http://dan.drown.org/stm32duino/package_STM32duino_index.json
3. Go to Tools, Board and click on Boards Manager…
    a. Wait for the downloads to end.
    b. Search STM32, select and install the package with STM32F1xx.
4. Click on Tools again:
    a. Set Board to Generic STM32F103C series.
    b. Set variant STM32F103C8 (20K RAM. 64K Flash).
    c. Set Upload method to STM32duino bootloader.
    d. Set Port to the one that you installed in the HW setup for ‘Black Pill’.

Development

Controller

BMI160 library

A few changes were made in the BMI160 library so we will be able to work with this IMU using a multiplexer.

Arduino sketch

A struct called BMI was created, it contains a BMI160GenClass instance (responsible for communicating with its IMU) and 3 floats variable that contains the current yaw, pitch and roll values.

Setup:

After the initialization of the serial port and the IMUs we empty the buffer.

Loop:

In each loop the controller handles one of the IMUs. The controller addresses the relevant multiplexer's entry, which is connected to an IMU and reads the raw data of its gyro and accelerometer. The accumulated yaw, pitch and roll angles (or the vectors) are calculated as follow:

The rotation is calculated by adding the change in direction to the last direction, the change is calculated by multiply the current angular velocity to the constant time interval between 2 sample.

$$rotation \; = angular\ velocity * loop\ time * num\ of\ gyros$$

With the accelerometer we can calculate the inclination of the IMUs, the formula of the inclination is for example in X direction is:

$$accelAngleX = arctan\left(\frac{accelAngleRawY}{accelAngleRawZ}\right)$$

$$CFangleX \; = \; (1 - weight) * (CFangleX \; + \; rotation\,) \; + \; weight \; * accelAngleX$$

After the calculation of yaw, pitch and roll we calculate the values of sin and cos function in order to improve performance and we use these values later when converting the yaw, pitch and roll values to unit vectors.

Because float values cannot be sent via serial port these values are converted to array of bytes so it can be sent to the adapter. In order to keep high precision, we multiply each float value by 100,000,000, convert it to unsinged long and set the MSB to 1 if the original number is a negative number, this value is the one that will be sent to the adapter (that will divide it by 100,000,000 with special consideration to the MSB to return it to the original value).

Two special values will be transmitted before and after the relevant data, e.g. for an entry whose entry is x the first byte is $[x\ mod2 \mid not(x) \mid x\ ]$ and the second is $not[first\ byte\ ]$. When performing XOR of these values the result is 0xFF. The reason for using these values is so the adapter will be able to validate the data that it receives.

Finally, the function waits until it completed a full sample interval for the validity of the calculations.

Adapter

We created a class for the adapter called GloveAdapter. Within this class we created another class called GyroData. An instance of GyroData is associated with an IMU and contains 2 fields: the first is asix_vec which contains the directions vectors of each IMU and the second is is_valid, which indicated if the current data in asix_vec is valid.

GloveAdapter contains an array of GyroData instances for each IMU in the glove. In its start function it initializes its GyroData array (all is_valid are false at this point) and opens a communication with the serial port.

In the Update function the adapter tried to read 38 bytes (1 start validation byte + 3 directions * 3 vectors for each direction * 4 bytes for each vector + 1 end validation byte). The adapter performs XOR on the validation bytes, if the result is not 0xFF then it means that these 38 bytes are invalid and the adapter will continue to read one byte at a time until it find a start and XOR it to the byte that was received 38 reads before until the XOR returns 0xFF and then we back to red new 38 bytes. If the result of the XOR is 0xFF then it means we received a valid data of single gyro whose index is $[first\ byte\ \&\ 7\ ]$ and now we can update the value its correspond GyroData. Before the update the adapter will set is_valid to false and after the update to true.

When a user wants to use the adapter in Unity he should add a public GameObject field which will be the adapter GameObject and add a private GloveAdapter field for the adapter. In Start function he should find the adapter's GameObject and assign it to the corresponded field and use assign its private adapter with the result of GetComponent of his GameObject.

In update function the user will call the adapter's getGyroValues with 2 parameters – the first is the desired IMU index and the second is 2-dimensional array with size 3x3. If the return value is true, then the array is filled with the unit vectors of the IMUs directions and if the return value is false then the array remains untouched.