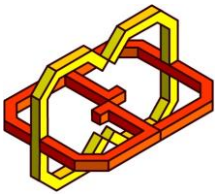


B-Glove 3D

Students

Aviel Simchi

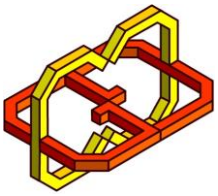
Tal Leibovitz

Supervisors

Yaron Honen

Boaz Sternfeld

Boris Sosin



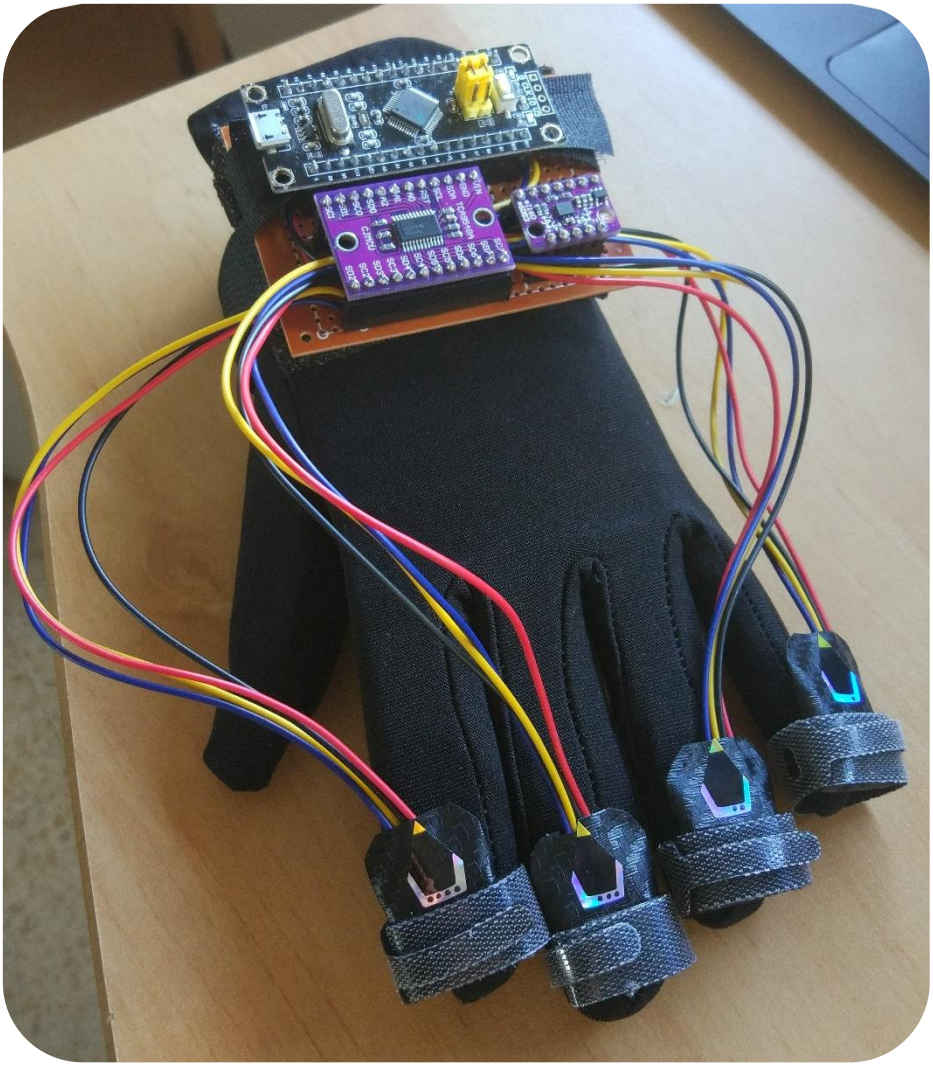
Contents

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.

This follow-up project designed to continue the work we have done as members of the B-Glove hardware and software teams. In our main project, we looked for creating a solution for the finger-tracking problem using IMUs (Inertial Measurement Unit) located on the fingers and the back of the hand.

With the help of IMUs, we were able to determine the orientation of each finger and the entire hand. These orientations are sent to the computer which displays a 3D hand in Unity scene.



Project Timeline

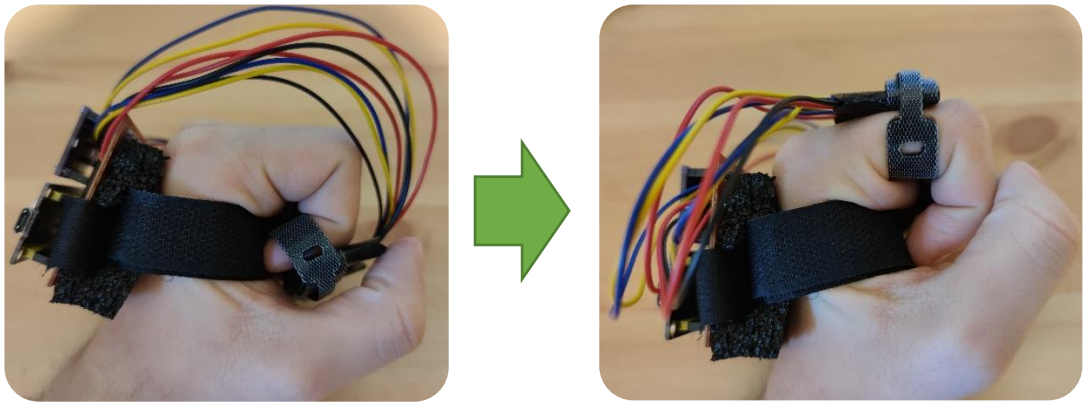
IMU Orientation to Finger Orientation

- **Description of the issue - 1 IMU for the 3 parts of the finger:** each finger is composed of **3** little parts. In our Unity scene, every part of each finger needs to know how to behave according to the **1** orientation we get from one IMU on each finger.

- **Description of the solution - simplification of the hand movement**: the way we handled this issue was to think as every one of the 3 finger parts as basically the same. We gave one script to each finger and all parts used the same logic. This was possible due to simplification we thought of in the hand movement.
  The simplification is that a normal finger posture is attributed with the same angle (more or less) for each finger part. For example, if we want a full closure (punch) posture so the first part of the finger will be 90 degrees rotated from the palm, the next part will be 90 degrees from the first part and the same for the last part.
  Of course, we are losing some finger abilities with this simplification, but the overall movement is compatible with the actual glove.

Initial Integration

- First, we were required to learn about each other's work in the first part of the project and plan the changes and improvements required for integration.
- Change IMU's finger position: from fingertip to the first part of the finger



And in doing so also changed the way in which the orientation was calculated.

First impressions

We evaluated the quality of performance by examining the following

- **Data path**: serial port had some synchronization issue and the reading rate did not match the reading rate.
- **Data Integrity**: sometimes packages were incomplete.
- **Data quality**: gyro had bias and orientation of the Unity 3D-hand was not as in reality.

In order to solve all the above and more, we added and change some features.

Firmware Improvements

- **Change IMU's data integration method - from varying to constant time interval**. This allow us to better control the writing to serial port and also to get more accurate data integration. We conclude that the latter indicates that giving the timestamps to samples in the old method was not accurate enough and probably should have been done in a lower firmware layer.

- **Add IMU's data filter** - using Madgwick AHRS IMU fusion algorithm we were able to get more accurate and reliable data, even after long operation time, and so the bias issue is solved. This algorithm uses an accelerometer in addition to the gyroscope and hence the improvement.

- **Serial data validation** - 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 data. waited for the next opening word. Then again, we waited for the next opening 'safe-word'.

Software Improvements

- **Adding the glove adapter host** - we have created an empty object in unity to hold the script that handles the data arriving from the hardware.
  The script attached to this object contacts the glove port, fetched the data from it and store it in a collection. Other objects can use this adapter to pull data in each update run.

- **Adding the mock adapter** - early in the process we started working separately. Since we only had one glove, one of us had to work with the unity without the input from the glove. We wanted to simulate the input and so we have created the mock adapter.

- We have created a C# interface with a method for supplying the input for the hand and the fingers. Both the glove adapter and the glove adapter mock implement the method for pulling data.

- **Creating Prefabs** - during our development process we needed to split some scenes and run some tests separately on different versions of the hand. This made us replicate changes we did every time. Later in time we learned about prefabs and the ability to only handle a base object and every change reflected in all our replicas.

- **Adding control abilities** - after the first integration succeeded, we wanted to test the hand in interaction with other objects in unity. Since we didn't have a component of location for the arm in the world, the interaction was missing. Hence, we added more controlling options to the hand and more important – to the camera. We have also added shortcuts for fixed hand and camera positions.

- **Making configuration easy** - switching between mock and glove configuration took us some time as well as other configs. We have created a constants file that made the development process easier.

- **Adding new objects for interaction** - we have added a 'piano' (a table with white stripes that changes color from white to black when touched by fingers) and a door that can be open by the hand. we also added a small non regular object ('Funny object') that its position is fixed and can rotate with the hand touch.

Results and conclusions

- In summary, in our estimation and considering the accuracy and reliability of the data, it is good enough to prove the concept proposed.

- We believe that further investment and consideration of the suggestions in the next section can be a start on the path to a glove that will determine the orientation of the hand by using accelerometers.

Suggestions for follow-up project

Firmware

1. Add IMUs for fingertips as well (a total of 11 IMUs for each hand) in order to get the full modeling of the hand.

2. Using an IMUs with build-in orientation calculations or at least to use sensors with magnetometer for improved accuracy.

3. Adding a Bluetooth module and battery.

Software

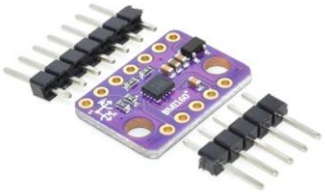
4. **Using a 'professional' Unity objects**
   In this project, everything that's in the unity environment was our work. Because the project purpose was POC of integration between the glove and the code, there was no need in more sophisticated Unity objects. The next step will be to use a richer environment and perhaps a more professional hand.
   Of course, well-made objects in Unity are not free and that should be considered.

5. **Initial Unity orientation calibration**
   One small but very significant enhancement would be to calibrate the hand to match the actual glove orientation. The unity hand already has the manual control options so one way to achieve calibration is to rotate the hand to the right orientation and let it be a fixed rotation gap throughout the scene.

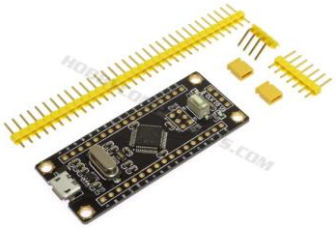
Hardware

Components

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_IXYikII/view

   Or use the one we provided at: \Initial setup\Bootloader

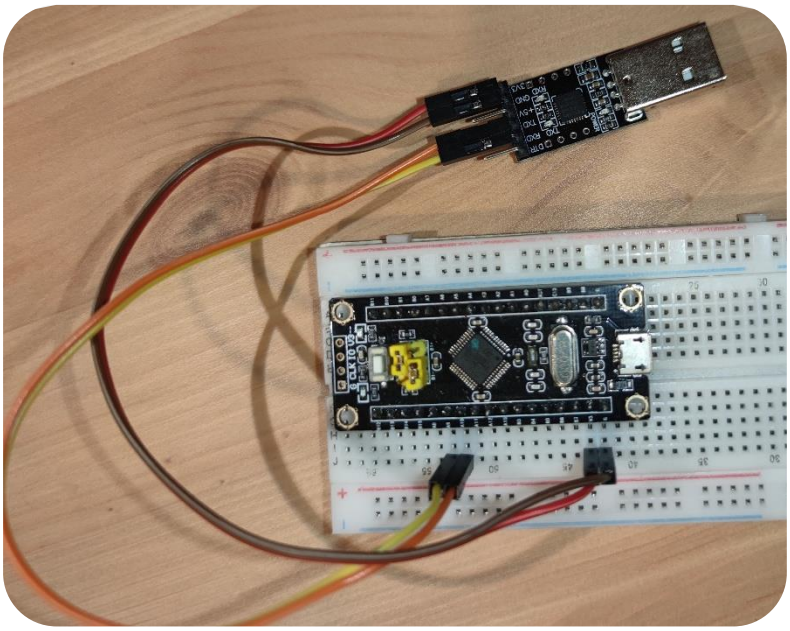
2. Download drivers for Arduino STM32:
   https://github.com/rogerclarkmelbourne/Arduino_STM32

   Or use the one we provided at: \Initial setup\Hardware\Arduino_STM32-master

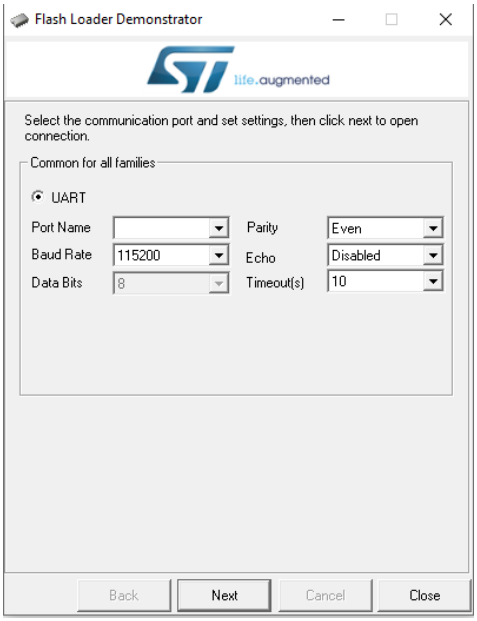
3. Set the hardware in the following order:
   a. Set Boot0 to 1 in the controller.

   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

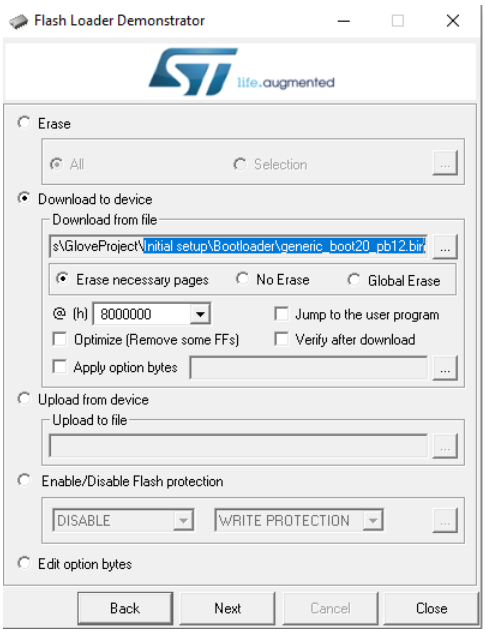
   c. Connect the TTL adapter to the computer via USB port.
      This should be look like the following:



4. Open the Flash Loader Demonstrator:
    a. Select the port and set baud rate to 115200:



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



    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 – debug serial port and data.

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 IMU was created, it contains a BMI160GenClass instance (responsible for communicating with its IMU), Madgwick filter – the reading data filter, timestamp of the latest reading, counter of the latest update to the serial, unique start and end sequence to each IMU that used to validate data transfer via serial port.

Setup:

Initialization of the serial port, filter, and the IMUs.

Loop:

In each loop the controller handles one of the IMUs.

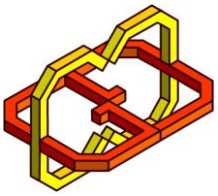
- First, we check if it's time to read data and update the filter.
- Controller addresses the relevant multiplexer's entry, which is connected to an IMU and reads the raw data of its gyro and accelerometer.
- Raw data is converted to gravity and degrees/second units.
- Filter is updated with the new reading, which computes orientation
- If this loop is the time that we need to send data to serial, we do as follow:
    - We get the current yaw, pitch and roll in float. 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.
    - 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 $[x\ mod 2 \mid not(x) \mid x\ ]$ 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.

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 angles which contains the orientation angels of each IMU and the second is is_valid, which indicated if the current data in angles 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 14 bytes (1 start validation byte + 3 directions * 4 bytes for each angle + 1 end validation byte). The adapter performs XOR on the validation bytes, if the result is not 0xFF then it means that these 14 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 14 reads before until the XOR returns 0xFF and then we back to red new 14 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 1-dimensional array with size 3 for x,y,z orientations. 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.

Unity

Controlling the hand and camera - we have created a lot of controlling possibilities for hand and camera, some of them are now silenced as the actual glove is dictating the movement but can be used again very easily. Adding other abilities to the existing mechanism is simple (just add more if-else statements)

Hand movement:

- Down – Page down key
- Up – Page up key
- Left – left key
- Right – right key
- Forward – up key
- Backward – down key

Hand interaction with object:

- Release object – R

Camera movement:

- Down – E
- Up – Q
- Left – A
- Right – D
- Forward – W
- Backward – S
- Rotate left – Z
- Rotate right – X
- Rotate up – V
- Rotate down - C

Fingers movement:

(not in use but in the code)

- Finger 1 (pinky)
  - Down – semi colon (“;”)
  - Up – P
- Finger 2 (ring)
  - Down – L
  - Up – O
- Finger 3 (middle)
  - Down – K
  - Up – I
- Finger 4 (index)
  - Down – J
  - Up –U
- Finger 5 (thumb)
  - Down – H
  - Up – Y

Camera & Hand fixed points:

- Door – 1 (above ‘Q’)
- Bowling game - 2
- Simple ball with table – 3
- Funny object – 4
- Basketball game – 5
- Piano – 7
- Overall view – backspace (just the camera)