EECS 249A Project Report: CubicHand

Kevin Albers, Robert Bui, José Oyola, Naren Vasanad

I. Introduction

This project aimed at creating an interactive light-emitting diode (LED) cube that can be controlled with hand gestures using a data glove. The motivation behind this project was to explore technologies in the realm of the Internet of Things (IoT) to create an application controlled through the internet using a model-based approach. A data glove is used for gesture recognition as its sensors play an important role in IoT applications, especially in wearable technology. The project models the turning on and off of LEDs based on the position and movement of the data glove. The goal is to accurately monitor the data gloves orientation and movement and display a smaller cube of LEDs (referred to a square of LEDs to avoid confusion) on the cube that will be updated based on information collected from the bend sensors, gyroscope, and the accelerometers of the data glove through wireless communication over Wi-Fi. In particular, the movement of the glove translates to movement of the square of LEDs in the cube, and the bending of different fingers in the glove translates to color and size changes of the square. The structure of the software was designed to be modular to serve as a basis for future work in code generation for embedded devices.

II. Focus Areas

A. Real-Time Networks

As shown in Fig 1 in the Appendix, the system consists of a real time network where packets of data are being transmitted over Wi-Fi using the TCP/IP socket communication protocol. The data glove continuously sends sensor data to the laptop which then routes the data to an mbed device, the Freescale FRDM-KL25Z. The mbed is connected to a CC3000 Wi-Fi module which allows it to receive packets of data using socket communication. Based on the information stored in the packets, the mbed device runs algorithms necessary to interpret the raw sensor data and update the LED cube.

B. Design Methodologies for Embedded System Design

The entire system can also be modeled as a synchronous data-flow (SDF) model with each part of the system representing an actor that fires when triggered with a certain number of inputs. An SDF clearly depicts the modules present in the system and shows how the entire structure has to be sequential. The SDF will be explained in more detail in Section III.

C. Modeling of Physical Dynamics

Since the data glove gives only raw sensor data, sensor data needed to be measured for the minimum and maximum values along with the precision determine how different gestures should be classified. Fig 2 in the Appendix shows these values for the data glove. As shown in Figure 2, the finger sensor data had a step size of 1 and ranged from 0 to 1000, giving an accuracy of 0.1%. Also, the angles for roll, pitch, and yaw had a precision of 0.01 degrees and varied between -90.00 and +90.00 degrees.

Roll, pitch and yaw are obtained from the quaternion data. Quaternions represent the accelerometer and gyroscope readings in the form of complex numbers and can then be converted into angular position using the following formulae:

$$roll = tan^{-1}\left(\frac{2(q_{00}q_{11} + q_{22}q_{33})}{1 - 2(q_{11}q_{11} + q_{22}q_{22})}\right)\frac{180}{\pi}$$

$$pitch = sin^{-1}\left(2\left(q_{00}q_{22} - q_{33}q_{11}\right)\right)\frac{180}{\pi}$$

$$yaw = tan^{-1}\left(\frac{2(q_{00}q_{33} + q_{11}q_{22})}{1 - 2(q_{22}q_{22} + q_{33}q_{33})}\right)\frac{180}{\pi}$$

q_{00}, q_{11}, q_{22} and q_{33} represent the four quaternion values sent by the data glove. The quaternion data also needs to be normalized since the data sent by the glove is in the range of 0 to 32768 (2^{16}). Hence, the sensor data received from the data glove is first divided by 2^{16} before it is used in the above formulae.

This sensor data is then filtered and fed into a gesture recognition module. These topics will be covered in more detail later in the document.

III. Model-Based Design

A. Overall Dataflow Model

The system was modeled using a SDF as shown Fig 3, Fig 4, Fig 5, and Fig 6.

B. Data Glove

As shown in Fig 3 and Fig 4 in the Appendix, the first actor, "Glove", gives three angular positions (roll, pitch and yaw) with the bend sensor data. This actor can be further divided into another data flow model that consists of a "Wi-Fi Socket" actor that receives raw packets via TCP/IP communication and sends it to a packet parser actor that interprets the raw packets and converts them into bend sensor and angular data. The Wi-Fi actor is blocked and cannot fire until a packet has been received via Wi-Fi. The packet parser produces one packet on each of its output ports containing information about finger bend and angle of movement.

C. Correction

The second actor is "Correction" as shown in Fig 5. "Correction" is a finite state machine that consists of two states: train and filter. The sensor data is normalized before

it can be fed to "Gesture Recognition". Since the initial state of the sensors may not be the same on every reset of the data glove, a reference zero is required in order to have consistent models for gesture recognition. After the data is trained, it can be smoothened using filters such as alpha filters or Kalman filters to eliminate jitters in sensor data.

D. Gesture Recognition

The third and fourth actors deal with gesture recognition. For every ten packets produced by "Correction", "Gesture Recognition" fires once. This procedure is used to down sample data obtained from the data glove. This allows for gestures to be interpreted more accurately. The project currently supports the use of ten gestures based on thresholds shown in Figure 7 in the Appendix. The "Tilt Detection" actor uses angular data to determine if the LED square should move in the x, y or z axis depending on roll, yaw and pitch, respectively. The LED square moves if the roll, pitch, or yaw becomes greater than +/- 10 degrees from the calibrated initial position, for a total of six gestures. Figure 8 shows how data glove movements correspond to movements on the LED cube. "Gesture Recognition" reads the bend sensor data to determine if the size or hue of the cube should be changed based on which fingers are bent. The thumb is considered to be bent if the value obtained from the data glove exceeds 200 ADC units. Whereas, the other four fingers are considered to be bent if the value from the data glove exceeds 350 ADC units. Figure 9 shows the four gestures based on the bend sensors. By keeping fingers 2 and 3 unbent and the others bent, the size of the LED square can be increased, while keeping only finger 3 unbent will decrease the size of the cube. The color of the LED square is based on how bent finger 3 was when the finger 4 is unbent and the other fingers are bent. Figure 10 shows how gestures are recognized in the "Gesture Recognition" actor.

E. Update Cube

The final actor, "Update Cube", fires for every output from the gesture recognition block. Since it is an actor that does not produce any tokens, it is a terminal actor. It requires a total of 3 tokens at the input: hue, change in size, change in the x-axis direction, change in the y-axis direction, and change in the z-axis direction as one packet. Figure 6 shows a flow diagram for this actor. As shown in the diagram, "Update Cube" has several actors within it: one for the setting color based on hue, one for incrementing or decrementing the size based on the size input, and one for moving the square based on the changes in the x, y and z-axis directions. A final actor inside "Update Cube", called cubeUpdate, will handle the updating of the LED cube itself by clearing the previous state of the square from the LEDs and re-drawing the new state of the square based on the new values for size, color, and position. In order to better display the 3D effect, the square of LEDs is drawn on all sides of the cube, with the brightness corresponding to the distance between the square and the surface of the cube. This allows for having the lighted square be "inside" the cube without being attached to any of the surfaces. Figure 11 shows flow diagrams for each of the first three actors, and Figure 12 shows the flow diagram for the final actor.

IV. Hardware

A. Data glove

The VirtualRealities DG5 Dataglove is used to determine the state of the users hand based on its internal sensors. Accelerometer and gyroscope data are used to output rotational quaternion data internally, which is sent to a client over Wi-Fi, along with finger bendness data as a percentage. As mentioned before, the quaternion data was used to determine movement on the cube, and the bend sensor data determined the gestures. The Dataglove offered a simple packet structure for data sent over the TCP Wi-Fi network. The structure is as follows:

[Header ($)] [Command] [Package Length] [Package Data] [Checksum] [End Character (#)]

A simple start command begins streaming requested data from the the Dataglove over a Wi-Fi network. The data packet provides information not only about quaternion and finger sensor data, but also a clock counter that timestamps when the data was logged. For example, while receiving quaternion and bend sensor data from the data glove, the received packet size is 42 bytes, 8 bytes of header information and 34 bytes of actual sensor data.

B. mbed

The mbed device, a Freescale FRDM-KL25Z, was chosen to be the central processor for the application. It receives the information transmitted from the Dataglove to the CC3000 and processes this information to determine the status of the Neopixel LED strips that form the cube. This embedded platform was chosen because it provides the low level timing control required for the NeoPixels and a processor faster than competing devices such as the Arduino. It is also a new platform with an online development environment based around applications for the IoT realm which this project is based.

C. Neopixel LEDs

The Neopixel LED strip was used to create three surfaces of the 10x10x10 cube. The LEDs are controlled by an mbed using the Multi-WS2811 library by Richard Thompson. This library is inherently limited to a maximum of 240 LEDs per strip due to the 16 kB ram on the device. Since the LED cube requires 300 LEDs, two digital pins on the mbed device are used. One pin controls a strip of 100 LEDs (10 strips of 10 LEDs connected) for the top of the cube and another pins controls a strip of 200 LEDs (10 strips of 20 LEDs connected) that is shared between the two sides of the cube. The library allows the mbed to individually address each LED and set each color with 8 bits for per color (RGB).

D. CC3000

The CC3000 Wi-Fi chip connected to the mbed works as a server when connected to the client PC over a LAN created by the team. This is being done by using the cc3000_hostdriver_mbedsocket library by Martin Kojtal. The Dataglove sends quaternion and bend sensor packets to the client PC, which forwards these packets to the mbed for parsing.

V. Software

A. Version control

Software version control was done by using the built-in version control on the mbeds online developer site. Version control helped keep track and merge the code from different aspects of the project that were developed in parallel. It was important that the code was maintained in a structured format in order to be able to revert back to changes. Code created locally to create a client on the computer in order to emulate a communication network with an intermediary system was maintained using Github. The links to the mbed project and the Github repository are mentioned in the Appendix.

B. C/C++

C/C++ was used for the development of the software for programming the mbed device and in the creation of the TCP/IP client on the computer. Using object oriented programming concepts, classes were developed for each of the modules. The code was structured such that each of the actors was an object of a class. A single function of the class would be run for firing the actor. This function returns data which needs to be sent to the following actor in the chain. Such structuring will allow ease in replacement of blocks or actors. It will also make automatic code generation from actor models easier for future development as will be discussed later in the report.

VI. Results

A. Wi-Fi

The Wi-Fi connection to the glove turned out to be the most troublesome part of the system in terms of latency. The data glove required the use of a TCP connection in order to receive data from the glove. This can cause problems with real time systems in embedded systems. The system on the mbed device is triggered by receiving packets from the data glove, which means the timing of the entire system is reliant on receiving data over Wi-Fi. When packets are missed, TCP will request for them to be redelivered, and wait for the new packets to arrive even if new packets have arrived, so that they are not received out of order. This causes delays up to 1 to 2 seconds which will cause the application to freeze occasionally throughout the process.

In order to see the consistency of data delivery from the Dataglove, an LED on the mbed was set to blink for every 50 packets received. Since the PC client sends 50 packets per second, this will blink once a second. When observed during runtime, the LED blinked inconsistently. For short periods it would blink at a constant 1 Hz, then it would slow down or stop blinking for a moment, then recover by blinking faster over the next period, which reflects the problems that would be expected by using TCP. Preferably, UDP should be used for this type of application, since it will simply ignore dropped packets. The DataGlove provides time data which can then be used to properly time the packets and adjust the algorithm for any dropped packets detected by this method.

B. LED Cube

The LED cube, shown in Figure 13, was built with black foam board cut with an X-Acto knife and glued together with hot glue. The cube was designed to have all LEDs on all three sides equally spaced, to maintain the 3D illusion of the lighted square moving inside of the cube. As mentioned in the Hardware section, the Neopixel LED strips were cut into 10 strips of 10 for the top of the cube, and 10 strips of 20 for the two sides of the cube. Female headers were soldered to each of the strips to connect them together. For each of the strips, double-sided tape was laid out on the foam board to attach the strip. The 10 strips of 10 for the top of the cube were were then wired together using black hookup wire, as were the 10 strips of 20 for the sides of the cube. Both of these longer strips were then connected to the mbed to control the LEDs. The cube is powered by a 5V wall adapter to provide the current necessary to power all 300 LEDs simultaneously at full brightness.

C. Final Project Demo

For the final project demo, the team showcased the interaction between the data glove and the LED cube. The two wifi enabled devices communicated through a computer on a LAN network. The data glove transmits quaternion and bend sensor data to the mbed device using TCP/IP protocol. The mbed interprets the data through gesture recognition to modify and move an LED square. For an added 3D effect, the lighted square is always shown on all three sides of the cube, with the brightness varying based on the distance from the square to the surface of the cube. The wearer of the data glove can successfully move the lighted square on all three axes of the cube and change its size and color based on the gestures mentioned previously. The demo can be viewed in the YouTube link mentioned in the Appendix.

VII. Conclusion

The project involved understanding constraints of real time systems through TCP/IP communication, modeling physical systems through gesture recognition and also in creating a model-based structure to the entire system using SDF models. These concepts helped in solving the problem in the right way. It also helped in making the system scalable and allowed introduction of new concepts into the flow much more easily. The final application was successful and showcased these concepts well by using Wi-Fi communication to successfully transmit sensor data and interpret the data to classify gestures for controlling an LED cube.

VIII. Future plans

The CubicHand project is a part of a larger project for code generating applications for the IoT. Since the project was constructed using a model based structure, it serves as basis of comparison for future code generation. Using code generation concepts for SDF models, the same application should be reproducible.

Another focus of the project is on improving gesture recognition using machine learning. The gestures in the current

model are simple and can be extended and made more robust by using machine learning algorithms. By performing unsupervised learning on sensor data collected from the data glove, it will be possible to minimize errors and maximize accuracy of gestures.

Finally, to be able to handle gestures more effectively and avoid connectivity problems, a data glove would be built to enable communication using Wi-Fi and also Bluetooth Low Energy (BLE) to evaluate different communication technologies. The data glove would include individually-selected bend sensors, accelerometers, gyroscopes and IMUs to obtain better orientation information.

IX. Appendix

Video: https://www.youtube.com/watch?v=eEFVdnKhD9I
mbed Code: https://developer.mbed.org/teams/Model-Based-Team/code/CubicHandServer/
Github: https://github.com/calnaren/CubicHand.git

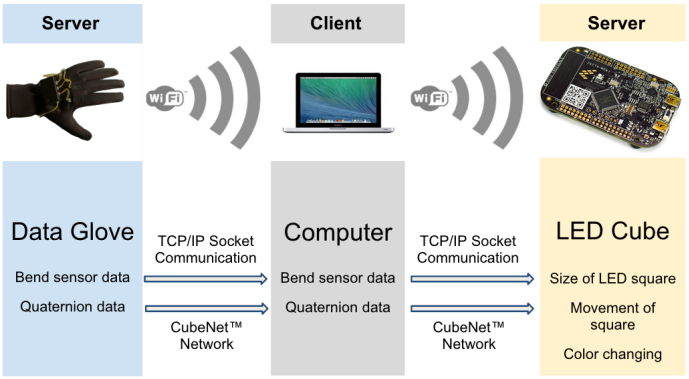

Fig. 1: Real-time TCP/IP Network

Sensor Data Specifications				
	Minimum	**Maximum**	**Precision**	**Units**
Finger Sensors	0	1000	1	ADC Units
Roll	-90.00	90.00	0.01	degrees
Pitch	-90.00	90.00	0.01	degrees
Yaw	-90.00	90.00	0.05	degrees

Fig. 2: Sensor Data Specifications

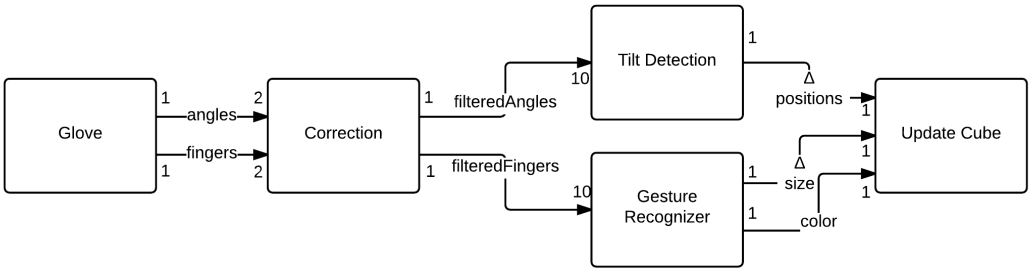

Fig. 3: Dataflow Diagram for CubicHand

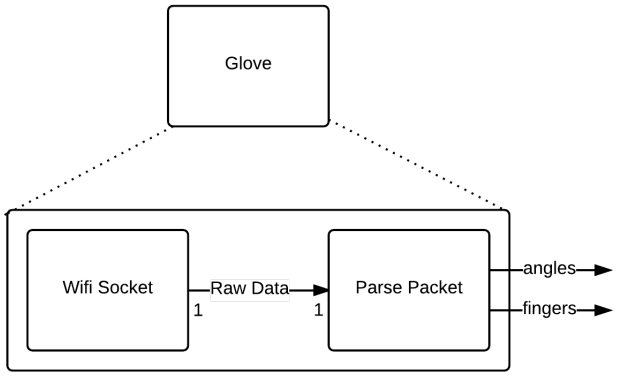

Fig. 4: Glove SDF

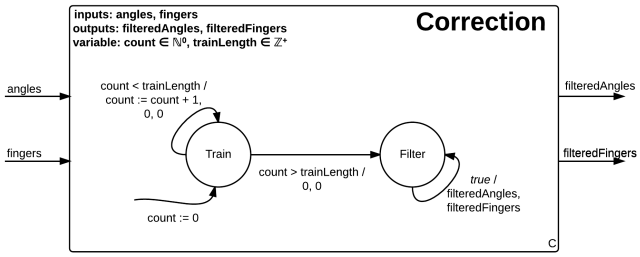

Fig. 5: Correction Actor

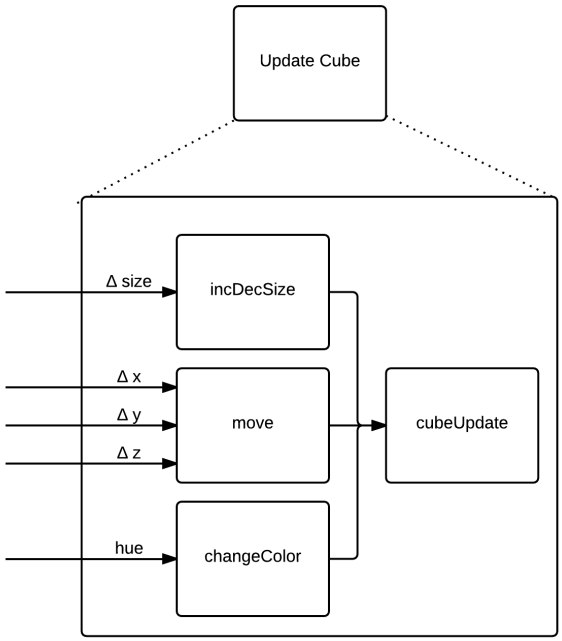

Fig. 6: Update Cube Actor

Gesture Recognition Thresholds			
	Unbent	**Bent**	**Units**
Fingers 0-3	< 350	≥ 350	ADC units
Finger 4 (Thumb)	< 200	≥ 200	ADC units
	Neg. Movement	**Pos. Movement**	**Units**
Roll, Pitch, Yaw	< -10	> 10	degrees

Fig. 7: Gesture Recognition Thresholds

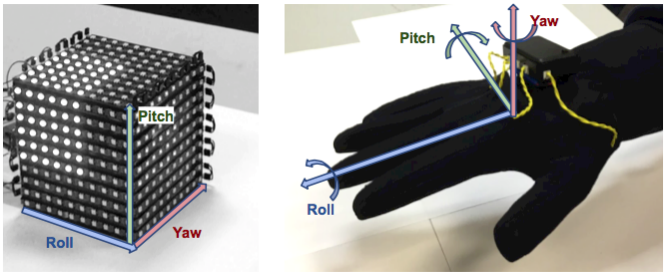

Fig. 8: Gestures based on Quaternion Data

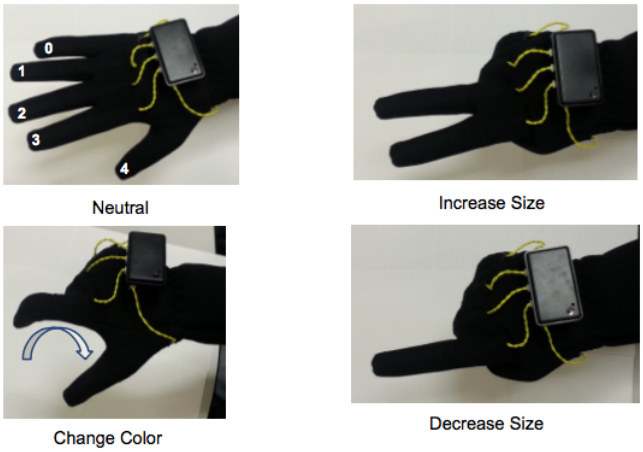

Fig. 9: Gestures based on Bend Sensors

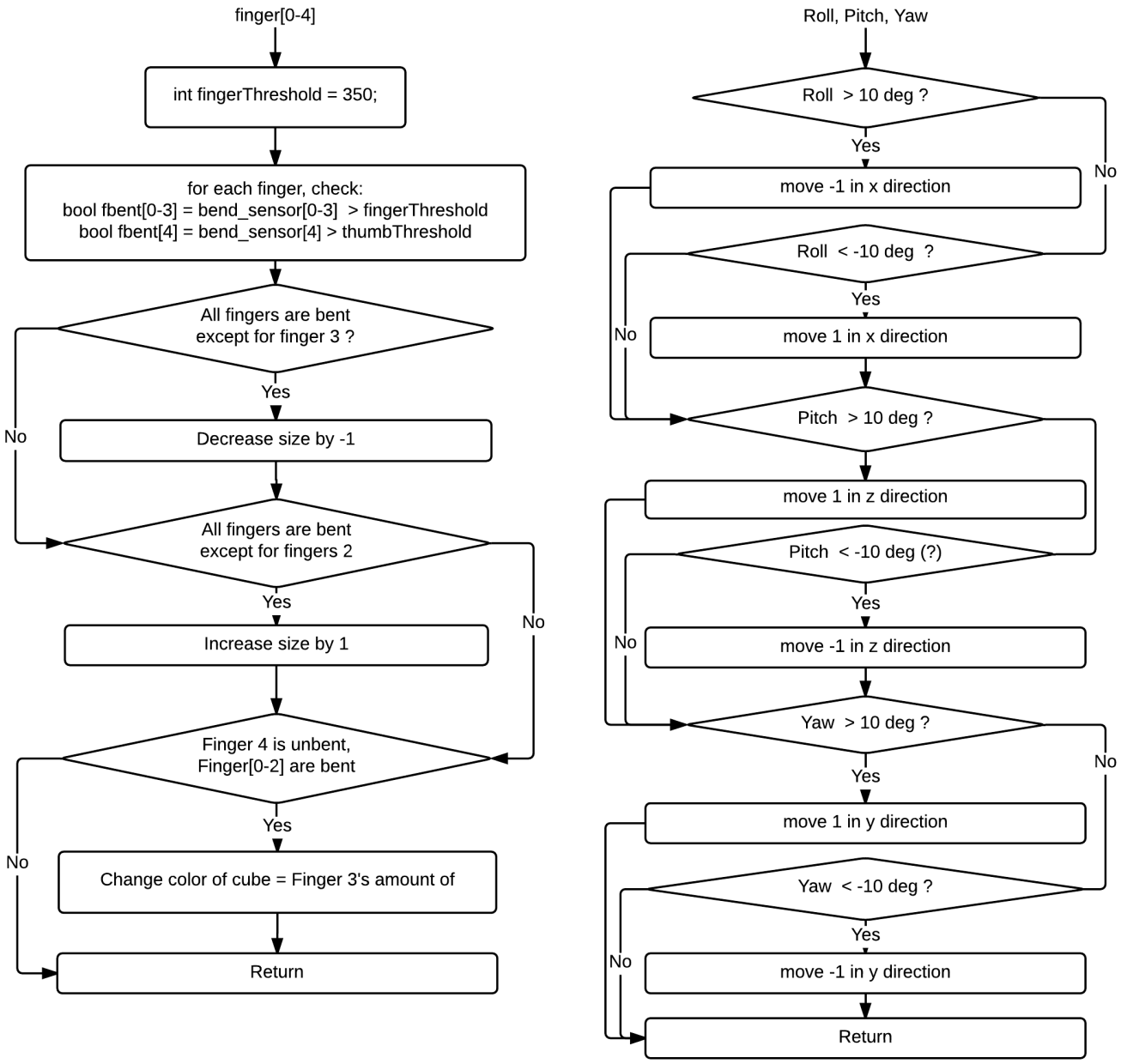

Fig. 10: Gesture Recognition Flow Diagram

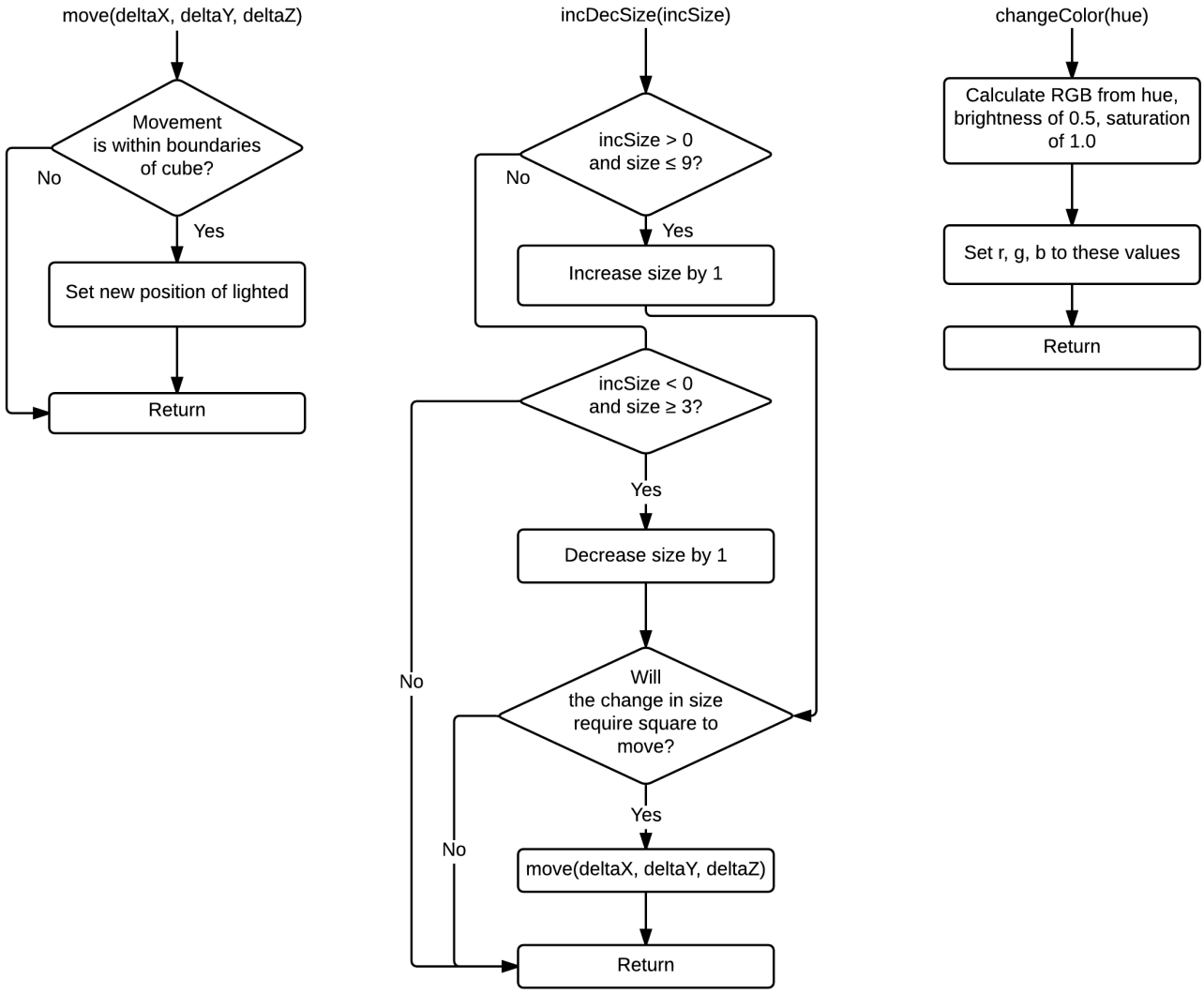

Fig. 11: Flow diagrams for Move, Size and Color

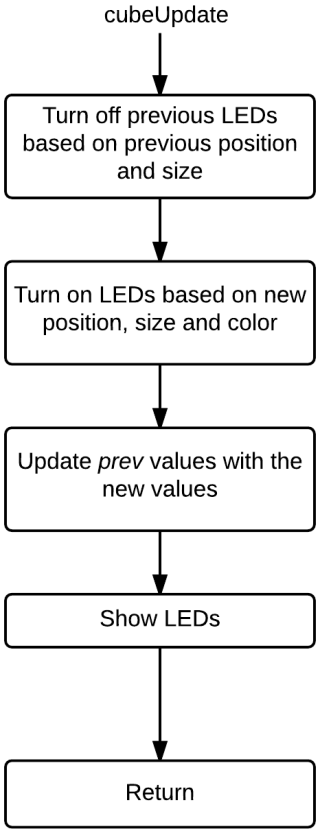

Fig. 12: CubeUpdate flow diagram

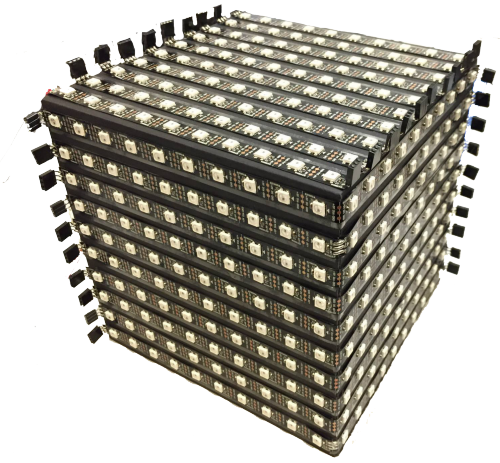

Fig. 13: The LED Cube