EE P 595A: TinyML
Lecture 6: Wizard Magic Wand

Dept. of Electrical and Computer Engineering
University of Washington

Instructor: Dinuka Sahabandu (sdinuka@uw.edu)

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

NSL

Topics Covered

- Introduction to Gesture Recognition and its Applications
- Gesture Recognition Techniques
- Key Components of a Gesture Recognition System
- Lab 6: Make a Magic Wand!
  - Data Collection and Labeling (crowd sourced by the class!)
  - Model Training
  - Evaluation Metrics
  - Deployment

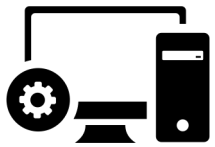
❖Relevant readings from the Textbook [Warden and Situnayake; O'REILLY Publisher] are Chapters 11 and 12

Introduction to Gesture Recognition

Gesture recognition is a type of perceptual computing user interface that allows computers to capture and interpret human gestures as commands.

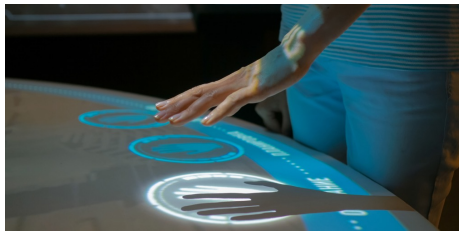
In what situations might gesture recognition be more convenient or practical than traditional input methods?

Gesture Recognition Applications



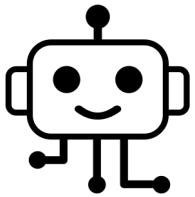
Human-Computer Interaction

- Touchless Interfaces
- Virtual and Augmented Reality
- Smart Home Control



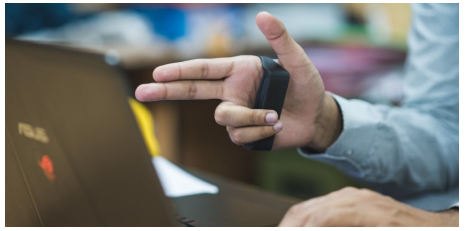
Gaming and Entertainment

- Motion-controlled Gaming
- Virtual Instruments
- Interactive Art Installations



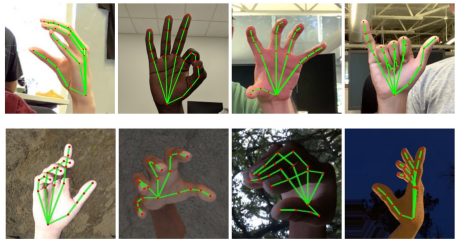
Robotics and Automation

- Gesture-based remote control
- Human-robot collaboration
- Sign language recognition

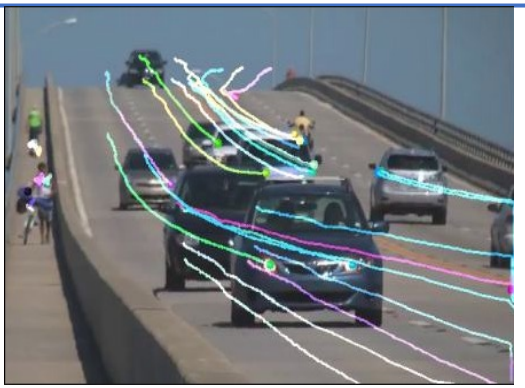


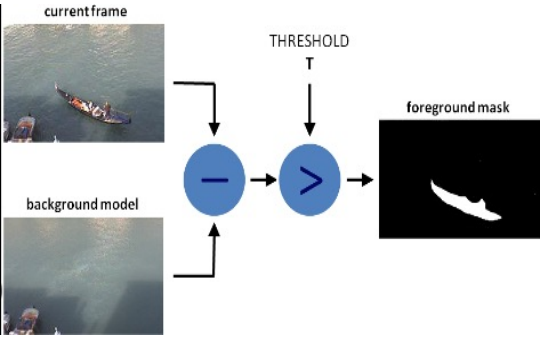
Accessibility and assistive technologies

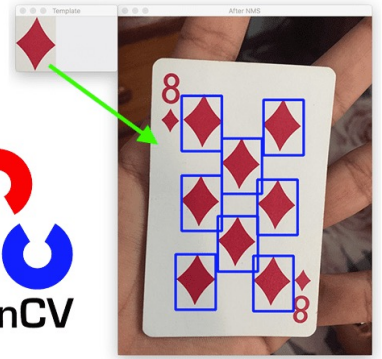
- Communication aids for people with speech or motor impairments
- Hands-free navigation for the visually impaired



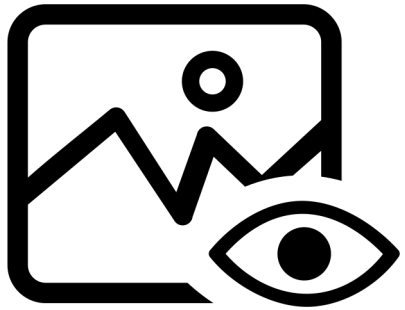
Gesture Recognition Techniques





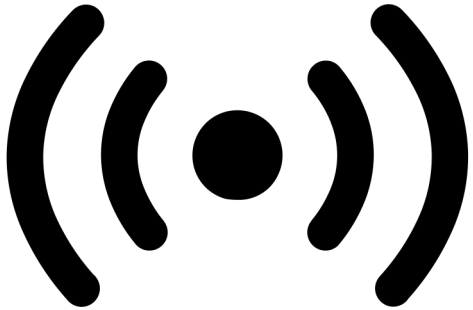


OpenCV



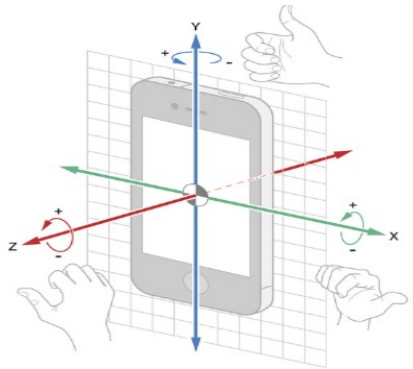
Vision-based Methods

Optical Flow
Background Subtraction
Template Matching

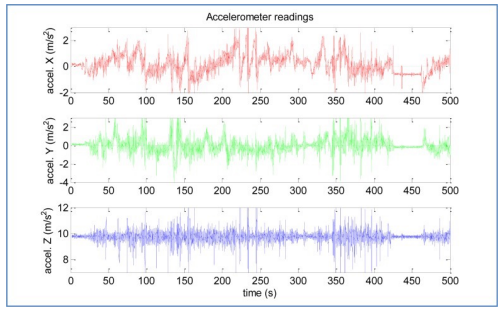


Sensor-based Methods

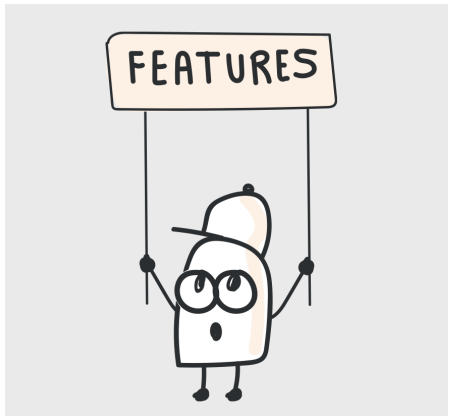
Data Gloves
Accelerometers and gyroscopes
Electromyography (EMG) sensors



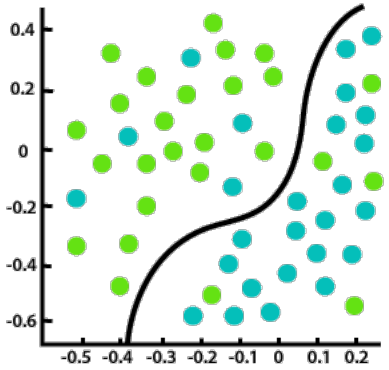
Key Components of a Gesture Recognition System



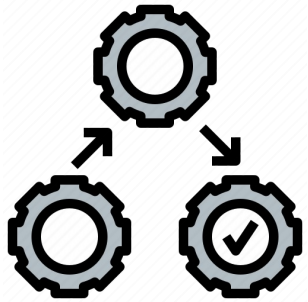
Data acquisition



Feature Extraction



Classification

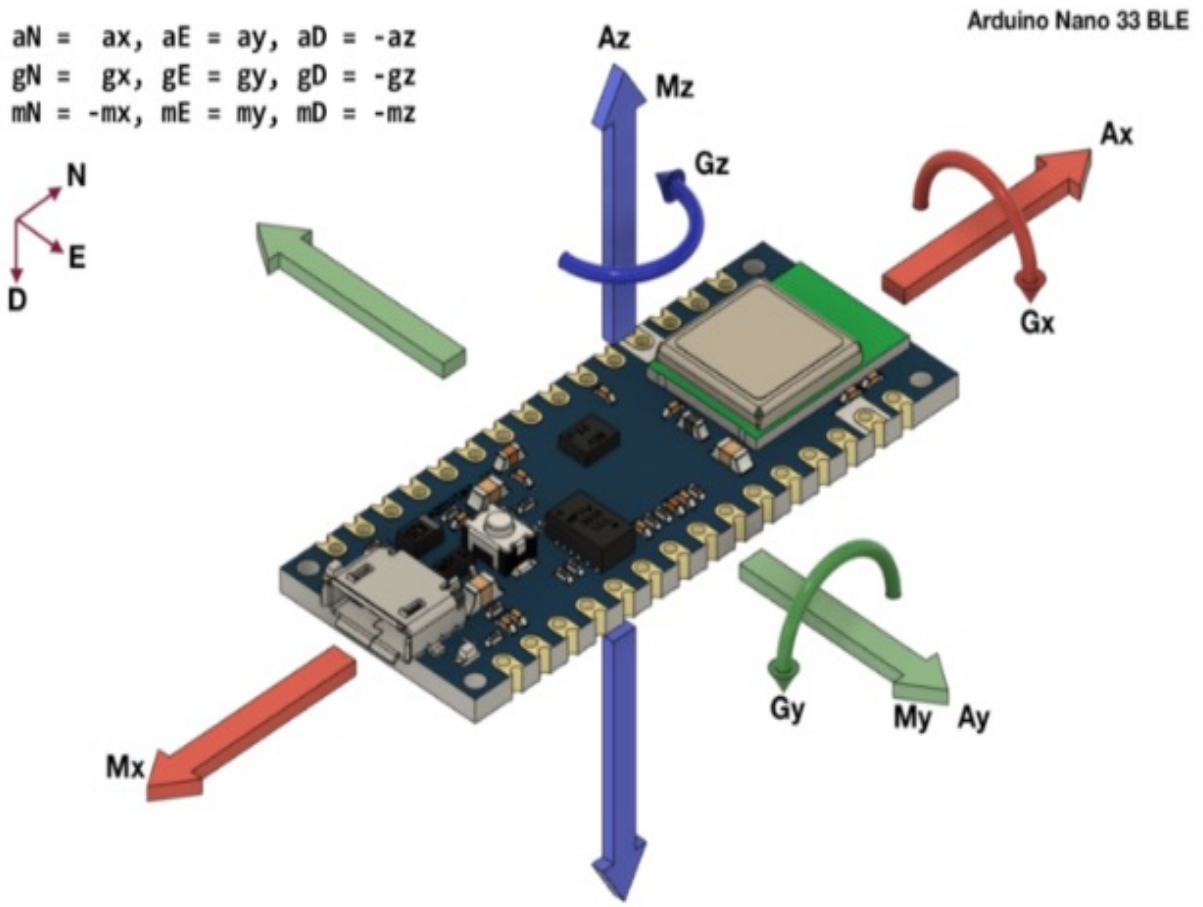


Post-Processing

Can we combine them together?

Now we take Magic Wand as as example

Inertial Measurement Unit (IMU)



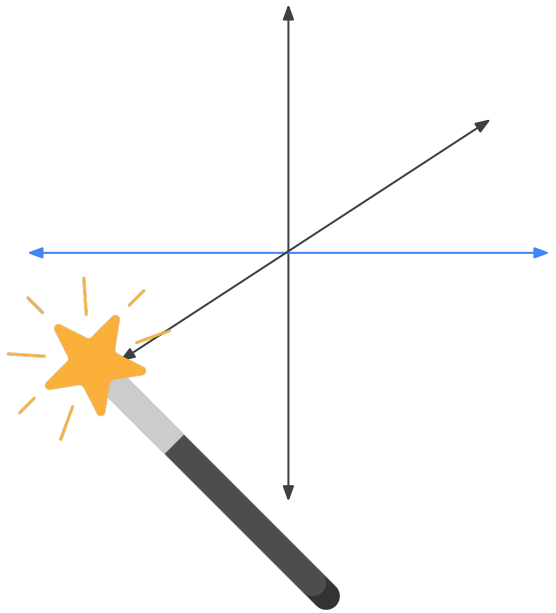
- **Accelerometer**: measures the linear acceleration along the x, y and z axes.
  Detects movements such as tilting or shaking

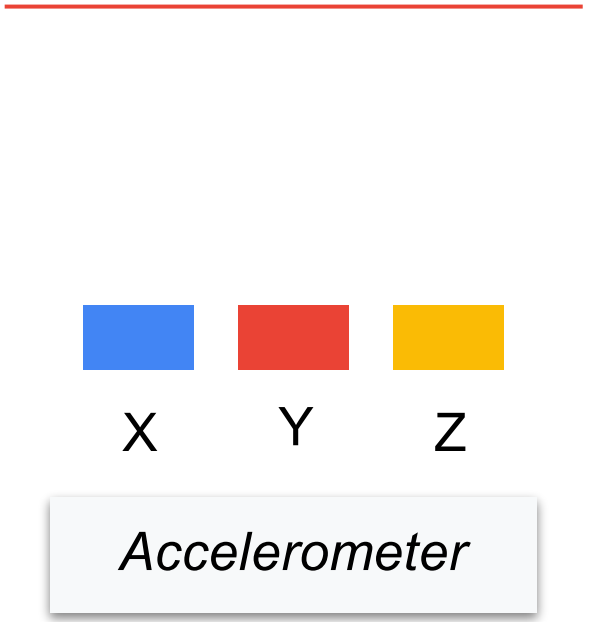
- **Gyroscope**: measures the rate of rotation around x, y and z axes, providing data about angular velocity.
  Detects complex gestures that involve twisting or rotating the device

- **Magnetometer:** measures the strength of magnetic fields in three dimensions.
  Helps in determining the orientation of the board relative to the Earth's magnetic field.

Capture the Gesture using Accelerometers

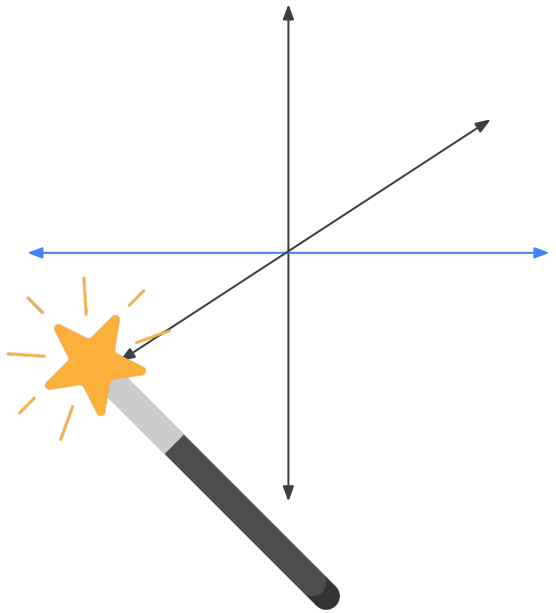
A simple movement

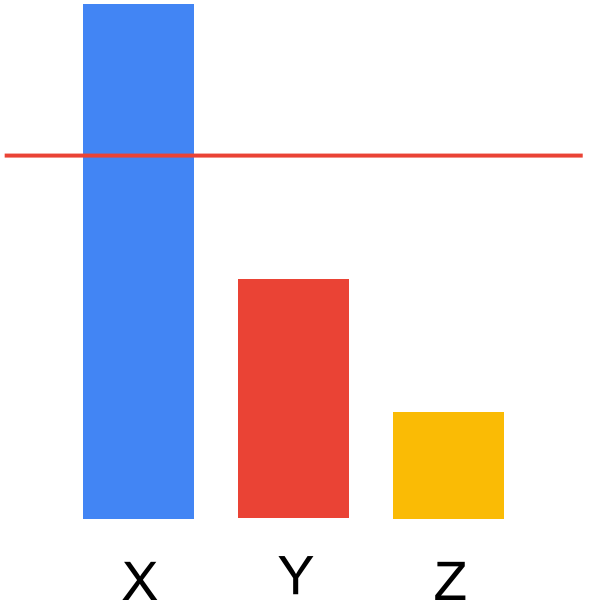




Capture the Gesture using Accelerometers

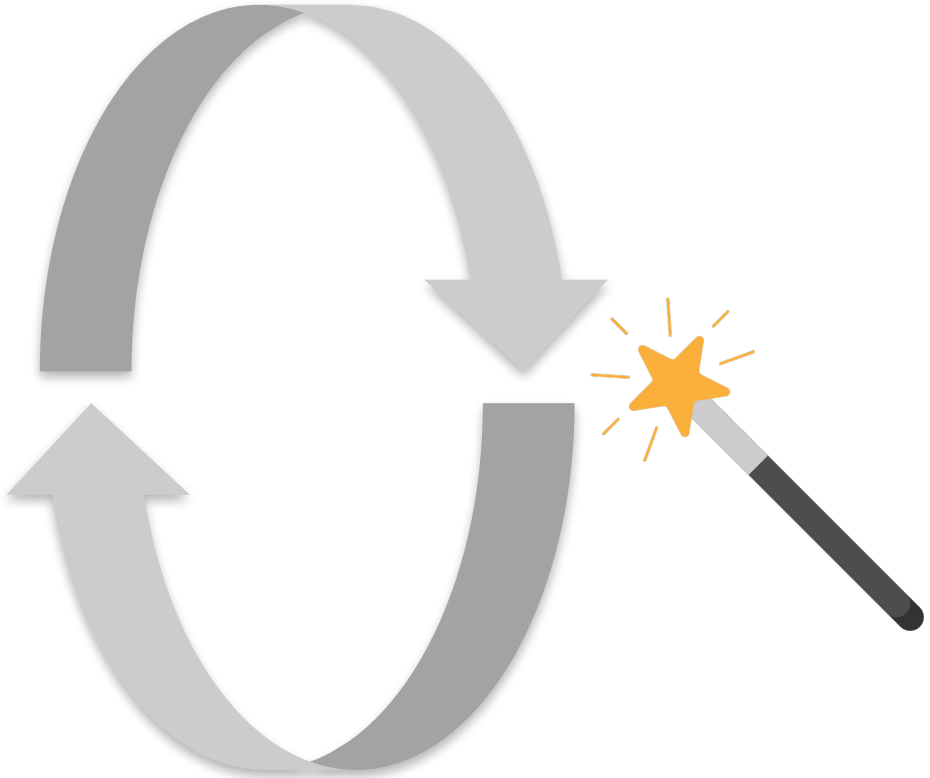
A simple movement

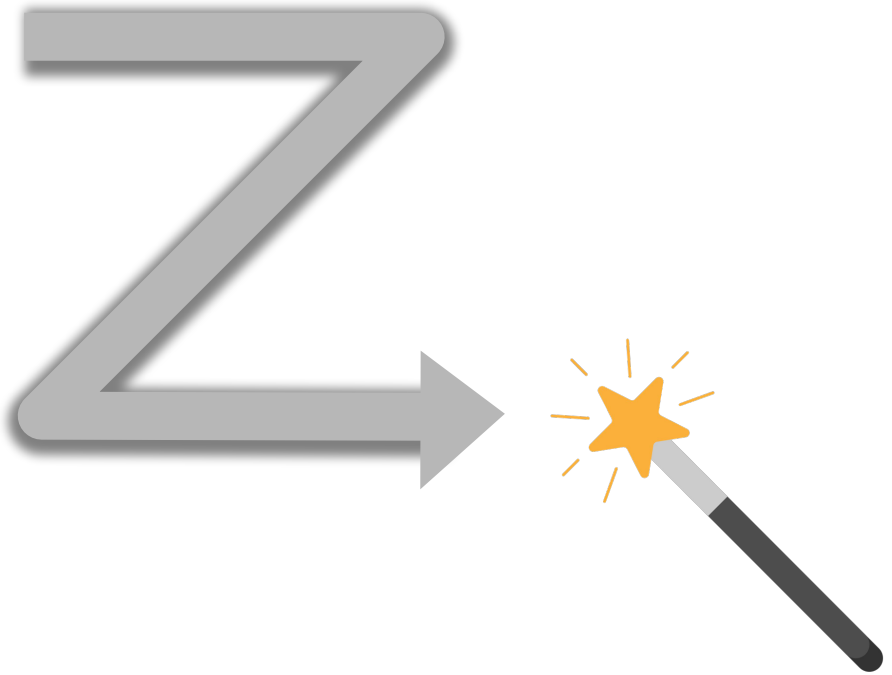




Capture the Gesture using Accelerometers

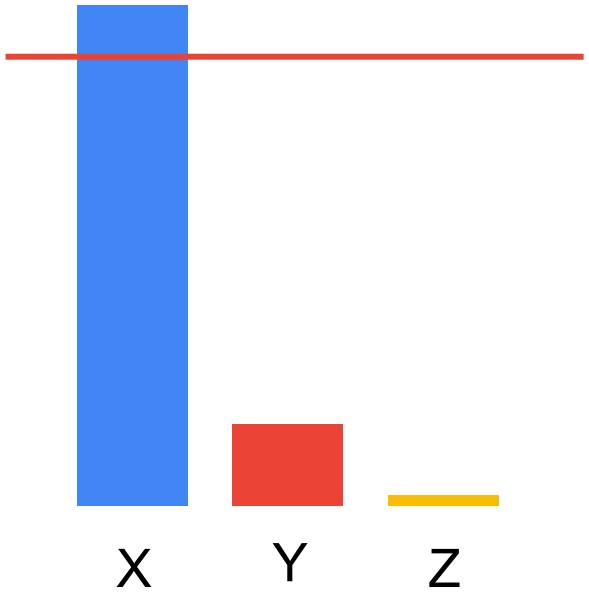
How about more complex movement?

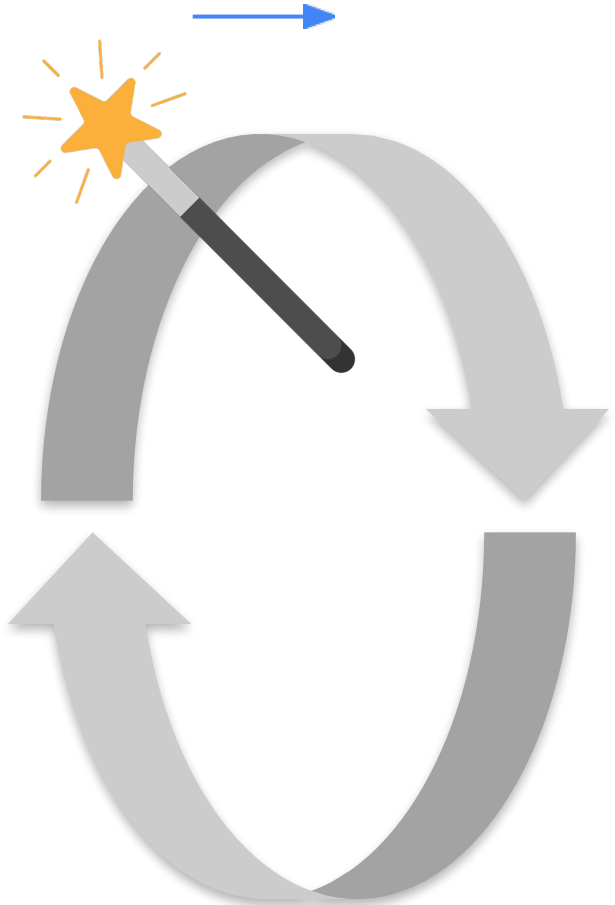




Capture the Gesture using Accelerometers

How about more complex movement: threshold sequence

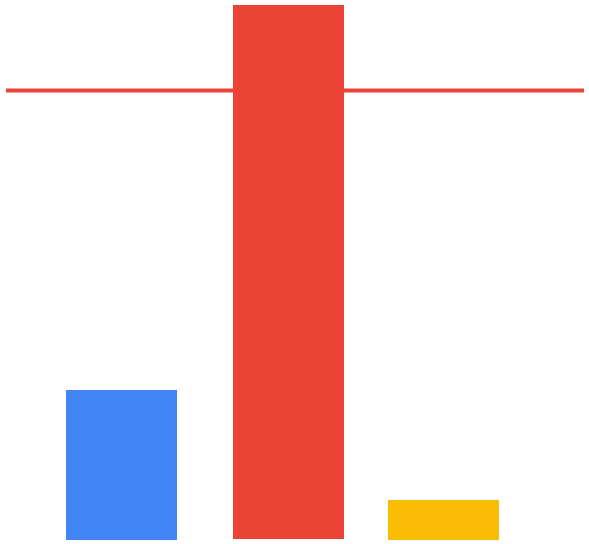


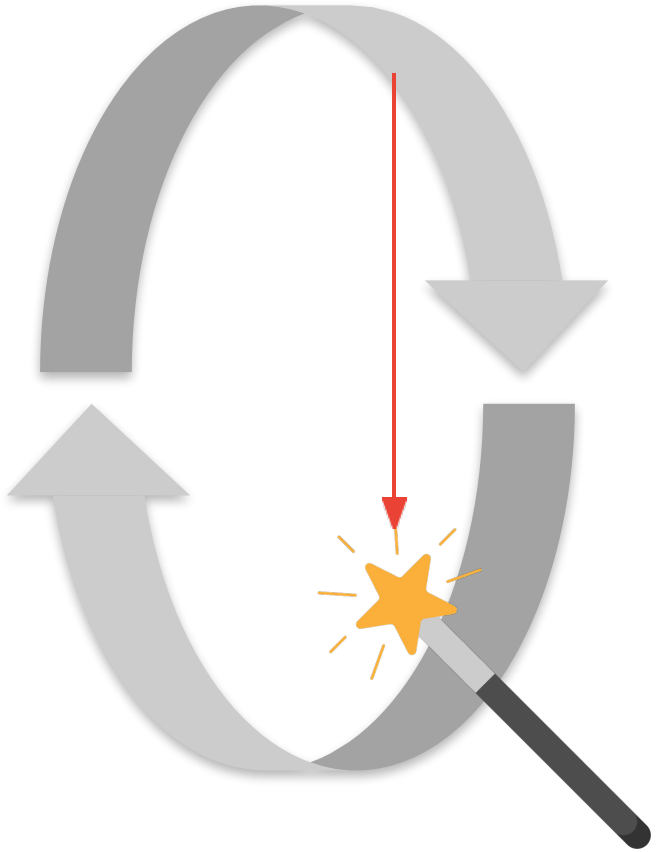


Sequence
→ X

Capture the Gesture using Accelerometers

How about more complex movement: threshold sequence

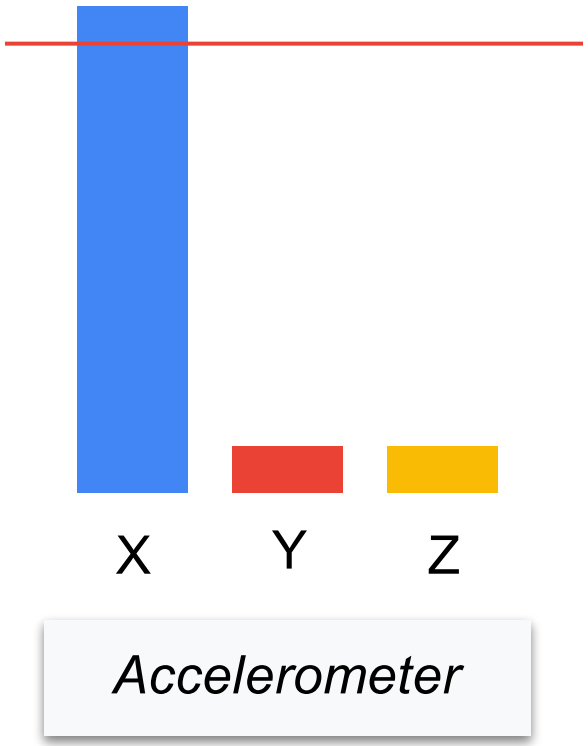


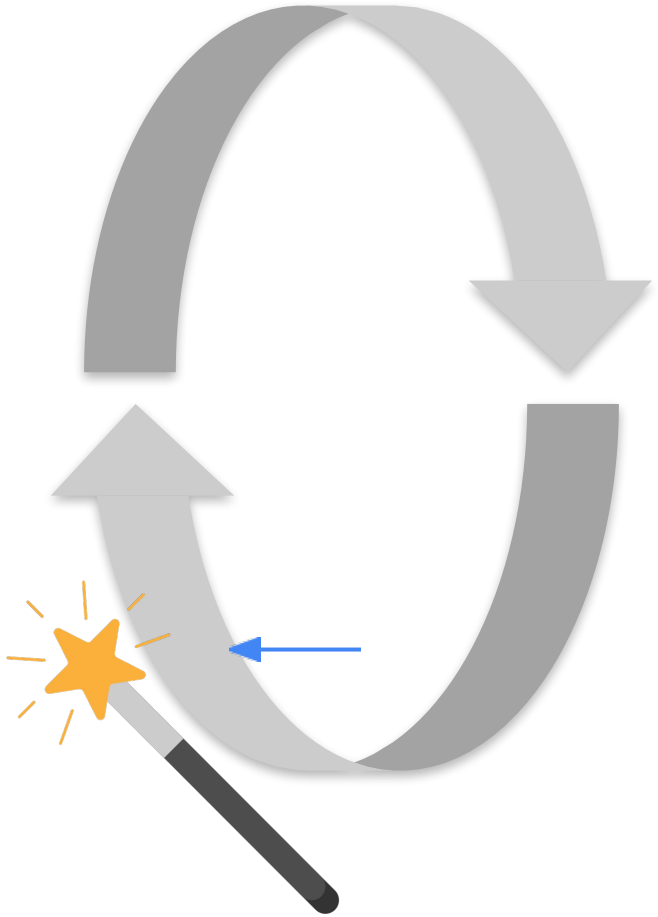


Sequence
X
↓ Y

Capture the Gesture using Accelerometers

How about more complex movement: threshold sequence

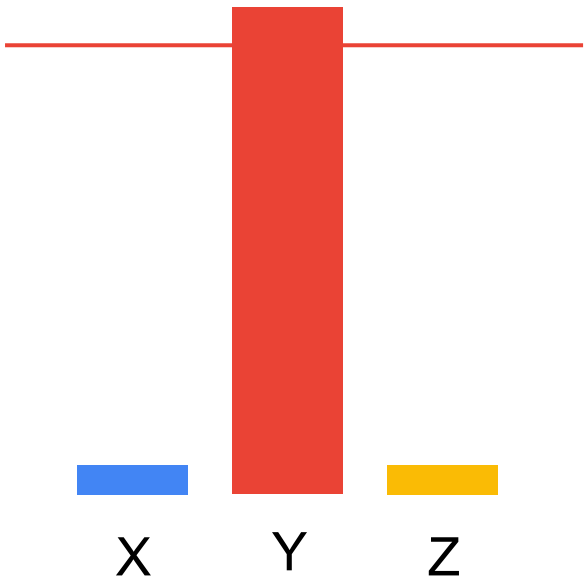


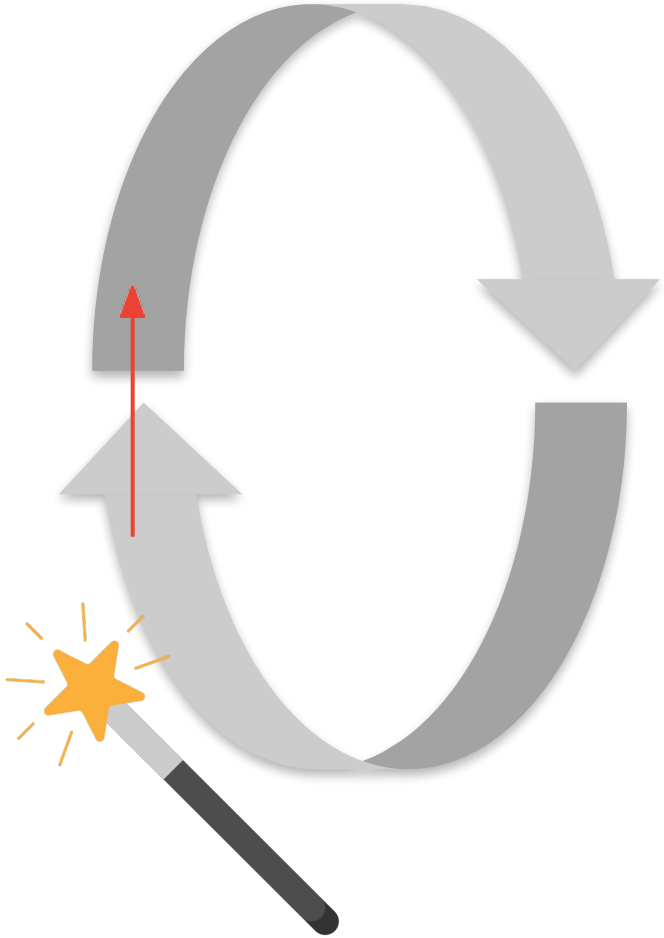


Sequence
X
Y
→ X

Capture the Gesture using Accelerometers

How about more complex movement: threshold sequence



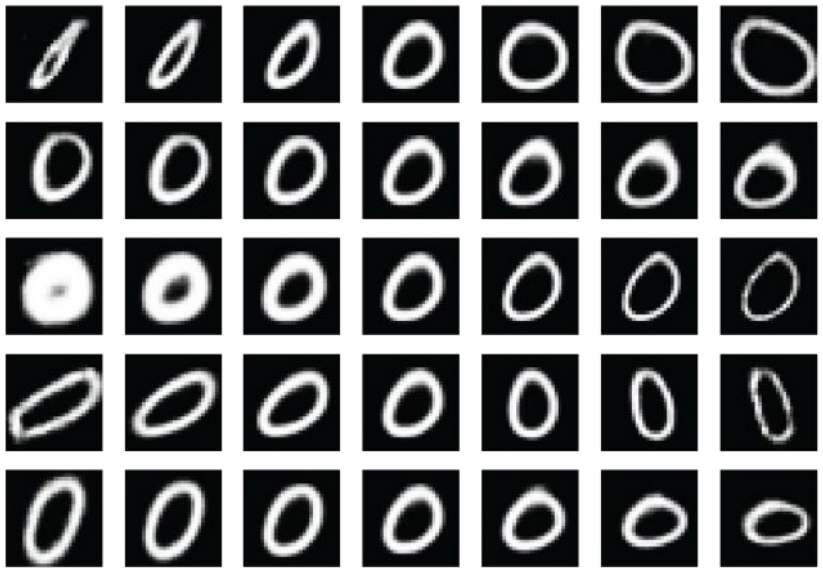


Sequence
X
Y
X
↓ Y

Capture the Gesture using Accelerometers

There are many ways to paint a picture - No drawing sequence is unique.

Can we make it robust to variations of movements and sequences?

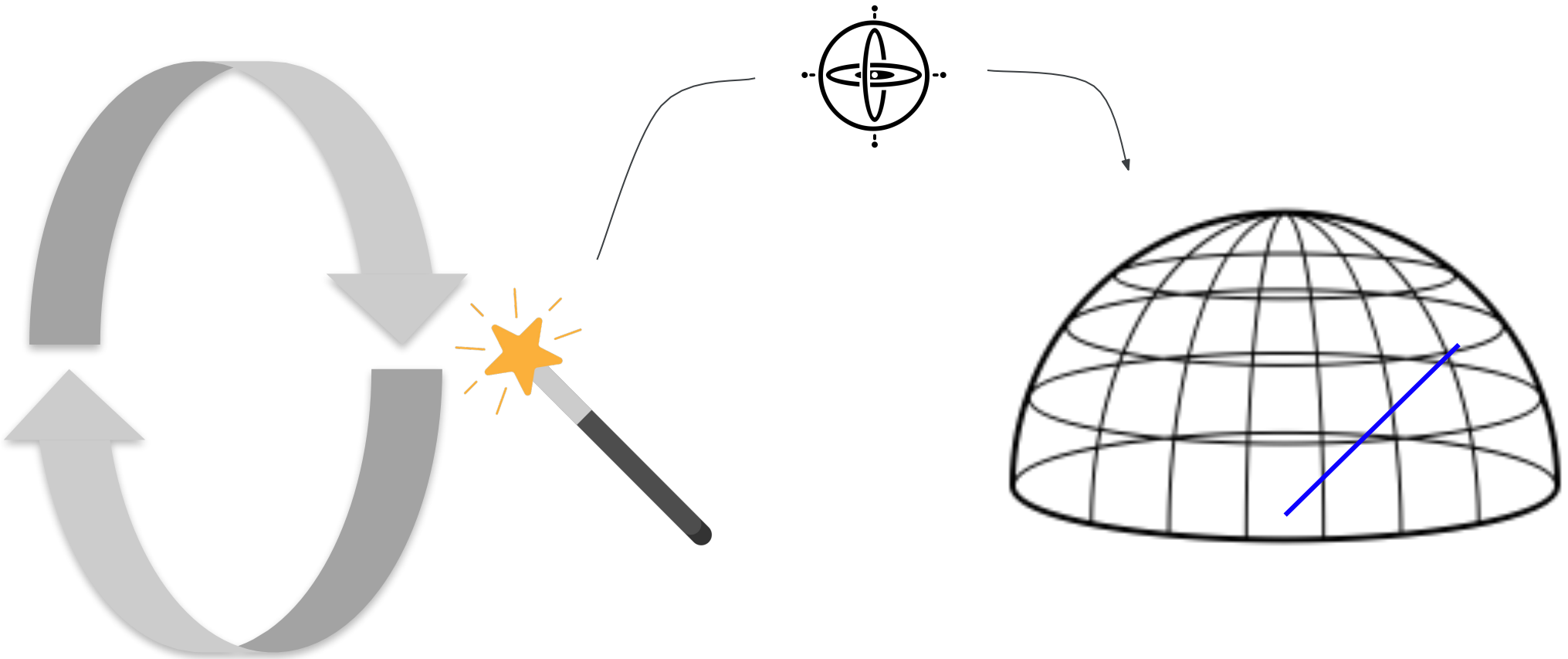


1st
Sequence
X
Y
X
Y

→

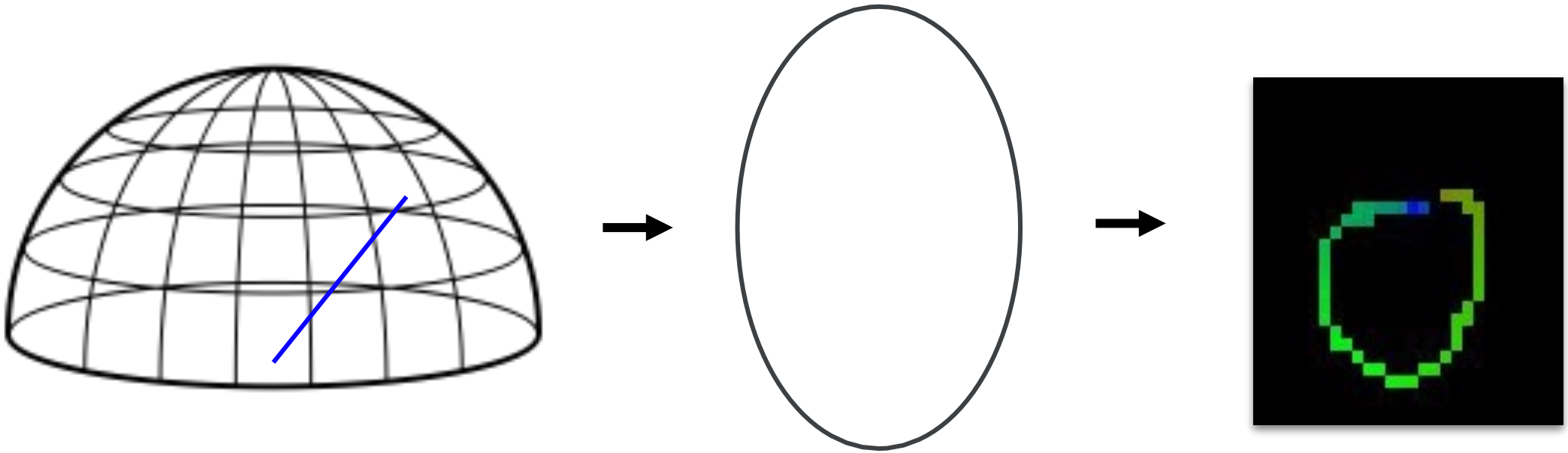
nth
Sequence
XY (?)
Y
X
XY (?)

Angular Position



But it is still hard to visualize and classify

Flatten and Rasterize



Flatten **Rasterize**

Flattening Example:
3D to 2D using Accelerometer and Gyroscope Data

- Accelerometer providing three components of acceleration: a_x, a_y, and a_z
- Gyroscope providing angular velocities ω_x, ω_y, and ω_z
- **Objective**: Adjust the 3D accelerometer data to account for rotations and then flatten this data onto the XY plane.

- **Step 1**: Compute the cumulative rotation angles for each axis over a time interval Δt:

$$\theta_x = \omega_x \times \Delta t$$
$$\theta_y = \omega_y \times \Delta t$$
$$\theta_z = \omega_z \times \Delta t$$

Flattening Example: 3D to 2D using Accelerometer and Gyroscope Data

- **Step 2**: To adjust for rotations around all three axes, construct a composite rotation matrix from the individual rotation matrices

Rotation Matrix around X-axis $R_x(\theta_x)$:

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix}$$

Rotation Matrix around Y-axis $R_y(\theta_y)$:

$$R_y(\theta_y) = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_x) \end{bmatrix}$$

Rotation Matrix around Z-axis $R_z(\theta_z)$:

$$R_z(\theta_z) = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Flattening Example:
3D to 2D using Accelerometer and Gyroscope Data

- **Step 2 (cont.)**: Composite the matrices (assuming rotations are small or order does not introduce significant error):

$$R = R_z(\theta_z) \times R_y(\theta_y) \times R_x(\theta_x)$$

- **Step 3:** Adjust the accelerometer data by applying the inverse of the composite rotation matrix to revert to the original reference frame:

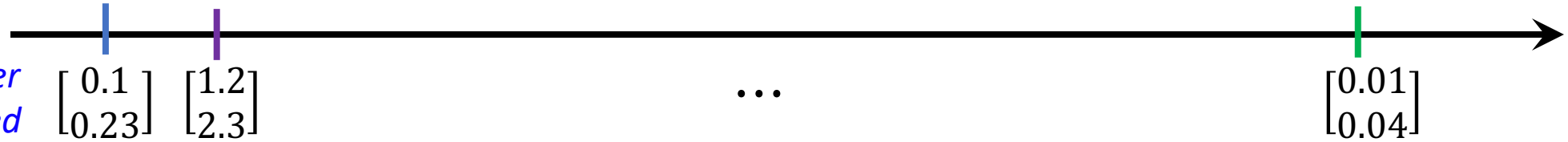
$$\begin{bmatrix} a'_x \\ a'_y \\ a'_z \end{bmatrix} = R^{-1} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

- **Step 4**: To flatten the adjusted 3D data onto the XY plane, simply ignore the Z-component

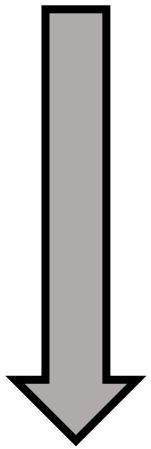
$$\begin{bmatrix} a'_x \\ a'_y \\ a'_z \end{bmatrix} \Rightarrow \begin{bmatrix} a'_x \\ a'_y \end{bmatrix}$$

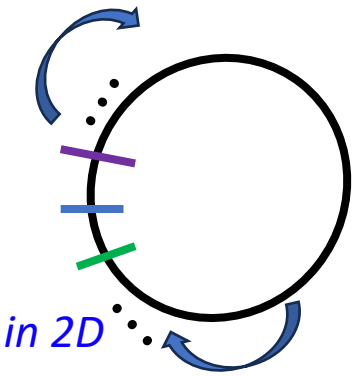
Accelerometer Data flattened to XY Plane !!!

2D Accelerometer Data to 2D Gesture



Accelerometer Data flattened to XY Plane





Gesture in 2D

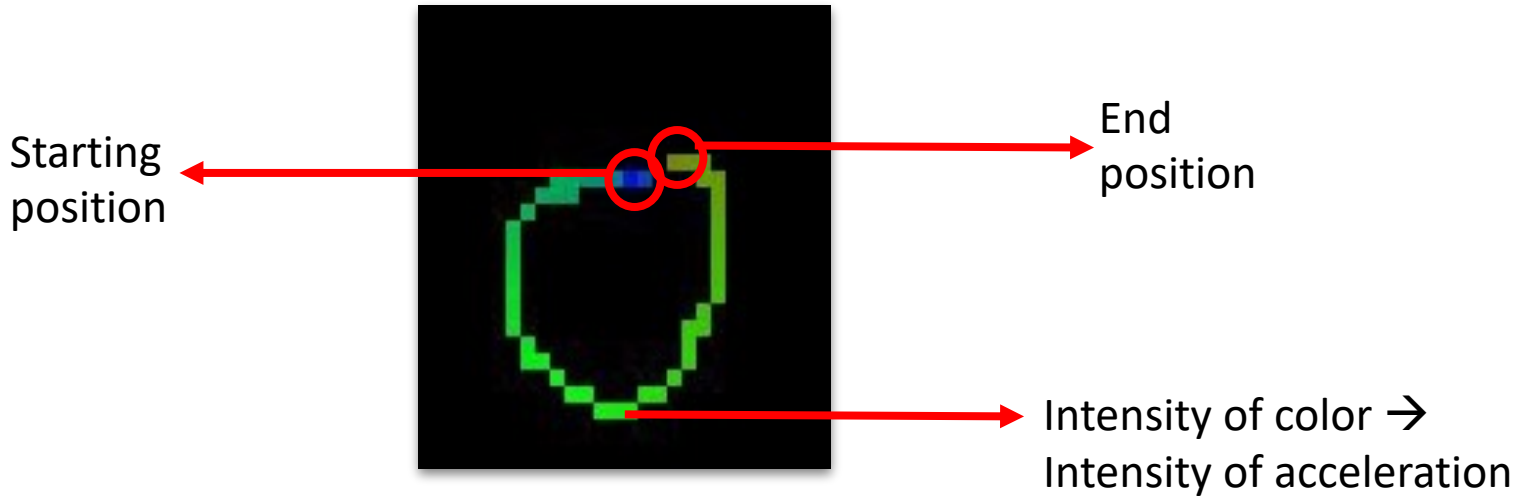
$$\mathbf{v} = \mathbf{a}t + \mathbf{v}_0$$
$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{a}t^2$$
$$\mathbf{r} = \mathbf{r}_0 + \frac{1}{2}(\mathbf{v} + \mathbf{v}_0)\,t$$
$$\mathbf{v}^2 = \mathbf{v}_0^2 + 2\mathbf{a}\cdot(\mathbf{r} - \mathbf{r}_0)$$
$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}t - \frac{1}{2}\mathbf{a}t^2$$

Equations of motion

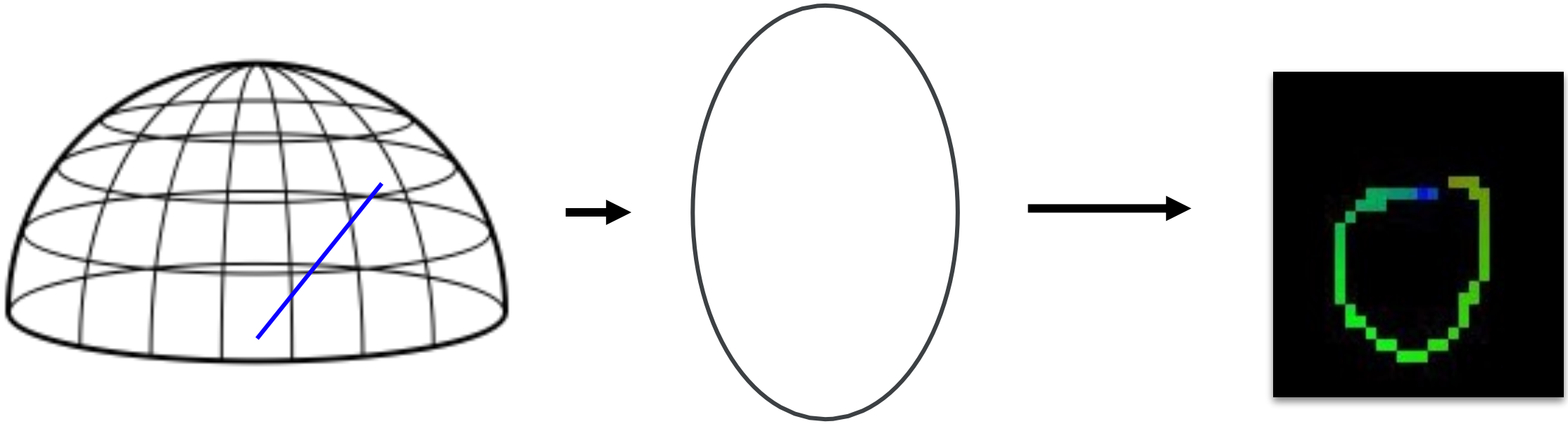
a: acceleration t: time
v: velocity v_0: initial velocity (set to zero)
r: position r_0: initial position

Rasterization

- Convert the gesture data into a pixel-based image format
- The flattened gesture (in 2D), is transformed into a grid of pixels where the presence and path of the gesture are marked
- Useful when using image processing techniques or convolutional neural networks (CNNs)
- Easier to standardize and normalize compared to raw sensor data, which might vary greatly depending on the sensor's calibration and environment



Flatten and Rasterize: Benefits



Flatten

Makes the data easier to handle and process.

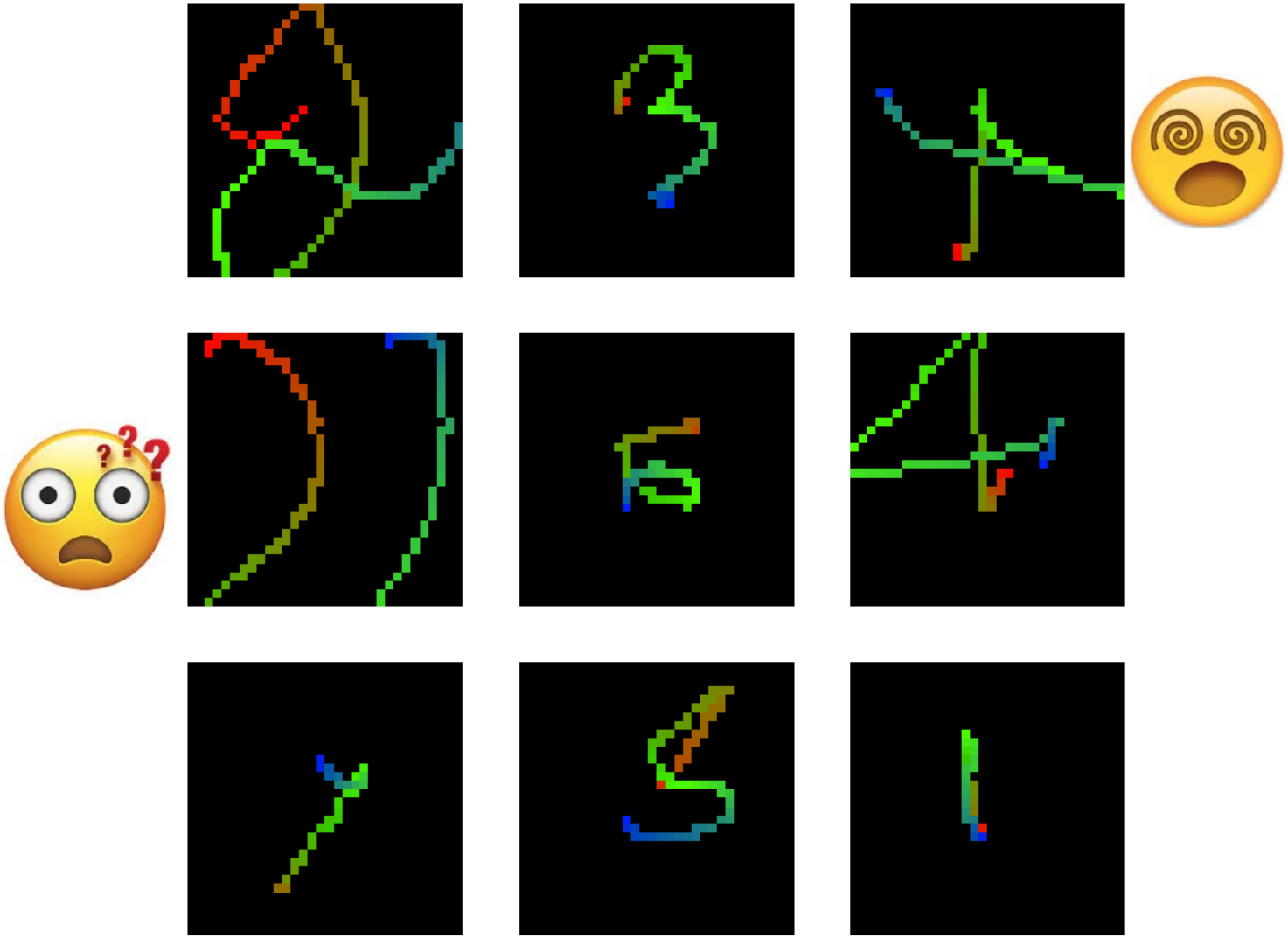
Rasterize

- Converting gesture data into an image format makes it possible to leverage computer vision models (e.g., CNNs) for gesture recognition.
- Easier to standardize and normalize.

Sample of Data Collected by You: Raw Data

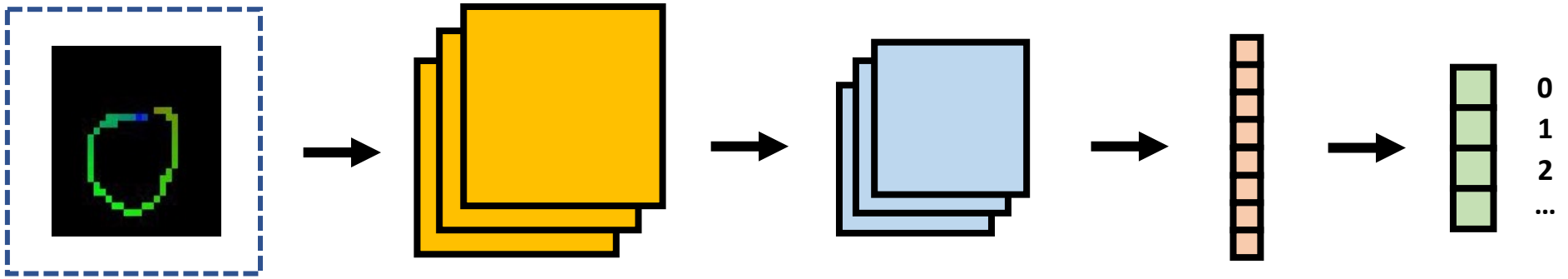

```
{"strokes":[{"index":0,"strokePoints":[{"x":0.1875,"y":0},{"x":0.1875,"y":0},{"x":0.1875,"y":0},
{"x":0.1796875,"y":0},{"x":0.1796875,"y":0.0078125},{"x":0.171875,"y":0.0078125},
{"x":0.171875,"y":0.015625},{"x":0.171875,"y":0.0234375},{"x":0.171875,"y":0.03125},
{"x":0.171875,"y":0.03125},{"x":0.171875,"y":0.0390625},{"x":0.171875,"y":0.046875},
{"x":0.171875,"y":0.046875},{"x":0.171875,"y":0.046875},{"x":0.171875,"y":0.046875},
{"x":0.1640625,"y":0.0546875},{"x":0.15625,"y":0.0625},{"x":0.1484375,"y":0.078125},
{"x":0.1484375,"y":0.0859375},{"x":0.1484375,"y":0.09375},{"x":0.1484375,"y":0.1015625},
{"x":0.1484375,"y":0.1015625},{"x":0.1484375,"y":0.1015625},{"x":0.140625,"y":0.109375},
{"x":0.140625,"y":0.109375},{"x":0.140625,"y":0.1015625},{"x":0.140625,"y":0.0859375},
{"x":0.1328125,"y":0.078125},{"x":0.125,"y":0.0625},{"x":0.1171875,"y":0.0546875},
{"x":0.109375,"y":0.046875},{"x":0.109375,"y":0.046875},{"x":0.109375,"y":0.0546875},
{"x":0.1171875,"y":0.0625},{"x":0.125,"y":0.0703125},{"x":0.1328125,"y":0.0703125},
{"x":0.140625,"y":0.0625},{"x":0.140625,"y":0.046875},{"x":0.1484375,"y":0.0390625},
{"x":0.1484375,"y":0.0234375},{"x":0.1484375,"y":0.015625},{"x":0.1484375,"y":0.0078125},
{"x":0.1484375,"y":0.0078125},{"x":0.1484375,"y":0.0078125},{"x":0.15625,"y":0.0078125},
{"x":0.15625,"y":0.0078125},{"x":0.1640625,"y":0.0078125},{"x":0.1640625,"y":0.015625},
{"x":0.15625,"y":0.0234375},{"x":0.1484375,"y":0.03125},{"x":0.140625,"y":0.0390625},
{"x":0.125,"y":0.0390625},{"x":0.109375,"y":0.0390625},{"x":0.09375,"y":0.046875},
{"x":0.0703125,"y":0.046875},{"x":0.046875,"y":0.0546875},{"x":0.03125,"y":0.0546875},
{"x":0.0078125,"y":0.0625},{"x":-0.0078125,"y":0.0625},{"x":-0.03125,"y":0.0625},
{"x":-0.0625,"y":0.0625},{"x":-0.0859375,"y":0.0625},{"x":-0.109375,"y":0.0546875},
{"x":-0.125,"y":0.046875},{"x":-0.1484375,"y":0.03125},{"x":-0.171875,"y":0.0234375},
{"x":-0.1875,"y":0.015625},{"x":-0.203125,"y":0.0078125},{"x":-0.21875,"y":-0.0078125},
{"x":-0.2265625,"y":-0.015625},{"x":-0.234375,"y":-0.03125},{"x":-0.2421875,"y":-0.046875},
{"x":-0.2421875,"y":-0.0625},{"x":-0.2421875,"y":-0.078125},{"x":-0.2421875,"y":-0.1015625},
{"x":-0.234375,"y":-0.1171875},{"x":-0.21875,"y":-0.1328125},{"x":-0.2109375,"y":-0.140625},
{"x":-0.1953125,"y":-0.15625},{"x":-0.1796875,"y":-0.171875},{"x":-0.15625,"y":-0.1796875},
{"x":-0.140625,"y":-0.1953125},{"x":-0.1171875,"y":-0.203125},{"x":-0.0859375,"y":-0.2109375},
{"x":-0.0625,"y":-0.21875},{"x":-0.0390625,"y":-0.2265625},{"x":-0.0078125,"y":-0.2265625},
{"x":0.0234375,"y":-0.2265625},{"x":0.046875,"y":-0.21875},{"x":0.0703125,"y":-0.203125},
{"x":0.0859375,"y":-0.1875},{"x":0.1015625,"y":-0.171875},{"x":0.1171875,"y":-0.15625},
{"x":0.1171875,"y":-0.1328125},{"x":0.1171875,"y":-0.109375},{"x":0.1171875,"y":-0.0859375},
{"x":0.109375,"y":-0.0703125},{"x":0.1015625,"y":-0.046875},{"x":0.0859375,"y":-0.03125},
{"x":0.0703125,"y":-0.015625},{"x":0.046875,"y":0.0078125},{"x":0.03125,"y":0.0234375},
{"x":0.0078125,"y":0.0390625},{"x":-0.0078125,"y":0.0546875},{"x":-0.0234375,"y":0.0703125},
{"x":-0.046875,"y":0.0859375},{"x":-0.0703125,"y":0.09375},{"x":-0.1015625,"y":0.09375},
{"x":-0.1328125,"y":0.0859375},{"x":-0.15625,"y":0.078125},{"x":-0.1796875,"y":0.0625},
{"x":-0.1953125,"y":0.0546875},{"x":-0.203125,"y":0.046875},{"x":-0.203125,"y":0.0390625},
{"x":-0.203125,"y":0.03125},{"x":-0.203125,"y":0.0234375},{"x":-0.1953125,"y":0.015625},
{"x":-0.1953125,"y":0.0078125},{"x":-0.1953125,"y":0.0078125},{"x":-0.1875,"y":0},
{"x":-0.1875,"y":0},{"x":-0.1796875,"y":0},{"x":-0.1796875,"y":0},{"x":-0.1796875,"y":0.0078125},
{"x":-0.1796875,"y":0.0078125},{"x":-0.1796875,"y":0.0078125},{"x":-0.1796875,"y":0.0078125},
{"x":-0.1796875,"y":0.015625},{"x":-0.1796875,"y":0.015625},{"x":-0.1796875,"y":0.015625},
{"x":-0.1796875,"y":0.015625},{"x":-0.1796875,"y":0.0234375},
{"x":-0.1796875,"y":0.0234375}],"label":"0"},{"index":2,"strokePoints":[{"x":0,"y":-0.078125},
{"x":-0.0078125,"y":-0.078125},{"x":-0.0078125,"y":-0.078125},{"x":-0.0078125,"y":-0.078125},
{"x":-0.015625,"y":-0.078125},{"x":-0.015625,"y":-0.078125},{"x":-0.015625,"y":-0.078125},
{"x":-0.0234375,"y":-0.078125},{"x":-0.0234375,"y":-0.078125},{"x":-0.03125,"y":-0.078125},
{"x":-0.03125,"y":-0.078125},{"x":-0.0390625,"y":-0.078125},{"x":-0.0390625,"y":-0.078125},
{"x":-0.046875,"y":-0.078125},{"x":-0.0546875,"y":-0.0703125},{"x":-0.0625,"y":-0.0703125},
```

Sample of Data Collected by You: Rasterize Data



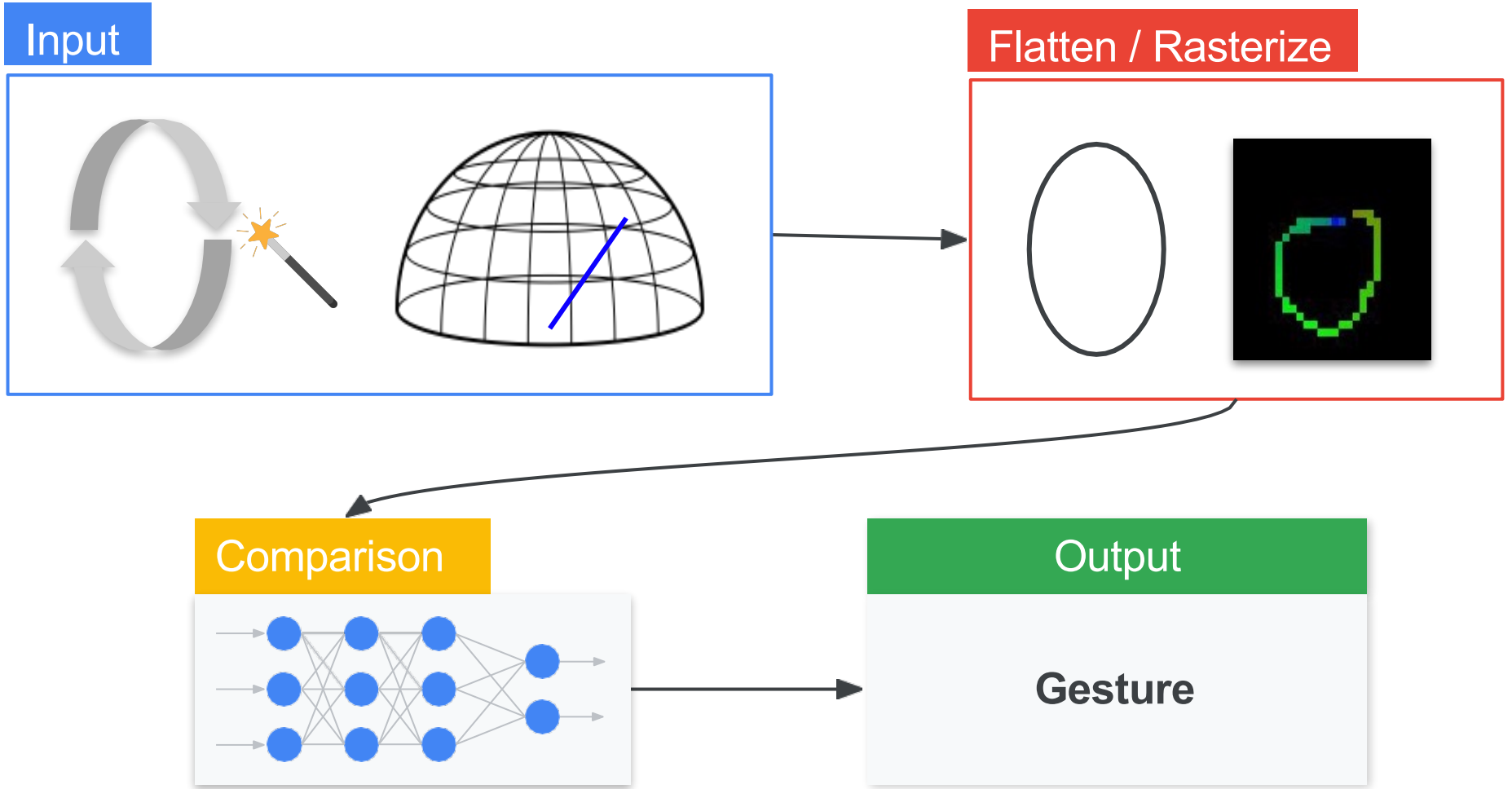
CNN for Magic Wand Sketch

Convolutional Neural Networks (CNNs)

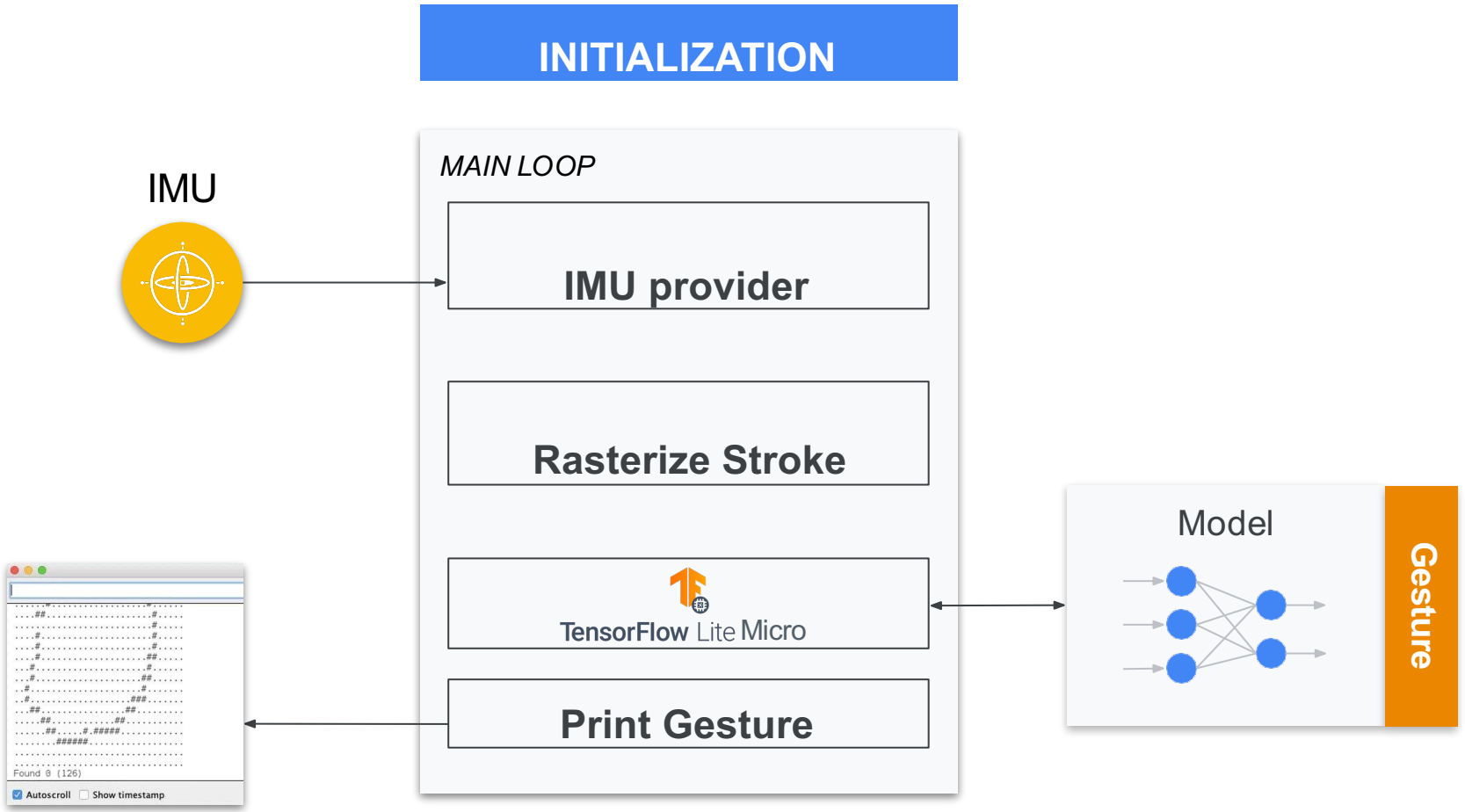


Input Layer (Rasterize Image) | Convolution Layer | Pooling Layer | Dense Layer | Output Layer

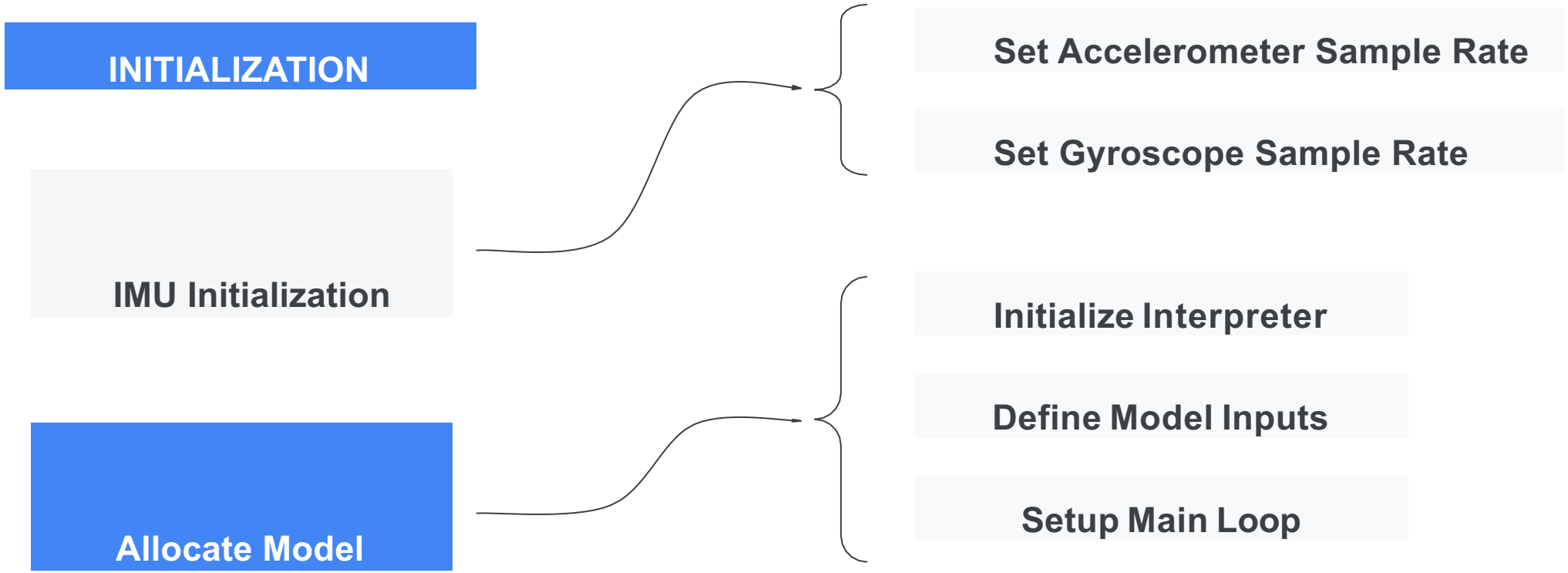
Final Robust Design



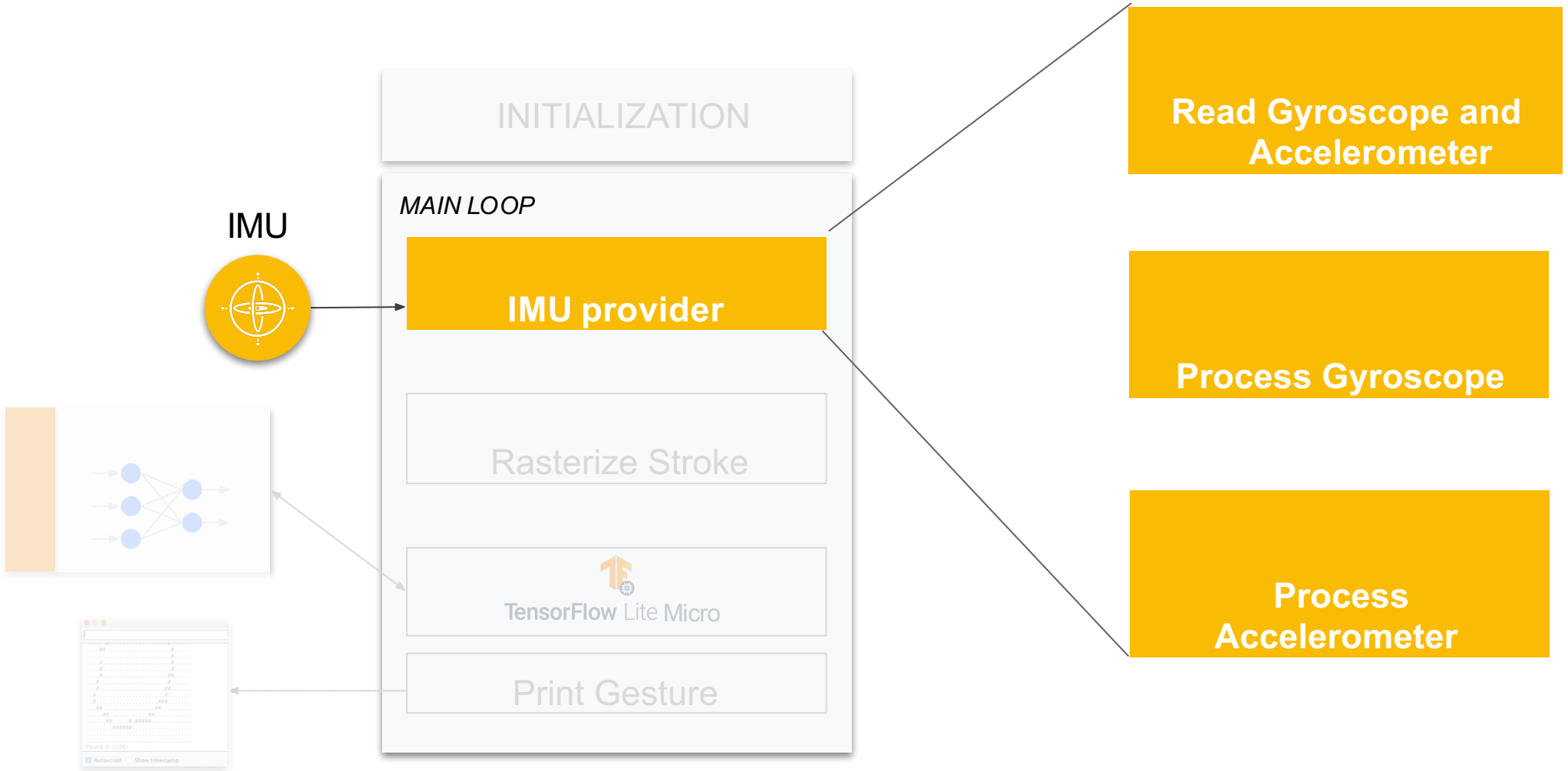
Deconstructing Magic Wand Application



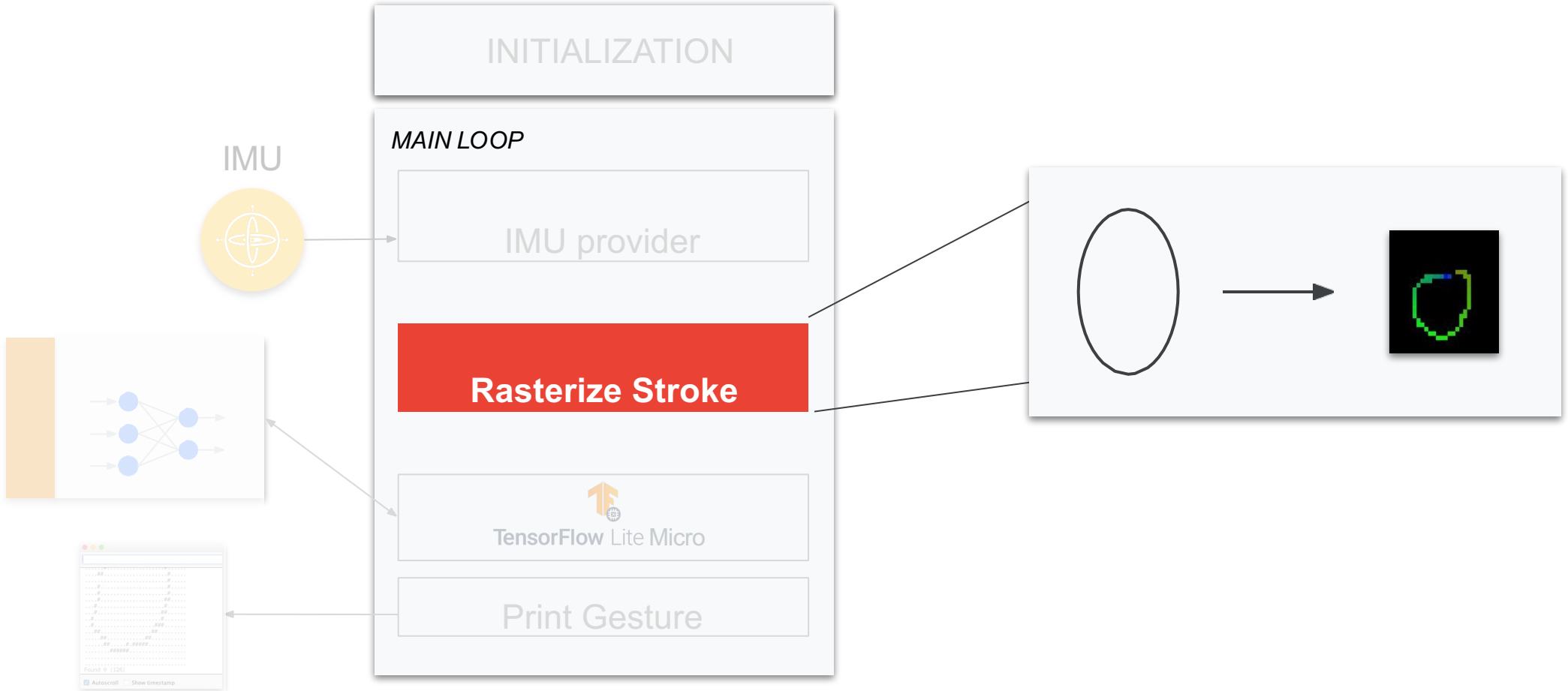
Magic Wand Application - Initiation



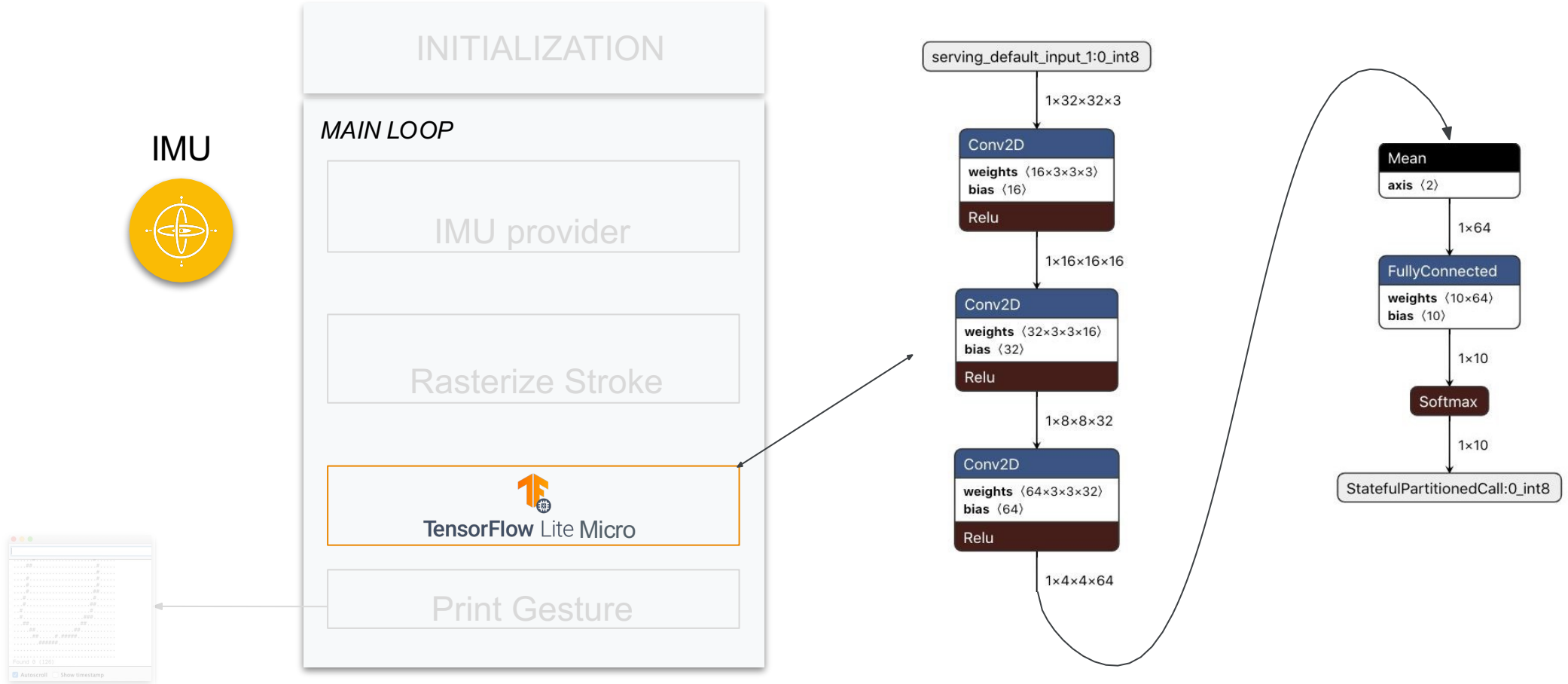
Magic Wand Application – IMU Provider



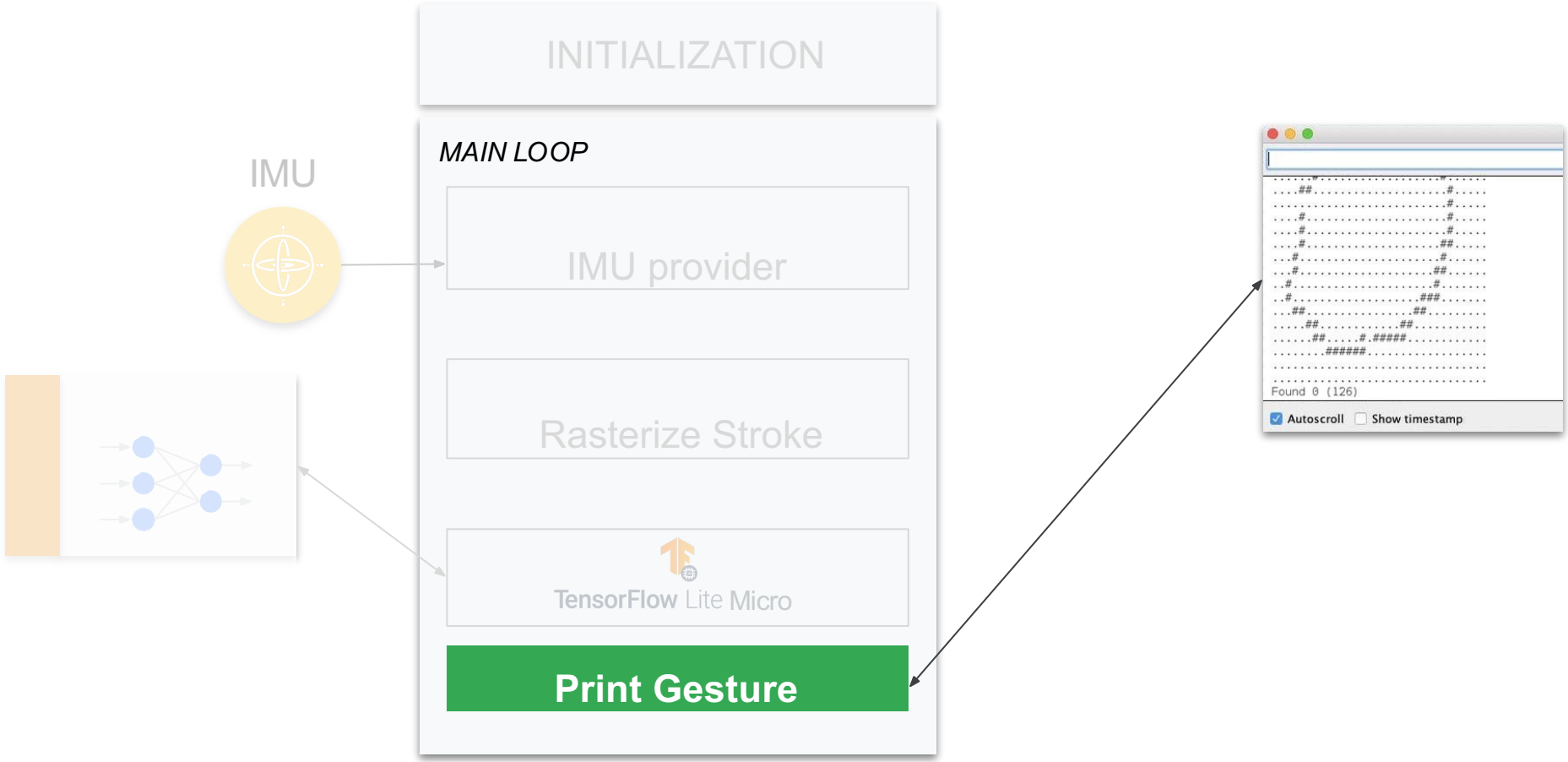
Magic Wand Application – Rasterize Stroke



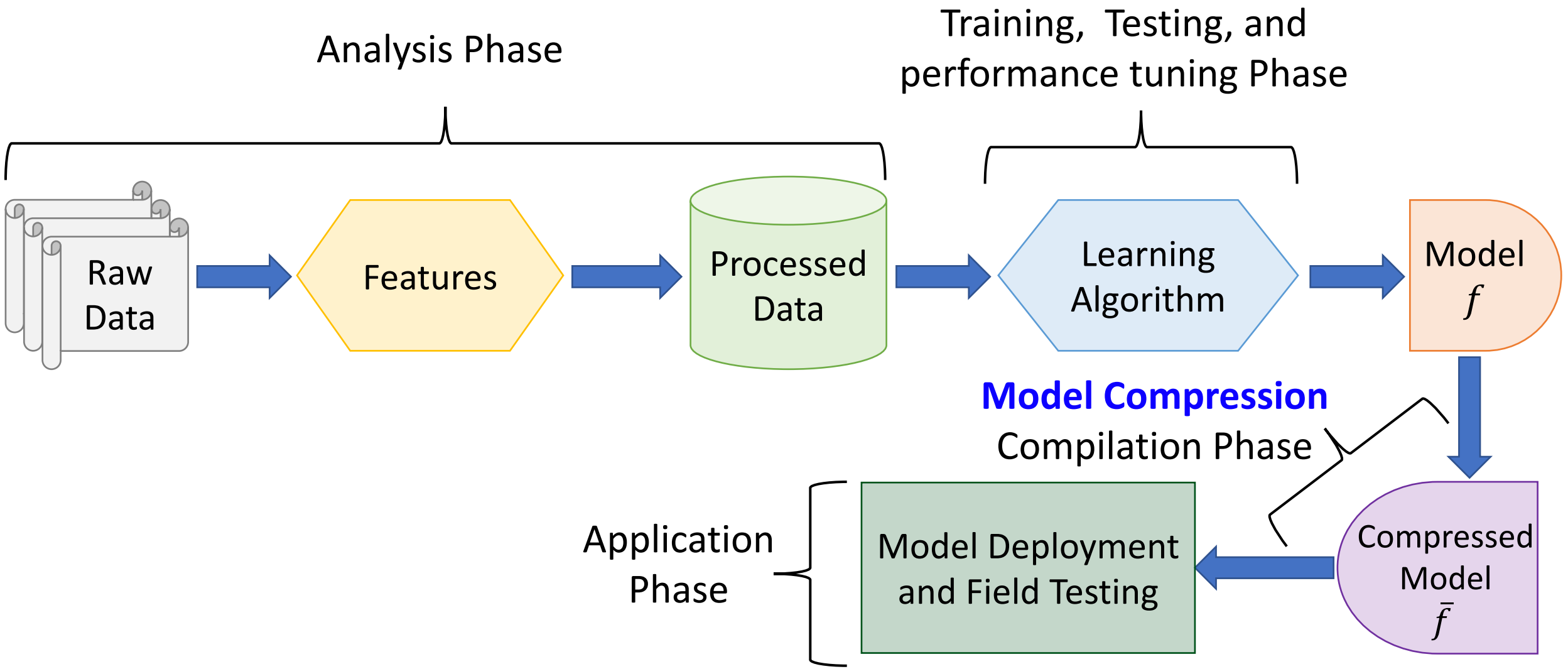
Magic Wand Application – TFLite Model



Magic Wand Application – Output



A Schematic View of TinyML Pipeline



5 min Break

Today's Lab

- Both Software and Hardware Lab!
- Gathering Data
- Training a model to recognize gesture
- Quantize the model for TinyML
- Deploy the model to the Arduino board

Lab 6 - Software - Road Map

1. Data Collection using Bluetooth
2. Labelled the data and Upload it to Google Drive
3. Rasterize the JSON files to Images
4. Train a CNN Model
5. Model Evaluation
6. Covert the TensorFlow model to TensorFlow Lite Micro model

Lab 6 - Hardware - Road Map

1. Connect Arduino Boards to your computer

2. Open magic wand example

3. Update the model in the example and deploy the model

4. Open Serial Monitor and test the model

What you need

- Mac, Linux or Windows laptop
- Arduino Nano BLE Sense 33
- MicroUSB cable
- Possibly a USB C to USB A converter, if you're on a USB C laptop
- A stick, about 12 inches (30 cm long)
- Sticky tape

Source: https://towardsdatascience.com/k-means-a-complete-introduction

Recording Gestures

1. Connect the board to the webpage using Bluetooth and Chrome's WebBLE API.
2. Moving the wand to collect the data.
3. The gestures are automatically split up by times when the wand is kept still.
4. Aim for at least 100 clean gestures of each type
5. Review and remove any sloppy or unrecognizable gestures using the trash can button
6. Label each gesture with its correct name (e.g., 0 or 1)
7. Download the reviewed and labeled data as a JSON text file for training purposes

Source: https://towardsdatascience.com/k-means-a-complete-introduction

Training and Quantize the Model

Now Open EEP595-TinyML-Lab6 (Magic Dataset Training) in Colab

Let’s train our magic wand!

Source: https://towardsdatascience.com/k-means-a-complete-introduction

Deploy the Model

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.

2. Open the magic_wand.ino sketch, which you can find in the Slack channel.

3. You'll then need to make two changes to the magic_wand.ino file to alert it of your number of gestures and gesture labels. These changes occur on lines 55-59, which currently read as:

```
//
------------------------------------------------------------
------------------- //

// UPDATE THESE VARIABLES TO MATCH THE NUMBER AND LIST OF
GESTURES IN YOUR DATASET //

//
------------------------------------------------------------
------------------- //
```

Deploy the Model

```
constexpr int label_count = 10;

const char * labels[label_count] =
{“0”,"1","2","3","4","5","6","7","8","9"};
```

a. Update the label_count to reflect the number of gestures in your dataset.

b. Update the list of labels to reflect the gestures in your dataset. Note: the order matters! Make sure it matches the alphanumeric order as printed out in the training script!

Deploy the Model

4. You'll also need to update the model data as we've done with the previous examples. The model data can be found in the magic_wand_model_data.cpp file. As we've done in the past, make sure to only update the binary values and leave the cpp syntax constant.

5. When you save, just like with the KWS examples, you will be asked to save a copy. Again, we suggest that you make a folder called e.g., TinyML inside of your Arduino folder. You can find your main Arduino folder either inside of your Documents folder or in your Home folder, and save it in that folder with a descriptive name like magic_wand_custom. That said, you can save it wherever you like with whatever name you want!

Deploy the Model

6. As always, use the Tools drop-down menu to select the appropriate Port and Board, and use the rightward arrow to upload / flash the code.

7. Now, open the serial monitor and test out your custom model. As a reminder, the serial monitor will output first ASCII art showing the gesture you just performed and below it will be the best match label as well as a confidence score between -128 and 127. The confidence score indicates how strongly the model believes you performed the gesture. Do note that every time you move the board and then stop, a new gesture will be processed, so don't be surprised to get some odd results as you move the board to prepare for a gesture.